1

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

Number Bases

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 underlying transitor-level components.

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.

The purpose of thise chapter is to understand binary numbers. Examples will describe how to convert from decimal numbers to binary. Alternative representations, such as hexadecimal, will also be covered. Finally, TODO: how they are grouped in the computer using bytes or bibytes,

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. When you work with numbers of different bases, write the number in parenthesis with the base in subscript, for example: 667_{10}. Binary numbers are base 2. 667_{10} 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 is called the least significant bit (LSB) represents the power of 2^0. The left-most digit is 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. Typically with computers, there is a fixed number of digits.

Example 1.1 Convert 1010011011_2 to decimal by expanding the powers of two. The most straightforeward way to convert this number is to expand it in of powers of two:

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

Note that we write out all powers of two starting with $n - 1$. In this case $n = 10$ because there ten digits. Begining with the MSB and ending with the LSB copy 0 or 1 based on the corresponding digit in the binary representation of the number. This may seem cumbersome but after some time you will memorize the powers of 2 and it will become easier.

It is important to note the largest positive integer that can be stored for a binary sequence of a given length to avoid a concept called overflow. For unsigned numbers this is defined as:

$$2^n - 1 \quad (1.2)$$

Where n is the number of digits in the binary sequence.

Example 1.2 What is the largest number that you can store in an unsigned C-language `int` data type? Note that `int` is 32 bits, so $n = 32$

$$2^n - 1$$
$$2^{32} - 1 = 4,294,967,295$$

Perhaps this is somewhat shocking. A C-language `int` cannot hold numbers greater than four billion.

Example 1.3 Suppose that you want to use binary numbers to encode specific letters of the Spanish language alphabet which has 27 letters. At least how many bits will you need to uniquely encode each letter?

$$2^n - 1 \geq 27$$
$$2^n \geq 28$$
$$n \geq \log_2 28$$
$$n \geq 4.8 \approx 5$$

You cannot have a fractional number of digits, so it must be at least 5 bits.

1.1.1 Converting to Binary Numbers

Expanding the number in terms of powers of two is the simplest way to convert a decimal number to binary (such as in Equation 1.1). To do so, 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, called the **divide-by-2** method. Essentially, with the previous algorithm 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.

Start with the number you want to convert. Perform an integer division by 2. Note that an integer division does not produce a fractional result. It should produce an integer and a remainder if the divisor cannot cleanly divide the dividend. Write the remainder to the right of the calculation, and write the result *below* your calculation. Continue dividing by 2 until you reach a dividend of 1. The binary representation of the number is read from top-to-bottom of the remainder values.

Homework Questions

1.1 Conduct a search on the historical basis for binary numbers before they were used in computing and explain how they were used by society.

1.2 Explain in your own words the following concepts:

(a) Most significant bit

(b) Least significant bit

(c) The largest unsigned number that can be stored in n bits.

1.3 Define the divide-by-2 method in pseudo-code.

1.4 What is the largest unsigned number that can be stored in the following data types?

(a) `char`

(b) 2-byte `int`

(c) 48-bit integer

1.5 Convert the following decimal numbers to binary:

(a) 0

(b) 1

(c) 23

(d) 59

(e) 100

(f) 1200

(g) 1092

(h) 1000000

1.6 Convert the following binary numbers to decimal using the table method:

(a) 0

(b) 1

(c) 1010

(d) 1111

(e) 1110101

(f) 1010100101

(g) 1000000000

(h) 1000100101

1.7 Repeat Question 1.6 using the divide-by-2 method.

Acronyms

LSB least significant bit. 4

MSB most significant bit. 4

Glossary

overflow Exceeding the capacity of a binary number with fixed number of digits. 4

unsigned A data type that allows for only positive numbers or operation that assumes the operands are positive numbers. 4