ICS Lecture Review 1

Chapter 1 -- Welcome Aboard

1. Two Recurring Themes

1.1 Abstraction

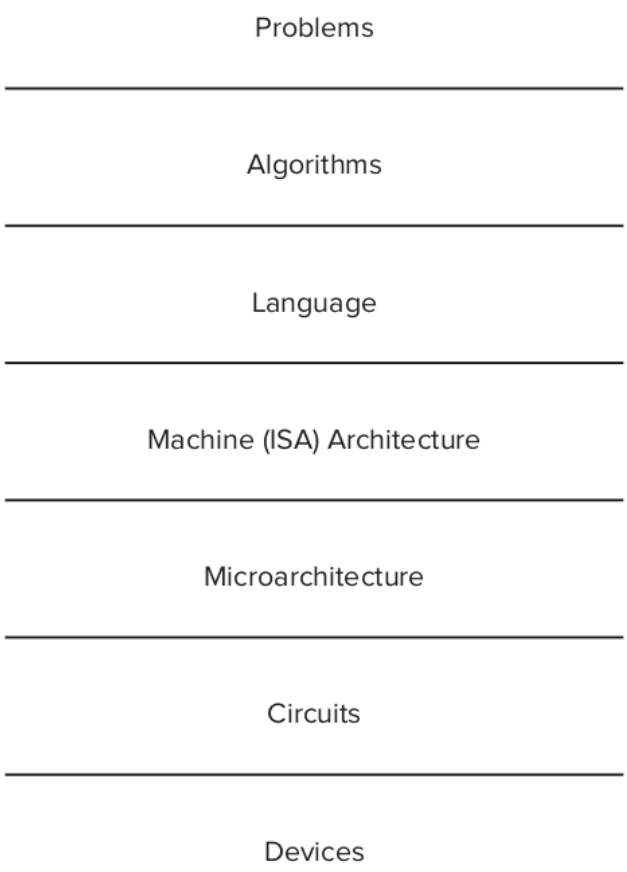
Too much or too tedious **low-level details** are abstracted into **high-level interfaces** that are easy to understand and use, thereby improving **efficiency**.

1.2 Hardware vs. Software

- If software developers can understand the hardware implementation, they can use its characteristics to write more efficient code.
- If hardware developers can understand the needs of the software, they can make optimizations that are more conducive to the software.

2. How Do We Get the Electrons to Do the Work?

2.1 Transformation Level



2.2 The Statement of the Problem

The statement of the problem should have **No Ambiguity**.

2.3 The Algorithm

- definiteness: each step is precisely stated.
- effective computability: each step can be carried out by a computer.
- finiteness: the procedure terminates after finite steps.

2.4 The Program

Transform the algorithm into a computer program in one of the programming languages precisely.

- high-level languages: they are independent of the computer on which the programs will execute. We say the language is “machine independent.

- low-level languages: they are tied to the computer on which the programs will execute. There is generally one such low-level language for each computer. That language is called the *assembly language* for that computer.

2.5 The ISA

The link between software and hardware (I think), which contains:

- Set of instructions
- Data types
- Address ability: how many bytes per memory-slot
- Address mode: to figure out where the operands are located

.....

If you want to understand what ISA includes and how a specific ISA (LC-3) is implemented, please read Appendix A of this book.

2.6 The Microarchitecture

The implementation of ISA. Generally, one microarch only supports one ISA, but one ISA can be supported by many microarchs. These depend on the cost/performance tradeoffs.

Microarch has many components, in this course, we only discuss the **CPU**

2.7 The Logic Circuit

2.8 The Devices

Discuss later in Chapter 3.

Chapter 2 -- Bits, Data Types, and Operations

1. Bits, Bytes

1. bit: only 0/1
2. byte: 1 byte = 8 bits

2. Data Type

2.1 Integer

2.1.1 unsigned integer

n-bit unsigned integer can represent: $[0,\ 2^n - 1]$

2.1.2 signed integer

signed-magnitude: a leading 0 signifies a positive integer, a leading 1 signify a negative integer. In a 4-bit example, $0110 = 6,\ 1110 = -6$， can represent [-7, 7], has the problem of "positive zero" and "negative zero".

1's Complement: For a non-negative number, its opposite number is obtained after bitwise inversion。 In a 4-bit example, $0110 = 6,\ 1001 = -6$， can represent [-7, 7], also has the problem of "positive zero" and "negative zero".

Now focus on the most widely used representation: 2's Complement

The highest bit of the 2's complement is the sign bit. The sign bit is 0 for non-negative numbers, and 1 for negative numbers. For an n-bit signed number, the weight of the sign bit is -2^{n-1}.

Representation range of n-digit signed number:
$[-2^{n-1},\ 2^{n-1}-1]$

From an intuitive point of view:

- The 2's complement of a non-negative number is to directly convert it to binary
- The 2's complement of a negative number is:

  Find the binary representation of its absolute value;

  invert every bits, then +1.

  In a 4-bit example, $0110 = 6,\ 1010 = -6$，

2.1.3 Extension

- sign extension: Fill sign bit when extending.
- zero extension: Fill 0 when extending.

2.1.4 overflow

The only possible overflow situations:

- positive + positive == negative, that is, carry to the sign bit, and the sign bit becomes 1 after adding
- negative + negative == positive，the sign bit becomes 0 after adding

2.2 Floating Point

We mainly introduce the **32-bit** normalized floating-point number representation under the **IEEE754 standard**

Sign	Exponent	Fraction
[31]，1 bit	[30:23]，8 bits	[22:0]，23 bits

Sign: 0 for non-negative, 1 for negative.

Exponent: 8 bits **unsigned integer**，can represent 0 ~ 255，When converting to the real exponent, subtract 127, and the final representation range is -127 ~ 128 (Note: 8-bit signed numbers represent the range -128 ~ 127)

Fraction: digits after point

the digit before point is always 1 so it's unnecessary to allocate a bit for it.

Example:

Convert 1 10000001 1010 0000 0000 0000 0000 000 to decimal representation

Sign	Exponent	Fraction
1	1000 0001	1010 0000 0000 0000 0000 000

$$-(1.101)_2 \times 2^{10000001_2 - 127_{10}} \ = \ -1.625 \times 2^2 = -6.5$$