

Lecture 3: 知识 - 数制与编码

大纲: (一) 数值与编码

- 进位计数制及其数据之间的相互转换;
- 定点数的编码表示;

进位计数制及其相互转换

概念: 二进制编码

- 真题: 2018年
- 在计算机系统内部, 所有信息都是用二进制进行编码的, 这样做的原因有以下几点:
 - (1) 二进制只有两种状态, 使用有两个稳定状态的物理器件就可以表示二进制数的每一位, 制造成本比较低, 例如用高低电平或电荷的正负极性都可以很方便地表示 0 和 1;
 - (2) 二进制位 1 和 0 正好与逻辑值“真”和“假”相对应, 为计算机实现逻辑运算和程序中的逻辑判断提供了便利条件;
 - (3) 二进制的编码和运算规则都很简单, 通过逻辑门电路能方便地实现算术运算;

进位计数法

概念: 进制基本概念

- 概念:
 - 常用的进位计数法有十进制、二进制、八进制、十六进制等;
- 基数:
 - 每个数位所用到的不同数码的个数称为基数;
 - 十进制的基数为 10 (0~9);
- 位权:
 - 概念:
 - 每个数码所表示的数值等于该数码本身乘以一个与它所在数位有关的常数, 这个常数称为位权;
 - 一个进位数的数值大小就是它的各位数码按权相加;
 - 公式:
 - 一个 r 进制: $K_n K_{n-1} \dots K_0 K_{-1} \dots K_{-m}$
 - r 是基数; r^j 是第 i 位的位权; K_i 的取值可以是 $0, 1, \dots, r-1$ 共 r 个数码中的任意一个
 - r 的数值:

$$K_n r^n + K_{n-1} r^{n-1} + \dots + K_0 r^0 + K_{-1} r^{-1} + \dots + K_{-m} r^{-m} = \sum_{i=n}^{-m} K_i r^i$$

概念: 三种进制

- 二进制:

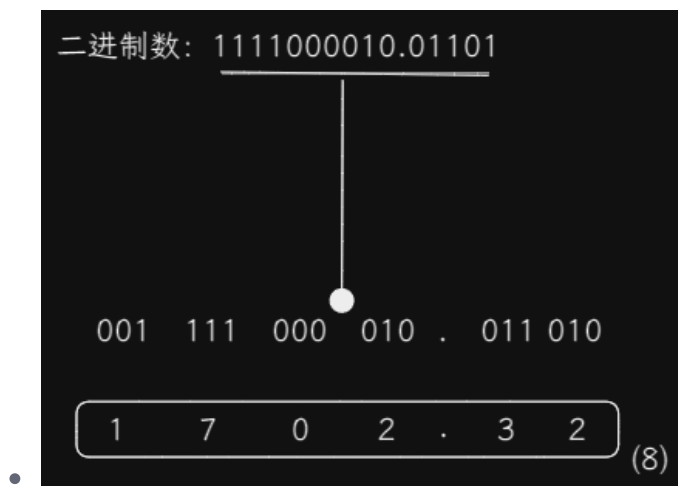
- 任意位数的权为 2^i
- i 为所在的位数;
- 八进制:
 - 二进制中的 3 位数码编为一组就是 1 位八进制数码;
- 十六进制:
 - 基数为 16, 有 0~9、A~F 共 16 个不同的数码, 其中 A~F 分别表示 10~15;
 - 4 位二进制数码与 1 位十六进制数码相对应;
- 后缀字母:
 - 用 B 表示二进制, 用 O 表示八进制, 用 D 表示十进制 (通常直接省略), 用 H 表示十六进制;
 - 有时也用前缀 0x 表示十六进制数;

进制数转换

二、八、十六转换

概念: 二进制数 $\rightarrow$ 八进制 & 十六进制

- 概念:
 - 二进制混合数是既有整数、又有小数的数, 转换时需要以小数点为分界;
- 实现:
 - (1) 拆分整数、小数
 - 整数部分:
 - 其整数部分, 从小数点开始往左数, 将一串二进制数分为 3 位 (八进制) 一组或 4 位 (十六进制) 一组;
 - 在数的最左边可根据需要加“0”补齐;
 - 小数部分:
 - 对于小数部分, 从小数点开始往右数, 也将一串二进制数分为 3 位一组或 4 位一组;
 - 在数的最右边也可根据需要加“0”补齐;
 - (2) 转换
 - 第 (1) 步得到的总位数为 3 或 4 的整数倍, 分别用对应的八进制数或十六进制数取代;
- 图示:
 - 二进制 $\rightarrow$ 八进制



概念: 八进制 & 十六进制 $\rightarrow$ 二进制

- 实现: 将**每位**逐个改为 3 位或 4 位二进制数即可 (必要时去掉整数最高位或小数最低位的 0);

概念: 八进制 $\leftrightarrow$ 十六进制

- 实现: 使用二进制数作为中介, 实现八进制 $\leftrightarrow$ 十六进制数的转换;
- 1. 将十六进制 (八进制) 数转换为二进制数;
- 2. 由二进制数, 转换为八进制 (十六进制)数;

十进制转换

概念: 任意进制 $\rightarrow$ 十进制

- 实现: 按权展开法
 - 将任意进制数的**各位数码与它们的权值相乘**, 再把乘积相加, 就得到了一个十进制数;
- 举例:

$$(11011.1)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} = 27.5$$

- 补充: 二进制小数 $\rightarrow$ 十进制数

$$[1.01]_B = 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 1.25$$

- 注意:
 - 整数部分的**第一位是从 2^0 次方**开始计算, 小数部分是从 2^{-1} 次方开始计算的;

概念: 十进制 $\rightarrow$ 二进制

- 实现: 基数乘法
 - 整数部分: 除基取余法
 - 整数部分除基取余;
 - 最先**取得的余数为数的**最低位**;
 - 最后**取得的余数为数的**最高位**;
 - 即: 除基取余, 先余为低, 后余为高, 商为 0 时结束;

- 小数部分：乘积取整法
 - 小数部分乘基取整；
 - 最先取得的整数为数的最高位；
 - 最后取得的整数为数的最低位；
 - 即：乘基取整，先整为高，后整为低，乘积为 1.0 (或满足精度要求) 时结束；

• 注意：

- 整数可以连续表示，但小数是离散的，所以并不是每个十进制小数都可以准确地用二进制表示；
- 任意一个二进制小数都可以用十进制小数表示，但一个十进制的小数不一定可以被二进制数精确表达；

• 图示：

• 整数部分：

- $123 = (1111011)_2$
- 注意：余数从下往上看，为从大到小；

除基	取余	
2 1 2 3	1	最低位
2 6 1	1	
2 3 0	0	
2 1 5	1	
2 7	1	
2 3	1	
2 1	1	最高位
0		

• 小数部分：

- $0.6875 = (0.1011)_2$
- 数字从上往下看，为从大到小

乘基	取整	
0.6875		
× 2	1	最高位
1.3750		
0.3750		
× 2	0	
0.7500		
× 2	1	
1.5000		
0.5000		
× 2	1	最低位
1.0000		
.....		

定点数的编码表示

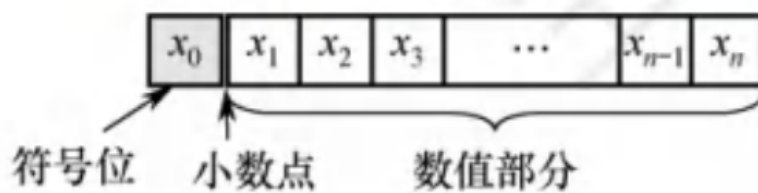
基础概念

概念：真值和机器数

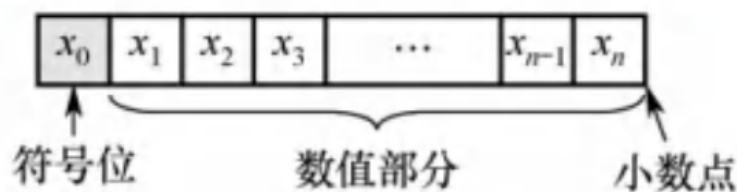
- 在日常生活中，通常用 **正号、负号** 来分别表示正数 (正号可省略) 和负数，如 +15、-8 等；
- 这种带“+”或“-”符号的数称为真值；
- **真值是机器数所代表的实际值；**

概念：机器数的定点表达

- 概念：
 - 根据小数点的位置是否固定，在计算机中分为定点表示和浮点表示；
 - 通常用**补码整数表示整数**，用**原码小数表示浮点数的尾数部分**，用**移码表示浮点数的阶码部分**；
 - 定点表示法用来表示**定点小数和定点整数**；
- 定点表示法：
 - 作用 -> 用来表示 **定点小数** 和 **定点整数**；
 - 定点小数：
 - 定点小数是纯小数，约定小数点位置在符号位之后、**有效数值部分最高位之前**；
 - 图示：



- 定点整数：
 - 定点整数是纯整数，约定小数点位置在**有效数值部分最低位之后**；
 - 图示：



举例：计算实例

- 搞懂下面例子，四种编码的互相转换就搞懂了
 - 计算器：
 - [原码,反码,补码相互转换在线计算器 \(23bei.com\)](http://23bei.com)
- 计算 $x_1 = -10101$ 的移码
 - (1) 真值 -> 原码: 1,0010101
 - (2) 原码 -> 反码: 原码的除符号位外，取反 -> 1,1101010
 - (3) 反码 -> 补码: 反码的数值位加 1 -> 1,1101011
 - (4) 补码 -> 移码: 补码的符号位取反 -> 0,1101011

- 注意：
 - 移码是在补码的基础上，只**对符号位取反**；

原码

定义： #原码

描述：

$$[x]_{\text{原}} = \begin{cases} 0, x, & 0 \leq x < 2^n \\ 2^n - x = 2^n + |x|, & -2^n < x \leq 0 \end{cases} (x \text{是真值, 字长为 } n+1)$$

解释

- 概念：
 - 用机器数的**最高位**表示数的**符号**，其余各位表示数的绝对值；

• 范围：

- 定义：
 - 若字长为 n 位，则原码整数的表示范围：

$$-(2^{n-1} - 1) \leq x \leq 2^{n-1} - 1$$

• 解释：

- **-1**：因为还需要表示 **+0** 和 **-0**，所以需要在正负域上各自 **-1**；
- **n-1**：因为需要用一位来表示符号，所以范围上 **n-1**；

举例

- 若 $x_1 = +1110$ ，字长为 **8**，则： $[x_1]_{\text{原}} = 0,0001110$

特点

- 优点：
 - 1. 与真值的对应关系简单、直观，与真值的转换简单；
 - 2. 用原码实现乘除运算比较简便；
- 缺点：
 - 1. 原码的 **0** 的表示不唯一，有 $[\pm 0]_{\text{原}}$ 两个编码；
 - 2. 原码的**加减运算比较复杂**，在原码加减运算中，对于两个不同符号数的加法（或同符号数的减法），先要比较两个数的绝对值大小，然后用绝对值大的数减去绝对值小的数，最后还要为结果选择合适的符号；

反码

概念：反码

- 概念：
 - 正数的反码是其本身，和原码表示法一致；
 - 负数的反码是在其原码的基础上，**符号位不变，其余各个位取反**；

- 实现：
 - 正数时，和原码一样计算；
 - 负数时，先求原码，然后将其除了符号位外，各个位取反；
- 举例：
 - $[+1] = [0, 0000001]_{\text{原}} = [0, 0000001]_{\text{反}}$
 - $[-1] = [1, 0000001]_{\text{原}} = [1, 1111110]_{\text{反}}$

补码

定义： #补码

描述：

$$[x]_{\text{补}} = \begin{cases} 0, x, & 0 \leq x < 2^n \\ 2^{n+1} + x = 2^{n+1} - |x|, & -2^n \leq x < 0 \end{cases} \pmod{2^{n+1}}$$

解释

- 概念：
 - 补码表示法中的加减运算**统一采用加法操作**实现；
 - 负数的补码等于**模与该负数绝对值之差**；
 - $n+1$ 位补码的模为 2^{n+1}
 - 无论整数还是负数， $[x]_{\text{补}} = 2^{n+1} + x \pmod{2^{n+1}}$ ($-2^n \leq x < 2^n$)
- 范围：
 - 定义：
 - 若字长为 n ，则补码的表示范围：

$$-2^{n-1} \leq x \leq 2^{n-1} - 1$$

- 解释：
 - -1 ：因为 0 的表示唯一，所以只需要在正数域上 -1 ；
 - $n-1$ ：因为需要用一位来表示符号，所以范围上 $n-1$ ；

举例

- $[+0]_{\text{补}} = [-0]_{\text{补}} = 0, 00 \dots 0 \rightarrow$ 说明 0 的补码是唯一的；
- $[-1]_{\text{补}} = 2^{n+1} - 1 = 1, 11 \dots 1$
- $[2^n - 1]_{\text{补}} = 0, 11 \dots 1 (n \text{ 个 } 1) \rightarrow$ 最大的整数，可以知道补码在正数部分的运算结果和原码一致；
- $[-2^n]_{\text{补}} = 1, 00 \dots 0 (n \text{ 个 } 0) \rightarrow$ 最小的整数，可以知道最小的负数其运算结果；

补充：求补码的负数 $[-A]_{\text{补码}}$

- 已知一个数的补码表示为 Y ，则这个数的负数的补码为 Y 取反+1，即逐位取反再加一；

定义： #变形补码

描述:

$$[x]_{\text{变补}} = \begin{cases} 00, x, & 0 \leq x < 2^{n-1} \\ 2^{n+1} + x = 2^{n+1} - |x|, & -2^{n-1} \leq x < 0 \end{cases} \pmod{2^{n+1}}$$

解释

- 变形补码是一种采用双符号位的补码表示, 也称模 4 补码;
- 假定变形补码的位数为 $n+1$ (其中符号位占 2 位, 数值位占 $n-1$ 位);
- 模 4 补码双符号位 00 表示正, 11 表示负, 用在执行算术运算的 ALU 中;

移码

定义: #移码

描述:

$$[x]_{\text{移}} = 2^n + x (-2^n \leq x < 2^n, \text{其中机器字长为 } n+1)$$

解释

- 概念:
 - 移码常用来表示浮点数的阶码, 它只能表示整数;
 - 移码就是在真值 x 上加上一个常数 (偏置值), 通常这个常数取 2^n ;
 - 相当于 x 在数轴傻瓜向正方向偏移了若干单位;
- 实现:
 - 先求出当前数字的补码, 然后在补码的基础上, 将其符号位取反;

特点

- 1. 移码中零的表示唯一;
- 2. 一个真值的移码和补码仅差一个符号位 $\rightarrow$ 先求补码, 即可得到移码;
- 3. 移码全 0 时, 对应真值的最小值 -2^n , 移码全 1 时, 对应真值的最大值 $2^n - 1$
- 4. 移码保持了数据原有的大小顺序, 移码大真值就大, 移码小真值就小;

总结

1. 原码、补码、反码的符号位相同, 正数的机器码相同。
2. 原码、反码的表示在数轴上对称, 二者都存在 +0 和 -0 两个 0。
3. 补码、移码的表示在数轴上不对称, 零的表示唯一, 它们比原码、反码多表示一个数。
4. 整数的补码、移码的符号位相反, 数值位相同。
5. 负数的补码、反码末位相差 1
6. 原码很容易判断大小。而负数的补码、反码很难直接判断大小, 可采用如下规则快速判断: 对于负数, 数值位部分越小, 其绝对值越大, 即负得越多;

常用校验码

概念：常用校验码

- 检验码是能够发现或能够自动纠正错误的数据编码；
- 码距不小于 2 时开始有了检错能力，码距越大，检错能力就越强，且检错能力总是大于纠错能力；
- 常用校验码有奇偶校验码、汉明码、CRC 循环码。
- 奇偶校验码，在原始数据的最高位添加 1 或者 0，使其数据中的 1 恰好为奇数或者偶数；
- 只能发现代码中奇数位出错的情况。常用在对存储器数据的检查或传输数据的检查中；
- CRC 循环码，通常用于数据链路层，适合对大量数据校验。

获取笔记持续更新可加微信：WandD2277