
Catalog Description
The objective of this course is to teach basic CS concepts by constructing a general-purpose computer system from the ground up. By introducing computer architecture, compilers, operating systems, and algorithms through hands-on projects, students will explore the interdependency of hardware and software design techniques and balance tradeoffs between competing design constraints. Students will learn about the engineering of computer systems across many levels of abstraction, from digital logic to software design. Co-requisites: CS2020 and MA2025.

Program Learning Objectives
This course 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 an important part of the foundation for all ESR and degree requirement courses.

Learning Outcomes
Upon successfully completing the course, the student will be able to:
- Apply the von Neumann model to construct a general-purpose computer system.
- Apply the principle of abstraction to solve computer system design problems.
- Understand how processors are designed and organized.
- Understand how the processor and memory unite to execute programs.
- Apply Boolean logic to the design of digital circuits.
- Construct complex systems from primitive digital circuits.
- Understand how programs work at different levels of abstraction.
- Translate programs to progressively lower levels of abstraction.
- Describe and explain the dynamics of procedure invocation and evaluation.

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 Learning Outcomes:
- Apply von Neumann model: Labs 1-9 and Tests
- Apply abstraction principle: Labs 1-9 and Tests
- Understand processor design and organization: Labs 1-5 and Tests
- Understand program execution: Labs 1-9 and Tests
- Apply Boolean logic: Labs 1-5 and Tests
- Construct complex systems from primitive components: Labs 1-9 and Tests
- Understand programs at different abstraction levels: Labs 4-9 and Tests
- Translate programs: Labs 6-8 and Tests
- Procedure invocation and evaluation: Lab 8.2 and Tests

Credit Hours
This course will meet for six hours per week, organized as three lecture hours and three lab hours.
Co-requisites
CS2020 and MA2025 (may be enrolled concurrently).

Textbook

Topics Covered

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Topic</th>
<th>Subtopics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Elementary Logic Gates</td>
<td>Demonstration of the system to be built; Abstraction-implementation duality; Simulator; Lab 1: Implementation of elementary logic gates (Not, And, Or, Xor, Mux, DMux)</td>
</tr>
<tr>
<td>1</td>
<td>Combinational Circuits</td>
<td>Signed integer representation; Carry; Overflow; Control; Lab 2: Implementation of combinational circuits (Adder, Incrementer, and ALU)</td>
</tr>
<tr>
<td>1</td>
<td>Sequential Circuits</td>
<td>Address; Flip-flop; Lab 3: Implementation of sequential circuits (Bit, Register, RAM, and PC)</td>
</tr>
<tr>
<td>1</td>
<td>Machine Language</td>
<td>Instruction set; PC; Lab 4: Implementation of assembly language programs (Multiply and Fill)</td>
</tr>
<tr>
<td>1</td>
<td>Computer Architecture</td>
<td>Von Neumann architecture; Fetch-decode-execute cycle; I/O; Machine code; Lab 5: Implementation of complete machine (Memory, CPU, Computer)</td>
</tr>
<tr>
<td>1</td>
<td>Assembler</td>
<td>Binary vs. symbolic; Symbol table; Parsing text; Lab 6: Implementation of two-pass assembler</td>
</tr>
<tr>
<td>1</td>
<td>VM Stack Arithmetic</td>
<td>Two-tier compilation; Stack; Lab 7.1: Translation of arithmetic commands to assembly (add, subtract, additive inverse, bitwise-and, bitwise-or, bitwise-not, equals, less than, greater than)</td>
</tr>
<tr>
<td>1</td>
<td>VM Stack Memory Access</td>
<td>Memory segments; Lab 7.2: Translation of memory access commands (push, pop)</td>
</tr>
<tr>
<td>1</td>
<td>VM Branching Commands</td>
<td>Conditional branch; Unconditional branch; Lab 8.1: Translation of branching commands to assembly (label, if-goto, goto)</td>
</tr>
<tr>
<td>1</td>
<td>VM Function Call and Return</td>
<td>Stack frame; Subroutine; Argument; Lab 8.2: Translation of function call and return commands to assembly (function, call, return)</td>
</tr>
<tr>
<td>1</td>
<td>High-Level Language</td>
<td>Syntax vs. grammar; API; Constructor vs. method vs. function; Class; Scope; Object; Array; Memory leak; Data type; Precedence; Lab 9: Implementation of program in high-level language</td>
</tr>
</tbody>
</table>