第 5 章　数据表示方法

计算机科学要在计算机系统上实现处理数据任务，面临三方面研究课题。用什么方法来表示数据？用什么方法来表示对数据进行加工的过程？数据和加工数据的表示如何最终在机器上实现？

前面三章介绍了计算机系统的组成，这是数据和数据加工表示方法的最终载体。第 2 章谈及计算机硬件，第 3 章谈及计算机软件，第 4 章谈及计算机网络。本章要介绍计算机科学如何使用分层次的方法来解决数据表示问题。至于数据加工的表示方法，在第 6 章再讲。

5.1　数据的分层表示方法学

计算机科学用“数据”作为手段来表示客观世界里要处理的对象。而在计算机的内部，能够物理实现的数据记号只有两个二进制数字“0”和“1”。因此，数据表示面临的任务是用最简单的记号表示出内容复杂而形式多变的对象。计算机科学的解决方法是：

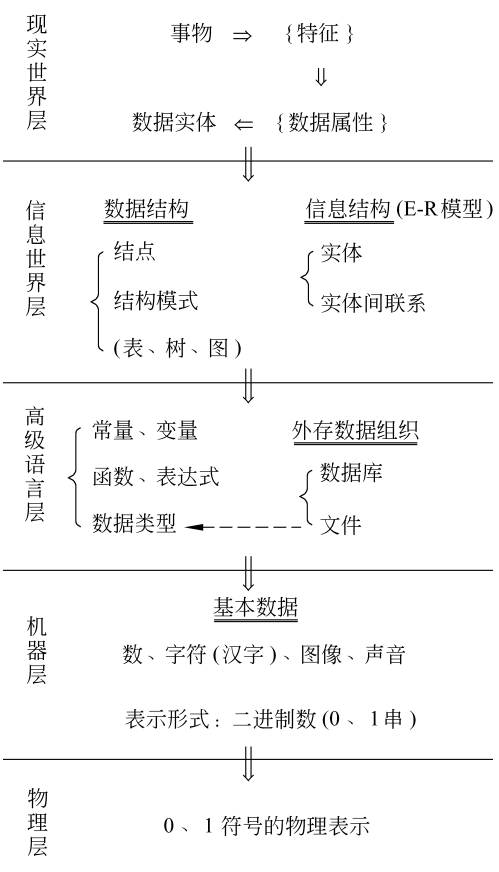

图 5-1　数据的分层表示方法

（1）划分出不同的数据表示抽象层次；

（2）每个层次上都各自定义数据的表示概念和手段；

（3）这些表示手段既相对独立，又可以从上一个表示层次映射到下一个层次上去；

（4）从现实世界到计算机内部物理实现，数据表示的抽象程度逐层降低，直到“0”、“1”记号能够在硬件的层次上实现为止。

这样，在完成数据表达任务过程中，人们可以根据需要选择适当的表达层次。而层次之间不同表示的转换，由人或计算机系统本身按照明确定义的映射规则来完成。

可以把不同时期计算机科学提出的数据表示方法总结为上述分层次数据表示方法学。图 5-1 描述了这种表示层次。当然，它们并不是由个别人预先提出来的一个表示体系，而是经历一段历史后，我们对计算机科学完成数据表示任务的方法和体系的理解结果。

5.1.1　现实世界层

正如本书一开始所指出的，计算机科学把

现实世界里的客观事物等同于事物的一组特征。这就是说，不管事物是有形的还是无形的，总是用**事物特征**的集合来表示事物本身。图 5-2 中的例子描述了数据表示方法的出发点。用合同的一组特征等同表示现实世界里的一纸合同。这是数据表示体系的出发点。

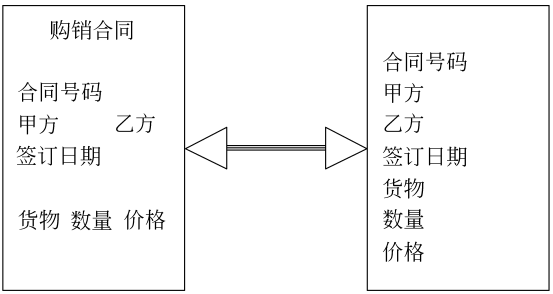

图 5-2　把“合同”等同于“合同的一组特征”

5.1.2　信息世界层

两个原因促使人们要在另一个抽象层次上考虑数据表示问题。首先，一种事物的特征几乎有无限多个，人们必须按照数据处理任务的需要选取出恰当的特征来代表事物；其次，一个数据处理任务不可能只面对一种事物，所以有必要刻画各个数据对象之间存在的关联，以便组成一个统一的数据表示构造，不妨称之为**信息结构**。

在提出的众多信息结构中，有两种被人们广泛接受，应用至今，它们是**实体-联系模型**和**数据结构**。

1. E-R 模型

正如名称所提示的那样，实体-联系模型由两个要素组成：**实体**(entity)以及实体之间的**联系**(relationship)，因此又常称为 **E-R 模型**。

实体是一组**数据属性**(attribute)的集合，是客观事物的一种表示手段；而联系则从实体的对应规律角度出发，刻画实体之间的关联状况。E-R 模型成功地把千千万万种事物之间存在的关联模式归结为五六种不同的联系类型。

注意，数据属性对应着事物特征，数据实体对应着客观事物。但是，它们代表不同层次上的对象，如何对应取决于专业人士对有关事物的深刻了解。

以“合同”为例，表达的 E-R 模型可以有两个不同方案。一个方案是用一个实体来表示合同，如图 5-2 所示。实体包含合同号码、甲方、乙方、签订日期、货物、数量和价格等数据属性。另外一个表示方案则由两个存在相互关联的实体组成，如图 5-3 所示。

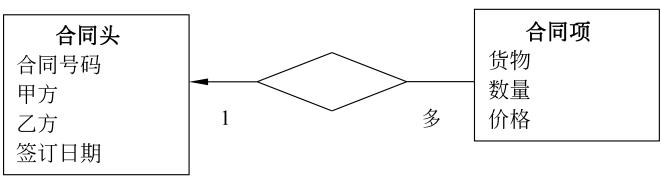

图 5-3　表示“合同”的一种 E-R 模型

在第二个方案里，现实世界里的“合同”在 E-R 模型中用“合同头”和“合同项”两个数据

实体表示。这样可以更准确地刻画出合同内各个数据之间的关联关系，即一份合同里面可以包含数目不确定的合同项。一份合同可能只涉及一种货物，而另一份会买卖十种八种。对合同数据的这种准确描述，就为以后的工作奠定了基础，在设计在计算机内存放合同数据的存储空间结构时，能够得到更加合理的设计方案。

直到今天，E-R 模型仍然是对存储数据进行前期分析时可以采用的工具，简单而有效。

2. 数据结构

数据结构(data structure)是另外一种应用很广泛的信息结构，对它的学习极为重要。数据结构用**结点**(node)这个抽象概念来表示数据构造的基本元素，用结点之间的联系模式来描写数据构造的内部关联。结点是事物的抽象，联系模式是事物之间关联方式的抽象。

例如，要描述公司日积月累、数以千计的合同，可以把每一份合同抽象为一个数据结点，如果只需要按签订日期的先后来保存这些合同，所有合同之间就存在一种顺序关联的关系。不难想象，具有同样顺序关联的事物数不胜数。因此，归纳出一种称为线性表的抽象结构，就可以表示一大类内容虽然各异，但关联构造却完全相同的数据对象。

更重要的是，人们发现只要定义三种抽象数据结构，就足以描写现实世界中的任何事物对象，它们是**线性表**(list)、**树**(tree)和**图**(graph)。因此，只要把握三种数据结构的特征、性质、实现方法和相关的典型算法，就可以应用到任何一个需要表达数据的场合中。“数据结构”一直是计算学科类专业的核心基础课程，原因就在这里。

本章的 5.6 节会更详细地讲述数据结构的主要概念。

5.1.3 高级语言层

数据结构也好，E-R 模型也好，说它们是“抽象”的，是指表达数据的概念和方法仍然没有进入到计算机系统的范畴。如果在程序设计时，能够利用程序设计语言提供的数据表达手段来表示这些抽象的信息结构，那么我们向数据表达的终极目标又前进了一步。

今天，高级语言仍然是程序设计的主要工具。尽管可以选用的高级语言很多，语言机构各有差别。但是可以归纳出高级语言表示数据对象的 5 个共同手段：**常量**、**变量**、**表达式**、**函数**和**数据类型**。

高级语言中，常量和变量是数据表示的基本概念；表达式和函数都表示了经过特定操作过程而得到的一个结果数据；而数据类型刻画出不同种类数据的基本特征、一类数据合法的取值范围以及合法的数据操作种类。

5.5 节会更详细地讲述高级语言的数据表示的 5 大手段。

5.1.4 机器层

高级语言的数据表示手段已经进入计算机系统的范围了。经过编译，高级语言程序可以转变为对应的机器语言程序。转变对象包括程序里的操作语句，也包括程序里定义的数据。

经过转换在机器里出现的所有数据，不管具有什么样的信息语义，都有两个共同特点：数据要存储在一组顺序组织的、可寻地址的存储单元当中；当然，任何数据的表现形式最终只能是二进制数。

那么，如何来确定一个二进制数形式数据的信息含义呢？靠各种各样的**数据编码规则**。

又是谁来把握这些编码规则呢？处理数据的程序。当然，其实是编写程序的人。编写程序时，按照预先约定的数据表示规则，对一个二进制数进行操作，就是按正确的语义处理一个数据。

机器层次上要考虑几类基本数据的编码表示方法：**数**、**字符**和**汉字**、**图像**、**声音**。本章的 5.2 节～5.4 节分别讲述它们的编码规则。

5.1.5 物理层

硬件是数据表示的最终层次。在这个层次上，数据表示的目标集中而单纯，就是如何在计算机的各种物理元件上表示二进制数字“0”和“1”。共同原则是元件必须具有两个稳定的物理状态，并且两种状态可以按照处理需要相互转换。这样，硬件材料的物理状态就可以和二进制数字挂钩了。

比如，电子电路的高电平、低电平；电磁材料的两个不同磁化方向；电容的充电、放电；光的通、断或强、弱；半导体材料“微穴”中电子的“满”、“空”等，都是计算机各类硬件里常见的物理状态。

至此，只用硬件的两种物理状态表示出世界万物的艰难任务，依靠划分数据表达层次的方法学得以顺利完成。以下各节将对各个层次上数据的表达方法进一步展开。至于更详细的学习，就要依靠以后开设的相关专业课程了。

习　题

1. 列举数据表示的层次，并解释为什么要分层次表示数据？

2. 挑选现实世界的三种事物加以描述。

3. 一个系有多个学生，每个学生只能在一个系注册；一个学生可以选修多门课程，而每门课程可以有许多个学生选修。用一个 E-R 模型表示“系”、“学生”、“课程”的数据联系。

4. 在磁盘和光盘上如何表示二进制数字“0”和“1”？

5. 各种数据在计算机内的共同表示特点是什么？

6. 如何确定计算机内一个二进制数究竟是代表一个数，还是代表一个汉字？谁负责确定？

5.2 数的表示

数(numbe)是数据的一种表示形式，计算机内需要以二进制数为手段来表示整数、实数和复数等各类数的数值，以方便对数进行算术运算。本节讨论的实质是**数值**的表示，而不是表现为**数字**(digit)**串**的数的记录形式。因为以数字为单位存储一个数，不但需要更多空间，而且不利于对数进行算术运算。

5.2.1 无符号整数的表示

一个无符号整数自然必须以二进制数的形式在计算机内出现。第 1 章的 1.3 节里已经讲过把一个日常使用的无符号整数转换为二进制形式的方法。要强调的是，注意数的**字长**。

当然可以说数 5 的二进制形式是 101。但一个数所驻留的内存单元、寄存器、运算器总是有固定字长的。存储空间里每个位上不是 0 就是 1，不可能是不确定的其他状态。

例如，计算机字长为 16 位，5 这个无符号整数在内存单元里的存储形式是：

0000000000000101

注意，在有效数字 101 的前面，有 13 个对数值计算无效但确实存在的“左零”。

因为存放在计算机内的数必有预先确定的字长，所以数的表示范围总是有限的。如果规定用 16 位来存放一个无符号整数，计算机内只能表示出 0～65 535，一共 65 536 个整数。这一点和数学的概念不一样，数学上一种数的所有可能值总是构成一个无穷集合。

5.2.2 有符号整数的表示

那么，如何表示有符号整数所带的正负号呢？能够使用的记号仍然只有 0 和 1。于是就产生一个问题，如何区分表示数绝对值的 0、1 和表示正负号的 0、1 呢？区分的唯一手段是数的**编码规则**。必须按照预先设计的一种规则来表示数，对数进行运算，理解运算的结果。

在编码表示方法里有两个极其重要的概念：**机器数**和**真值**。机器数是指被表示的数在机内呈现的表面形式。真值则是机器数的信息含义。两者依靠预先确定的编码规则互相对应。

有符号整数的编码规则有多种。先以**原码规则**为切入点。原码容易理解，但缺点很多，并不实用。然后介绍广为应用的**补码规则**。

1. 原码

原码规则规定：在数的绝对值以外再增加一个**符号位**，用 0 表示正号、用 1 表示负号。这样，额外符号位上的 0、1 就和数的正、负值对应了。

【例 5-1】 假设字长为 4 位，写出＋5 和－5 的原码表示式。

字长 4 位，必须留出一个符号位，其余 3 位用来表示数的绝对值。按原码规则：

＋5 的原码是$\underline{0}$101　　－5 的原码是$\underline{1}$101

不难理解，4 位原码能够表示的最大整数是＋7(0111)，最小整数是－7(1111)，一共 15 个数。0 的原码有两个：0000 和 1000。这是原码规则的毛病之一。

重要的是，通过对原码的认识，要区分**机器数**和**真值**这两个不同的概念，如图 5-4 所示。机器里的一个二进制数 1101，只有引用原码规则才能确定它代表的真值是－5。如果引用的是无符号整数编码规则，机器数 1101 的真值是 13。

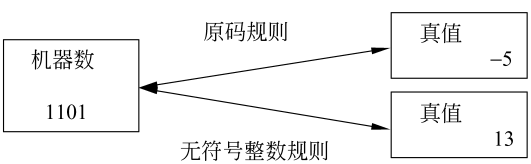

图 5-4　机器数和真值的对应

可见，机器数的形式是单一的，离开了编码规则，一个机内二进制数所代表的数据真值无从谈起。至于规则的引用，则由处理数据的程序负责。当然，实质上是由程序的设计者来负责的。

和第 1 章里讲过的基本概念对照，机器数其实就是一种数据，而真值则是对应的信息，编码规则就是信息的理解规则。

2. 补码

和原码相比，补码的优点很多。所以机内表示有符号整数时，大多数情况下会采用**补码**(complement notation)的形式。想透彻理解补码规则，要认识以下 5 个概念：

1) 模

一个预先确定的无符号整数 m。模的值可以随意规定。但是用二进制形式表示数时，如果字长规定为 n 位，那么补码系统的模就是 2^n。

2) 模 m 运算系统

模为 m 的运算系统有两个基本特点：

(1) 系统处理的是无符号的整数。

(2) 操作数和运算结果都必须在$0\sim m-1$之间取值。如果运算结果等于或者大于 m，则必须用除以 m 的余数取代。

模 m 运算系统的特点如图 5-5 所示。

操作数：$0\sim m-1$

运算结果：$N=\begin{cases} N \bmod m & (N\geqslant m) \\ 0\sim m-1 & (N<m) \end{cases}$

图 5-5　模 m 运算系统的特点

可见，在一个模为 m 的运算系统中，只能有 m 个数存在。不管是运算对象还是运算结果，它们的数值分布在 $0\sim m-1$ 的范围内。

模运算系统并不抽象，日常生活中也能找到。“时钟”就是一个 $m=12$ 的模运算系统。钟只有 12 个数值，即 0～11。把时针顺时针走动当作加法，逆时针走动当作减法。显然做加、减法的时候，和与差都只能在 0～11 范围内的 12 个数中取值。

在这个系统里，$3+9=0$；而 $3-4=11$。(请思考其原因。)

3) 补数

在模 m 系统中，如果 $a+b=m$，则 a 为 b 的补数，b 为 a 的补数。

因此，在模 12 系统里，3 与 9、1 和 11 互为补数，其余类推。注意，这里讲的是**补数**，而不是补码。

数与它的补数的和等于模，如果规定数的表示是定长的，这个和呈现为 0。想想“时钟”这个模 12 系统，3 和 9 互为补数，它们的和为 12，不就是零点吗？

4) 求补

所谓“求补”，就是指求出一个数的补数(不是补码)的操作过程。

模 m 系统中，如 a、b 互为补数，则 $a+b=m$。因此可以用减法求出一个数的补数。即：

$$b=m-a,\quad a=m-b$$

使用二进制表示的情况下，可以用更简单的操作取代减法。即模减去一个数得到的差等同于对这个数**“求反加 1”**。即把二进制数每一位，0 变 1、1 变 0 地“取反”，然后再在结果的最右位加上 1，就可以得到原来二进制数的补数。

【例 5-2】 在字长为 8 位的模运算系统中，求二进制数 01101010 的补数。

这个系统的模 $m=2^8=256$。

二进制数	01101010
对二进制数“求反”	10010101
结果再加 1	10010110

验证：两个互补的数，其和等于模。如果数的长度定为 8 位，则最左的 1 丢失，结果为全 0。

$$01101010+10010110=100000000=2^8$$

5）补码

在上述 4 个概念的基础上，现在可以给出补码的定义规则了，即在一个模 m 的系统中，正数的补码是自身的绝对值，负数的补码是绝对值的补数。这样，就可以用无符号的机器数来表示有符号的整数真值了。

【例 5-3】 一个模 12 的补码表示系统。

系统 $m=12$，共有 12 个无符号整数，记为 0～11。用其中 6 个表示 0～+5 这 6 个正数，其余 6 个表示－1～－6 这 6 个负数。各个补码和真值的对应关系如图 5-6 所示。

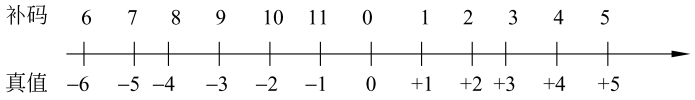

图 5-6 模 12 的补码系统

按补码编码规则，+1～+5 这 5 个正数的补码是各数的绝对值 1～5。－1 的绝对值是 1，1 的补数是 11，所以 11 是－1 的补码。其他负数的补码表示可以依此类推。

【例 5-4】 字长为 4 位的二进制数补码系统（如图 5-7 所示）。

系统的模是 $m=2^4=16$，一个有 16 个补码，表示－8～+7 共 16 个有符号整数。

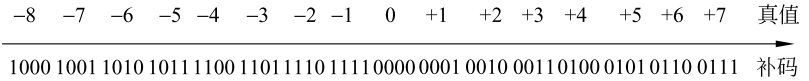

图 5-7 字长 4 位、模 16 的补码系统

（1）补码的优点。

计算机运算器的字长 n 是固定的，所以是个模为 2^n 的模运算系统。用补码表示正负数有许多优点，因而被广泛使用。

在使用补码表示的前提下，最大的两个好处是：

① 对任意的正、负整数，可以不加区分地进行机械式的加法运算，结果总是对的；

② 可以用加法替代减法，减去一个数等同于加它的补数。

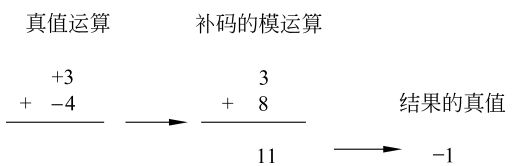

图 5-8 模 12 系统中的加法运算示例

【例 5-5】 在例 5-3 的模 12 补码系统中，验证补码运算的特点。

如图 5-8 所示，不难验证在补码体系里只要按模运算的规则进行加法运算，得到的和总是符合有符号整数的真值运算意义的。如果用原码来表示有符号整数，就没有这个好处了，请读者自行验证。

但是再看看 3＋4＝7，3 和 4 分别是＋3 和＋4 的补码，而 7 是－5 的补码，结果错了。原因在哪里呢？

【例 5-6】 在例 5-4 给出的字长为 4 位的二进制补码系统中，验证补码运算的特点。

用补码表示正、负数，除了可以机械地、不加判断地执行加法运算之外，还可以用加法来替代减法(如图 5-9 所示)。当然，这一点没有那么重要，毕竟多数 CPU 都会提供减法指令。通过这个例子，请读者更清晰地区分“补数”和“补码”这两个不同的概念。

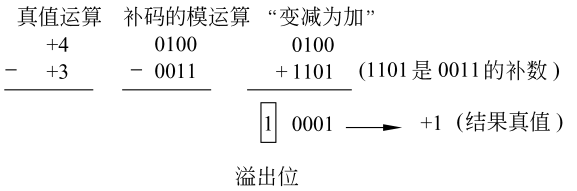

图 5-9　模 2^4 的补码系统的减法示例

对于其他的运算数，上述运算方法当然也是对的。例如：

$$(-7)-(-6)=-1$$

即

$$1001-1010=1001+0110=1111(-1\text{ 的补码})$$

再看一个出错的例子：

$$(-3)-(+6)=-9$$

$$1101-0110=1101+1010=0111\quad(+7\text{ 的补码，结果错了})$$

出错的原因是什么？

(2) 补码和真值的转换。

前面已讲过，所谓的“求补”是一种操作，把一个数的补数求出来。对于二进制数，可以用“求反加 1”的动作完成“求补(数)”。

使用求补操作，既可以把一个负数的补码写出来，也容易确定负数补码的真值。

【例 5-7】 求+3 和-3 在字长 4 位、模 2^4 的二进制数补码系统中的补码形式。

根据前述补码编码规则：

+3 的绝对值是 3，所以+3 的补码是 0011；

-3 的绝对值是 3，　0011 的补数是 1101，所以-3 的补码是 1101。

【例 5-8】 已知 0001 是正数的补码，1000、1111 是负数的补码，写出它们的真值。

显然，0001 是+1 的补码；因为 1000 的补数是 1000(0111+1)，1000 是 8，所以 1000 是-8 的补码；1111 的补数是 0001(0000+1)，0001 是 1，所以 1111 是-1 的补码。

(3) 补码的几个性质。

① 符号位。补码的最左位可以指示真值正、负。正数补码的最左位必为 0，负数补码的最左位必为 1。要注意的是，和原码里额外添加的符号位不同，补码的这个“符号位”不仅仅指示数的正、负，而且是真值里不可切割的一个组成部分。

【例 5-9】 写出补码 1011 的真值。

它不是-3。从最左位判断，1011 是个负数的补码，因此对它求补：

1011 的补数是 0101(5)，所以 1011 是-5 的补码。

② 不同补码的真值比较。比较两个同符号数的补码，补码越大对应的真值也越大。再看例 5-4 里面的补码系统，从 0000 到 0111 补码越来越大，对应的真值也从 0 增大到+7。在负数的范围里，补码从 1000 渐增到 1111，对应的真值也从-8 增大到-1。

③ 补码系统的表示范围。字长为 n 位的补码系统，模为 2^n，共有 2^n 个码，表示有符号整数的真值范围为：

$$-2^{n-1}\sim+2^{n-1}-1$$

因此，例 5-3 里的模 12 系统，只能表示 12 个数，从－6 到＋5；而例 5-4 里的模 2^4 系统，只能表示 16 个数，从－8 到＋7。

【例 5-10】 用 16 位的补码形式来表示有符号整数，指出数的表达范围。

因为 16 位补码系统的模为 $2^{16}=65\,536$，所以总共可以表示 65 536 个正、负数。

- 能够表示的最小负数是 $-2^{15}=-32\,768$；
- 能够表示的最大正数是 $+2^{15}-1=+32\,767$。

想想为什么能够表示的最大正数的绝对值会比最小负数的绝对值小 1？

5.2.3 实数的表示

实数包括有符号的整数和非整数。数学上的实数是连续的，而且组成了一个无限集合。计算机内的实数则不然，离散而且数量有限。实数的机内表示形式分**定点数**和**浮点数**两种。表示实数时，两种形式常常混合着使用。

1. 定点形式

在一个二进制定点数里，人为认定在某两位数字之间存在一个小数点。小数点左边是整数部分，右边是小数部分。整数位上的权值自右到左是 2^0、2^1、…、2^{n-1}。小数位上的权值自左到右是 2^{-1}、2^{-2}、…、2^{-m}。

【例 5-11】 二进制定点数 $10_\wedge 111$ 的值。

$$\begin{aligned}10_\wedge 111 &=1\times2^1+0\times2^0+1\times2^{-1}+1\times2^{-2}+1\times2^{-3}\\&=2+0.5+0.25+0.125=2.875\end{aligned}$$

表面上看定点方式挺方便，小数点只是认定的，不需要有具体表现形式，也不占用空间。但是，实际操作时却面临很多麻烦，每个实数小数点前后的位数都可能不一样，如何“记住”每个数的小数点位置呢？

所以定点方式通常只会在实数表示中起辅助作用，尤其多使用以下两种形式的定点数：

(1) 把小数点认定在数字串的最右端，这种定点数实质上是个整数，即：

$$a_{n-1}\ a_{n-2}\cdots\ a_1a_0{}_\wedge$$

这里“$_\wedge$”表示人为认定的小数点在定点数里的出现位置。

(2) 小数点认定在高端第 1 位数字和第 2 位数字之间，即：

$$a_{n-1}{}_\wedge\ a_{n-2}\cdots\ a_1a_0$$

如二进制的定点数 $1_\wedge 01$ 应理解为 1.01，也就是十进制实数 1.25。

2. 浮点形式

可以把实数 12.34 用另外一种形式表示，即 $0.1234\times10^{+2}$，称之为浮点数形式。0.1234 表示了实数值的各位数字，叫做**尾数**；＋2 表示了小数点在实数里的实际位置，叫做**阶码**。

二进制的情况也一样。一个二进制数－0.00101 可以表示为 -0.101×2^{-2}，－0.101 是尾数，－2 是阶码。

一般来说，一个二进制实数 N 总可以表示为：

$$N=R\times 2^{k}$$

因此，只要记录尾数和阶码就可以表示一个实数。(尾数 R，阶码 k)就是实数的浮点表示式。阶码的底 2 只需约定，不必出现在浮点数表示当中。当然，运算时不能忽略它。

尾数和阶码都用定点数形式来表示。阶码总是整数，而尾数的习惯表示是把小数点定在左起第一位和第二位之间，见下例。

【例 5-12】 约定每一个浮点数占 8 位，左端 3 个位表示阶码，右端 5 个位表示尾数。给出二进制实数 10.11 的浮点形式。

因为

$$10.11=+0.1011\times 2^{+2}$$

即尾数是+0.1011，阶码是+2，所以它的浮点数形式是01001011。

这里，尾数和阶码都是用补码表示的定点数。左 3 位阶码的隐含小数点位置在最右端，右 5 位尾数的隐含小数点位置在左起第一位数字和第二位数字之间。

3. 规格化的浮点数

实数的浮点表示并不唯一。12.34 可以写成 $0.1234\times 10^{+2}$，可以写成 123.4×10^{-1}，喜欢的话，也可以写成 $0.000\,012\,34\times 10^{+6}$。尽管数值并不因为形式的不同而变化，但是尾数的位数是固定的，不恰当的浮点形式会导致有效数字的丢失。解决的办法就是规格化，即做出各种规定，使浮点数的表示标准化。

可以规定不同的规格化标准，下面是可行的一种，其要点是：

(1) 浮点数的字长为 $m+n$ 位，左起 m 位是阶码，右起 n 位是尾数；

(2) 阶码用模为 2^m 的补码表示，是有符号的整数；

(3) 尾数用定点形式的补码来表示，隐含的小数点位于左起第一位和第二位之间，因此尾数是模为 2 的补码系统；

(4) 尾数头两位数字必须相反，即正尾数头两位是 $0_{\wedge}1$，负尾数头两位是 $1_{\wedge}0$。

符合上述要点的规格化浮点数，形如图 5-10 所示。

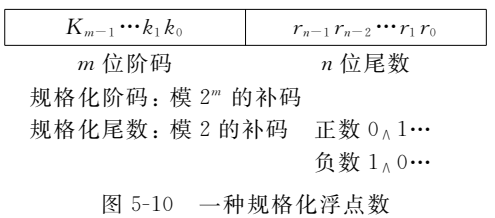

图 5-10 一种规格化浮点数

【例 5-13】 浮点数字长 8 位，左起 3 位是阶码，右起 5 位是尾数。按上述规定，给出+1/2、+1/4、−1、−1/2 的规格化形式。

+1/2	00001000	即	$+1/2\times 2^{0}=+1/2$
+1/4	11101000		$+1/2\times 2^{-1}=+1/4$
−1	00010000		$-1\times 2^{0}=-1$
−1/2	11110000		$-1\times 2^{-1}=-1/2$

注意，+1/4 的规格化浮点数不是 00000100。虽然尾数 $0_{\wedge}0100$ 确是+1/4 的模 2 补码形式，但尾数头两位数字都是 0，不符合上述规格化约定。

4. 实数表示的截断误差

计算机内表示的实数和数学上的实数的差别，除了不连续之外，截断误差也是一个不能忽略的问题。首先，一些实数本身就没有准确的二进制形式。比如 0.3 的二进制数形式是个无限循环小数，即 0.0 10011001…，在机内出现必然会引起截断误差。其次，即使实数有准确的二进制形式，但预定存储实数的字长是固定的，尾数字段空间不够大时，部分数字位