COMP 2280 – Introduction to Computer Systems

Course Description

Calendar entry

(Lab Required) Data representation and manipulation, machine-level representation of programs, assembly language programming, and basic computer architecture. Not available to students who have previously completed ECE 3610. Prerequisites: COMP 2140, COMP 2160, and one of MATH 1240, MATH 1241 or COMP 2130.

General Course Description

By this point students have learnt how to implement data structures, implement data structures safely, and manipulate memory through pointers; all in high-level languages such as C and Java. In doing so we've been treating the computer as a black box that magically does what we need it to do. Now it's time to show how we make that magic happen.

In this course we teach how code gets turned into something a computer can use. We show how data structures are represented, how function calls are implemented, and how recursion is realized. All through the lens of assembly language.

We don't stop there. We also introduce the fundamental components of a computer and how they combine to provide us with the modern computers we use today. How an instruction is managed and moves through these components in a structured way is at the core of what we call Computer Science.

Detailed Prerequisites

Before entering this course, a student should be able to:

- Implement data structures such as lists, stacks, and binary trees in a high-level language.
- Manipulate objects in memory using pointers.
- Manipulate memory buffers through pointers and simple address arithmetic.
- Perform well-defined and structured tests on code.
- Use a debugger to inspect program state and step through code line-by-line.
- Evaluate Boolean algebra expressions.

Course Goals

By the end of this course students will:

- See how the code they write gets interpreted and executed by a computer.
- Understand what the compiler does to ensure that they can implement data structures and manipulate them using techniques such as recursion.
- Experience the need for an operating system if they want to provide anything beyond manually loading and running one application at a time.
- See how a simple set of circuits leads to the creation of Memory, the Central Processing Unit, and everything in between.
- Be introduced to the notion of a state machine through the controlled execution of instructions.

Learning Outcomes

Data Representation and Boolean Algebra

Students should be able to:

1. Provide the binary representation of 2's complement integers and fixed- and floating-point reals.
2. Interpret binary representations of values to provide the decimal/human-readable equivalent.
3. Perform standard mathematical operations (+, −, ×, ÷) at the binary level.
4. Perform Boolean operations (AND, OR, NOT, XOR) and standard manipulations (such as DeMorgan's Law) on/between binary values.
5. Use logical operations to mask a binary value's individual bits to 0 or 1 and compare bits between binary values.

Introduction to Circuits

Students should be able to:

1. Draw and explain combinational circuits that implement Multiplexers, Decoders, and Adders.
2. Draw and explain sequential circuits, through timing diagrams, that implement a flip-flop.
3. Combine flip-flops to implement a register.
4. Use multiplexers, decoders, and flip-flops to implement a memory chip and explain how words are stored and read.
5. Combine memory chips to build/organize a memory with standard size words (e.g., 32-bits).
6. Show how a memory address, a.k.a. the contents of a pointer, is interpreted to access a word given a particular organization of memory chips.

Machine Level Program Representation

Students should be able to:

1. Convert high-level code patterns and control structures into assembly language.
2. Represent and manipulate data structures such as arrays, linked lists, and binary trees using assembly language.
3. Use a run-time stack to pass arguments, allocate local variables, and return results in assembly language.
4. Implement a run-time stack, in assembly language, that fully supports all function call behaviour required by a high-level language such as C or Java.
5. Manually perform a two-pass scan of assembly language instructions to create a symbol table, generate machine code, and link the code so that it is ready to be loaded into memory.

Systems Programming

Students should be able to:

1. Differentiate between user code and system code by explaining the flow of control from user mode to system mode and back again.
2. Differentiate between polling and interrupt driven input/output (I/O).
3. Update the system's interrupt vector table and activate/deactivate an interrupt service routine (ISR).
4. Implement an ISR that accepts and processes basic input.
5. Identify inappropriate coding within an ISR and explain why system/device driver implementations need to avoid such coding pitfalls.

Computer Organization

Students should be able to:

1. Identify the registers, data paths, control points and logical units that make up a Central Processing Unit (CPU).
2. Explain the CPU<->Memory interface and how data is transferred to/from memory.
3. Explain the Fetch-Decode-Execute-Store instruction lifecycle and how it captures all possible CPU functionality.
4. Explain the functionality of the Control Unit in terms of the timed activation of control points to execute machine instructions.
5. Provide micro-operation sequences to perform read from memory, write to memory, register transfer, and arithmetic/logical operations.
6. Work with the Control Unit's state machine to identify the sequence of steps needed to complete a machine instruction.
7. Convert a machine instruction into micro-operation sequences that correctly manipulate control points to perform the steps required by the machine instruction.