CSCE 312 – Introduction to Computer Systems
Spring 2016

Course Logistics

1. Time and Place
   a. **Lecture**: Mon, Wed, Fri 3:00pm – 3:50pm Room: HRBB 124
   b. **Lab**: Various
      i. Section 501: Mon 4:10pm-5:50pm RDMC 111C
      ii. Section 502: Wed 4:10pm-5:50pm RDMC 111C
      iii. Section 503: Thu 12:40pm-2:20pm RDMC 111C
2. Course Material and Discussion Forum (Piazza): TBD
3. Assignments and Grades (eCampus): TBD
4. Instructor: Aakash Tyagi (tyagi@cse.tamu.edu) HRBB-515A
5. Teaching Assistant: TBD
6. Peer Teacher: Austin Hamilton (karasaj@email.tamu.edu)
7. Office Hours
   a. **Aakash**: Appointment by e-mail (tyagi@cse.tamu.edu)
   b. **TA**: TBD
   c. **Austin**: TBD (Peer Teachers Central in HRBB Building 1st Floor)

Prerequisites

All the computer science knowledge necessary for completing this course is given in the course lectures, projects, and textbook. Lab assignments will require basic computer programming skills. Some basic background in data structures and algorithms will be appreciated.

Description

The course objective is to integrate key notions from algorithms, computer architecture, operating systems, compilers, and software engineering in one unified framework. This will be done constructively, by building a general-purpose computer system from the ground up. In the process, we will explore many ideas and techniques used in the design of modern hardware and software systems, and discuss major trade-offs and future trends. Throughout this journey, you will gain many cross-section views of the computing field, from the bare bone details of switching circuits to the high level abstraction of object-based software design. The course consists of materials on the following topics: introduction to computer systems, data representation, machine language, processor architecture, memory hierarchy, assembler, compiler, virtual memory, system level I/O. Several laboratory assignments will provide hands-on experience many of the above topics.

Catalog Description

Introduction to computer systems from programmer's perspective: simple logic design, data representation and processor architecture, programming of processors, memory, control flow, input/output, and performance measurements; hands-on lab assignments.

Textbooks

- **Main Textbook**: The Elements of Computing Systems: Building a Modern Computer from First Principles. Noam Nisan, Shimon Schocken, The MIT Press, 2005.
- **Reference Texts:**
  - Computer Systems: A Programmer's Perspective, Randal E. Bryant and David R. O'Hallaron, Prentice Hall, 2015.
  - Digital Design, 2nd Ed, by Frank Vahid, Wiley publication, 2010

Methodology

This is mostly a hands-on course, evolving around building a series of hardware and software modules. Each module development task is accompanied by a design document, an API, an executable solution, a test script (illustrating what the module is supposed to do), and a detailed implementation plan (proposing how to build it). The projects are spread out evenly, so there will be no special pressure towards the semester's end. Each lecture will start by reviewing the work that was done thus far, and giving guidelines on what to do next.

Programming

The hardware projects will be done in a simple Hardware Description Language (HDL) that can be learned in a few hours. It is covered in the Appendix in our Textbook and we will cover some basics in class as well. The resulting chips (as well as the topmost computer-on-a-chip system) will be tested and simulated on a supplied hardware simulator, running on the student's computer. The software projects can be done in either Java or Python or C++.

Resources

All the baseline course materials – lecture notes, book chapters, simulators, software tools, tutorials and test programs – can be downloaded freely from the course web site http://www.nand2tetris.org. The supplied software can run as is on either Linux or Windows. The lecture notes presented in class are modified with Instructor's own thoughts and inputs as deemed necessary.

Preliminary Course Content (subject to changes)

1. Hello, World: Demonstration of some interactive games (like *Pong*, *Tetris, Sokoban*) running on the computer built in the course, tracing their execution from the object-oriented language level down to the Nand level; The abstraction / implementation paradigm and its role in systems design; Overview of the Hardware Description Language (HDL) used in the course; Designing a set of elementary logic gates from primitive Nand gates; Implementing the gates in HDL.

2. Combinational logic: using the logic gates to design and implement a family of binary adders, culminating in the construction of a simple ALU (Arithmetic-Logic Unit).

3. **Sequential logic:** using the logic gates to design and implement a memory hierarchy, from elementary flip-flop gates to registers and RAM units of arbitrary sizes.

4. **Machine language:** introducing an instruction set, in both binary and assembly (symbolic) versions; Writing some low-level assembly programs and running them on a supplied CPU emulator.

5. **Computer architecture:** Integrating the chip-sets into a computer platform capable of running programs written in the machine language. We will spend some additional discretionary time discussing modern computer architecture principles and components like Pipelining and Memory Hierarchy.

6. **Assembler:** Basic language translation techniques: parsing, symbol table, macro-assembly; Building an assembler for the assembly language.

7. **Virtual machine I:** The role of virtual machines in modern software architectures like Java and .NET; Introduction of a typical VM language, focusing on stack-based arithmetic, logical, and memory access operations; Implementing part I of a VM translator that translates from the VM language into the assembly language.

8. **Virtual machine II:** Continued discussion of the VM abstraction and implementation, focusing on stack-based flow-of-control and subroutine call-and-return techniques; Extending the VM translator into a complete VM implementation that serves as the back-end component of the compiler built later in the course.

9. **High Level Language:** Introducing *Jack*, a simple high-level object-based language with a Java-like syntax; discussing various trade-offs related to the language design and implementation; using Jack to write a simple interactive game and running it on the computer built in earlier weeks. This project makes use of the compiler and operating systems built in the remainder of the course.

10. **Compiler I:** Context-free grammars and recursive parsing algorithms; Building a syntax analyzer (tokenizer and parser) for the jack language; the syntax analyzer will generate XML code reflecting the structure of the translated program.

11. **Compiler II:** Code generation, low-level handling of arrays and objects; Morphing the syntax analyzer into a full-scale compiler; This is done by replacing the routines that write passive XML with routines that generate executable VM code for the stack machine presented earlier in the course.

12. **Operating system:** Discussion of OS/hardware and OS/software design trade-offs, and time/space efficiency considerations; Design and implementation of some classical arithmetic and geometric algorithms that come to play in the implementation of our OS, as well as classical mathematical, memory management, string processing, and I/O handling algorithms, needed for completing the OS implementation.

<u>Likely order of coverage will be as follows:</u>

1. Midterm Exam: Chapters 1-6
2. Final Exam: Chapters 7-12 and Content on Pipelining and Memory Hierarchy

Course Grade

A: 90+, B: 80-89, C: 70-79, D: 60-69, F: 59-

50% lab assignments (12 lab assignments), 40% exams (2 x 20% each), 10% quizzes (several in-class quizzes).

The first six lab assignments will be done individually. The last six lab assignments will be done in teams of 2. The exam content will be roughly divided in half.

The final examination will be in the classroom during its allocated time in the Final's Week. See http://registrar.tamu.edu/Courses,-Registration,-Scheduling/Final-Exam-Schedule for details.

Late Submissions

Penalty: Unless stated otherwise, lateness is penalized as a simple linear function of minutes late. **You lose a percentage point of the score for every hour you are late.** Roughly speaking, if you are 4 days late you end up losing all the points for that assignment.

Communication

Instructor, Teaching Assistants, and Peer Teachers for this course will do their best to communicate relevant administrative information (deadlines, information about posted material, details about projects, locations of tutorials, and so on) in an effective and timely manner. We will be using announcements in class, and material and communication related postings on Piazza. E-mails listed on course roster are used in Piazza. If you are not receiving Piazza updates, please contact the instructor. **Please use Piazza for all correspondence related to the course**. You can post Public or Private. This is a very convenient and potentially effective way to communicate with instructor and TA.

Academic Integrity Statement and Policy

"An Aggie does not lie, cheat or steal, or tolerate those who do." For additional information, please visit: http://aggiehonor.tamu.edu.

Americans with Disabilities Act (ADA) Policy Statement

The Americans with Disabilities Act (ADA) is a federal anti-discrimination statute that provides comprehensive civil rights protection for persons with disabilities. Among other things, this legislation requires that all students with disabilities be guaranteed a learning environment that provides for reasonable accommodation of their disabilities. If you believe you have a disability requiring an accommodation, please contact Disability Services, in Cain Hall, Room B118, or call 845-1637. For additional information visit http://disability.tamu.edu.

Acknowledgments

The course owes its existence to the Nand2Tetris organizers Prof. Nisan and Schocken. Additional material taught in this course is taken from the 312 course taught by our own Prof. E.J. Kim.