HUI BIAN YU YAN CHENG XU SHE JI

汇编语言

程序设计

邵玉祥　陈　青　主编

科学出版社
www.sciencep.com

汇编语言程序设计

邵玉祥　陈　青　主编

科学出版社
北京

内 容 简 介

汇编语言是一种执行效率高、功能强的程序设计语言，它能够直接控制计算机硬件，并最大限度地发挥机器硬件的功能。它是计算机相关专业的基础课，是深入学习计算机其他专业课的前提及关键。全书共分三部分，包括汇编语言的组成成分、汇编语言的编程技术以及汇编语言的实际应用，共有8章，分别为汇编语言基础、8086微处理器、寻址方式、汇编语言程序、汇编语言程序设计、汇编语言高级编程、I/O程序设计、中断及中断系统。在编写过程中，采用循序渐进的叙述方法，配以大量的实例来帮助读者对各个知识点的理解。

本书可作为普通高等学校本、专科计算机及相关专业课程的教材，也可供从事计算机设计与应用的人员学习使用。

图书在版编目（CIP）数据

汇编语言程序设计/邵玉祥，陈青主编.—北京：科学出版社，2007
ISBN 978-7-03-019723-8

Ⅰ.汇… Ⅱ.①邵…②陈… Ⅲ.汇编语言－程序设计 Ⅳ.TP313

中国版本图书馆CIP数据核字（2007）第129388号

责任编辑：江 兰／责任校对：梅 莹
责任印制：高 嵘／封面设计：苏 波

科学出版社出版
北京东黄城根北街16号
邮政编码：100717
http://www.sciencep.com
湖北京山德兴印刷有限公司印刷
科学出版社发行 各地新华书店经销
*
2007年8月第 一 版 开本：787×1092 1/16
2007年8月第一次印刷 印张：15 1/4
印数：1—3 000 字数：371 000
定价：21.80元
（如有印装质量问题，我社负责调换）

前　言

汇编语言是一种执行效率高、功能强的程序设计语言，它能够直接控制计算机硬件，并最大限度地发挥机器硬件的能力。在对程序的执行时间和占用空间要求很高的场合，必须使用汇编语言才能满足要求。汇编语言还可以与高级语言进行混合编程，以便发挥各自的优势。通过学习汇编语言，有助于理解操作系统和应用程序的运行原理，准确地分析程序错误。它也是掌握加密/解密技术、病毒蠕虫机理、剖析关键代码等高级技术的必备基础。

汇编语言是计算机相关专业的基础课，是深入学习计算机其他专业课的前提及关键，汇编语言的发展呈两极趋势，要么控制硬件底层，直接编写输入输出控制程序，要么与高级语言混合编程，实现 Windows 环境下的 32 位保护模式下的编程。但是在高校教学中，首先要求学生具有扎实的编程基本功，其次要有较强的实际动手能力，更要培养学生的程序设计创新思维，这就更需要学习汇编语言程序设计。笔者认为汇编语言的掌握程度可以衡量学生对计算机及相关专业的理解深度，控制硬件底层，编写设备驱动程序是汇编语言的特长，Windows 环境下的汇编语言编程更是发展趋势。

本书以 80x86 系列 CPU 为准，以 DOS 操作系统为支撑环境，详细讲述了 16 位的汇编语言编程，同时实现了从 DOS 下的 16 位实模式编程过渡到与高级语言的混合编程，以及 Windows 环境下的 32 位保护模式下的编程。全书以培养学生的创新编程思维为核心，以掌握汇编语言的思考方式为目的，通过精选的例题编程实现内容的理论与实践的结合。

全书共分三部分，包括汇编语言的组成成分、汇编语言的编程技术以及汇编语言的实际应用，第一部分内容由汇编基础知识、寄存器、存储器、变量、指令系统、伪指令组成，第二部分内容由顺序、分支、循环、子程序、宏定义组成，还包括结构、记录数据类型，第三部分内容包括输入输出程序设计，以及中断系统功能调用。在编写过程中，采用循序渐进的叙述方法，配以大量的实例来帮助读者对各个知识点的理解。书中的许多习题都是以例题程序为基础的，可以达到举一反三的效果。

本书共分 8 章，第 1、2 章及附录由张清战编写，第 3、4 章由陈青编写，第 5、6 章由朱华编写，第 7、8 章由邵玉祥编写。该书由主编邵玉祥负责全面内容的规划、编排及程序设计，书中的许多观点和设计思想，都源于主编自身对程序设计和计算机学科的感悟和认知，难免存在一些片面性甚至错误，希望读者能够批评指正，主编将尽量给予纠正。另外，本书也引用了某些名家和同行的研究成果，这些成果经过自身的理解后进行阐述，相应的文献均列入书后的参考文献之中。在此向这些作者表示衷心的感谢！

由于水平所限，加之时间紧迫，书中缺点和问题再所难免，恳请读者批评指正。

编　者

2007 年 5 月于东湖

目　录

第 1 章　汇编语言基础

作为最基本的编程语言之一，汇编语言虽然应用的范围不算很广，但重要性却毋庸置疑，因为它能够完成许多其他语言所无法完成的功能。就拿 Linux 内核来讲，虽然绝大部分代码是用 C 语言编写的，但仍然不可避免地在某些关键地方使用了汇编代码，其中主要是在 Linux 的启动部分。由于这部分代码与硬件的关系非常密切，即使是 C 语言也会有些力不从心，而汇编语言则能够很好地扬长避短，最大限度地发挥硬件的性能。汇编语言的优点是速度快，可以直接对硬件进行操作，能够充分利用计算机硬件特性，是一种面向机器的低级语言，它随机器结构的不同而不同。因此，要学会一种汇编语言，就必须首先了解与该机器有关的硬件结构、数据类型及表示方法等。本章将从机器语言讲解开始，然后介绍汇编语言就是机器语言的符号化，最后介绍与汇编语言有关的基础知识，包括数制、机器编码以及 ASCII 码。

1.1　机器语言与汇编语言

1.1.1　机器语言

计算机正常工作必须接受用户的操纵和控制，而用户为了让计算机实现自己的想法和意愿，就必须与计算机之间进行信息交流，而交流信息的工具就是计算机语言。目前所使用的计算机语言可分为三类：机器语言、汇编语言和高级语言。高级语言接近于自然语言，容易掌握，使用方便，通用性强，而且不依赖于具体的计算机，但是计算机并不能直接识别高级语言，它只能识别在设计该机器时事先规定好的机器指令。

机器指令是指挥计算机完成某一基本操作的命令，每一条机器指令的执行都对应着 CPU 的一种具体操作。机器指令的一般形式为：

操作码	地址码

其中，操作码和地址码均是由 0 和 1 组成的二进制代码，每一条机器指令都是用一组二进制代码表示的。操作码指出了操作的种类，如加、减、传送、移位等，地址码指出了参与操作的操作数和运算结果存放的位置。例如，将变量 x 的内容加 2，结果仍保留在 x 存储单元中，其中变量 x 的偏移地址为 1000H，且为字类型存储单元，如果用 Intel 8086 的机器指令来完成该操作，则相应的指令码为：

```
10000011
00000110
00000000
00010000
00000010
```

其中，第 1、2 行中的两个 8 位二进制数是操作码，表示要进行“加”操作；第 3、4 行中的两个 8 位二进制数指出了第一个加数(称目的操作数)所存放的偏移地址 1000H，相加的结果也送入该存储单元中；第 5 个字节的 8 位二进制数指出了第二个加数(称源操作数)是 2。

机器指令也常常被称为硬指令，它是面向机器硬件的，即每台计算机都规定了自己所特有的一定数量的基本指令，这批指令的全体即为计算机的指令系统，这种机器指令的集合就是机器语言。机器语言是最低级的语言，是用二进制代码表示的计算机能直接识别和执行的一种机器指令的集合。用机器语言编写的、计算机能直接执行的程序称为机器语言程序。

通过上例可以看出，机器指令是用许多二进制数表示的，用机器语言编程必然很烦琐，非常消耗精力和时间，难记忆，易弄错，并且难以检查程序和调试程序，工作效率低。因此，现在一般不使用机器语言编写程序。

1.1.2　汇编语言

因为机器指令是用二进制表示的，编写程序相当麻烦，而且写出的程序也难以阅读和调试，所以为了克服这些缺点，人们就想出了用“助记符”表示机器指令的操作码，用“变量”代替操作数的存放地址，另外还可以在指令前加上标号，用来代表该指令的存放地址等。这种用符号书写的，其主要操作与机器指令基本一一对应的，并遵循一定语法规则的计算机语言就是汇编语言，用汇编语言编写的程序称为汇编源程序。

由此可见，汇编语言也是低级语言，是面向机器的语言，实质是机器语言的符号化。本书重点介绍的是 Intel 8086/8088 宏汇编语言，它是面向 8086/8088 微处理器的。例如，对于前面的例子，用宏汇编语言来书写则应为：

```
ADD WORD PTR DS:[1000],2
```

其中，“ADD”为加指令的助记符；“DS:[1000]”表示在当前数据段中偏移地址为 1000H 存储单元中的内容，是目的操作数；“WORD PTR”说明了这个目的操作数是字类型，而源操作数是 2，相加的结果送入目的操作数所在的原存储单元中。

由于汇编语言是为了方便用户而设计的一种符号语言，因此，用它编写的源程序并不能直接被计算机所识别，必须将它“翻译”成由机器指令组成的机器语言程序后，计算机才能执行。这种由汇编源程序经过“翻译”转换成的机器语言程序也称为目标程序，目标程序中的二进制代码(即机器指令)称为目标代码，一般以 OBJ 作为文件扩展名。这个“翻译”工作又称为“汇编”，在高级语言中又称为“编译”。

汇编源程序经汇编后所生成的目标代码，还不能直接交给计算机去执行，还需要通过连接程序的装配才具备可执行性，装配结果称为“执行文件”，一般以 EXE 作为文件的扩展名。同时，连接程序还具有把多个目标程序装配在一起的功能，或者把目标程序与预先编写好的子程序库中的子程序连接在一起，构成较大的执行文件。汇编语言源程序、汇编程序、目标程序、连接程序、执行文件的关系如图 1.1 所示。

在汇编程序过程中，为了让汇编程序正确地完成翻译工作，必须告诉汇编程序，源程序应从什么位置开始存放，汇编到什么位置结束，数据应放在什么位置，数据的类型是什么，留多少内存单元作临时存储区等。这就要求源程序中应该有一套告诉汇编程序

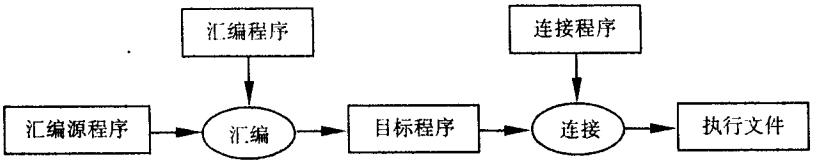


图 1.1　程序汇编连接图

如何进行汇编工作的命令，这种命令称为伪指令(或称汇编控制命令)。汇编指令与汇编伪指令的本质区别为是否产生目标代码。由此可见，指令助记符、语句标号、数据变量、伪指令以及它们的使用规则构成了整个汇编语言的内容。

由于汇编语句基本上与机器指令相对应，因而它的编写也是相当麻烦的。为了简化程序的编写，提高编程效率，现代的计算机系统一般都提供了宏汇编程序。它允许程序员用一个名字(宏指令名)来代替程序中重复出现的一组语句，然后在源程序中所需要的地方使用宏指令名及不同的参数进行宏调用。熟练灵活地使用宏调用功能可使汇编源程序编写得像高级语言一样清晰、简洁，且容易使算法标准化、处理问题灵活，这样，不仅加速了程序的编写进程，而且有利于阅读、修改和调试源程序。Intel 8086 把用宏汇编语言编写的源程序翻译成目标程序的工作是由名称为MASM.EXE的汇编程序来完成的。

与机器语言相比，汇编语言易于理解和记忆，所编写的源程序也容易阅读和调试，所占用的存储空间、执行速度与机器语言相仿。与高级语言相比，汇编语言具有直接和简洁的特点，用它编制程序能精确地描述算法，充分发挥计算机硬件的功能。汇编语言直接同计算机的底层软件甚至硬件进行交互，它具有如下一些优点：

(1) 能够直接访问与硬件相关的存储器或 I/O 端口；

(2) 能够不受编译器的限制，对生成的二进制代码进行完全的控制；

(3) 能够对关键代码进行更准确的控制，避免因线程共同访问或者硬件设备共享引起的死锁；

(4) 能够根据特定的应用对代码做最佳的优化，提高运行速度；

(5) 能够最大限度地发挥硬件的功能。

1.2　数制与数制转换

在日常生活中每天都要和数打交道，通常书写的都是十进制数，但是计算机内部使用的却是由 0 和 1 两种符号构成的二进制数。二进制数在书写时显得过于冗长、麻烦，所以在与计算机打交道时，经常使用八进制数和十六进制数。同一数值可以用不同的数制表示出来，不同数制之间可以相互转换。

1.2.1　数　制

数制是人们常用的一种计数方法，任何一种数制都涉及以下三个问题：

1. 计数符号

这是用于书写数值的各个符号，所有计数符号构成的集合称做数符集，k 进制的数符集中必然包含 k 个符号。例如：

二进制数符集中有 2 个符号：0 和 1；

八进制数符集中有 8 个符号：0，1，2，3，4，5，6，7；

十进制数符集中有 10 个符号：0，1，2，3，4，5，6，7，8，9；

十六进制数符集中有 16 个符号：0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F。

显然，任意进制的各个计数符号是有顺序的，二进制、八进制、十进制的每个计数符号就是它的序号值，十六进制数的后 6 个计数符号 A，B，C，D，E，F 的序号值依次是 10、11、12、13、14、15。

2. 基数和权

如果把用 k 进制书写的一个整数从右往左依次记做第 0 位、第 1 位、……、第 n 位，则第 i 位上的数符 a_i 所代表的含义是 a_i*k^i。在此，我们把 k 称为一个数制的基数，而把 k^i 称为 k 进制数第 i 位的权。

3. 计数规则

简单地说，就是“逢 k 进 1，借 1 当 k”。

1.2.2　数制转换

1. 将非十进制数转换成十进制数

不难看出，一个任意 k 进制数可以写成以下形式：

$$a_na_{n-1}a_{n-2}\cdots a_1a_0=a_n*k^n+a_{n-1}*k^{n-1}+a_{n-2}*k^{n-2}\cdots+a_1*k^1+a_0*k^0$$

其中，首先依照计数符号的次序把 k 进制的符号 a_i 转换成十进制数，然后把上式的右边按照十进制的运算规则进行计算，求得的结果就是相应的十进制数。

例 1.1　把十六进制数 1AB2C 转换成为十进制数。

解　十六进制数 1AB2C 可以写成如下形式：

$$1AB2CH=1*16^4+10*16^3+11*16^2+2*16^1+12*16^0=109356D$$

2. 把十进制数转换成非十进制数

根据上式，如果能把一个十进制数写成上式右边的形式，当缺少某一项权值 k^j 时，就在它相应的位置上写 $0\times k^j$，并且保证每个权项值 k^j 前面的系数 a_j 都是小于 k 的非负整数，这样就可以先把每个系数转换成 k 进制数相应的数符，然后从最大权开始，按照权从大到小的次序，依次记录下各系数对应的数符，即可得到相应的 k 进制数。

例 1.2　把十进制数 15370 转换成十六进制数。

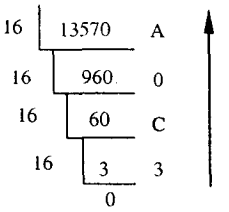
解　十进制数 15370 可以写成如下形式：

$$15370D=3*16^3+12*16^2+0*16^1+10*16^0=3C0AH$$

按十六进制数符的排列次序可知：十进制数 12 对应十六进制数数符 C，十进制数 10 对应十六进制数数符 A，对于式中缺少的权 16^1，以 0 作为它前面的系数，而最后一项 10 可以看做 10×16^0。所以与十进制数 15370 相等的十六进制数是 3C0A。

注意　在上例求解过程中，关键是如何求出右式中各权值项前面的系数 3、12、10，求解方法可以归结为“除 k 取余”法。

求解过程如下：



计算所得系数值为 3、C、0、A。

3. 二进制、八进制、十六进制之间的转换

在十进制以外的其他数制之间，如果要把一个 k 进制数转换成 r 进制数，一般以十进制作为中间媒介。先把 k 进制数转换成十进制数，再把等值的十进制数转换成 r 进制数。但是，在二进制数、八进制数、十六进制数之间实现数制转换，存在着非常简便的转换方法即“比例系数”法，即二进制数与八进制数之间比例系数为 3:1，二进制数与十六进制数之间的比例系数为 4:1，八进制数与十六进制数之间转换以二进制数为中间媒介。

例 1.3　把二进制数 101100110101111 转换成八进制数和十六进制数。

解　按比例系数 3:1 分组，101，100，110，101，111

八进制数，　5　　4　　6　　5　　7

按比例系数 4:1 分组，0101，1001，1010，1111(前边不足 4 位补 0)

十六进制数，　5　　9　　A　　F

所以，等值的八进制数是 54657，十六进制数是 59AF。

八进制数、十六进制数转换成二进制数则更简单，只需从左到右把每位八进制数和十六进制数按 8421 码转换成相应的 3 位二进制或 4 位二进制，不足 3 位或 4 位的左补 0 凑足即可，并把结果写在一起。

1.2.3　数的书写方法

由于计算机中经常使用的数制有二进制、八进制、十进制、十六进制，所以有时写出一个数时，却无法分辨到底使用了哪一种数制，比如 377。为此需要在书写形式上加以区别。一般来说，书写方法有三种：下标法、前导法、后缀法。

1. 下标法

这是日常书写的一种常用方法，是在写完表示数值的符号序列之后，在其右下角以角标的形式写上所使用数制的基数，比如：

$(1001)_2$ 表示二进制数 1001；

$(377)_8$ 表示八进制数 377；

$(377)_{16}$ 表示十六进制数 377。

对于日常使用的十进制数，可以按上述方法加下标表示，也可以不加；反之，如果一个数在书写时没有加上任何标记，则表示是十进制数。

2. 前导法

有些程序设计语言中使用前导法，即在表示数值的数符序列前面加上特定的前导符号以区分不同的数制。比如，在 C 语言中就用 0x 作为十六进制数的前导符号，在 Turbo

Pascal 中用$作为十六进制数的前导符号。十六进制数 123 在 C 语言中写做 0x123，在 Turbo Pascal 中则写做$123。

3. *后缀法*

与前导法相反的另一种数的写法是后缀法，即在写完表示数值的数符序列后，加上一个特殊的符号表示不同的数制。汇编语言中就采取这种写法。二进制、八进制、十进制和十六进制的后缀符号分别是 B、Q、D 和 H。比如：

1011B 表示二进制数 1011；

1011H 表示十六进制数 1011。

特别强调的是，如果一个十六进制数以字母数符开头，会与后面章节中的变量名、标号等标识符相混淆。为了区分这种情况，在所有字母数符开头的十六进制数的前面再加上一个 0。当然在一个整数的最高位前面加 0 是不影响数值本身的。比如，十六进制数 ABC 必须写成 0ABCH。

注意，如果一个数不加任何后缀，则被默认为是十进制数。

1.3　有符号数与无符号数

1.3.1　有符号数

负数是程序设计中必须面对的问题，但是计算机内部并没有正负号，只有 0 和 1。计算机内部区分正负数的方法是，在存放数的若干个二进制位(一般是 8 位、16 位或 32 位)中，用最高位作为符号位，这一位是 1 表示该数是负数，而这一位是 0 则表示该数非负。对于有符号数而言，负数除了最高位为 1 以外，还采用 *n* 位二进制补码表示。同时，计算机中无论是字符类型，还是整数类型，也无论这个整数占几个字节，它都是用全 1 来表示-1 的。比如一个字节的数值中，1111 1111 表示-1，那么，1111 1111-1 是什么呢？和现实中的计算结果完全一致。1111 1111 -1 = 1111 1110，而 1111 1110 就是-2。这样一直减下去，当减到只剩最高位用于表示符号的 1 以外，其他低位全为 0 时，就是最小的负值了，在一个字节中，最小的负值是 1000 0000，也就是-128。所以，

8 位补码所能表示的有符号数范围为 80H~7FH，即$-128 \leqslant N \leqslant 127$；

16 位补码所能表示的有符号数范围为 8000H~7FFFH，即$-32768 \leqslant N \leqslant 32767$；

32 位有符号数的表示范围为 80000000H~7FFFFFFFH。

1.3.2　无符号数

但是，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，为此可以把最高有效位作为数值位处理，这样的数称为无符号数。其中：

8 位无符号数的表示范围为 00H~0FFH，即 $0 \leqslant N \leqslant 255$；

16 位无符号数的表示范围为 0000H~0FFFFH，即 $0 \leqslant N \leqslant 65535$；

32 位无符号数的表示范围为 00000000H~0FFFFFFFH，$0 \leqslant N \leqslant 2^{32}-1$。

注意　Intel 8086 微处理器是定点机，没有处理浮点数的专门指令，对于浮点数的运算只能用程序来实现，而且是将小数点位固定在第 0 位的后面，所以对 8086 宏汇编语言而言，处理的数据都是有符号定点整数，无符号整数通常用来表示地址和进行逻辑运算。

存放在计算机内部的数是否有符号是人为看待的问题，而不是数据本身所具备的属性。

1.4　原码、反码、补码

前已述及，计算机中的数是用二进制数来表示的，数的符号也是用二进制表示，一般用最高有效位来表示机器数的符号(“0”表示正数，“1”表示负数)。一个数连同其符号在内均用二进制表示，这样的数称为机器数。机器数原来的实际值(符号仍用“+”和“-”)称为真值。机器数可以用不同的编码来表示，常用的有原码、反码及补码。

1. 机器码的表示方法

机器编码又称机器码，它有三种表示方法，即原码、反码及补码。原码的编码规则为：保持真值的数值部分不变，最高位为符号(0或1)的数值化表示。反码的编码规则为：对于正数，保持真值的数值部分不变，最高位加符号位“0”；对于负数，对其真值的数值部分按位取反(将1变为0，0变为1)，最高位加符号位“1”。补码的编码规则为：对于正数，保持真值的数值部分不变，最高位加符号位“0”；对于负数，对其真值的数值部分按位取反后加1，最高位加符号位“1”。

例 1.4　设 M = + 61= +3DH，

则 n=8 时，M 的 8 位原码表示为：$[M]_{原}$=3DH；

M 的 8 位反码表示为：$[M]_{反}$=3DH；

M 的 8 位补码表示为：$[M]_{补}$=3DH。

而 n=16 时，M 的 16 位原码表示为：$[M]_{原}$=003DH；

M 的 16 位反码表示为：$[M]_{反}$=003DH；

M 的 16 位补码表示为：$[M]_{补}$=003DH。

例 1.5　设 M=-61=-3DH，

则 n=8 时，M 的 8 位原码表示为：$[M]_{原}$=0BDH；

M 的 8 位反码表示为：$[M]_{反}$=0C2H；

M 的 8 位补码表示为：$[M]_{补}$=0C3H。

而 n=16 时，M 的 16 位原码表示为：$[M]_{原}$=803DH；

M 的 16 位反码表示为：$[M]_{反}$=0FFC2H；

M 的 16 位补码表示为：$[M]_{补}$=0FFC3H。

2. 机器码间的相互关系

在计算机中，采用什么编码方式，主要的目的是便于运算，简化逻辑结构，提高运算精度和可靠性。实践证明，补码运算比较方便，所以负数在计算机中常用二进制补码表示。而原码表示最为简单，反码与原码、补码之间存在简单的对应关系。因此，可先求一个数的原码，通过反码过渡而获得它的补码。三种编码之间有如下关系：

正数：$[M]_{原}=[M]_{反}=[M]_{补}$

负数：反码是原码除符号位外的按位取反，补码是原码的“取反加1”，保留符号位。

3. 符号扩展

符号扩展是指一个数从较少位数增加到较多位数的过程，但增加的空位均由最高符号位填充。一般是指一个数由8位扩展到16位，或从16位扩展到32位。从本节以上的

两个实例中可以看出，M 的 16 位补码实际上是其 8 位补码的符号扩展，由此可以得出以下重要结论：一个二进制数的补码表示中的最高位(即符号位)向左扩展若干位(即符号扩展)后，所得到的数仍是该数的补码。

1.5　ASCII 码

计算机中所处理的数据包括数值数据和字符数据，数据在计算机内的存储都以二进制的形式。对于字符数据而言，是用一个字节表示一个字符的，其字符的数值表示规律是以美国信息标准交换代码 ASCII 码为标准的。在 8086/8088 中，从键盘输入的任意字符，在计算机内的表示都是 ASCII 码的形式。例如，当从键盘输入字符串“1234ABCD”时，它们立即被转换成与之对应的 ASCII 31H，32H，33H，…，44H，如果在程序中需要用到数字 1234 时，必须把相应的 ASCII 码转换成原来的数字。同样，当用户需要在显示器上显示自己程序的运行结果 1234 时，只要将它们逐一转换成与其对应的 ASCII 码后，存放在主存中，然后送显示器显示即可。

在高级语言中，这些转换工作由系统独立完成，而不需用户作处理，但在汇编语言中，这些工作只能由用户自己完成。为了区别数值数据，程序中的字符数据全部以单引号或双引号括起来，常用字符的 ASCII 码如表 1.1 所示。

表 1.1　常用字符 ASCII 表(16 进制数)

字符	ASCII 码	字符	ASCII 码	字符	ASCII 码	字符	ASCII 码
0	30	a	61	k	6B	u	75
1	31	b	62	l	6C	v	76
2	32	c	63	m	6D	w	77
3	33	d	64	n	6E	x	78
4	34	e	65	o	6F	y	79
5	35	f	66	p	70	z	7A
6	36	g	67	q	71	A	41
7	37	h	68	r	72	B	42
8	38	i	69	s	73	C	43
9	39	j	6A	t	74	D	44
E	45	J	4A	O	4F	T	54
F	46	K	4B	P	50	U	55
G	47	L	4C	Q	51	V	56
H	48	M	4D	R	52	W	57
I	49	N	4E	S	53	X	58
Y	59	Z	5A	回车	0D	换行	0A
:	3A	;	3B	<	3C	=	3D
>	3E	?	3F	@	40	[	5B
]	5D	)	7D	{	7D	~	7E
!	21	#	23	$	24	%	25
&	26	(	28	)	29	*	2A
+	2B	,	2C	-	2D	/	2F

从表 1-1 中可以看出，数字 0~9 的 ASCII 码值是 30H~39H，当使用由键盘输入的字符数字原数字值时，相应 ASCII 码值减 30H 即可。小写字母 a~z 的 ASCII 码值为 61H~7AH，大写字母 A~Z 的 ASCII 码值为 41H~5AH，其中小写字母的 ASCII 码值大于