Course Logistics

• Prof Szajda
  Jepson 219
dszajda@richmond.edu
  287-6671

• Meeting Times
  Lecture: TR 10:30–11:45 (in BML–CV 176)
  Lab: F 10:30–11:20 (online via Zoom)
  Office Hours:
    email and we’ll Zoom
    and by appointment
Course Logistics

**Grading**
- Two Tests 30%
- Final Exam 25%
- Labs 20%
- Homework 10%
- Final Programming Project 15%

**Important Dates**
- Thurs., Oct. 1
  Exam 1
- Tues., Nov. 10
  Exam 2
- Thurs., Dec. 10
  Final Exam (2–5pm)
Textbook

- Computer Organization and Design: The Hardware/Software Interface, 5th Edition, by Patterson and Hennessy
Graded Work Policies

• Collaboration
  You may discuss homework and other non-exam assignments with other students in this class
  “Empty Hands” policy (see syllabus)
    Must leave any discussion/communication without any written or otherwise recorded material
    Must note who you worked with on assignment

• Late Work is absolutely not accepted!
  It is difficult enough to keep up with grading in this course even when material is submitted on time!
Attendance Policy

• I expect you to attend class and to participate (meaning, don’t come if you’re going to sleep)

• DO NOT use your laptop/tablet/digital device during class for any function other than taking notes (see syllabus)
  Surf the web and read email on your own time

• If you miss 4 or more days of class (including labs), I can (and will) give you a grade of “V”
Reading

• Read over syllabus (you are responsible for the material on this!)
• Read Chapter 1
What is this Course About?

• How do we design today’s computer systems?
  Intel Core i7 at 3.3 GHz
  6 cores – 2 threads each
  15 MB Shared cache
  64 GB Main Memory

• Starting with something that can only represent a 0 or a 1?

Transistor
Basics Behind Lots of Processors
High Level Info About Course

• Required for major or minor
• Prerequisite for many upper level courses

Material and skills you learn will be necessary for later courses

Of course, what you get out of this course depends on the effort you put into it!

You are graded on competence, not effort! (but you’ll need the latter for the former)
Specific Topics

- **SW/HW Interface**
  Assembly languages and instruction encoding

- **Processor Construction and Design**
  How to build simple processor from simple circuits

- **Memory System Design**
  How to construct a memory system that keeps processor fed

- **I/O Devices**
  How processor interacts with disk, mouse, etc.
Skill Development

- Many of these are taught in CS240 and will be further developed via assignments in this course
  - Object oriented design
  - Systematic testing
  - Debugging with a debugger
  - Learning on your own
Why Learn This?

• “I’m only going to do application programming, so why should I learn this?”

  Because you are studying the science of computing, so should understand how computing (currently) actually occurs.

Because you will need to write code that is secure, so need to understand how code is exploited (much of which requires knowledge of architecture).
Why Learn This?

• “I’m only going to do application programming, so why should I learn this?”

  Because you might someday find yourself having to code embedded devices, or device drivers, or parts of a compiler, or optimize code, or understand at a low level what is happening in a database
Why Learn This?

• “I’m only going to do application programming, so why should I learn this?”

Because you can, without realizing it, write code that performs poorly because it is badly matched to some aspect of the hardware (e.g., maps poorly to a specific memory hierarchy)

And if you don’t know architecture, you’ll have no idea what is causing the poor performance
Why Learn This?

• “I’m only going to do application programming, so why should I learn this?”

Because there are many lessons to be learned about how to design and build complex systems, and how specific design choices impact overall system performance.
Do I Have to Be An Expert?

• Depends on what you end up doing
• For most application programmers, no
  But you do need to have a basic understanding of how your code is actually being executed
Computer Architecture Overview
What is a Computer? Architecture?

- Design transistor technology
- Optimize layout, circuits, etc
- Design processor
- Design assembly language
- Write compilers
- Program software

Architecture
Where do Logic Circuits Fit?

1. Program software
2. Write compilers
3. Design assembly language
4. Design processor
5. Optimize layout, circuits, etc
6. Design transistor technology

How do I put together registers, adders, SRAM, etc?
Where do Logic Circuits Fit?

- Program software
- Write compilers
- Design assembly language
- Design processor
- Optimize layout, circuits, etc.
- Design transistor technology

How do I design register files, adders, etc. out of boolean gates?
Where do Logic Circuits Fit?

- Program software
- Write compilers
- Design assembly language
- Design processor
- Optimize layout, circuits, etc
- Design transistor technology

How do I design boolean gates out of transistors and put them on silicon?
What about Software?

- Program software
- Write compilers
- Design assembly language
- Design processor
- Optimize layout, circuits, etc
- Design transistor technology
Where do HW and SW Meet?

- Program software
- Write compilers
- Design assembly language
- Design processor
- Optimize layout, circuits, etc
- Design transistor technology

Hardware / Software Interface
System Components

• Processor
  Datapath
  Control
• Memory
• Input and output (I/O)
Mice

- **Optical mouse**
  - LED illuminates desktop
  - Small low-res camera
  - Basic image processor
    - Looks for x, y movement
  - Buttons & wheel

- **Supersedes roller-ball mechanical mouse**
Display

- LCD screen: picture elements (pixels)
  Mirrors content of frame buffer memory

Frame buffer

Raster scan CRT display
Nonvolatile Storage

- **Volatile main memory**
  Loses instructions and data when power off
- **Non-volatile secondary memory**
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)
2018 Macbook Pro
Networks

- Communication and resource sharing
- Local area network (LAN): Ethernet
  Within a building
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth
Software Terminology

Instruction Set Architecture (ISA) : note that this abstraction allows different chips with same ISA to run the same programs

Operating System vs. User program

System Software: includes OS and compiler
Software Terminology

- Binary or executable

  Compiler: translates high level language into binary representation

  Assembler: translates low level language into binary representation
In order to build circuits that implement logic, we need voltage-controlled switches

- Control input = 1 → Switch is closed
- Control input = 0 → Switch is open

This can be accomplished with electro-mechanical relays

- Large, clunky, power-hungry
- Transistors are a better way
  - Tiny, efficient, fast
MOS: “Metal Oxide Semiconductor”
this is nMOS (source/drain n-type)

P-type silicon: Excess positive charges (electron holes)
N-type silicon: Excess negative charges (electrons)
Oxide: Insulator
Gate: Metal pad

In this state, current (electrons) cannot flow between source and drain – switch is OPEN
Place a **positive** charge on the gate wire (gate = +5V)

The gate’s **positive** charge attracts **negatively-charged** electrons

This **row of electrons** forms a **channel** connecting the Source and Drain – **Current can flow** – Switch is **CLOSED**
Transistors

- Transistors
  - Store 0 or 1 when on or off
  - Can connect transistors in series or parallel to create larger building blocks called gates

CMOS Inverter created from two transistors

CMOS: Complementary Metal Oxide Semiconductor
Your Turn

• Assuming you know how to build an inverter from transistors (you do), show how to build an AND gate using nothing but a voltage source, inverter(s), and nMOS transistors.
And Gate
As of 2016, largest transistor count on commercially available single chip processor is 7.2 billion, on the Intel Broadwell-EP Xeon.
Moore’s Law – The number of transistors on integrated circuit chips (1971-2018)

Moore’s law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore’s law.

As of 2017, largest transistor count on commercially available single chip processor is 19.2 billion, on the AMD Ryzen-based Epyc
Technology: Microprocessor Logic Density

Source:
http://www.nature.com/nature/journal/v479/n7373/full/nature10676.html
Microprocessor Logic Density

Transistor Density \( MTr/\text{mm}^2 \)

- 2007: 3.3
- 2009: 7.5
- 2011: 15.3
- 2013: 37.5
- 2015
- 2017: 100.8

HVM Wafer Start Date

- 2007
- 2009
- 2011
- 2013
- 2015
- 2017
- 2019
- 2020

Intel
120 Years of Moore’s Law

[Graph showing the history of computing technology over 120 years, from mechanical devices to integrated circuits, with calculations per second per constant dollar.
Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Billion Transistor Chips

Intel Core i7 Extreme Edition - 2.27 billion transistors, 435 mm^2 area
Growing Silicon

- Silicon is a crystal grown in a vat
- It comes out the shape of a cylinder
- This is called an ingot
Creating Chips

- Sliced into thin discs called wafers
- Etch grooves and pour metal, etc
  20 - 40 steps
- Cut the wafer into dies or chips
- A flaw is called a defect
- The percentage of good ones is yield

Yield: 8/10 = 80% (not realistic)
Manufacturers secretive about yields, but can be as low as 30%
Chip Manufacturing

- https://www.youtube.com/watch?v/fwNkg1fsqBY
Cost

- **Cost per die**
  \[
  \frac{\text{CostPerWafer}}{(\text{DiesPerWafer}) \times \text{Yield}}
  \]
- **Dies per wafer**
  \[
  \frac{\text{Wafer area}}{\text{Die area}} - \text{wasted edge space}
  \]
- **Yield**
  \[
  \frac{1}{1 + \left(\frac{\text{DefectPerArea} \times \text{DieArea}}{2}\right)^2}
  \]
  2 in denominator is "\text{alpha}" which is determined by number of masking levels used (a measure of manufacturing complexity)
- **Cheapest when yield is high and dies per wafer are high**
Current Chip Trends

- Shrinking Technology
  - Reported in microns (width of wire)
  - Each generation allows more to fit in same space
  - Defect rate gradually falls in time with same technology

- Increasing Area
  - Yield – Increases chance of a defect on die
  - Dies/wafer – fewer dies, more wasted space