0.1 Preface

This textbook is an open-source document that meets the requirements of COMP 142 Computer Architecture and Organization. The goal of the class is to study the organization and behavior of real computer systems at the assembly-language level, the mapping of statements and constructs in a high-level language onto sequences of machine instructions is studied, as well as the internal representation of simple data types and structures. It also covers numerical computation, noting the various data representation errors and potential procedural errors.

The general topics covered by the class are:

1. Bits, bytes, and words
2. Numeric data representation and number bases
3. Fixed- and floating-point systems
4. Signed and twos-complement representations
5. Representation of nonnumeric data (character codes, graphical data)
6. Representation of records and arrays
7. Basic organization of the von Neumann machine
8. Control unit; instruction fetch, decode, and execution Instruction sets and types (data manipulation, control, I/O)
9. Assembly/machine language programming
10. Instruction formats
11. Addressing modes
12. Subroutine call and return mechanisms
13. I/O and interrupts

Chapter 1

Data Representation

Computers use binary numbers because they are made up of electronic components called transistors, which can be either in an on" or "off" state. By using a binary numbering system, which only has two digits (0 and 1), computers can represent and process information using these "on" and "off" states of transistors.

A binary number is a number expressed in the base-2 numeral system, a method of mathematical expression which uses only two symbols: typically 0 or 1. In more mathematical or logical contexts 0 is treated as False and 1 is treated as True.

The base-2 numeral system is a positional notation with a base of 2. Each digit is referred to as a bit, or binary digit. Because of its straightforward implementation in digital electronic circuitry using logic gates, the binary system is used by almost all modern computers and computer-based devices, as a preferred system of use, over various other human techniques of communication, because of the simplicity of the language.

The modern binary number system was studied in Europe in the 16th and 17th centuries by Thomas Harriot, Juan Caramuel y Lobkowitz, and Gottfried Leibniz. However, systems related to binary numbers have appeared earlier in multiple cultures including ancient Egypt, China, and India. Leibniz was specifically inspired by the *I Ching*, a classical Chinese text from the 9th century BCE.

1.1 Binary Numbers

You can think of a number base as the way to represent a number. The value of the number does not change when transitioning between number

bases. Most people use a base of 10, also known as decimal. For example, 667 is a decimal number. In more mathematical contexts with numbers of different bases, you can write the number in parenthesis with the base in subscript, for example: 667_{10}. Binary numbers are base 2. 667 in base 2 is representated as 1010011011_2. When spoken, binary numerals are usually read digit-by-digit, in order to distinguish them from decimal numerals. For example, 1010011011_2 is pronounced *one zero one zero zero one one zero one one*. It would be confusing to refer to the number as *one billion ten million eleven thousand and eleven* which represents a different value. Note that the number base should not change the intrinsic value of a number.

The order of the digits in a binary number represents their power of two. The right-most digit called the least significant bit (LSB) represents the power of 2^0. The left-most digit called the most significant bit (MSB) represents the power of 2^{n-1} where n is the length of the sequence. For example, with 1010011011_2 the LSB is 1, representing 2^0, and the MSB is 2^9 because the sequence is 10 bits long. A sequence of binary numbers can be written as an expansion of power of two:

$$1\times2^9+0\times2^8+1\times2^7+0\times2^6+0\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0 \quad (1.1)$$

When converting from binary to decimal, expand the terms as above. This is the simplest way to convert a binary number to decimal.

1.1.1 Converting to Binary Numbers

Suppose that you want to convert 667_{10} to decimal. Expanding the number in terms of powers of two is the simplest way to convert a decimal number to binary (such as in Equation). Prepare a table with the power of two that is just less than the magnitude of the number you are converting. For 667_{10}, this is 512_{10} or 2^9:

Power	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal	512	256	128	64	32	16	8	4	2	1
Digit										

Start from left to right. If you can subtract the number without it becoming negative, indicate 1 for the digit. Then, carry out the subtraction and use this new value for the next column. If you cannot carry out the subtraction without the number becoming negative, skip to the next column. Repeat this procedure for the next column.

$667 - 512 = 155$, so we can indeed subtract 512_2 from 667_2. We can note 1 in the digit, and use the value of 155 for the next column. $155 - 256 < 0$. We cannot subtract without the number becoming negative, so we note 0 as the digit for this current column. However, $155 - 128 = 27$. So we can note 1 in the digit for this next column. And so on until the last digit. If you have any remaining value beyond the right-most column you have made a mistake, check your work.

Power	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal	512	256	128	64	32	16	8	4	2	1
Digit	1	0	1	0	0	1	1	0	1	1

There is a more formal way to carry out this procedure. Essentially, the digit column is noting if there would be a remainder when carrying out integer division by its respective power of 2. An alternative way to convert the number is as follows.