Nicholas Hunt

Independent Study Report

Elements of Computing Systems

The Elements of Computing Systems is a thorough and easily digestible book on what it takes for a computer to recognize a set of instructions. Machine language, assembly, and high-level languages, are all covered within this book. By going through and completing the projects in each chapter, I have come to learn what it means to compile, translate, and assemble code. Other courses, such as Mechanics of Programming (MOP) and Concepts of Computing Systems (CCS), do a great job explaining the semantics of these operations. But this book forces you to complete these tasks through practice. There was no point where I felt lost in the book; it is thorough and explains every concept to its fullest. With that said, there are some things I would change before adapting this book to a graduate-level course.

I will start with what I felt weird about/did not like. This book goes well into hardware, and you will learn how to build the several types of logic gates that come into play within the hardware. However, I think it could do more. The CPU we generate supplies a minimalistic CPU that cannot multiply or divide, which is a missed opportunity, especially when the CPU developed in most CCS courses includes this arithmetic in the ALU. Furthermore, the book does not touch on floating-point numbers within the hardware. It gets around this by implementing a fraction library within the OS. It that takes in a numerator and denominator within its constructor. Having multiplication, division, and decimal numbers represented this high up in a language is not optimal for performance and not practical in the real world. If I were to present this as a viable course, we would have to address these issues. I suggest Intel's 8086 or 80186 CPUs, as they are old enough to be a bit tame and young enough to contain multiplication and division. Updating the CPU would also mean updating the book's Assembly instruction (It would just be a portion on the x86 language rather than the Hack language). This change would do the hardware portion of this class justice. I completed the hardware portion in six weeks, and these additions would take longer to go over. But students will have an extra week as we started this independent study a bit late. But of course, you could also add these instructions to the Hack language as it is already a made-up language!

As for what I found good, I feel confident with my understanding of compiler design and plan on taking more classes within the area. Elements from CS Theory, such as language recognition, came into play as I had to develop a syntax analyzer for the compiler portion of this course. This discovery was exciting when it all clicked. This book and its weekly projects always left me in awe as I saw everything connecting together. This book also starts off at breathable pace. Nothing should change within the first and third chapters and within the software section (chapters 6-12). Other chapters contain information on the ALU, machine code and assembly language specification (which I discussed above)! The software portion is easy to follow but very hard to implement and took me several hours and sessions to get working. It is definitely up to standard - in my humble opinion - for what RIT expects from a graduate level course.

Over the course of this semester, I took around three hours a week reading through each chapter, taking detailed notes. On top of that there were weekly projects. Hardware projects ranged from three to seven hours and 100 to 300 lines, and the software projects ranged from six to ten hours and 500 to 1500 lines. Notes range from four to ten pages. I estimate I spent around seven to fourteen hours a week working towards completing these assignments.

If I had to create a Mock Weekly schedule for the 15-week course it would be something like this:

Week #	Topics	Assignments Given	Events and Assignments Due
Week 1	Boolean Logic: Composition of Logic gates	Project 1: Logic Gates from Nand Gate	
Week 2	Boolean Arithmetic I: Composition of ALU without multiplication/division	Project 2: ALU design within HDL (without mult and div).	Project 1: Due
Week 3	Boolean Arithmetic II: Composition of Full ALU	Project 3: ALU design within HDL (complete)	Project 2: Due
Week 4	Memory: Memory devices and combination and sequential logic	Project 4: Memory and logic device design within HDL.	Project 3: Due
Week 5	Machine Learning: Going over the Assembly language of the processor chosen.	Project 5: Complete the selected assembly programs.	Project 4: Due
Week 6	Computer Architecture: Data busses	Project 6: Build your CPU within HDL language using your created ALU and memory devices.	Project 5: Due
Week 7	Computer Architecture II: Finish the processor described.		
Week 8	Assembler: How to implement an assembler	Project 7: Construct an assembler using Java or C++.	Project 6: Due
Week 9	Virtual Machine I: How to implement a virtual machine for the sake of easier compilation.	Project 8: Construct Push, Pop, and arithmetic VM translator instructions.	Project 7: Due
Week 10	Virtual Machine II: VM instructions such as function, goto, if-goto, call, etc.	Project 9: Complete VM Translator construction.	Project 8: Due
Week 11	High-Level Language: The Jack language.	Project 10: Complete the selected Jack language programs.	Project 9: Due
Week 12	Compiler I: Syntax Analysis: building a syntax Analyzer to recognize or reject a language.	Project 11: Complete the syntax analyzer within Java or C++.	Project 10: Due
Week 13	Compiler II: Code Generation: Code Compilation, writing VM instruction.	Project 12: Complete the Compiler within Java or C++.	Project 11: Due
Week 14	Operating System: Learn what an OS library needs to operate.	Project 13: Build the OS library within the Jack language, compile it into VM code.	Project 12: Due
Week 15	Finals Week		Project 13: Due