第 2 章 定位数系与数据存储

本章概述：本章首先介绍了人类对数的认识过程中，由简单分群数系到乘法分群数系再到定位数系的演变，总结了当今通用的十进制系统的基本记数原理；然后介绍了参照十进制系统创建的二进制、八进制以及十六进制的基本记数原理以及这几种进位制系统与十进制系统之间相互转换的方法；接下来本章还介绍了多媒体数据在计算机中的存储方式，详细阐述了数值型数据（整数类型与实数类型）在计算机中存储和运算，其它类型的多媒体数据的存储仅仅作了简要概述。

数据（data）可以简单地理解为对人类有用的信息，比如科学计算或实验中的各种结果以及新闻报道中发布的文字、图片以及视音频这些都是数据的实例，计算机处理的数据从形式上可归纳为数字、文本、图像、音频、视频这五种类型，计算机科学领域将这五类数据统称为**多媒体数据**（multimedia data）。

数据既然是对人类有用的信息，因此是需要保存的。即便是在发明文字之前，人类就通过在绳子上打结的方式计数（结绳记事），在树干上刻线的方式记录时间。文字出现之后，数据的保存开始变得系统化，人类使用得最多的数据就是各种“数量”，比如采摘的果实有多少个？一根树枝有多长？一块土地有多大？人类发明了很多符号表示“数量”，比如阿拉伯数字 0、1、…、9，比如古罗马数字 I、II、III、TV、V、VI 等，还有中国古代人使用的汉字一、二、…、十、百、千、万等。随着进化程度的深入，人类不仅可以认识理解“数量”，还能使用“数量”进行一些简单的运算，进一步发展则形成了完整的**数系**（numeral system），数系可以简单地理解为使用哪些符号遵循什么规则表示自然界存在的“数量”。

2.1 定位数系

纵观人类发展历史，不同的文明创造过各类不同的数系，比较典型的有**简单分群数系**（simple grouping system）、**乘法分群数系**（multiplicative grouping system）、**定位数系**（positional numeral system），其中由印度人发明经过阿拉伯人推广的**十进制定位数系**就是今天全球统一使用的十进制自然数系统。

2.1.1 十进制定位数系

印度人发明的十进制定位数系很有可能是建立在对更古老数系的改进基础之上，所以接下来介绍一下简单分群数系到乘法分群数系再到定位数系的演变过程。

2.1.1.1 简单分群数系

简单分群数系的记载出现在人类早期文明的遗迹中，古埃及人、古巴比伦人、古希腊雅典人以及古罗马人就是使用的简单分群数系，古埃及人与古巴比伦人使用的都是象形文字，古埃及人使用形似一根竖线的符号|表示单个数量，这和其他一些文明中表示单个数量的数字符号不谋而合，比如前面提到的古希腊雅典人和古罗马人使用 I 表示单个数量，中国古代大约是春秋战国时期开始作为计算工具使用的算筹，就是用一根小木棍表示单个数量。各个不同的文明对于各自创建的简单分群数系有着不同的数字符号集合以及表示数的规则，不过大致思路还是类似的，本章介绍的一种简单分群数系，在

记数规则上参照古埃及人创建的简单分群数系，只不过使用不同的符号集合，图 2-1 中显示的第一排符号就是古埃及人使用的符号，第二排就是本文选取的替代符号，替代符号中保留古埃及符号中的|，其余的符号则使用大写英文字母替代，每个古埃及符号下面对应的英文字母即为其替代者。

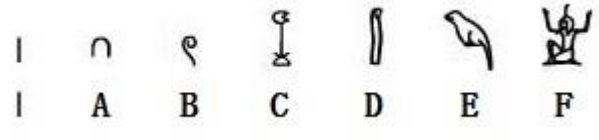


图 2-1 古埃及人创建的简单分群数系中使用的符号

人类进化过程的早期，觅食是人生主要活动，所关心的“数量”多是与食物有关，比如捕获了多少只野兽，采摘了多少颗野果。这类与物体个数有关的“数量”后来发展成为自然数，所以接下来将与物体个数相关联的“数量”称为“自然数量”（具有离散变化特性），以区别于自然界中存在的诸如树枝长度以及土地面积等连续变化的“数量”。

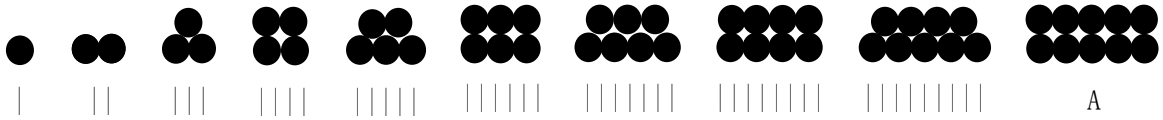


图 2-2 简单分群数系中“自然数量”的表示

图 2-2 给出了直观感受到的“自然数量”与符号的对应关系，图中黑色圆圈表示抽象的物体，单个圆圈对应的符号是|，那么有多少个圆圈就使用多少个|来表示直觉上的“自然数量”，这是一种最朴素的使用符号表示“自然数量”的思路，值得一提地是，人类最早认识自然数时并没有零的概念。随着“自然数量”的逐渐增加（代表物体的黑色圆圈越来越多），使用的|也会越来越多，这显然不是一种理想的记数方式，于是当|的个数累计到十个时，就使用字母 A 取代十个|，也就是说十个|表示的“自然数量”相当于一个 A 表示的“自然数量”；这样的替换大大减少了表示“自然数量”的符号长度，接下来将使用 A 与若干个|组成的字符串表示继续增加的“自然数量”。表 2-1 给出了十进制自然数(1-20)在简单分群数系中的表示形式。

表 2-1 十进制自然数在简单分群数系中的表示

十进制自然数	1	2	3	4	5	6	7	8	9	10
简单分群数系	|	||	|||	||||	|||||	||||||	|||||||	||||||||	|||||||||	A
十进制自然数	11	12	13	14	15	16	17	18	19	20
简单分群数系	A|	A||	A|||	A||||	A|||||	A||||||	A|||||||	A||||||||	A|||||||||	AA

表 2-1 所示的简单分群数系中，“自然数量”每增加一个就在其对应的符号串中加一个|，累积到十个|就用 A 替代，累积到十个 A 就用 B 替代，其余以此类推，可以推测 20 之后的十进制自然数在简单分群数系中的表示形式：

21、…、29、30 在简单分群数系中表示为：AA|、…、AA|||||||||、AAA；

31、…、39、40 在简单分群数系中表示为：AAA|、…、AAA|||||||||、AAAA；

……

91、…、99、100 在简单分群数系中表示为：AAAAAAAAA|、…、AAAAAAAAA|||||||||、B；

和今天使用的十进制对比，|就相当于 1，A 就相当于十，B 就相当于百，并以此类推；使用这些符号排列在一起表示“自然数量”时，相同符号摆放在一起，不同的符号遵循从右往左表示的“自然数量”依次递增的顺序，比如自然数 234 在简单分群数系中表示为 BBAAA||||。

在使用符号表示“自然数量”的体系出现之后，表示“自然数量”的符号串被称为**自然数**（natural number），针对自然数的各类运算也被确立，其中加法（+）是最基本的运算，加法在简单分群数系中的意义通过图示（图 2-3）最直观，**简单分群数系中两个自然数表示的“自然数量”合在一起之后形成的“自然数量”对应的自然数即为这两数之和**。简单分群数系中任意两自然数之间做加法比较简单直接，只需要将表示两个数的符号摆在一起（有必要地话再进行符号变换并重新排序）即可，图 2-3(a)中“自然数量”（三个圈）与“自然数量”（四个圈）合在一起后的“自然数量”不到十个，所以对应的两个数（|||和||||）相加之后仅需要将所有的|摆在一起即可，相加后不涉及符号变换；图 2-3(b)中两个“自然数量”合在一起后的“自然数量”超过十个，所以需要将其中十个|用 A 代替，相加后涉及符号变换。

(a)没有涉及符号变换的加法　　(b)涉及符号变换的加法

图 2-3 简单分群数系中加法运算的意义

再看较大自然数之间相加的例子，比如十进制自然数 1234+234 的计算过程如下：

CBBAAA||||+BBAAA||||=CBBAAA||||BBAAA||||=CBBBBAAAAAA||||||||（相加后符号需要重新排序）

有时相加后除了符号重新排序外还要做一些符号替换，比如 1234+5678 的计算过程如下：

CBBAAA||||+CCCCCBBBBBBAAAAAAA||||||||=CBBAAA||||CCCCCBBBBBBAAAAAAA||||||||

=CCCCCCBBBBBBBBAAAAAAAAAA||||||||||=CCCCCCBBBBBBBBAAAAAAAAAA**A**||=CCCCCCBBBBBBBB**B**A||

上述两数相加后，|的数量超过了十个，所以其中十个|被一个 A 替换（新增的 A 使用加粗表示），然后 A 的数量也超过十个，所以其中十个 A 被一个 B 替换（新增的 B 使用加粗表示），经过这两步替换之后得到的就是两数之和的最终表示形式，符号替换实际上就是后来自然数相加时数字进位的雏形。

可以发现简单分群数系表示数的规则就是通过表示不同“自然数量”的符号的罗列，所有符号表示的“自然数量”合在一起就是该数表示的“自然数量”。以自然数 234 为例，其在简单分群数系中的表示规则暗含了如下等量关系：

BBAAA||||= B+B+A+A+A+|+|+|+|

推广到一般情形则有，简单分群数系中的自然数与组成该数的各个符号之间满足如下等量关系：

$a_n a_{n-1}\cdots a_1=a_n+a_{n-1}+\cdots+a_1$

回过头来思考一个有趣的问题，埃及人使用的简单分群数系里面为什么是十个|相当于一个 A，而不是更多或更少的|相当于一个 A？这很有可能与早期人类使用十个手指计数的习惯有关，古埃及人、古巴比伦人、古希腊雅典人以及古罗马人使用的简单分群数系，都是“十个换一个”，古罗马数字中 I

表示自然数 1，十个 I 相当于一个 X，十个 X 相当于一个 C，十个 C 相当于一个 M，不过古罗马数字中还包括 V（相当于五个 I）、L（相当于五个 X）、D（相当于五个 C）。

从简单分群数系开始使用符号表示自然数时，累计到 b 个符号时使用表示更大数的符号替换，可以称该简单分群数系以 b 为**底**（**基**），这就是后来进位制的雏形，本节以及后面介绍的数系如果没有特别强调以多少为底，都默认表示为以十为底。

简单分群数系虽然在记数方式以及加法运算上简单直接，但是在自然数的表示上依然比较繁琐，每种符号最多可以罗列9个,比如自然数999在本章所介绍的简单分群数系中表示时需要27个符号(B、A、|各 9 个)，书写起来是相当麻烦的，所以简单分群数系在随后的发展过程中慢慢做了改进，形成了乘法分群数系。

2.1.1.2 乘法分群数系

可以对简单分群数系在数的表示上做些改进，对十以内的九个自然数（没有零）分别使用不同的单个符号表示，而不是只使用一个符号|；对于上一节中介绍的简单分群数系，使用阿拉伯数字 1-9 表示十以内的自然数（如图 2-4 所示）。

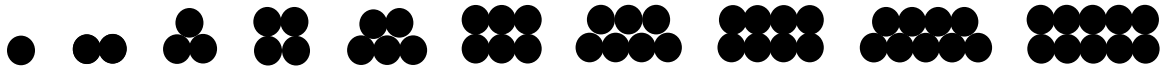


图 2-4 使用乘法分群数系表示“自然数量”

A、B、C 等表示更大自然数的符号依然保留，对应不同自然数的符号只出现一次，而不是像简单分群数系里面那样会出现多个相同符号罗列的情况，比如简单分群数系表示的数 BBAAA||||使用乘法分群数系表示为 2B3A4（2 个 B、3 个 A、4 个|的简约表示形式），每个大写英文字母前面的数字表示该字母的个数，这样的改进极大地减少了表示一个数需要的平均符号长度，这种改进后的数系被称为乘法分群数系。

十进制自然数（1-20）的简单分群数系表示法与乘法分群数系表示法对照表见表 2-2 所示，可以推测 20 之后的十进制自然数在乘法分群数系中的表示形式：

表 2-2 十进制自然数在简单分群数系与乘法分群数系中的表示

十进制自然数	1	2	3	4	5	6	7	8	9	10
简单分群数系	|	||	|||	||||	|||||	||||||	|||||||	||||||||	|||||||||	A
乘法分群数系	1	2	3	4	5	6	7	8	9	A
十进制自然数	11	12	13	14	15	16	17	18	19	20
简单分群数系	A|	A||	A|||	A||||	A|||||	A||||||	A|||||||	A||||||||	A|||||||||	2A
乘法分群数系	A1	A2	A3	A4	A5	A6	A7	A8	A9	2A

21、22、…、29、30 在乘法分群数系中表示为：2A1、2A2、…、2A9、3A；

31、32、…、39、40 在简单分群数系中表示为：3A1、3A2、…、3A9、4A；

……

91、92、…、99、100 在简单分群数系中表示为：9A1、9A2、…、9A9、B；

在乘法分群数系中，可以将 1、A、B、C 等符号看做是从小到大的**数量单位**，乘法分群数系中的数就由数量单位和数字（1-9）交错排列的方式组成，1 是最特殊的符号，既是数字也是最小的数量单位，通常单个数量单位前面的数字 1 会省略，比如说某数由 1 个 B、1 个 A、4 个 1 组成，在乘法分群数系中表示为 BA4，而不是 1B1A4，乘法分群数系中的某数中出现数字 1 必定是在最右边的数位上。

以自然数 234 为例，其在乘法分群数系中的表示形式与组成该数的数量单位以及数字之间存在如下等量关系（中间转换形式为简单分群数系的表现形式）：

2B3A4=BBAAA||||=B+B+A+A+A+|+|+|+|=2 个 B+3 个 A+4 个|=2B+3A+4

可以在加法运算的基础上定义乘法运算，则 2 个 B 可以表示为 B×2，再根据乘法满足交换律，进一步表示为 2×B，所以上述等量关系还可以表示为：

2B3A4=2×B+3×A+4×1

推广到一般情形则有，乘法分群数系中的自然数与组成该数的各个符号之间满足如下等量关系：

$a_nX_na_{n-1}X_{n-1}\cdots a_1X_1a_0=a_n\times X_n+a_{n-1}\times X_{n-1}+\cdots+a_1\times X_1+a_0$

其中 X_i（i=1,2,…,n）是数量单位，a_i（i=0,1,…,n）是数字。

乘法分群数系中的数等于其包含的数量单位及其前面数字乘积之和，这也是乘法分群数系命名的由来。

乘法分群数系使得数的平均表示长度大大缩短，但是在实施加法运算时就没有简单分群数系来的直接了，这涉及到相同符号前面的数字“合并”（十以内的数字相加）的问题，这个在今天不算个事，但是在处于当时的人类来说则有不小的难度，这其中关键的受制因素并非早期人类的智力，而是物质上的困难。

早期人类普遍没有手动计算能力（或者说是意识），最大的原因受制于当时落后的书写条件。虽然古埃及人就已经发明了**莎草纸**[1]（papyrus），但是由于造纸的原材料有限以及在潮湿环境下不易保存，所以并不如今天的纸张这样普及，且书写质量更是远不如今天使用的纸张[2]；古希腊人因为受到古埃及人对莎草纸出口的限制（看来贸易战自古就有），发明了羊皮纸以及牛皮纸，这两种纸张书写质量很高，但是因为原材料来自动物皮，所以成品比较稀少且价格也很昂贵，通常是贵族使用用来记录重要的文件，没有在民间普及；所以早期人类进行计算都是通过工具，比如中国春秋战国时期开始使用的算筹、古希腊人使用的沙盘、古罗马人使用的沟算盘和中国唐朝开始使用的算盘，使用计算工具使得早期人类并没有过多地琢磨如何手动计算，今天的人类手动计算的能力秒杀祖先其主要原因在于教育体系的完善，学生刚进入小学阶段就已经将加法和乘法口诀背得滚瓜烂熟，并在此基础上演练多位数的计算，在手动计算方面训练有素，随着年龄增长之后这种能力成为本能。

---

[1] 古埃及人记录文字的载体，使用盛产于尼罗河三角洲的纸莎草的茎制成，曾出口到古希腊以及地中海各古文明地区。

[2] 造纸术经过我国东汉时期蔡伦的改进后才在全世界范围推广普及。

根据《孙子算经》[1]中记载，中国春秋战国时期就开始使用算筹作为计算工具，结合书中记载和考古材料的发现，古代的算筹实际上是一根根同样长短和粗细的小棍子，一般长为 13-14 厘米，直径粗 0.2-0.3 厘米，多用竹子制成，也有用木头、兽骨、象牙等材料制成。使用算筹表示的数字 1 至 9 的形式如图 2-5 所示，每个数字有横式和竖式两种形式，算筹使用符号‘|’按照纵横两种方式的组合形成十以内的数字，这种思路接近简单分群数系；但是书中还记载了算筹表示的多位数从左往右依次为纵式与横式的顺序交替，比如说数 24 使用算筹表示为“=||||”，左起第一位的数字 4 使用竖式‘||||’表示，左起第二位的数字 2 使用横式‘=’表示，以此类推。《孙子算经》中对于算筹计算法则的记录为：凡算之法，先识其位，一纵十横，百立千僵，千十相望，万百相当。算筹计数采用的是十进制，其表示数的纵横排列的方式和定位数系很相似，不过缺乏更多的证据认定其为十进制定位数系。算筹退出历史舞台的具体时间不详，最迟在唐朝使用算盘作为计算工具之后。

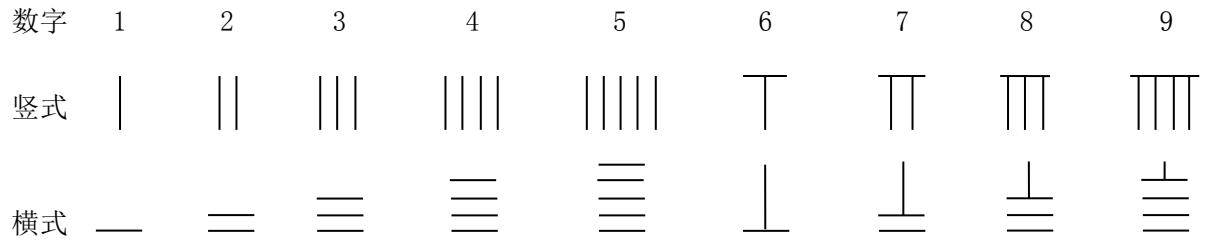


图 2-5 数字 1 到 9 在算筹中的表示

中国古代在数表示形式上，书面文字表示的数与算筹或算盘等计算工具表示的数是不一样的，书面文字中表示数使用的是汉字：零、一、二、三、四、五、六、七、八、九、十、百、千、万、亿、兆等，比如自然数 234 在中国古代被表示为二百三十四，这属于典型的乘法分群数系表示法。

2.1.1.3 定位数系

定位数系可以看做是在乘法分群数系的基础上再次改进，就以本章介绍的乘法分群数系为例，该数系表示的数中数字与其后面紧邻的数量单位（大写英文字母）联系紧密，比如 3B2A5，数字 3 表示 B 的个数，数字 2 表示 A 的个数，而且从小到大的数量单位 1、A、B、C 等都是按照从右到左的顺序排列，何不干脆将数量单位去掉，数量单位前面表示其个数的数字采取固定位置摆放，具体来说右边第一位对应十以内数字（也就是数量单位 1 的个数），右边第二位固定为数量单位 A 的个数，右边第三位固定为数量单位 B 的个数，右边第四位固定为数量单位 C 的个数，以此类推。因为这种表示法中包含的数量单位的个数与数字位置固定，所以被称为**定位数系**（有的书籍里也称为**位置制数系**或**位值制数系**）。比如乘法分群数系中的数 2B3A4，转换为定位数系可表示为 234，表示 2 个 B、3 个 A、4 个 1 之和。

不过这样改进后的记数方法和今天的十进制还有一点不一样，就是没有数字 0。实际上当使用定位数系表示数的时候，0 的出现就是水到渠成的事。比如乘法分群数系表示的数 3B5，看上去没有什么问题，转换成简单分群数系表示 BBB|||||看上也没有什么问题，但是转换成定位数系就有问题啦，左起第一位是 5，左起第二位没有，左起第三位是 3，那就应该是 3 5（左起第二位是空格），而不是 35（中间没有空格，这个对应的乘法分群数系表示的数是 3A5），3 5（左起第二位是空格）和 35（中间没有

[1] 这部著作大约成书于公元四五世纪，作者信息不详。

空格）看上去有区别吗？眼神不好的估计看不出区别，如果是手写稿更容易产生混淆，最开始处理这个状况是使用‘.’代替某位置上的空缺数字，也就是说乘法分群数系表示的数 3B5 转换成定位数系就是 3.5，这和后来十进制小数的表示发生冲突，所以再往后来干脆引入一个新的符号 0，某位置上没有数字的就用 0 填充，也就是说乘法分群数系表示的数 3B5 转换成定位数系就是 305，中间的 0 表示该位置上没有数字，0 也就逐渐变成表示“无”的数字；不过对于定位数系表示的数而言，最左边数位开始的若干个 0 是没有意义的，所以约定定位数系表示的数中，0 只能出现在数字中间或末尾，这就是今天我们所熟悉的十进制自然数系统，这也是印度人最先发明然后由阿拉伯人推广的十进制自然数系统。

下面分析下定位数系中的数与其各数位上的数字之间存在什么样的关系。尽管定位数系中去掉了乘法分群数系中的数量单位，但是这些数量单位并没有凭空消失，而是与数位具有关联性，根据乘法分群数系转换为定位数系的规则可以知道，乘法分群数系中的数 A 对应的定位数系表示法为 10，这是因为该数除了一个 A 没有其它任何数量单位，而数量单位 A 对应的数量放在右起第二位，所以右起第二位上数字为 1，右起第一位因为没有数字所以补 0；同样道理，乘法分群数系中的数 B 对应的定位数系表示法为 100；乘法分群数系中的数 C 对应的定位数系表示法为 1000；其余以此类推。

在清楚了乘法分群数系中每个数量单位在定位数系中的表示形式后，就可以开始着手分析定位数系中的数与其各数位上的数字之间存在的关系了。以自然数 234 为例，其与各位数字之间存在如下等量关系（中间转换形式为乘法分群数系的表现形式）：

234=2B3A4=2B+3A+4=2×B+3×A+4×1=2×100+3×10+4×1

可以将乘法分群数系中数量单位的概念扩展到定位数系中来，将 1、A、B、C 等在定位数系中的对应数 1、10、100、1000 等作为定位数系中的数量单位，每个数量单位都和数位联系起来，右边第 1 位对应的数量单位为 1，右边第 2 位对应的数量单位为 10，右边第 3 位对应的数量单位为 100，右边第 n 位对应的数量单位为 100…0(n-1 个 0);可以在乘法运算的基础上定义乘方运算,$100=10\times10=10^2$,$10=10^1$,$1=10^0$,则定位数系中的有 n 个数位组成的数从右往左每个数位对应的数量单位分别为 10^0、10^1、10^2、…、10^{n-1}。

定位数系中的数可以看做是每个数位上的数字与对应数量单位乘积之和，比如

$234=2\times100+3\times10+4\times1=2\times10^2+3\times10^1+4\times10^0$

推广到一般情况，定位数系中的数与组成该数的位数字之间满足如下等量关系：

$a_na_{n-1}\cdots a_1a_0=a_n\times10^n+a_{n-1}\times10^{n-1}+\cdots+a_1\times10^1+a_0\times10^0$

2.1.1.4 十进制小数

自然界除了存在表示物体个数的“自然数量”，还存在其它诸如树枝长度以及土地面积等连续变化的“连续数量”。对于线段长度的度量（也就是如何使用数来表示或标记线段长度），通常选取一段固定的长度作为 1 个“单位长度”（比如 1 毫米、1 厘米、1 分米、1 米、1 公里等），使用数来标记线段长度时，看该线段长度是“单位长度”的“几”倍，就用“几”表示，如果某线段长度恰好是“单位长度”的整倍数，则标记出来的是自然数。不过很多线段长度并不恰好是“单位长度”的整倍数，总会多出一部分，多出的这部分比一个“单位长度”要短，遇到这种情况怎么办？

前人对这个问题的解决方案是将“单位长度”平均分成几等分，将其中一份作为新的尺度标准，

看看多出的部分是“缩小单位长度”的几倍，如果还有多出的不足一个“缩小单位长度”的部分，则重复前面的过程。为了将“缩小单位长度”标记出来的数与“单位长度”标记出来的数（自然数）做区分，在后者的最右边加个点，使用“缩小单位长度”标记出来的数放在这个点的后面，使用这种方法表示的数被称为**小数**（decimal），数字中间出现的点被称为**小数点**（decimal point），小数点左边的部分被称为整数部分，小数点左边的部分被称为小数部分。

十进制系统中是将“单位长度”逐渐十等分作为更小的尺度标准来度量小数。可以使用数轴将线段长度与数字统一起来，数轴上每个点的坐标用该点到坐标原点的距离并结合方向（往右是正方向）表示。如图 2-6 所示，0 点为坐标原点，图中较长的刻度线标记“单位长度”，下面对应的数字是按照“单位长度”计算的坐标（称之为大坐标）；将大坐标 2 与 3 之间的“单位长度”十等分得到“1/10 单位长度”，并增加 9 条较短的刻度线标记“1/10 单位长度”，较短刻度线上的数字 1-9 表示的是按照“1/10 单位长度”计算的坐标（称之为小坐标），之所以没有小坐标 0 是因为其与大坐标 2 重合，也就是说大坐标 2 标记的点同时也是小坐标 0 标记的点，接下来看看数轴上 A、B、C 三点的坐标怎么表示：

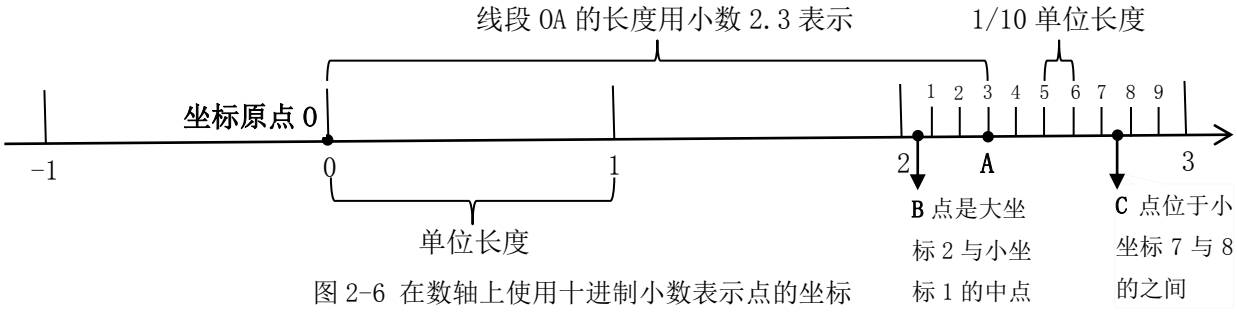


图 2-6 在数轴上使用十进制小数表示点的坐标

①A 点在大坐标 2 与 3 之间，恰好位于小坐标 3 的位置，所以表示线段 OA 的长度时，使用了“整数部分+小数点+小数部分”的表示方式，线段 OA 的长度为 2 个“单位长度”再加上 3 个“1/10 单位长度”，因此 A 点坐标可表示为 2.3，整数部分的 2 表示“2 个单位长度”，小数点后面的 3 表示“3 个 1/10 单位长度”，小数点后面第一位被称为**十分位**（tenths digit）；

②B 点位于大坐标 2 与 3 之间，但是其并没有落在任何一个小坐标上，位于大坐标 2（也就是小坐标 0）与小坐标 1 的中点，那么线段 OB 的长度该如何表示呢？遇到这种情况的处理方式是将“1/10 单位长度”继续缩小十倍得到“1/100 单位长度”，并增加 9 条更短的刻度线标记“1/100 单位长度”，更短刻度线上的数字 1-9 表示的是按照“1/100 单位长度”计算的坐标（称之为小小坐标），由于“1/100 单位长度”、更短刻度线、小小坐标这些太细小，所以图中并没显示出来。因为 B 点位于大坐标 2（也就是小坐标 0）与小坐标 1 的中点，所以 B 点恰好落在小小坐标 5 上，那么线段 OB 的长度为 2 个“单位长度”再加上 0 个“1/10 单位长度”（多出的部分不足 1 个“1/10 单位长度”）再加上 5 个“1/100 单位长度”，因此 B 点坐标可表示为 2.05（注意小数点后面第一位上的 0 不能省略）；整数部分的 2 表示“2 个单位长度”，小数点后面第一位上的 0 表示“0 个 1/10 单位长度”，小数点后面第二位上的 5 表示“5 个 1/100 单位长度”，新增的数位（小数点后第二位）被称为**百分位**（percentile）；

③C 点位于大坐标 2 与 3 之间，所以其整数部分为 2；进一步根据其位于小坐标 7 与 8 之间，所以其小数点后第一位是 7；接下来将小坐标 7 与 8 之间的“1/10 单位长度”进一步缩小十倍并标注小小坐标后，如果 C 点也没有位于任何一个小小坐标之上，假定其位于小小坐标 b 与 b+1 之间，则 C 点小数点后第二位是 b；然后继续将小小坐标 b 与 b+1 之间的“1/100 单位长度”再缩小十倍，重复前面的过程，随后增加的数位依次被称为千分位、万分位、十万分位等。

可以将数量单位的概念扩充到小数中，只不过自然数中的数量单位是针对“自然数量”而论，小数表示的是“连续数量”，而且在标记数时经过多次缩小十等分的操作，因此小数里面的“数量单位”也可以被称为**小数单位**，小数点后第 1 位对应的小数单位 0.1，使用分数表示为 10^{-1}；小数点后第 2 位对应的小数单位 0.01，使用分数表示为 10^{-2}；…；小数点后第 m 位对应的小数单位为 0.00…1（小数点后有 m-1 个 0），使用分数表示为 10^{-m}。

十进制小数与各个数位上的数字之间存在如下等量关系：

$$\pm a_n a_{n-1}\cdots a_1 a_0.\, a_{-1}a_{-2}\cdots a_{-m}=\pm(a_n\times 10^{n}+a_{n-1}\times 10^{n-1}+\cdots+a_1\times 10^{1}+a_0\times 10^{0}+a_{-1}\times 10^{-1}+a_{-2}\times 10^{-2}+\cdots+a_{-m}\times 10^{-m})$$

下面请读者思考一个问题：数轴上所有的点是否都可以通过上面所述的方式得到精确地表示？

十进制小数是对十进制定位数系（自然数）的扩充，也就是将分数使用类似于定位数系的方式表示，也就是说小数其实是分数的“定位数系化”的表现形式，但是分数概念的形成则比十进制定位数系的出现要早，古埃及人在使用简单分群数系的时候就已经使用分数了，不过古埃及人使用的分数都是分子为 1 的分数，也就是今天所说的分数单位。

古希腊人研究几何时计算线段长度时，就采用“单位长度”线段的若干倍或若干等分的方式，如果两条线段都是第三条线段的整数倍，则称这两条线段长度为**可通约量**（commensurable quantities），古希腊数学家**毕达哥拉斯**（Pythagoras）就认为，只要第三条线段取得足够小，任何两条线段都可以作为第三条线段的整数倍，使用“单位长度”度量任意线段 L 得长度时，如果 L 不是“单位长度”的整数倍，则将“单位长度”若干等分，只要分的足够细小，就可以使得线段 L 是“单位长度”若干分之一的整数倍，“单位长度”用 1 表示，这样任意线段长度都可以表示称为 q/p 的分数形式（其中 p 与 q 是互质[1]的正整数）。毕达哥拉斯提到的分数中，如果 p 为 1 就是自然数，加上负分数和零就是今天数学中定义的**有理数**（**rational number**），rational 是以 **ratio**（**比率**）为词根，所以 rational number 的意思是可以表示成（互质两数之）比的数，而不是“理性的”数。按照今天的数学术语，毕达哥拉斯的观点是任何线段长度都可以表示成有理数（最简分数）的形式。

不过毕达哥拉斯的这个观点很快就被自己学派的一个弟子**希帕索斯**（Hippasus）发现是谬论，毕达哥拉斯在几何中的一大贡献就是证明了直角三角形两条直角边的平方和等于斜边的平方，这个被称为**毕达哥拉斯定理**；根据毕达哥拉斯定理，边长为 1 的等腰直角三角形的斜边平方就是 2，问题是存在一个有理数（最简分数）的平方是 2 吗？答案是否定的，可以证明不存在任何一个最简分数的平方等于 2，所以边长为 1 的等腰直角三角形的斜边长度（根号 2）是不能表示成最简分数的形式的，但是所以边长为 1 的等腰直角三角形的斜边长度确是客观存在的，所以毕达哥拉斯对于任意线段长度都可以

[1]如果两个正整数的最大公约数为 1，就称这两个正整数互质。

表示成有理数（最简分数）的论断是错误的，据说希帕索斯刚发现这个谬误时在学派内被下达封口令，不过因为他没有遵循封口令，被毕达哥拉斯的其他门徒扔进大海淹死。

根号 2 的发现是数学史上出现的第一次危机，它使得人类意识到数轴上不是所有的点的坐标都可以使用有理数表示，比如与坐标原点距离长度为根号 2 的点的坐标就无法使用有理数表示，所以这类数被称为**无理数**（irrational number），古希腊时代无理数就已经被发现随后在实际应用中被使用（比如圆周率），但是直到到**微积分**（calculus）出现后的 19 世纪末，德国数学家**戴德金**（Dedekind）提出的**戴德金分割**（Dedekind cut）的概念才给出了无理数的严格定义，完善了实数的连续性理论，第一次数学危机得以解除。

2.1.2 其它进位制系统

当今人类在日常交流中使用的是十进制系统，参照十进制系统记数原理创建的**二进制系统**（binary system）、**八进制系统**（octal system）和**十六进制系统**（hexadecimal system）被广泛应用于数学和计算机等科学领域。

通常而言，b 进制系统是由 0、1、…、b-1 这 b 个数字组成的记数系统，如果 b 小于 10，则这 b 个数字就是阿拉伯数字中的前 b 个数字，例如八进制系统（b=8）就是由 0、1、…、7 这 8 个数字组成的记数系统；如果 b 大于 10，则这 b 个数字就由十个阿拉伯数字（0-9）再增补 b-10 个数字（通常是英文字母）组成，例如十六进制系统（b=16）就是由 0、1、…、9、a、b、c、d、e、f（大小写字母均可）这 16 个数字组成的记数系统，通常使用$(N)_b$表示 b 进制数 N，没有下标的数默认为十进制数。前面章节分析讨论的十进制系统中（乘法分群数系与定位数系）的数量单位以及小数单位适用于任意 b 进制系统。

2.1.2.1 二进制系统

（1）二进制自然数

二进制自然数系统是采用数字 0 和 1（参照十进制）表示数的二进制定位数系，而十进制定位数系是建立在十进制简单分群数系与十进制乘法分群数系的改进基础上，因此了解二进制定位数系怎么表示自然数，就从二进制的简单分群数系和乘法分群数系表示自然数开始，表 2-3 给出了十进制自然数在二进制简单分群数系中的表示形式，然后将其转换为二进制乘法分群数系中的表示形式，最后再转换为二进制定位数系的表示形式（二进制自然数）。

表 2-3 十进制与二进制自然数在二进制简单分群数系与乘法分群数系中的表示

十进制自然数	1	2	3	4	5	6	7	8	9	10	11	12	13
二进制简单分群数系	|	A	A|	B	B|	BA	BA|	C	C|	CA	CA|	CB	CB|
二进制乘法分群数系	1	A	A1	B	B1	BA	BA1	C	C1	CA	CA1	CB	CB1
二进制自然数	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101

二进制简单分群数系中，每 2 个|用 A 替换，每 2 个 A 用 B 替换，每 2 个 B 用 C 替换，其余以此类推。就以十进制自然数 9 为例，全部使用|表示为|||||||||，转换成二进制简单分群数系的过程为：

|||||||||⇔AAAA|⇔BB|⇔C|，进一步转换为乘法分群数系中的形式为 C1；最后转换为二进制定位数系中的形式为 1001，所以 9=$(1001)_2$。

（2）二进制整数转换为十进制整数

将一个二进制自然数转换为十进制自然数，可以遵循将其一步步转换为二进制乘法分群数系表示形式、二进制简单分群数系表示形式、全部由|组成的形式，最后数一数有多少个|即可。

比如二进制自然数$(111)_2$，其二进制乘法分群数系的形式为 BA1，其二进制简单分群数系的形式为 BA|，其中一个 A 相当于 2 个|，一个 B 相当于 2 个 A,相当于 4 个|，所以全部转换为符号|表示的形式为|||||||，共有 7 个|，所以$(111)_2$=7。

上面过程使用算式可以表示为：

$(111)_2$=BA1=B+A+1=4+2+1=7

可以将乘法分群数系中的数量单位 B 和 A 替换为定位数系中对应的数量单位，因为二进制定位数系中相邻的数量单位之间是 2 倍的关系，所以数量单位 1、10、100、1000、…、100…0（n 个 0）对应的十进制数分别为 2^0、2^1、2^2、2^3、…、2^n。

所以$(111)_2$转换为十进制数的计算过程可以表示为：

$(111)_2=(100)_2+(10)_2+(1)_2=4+2+1=7$

再举个例子：

$(1010)_2=(1000)_2+(10)_2=2^3+2^1=10$

为了归纳出一般性形式，将$(1010)_2$转换为十进制数的过程还可以表示为如下形式：

$(1010)_2=(1000)_2+(10)_2=2^3+2^1=1\times2^3+1\times2^1=1\times2^3+0\times2^2+1\times2^1+0\times2^0=10$

那么对于任意二进制自然数转换为十进制数的计算方法为：

$(a_na_{n-1}\cdots a_1a_0)_2=a_n\times(10\cdots0)_2+a_{n-1}\times(10\cdots0)_2+\cdots+a_1\times(10)_2+a_0\times(1)_2=a_n\times2^n+a_{n-1}\times2^{n-1}+\cdots+a_1\times2^1+a_0\times2^0$

自然数的转换很容易扩充到整数上，二进制整数转换为十进制整数的计算方法为：

$\pm(a_na_{n-1}\cdots a_1a_0)_2=\pm(a_n\times2^n+a_{n-1}\times2^{n-1}+\cdots+a_1\times2^1+a_0\times2^0)$

例 2-1 将二进制整数$(11101)_2$与$(-101)_2$转换成十进制整数。

[解] $(11101)_2=2^4+2^3+2^2+2^0=16+8+4+1=29$

$(-101)_2=-(2^2+2^0)=-(4+1)=-5$

（3）十进制整数转换为二进制整数

先讨论十进制自然数转换为二进制自然数的方法。

假定某十进制自然数 N 对应的二进制数为$(a_na_{n-1}\cdots a_1a_0)_2$，则有：

$(a_na_{n-1}\cdots a_1a_0)_2=a_n\times2^n+a_{n-1}\times2^{n-1}+\cdots+a_1\times2^1+a_0\times2^0=N$

注意观察下面这个等式

$a_n\times2^n+a_{n-1}\times2^{n-1}+\cdots+a_1\times2^1+a_0\times2^0=N$

等式左边有 n+1 项相加，其中前 n 项都至少乘以了 1 个 2，所以每一项都是偶数，那么最后一项$a_0\times2^0$（也就是 a_0）是奇数还是偶数取决于 N 是奇数还是偶数，将 N 除以 2，余数是 0 则 a_0=0，余数是 1 则 a_0=1；求出 a_0后，将其移到等式右边，然后两边再同时除以 2 得：

$a_n\times2^{n-1}+a_{n-1}\times2^{n-2}+\cdots+a_1=(N-a_0)\div2$

接下来可以和求 a_0 同样的思路求出 a_1，其余以此类推可分别求出 a_2、…、a_n。

下面找一个具体的例子演示一下上述过程，例如求 14 的二进制数，因为 $14<2^4=16=(10000)_2$,所以 14 对应的二进制数最多是 4 位，假定 $14=(a_3a_2a_1a_0)_2$，则有：

$a_3\times2^3+a_2\times2^2+a_1\times2^1+a_0=14$

因为 14 除以 2 余数为 0，所以 $a_0=0$，两边同时再除以 2 得：

$a_3\times2^2+a_2\times2^1+a_1=14\div2=7$

因为 7 除以 2 余数为 1，所以 $a_1=1$，将 a_1 移到等式右边，然后两边同时再除以 2 得：

$a_3\times2^1+a_2=(7-1)\div2=3$

因为 3 除以 2 余数为 1，所以 $a_2=1$，将 a_2 移到等式右边，然后两边同时再除以 2 得：

$a_3=(3-1)\div2=1$

综上所述：$a_3=1$，$a_2=1$，$a_1=1$，$a_0=0$，$14=(1110)_2$。

也可以按照下面表格的方式记录 14 对应的二进制数的求解过程：如表 2-4 所示。

表 2-4 十进制数 14 对应二进制数的求解过程

除以 2	商(舍去余数)	余数	对应数位
14÷2	7	0	a_0
7÷2	3	1	a_1
3÷2	1	1	a_2
1÷2	0	1	a_3
商为 0 时终止			

对于十进制负整数转换成二进制负整数的方法是先将对应的正整数转换为二进制正整数，前面再加个负号即可。十进制的整数转换为二进制整数的方法可以总结为“**除以 2 取余**”。

例 2-2 将十进制整数-35 转换为二进制整数。

[解] 先将-35 的相反数 35 转换为二进制，然后前面加上负号即可。

35÷2=17 余 1($a_0=1$)

17÷2=8 余 1($a_1=1$)

8÷2=4 余 0($a_2=0$)

4÷2=2 余 0($a_3=0$)

2÷2=1 余 0($a_4=0$)

1÷2=0 余 1($a_5=1$)

商为 0 时终止，所以 $35=(a_5a_4a_3a_2a_1a_0)_2=(100011)_2$，则$-35=(-100011)_2$

关于解题过程并没有固定的格式，前面求 14 的二进制数时用到了列表，本例则使用的是分步列式计算，将求解过程描述清楚明确即可。

（4）二进制小数

前面探讨了十进制小数如何表示连续的数量，下面将其拓展到二进制中。十进制表示小数的方法中，不足一个“单位长度”的数量，会将其逐步缩短十分之一继续进行度量；那么在二进制表示小数的方法中，对于同样情况的处理，是将“单位长度”逐步缩短二分之一继续进行度量，对于二进制形式的小数而言，小数点后面的数位依次为二分位、四分位、八分位等。

如图 2-7 所示，将数轴上的坐标全部改为二进制数，坐标原点为 0，坐标刻度线从长到短依次标记“单位长度”、“1/2 单位长度”、“1/4 单位长度”、“1/8 单位长度”，其余若干个点分为位于坐标 0 和 1 之间，接下来依次求每个点用二进制形式表示的坐标。

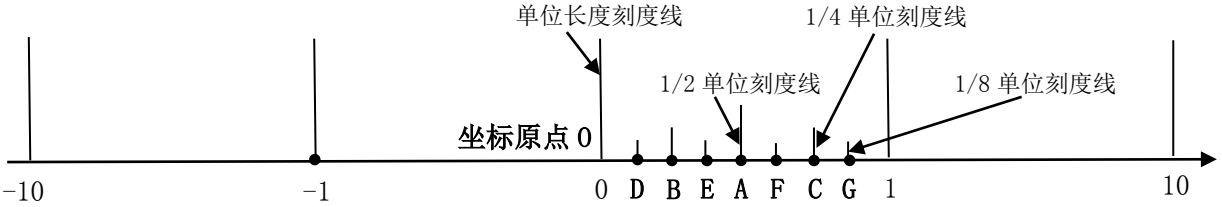


图 2-7 在数轴上使用二进制小数表示点的坐标

①A 点位于 1/2 单位刻度线上，所以 A 的坐标用二进制表示为$(0.1)_2$；

②B 点位于 1/2 单位刻度线的左边（不足 1/2 单位长度），所以对应二分位上数字是 0，其位于 1/4 单位刻度线上，所以对应四分位上数字是 1，所以 B 的坐标用二进制表示为$(0.01)_2$；

③C 点位于 1/2 单位刻度线的右边（超出 1/2 单位长度），所以对应二分位上数字是 1，其位于 1/4 单位刻度线上，所以对应四分位上数字是 1，所以 C 的坐标用二进制表示为$(0.11)_2$；

④D 点位于 1/2 单位刻度线的左边（不足 1/2 单位长度），所以对应二分位上数字是 0，其位于 1/4 单位刻度线的左边（不足 1/4 单位长度），所以对应四分位上数字是 0，其位于 1/8 单位刻度线上，所以对应八分位上数字是 1，所以 D 的坐标用二进制表示为$(0.001)_2$；

⑤E 点位于 1/2 单位刻度线的左边（不足 1/2 单位长度），所以对应二分位上数字是 0，其位于 1/4 单位刻度线的右边（超出 1/4 单位长度），所以对应四分位上数字是 1，其位于 1/8 单位刻度线上，所以对应八分位上数字是 1，所以 E 的坐标用二进制表示为$(0.011)_2$；

⑥F 点位于 1/2 单位刻度线的右边（超出 1/2 单位长度），所以对应二分位上数字是 1，其位于 1/4 单位刻度线的左边（扣除 1/2 单位长度剩余部分不足 1/4 单位长度），所以对应四分位上数字是 0，其位于 1/8 单位刻度线上，所以对应八分位上数字是 1，所以 F 的坐标用二进制表示为$(0.101)_2$；

⑦G 点位于 1/2 单位刻度线的右边（超出 1/2 单位长度），所以对应二分位上数字是 1，其位于 1/4 单位刻度线的右边（扣除 1/2 单位长度剩余部分超出 1/4 单位长度），所以对应四分位上数字是 1，其位于 1/8 单位刻度线上，所以对应八分位上数字是 1，所以 G 的坐标用二进制表示为$(0.111)_2$；

（5）二进制小数转换为十进制小数

借助于图 2-7 来分析一下二进制小数转换为十进制小数的方法，比如 G 点的坐标$(0.111)_2$表示原点 0 到 G 点的线段长度（使用二进制小数表示），从十进制的角度重新计算一下 OG 的长度，就可以求出其使用十进制表示的坐标。图中可以看出 OG 由 1 个“1/2 单位长度”和 1 个“1/4 单位长度”以及 1 个

“1/8 单位长度”组成，“单位长度”为 1，所以计算 OG 长度的算式为：

OB=1/2+1/4+1/8=0.5+0.25+0.125=0.875

所以$(0.111)_2$转换为十进制数的计算过程为：

$(0.111)_2$=1/2+1/4+1/8=0.5+0.25+0.125=0.875

可以将十进制中小数单位的概念移植到二进制中，由前面的分析可知，$(0.1)_2$表示将“单位长度”二等分，所以$(0.1)_2=1/2=2^{-1}$；$(0.01)_2$表示将“单位长度”四等分，所以$(0.01)_2=1/4=2^{-2}$；依次类推可知，整数部分为 0 且小数点后有 n 个 0 和一个 1 组成的二进制小数$(0.0\cdots1)_2=2^{-n}$，那么整数部分为 0 的二进制小数可以通过拆分为小数单位之和的方式转换为对应的十进制小数。

所以$(0.111)_2$转换为十进制数的计算过程还可以表示为：

$(0.111)_2=(0.1)_2+(0.01)_2+(0.001)_2=2^{-1}+2^{-2}+2^{-3}$=0.5+0.25+0.125=0.875

图 2-7 中 F 点坐标$(0.101)_2$转换为十进制数的计算过程可以表示为：

$(0.101)_2=(0.1)_2+(0.001)_2=2^{-1}+2^{-3}$=0.5+0.125=0.625

为了追求形式上的完备，上面两个算式可以变为以下形式：

$(0.111)_2=(0.1)_2+(0.01)_2+(0.001)_2=2^{-1}+2^{-2}+2^{-3}=1\times2^{-1}+1\times2^{-2}+1\times2^{-3}$=0.5+0.25+0.125=0.875

$(0.101)_2=(0.1)_2+(0.001)_2=2^{-1}+2^{-3}=1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$=0.5+0.125=0.625

概括起来可知，对于形如“$0.a_1a_2\cdots a_n$”的二进制小数，其转换为十进制小数的计算方法为：

$(0.a_1a_2\cdots a_n)_2=a_1\times2^{-1}+a_2\times2^{-2}+\cdots+a_n\times2^{-n}$

那么对于既有整数部分又有小数部分，且既有正又有负的的任意二进制数而言，其转换为十进制数的计算方法为：

$\pm(a_na_{n-1}\cdots a_1a_0.a_{-1}a_{-2}\cdots a_{-m})_2=\pm(a_n\times2^n+a_{n-1}\times2^{n-1}+\cdots+a_1\times2^1+a_0\times2^0+a_{-1}\times2^{-1}+a_{-2}\times2^{-2}+\cdots+a_{-m}\times2^{-m})$

例 2-3 分别将二进制数$(101.011)_2$与$-(110.101)_2$转换为十进制数。

[解] $(101.011)_2=2^2+2^0+2^{-2}+2^{-3}$=4+1+0.25+0.125=5.375

$-(110.101)_2=-(2^2+2^1+2^{-1}+2^{-3})=-(4+2+0.5+0.125)$=-6.625

（6）十进制小数转换为二进制小数

将一个十进制小数转换为二进制小数时，遵循整数部分与小数部分分开转换的原则，十进制整数转换为二进制整数的方法前面已经讨论过，下面来分析一下十进制小数转换为二进制小数的方法。

以十进制小数 0.375 为例，其整数部分为 0，因此对应的二进制形式整数部分也为 0，假定其对应的二进制小数为$(0.a_1a_2\cdots a_n)_2$，根据二进制小数转换为十进制小数的方法可知：

$a_1\times2^{-1}+a_2\times2^{-2}+\cdots+a_n\times2^{-n}$=0.375

等式两边同时乘以 2

$a_1+a_2\times2^{-1}+\cdots+a_n\times2^{-(n-1)}$=0.375×2=0.75

可否通过对比等式两边求出 a_1呢？

a_1、a_2、…、a_n都是 0 或 1，如果 a_1=0，则 $a_1+a_2\times2^{-1}+\cdots+a_n\times2^{-(n-1)}=a_2\times2^{-1}+\cdots+a_n\times2^{-(n-1)}<2^{-1}+\cdots+2^{-(n-1)}=1-2^{-(n-1)}<1$

如果 a_1=1，则 $a_1+a_2\times2^{-1}+\cdots+a_n\times2^{-(n-1)}=1+a_2\times2^{-1}+\cdots+a_n\times2^{-(n-1)}\geq1$

实际情况是 $a_1+a_2\times2^{-1}+\cdots+a_n\times2^{-(n-1)}=0.75<1$，因此可以推断 $a_1=0$；

将 $a_1=0$ 代入到前一个等式

$a_2\times2^{-1}+\cdots+a_n\times2^{-(n-1)}=0.75$

两边继续乘以 2

$a_2+a_3\times2^{-1}+\cdots+a_n\times2^{-(n-2)}=0.75\times2=1.5$

如果 $a_2=0$，则 $a_2+a_3\times2^{-1}+\cdots+a_n\times2^{-(n-2)}=a_3\times2^{-1}+\cdots+a_n\times2^{-(n-2)}<2^{-1}+\cdots+2^{-(n-2)}=1-2^{-(n-2)}<1$

如果 $a_2=1$，则 $a_2+a_3\times2^{-1}+\cdots+a_n\times2^{-(n-2)}=1+a_3\times2^{-1}+\cdots+a_n\times2^{-(n-2)}\geq1$

实际情况是 $a_2+a_3\times2^{-1}+\cdots+a_n\times2^{-(n-2)}=1.5>1$，因此可以推断 $a_2=1$；

将 $a_2=1$ 代入到前一个等式

$1+a_3 2^{-1}+\cdots+a_n\times2^{-(n-2)}=1.5$

将 1 移到等式右边

$a_3\times2^{-1}+a_4\times2^{-2}+\cdots+a_n\times2^{-(n-2)}=1.5-1=0.5$

两边同时乘以 2

$a_3+a_4\times2^{-1}+\cdots+a_n\times2^{-(n-3)}=0.5\times2=1$

如果 $a_3=0$，则 $a_3+a_4\times2^{-1}+\cdots+a_n\times2^{-(n-3)}=a_4\times2^{-1}+\cdots+a_n\times2^{-(n-3)}<2^{-1}+\cdots+2^{-(n-3)}=1-2^{-(n-3)}<1$

如果 $a_3=1$，则 $a_3+a_4\times2^{-1}+\cdots+a_n\times2^{-(n-3)}=1+\cdots+a_n\times2^{-(n-3)}\geq1$

实际情况是 $a_3+a_4\times2^{-1}+\cdots+a_n\times2^{-(n-3)}=1$，因此可以推断 $a_3=1$，且 $a_4\times2^{-1}+\cdots+a_n\times2^{-(n-3)}=0$，则 $a_4=\cdots=a_n=0$，也就是说 a_3 之后没有其它数位了。

综上所述，$0.375=(0.011)_2$

上述计算过程可以表示成表 2-5 中的步骤。

表 2-5 十进制小数 0.375 转换为二进制小数的过程

小数部分乘以 2	乘积	整数部分(对应数位)	小数部分
0.375×2	0.75	$0(a_1)$	0.75
0.75×2	1.5	$1(a_2)$	0.5
0.5×2	1	$1(a_3)$	0.0
小数部分为 0 时终止			

十进制小数中负数的转换方式与正数一样，转换后保留负号即可。十进制小数转换为二进制小数方法可以总结为“**乘以 2 取整**”。需要强调地是，十进制小数转换为二进制小数的过程并非总可以在有限地步骤内终止，有时会陷入无限循环的过程，比如十进制小数 0.6 转换成二进制小数的过程如表 2-6 所示。转换过程的第 5 步与第 1 步完全相同，都是 0.6×2，因此可以推断后面开始将重复前面的步骤，这个过程会无限循环，并不会终止，也就是说十进制小数 0.6 对应的二进制小数是一个循环小数“0.100110011001…”，小数部分按照“1001”的顺序一直向后延伸。通常遇到这种情况是保留小数点后若干位，求出近似值代替。

表 2-6 十进制小数 0.6 转换为二进制小数的过程

小数部分乘以 2	乘积	整数部分(对应数位)	小数部分
0.6×2	1.2	1(a_1)	0.2
0.2×2	0.4	0(a_2)	0.4
0.4×2	0.8	0(a_3)	0.8
0.8×2	1.6	1(a_4)	0.6
0.6×2	1.2	1(a_5)	0.2
开启无限循环模式…			

将一个既有整数部分又有小数部分的十进制数转换为二进制数的方法是，先将整数部分按照“**除以 2 取余**”的方法转换为二进制整数部分，然后再将小数部分按照“**乘以 2 取整**”的方法转换为二进制小数部分，最后将两部分合起来即可。

例 2-4 将十进制数 10.625 转换为二进制数。

[解]（1）先转换整数部分 10

10÷2=5　5÷2=2　2÷2=1　1÷2=0

余 0　余 1　余 0　余 1

将余数序列反过来排列构成二进制数的整数部分，即 $10=(1010)_2$

（2）接下来转换小数部分 0.625

0.625×2=1.25　0.25×2=0.5　0.5×2=1.0

整数部分 1　整数部分 0　整数部分 1

整数部分的数字正序排列二进制数的小数部分，即 $0.625=(101)_2$, $10.625=(1010.101)_2$

2.1.2.2 八进制系统

（1）八进制自然数

八进制自然数系统是采用数字 0、1、2、3、4、5、6、7（参照十进制）表示数的八进制定位数系，想要了解八进制系统怎么表示自然数，就从八进制的简单分群数系和乘法分群数系表示自然数开始，表 2-7 给出了十进制自然数在八进制简单分群数系中的表示形式，然后将其转换为八进制乘法分群数系中的表示形式，最后再转换为八进制定位数系的表示形式（八进制自然数）。

八进制简单分群数系中，每 8 个|用 A 替换，每 8 个 A 用 B 替换，每 8 个 B 用 C 替换，其余以此类推。就以十进制自然数 18 为例，全部使用|表示为|||||||| |||||||| ||，转换成八进制简单分群数系（每 8 个|用 A 替换）的形式为 AA||；转换成二进制乘法分群数系的形式为 2A2，进一步转换为八进制定位数系中的形式为 22，所以 $18=(22)_8$。

表 2-7 十进制与八进制自然数在八进制简单分群数系与乘法分群数系中的表示

十进制自然数	1	2	3	4	5	6	7	8	9	10	11	12
八进制简单分群数系	|	||	|||	||||	|||||	||||||	|||||||	A	A|	A||	A|||	A||||
八进制乘法分群数系	1	2	3	4	5	6	7	A	A1	A2	A3	A4
八进制自然数	1	2	3	4	5	6	7	10	11	12	13	14

十进制自然数	13	14	15	16	17	18	19	20
八进制简单分群数系	A|||||	A||||||	A|||||||	AA	AA|	AA||	AA|||	AA||||
八进制乘法分群数系	A5	A6	A7	2A	2A1	2A2	2A3	2A4
八进制自然数	15	16	17	20	21	22	23	24

（2）八进制整数转换为十进制整数

将一个八进制自然数转换为十进制自然数，可以遵循将其一步步转换为八进制乘法分群数系表示形式、八进制简单分群数系表示形式、全部由|组成的形式，最后数一数有多少个|即可。

比如八进制自然数$(234)_8$，其八进制乘法分群数系的形式为 2B3A4，其八进制简单分群数系的形式为 BBAAA||||，其中一个 A 相当于 8 个|，一个 B 相当于 8 个 A,相当于 64 个|，所以全部转换为符号|表示的形式共有 64+64+8+8+8+4=164 个|，所以$(234)_8$=164。

上面过程使用算式可以表示为：

$(234)_8$=2B3A4=2B+3A+4=2×B+3×A+4=2×64+3×8+4=164

可以将乘法分群数系中的数量单位 B 和 A 替换为定位数系中对应的数量单位，因为八进制定位数系中相邻的数量单位之间是 8 倍的关系，所以八进制数量单位 1、10、100、1000、…、100…0（n 个 0）对应的十进制数分别为8^0、8^1、8^2、8^3、…、8^n。

所以$(234)_8$转换为十进制数的计算过程可以表示为：

$(234)_8=2\times B+3\times A+4=2\times(100)_8+3\times(10)_8+4\times(1)_8=2\times8^2+3\times8^1+4\times8^0=164$

再举个例子：

$(205)_8=2\times(100)_8+5\times(1)_8=2\times8^2+5\times8^0=133$

为了归纳出一般性，将$(205)_8$转换为十进制数的过程还可以表示为如下形式：

$(205)_8=2\times(100)_8+5\times(1)_8=2\times8^2+5\times8^0=2\times8^2+0\times8^1+5\times8^0=133$

那么对于任意八进制自然数转换为十进制数的计算方法为：

$(a_na_{n-1}\cdots a_1a_0)_8=a_n\times(10\cdots0)_8+a_{n-1}\times(10\cdots0)_8+\cdots+a_1\times(10)_8+a_0\times(1)_8=a_n\times8^n+a_{n-1}\times8^{n-1}+\cdots+a_1\times8^1+a_0\times8^0$

自然数的转换很容易扩充到整数上，八进制整数转换为十进制整数的计算方法为：

$\pm(a_na_{n-1}\cdots a_1a_0)_8=\pm(a_n\times8^n+a_{n-1}\times8^{n-1}+\cdots+a_1\times8^1+a_0\times8^0)$

例 2-5 分别将八进制数$(127)_8$与$-(176)_8$转换为十进制数。

[解] $(127)_8=1\times8^2+2\times8^1+7\times8^0=64+16+7=87$

$-(176)_8=-(1\times8^2+7\times8^1+6\times8^0)=-(64+56+6)=-126$

（3）十进制整数转换为八进制整数

十进制整数转换为八进制整数的思路和十进制整数转换为二进制整数类似，遵循的是“**除以 8 取余**”的方法，所以接下来直接举例说明转换过程，省略了分析步骤。

例 2-6 分别将十进制数 1156 与-99 转换为八进制数。

[**解**] 先将 1156 转换为八进制数

1156÷8=144 余 4(a_0=4)

144÷8=18 余 0(a_1=0)

18÷8=2 余 2(a_2=2)

2÷8=0 余 2(a_3=2)

商为 0 时终止，所以 1156=$(a_3a_2a_1a_0)_8$=$(2204)_8$

接下来将-99 转换为八进制数，先将 99 转换为八进制数

99÷8=12 余 3(a_0=3)

12÷8=1 余 4(a_1=4)

1÷8=0 余 1(a_2=1)

商为 0 时终止，所以 99=$(a_2a_1a_0)_8$=$(143)_8$，则-99=$(-143)_8$。

（4）八进制小数

八进制表示小数的方法中，不足一个“单位长度”的数量，会将其逐步缩短八分之一继续进行度量，小数点后面的数位依次为八分位、六十四分位、五百一十二分位等。如图 2-8 所示，将数轴上的坐标改为八进制，坐标原点为 0，图中最长的刻度线标记“单位长度”，下面对应的数字是按照“单位长度”计算的坐标（称之为大坐标）；将大坐标 0 与 1 之间的“单位长度”八等分得到“1/8 单位长度”，并增加 7 条较短的刻度线标记“1/8 单位长度”，较短刻度线上的数字 1-7 表示的是按照“1/8 单位长度”计算的坐标（称之为小坐标）；将小坐标 3 与 4 之间的“1/8 单位长度”八等分得到“1/64 单位长度”，并增加 7 条最短的刻度线标记“1/64 单位长度”，按照“1/64 单位长度”计算的坐标称之为小小坐标（图 2-8 中省略了小小坐标对应的数字）；接下来看看数轴上 A、B、C 三点的坐标怎么表示：

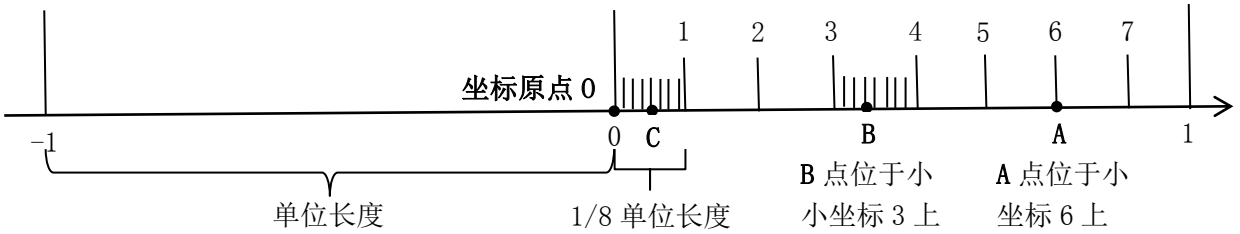


图 2-8 在数轴上使用八进制小数表示点的坐标

①A 点位于位于小坐标 6 上，所以 A 的坐标用八进制数表示为$(0.6)_8$；

②B 点位于大坐标 0 与 1 之间，且位于小坐标 3 与 4 之间，位于小小坐标 3 上，所以 B 的坐标用八进制数表示为$(0.33)_8$；

③C 点位于小坐标 0 与 1 中点，因为距离原点不到 1 个“1/8 单位距离”，所以小数点后第 1 位上数字为 0，小坐标 0 与 1 中点对应的小小坐标是 4，所以 C 的坐标用八进制数表示为$(0.04)_8$。

（5）八进制小数转换为十进制小数

借助于图 2-8 来分析一下八进制小数转换为十进制小数的方法，比如 B 点的坐标$(0.33)_8$表示原点 O 到 B 点的线段长度（使用八进制小数表示），从十进制的角度重新计算一下 OB 的长度，就可以求出其使用十进制表示的坐标。图中可以看出 OB 由 3 个“1/8 单位长度”和 3 个“1/64 单位长度”组成，“单位长度”为 1，所以计算 OB 长度的算式为：

OB=3×(1/8)+3×(1/64)=3/8+3/64=0.375+0.046875=0.421875

所以$(0.33)_8$转换为十进制数的计算过程为:

$(0.33)_8$=3×(1/8)+3×(1/64)=3/8+3/64=0.375+0.046875=0.421875

可以将十进制中小数单位的概念移植到八进制中，由前面的分析可知，$(0.1)_8$表示将“单位长度”八等分，所以$(0.1)_8=1/8=8^{-1}$；$(0.01)_8$表示将“单位长度”六十四等分，所以$(0.01)_2=1/64=8^{-2}$；依次类推可知，整数部分为 0 且小数点后有 n 个 0 和一个 1 组成的二进制小数$(0.0\cdots1)_2=8^{-n}$，那么整数部分为 0 的八进制小数可以通过拆分为各个小数位上的数字与小数单位乘积之和的方式转换为对应的十进制小数。

所以$(0.33)_8$转换为十进制数的计算过程还可以表示为:

$(0.33)_8=3\times(0.1)_8+3\times(0.01)_8=3\times8^{-1}+3\times8^{-2}=3/8+3/64=0.375+0.046875=0.421875$

图 2-8 中 C 点坐标$(0.04)_8$转换为十进制数的计算过程还可以表示为:

$(0.04)_8=4\times(0.01)_8=4\times8^{-2}=4/64=1/16=0.0625$

为了追求形式上的完备，上面计算过程可以变为以下形式：

$(0.04)_8=4\times(0.01)_8=4\times8^{-2}=0\times8^{-1}+4\times8^{-2}=4/64=1/16=0.0625$

概括起来可知，对于形如“$0.a_1a_2\cdots a_n$”的八进制小数，其转换为十进制小数的计算方法为：

$(0.a_1a_2\cdots a_n)_2=a_1\times8^{-1}+a_2\times8^{-2}+\cdots+a_n\times8^{-n}$

那么对于既有整数部分又有小数部分，且既有正又有负的的任意八进制数而言，其转换为十进制数的计算方法为：

$\pm(a_na_{n-1}\cdots a_1a_0.a_{-1}a_{-2}\cdots a_{-m})_8=\pm(a_n\times8^n+a_{n-1}\times8^{n-1}+\cdots+a_1\times8^1+a_0\times8^0+a_{-1}\times8^{-1}+a_{-2}\times8^{-2}+\cdots+a_{-m}\times8^{-m})$

例 2-7 将八进制数$(23.11)_8$转换为十进制数。

[解] $(23.11)_8=2\times8^1+3\times8^0+1\times8^{-1}+1\times8^{-2}=16+3+0.125+0.015625=19.140625$

（6）十进制小数转换为八进制小数

十进制小数转换成八进制小数的思路，与十进制小数转换成二进制小数的思路完全一样，遵循的是**“乘以 8 取整”**的方法。将一个既有整数部分又有小数部分的十进制数转换为八进制数的方法是，先将整数部分按照**“除以 8 取余”**的方法转换为八进制整数部分，然后再将小数部分按照**“乘以 8 取整”**的方法转换为八进制小数部分，最后将两部分合起来即可。

例 2-8 将十进制数 126.634 转换为八进制数（保留小数点后 4 位）。

[解] 先将整数部分 126 转换为八进制整数

126÷8=15　　15÷8=1　　1÷8=0

余 6　　余 7　　余 1

则 126=$(176)_8$

再将小数部分 0.634 转换为八进制小数

0.634×8=5.072　0.072×8=0.576　0.576×8=4.608　0.608×8=4.864

整数部分 5　　整数部分 0　　整数部分 4　　整数部分 4

转换过程并没有终止，根据题目保留小数点后 4 位的要求，0.634 转换成八进制约等于$(0.5044)_8$，因此 126.634 转换为八进制数约等于$(176.5044)_8$。

2.1.2.3 十六进制系统

十六进制系统是采用数字 0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f（参照十进制）表示数的记数系统，十六进制系统的规律和八进制完全类似，只不过使用的数字比八进制多出 8 个而已，所以关于十六进制数与十进制数之间互相转换的方法可以直接套用八进制数与十进制数之间互相转换的方法。

十六进制系统中的 16 个数字对应的十进制数如表 2-8 所示。

表 2-8 十六进制数字对应的十进制自然数

十六进制数字	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
十进制自然数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

（1）十六进制整数转换为十进制整数

十六进整数中相邻的数量单位之间是 16 倍的关系，所以十六进制数量单位 1、10、100、1000、…、100…0（n 个 0）对应的十进制数分别为 16^0、16^1、16^2、16^3、…、16^n。

十六进制整数转换为十进制整数的计算方法为：

$\pm(a_n a_{n-1}\cdots a_1 a_0)_{16}=\pm(a_n\times16^n+a_{n-1}\times16^{n-1}+\cdots+a_1\times16^1+a_0\times16^0)$

需要强调地是，如果某个 a_i 是大于 9 的数字，上述计算过程中要将其转换成对应的十进制数。

例 2-9 分别将十六进制数$(12f)_{16}$与$-(1a6)_{16}$转换为十进制数。

[解] $(12f)_{16}=1\times16^2+2\times16^1+15\times16^0=256+32+15=303$

$-(1a6)_{16}=-(1\times16^2+10\times16^1+6\times16^0)=-(256+160+6)=-422$

上述两个转换的计算过程中，$(12f)_{16}$中的 f 与$-(1a6)_{16}$中的 a 要变成对应的十进制数 15 和 10。

（2）十进制整数转换为十六进制整数

十进制整数转换为十六进制整数的思路和十进制整数转换为二进制以及八进制整数类似，遵循的是“**除以 16 取余**”的方法，所以接下来直接举例说明转换过程，省略了分析步骤。

例 2-10 分别将十进制数 1156 与-59 转换为十六进制数。

[解] 先将 1156 转换为十六进制数

1156÷16=72 余 4(a_0=4)

72÷16=4 余 8(a_1=8)

4÷16=0 余 4(a_2=4)

商为 0 时终止，所以 1156=$(a_2a_1a_0)_{16}$=$(484)_{16}$

接下来将-59 转换为十六进制数，先将 59 转换为十六进制数

59÷16=3 余 11(a_0=b)，注意余数 11 对应的十六进制数字是 b

3÷16=0 余 3(a_1=3)

商为 0 时终止，所以 59=$(a_1a_0)_{16}$=$(3b)_{16}$，则-59=$(-3b)_{16}$。

（3）十六进制小数转换为十进制小数

可以将十进制中小数单位的概念移植到十六进制中，$(0.1)_{16}$ 表示将“单位长度”十六等分，所以$(0.1)_{16}=1/16=16^{-1}$；$(0.01)_{16}$表示将“单位长度”二百五十六等分，所以$(0.01)_{16}=1/256=16^{-2}$；依次类推可知，整数部分为 0 且小数点后有 n 个 0 和一个 1 组成的十六进制小数$(0.0\cdots1)_{16}=16^{-n}$，那么整数部分为 0 的十六进制小数可以通过拆分为各个小数位上的数字与小数单位乘积之和的方式转换为对应的十进制小数。

概括起来可知，对于形如“$0.a_1a_2\cdots a_n$”的十六进制小数，其转换为十进制小数的计算方法为：

$(0.a_1a_2\cdots a_n)_{16}=a_1\times16^{-1}+a_2\times16^{-2}+\cdots+a_n\times16^{-n}$

那么对于既有整数部分又有小数部分，且既有正又有负的的任意十六进制数而言，其转换为十进制数的计算方法为：

$\pm(a_na_{n-1}\cdots a_1a_0.a_{-1}a_{-2}\cdots a_{-m})_{16}=\pm(a_n\times16^n+a_{n-1}\times16^{n-1}+\cdots+a_1\times16^1+a_0\times16^0+a_{-1}\times16^{-1}+a_{-2}\times16^{-2}+\cdots+a_{-m}\times16^{-m})$

如果其中某个 a_i 是大于 9 的数字，上述计算过程中要将其转换成对应的十进制数。

例 2-11 将十六进制数$(2c.5)_{16}$转换为十进制数。

[解] $(2c.5)_{16}=2\times16^1+12\times16^0+5\times16^{-1}=32+12+0.3125=44.3125$

（4）十进制小数转换为十六进制小数

十进制小数转换成十六进制小数的思路，与十进制小数转换成二进制以及八进制小数的思路完全一样，遵循的是“**乘以 16 取整**”的方法。将一个既有整数部分又有小数部分的十进制数转换为十六进制数的方法是，先将整数部分按照“**除以 16 取余**”的方法转换为十六进制整数部分，然后再将小数部分按照“**乘以 16 取整**”的方法转换为十六进制小数部分，最后将两部分合起来即可。

例 2-12 将十进制数 178.6 转换为十六进制数（保留小数点后 2 位）。

[解] 先将整数部分 178 转换为十六进制整数

178÷16=11　　11÷16=0

余 2　　余 b

则 178=$(b2)_{16}$

再将小数部分 0.6 转换为十六进制小数

0.6×16=9.6　　0.6×16=9.6

整数部分 9　　整数部分 9

按照题目保留小数点后 2 位的要求，0.6 转换成十六进制约等于$(0.99)_{16}$，因此 178.6 转换为十六进制数约等于$(b2.99)_{16}$。

2.1.2.4 二进制与八进制之间的互相转换

首先观察一下八进制自然数与二进制自然数对照表（如表 2-9 所示），

2-9 八进制自然数与二进制自然数对照表

八进制	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
二进制	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

其次可以归纳出如下规律：

①组成八进制的单个数字共有 8 个，每个八进制数字转换成二进制数最多需要 3 位（$2^3=8$）；

②随机选择一个八进制的两位数比如$(15)_8$，左边数位上的数字 1 对应的二进制数也是 1，右边数位上的数字 5 对应的二进制数是 101，然后将两者合到一起组成的数 1101 恰好是$(15)_8$对应的二进制数；

③观察表 2-9 中其它的两位八进制，发现具有同样规律，八进制数转换成二进制数时，先将该八进制数每个数位上的数字分别转换成二进制数，然后按照数位顺序合并在一起，就完成了转换。

接下来进行大胆地猜测，任意位数的八进制数（包括小数），将每个数位上的数字转换成二进制数，然后依照数位顺序合并在一起，就构成了该数对应的二进制数。

最后来进行求证，证明思路是先将八进制数转换成十进制数，然后再转换成二进制数的形式。由于正数与其相反数在转换方式上完全相同，所以后面只讨论正数的情况，同理可推得负数的情况，零在所有的进位制系统中都是 0。

对于任意八进制正数$(a_m a_{m-1}\cdots a_1 a_0.a_{-1}a_{-2}\cdots a_{-n})_8$，其对应的十进制数为：

$$(a_m a_{m-1}\cdots a_1 a_0.a_{-1}a_{-2}\cdots a_{-n})_8=a_m\times8^m+a_{m-1}\times8^{m-1}+\cdots+a_1\times8^1+a_0\times8^0+a_{-1}\times8^{-1}+a_{-2}\times8^{-2}+\cdots+a_{-n}\times8^{-n}$$

接下来将等式右边做如下变换：

①对于整数部分的每个数字$a_i(i=0,1,\cdots,m)$，假定其对应的二进制数为$(a_{i1}a_{i2}a_{i3})_2$，则有

$a_i=a_{i1}\times2^2+a_{i2}\times2+a_{i3}$；

②对于小数部分的每个数字$a_{-j}(j=1,\cdots,n)$，假定其对应的二进制数为$(a_{-j1}a_{-j2}a_{-j3})_2$，则有

$a_{-j}=a_{-j1}\times2^2+a_{-j2}\times2+a_{-j3}$；

将上述两种变换代入到等式中可得：

$$(a_m a_{m-1}\cdots a_1 a_0.a_{-1}a_{-2}\cdots a_{-n})_8=a_m\times8^m+a_{m-1}\times8^{m-1}+\cdots+a_1\times8^1+a_0\times8^0+a_{-1}\times8^{-1}+a_{-2}\times8^{-2}+\cdots+a_{-n}\times8^{-n}=$$

$$(a_{m1}\times2^2+a_{m2}\times2+a_{m3})\times8^m+\cdots+(a_{01}\times2^2+a_{02}\times2+a_{03})\times8^0+(a_{-11}\times2^2+a_{-12}\times2+a_{-13})\times8^{-1}+\cdots+(a_{-n1}\times2^2+a_{-n2}\times2+a_{-n3})\times8^{-n}=$$

$$(a_{m1}\times2^2+a_{m2}\times2+a_{m3})\times(2^3)^m+\cdots+(a_{01}\times2^2+a_{02}\times2+a_{03})\times(2^3)^0+(a_{-11}\times2^2+a_{-12}\times2+a_{-13})\times(2^3)^{-1}+\cdots+$$

$$(a_{-n1}\times2^2+a_{-n2}\times2+a_{-n3})\times(2^3)^{-n}=$$

$$(a_{m1}\times2^2+a_{m2}\times2+a_{m3})\times2^{3m}+\cdots+(a_{01}\times2^2+a_{02}\times2+a_{03})\times2^0+(a_{-11}\times2^2+a_{-12}\times2+a_{-13})\times2^{-3}+\cdots+(a_{-n1}\times2^2+a_{-n2}\times2+a_{-n3})\times2^{-3n}=$$

$$a_{m1}\times2^{3m+2}+a_{m2}\times2^{3m+1}+a_{m3}\times2^{3m}+\cdots+a_{01}\times2^2+a_{02}\times2^1+a_{03}\times2^0+a_{-11}\times2^{-1}+a_{-12}\times2^{-2}+a_{-13}\times2^{-3}+\cdots+a_{-n1}\times2^{-(3n+2)}+a_{-n2}\times2^{-(3n+1)}+a_{-n3}\times2^{-3n}$$

$$=(a_{m1}a_{m2}a_{m3}\cdots a_{01}a_{02}a_{03}.a_{-11}a_{-12}a_{-13}\cdots a_{-n1}a_{-n2}a_{-n3})_2$$

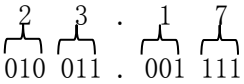
上面等式变换过程的最后结果是形如“$a_{m1}a_{m2}a_{m3}\cdots a_{01}a_{02}a_{03}.a_{-11}a_{-12}a_{-13}\cdots a_{-n1}a_{-n2}a_{-n3}$”的数字串，它正是八进制数$(a_m a_{m-1}\cdots a_1 a_0.a_{-1}a_{-2}\cdots a_{-n})_8$按照数位顺序将每个数位上的数字转换成二进制数后合并在一起形

成的数字串。

下面通过一些例子来详述二进制数与八进制数之间的互相转换方法。

例 2-13 将八进制数$(23.17)_8$转换成二进制数。

[解] 将$(23.17)_8$的每位数字分别转换成二进制数，然后按照数位顺序排列，需要强调地是每个数位都要转换成 3 位二进制数的形式，转换后不足 3 位的需要在左边补 0（比如数字 1 转换成二进制后的形式是 001）

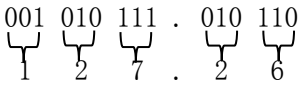


然后将排列的二进制数字串中最左边的若干个 0 去掉，即完成转换。

$(23.17)_8=(10011.001111)_2$

例 2-14 将二进制数$(1010111.01011)_2$转换成八进制数。

[解] 将二进制数转换为八进制数时，以小数点为起点，左起和右起分别按照 3 个数字一组的方式分组，整数部分最左边一组不足 3 位的需要在前面补 0 凑足 3 位，小数部分最右边一组不足 3 位的需要在后面补 0 凑足 3 位，比如本例中的二进制数分组后形式如下（每组之间使用空格分开）：



然后将每组中 3 位二进制数转换成八进制数字并按既定顺序排列起来即可。

$(1010111.01011)_2=(127.26)_8$

2.1.2.5 二进制与十六进制之间的互相转换

十六进制自然数与二进制自然数对照关系如表 2-10 所示。

2-10 十六进制自然数与二进制自然数对照表

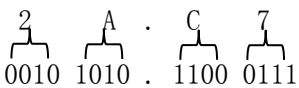
十六进制	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
二进制	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

二进制数和十六进制数之间的相互转换方法与二进制数和八进制数之间的相互转换方法完全类似，只不过每个十六进制数字对应的二进制数最多有 4 位，所以将十六进制数转换成二进制数时是将每个数字分别转换成 4 位二进制数然后按数位顺序合并在一起，而将二进制数转换成十六进制数时是按照 4 个数位一组的方式分别将每组的 4 位二进制数转换成十六进制数字。

下面通过一些例子来详述二进制数与十六进制数之间的互相转换方法。

例 2-15 将十六进制数$(2A.C7)_{16}$转换成二进制数。

[解] 将$(2A.C7)_{16}$的每位数字分别转换成二进制数，然后按照数位顺序排列，需要强调地是每个数位都要转换成 4 位二进制数的形式，转换后不足 4 位的需要在左边补 0



然后将排列的二进制数字串中最左边的若干个 0 去掉，即完成转换。

$(2A.C7)_{16}=(101010.11000111)_2$

例 2-16 将二进制数$(1010111.01011)_2$转换成十六进制数。

[解] 将二进制数转换为十六进制数时，以小数点为起点，左起和右起分别按照 4 个数字一组的方式分组，整数部分最左边一组不足 4 位的需要在前面补 0 凑足 4 位，小数部分最右边一组不足 4 位的需要在后面补 0 凑足 4 位，比如本例中的二进制数分组后形式如下（每组之间使用空格分开）：

```
0101 0111 . 0101 1000
 5    7   .  5    8
```

然后将每组中 4 位二进制数转换成十六进制数字并按既定顺序排列起来即可。

$(1010111.01011)_2=(57.58)_{16}$

2.2 数据存储

使用某些符号按照既定格式表示的数据称为该数据的**编码**（code），使用阿拉伯数字表示的数与使用罗马数字表示的数，就是针对数值型数据的两种不同的编码，将数据由一类格式转变成另一类格式的过程也称为**编码**（encoding）,此处出现了两个编码，显然前一个编码（code）是名词，后一个编码（encoding）是动词，英文词汇的使用上不会出现混淆，翻译成中文词汇时难免会出现一词多义的情况，有时要结合上下文注意区分。

构成计算机存储系统的最基本的物理元器件通常只有两种状态，使用这两种状态模拟数据的表示，最简单直接的方法就是将一种状态表示 0，另一种状态表示 1，多个物理元器件的排列就可以表示 0 和 1 组成的数字串，所以计算机存储设备可以存储的信息是 0 和 1 组成的数字串信息。那么存储在计算机中的数据必需能够被编码成 0 和 1 组成的数字串形式。不同的数据类型编码成 0-1 数字串的方法也不同，但是都与二进制系统存在着千丝万缕的联系。

计算机存储的信息中，单个数字 0 或 1 被称为**位**（**binary digit**，英文缩写位 **bit**），因为 bit 直译可读作**比特**，所以 binary digit 有时也被称为**比特位**，比特位是计算机系统中最小的存储单位，多个比特位组成的数字串被称为**位模式**（**bit pattern**），8 个比特位构成的位模式被称为**字节**（**byte**）。因为计算机存储的容量是有限的，并且为了方便数据在计算机系统中的存取和处理，保存同类数据所占用的比特位数也是相同的，每种类型的数据存储时占用多少比特位，不同的系统有不同的约定。在计算机系统中约定字节（8 个比特位）是最小的存储单位，也就是说任何类型的数据保存在计算机中都至少占据 1 个字节的存储容量（也称为存储空间），且都是字节的整数倍。

2.2.1 数值型数据在计算机中的存储

2.2.1.1 整数的存储与运算

整数的存储不仅涉及到数的存储，还涉及到符号（正负号）的存储，接下来介绍几种常见的将整数编码为位模式的方法。

（1）无符号表示法

存储到计算机中的整数，最简单的一种编码方式就是将该整数转换成二进制形式，不过这种方法

不能区分具有相同绝对值的正负整数，也就是说这种编码形式适合于保存非负整数，因此这种编码方式也称为**无符号表示法**（unsigned representation）。

将一个整数按照无符号表示法编码成 n 位位模式的方法是：将该数转换成二进制形式，如果转换后的数位不足 n 位，则在该数的左边补 0 凑足 n 位。

反过来将一个使用 n 位无符号表示法表示的位模式还原为整数（解码）的方法是：将位模式最左边开始的连续若干个 0 舍去，然后将剩余的位模式（当做二进制整数）转换为十进制整数。

n 位位模式可以有 2^n 种不同的排列，共可以表示 2^n 个非负整数，从小到大排列为 0 到 2^n-1，所以使用 n 位无符号表示法可以表示的最小整数是 0，可以表示的最大整数是 2^n-1。

大于 2^n-1 的整数转换为二进制整数时数位会超出 n 位，此时如果还使用 n 位无符号表示法编码，则选取最右边的 n 位位模式作为编码信息，那么左边若干位会被舍去，再反过来解码后得到的整数并不是原来的整数，这种因为超出存储限制导致的编码错误被称为**溢出**（overflow）。

例 2-17 求整数 7 按照 8 位无符号表示法编码后的位模式。

[解] $7=(111)_2$，左边补 0 凑足 8 位，则 00000111 即为所求的位模式。

例 2-18 某整数使用无符号表示法编码后的位模式为 00101011，求该整数。

[解] 去掉编码串的最左边的两个 0 得到位模式 101011，$(101011)_2=2^5+2^3+2^1+2^0=32+8+2+1=43$，则该整数为 43。

（2）符号加绝对值表示法

只能存储非负整数的计算机系统显然是功能不完整的，为了解决负整数的编码问题，可以考虑将符号位引入到编码后的位模式，也就是说将最左边的一位作为符号位，0 表示非负整数，1 表示负整数，其余的位模式作为存储这个数绝对值的数据位，这种编码方法被称为**符号加绝对值表示法**（sign-and-magnitude representation）。

比如整数-7，如果还是用 8 位位模式来编码，就将编码后最左边的一位作为符号位，用 1 表示。然后将-7 的绝对值 7 转换成二进制形式$(111)_2$，除去最左边的符号位，还有 7 位用于保存$(111)_2$，于是将 111 左边补 4 个 0 凑足 7 位得到 0000111，再加上最左边的符号位 1，最终-7 的编码形式是 10000111。对于整数 7，其编码后的形式是 00000111。

将一个整数按照符号加绝对值表示法编码成 n 位位模式的方法是：先将该数的绝对值转换成二进制形式，如果转换后的数位不足 n-1 位，则在该数的左边补 0 凑足 n-1 位，然后根据该数的正负性确定最左边符号位的值，非负为 0，负则为 1。

反过来将一个使用 n 位符号加绝对值表示法表示的位模式还原为整数（解码）的方法是：将符号位除外的 n-1 位位模式当做二进制整数（这 n-1 位位模式种最左边的若干个 0 舍去）转换为十进制整数，然后根据符号位上数字是 0 还是 1，决定是否在转换后的整数前面加上负号。

接下来讨论下 n 位符号加绝对值表示法能够表示的整数的范围，能表示的最大整数编码后的位模式是 01…1（n-1 个 1），解码后的值是 $2^{n-1}-1$；能表示的最小整数编码后的位模式是 11…1（n 个 1），解码后的值是$-(2^{n-1}-1)$。

不过随之出现了一个新的问题，整数列$-(2^{n-1}-1)$到 $2^{n-1}-1$ 共有 $2\times(2^{n-1}-1)+1=2^n-1$ 个数（正负各有

2^{n-1}-1 个，再加上 1 个 0），换个角度从排列组合的层面思考，n 位位模式能够组成 2^n 个不同的排列，每一种对应一个整数的编码而论，n 位位模式应该可以表示 2^n 个整数（见面介绍的 n 位无符号表示法就能表示 2^n 个整数），为什么 n 位符号加绝对值表示法能够表示的整数个数会少一个呢？

稍加思考即可发现，n 位符号加绝对值表示法中位模式 00…0 表示 0，而 10…0 表示-0，实际上 0 是没有正负性的，也就是说位模式 10…0 没有表示一个有效的负整数，因此导致能够表示的整数少了一个。

例 2-19 求整数 9 和-9 按照 8 位符号加绝对值表示法编码后的位模式。

[解] $9=(1001)_2$，将位模式 1001 左边补 0 凑足 8 位，则 00001001 即为 9 对应的位模式编码；

将位模式 1001 左边补 0 凑足 7 位，符号位用 1 表示，则 10001001 即为-9 对应的位模式编码。

例 2-20 某整数使用符号加绝对值表示法编码后的位模式为 10101011，求该整数。

[解] 最左边一位是 1，所以该位模式表示的十个负整数；

去掉符号位以及左边第二位上的 0 得到位模式 101011，$(101011)_2=2^5+2^3+2^1+2^0=32+8+2+1=43$，

则该整数为-43。

（3）二进制补码表示法

前面的讨论可知，符号加绝对值表示法有一个小漏洞，就是其编码的整数中会出现正负两个 0，这导致了位模式 1…0 的无效性（数学中并不存在-0 的概念），下面以 4 位编码长度为例尝试着修复一下符号加绝对值表示法，如表 2-11 所示。

表 2-11 符号加绝对值表示法修复方案

位模式	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
无符号表示法	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
符号加绝对值表示法	0	1	2	3	4	5	6	7	-0	-1	-2	-3	-4	-5	-6	-7
修复方案 1	0	1	2	3	4	5	6	7	-8	-1	-2	-3	-4	-5	-6	-7
修复方案 2	0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1

表 2-11 中第一行给出了所有 4 位位模式的递增序排列，第二行给出的是首行中每个位模式按照无符号表示法解码后对应的整数，第三行给出的是首行中每个位模式按照符号加绝对值表示法解码后对应的整数。接下来讨论如何对符号加绝对值表示法进行修复，使得每一个位模式都能有效利用起来，修复遵循“0 开头的位模式对应非负整数，1 开头的位模式对应负整数”的规律，0 开头的位模式共有 8 个，可以表示 8 个非负整数，因此对应的整数的范围是 0 到 7；1 开头的位模式共有 8 个，可以表示 8 个负整数，因此对应的整数的范围是-1 到-8；为什么是这两个整数区间，因为所有 16 个位模式对应的整数范围应该是一个正负尽量对称且保持递连贯的整数集。基于上述原则提出以下两种修复方案：

①直接在符号加绝对值表示法的基础上仅仅修改一个位模式 1000 对应的整数，将其由原来对应的-0 改为-8；这个修改方案看似简单，但是却带来一个“不和谐”的情况；所谓“和谐”的情况，就是有序的位模式与其对应的整数也保持正序（或逆序），比如无符号表示法中位模式与其对应的整数是正

序关系（均保持递增序）；符号加绝对值表示法中，位模式被分为 0 开头与 1 开头两种，0 开头的位模式与其对应的整数也是正序关系，1 开头的位模式则与其对应的整数是反序关系（位模式 1000 到 1111 是递增序，对应的整数从-0 到-7 是递减序）；现在将位模式 1000 对应的整数改成-8，在所有 1 开头的位模式对的整数序列中，-8 是个“突兀”的存在。

②在修复方案 1 的基础上对 1 开头的位模式对应的整数在排序上做个调整，也就是将原本的位模式 1001 到 1111 对应的-1 到-7 反序，则 1001 到 1111 对应-7 到-1，那么 1000 到 1111 对应-8 到-1，这样就使得 1 开头的位模式也保持了“和谐性”。这种编码方案就是现在几乎所有的计算机系统都采用的存储整数的**二进制补码表示法**（two’s complement representation）。

下面分析一下二进制补码表示法编码与解码的方法，将无符号表示法与二进制补码表示法对照起来（如表 2-12 所示）观察一下，看能否发现什么规律？之所以选择无符号表示法，是因为无符号表示法就是二进制表示法。通过观察可以发现如下规律：

表 2-12 无符号表示法与二进制补码表示法对照表

位模式	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
无符号表示法	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
二进制补码表示法	0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1
负整数的另一种表示	0	1	2	3	4	5	6	7	-8+0	-8+1	-8+2	-8+3	-8+4	-8+5	-8+6	-8+7
2 的幂之和形式	0	2^0	2^1	2^0+2^1	2^2	2^0+2^2	2^1+2^2	$2^0+2^1+2^2$	-2^3+0	-2^3+2^0	-2^3+2^1	$-2^3+2^0+2^1$	-2^3+2^2	$-2^3+2^0+2^2$	$-2^3+2^1+2^2$	$-2^3+2^0+2^1+2^2$

①非负整数的二进制补码表示法就是直接将该数转换成二进制形式；

②负整数的二进制补码表示法的规律不那么明显，观察表 2-12 中最右边的 8 列，也就是所有 1 开头的位模式所在列，每一列中无符号表示法对应的整数（第 2 行）与二进制补码表示法对应的整数（第 3 行）比较，两者恰好相差 16，比如 1010 使用无符号表示法对应的整数是 10，使用二进制补码表示法对应的整数是-6，10 比-6 大 16；再比如比如 1111 使用无符号表示法对应的整数是 15，使用二进制补码表示法对应的整数是-1，15 比-1 大 16；反之，求-6 对应的 4 位二进制补码时，可以先将-6 加上 16 等于 10，然后直接将 10 转换成二进制形式 1010，即为所求。因此任意负整数对应的 4 位二进制补码是将其先加上 16，然后再转换成二进制形式；

③二进制补码表示法的解码过程可以按照前面所述的方式分非负整数与负整数两种情况来描述，不过还可以寻找出更加统一的规律（也就是不分正负的解码方法），观察表 2-12 中倒数第 2 行行可以发现，4 位二进制补码表示法对应的整数中，0 开头的位模式对应的整数依次为 0、1、…、7，1 开头的位模式对应的整数依次可表示为-8+0、-8+1、…、-8+7；然后将上述整数全部拆分成 2 幂之和的形式（表 2-12 中最后一行），左起开始分别观察第 1 列与第 9 列、观察第 2 列与第 10 列、…、观察第 8 列与第 16 列，随机选出一对作比较，比如第 4 列与第 12 列对应的位模式分别为 0011 和 1011，两者仅仅最左边一位不同，其余 3 位相同，变成 2 幂之和的形式分别为 2^1+2^0 与 $-2^3+2^1+2^0$，也就是说后者比前

者多加了一个-2^3，而两者相同的部分 2^1+2^0，恰好是（0011 和 1011 两个位模式相同的右边 3 位）011 转换成二进制数时的算式，其它每一对也都具有相同规律；归纳起来就是：形如 $0a_2a_1a_0$的 4 位二进制补码表示法对应的整数为 $a_2\times2^2+a_1\times2^1+a_0\times2^0$，形如 $1a_2a_1a_0$ 的 4 位二进制补码表示法对应的整数为 $-2^3+a_2\times2^2+a_1\times2^1+a_0\times2^0$，进一步可统一为：形如 $a_3a_2a_1a_0$ 的 4 位二进制补码表示法对应的整数为 $a_3\times(-2^3)+a_2\times2^2+a_1\times2^1+a_0\times2^0$；

上述归纳出来的规律可以推广到 n 位二进制补码表示法的情形。

将一个整数按照二进制补码表示法编码成 n 位位模式的方法是：如果该数为非负整数则将该数转换成 n 位二进制形式，如果该数是负整数则将 2^n加上该数的和转换成 n 位二进制形式；

反过来将一个使用 n 位二进制补码表示法表示的位模式 $a_{n-1}\cdots a_1a_0$解码后得到的整数是[3]：

$a_{n-1}\times(-2^{n-1})+a_{n-2}\times2^{n-2}+\cdots+a_1\times2^1+a_0\times2^0$

例 2-21 求整数 9 和-9 按照 8 位二进制补码表示法编码后的位模式。

[解] $9=(1001)_2$，将位模式 1001 左边补 0 凑足 8 位，则 00001001 即为 9 对应的二进制补码；

$2^8+(-9)=256-9=247=(11110111)_2$，则 11110111 即为-9 对应的二进制补码。

例 2-22 两整数使用二进制补码表示法编码后的位模式分别为 00101011 和 10101011,求这两整数。

[解] 位模式 00101011 解码后对应的整数为：$2^5+2^3+2^1+2^0=32+8+2+1=43$

位模式 10101011 解码后对应的整数为：$-2^7+2^5+2^3+2^1+2^0=-128+32+8+2+1=-128+43=-85$

（4）二进制补码表示的整数的运算

加法运算是四则混合运算里面最基础的运算，减法一个数等于加上这个数的相反数，所以减法运算可以转换成加法运算实施，乘法运算可以分解为多步加法运算，除法运算则可以分解为多步减法运算，所以对于二进制补码表示的整数的运算只分析讨论加法运算。

比特位之间的加法运算法则为 0+0=0，0+1=1，1+0=1，1+1=10，多位位模式对位加时采取数位右对齐的方式，有进位时向左边相邻的一位进位。

二进制补码最大的优点在于实施加法运算时可以直接将补码形式的位模式对位加，不需要解码成统一的二进制数后再做加法。接下来看几个例子，所有的例子均使用 8 位二进制补码。

例 2-23 分别将 17 和 22 对应的 8 位二进制补码形式对位相加，然后将相加结果解码，得到的整数是多少？

[解] 17 和 22 都是正整数，所以对应二进制补码就是其转换成二进制后的形式，

$(17)_2=10001$，补 0 凑足 8 位得到其二进制补码形式 00010001；

$(22)_2=10110$，补 0 凑足 8 位得到其二进制补码形式 00010110；

两者对位相加

```
进位        1
      0 0 0 1 0 0 0 1
    + 0 0 0 1 0 1 1 0
    -----------------
      0 0 1 0 0 1 1 1
```

相加后的结果 00100111 解码后对应整数为 $2^5+2^2+2^1+2^0=32+4+2+1=39$，恰好是 17+22 的结果。

例 2-23 的结果比较好理解，因为正整数的二进制补码就是用二进制数表示，补码相加实际上就是二进制数相加，相加后结果解码实际上就是二进制转换为十进制的过程。不过两个正整数相加时有时会超出整数能够表示的范围，这种情况被称为**溢出**（flow）。

例 2-24 分别将 117 和 22 对应的 8 位二进制补码形式对位相加，然后将相加结果解码，得到的整数是多少？

[解] 117 和 22 都是正整数，所以对应二进制补码就是其转换成二进制后的形式，

$(117)_2$=1110101，补 0 凑足 8 位得到其二进制补码形式 01110101；

$(22)_2$=10110，补 0 凑足 8 位得到其二进制补码形式 00010110；

两者对位相加

```
进位   1 1 1    1
       0 1 1 1 0 1 0 1
     + 0 0 0 1 0 1 1 0
     -----------------
       1 0 0 0 1 0 1 1
```

相加后的结果 10001011 解码后对应整数为$-2^7+2^3+2^1+2^0$=-128+8+2+1=-117，结果明显是错误的，原因在于 117+22=139，超出了 8 位二进制补码能够表示的最大整数 127，出现了溢出错误。

例 2-25 分别将-17 和 22 对应的 8 位二进制补码形式对位相加，然后将相加结果解码，得到的整数是多少？

[解] -17 是负整数，所以对应二进制补码就是 2^8-17=239 转换成二进制后的形式，

$(239)_2$=11101111，所以-17 的二进制补码形式为 11101111；

22 是正整数，所以对应二进制补码就是其转换成二进制后的形式，

$(22)_2$=10110，补 0 凑足 8 位得到其二进制补码形式 00010110；

两者对位相加

```
进位   1 1 1 1 1 1 1
         1 1 1 0 1 1 1 1
       + 0 0 0 1 0 1 1 0
       -----------------
         0 0 0 0 0 1 0 1
```

相加后的结果 00000101，解码后对应整数为 2^2+2^0=4+1=5，恰好是(-17)+22 的结果。

从上面的加法竖式可以看到，运算结果如果加上最左边的进位 1，则超出 8 位的数位限制，因此最左边的进位 1 被舍去，舍去后的结果正确，因此这种情况不属于溢出错误。

例 2-26 分别将 17 和-22 对应的 8 位二进制补码形式对位相加，然后将相加结果解码，得到的整数是多少？

[解] 17 是正整数，所以对应二进制补码就是其转换成二进制后的形式，

$(17)_2$=10001，补 0 凑足 8 位得到其二进制补码形式 00010001；

-22 是负整数，所以对应二进制补码就是 2^8-22=234 转换成二进制后的形式，

$(234)_2$=11101010，所以-22 的二进制补码形式为 11101010；

两者对位相加

```
  0 0 0 1 0 0 0 1
+ 1 1 1 0 1 0 1 0
-----------------
  1 1 1 1 1 0 1 1
```

相加后的结果 11111011，解码后对应整数为$-2^7+2^6+2^5+2^4+2^3+2^1+2^0$=-128+64+32+16+8+2+1=-5，恰好是 17+(-22)的结果。

例 2-27 分别将-17 和-22 对应的 8 位二进制补码形式对位相加，然后将相加结果解码，得到的整数是多少？

[解] -17 是负整数，所以对应二进制补码就是 2^8-17=239 转换成二进制后的形式，

$(239)_2$=11101111，所以-17 的二进制补码形式为 11101111；

-22 是负整数，所以对应二进制补码就是 2^8-22=234 转换成二进制后的形式，

$(234)_2$=11101010，所以-22 的二进制补码形式为 11101010；

两者对位相加

```
进位  1 1 1    1 1 1
        1 1 1 0 1 1 1 1
      + 1 1 1 0 1 0 1 0
      -----------------
        1 1 0 1 1 0 0 1
```

相加后的结果 11011001 解码后对应整数为$-2^7+2^6+2^4+2^3+2^0$=-128+64+16+8+1=-128+89=-39，恰好是(-17)+(-22)的结果，最左边多出的进位 1 舍去，这种情况也不属于溢出错误。

例 2-28 分别将-117 和-22 对应的 8 位二进制补码形式对位相加，然后将相加结果解码，得到的整数是多少？

[解] -117 是负整数，所以对应二进制补码就是 2^8-117=139 转换成二进制后的形式，

$(139)_2$=10001011，所以-117 的二进制补码形式为 10001011；

-22 是负整数，所以对应二进制补码就是 2^8-22=234 转换成二进制后的形式，

$(234)_2$=11101010，所以-22 的二进制补码形式为 11101010；

两者对位相加

```
进位 1         1    1
       1 0 0 0 1 0 1 1
     + 1 1 1 0 1 0 1 0
     -----------------
       0 1 1 1 0 1 0 1
```

相加后的结果 01110101（舍去了最左边的进位 1）不用解码就知道结果是正整数，这明显是错误的，两个负整数加起来结果不可能是正整数。原因在于(-117)+(-22)=-139，超出了 8 位二进制补码能够表示的最小整数-128，出现了溢出错误。

从前面的几个例子可以看出，二个整数相加，两个加数同为正时，相加后的结果如果大于系统能够存储的最大整数值，会发生溢出错误，这种情况被称为**上溢**（overflow）；两个加数同为负时，相加

后的结果如果小于系统能够存储的最小整数值，也会发生溢出错误，这种情况被称为**下溢**（underflow）；两个加数为一正一负时，不会出现溢出。在运算结果没有溢出错误时，不管两个加数的正负性如何，直接将加数的补码对位加的结果（有时需要舍去最左边数位的进位）恰好就是和的补码，假定两个加数为 A 与 B，则有等量关系成立：A 的补码+B 的补码=(A+B)的补码。

尽管是前面的例子观察出上面的规律，但是却不能作为最终结论，个例不具备整体上的代表性，接下来从抽象的角度来分析。

例 2-29 整数 A 与 B 使用 n 位二进制补码表示，假定 A+B 的结果不会溢出，试分析一下 A 的补码+B 的补码与(A+B)的补码之间存在的关系。

[解] 首先对相关符号作一些说明，A 的补码用 $A_{补}$ 表示，A 的二进制形式用 A_{bin} 表示，，$A_{bin}+B_{bin}=(A+B)_{bin}$ 作为不证自明的结论使用。分以下 4 种情况讨论：

①A≥0，B≥0

由 A≥0 可知 $A_{补}=A_{bin}$，由 B≥0 可知 $B_{补}=B_{bin}$，则 $A_{补}+B_{补}=A_{bin}+B_{bin}=(A+B)_{bin}$

由 A≥0，B≥0 可知 A+B≥0，所以 $(A+B)_{bin}=(A+B)_{补}$，则 $A_{补}+B_{补}=A_{bin}+B_{bin}=(A+B)_{bin}=(A+B)_{补}$

这种情况下 $A_{补}$ 与 $B_{补}$ 对位加的结果就是 $(A+B)_{补}$，这与例 2-23 中显示的情况相同。

②A≥0，B<0，|A|≥|B|

由 A≥0 可知 $A_{补}=A_{bin}$，由 B<0 可知 $B_{补}=(2^n+B)_{bin}$，

则 $A_{补}+B_{补}=A_{bin}+(2^n+B)_{bin}=(A+2^n+B)_{bin}=(2^n+A+B)_{bin}=(2^n)_{bin}+(A+B)_{bin}$，

由|A|≥|B|可知 A+B≥0，则 $(A+B)_{bin}=(A+B)_{补}$，则 $A_{补}+B_{补}=(2^n)_{bin}+(A+B)_{bin}=(2^n)_{bin}+(A+B)_{补}$，

$(2^n)_{bin}=(10\cdots0)_2$ 是一个 n+1 位数，所以 $(2^n)_{bin}+(A+B)_{补}$ 也是 n+1 位数，这说明 $A_{补}$ 与 $B_{补}$ 在对位加之后最左边一位产生了进位，舍去这个进位之后的结果就是 $(A+B)_{补}$，这与例 2-25 中显示的情况相同。

③A≥0，B<0，|A|<|B|

由 A≥0 可知 $A_{补}=A_{bin}$，由 B<0 可知 $B_{补}=(2^n+B)_{bin}$，

则 $A_{补}+B_{补}=A_{bin}+(2^n+B)_{bin}=(A+2^n+B)_{bin}=[2^n+(A+B)]_{bin}$，由|A|<|B|可知 A+B<0，则 $[2^n+(A+B)]_{bin}=(A+B)_{补}$，

则 $A_{补}+B_{补}=[2^n+(A+B)]_{bin}=(A+B)_{补}$

这种情况下 $A_{补}$ 与 $B_{补}$ 对位加的结果就是 $(A+B)_{补}$，这与例 2-26 中显示的情况相同。

④A<0，B<0

由 A<0 可知 $A_{补}=(2^n+A)_{bin}$，由 B<0 可知 $B_{补}=(2^n+B)_{bin}$，

所以 $A_{补}+B_{补}=(2^n+A)_{bin}+(2^n+B)_{bin}=(2^n+2^n+A+B)_{bin}=[2^n+2^n+(A+B)]_{bin}=(2^n)_{bin}+[2^n+(A+B)]_{bin}$，

由 A<0 且 B<0 可知 A+B<0，则 $[2^n+(A+B)]_{bin}=(A+B)_{补}$，

则 $A_{补}+B_{补}=(2^n)_{bin}+[2^n+(A+B)]_{bin}=(2^n)_{bin}+(A+B)_{补}$

这种情况下 $A_{补}$ 与 $B_{补}$ 对位加之后最左边一位会产生进位，舍去这个进位之后的结果就是 $(A+B)_{补}$，这与例 2-27 中显示的情况相同。

2.2.1.1 实数（小数）的存储与运算

按照数学中的定义，实数包含有理数和无理数，而有理数包含整数和分数，而小数是分数使用定位数系表示的形式，所以实数包含整数、分数（小数）和无理数。计算机存储实数时，由于计算机的

存储格式仅限于 0 和 1 组成的数字串（位模式），所以对于分数和无理数必需先转换成小数形式后再考虑如何编码成位模式，对于部分只能转换成循环小数的分数以及所有的无理数而言，计算机只能存储其近似值。所以从计算机系统的视角看到的数，只有整数（没有小数部分）和实数（有小数部分）两种，对这两种数据类型的编码方法是截然不同的，整数的存储前面介绍过，是采用二进制补码表示法编码，接下来介绍实数的编码方法。

十进制小数被存储到计算机中之前先要被转换成二进制小数，然后将其转换成规范化形式，首先来看看二进制数的规范化形式是怎么样的？

（1）二进制数的规范化形式

二进制数的规范化形式类似于十进制数的科学计数法形式，科学计数法是将十进制数表示成“$\pm a.xx\cdots x\times 10^n$”的形式（其中 a 是 1-9 中的数字，n 是整数）；二进制数的规范化形式是将二进制数表示成“$\pm a.xx\cdots x\times 2^n$”的形式（其中 a 是数字 1，n 是整数）。

十进制数中小数点左移 1 位则该数缩小 10 倍，右移 1 位则该数扩大 10 倍；同样规律也适用于二进制数，二进制数中小数点左移 1 位则该数缩小 2 倍，右移 1 位则该数扩大 2 倍。

假定$(111.101)_2$变成规范化形式为$(1.11101)_2\times 2^n$，由于$(111.101)_2$是通过左移 2 位后变成$(1.11101)_2$，所以$(111.101)_2$是在缩小2^2倍后变成$(1.11101)_2$，所以$(111.101)_2=1.11101\times 2^2$；这个过程可以理解为小数部分因为小数点左移 2 位缩小了2^2倍，那么在保持大小不变的情况下指数部分就要扩大2^2倍，所以指数上要加 2，原数$(111.101)_2$可以看做$(111.101)_2\times 2^0$（指数部分为 0）。

同理$(0.00101)_2$通过小数点向右移动 3 位后变成规范化形式为$(1.01)_2\times 2^{-3}$，向右移动 3 位后小数部分扩大2^3倍，保持大小不变的话，指数部分要缩小2^3倍，指数由原来的 0 变为-3。

二进制小数变成规范化形式后由符号位（负还是非负）、小数（$a.xx\cdots x$）和指数（2^n）这三部分组成，二进制小数的存储涉及到这三部分的分别存储。符号位的存储需要保存的是该数“非负”还是“负”，所以仅需占用 1 个比特位足以；小数部分仅需要存储小数点后面的数字，因为整数部分固定是 1，这可以当做默认信息不需要存储；指数部分仅需要存储指数 n 即可，2 作为固定的幂也可当做默认信息不需要存储。

指数 n 尽管是整数，但是存储其采用的编码并不是二进制补码，而是余码。

（2）余码表示法

通常固定位数的整数编码基本上都差不多是“正负各半”，比如 4 位二进制补码表示的整数范围是-8 到 7，还有一种思路表示-8 到 7 的所有整数，就是将每个整数先加上 8 然后再变成二进制形式，-8 到 7 的所有整数全体加上 8 之后变成 0 到 15 的所有整数，对应的二进制形式为 0000 到 1111；解码时先将其转换成十进制数然后减去 8。

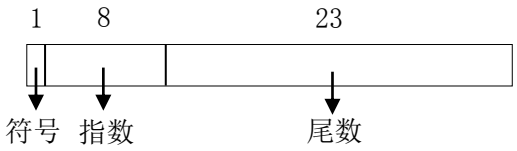
通过将原本大体上“正负各半”的整数集通过全体成员加上一个固定的正整数后，变成非负整数集，然后再转换成二进制数进行编码的方法就是**余码表示法**（excess representation）。

余码表示法中被加入的正整数被称为偏移量，编码位数不同，要表示的整数范围不同，使用的偏移量就不同。同样是 4 位编码长度，如果要使用余码表示法编码整数-7 到 8，则使用的偏移量是 7。可以用符号 excess_k 表示偏移量为 k 的余码表示法。

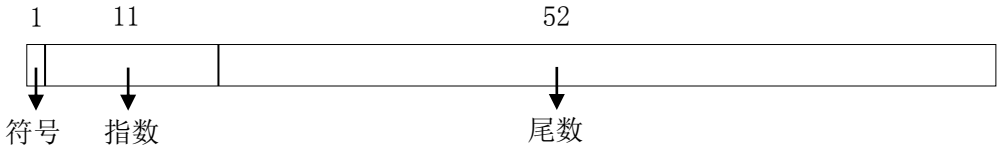
现在我们了解到了，计算机存储实数的大致方法是将该数使用规范化表示后，存储符号信息、小数点后面的数字以及指数这三部分信息，这种存储实数的方法被称为**浮点表示法**（floating-point representation）。

（3）存储实数（小数）的单精度和双精度格式

浮点表示法的格式遵循美国**电气和电子工程师协会**（Institute of Electrical and Electronics Engineers，英文缩写为 IEEE）提出的两种标准（如图 2-9 所示），一种是**单精度**（single）表示法，使用 32 位位模式；另一种是**双精度**（double）表示法，使用 64 位位模式。



单精度表示法
符号占 1 位：0 表示非负，1 表示负
指数占 8 位：使用 excess_127 码
尾数占 23 位：不足部分末尾（右边）补 0



双精度表示法
符号占 1 位：
指数占 11 位：
使用 excess_1023 码
尾数占 52 位：

图 2-9 浮点数表示法的 IEEE 标准

单精度格式共占 32 位，最左边 1 位是符号位，0 表示非负，1 表示负；左边第 2 位到第 9 位（共 8 位）是存储指数，使用 excess_127 码；左边第 10 位到末尾的最后 23 位存储尾数（也就是小数点后面的数字），尾数信息一定要从左边第 10 位开始存储，不足 23 位的话在末尾补 0。

双精度格式共占 64 位，最左边 1 位是符号位，0 表示非负，1 表示负；左边第 2 位到第 12 位（共 11 位）是存储指数，使用 excess_1023 码；左边第 13 位到末尾的最后 52 位存储尾数（也就是小数点后面的数字），尾数信息一定要从左边第 13 位开始存储，不足 52 位的话在末尾补 0。

例 2-30 写出 9.75 使用单精度表示法编码后的信息

[解] 9.75 是非负数，所以符号位是 0；

$9.75=(1001.11)_2=(1.00111)_2\times2^3$

指数 3 使用 excess_127 表示为 3+127=130 的二进制形式 10000010

尾数部分为 00111000000000000000000（后面补 0 凑足 23 位）

9.75 使用单精度表示法表示为 0 10000010 00111000000000000000000

例 2-31 写出-9.75 使用单精度表示法编码后的信息

[解] -9.75 是负数，所以符号位是 1；

参考例 2-30 可知，-9.75 使用单精度表示法表示为 1 10000010 00111000000000000000000

例 2-32 某十进制数使用单精度表示法表示为 11001010000000000111000100001111，求出该数。

[解] 将单精度编码后的位模式分割成符号位、指数、尾数三部分

1 10010100 00000000111000100001111

符号位为 1，所以该数为负数；

指数位信息 10010100 变成十进制整数为$(10010100)_2$=128+16+4=148，148-127=21，所以指数位 21；

结合尾数部分信息，小数部分为$(1.00000000111000100001111)_2$；

该数的二进制形式的规范化表示为$-(1.00000000111000100001111)_2\times2^{21}$；

转换成普通二进制小数$-(1.00000000111000100001111)_2\times2^{21}=-(10000000011110001000011.11)_2$；

转换成十进制数$-(10000000011110001000011.11)_2=-(2^{21}+2^{12}+2^{11}+2^{10}+2^6+2^1+2^0+2^{-1}+2^{-2})=-2104378.75$；

（4）实数（小数）的运算

对于浮点表示法表示的实数之间的运算也仅仅讨论加法运算，两个实数之间的加法最关键地是要将小数点对齐，如果两个实数的指数不同，则需要调整其中一个数的小数点位数使得两个数的指数部分相同，再将小数部分相加，如有必要则将相加后的结果变成规范化形式，最后再编码成浮点（单精度或双精度）表示法形式。下面看两个例子。

例 2-33 实数 A 与 B 使用单精度表示法编码的形式分别为

A 0 10000001 01110000000000000000000

B 0 10000110 01000011110000000000000

求 A 与 B 之和的单精度表示形式

[解] 根据 A 的单精度表示形式可知，符号位是 0，指数位信息是 10000001，尾数位信息是 0111；指数位信息 10000001 转换成二进制数为 129，按照余 127 码解码后得 129-127=2，所以 A 的规范化表示形式为$(1.0111)_2\times2^2$；

根据 B 的单精度表示形式可知，符号位是 0，指数位信息是 10000110，尾数位信息是 0100001111；指数位信息是 10000110 转换成二进制数为 134，按照余 127 码解码后得 134-127=7，所以 B 的规范化表示形式为$(1.0100001111)_2\times2^7$；

接下来对齐指数，有两种思路：

①将 A 的小数点左移 5 位，使得指数变为 7 向 B 看齐；$A=(1.0111)_2\times2^2=A=(0.000010111)_2\times2^7$；然后将移位后的 A 与 B 对位加$(0.000010111)_2\times2^7+(1.0100001111)_2\times2^7=(0.000010111+1.0100001111)_2\times2^7=(1.0100111101)_2\times2^7$；

②将B的小数点右移5位,使得指数变为2向A看齐;$B=(1.0100001111)_2\times2^7=B=(101000.01111)_2\times2^2$；然后将 A 与移位后的 B 对位加$(1.0111)_2\times2^2+(101000.01111)_2\times2^2=(1.0111+101000.01111)_2\times2^2=(101001.11101)_2\times2^2$；

观察上面的两种思路发现，后一种思路要多算一步，即将 B 的小数点移位和 A 相加之后的结果多一步转换成规范化形式的步骤，按照第一种思路往下计算时绝大多数情况下相加后的结果依然是规范化形式，不需要再转换；因此在两个实数相加时，对齐指数这一步，优先选择指数小的数左移小数点向指数较大的数看齐。

最后将$(1.0100111101)_2\times2^7$变成单精度表示形式为 0 100000110 01001111010000000000000

验证一下，A 解码后是 5.75，B 解码后是 161.875， A+B 解码后是 167.625，结果无误。

对于正负号相反的数相加，除了对齐指数外，还要将两个数的绝对值相减，这时一定要注意先判断哪个数的绝对值较大，一定要用绝对值较大的减去绝对值较小的，然后确定相减后的符号是正还是负，相减时采取小数点前后对位减，不够减从左边相邻的一位借位。

例 2-34 实数 A 与 B 使用单精度表示法编码的形式分别为

A 0 10000001 01110000000000000000000

B 1 10000110 01000011110000000000000

求 A 与 B 之和的单精度表示形式

[解] 本例中的 A 与上例中的 A 相同,本例中的 B 是上例中 B 的相反数,参考上例结果对其指数后：

$A=(0.000010111)_2\times2^7$，$B=-(1.0100001111)_2\times2^7$；

显然 B 的绝对值更大，所以 A 与 B 相加后结果是负数，接下来将 B 的小数部分的绝对值减去 A 的小数部分：

```
借位              1 1
|B|         1. 0 1 0 0 0 0 1 1 1 1
 A        - 0. 0 0 0 0 1 0 1 1 1
-------------------------------------
             1. 0 0 1 1 1 0 0 0 0 1
```

则 $A+B=(A-|B|)=-(|B|-A)=-((1.0100001111)_2-(0.000010111)_2)\times2^7=-(1.0011100001)_2\times2^7$；

将其表示成单精度形式为 1 10000110 00111000010000000000000，解码后为-156.125；

最后验证一下，A 解码后是 5.75，B 解码后是-161.875，5.75+(-161.875)=-156.125。

2.2.2 其它多媒体数据在计算机中的存储

2.2.2.1 文本的存储

文本是由各种各样的字符组成，因此字符存储的问题解决了，文本存储的问题就迎刃而解。绝大多数文本都是人类发明的自然语言，语言则是由文字组成，构成语言的文字越少，存储起来就越方便，编码方案就越简单。

比如英语是由 26 个英文字母组成，若干个字母组成单词，若干个单词组成短语或语句，若干条语句组成一篇文本。所以英文文本的存储仅需要考虑 26 个英文字母以及若干标点符号的存储即可，给这些字符（英文字母和标点符号）设计编码方案也不太复杂。

（1）ASCII

美国国家标准学会（American National Standards Institute，英文缩写为 ANSI）在 1967 年发布了**美国信息交换标准代码**（American Standard Code for Information Interchange，英文缩写为 ASCII），它包含了 128 个字符在计算机中存储时对应的位模式编码，编码方式很简单，给每个字符分配一个十进制自然数编号，从 0 开始到 127 为止，然后将这些自然数编号转换成二进制形式，即为每个字符对应的 ASCII 码。$128=2^7$，所以理论上来说存储这 128 个字符只需要 7 位即可，但是在计算机系统中约定了 8 个比特位（也就是 1 个字节）是存储数据的最小单位，所以 ASCII 中的每个字符也采用 8 位存储格式，每个字符占据 1 个字节的存储容量。ASCII 中的这 128 个字符里面包含全体英文字母的大小写，0-9 的数字，还有一些基本的标点符号以及数学运算符号，计算机键盘上大部分按键对应的字符都在 ASCII 码表里可以找到，完整的 ASCII 码信息如表 2-13 所示。

表 2-13 中每个字符对应的编码都是转换成十进制后的形式,其中前 32 个(0-31)和最后 1 个(127)字符是控制字符或通信专用字符，控制字符包括 LF（换行）、CR（回车）、FF（换页）、DEL（删除）、BS

（退格)、BEL（振铃）等，通信专用字符包括 SOH（文头)、EOT（文尾)、ACK（确认）等；这些字符通常都是不可显示的字符，也就是说输出时屏幕上看不到这些字符，但是这些字符会控制光标的位置，比如 LF（换行）符，会让光标从上一行移到下一行。ASCII 码表中其它字符都是可以显示的字符，包括 52 个英文字母（大小写各 26 个)、0-9 的数字、基本数学运算符和一些标点符号等。

有了 ASCII 码表英文文本的存储就比较简单了，直接将文本中的每个字符转换成对应的 ASCII 码即可。按照这种格式保存在计算机中的文件被称为**文本文件**（**text file**)，最常见的文本文件是后缀名为“.txt”的文件。

表 2-13 ASCII 码表

编码	字符	编码	字符	编码	字符	编码	字符	编码	字符	编码	字符	编码	字符	编码	字符
0	NUL	16	DLE	32	SPC	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	”	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	’	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	S1	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

8 位位模式可以对 256（2^8=256）个字符进行编码，ASCII 里面只有 128 个字符，对应的 8 二进制编码都是 0 开头的位模式，1 开头的位模式并没有使用到。所以 1981 年 IBM 公司发布了扩展的 ASCII 码表，增加了 128 个新的字符，包括了某些带重音的字元、一个常用于表示数学符号的小写希腊字母表、一些块型和线状图形字元等。不过扩展的 ASCII 码表并非国际标准。

（2）汉字编码

ASCII 码解决了英文文本的编码问题，非英文的语言文字的编码问题则需要另想办法，和英文类似由字母构成的语言在文本编码方面处理起来有一定优势，而汉字构成的中文则麻烦许多。常用的汉字

有几千个，给每一个汉字设计一个位模式编码是一项很繁琐的工作，我国**国家标准化管理委员会**（Standardization Administration of China，英文缩写为SAC）在1981年5月1日发布了《信息交换用汉字编码字符集（基本集）》（简称GB2312），现在常说的**国标码**就是指的这个编码字符集。不过国标码并不是汉字在计算机中保存的编码形式，而且汉字的计算机化也不止存储需要编码，输入输出也需要编码，下面简要介绍这些编码的来龙去脉。

①输入码

计算机键盘上有26个英文字母键，所以英文的输入不需要编码。但是汉字就不一样了，目前键盘依然是主要的输入工具，所以如何将汉字转换成键盘上的键码是汉字输入要解决的问题，每个汉字都有对应的键盘输入码（简称输入码）。目前最常用的输入码就是汉字对应的汉语拼音，上世纪八九十年代拼音输入法还不成熟时，五笔字型是比较流行的输入码，其它流行过的输入码还有接下来要介绍的区位码。

②区位码

区位码是对汉字的一种十进制数字编码，其编码方案是将6763个汉字以及682个其它符号（共计7445个符号）放入一个94行94列的表格中，每个汉字或其它符号占据一个单元格；这6763个汉字中一级汉字有3755个，均为常用汉字，按拼音顺序排列，分布在表格的第16-55行；二级汉字3008个，为不太常用汉字，按部首排列，分布在表格的第56-87行；682个其它符号分布在表格的前9行，其余所在行空置，上述分布了图形字符与汉字的所在行的少数部分列也是空置的，这些空位是留待将来新增符号使用。

对表格中每个符号的编码使用4位十进制数，左边2位数对应其所在行，右边2位数对应其所在列；比如第1行第9列的符号‘々’对应区位码是0109，计算思维的‘计’字对应的区位码是2838，所以它位于第28行38列。汉字与区位码的双向查询可以查阅区位码表，不过也很容易在互联网上在线查询。区位码可以作为汉字的输入码（需要在计算机上安装区位输入法），不过记住3000多个常用汉字的区位码也不是一件很容易的事，所以使用区位输入法的用户越来越少，不过区位码最主要的功能还是是作为接下来要介绍的国标码的参考。

③国标码

区位码是十进制编码，如果要转换成计算机存储的位模式编码，不是简单地将区位码对应的十进制数转换成二进制数，而是要经历一系列眼花缭乱的操作，这第一步就是由区位码转换成国标码。国标码就是前面提到的我国在1981年发布的GB2312，具体转换方式是将区位码前两位数和后两位数分别转换成十六进制数，然后再分别加上$(20)_{16}$后再合在一起就构成了国标码（十六进制表示形式），所以国标码采取的是2个字节（16位）的存储容量。下面看一个例子：

例2-35 ‘计’字的区位码是2838，求其对应的国标码。

[解] 区位码的前两位数28=$(1c)_{16}$，$(1c)_{16}+(20)_{16}=(3c)_{16}$；

区位码的后两位数38=$(26)_{16}$，$(26)_{16}+(20)_{16}=(46)_{16}$；

则‘计’字的国标码是$(3c46)_{16}$。

将‘计’字的国标码转换成2进制数，是不是就是‘计’字在计算机中的存储形式了呢？答案是

否定的，下面来分析下原因。

‘计’字的国标码是$(3c46)_{16}$转换成二进制的形式为 0011 1100 0100 0110，使用计算机存储需要 16 位（2 个字节），不过有一个背景信息需要强调一下，前面介绍的 ASCII 码是国际化字符编码的标准，国际化意味着全球通用的标准，所以其它后来出现的字符编码都必需兼容 ASCII 码，再来看看‘计’字的国标码 0011 1100 0100 0110，前 8 位（转换成十进制是 60）对应 ASCII 码表中的字符‘<’，后 8 位（转换成十进制是 70）对应 ASCII 码表中的字符‘F’，因此计算机系统在对表示符号的位模式 0011 1100 0100 0110 进行解码时，究竟是按照国标码的标准解码成汉字‘计’呢？还是按照 ASCII 码的标准解码成字符‘<’和‘F’呢？出现这种解码上的二义性，就表明国标码不兼容 ASCII 码，所以汉字‘计’想要存储在计算机中，需要对其国标码进一步转换成机内码。

④机内码

ASCII 码表中每个字符虽然占据 1 个字节，但是其位模式编码的最高位都是 0，所以出于兼容 ASCII 码的考虑，将汉字国标码的两个字节中每个字节的最高位都设置为 1，就不会在解码上与 ASCII 码出现二义性了。

例 2-36 ‘计’字的国标码是$(00111100\ 01000110)_2$，求其对应的机内码。

[解] ‘计’字的国标码 00111100 01000110，将其前一个字节 00111100 的最高位改成 1 变成 10111100，后一个字节 01000110 的最高位改成 1 变成 11000110，再合在一起构成的位模式 10111100 11000110，即为‘计’字存储在计算机中的机内码形式。

不过有一个问题可以深入思考下，有没有可能出现某个汉字国标码的 2 个字节最高位都是 1，而不需要将其转换成机内码的情况？

⑤字型码

汉字字型码用于汉字的输出，通常有点阵和矢量两种表示方式。

使用点阵表示的字型也称为字模，简易型汉字为 16×16 点阵，提高型汉字为 24×24 点阵以及 32×32 点阵等。图 2-10 给出了汉字‘中’的字模及其对应的字型码，该字模使用了 16×16 点阵，点阵由 16×16 的方格组成，涂黑的小方格描出了‘中’字的字型，点阵中白方格用 0 表示，黑方格用 1 表示，将 16 行 16 列的方格转换成了 16 个 16 位的位模式，就构成了‘中’字的字型码。

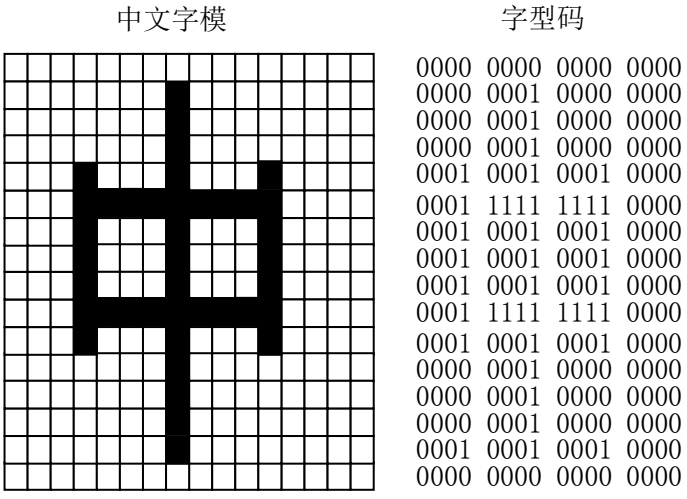


图 2-10 汉字‘中’的字模与字型码

矢量表示方式存储的是描述汉字字型的轮廓特征，当要输出汉字时，通过计算机的计算，由汉字字型描述生成所需大小和形状的汉字点阵。矢量化字型描述与最终文字显示的大小，分辨率无关，因此可以产生高质量的汉字输出。Windows 中使用的 TrueType 技术就是汉字的矢量表示方式。

国标码 GB2312 中只包含简体字，1984 年台湾发布实施了《台湾地区繁体中文标准字符集》（该编码最初为五大中文套装软件设计，所以简称为**BIG5**），采用双字节编码，共收录 13053 个中文字；我国在 1995 年 12 月发布了 GB2312 的扩充版本**GBK**，收录 21003 个汉字，也包含 BIG5 中的所有繁体字。2000 年 3 月再次发布了 GBK 的扩充版本**GB18030**，其中收录 27484 个汉字，还包括中国少数民族文字以及日本和朝鲜文字，采用单字节、双字节和四字节三种方式对字符编码，兼容 ASCII、GB2312 和 GBK。

（3）UNICODE 与 UCS

随着计算机在全球范围的普及，各国为了解决本国语言的输入输出以及存储问题，设计了各种各样的字符编码集，这些编码集虽然都兼容 ASCII 码表，但是各自之间并不能互相兼容，使用一种编码保存的文件如果用不兼容编码集的软件打开就会出现乱码，所以计算机系统需要一个全球统一的字符编码集，里面能包含世界上每个人使用的所有字符。

国际非盈利组织**统一码联盟**（the Unicode Consortium）在上世纪 90 年代初期发布的 **UNICODE**（**万国码**）就在这种背景下应运而生，它为每种语言中的每个字符设定了统一并且唯一的编码，以满足跨语言、跨平台进行文本转换、处理的要求。统一码联盟提出的 UNICODE 标准中给出了收录的字符信息，并不包括这些字符的存储方法，针对 UNICODE 字符存储的编码实现方案目前常用的有 UTF8、UTF16、UTF32。

国际标转化组织（International Organization for Standardization，英文缩写为 ISO）出于统一全球字符集的考虑也发布了一种被称为**基于通用字符集**（Universal Character Set，英文缩写为 **UCS**）的字符集，其对应的编码实现方案有 **UCS2** 和 **UCS4**。

UNICODE 与 UCS 两者早在发布初期已经有了互相兼容的意识，目前两者正在携手发展。

2.2.2.2 音频的存储

声音是由物体振动产生的波动现象，它通过空气、液体或固体等介质的传播并能被人和动物的听觉器官所感知。最初发出振动的物体叫声源，声音以波的振动形式传播，频率在 20 Hz~20 kHz 之间的声音是可以被人耳识别的。人能够识别的声音也称为**音频**（audio）。

音频存储到计算机中通常需要经过**采样**（sampling）、**量化**（quantization）、**编码**（encoding）三个步骤[4]。

（1）采样

声波是一种模拟信号，将随时间连续变化的声波的振幅，按一定的时间间隔采集样值，从而形成在时间上不连续的脉冲序列，这一过程被称为采样。对声波的采样是不断进行的，相邻两个采样点的时间间隔也是相同的，采样点选取通常有一定的规律，每隔一段时间就进行一次采样，单位时间对声波的采样次数被称为**采样频率**（sampling frequency），采样频率越高越能捕捉到声波的细微变化。

（2）量化

采样之后得到数量相当大的采样值，这些采样值不可能恰好都是整数，所以需要使用一定的转换

和取舍将这些采样值全部转换成整数，因为整数的编码过程比小数的编码过程简单许多，所以量化环节主要是给后续的编码环节提供便利。在采用这种四舍五入的方法量化后，相比原声波信号产生了一定程度的误差，当这种误差值小到一定程度时，人耳也不能加以辨别，这样对还原声波信号不会产生过分的影响，是人耳听觉器官所能接受的。

（3）编码

音频的编码则是将量化后的样本值表示成位模式，通常量化后的值如果是非负整数，则采用无符号整数表示法给样本值编码；量化后的值如果正负整数都包含，则采用符号加绝值表示法给样本值编码；对每个采样值的编码长度被称为采样位数，采样的频率越高，采集到的不同大小的样本值就越多，需要使用的采样位数也越多。

目前主流的音频格式是 MP3，它采用每秒 44100 个样本的采样频率，采样位数是 16 位。

2.2.2.3 图片的存储

计算机能够存储和处理的图片主要有 2 类：**矢量图**（vector graphic）和**位图**（bitmap）。矢量图形使用直线和曲线来描述图形，这些图形的元素是一些点、线、矩形、多边形、圆和弧线等等，它们都是通过数学公式计算获得的[5]。矢量图形在计算机处理的图形中所占比例较小，尤其是色彩多样的图片不适合存储成矢量图的格式，接下来主要介绍位图的存储原理。

呈现在我们眼中的现实世界是一幅五彩斑斓的画卷，画家使用各种色彩描绘出栩栩如生的人或物，因此图片可以看做是各种各样的色彩组成。为了记录下构成图片的所有色彩，可以将图片按照行和列分成若干小方格，分的越细，每个小方格中包含的色彩就越接近，直到最后每个小方格缩小成肉眼所见的单一颜色的点，然后将所有点的颜色量化（和音频的量化类似），最后再对所有点的颜色值编码，就得到了图片的数字化存储格式。

将图片输入到计算机中通常要借助于专门的设备，比如扫描仪和数码相机等。扫描仪扫描一张图片的过程实际上就是一个将图片进行编码并存储的过程。将图片分成小方格的细密程度使用**分辨率**（resolution ratio）来衡量，比如将图片的行 800 等分，列 600 等分，被分成 480000 个小格子，其中每个小格子被称为一个**像素**（pixel），对应的分辨率就用 800×600 表示，分的像素越多分辨率就越高，按照存储格式还原输出后的图片的质量就越好。图片的输出质量除了取决于分辨率的大小，还取决于颜色值的细化程度，每一种颜色也是由深到浅连续变化的，对同一种颜色，使用不同的数描述不同深浅程度的颜色值，就是对颜色进行量化，对颜色由深到浅的变化程度分的越细，图片的输出质量也越高。

将颜色数字化的方法有很多，其中最常用的是**真彩色**（true color）。美术学观点所有的颜色都可以由红、绿、蓝这三种颜色按照一定比例调和而成，红、绿、蓝这三种颜色也称为**三基色**（three primary colours），将每一种基色按照深浅程度划分为 256 各等级，分别使用数字 0-255 表示，比如 0 级红色表示最浅的红色（实际上就是白色），255 级红色表示最深的红色（鲜红），使用三元组（r,g,b）则可以表示任意颜色，其中 r 表示红色的颜色值，g 表示绿色的颜色值，b 表示蓝色的颜色值，比如颜色值（255,255,255）表示由最深的红色加上最深的绿色再加上最深的蓝色调和在一起的颜色，实际上它还是白色。三元组中每个分量 256 个值，所以使用二进制编码需要 8 位（2^8=256），那么表示三个分量一

共需要24位,也就是说每个像素对应的颜色值需要使用24位位模式表示,如果图片分辨率是800×600,那么存储所有的像素的颜色值需要 24×800×600=11520000 个比特位，折合 1440000 个字节，每个汉字占 2 字节，折合 720000 个汉字，相当于含标点符号约 70 万字的文章，可见比起存储纯文字的文本文件，存储图片的位图（后缀名“.bmp”）文件占据的容量要大得多。

不过当前计算机处理图片时都会进行压缩，目前常见的视频压缩格式有 jpeg 和 jpg 等。

2.2.2.4 视频的存储

视频（video）是图像在时间上的表示，一部视频就是一系列的图像一张接一张地播放而形成的运动画面，其中每张图像称为该视频的一个**帧**（frame）。

因此对视频的存储是建立在图像存储的基础之上的。由于一部视频包含成千上万的帧，因此保存视频的信息量相当巨大，通常将视频转换成二进制格式之后都要进行压缩来节省空间，目前常见的视频压缩格式有 mpeg4、rmvb、mkv 、avi 等。

2.3 结束语

位置化数字系统用一组有顺序的数字来表示数，每个数字所表示的大小，既取决于它本身的值，又取决于其所在的位置。十进制数转换成 b 进制数时，整数部分与小数部分要分别转换，整数部分采取“除以 b 取余”的方法，小数部分采取“乘以 b 取整”的方法；二进制与八进制以及十六进制之间的互相转换不需要通过十进制作为参考，可以直接转换。

所有类型的数据存储在计算机中时必需先要转换成位模式（0 和 1 组成的数字串），整数存储到计算机中采用的是二进制补码编码；实数（小数）存储到计算机中采用的是单精度或双精度的格式编码；计算机存储文本是以字符作为最小存储单位,ASCII 码表是最早形成国际标准的计算机可处理的字符集,随着计算机技术的发展，能够存入计算机中保存和处理的字符集越来越大，它们都兼容 ASCII 码表，也都被纳入 UNICODE 的囊中，UNICODE 几乎包括全世界人类都在使用的所有字符；音频存储到计算机中要经历采样、量化、编码三个步骤；图片存储到计算机中主要有位图和矢量图两类，大多数图片适合存储位位图格式；视频可以看做一系列的图像一张接一张地播放而形成的运动画面，所以视频存储是建立在位图存储的基础之上。

参考文献:

[1](美)Howard Eves(著). 欧阳绛(译).数学史概论(第六版)[M].哈尔滨:哈尔滨工业大学出版社,2008.

[2](美)Behrouz Forouzan,Firouz Mosharraf(著). 刘艺,瞿高峰(译).计算机科学导论(第三版)[M].北京:机械工业出版社,2010.

[3](美)Randal E Bryant,David O Hallaron(著). 龚奕利,雷迎春(译).深入了解计算机系统 [M].北京:中国电力出版社,2004.

[4]鲁宏伟,汪厚祥.多媒体计算机技术[M].北京:电子工业出版社,2011.

[5]郭艳华,马海燕.计算机与计算思维导论[M].北京:电子工业出版社,2014.

习题

1.将下列各数转换成十进制数。

（1）$(101011.011)_2$ （2）$(-111111.111)_2$ （3）$(21.11)_8$ （4）$(-617.7)_8$ （5）$(ABC.D)_{16}$ （6）$(-DE.F)_{16}$

2.将下列各十进制数分别转换为二进制数、八进制数、十六进制数（结果不能够用有限小数表示时，二进制数保留小数点后 10 位，八进制和十六进制数保留小数点后 4 位）。

（1）123.45 （2）71.875 （3）-114.06 （4）-126.406

3.将下列各二进制数分别转换为八进制数和十六进制数

（1）$(101011.01101)_2$ （2）$(111111.110110)_2$ （3）$(1111100.000001)_2$

4.将八进制数$(21.11)_8$与$(617.7)_8$转换为十六进制数（不能通过十进制数作为中转）

5.将十六进制数$(ABC.D)_{16}$与$(DE.F)_{16}$转换为八进制数（不能通过十进制数作为中转）

6.将下列各整数分别使用 8 位符号加绝对值和 8 位二进制补码两种编码方法表示出来

（1）123 （2）71 （3）-114 （4）-126

7.已知 A 为负整数，将|A|使用 n 位二进制（不足 n 位时左边补 0 凑足 n 位）表示为位模式“$a_{n-1}a_{n-2}\cdots a_0$”，然后将“$a_{n-1}a_{n-2}\cdots a_0$”各位上数字取反（0 变 1，1 变 0）得到位模式“$b_{n-1}b_{n-2}\cdots b_0$”，再将“$b_{n-1}b_{n-2}\cdots b_0$”加 1 得到位模式“$c_{n-1}c_{n-2}\cdots c_0$”，证明：

（1）位模式“$c_{n-1}c_{n-2}\cdots c_0$”即为 A 的 n 位二进制补码；

（2）将位模式“$c_{n-1}c_{n-2}\cdots c_0$”各位上数字取反再加 1 之后得到的结果恰好是位模式“$a_{n-1}a_{n-2}\cdots a_0$”。

8.写出下面各数使用单精度表示法编码的过程

（1） 7.1815 （2）11.40625 （3）-0.375 （4）-12.625

9.实数 A 使用单精度表示法编码的形式为 0 10001000 00110001100000000000000，实数 B 使用单精度表示法编码的形式为 0 10000110 01000011110000000000000，

（1）求 A 与 B 之和的单精度表示形式

（2）求 A 与-B 之和的单精度表示形式

10.汉字‘思’的区位码是 4328，然后求出‘思’字的国标码和机内码，并给出详细求解过程。想一想如何求出词语“计算思维”存储在计算机中的编码信息。

11.可否考虑汉语拼音组成的汉字输入码转换成的 ASCII 码作为汉字存储在计算机中的编码？你认为这样的设计方案好还是不好？请说明理由。

12.GB2312 中所有的汉字都要转换成机内码才能存储到计算机中吗？也就是说是否存在某个汉字的国标码因为其 2 个字节的最高位都是 1，而不需要转换成机内码就直接存储到计算机中的？有的话请找出来，没有的话请分析其原因。