HOPE

（共二册）

# IBM PC 8086、8088、80286

# 微机原理

# 与宏汇编语言的DOS BIOS程序设计教程

（上册）

北京希望电脑公司

# IBM PC 8086、8088、80286

# 微机原理

# 与宏汇编语言的DOS BIOS程序设计教程

（上册）

钟应木　章忆文　编译

北京希望电脑公司

一九九一年十月

# 前　言

本书以已普及的 80286 为基础，介绍 IBM PC 的系统结构及对各个外设的控制。本书还以最流行的操作系统 DOS 3.3(或以上版本)和标准的 BIOS 为背景，以宏汇编语言(MASM)为工具，介绍对外设的控制编程。内容兼容于 8086、8088 和基础的 80386。

第一部分首先介绍 IBM PC 80286 的数据表示法以及硬件结构。然后说明宏汇编语言的编写、汇编、连接、执行的基础知识。按功能分成数据的传送，简单的四则运算，分支与循环，位运算指令，字串指令，子程序和宏调用，分别介绍汇编语言指令。

第二部分围绕 DOS 系统资源与应用，按不同设备分别介绍 DOS 的系统调用，说明基本字符的输入/输出、日期与时间的管理、驱动器目录管理、使用文件句柄的文件管理、驱动器的管理、进程管理、设备的管理、使用文件控制块的文件管理以及 DOS 其它资源的管理。

第三部分以 BIOS 服务为主线，介绍声音的产生，文本模式下的屏幕处理，图形模式下的屏幕绘图，键盘的 BIOS 服务，打印机的 BIOS 服务，磁盘的 BIOS 服务，时间与日期的 BIOS 服务以及硬盘等其它 BIOS 服务。

在第一部分介绍汇编指令时用丰富的例子说明用法，在第二、第三部分介绍 DOS 和 BIOS 对外设的控制时定义了每个调用的宏，并列举了精巧的宏调用示例。

本书言简意赅，循序渐进，示例丰富，只要稍有一点 PC 或汇编语言知识的人，就可以做为以系统资源控制和应用编程的参考书。它又是一本教程，可以作为授课和培训之教材。

编者

# 目　录

# 第一篇　基本汇编语言程序

本篇内容:

第 1 章: IBM PC 80286 数据的表示法
第 2 章: IBM PC 80286 的硬件结构
第 3 章: 编写、汇编、连接、执行汇编语言的基础知识
第 4 章: 数据的传送
第 5 章: 简单的四则运算
第 6 章: 分支与循环
第 7 章: 位运算指令
第 8 章: 字串指令的应用
第 9 章: 子程序和宏调用的应用
第 10 章: 杂项总结

学习汇编语言的首要工作是，了解数据的表示法和基本的硬件结构，因此本书特将这两部分安排在第一和第二章内说明。另外，我们在本书的第三章说明了编写、汇编、连结、执行汇编语言的基础知识，有了以上概念后，就可正式介绍汇编语言程序设计了。

本书将较常用的汇编语言指令，分门别类的分配在第四章至第九章内。第十章则将较不常用的指令做综合性的说明以期读者能对整个汇编语言结构有一综合性的了解。

当你熟读了本部分指令后，那是不够的，因为本部分只是一个基础，学习汇编语言最终的目的是要能灵活的利用系统资源 DOS 和 BIOS，以达到以软件取代硬件的目标，因此读完本篇之后，应继续阅读本书的第二和第三篇(DOS 和 BIOS 及声音产生)。

# 第一章　IBM PC 80286 数据的表示法

**本章学习目标**

**1：使读者熟悉基本数字，非数字(文字码)的数据表示法。**

**2：读者可了解如何进行各数字系统的转换。**

**本章内容**

**1-1：二进制系统**

**1-2：十六进制系统**

**1-3：二进码十进制(BCD)系统**

**1-4：浮点数数据表示法**

**1-5：字母数字数据的表示法**

当人们互相勾通时，所使用的都是自然语言(Natural language)，像中文、英文、西班牙文等。然而计算机它只能了解 0 与 1，所以当我们想使用计算机时(使用汇编语言 Assembly language)就必须要熟悉二进制数字，除了二进制系统外汇编语言程序设计师还使用 16 进制，BCD 格式与一些字母数字，所以你也必须对此有所了解。

由于计算机内部采用两种状态的装置，因而使得它的结构较为简单，而且它的可靠性也大为提高。通常我们使用 0 与 1 来代表这二种状态，如果我们想要表示一个较大的数字，只需多将几个 0 或 1 串接起来就可以了。

图 1-1 说明计算机利用二进制系统的情形：

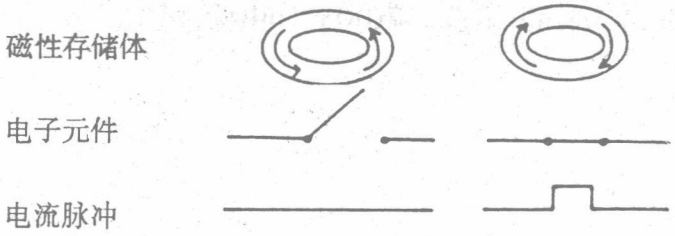



图 1-1　计算机利用二进制数字系统的方式

每一个单一的 0 或是 1 称为一个位(Bit)，位是计算机数据的最小单位，虽然一个位不能提供很多信息，但是若将位串接起来，所能做的事情却令人惊讶!n 位可形成 $2^n$ 个不同的数字，每多加一位，就使得可形成的数字多增加一倍。

在计算机内，任何数据，如字母、数字、标点符号等都是利用这些位串来代表。而同样位字串，我们根据不同的数字格式来解释，可以获得不同的结果。常用的四种数字格式：

1. 二进制(Binary)
2. 十六进制(Hexidecimal)

3. 二进制编码的十进制(BCD)

4. 浮点式(Floating point)。

另一种计算机资料称做字母数字码(Alphanumeric code)，所谓字母数字就是以不同的位组合，分别代表字母和其它各种字符(character)。由于字母数字也包括十进制的十个数字，因此这种码也可用来储存和处理数字上的问题。

## 1.1 二进制系统

数字，其本质上是相当抽象的，因此我们必须利用一些规则和形式将其表现出来，才能对它有较为具体的概念。计算机存贮部件“ON”和“OFF”的设定是与二进制系统有关的二数位，为计算机的基本系统，即1代表ON，0代表OFF。二进位系统又称为一基底为2的系统，这与人类常用的十进制系统不同，因为十进制系统有0到9等10个数位。

诚如规则所述，一位若是在ON状态其值为1，若在是OFF状态其值为0。这看起来好像是个限制，正如十进制只能代表0到9的数位。但是，若你考虑合并十进制数字去形成大于9的数，你也可以合并二进制数去形成大于1的数，因此这就等于没有限制，且可以表示任何大小的数了。

而一个二进制数字的数值是根据每个位的“相对位置”和“1”位存在的位置而定的。下面的8位数含有8个1:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 位 | : | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 位对应值 | : | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

最右边的位代表数值1，而它左边的下一个位代表2，再下一位则代表4，依此类推，最后一个位代表128。以此数为例，这些位的总数值为1+2+4+……+128=255。

对于一个二进制数01000001，其值为1加上64，所以是65。

在计算机的专有名词中，8位的数据单位称为一字节(byte)，使用8位，即一个字节可以表示由0到255个数字。因为字节是最基本的数据处理单位，所以微计算机的存贮容量均以字节数来表示。

$2^{10}$等于1024，1024值通常用字母“K”来表示，因此一计算机有256K即表示此计算机有256x1024(或262144)字节的存贮容量。$2^{10}$等于1024x1024，此值通常用字母“M”来表示，一计算机容量是1M即表示此计算机有1024x1024字节存贮容量。

### 1.1.1 二进制的算术运算

计算机是以二进制方式运算的，所以身为汇编语言程序员必须熟悉二进制格式与算术运算。下面是四种二进制加法的基本格式:

0+0=0

0+1=1

1+0=1

1+1=0　　进位1

对于以上二时制格式了解了以后，我们就可真正处理二进制1000001与00101010的相加了。(若以10进制的观点而言，这是65和42的相加)。

```
      二进制
     01000001          65
   + 00101010        + 42
   ----------        ----
     01101011          107
```

### 1.1.2 带正负号的数字

到目前为止，我们所讨论的都是二进制无正负号的数，因此若一字节所有 8 位皆为 0，该值为 0，若全部是 1，该字节的值为 255。

但是我们常以正值或负值来运算，换句话说是包含正负号的数。当一个字节包含正负号时，则只有 7 位有效位(从 0 到 6)表示数据，最高有效位(位 7，又称符号位)表示该数的正负号，若该数大于或等于零，则符号位为 0，若该数小于零，则符号位是 1。

图 1-2 是有正负号或无正负号的数字表示法：

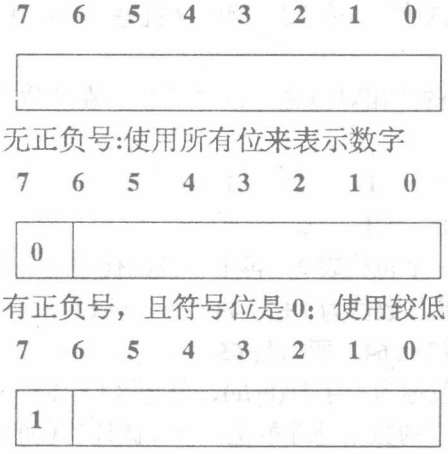



图 1-2 有正负号或无正负号的数字表示法

在 IBM PC 系统中，所有表示负数的方法是采用 2 的补码(2'S Complement)来表示。也就是说，要将一个二进制数字表示成负数，必须把所有位取反(reverse，即 0 变 1，1 变 0)并将取反结果加上 1。现以 65 来做说明：

```
数值65二进制表示   :  01000001
取反               :  10111110
将取反结果加1      :  10111111      (等于-65)
```

若要算出一个负的二进制数字的绝对值，只要重复上述的过程即可。

```
  数值-65二进制表示方式  :  10111111
  取反                   :  01000000
  将取反结果加1          :  01000001      (等于+65)
```

若将+65 与-65 相加，所得结果应为 0，如下所示：

```
  01000001   (+65)
+ 10111111   (-65)
-----------------
(1)00000000
进位 1
```

另一常用运算是减法运算。其实二进制的减法也是一件简单的事：只要将被减数的正负号取反，然后将二数相加就可以了。现我们举 65 减去 42 的例子来做说明：42 的二进制表示法为 00101010。其 2 的补码是 11010110。

```
    65              01000001
+ (-42)             11010110
--------          ----------
    23          (1) 00010111
```

由上可知，所得结果 23 是正确的。

图 1-3 是 4 位带正负号数字的表示法，从此图可知 4 位可代表+7～-8 间所有整数。若将其扩充，可知 1 字节可代表+127～-128 间所有的整数。

| 二进制 | 十进制 | 二进制 | 十进制 |
|---|---|---|---|
| 0000 | 0 | 1000 | -8 |
| 0001 | 1 | 1001 | -7 |
| 0010 | 2 | 1010 | -6 |
| 0011 | 3 | 1011 | -5 |
| 0100 | 4 | 1100 | -4 |
| 0101 | 5 | 1101 | -3 |
| 0110 | 6 | 1110 | -2 |
| 0111 | 7 | 1111 | -1 |

图 1-3　4 位带正负号数字的表示法。

## 1-2　十六进制系统

对计算机而言，二进制算术运算相当好，但是对人类而言，而需要较简捷的表示方法，于是我们便采用了十六进制的表示法。

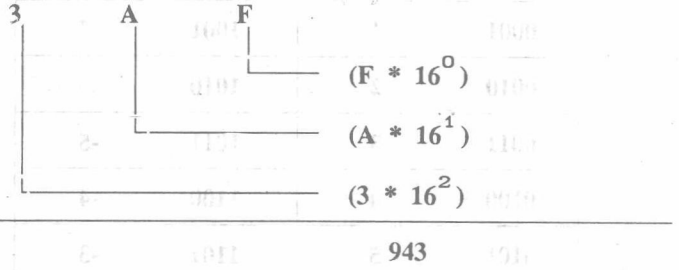
十六进制表示法是以 16 为底的数字系统，其中前面 10 个以 0 到 9 表示，最后 6 个则以 A 到 F 表示。图 1-4 是十六进制的数字表示法。

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

图 1-4　十六进制数字的表示法

由于十六进制系统是以 16 为基底，所以所有数位皆是其右邻数位的 16 倍。即最右边数位值为 $16^0$，其次为 $16^1$，其他依此类推。

例如求十六进制的 3AF 之十进制值:




一般高级语言通常使用十进制来表示数字，汇编语言则使用十六进制来表示数据。例如，存贮体地址和内容，寄存器，指令码等。因此刚开始使用十六进制来设计程序时会有一点困难，但习惯以后就会变得很简单。

## 1-3　BCD 系统

BCD 格式(Binary Code decimal)是 IBM/PC 系统另一种数字的表示法，这种格式对于处理十进制小数点数字非常好用。

所谓 BCD 格式就是用四个二进制位，来表示一个十进制数字，这也是为什么它会被称为“二进制码十进制”。BCD 格式又可分为压缩式(Packed)和非压缩式(Unpacked)两种基本格式。

压缩式 BCD 格式，就是以一连串四位为一组的二进制位，来代表一个十进制数字。例如，十进制的 9502 可以写成:

至于非压缩 BCD 格式，每一个十进制数字，则是存放在字节的低位部分，至于高位部分存放什么东西就不重要了。因此 9502 可以表示成：

```
uuuu1001    uuuu0101    uuuu0000    uuuu0010
    ↑           ↑           ↑           ↑
    9           5           0           2
```

(*u 表示我们并不关心它所放的内容)

储存或执行 BCD 运算时须特别小心，因计算机并不知道这个二进制代码是一 BCD 数字。因此要用 BCD 数字运算，尤其是算术运算，程序员须特别多做一些笔记工作。

## 1-4 十进制浮点数据表示法

储存“浮点数值”与储存有正负号的数值所面临的问题相同。也就是我们必须找出十进制小数点的表示方法。这意谓着我们必须将一个浮点数值分成二部分：十进制小数点的右边为小数为部分，称为“尾数(mantissa)”，十进制小数点的左边称为整数部分。

由于以二进制表示浮点数值的表示法，有许多不同的编码方式，以致美国 IEEE 协会(Institute of Electrical and Electronic Engineer)提出了一个标准格式。IBM PC 使用这个标准格式时，需要一个额外芯片，同时需要额外的指令以完成其运算，这个芯片称为 80287 处理机。

实际上 IEEE 对浮点数值的表示法有二种标准格式：一个使用 32 位格式，另一个使用 64 位的格式，其中使用 32 位的格式称为“短实数格式”(short real format)是 IBM PC 所使用的格式，以下就是短实数格式的说明。

### 1.4.1 短实数格式

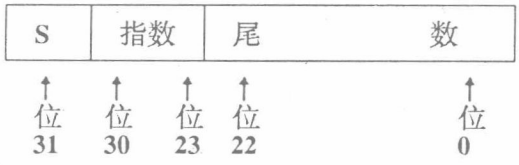
利用短实数格式储存二进制部分，第一个步骤是进行“规范化”(normalization)。这与将十进制小数点数字，转换成我们熟悉的以正负号指数与尾数(mantissa)表示的科学记数法之过程相似。将二进制小数部分规范化时，要将第一个“1”置于小数点的右边。例如：

0.000111101 规范化后的结果是：$0.111101*2^{-3}$

下一步是将规范化后的重要部分以 32 位表示。这些重要部分是：

1. 正负号。
2. 指数部分(基底是 2)。
3. 尾数部分。

IEEE 建议的短实数格式中，正负号储存在最左边的位内，指数部分储存在接着的 8 位内(不是直接存入，而是稍有改变，而后将会说明)，尾数部分则储存在剩下最右边的 23 位内(亦非直接存入，而是稍有改变)。32 位格式如下所示：




1. S 表示正负号：0 表示正号，1 表示负号。