第 1 章

数据表示与数值运算

本章主要介绍数值数据和字符数据在计算机内部的表示方法及数值运算的实现方法，包括各种进制数和相互转换方法及计算机的性能指标。数值数据详细介绍了整数和浮点数（实数）的表示方法及显示方法；字符数据详细介绍了 ASCII 码、机内码、Unicode、UTF-8、点阵字形码、矢量字形码的表示方法及显示方法；校验码详细介绍了奇偶校验码、海明校验码、循环冗余校验码；数值运算详细介绍了定点整数的加减运算和溢出判断方法及硬件实现方法、定点整数的乘除运算和浮点数的加减乘除运算。通过本章的学习，读者应该能完成以下学习任务。

（1）掌握十进制、二进制、十六进制的表示方法及其相互转换方法。

（2）了解八进制，掌握二进制口算方法。

（3）了解计算机的主要性能指标。

（4）掌握无符号整数、有符号整数和移码的表示方法，了解 BCD 码的表示方法。

（5）掌握浮点数（实数）的表示方法。

（6）了解 ASCII 码常用字符的编码规律。

（7）了解机内码和区位码的编码规律，掌握其相互转换方法。

（8）了解 Unicode 和 UTF-8 的编码规律，掌握其相互转换方法。

（9）了解点阵字形码和矢量字形码表示字形的相关原理。

（10）了解码距概念，掌握奇偶校验码、海明校验码、循环冗余校验码的编码方法。

（11）掌握定点整数的加减运算和溢出判断方法，理解硬件实现方法。

（12）掌握定点整数的乘除运算。

（13）掌握浮点数的加减乘除运算。

（14）了解相关理论知识在编程语言中的使用和验证方法。

1.1 计 数 制

计数制是计算机中用来表示数值的方法，有十进制、二进制、八进制、十六进制等。

1.1.1 十进制（decimal）

十进制用 0~9 共 10 个数码表示，基数为 10，运算规则为逢十进一，各位位权是 10^i（整数部分最低位位权为 10^0，其余各位位权从右至左指数依次增大；小数部分最高位位权为 10^{-1}，其余各位位权从左至右指数依次减小）。十进制数后缀字母为 D，一般省略。n 位整

数 m 位小数的十进制数 N 表示如下。

$$N = a_{n-1}a_{n-2}\cdots a_0.a_{-1}\cdots a_{-m}\mathrm{D}$$
$$= a_{n-1}\times 10^{n-1} + a_{n-2}\times 10^{n-2} + \cdots + a_0\times 10^0 + a_{-1}\times 10^{-1} + \cdots + a_{-m}\times 10^{-m}$$

例如：

$$N = 325.46\mathrm{D}$$
$$= 3\times 10^2 + 2\times 10^1 + 5\times 10^0 + 4\times 10^{-1} + 6\times 10^{-2}$$
$$= 325.46$$

1.1.2 二进制（binary）

二进制用 0 和 1 两个数码表示，基数为 2，运算规则为逢二进一，各位位权是 2^i。二进制数后缀字母为 B。n 位整数 m 位小数的二进制数 N 表示如下。

$$N = a_{n-1}a_{n-2}\cdots a_0.a_{-1}\cdots a_{-m}\mathrm{B}$$
$$= a_{n-1}\times 2^{n-1} + a_{n-2}\times 2^{n-2} + \cdots + a_0\times 2^0 + a_{-1}\times 2^{-1} + \cdots + a_{-m}\times 2^{-m}$$

例如：

$$N = 10111101.11\mathrm{B}$$
$$= 1\times 2^7 + 0\times 2^6 + 1\times 2^5 + 1\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2}$$
$$= 1\times 128 + 0\times 64 + 1\times 32 + 1\times 16 + 1\times 8 + 1\times 4 + 0\times 2 + 1\times 1 + 1\times 0.5 + 1\times 0.25$$
$$= 189.75$$

注意

（1）二进制整数部分各位的位权，特别是末 4 位，分别是 8、4、2、1，这是二进制数口算时常用的数值。例如，1111B=8+4+2+1=15。

（2）非十进制数不能读成几百几十，例如 111B 不能读成“一百一十一”，只能读成“一一一 B”，其他非十进制数读法类似。

（3）很多教材用下标数字 2 表示二进制，例如$(10111101.11)_2$，该表示方式不能用在任何源程序中，因此，建议不要使用该表示方式，其他进制也有类似情况。

（4）计算机底层无论是数值还是字符，甚至是音频、视频等信息，最终都是用二进制表示和存储的，只是为了阅读或使用方便，应用软件把它们转换为十进制或十六进制进行表示和显示，因此，我们使用计算机时一般看不到二进制数。

1.1.3 八进制（octal）

八进制用 0~7 共 8 个数码表示，基数为 8，运算规则为逢八进一，各位位权是 8^i。八进制数后缀字母为 O 或 Q。n 位整数 m 位小数的八进制数 N 表示如下。

$$N = a_{n-1}a_{n-2}\cdots a_0.a_{-1}\cdots a_{-m}\mathrm{Q}$$
$$= a_{n-1}\times 8^{n-1} + a_{n-2}\times 8^{n-2} + \cdots + a_0\times 8^0 + a_{-1}\times 8^{-1} + \cdots + a_{-m}\times 8^{-m}$$

例如：

$$N = 377.34\mathrm{Q}$$
$$= 3\times 8^2 + 7\times 8^1 + 7\times 8^0 + 3\times 8^{-1} + 4\times 8^{-2}$$
$$= 255.4375$$

注意

（1）在 C 语言（本书所提到的 C 语言多数指 VC 或 VS 下的 C++）中，八进制数用前缀 0（数字零）表示，例如 0175。

（2）八进制比较少用。

1.1.4　十六进制（hexadecimal）

十六进制用 0~9、A、B、C、D、E、F 共 16 个数码（字母一般不区分大小写）表示。A~F 相当于十进制数的 10~15，十六进制基数为 16，运算规则为逢十六进一，各位位权是 16^i。十六进制数后缀字母为 H。n 位整数 m 位小数的十六进制数 N 表示如下。

$$N = a_{n-1}a_{n-2}\cdots a_0.a_{-1}\cdots a_{-m}\mathrm{H}$$
$$= a_{n-1}\times 16^{n-1} + a_{n-2}\times 16^{n-2} + \cdots + a_0\times 16^0 + a_{-1}\times 16^{-1} + \cdots + a_{-m}\times 16^{-m}$$

例如：

$$N = \mathrm{FA.8H}$$
$$= 15\times 16^1 + 10\times 16^0 + 8\times 16^{-1}$$
$$= 250.5$$

注意

（1）在汇编指令中，若十六进制数的第一个数码是字母，则必须加前缀 0（数字零），以区别于标识符（变量名等）。例如，将常量 FAH 赋值给寄存器 EAX 的指令应写成：MOV EAX,0FAH。平时书写时可不加前缀 0。

（2）在 C 语言和调试器等软件中，十六进制数用前缀 0x（数字零和字母 x）表示，例如 0xFA。

（3）计算机内部数据经常用十六进制表示，且经常没有前缀 0x 或后缀 H。

1.2　进制数间的转换


第 11 讲 2

1.2.1　十进制转二进制

十进制转换为二进制时，若数据包含整数和小数，则整数部分和小数部分要分别转换，再合并。

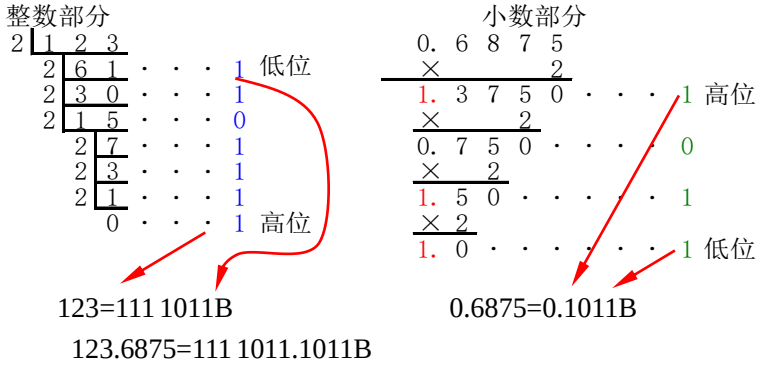
整数和小数转换规则如下。

（1）整数部分：除 2 取余，先低（位）后高（位）。

（2）小数部分：乘 2 取整，先高（位）后低（位）。

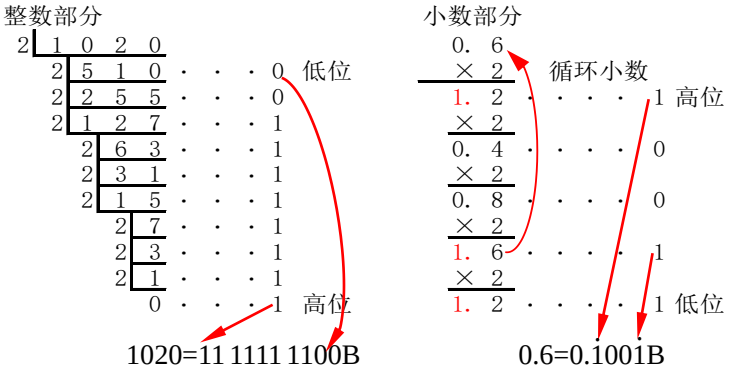
例 1-01　将十进制数 123.6875 转换为二进制数。

首先将十进制数 123.6875 分成整数 123 和小数 0.6875，然后分别将其转换为二进制数，转换过程如下。



123.6875=111 1011.1011B

例 1-02　将十进制数 1020.6 转换为二进制数。



1020.6=11 1111 1100.$\dot{1}00\dot{1}$B

说明

将十进制数 0.6 转换为二进制数，其结果是无限循环小数，默认按就近舍入（round to nearest）的方法处理其尾数，具体规定详见浮点控制寄存器（rounding control，RC）字段的说明。

1.2.2　十进制转八进制和十六进制

十进制转换为八进制或十六进制的规则如下。

（1）整数部分：除八或十六取余，先低后高。

（2）小数部分：乘八或十六取整，先高后低。

由于数字太大，十进制转八进制或十六进制时经常先转换为二进制再转换为八进制或十六进制，规则如下。

整数部分：

（1）将二进制整数从右往左每 3 位一组（不够时左边补 0）分别转换为八进制数。

（2）将二进制整数从右往左每 4 位一组（不够时左边补 0）分别转换为十六进制数。

小数部分：

（1）将二进制小数从左往右每 3 位一组（不够时右边补 0）分别转换为八进制数。

（2）将二进制小数从左往右每 4 位一组（不够时右边补 0）分别转换为十六进制数。

例 1-03　将 1111111100.1001B 转换为八进制数和十六进制数。

1111111100.1001B=001 111 111 100.100 100B

=1774.44Q

1111111100.1001B=0011 1111 1100.1001B

=3FC.9H

说明

相传上古伏羲用 8 个符号来表示先天八卦，对应 8 种特定意义，卦序、卦名、卦象分别为：一乾☰、二兑☱、三离☲、四震☳、五巽☴、六坎☵、七艮☶、八坤☷。卦象中“⚊”代表阳爻，类似二进制数 0；“⚋”代表阴爻，类似二进制数 1。每一个卦象的三爻类似一组三位二进制数。例如，震卦☳从下往上对应二进制 011B，其他类似。1679 年莱布尼茨提出二进制，1951 年二进制电子计算机 EDVAC 诞生。

1.2.3　十进制转二进制加法口算

首先，二进制数各位的位权 2^n 与十进制数的对应关系如下。

$2^0=1$	$2^4=16$	$2^8=256$	$2^{12}=4096$	$2^{16}=65536$
$2^1=2$	$2^5=32$	$2^9=512$	$2^{13}=8192$	
$2^2=4$	$2^6=64$	$2^{10}=1024$	$2^{14}=16384$	
$2^3=8$	$2^7=128$	$2^{11}=2048$	$2^{15}=32768$	

其次，观察以下两种进制数的规律。

$10^1=10$	$2^1=10\text{B}$
$10^2=100$	$2^2=100\text{B}$
$10^3=1000$	$2^3=1000\text{B}$
…	…
$10^n=10\cdots0$（n 个 0）	$2^n=10\cdots0\text{B}$（n 个 0）

由此可得二进制加法口算方法：2^n 由 1 个 1 和 n 个 0 构成的二进制数组成，即：

$$2^n=10\overbrace{\cdots}^{n}0\text{B}$$

因此，比该数略大的数只要将若干相应位权的 0 改为 1 即可。例如：

$$\because 128=2^7=1\overbrace{000\ 0000}^{7}\text{B}$$

$$\therefore 130=128+2=2^7+2=1000\ 0000\text{B}+10\text{B}=1000\ 0010\text{B}$$

相当于将 1000 0000B 倒数第 2 位的 0 改为 1。

$$\because 256=2^8=1\overbrace{0000\ 0000}^{8}\text{B}$$

$$\therefore 261=256+5=2^8+(4+1)=1\ 0000\ 0000\text{B}+101\text{B}=1\ 0000\ 0101\text{B}$$

相当于将 1 0000 0000B 倒数第 1、3 位的 0 改为 1。

同理：

$$525=512+13=2^9+(8+4+1)=10\ 0000\ 0000\text{B}+1101\text{B}=10\ 0000\ 1101\text{B}$$

$$1030=1024+6=2^{10}+(4+2)=100\ 0000\ 0000\text{B}+110\text{B}=100\ 0000\ 0110\text{B}$$

1.2.4 十进制转二进制减法口算

首先，观察以下两种进制数的规律。

$10^1-1=9$	$2^1-1=1B$
$10^2-1=99$	$2^2-1=11B$
$10^3-1=999$	$2^3-1=111B$
…	…
$10^n-1=9\cdots9$（n 个 9）	$2^n-1=1\cdots1B$（n 个 1）

由此可得二进制减法口算方法：2^n-1 由 n 个 1 构成的二进制数组成，即：

$$2^n-1=\overset{n}{1\cdots1}B$$

因此，比该数略小的数只要将若干相应位权的 1 改为 0 即可。例如：

$$\because 127=2^7-1=\overbrace{111\ 1111}^{7}B$$

$$\therefore 120=127-7=(2^7-1)-(4+2+1)=111\ 1111B-111B=111\ 1000B$$

$$\because 255=2^8-1=\overbrace{1111\ 1111}^{8}B$$

$$\therefore 250=255-5=(2^8-1)-(4+1)=1111\ 1111B-101B=1111\ 1010B$$

同理：

$$505=511-6=(2^9-1)-(4+2)=1\ 1111\ 1111B-110B=1\ 1111\ 1001B$$

$$1020=1023-3=(2^{10}-1)-(2+1)=11\ 1111\ 1111B-11B=11\ 1111\ 1100B$$

1.2.5 十进制转二进制其他口算

（1）乘法口算方法。类似数学的科学记数法，若一个大整数 N 和一个小整数 a 可以表示成 $N=a\times2^n$，要将 N 转换成二进制数，则可先将小整数 a 转换成二进制数，然后在右边补 n 个 0 即可，相当于将其左移 n 位或将小数点右移 n 位。例如：

$$40=10\times2^2=(8+2)\times2^2=1010B\times2^2=1010\ 00B$$

$$88=11\times2^3=(8+2+1)\times2^3=1011B\times2^3=1011\ 000B$$

（2）除法口算方法。若一个分数 F 和一个整数 a 可以表示成 $F=a/2^n$，要将 F 转换成二进制数，则可先将整数 a 转换成二进制数，然后将其右移 n 位或将小数点左移 n 位。例如：

$$11/16=(8+2+1)/2^4=1011B\times2^{-4}=0.1011B$$

$$23/64=(16+4+2+1)/2^6=10111B\times2^{-6}=0.010111B$$

（3）其他口算。若整数 $N=a\times16+b$，即 16 的整数 a 倍加零头 b，则可先将其转换成十六进制数再转换成二进制数。例如：

$$161=10\times16^1+1\times16^0=0A1H=1010\ 0001B$$

$$176=11\times16^1+0\times16^0=0B0H=1011\ 0000B$$

1.3 计算机的性能指标

要衡量一台计算机的性能，需要考虑多项不同的指标，主要有字长、存储容量、主频、

运算速度、可靠性、系统可维护性等。

1.3.1　字长

计算机 CPU 在同一时刻最多能实现的算术逻辑运算的二进制位数称为字长，例如 80386/80486 CPU 的字长为 32 位，80586 CPU 的字长为 64 位。32 位系统对应的寄存器、数据总线一般是 32 位的，而 64 位系统对应的寄存器、数据总线一般是 64 位的。字长影响着计算机的运算精度。

原则上，32 位系统的一个字指 32 位，但习惯上，一个字（WORD）指 16 位，双字（DWORD）指 32 位。

1.3.2　存储容量

影响计算机性能的存储容量主要指主（内）存容量。计算机中当前正在运行的程序和数据都要存放在内存中，因此，存储容量决定着计算机在同一时间内所能运行的程序的数量和大小。

存储容量常用单位及其换算关系如下。

1 字节（Byte）=8 比特（二进制位，bit）

$1024B=2^{10}B=1KB\approx10^{3}B$	（K-Kilo）	$1024TB=2^{50}B=1PB\approx10^{15}B$	（P-Peta）
$1024KB=2^{20}B=1MB\approx10^{6}B$	（M-Mega）	$1024PB=2^{60}B=1EB\approx10^{18}B$	（E-Exa）
$1024MB=2^{30}B=1GB\approx10^{9}B$	（G-Giga）	$1024EB=2^{70}B=1ZB\approx10^{21}B$	（Z-Zetta）
$1024GB=2^{40}B=1TB\approx10^{12}B$	（T-Tera）	$1024ZB=2^{80}B=1YB\approx10^{24}B$	（Y-Yotta）

现在个人计算机的主（内）存容量一般为 8~32GB，辅存容量则可达到 TB 级，而有些大数据公司，其每天的数据处理量就达到 100PB。

1.3.3　主频

主频是计算机 CPU 在单位时间内输出的脉冲数。主频决定着计算机在单位时间内所能产生的操作数量或所能运行的指令数量，因此，早期计算机的主频是决定计算机运算速度的一个重要指标。其最小单位为赫兹（Hz），早期单位为 MHz，现在常用单位为 GHz。

主频一个脉冲的时间称为时钟周期，常用单位及其换算关系如下。

$1ms=10^{-3}s$	（m-mili）	$1ps=10^{-3}ns=10^{-12}s$	（p-pico）
$1\mu s=10^{-3}ms=10^{-6}s$	（μ-micro）	$1fs=10^{-3}ps=10^{-15}s$	（f-femto）
$1ns=10^{-3}\mu s=10^{-9}s$	（n-nano）	$1as=10^{-3}fs=10^{-18}s$	（a-atto）

现在一般个人计算机的时钟周期为 ns 级，即 CPU 主频一般为 GHz。

1.3.4　运算速度

运算速度一般指 CPU 在单位时间内执行的指令条数，常用单位为 MIPS（million instructions per second，每秒百万条指令）或 MFLOPS（million floating-point operations per second，每秒百万个浮点操作）。

1.3.5　可靠性

可靠性指标主要指平均无故障时间（MTBF）。若 t_i 为第 i 次无故障间隔时间，N 为故

障数，则：

$$\mathrm{MTBF}=\sum_{i=1}^{N}t_i / N$$

因此，MTBF 越大越好。

1.3.6 系统可维护性

系统可维护性是指发生故障后能尽快恢复正常的程度，常用平均修复时间（MTTR）表示。

$$\mathrm{MTTR}=\sum_{i=1}^{M}T_i / M$$

其中，T_i 为从第 i 次故障开始至投入运行的时间，M 为修复总次数。

因此，MTTR 越小越好。

其他指标还有兼容性、性价比等。

1.4 数值的表示

计算机中所表示的数值数据有整数和浮点数（实数），整数包括无符号整数和有符号整数，浮点数包括单精度、双精度、扩展精度 3 种。无论是何种类型数据，最终都用二进制表示和存储，但为了方便阅读，一般用十进制显示其真值，用十六进制表示其机器数。

1.4.1 无符号整数的表示

无符号整数本质上就是机器数，因为不存在符号占用一位或两位二进制数，所以存储的内容就是整数本身，只需将其数值直接转换为二进制数即可。当然，为了方便阅读，也可以用十进制或十六进制表示，只是使用时要注意可表示数的范围，超出范围时权值高的二进制位将被舍去，或者编译时出错并提示数据太大。n 位二进制数可表示的无符号整数的范围是 $0\sim2^n-1$，n 的取值由数据类型决定。例如 BYTE 类型为 8 位，不同数据类型表示范围如表 1-1 所示。

表 1-1 无符号整数表示范围

类　型	位　数	表示范围	备　注
BYTE（或 DB）	8	0~255（2^8-1）	用于字符（串），类似 C 程序中的 unsigned char
WORD（或 DW）	16	0~65535（$2^{16}-1$）	常用于汉字，类似 C 程序中的 unsigned short= wchar_t
DWORD（或 DD）	32	0~4294967295（$2^{32}-1$）	用于整数，类似 C 程序中的 unsigned int
QWORD（或 DQ）	64	0~18446744073709551615（$2^{64}-1$）或$-1.79\times10^{308}\sim1.79\times10^{308}$	用于大整数或浮点数，类似 C 程序中的__int64 或 double，做大整数时用%I64u 输入输出，做浮点数时用%lf 输入输出

注意

表 1-1 中的各数据类型一般只决定数据的位数，至于是否按无符号数处理由具体的汇编指令决定。例如，若.IF、.WHILE 等伪指令条件表达式中出现无符号类型变量，则表达式按无符号比较；若无符号变量比较之后用 Jcc 指令判断，则是否按无符号处理要看所用的转移指令。

在 C 程序中，无符号整数可以用任意进制数给无符号变量赋值，甚至用有符号数给无符号变量赋值，编译器会自动将其转换成相应的机器数再存入变量，具体所表示的十进制数值可以用%u 输出。

例 1-04　用 C 程序输出 250、−6、0FAH、11111010B、372Q 对应的 8 位无符号数的值。这里用 C 程序嵌入汇编指令 MOV 对变量进行赋值，相关知识见后续章节。

源程序如下：

例 1-04

```
#include "stdio.h"
void main(int argc, char* argv[])
{
    unsigned char a,b,c,d,e;
    __asm
    {
        MOV   a,250
        MOV   b,-6
        MOV   c,0FAH
        MOV   d,11111010B
        MOV   e,372Q
    }
    printf("250、−6、0FAH、11111010B、372Q 对应无符号数为%u、%u、%u、%u、%u\n",a,b,c,d,e);
}
```

输出结果为：

```
250、−6、0FAH、11111010B、372Q 对应无符号数为 250、250、250、250、250
```

8 位无符号数按 32 位整数输出过程如图 1-1 所示。

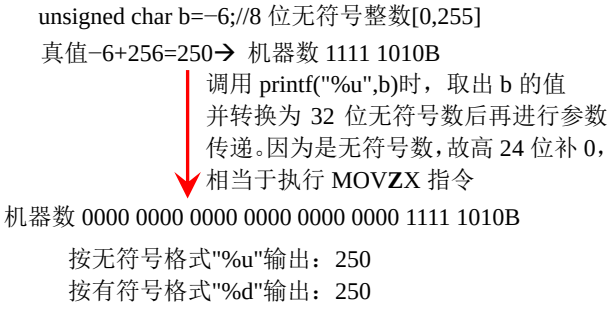


图 1-1　8 位无符号数按 32 位整数输出过程

由以上运行结果可知，若机器数对应的 8 位无符号整数的数值不在[0,255]区间中，则可通过加或减 2^8（=256）进行调整，即可得到相应的无符号整数。

1.4.2 有符号整数的表示（补码等）

计算机为表示整数的符号，约定最高位为 0 表示正数，最高位为 1 表示负数。

有符号整数的常用编码有原码、反码、补码，一般用补码。例如，C 语言、Python 等编程语言都用补码表示有符号整数。

n 位补码的取值范围是-2^{n-1}~$+2^{n-1}-1$，n 的取值由数据类型决定。例如，SBYTE 类型为 8 位，不同数据类型表示范围如表 1-2 所示。

表 1-2 有符号整数表示范围

类 型	位 数	表 示 范 围	备 注
SBYTE（或 DB）	8	−128~+127，即-2^7~$+2^7-1$	用于字符（串），类似 C 程序中的 char
SWORD（或 DW）	16	−32768~+32767，即-2^{15}~$+2^{15}-1$	用于汉字，类似 C 程序中的 short
SDWORD（或 DD）	32	−2147483648~+2147483647，即-2^{31}~$+2^{31}-1$	用于整数，类似 C 程序中的 int
QWORD（或 DQ）	64	−9223372036854775808~9223372036854775807，即-2^{63}~$+2^{63}-1$ 或-1.79×10^{308}~1.79×10^{308}	作用同表 1-1，只是做大整数时用%I64d 输入输出

有符号整数 X 用 n 位二进制数表示的补码的计算公式如下。

$$[X]_{补码}=2^n+X$$

例如，求−6 用 8 位二进制数表示的补码的计算过程如下。

$$[-6]_{补码}=2^8+(-6)=250$$

这里，真值−6 的补码在数值上等于机器数 250，至于 250 要表示成什么进制、以什么形式表示，可根据需要和使用环境决定。例如，在汇编语言中可以表示成 0FAH、11111010B 等。

下面观察 ±1 在不同二进制位数时的补码表示。

当 n=8 时，即 SBYTE 类型 ±1 的补码表示如下（超过 8 位部分即高位 1 被舍去）。

$[+1]_{补码}=2^8+(+1)=256+1=0\ 000\ 0001B$

$[-1]_{补码}=2^8+(-1)=256-1=1\ 111\ 1111B$

当 n=16 时，即 SWORD 类型 ±1 的补码表示如下。

$[+1]_{补码}=2^{16}+(+1)=65536+1=0\ 000\ 0000\ 0000\ 0001B$

$[-1]_{补码}=2^{16}+(-1)=65536-1=1\ 111\ 1111\ 1111\ 1111B$

当 n=32 时，即 SDWORD 类型 ±1 的补码表示如下。

$[+1]_{补码}=2^{32}+(+1)=4294967296+1=0\ 000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001B$

$[-1]_{补码}=2^{32}+(-1)=4294967296-1=1\ 111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111B$

对于十进制数，左边补多少个 0，其数值都不变；对于补码，左边补多少个符号，其数值都不变，这就是符号扩展，即补码在不改变其符号的前提下，其符号位可以任意增删而不改变其真值。

在使用有符号整数除法指令 IDIV 时，用 CDQ 指令实现将 EAX 寄存器中 32 位有符号数扩展为 64 位有符号数并存于 EDX 和 EAX 两个寄存器中（CDQ: Convert DWORD(EAX)→QWORD(EDX|EAX)），详见后续章节。

由于补码计算公式不方便计算负数，因此常用原码和反码推出补码。

原码、反码、补码都用最高位表示符号，0 表示正数，1 表示负数，其余 $n-1$ 位表示数值。

n 位原码直接用 $n-1$ 位表示数值。8 位原码表示的范围如下。

最小正数[+0]$_{原码}$=0000 0000B=00H=0
最大正数[+127]$_{原码}$=0111 1111B=7FH=127　　正数 128 个编码
最大负数[−0]$_{原码}$=1000 0000B=80H=128
最小负数[−127]$_{原码}$=1111 1111B=FFH=255　　负数 128 个编码

256 个编码表示 255 个数
+0 和−0 各占一个编码
真值范围是[−127,+127]

n 位反码的正数同原码，负数将原码的数值位按位取反（符号位不变）。8 位反码表示的范围如下。

最小正数[+0]$_{反码}$=0000 0000B=00H=0
最大正数[+127]$_{反码}$=0111 1111B=7FH=127　　正数 128 个编码
最大负数[−0]$_{反码}$=1111 1111B=FFH=255
最小负数[−127]$_{反码}$=1000 0000B=80H=128　　负数 128 个编码

256 个编码表示 255 个数
+0 和−0 各占一个编码
真值范围是[−127,+127]

n 位补码的正数同原码，负数由相应的反码末位加 1 得到。8 位补码表示的范围如下。

最小正数[+0]$_{补码}$=0000 0000B=00H=0
最大正数[+127]$_{补码}$=0111 1111B=7FH=127　　正数 128 个编码
最小正数[−0]$_{补码}$=0000 0000B=00H=0
最大负数[−1]$_{补码}$=1111 1111B=FFH=255
[−127]$_{补码}$=1000 0001B=81H=129　　负数 128 个编码
最小负数[−128]$_{补码}$=1000 0000B=80H=128

256 个编码表示 256 个数
+0 和−0 只占一个编码
真值范围是[−128,+127]

用补码计算公式[X]$_{补码}$=2^n+X 求−128 的补码，可以验证如下。结果与上面的计算结果相同。

[−128]$_{补码}$=2^8+(−128)=256−128=128=1000 0000B

由于正数的最高位为 0，负数的最高位为 1，所以各类型的取值范围如下。

SBYTE（或 DB）类型：正数为 00H~7FH，负数为 80H~FFH。

SWORD（或 DW）类型：正数为 0000H~7FFFH，负数为 8000H~FFFFH。

SDWORD（或 DD）类型：正数为 00000000H~7FFFFFFFH，负数为 80000000H~FFFFFFFFH。

在 C 程序中，有符号整数同样可以用任意进制数给有符号变量赋值，甚至用无符号数给有符号变量赋值，编译器会自动将其转换成相应的机器数再存入变量，具体所表示的十进制数值可以用%d 输出。

例 1-05　用 C 程序输出 250、−6、0FAH、11111010B、372Q 对应的 8 位有符号数的值。

源程序如下：


例 1-05

```
#include "stdio.h"
void main(int argc, char* argv[])
{
    char a,b,c,d,e;
    __asm
    {
```

```
        MOV    a,250
        MOV    b,-6
        MOV    c,0FAH
        MOV    d,11111010B
        MOV    e,372Q
    }
    printf("250、−6、0FAH、11111010B、372Q 对应有符号数为%d、%d、%d、%d、%d\n",a,b,c,d,e);
}
```

输出结果为：

```
250、−6、0FAH、11111010B、372Q 对应有符号数为−6、−6、−6、−6、−6
```

8 位有符号数按 32 位整数输出过程如图 1-2 所示。

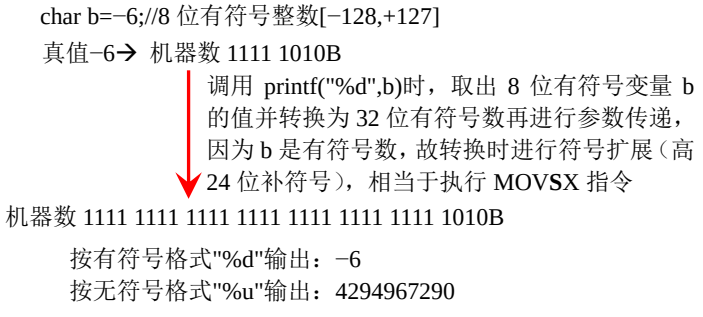


图 1-2　8 位有符号数按 32 位整数输出过程

由以上运行结果可知，若机器数对应的 8 位有符号数的数值不在[−128,127]区间中，则可通过加或减 2^8（=256）进行调整，即可得到相应的有符号数。

1.4.3　移码

用补码表示整数，可以很好地实现有符号整数的加减运算（可通过后续章节进行学习），但也存在一个问题——整数的真值不是随着机器数（即补码）的增大而增大。当机器数最小时，其对应的真值不是最小值；当机器数最大时，其对应的真值不是最大值。如图 1-3 所示，用补码表示 SBYTE 类型整数，当机器数最小时是 00000000B，其对应的真值是 0，不是最小值；当机器数最大时是 11111111B，其对应的真值是−1，不是最大值。

用原码表示整数也有类似问题。

在浮点数的指数表示中，希望能够得到一种编码，让真值随机器数单调递增，这样，可以直接通过机器数的大小关系确定真值的大小关系，以便于比较浮点数的大小。虽然无符号整数是随机器数单调递增的，但是无符号整数真值=机器数$\in[0,2^n-1]$，没有负数，不能作为指数。为了得到负数，可以将该函数关系向下平移一个偏置量，得到：

$$真值=机器数-偏置量$$

这样，真值随机器数单调递增，同时又有正负数。由上式可得：

$$机器数=真值+偏置量$$

8 位补码、移码机器数与真值关系如图 1-3 所示。

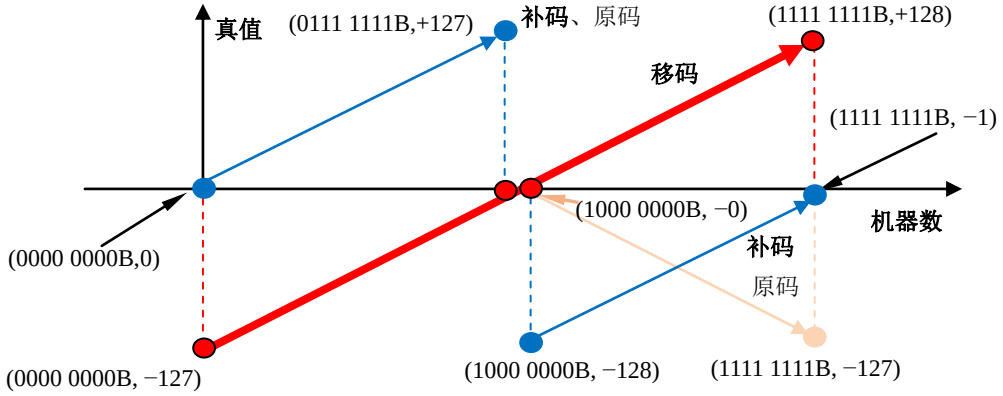


图 1-3　8 位补码、移码机器数与真值关系示意图

若真值为 X，则机器数=X+偏置量，由于这个编码（即机器数）是通过真值平移产生的，所以称为移码，即：

$$[X]_{移码}=X+偏置值$$

IEEE 754 用作浮点数指数的移码偏置值为 $2^{n-1}-1$，其中 n 为移码的位数。例如，单精度浮点数使用 8 位移码，偏置值为 $2^{n-1}-1=2^{8-1}-1=127$，其计算公式如下：

$$[X]_{移码}=X+(2^{8-1}-1)=X+127$$

例如：

$[-127]_{移码}=-127+127=0=0000\ 0000B$

$[+128]_{移码}=+128+127=255=1111\ 1111B$

由此可见，对于 8 位移码，当真值最小（−127）时，对应移码机器数最小（0000 0000B）；当真值最大（+128）时，对应移码机器数最大（1111 1111B）。

双精度浮点数使用 11 位移码，偏置值为 $2^{n-1}-1=2^{11-1}-1=1023$；扩展精度浮点数使用 15 位移码，偏置值为 $2^{n-1}-1=2^{15-1}-1=16383$。

1.4.4　BCD 码

BCD 码（binary-coded decimal）又称二−十进制编码，即用 4 位二进制数来表示 1 位十进制数的编码。4 位二进制数各位的位权分别为 8、4、2、1 的 BCD 码称为 8421BCD 码。

两位 BCD 码存于一个字节的称为压缩 BCD 码。例如，十进制数 23 的压缩 BCD 码为 0010 0011B，即 23H。

每位 BCD 码存于一个字节的称为非压缩 BCD 码（每字节高 4 位补 0）。例如，十进制数 23 的非压缩 BCD 码为 0000 0010 0000 0011B，即 0203H。

BCD 码与二进制数编码不同。例如，十进制数 23 的二进制数编码为 0001 0111B，即 17H，汇编指令 AAM 可以实现将此二进制数编码转换为 BCD 码。

1.4.5　浮点数

在数学中，一个浮点数 N 可以用科学记数法表示如下。

$$N=a\times10^{n}$$

其中，|*a*|为[1,10)的有效数字，*n* 为整数表示的指数，例如 2.5×10^{13}。

在计算机中，浮点数用类似于有效数字的尾数和类似于指数的阶码表示，对应的真值如下。

$$浮点数真值（N）=尾数真值（m）\times2^{阶码真值（e）}$$

一般，尾数真值（*m*）的绝对值为[1,2)的有效数字，阶码真值（*e*）用移码表示的机器数（*E*）减去偏置量作为指数，例如+12.0=$+1.5\times2^{+3}$。

汇编语言浮点数使用 IEEE 754 规范，有单精度（REAL4）、双精度（REAL8）和扩展精度（REAL10）共 3 种类型，分别用于定义 4 字节、8 字节和 10 字节的浮点数变量（同 DWORD、QWORD、TBYTE 或 DD、DQ、DT 3 种类型）。浮点数的机器数格式如图 1-4 所示，其中尾数分为尾数符号（*S*）和尾数数值（*M*）两部分。*S* 用 1 表示负数，用 0 表示正数；*M* 用原码表示；阶码（*E*）用移码表示。

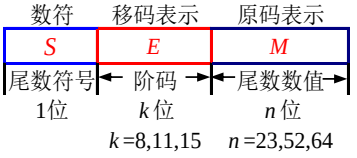
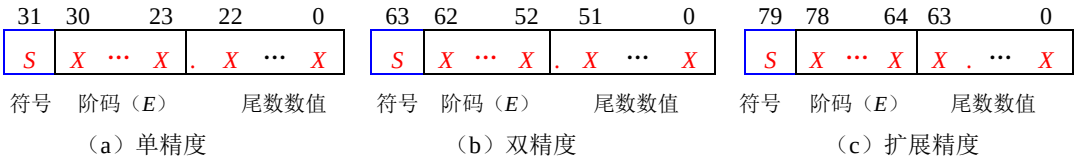


图 1-4　IEEE 754 规范中浮点数的机器数格式

单精度时，用 8 位表示阶码，23 位表示尾数数值；双精度时，用 11 位表示阶码，52 位表示尾数数值；扩展精度时，用 15 位表示阶码，64 位表示尾数数值。具体格式如图 1-5 所示。



（a）单精度　　（b）双精度　　（c）扩展精度

图 1-5　3 种浮点数的机器数格式

注意

其中小数点是隐含的，实际机器数中是不存在的，一般不标注，这里仅用于提醒数值各位的位权。

对于单、双精度，因尾数真值 *m* 即有效数字的整数部分固定为 1，所以表示成机器数时将其隐含（因此称为尾数）。例如，若尾数真值 *m* 即有效数字为+1.5，则尾数符号 *S* 为 0，表示正数，尾数数值 *M* 为.5 表示实际存储的内容，小数点左边的 1 不保存。因此，浮点数的真值表示为：

$$浮点数真值=(-1)^S\times1.M\times2^{移码表示的阶码（E）-偏置量}$$

例如，若单精度浮点数的机器数为 1 1000 0000 .110 0000 0000 0000 0000 0000B，则根据图 1-5（a）单精度的格式可知，该机器数对应的尾数符号 *S*=1，*M* 的实际存储为 .110 0000 0000 0000 0000 0000B=0.75，故尾数真值 $m=(-1)^S\times1.M=(-1)^1\times1.75=-1.75$；阶码 *E*=1000 0000B=128，单精度浮点数的偏置量为 127，指数即阶码真值 *e*=阶码（*E*）−偏置量

=128−127=1。因此，该浮点数的真值为：

$$(-1)^1\times1.75\times2^{128-127}=-1.75\times2^1=-3.5$$

若单精度浮点数的机器数为 0 0000 0000 .000 0000 0000 0000 0000 0000B，则该机器数对应的尾数符号 S=0，尾数数值 M 的实际存储为.000 0000 0000 0000 0000 0000B=0，阶码 E=0000 0000B=0，单精度浮点数的偏置量为 127，指数即阶码真值 e=阶码（E）−偏置量=0−127=−127。因此，该浮点数理论上的真值为：

$$(-1)^0\times1.0\times2^{0-127}=2^{-127}$$

这是浮点数所能表示的最小正数，因为没有比这更小的正数可以表示浮点数 0。因此，IEEE 754 规定，当尾数数值和阶码全为 0 时表示机器零。其他特殊浮点数详见 5.5.6 节。

例 1-06　将+12.0 表示成浮点数对应的单精度机器数。

要将浮点数表示成机器数，须将该浮点数表示成尾数真值×$2^{\text{阶码真值}}$，当浮点数的绝对值大于 2 时，可用 2^{+n} 去除，直到结果为[1,2)的有效数字即尾数真值，则小数部分作为尾数数值（M），而+n 作为阶码的真值；当浮点数的绝对值小于 1 时，可用 2^{+n} 去乘，直到结果为[1,2)的有效数字，则小数部分作为尾数数值（M），而−n 作为阶码的真值。因此，+12.0=+12/2^{+3}×2^{+3}=+1.5×2^{+3}，有效数字为+1.5，尾数符号 S=0，对应尾数实际存储的数值 M=0.5（小数点左边的整数 1 隐含），M 用 23 位原码表示为：

$$[0.5]_{\text{原码}}=.100\ 0000\ 0000\ 0000\ 0000\ 0000\text{B}$$

阶码真值为+3，用 8 位移码表示为：

$$[+3]_{\text{移码}}=+3+127=130=128+2=1000\ 0010\ \text{B}$$

所以，+12.0 对应机器数为 0 1000 0010 .100 0000 0000 0000 0000 0000B=4140 0000H

计算过程如图 1-6 所示。

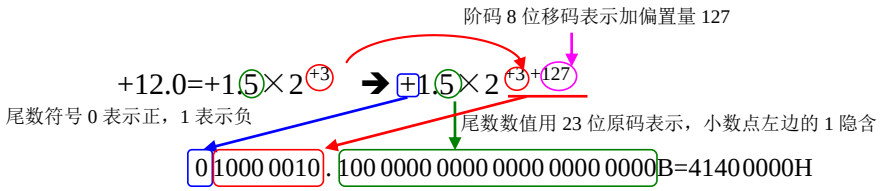


图 1-6　浮点数表示成单精度机器数的过程

机器数 4140 0000H 所表示的浮点数可用以下 C 程序进行验证。

例 1-07　将机器数 0x4140 0000 转换为浮点数并输出。

源程序如下：


```
#include"stdio.h"
void main()
{
    int d=0x4140 0000;              //浮点数对应的机器数
    float *p=(float*)&d;            //将 int 地址强制转换为 float 地址，&d 的完整表示为(int*)&d
    printf("%f %f\n",*p,*(float*)&d);   //再取内容输出
}
```

输出结果为：

```
12.000000 12.000000
```