CS 271: Computer Architecture and Assembly Language
Winter 2013

January 7, 2013
Introduce yourself!

On a piece of paper . . .

- Name, year
- CS/EE classes you’ve taken
- Are these office hours OK?
  - Mon Noon–2pm, Wed 4pm—5pm
  - Tues 9am–1am, Thurs 4pm–5pm
- Experience programming in assembly?
  If so, for what architecture?
- What do you hope to learn in this class?

Turn in to me before you leave!

(I’ll give you some time at the end of class)
Course logistics

What is this course about?
  What is the scope?
  What is a computer architecture?
  What is an assembly language?

Learning objectives
## Contact info and office hours

<table>
<thead>
<tr>
<th>Instructor</th>
<th>Eric Walkingshaw</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="mailto:walkiner@eecs.oregonstate.edu">walkiner@eecs.oregonstate.edu</a></td>
</tr>
<tr>
<td>Office hours</td>
<td>Mon: Noon – 2pm</td>
</tr>
<tr>
<td>(KEC 3093)</td>
<td>Wed: 4pm – 5pm</td>
</tr>
<tr>
<td></td>
<td><strong>or by appointment</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Teaching Asst.</th>
<th>Yaofei Feng</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="mailto:fengy@engr.oregonstate.edu">fengy@engr.oregonstate.edu</a></td>
</tr>
<tr>
<td>Office hours</td>
<td>Tues: 9am – 11am</td>
</tr>
<tr>
<td>(KEC Atrium)</td>
<td>Thurs: 4pm – 5pm</td>
</tr>
</tbody>
</table>
# Course details

<table>
<thead>
<tr>
<th>Lectures</th>
<th>Strand Agriculture Hall 203</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MWF 3:00–3:50 pm</td>
</tr>
</tbody>
</table>

| Mailing list | cs271-w13@engr.orst.edu       |

| Web page     | eecs.oregonstate.edu/~walkiner/cs271-w13/ |
Materials, tests, and coursework

- No textbook!
- Slides and links will be posted to the course web page (possibly some required reading)

Estimated grading breakdown

- 10% – written homework
- 30% – programming assignments
- 30% – midterms (2 × 15% each)
- 30% – final exam

Subject to change! Check the class web page
Academic honesty

For written homework and programming assignments:

- **Discussion is encouraged!**
- Each student should submit their own final work
- Should understand and be able to reproduce your answers
- Goal is to **learn** the material

If you work with other students, **list them** on your submission!
Important dates

No class!
Jan 21 – MLK Jr. Day

I’m out of town
Feb 25 – Mar 1  (More details when we get closer.)

Final exam
Tues, Mar 19, Noon–2pm

Check the class web page regularly!
Outline

Course logistics

What is this course about?
  What is the scope?
  What is a computer architecture?
  What is an assembly language?

Learning objectives
Levels of abstraction from computer hardware

**Natural language**
- English, Spanish, Chinese

**Declarative programming language**
- Haskell, Prolog, MySQL

**Imperative programming language**
- C, Java, Python, Javascript

**Assembly language**
- GAS, MASM, MIPS assembly

**Machine code**
- x86 instructions, MIPS instructions

\{ my research \}

\{ this class \}
## Moving down the hierarchy

### Natural language
- Used by humans, ambiguous semantics
- Translated to programming language by a **programmer**

### Programming language
- Well-defined syntax/semantics, portable to different architectures
- Translated to assembly by a **compiler**

### Assembly language
- Mnemonic instructions for a specific architecture
- Translated to machine code by an **assembler**

### Machine code
- Binary instructions for a specific architecture
What is a computer architecture?

One view: The machine language the CPU implements

Instruction set architecture (ISA)

- Built in data types (integers, floating point numbers)
- Fixed set of instructions
- Fixed set of on-processor variables (registers)
- Interface for reading/writing memory
- Mechanisms to do input/output
What is a computer architecture?

Another view: The implementation of the CPU in hardware

Microarchitecture – implements the ISA
In this course . . .

MIPS architecture

- RISC architecture – reduced instruction set computer
  - vs. CISC – *complex* instruction set computer
- Very widely used in embedded systems

We’ll study:
- the ISA in gory detail
- the microarchitecture at a higher level
What is an assembly language?

A **programming interface** to the ISA

An assembly language provides:

- A set of **mnemonics** for machine instructions
  - Opcodes, register names, addressing modes
- A way to **name** memory addresses and constants
- Other conveniences for generating machine code
What is an assembler?

An **assembler** is software that translates assembly code to machine code.
Assembly vs. programming languages

Why use assembly?
- Easier than writing machine code!
- Provides direct control of hardware components
  - Access to features not exposed in a higher-level language
- Performance (dubious)
- A good way to learn a computer architecture :)

Common uses of assembly
- Embedded systems – size/speed efficiency
- Device drivers – direct control
Outline

Course logistics

What is this course about?
  What is the scope?
  What is a computer architecture?
  What is an assembly language?

Learning objectives
What should you learn in this class?

1. Understand how data is represented in computers.
   - Programs, integers, and floating point numbers.
   - Big-endian vs. little-endian.
   - Binary, hex, and decimal number systems.
   - Parity bits, error-correcting codes.

2. High-level understanding of a computer architecture.
   - What are the major components?
   - Instruction execution cycle and pipelining.
   - Relationship of assembly to an instruction set architecture.
   - Role of the operating system.
What should you learn in this class?

3. Understand exactly what an assembler does.
   • Translation from assembly instructions to machine code.
   • Operation and register mnemonics.
   • Replacing labels with offsets.
   • Expansion of macro instructions.

4. Experience programming in an assembly language.
   • Instruction formats and register conventions.
   • Implementing branches, loops, and procedure calls.
   • Interacting with the operating system through system calls.

5. Understand the mechanics of procedure calls.
   • Simulate the system stack in assembly language.
   • Return values and parameter passing.
   • Alternative procedure call mechanics.