ECE437: Introduction to Digital Computer Design and Prototyping

Instructor: T. N. Vijaykumar

Ch 1a (intro)

Fall 2016

Course administration: via Blackboard

Acknowledgements and Disclaimer

- Many slides are adapted and extended from multiple sources
  - Publisher resources
  - Profs. Thottethodi, Pai, and Patterson
  - Copyright message applies only to additions/extensions

Performance of Computers

- What do these two intervals have in common?
  - 1971 - 2014 (43 years)
  - 2015 – 2016 (2 years)

  - 1971 – first microprocessor

Performance of Computers

- 2x faster every 18 months (exponential trend: 2^n faster after 18n months)
- → 2x means double → although the intervals are different in length, absolute speed improvements of computers similar in the intervals!
- The amount of improvement in 43 years (1971 -2014) is repeated in just 2 years (2015-2016)!
- Bottomline: Computer architects rule!

Ubiquity of Computers

- Originally used by governments for military applications
  - a few handful in number
- Now used by people for all kinds of things
  - business, entertainment, engineering, science, …
  - visible and invisible
  - billions in number

So what?

- Higher performance leads to
  - Better utilization of all those billions (and more) computers
  - enable applications previously considered infeasible
- In ECE437, you will learn key techniques that help today's microprocessors achieve high performance
  - Many students get jobs based on 437

Outline

- Administrivia
- Very brief introduction

Instructor

- Instructor: Prof. T. N. Vijaykumar
  - 320 EE Building, 494 0592, vijay@ecn.purdue.edu
  - Admin. Assistant: Mary Ann Satterfield, EE325
  - Office hours: Tu 4:30-5:30 and Wed 4:30-5:30
- Lecture TA: Gowtham Baskara (details on BB)
- Text book
  - Computer Organization and Design- H/W S/W interface, Hennessy and Patterson
  - Fifth Edition – DIFFERENT from other Ed. – the exercises are different
  - You should get "5th Ed." else you will get zero on homeworks

Lectures

- I like my classes to be highly interactive
- Please **ask if something is not clear**
  - **DO NOT be intimidated**
- No question is stupid or trivial
- No confusion is small or unimportant
- Don't think everybody else gets it except you – probably some others don't get it too
- The course builds on itself so ask right away else you will not get later chapters

Lectures

- If you come to class, your work will reduce by 10x than trying to learn on your own
  - Engage/understand/internalize IN the lectures
  - NOT "go back to slides later" – INEFFECTIVE
- If there is an issue with the lab, lectures, homeworks, or exams please do tell me
  - I usually know and avoid problems but sometimes I may be unaware
  - Lab and lectures are VERY carefully synchronized so lecture is ahead
  - Lab TAs and I meet every week (+ many emails)
- I want to make sure you learn 437 very well

My personal goal

- Computer architecture (437) is my craft
- I want you to learn my craft well enough that many of you will choose to build a career on it for the rest of your lives

Tips for lab

- Learn to debug efficiently
  - If your debugging is poor, you will spend 20 hours on a bug that should take 20 minutes
  - Lab will become impossible
- Every lab code is used in later labs
  - so bugs in early labs will show up later and make your life miserable

Tips for lab

- Very hard to find the bugs from 2 months ago
  - Very hard to even realize that lab 4 code is broken when your lab 8 code does not run
- How do we combat this?
  - THOROUGHLY test code for EACH lab
  - write MANY MANY tests for EACH lab
  - WELL beyond the few tests we give you
  - Beat up your design for EACH lab

Administration

- Communication/Administration: Blackboard
  - Questions on lectures, labs, homeworks, solutions
  - Important announcements
  - Grades
  - Anonymous rants discussion board
- Confidential and/or individual correspondence -- email

ABET Course Objectives (Outcomes)

- Informally stated here
  - Formal outcome statements on Blackboard
- Three outcomes assessed in the lecture
  - Computer Arithmetic
  - Single-cycle processor + cache
  - Pipelined processor + cache
- One in the lab
  - Sub outcomes to satisfy the one major lab outcome
- Lecture outcomes:
  - Mastery questions in Homeworks/Midterms/Final Exam
  - Remediation question
  - If close but not quite there, there may be verbal remediation

Homeworks

- You can VERBALLY discuss BEFORE you write the FIRST word for ANY problem in a homework
  - You can talk about how to approach a problem and ideas for solution
  - No discussions after you start writing
  - No exchange of anything written EVER

Grading

- Grading
  - 40% lab, 10% homework, 25-30% midterms (2), 20-25% final
  - Part of homework grade will be used to incentivize
    - Attendance at visitor talks
    - Course feedback
  - Curve based, plus/minus (cluster driven)
  - Historical grade distribution
    - 20-25% A's, 20-30% B's, Rest C's, D's, F's
    - Percentages **NOT** meant as guide for this offering
  - Guaranteed F's : Failure to satisfy outcomes despite multiple opportunities, absence from exams/labs

Outline Fall

•Mostly accurate
• Important Weeks

•Midterms (tentative)
• Breaks

Week	Lectures	Lab
1	Ch 1a (intro), 2	Tutorial, Reg File, ALU
2	Ch. 4a (single cycle)	ISA
3	Ch. 4a, 1b (performance)	Mem Arbiter
4	Ch. 4b (pipeline)	Single cycle
5	Ch 4b	Pipeline
6	**Ch. 4b, 5a (cache)**	Pipeline
7	**Ch. 5a, Midterm?**	Pipeline
8	**Oct Break M-T, ch 6 (multicore), Midterm?**	Midterm report
9	**Ch 6**	Cache
10	Ch 6	Cache
11	**Ch 3a (Int ALU)**	Multicore s/w
12	Ch 5b (virtual mem),	Multicore
13	**Ch (I/O), Midterm?**	Multicore
14	Ch 3b (FP ALU) **Midterm?**	Multicore
15	**Midterm? Thanksgiving W-F**	Thanksgiving
16	Ch 3b	Final report

Outline Spring

•Mostly accurate
• Important Weeks

•Midterms (tentative)
• Breaks

Week	Lectures	Lab
1	Ch 1a (intro), 2	Tutorial, Reg File, ALU
2	Ch. 4a (single cycle)	ISA
3	Ch. 4a, 1b (performance)	Mem Arbiter
4	Ch. 4b (pipeline)	Single cycle
5	Ch 4b	Pipeline
6	**Ch. 4b, 5a (cache)**	Pipeline
7	**Ch. 5a, Midterm?**	Pipeline
8	**Ch. 6 (multicore), Midterm?**	Midterm report
9	**Ch. 6**	Cache
10	Spring Break,	
11	**Ch 6**	Cache
12	**Ch 3a (Int ALU)**	Multicore s/w
13	Ch 5b (virtual mem), Midterm?	Multicore
14	**Ch (I/O), Midterm?**	Multicore
15	Ch 3b (FP ALU)	Multicore
16	Ch 3b	Final report

Introduction of ECE437

- ECE270 - Transistors up to multiplexors
- ECE362 - Assembly Language to Instruction Set Architecture
- ECE337 – VHDL, ASIC design
- ECE437 - Puts the three together
  - great with ECE468 (compilers), ECE469 (OS)
  - if you are interested in hardware, you MUST do software!
- Requires managing complexity through ABSTRACTION

Why Study Computer Design

- To design new computers: old designs become obsolete fast
- To be an informed user
  - a little auto mechanics helps owner infrequently, but importantly
- To learn to deal with complexity via abstraction
  - problems that take months and years to complete

Why don't old designs suffice?

- (R)evolutionary changes in technology

1947	1st transistor	Bell Labs
1958	1st integrated circuit	Texas Instruments
1971	1st microprocessor Intel 4004	2300 transistors, 108 kHz
1978	Intel 8086	29K Transistors
1989	Intel 80486	1.2M Transistors
1995	Intel Pentium Pro	5.5M Transistors
2003	Intel Pentium 4	125M Transistors
2007	QuadCore Xeon	820M Transistors
2010	8-core Nehalem-EX	2.3B transistors
2014	15-core Xeon Ivy Bridge-Ex	4.3B transistors
2016	22-core Xeon Broadwell-E5	7.2B - lead feature Turbo Boost Max 3.0 developed by Purdue PhD Mike Powell

Why don't old designs suffice?

- Application needs change
  - Missile trajectories
  - Payroll processing
  - Spread sheets
  - Desktop publishing
  - Collaborative computing, Internet
  - Games
  - Facebook, Twitter, Web Search
- Next "killer app"?

Why don't old designs suffice?

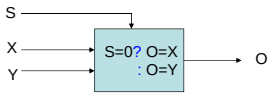
- It's not just about scale
  - More hardware, larger software
- New constraints
  - Power
    - Server-with-built-in-skillet?
  - Faults
    - What if one bit out of a million flips its state randomly?

Abstraction

- Black boxes
- Difference between interface and implementation
  - Interface - WHAT something does
  - Implementation - HOW it does so

Abstraction - Example

- 2-to-1 Mux
- Interface:



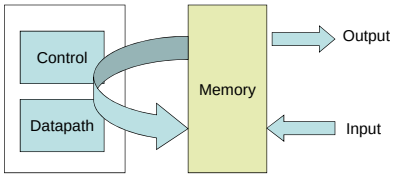
- Implementations
  - gates (fast or slow), pass transistors

What's the Big Deal?

- A real processor's interface specification
  - Huge volumes
- Worse for full computers, in general - a tower of abstraction
  - Application software
  - System software (OS and compiler/assembler/linker)
  - Hardware (CPU, memory, I/O)
- Each interface is complex and implemented with layer below
  - Abstraction keeps unnecessary details hidden
- Thousands of engineers to build one product

Basic Division of Hardware

- In space and time
  - In space



Basic Division of Hardware

- In time
  - Fetch the instruction from memory — add r1, r2, r3
  - Decode the instruction - what does this mean?
  - Read input operands — read r2, r3
  - Perform operation — add
  - Write results — write to r1
  - Determine next instruction — pc := pc + 4

Recap

- Dealing with complexity
  - Abstraction
- Machine independent HLL
  - Variables, operations
- Compiler/Assembler/linker
  - Machine (interface) dependent instructions
  - Implementation independent (backward compatibility)
- Hardware
  - Instructions

Sea Change: Multicores

- Multicore vs. Uniprocessor
  - For 2 decades uniprocessors but now multicores
  - Why do you think it changed?
    - Is it a good idea?
    - Two is better than one?
  - Why not earlier?
    - Transistors were not a problem
    - Intel Pentium4 = 100M – 125M transistors
    - Intel Pentium3 = 9.5M transistors
- Only two possibilities
  - Multicore was not a choice
  - Engineers could not see a reasonable design

Latency vs. Throughput

- Better uniprocessor (twice as fast)
  - 2x latency improvement = 2x throughput improvement
- Multicore
  - 2x throughput improvement
  - Each task takes the same amount of time → same latency
  - Can we make each task faster -- lower latency?
- Again: Why multicore?

Multicores

- We cannot make the clock any faster because power will blow up
- 4 cores can give better performance at less power than a 4-times faster clock
- But multicores need parallel programming (break one program into many threads) which is MUCH harder than sequential programming – BIG issue
  - cannot be programmed → cannot be sold!
- More in Chapter 6

Chapter order in lectures

- To stay ahead of the lab, I will follow: Chapters 1a, 2, 4a, 1b, 4b, 5a, 6, 3a, 5b, I/O, 3b
  - Maintains logical flow of the material
  - Ch. 4a+b, 5a, and 6 are crucial for the lab, so we cover them as early as possible
- For now, remember that performance means time to execute a program
  - More details later (ch. 1b, 1.6 onwards)

Bottomline

- Designers must know BOTH software and hardware
  - Compilers, Operating Systems, Networks
- Both contribute to layers of abstraction of computers
- Read the book - ~25 pages of Chapter 1 done!!
  - 1.1-1.4 done
- Through the course, read the book (preferably BEFORE lecture, or at least after lecture)