1. Introduction and Overview

EECS 370 – Introduction to Computer Organization – Winter 2023

EECS Department
University of Michigan in Ann Arbor, USA

Who are the faculty instructors?

- Mark Brehob, brehob@umich.edu
  - Teaching section 1, 3, and 4.
- Jonathan Beaumont, jbbeau@umich.edu
  - Teaching section 2
- Krisztian Flautner, manowar@umich.edu
  - Teaching section 5

Who am I?

- Dr. Mark Brehob
  - Full-time teacher (lecturer)
    - Been here for 22 years and I've taught a wide variety of courses (100, 101, 203, 270, 281, 370, 373, 376, 452, 470, 473)
  - PhD is in the intersection of computer architecture and theoretical computer science as it relates to caches.
  - See http://web.eecs.umich.edu/~brehob/

Student staff

- GSIs
  - Jonathan Bailey jbaile
  - Musa Haydar musah
  - Chen Huang kouchin
  - Sunny Nayak sanketn
  - Mason Nelson nelsontm
  - Mikhail Sashko msashko
  - Huanchen Sun huanchen
- IAs
  - Ibrahim Alnassar alnassar
  - Julia Aoun jcaoun
  - Dharivi Bansal dharivib
  - Caroline Bromberg bromberc
  - Kevin Choong kevincc
  - Sarah Fayyad shfayyad
  - John Kyle johnkyle
  - Luke Lesh leshlu
  - Hunter Muench hmuench
  - Emily Nagy nagyem
  - Maximos Nolan maximosn
  - Daniel Stefanescu dstefane
  - Amrita Thirumalai aathiru
  - Jennifer Williams jenwill
  - Dharivi Bansal dharivib

Class resources

- Course homepage: https://eecs370.github.io/
  - All assignments will be posted here. HW0 is there!
  - Also a link for administrative requests (SSD, Medical emergencies, etc.)
  - Exam accommodation issues need to fill out our form.

- Piazza: https://piazza.com/umich/winter2023/eecs370
  - Use for general questions on lectures, projects and homework assignments. Can discuss with your classmates.

- Gradescope: https://www.gradescope.com
  - Turn in homework assignments.
  - Details about joining Gradescope are forthcoming.

Goals of the course

- To understand how computer systems are organized and what tradeoffs are made in the design of these systems

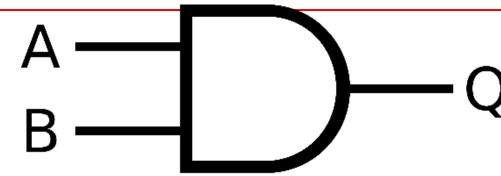
  - Instruction set architecture
  - Processor microarchitecture
  - Memory systems
  - System architecture

Where does EECS 370 fit in our curriculum?

- Software view
  - EECS 183/ENGR 101, EECS 280, EECS 281
  - Turning specs into high-level language
- Hardware view
  - EECS 270, **EECS 370**
  - gates → logic circuits → computing structures
- Prereqs:
  - C or C++ programming experience (EECS 280)
  - Basic logic (EECS 203 or EECS 270)
- Projects will be in C — not C++

Logic done quickly

S	C	X
F	F	
F	T	
T	F	
T	T	



	Math/Philosophy	Electrical/Computer Engineering	Gate
Y AND Z			
Y OR Z			
NOT Y			
Y XOR Z			

Example

S	C	B	
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

S	C	B	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

S OR (C AND B)

Basics: lectures and discussions

- Lectures:
  - Come to lecture. Expect some active learning.
- Discussions:
  - Go to your discussion section
  - Discussion sections begin meeting tomorrow
  - They are really helpful

Office hours

- Still to come. Will start on Saturday.

Calendar (from website)

Day	Topic	Assignment Released	Assignment Deadline	Petterson & Hennessey ARM 5th Readings
Week 0: Jan 6 – Jan 9				
Thursday January 5	Lecture 1: Introduction Slides	HW 0 Out		Sections 1.1, 1.2, 1.3, 1.4, 1.7
Friday January 6	Discussion 1: C & Binary (EECS 280 Review) Slides			
Monday January 9		HW1 out		
Week 1: January 10 – January 16				
Tue January 10	Lecture 2: ISA and Memory Slides	Project 1 out	HW 0 due (Wednesday the 11th)	Sections 2.2, 2.3, 2.4, 2.5
Thu January 12	Lecture 3: LC2K Slides			Sections 2.2, 2.3, 2.5, 2.6
Fri January 13	Friday setup clinic; nothing on Monday (MLK)			
Mon January 16				

Your work in 370—

- Programming assignments (4 x 10% each)
  - Assembly / functional processor simulation
  - Linker and load editor
  - Pipeline simulation
  - Cache simulation
- One midterm and a final exam
  - Midterm @ 24% — March 9th (evening)
  - Final @ 24% — April 23rd 10:30am-12:30pm
- Homeworks (12% total)
  - Total of 7 homework assignments, drop lowest
    - Assignments will have group work.

Programming assignments

- 4 programming assignments simulating the execution of a simple microprocessor
- First programming assignment posted Tuesday 1/10.
- Using C to program, C is a subset of C++ without several features like classes.
- The challenge is to understand computer organization enough that you can build a complete computer emulator.

Auto-grading projects

- We use an autograder to grade your projects
- **Projects due at 11:59pm on due date**.
  You may use up to 4 late days just for projects over the course of the semester
- Help on C available from GSIs and IAs

Academic Integrity

- We encourage collaboration in EECS 370, especially on concepts, tools, specifications, and strategies.

- See the Syllabus for examples of what collaboration is encouraged and what constitutes unacceptable collaboration.

Encouraged Collaboration	Unacceptable collaboration
Sharing high-level design strategies, e.g., helper function organization or data structure choices	Walking through an important piece of code step-by-step, sharing pseudocode, sharing comments
Helping others understand the spec or project nuances	Providing your code as a reference

- All work you submit must be your own. Collaboration must not result in code that is identifiably similar to other solutions, past or present.

EECS 370 had ~30 honor council cases last semester.

Homework assignments

- 7 written homework assignments
  - Cover lecture material
  - Good practice problems

- We will only use your 6 best homework grades

- You can discuss homework problems with your classmates, however you need to turn in your own write-up of the solution.

- First homework is posted.

How we assign course grades

- Class has a fixed grading scale
  - We may adjust thresholds in your favor
  - See the Syllabus

- Historically, about 1/3 As, 1/3 Bs, 1/3 other

- Disclaimer: Past grading is no evidence of future results!

Course textbooks

Computer Organization and Design
ARM Edition by Patterson and Hennessy

- A very good book.
- The authors have two books:
  - Ours is the intro book "Computer Organization"
  - The second book, "Computer Architecture" is used in EECS 470
  - I can only think of a handful of textbooks that have had the impact on a field like these (especially the Computer Architecture book).
  - Also good to learn from.

Reading assignments

- Reading assignments will be posted on the website, along with the slides
  - Text is optional, but quite useful.

Approximate Breakdown of Course Topics

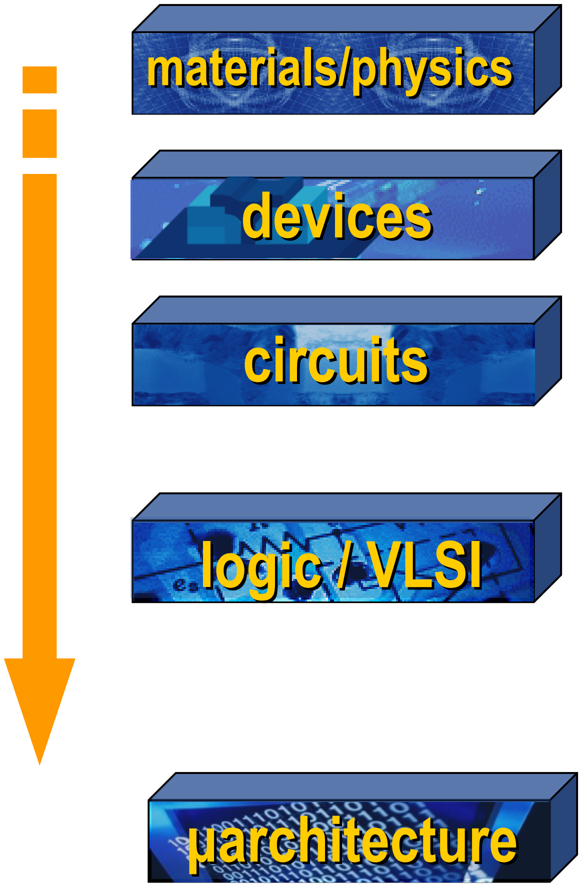
- Introduction (this lecture)
- ISAs and Assembly (~6 lectures)

- Processor implementation (~9 lectures)

- Memory (~7 lectures)
- Hot Topics (1 lecture)

- Exams/reviews (3 lectures)

Where this course material fits in the system stack



- Quantum-level, solid state physics
- Conductors, Insulators, Semiconductors.
- Doping silicon to make diodes and transistors.
- Building simple gates, boolean logic and truth tables
- Combinatorial logic: muxes, decoders, adders
- Clocks
- Sequential logic: latches memory
- State machines
- Processor Control: Machine instructions
- Computer Architecture: Defining a set of instructions

What is 370 about?

- You're used to writing programs like this

```
void daxpy(int n, double a,
           double *x, double *y)
{
  for (int i = 0; i < n; ++i)
      y[i] = a*x[i] + y[i];
}
```

- But what's actually happening ***inside*** the computer when we compile / run this?
  - Come to think it… what does "compiling" even mean??

- 370 in a nutshell: **How do computers execute programs?**

You will need to know this stuff if…

- You work in designing processors at Intel, ARM, NVIDIA, etc.
- You write optimized library code
- You work on designing operating systems or compilers
- You work in computer security
  - Remember this?
- You work in designing embedded systems (IOT, etc.)

You might need to know this stuff if…

- Even if you just write software for the rest of your life
  - Important to know what your computer is doing when it executes your code!
  - It can make a big difference in your performance

- A great example comes from the #1 StackOverflow question of all time…

Example

- Consider 1 version of code that loops through an array of random values and adds all elements larger than 128
  - Takes 1.5 seconds to sum values

```
for (unsigned c = 0; c < arraySize; ++c)
    data[c] = std::rand() % 256;

// Test
clock_t start = clock();
long long sum = 0;
// Primary loop
for (unsigned c = 0; c < arraySize; ++c)
{
    if (data[c] >= 128)
        sum += data[c];
}

double elapsedTime =
  static_cast<double>(clock() - start);
```

https://stackoverflow.com/questions/11227809/

Example

- What will happen to the execution time of the loop if we sort the array beforehand?

```
for (unsigned c = 0; c < arraySize; ++c)
    data[c] = std::rand() % 256;
std::sort(data, data + arraySize);

// Test
clock_t start = clock();
long long sum = 0;
// Primary loop
for (unsigned c = 0; c < arraySize; ++c)
{
    if (data[c] >= 128)
        sum += data[c];
}

double elapsedTime =
  static_cast<double>(clock() - start);
```

What do you think will happen?

A. Sorted array sums much faster

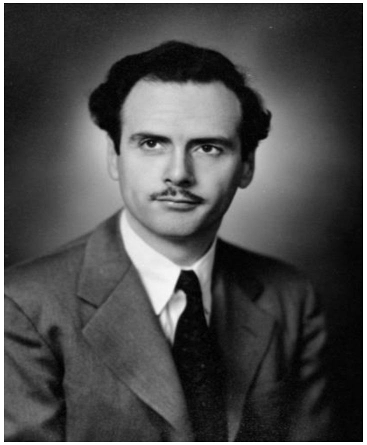
B. Sorted array sums much slower

C. No significant difference between sorted and unsorted arrays

D. I have no idea!

Why take 370?

- Inherent value in understanding how the tools we use work



"We shape our tools and thereafter our tools shape us"

- Marshal McLuhan

Let's take a short break

- In the meantime…
- What processor are you using?
  - Windows: Control Panel > System and Security > System
  - Mac: Click top-left Apple icon > About this Mac
  - Linux: lscpu
- How many cores does it have?
  - Windows: Ctrl-Shift-Esc > Performance

What hardware do you have with you today?

How programs work in a nutshell

- We're used to seeing programs like this

```
void daxpy(int n, double a,
           double *x, double *y)
{
  for (int i = 0; i < n; ++i)
      y[i] = a*x[i] + y[i];
}
```

- But computer hardware is limited and can't understand code this complicated
  - Tons of different keywords and variable names…
  - Matching up different parentheses
  - Do we have to hardwire all this in logical circuits???

How programs work in a nutshell

- High level languages are intended to be easy for humans to understand / write
  - **NOT** for hardware to execute
- To be executed, must **compile** this complicated code into a set of **very simple** and easy to understand instructions that hardware can execute

```
void daxpy(int n, double a,
           double *x, double *y)
{
  for (int i = 0; i < n; ++i)
     y[i] = a*x[i] + y[i];
}
```

How programs work in a nutshell

- THIS is what computers can understand and execute

example.exe

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000510	48	83	ec	08	48	8b	05	cd	0a	20	00	48	85	c0	74	02
00000520	ff	d0	48	83	c4	08	c3	00	00	00	00	00	00	00	00	00
00000530	ff	35	8a	0a	20	00	ff	25	8c	0a	20	00	0f	1f	40	00
00000540	ff	25	8a	0a	20	00	68	00	00	00	00	e9	e0	ff	ff	ff
00000550	ff	25	a2	0a	20	00	66	90	00	00	00	00	00	00	00	00

- This is called "machine code": just a bunch of 0s and 1s (easier for us to read if we convert it into hex digits)
- It's what's produced when you type:
  - g++ example.c -o example.exe

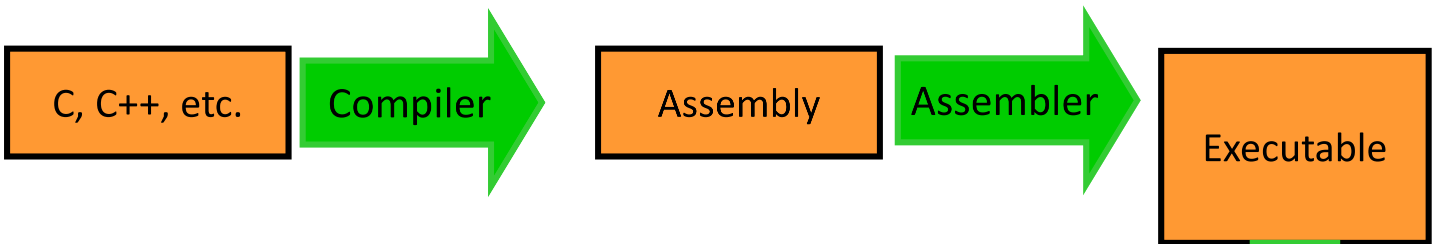
Binary, Hex, etc.

How programs work in a nutshell

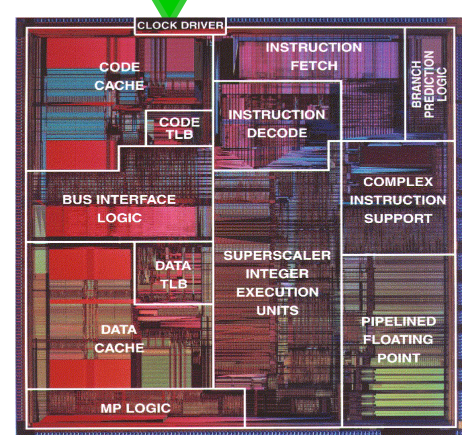
- Humans often work at an intermediate level by writing **assembly code**
- Usually has a 1-1 correspondence with machine code instructions
  - Gives the programmer fine control over the final executable
- But it's (relatively) easy to read
- You can view generated assembly code with -s flag in g++:
  - g++ -s example.c

```
daxpy:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    %edi, -20(%rbp)
        movsd   %xmm0, -32(%rbp)
        movq    %rsi, -40(%rbp)
        movq    %rdx, -48(%rbp)
        movl    $0, -4(%rbp)
        jmp     .L2
```

Source Code to Execution (simplified)



- There are some things missing here... we'll fill those in later
- First 2 weeks of the course will cover this process
- Remainder of the class will be... how do you build this thing?? (and memory)



Architectures

- Not just one type of machine code produced for all types of computers
- Just like how there are several different programming languages (C/C++, Java, Python, etc)…
  - there are also many different types of **architectures** that code can be compiled to run on
- Popular architectures:
  - x86, ARM, RISC-V
- Code compiled for one architecture will not run on another

x86

- Designed by Intel (AMD designed 64-bit version)
- Beefy, complex, fast, power-hungry
- Used in:
  - Desktops
  - Most laptops
  - Servers
  - PlayStation 4/5, Xbox One

ARM

- Designed by… ARM
- Versatile: can be used for higher performance or low-power usage
- Used in:
  - Most smartphones
  - Recent Macbooks
  - Recent supercomputer clusters
  - Nintendo Switch

RISC-V

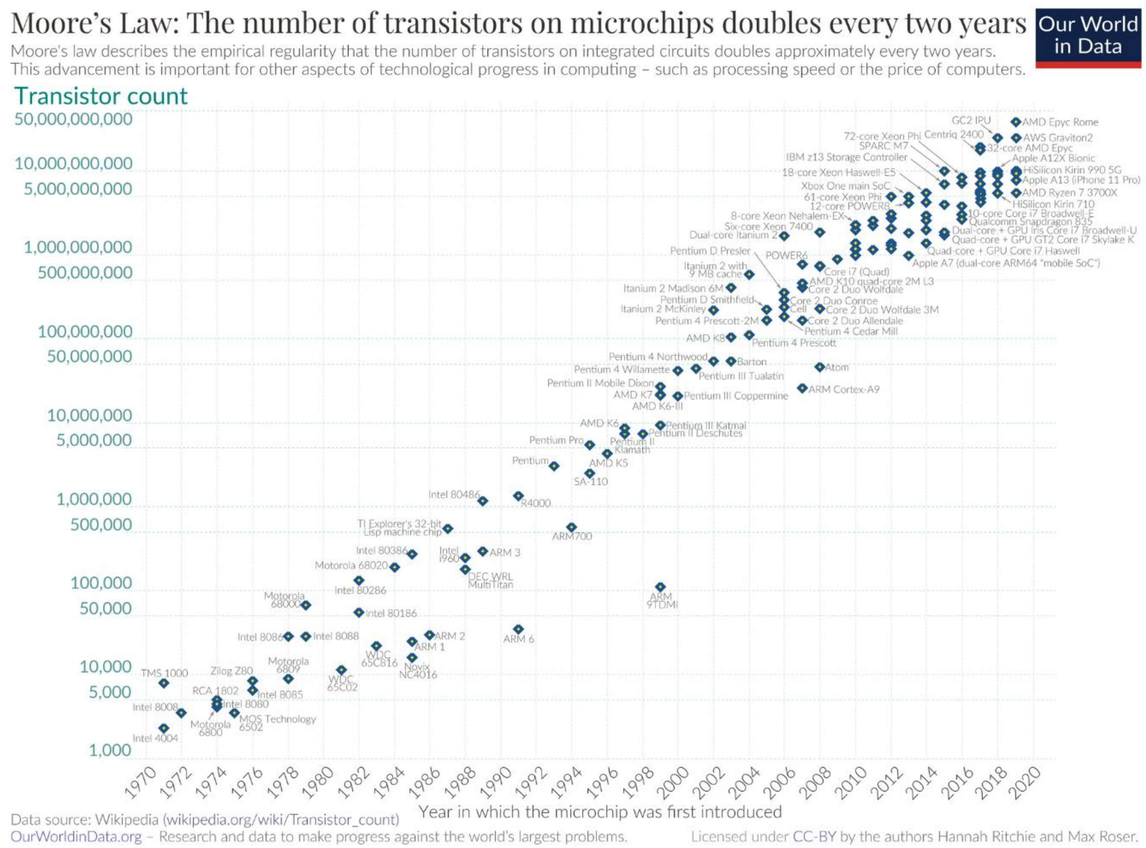
- Open source
- Very popular in academia
  - Don't need to pay super-expensive licensing fees
- Starting to make its way into actual products

Architectures Discussed in this Class

- We primarily focus on:
  - A subset of ARM called "LEG" (hardy-har-har)
  - A made-up ISA we call LC2K (Little Computer 2000)
    - Extremely simple, lets us focus on the concepts
    - Not practical for real applications

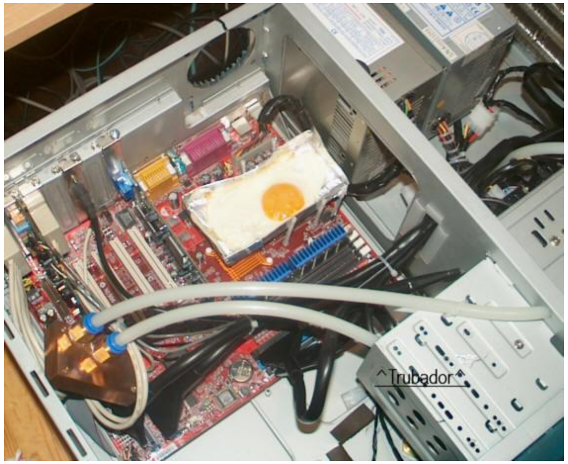
The Trend of Computing

- Moore's Law



The End of Moore's Law?: Dennard Scaling

- Dennard Scaling: as transistors get smaller their power density stays constant
- Translation: as the number of transistors on a chip grows (Moore's Law), the power stays roughly constant
- Mid-2000's Dennard Scaling broke. Why? Transistors got so small that they began to leak a lot of power. Leaking lots of power caused a chip heat up a lot.
- Conclusion: you can put lots of transistors on a chip, but you can't use them all at full power at the same time.
  - You'll melt the processor!



Reminders

- Project 1 posted Tuesday
- Homework 0 posted; due Wednesday
- Discussion starts this week
  - learn about C programming, debugging methods and tools, and more.