# IBM-PC机
# 汇编语言程序设计

HUIBIANYUYANCHENGXUSHEJI

余朝琨 编著

厦门大学出版社

# IBM－PC机汇编语言程序设计

余朝琨　　编著

厦门大学出版社

**图书在版编目(CIP)数据**

IBM—PC 机汇编语言程序设计/余朝琨编著. —厦门:厦门大学出版社,2001.7
ISBN 7-5615-1766-1

Ⅰ.I… Ⅱ.余… Ⅲ.汇编语言-程序设计-高等学校-教材 Ⅳ.TP313

中国版本图书馆 CIP 数据核字(2001)第 027935 号

厦门大学出版社出版发行
(地址:厦门大学 邮编:361005)
http://www.xmupress.com
xmup @ public.xm.fj.cn
三明日报社印刷厂印刷
2001 年 7 月第 1 版 2001 年 7 月第 1 次印刷
开本:787×1092 1/16 印张:28
字数:710 千字 印数:1—1 500 册
定价:35.00 元

# 内容简介

本书阐述和讨论了 8086 汇编语言程序设计的原理、方法和技巧。主要内容包括：编程的基础知识；指令的寻址方式和操作数的寻址方式；较系统地介绍了 80X86 的指令系统；汇编语言的程序格式；注重一个完整的源程序的编写过程，对顺序、分支、循环和子程序的基本结构形式以及程序设计都作了较为详细的介绍；同时对 I/O 端口与 CPU 间数据交换方式及中断、高级宏汇编技术和 BIOS 及 DOS 功能调用都进行了适当的介绍。

本书可作为高等院校汇编语言程序设计的教材，也可供使用汇编语言的工程技术人员参考。

# 前 言

本书付梓之际，人类已经迈进 21 世纪，迎接了信息时代。随着智能计算机、网络和多媒体技术的不断发展，人类社会的信息化程度将越来越高，人们也越来越重视对计算机知识的掌握。

IBM－PC 汇编语言程序设计是计算机及相关专业必修的一门核心专业基础课程，学习它对掌握程序设计方法、从事软件开发和应用都有重要作用。

同其他高级语言相比，汇编语言是属于低层次的程序设计语言。所谓低层次，主要表现在它同具体的机器联系密切。正因为如此，它可以更加充分地发挥机器的功能和特点。本书是以 Intel 8086 系列微型机为背景，介绍其汇编语言程序设计，这对今后的实际工作是有利的。即使是在更高档的微机上开发软件，先掌握 8086 汇编语言程序设计必将是以后得心应手工作的基础。

本书共分十二章，第一章主要介绍与编程有关的基础知识，重点讨论 CPU 中寄存器组与存储器的分段管理的要点；第二章阐述指令的寻址及操作数的寻址方式，这是重要的编程基础；第三章用了较大的篇幅介绍了 80X86 的指令系统，采取边讲指令边练习编程的方法，由浅入深，循序渐进，使初学者不会感到抽象。突出指令的使用规则，让读者明了程序编写重在“算法与语法”，两者缺一不可；第四章介绍汇编语言的程序格式。一个完整的源程序，必须向汇编程序提供各种信息，所以说伪指令与指令是同等重要，必须足够重视；汇编语言是一门实践性强的课程，为此在第五章介绍与上机实践有关的知识供读者参考；从第六到第十章可以说是汇编语言程序设计的具体实例，分门别类地进行理论分析及实例分析，旨在对读者有抛砖引玉的作用；第十一章讲述高级宏汇编语言技术，介绍了结构、记录以及宏指令的使用方法；第十二章 BIOS 和 DOS 功能调用是在读者已有初步汇编语言程序设计知识和技术之后的扩展和延伸。

在本书的编写过程中得到吴锤红教授的大力支持与帮助；得到我系领导的鼓励与支持。在书稿的录入等工作中张祖曦、杨建成、余力、林剑、林钦、刘必渠、余芳等同志做了大量工作，在此深表谢意。

本书自立项以及在书稿形成过程中都适时地得到厦门大学出版社宋文艳副总编的指教，作者特表谢意。

由于编者的水平有限，书中不妥甚至谬误之处在所难免，敬请读者不吝赐教为感。

余朝琨

于 2001 年春

# 目 录

# 第一章　汇编语言编程的基础知识

## 1.1　汇编语言程序的概念

现今我们社会中无论哪个行业、部门、地区，到处可见计算机的应用，尤其是多媒体、网络的应用，计算机已进入千家万户，随着上网人数的日益增多，计算机也就成为人们生活中一种重要的学习、工作及娱乐的工具。

人们可以学会某种高级语言进行编制程序，与计算机通信，从而感受到高级语言容易学习，用起来很顺手。这是因为这些高级语言的语句是面向数学语言或自然语言的，因此容易接受、掌握。他们都把实际的计算机当成“虚拟机”，这其中有许多人对计算机是如何工作的还不甚了解。

相对来讲，汇编语言是面向机器的低级语言，其编程要比高级语言困难些。既然如此，那我们为什么至今还要学习和使用汇编语言呢？汇编语言到底有什么特点？

1. 学习和使用汇编语言可以从根本上认识、理解计算机的工作过程。因为一台计算机不管执行什么任务，诸如显示五彩缤纷的画面，或是收发电子邮件等等，归根结底都要执行相对应的计算机机器程序，而机器指令是由 CPU 中的控制器直接提供的，我们可以把控制器(CPU)看成是机器指令的解释器。机器指令是以代码的形式表示的，是面向设计者的，对于用户来说，编制程序和阅读程序都是相当困难的。然而，汇编语言程序是把由机器指令组成的机器语言程序“符号化”，并与机器语言程序一一对应。

用汇编语言编写程序，可以更加清楚地了解计算机是怎样完成各种复杂工作的。在此基础上，程序员更能充分地利用机器硬件的全部功能，发挥机器的长处。

2. 现在的计算机系统中，某些功能仍然是靠汇编语言程序来实现的。例如机器自检，系统初始化，实际的输入输出设备的操作，至今仍然用汇编语言编制的程序来完成。

3. 汇编语言程序的效率通常高于高级语言程序。这里所提到的“效率”主要指产生的程序目标代码长度和程序运行的速度。所以在需要节省内存空间和提高程序运行速度等重要场合，如实时过程控制，则常常采用汇编语言来编制控制程序。

4. PC－DOS 系统中设置了两层内部子程序可供用户使用，即基本输入输出子程序 BIOS 和 DOS 层功能模块。这些子程序对用户来说均可看成中断处理程序。它们的入口都安排在中断入口表中。当我们使用汇编语言编程时可以直接调用它们，这就极大地方便了用户对这些微机系统资源的利用。

鉴于以上原因，现在许多高级语言都设置有与汇编语言程序接口的功能，以便于程序员用汇编语言编制一些子程序，完成与机器紧密联系的特定功能，提高高级语言程序的效率。

这里试举一例说明用高级语言和用汇编语言编程的情况。例如要完成三个数相加运算的 BASIC 语言的一种编程方法如下：

```
10  READ  A, B, C
20  LET  S=A+B+C
30  DATA  18, 28, 38
40  END
```

同样的运算，用 8086 汇编语言编写的完整的源程序如下：

```
DATA  SEGMENT
A  DB  18                                    ; 数据 A
B  DB  28                                    ; 数据 B
C  DB  38                                    ; 数据 C
S  DB  ?                                     ; 存放和单元
DATA  ENDS                                   ; 数据段定义
STACK  SEGMENT  PARA  STACK  'STACK'         ; 定义堆栈段
DB  100  DUP(?)
STACK  ENDS
CODE  SEGMENT
      ASSUME  CS:CODE, DS:DATA, SS:STACK
START: MOV  AX, DATA
      MOV  DS, AX                            ; 数据段赋值
      MOV  AL, A                             ; 取第一个数
      ADD  AL, B                             ; 与第二个数相加
      ADD  AL, C                             ; 与第三个数相加
      MOV  S, AL                             ; 存放结果
      MOV  AH, 4CH                           ; 返回 DOS
      INT  21H
CODE  ENDS
      END  START
```

上述汇编语言源程序经MASM的汇编成目标程序，再经LINK连接而产生可执行的程序。

从这个例子可以看出：用汇编语言编制的程序要比用高级语言编制的程序复杂。

由于每种计算机的设计者有不同的设计思想与应用目的，因此每种计算机有它自己的机器指令系统，相应地有自己的机器语言和汇编语言。各机种之间的机器语言和汇编语言一般是不兼容的。我们用汇编语言编制程序时要涉及到计算机硬件内部结构和功能。一台计算机能执行的机器语言程序，主要决定于组成这台计算机的中央处理器 CPU。因此，我们需要了解并熟悉计算机的内部结构，主要是指 CPU 的功能结构：有哪些寄存器，以及这些寄存器的作用；为了访问主存储器，CPU 如何形成访存的地址；在进行输入或输出操作时有些什么工作方式等等。本书以 IBM－PC Intel 8086 CPU 的汇编语言为例进行阐述。8086 的汇编语言是向上兼容的，它在 80X86 机上都可运行。

# 1.2　数据表示法

计算机内部流动的信息可分成两大类：控制信息和数据信息。数据信息是计算机加工处理的对象，控制信息用来控制数据加工处理的过程。数据信息包括数值数据和非数值数据。数值数据是有确定的数值、能表示数量大小的数据；非数值数据又称为符号数据，用来表示一些符号、记号，它不是数值，不能表示数量的大小，例如英文字母、汉字等等。

在汇编语言程序设计中，了解和掌握数据的表示方法是必不可少的。计算机内部只处理二进制数代码。而我们从键盘输入接收的仅是 ASCII 码；CPU 处理的数值结果要送到 CRT 显示或打印机，也应送 ASCII 码；而且为了书写方便或习惯，二进制数可用八进制、十进制、十六进制数表示。这些码制转换都需要靠编程来实现，若不了解这些进位数制或码制之间的相互转换的规律，就难以编写转换程序。同时如果对一个数值数据的真值与机器中的数如何表示不了解，则直接影响数值处理的汇编语言程序设计。

## 1.2.1　进位计数制及其转换

表示一个数值数据有三个要素：进位计数制、小数点和数的正负符号。

1. 进位计数制、基数和位权

按进位方式计数的数制叫做进位计数制，简称进位制。在我们日常生活中，人们习惯于用十进制，即“逢十进一”的数来表示数据。但在计算机内部，数据是以二进制形式表示的，二进制数只有“0”和“1”两个数字，便于用具有两种稳定状态的物理元件或数字电路中如双稳态电路来保存二进制信息。二进制数运算简便，相应的运算线路也十分简单。为了使用户便于输入输出或书写数据，也常用八进制和十六进制。同一个数用不同的数制表示，这就存在着它们之间的相互转换问题。

一个任意的 R 进制数 N，都可写成：

$$\begin{aligned} N &= K_n K_{n-1} \cdots K_1 K_0 K_{-1} K_{-2} \cdots K_{-m} \\ &= K_n \cdot R^n + K_{n-1} \cdot R^{n-1} + \cdots + K_1 \cdot R^1 + K_0 \cdot R^0 + K_{-1} \cdot R^{-1} \\ &\quad + K_{-2} \cdot R^{-2} + \cdots + K_{-m} \cdot R^{-m} \\ &= \sum_{i=-m}^{n} K_i \cdot R_i \end{aligned}$$

式中 $m$、$n$ 为正整数；$R^i$ 是对应位的位权；R 为 R 进制的基数。所谓基数，就是指在该计数制中每数位 $K_i$ 可能用到的数字符号的个数。每个数位计满 R 后就向高位进位，即“逢 R 进一”，在 R 进制数中相邻两个数位的权相差 R 倍，亦即当小数点向左移一位时，数值缩小 R 倍；而当小数点向右移一位时，数值扩大 R 倍。

基数 R = 2 时，为二进制数，权为 $2^i$，$K_i$ 为 0，1 二个数字中的一个，逢二进位。

基数 R = 8 时，为八进制数，权为 $8^i$，$K_i$ 可为 0，1，…，7 八个数字中的一个，逢八进位。

基数 R = 10 时，为十进制，权为 $10^i$，$K_i$ 可为 0，1，…，9 十个数字中的一个，逢十进位。

基数 R = 16 时，为十六进制数，权为 $16^i$，$K_i$ 可为 0，1，…，9，A，B，C，D，E，F 十六个数字中的一个(A、B、C、D、E、F 与十进制数字的对应关系为：10、11、12、13、14、15)，逢十六进位。

在书写不同进位计数制的数值时，通常在一个数的末尾用一个标识字母来表示该数是什么进位计数制的数。二进制数用字母 B(Binary)，八进制数用字母 O(Octal)，十进制数用字母 D(Decimal)，十六进制数用字母 H(Hexadecimal)。如果数的尾部没有任何字母，那么计算机接收到这样的数就默认为是十进制数。例如：101101B、127O、178D 或 178、3CH 等。

### 1.2.2 各种数制间的相互转换

由于八进制数，十六进制数与二进制数之间有固定的对应关系，从小数点开始分别向左右按每 3 位或按每 4 位二进制数为一组就可以方便地完成八进制、十六进制与二进制之间的相互转换。因此，各种数制间的相互转换主要是十进制与二进制之间的相互转换。这两种数制间的相互转换方法可以很方便地引入到十进制数与八、十六进制数之间的相互转换。

1. R 进制数据转换成十进制数

把任意 R 进制数写成按权展开式后，再求和，就可以得到该 R 进制数对应的十进制数。

例：将十六进制数 3A9.3C 转换成十进制数。

解：$(3A9.3C)_{16} = 3\times16^2 + 10\times16^1 + 9\times16^0 + 3\times16^{-1} + 12\times16^{-2}$

$= 768 + 160 + 9 + 0.1875 + 0.046875$

$= (937.234375)_{10}$

2. 十进制整数转换为二进制整数

有两种转换方法：

(1) 减权定位法

对于二进制数，其多项式可写为：

$K_n2^n + K_{n-1}2^{n-1} + \cdots + K_22^2 + K_12^1 + K_02^0$

多项式中每项的系数 $K_i$ 仅取 0 或 1，由系数组成的一个二进制数：

$K_nK_{n-1}\cdots K_2K_1K_0$

而 $2^n2^{n-1}\cdots2^22^12^0$ 则是每项的位权。若该项系数 $K_i = 0$，则该项的值为 0，若 $K_i = 1$，则该项的值为对应项位权的值 $2^i$。二进制数中各位的权值为：纯整数的最低位为 1，往左每位以 2 的倍率递增。这种权值是很好记的。只要画出下列草图，即可方便地用减法完成转换。

| 位权： | | | … | | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | $D_{10}$ | $D_9$ | $D_8$ | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |

根据二进制的位与权的对应关系可导出十进制整数转换成二进制整数的规律：把待转换的十进制数可能达到二进制的最高位权开始，依次用待转换的十进制数与各位的权值进行比较，如够减，则该位系数 $K_i$(或 $D_i$)确定为 1，同时将待转换的十进制数减去该位权值，用余下的数继续往下比较；若不够减，则该数位的 $K_i$(或 $D_i$)定为 0。依此类推，可进行到所有二进制数位都能给予确定为止。

例如：十进制数 338 转换成二进制数。

从权位表可知 338<512；　$K_8 = 1$

338 − 256 = 82 <128　　；　$K_7 = 0$

而 82 > 64　　；　$K_6 = 1$

82 − 64 = 18<32　　；　$K_5 = 0$

18 > 16　　；　$K_4 = 1$

18 − 16 = 2<8　　；　$K_3 = 0$

2 < 4　　；　$K_2 = 0$

很明显　　；　$K_1 = 1$

；　$K_0 = 0$

经过上述转换，338D = 101010010B。这种转换方法比较直观、简便，且易于验算。尤其十进制数较大时，明显优于“除基取余法”。

(2) 除基取余法

若待转换的十进制数为 S，可写成二进制的多项式形式如下：

$S = K_n2^n + K_{n-1}2^{n-1} + \cdots + K_22^2 + K_12^1 + K_02^0$

上式两边同除以基数 2 后写成：

$S/2 = (K_n2^{n-1} + K_{n-1}2^{n-2} + \cdots + K_22^1 + K_12^0) + K_0/2$

显然，等式右边括号内的数为除 2 的商，$K_0$ 是余数。如余数为 0，则 $K_0 = 0$，余数为 1，则 $K_0 = 1$。这样可以依次判定 $K_1$，$K_2$，…，$K_n$ 等各项系数。

| 除基数 | | 余数 | $K_i$ |
|---|---|---|---|
| 2　338 | | | |
| 2 ⌊ 169 | … | 0 | $K_0$ |
| 2 ⌊ 84 | … | 1 | $K_1$ |
| 2 ⌊ 42 | … | 0 | $K_2$ |
| 2 ⌊ 21 | … | 0 | $K_3$ |
| 2 ⌊ 10 | … | 1 | $K_4$ |
| 2 ⌊ 5 | … | 0 | $K_5$ |
| 2 ⌊ 2 | … | 1 | $K_6$ |
| 2 ⌊ 1 | … | 0 | $K_7$ |
| 0 | … | 1 | $K_8$ |

转换后 338D = 101010010B。这种转换方法操作步骤统一、规范，较方便于用程序设计来实现数制转换。

### 1.2.3　十进制小数转换为二进制小数

类似于整数转换，也有两种转换方法：

1. 减权定位法

与整数转换的减权定位法基本相同，差异在于：①位权值不同。二进制纯小数部分的权值从小数点向右第一位的权值为 0.5，尔后向右每位以除以 2 递缩，如：

| 0.5 | 0.25 | 0.125 | 0.0625 | |
|---|---|---|---|---|
| $K_{-1}$ | $K_{-2}$ | $K_{-3}$ | $K_{-4}$ | … |

小数点

②转换的小数需要根据程序要求或计算机字长来确定小数点的位数。例如：把十进制数 0.875 转换成二进制数的操作步骤为：

0.875 大于 $K_{-1}$ 处的权 0.5，所以 $K_{-1}=1$；$0.875-0.5=0.375$ 大于 $K_{-2}$ 处的权 0.25，所以 $K_{-2}=1$；$0.375-0.25=0.125$ 等于 $K_{-3}$ 处的权 0.125，所以 $K_{-3}=1$。因此，转换的结果为 0.875D = 0.111B。

2. 乘基取整法

设待转换的十进制纯小数 S 写成下列等式：

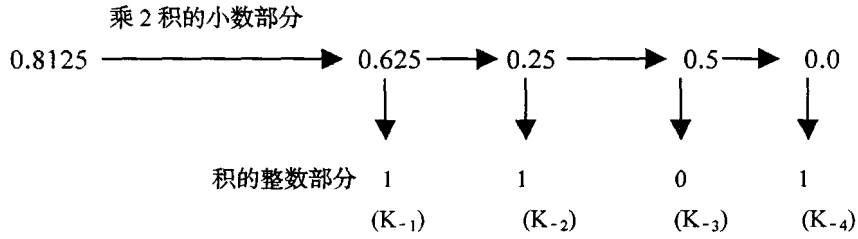
$S=K_{-1}2^{-1}+K_{-2}2^{-2}+\cdots+K_{-m}2^{-m}$

等式两边同乘以基数 2，可得：

$2S=K_{-1}+(K_{-2}2^{-1}+K_{-3}2^{-2}+\cdots+K_{-m}2^{-m+1})$

从上式可见，等式右边括号内为小数部分，而括号外 $K_{-1}$ 为整数部分。也就是说，S 乘以基数 2 后，如整数部分为 1，那么 $K_{-1}=1$，否则 $K_{-1}=0$。重复上述操作，把余下的小数部分继续乘以基数 2，这样就可依次确定 $K_{-2}$，…，$K_{-m}$ 的值。

例如：转换十进制小数 0.8125 为二进制小数的具体操作如下：




转换后的 0.8125D = 0.1101B。

### 1.2.4 二进制整数转换为十进制整数

有两种转换方法：

1. 按权值相加法

首先二进制数各位的系数乘以该位的位权值，尔后把各位的乘积相加即得转换后的十进制数。例如：

$$10110101B=1\times2^7+1\times2^5+1\times2^4+1\times2^2+1\times2^0$$
$$=128+32+16+4+1$$
$$=181D$$

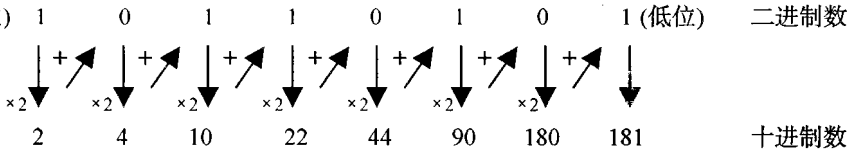
2. 逐次乘基相加法

由于二进制数相邻数位的位权之间的比值是基数 2，因此二进制数的展开多项式可以改写成：

$$K_n2^n+K_{n-1}2^{n-1}+\cdots+K_12^1+K_02^0=(\cdots((K_n\times2+K_{n-1})\times2+K_{n-2})\times2\cdots)\times2+K_0$$

从右边式子可知转换的方法是：从最高位开始，逐次乘以 2 再与次高位的系数相加，所

得结果再乘以 2 并与相邻的低位相加，一直进行到加上最低位为止。其操作示意图如下：




### 1.2.5　二进制小数转换为十进制小数

有两种转换方法

1. 按权相加法

这种方法与整数的按权相加法类似。例如：

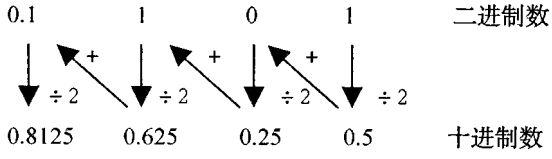
$0.1011B = 1 \times 2^{-1} + 1 \times 2^{-3} + 1 \times 2^{-4}$

$= 0.5 + 0.125 + 0.062\ 5$

$= 0.6875D$

2. 逐次除基相加法

操作的具体步骤是：从小数点后的最低位开始，除以基数 2 后所得数值再与次低位相加，所得结果再除以 2 并又与相邻高位相加，如此继续到小数点后最高位除以 2 为止。例如：0.1101B 转换成十进制数的操作过程可用下面的图解来示意：




转换结果：0.1101B = 0.8125D。

## 1.3　数的符号表示

### 1.3.1　机器数与真值

数值数据可分为无符号数和有符号数两类。无符号数值是：所有的数位都用来表示数据的数值，它没有符号位，为了方便，通常将它作为正数处理。如 n = 8，其数值范围为 0 ~ 255，若 n = 16，其数值范围为 0 ~ 65535。

有符号数分为正、负数两种，为使计算机能够表示和识别，把符号进行数字化，一般规定：用“0”表示正号“ + ”；用“1”表示负号“ - ”。这正好是二进制数中的两个不同数码，符号位放在数值的最高位，即最左边的一位。将符号数码化了的数在机器中的表示形式称为“机器数”，而用“ + ”、“ - ”符号表示再加上绝对值的数值称之为机器数的“真值”。例如：真值 $x_1$ = - 1011011B，其对应的机器数原码为 11011011B；真值 $x_2$ = + 0.1011B，其机器数的原码为 0.1011B。其中，机器数中的小数点实际上是不存在的，这里用“.”仅表示它是小数。

带符号的机器数可以用原码、反码和补码三种不同的码制来表示，由于补码表示法在加、

减运算中的优点，现在多数计算机都是采用补码表示法。

### 1.3.2 数的原码表示

原码表示法是一种比较直观的机器数表示方法。原码与真值的区别仅仅是原码把符号位数字化了。如：－10110 的原码为 110110，＋10110 的原码为 010110。

采用原码作乘、除运算比较方便，可单独处理符号位，数值部分即为绝对值，可直接进行乘、除运算。但对于应用最多的加、减运算，原码表示就不太方便了。

### 1.3.3 数的反码表示

1. 正数的反码与其原码相同。

例如：X = ＋1001B，则

$[X]_{反} = [X]_{原} = 01001B$

2. 负数的反码等于对应正数“按位求反”。

如：X = －1011011B，则

$[-X]_{原} = 01011011B$(即正数)，$[X]_{反} = 10100100B$

### 1.3.4 数的补码表示

补码表示的数在加、减运算中符号位一同参与运算，且可将减法化成加法运算。

1. 正数的补码形式与原码相同。如 X = ＋1001101B，则$[X]_{补} = [X]_{反} = [X]_{原} = 01001101B$。

2. 负数的补码符号位与原码相同，其余各位取反，然后末位加 1；或者说：负数的补码等于对应的正数“按位求反加 1”。

记住以上的规则就不难求一个数的原码、反码和补码了。

### 1.3.5 十进制数的二进制码

十进制数的二进制编码称为 BCD 码。引入 BCD 码的目的是为解决日常习惯的十进制数与机器内的二进制数之间的矛盾，使十进制数与二进制数之间的转换更为方便。最常用的是 8421BCD 码。它对每一位十进制数码用 4 位二进制编码来表示，十进制数的 0～9 分别对应于 0000～1001。

十进制数的 7368 用 8421BCD 码表示为：

0111　0011　0110　1000

BCD 码有压缩 BCD 码和非压缩 BCD 码两种形式，压缩 BCD 码的每个字节存放两位 BCD 码，而非压缩 BCD 码的每个十进制数字占用一个字节的低 4 位，其高 4 位的内容不代表 BCD 的信息。掌握了压缩与非压缩 BCD 码的特点，对于用十进制数进行加减乘除运算的编程就能有的放矢。

### 1.3.6 字符编码

现在计算机中通常采用的字符编码是 ASCII 码(American Standard Code Information Interchange)。它是美国信息交换标准代码，是最主要的字符编码方式。对字符进行编码是进行非数值处理、实现输入输出的基本手段。

标准的 ASCII 码(见附录 A)在一个字节中用七位二进制数表示字符编码，可以用一位(最

高位)表示奇偶校验位(Parity bit)。

标准的 ASCII 码共有 128 个字符，可分为两类：非打印的 ASCII 码和可打印的 ASCII 码。

非打印 ASCII 码：这类编码用于控制性代码，共 33 个，如 BEL(响铃，07H)、DEL(删除，7FH)、CR(回车，0DH)、LF(换行，0AH)等。

可打印 ASCII 码共有 95 个。其中有：

| | |
|---|---|
| 数字 0～9 的编码 | 30H～39H |
| 大写字母 A～Z 的编码 | 41H～5AH |
| 小写字母 a～z 的编码 | 61H～7AH |
| 空格(Space)的编码 | 20H |

以上介绍了无符号数、有符号数、8421BCD 码、ASCII 码。若机器只能识别二进制的补码，那么对 0FFH，CPU 认为它是－1；而程序员又可以怎么看呢？看法有：

(1) 0FFH 看成是 255，即把 0FFH 当成无符号数；

(2) 把 0FFH 看成－127，即把 0FFH 当成有符号数原码；

(3) 把 0FFH 看成－0，即把 0FFH 看成反码。

以上三种看法与机器的默认不一致，程序员必须用相应的指令编程才可识别。

注意，在 FFH 前面加上 0 是为了让 CPU 知道其后是数据，而不是符号名。十六进制数中以 A～F 打头的数前都必须加 0，否则 CPU 无法识别，因为写成 FFH，它可以是变量，也可以是代表立即数的符号。要看具体定义来确定，写成 0FFH 时，它就是数据。

又如 88H，看成补码是－120，看成原码是－8，看成反码是－119，看成压缩的 8421BCD 码那就是 88，除补码之外的其他看法都必须由程序员用相应指令编程来解释。

## 1.4　计算机系统结构

计算机系统结构是指计算机主要部件的特性布局及连接，是汇编语言程序设计者必须了解的计算机资源及其源部件，如地址空间、寄存器组、寻址方式、指令系统等。计算机通常由五大部分组成：控制器、运算器、存储器、输入设备和输出设备。其中把控制器和运算器两部分集成在一个芯片上，称为微处理器，即中央处理器 CPU(Central Processing Unit)，CPU 是整个系统的核心，指令的执行、机器的工作都是由其控制完成的。

80X86 是以 Intel 公司 1978 年推出的 8086 微处理器为基础，作纵横两个方向扩展发展起来的微处理器系列，其 CPU 为 8086/8088、80286、80386、80486 和目前最新的奔腾与高性能奔腾中的一种，配以适当数量的支持芯片，构成功能更为强大的微型计算机系统。

8086 CPU 是 16 位微处理器，采用 16 位数据总线。8088 CPU 是准 16 位的微处理器，采用 8 位数据总线，而使用 16 位内部总线。8086 具有 6 个字节指令流队列，8088 则是 4 个字节。两者指令系统相同，汇编语言一致，除了运行速度外，其他方面无区别。

1. 8086 CPU 结构

8086 CPU 由指令执行部件 EU 与总线接口部件 BIU 两部分组成，其结构示意图如图 1－1 所示。