第1章

CHAPTER 1

微型计算机基础知识

电子计算机的诞生和发展是20世纪最重要的科技成果之一。微型计算机(简称“微机”)是计算机的一个重要分支。本章介绍微型计算机系统的基础知识,概述其发展变化,内容包括计算机中的数制和编码,微型计算机的结构、工作原理、分类及主要性能指标,典型微型计算机系统。

1.1 计算机中数的表示与编码

1.1.1 数制及其转换

数制是人们利用符号来计数的科学方法。数制有很多种,在日常生活中人们常用十进制计数,在计算机内部,一切信息的存储、处理与传送均采用二进制的形式。一个具有两种不同稳定状态且能相互转换的器件即可以用来表示一个二进制数,因而一个二进制数在计算机内部是以电子器件的物理状态来表示的,二进制的表示是最简单且最可靠的。由于八进制、十六进制与二进制之间有非常简单的对应关系,而且位数相对较少,在阅读与书写时常常采用八进制或十六进制。因而在计算机的设计及使用中,通常使用的计数方法是二进制、八进制、十进制和十六进制。

1. 进位计数制

进位计数制是采用位置表示法,即处于不同位置的同一数字符号所表示的数字不同。一般说来,如果数制只采用 R 个基本符号,则称为基 R 数制,R 称为数制的“基数”或简称“基”;而数制中每一个固定位置对应的单位值称为“权”。

对 R 进制数来说,有以下特点:

① 能选用的数码的个数等于基数 R,即各数位只允许是 $0,1,\cdots,R-1$;

② 各位的权是以 R 为底的幂;

③ 计数规则是“逢 R 进一”。

常用的四种数制的表示法如表1.1所示。

对任意一个进制数,都可以按权展开成多项式,其中每一项表示相应数位代表的数值。例如“逢十进一”的十进制数258.5可写为

$$258.5=2\times10^{2}+5\times10^{1}+8\times10^{0}+5\times10^{-1}$$

对 R 进制数 N，若用 $n+m$ 个代码 $D_i(-m\leqslant i\leqslant n-1)$ 表示，从 D_{n-1} 到 D_{-m} 自左至右排列，则其按权展开多项式为

$$N=D_{n-1}R^{n-1}+D_{n-2}R^{n-2}+\cdots+D_0R^0+D_{-1}R^{-1}+\cdots+D_{-m}R^{-m} \qquad (1.1)$$

式中，D_i 为第 i 位代码，它可取 0～$(R-1)$之间的任何数字符号；m 和 n 均为正整数，n 表示整数部分的位数，m 表示小数部分的位数。表 1.2 列出了四种数制表示的数的对应关系。

表 1.1　常用四种数制的表示法

数　　制	二进制	八进制	十进制	十六进制
进位规则	逢二进一	逢八进一	逢十进一	逢十六进一
基数	2	8	10	16
所用符号	0,1	0,1,2,…,7	0,1,2,…,9	0,1,2,…,9,A,B,…,F
权	2^i	8^i	10^i	16^i
数制标识	B	Q	D	H

表 1.2　四种数制表示的数的对应关系

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0	0	0	0	8	1000	10	8
1	1	1	1	9	1001	11	9
2	10	2	2	10	1010	12	A
3	11	3	3	11	1011	13	B
4	100	4	4	12	1100	14	C
5	101	5	5	13	1101	15	D
6	110	6	6	14	1110	16	E
7	111	7	7	15	1111	17	F

2. 数制之间的转换

1) 二、八、十六进制数转换成十进制数

转换规则为“按权相加”，即将二、八、十六进制数按权展开，求出按权展开式的值就是该数转换为十进制数的等价值。

例 1.1　$(1011.01)_2=1\times2^3+0\times2^2+1\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}=(11.25)_{10}$

$(54)_8=5\times8^1+4\times8^0=(44)_{10}$

$(7A)_{16}=7\times16^1+10\times16^0=(122)_{10}$

2) 十进制数转换成二、八、十六进制数

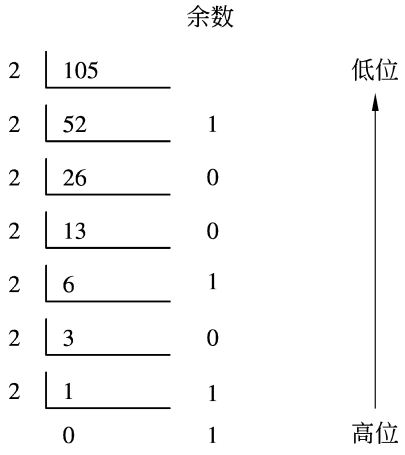
十进制数转换成二、八、十六进制数时，需要把整数部分与小数部分分别转换，然后拼接起来。

(1) 整数部分的转换

十进制整数转换为二进制整数的方法为：把被转换的十进制整数反复地除以 2，直到商为 0，所得的余数(从末位读起)就是这个数的二进制表示。简单地说，就是“除 2 取余法”。

例 1.2　将十进制整数 105 转换成二进制整数。

解：按“除 2 取余”方法进行转换的过程如下：

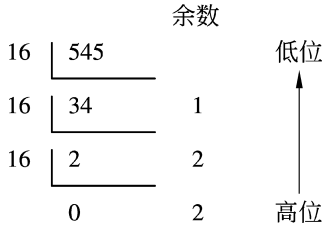


转换结果为：$(105)_{10}=(1101001)_2$。

同理，将十进制整数转换为 R 进制数，按照“除 R 取余”规则即可。

例 1.3　将十进制数 545 转换为十六进制数。

解：按“除 16 取余”方法进行转换的过程如下：



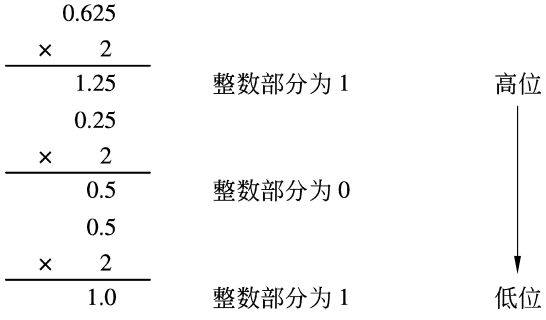
转换结果为：$(545)_{10}=(221)_{16}$。

(2) 小数部分的转换

十进制小数转换成二进制小数的方法：将十进制小数连续乘以 2，选取进位整数，直到满足精度要求为止，简称“乘 2 取整法”。

例 1.4　将十进制小数 0.625 转换成二进制小数。

解：转换过程如下：



将十进制小数 0.625 连续乘以 2，把每次所进位的整数，按从上往下的顺序写出。于是，转换结果为：$(0.625)_{10}=(0.101)_2$。

同理，十进制小数转换成 R 进制小数的方法是“乘 R 取整法”。

例 1.5　将十进制小数 0.145 转换成十六进制小数。

解：转换过程如下：

```
    0.145
  ×   16
  ------
    2.32      整数部分为 2          高位
    0.32                              |
  ×   16                              |
  ------                              |
    5.12      整数部分为 5            |
    0.12                              |
  ×   16                              |
  ------                              |
    1.92      整数部分为 1            |
    0.92                              |
  ×   16                              |
  ------                              |
   14.72      整数部分为 14(E)        低位
```

将十进制小数 0.145 连续乘以 16，取每次进位的整数，直到满足精度要求为止，因而转换结果为：$(0.145)_{10}=(0.251\mathrm{E})_{16}$。

由上可知，十进制整数部分的转换采用基数不断去除要转换的十进制数，直到商为 0 为止，将各次计算所得的余数，按最后的余数为最高位，第一位为最低位，依次排列，即得转换结果。十进制小数部分的转换采用基数不断去乘需要转换的十进制小数，直到满足要求的精度或小数部分等于 0 为止，然后取每次乘积结果的整数部分，以第一次取整位为最高位，依次排列，即可得到转换结果。

如果一个数既有小数又有整数，则应将整数部分与小数部分分别进行转换，然后用小数点将两部分连起来，即为转换结果。

例如：$(42.125)_{10}=(42)_{10}+(0.125)_{10}$

$$\downarrow \qquad \downarrow$$

$$(101010)_2 \quad (0.001)_2$$

所以　$(42.125)_{10}=(101010.001)_2$

3）二进制数与八进制数、十六进制数间的相互转换

由于 $2^3=8$，$2^4=16$，因此二进制数与八进制数、十六进制数之间的转换很简单。将二进制数从小数点位开始，向左每 3 位产生一个八进制数字，不足 3 位的左边补零，这样就得到整数部分的八进制数；向右每 3 位产生一个八进制数字，不足 3 位的右边补 0，就得到小数部分的八进制数。同理，向左每 4 位产生一个十六进制数字，不足 4 位的左边补零，就得到整数部分的十六进制数；向右每 4 位产生一个十六进制数字，不足 4 位的右边补 0，就得到小数部分的十六进制数。

例如：$(01111100.1001001)_2=(174.444)_8=(7\mathrm{C}.92)_{16}$

八进制数要转换成二进制数，只需将八进制数分别用对应的三位二进制数表示即可；十六进制数要转换成二进制数，只需将十六进制数分别用对应的四位二进制数表示即可。

为了便于区别不同数制表示的数，规定在数字后面用一个 H 表示十六进制数，用 B 表示二进制数，用 D(或不加标志)表示十进制数，如 64H、1101B、369D 分别表示十六进制数、

二进制数和十进制数。另外，规定当十六进制数以字母开头时，为了避免与其他字符相混，在书写时前面加一个数0，如十六进制数B9H，应写成0B9H。

1.1.2　带符号数的表示

1. 机器数与真值

日常生活中遇到的数，除了上述无符号数外，还有带符号数。对于带符号的二进制数，其正负符号如何表示呢？在计算机中，为了区别正数和负数，通常用二进制数的最高位表示数的符号，对于一个字节型二进制数来说，D_7 位为符号位，D_6～D_0 位为数值位。在符号位中，规定用“0”表示正，“1”表示负，而数值位表示该数的数值大小。把一个数及其符号位在机器中的一组二进制数表示形式，称为“机器数”，机器数所表示的值称为该机器数的“真值”。

2. 机器数的表示方法

机器数可以用不同的表示方法，常用的有原码表示法、反码表示法和补码表示法。

1）原码表示法

最高位为符号位（正数为0，负数为1），其余数字位表示数的绝对值。

例如，当机器字长为8时：

$[+0]_{原}=00000000B$　　$[-0]_{原}=10000000B$

$[+4]_{原}=00000100B$　　$[-4]_{原}=10000100B$

$[+127]_{原}=01111111B$　　$[-127]_{原}=11111111B$

注意：

① “0”的原码有两种表示法：00000000表示+0，10000000表示-0。

② 若微机字长为8位，则原码的表示范围为-127～+127；若字长为16位，则原码的表示范围为-32767～+32767。

原码表示法简单直观，且与真值的转换很方便，但不便于在计算机中进行加减运算。如进行两数相加，必须先判断两个数的符号是否相同。如果相同，则进行加法运算，否则进行减法运算。如进行两数相减，必须比较两数的绝对值大小，再由大数减小数，结果的符号要和绝对值大的数的符号一致。按上述运算方法设计的算术运算电路很复杂。因此，计算机中通常使用补码进行加减运算，这样就引入了反码表示法和补码表示法。

2）反码表示法

正数的反码与其原码相同，负数的反码是在原码基础上，符号位不变（仍为1），数值位则按位取反。例如，当机器字长为8时：

$[+0]_{反}=[+0]_{原}=00000000B$　　$[-0]_{反}=11111111B$

$[+4]_{反}=[+4]_{原}=00000100B$　　$[-4]_{反}=11111011B$

$[+127]_{反}=[+127]_{原}=01111111B$　　$[-127]_{反}=10000000B$

注意：

① “0”的反码有两种表示法：00000000表示+0，11111111表示-0。

② 若微机字长为8位，则反码的表示范围为-127～+127；若字长为16位，则反码的表示范围为-32767～+32767。

3）补码表示法

正数的补码与其原码相同，负数的补码是在原码基础上，符号位不变（仍为1），数值位

则按位取反再加一。例如,当机器字长为8时:

$[+0]_{补}=[+0]_{原}=00000000B$　　$[-0]_{补}=[-0]_{反}+1=00000000B$

$[+4]_{补}=[+4]_{原}=00000100B$　　$[-4]_{补}=[-4]_{反}+1=11111100B$

$[+127]_{补}=[+127]_{原}=01111111B$　　$[-127]_{补}=[-127]_{反}+1=10000001B$

注意:

① $[+0]_{补}=[-0]_{补}=00000000$,无+0和-0之分。

② 正因为补码中没有+0和-0之分,所以8位二进制补码所能表示的数值范围为-128～+127;16位二进制补码所能表示的数值范围为-32768～+32767。

3. 机器数与真值之间的转换

1) 原码转换为真值

根据原码定义,将原码数值位各位按权展开求和,由符号位决定数的正负,即可由原码求出真值。

例 1.6　已知$[X]_{原}=00011111B$,$[Y]_{原}=10011101B$,求X和Y。

解:

$$X=0\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0=31$$

$$Y=-(0\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+0\times2^1+1\times2^0)=-29$$

2) 补码转换为真值

求补码的真值,先求出补码对应的原码,再按原码转换为真值的方法即可求出其真值。正数的原码与补码相同。负数的原码可在补码基础上再次求补。

例 1.7　已知$[X]_{补}=00001111B$,$[Y]_{补}=11100101B$,求X和Y。

解:

$$[X]_{原}=[X]_{补}=00001111B,\quad X=15$$

$$[Y]_{原}=[[Y]_{补}]_{补}=10011011B,\quad Y=-27$$

4. 补码的加减运算

在计算机中,凡是带符号的数一律用补码表示,运算结果自然也是补码。其运算特点是:符号位和数值位一起参加运算,并且自动获得结果(包括符号位与数值位)。

补码加法的运算规则为:$[X+Y]_{补}=[X]_{补}+[Y]_{补}$

补码减法的运算规则为:$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$

当运算结果不超出补码所表示的范围时,运算结果是正确的补码形式。但当运算结果超出补码表示范围时,结果就不正确了,这种情况称为溢出。当最高位向更高位的进位由于机器字长的限制而自动丢失时,并不会影响运算结果的正确性。

计算机中带符号数用补码表示时有如下优点:

① 可以将减法运算变为加法运算,因此可使用同一个运算器实现加法和减法运算,简化了电路;

② 无符号数和带符号数的加法运算可以用同一个加法器实现,简化了微机内部的电路结构。

5. 溢出及其判断方法

1) 进位与溢出

所谓进位是指运算结果的最高位向更高位的进位,用来判断无符号数运算结果是否超

出计算机所能表示的最大无符号数的范围。

溢出是指带符号数的补码运算溢出，用来判断带符号数补码运算结果是否超出了补码所能表示的范围。例如，字长为 n 位的带符号数，它能表示的补码范围为 $-2^{n-1}\sim+2^{n-1}-1$，如果运算结果超出了此范围，就称为补码溢出，简称溢出。

2）溢出的判断方法

判断溢出常见的方法有：①直接由参加运算的两个数的符号及运算结果的符号进行判断；②通过符号位和数值部分最高位的进位状态来判断。第一种方法适用于手工运算时对结果是否溢出的判断，第二种方法通常在计算机中使用。

设符号位进位状态用 CF 来表示，当符号位向前有进位时，CF＝1，否则 CF＝0；数值部分最高位的进位状态用 DF 来表示，当该位向前有进位时，DF＝1，否则 DF＝0；溢出标志位 OF＝ CF ⊕ DF，若 OF＝1，则说明结果溢出，OF＝0，说明结果未溢出。也就是说，当符号位和数值部分最高位同时有进位或同时没有进位时，运算结果不溢出；否则结果溢出。

例 1.8　设有两个操作数 x＝01000100B，y＝01001000B，将这两个操作数送运算器做加法运算：①若为无符号数，计算结果是否正确？②若为带符号补码数，计算结果是否溢出？

解：① 若为无符号数，由于 CF＝0，说明结果未超出 8 位无符号数所能表达的数值范围（0～255），计算结果 10001100B 为无符号数，其真值为 140，计算结果正确。

② 若为带符号补码数，由于 OF＝1，表明结果溢出；也可通过参加运算的两个数的符号及运算结果的符号进行判断。由于两操作数均为正数，而结果却为负数，所以结果溢出；＋68 和＋72 两数补码之和应为＋140 的补码，而 8 位带符号数补码所能表达的数值范围为－128～＋127，结果超出该范围，因此结果是错误的。

1.1.3　定点数与浮点数

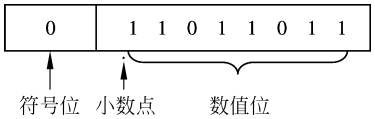
在一般书写中，小数点是用记号“.”来表示的，但在计算机中表示任何信息只能用 0 或 1 两种数码。如果计算机中的小数点用数码表示的话，则不易与二进制数位区分开，所以在计算机中小数点不能用记号“.”表示，那么在计算机中小数点又如何确定呢？

为了确定小数点的位置，在计算机中，数的表示有两种方法，即定点表示法和浮点表示法。

1. 定点表示法

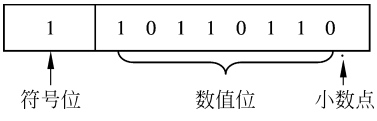
所谓定点表示法，是指在计算机中所有数的小数点的位置人为约定固定不变。一般来说，小数点可约定固定在任何数位之后，但常用下列两种形式：

① 定点纯小数，约定小数点位置固定在符号之后，如＋0.11011011 在机内表示为



假设字长为 n 时，定点小数表示范围为 $1-2^{n-1}\sim-(1-2^{n-1})$。

② 定点纯整数，约定小数点位置固定在最低数值位之后，如－10110110 在机内表示为

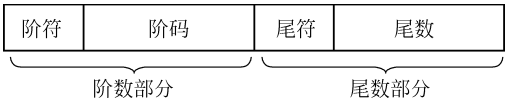


假设字长为 n 时,定点小数表示范围为$-(2^{n-1}-1)\sim 2^{n-1}-1$。

显然,定点数表示法使计算机只能处理纯整数或纯小数,限制了计算机处理数据的范围。为了使得计算机能够处理任意数,事先要将参加运算的数乘上一个“比例因子”,转化成纯小数或纯整数后进行运算,运算结果再除以“比例因子”还原成实际数值。比例因子要取得合适,使参加运算的数、运算的中间结果以及最后结果都在该定点数所能表示的数值范围之内。

2. 浮点表示法

在浮点表示法中,小数点的位置是浮动的。为了使小数点可以自由浮动,浮点数由两部分组成,即尾数部分与阶数部分。浮点数在机器中的表示方法如下:



其中,尾数部分表示浮点数的全部有效数字,它是一个有符号位的纯小数;阶数部分指明了浮点数实际小数点的位置与尾数(定点纯小数)约定的小数点位置之间的位移量 P,该位移量 P(阶数)是一个有符号位的纯整数。尾符则决定了整个数的正负。

当阶数为$+P$ 时,则表示小数点向右移动 P 位;当阶数为$-P$ 时,则表示小数点向左移动 P 位。因此,浮点数的小数点随着 P 的符号和大小而自由浮动。

任意一个二进制数总可以表示为纯小数(或纯整数)和一个 2 的整数次幂的乘积。例如,任意一个二进制数 N 可写成

$$N=S\times 2^{P} \tag{1.2}$$

式中,S 称为数 N 的尾数,P 称为数 N 的阶码,P、S 都是用二进制表示的数。尾数 S 表示了数 N 的全部有效数字,显然 S 采用的数位越多,则数 N 表示的数值精确度越高。阶码 P 指明了数 N 的小数点的位置,显然 P 采用的数位越多,则数 N 表示的数值范围就越大。

如假定 $P=0$,此时,$N=S\times 2^{0}=S$。若尾数 S 为纯小数,这时数 N 为定点纯小数。

如假定 $P=0$,此时若尾数 S 为纯整数,则数 N 为定点纯整数。

如假定 $P=$任意整数,此时,数 N 需要尾数 S 和阶数 P 两部分共同表示,即数 N 为浮点数。

显然,浮点数表示的数值范围比定点数表示的数值范围大得多。设浮点数的阶数位数为 $m+1$ 位,尾数的位数为 $n+1$ 位,则浮点数的取值范围为

$$2^{-n}\cdot 2^{-(2^{m}-1)}<|N|<(1-2^{-n})\cdot 2^{+(2^{m}-1)}$$

虽然浮点数具有表示数值范围大的突出优点,但是,浮点数的运算较为复杂。当计算机进行一次浮点数运算时,需要分别进行两次定点数运算。

例如,设两个浮点数为

$$N_1=S_1\times 2^{P_1}$$

$$N_2=S_2\times 2^{P_2}$$

如 $P_1\neq P_2$,则两数就不能直接相加、减,必须首先对齐小数点(即对阶)后,才能作尾数间的加、减运算。对阶时,小阶向大阶看齐,即把阶小的小数点左移,在计算机中是尾数数码右移,右移 1 位,阶码加 1,直至两数的阶码相同为止,然后两数才能相加减。

浮点数的乘除法，阶码和尾数要分别进行运算。

为了使计算机运算过程中不丢失有效数字，提高运算的精度，一般都采用二进制浮点规格化数。所谓浮点规格化，是指尾数 S 的绝对值小于 1 而大于或等于 1/2，即小数点后面的第一位必须是“1”。

例 1.9　将十进制数 24.125 化为二进制形式的规格化浮点数。

解：① 将该数化为二进制数：

$$24.125 = 11000.001\text{B}$$

② 将此数规格化，则所得规格化浮点数的尾数 $S = +0.11000001$，阶数 $P = +5 = +101$，因此该数的规格化浮点表示为

$$+0.11000001 \times 2^{+101}$$

由于浮点数运算复杂，运算器中除了尾数运算部件外，还有阶码运算部件，控制部件也相应地复杂了，故浮点运算计算机的设备增多，成本较高。

在计算机中，究竟采用浮点制还是定点制，必须根据使用要求设计。目前，一般小型机、微型机多采用定点制，而大型机、巨型机及高档微型机多采用浮点制。

1.1.4　计算机中的编码

由于计算机只能识别二进制数，因此，计算机进行人机交换信息时用到的信息，如数字、字母、符号等都要以特定的二进制编码来表示，这就是信息的编码。

1. 二进制编码的十进制数字

虽然二进制数对计算机来说是最佳的数制，但是人们却不习惯使用它。为了解决这一矛盾，提出了一个比较适合于十进制系统的二进制编码的特殊形式，即将 1 位十进制的 0～9 这 10 个数字分别用 4 位二进制码的组合来表示，在此基础上可按位对任意十进制数进行编码，这就是采用二进制编码的十进制数，简称 BCD(binary-coded decimal)码。

4 位二进制数码有 16 种组合(0000～1111)，原则上可任选其中的 10 个来分别代表十进制中的 0～9 这 10 个数字。我们常用的 BCD 码实际上是指 8421BCD 码，这种编码从 0000～1111 这 16 种组合中选择前 10 个即 0000～1001 来分别代表十进制数码 0～9，8、4、2、1 分别是这种编码从高位到低位每位的权值。BCD 码有两种形式，即压缩型 BCD 码和非压缩型 BCD 码。压缩型 BCD 码用一个字节表示两位十进制数，例如，10000110B 表示十进制数 86D；非压缩型 BCD 码用一个字节表示一位十进制数，高 4 位总是 0000，低 4 位用 0000～1001 中的一种组合来表示 0～9 中的某一个十进制数。表 1.3 给出了 8421BCD 码与十进制数字的编码关系。

BCD 码的优点是与十进制数转换方便，容易阅读；缺点是用 BCD 码表示的十进制数的数位要较纯二进制表示的十进制数位更长，使电路复杂性增加，运算速度减慢。

需要说明的是，虽然 BCD 码可以简化人机联系，但它比纯二进制编码效率低。对同一个给定的十进制数，用 BCD 码表示时需要的位数比用纯二进制码多，而且用 BCD 码进行运算所花的时间也更多，计算过程更复杂。BCD 码是将每个十进制数用一组 4 位二进制数来表示，若将这种 BCD 码送计算机进行运算，由于计算机总是将数当作二进制数而不是当作 BCD 码来运算，所以结果可能出错，因此需要对计算结果进行必要的修正，以得到正确的 BCD 码形式。

表 1.3 8421BCD 码与十进制数的编码关系

十进制数	8421BCD 码	十进制数	8421BCD 码
0	0000	8	1000
1	0001	9	1001
2	0010	10	00010000
3	0011	11	00010001
4	0100	20	00100000
5	0101	45	01000101
6	0110	68	01101000
7	0111	92	10010010

2. 字母与符号的编码

字符是指数字、字母以及其他的一些符号的总称。

现代计算机不仅用于处理数值领域的问题,而且要处理大量非数值领域的问题。这样一来,必然需要计算机能对数字、字母、文字以及其他一些符号进行识别和处理,而计算机只能处理二进制数,因此,通过输入输出设备进行人机交换信息时使用的各种字符也必须按某种规则,用二进制数码 0 和 1 来编码,计算机才能进行识别与处理。

目前国际上使用的字符编码系统有许多种。在微机、通信设备和仪器仪表中广泛使用的是 ASCII(America standard code for information interchange)码,即美国标准信息交换码。ASCII 码用一个字节来表示一个字符,采用 7 位二进制代码来对字符进行编码,最高位一般用作检验位(旧称校验位)。7 位 ASCII 码能表示 $2^7=128$ 种不同的字符,其中包括数码 0～9,英文大、小写字母,标点符号及控制字符等,如表 1.4 所示。

表 1.4 ASCII 码字符表

高 3 位 / 低 4 位	000	001	010	011	100	101	110	111
0000	NUL	DEL	SP	0	@	P	.	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	DOT	DC4	$	4	D	T	d	t
0101	ENG	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	↑	n	～
1111	SI	US	/	?	O	↓	o	DEL