# Introduction to Computer System Organization

## SZU Review

## Chapter2

## 数据表示

## Data Expression

## 1.数值表示:

进制转换(e.g.Dec. to bin.)

## 2.符号表示:

最高位(Sign Bit)

1:negative(代表为负数)

0:positive(代表为正数)

## 3.小数点的表示:

1. 定点数: 约定好小数点的位置，不去表达

   缺陷:能表达的数据类型十分有限:正整数、纯小数（int long etc.）

2. 浮点数: 小数点的位置可以浮动(原理:科学计数法)

## 4.定点数的表示:

1. 原码: 直观、不易于计算(不利于在硬件上实现)

   相关问题:正0与负0的表示

   以 8 bit 为例:

   正0: 00000000

   负0: 10000000

2. 补码: 解决在硬件电路上的计算问题

   (1). positive(正数): 补码与原码相同.

   (2). negative(负数): 原码各位按位取反,末位+1

   优点: 方便在硬件上实现计算

   **comment:**

   1. 符号位具有数值含义,可以直接参与运算.
   2. "0" 表示唯一，避免在硬件上实现时，因为"0"的表示的二义性而造成一系列不必要的麻烦。
   3. 减法可以转换为加法来进行计算(**Point:补码运算**)

3. 反码:主要功能是用于求补码。

   将负数的求补操作转化为"取反加1"的操作(更快捷地求负数的补数)

4. 移码:(浮点数的)阶码

# 5.补码运算

**Formula:**

> (A+B)补 = A补 + B补
>
> (A-B)补 = A补 + (-B)补

**comment:在未溢出情形下(Correct!)**

**Points:**

1. 每次运算后都要**判溢出**(补码运算后作溢出判断)
2. 常见的判溢出的方法:

   针对运算过程及其对应的运算结果:

   **两正得一负**

   **两负得一正**

   **conclusion:一定溢出.**

   **一正一负**

   **conclusion:永不溢出.**

3. 计算机中的完整运算过程包括:**结果+溢出判断**

# 6. 浮点数的表示:

任意小数 --> M · $2^E$

唯一性处理:

s.t. M 必须是**规格化**的.

规格化(正则化): only this type of number can we accept:

Normalization

> 0.x × $2^E$

x为1-9的正整数

非规格化数字带来的意外(accidents):

0.09 × $10^2$ = 0.9 × $10^1$

前者为非规格化的M(0.09)，显然等式左右两端想等，由此产生的同一个数字的多种表达使我们不希望在硬件实现时所看到的。

### IEEE 754

附一个十分好用的网站:https://www.h-schmidt.net/FloatConverter/IEEE754.html

1. 32 bit: 1 + 8 + 23 (float)

1(S):符号位

8(E)(exponent):指数(1<=指数<=254)

23(fraction):尾数(数值精度)

2. 64 bit: 1 + 11 + 52 (double)

**Formula:**

> (-1)$^s$·(1.M) × $2^{E-127}$

s:符号位

(1.M):隐藏位计数

E-127:移码

3. **Examples:**

   1. 0 10000011 00101000000000000000000

   指数字段的内容是131,131-127 = 4,故指数值为+4,在小数点左边补1，并将尾数字段放在小数点右边,则得1.00101.如果将小数点右移4位(等价于1.x·$2^4$),则得10010.1即18.5.

   2. 1 10000010 00101000000000000000000

   符号位为1表示负数。指数字段为130,表示指数值等于130-127或+3.在尾数字段前小数点左边添加1，得1.00101.将小数点右移+3位,则得1001.01.所以结果为-9.25.

   3. 试用IEEE浮点数标准表示-$6\frac{5}{8}$.

   首先将-$6\frac{5}{8}$表示成二进制数-110.101，即:

   -$6\frac{5}{8}$ = -(1 × $2^2$ + 1× $2^1$ + 0 × $2^0$ + 1 × $2^{-1}$ + 0 × $2^{-2}$ + 1 × $2^{-3}$)

   正则化处理之后，即为-1.10101 × $2^2$。

符号位为1，表示 $-6\frac{5}{8}$ 是一个负数。指数部分位10000001，即无符号数129，代表实际指数为+2(129-127 = +2)。尾数部分省略小数点左边的1之后，精度为23位。因而，尾数部分等于10101000000000000000000.由此，$-6\frac{5}{8}$ 的IEEE浮点数表示为:1 10000001 10101000000000000000000

**comment:正常的思维顺序应该是:**

**先将数据正则化处理，将实际指数+127，从而得到无符号的指数，再将无符号的指数转换为2进制作为IEEE754浮点数下的指数部分，符号位(首位)由原数据的正负来确定，最后再取正则化过后数据的小数点后部分的数据，补全0至23位得到IEEE754浮点数下的尾****数部分。**

## 7. 位运算

### 常用位运算操作总结

| 运算符 | 名称 | 描述 | 示例（二进制） | 示例（十进制） | 常见应用场景 |
| --- | --- | --- | --- | --- | --- |
| `&` | 按位与 | 两位都为1时结果为1 | `1010 & 1100 = 1000` | `10 & 12 = 8` | 掩码操作、奇偶判断 |
| `\|` | 按位或 | 任意一位为1时结果为1 | `1010 \| 1100 = 1110` | `10 \| 12 = 14` | 设置特定位 |
| `^` | 按位异或 | 两位不同时结果为1 | `1010 ^ 1100 = 0110` | `10 ^ 12 = 6` | 交换变量、数据加密 |
| `~` | 按位取反 | 0变1，1变0 | `~1010 = 0101` (4位示例) | `~10 = -11` | 求补码、位翻转 |
| `<<` | 左移 | 高位丢弃，低位补0 | `1010 << 2 = 101000` | `10 << 2 = 40` | 快速乘2的幂、位域组合 |
| `>>` | 算术右移 | 低位丢弃，高位补符号位 | `1010 >> 2 = 1110` (有符号) | `-6 >> 1 = -3` | 快速除2的幂（有符号数） |
| `>>>` | 逻辑右移 | 低位丢弃，高位补0 | `1010 >>> 2 = 0010` | `-1 >>> 1 = 2147483647` | 无符号右移、高位清零 |