第二章 数制和编码

By pepperRabbit

● 进位数制

位置表示法：

$N = (a_{n-1}a_{n-2}\cdots a_1a_0.a_{-1}a_{-2}\cdots a_{-m})_r$

最高有效数字 Most significant digit (MSD): a_{n-1}

最低有效数字 Least significant digit (LSD): a_{-m}

二进制下，称为 MSB 和 LSB

多项式表示法：

$$N = a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \ldots + a_0 \times r^0 + a_{-1} \times r^{-1} \ldots + a_{-m} \times r^{-m} \quad (1.2)$$

$$= \sum_{i=-m}^{n-1} a_i r^i$$

r^i:weight(权)

二进制、八进制和十六进制：略

二进制到八进制、十六进制间的相互转换：

“分组对应”法：从**小数点**起，每隔 3 位或 4 位转换为 1 位八进制或十六进制的数。

$(1011101000.011)_2 = (\underline{0010}\ \underline{1110}\ \underline{1000}.\underline{0110})_2$

$= (2E8.6)_{16}$

$(3FD.B)_{16} = (\underline{0011}\ \underline{1111}\ \underline{1101}.\underline{1011})_2$

$= (1111111101.1011)_2$

- **整数部分：除以基数取余法**
  - A->B转换的理论基础:
    - $(N_I)_A = b_{n-1}B^{n-1} + \ldots + b_0B^0$
      这里, b_i's represents the digits of $(N_I)_B$ in base A.
    - $N_I / B = (b_{n-1}B^{n-1} + \ldots + b_1B^1 + b_0B^0) / B$
      $=$ (商 Q_1: $b_{n-1}B^{n-2} + \ldots + b_1B^0$) + (余数 R_0: b_0)
    - 一般来说, $(b_i)_A$ 就是Q_i 除以$(B)_A$ 的余数R_i.
  - **转换过程**
    1. $(N_I)_A$ 除以 $(B)_A$, 得到Q_1 和 R_0。
       R_0 是结果的最低位（LSD）d_0。
    2. 计算d_i, $i = 1 \ldots n-1$
       用Q_i 除以 $(B)_A$, 得到 Q_{i+1} 和R_i, R_i就是d_i。
    3. 当$Q_{i+1} = 0$时，停止过程.

2020/9/23

- **小数部分：乘以基数取整法**
  - 理论基础:
    - $(N_F)_A = b_{-1}B^{-1} + b_{-2}B^{-2} + \ldots + b_{-m}B^{-m}$

    这里, $(N_F)_A$ 是*A*进制小数， b_i's 是 $(N_F)_B$ *的A进制数字。*
    - $B \times N_F = B \times (b_{-1}B^{-1} + b_{-2}B^{-2} + \ldots + b_{-m}B^{-m})$

    = (整数 I_{-1}: b_{-1}) + (小数 F_{-2}: $b_{-2}B^{-1} + \ldots + b_{-m}B^{-(m-1)}$)
    - 一般来说, $(b_i)_A$ 是$F_{-(i+1)} \times (B_A)$的整数部分 I_{-i}.
  - 转换过程：

    1. 设 $F_{-1} = (N_F)_A$.
    2. 用 F_{-i} 乘以 $(B)_A$ 计算 $(b_{-i})_A$, 其中$i = 1 \ldots m$,
       得到整数 I_{-i}, 表示 $(b_{-i})_A$, 以及小数$F_{-(i+1)}$.
    3. 把 $(b_{-i})_A$ 改写成*B进制数.*
    4. 直到 fraction=0或到达最大有效数字

- **浮点数**

- IEEE754浮点标准形式V=$(-1)^s$ ***M*** 2^E
  - **符号位S**(Sign)：决定这个数是正数(s=0)还是负数(s=1) 。
  - **尾数M**(significand)：是一个二进制小数，范围在[1.0，2.0）
  - **阶码E**(exponent)：对浮点数加权重，权重是2的E次幂。
- 编码表示，分3个字段：
  - 1位符号字段s
  - k位阶码字段exp：阶码E的编码
  - n位小数字段frac： 尾数M的编码

s	exp	frac

$$N = (S_M b_{e-1} b_{e-2} \ldots b_0 a_{n-1} \ldots a_{-m})_r$$

- 阶码编码exp，无符号数表示，既不全0又不全1。
- 阶码编码exp表示阶码E的偏置(biased)形式: ***exp*** =E+ ***Bias***
  - exp：无符号数
  - Bias=2^{k-1}-1，k为阶码的位宽
    - 单精度：127 (Exp: 1…254, E: -126…127)
    - 双精度: 1023 (Exp: 1…2046, E: -1022…1023)
- 小数字段frac表示尾数M的编码M= $1.xxx\ldots x_2$
  - xxx…x: 小数字段数值
  - 最小值：000…0 (*M* = 1.0)
  - 最大值：111…1 (*M* = 2.0 – ε)

- 单精度：32位

s	exp	frac
1	8-bits	23-bits

- 双精度：64位

s	exp	frac
1	11-bits	52-bits

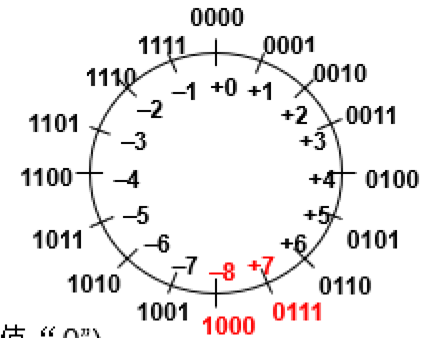
$N = (Sb_{k-1}b_{k-2} \ldots b_0a_{n-1} \ldots a_{-m})_2$

$$N = (-1)^S \times (1.a_{n-1} \ldots a_{-m})_2 \times 2^{(b_{k-1}b_{k-2} \ldots b_0)-2^{k-1}+1}$$

带符号数的表示

补码：

- 基数补码表示法（同余数、模）：
  1. 基数为R的n位数的补码等于从R^{n}中减去该数。
     $D_补=R^n-D$
  2. **按位取反加一。**
     - R^n-D=((R^n-1)-D)+1; **(R^n-1)-D表示反码**，R-1-d/位
- 一个数的补码的补码保持不变。



- 二进制补码的性质：
  - 0是唯一表示的(用全0表示数值“0”)
  - 不对称，负整数比正整数多1个（-2^{n-1}）
  - n个二进位的补码可表示的数值范围是：$-2^{n-1} \sim 2^{n-1}-1$

二进制数的运算

- 现代计算机中，带符号整数都使用补码表示。
- **CPU直接对补码进行运算和处理！**
- 采用补码运算具有如下优势：
  - 符号位和数值位统一处理，使符号位能与数值一起参加运算，从而简化运算规则，简化运算器的结构，提高运算速度；
  - 减法可以按加法来处理。加法运算比减法运算更易于实现。
  - 保证了 0 编码的唯一性。

- 补码加法规则
  - 二进制补码数可以按照普通的二进制加法相加，只要**不超过**补码定义的范围，该结果就是正确的。

- 补码减法处理方法：
  - (1)将减数变负取补码;
  - (2)将减数和被减数按正常的加法规则相加即可。

溢出检测：

符号位相同的数相加可能溢出，符号位不同的数相加不会溢出。

溢出检测方法：

1、和的符号位与加数的符号位不同，则产生溢出。

2、如果向符号位的进位输入C_{in}和从符号位的进位输出C_{out}不同，则产生溢出。

二进制乘法：

- 采用移位—累加方法实现无符号数的乘法。
- 有符号数：
  - 同号相乘为正，异号为负；
  - 取操作数的绝对值，按无符号数乘法计算积。

二进制除法：

- 基本方法：移位—减法。

- 除数为零会产生除法溢出。

- 有符号数的处理方法和乘法相同。

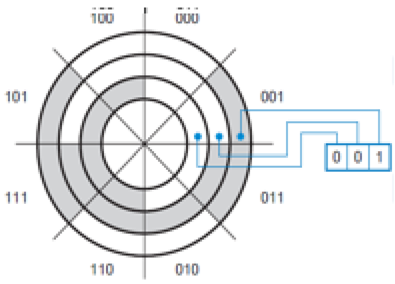
- **编码**

BCD 码：用二进制数位表示十进制数

- BCD码运算：类似于4位无符号数加法
  - 加法有进位，**加6修正**
  - 减法有借位，**减6修正**
  - 结果超过1001，需要校正，**加6修正**

格雷码：

通过设计数字编码使得每对连续的码字之间**只有一个数位不同**，这样的编码叫做格雷码（**循环码**）。



- 利用Gray码的反射特性
  - 若以高位0和1的交界为轴，低位的代码是轴对称的；
  - 高位被称为“反射位”；
  - 利用反射特性可以较容易地构成任意位循环格雷码。

0	00	000	0000	1100
1	01	001	0001	1101
		011	0011	1111
	11	010	0010	1110
	10		0110	1010
		110	0111	1011
		111	0101	1001
		101	0100	1000
		100		

- 利用二进制转换到Gray码

  其公式是：$G_n=B_n$

  $G_i=B_{i+1}\oplus B_i$

  $\oplus$称为异或运算(模2加法，即不考虑进位的二进制加法)，运算规则为：$0\oplus 0=0$；$0\oplus 1=1$；$1\oplus 0=1$；$1\oplus 1=0$。

  例：二进制数为　1　0　1　1　0　1　0　0

  Gray码　1　1　1　0　1　1　1　0

汉明码：

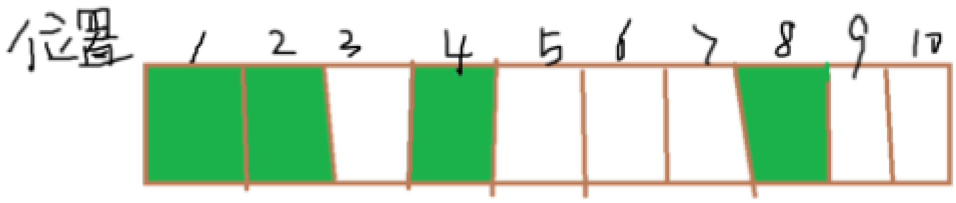
- 汉明距离（Hamming distance）
  - 两个位串逐位比较，不同位的数目叫做这两个位串的距离。
- 编码方法：
  - 从右向左给每一信息位编号：1——2^i-1；
  - 其中2的幂次位置安排校验位，其余信息位；
  - 每个校验位与部分信息位联合成一组，这些信息位的编号中在校验位为1的位置上都是1。
    - 校验位2（010），与信息位011，110，111组成一组。
- 纠错处理：
  - 检验所有奇偶检验组：
    - 如果都是偶校验，则码字是正确的；
    - 如果有一组或多组是奇校验，可认为出现了单错。具有奇校验的组必然和奇偶校验矩阵中的某一列相匹配，则对应的位置号包含错误值，取反即可。
- 例：
  - 接受编码：1110101
  - 则： C组：1110***→1(奇数个1)
    B组：11**10*→1 (奇数个1)
    A组：1*1*1*1→0 (偶数个1)

出错位置号110，第6位错，修改1为0，则正确的编码是：1010101

距离为3的汉明码扩展一个全局偶校验位后，可把距离扩展为4，扩展检错能力（纠1位错、检1位错）。

汉明码老师讲的还是不是很清楚…参考这篇博客
https://blog.csdn.net/Yonggie/article/details/83186280?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.channel_param&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.channel_param
再理解一下…

首先，校验码的位置，**2 的 i 次方的位置上放校验码**，即 1,2,4,8，…



第二个问题，汉明码怎么分组!!!

凡是位次是 xxx1 的放在第一组，
凡是位次是 xx1x 的放在第二组，
凡是位次是 x1xx 的放在第三组，
凡是位次是 1xxx 的放在第四组…

例如，XX1X101X011，共 11 位
1,3,5,7,9,11 放在第一组（xxx1）
2,3,6,7,10,11 放在第二组（xx1x）
4,5,6,7 放在第三组（x1xx）
8,9,10,11 放在第四组（1xxx）

然后，保证偶校验，根据每组 1 的个数决定校验码是 1 还是 0，就是**对每组的各个位进行异或操作**再填进去就可以了。

那么，接受者收到以后怎么检错呢…
首先，分组，例如，P1,P2,P3,P4,P5
然后，对每个组进行校验，错的写 1，对的写 0

把P从大到小排列起来，得到一串1010，

for example:

组别:	P5	P4	P3	P2	P1
标志:	1	0	1	0	1

从大到小排列起来，标志排成了一串一零串。这个数就是出错的数据的位置。

本例中，10101位置上的位错了，换成十进制是第21个位置上的数错了。

然后，我们发现了它错误的位置，又因为它是二进制的，不是0就是1，所以，可以顺便把它纠错。