CSC 252: Computer Organization
Spring 2018: Lecture 1

Instructor: Yuhao Zhu

Department of Computer Science
University of Rochester

Action Items:
• Get CSUG account
• Sign up for Blackboard

Slide credits: Leo Porter, Mattan Erez, Derek Chiou, Keshav Pingali
Outline: Class Introduction

• Introduction
  – What Are You Supposed to Learn in this Class?
    • What Is Computer Systems?
  – Instructor & TAs
  – What Do I Expect From You?
  – How am I Going to Teach?
  – Grading, Policies

• Action items:
  – Get a CSUG account.
    – cycle1.csug.rochester.edu (or cycle2, cycle3)
    – Talk to Brynn Wilkins (bwilkins@cs.rochester.edu) if you don’t already have one
  – Sign up for Blackboard (https://learn.rochester.edu/)
Where to Find Stuff

• http://cs.rochester.edu/courses/252/spring2018/
  - General info
  - Programming assignments details
  - Various course materials
  - Opportunities and “ads”

• Blackboard for all communication
  - Only place for all announcements (e.g., when an assignment is out)
  - Only place for digitized Q&A
  - Except personal issues
  - Don’t waste time: check/search previous posts before posting

• CSUG machines for programming assignments submissions
What is Computer Systems?
What is Computer Systems?

Problem

Who scores the highest on the exam?
What is Computer Systems?

Problem

Who scores the highest on the exam?

Circuit

Electrons, Resistors, Capacitors, etc.
What is Computer Systems?

Problem

Who scores the highest on the exam?

Circuit

Electrons, Resistors, Capacitors, etc.
What is Computer Systems?

Problem

Who scores the highest on the exam?

Quicksort

Circuit

Electrons, Resistors, Capacitors, etc.
What is Computer Systems?

- Problem
- Algorithm
- Circuit

Who scores the highest on the exam?
Quicksort
Electrons, Resistors, Capacitors, etc.
What is Computer Systems?

Problem

Who scores the highest on the exam?

Algorithm

Quicksort

Human-readable language (Java, C)

Circuit

Electrons, Resistors, Capacitors, etc.
What is Computer Systems?

Problem

Algorithm

Program

Who scores the highest on the exam?

Quicksort

Human-readable language (Java, C)

Electrons, Resistors, Capacitors, etc.
What is Computer Systems?

Who scores the highest on the exam?

Problem

Algorithm

Program

Quicksort

Human-readable language (Java, C)

Machine Language

Electrons, Resistors, Capacitors, etc.

Circuit

Problem

Algorithm

Program
What is Computer Systems?

- Problem
- Algorithm
- Program
- Instruction Set
- Architecture

Who scores the highest on the exam?

- Quicksort
- Human-readable language (Java, C)
- Machine Language
- Electrons, Resistors, Capacitors, etc.
What is Computer Systems?

- Problem
- Algorithm
- Program
- Instruction Set
- Architecture
- Circuit

Who scores the highest on the exam?

- Quicksort
- Human-readable language (Java, C)
- Machine Language
- Hardware Design
- Electrons, Resistors, Capacitors, etc.
What is Computer Systems?

Who scores the highest on the exam?

Quicksort

Human-readable language (Java, C)

Machine Language

Hardware Design

Electrons, Resistors, Capacitors, etc.

Problem

Algorithm

Program

Instruction Set Architecture

Microarchitecture

Circuit

Problem

Algorithm

Program

Instruction Set Architecture

Microarchitecture

Circuit
What is Computer Systems?

Problem

Who scores the highest on the exam?

Algorithm

Quicksort

Program

Human-readable language (Java, C)

Instruction Set

Machine Language

Architecture

Hardware Design

Microarchitecture

Electrons, Resistors, Capacitors, etc.

Circuit
Who scores the highest on the exam?

- Quicksort

Program

- Instruction Set
- Architecture

Problem

- Algorithm

Computer Systems Match User Requirements to Hardware Technologies

- Hardware Design
- Electrons, Resistors, Capacitors, etc.
- Machine Language
- Human-readable language (Java, C)

Circuit

- Microarchitecture
## Two Fundamental Aspects of Computer Systems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Who scores the highest on the exam?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Quicksort</td>
</tr>
<tr>
<td>Program</td>
<td>Human-readable language (Java, C)</td>
</tr>
<tr>
<td>Instruction Set Architecture</td>
<td>Machine Language</td>
</tr>
<tr>
<td>Microarchitecture</td>
<td>Hardware Design</td>
</tr>
<tr>
<td>Circuit</td>
<td>Electrons, Resistors, Capacitors, etc.</td>
</tr>
</tbody>
</table>
Two Fundamental Aspects of Computer Systems

- How is a human-readable program translated to a representation that computers can understand?

Problem

---

Algorithm

---

Program

---

Instruction Set

---

Architecture

---

Microarchitecture

---

Circuit

Who scores the highest on the exam?

Quicksort

Human-readable language (Java, C)

Machine Language

Hardware Design

Electrons, Resistors, Capacitors, etc.
Two Fundamental Aspects of Computer Systems

- How is a human-readable program translated to a representation that computers can understand?
- How does a modern computer execute that program?

Problem

Algorithm

Program

Instruction Set Architecture

Microarchitecture

Circuit

Who scores the highest on the exam?

Quicksort

Human-readable language (Java, C)

Machine Language

Hardware Design

Electrons, Resistors, Capacitors, etc.
The “Translation” Process, a.k.a., **Compilation**

- It translates a text file to an executable binary file (a.k.a., executable) consisting of a sequence of **instructions**
- Why binary? Computers understand only 0s and 1s
  - The subject of next lecture
The “Translation” Process, a.k.a., **Compilation**

- It translates a text file to an executable binary file (a.k.a., executable) consisting of a sequence of **instructions**
- Why binary? Computers understand only 0s and 1s
  - The subject of next lecture

**Example: add.c**

(Human-readable)

```c
#include <stdio.h>
int main() {
    int a = 1;
    int b = 2;
    int c = a + b;
    printf(“%d\n”, c);
}
```
The “Translation” Process, a.k.a., Compilation

- It translates a text file to an executable binary file (a.k.a., executable) consisting of a sequence of instructions
- Why binary? Computers understand only 0s and 1s
  - The subject of next lecture

*Example: add.c*
(Human-readable)

```c
#include <stdio.h>
int main() {
  int a = 1;
  int b = 2;
  int c = a + b;
  printf("%d\n", c);
}
```

*Executable Binary*
(Machine-readable)

```
00011001   …
01101010   …
11010101   …
10100100   …
```
The “Translation” Process

Example: add.c

```c
#include <stdio.h>
int main() {
    int a = 1;
    int b = 2;
    int c = a + b;
    printf("%d\n", c);
}
```

Executable Binary

```
00011001   ...
01101010   ...
11010101   ...
10100100   ...
```
The “Translation” Process

Example: add.c

```c
#include <stdio.h>
int main() {
    int a = 1;
    int b = 2;
    int c = a + b;
    printf("%d\n", c);
}
```

Executable Binary

```plaintext
00011001   ...
01101010   ...
11010101   ...
10100100   ...
```
The “Translation” Process

Example: add.c

```c
#include <stdio.h>
int main() {
    int a = 1;
    int b = 2;
    int c = a + b;
    printf("%d\n", c);
}
```
The “Translation” Process

Example: add.c

```c
#include <stdio.h>
int main() {
    int a = 1;
    int b = 2;
    int c = a + b;
    printf("%d\n", c);
}
```
The “Translation” Process

Example: add.c

```c
#include <stdio.h>
int main() {
    int a = 1;
    int b = 2;
    int c = a + b;
    printf("%d\n", c);
}
```

Assembly program: add.s

```assembly
movl $1, -4(%rbp)
movl $2, -8(%rbp)
movl -4(%rbp), %eax
addl -8(%rbp), %eax
...
callq _printf
```
The "Translation" Process

**Assembly program: add.s**

```assembly
movl $1, -4(%rbp)
movl $2, -8(%rbp)
movl -4(%rbp), %eax
addl -8(%rbp), %eax
...  
callq _printf
```
The “Translation” Process

**Source program (text)**

- `add.c`

**Pre-processor (cpp)**

- `add.i`

**Compiler (cc1)**

- `add.s`

**Assembler (as)**

- `add.o`

**Linker (ld)**

- `add`

**Assembly program: add.s**

```
movl $1, -4(%rbp)  
movl $2, -8(%rbp)  
movl -4(%rbp), %eax  
addl -8(%rbp), %eax  
...

callq _printf
```
The “Translation” Process

Assembly program: add.s

```
movl $1, -4(%rbp)
movl $2, -8(%rbp)
movl -4(%rbp), %eax
addl -8(%rbp), %eax
...
callq _printf
```

Relocatable Binary

```
00011001 ...
01101010 ...
11010101 ...
stub_printf
10100100 ...
```
The “Translation” Process

Assembly program: add.s

```
movl $1, -4(%rbp)
movl $2, -8(%rbp)
movl -4(%rbp), %eax
addl -8(%rbp), %eax
...  
callq _printf
```

Relocatable Binary

```
00011001 ...
01101010 ...
11010101 ...
stub _printf
10100100 ...
```
The “Translation” Process

Assembly program: add.s

<table>
<thead>
<tr>
<th>Assembly Code</th>
<th>Relocatable Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>movl $1, -4(%rbp)</td>
<td>00011001 ...</td>
</tr>
<tr>
<td>movl $2, -8(%rbp)</td>
<td>01101010 ...</td>
</tr>
<tr>
<td>movl -4(%rbp), %eax</td>
<td>11010101 ...</td>
</tr>
<tr>
<td>addl -8(%rbp), %eax</td>
<td>stub _printf</td>
</tr>
<tr>
<td>...</td>
<td>10100100 ...</td>
</tr>
<tr>
<td>callq _printf</td>
<td>...</td>
</tr>
</tbody>
</table>
The “Translation” Process

Relocatable Binary

00011001   …
01101010   …
11010101   …
stub _printf
10100100   …
The “Translation” Process

Relocatable Binary

00011001 …
01101010 …
11010101 …
stub _printf
10100100 …
The “Translation” Process

Relocatable Binary

00011001   ...
01101010   ...
11010101   ...
stub _printf
10100100   ...

Relocatable Binary for printf.o
The “Translation” Process

Relocatable Binary

```
00011001 ...
01101010 ...
11010101 ...
stub _printf
10100100 ...
```

Relocatable Binary for `printf.o`
The “Translation” Process

Relocatable Binary

00011001 ... 
01101010 ... 
11010101 ... 
stub _printf 
10100100 ... 

Relocatable Binary for printf.o
The “Translation” Process

Source program (text)

Pre-processor (cpp)

Compiler (cc1)

Assembler (as)

Linker (ld)

Executable Binary

Relocatable Binary

00011001 ...
01101010 ...
11010101 ...
stub _printf
10100100 ...

00011001 ...
01101010 ...
11010101 ...
01110001 ...
01101010 ...
10100100 ...

Relocatable Binary for printf.o
The “Translation” Process

Executable Binary

Relocatable Binary

Questions?

Source program (text) → Pre-processor (cpp) → Modified source program (text) → Compiler (cc1) → Assembly program (text) → Assembler (as) → Relocatable object programs (binary) → Linker (ld) → Executable object program (binary)

Relocatable Binary for printf.o

00011001 ...
01101010 ...
11010101 ...
stub _printf
10100100 ...

Executable Binary

00011001 ...
01101010 ...
11010101 ...
01110001 ...
01101010 ...
10100100 ...
Back to Layers of Transformation…

How is a human-readable program translated to a representation that computers can understand?

How does a modern computer execute that program?

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>Program</th>
<th>Instruction Set</th>
<th>Architecture</th>
<th>Microarchitecture</th>
<th>Circuit</th>
</tr>
</thead>
</table>

Who scores the highest on the exam?

- Quicksort
- Human-readable language (Java, C)
- Machine Language
- Hardware Design
- Electrons, Resistors, Capacitors, etc.
Back to Layers of Transformation...

How is a human-readable program translated to a representation that computers can understand?

Who scores the highest on the exam?

Quicksort

Human-readable language (Java, C)

Machine Language

Hardware Design

Electrons, Resistors, Capacitors, etc.

How is a human-readable program translated to a representation that computers can understand?

How does a modern computer execute that program?
The Fundamental Idea of Computers

- Executables (i.e., instructions) are stored in “memory”
- Processors read instructions from memory and execute instructions one after another

Assembly program: add.s

```
movl $1, -4(%rbp)
movl $2, -8(%rbp)
movl -4(%rbp), %eax
addl -8(%rbp), %eax
...
callq _printf
```
The Fundamental Idea of Computers

Assembly program: add.s

movl $1, -4(%rbp)
movl $2, -8(%rbp)
movl -4(%rbp), %eax
addl -8(%rbp), %eax
...
callq _printf
The Fundamental Idea of Computers

![Diagram showing memory and instruction]

- Assembly program:
  - `add.s`
  - `movl $1, -4(%rbp)`
  - `movl $2, -8(%rbp)`
  - `movl -4(%rbp), %eax`
  - `addl -8(%rbp), %eax`
The Fundamental Idea of Computers

Processor

Memory

Instruction
The Fundamental Idea of Computers

Processor

Program counter (PC)

Memory

Instruction
High-level Organization of Computer Hardware a.k.a., The Von Neumann Model

Processor

Program counter (PC)

Memory

Instruction
High-level Organization of Computer Hardware
a.k.a., The Von Neumann Model

![Diagram showing the high-level organization of computer hardware.]

- Processor
  - Program counter (PC)
  - Arithmetic logic unit (ALU)

- Memory
  - Instruction
High-level Organization of Computer Hardware a.k.a., The Von Neumann Model
High-level Organization of Computer Hardware a.k.a., The Von Neumann Model

Processor
- Program counter (PC)
- Arithmetic logic unit (ALU)
- Control Path

Memory
- Instruction
- Data
High-level Organization of Computer Hardware a.k.a., The Von Neumann Model

- Processor
  - Program counter (PC)
  - Arithmetic logic unit (ALU)
  - Control Path

- Memory
  - Instruction
  - Data

- Input / Output Device
  - Touchscreen
  - Disk
  - Sensor
  - Ethernet
  - Camera

Bus
High-level Organization of Computer Hardware
a.k.a., The Von Neumann Model

Processor
- Program counter (PC)
- Arithmetic logic unit (ALU)

Control Path

Memory
- Instruction
- Data

Bus

Input / Output Device
- Touchscreen
- Disk
- Sensor
- Ethernet
- Camera
High-level Organization of Computer Hardware a.k.a., The Von Neumann Model

Processor:
- Program counter (PC)
- Arithmetic logic unit (ALU)
- Control Path

Memory:
- Instruction
- Data

Input / Output Device:
- Touchscreen
- Disk
- Sensor
- Ethernet
- Camera

I/O Bus
High-level Organization of Computer Hardware

a.k.a., The Von Neumann Model

**Processor**
- Program counter (PC)
- Arithmetic logic unit (ALU)

**Memory**
- Instruction
- Data

**Input / Output Device**
- Touchscreen
- Disk
- Sensor
- Ethernet
- Camera

**I/O Bus**

Diagram: Processor connected to Memory via I/O Bus, Processor and I/O Device connected by Control Path.
High-level Organization of Computer Hardware a.k.a., The Von Neumann Model

Processor
- Program counter (PC)
- Arithmetic logic unit (ALU)

Memory
- Instruction
- Data

I/O Bridge

Control Path

I/O Bus

Input / Output Device
- Touchscreen
- Disk
- Sensor
- Ethernet
- Camera
High-level Organization of Computer Hardware a.k.a., The Von Neumann Model

- Processor
- Memory
- Control Path
- I/O Bus
- I/O Bridge
- Instruction
- Data
- Input / Output Device
  - Touchscreen
  - Disk
  - Sensor
  - Ethernet
  - Camera
High-level Organization of Computer Hardware a.k.a., The Von Neumann Model

- Processor
- Memory
- Program counter (PC)
- Input / Output Device
- Instruction
- Data
- Arithmetic logic unit (ALU)
- Control Path
- Touchscreen
- Sensor
- Disk
- Ethernet
- Camera
- I/O Bus
- Memory
  - Instruction
  - Data
- I/O Bridge
High-level Organization of Computer Hardware
a.k.a., The Von Neumann Model

Processor
Memory
Program counter (PC)
Input / Output Device
Instruction Data
Arithmetic logic unit (ALU)
Control Path

I/O Bridge
I/O Bus

Input / Output Device
Touchscreen
Disk
Sensor
Ethernet
Camera

Memory
Back to Layers of Transformation…

How is a human-readable program translated to a representation that computers can understand?

How does a modern computer execute that program?

Problem

Algorithm

Program

Instruction Set

Architecture

Microarchitecture

Circuit

Who scores the highest on the exam?

Quicksort

Human-readable language (Java, C)

Machine Language

Hardware Design

Electrons, Resistors, Capacitors, etc.
Instruction Set Architecture
Instruction Set Architecture

• The assembly language’s view of the computer is called the “instruction set architecture” (ISA)

Assembly program: add.s

```
movl $1, -4(%rbp)
movl $2, -8(%rbp)
movl -4(%rbp), %eax
addl -8(%rbp), %eax
...
callq _printf
```
Instruction Set Architecture

- The assembly language’s view of the computer is called the “instruction set architecture” (ISA)
- No need to care how the instructions are implemented as long as they are somehow implemented

Assembly program: add.s

```
movl $1, -4(%rbp)
movl $2, -8(%rbp)
movl -4(%rbp), %eax
addl -8(%rbp), %eax
...
callq _printf
```
Instruction Set Architecture

- The assembly language’s view of the computer is called the “instruction set architecture” (ISA)
- No need to care how the instructions are implemented as long as they are somehow implemented
- Implementation of an ISA is called microarchitecture

Assembly program: add.s

```
movl $1, -4(%rbp)
movl $2, -8(%rbp)
movl -4(%rbp), %eax
addl -8(%rbp), %eax
...
callq _printf
```
Instruction Set Architecture

• The assembly language’s view of the computer is called the “instruction set architecture” (ISA)

• No need to care how the instructions are implemented as long as they are somehow implemented

• Implementation of an ISA is called microarchitecture

• ISAs abstract away details of microarchitecture
Abstraction
Abstraction

• Think of car versus engine, transmission, brakes, …
Abstraction

- Think of car versus engine, transmission, brakes, …
- Bridges of Konigsberg
  - Is there a walk that crosses each bridge exactly once?
Abstraction

• Think of car versus engine, transmission, brakes, …
• Bridges of Konigsberg
  • Is there a walk that crosses each bridge exactly once?
• Solution by Euler
  • Key insight: connectivity between land masses is what is important, not the actual distances or the orientations of the bridges
Abstraction

• **Create an abstraction**
  • One node for each land mass
  • Edge between two nodes if there is a bridge connecting the two land masses

• Graph has nodes of odd degree, so there is no walk with desired property.

• Led to field we now call topology.
Abstraction
Abstraction

• The act or process of leaving out of consideration one or more properties of a complex object so as to focus on others
  • Euler left out distances and orientations
  • ISA leaves out how “ADD” is implemented
  • ISA also leaves our how long an “ADD” instruction takes
Abstraction

• The act or process of leaving out of consideration one or more properties of a complex object so as to focus on others
  • Euler left out distances and orientations
  • ISA leaves out how “ADD” is implemented
  • ISA also leaves our how long an “ADD” instruction takes

• Bad abstractions throw away essential features of problem
  • Topologist is someone who does not know the difference between a doughnut and coffee-cup
  • Bad ISAs don’t tell you the hardware can do multiplication
Every Layer in CS is an Abstraction

Problem

Algorithm

Program

Instruction Set Architecture

Microarchitecture

Circuit

Who scores the highest on the exam?

Quicksort

Human-readable language (Java, C)

Machine Language

Hardware Design

Electrons, Resistors, Capacitors, etc.
Every Layer in CS is an Abstraction

- Depend on which layer you want to live at, you have different views of the computer

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Instruction Set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Architecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Microarchitecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Circuit</td>
</tr>
</tbody>
</table>

Who scores the highest on the exam?

- Quicksort
- Human-readable language (Java, C)
- Machine Language
- Hardware Design
- Electrons, Resistors, Capacitors, etc.
Every Layer in CS is an Abstraction

- Depend on which layer you want to live at, you have different views of the computer
- This course expands your layers of abstractions

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Quicksort</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Human-readable language (Java, C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Machine Language</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hardware Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Electrons, Resistors, Capacitors, etc.</td>
</tr>
</tbody>
</table>

Who scores the highest on the exam?
Instruction Set Architecture
Instruction Set Architecture

• There used to be many ISAs
  – x86, ARM, Power/PowerPC, Sparc, MIPS, IA64, z
  – Very consolidated today: ARM for mobile, x86 for others
Instruction Set Architecture

• There used to be many ISAs
  – x86, ARM, Power/PowerPC, Sparc, MIPS, IA64, z
  – Very consolidated today: ARM for mobile, x86 for others

• There are even more microarchitectures
  – Apple/Samsung/Qualcomm have their own microarchitecture (implementation) of the ARM ISA
  – Intel and AMD have different microarchitectures for x86
Instruction Set Architecture

• There used to be many ISAs
  – x86, ARM, PowerPC, Sparc, MIPS, IA64, z
  – Very consolidated today: ARM for mobile, x86 for others

• There are more microarchitectures
  – Apple/Samsung/Qualcomm have their own microarchitecture (implementation) of the ARM ISA
  – Intel and AMD have different microarchitectures for x86
Instruction Set Architecture

• There used to be many ISAs
  – x86, ARM, Power/PowerPC, Sparc, MIPS, IA64, z
  – Very consolidated today: ARM for mobile, x86 for others

• There are even more microarchitectures
  – Apple/Samsung/Qualcomm have their own microarchitecture (implementation) of the ARM ISA
  – Intel and AMD have different microarchitectures for x86

• ISA is lucrative business: ARM’s Business Model
  – Patent the ISA, and then license the ISA
  – Every implemementer pays a royalty to ARM
  – Apple/Samsung pays ARM whenever they sell a smartphone
Instruction Set Architecture

• Little research on ISA, much more microarch. research
  – ISA is stable now. “One ISA rules them all.”
  – Free, open ISA: RISC V (UC Berkeley, https://riscv.org/)
  – Instead, focus on optimizing the implementation.
Instruction Set Architecture

- Little research on ISA, much more microarch. research
  - ISA is stable now. “One ISA rules them all.”
  - Free, open ISA: RISC V (UC Berkeley, https://riscv.org/)
    - Instead, focus on optimizing the implementation.

- Interesting question: can we have one microarchitecture (implementation) for different ISAs?
Instruction Set Architecture

- Little research on ISA, much more microarch. research
  - ISA is stable now. “One ISA rules them all.”
  - Free, open ISA: RISC V (UC Berkeley, https://riscv.org/)
  - Instead, focus on optimizing the implementation.

- Interesting question: can we have one microarchitecture (implementation) for different ISAs?
  - Can a microarchitecture designed for ISA $X$ execute ISA $Y$?
Instruction Set Architecture

• Little research on ISA, much more microarch. research
  – ISA is stable now. “One ISA rules them all.”
  – Free, open ISA: RISC V (UC Berkeley, https://riscv.org/)
  – Instead, focus on optimizing the implementation.

• Interesting question: can we have one microarchitecture (implementation) for different ISAs?
  – Can a microarchitecture designed for ISA X execute ISA Y?
  – Yes but you need something that translates programs written in ISA Y to ISA X while you are executing it: *dynamic binary translator*
Instruction Set Architecture

• Little research on ISA, much more microarch. research
  – ISA is stable now. “One ISA rules them all.”
  – Free, open ISA: RISC V (UC Berkeley, https://riscv.org/)

  – Instead, focus on optimizing the implementation.

• Interesting question: can we have one microarchitecture (implementation) for different ISAs?
  – Can a microarchitecture designed for ISA $X$ execute ISA $Y$?
  – Yes but you need something that translates programs written in ISA $Y$ to ISA $X$ while you are executing it: *dynamic binary translator*
  – E.g., Transmeta executes x86 ISA programs on their in-house ISA
Instruction Set Architecture

• Little research on ISA, much more microarch. research
  – ISA is stable now. “One ISA rules them all.”
  – Free, open ISA: RISC V (UC Berkeley, https://riscv.org/)

  – Instead, focus on optimizing the implementation.

• Interesting question: can we have one microarchitecture (implementation) for different ISAs?
  – Can a microarchitecture designed for ISA X execute ISA Y?
  – Yes but you need something that translates programs written in ISA Y to ISA X while you are executing it: dynamic binary translator
  – E.g., Transmeta executes x86 ISA programs on their in-house ISA
  – Think opportunities, not overhead
The Role of a Computer System Designer

Problem

Algorithm

Program

Instruction Set
Architectural

Microarchitecture

Circuit
The Role of a Computer System Designer

- **Look Up** (Nature of the problems)

![Diagram of the role of a computer system designer]

- Problem
- Algorithm
- Program
  - Instruction Set
  - Architecture
  - Microarchitecture
  - Circuit
The Role of a Computer System Designer

- **Look Up** (Nature of the problems)
- **Look Down** (Nature of the circuit technology and physics)
The Role of a Computer System Designer

- **Look Up** (Nature of the problems)
- **Look Down** (Nature of the circuit technology and physics)
- **Look Backward** (Evaluating old ideas in light of new technologies)
The Role of a Computer System Designer

- **Look Up** (Nature of the problems)
- **Look Down** (Nature of the circuit technology and physics)
- **Look Backward** (Evaluating old ideas in light of new technologies)
- **Look Forward** (Listen to dreamers and predict the future)
Why This Course Matters to You

• It’s just a lot of fun. You get to look up and down, forward and backward.

• Without understanding Computer Systems you may write grossly inefficient code and have no idea why (this could easily cost you a job OR, in contrast, if you know why - get you a promotion).

• Could be your favorite subject in CS, or the area you want to do research in.
Why This Course Matters to You

• It’s just a lot of fun. You get to look up and down, forward and backward.

• Without understanding Computer Systems you may write grossly inefficient code and have no idea why (this could easily cost you a job OR, in contrast, if you know why - get you a promotion).

• Could be your favorite subject in CS, or the area you want to do research in.

Questions?
Who Are We?
Who Are We?

• Myself: Yuhao Zhu
  – WH 3501, yzhu@rochester.edu
  – Office hours TBD (soon) or by appointment
  – Got a good education
  – Got some industry experience
  – Researching computer systems, especially mobile systems, for emerging visual applications: Augmented/Virtual Reality, Computational Imaging, etc.
Who Are We?

• Myself: Yuhao Zhu
  – WH 3501, yzhu@rochester.edu
  – Office hours TBD (soon) or by appointment
  – Got a good education
  – Got some industry experience
  – Researching computer systems, especially mobile systems, for emerging visual applications: Augmented/Virtual Reality, Computational Imaging, etc.

• TAs: 3 Grads + 5 UGs
  – Alan, Sayak, Michael, Alan, Akshay, Benjamin, Eric, Jie
  – Really care about you learning the material and succeeding
Who Are We?

• Myself: Yuhao Zhu
  – WH 3501, yzhu@rochester.edu
  – Office hours TBD (soon) or by appointment
  – Got a good education
  – Got some industry experience
  – Researching computer systems, especially mobile systems, for emerging visual applications: Augmented/Virtual Reality, Computational Imaging, etc.

• TAs: 3 Grads + 5 UGs
  – Alan, Sayak, Michael, Alan, Akshay, Benjamin, Eric, Jie
  – Really care about you learning the material and succeeding

• Coming to office hours does NOT mean you are weak!
What Do I Expect From You?

• *Learn the material*
• *Do the work*
  • If you’re going to come to class, come on time
    • I do not take attendance and you will not be graded on it
    • However, all of the top students from previous years regularly attended class
  • Big believer in communicating
    • Probably the most important thing to know
      • Speak up, ask questions, participate in class
  • You can eat, but please by quiet
  • There will be a minimum standard to pass the class
• This class will definitely be tough
What Should You Expect From Me?

- **Think of me as your tutor**
  - Be your guide in inducing you to explore concepts
  - Create situations and post problems that set the scene for your exploration
  - Answer your questions

- Not spend lecture reading the textbook to you with slightly different words
A Word about Lectures and Medieval Times

• Lecture: It’s a large part of what you pay for

• But why do we have the “lecture” format?
  – Why does someone stand at the front and tell you things?
  – Why do you take “notes” on what they say?
A Word about Lectures and Medieval Times

• Lecture: It’s a large part of what you pay for

• But why do we have the “lecture” format?
  – Why does someone stand at the front and tell you things?
  – Why do you take “notes” on what they say?

Because that’s how the teacher learned, and they learned that way...
All The Way Back to Medieval Times..
All The Way Back to Medieval Times..
All The Way Back to Medieval Times..

Lecture Halls

people
Modern Times

• You don’t have to trust the monk!
  – The printing press: a revolutionizing development
  – The web: order your knowledge up for yourself on Amazon!

• Read books and analyze for YOURSELF!
  – If I rephrase it for you, what purpose does that serve?
FAQ: "But professor, wouldn't it be more efficient if you just taught us with the right answer to begin with?"
FAQ: "But professor, wouldn't it be more efficient if you just taught us with the right answer to begin with?"

- Have you ever heard of a workout class where the instructor did all the exercises while everyone else just watched attentively?
FAQ: "But professor, wouldn't it be more efficient if you just taught us with the right answer to begin with?"

- Have you ever heard of a workout class where the instructor did all the exercises while everyone else just watched attentively?

- To learn, you must do the work with your own muscle (brain)!
How Will You Be Evaluated?
How Will You Be Evaluated?

- Programming Assignments: 54%
  - 6 assignments, 9% each
  - *This is where you really learn*
  - Each assignment has two deadlines
  - One for the “pre-assignment” (a.k.a. trivia), whose point is to get you start thinking about the assignment (1.5%)
  - The other for the main assignment (7.5%)
  - By just thinking about the lab, you get 1/6 of the points! So do the trivia!
How Will You Be Evaluated?

- Programming Assignments: 54%
  - 6 assignments, 9% each
  - This is where you really learn
  - Each assignment has two deadlines
  - One for the “pre-assignment” (a.k.a. trivia), whose point is to get you start thinking about the assignment (1.5%)
  - The other for the main assignment (7.5%)
  - By just thinking about the lab, you get 1/6 of the points! So do the trivia!
- 1 midterm exam, 16%
How Will You Be Evaluated?

- Programming Assignments: 54%
  - 6 assignments, 9% each
  - This is where you really learn
  - Each assignment has two deadlines
  - One for the “pre-assignment” (a.k.a. trivia), whose point is to get you start thinking about the assignment (1.5%)
  - The other for the main assignment (7.5%)
  - By just thinking about the lab, you get 1/6 of the points! So do the trivia!
- 1 midterm exam, 16%
- 1 comprehensive final exam, 30%
How Will You Be Evaluated?

• Programming Assignments: 54%
  – 6 assignments, 9% each
  – This is where you really learn
  – Each assignment has two deadlines
  – One for the “pre-assignment” (a.k.a. trivia), whose point is to get you start thinking about the assignment (1.5%)
  – The other for the main assignment (7.5%)
  – By just thinking about the lab, you get 1/6 of the points! So do the trivia!
• 1 midterm exam, 16%
• 1 comprehensive final exam, 30%
• Observations:
  – Programming assignments have the most weight because that’s where you really learn and will benefit the most
  – Final is almost twice as important as the midterm because we care about what you know in the end
  – All three parts are important. You can’t pass the class while focusing only on one part.
Programming Assignments
Programming Assignments

- Check syllabus and labs to figure out when they are due (there is a date and a time specified)
  - Days you have for each lab: 10, 15, 15, 13, 13, 13
Programming Assignments

• Check syllabus and labs to figure out when they are due (there is a date and a time specified)
  • Days you have for each lab: 10, 15, 15, 13, 13, 13

• They take time, so start early!
  – Thinking/design time
  – Programming time
  – Test design + debug (and repeat)
Programming Assignments

• Check syllabus and labs to figure out when they are due (there is a date and a time specified)
  • Days you have for each lab: 10, 15, 15, 13, 13, 13
• They take time, so start early!
  – Thinking/design time
  – Programming time
  – Test design + debug (and repeat)
• 3 slip days. Use it wisely!
  • Other than slip days, late submission counts 0 point
Programming Assignments

• Check syllabus and labs to figure out when they are due (there is a date and a time specified)
  • Days you have for each lab: 10, 15, 15, 13, 13, 13
• They take time, so start early!
  – Thinking/design time
  – Programming time
  – Test design + debug (and repeat)
• 3 slip days. Use it wisely!
  • Other than slip days, late submission counts 0 point
• You could work in pairs
  – Only 1 submission per pair
Programming Assignments

- Check syllabus and labs to figure out when they are due (there is a date and a time specified)
  - Days you have for each lab: 10, 15, 15, 13, 13, 13
- They take time, so start early!
  - Thinking/design time
  - Programming time
  - Test design + debug (and repeat)
- **3 slip days.** Use it wisely!
  - Other than slip days, late submission counts 0 point
- You could work in pairs
  - Only 1 submission per pair
- Share ideas but not artifacts (e.g., code, sketch)
Programming Environment

• Develop code (or at least test it) on the CSUG Linux boxes (csug.rochester.edu)
  – Microsoft Visual Studio could be nice, but it’s not what we use
  – The lack of Unix knowledge is a major problem according to our industry contacts

• Projects will be mostly in C and x86 assembler.

• We only accept ANSI-C that can be compiled by the default GCC on the CSUG Linux boxes
Exams
Exams

- Two exams: one in-class midterm and one final
  - Midterm covers everything up until the second last lecture
  - Finally will cover everything, including materials before midterm
Exams

• Two exams: one in-class midterm and one final
  – Midterm covers everything up until the second last lecture
  – Finally will cover everything, including materials before midterm

• “I don’t know” is given 15% partial credit
  – You need to decide if guessing is worthwhile
  – Saves grading time
  – You have to write “I don’t know” and cross out /erase anything else to get credit: A blank answer doesn’t count
Exams

• Two exams: one in-class midterm and one final
  – Midterm covers everything up until the second last lecture
  – Finally will cover everything, including materials before midterm

• “I don’t know” is given 15% partial credit
  – You need to decide if guessing is worthwhile
  – Saves grading time
  – You have to write “I don’t know” and cross out /erase anything else to get credit: A blank answer doesn’t count

• All exams are open book (means your book won’t help)
  – They will in fact probably hurt
Exams

- **Two exams: one in-class midterm and one final**
  - Midterm covers everything up until the second last lecture
  - Finally will cover everything, including materials before midterm
- **“I don’t know” is given 15% partial credit**
  - You need to decide if guessing is worthwhile
  - Saves grading time
  - You have to write “I don’t know” and cross out /erase anything else to get credit: A blank answer doesn’t count
- **All exams are open book (means your book won’t help)**
  - They will in fact probably hurt
- **No collaboration on exams**
Textbook

• Required textbook
  • Bryant and O'Hallaron's *Computer Systems: A Programmer's Perspective* (3rd edition)

• Some recommended (but not required) textbooks
  – *Introduction to Computing Systems: From Bits and Gates to C and Beyond*, 2/e. This is where I learnt Computer Systems.
  
  – *Structured Computer Organization*, 6/e. More emphasis on SW.
  
Where to Find Stuff

• http://cs.rochester.edu/courses/252/spring2018/
  – General info
  – Programming assignments details
  – Various course materials
  – Opportunities and “ads”

• Blackboard for all communication
  – Only place for all announcements (e.g., when an assignment is out)
  – Only place for digitized Q&A
  – Except personal issues
  – Don’t waste time: check/search previous posts before posting

• CSUG machines for programming assignments submissions