Catalog Description

This is the second course of the CS2001-CS2002 course sequence. The objective of this course sequence is to teach foundational CS concepts by constructing a general-purpose computer system from the ground up. By integrating computer architecture, compilers, operating systems, and algorithms through hands-on projects, students will explore hardware and software design techniques and balance tradeoffs between competing design constraints. Students will learn about the engineering of systems across many levels of abstraction, from digital logic to software design. In this course, the von Neumann architecture is explored in depth through building a series of software translation programs and aspects of an operating system. Pre-requisites: CS2001 and CS2020.

Program Learning Objectives

This course sequence presents foundational knowledge in a Computer Science degree program necessary for understanding topics in more specialized courses. The course does not individually or directly satisfy ESRs or degree requirements, but its fundamental nature lays and important part of the foundation for all ESR and degree requirement courses.

Learning Outcomes

Upon successfully completing the course sequence, the student will be able to:

• Understand foundational principles of computer science, including computer architecture, compilers, operating systems, and algorithms, by constructing a general-purpose computer system from the ground up.
• Balance tradeoffs between key design constraints.
• Apply the von Neumann Model and the concepts of cache memory, virtual memory, memory segmentation, paging, and address translation to the construction of a general-purpose computer system.
• Apply the principle of abstraction to solve computer system design problems.
• Accelerate programs by identifying hardware and software bottlenecks and understanding the interface between software and physical devices.

Upon successfully completing the course, the student will be able to:

• Design programs in a simple, object-based, high-level language.
• Extend the functionality of the high-level language using a library.
• Translate programs to progressively lower levels of abstraction.
• Understand how programs work at different levels of abstraction.
• Describe and explain the dynamics of procedure invocation and evaluation.
• Describe and explain how program components are linked (bound together) at run time.
Assessment

To assess the learning outcomes, students will demonstrate their knowledge on a sequence of tests. Students will also perform laboratory exercises, and lab reports are part of the assessment. The projects are described in the textbook and are supported by machine simulation software provided by the authors. Running code the students have translated on the simulated machines provides immediate feedback on correctness and basic troubleshooting information in cases of incorrect translations.

Assessments for Course Sequence Learning Outcomes:
• Understand foundational principles of CS: Labs 7-12 and Tests
• Balancing of tradeoffs: Labs 7-12 and Tests
• Apply von Neumann Model, etc.: Labs 7-12 and Test on material from Weeks 1 and 2: "Memory"
• Apply abstraction to solve design problems: Labs 7-12 and Tests
• Accelerate programs: Labs 7-12 and Tests

Assessments for Course Learning Outcomes:
• Design programs in a simple, object-based, high-level language: Lab 9
• Extend the functionality of the high-level language using a library: Lab 10
• Translate programs to progressively lower levels of abstraction: Labs 7, 8, 10, and 11.
• Understand how programs work at different levels of abstraction: Labs 7-12 and Tests
• Describing the dynamics of procedure invocation and evaluation: Test on Week 3 and 4 material: "Execution Dynamics"
• Describing how program components are linked: Test on material from Weeks 8 and 9: "Linking and Loading"

Credit hours

This is a 3-3 class. The course will meet for six hours per week, organized as three lecture hours and three lab hours.

Prerequisites


Textbook

## Topics Covered

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Topic</th>
<th>Subtopics</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Memory</td>
<td>Caches, associativity, locality principle, memory hierarchy, virtual memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Project 7: VM I: Stack Arithmetic</em></td>
</tr>
<tr>
<td>2</td>
<td>Execution Dynamics</td>
<td>Managing storage, stack, procedure call and return, read-only instruction code</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Project 8: VM II: Program Control</em></td>
</tr>
<tr>
<td>1</td>
<td>Speed-up</td>
<td>Pipelining, branch prediction, prefetching, out-of-order execution, superscalar processors</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Project 9: High-Level Language</em></td>
</tr>
<tr>
<td>2</td>
<td>Input/Ouput and Storage Systems</td>
<td>I/O and performance, Amdahl's Law, I/O architectures, data transmission modes, storage hierarchy, file systems</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Project 10: Compiler I: Syntax Analysis</em></td>
</tr>
<tr>
<td>2</td>
<td>Linking and Loading</td>
<td>Creating executable files from separately compiled components, single-job vs. concurrent operations, services and facilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Project 11: Compiler II: Code Generation</em></td>
</tr>
<tr>
<td>2</td>
<td>Operating Systems</td>
<td>Organization and types of operating systems, tradeoffs of different OSes, bootstrapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Project 12: Operating System</em></td>
</tr>
</tbody>
</table>