Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Course Outline
Online - Summer 2020

Instructor

Dr. Larry Hughes
Department of Electrical and Computer Engineering
Dalhousie University
Room C-369
Email: larry.hughes@dal.ca

Introduction

For at least 4500 years, humans have employed a variety of techniques, most notably tables (such as astronomical, logarithmic, and trigonometric), to reduce the drudgery and complexity of repeating commonly performed calculations, be they scientific, engineering, or economic. In 1821, Charles Babbage, after finding error after error in hand-calculated astronomical tables exclaimed:

I wish to God these calculations had been executed by steam.

Because of this, Babbage proposed, designed, and began building the Difference Engine, essentially a simple programmable calculator, with the intention of putting Victorian computers out of work (a Victorian computer was "a person who makes a living by calculating mathematical tables by hand").

Work on the Difference Engine subsequently stopped with the withdrawal of government funding. Babbage went on to design the Analytical Engine, a stored-program computer powered by steam, while Ada, Countess of Lovelace, designed and wrote software intended for the machine. Ada has been referred to as the first computer programmer.

Today's computers, based concepts first proposed before, during, and shortly after the Second World War by people such as Turing and von Neumann, are intended to do what tables have done for centuries and human computers did in Victorian times:

Computers can perform complex and repetitive procedures quickly, precisely and reliably, and can quickly store and retrieve large amounts of data.

Ever since the development of the electronic computer, hardware and software designers have had one overriding objective: how to make computers perform complex and repetitive procedures more quickly (and, ideally, at a lower cost).

Increasing the speed of computers is due, in part, to improvements in the design of integrated circuits, as predicted by Gordon Moore in 1965 (the 20-24 month doubling time is now referred to as Moore's Law):

The complexity for minimum component costs has increased at a rate of roughly a factor of two per year… Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer.

Although increasing the number of transistors on an integrated circuit has contributed to the goal of making computers operate more quickly, it is not the only reason. Over the past sixty years, there has been a significant evolution in the organization of the basic architecture of computers as described by von Neumann and supported by concomitant changes in compiler design. It is important for computer engineers to understand these changes, not only to appreciate what has been done but to propose and develop innovative designs for the future.

Since Moore's Law appears to be reaching its limit, there is a need for a new generation of computer, once again for reasons of speed, for applications in fields such as AI (artificial intelligence) to control, for example, AVs (autonomous vehicles).

ECED 3403 is intended for third year Computer Engineering students; it examines computers in terms of their architecture and organization. As the first step, it is necessary to distinguish between computer architecture and computer organization:

- A computer's **architecture** is its abstract model and is the programmer's view in terms of instructions, addressing modes, and registers.
- A computer's **organization** expresses the realization of the architecture.

In other words, architecture describes *what* a computer does, and organization describes *how* it does it.

Objectives

This summer's ECED 3403 an online version of a classroom-based course intended for third-year undergraduate Computer Engineer students.

Through a series of modules, labs, and assignments, students will develop an understanding of computer architecture, the evolution of computers, and the current state of the art.

In addition, the labs and assignments are designed to improve each student's software design, implementation, and testing skills.

The major learning outcomes, based on CEAB graduate attributes of ECED 3403 are as follows:

Outcomes	CEAB graduate attributes
• Investigate, design, and apply data and control structures for symbolic translation • Appraise and develop emulators and emulation techniques • Compare and rank central processing unit design and instruction set architectures • Design and implement techniques to overcome memory access limitations	Problem Analysis (2B): Break a complex problem into manageable elements, using appropriate assumptions. Investigation (3C): Analyze and interpret data and form conclusions Design (4B): Conceptualize and evaluate alternative approaches; perform appropriate design to fit requirements Tools (5A): Demonstrate proficiency in appropriate current computer- based packages.

Course Structure

ECED 3403 has been redesigned and modularized because of the coronavirus pandemic. The course consists of five parts:

- A series of eleven modules, each covering a specific topic in computer architecture and includes links to videos and required readings.
- Four assignments, applying software design, implementation, and testing techniques to examine different aspects of computer architecture. The assignments are built around XM3 to gain further understanding of computer architecture and software design.
- Five labs on software design techniques, implementation, and testing, all intended to make you better Computer Engineers and to help in the development of solutions for the course's assignments.
- Two quizzes scheduled to be conducted during the first week of June and the first week of July.
- An open-notes final examination. The details of the final examination will be announced towards the end-of-term.

Contact hours

Live classroom sessions will be held at the regularly scheduled times of 1:35pm to 2:55pm (ADT) on Tuesdays and Thursdays. These lectures will be conducted using Teams and recorded for later viewing.

Dr. Hughes and the TAs, Gary Hubley and Zach Thorne, will be available on Tuesdays during the regular lab hours, 3:05pm to 4:55pm (ADT).

Dr. Hughes will be available on Tuesday and Thursday evenings between 6:00pm and 7:00pm (ADT). The TAs will be available on Monday and Wednesday evenings at the same times.

If extra assistance is required outside these hours, please contact any of us and we either answer your questions directly or set up a time to meet via Teams.

Supporting material

The X-Makina assembler and emulator and the course text, *Introduction to XM3 Instruction Set Architecture*, are available, free of charge, from the course Brightspace website. Links to additional material will be supplied with each module.

Assessment

The marking scheme is as follows:

Quizzes	15%
Labs	15%
Assignments	45%
Final examination	25%
Total	100%

To pass the course, passing grades must be attained in the assignments and the final examination. The final letter-grade will be obtained from Dalhousie's grading system using the percentage total of the modules, labs, assignments, and final examination:

A+ 90%-100%	A 85%-89%	A- 80%-84%
B+ 77%-79%	B 73%-76%	B- 70%-72%
C+ 65%-69%	C 60%-64%	C- 55%-59%
D 50%-54%		
F <50%		

There is no Supplementary Examination in this course.

Course modules

The course consists of eleven modules.

Module 1: Software
a) Data structures
b) Code structures
c) Design tools
  - Compilers, assemblers, preprocessors, linkers, and librarians
  - Modules: source, object, library, and load (executable)

Module 2: Hardware
a) Memory
  - Organization (von Neumann [Princeton] and Harvard)
b) Software (again!)
  - The load module and the loader
  - The running program:
    – Subroutines: functions, the stack, arguments, automatics
    – Dynamic memory: pointers
c) The Central Processing Unit
d) Bus: Data and Address
e) Devices

f) Emulators
g) Flynn’s taxonomy

Modules 3 and 4: Central Processing Unit and Instruction Set Architectures (ISAs)
a) The Central Processing Unit
b) Accumulators, index registers, registers (Program Counter, Stack Pointer, and general purpose)
c) Subroutine support (Stack and Link registers)
d) 0 (Stack), 1, 2, and 3 address machines
e) Addressing modes
f) Exceptions: Faults and Traps
g) Examples
h) X-Makina

Module 5: Arithmetic and Logic Unit (ALU)
a) 74181 4-bit ALU
b) Bit-slicing
c) Program Status Word
d) Subtraction and comparison
e) Branching

Modules 6 and 7: Reducing CPU idle time and increasing parallelism
a) The problem: devices
b) Exceptions: Interrupts
c) Interrupt controllers
d) Double buffering
e) Direct Memory Access (DMA)
f) Processes
g) Memory Management: Segmentation and paging

Module 8: Cache
a) The “memory wall”
b) Associative memory and direct mapping
c) Cache policies (Write-through and Write-back)
d) Cache consistency
e) N-way associative
f) The Least Recently Used (LRU) algorithm

Modules 9 and 10: Pipeline architectures
a) Stages
b) Register renaming
c) Conditional branches and branch prediction
d) Out-of-order execution and code optimization
e) Converting X-Makina to pipeline

Module 11: Reduced Instruction Set Computers (RISC)
a) Limitations of Complex Instruction Set Computers (CISC)

b) Fixed instruction sizes
c) Register stacks
d) Compiler design
e) X-Makina