ECE 253 Lecture Notes

Hei Shing Cheung

Digital and Computer Systems, Fall 2025 ECE253

The up-to-date version of this document can be found at `https://github.com/HaysonC/skulenotes`

Chapter 1

Digital Circuits that Compute, Store, and Control

Introduction

Layers of Computation In hardware, we have the following layers of abstraction:

- Computation
- Adders
- Logic Gates
- Transistors
- Silicon

In this course, we will focus on the first three layers, on top of the logic gate level.

Layer of abstraction At this course, for the digital systems part, we would start from understanding logic gates, all the way to understanding computer architecture, with each level of abstraction hiding the details of the lower level.

1.0.1 Hierarchy, Modularity, and Regularity

Definiton 1.0.1.1 (Hierarchy)**.** The division of system into a set of modules, then further subdividing each module into smaller modules, and so on, until pieces are *easy* to understand.

Definiton 1.0.1.2 (Modularity)**.** The design principle that modules have well-defined functions and interfaces so they connect easily without unintended side effects.

Definiton 1.0.1.3 (Regularity)**.** The uniformality of modules, such that the reusability of common modules reduces the number of distinct modules to be designed.

1.0.2 Digital Logic Gates

Logic gates are made out of transistors:

Definiton 1.0.2.1 (Transistor)**.** A transistor is a 3-terminal device behaving as a switch. When the voltage on the terminal is HI, the switch is closed, and when the voltage is LO, the switch is open.

Factors Affecting Speed of Digital Circuits

- **Transistors and Electrons take time to switch.** A transistor (State of the Art) takes 2-3 picoseconds to switch. Gates takes 40 ps and an 8-bit adder takes 300 ps.
- **Wires take time to propagate signals.** Signals travel at approximately 2/3 the speed of light in a vacuum, which is about 200,000 kilometers per second in a typical silicon wire.
- **Capacitance** There would be RCL circuits formed by the wires and transistors, which would cause delay.

1.1 Information Representation

1.1.1 Number Systems

Definiton 1.1.1.1 (Number System)**.** A number system is a way of representing numbers using a set of symbols (digits) and a base (radix). The base determines the number of unique digits that can be used in the number system.

Common Number Systems You should be familiar with the following number systems:

- Decimal (Base 10): Digits 0-9
- Binary (Base 2): Digits 0-1
- Hexadecimal (Base 16): Digits 0-9, A-F

In computer systems, we use binary to represent information, and we would often use hexadecimal to represent binary numbers in a more compact way - a group of 4 bits (a nibble) can be represented by a single hexadecimal digit.

Example 1.1.1.2 (Binary, Decimal, and Hexadecimal Numbers)**.** Below is a table showing the conversion of binary numbers to decimal numbers, along with their hexadecimal representation.

Binary	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Table 1.1: Binary to Decimal and Hexadecimal Conversion

Example 1.1.1.3 (Decimal to Binary Conversion)**.** To convert a decimal number to binary, we can use the method of successive division by 2. For example, to convert the decimal number 437 to binary:

$$\begin{aligned}
437 \div 2 &= 218 && \text{remainder } 1 \\
218 \div 2 &= 109 && \text{remainder } 0 \\
109 \div 2 &= 54 && \text{remainder } 1 \\
54 \div 2 &= 27 && \text{remainder } 0 \\
27 \div 2 &= 13 && \text{remainder } 1 \\
13 \div 2 &= 6 && \text{remainder } 1 \\
6 \div 2 &= 3 && \text{remainder } 0 \\
3 \div 2 &= 1 && \text{remainder } 1 \\
1 \div 2 &= 0 && \text{remainder } 1
\end{aligned}$$

Reading the remainders from bottom to top, we get the binary representation of 437

Example 1.1.1.4. To convert $(512000)_{10}$ to binary, we recognize that $512000 = 2^9 \times 1000$. We know that $2^9 = 512$ and $1000_{10} = 1111101000_2$ (by method outlined above). Therefore, we can shift the binary representation of 1000 left by 9 bits to get the binary representation of 512000:

$$(512000)_{10} = (1111101000000000000)_2$$

1.2 Digital Storage Elements

1.3 Finite State Machines (FSM)

Chapter 2

Computer Organization and Assembly Language

What is Assembly Language? We know that we can run C/C++ on any computer (Machine Agnostic), but how does the computer understand C/C++? The answer is the compiler that parse it to assembly through:

1. **Front-end Parser:** The front-end parser would parse the C/C++ code into an intermediate representation (IR), which is a low-level representation of the code that is easier to optimize. The front-end parser would also perform optimizations on the IR, such as loop unrolling, inlining, and dead code elimination.

2. **Back-end Parser:** The back-end parser would take the optimized IR and generate assembly code for a specific architectures.

The assembly code is then assembled into machine code, which is a series of 0s and 1s that the computer can understand. The assembly would be specific to the architecture of the computer (machine dependent), which is why we have different assembly languages for different architectures (e.g., x86, RISC-V, ARM).

2.1 Computer Organization

2.2 Assembly Language