

Dalhousie University
Department of Electrical and Computer Engineering
ECED 3403 – Computer Architecture
Course Outline
Summer 2018

Instructor

Dr. Larry Hughes
Department of Electrical and Computer Engineering
Dalhousie University
Room C-369
Email: larry.hughes@dal.ca
Course URL: <http://lh.ece.dal.ca/eced3403>

Office hours

Tuesday

9:00am to 10:00am
11:30am to 2:15pm

Thursday

11:30am to 2:15pm
4:30pm to 6:00pm (Room C332, LSC)

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.¹ In 1821, Charles Babbage, after finding error after error in hand-calculated astronomical tables exclaimed:²

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”³).

(Work on the Difference Engine subsequently stopped with the withdrawal of government funding. Babbage subsequently designed 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.⁴)

¹ *Introduction, The History of Mathematical Tables from Sumer to Spreadsheets*, <http://uruk-warka.dk/mathematics/ER7%20etal-tables.pdf> (accessed 29 April 2017)

See also, <https://www.amazon.ca/History-Mathematical-Tables-Sumer-Spreadsheets/dp/0198508417>

² *The Babbage Engine – Overview*, <http://www.computerhistory.org/babbage/overview/> (accessed 29 April 2017)

³ *A Victorian "Calculator"*, <http://www.nzeldes.com/HOC/RoppCalc.htm> (accessed 29 April 2017)

⁴ *Ada Lovelace*, <http://www.computerhistory.org/babbage/adalovelace/> (accessed 29 April 2017)

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 computers did in Victorian times:⁵

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**):⁶

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.⁷

ECED 3403 is intended for third year Computer Engineering students; it examines computers in terms of their architecture. 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.

⁵ Definition of "computer" from the *The Free On-line Dictionary of Computing*. Denis Howe, <http://foldoc.org/computer> (accessed 29 April 2017)

⁶ Moore, Gordon E. (1965). "Cramming more components onto integrated circuits", *Electronics Magazine*.

⁷ Since Moore's Law appears to be reaching its limit, there is a need for a new general of computer, once again for reasons of speed, for applications in fields such as AI (artificial intelligence) to control, for example, AVs (autonomous vehicles).

Text

A text describing the machine used in the course is available, free of charge, from the course website. Additional reading material will be posted on the course website:

<http://lh.ece.dal.ca/eced3403>

Objectives

ECED 3403 allows the undergraduate Computer Engineer to develop an understanding of computer architecture and organization, the evolution of computers, and the current state of the art.

By the end of ECED 3403, the student will be able to (CEAB Accreditation sub-attributes in parenthesis):

- Investigate, design, and apply data and control structures for symbolic translation (System and software development 3.1.5 C, D, E);
- Appraise and develop emulators and emulation techniques (Software development 3.1.4 A, B, C);
- Compare and rank central processing unit design and instruction set architectures (Software development 3.1.4 E);
- Describe, create, and implement software and hardware interaction in pipeline architectures (Software development 3.1.2 C, D);
- Design and implement techniques to overcome memory access limitations (System and software development 3.1.3 F; 3.1.5 E).

Course Structure

ECED 3403 is a one-semester course, consisting of five parts:

- Two classroom lectures a week.
- A weekly tutorial to discuss assignments and other course-related material.
- Two quizzes, each 60 minutes in length, covering classroom lectures and tutorials.
- Three assignments, applying software to examine different aspects of symbolic translation and computer architecture.
- A final examination. The time and location of the final examination will be announced towards the end-of-term.

The marking scheme is as follows:

Quizzes	10%
Assignments	50%
Final examination	40%
Total	100%

In order 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 (A+, A, A-, B+, B, B-, C+, C, C-, D, FM, and F) using the percentage total of the quizzes, assignments, and final examination.

Course content and topics to be covered

Note: The material listed here will be covered, although not necessarily in the order presented.

1. Introduction to course
 - a) Knowledge survey
 - b) Course outline
 - c) What do we know?
2. What does software need?
 - a) Data types and representation
 - b) Code structures
 - c) Subroutines and parameters (by value, reference)
 - d) Static, dynamics, typed structures, and objects
3. Design tools
 - a) Emulators
 - b) Compilers, assemblers, pre-compilers, pre-assemblers, linkers, and loaders
4. Machine architectures
 - a) Von Neumann (Princeton) and Harvard
 - b) Central Processing Unit (CPU)
 - c) Data (and address) bus
 - d) Memory
 - e) Devices
5. Instruction Set Architectures (or ISAs)
 - a) Basic requirements
 - b) Stack machines (zero address instructions)
 - c) 1, 2, and 3 address machines
 - d) Addressing modes
 - e) Subroutine support
6. Arithmetic and Logic Unit (ALU)
 - a) 74181 4-bit ALU
 - b) Subtraction
 - c) Bit-slicing
7. Cache
 - a) The “memory wall”
 - b) Associative memory
 - c) Cache policies (Write-through and Write-back)
8. Reducing CPU idle time and increasing parallelism
 - a) Interrupts
 - b) Double buffering
 - c) Direct Memory Access (DMA)
 - d) Processes
 - e) Memory Management: Segmentation and paging

- f) Memory organization
- 9. Pipeline
 - a) Stages
 - b) Register renaming
 - c) Conditional branches and branch prediction
 - d) Out-of-order execution and code optimization
- 10. Reduced Instruction Set Computers (RISC)
 - a) Limitations of Complex Instruction Set Computers (CISC)
 - b) Fixed instruction sizes
 - c) Register stacks
 - d) Compiler design
- 11. Flynn's taxonomy
 - a) SISD – uniprocessors
 - b) SIMD – array (vector) processors
 - c) MISD – pipelined
 - d) MIMD – parallel machines (distributed systems and the internet)
 - e) Extensions to the taxonomy
- 12. Miscellaneous
 - a) Multimedia (graphical) operations
- 13. Course review
- 14. Final examination