IBM—PC
汇编语言程序设计

沈美明　温冬婵　编著

清华大学出版社

IBM PC 汇编语言程序设计

沈美明　温冬婵　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书为清华大学计算机汇编语言程序设计课教材，主要阐述 IBM PC 及其兼容机汇编语言程序设计的方法和技术。全书共 13 章：第一、二章介绍基础知识；第三、四章说明 IBM PC 机的指令系统及包括伪操作在内的汇编语言程序格式；第五、六章叙述循环、分支、子程序等基本程序结构以及程序设计的基本方法和技术；第七章为宏汇编技术；第八章说明以中断为主的输入/输出程序设计方法；第九章介绍 BIOS 和 DOS 系统功能调用的使用方法；第十～十二章分别说明图形显示、发声和磁盘文件存取的程序设计方法及程序实例；第十三章为包括汇编语言和高级语言相连接在内的多个模块的连接技术。全书提供了大量程序实例，每章后均附有习题。

本书适于初学者使用，只要具有一种高级语言程序设计基础的读者，都可通过学习本书掌握汇编语言程序设计技术，因此，本书除适用于高等院校“汇编语言程序设计”课程教材外，也可供使用汇编语言的工程技术人员参考。

IBM PC 汇编语言程序设计

沈美明　温冬婵　编著

责任编辑　贾仲良

☆

清华大学出版社出版

北京　清华园

化学工业出版社印刷厂印装

新华书店总店科技发行所发行

☆

开本：787×1092 1/16　印张：26.75　字数：684 千字

1991 年 6 月第 1 版　1994 年 3 月第 6 次印刷

印数：197001—247000

ISBN 7-302-00829-9/TP・301

定价：11.80 元

前　言

汇编语言是计算机能提供给用户的最快而又最有效的语言，也是能够利用计算机所有硬件特性并能直接控制硬件的唯一语言，因而在对于程序的空间和时间要求很高的场合，汇编语言是必不可少的，至于对于很多需要直接控制硬件的应用场合，则更是非用汇编语言不可了。

“汇编语言程序设计”是高等院校电子计算机硬、软件及应用专业学生必修的核心课程之一。它不仅是计算机原理、操作系统等其它核心课程的必要先修课，而且对于训练学生掌握程序设计技术，熟悉上机操作和程序调试技术都有重要作用。由于汇编语言本身的特点，使这一课程必须结合一台具体的计算机来组织教学。我们选择了目前国内最广泛使用的 IBM PC 机（包括 XT，AT，IBM PC 兼容机及国产 0520 微型机系列）作为基础机型而编写了本书，因而本书中所有例题均可在上述各种计算机上运行。本书适于初学者使用，只要有一种高级语言程序设计基础，都可以通过学习本书掌握汇编语言程序设计技术。因此，本书不仅可作为高等院校“汇编语言程序设计”课程的教材，也可以供需用汇编语言的工程技术人员及科研人员使用。

本书共有十三章。第一、二章为汇编语言所用的基础知识，已了解计算机基本原理的读者可以跳过这两章。第三章详细介绍 IBM PC 机的指令系统和寻址方式，并给出各种指令的使用举例。第四章介绍伪操作、汇编语言程序格式及汇编语言的上机过程。第五、六章说明循环、分支、子程序结构和程序设计的基本方法，并提供起泡排序、折半查找等多种常用算法的程序举例。第七章说明宏汇编、重复汇编及条件汇编的设计方法。第八章叙述输入/输出程序设计方法，重点说明中断原理、中断过程及中断程序设计方法。第九章说明 BIOS 和 DOS 系统功能调用的使用方法。第十～十二章分别说明图形显示、发声及磁盘文件存取的程序设计方法，同时提供各种程序设计方法和程序实例。第十三章主要说明多个模块相连接时有关的程序设计技术以及汇编语言程序与高级语言程序的连接技术。最后简单介绍模块化程序设计及结构程序设计的基本方法。书中提供了大量程序例题，每章后均有习题，便于读者复习及检查学习效果。

本书的第一～七章及十三章由沈美明同志编写，第八～十二章由温冬婵同志编写。在本书定稿的过程中吴家勋同志提出了很多宝贵意见，特在此表示感谢。

编　者

目　录

第一章　基础知识

1.1　进位计数制与不同基数的数之间的转换

1.1.1　二进制数

进位计数制是一种计数的方法，习惯上最常用的是十进制计数法。一个任意的十进制数可以表示为：

$$a_n a_{n-1}\cdots a_0 . b_1 b_2 \cdots b_m$$

其含意是：

$$a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \cdots + a_0 \cdot 10^0 + b_1 \cdot 10^{-1} + b_2 \cdot 10^{-2} + \cdots + b_m \cdot 10^{-m}$$

其中 $a_i(i=0,1,\cdots,n)$，$b_j(j=1,2,\cdots,m)$是 0,1,2,3,4,5,6,7,8,9 十个数码中的一个。

十进制数的基数为 10，即其数码的个数为 10，且遵循逢十进一的规则。上式中相应于每位数字的 10^k 称为该位数字的权，所以每位数字乘以其权所得到的乘积之和即为所表示数的值。例如：

$$12345.67 = 1 \times 10^4 + 2 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0 + 6 \times 10^{-1} + 7 \times 10^{-2}$$

十进制数是人们最熟悉、最常用的一种数制，但它不是唯一的数制。例如计时用的时、分、秒就是按 60 进制计数的。基数为 r 的 r 进制数的值可以表示为：

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_0 \cdot r^0 + b_1 \cdot r^{-1} + b_2 \cdot r^{-2} + \cdots + b_m \cdot r^{-m}$$

其中 a_i, b_j 可以是 $0,1,\cdots,r-1$ 中的任一个数码，r^k 则为各位数相应的权。

计算机中为便于存储及计算的物理实现，采用了二进制数。二进制数的基数为 2，只有 0，1 两个数码，并遵循逢二进一的规则，它的各位权是以 2^k 表示的，因此二进制数 $a_n a_{n-1}\cdots a_0 . b_1 b_2 \cdots b_m$ 的值是：

$$a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \cdots + a_0 \cdot 2^0 + b_1 \cdot 2^{-1} + b_2 \cdot 2^{-2} + \cdots + b_m \cdot 2^{-m}$$

其中 a_i, b_j 为 0,1 两个数码中的一个。例如：

$$101101_2 = 1 \times 2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 45_{10}$$

其中数的下标表示该数的基数 r，即二进制的 101101 与十进制的 45 等值。

n 位二进制数可以表示 2^n 个数。例如 3 位二进制数可以表示 8 个数，它们是：

二进制数	000	001	010	011	100	101	110	111
相应的十进制数	0	1	2	3	4	5	6	7

而 4 位二进制数则表示十进制的 0～15 共 16 个数如下：

二进制数	0000	0001	0010	0011	0100	0101	0110	0111
相应的十进制数	0	1	2	3	4	5	6	7
二进制数	1000	1001	1010	1011	1100	1101	1110	1111
相应的十进制数	8	9	10	11	12	13	14	15

为便于人们阅读及书写，经常使用八进制数或十六进制数来表示二进制数。它们的基数和数码表示如表 1.1 所示。

表 1.1　几种常用的进位计数制的基数和数码

进位计数制	基　数	数　　码
十六进制数	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
十进制数	10	0,1,2,3,4,5,6,7,8,9
八进制数	8	0,1,2,3,4,5,6,7,
二进制数	2	0,1

按同样的方法，读者可以很容易地掌握八进制和十六进制数的表示方法。可以看出：23_{10}可以表示为 17_{16}、27_8及 10111_2，1.375_{10}可以表示为 1.6_{16}、1.3_8 及 1.011_2 等。在计算机里，通常用数字后面跟一个英文字母来表示该数的数制。十进制数一般用 D(Decimal)、二进制数用 B(Binary)、八进制数用 O(Octal)、十六进制数用 H(Hexadecimal)来表示。例如：117D，1110101B，0075H，…。当然也可以用这些字母的小写形式，本书的后面就采用了这种表示方法。

1.1.2　二进制数和十进制数之间的转换

1.1.2.1　二进制数转换为十进制数

各位二进制数码乘以与其对应的权之和即为与该二进制数相对应的十进制数。例如：

$$1011100.10111B = 2^6 + 2^4 + 2^3 + 2^2 + 2^{-1} + 2^{-3} + 2^{-4} + 2^{-5} = 92.71875D$$

1.1.2.2　十进制数转换为二进制数

十进制数转换为二进制数的方法很多，这里只说明比较简单的降幂法及除法两种。

·降幂法

首先写出要转换的十进制数，其次写出所有小于此数的各位二进制权值，然后用要转换的十进制数减去与它最相近的二进制权值，如够减则减去并在相应位记以 1；如不够减则在相应位记以 0 并跳过此位；如此不断反复，直到该数为 0 为止。

例 1.1　N＝117D，小于 N 的二进制权为：

64　32　16　8　4　2　1

对应的二进制数是 1　1　1　0　1　0　1

计算过程如下：

$$117 - 2^6 = 117 - 64 = 53 \qquad (a_6 = 1)$$

$$53 - 2^5 = 53 - 32 = 21 \qquad (a_5 = 1)$$

$$21 - 2^4 = 21 - 16 = 5 \qquad (a_4 = 1)$$

$$(a_3 = 0)$$

$$5 - 2^2 = 5 - 4 = 1 \qquad (a_2 = 1)$$

$(a_1 = 0)$

$1 - 2^0 = 1 - 1 = 0$ $(a_0 = 1)$

所以 N=117D=1110101B

例 1.2 N=0.8125D，小于此数的二进制权为：

0.5 0.25 0.125 0.0625

对应的二进制数是 1 1 0 1

计算过程如下：

$0.8125 - 2^{-1} = 0.8125 - 0.5 = 0.3125$ $(b_1 = 1)$

$0.3125 - 2^{-2} = 0.3125 - 0.25 = 0.0625$ $(b_2 = 1)$

$(b_3 = 0)$

$0.0625 - 2^{-4} = 0.0625 - 0.0625 = 0$ $(b_4 = 1)$

所以 N=0.8125D=0.1101B

· 除法

把要转换的十进制数的整数部分不断除以 2，并记下余数，直到商为 0 为止。

例 1.3 N=117D

$117/2 = 58$ $(a_0 = 1)$

$58/2 = 29$ $(a_1 = 0)$

$29/2 = 14$ $(a_2 = 1)$

$14/2 = 7$ $(a_3 = 0)$

$7/2 = 3$ $(a_4 = 1)$

$3/2 = 1$ $(a_5 = 1)$

$1/2 = 0$ $(a_6 = 1)$

所以 N=117D=1110101B。

对于被转换的十进制数的小数部分则应不断乘以 2，并记下其整数部分，直到结果的小数部分为 0 为止。

例 1.4 N=0.8125D

$0.8125 \times 2 = 1.625$ $(b_1 = 1)$

$0.625 \times 2 = 1.25$ $(b_2 = 1)$

$0.25 \times 2 = 0.5$ $(b_3 = 0)$

$0.5 \times 2 = 1.0$ $(b_4 = 1)$

所以 N=0.8125D=0.1101B。

1.1.3 十六进制数及其与二进制、十进制数之间的转换

我们知道，在计算机内部，数的运算和存储都是采用二进制的。但是，二进制数对于人的阅读、书写及记忆都是很不方便的。十进制数虽然是人们最熟悉的一种进位计数制，但它与二进制数之间并无直接的对应关系。为了便于人们对二进制数的描述，应该选择一种易于与二进制数相互转换的数制。显然，使用 2^n 作为基数的数制是能适合人们的这种要求的，常用的有八进制数和十六进制数，我们在这里主要介绍十六进制数。

1.1.3.1 十六进制数的表示

计算机中存储信息的基本单位为一个二进制位(Bit),它可以用来表示 0 和 1 两个数码。此外,由于计算机中常用的字符是采用由 8 位二进制数组成的一个字节(Byte)来表示的,因此字节也成为计算机中存储信息的单位。计算机的字长一般都选为字节的整数倍,如 16 位、32 位、64 位等。一个字节由 8 位组成,它可以用两个四位组(又称半字节)来表示,所以用十六进制数来表示二进制数是比较方便的。

十六进制数的基数为 16,共有 16 个数码,它们是 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F。其中 A 表示十进制的 10,余类推。它们与二进制和十进制数的对应关系如下:

二进制数	0000	0001	0010	0011	0100	0101	0110	0111
十进制数	0	1	2	3	4	5	6	7
十六进制数	0	1	2	3	4	5	6	7
二进制数	1000	1001	1010	1011	1100	1101	1110	1111
十进制数	8	9	10	11	12	13	14	15
十六进制数	8	9	A	B	C	D	E	F

对应于十六进制数中各位的权是 16^k。

1.1.3.2 十六进制数和二进制数之间的转换

由于十六进制数的基数是 2 的幂,所以这两种数制之间的转换是十分容易的。一个二进制数,只要把它从低位到高位每 4 位组成一组,直接用十六进制数来表示就可以了。

例 1.5

0011 0101 1011 1111
3 5 B F

亦即 0011010110111111B=35BFH

反之,把十六进制数中的每一位用 4 位二进制数表示,就形成相应的二进制数了。

例 1.6

A 1 9 C
1010 0001 1001 1100

亦即 A19CH=1010000110011100B

1.1.3.3 十六进制数和十进制数之间的转换

各位十六进制数与其对应权值的乘积之和即为与此十六进制数相对应的十进制数。

例 1.7 N=BF3CH

$=11\times16^3+15\times16^2+3\times16^1+12\times16^0$

$=11\times4096+15\times256+3\times16+12\times1$

=48956D

十进制数转换为十六进制数也可使用降幂法和除法。

· 降幂法

首先写出要转换的十进制数,其次写出小于该数的十六进制权值,然后找出该数中包含多少个最接近它的权值的倍数,这一倍数即对应位的值,用原数减去此倍数与相应位权值的乘积得到一个差值,再用此差值去找低一位的权值的倍数,如此反复直到差值为 0 为止。

例 1.8

N=48956D 小于 N 的十六进制权值为

	4096	256	16	1
对应的十六进制数	B	F	3	C

计算过程如下：

$$48956 - 11 \times 4096 = 3900$$
$$3900 - 15 \times 256 = 60$$
$$60 - 3 \times 16 = 12$$
$$12 - 12 \times 1 = 0$$

所以 N＝48956D＝(11)(15)(3)(12)

＝BF3CH

· 除法

把要转换的十进制数的整数部分不断除以 16，并记下余数，直到商为 0 为止。

例 1.9

N＝48956D

$48956/16 = 3059$　　$(a_0 = 12)$

$3059/16 = 191$　　$(a_1 = 3)$

$191/16 = 11$　　$(a_2 = 15)$

$11/16 = 0$　　$(a_3 = 11)$

所以　N＝48956D＝BF3CH。

对于要转换的十进制数的小数部分，则应不断地乘以 16，并记下其整数部分，直到结果的小数部分为零为止。由于其方法与二、十进制数的转换的方法是相同的，这里不再举例说明。显然，为把一个十进制数转换为二进制数，可以先把该数转换为十六进制数，然后再转换为二进制数，这样可以减少计算次数，反之，要把一个二进制数转换为十进制数，也可采用同样的办法。

1.2　二进制数和十六进制数运算

1.2.1　二进制数的运算

加法规则：

0＋0＝0

0＋1＝1

1＋0＝1

1＋1＝0 (进位 1)

乘法规则：

0×0＝0

0×1＝0

1×0＝0

1×1＝1

1.2.2　十六进制数的运算

十六进制数的运算可以采用先把该十六进制数转换为十进制数，经过计算后再把结果转换为十六进制数的方法，但这样做比较繁琐。其实，只要按照逢十六进一的规则，直接用十六进制数来计算也是很方便的。

十六进制加法：

当两个一位数之和 S 小于 16 时，与十进制数同样处理，如两个一位数之和 S≥16 时，则应

该用S－16及进位1来取代S。

例 1.10

$$\begin{array}{r} 05C3H \\ +3D25H \\ \hline 42E8H \end{array}$$

十六进制数的减法也与十进制数类似，够减时可直接相减，不够减时服从向高位借1为16的规则。

例 1.11

$$\begin{array}{r} 3D25H \\ -05C3H \\ \hline 3762H \end{array}$$

十六进制数的乘法可以用十进制数的乘法规则来计算，但结果必须用十六进制数来表示。

例 1.12

$$\begin{array}{r} 05C3H \\ \times 00ABH \\ \hline 3F61 \\ +399E \\ \hline 3D941H \end{array}$$

十六进制数的除法可以根据其乘法和减法规则处理，需要时读者可自行处理，这里不再赘述。

1.3 计算机中数和字符的表示

1.3.1 数的补码表示

计算机中的数是用二进制来表示的，数的符号也是用二进制表示的。把一个数连同其符号在内在机器中的表示加以数值化，这样的数称为机器数。一般用最高有效位来表示数的符号，正数用0表示，负数用1表示。机器数可以用不同的码制来表示，常用的有原码、补码和反码表示法。由于多数机器的整数采用补码表示法，IBM PC机也是这样，所以我们在这里只介绍补码表示法。

补码表示法中，正数采用符号-绝对值表示，即数的最高有效位为0表示符号为正，数的其余部分则表示数的绝对值。例如，假设机器字长为8位，则$[+1]_{补}=00000001$，$[+127]_{补}=01111111$，$[+0]_{补}=00000000$。

当用补码表示法来表示负数时则要麻烦一些。负数X用$2^n-|X|$来表示，其中n为机器的字长。当n=8时，$[-1]_{补}=2^8-1=11111111$，而$[-127]_{补}=2^8-127=10000001$，显然，最高有效位为1表示该数的符号为负。应该注意，$[-0]_{补}=2^8=00000000$，所以在补码表示法中0只有一种表示，即00000000。对于10000000这个数，在补码表示法中被定义为－128。这样，8位补码能表示数的范围为－128～＋127。

我们可以用一种比较简单的办法来写出一个负数的补码表示：先写出与该负数相对应的正数的补码表示(用符号-绝对值法)，然后将其按位求反(即0变为1，1变为0)，最后在末位(最低位)加1，就可以得到该负数的补码表示了。

例 1.13 机器字长为16位，写出N＝－117D的补码表示。

＋117D可表示为　0000　0000　0111　0101

按位求反后为　1111　1111　1000　1010

末位加 1 后为　　1111　1111　1000　1011

用十六进制数表示为　F　　F　　8　　B

即 $[-117]_{补}=FF8BH$

例 1.14　如机器字长为 8 位，则－46D 的补码表示为：

＋46 的补码表示　　0010　1110

按位求反　　1101　0001

末位加 1　　1101　0010

用十六进制数表示　　D　　2

即 $[-46]_{补}=D2H$

至此，读者应该已经学会了一个数的补码表示法。在这里，我们顺便说明一下，用补码表示数时的符号扩展问题。所谓符号扩展是指一个数从位数较少扩展到位数较多（如从 8 位扩展到 16 位，或从 16 位扩展到 32 位）时应该注意的问题。对于用补码表示的数，正数的符号扩展应该在前面补 0，而负数的符号扩展则应该在前面补 1。例如，我们已经知道如机器字长为 8 位，则 $[+46]_{补}=00101110$，$[-46]_{补}=11010010$；如果要把它们从 8 位扩展到 16 位，则

$$[+46]_{补}=0000000000101110=002EH \quad [-46]_{补}=1111111111010010=FFD2H$$

下面，我们再来讨论一下 n 位补码表示数的范围问题。8 位二进制数可以表示 $2^8=256$ 个数，当它们是补码表示的带符号数时，它们的表数范围是 $-128\leqslant N\leqslant +127$。一般说来，n 位补码表示的数的表数范围是：

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

所以 n＝16 时的表数范围是：

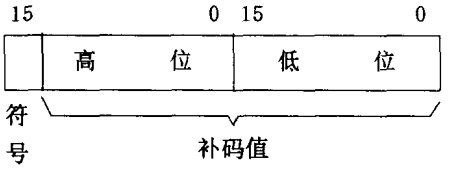
$$-32768\leqslant N\leqslant +32767$$

表 1.2　n 位二进制补码数的表数范围

十进制数	二进制数	十六进制数	十进制数	十六进制数
	n＝8		n＝16	
＋127	01111111	7F	＋32767	7FFF
＋126	01111110	7E	＋32766	7FFE
⋮	⋮	⋮	⋮	⋮
＋2	00000010	02	＋2	0002
＋1	00000001	01	＋1	0001
0	00000000	00	0	0000
－1	11111111	FF	－1	FFFF
－2	11111110	FE	－2	FFFE
⋮	⋮	⋮	⋮	⋮
－126	10000010	82	－32766	8002
－127	10000001	81	－32767	8001
－128	10000000	80	－32768	8000

表 1.2 表示当 n＝8 和 16 时 n 位二进制补码数的表数范围。

在机器里，为了扩大表数范围，可以用二个机器字来表示一个机器数，这种数称为双字长数或双精度数，其格式为：



其中高位字的最高有效位为符号位，高位字的低 15 位和整个低位字的 16 位联合组成 31 位数来表示数值，因而低位字的最高有效位没有符号意义，只有数值意义。双字长数的表数范围可扩大到：

$$-2^{31} \leqslant N \leqslant 2^{31}-1$$

$2^{31} \approx 2.15 \times 10^9$，可见表数范围扩大了许多。上图中每个字上的 0，15 是位编号，每个机器字都从低位开始给每一位以编号，所以从右至左依次编号为 0，1，…，15。

1.3.2 补码的加法和减位

我们知道，对一个正数的补码表示按位求反后再在末位加 1，可以得到与此正数相应的负数的补码表示。我们把这种对一个二进制数按位求反后在末位加 1 的运算称为求补运算，可以证明补码表示的数具有以下特性：

$$[X]_{补} \overset{求补}{\Rightarrow} [-X]_{补} \overset{求补}{\Rightarrow} [X]_{补}$$

在这里，只用例子来说明。由例 1.13 可见：

$$[117]_{补} = 0075H$$

$$[-117]_{补} = FF8BH$$

现对$[-117]_{补}$作求补运算：

$[-117]_{补}$为	1111	1111	1000	1011
按位求反后得	0000	0000	0111	0100
末位加 1 后得	0000	0000	0111	0101

此数正是$[+117]_{补}=0075H$。

这一特性在补码的加、减法运算中很有用。

补码的加法规则是：

$$[X+Y]_{补} = [X]_{补} + [Y]_{补}$$

补码的减法规则是：

$$[X-Y]_{补} = [X]_{补} + [-Y]_{补}$$

其中的$[-Y]_{补}$只要对$[Y]_{补}$求补就可得到。对于这两个规则我们只用例子来说明。读者可以从下面的例子中认识到由于用补码表示数，使计算机中的加、减法运算十分简便，它不必判断数的正负，只要符号位参加运算能自动地得到正确的结果。假设机器字长为 8 位，用下列例子说明补码的加法运算。

例 1.15

十进制	二进制
25	00011001
+32	+00100000
57	00111001

例 1.16

$$\begin{array}{r} 32 \\ +(-25) \\ \hline 7 \end{array} \qquad \begin{array}{r} 00100000 \\ +11100111 \\ \hline \text{C } 00000111 \\ 1 \end{array}$$

例 1.17

$$\begin{array}{r} 25 \\ +(-32) \\ \hline -7 \end{array} \qquad \begin{array}{r} 00011001 \\ +11100000 \\ \hline 11111001 \end{array}$$

例 1.18

$$\begin{array}{r} -25 \\ +(-32) \\ \hline -57 \end{array} \qquad \begin{array}{r} 11100111 \\ +11100000 \\ \hline \text{C } 11000111 \\ 1 \end{array}$$

可以看出，上述 4 个例子的计算结果都是正确的，在例 1.16 和例 1.18 中，从最高有效位向高位的进位由于机器字长的限制而自动丢失，但这并不会影响运算结果的正确性。同时，机器为了某种需要（将在第三章中说明）将把这一进位值保留在标志寄存器的进位位 C 中。

下面，我们再用四个例子说明补码的减法运算，这里机器字长仍假定为 8 位。

例 1.19

十进制			二进制
25	00011001	计算机中用对减数求补的方法把减法转化为加法	00011001
−32	−00100000		+11100000
−7			11111001

例 1.20

$$\begin{array}{r} 32 \\ -(-25) \\ \hline 57 \end{array} \qquad \begin{array}{r} 00100000 \\ -11100111 \end{array} \qquad \begin{array}{r} 00100000 \\ +00011001 \\ \hline 00111001 \end{array}$$

例 1.21

$$\begin{array}{r} -25 \\ -(+32) \\ \hline -57 \end{array} \qquad \begin{array}{r} 11100111 \\ -00100000 \end{array} \qquad \begin{array}{r} 11100111 \\ +11100000 \\ \hline \text{C } 11000111 \\ 1 \end{array}$$

例 1.22

$$\begin{array}{r} -25 \\ -(-32) \\ \hline 7 \end{array} \qquad \begin{array}{r} 11100111 \\ -11100000 \end{array} \qquad \begin{array}{r} 11100111 \\ +00100000 \\ \hline \text{C } 00000111 \\ 1 \end{array}$$

可以看出，在机器里，补码减法是用对减数求补后把减法转换为加法进行的，它同样能自动地得到正确的结果。其中例 1.21 和例 1.22 中的由最高有效位向高位的进位同样自动丢失而不

会影响运算的结果。

1.3.3 无符号整数

在某些情况下，要处理的数全是正数，此时再保留符号位就没有意义了。我们可以把最高有效位也作为数值处理，这样的数称为无符号整数。16 位无符号数的表数范围是 0≤N≤65535，8 位无符号数的表数范围是 0≤N≤255。

在计算机中最常用的无符号整数是表示地址的数。此外，如双精度数的低位字也是无符号整数等。在某些情况下，带符号的数（在机器中用补码表示）与无符号数的处理是有差别的，读者在处理数时，应注意它们的区别。

1.3.4 字符表示法

计算机中处理的信息并不全是数，有时需要处理字符或字符串，例如从键盘输入的信息或打印输出的信息都是字符方式输入输出的，因此，计算机必须能表示字符。字符包括：

字母： A、B、…、Z，a、b、…、Z；

数字： 0、1、…、9；

专用字符： +、-、*、/、↑、SP(space 空格)…

非打印字符：BEL(Bell 响铃)、LF(Line Feed 换行)，CR(Carriage Return 回车)、……

这些字符在机器里必须用二进制数来表示。IBM PC 机采用目前最常用的美国信息交换标准代码 ASCII(American Standard Code for Information Interchange)来表示。这种代码用一个字节(8 位二进制码)来表示一个字符，其中低 7 位为字符的 ASCII 值，最高位一般用作校验位。表 1.3 列出了用十六进制数表示的部分常用字符的 ASCII 值。

表 1.3 常用字符的 7 位 ASCII 值(用十六进制数表示)

字符	ASCII	字符	ASCII	字符	ASCII	字符	ASCII
NUL	00	4	34	M	4D	f	66
BEL	07	5	35	N	4E	g	67
LF	0A	6	36	O	4F	h	68
FF	0C	7	37	P	50	i	69
CR	0D	8	38	Q	51	j	6A
SP	20	9	39	R	52	k	6B
!	21	:	3A	S	53	l	6C
"	22	;	3B	T	54	m	6D
#	23	<	3C	U	55	n	6E
$	24	=	3D	V	56	o	6F
%	25	>	3E	W	57	p	70
&	26	?	3F	X	58	q	71
'	27	@	40	Y	59	r	72
(	28	A	41	Z	5A	s	73
)	29	B	42	[	5B	t	74
*	2A	C	43	\	5C	u	75
+	2B	D	44	]	5D	v	76
,	2C	E	45	↑	5E	w	77
-	2D	F	46	←	5F	x	78
.	2E	G	47	'	60	y	79
/	2F	H	48	a	61	z	7A
0	30	I	49	b	62	{	7B
1	31	J	4A	c	63	|	7C
2	32	K	4B	d	64	}	7D
3	33	L	4C	e	65	~	7E