Contents

1 CS33 Introduction and Course Overview

Lecturer: Glen Ryman, a faculty at UCLA since 2001, specializing in microprocessor design. Although his main research area is not the primary focus of this class, the knowledge influences his teaching approach.

2 Course Description

This course serves as an introduction to systems, going deeper into the computer science abstraction than before. It covers both the software and hardware system sides, with the aim of giving a foundation on which later topics like compilers, microprocessor design, and networking will be built.

3 Key Topics and Concepts

- **Abstraction Layers in Computing**: The course delves into various levels of abstraction, from application and algorithm to instruction set architecture (ISA) and microarchitecture.
- **Instruction Set Architecture (ISA)**: The interface between hardware and software, defining the capabilities of hardware and how software controls it.
- **Software Stack and Hardware Implementation**: Discussion on how software layers interface with the physical hardware and the implications of physical aspects like electromigration in hardware design.

4 Course Structure and Assessments

- **Lectures**: Attendance is encouraged but not mandatory, with sessions available on BruinCast.
- **Discussion Sections**: Mandatory participation, aiming to reinforce lecture content and prepare for labs.
- **Labs**: Emphasize starting early due to the unfamiliarity of the material. Labs build on lecture concepts and provide practical experience.
- **Exams**: Midterm and final exams, with the final being comprehensive. Open book/notes but no electronic devices allowed.
- **Grading**: Based on exams, labs, discussion participation, and homework assignments. A curve may be applied depending on overall performance.

5 Lab Quick Tips

- Start labs early to avoid misunderstandings and last-minute pressure.
- Pay attention to the instructions and details provided in the lab manuals.
- Utilize discussion sections for clarifications and additional support.

6 Tools and Resources

- **Emacs and Vim**: Basic understanding of these text editors can be beneficial for editing code directly on the school's servers.
- **GCC**: The GNU Compiler Collection used for compiling C programs.
- **GDB**: The GNU Debugger for debugging programs, allowing you to step through code and inspect variables and memory.

7 Compiler and Memory Insights

Understanding the compiler's role in transforming high-level code into machine instructions and allocating memory is crucial. This involves stages like preprocessing, compiling, assembling, and linking.

Key Practices to Understand:

- **Memory Dump Analysis**: Interpreting data and text segments of compiled programs to understand memory allocation and execution flow.
- **Register and Memory Inspection with GDB**: Learning how to inspect register values and memory locations to trace program execution.

8 Study and Learning Tips

- **Understand Key Concepts**: Focus on building a solid understanding of core concepts like ISA, software/hardware stack, and binary operations.
- **Practice with Tools**: Hands-on practice with tools like GCC, GDB, Emacs/Vim will solidify your understanding and improve your efficiency.
- **Engage in Labs**: Labs are designed to reinforce lecture content. Engaging deeply with lab tasks will bridge theory with practical skills.
- **Participate and Ask Questions**: Active participation in discussions and asking questions when in doubt can significantly enhance your understanding.

9 First Lab (Lab Zero): A Sneak Peek

The first lab focuses on bitwise operations and how to manipulate bits without using high-level arithmetic operations. It's designed to foster an understanding of how low-level computing processes work and prepare you for more complex tasks ahead.

Lab Zero Challenge:

- Multiply a number by 3/4 and correctly round towards zero using only bitwise operations, addition, and logical NOT. This requires a creative approach, possibly involving bit shifts as a proxy for multiplication and division.

Approach Tips:

- Start by considering how left and right shifts can replicate multiplication and division's effects on binary numbers.
- Experiment with addition and shifting to account for both positive and negative numbers, watching out for the rounding behavior.

Reminder: This lab emphasizes the importance of understanding the underlying principles of computer architecture and systems programming, setting the stage for future topics.

10 Final Notes

This course, through its structured lectures, labs, and assessments, aims to deepen your understanding of systems programming and computer architecture. Engage with the course material actively, make use of the resources provided, and don't hesitate to seek help when needed. This foundational knowledge will be invaluable as you progress through your computer science career.