# Welcome! CS240
# Principles of Computer Organization

Instructor: Aline Normoyle

Textbooks:

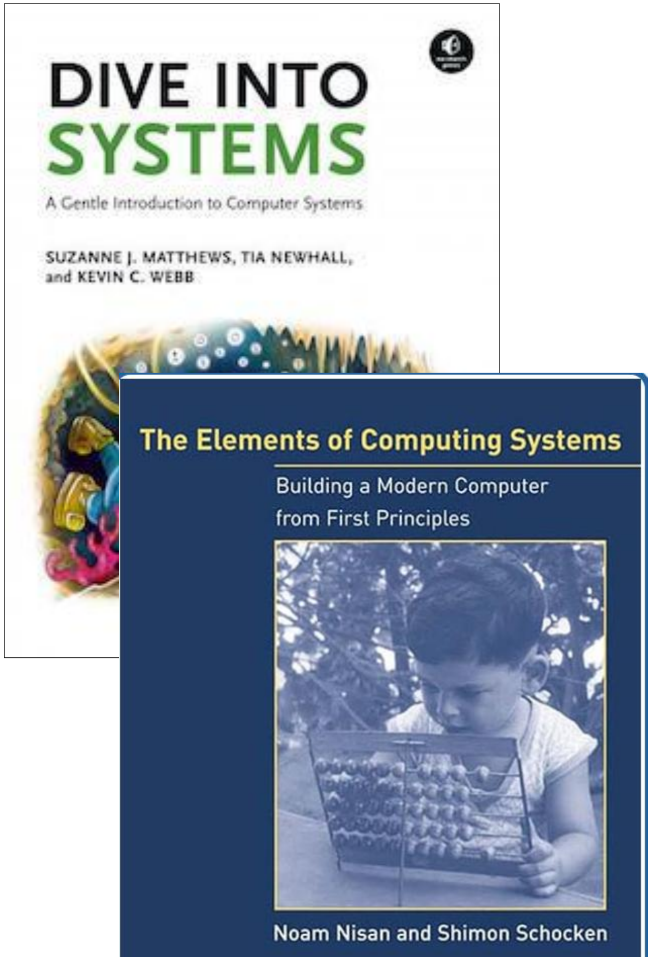
- Dive into Systems
- Elements of Computing Systems

Slack: Announcements, links, etc

Website: Policies, syllabus, etc

Github: Code repository

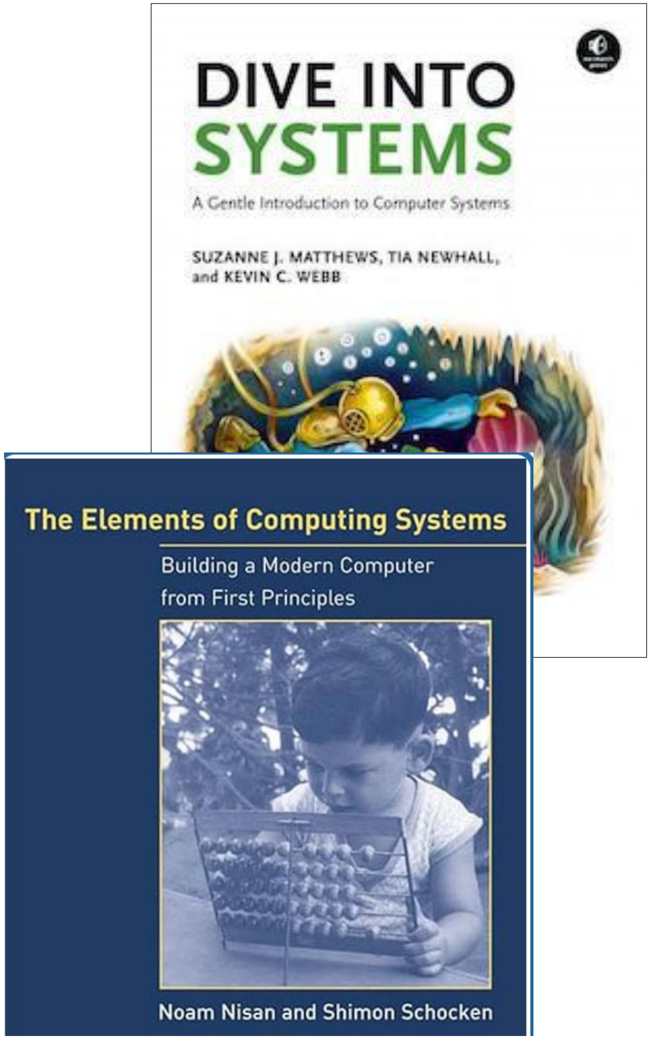
Lab: Park 231


DIVE INTO SYSTEMS
A Gentle Introduction to Computer Systems
SUZANNE J. MATTHEWS, TIA NEWHALL, and KEVIN C. WEBB
The Elements of Computing Systems
Building a Modern Computer from First Principles
Noam Nisan and Shimon Schocken

# Book Resources

https://diveintosystems.org/

https://nand2tetris.org



DIVE INTO SYSTEMS

A Gentle Introduction to Computer Systems

SUZANNE J. MATTHEWS, TIA NEWHALL, and KEVIN C. WEBB

The Elements of Computing Systems

Building a Modern Computer from First Principles

Noam Nisan and Shimon Schocken

# Course Resources

**Webpage**

https://brynmawr-cs240-f25.github.io/website/

**Github**

https://github.com/BrynMawr-CS240-f25/

**Slack**

https://BrynMawr-CS240-f25.slack.com

# What you will learn

- C/C++ programming
- How computers work and how they are built in layers
  - Boolean logic, gates, arithmetic
  - Machine language, assembly, virtual machines
  - High-level language, compilers
  - Operating system
- Skills: UNIX, git, basic hardware

# Computers: Layers of Abstraction

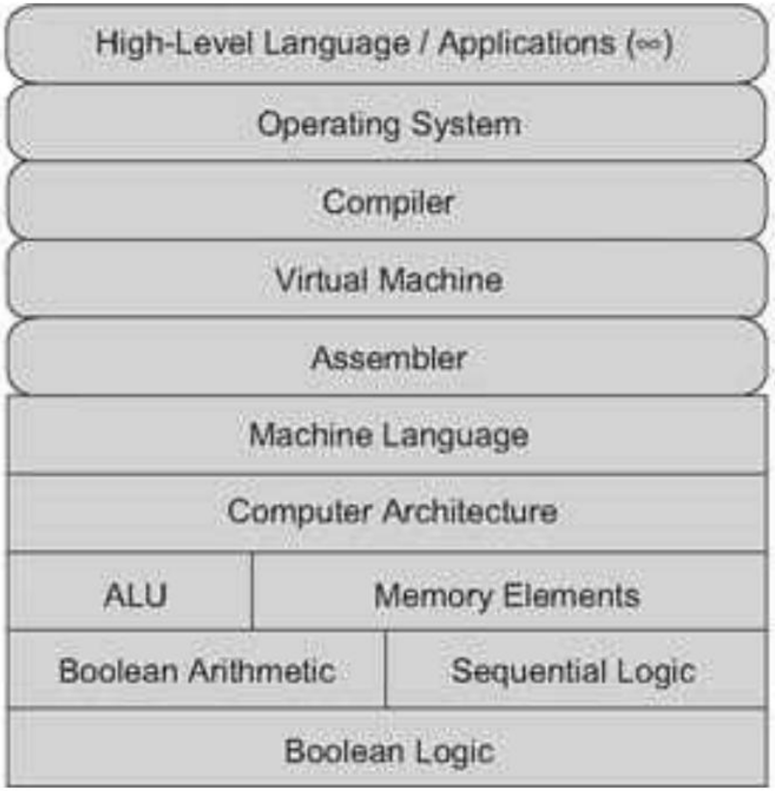
Weeks 01-02: Basic C, Binary Representations, Principles of computer architecture

Weeks 03-12:

Build a full computer emulator in C

Software (Weeks 07-12)

Hardware (Weeks 03-05)



# Course topics:
# From bits to apps

Nand
Nand to Tetris

| a | b | Nand |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Hardware: Logic gates, Boolean arithmetic, multiplexors, flip-flops, registers, RAM units, counters, clock

Architecture: ALU/CPU design and implementation, addressing modes, memory-mapped I/O, machine code, assembly language programming

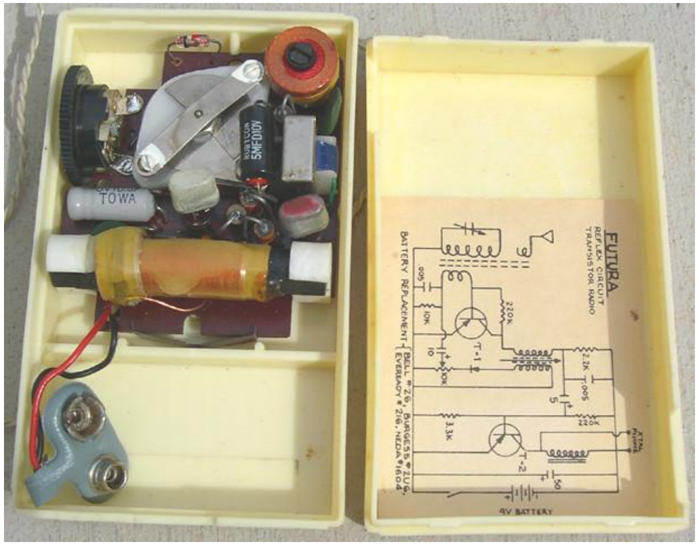
Programming Languages: Object-based design and programming, abstract data types, scoping rules, syntax and semantics, references.

Compilation: Lexical analysis, top-down parsing, symbol tables, pushdown automata, virtual machine, code generation, implementation of arrays and objects.

Data structures and algorithms: Stacks, trees, hash tables, lists, recursion, arithmetic algorithms, geometric algorithms,

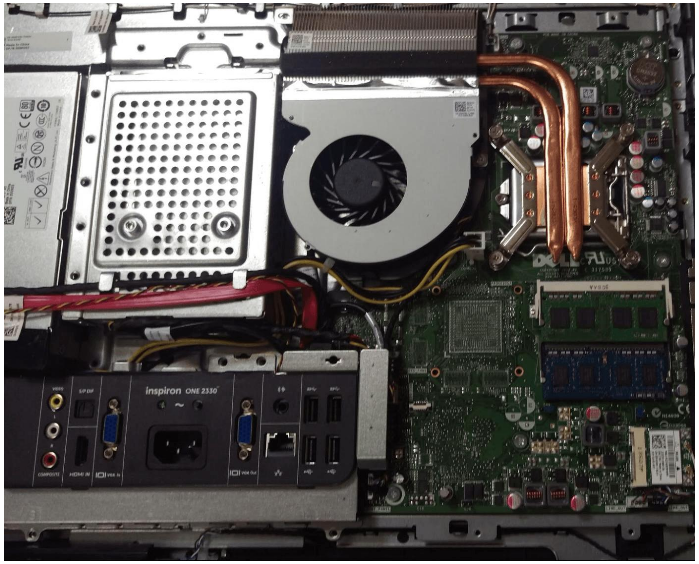
Engineering: Abstraction /implementation, modular design, API design and documentation, unit testing, quality assurance, programming at the large.

# Philosophy: Learning through dissection

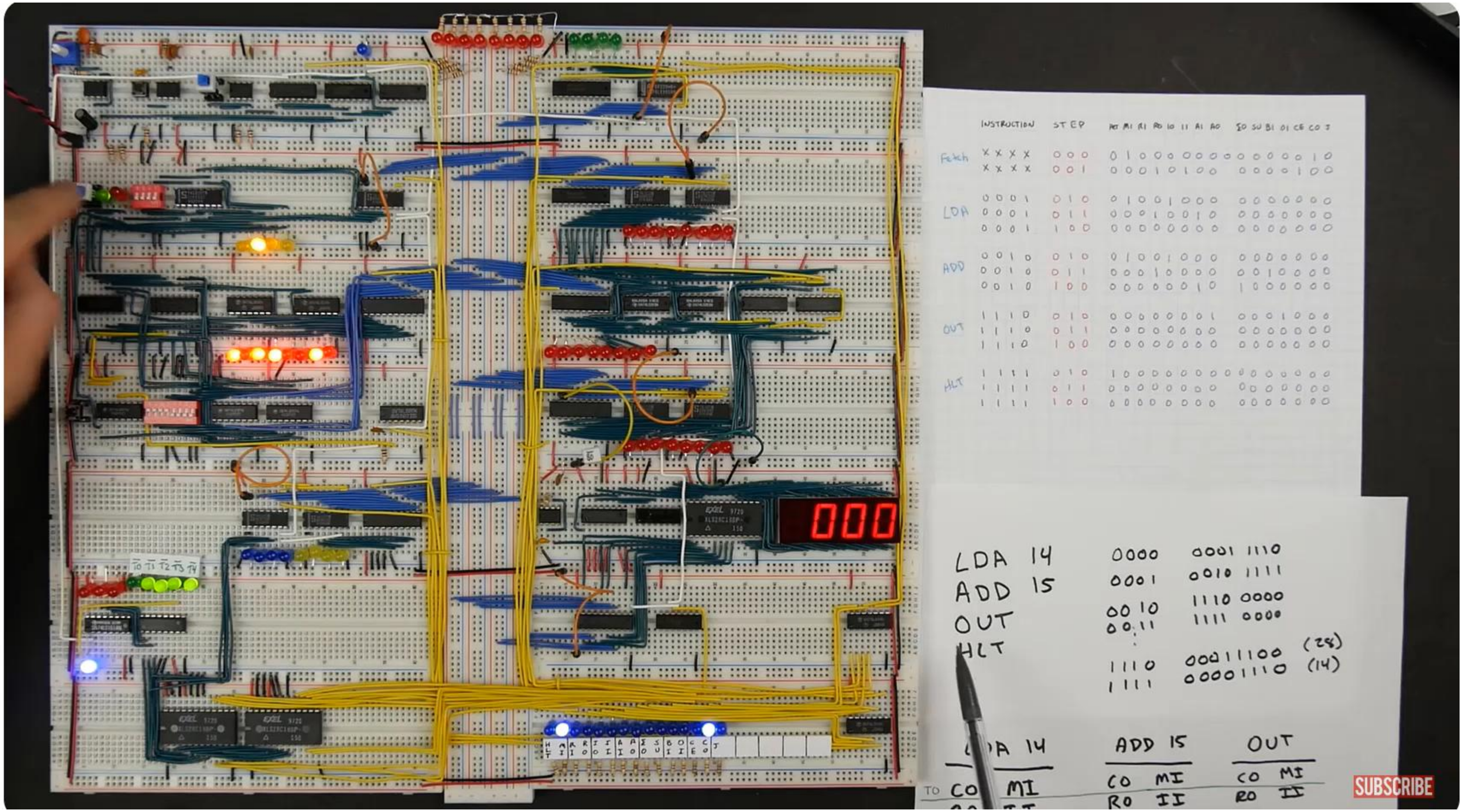


1950s transistor radio could be taken apart to see how they work

Modern computers consist of small components – it’s possible to customize and make your own devices but not novice-friendly



# Demo: 8-Bit Computer



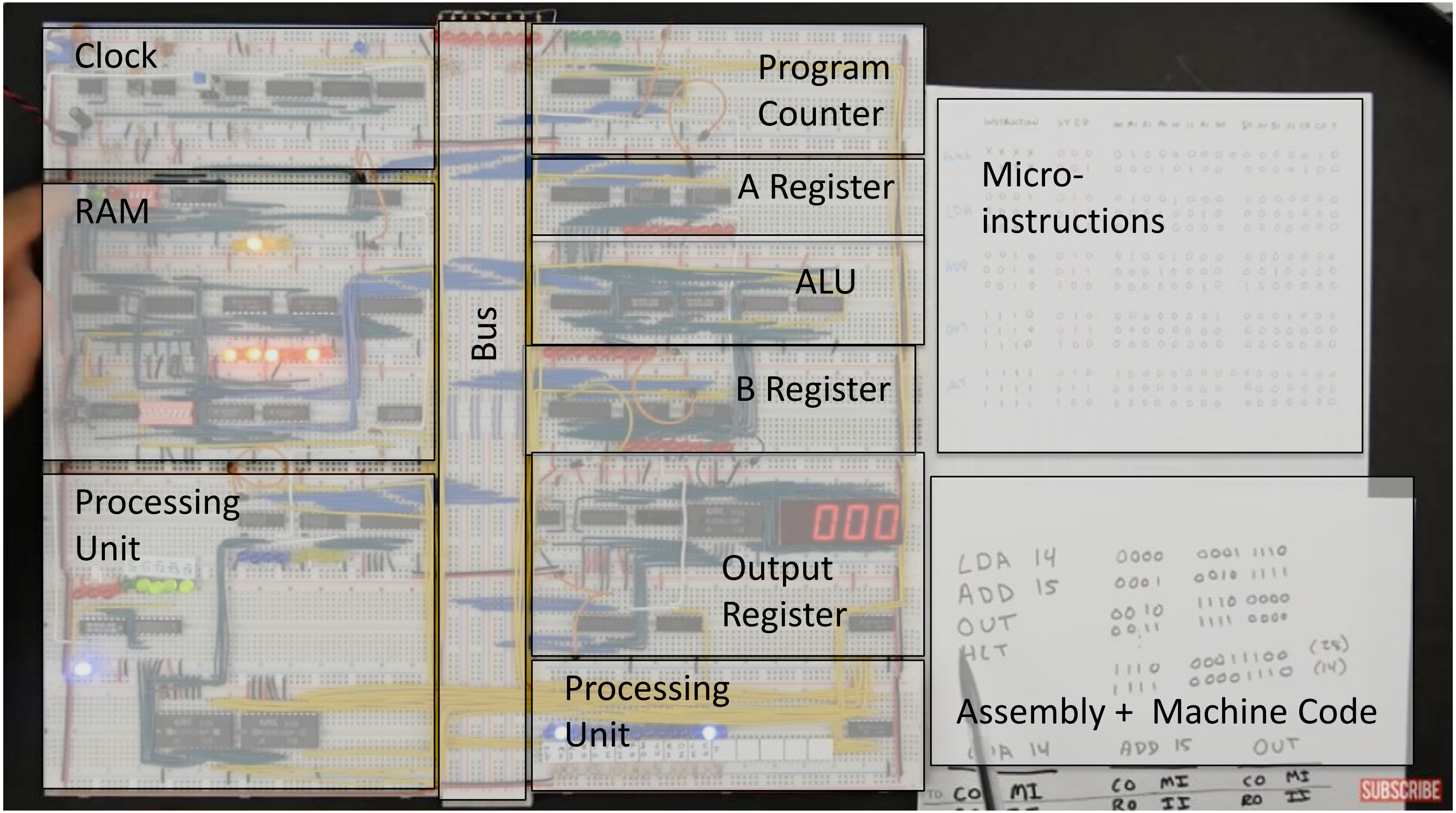
| | INSTRUCTION | STEP | HLT MI RI RO IO II AI AO | ΣO SU BI OI CE CO J |
|---|---|---|---|---|
| Fetch | X X X X | 0 0 0 | 0 1 0 0 0 0 0 0 | 0 0 0 0 0 0 1 0 |
| | X X X X | 0 0 1 | 0 0 0 1 0 1 0 0 | 0 0 0 0 1 0 0 |
| LDA | 0 0 0 1 | 0 1 0 | 0 1 0 0 1 0 0 0 | 0 0 0 0 0 0 0 |
| | 0 0 0 1 | 0 1 1 | 0 0 0 1 0 0 1 0 | 0 0 0 0 0 0 0 |
| | 0 0 0 1 | 1 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 |
| ADD | 0 0 1 0 | 0 1 0 | 0 1 0 0 1 0 0 0 | 0 0 0 0 0 0 0 |
| | 0 0 1 0 | 0 1 1 | 0 0 0 1 0 0 0 0 | 0 0 1 0 0 0 0 |
| | 0 0 1 0 | 1 0 0 | 0 0 0 0 0 0 1 0 | 1 0 0 0 0 0 0 |
| OUT | 1 1 1 0 | 0 1 0 | 0 0 0 0 0 0 0 1 | 0 0 0 1 0 0 0 |
| | 1 1 1 0 | 0 1 1 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 |
| | 1 1 1 0 | 1 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 |
| HLT | 1 1 1 1 | 0 1 0 | 1 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| | 1 1 1 1 | 0 1 1 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 |
| | 1 1 1 1 | 1 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 |

| | | | |
|---|---|---|---|
| LDA 14 | 0000 | 0001 1110 | |
| ADD 15 | 0001 | 0010 1111 | |
| OUT | 0010 | 1110 0000 | |
| HLT | 0011 | 1111 0000 | |
| | ⋮ | | |
| | 1110 | 00011100 | (28) |
| | 1111 | 00001110 | (14) |

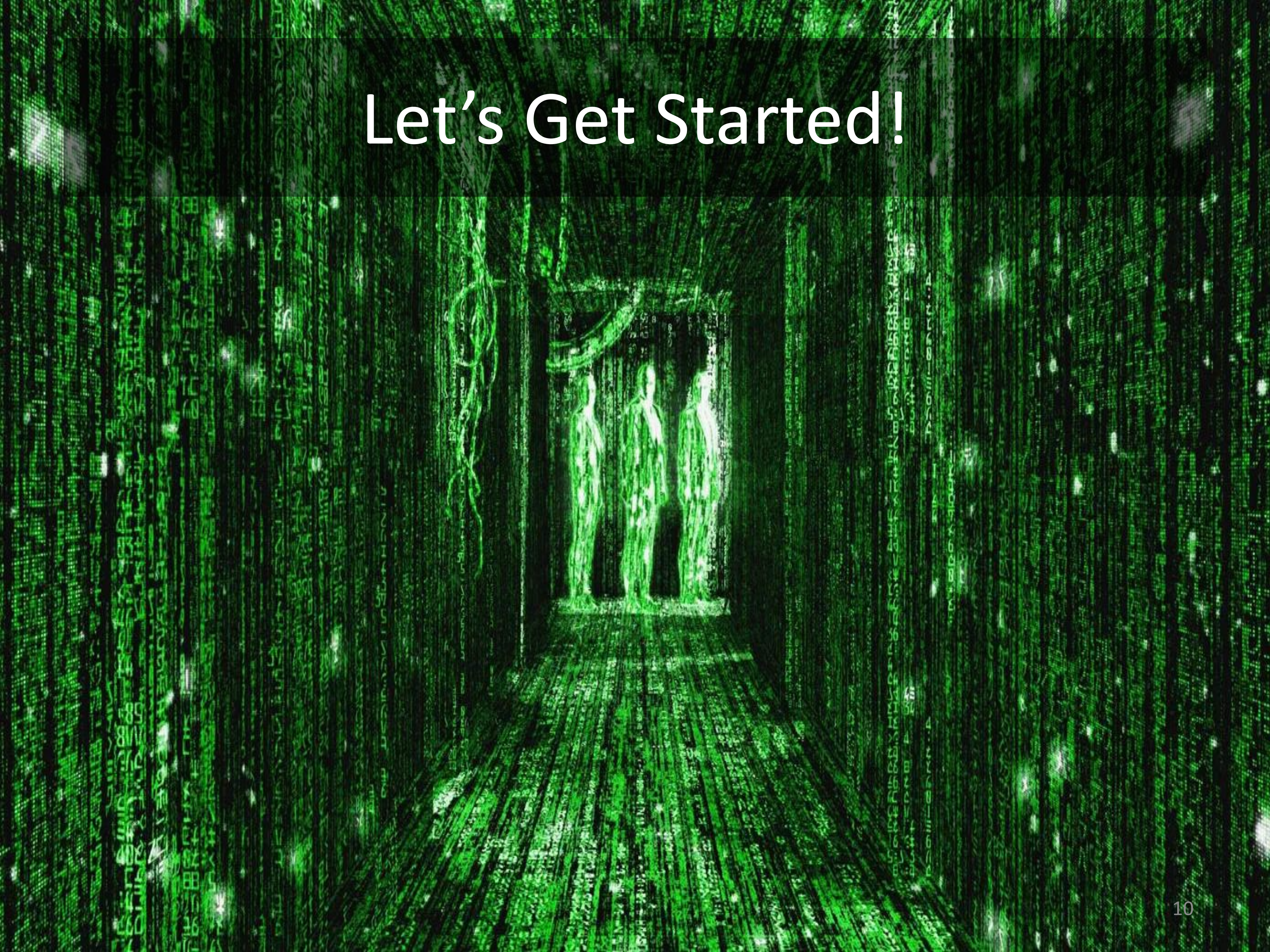
| | LDA 14 | | ADD 15 | | OUT | |
|---|---|---|---|---|---|---|
| TO | CO | MI | CO | MI | CO | MI |
| | | | RO | II | RO | II |

https://eater.net/8bit/control

# Demo: 8-bit computer



# Let’s Get Started!



# Development Environment

A **development environment** consists of the platform and tools that you use to write software

Systems programmers need to be able to

- work from terminal using shell commands
- program in low-level languages
- use debugging and profiling tools

This class:

- Operating system: Ubuntu (Linux)
- Programming languages: C, x86_64 assembly language
- Editor: nano, vim, or emacs
- Makefiles for compiling and linking
- git for source control

# C

- High-level programming language
  - Java, python, ruby, Javascript, C++, etc
  - Imperative (sequence of statements)
  - Procedural (structured using functions)
  - No classes, built-in types such as strings, lists

- Less abstracted than other languages
  - easier to see relationship between code and the computer's running of it
  - capable of more efficient code

# From Java to C: Hello World

```
class Hello {
  public static void main(String[] args) {
    System.out.println(“Hello World”);
  }
}
```

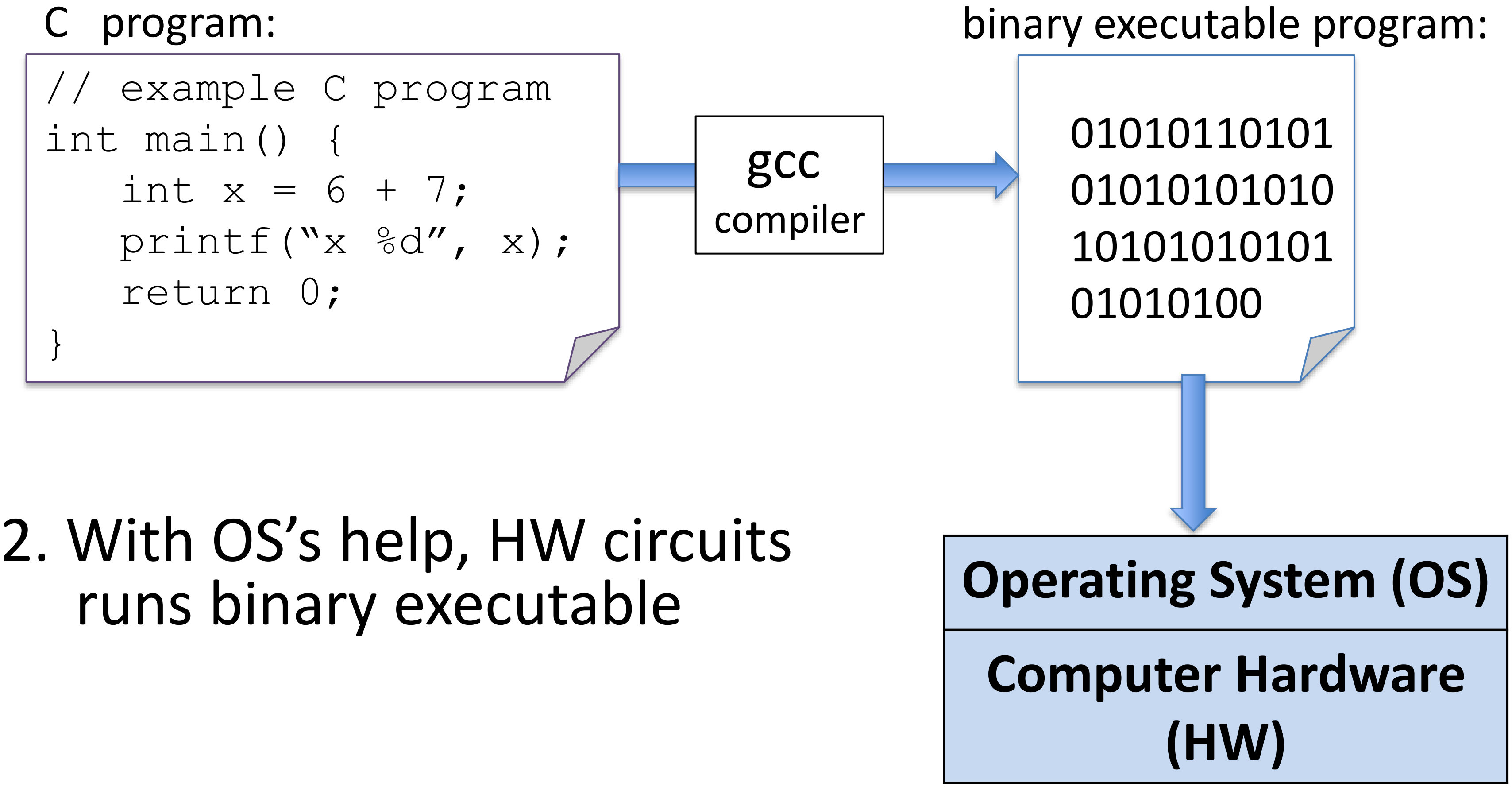
```
#include <stdio.h>

int main(int argc, char** argv) {
  printf("Hello World!\n“);
  return 0;
}
```

To compile: javac hello.java
To run: java Hello

To compile: gcc hello.c
To run: ./a.out

# Building and Running a C program

1. **Compiling** a C program translates it to binary (0’s and 1’s )
   - The binary file is an **executable**, meaning “we can run it”



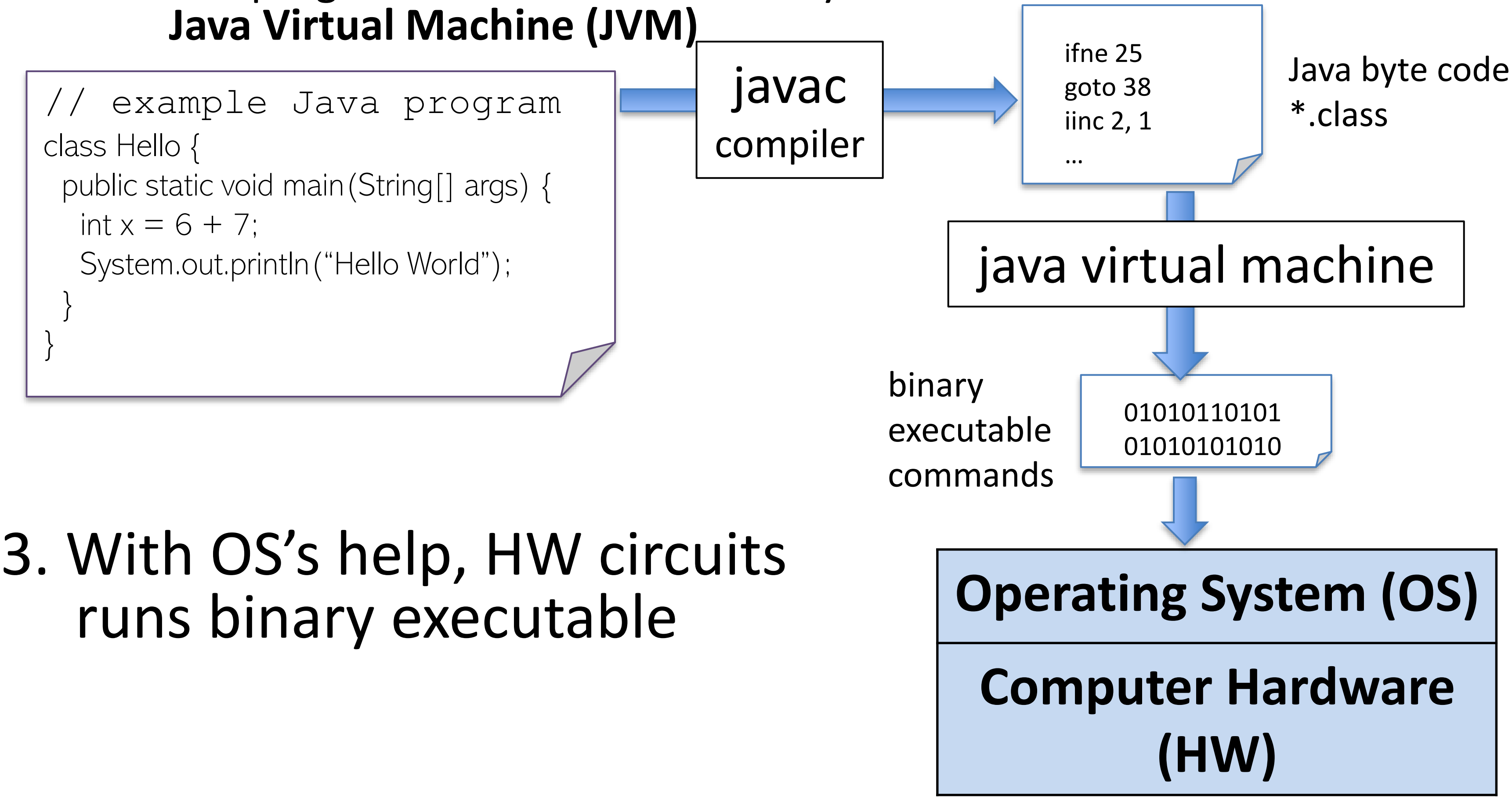
C program:

```
// example C program
int main() {
   int x = 6 + 7;
   printf(“x %d”, x);
   return 0;
}
```

2. With OS’s help, HW circuits runs binary executable

# Building and Running a Java program

1. **Compiling** (javac) a Java program translate it to Java byte code
2. **Running** (java) translates the program to binary (0’s and 1’s )
   - The program that translate from byte code to machine code is called the **Java Virtual Machine (JVM)**



```
// example Java program
class Hello {
  public static void main(String[] args) {
    int x = 6 + 7;
    System.out.println(“Hello World”);
  }
}
```

3. With OS’s help, HW circuits runs binary executable

Slide based on those by Tia Newhall, CS31, Swarthmore

# All programs must eventually become binary (0’s and 1’s) to run on a computer

- The binary code is specific to the hardware
- Higher-level languages (e.g. Java) have more **layers of abstraction** between the programmer’s code and the binary code
  - higher-level languages are **cross-platform**, e.g. the same program can run on different hardware
    - ex. Our C and Java programs run on mac, windows, and linux

# Makefiles

Idea: Put all build commands into a file

```
$ nano Makefile
$ make hello
```

```
CC=gcc
% :: %.c
  $(CC) -g -Wall -Wvla -Werror -Wno-unused-variable $< -o $@

all: hello

clean :
  rm hello
```

# Review: UNIX basics

Ubuntu Desktop has a window manager (lab machines) but we will mostly be using **command-line interfaces (CLI)**

**terminal** – text-based interface for the OS

**command line** – current line in the terminal; where we issue a command

**command prompt** – prefix text at the beginning of the command line

**shell** – program that executes commands from terminal

- **bash –** the shell we will use in this class!
- **zsh** – mac shell
- **powershell** – windows shell

# Exercise: Connect to a server

On a laptop or home desktop computer, open a terminal and ssh to comet

```
$ ssh <username>@comet.cs.brynmawr.edu
```

# Exercise: Edit a file

Write and compile a program, `hello.c`, that prints “Hello World”

```
$ nano hello.c

$ gcc hello.c
$ ./a.out

$ gcc hello.c –o hello
$ ./hello
```

# Reference: Some useful commands

- ls – list all directories
- cd, mkdir, mv, cp, rm – change directory, make directory, move, copy, remove
- cat, less, more – showing files
- javac, gcc, make – compiling programs
- vi, nano, emacs – editing files
- grep, find – searching files
- man – read documentation (RTFM: “Read the fine manual”)
- ssh <username>@goldengate.cs.brynmawr.edu – log into CS server
- git – source control

# Working with paths from terminal

- What are files? What are directories?

- path - full name of a file or directory that indicates the file/directory location within the file system
  - Absolute paths: path from the root of the file system to the file
  - Relative paths: path from **current working directory** to the file

- File extension: Tells the OS what type of data is in the file (ex: *.txt, *.jpg, etc)

# Special directories

**..** ← the parent directory (two dots)

. ← the current directory (one dot)

/ ← the root directory

/home/<username> ← your home directory

~ ← your home directory

# Example

```
root
-- A
---- hello.txt
-- B
```

What  is the absolute path of hello.txt?

What  is the absolute path of hello.txt from the A directory?

What is the relative path of `hello.txt` from

- the root directory?
- the A directory?
- the B directory?

# Working with paths

```
root
-- home
---- ren
------ A
---- stimpy
------ B
------ C
-------- hello.txt
```

What is the absolute path of hello.txt?

If we are in the directory A, what is the relative path of hello.txt?

If we are in the directory B, what is the relative path of hello.txt?

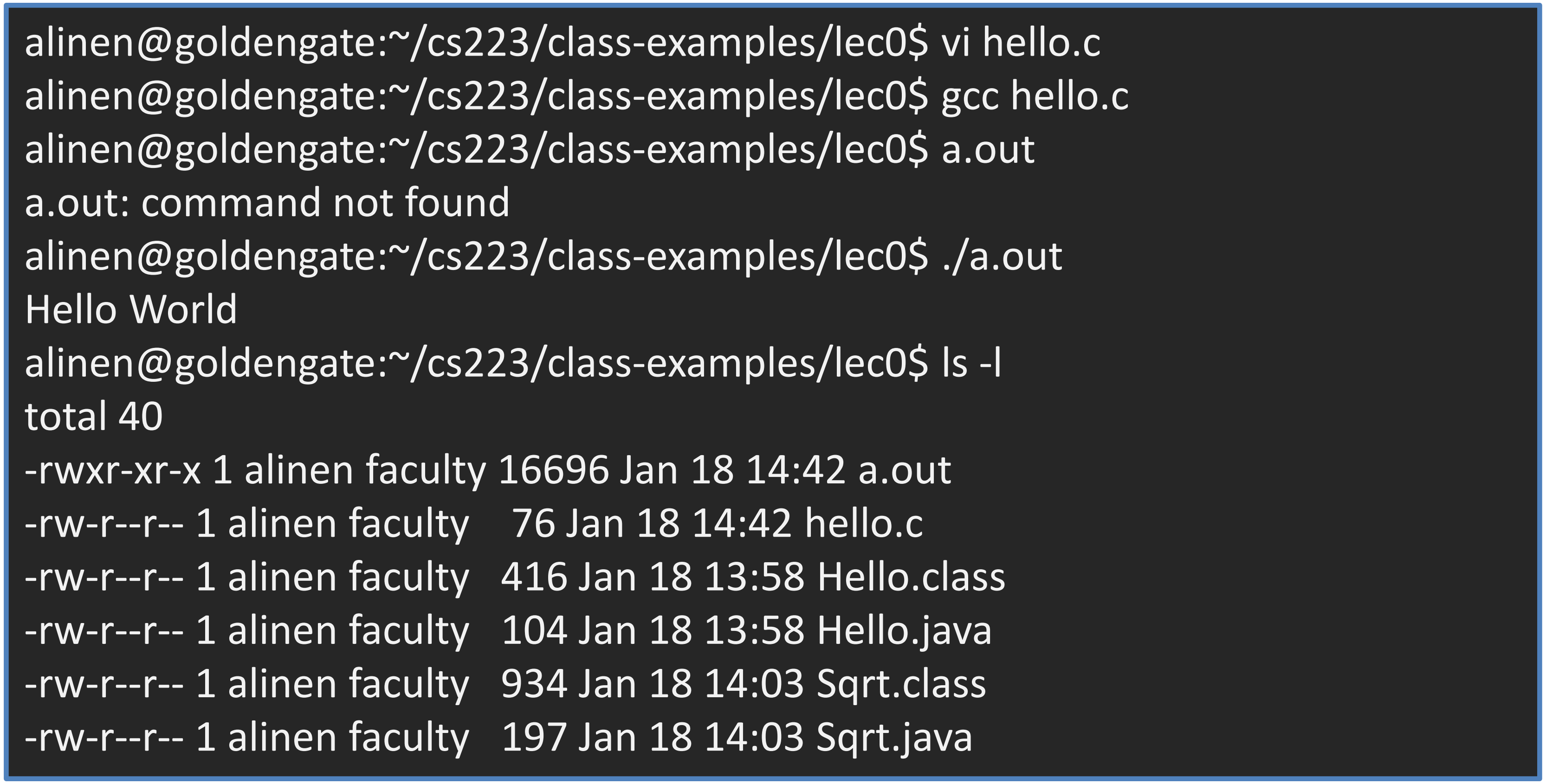
# Example: Working with paths

```
alinen@goldengate:~/cs223/orig-class-examples/lec0$ cat ../../hello.c
#include <stdio.h>
int main() {
  printf("Hello World\n");
}
alinen@goldengate:~/cs223/orig-class-examples/lec0$ cat ~/cs223/hello.c
#include <stdio.h>
int main() {
  printf("Hello World\n");
}
alinen@goldengate:~/cs223/orig-class-examples/lec0$ cat /home/alinen/cs223/hello.c
#include <stdio.h>
int main() {
  printf("Hello World\n");
}
```

# Draw the directory hierarchy after the following commands

```
$ pwd
/home/alinen
$ mkdir A
$ cd A
$ mkdir Z
$ touch talk.c
$ cd ..
$ touch listen.c
$ cd
$ touch sing.c
```

# File properties



```
alinen@goldengate:~/cs223/class-examples/lec0$ vi hello.c
alinen@goldengate:~/cs223/class-examples/lec0$ gcc hello.c
alinen@goldengate:~/cs223/class-examples/lec0$ a.out
a.out: command not found
alinen@goldengate:~/cs223/class-examples/lec0$ ./a.out
Hello World
alinen@goldengate:~/cs223/class-examples/lec0$ ls -l
total 40
-rwxr-xr-x 1 alinen faculty 16696 Jan 18 14:42 a.out
-rw-r--r-- 1 alinen faculty    76 Jan 18 14:42 hello.c
-rw-r--r-- 1 alinen faculty   416 Jan 18 13:58 Hello.class
-rw-r--r-- 1 alinen faculty   104 Jan 18 13:58 Hello.java
-rw-r--r-- 1 alinen faculty   934 Jan 18 14:03 Sqrt.class
-rw-r--r-- 1 alinen faculty   197 Jan 18 14:03 Sqrt.java
```

# Your editor and you!

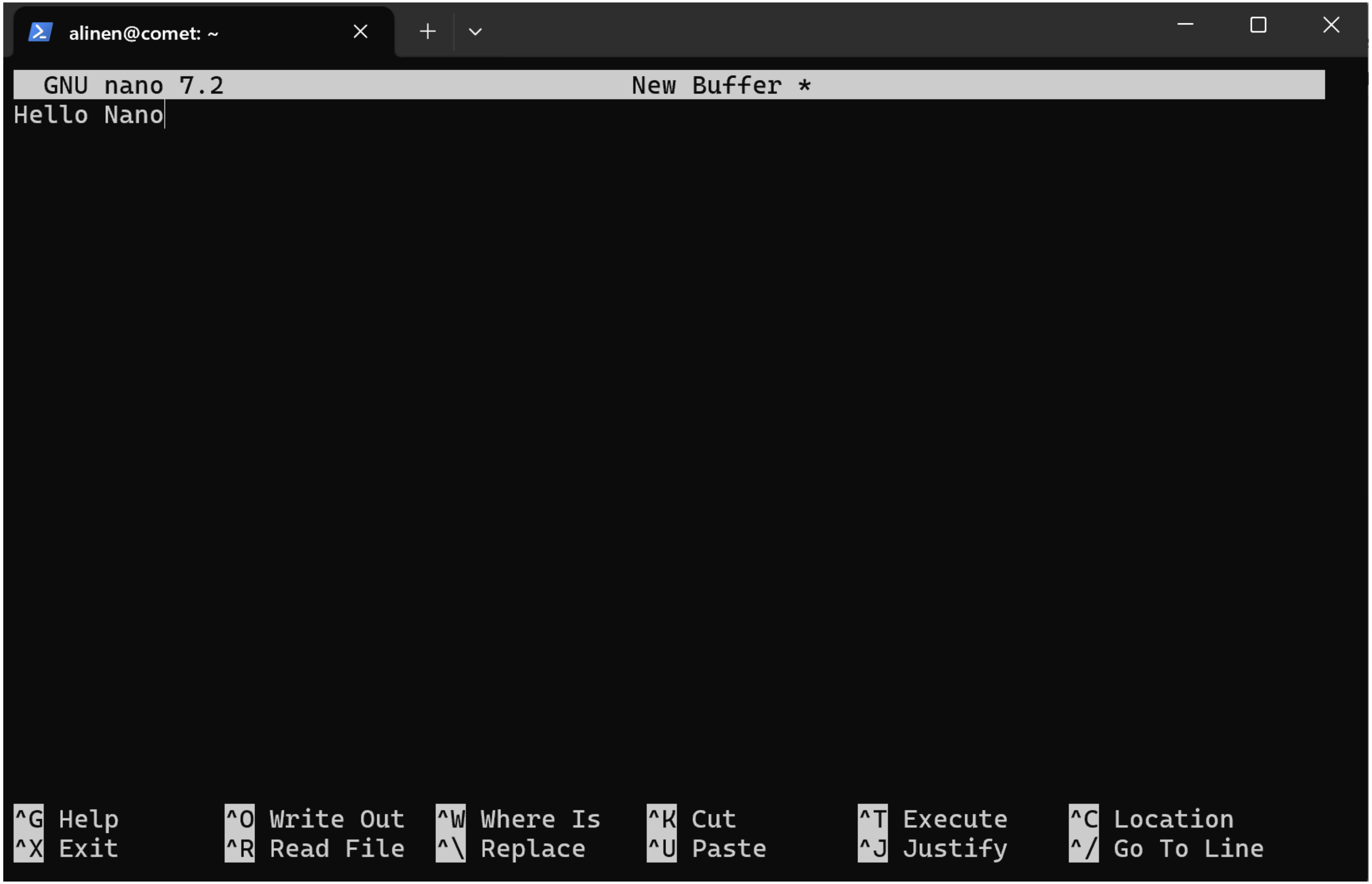
You are encouraged to learn a terminal editor this semester

- Nano
- Emacs
- Vim

Learning a good editor will help you write code faster

You will need to use one of these editors for coding activities in lab

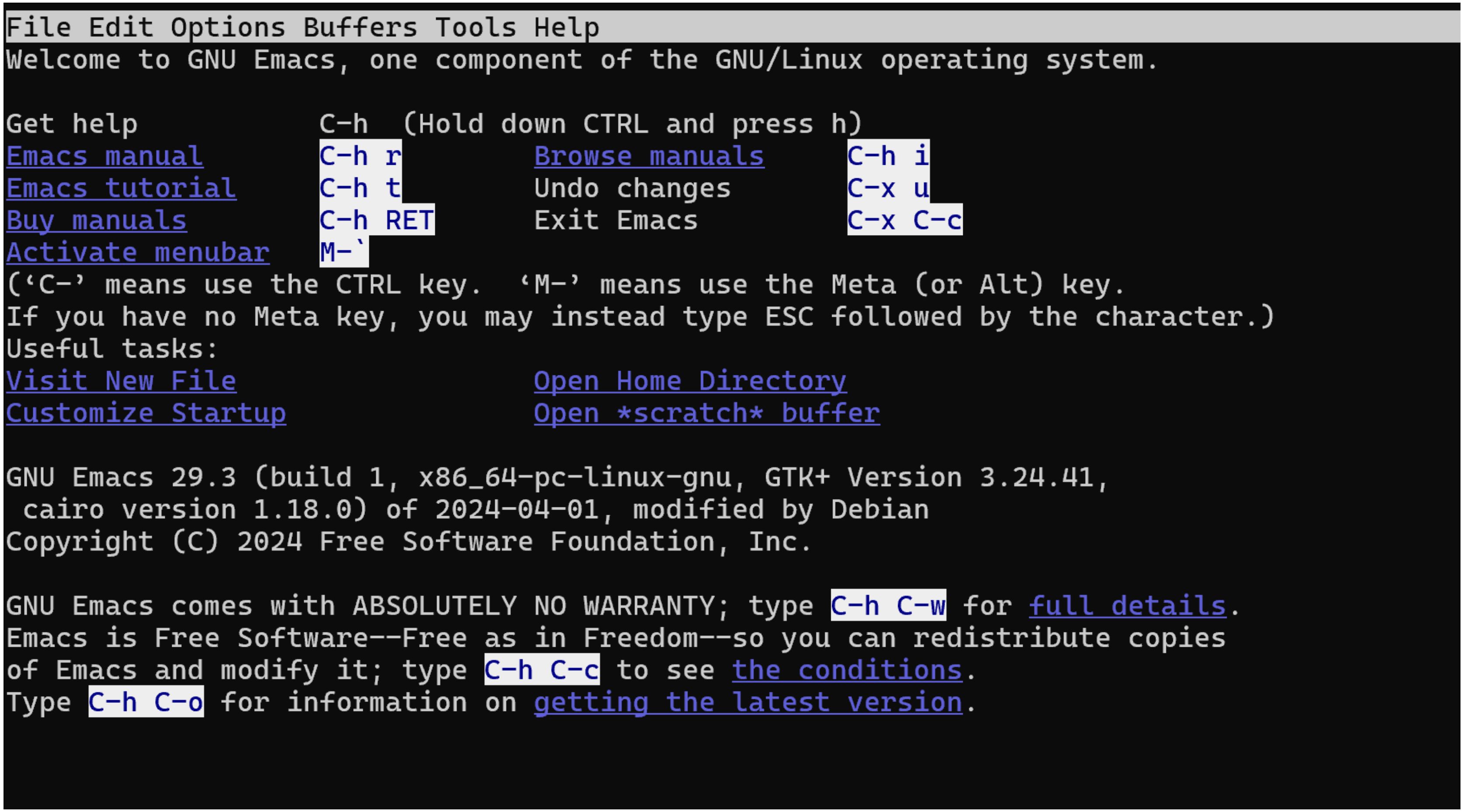
# Nano



```
  GNU nano 7.2                            New Buffer *
Hello Nano
```

```
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line
```

# Emacs



```
File Edit Options Buffers Tools Help
Welcome to GNU Emacs, one component of the GNU/Linux operating system.

Get help            C-h  (Hold down CTRL and press h)
Emacs manual        C-h r         Browse manuals     C-h i
Emacs tutorial      C-h t         Undo changes       C-x u
Buy manuals         C-h RET       Exit Emacs         C-x C-c
Activate menubar    M-`
(‘C-’ means use the CTRL key.  ‘M-’ means use the Meta (or Alt) key.
If you have no Meta key, you may instead type ESC followed by the character.)
Useful tasks:
Visit New File                    Open Home Directory
Customize Startup                 Open *scratch* buffer

GNU Emacs 29.3 (build 1, x86_64-pc-linux-gnu, GTK+ Version 3.24.41,
 cairo version 1.18.0) of 2024-04-01, modified by Debian
Copyright (C) 2024 Free Software Foundation, Inc.

GNU Emacs comes with ABSOLUTELY NO WARRANTY; type C-h C-w for full details.
Emacs is Free Software--Free as in Freedom--so you can redistribute copies
of Emacs and modify it; type C-h C-c to see the conditions.
Type C-h C-o for information on getting the latest version.
```

NOTE: F10 to use the menu

# Vim

- To open: `vi <filename>`
- To quit: Press escape, then `:q!`
- To save: Press escape, then `:w`
- Two modes: **insert** and **command mode**
  - insert mode: type text in the usual way: ‘i’ enters **insert mode** at current cursor position
  - Escape enters **command mode**: search, navigate, copy/paste/delete, etc

# Course Philosophy: Practice!

- Lectures (mandatory): Slides with integrated activities
- Labs (mandatory): Check-ins, hands-on projects
- 2 midterms, oral exam during exam week
- Accommodations: Need at least 2 weeks prior notice to make arrangements

Important: Learn independence, e.g. doing the work yourself

- CS Goals: UNIX, git, terminal editors, C programming, how computing systems work
- Life Goals: Develop skills, strategies, and knowledge for analytical thinking

# My advice

Read the textbook!

Show up: lectures and labs

Do the work:

- ~ 10 hour week commitment (4.5 hrs + 5 hrs)
- Lots of support is available: slack, pre/post class/lab, office hours
- Take hand-written notes
- Practice exams, quizzes and coding activities in ways that mimic the test environments
- Do homework and study with phones LOCKED AWAY IN ANOTHER ROOM and distracting web pages CLOSED.

# My advice

Keep your commitments: 80% of work is consistently showing up

Try your best without beating yourself up. Keep your sense of humor!

Assess your pre-requisite knowledge and fill gaps – allow for more time if necessary

Find community: get to know your classmates and fellow majors. Form a study group. Work together in the labs.

Build habits that are forward leading

- Focus, organization, taking responsibility for your choices
- Resist short-term gains that can sabotage you in the long-term. **Avoid over-committing.**
- Take care of yourself: sleep, exercise, socialize