第 3 章 数据表示、运算算法和线路实现

运算器最重要的功能是加工数据，为此必须很好地掌握各种基本类型的数据及其在计算机内的表示和存储方式，以及完成运算所用的算法和实现这些算法所用的原理性电路。这些内容又以数字化信息编码、布尔代数、数字电路与逻辑设计为基础。布尔代数、数字电路与数字逻辑设计在本教材第 2 章进行了简要介绍。因此，本章将从数字化信息编码讲起，引出二进制编码，数制转换，插入部分检错纠错码知识；接下来介绍各种基本类型数据的表示；数值数据算术运算的有关方法。

二进制编码，数制转换，定点小数和整数的原码、反码、补码表示是本章的核心重点内容。

检错纠错码概念，讲到的两种常用的检错、纠错码的实现原理应较好理解，而对它们所用的具体线路简单了解即可。

应该掌握定点小数、整数、浮点数在计算机内的表示，补码加减法的运算规则，原码一位乘除法的原理性算法和完成算术运算用到的原理性逻辑线路。补码乘除法、加速乘除运算的可行方案简单了解即可。

本章作业较多，对深入理解所学知识和比较熟练地完成必要的计算很有必要，也是更好地学习运算器知识的基础，应认真完成。

3.1 数字化信息编码的概念和二进制编码知识

3.1.1 数字化信息编码的概念

计算机最重要的功能是处理信息，如数值、文字、符号、语音、图形和图像等。在计算机内部，各种信息都必须采用数字化的形式被保存、加工与传送。掌握信息编码的概念和技术是至关重要的。

所谓编码，就是用**少量、简单的基本符号**，选用**一定的组合规则**，以表示大量复杂多样的信息。基本符号的种类和这些符号的组合规则构成编码的两大要素。例如，用 10 个阿拉伯数字表示数值，用 26 个英文字母构成英文词汇，就是现实生活中编码的典型例子。

当要使用的基本符号数量较多时，往往还要采取措施，以便首先使用更少量的简单符号来编码以表示那些量大而复杂的基本符号，再用这些基本符号来表示信息，这就构成了多重编码。多重编码的典型例子是汉字编码，若把上万个汉字都作为基本符号就太多了，可以首

先用笔形字画、偏旁部首、拼音或其他方式对其进行编码,以解决汉字输入问题。

在计算机中,广泛采用的是仅用0和1两个基本符号组成的基二码,亦称为二进制码。主要有以下3个方面的原因。

(1) 基二码在物理上最容易实现,即容易找到具有两个稳定状态且能方便地控制状态转换的物理器件;可以用两个状态分别表示基本符号0和1。

(2) 用基二码表示的二进制数,其编码、计数和算术运算规则简单,容易用数字电路实现,为提高计算机的运算速度和降低实现成本奠定了基础。

(3) 基二码的两个基本符号0和1能方便地与逻辑命题的"否"和"是",或称"假"和"真"相对应,为计算机中的逻辑运算和程序中的逻辑判断提供了便利条件。

计算机中的各种类型的数据,通常都是用二进制编码形式来表示、存储、处理和传送的。

3.1.2 二进制编码和码制转换

1. 数制与进位计数法

在采用进位计数的数字系统中,如果只用 r 个基本符号(例如0,1,2,…,$r-1$),通过排列起来的符号串表示数值,则称其为基 r 数制(Radix-r Number System),r 被称为该数制的基(Radix)。假定用 $m+k$ 个自左向右排列的符号 $D_i(-k\leqslant i\leqslant m-1)$ 表示数值 N,即

$$N = D_{m-1}D_{m-2}\cdots D_1D_0D_{-1}D_{-2}\cdots D_{-k} \tag{3.1}$$

式中的 $D_i(-k\leqslant i\leqslant m-1)$ 为该数制采用的基本符号,可取值0,1,2,…,$r-1$,小数点位置隐含在 D_0 与 D_{-1} 位之间,则 $D_{m-1}\cdots D_0$ 为 N 的整数部分,$D_{-1}\cdots D_{-k}$ 为 N 的小数部分。

如果每个 D_i 的单位值都赋以固定的值 W_i,则称 W_i 为该位的权(Weight),此时的数制称为有权的基 r 数制(Weighted Radix-r Number System)。此时 N 代表的实际值可表示为

$$N = \sum_{i=-k}^{m-1} D_i \times W_i \tag{3.2}$$

如果该数制编码还符合"逢 r 进位"的规则,则每位的权(简称位权)可表示为

$$W_i = r^i$$

式中的 r 是数制的基,i 是位序号。式3.2又可以写为

$$N = \sum_{i=-k}^{m-1} D_i \times r^i \tag{3.3}$$

式中的符号:

r——是这个数制的基;

i——表示这些符号的排列次序,即位序号;

D_i——是位序号为 i 的一位上的符号;

r^i——是第 i 位上的一个1所代表的值(位权);

$D_i\times r^i$——是第 i 位上的符号所代表的实际值;

$\sum$—— 表示对 $m+k$ 位的各位的实际值执行累加求和;

N——代表一个数值。

此时该数制被称为 r 进位数制(Positional Radix-r Number System),简称 r 进制。下面是计算机中常用的几种进位数制:

二进制 $r=2$,基本符号 0,1

八进制 $r=8$，基本符号　0,1,2,3,4,5,6,7

十六进制 $r=16$，基本符号　0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

其中 A～F 分别表示十进制数 10,11,12,13,14,15

十进制 $r=10$，基本符号　0,1,2,3,4,5,6,7,8,9

如果每一数位都具有相同的基，即采用同样的基本符号集来表示，则称该数制为固定基数制（Fixed Radix Number System），这是计算机内普遍采用的方案。在个别应用中，也允许对不同的数位或位段选用不同的基，即混合采用不同的基本符号集来表示，则该数制被称为混合基数制（Mixed Radix Number System）。

2. 二进制编码和二进制数据

二进制编码是计算机内使用最多的码制。它只使用两个基本符号 0 和 1，并且通过由这两个符号组成的符号串来表示各种信息（各种类型的数据）。二进制的数值类型的数据亦是如此，计算其所代表的数值的运算规则是

$$N=\sum_{i=-k}^{m-1}D_i\times 2^i \qquad D_i \text{ 的取值为 0 或 1} \tag{3.4}$$

例如　$(1101.0101)_2=(13.3125)_{10}$。

等号左右两边括号内的数字为两个不同进制的数字，括号右下角的 2 和 10 分别指明左右两边的数字为二进制和十进制的数。按式(3.4)，计算二进制数 1101.0101 的实际值为

$$1\times 2^3+1\times 2^2+0\times 2^1+1\times 2^0+0\times 2^{-1}+1\times 2^{-2}+0\times 2^{-3}+1\times 2^{-4}$$

$$=8+4+1+0.25+0.0625=13.3125$$

从式中可以进一步看到，由于二进制只用 0 和 1 两个符号，在计算二进制位串所代表的实际值时，只需把符号为 1 的那些位的位权相加即可，则上式变为：

$$2^3+2^2+2^0+2^{-2}+2^{-4}=13.3125$$

熟练地记清二进制数每位上的位权是有益的。当位序号为 0～12 时，其各位上的位权分别为 1、2、4、8、16、32、64、128、256、512、1024、2048 和 4096。

3. 数制转换

计算机中常用几种不同的进位数制，包括二进制、八进制、十进制和十六进制。二进制数据更容易用逻辑线路处理，更接近计算机硬件能直接识别和处理的数字信息的使用要求，而使用计算机的人更容易接受十进制的数据类型。二者之间的进制转换是经常遇到的问题。

1）十进制数据与其他进制数据的转换

式(3.3)确定的运算规则，是不同进位计数制数据之间完成进位制转换的依据。

十进制到二进制的转换，通常要区分数的整数部分和小数部分，并分别按除 2 取余数部分和乘 2 取整数部分两种不同的方法来完成。

（1）**对整数部分**，要用除 2 取余数办法完成十进制到二进制的转换，其规则是：

① 用 2 除十进制数的整数部分，取其余数为转换后的二进制数整数部分的低位数字；

② 再用 2 去除所得的商，取其余数为转换后的二进制数高一位的数字；

③ 重复执行第②步的操作，直到商为 0，结束转换过程。

例如，将十进制的 37 转换成二进制整数的过程如下：

```
2 | 37 -------1        低位
  2 | 18 -------0        |
    2 | 9 -------1       |
      2 | 4 -------0     |
        2 | 2 -------0   |
          2 | 1 -------1 |
            0            高位
```

余数部分，即转换后的结果为$(100101)_2$

(2) **对小数部分**，要用乘2取整数办法完成十进制到二进制的转换，其规则是：

① 用2乘十进制数的小数部分，取乘积的整数为转换后的二进制数的最高位数字；

② 再用2乘上一步乘积的小数部分，取新乘积的整数为转换后二进制小数的低一位数字；

③ 重复第②步操作，直至乘积部分为0，或已得到的小数位数满足要求，结束转换过程。

例如，将十进制的0.43，转换成二进制小数的过程如下(假设要求小数点后取5位)：

```
          0.43×2
高位  0   0.86×2
      1   0.72×2
      1   0.44×2
      0   0.88×2
低位  1   0.76
```

整数部分，即转换后的二进制小数为$(0.01101)_2$

对小数进行转换的过程中，转换后的二进制已达到要求位数，而最后一次的乘积的小数部分不为0，会使转换结果存在误差，其误差值小于求得的最低一位的位权。

对既有整数部分又有小数部分的十进制数，可以先转换其整数部分为二进制数的整数部分，再转换其小数部分为二进制的小数部分，通过把得到的两部分结果合并起来得到转换后的最终结果。例如，$(37.43)_{10}=(100101.01101)_2$。

在实现手工转换时，如果对二进制数已经比较熟悉，基本上记住了以2为底的指数值，即二进制数每一位上的权，对十进制数进行转换时，也可以不采用上述规则，基本上可以直接写出来。例如：

$(45.625)_{10}=32+8+4+1+0.5+0.125$

$=(10\ 1\quad 1\ 01.\quad 10\quad\quad 1)_2$，即$(101101.101)_2$。

$(1105)_{10}=1024+81=1024+64+16+1$

$=(10001010001)_2$，即$(10001010001)_2$。

参照上述方法，也可以实现十进制到八进制、十进制到十六进制的转换过程。例如：

(1) 完成八进制和十进制数据之间的转换，具体如下。

```
8 | 1109 ----5
  8 | 138 ----2
    8 | 17 ----1
      8 | 2 ----2
        0          结果：(1109)10=(2125)8
```

```
    0.385×8
3   0.08×8
0   0.64×8
5   0.12           结果：(0.385)10=(0.305)8
```

(2) 完成十进制到十六进制数的转换方法与前述方法类似，只是乘除 16 时，手工运算不大方便。

2) 二进制与八进制、十六进制间的转换

用二进制表示一个数值 N，所用的位数 K 为 $\log_2 N$，如表示 4096，K 为 13，写起来位串较长。为此，计算机中也常常采用八进制和十六进制来表示数值数据，为表示数值 N，分别有如下对应关系：

$$N=\sum_{i=-k}^{m-1} D_i \times 8^i \qquad D_i \text{ 的取值为 0 到 7} \tag{3.5}$$

例如 $(7.44)_8=7\times8^0+4\times8^{-1}+4\times8^{-2}=(7.5625)_{10}$。

$$N=\sum_{i=-k}^{m-1} D_i \times 16^i \qquad D_i \text{ 的取值为 0 到 9 和 } A \text{ 到 } F \tag{3.6}$$

例如 $(1A.08)_{16}=1\times16^1+10\times16^0+8\times16^{-2}=(26.03125)_{10}$。

上述二式中所用符号的意义与式(3.3)中所用符号的意义类同，但此处 D_i 包含的基本符号分别限于 0～7 和 0～9、A～F，各位的码权分别为 8^i 和 16^i。

把用二进制、八进制、十六进制表示的数转换成十进制数的值，使人能更容易地衡量这个数值的大小。

由于 $\log_2 8=3$，$\log_2 16=4$，即 1 位八进制的数可以用 3 位二进制的数重编码来得到，1 位十六进制的数可以用 4 位二进制的数重编码得到，故人们通常认为，在计算机这个领域内，八进制和十六进制数只是二进制数的一种特定的表示形式。表 3.1 给出了少量二进制、八进制、十六进制和十进制数的对应关系。

表 3.1　二进制、八进制、十六进制和十进制的对应关系

二进制数	八进制数	十六进制数	十进制数的值
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12

续表

二进制数	八进制数	十六进制数	十进制数的值
1101	15	D	13
1110	16	E	14
1111	17	F	15

在把二进制数转换成八进制或十六进制表示时,应从小数点所在位置分别向左、向右对每3位或每4位二进制位进行分组,写出每一组所对应的1位八进制数或十六进制数。若小数点左侧(即整数部分)的位数不是3或4的整数倍,可以按在数的最左侧补零的方法理解;对小数点右侧(即小数部分),应按在数的最右侧补零的方法处理,否则容易转换错。对不存在小数部分的二进制数(整数),应从最低位开始向左把每3位划分成一组,使其对应一个八进制位,或把每4位划分成一组,使其对应一个十六进制位。例如,$(10.101)_2$变成八进制时,应把它理解为$(010.101)_2$,是$(2.5)_8$,即八进制的2.5。当把它转换为十六进制时,应首先变为$(0010.1010)_2$,是$(2.A)_{16}$,即十六进制的2.A,而不是$(2.5)_{16}$。又如:

$$(1100111.10101101)_2=(147.532)_8$$

$$(1100111.10101101)_2=(67.AD)_{16}$$

八进制和十六进制之间的转换不是很常用,二者经过二进制的中间结果进行转换是方便的。

4. 二进制数的运算规则

二进制数之间可以执行算术运算和逻辑运算,其规则简单,容易实现。

1) 加法运算规则

0+0=0
0+1=1
1+0=1
1+1=0 (产生进位)

例如:

```
  1101
+) 1001
 10110
```

2) 减法运算规则

0−0=0
0−1=1 (产生借位)
1−0=1
1−1=0

例如:

```
  1101
−) 0111
  0110
```

3) 乘法运算规则

二进制数乘法的计算方法,与十进制数乘法的计算方法类似,按乘数每一位的值计算出部分积,对全部部分积求累加和则得到最终乘积。

0×0=0
0×1=1
1×0=1
1×1=1

例如:

```
  1101
×1001
  1101
 0000
0000
1101
1110101
```

4）除法运算规则

二进制数除法的计算与十进制数除法类似，也由减法、逐位上商等操作分步完成。例如：

```
        1101
1001 ) 1110101
       1001
       ----
        1011
        1001
        ----
          1001
          1001
          ----
             0
```

逻辑运算是在对应的两个二进制位的逻辑值之间进行的，与相邻的高低位的值均无关，即不存在进位、借位等问题。

5）逻辑或运算规则（运算符为∨）

$0 \vee 0 = 0$

$0 \vee 1 = 1$

$1 \vee 0 = 1$

$1 \vee 1 = 1$

例如：

```
    1100
∨   1010
--------
    1110
```

6）逻辑与运算规则（运算符为∧）

$0 \wedge 0 = 0$

$0 \wedge 1 = 0$

$1 \wedge 0 = 0$

$1 \wedge 1 = 1$

例如：

```
    1100
∧   1010
--------
    1000
```

7）逻辑非运算规则（运算符为¬）

$$\neg 0 = 1$$

$$\neg 1 = 0$$

逻辑非实现对单个逻辑值的处理，而不是对两个逻辑值的运算，逻辑非又被称为逻辑取反操作。对逻辑数 1011 逐位进行取反，其结果为 0100。

8）逻辑异或运算规则（运算符为∀）

$0 \forall 0 = 0$

$0 \forall 1 = 1$

$1 \forall 0 = 1$

$1 \forall 1 = 0$

例如：

```
    1100
∀   1010
--------
    0110
```

与、或、非操作是三种最基本的逻辑操作，用它们可以组合出任何逻辑运算功能。某些情况下，还要用到逻辑异或操作。逻辑异或实现的是按位加功能，只有参与异或操作的两个逻辑值不同时（一个为 0，另一个为 1），结果才为 1，和或操作结果的差异表现在：或操作中 1 或 1＝1，而异或操作则是 1 异或 1＝0。

3.1.3 检错纠错码

1. 检错纠错的有关概念和实现思路

数据在计算机系统内加工、存取和传送的过程中可能产生错误。为减少和避免这类错

误,一方面是精心选择各种电路,改进生产工艺与测试手段,尽量提高计算机硬件本身的可靠性;另一方面是在数据编码上找出路,即采用带有某种特征能力的编码方法,通过少量的附加电路,使之能发现某些错误,甚至能准确地确定出错位置,进而提供自动纠正错误的能力。

数据校验码就是一种常用的带有发现某些错误,甚至带有一定自动改错能力的数据编码方法。它的实现原理是在合法的数据编码之间,加进一些不允许出现的(非法的)编码,使合法数据编码出现某些错误时,就成为非法编码。这样,则可以通过检查编码的合法性来达到发现错误的目的。合理地设计编码规则,安排合法或不合法的编码数量,就可以得到发现错误的能力,甚至达到自动改正错误的目的。这里用到一个**码距**(最小码距)的概念。码距是指任意两个合法码之间至少有几个二进制位不相同,仅有一位不同,称其(最小码距)为1,例如用4位二进制表示16种状态,则16种编码都用到了,此时码距为1。就是说,任何一个编码状态的4位码中的一位或几位出错,都会变成另一个合法码,此时无检错能力。若用4个二进制位表示8种合法状态,就可以只用其中的8个编码来表示之,而把另8种编码作为非法编码,此时可能使合法码的码距为2。一般说来,合理地增大编码的码距,就能提高发现错误的能力,但表示一定数量的合法码所使用的二进制位数要变多,增加了电子线路的复杂性和数据存储、数据传送的数量。在确定与使用数据校验码的时候,通常要考虑在不过多增加硬件开销的情况下,尽可能地发现较多的错误,甚至能自动改正某些最常出现的错误。常用的数据校验码是奇偶校验码、汉明校验码、循环冗余校验码等。纠错编码是对检错编码更进一步的发展和应用。

计算机内经常遇到的错误有两大类:随机错误和突发错误。前者指孤立出现的一个错误,后者指连续产生的一批(彼此之间可能有关联)错误。对它们处理的难度和复杂度会有很大不同,在我们的课程中基本不涉及对突发错误的检查与纠正问题,有兴趣者请自行查阅有关资料。纠错编码的分类方案如图3.1所示。

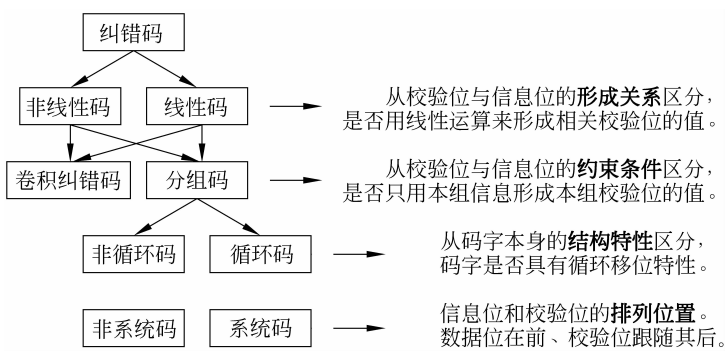

图3.1 纠错码的分类

2. 3种常用的检错纠错码

1) 奇偶校验码

奇偶校验码是一种开销最小,能发现数据代码中一位出错情况的编码,常用于存储器读写检查,或ASCII字符、其他类型信息传送过程中的出错检查。它的实现原理,是使原来合法编码码距由1增加到2。若合法编码中有一个二进制位的值出错了,由1变成

0，或由 0 变成 1，这个码必将成为非法编码。实现的具体方法，通常是为一个字节补充一个二进制位，称为校验位，通过设置校验位的值为 0 或 1 的方式，使字节自身的 8 位和该校验位含有 1 值的位数一定为奇数或偶数。在使用奇数个 1 的方案进行校验时，称为奇校验，反之则称为偶校验。依据 8 位的数据位中为 1 值的个数确定校验位的值，是由专设的线路实现的，通常使用几级异或门电路产生校验位的值，其输入是数据位的信号或异或运算的中间结果，最后的输出信号是校验位的值。例如，当要把一个字节的值写进主存时，首先用此电路形成校验位的值，然后将这 9 位的代码作为合法数据编码写进主存。当下一次读出这一代码时，再用相应线路检测读出 9 位码的合法性。若在主存写进、存储或读出的过程中，某一个二进制位上出现错误，得到的 9 位码必变成非法编码，从而发现一定是哪一位上出现了错误。这种方案只能发现一位错或奇数个位出错，但不能确定是哪一位错，也不能发现偶数个位出错。考虑到，一位出错的概率比多位同时出错的概率高得多，该方案还是有很好的实用价值。

下面给出对几个字节值的奇偶校验的编码结果：

数据	奇校验的编码	偶校验的编码
00000000	**1**00000000	**0**00000000
01010100	**0**01010100	**1**01010100
01111111	**0**01111111	**1**01111111
11111111	**1**11111111	**0**11111111

该例子中，码字的最高一位为校验位，其余低八位为数据位。从中可以看到，校验位的值取 0 还是 1，是由数据位中 1 的个数、是奇校验还是偶校验方案共同决定的。

2）汉明校验码

这是由 Richard Hamming 于 1950 年提出、目前还被广泛采用的一种很有效的校验方法。其只要增加少数几个校验位，就能检测出两位同时出错、亦能检测出一位出错并能自动恢复该出错位的正确值的有效手段，后者被称为自动纠错。它的实现原理，是在 k 个数据位之外加上 r 个校验位，从而形成一个 $k+r$ 位的新的码字，使新得到的码字的码距比较均匀地拉大。若把数据的每一个二进制位合理地分配在几个不同的偶校验位的组合中，使每一个数据位与不同的校验位组合建立准确的对应关系，则当某一位出错后，就会引起相关的几个校验位的值发生变化，这不但可以发现出错，还能指出是哪一位出错，为进一步自动纠错提供了依据。

假设为 k 个数据位设置 r 个校验位，则校验位能表示 2^r 个状态，可用其中的一个状态指出“没有发生错误”，用其余的 2^r-1 个状态指出有错误发生在某一位，包括 k 个数据位和 r 个校验位，因此校验位的位数应满足如下关系：

$$2^r \geqslant k+r+1 \tag{3.7}$$

如果要求在检出与自动校正一位错的基础上，也能同时发现两位错，此时应该在前一种条件下再增加一位校验位，称为总校验位，专用于区分是一位出错还是双位出错，则此时的校验位的位数 r 和数据位的位数 k 应满足下述关系：

$$2^{r-1} \geqslant k+r \tag{3.8}$$

式(3.8)是通过用 $r-1$ 替代式(3.7)中的 r 得到的。

按上述不等式，可计算出数据位 k 与校验位 r 的对应关系，如表 3.2 所示。

表 3.2　数值位 k 和校验位 r 的对应关系

k 值	最小的 r 值	k 值	最小的 r 值
3～4	4	27～57	7
5～11	5	58～120	8
12～26	6		

设计汉明码编码的关键技术,是合理地把每个数据位分配到 r 个校验组中,以确保能发现码字中任何一位出错;若要实现纠错,还要求能指出是哪一位出错,对出错位求反则得到该位的正确值。例如,当数据位为 3 位(用 $D_3\ D_2\ D_1$ 表示)时,检验位应为 4 位(用 $P_4\ P_3\ P_2\ P_1$ 表示)。可通过表 3.3 表示的关系,完成把每个数据位划分在形成不同校验位的偶校验值的逻辑表达式中。

表 3.3　校验位与数据位的对应关系

	D_3	D_2	D_1	P_4	P_3	P_2	P_1
	1	1	1	**1**	1	1	1
最低 3 行	1	1	0	0	**1**	0	0
	1	0	1	0	0	**1**	0
	0	1	1	0	0	0	**1**
低 3 行编码	6	5	3	0	4	2	1

表中 $D_3\ D_2\ D_1$ 为 3 位数据位,$P_4\ P_3\ P_2\ P_1$ 为 4 位校验位,其中低 3 位中的每一个校验位 $P_3\ P_2\ P_1$ 的值,都是用 3 个数据位中不同的几位通过偶校验运算规则计算出来的。其对应关系是:对 P_i(i 的取值为 1～3),总是用处在 P_i 取值为 1 的行中的、用 1 标记出来的数据位来计算出该 P_i 的值。最高一个校验位 P_4,被称为总校验位,它的值是通过对全部 3 个数据位和其他全部校验位(不含 P_4 本身)执行偶校验计算求得的。

在 P_1、P_2、P_3、P_4 竖列相应行分别填 1,在该 4 列的低 3 横行其他位置分别填 0,在最顶横行的每个尚空位置都分别填 1。**若只看低 3 横行**,右 4 竖列的 3 个位的组合值分别为十进制的 1、2、4、0,则分配 $D_1\ D_2\ D_3$ 列的组合值为 3 5 6,保证低 3 横行各竖列的编码值各不相同。

计算各校验位的值的过程叫作编码,按刚说明的规则,4 个校验位所用的编码方程为

$$P_4 = D_3 \oplus D_2 \oplus D_1 \oplus P_3 \oplus P_2 \oplus P_1$$

$$P_3 = D_3 \oplus D_2$$

$$P_2 = D_3 \oplus D_1$$

$$P_1 = D_2 \oplus D_1$$

由多个数据位和多个校验位组成的一个码字,将作为一个数据单位处理,例如被写入内存或被传送走。之后,在执行内存读操作或在数据接收端,则可以对得到的码字,通过偶校验来检查其合法性,通常称该操作过程为译码,所用的译码方程为

$$S_4 = P_4 \oplus D_3 \oplus D_2 \oplus D_1 \oplus P_3 \oplus P_2 \oplus P_1$$
$$S_3 = P_3 \oplus D_3 \oplus D_2$$
$$S_2 = P_2 \oplus D_3 \oplus D_1$$
$$S_1 = P_1 \oplus D_2 \oplus D_1$$

译码方程和编码方程的对应关系很简单。译码方程是用一个校验码和形成这个校验码的编码方程执行异或，实际上是又一次执行偶校验运算。通过检查 4 个 S 的结果，可以实现检错纠错的目的。实际情况是，当译码求出来的 S_4、S_3、S_2、S_1 的结果与表 3.3 中的哪一列的值相同，就说明是哪一位出错；故人们又称表 3.3 为出错模式表。若出错的是数据位，对其求反则实现纠错；若出错的是校验位则不必理睬。举例如下。

(1) 任何一位(含数据位、校验位)均不错，则 4 个 S 都应为 0 值(思考为什么会如此)。

(2) 任何单独一位数据位出错，4 个 S 中会有 3 个为 1；如 D_3 错，则 S_4 S_3 S_2 S_1 为 1110。

(3) 若单独一位校验位出错，4 个 S 中会有一个或两个为 1；如 P_1 错，S_4 S_3 S_2 S_1 为 1001，如 P_4 错，S_4 S_3 S_2 S_1 为 1000。

(4) 任何两位(含数据位、校验位)同时出错，S_4 一定为 0，而另外 3 个 S 位一定不全为 0，此时只知道是两位同时出错，但不能确定是哪两位出错，故已无法纠错。如 D_1 和 P_2 出错，会使 S_4 S_3 S_2 S_1 为 0001。请注意，S_4 的作用在于区分出错的位数是奇数还是偶数，S_4 为 1 是奇数个位出错，为 0 是无错或者偶数个位出错。这不仅为发现两位错所必需，也是为**确保**能发现并改正一位错所必需的。若不设置 S_4，某两位出错对几个 S 的影响与单独另一位出错可能是一样的(不必花费精力推敲)，此时若不加以区分，简单地按一位出错自动完成纠错处理反而会越纠越错。

3) 循环冗余校验码——CRC 码

二进制信息位串沿一条信号线逐位地在部件之间或计算机之间传送称为串行传送。CRC(Cyclic Redundancy Check)码可以发现并纠正信息串行读写、存储或传送过程中出现的一位、多位错误。因此，在外存储器设备读写和计算机之间串行通信的场合得到普遍应用。

CRC 码一般是指 k 位信息码之后拼接 r 位校验码。其特点是一个合法码字实现首尾连接的循环移位，每一步移位操作得到的都是合法码字。应用 CRC 码的关键是如何从 k 位信息位简便地得到 r 位校验位(编码)的值，以及如何判断 $k+r$ 位的码字是否正确。下面仅就 CRC 码应用中的问题做简单介绍(有关的理论问题请参阅有关书籍)。

(1) 模 2 运算是指以按位模 2 相加为基础的四则运算，运算时不考虑位间进位和借位。

(2) **CRC 码编码**是经过模 2 除运算来完成的。其得到商的规则是：当被除数或者部分余数的最高位为 1 时，商 1；为 0 时，商 0。当余数的位数小于除数的位数时结束运算，该余数即为校验位的值。实现这种运算用到的线路很简单，实现左移的移位寄存器(实现串行输出)加上与除数有关的异或控制门即可完成，关键是如何得到模 2 除运算的除数(称为生成多项式)，通常可去查数学表。

(3) **CRC 的译码与纠错**，在数据接收端，用约定的生成多项式去除收到的循环码，如果码字无误则余数应为 0，如有某一位出错，则余数不为 0，且不同数位出错余数会不同，余数

与出错位的对应关系是不变的,只与码制和生成多项式有关。还用这个电路,再经过几步循环移位和相关操作即可实现纠错。

3.2 数据表示

在这一节,重点讲解数据表示,即计算机内最常用的信息编码。其包括逻辑型数据表示、中西文字符编码表示、数值型数据的编码表示。这些内容是数字计算机设计与应用的基本知识之一,要求能熟练掌握并运用自如。

3.2.1 逻辑类型数据的表示

逻辑数据是用来表示二值逻辑中的"是"与"否"或称"真"与"假"两个状态的数据。很容易想到,用计算机中的基2码的两个状态1和0恰好能表示逻辑数据的两个状态。例如用1表示真,则0表示假。请注意,这里的1和0没有了数值有无或大小的概念,只有逻辑上的意义。在计算机内,可以用一位基2码表示逻辑数据,就是说,8个逻辑数据可以存放在1个字节中,可用其中的每一位表示一个逻辑数据。

正确地建立逻辑数据的概念是十分重要的。因为计算机内部的所有数字化信息,不论它们是数据、指令或控制信号,都是用基2码表示的,其存储与处理都是用逻辑线路实现的。可以说,逻辑线路是计算机硬件组成的基石,而逻辑线路处理的对象就是逻辑型数据,通称逻辑变量,即实现输出逻辑变量与输入逻辑变量之间一定的逻辑运算关系。这些内容在本教材的第2章已有一定阐述。计算机组成原理课程的相当大的一部分内容,涉及计算机各功能部件本身的逻辑功能与实现,以及计算机整机各部件之间的逻辑关系与连接。

3.2.2 字符类型数据的表示

字符是计算机中使用最多的信息形式之一,是人与计算机通信、交互作用的重要媒介。在计算机中,要为每个字符指定一个确定的编码,作为识别与使用这些字符的依据。这些编码的值,是用一定位数的基2码的两个基本符号1和0进行编码给出的。

1. ASCII码和EBCDIC码

使用得最多的、最普遍的是ASCII字符编码,即American Standard Code for Information Interchange,如表3.4所示。

表3.4 ASCII字符编码表

$b_6b_5b_4$ / $b_3b_2b_1b_0$	000	001	010	011	100	101	110	111
0 0 0 0	NUL	DLE	SP	0	@	P	、	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s

续表

$b_3b_2b_1b_0$ \ $b_6b_5b_4$	000	001	010	011	100	101	110	111
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	↑	m	~
1111	SI	US	/	?	O	-	o	DEL

从表中可以看到：每个字符是用7位基2码表示的，其排列次序为$b_6b_5b_4b_3b_2b_1b_0$，在表中的$b_6b_5b_4$为高位部分，$b_3b_2b_1b_0$为低位部分。而一个字符在计算机内实际上用8位表示。正常情况下，最高一位b_7为0。在需要奇偶校验时，这一位可用于存放奇偶校验位的值，此时称这一位为校验位。

ASCII是由128个字符组成的字符集。其中编码值0～31不对应任何可印刷(或称有字形)字符，通常称它们为控制字符，用于通信中的通信控制或对计算机设备的功能控制。编码值为32的是空格(或间隔)字符SP。编码值为127的是删除控制DEL码。其余的94个字符称为可印刷字符，有人把空格也计入可印刷字符时，则称有95个可印刷字符。请注意，这种字符编码中有如下两个规律。

(1) 字符0～9这10个数字符的高3位编码为011，低4位为000～1001。当去掉高3位的值时，低4位正好是二进制形式的0～9。这既满足正常的排序关系，又有利于完成ASCII码与二进制码数值之间的类型转换。

(2) 英文字母的编码值满足正常的字母排序关系，且大、小写英文字母编码的对应关系相当简便，差别仅表现在b_5一位的值为0或1，有利于大、小写字母之间的编码变换。

另有一种字符编码主要用在IBM计算机中的EBCDIC编码(Extended Binary Coded Decimal Interchange Code)。它采用8位码，有256个编码状态，但只选用其中一部分。0～9这10个数字符的高4位编码为1111，低4位仍为0000～1001。大、小写英文字母的编码同样满足正常的排序要求，而且有简单的对应关系，即同一个字母的大、小写的编码值仅最高的第二位的值不同，易于判别与变换。

2. 字符串的表示

随着计算机在文字处理与信息管理中的广泛应用,字符串已成为最常用的数据类型之一。许多计算机中都提供字符串操作功能,一些计算机还给出读写字符串的机器指令。

字符串是指连续的一串字符。通常方式下,它们占用主存中连续的多个字节,每个字节存一个字符。当主存字由2个或4个字节组成时,在同一个主存字中,既有按从低位字节向高位字节的顺序存放字符串内容的,也有按从高位字节向低位字节的次序顺序存放字符串内容的。这两种存放方式都是常用方式,不同的计算机可以选用其中任何一种。例如,IF A>B THEN READ(C)就可以有两种不同的存放方式,如图3.2所示。

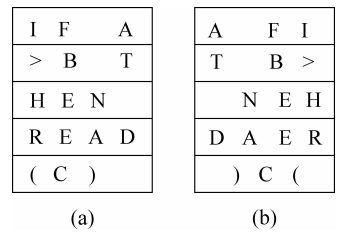

图3.2 字符串的两种存放方式

假定每个主存字由4个字节组成,图3.2(a)是按从高位字节向低位字符的次序存放字符串,图3.2(b)按从低位字节向高位字节的次序存放字符串。主存中每个字节存的都是相应字符的ASCII编码值。例如对图3.2(a)来说,每个字节分别存放的是十进制的73、70、32、65、62、66、32、84、72、69、78、32、82、69、65、68、40、67和41和32。

3. 中文汉字的编码

西文字符是一种拼音文字,用30个左右的字母可以拼写出所有单词,再加上一些数学符号、标点符号等辅助字符,就可以构成一个完整的西文字符集,字符总数不超过256个,所以使用7位或8位二进制位就可以表示。我国是个多民族的国家,使用包括汉、蒙、藏、朝鲜、僮、苗、哈尼、维吾尔等近60种民族文字,其中用得最多且最广的是汉字。汉字也是字符,但有其特殊性,汉字是表意文字,汉字的数量很多,总数超过6万个。一个字就是一个方块图形,编码就要复杂一些,也给汉字在计算机内部的表示、汉字的传输与交换、汉字的输入与输出等带来了一系列问题。下面对汉字的编码、输入、存储和输出编码进行介绍。

1) 汉字内码

汉字内码又称机内码,是指用于汉字信息的存储、交换、检索等操作的机内代码,一般采用2个字节表示。各种不同输入法输入的汉字,其内码在计算机中是相同的。汉字内码等于汉字国标码加上8080H,即表示汉字的两个字节信息的最高一个二进制的值必定为1,例如“中”字的机内码为D6D0H,这有利于在文字处理软件中区分中文字符和西文字母。

计算机中的汉字编码都是通过软件定义和处理的,硬件不直接提供对汉字处理的支持。

1981年,我国制定了“中华人民共和国国家标准信息交换汉字编码”,代号为GB 2312—1980。在这种编码的字符集中,一共收录了7445个汉字和图形,其中汉字6763个、图形682个。所谓区位码,是将上述国家标准局公布的6763个两级汉字分为94个区,每个区分94位,一个汉字所在的区号和位号简单地组合在一起就构成了这个汉字的区位码,其中高两位表示区号,低两位表示位号,都采用十进制数。实际上也就是把汉字表示成二维数组,每个汉字在数组中的下标就是区位码。区位码可以唯一地确定一个汉字或符号。例如“中”字位于54区48位,“中”字的区位码即为“5448”。区位码的值加上十六进制的2020成为国标码。

2000年公布的汉字编码的国家标准是GB 18030,该标准收录了27484个汉字。编码标准采用单字节、双字节(2B)、四字节(4B)。GB 18030是简体中文字符集GB 18030—2000

和 GB 18030—2005 的代号。GB 18030—2005 的标准名称是“中华人民共和国国家标准 GB 18030—2005：信息技术：中文编码字符集(Chinese National Standard GB 18030—2005：Chinese coded character set)”。GB 18030—2005 取代 GB2312、GB 1300、GBK、18030—2000 和 Big5 标准，支持 Unicode 的 CJK 统一汉字，共收录 70244 个汉字。GB 18030 编码在码位空间上与 Unicode 标准对应。

2) 汉字输入码

键盘是计算机系统最重要的输入设备，用于西文这种小字符集的文字输入是很方便的。由于汉字是大字符集，若设计并使用专门的汉字输入键盘则有许多困难，例如键数太多、查找不方便、成本过高等。经济又便捷的输入方式还是选用普通的键盘，采用几个键的组合代表一个汉字，这就需要解决汉字的输入码问题。

汉字输入码也称外码，是为了将汉字输入计算机而编制的代码，是代表某一汉字的一串键盘符号。汉字输入码的种类有很多，主要可划分为以下几种。

(1) 数字编码，如区位码、国标码、电报码等。

(2) 拼音编码，如全拼码、双拼码、简拼码等。

(3) 字形编码，如王码五笔、郑码、大众码等。

(4) 音形编码，如表形码、钱码、智能 ABC 等。

在输入汉字的过程中，输入一个汉字需要敲击的键的次数、用到的规则、方便程度和输入的速度，不同的输入码方案会有所不同，可谓各有长短，不同的人也会有自己的偏爱。

在键盘输入法之外，还有手写汉字联机识别输入和印刷汉字扫描输入后自动识别两种方法，现均已达到实用水平。现在还有一种用语音输入汉字的方法，虽然简单易操作，但离实用阶段尚有差距。

3) 汉字字形码

汉字字形码又称汉字字模码，是对汉字字形经过点阵数字化后形成的一串二进制数，用于汉字的显示和打印。每个汉字的点阵规模可以有所不同，通常有以下几种类型。

(1) 简易型汉字：16×16，32 字节/汉字。

(2) 普通型汉字：24×24，72 字节/汉字。

(3) 提高型汉字：32×32，128 字节/汉字。

将所有汉字的字模点阵代码按内码顺序集中起来，构成了汉字库。汉字库可以被保存在磁盘，固化在内存或使用时调入内存，固化在打印机的逻辑电路中。

3.2.3 多媒体信息编码

计算机能对图画、声音、音乐和电影等信息进行各种处理，如支持建筑/机械等图纸制作、图像扫描输入和处理、音乐合成、语音识别与合成、视频会议等。这势必涉及如何在计算机内部表示相关媒体信息的问题。

1. 图的编码

在计算机中，图可以分成图像(Image)和图形(Graphics)两种类型。通常可以用矢量图(Vector Graphics)表示，也可以用位图(Bitmap 或 Bitmapped Image)表示。

图像表示法类似于汉字的字模点阵码。只是汉字描述的仅仅是形状(即字模)，而对于图像除了要描述的形状外，还要描述其颜色或灰度。例如，用 8 位二进制数表示一个像素点

的灰度,用3个字节分别表示彩色图像一个像素点的3个彩色分量(R、G、B)的强度。

图形表示法是根据画面中所包含的内容,分别用几何要素(如点、线、面、体)和物体表面的材料与性质以及环境的光照条件、观察位置等来进行描述,如工程图纸、地图等可采用图形表示,它们易于加工处理,数据量也少。汉字字形的轮廓描述方法就属于图形表示。

2. 声音的编码

计算机处理的声音可以分为如下3种:第1种是语音,即人的说话声;第2种是音乐,即各种乐器演奏出的声音;第3种是效果声,如掌声、打雷、爆炸等声音。在计算机内部可以用波形法和合成法两种方法来表示声音。所有的声音都可用波形法来表示,但更多用于语音和效果声,对于音乐声,则用合成法来表示更好一些。

声音可以用一种连续的随时间变化的声波波形来表示。这种波形反映了声音在空气中的振动。计算机要能够对声音进行处理,必须将声波波形转换成二进制表示形式,这个转换过程称为声音的"数字化编码"。声音的数字化编码过程分为以下3步。

(1) 以固定的时间间隔对声音波形进行采样,使连续的声音波形变成一个个离散的采样信号(即样本值),每秒采样的次数被称为采样频率,通常采用44.1kHz、22.05kHz和11.025kHz这3种频率,也可以自行选择。采样频率越高,声音的保真度越好。

(2) 对得到的每个样本值进行模数转换(称为A/D转换),将每个样本值用一个二进制数字量来表示。这个过程即所谓的量化处理。转换后的二进制数的位数一般可以有两种选择:16位或8位。位数越多,量化精度越高,噪声越小,声音质量也就越好。

(3) 对产生的二进制数据进行编码(有时还需进行数据压缩),以按照规定的统一格式进行表示。表3.5是波形法采用的几种不同参数的数字化声音信息的比较。

表3.5 波形法采用的几种不同参数的数字化声音信息的比较

几种不同参数的数字化声音信息			
采样频率(kHz)	量化精度(bit)	每分钟的数据量(MB)	质量与应用
44.1	16	5.3	相当于激光唱片质量,用于最高质量要求的场合
22.05	16 8	2.65 1.32	相当于调频广播质量,可用作伴音和声响效果
11.025	16 8	1.32 0.66	相当于调幅广播质量,可用作伴音和解说词

声音的另一种表示方法是合成法。它主要适用于音乐在计算机内部的表示。它把音乐的乐谱、弹奏的乐器类型、击键力度等用符号进行记录。目前广为采用的一种标准为MIDI(Musical Instrument Digital Interface)。例如,在MIDI标准中,一个MIDI文件(扩展名为mid)中包含了一连串的MIDI消息。每个MIDI消息由若干字节组成,通常第一个字节为状态字节,其后为1个或2个数据字节。状态字节的特征是最高位为1,它用来指出紧随其后的数据字节的用途和含义。数据字节的最高位为0,后面是MIDI消息的信息内容。例如,表示一个"中央C"的MIDI消息可由以下3个字节组成:90h、3Ch、40h。其中90是状态字节,它表示一个音符的开始;3C表示击键的位置为"中央C";40表示击键的速度为中等。与波形表示方法相比,采用合成法的MIDI表示,其数据量要少得多(相差2~3个数量

级),编辑修改也比较容易。但它主要适用于表现各种乐器所演奏的乐曲,不能用来表示语音等其他声音。

为了处理上述两类数字声音信息,计算机内部有一个相应的声音处理硬件(如声卡),用来完成对各种声音输入设备输入的声音进行数字化编码处理,并将处理后的数字波形声音还原为模拟信号声音,经功率放大后输出。有的声音处理硬件可外接 MIDI 键盘,将弹奏的乐曲以 MIDI 形式输入计算机内,并将计算机处理后的 MIDI 乐曲经合成器(波表合成器或频率合成器)合成为音乐声音后输出。

3. 视频信息的编码

视频信息的信息量最丰富,它是一种最有感染力的承载信息媒体。视频信息的处理是多媒体技术的核心。计算机通过在内部安装一个视频获取设备(如视频卡),将各类视频源(如电视机、摄像机、VCD 机或放像机等)输入的视频信号进行相应的处理,转化为计算机内部可以表示的二进制数字信息。

视频获取设备将视频信号转换为计算机内部表示的二进制数字信息的过程被称为视频信息的"数字化"。视频信息的数字化过程比声音更复杂一些,它是以一幅幅彩色画面为单位进行的。每幅彩色画面有亮度(Y)和色差(U,V)3 个分量,对 Y、U、V 这 3 个分量需分别进行采样和量化,得到一幅数字图像。表 3.6 是几种常用数字视频的格式。

表 3.6　几种常用数字视频的格式

名　　称	分辨率	量化精度	数据量
CCIR601	720×576×25	8+4+4	124Mb/s
CIF	360×288×25	8+4+4	26Mb/s
QCIF	180×144×25	8+4+4	6.5Mb/s

从表 3.6 中可以看出,数字视频信息的数据量非常大。例如,一分钟 CCIR601 数字视频,其数据量约为 1Gb,这样大的数据量无论是存储、传输还是处理,都相当困难。要解决这个问题就必须对数字视频信息进行压缩编码处理。在获取数字视频的同时立即进行压缩编码的处理,称为实时压缩。有些视频获取设备具有实时压缩的功能。

3.2.4　数值类型数据的表示

数值类型数据是表示数量多少、数值大小的数据。它们有多种类型和不同的表示方法。

日常生活中,用得最多的是带正、负符号的十进制数字串形式的表示方法,例如 3.1416、−256 等。这种形式的数据难以在计算机内直接存储与运算,主要用于计算机的输入/输出操作,是人-机间交换数据的媒介。

在计算机内有时可以使用二-十进制编码,即用 4 位基 2 码编码一位十进制数,一个多位的十进制数被表示为这种编码的数串。4 位基 2 码组成 16 个状态,一位十进制数仅有 10 个状态,因此,怎样从 16 个状态中选用其中 10 个就有非常多的方案。下面还会详细讨论。

在计算机中最常用的方法,是用二进制码表示数据,包括整数、纯小数、实数(通称浮点数)。这有利于减少所用存储单元的数量,又便于实现算术运算。为了更有效地、方便地统一表示正数、负数和零,对二进制数又可以选用原码、反码、补码等多种编码方案表示。

数值数据的表示与编码方案,与计算机的设计、实现关系十分密切,且涉及一些基础理论与处理技术,有必要进行较为详细的讨论,放在3.3节专门讲解。

数值数据用于表示数量的大小。讨论数值数据时,经常用到数值范围和数据精度两个概念。数值范围是指一种类型的数据所能表示的最大值和最小值;数据精度,通常用实数所能给出的有效数字的位数表示。这两个概念是不同的。在计算机中,它们的值与用多少个二进制位表示某种类型的数据,以及怎么对这些位进行编码有关。

二进制数主要分成定点小数、整数与浮点数3类,还有用4位二进制表示一个十进制数位的压缩数字串。先简要说明这3类二进制数的一般表示,再详细讨论其具体编码形式。

1. 定点小数的表示方法

定点小数是指小数点准确固定在数据某个位置上的小数。从实用角度看,都把小数点固定在最高数据位的左边,小数点前边再设一位符号位。按此规则,任何一个小数都可以被写成

$$N = N_S N_{-1} N_{-2} \cdots N_{-M}$$

如果在计算机中用 $m+1$ 个二进制位表示上述小数,则可以用最高(最左)一个二进制位表示符号(如用0表示正号,则1就表示负号),而用后面的 m 个二进制位表示该小数的数值。小数点不用明确表示出来,因为它总是固定在符号位与最高数值位之间,已成定论。定点小数的取值范围很小,对用 $m+1$ 个二进制位的小数来说,其值的范围为

$$|N| \leqslant 1 - 2^{-m}$$

即小于1的纯小数,这对用户算题是十分不方便的,因为在算题前,必须把要用的数,通过合适的"比例因子"化成绝对值小于1的小数,并保证运算的中间和最终结果的绝对值也都小于1,在输出真正结果时,还要把计算的结果按相应比例加以扩大。

定点小数表示法主要用在早期的计算机中,它最节省硬件。随着计算机硬件成本的大幅度降低,现代的通用计算机都被设计成能处理与计算多种类型数值数据的计算机。我们主要通过定点小数讨论数值数据的不同编码方案,定点小数也被用来表示浮点数的尾数部分。

2. 整数的表示方法

整数表示的数据的最小单位为1,可认为它是小数点定在数值最低位右面的一种数据。

整数又被分为带符号和不带符号的两类。对带符号的整数来说,符号位被安排在最高位,任何一个带符号的整数都可以被写成

$$N = N_S N_n N_{n-1} \cdots N_2 N_1 N_0$$

对于用 $n+1$ 位二进制位表示的带符号的二进制整数,其值的范围为

$$|N| \leqslant 2^n - 1$$

对不带符号的整数来说,所有的 $n+1$ 个二进制位均被用于数值,此时数值的范围是

$$0 \leqslant N \leqslant 2^{n+1} - 1$$

即原来的符号位被解释为 2^n 的数值。

有时也用不带符号的整数表示另外一些内容,此时它不再被理解为数值的大小,而被看成一串二进制位的某种组合。

在很多计算机中,往往同时使用不同位数的几种整数,如用8位(称为字节)、16位(称为半字)、32位(称为字)或64位(称为双字)二进制来表示一个整数,它们占用的存储空间

和所表示的数值范围是不同的。

3. 浮点数的表示方法

浮点数是指小数点在数据中的位置可以左右移动的数据。它通常被表示成

$$N = M \times R^E$$

这里的 M(Mantissa)被称为浮点数的尾数;R(Radix)被称为阶码的基数;E(Exponent)被称为阶的阶码。计算机中一般规定 R 为 2、8 或 16,是一个确定的常数,不需要在浮点数中明确表示出来。因此,要表示浮点数,一是要给出尾数 M 的值,通常用定点小数形式来表示。它决定了浮点数的表示精度,即可以给出的有效数字的位数。二是要给出阶码,通常用整数形式表示。它指出的是小数点在数据中的位置,决定了浮点数的表示范围。浮点数也要有符号位。在计算机中,浮点数通常被表示成如图 3.3 所示的格式。

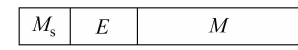

图 3.3　浮点数的格式

图 3.3 中 M_s是尾数的符号位,即浮点数的符号位,安排在最高一位;E 是阶码,紧跟在符号位之后,占用 m 位,含阶码的一位符号;M 是尾数,在低位部分,占用 n 位。

合理地选择 m 和 n 的值是十分重要的,以便在总长度为 $1+m+n$ 个二进制表示的浮点数中,既保证有足够大的数值范围,又保证有所要求的数值精度。例如,在 PDP-11/70 计算机中,用 32 位表示的一个浮点数,符号位占 1 位,阶码用 8 位,尾数用 23 位,数的表示范围约为 $\pm 1.7\times 10^{\pm 38}$,精度约为十进制的 7 位有效数字。

若不对浮点数的表示格式作出明确规定,同一个浮点数的表示就不是唯一的。例如 0.5 也可以表示为 0.05×10^{1}、50×10^{-2} 等。为了提高数据的表示精度,也为了便于浮点数之间的运算与比较,规定计算机内浮点数的尾数部分用纯小数形式给出,而且当尾数的值不为 0 时,其绝对值应大于或等于 0.5,这被称为浮点数的规格化表示。对不符合这一规定的浮点数,要通过修改阶码并同时左右移尾数的办法使其变成满足这一要求的表示形式,这种操作被称为浮点数的规格化处理,对浮点数的运算结果就经常需要进行规格化处理。

当一个浮点数的尾数为 0,不论它的阶码为何值,该浮点数的值都为 0。当阶码的值为它能表示的最小一个值或更小的值时,不管其尾数为何值,计算机都把该浮点数看成零值,通常称其为机器零,此时该浮点数的所有各位(包括阶码位和尾数位)都清为 0 值。

按国际电子电气工程师协会的 IEEE 标准,规定常用的浮点数的格式如表 3.7 所示。

表 3.7　IEEE 标准的浮点数格式

符号位	阶码	尾数	总位数	
短浮点数(单精度)	1	8	23	32
长浮点数(双精度)	1	11	52	64
临时浮点数	1	15	64	80

对短浮点数和长浮点数,当其尾数不为 0 值时,其最高一位必定为 1,在将这样的浮点数写入内存或磁盘时,不必保存该位,可左移一位尾数隐藏掉它,这种处理技术称为隐藏位技术。目的是用同样多位的尾数能多保存一个二进制位。为了保持浮点数的值不变,还要把原来的阶码值减 1。在将浮点数取回运算器执行运算时,再恢复该隐藏位的值和原来的阶码值。对临时浮点数(通常只出现在浮点运算器内部),不使用隐藏位技术。

从上述讨论可以看到,浮点数比定点小数和整数使用起来更方便。例如,可以用浮点数直接表示电子的质量 9×10^{-28} 克,太阳的质量 2×10^{33} 克,圆周率 3.1416 等。上述值都无法直接用定点小数或整数表示,要受数值范围和表示格式等各方面的限制。

4. 十进制数的编码与运算

十进制数的每一个数位的基为 10,但到了计算机内部,出于存储与计算方便的目的,必须采用基 2 码对每个十进制数位进行重编码,所需要的最少的基 2 码的位数为 $\log_2 10$,取整数为 4。4 位基 2 码有 16 种不同的组合,怎样从中选择出 10 个组合来表示十进制数位的 0～9,有非常多的可行方案,下面介绍其中的最常用的几种。

1) 十进制有权码

十进制有权码是指表示一个十进制数位的 4 位基 2 码的每一位有确定的位权。

用得最普遍的是 8421 码,即 4 个基 2 码位的权从高向低分别为 8、4、2 和 1,使用基 2 码的 0000、0001、…、1001 这 10 种组合,分别表示 0～9 这 10 个值。这种编码的优点是这 4 位基 2 码之间满足二进制的进位规则,而十进制数位之间则是十进制规则,故称这种编码为以二进制编码的十进制(Binary Coded Decimal)数,简称 BCD 码或二-十进制码。另一个优点是在数字符的 ASCII 码与这种编码之间的转换方便,即取每个数字符的 ASCII 码的低 4 位的值便直接得到该数字的 BCD 码,入/出操作简便。在计算机内实现 BCD 码之间的算术运算要复杂一些,在某些情况下,需要对加法运算的结果进行修正。修正规则如下。

(1) 若两个 8421 码每位数相加之和等于或小于 1001,即十进制的 9,不需要修正。例如,$(1)_{10}+(8)_{10}=(9)_{10}$ 的计算结果本身就是正确的。

(2) 若相加之和在 10 到 15 之间,一方面应向高位产生一进位,本位还要进行加 6 修正,进位是在进行加 6 修正时产生的。例如,$(4)_{10}+(9)_{10}=(\mathbf{1})_{10}(3)_{10}$ 就是如此。

(3) 若相加之和在 16 和 18 之间时,向高位的进位会在相加过程中自己产生,对本位还需进行加 6 修正。例如,$(7)_{10}+(9)_{10}=(\mathbf{1})_{10}(6)_{10}$ 就是如此。

另外几种有权码,如 2421、5211、84-2-1、4311 码(表 3.8),也都是用 4 位有权基 2 码表示一个十进制数位,但这 4 位基 2 码之间并不符合二进制规则。这几种有权码的特性如下。

(1) 当采用 2421、5211 和 4311 编码时,任何两个十进制数位相加产生 10 或大于 10 的结果,相应的基 2 码相加会向高一位产生进位,有利于实现逢十进位的计数和加法规则。

(2) 任何两个相加之和等于 9 的十进制数位的基 2 码,互为反码,即满足十进制数按 9 互补(9's Complement)的关系,有利于简化减法处理。表 3.8 给出的是上面提到的十进制数位的编码方案。

表 3.8　4 位有权码

十进制数	8421 码	2421 码	5211 码	84-2-1 码	4311 码
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0111	0001
2	0010	0010	0011	0110	0011
3	0011	0011	0101	0101	0100
4	0100	0100	0111	0100	1000