CS 271
Computer Architecture & Assembly Language

Lecture 1
Introduction and Course Syllabus
1/4/22, Tuesday
Lecture Topics:

• Syllabus
• Introduction to Hardware, Software, and Languages
• Setup Instructions
About Me

• 8\textsuperscript{th} year at OSU, got my Bachelor degree in Spring 2018, and Master Degree in Fall 2020
• Became a full-time instructor since Winter 2021☺
• Involved in First year CS program since 2017
  • TA 2017-2018
  • GTA 2018-2020

• Taught CS 161 in Fall 2019 and 2020, Winter 2021, and Spring 2021
• Taught CS 162 in Spring and Summer 2020, and Fall 2021
• Taught CS 271 in Winter 2021
• Taught CS 372 in Summer and Fall 2021
• Taught CS 444/544 in Spring 2021
Syllabus

IT’S IN THE SYLLABUS

This message brought to you by every instructor that ever lived.

WWW.PHDCOMICS.COM
Course Information

• **Canvas site:**
  - All course materials
  - Code submission (as .asm)
  - Must score 100% on syllabus quiz to unlock the rest

• **Discord:**
  - Online discussion forum

• **Textbook:**
  - Irvine, Kip R., *Assembly Language for x86 Processors* (8th ed.)
  - You may access the 7th edition [here](#)
Basics

• Instructor: Yipeng (Roger) Song
  • I go by Roger 😊

• Email
  • Instructor: songyip@oregonstate.edu
  • TAs: cs271-ta@engr.orst.edu (TAs and me)

• Office Hours: TBD

• Requirements: Laptop

• Programming Language: Assembly (MASM)
DOG_FACTOR = 7

.data
userName BYTE 33 DUP(0) ; string to be entered by user
userAge DWORD ? ; integer to be entered by user
intro_1 BYTE "Hi, my name is Lassie, and I'm here to tell you your age in dog years!", 0
prompt_1 BYTE "What's your name? ", 0
intro_2 BYTE "Nice to meet you, ", 0
prompt_2 BYTE "How old are you? ", 0
dogAge DWORD ?
result_1 BYTE "Wow ... that's ", 0
result_2 BYTE " in dog years!", 0
goodBye BYTE "Good-bye, ", 0
x DWORD 153461
y BYTE 37
z BYTE 90

.code
main PROC

; Introduce programmer
    mov AH, y
    mov AL, z

    mov edx, OFFSET intro_1
    call WriteString
    call CrLf

; Get user name
    mov edx, OFFSET prompt_1
    call WriteString
    mov edx, OFFSET userName
    mov ecx, 32
    call ReadString

; Get user age
    mov edx, OFFSET prompt_2
    call WriteString
    call ReadInt
    mov userAge, eax
More Basics...

• Be respectful (Establishing a Positive Community)
• Have a growth mindset
  • Most abilities could be developed through dedication and hard work
• Academic Misconduct (0 tolerance!!) (See section 17 of the syllabus)
  • https://engineering.oregonstate.edu/academic-misconduct
• Be Proactive
  • Take control and cause something to happen, rather than just adapt to a situation or wait for something to happen
Technology

• Laptops (Windows)
  • Phones needed for DUO
  • bypass DUO: Follow instructions [here](#)
Attendance

• **Lecture**: Strongly Encouraged
  • I will post lecture slides and demoed code on Canvas
Grade Breakdown

• 20% - Weekly Summaries
• 10% - Quizzes
• 15% - Midterm Exam
• 35% - Assignments
• 20% - Final Project
Weekly Summaries – 20%

• 10 in total (2% each)
  • Open book, open note, open internet, open lecture, open classmates/friends.

• Available from: Thur 12 pm (after lecture) to Sun 11:59 pm
  • Canvas is very unforgiving about due times -- don't push it.

• T/F, and multiple choices, short answers, covering assigned reading material and lectures from the week

• A time limit of 6 hours

• Two attempts, the higher score will be recorded

• Cannot be taken after the due
Quizzes – 10%

• 5 in total, including the syllabus quiz (2% each)
  • Open book, open note, open internet, but NOT open classmates/friends
• Available from: Thur 12 pm (after lecture) to Sun 11:59 pm
  • Canvas is very unforgiving about due times -- don't push it.
• T/F, and multiple choices, short answers, covering material taught from the
  previous quiz to that point
• 1 attempt, 60-minute time limit

• Refer to the Course Calendar for quiz due dates (weeks)
Midterm Exam – 15%

• One Mid Term (in Week 6)
  • During lecture time
  • In person, same classroom
    • T/F, and multiple choices, short answers
  • Close-everything
  • Allowed to use a calculator, and scratch paper
Programming Assignments – 35%

• 5 in the term
• Some are one-week, and some are two-week assignments
• All programming assignments must be submitted in order to pass the course – otherwise F
• Always due Sunday by midnight
• All code (.asm) must run on Visual Studio – otherwise 0
• Late assignments
  • 2 grace days throughout the term
  • Late work is penalized 15% per day
  • At max, 2 days late.
  • More than 2 days after due → 0

• Refer to section 13 on the syllabus
Final Project – 20%

• No final exam, but a project

• Due during final’s week (exact time: TBD)

• Fail to submit the final project \(\rightarrow F\)

• Not allowed to use grace days
Grading Philosophy*

• A  [93 or greater) mastery
• A- [90 – 93)
• B+ [87 – 90)
• B  [83 – 87) stable/proficient
• B- [80 – 83)
• C+ [77 – 80)
• C  [73 – 77) passable
• C- [70 – 73)

*Note: I do round ☺ (i.e. 89.45 → 89.5 → 90 ☺)
How to Be Successful

• Read and listen carefully
• Start assignments early
• Be proactive with absences and issues that arise in the term
• Get help when you need it
  • Make use of Discord and Office Hours

• Refer to section 14 on the syllabus
TAs

• Go see your TAs!!!
• Where: Varies
• When: Varies – check the Office Hours page on Canvas
Help Hierarchy

• Reread assignment, lecture slides, syllabus, textbook
• Google online
• Ask a friend
• Check Discord for relevant posts or create a new question
• Ask a TA
  • You can attend office hours
  • TAs will also be monitoring Discord
• Ask Roger
Lecture Topics:

• Syllabus

• Introduction to Hardware, Software, and Languages

• Setup Instructions
Intro to Problem-Solving Languages

• Viewed by “levels”

• Natural languages:
  • E.g.: English, Spanish, Chinese...
  • Used by humans
  • Many interpretations
  • Translated to programming languages by computer programmers
Intro to Problem-Solving Languages

• Viewed by “levels”
  • High-level computer programming languages
    • E.g.: Java, C/C++, Python...
    • English like, portable to various architectures
    • Strict rules of syntax and semantics
    • Translated to lower levels by compilers/translators

• Low-level computer programming languages
  • E.g.: Intel assembly, MacOS assembly...
  • Mnemonic instructions for specific computer architectures
  • Translated to machine languages by assemblers

```c
#include <stdio.h>

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

```java
public class Hello
{
    public static void main(String argv[])
    {
        System.out.println("Hello, World!");
    }
}
```

```python
print "Hello, World!"
```
Intro to Problem-Solving Languages

• Viewed by “levels”
  • Machine-level computer languages
    • E.g.: Intel machine instructions, MacOS machine instructions
    • Actual binary code instructions for specific architecture
Programming Tools/Environments for Various Language Levels

• Natural Language
  • Word processors

• High-level programming languages
  • Text editor, libraries, compiler, linker, loader, debugger
  • E.g.: Eclipse, Visual Studio, ...

• Low-level programming languages
  • Text editor, libraries, assembler, linker, loader, debugger
  • E.g.: any text editor together with MASM, Visual Studio, ...

• Machine-level computer languages
  • Some way to assign machine instructions directly into computer memory
  • E.g.: set individual bits (switches), loader
Computer Languages / Computer Hardware Viewed by “Levels” (simplified)

• Level 4: Problem solution in natural language
  • Description of algorithm, solution design
  • **Programmer** translates to …

• Level 3: Computer program in high-level computer programming language
  • Source code (machine independent)
  • **Compiler** translates to …

• Level 2: Program in assembly language
  • Machine specific commands to control hardware components
  • **Assembler** translates to …

• Level 1: Program in machine code
  • Object code (binary)
  • **Linker** / loader sets up …

• Level 0: Actual computer hardware
  • Program in electronic form
Assembly Language

• In this course...
  • Skip the “high-level language” level
  • Write programs in assembly language
  • Expand levels 2, 1, and 0
    • understand what happens inside the computer

• Use an assembly language to understand a specific architecture
• Concepts transfer to other architectures
Assembly Language

• Assembly language provides:
  1. Set of **mnemonics** for **machines instructions**
     • Opcodes and addressing modes

  2. Mechanism for naming **memory addresses** and other **constants**
     • Note: a named memory address is usually called a “**variable**”

  3. Other “conveniences” for developing **source code** for a particular **machine architecture**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A, byte</td>
<td>add A to byte, put result in A</td>
</tr>
<tr>
<td>ADDC A, byte</td>
<td>add with carry</td>
</tr>
<tr>
<td>SUBB A, byte</td>
<td>subtract with borrow</td>
</tr>
<tr>
<td>INC A</td>
<td>increment A</td>
</tr>
</tbody>
</table>
Assembler and assembly

• An assembler is a **software system** that takes assembly language as input and produces machine language as output.
Operating Systems (OS)

- Operating systems provide interfaces among users, programs, and devices (including the host computer itself).

- Implemented for specific architecture (in the host computer’s machine language).
Low-level programming

• Level 2: Program in assembly language
  • Assembler translates to ...

• Level 1: Program in machine code
  • Operating system does partial translation
  • The hardware’s instruction set architecture (ISA) provides a micro-program for each machine instruction (CISC*) or direct execution (RISC*)

• Level 0: Actual computer hardware
  • Digital logic (circuits)
  • Micro-architecture circuits control computer components

*More later on CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer)*
Relationship: Instruction Set ↔ Architecture

• A computer’s instruction set is defined by the computer’s architecture.
  • i.e.: each computer’s architecture has its own machine language.
  • E.g.: Sun machine instructions will not work on an Intel architecture

• **Cross-assemblers** (software) can be used to convert a machine language to another machine languages.

• **Virtual machines** (software) can be used to simulate another computer’s architecture
Relationship: Architecture ←→ Software

- **Hardware**: Physical devices
  - E.g.: circuits, chips, disk drives, printers...

- **Software**: Instructions that control hardware
  - E.g.: games, word processors, compilers, operating systems...

- Sometimes the line between hardware and software is not clear
  - E.g.: Parts of an operating system might be implemented in hardware
System Architectures

• Super-computer
• Mainframe
• Multiprocessor/Parallel (multi-core)
• Server
• Distributed (collection of Workstations)
  • Network
• Personal computer
  • Desktop, laptop, netbook …
• Micro-controller (Real-time/Embedded system)
  • Phone, car, watch, appliance …
• Etc.
Two architecture development tracks

- Build more **powerful** machines
  - Multi-core, etc.
- Build some machine **smaller/cheaper**
  - Nanotech

- *Moore’s Law*
  - the number of transistors in a dense integrated circuit (IC) doubles about every two years
Why use assembly language?

• Easier than machine code
• Access to all features of target machine
• Performance (maybe)
• Using mixed languages

• Note that assembly language tends to solve toward a high-level language
  • Advanced features (“auto” loop control, etc.)
  • Libraries
void SimdMul(float *a, float *b, float *c, int len)
{
    int limit = (len/SSE_WIDTH) * SSE_WIDTH;
    __asm
    {
        ".att_syntax\n\t"
        "movq -24(%rbp), %r8\n\t" // a
        "movq -32(%rbp), %rcx\n\t" // b
        "movq -40(%rbp), %rdx\n\t" // c
    }
    for(int i = 0; i < limit; i += SSE_WIDTH)
    {
        __asm
        {
            ".att_syntax\n\t"
            "movups (%r8), %xmm0\n\t" // load the first sse register
            "movups (%rcx), %xmm1\n\t" // load the second sse register
            "mulps %xmm1, %xmm0\n\t" // do the multiply
            "movups %xmm0, (%rdx)\n\t" // store the result
            "addq $16, %r8\n\t"
            "addq $16, %rcx\n\t"
            "addq $16, %rdx\n\t"
        }
    }
    for(int i = limit; i < len; i++)
    {
        c[i] = a[i] * b[i];
    }
}
Common uses of assembly language

• Embedded systems
  • Efficiency is critical
• Real-time applications
  • Timing is critical
• Interactive games
  • Speed is critical
• Low-level tasks, Device drivers
  • Direct control is critical
Main concepts:

• Hardware/software
• Languages (high-level, assembly, machine)
• How statements are translated from higher to lower levels
• Variety of architectures
  • Each has its own instruction set
• Applications of assembly language
Lecture Topics:

• Syllabus
• Introduction to Hardware, Software, and Languages
• Setup Instructions
Things to do before next lecture

• Complete the syllabus quiz (and make sure you get 100%)
• Complete the Visual Studio Setup

• Do the self-check exercise

• Continue on the assigned readings