# 「计算机系统概论」TA Session 1

2022-2023 短学期计算系统概论课程

By @HobbitQia

# Chapter 1 Welcome Abroad

# 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*.

***e.g.*** 打车时，告诉司机每一个路口怎么走 vs 直接告诉他目的地

There is an underlying assumption, when *everything about the detail is just fine.*

***e.g.*** 打车去机场，结果被绕路

在 ICS 这门课中，我们会从晶体管开始，一步一步提高我们的抽象层次，直到我们能够使用高级语言来编写程序。

# Hardware vs. Software

> *Someone says he is a hardware person, and someone says he is a software person. I would say I am a person! By Patt*

Both hardware and software are important!

# Computer System

计算机主要由这几部分组成

- CPU
- Memory 内存
- I/O
  - Input ***e.g.*** 键盘、鼠标
  - Output ***e.g.*** 显示屏

    More details will be covered in Chapter 4.

# Computing Ability

- 所有的计算机，他们的计算能力是相同的（能解决的问题相同），只是需要的时间和空间可能不同。
- 如果有足够多的时间和空间，所有的计算机都可以完成同样的任务。

# Level of Transformation

把一个问题转化为计算机能够解决的问题，需要经过多个层次的转化。

| **Problem (Natural language)** |
| --- |
| Algorithm (eliminate ambiguity) |
| Program (Python, C++, ...) |
| Instruction set architecture (ISA)( 指令集架构 ) |
| Micro-architecture ( 微架构 ) |
| Electronic circuit ( 通过电势差来使电子移动 ) |
| Electrons |

# Level of Transformation (Cont.)

- **Problem**
  用自然语言描述，可能会有歧义 ***e.g.*** "安"
- **Algorithm** Algorithm is a procedure step by step.
  - *definite* 没有歧义（和自然语言描述的问题的区别）
  - *effective computability* 每一步要能被计算机执行
  - *finite* 算法应该在有限时间内终止
- **Program**
  - high-level language ***e.g.*** Python, C++, Java...
  - low-level language ***e.g.*** LC-3, MIPS, RISC-V

# Level of Transformation (Cont.)

- **ISA**
  ISA is the interface between software(program) and hardware.
  - ISA contains opcode, data type, addressing mode, addressability.
- **Microarchitecture** ISA 对应的物理实现
- Circuit

# Level of Transformation (Cont.)

- **ISA**
  ISA is the interface between software(program) and hardware.
  - ISA contains opcode, data type, addressing mode, addressability.
- **Microarchitecture** ISA 对应的物理实现
- Circuit

当我们的程序转化为 01 字符串时，ISA 规定了 01 字符串的功能（如 add 两个数）而微结构是其对应的物理实现（比如如何实现加法、加法的两个算子存在哪里）。因此对于同样的 01 字符串，其实现的功能相同，但可以有各种不同的物理实现；但一套微结构只能实现一类 ISA。

# Chapter 2 Bits, Data Types and Operations

# Bits

计算机通过电子的流动来进行计算，计算机内部的各个部件会对电压的高低做出反应，因此控制电子的流动，我们简单的将电压存在（大于 0）定义为“1”，而电压不存在（为 0）定义为“0”，我们把这样的“1”和“0”称为 **bit （比特）**。

- 一个 bit 有两个状态，我们可以用多个 bit 组合来表示更多的状态。这样得到的组合就是二进制串（binary string）
  $n$ bits，可以表示 $2^n$ 个不同的状态。
- Bits are just bits. 关键在于我们如何理解这些位。

# Integers

- **Unsigned Integer 无符号数**：对于二进制串 $a_{n-1}a_{n-2}...a_1a_0$
  其中 $a_i$ 对应的权重为 $2^i$，因此这个二进制串对应的无符号数为
  $a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + ... + a_1 \times 2^1 + a_0 \times 2^0$
  因此 $n$ bits 的二进制串可以表示的数为 $[0, 2^{n-1}]$

***e.g.*** 4 位二进制串 $1110$ 表示的数字为 14。

- **Signed Integer 有符号数**：
  - Signed-Magnitude (原码)
  - 1's Complement (反码)
  - 2'S Complement (补码)

# Signed-Magnitude

$n$ 位，最高位（$a_{n-1}$）用来表示正负号（"0" 表示正数，"1" 表示负数），剩余 $n-1$ 位同无符号数。

***e.g.***

$$5 = 0101$$

$$-5 = 1101$$

缺点：

- 两个数相加不能直接得到以该表示方法表示的正确结果，需要额外做转化。
- 存在冗余：+0 (0000) or -0(1000)

# 1's Complement

- 对于一个非负数，它的反码是它本身。
- 对于一个负数，它的反码是它的相反数的二进制串按位取反。

***e.g.***

$$5 = 0101$$

$$-5 = 1010$$

缺点：

- 同样不能直接得到正确的结果。
- 存在冗余：+0 (0000) or -0(1111)

# 2'S Complement

- $n$ 位，$a_{n-1}$ 对应的权重为 $-2^{n-1}$，其余位 $a_i$ 对应的权重依然为 $2^i$，因此这个补码表示的数为 $a_{n-1} \times \left(-2^{n-1}\right) + a_{n-2} \times 2^{n-2} + \ldots + a_1 \times 2^1 + a_0 \times 2^0$
  因此，$n$ 位补码能表示的范围为 $\left[-2^n, 2^{n-1}\right]$
- 类似于反码，区别在于求一个数的相反数的补码是按位取反再加 1.

***e.g.***

$$5 = 0101$$

$$-5 = 1011$$

# 2'S Complement

- $n$ 位，$a_{n-1}$ 对应的权重为 $-2^{n-1}$，其余位 $a_i$ 对应的权重依然为 $2^i$，因此这个补码表示的数为 $a_{n-1} \times \left(-2^{n-1}\right) + a_{n-2} \times 2^{n-2} + \ldots + a_1 \times 2^1 + a_0 \times 2^0$
  因此，$n$ 位补码能表示的范围为 $\left[-2^n, 2^{n-1}\right]$
- 类似于反码，区别在于求一个数的相反数的补码是按位取反再加 1.

***e.g.***

$$5 = 0101$$

$$-5 = 1011$$

**Question**：假设 $n = 4$，-8 的补码是？8 的补码是？

# Arithmetic

- 对于加减法
  - 方式和十进制的类似，逢二进一。
  - 注意，两个二进制数位数不同的时候需要先将位数少的那个数符号扩展，再进行运算。
    - Signed-Extending 符号扩展对于正数来说是在高位补充 0；对于负数是在高位补充 1（即填充符号位）。
  - 计算 $A-B$ 相当于计算 $A+(-B)$，即先计算 B 的相反数（补码下只需要按位取反再加一），再将 B 的相反数和 A 相加。
  - Overflow 溢出：当两个正数相加得到的结果是负数，或者两个负数相加得到的结果是正数，就称为溢出。

***e.g.*** 假设 $n=4$，计算 $0111(7)+0010(2)=1001(-7)$

# Arithmetic (Cont.)

- 对于二进制数和十进制的转换
  - 二进制转十进制
    每一位的权重乘以对应的位数，再相加（$\sum_i a_i \times 2^i$）
  - 十进制转二进制
    用 2 乘十进制小数，将积的整数部分取出，再用 2 乘余下的小数部分，再将积的整数部分取出，如此直到积中的整数部分为 0/1，此时 0/1 为二进制的最后一位。或者达到所要求的精度为止。
    然后把取出的整数部分按顺序排列起来，先取的整数作为二进制小数的高位有效位，后取的整数作为低位有效位。

# Logical Operations

- 逻辑运算: AND, OR, NOT, XOR

$$X \cdot Y \Leftrightarrow X\ AND\ Y$$
$$X + Y \Leftrightarrow X\ OR\ Y$$
$$X \oplus Y \Leftrightarrow X\ XOR\ Y$$
$$\overline{X} \Leftrightarrow NOT\ X$$

- DeMorgan's Law
- $a\ AND\ 1 = a, a\ OR\ 0 = a$

# Floating Point Numbers

我们把浮点数的位码拆分为三个部分：

| | sign | exponent | fraction |
|---|---|---|---|
| float | 1 | 8 | 23 |
| double | 1 | 13 | 52 |

这样的位码得到的数为 $N = (-1)^S \times M \times 2^E$

- S: sign bit. 0 表示正数，1 表示负数。
- E: 阶码。$E = exp - Bias$，$exp$ 就是 exponent 部分表示的无符号数，$Bias$ 是一个固定的数，对于 float 是 127，对于 double 是 1023。
- M: 尾数。$M = 1.frac$，$frac$ 就是 fraction 部分的二进制串。

# Floating Point Numbers (Cont.)

上面是浮点数的 normalized form，即规格化表示。浮点数还有下面的非规格化的情况：

- $exp = 0$ 时规定 $M = 0.frac$
  其中 $frac = 0$ 时，表示的数字为 0.0 （有 +0.0 和 -0.0）
- $exp = 1111\ 1111$ 即全 1 时
  - 若 $frac = 0$ 则这个数表示 $+inf / -inf$
  - 若 $frac \neq 0$ 则这个数表示 $NaN$ （Not a Number）

# Hexadecimal Representation

- 十六进制表示法是为了让我们人类更轻松的读懂二进制 01 串。
- 以 4 个 bits 为单位，将这 4 bits 看成一个十六进制数。
  $A = 1010, B = 1011, C = 1100, D = 1101, E = 1110, F = 1111$

  ***e.g.***
  110101100010 $\rightarrow$ 1101 | 0110 | 0010 $\rightarrow$ D62

# Hexadecimal Representation

- 十六进制表示法是为了让我们人类更轻松的读懂二进制 01 串。
- 以 4 个 bits 为单位，将这 4 bits 看成一个十六进制数。
  $A = 1010, B = 1011, C = 1100, D = 1101, E = 1110, F = 1111$

  ***e.g.***
  110101100010 $\rightarrow$ 1101 | 0110 | 0010 $\rightarrow$ D62

**Question：** 1000111 对应的十六进制数是?

# 谢谢大家

Question?