第3章 chapter 3

指令系统

3.1 概　　述

1. 指令的书写格式

指令是CPU执行某种操作的“命令”，CPU全部指令的集合称为指令系统。

指令有两种书写格式：机器指令和符号指令。

机器指令用一串二进制数描述，计算机硬件只能识别、存储和运行机器指令。一般来说，不同系列的CPU完成相同的操作，其机器指令描述是不相同的。机器指令不论在书写、阅读和记忆方面都十分困难，使用机器指令编程是不可想像的，为此引出了另一种表示方法，即符号指令。

符号指令是用规定的助记符和规定的书写格式书写的指令。

表3.1列举了3种操作，同时也写出了完成这些操作的符号指令和机器指令。

表3.1　符号指令与机器指令对照表

操　　作	80486符号指令	80486机器指令
1234H→AX	MOV　AX,1234H	B8　34　12
AX+BX→AX	ADD　AX,BX	03　C3
CX−DX→CX	SUB　CX,DX	2B　CA

其中MOV、ADD、SUB分别是传送指令、加法指令和减法指令的操作码助记符。助记符右侧用逗号间隔的是两个操作数，逗号左边为目标操作数，逗号右边是源操作数。MOV指令的功能是把源操作数传送到目标寄存器中。ADD指令的功能是完成两个操作数相加，结果写入目标寄存器。SUB指令的功能是用目标操作数减源操作数，差值写入目标寄存器。很显然，符号指令比机器指令更容易理解，更容易记忆。

一条符号指令的机器指令有1～16个字节，它们在存储器中是连续存放的，CPU规定：存放指令第一字节的内存地址称为指令地址。

2. 符号指令的书写格式

符号指令的书写格式如下：

标号：操作码助记符 操作数助记符；注释

指令的核心要素是操作码和操作数。

标号代表该条指令的存放地址，它为程序分支、循环提供了转移目标。显然并非每一条指令都要设置标号，标号与符号指令之间用冒号做间隔符，标号的命名规则是：以字母或下划线开头，后跟字母、数字、下划线等，长度不超过 31 个字符。需要注意的是：指令的操作码助记符，伪指令助记符，CPU 寄存器的名称等“系统保留字”不能作为标号使用。

为了阅读方便，可以有注释，注释与符号指令用分号做间隔符，CPU 并不执行，只是在打印源程序清单时，照原样打印出来。

操作数是指令的操作对象，80486 指令操作数的长度可以是单字节、双字节或者四字节。多字节操作数是连续存放的，存放的规则是：低位字节存放在 i 单元中，高位字节存放在相邻的 i+1 单元中，这一规则在以后的程序设计中十分有用，请读者务必注意。

用机器指令编写的程序称为目标程序，用符号指令设计的程序称为符号程序或汇编源程序，汇编源程序要经过编辑、汇编和链接才能生成 CPU 可执行的目标程序。

3.2 80486 寻址方式

前文已叙述过，指令由操作码和操作数两部分组成，操作数是指令的操作对象，在微机系统中操作数有 3 种存放方式，即：

(1) 操作数就包含在本条指令当中，这种操作数，称为立即数。

(2) 操作数存放在 CPU 的某个寄存器中，这种操作数称为寄存器操作数。

(3) 操作数存放在存储器中，这种操作数，称为存储器操作数或内存操作数。

所谓寻址方式，就是在指令格式中用规定的助记符或者助记符表达式(即地址表达式)通知 CPU 怎样计算操作数的地址。

按粗线条划分，80486 有 7 种寻址方式，其中访问存储器有 5 种寻址方式。

3.2.1 立即寻址

立即寻址表示：操作数包含在本条指令中，是指令的一部分，完整地取出该条指令之后也就获得了操作数。在符号指令中如何表示立即寻址？下面列举了几条指令，它们的功能是把源操作数(立即数)写入目标寄存器之中，其中源操作数部分，就是立即寻址。

```
MOV     AL,01010101B
MOV     BX,1234H
MOV     CL,4
MOV     DL,'A'
MOV     BL,0A6H
MOV     CX, 3 * 5
MOV     EAX,12345678H
```

汇编语言规定：立即数必须以数字开头，以字母开头的十六进制数前面必须以数字0做前缀，数制用后缀表示，B表示二进制数，H表示十六进制数，D或者默认表示十进制数，Q表示八进制数，程序员可以用自己习惯的计数制书写立即数。汇编程序在汇编时，对于不同进制的立即数一律汇编成等值的二进制数，用单引号括起来的字符汇编成相应的ASCII码，此外立即数还可以是用＋、－、＊、/表示的算术表达式，汇编程序按照先乘除后加减的规则自动计算，也可以用圆括号改变运算顺序。

3.2.2 寄存器寻址

如果操作数存放在CPU的某个寄存器中，如何通知CPU？在符号指令的操作数部分，写出寄存器的名称即可，下面列举的5条指令，其源操作数、目标操作数，均为寄存器寻址方式。

```
MOV     DS,AX          ;AX寄存器的内容→DS
MOV     CL,BL          ;BL寄存器的内容→CL
INC     SI             ;SI寄存器的内容加1
DEC     DI             ;DI寄存器的内容减1
ADD     EAX,EBX        ;EAX,EBX的内容相加,结果→EAX
```

3.2.3 存储器操作数的寻址方式

本节首先给出“逻辑地址”的概念，然后论述逻辑地址与物理地址的关系，最后介绍存储器操作数的5种寻址方式。

程序设计中最常见的是：操作数存放在某个逻辑段（例如数据段）的存储单元，或者运算结果要写入存储单元，显然CPU要取出（或写入）操作数必须事先知道存放操作数的那个单元的物理地址。由于80x86系列对存储器采用分段管理，所以在指令格式中直接写出存储单元的物理地址是不现实的，程序员只能给出存储单元的逻辑地址。逻辑地址经过汇编，由CPU内部的段页式管理部件最终生成物理地址，然后CPU再根据操作码的要求对该单元进行操作。

逻辑地址包含两部分，一是存储单元所在逻辑段的段基址，二是该单元的偏移地址。逻辑地址的一般书写格式为：

```
段寄存器名称：偏移地址表达式
```

其中，“段寄存器名称：”称为段超越前缀。它表示指令要访问的是哪一个逻辑段。

在实模式下，每个逻辑段首单元的物理地址是一个能被16整除的数，首单元物理地址除以16之后，其商值被称为该逻辑段的段基址。偏移地址是存储单元相对于段首单元的地址偏移量。例如：假设数据段首单元的物理地址为10000H，那么数据段的段基址就是1000H，物理地址为12345H的存储单元，其偏移地址就是2345H。逻辑地址“1000H：2345H”就代表物理地址为12345H的那个内存单元。

在实模式下，段寄存器的内容就是相关逻辑段的段基址，CPU对某一单元进行操作

的时候，首先把该单元所在逻辑段的段寄存器内容乘以16然后加上该单元的偏移地址就得到该单元的物理地址。偏移地址有多种生成方法，每一种方法对应着一种寻址方式，我们的学习重点就是深入理解每一种寻址方式的含义，并且按照汇编语言的要求，规范地写出每种寻址方式的地址表达式。下面详细介绍80486存储器操作数的5种寻址方式。

1. 直接寻址

直接寻址方式有两种书写格式。

(1) 在这种格式中，偏移地址表达式中直接写出存储单元的偏移地址，段超越前缀不能省略，否则将出现寻址错误。例如：

```
MOV     BX,DS:[1234H]   ;取出数据段偏移地址为1234H字单元中的内容→BX
MOV     AL,ES:[2CH]     ;取出ES附加段偏移地址为2CH字节单元的内容→AL
```

这种书写格式，编程时很少使用，因为存储单元的分配是由DOS系统管理的，通常情况下程序员不知道要访问的那个存储单元的偏移地址是多少。

(2) 用变量名代表存储单元的偏移地址。

汇编语言规定，程序员可以为某个存储单元起一个名字，这个名字就称为存储单元的变量名。在一个源程序中，逻辑段之中或者逻辑段之间都不允许有重复定义的变量名，源程序经过汇编之后，存储单元的偏移地址就赋给了变量名，因此，在直接寻址方式中可以用变量名取代偏移地址表达式，又因为变量名是不允许重复定义的，所以段超越前缀可以省略。例如：取出数据段以BUF命名的双字单元的内容→EAX寄存器，可以写成：

```
MOV     EAX,DS: BUF
```

或者写成：

```
MOV     EAX,BUF
```

2. 寄存器间接寻址

寄存器间接寻址(简称间接寻址或间址)是访问存储器最常用的寻址方式，这种寻址方式要求事先把存储单元的偏移地址写入规定的寄存器(为了描述方便，我们称它为间址寄存器)。指令的逻辑地址表达式部分用下列格式描述：

```
段寄存器:[间址寄存器]
```

对于约定的逻辑段其段超越前缀可以省略。例如，假设数据段BUF字节单元有一个操作数N，要求用间接寻址取出N→AL寄存器，在实模式下如何编程？

```
MOV     DS,数据段段基址
        ⋮
MOV     BX,BUF单元的偏移地址
```

```
MOV     AL,[BX]                    ;访问数据段,用 BX 间址取数→AL
```

即：事先把 BUF 单元所在逻辑段的段基址→DS 寄存器，再把 BUF 单元的偏移地址→BX 寄存器，做了这两项准备工作之后，CPU 在执行 MOV AL,[BX]时，首先把 DS 中的段基址乘 16，然后加上 BX 寄存器中的偏移地址从而得出 BUF 单元的物理地址，最后根据物理地址取出该单元的操作数 N→AL 寄存器。

从以上描述可以看出，BX 寄存器中存放的是 BUF 单元的偏移地址，而不是 BUF 单元中的操作数，换句话说，为了取出操作数 N，指令中没有给出偏移地址，而是给出了存放偏移地址的寄存器，这就是寄存器间接寻址的概念。哪些寄存器可以做间址寄存器？80486 规定：

① 可以使用 BP,BX,SI,DI 4 个 16 位的寄存器做间接寻址寄存器，并且规定：使用 BP 寄存器间址，约定访问的是堆栈段，使用 BX、SI、DI 寄存器间址，约定访问的是数据段。

② 也可以使用 EBP、ESP 或者 EAX～EDX、ESI、EDI 这 8 个 32 位的寄存器做间接寻址寄存器，并且规定：使用 EBP、ESP 间接寻址，CPU 约定访问的是堆栈段，使用 EAX～EDX,ESI,EDI 间接寻址，CPU 约定访问的是数据段。例如：

```
MOV     BP,MESG 单元的偏移地址
MOV     CL,ES:[BP]                 ;从 ES 附加段 MESG 字节单元取数→CL
MOV     SI,20H
MOV     EBX,[SI]                   ;从数据段偏移地址 20H～23H 单元取数→EBX
```

3. 基址寻址

在这种方式中，存储单元的偏移地址为规定的基址寄存器的内容与一个常量(即位移量)之和，指令格式中，逻辑地址表达式写成：

```
段寄存器:[基址寄存器+位移量]
```

或者：

```
段寄存器:位移量[基址寄存器]
```

如果是访问约定的逻辑段，则段超越前缀可以省略。哪些寄存器可以做基址寄存器？80486 规定：

(1) 可以用 BP、BX 这两个 16 位的寄存器做基址寄存器，使用 BP 进行基址寻址，约定访问的是堆栈段，使用 BX 进行基址寻址，约定访问的是数据段。

(2) 也可以使用 EBP、ESP、EAX～EDX、ESI、EDI 这 8 个 32 位的寄存器做基址寄存器，若使用 EBP、ESP 进行基址寻址约定访问的是堆栈段，若使用 EAX～EDX、ESI、EDI 进行基址寻址约定访问的是数据段，例如：

```
MOV     BP,BUF 单元的偏移地址
MOV     DL,DS:[BP+10]              ;访问数据段,从 BUF+10 字节单元中取数→DL
MOV     EAX,NUM 单元的偏移地址
```

```
MOV     EDX,[EAX+10H]          ;从数据段 NUM+16～NUM+19 单元取数→EDX
```

4. 变址寻址

变址寻址有两种格式：

（1）有比例因子的变址寻址，在这种格式中：

存储单元的偏移地址＝比例因子×变址寄存器的内容＋位移量

指令格式中，完整的逻辑地址表达式写成如下形式：

```
段寄存器：[比例因子 * 变址寄存器＋位移量]
```

或者

```
段寄存器：位移量[比例因子 * 变址寄存器]
```

访问约定的逻辑段，段超越前缀可以省略。其中比例因子可以是 1、2、4、8 中的一个数，变址寄存器可以是 EBP 或者 EAX～EDX、ESI、EDI 这 7 个 32 位寄存器，并且规定：用 EBP 变址寻址，约定访问的是堆栈段，用 EAX～EDX、ESI、EDI 变址寻址，约定访问的是数据段。

（2）没有比例因子的变址寻址，在这种格式中：

存储单元的偏移地址＝变址寄存器的内容＋位移量

指令格式中，完整的逻辑地址表达式写成如下形式：

```
段寄存器：[变址寄存器＋位移量]
```

或者

```
段寄存器：位移量[变址寄存器]
```

访问约定的逻辑段，段超越前缀可以省略。

在这种格式中，变址寄存器只能选择 SI、DI 这两个 16 位的寄存器，约定访问的是数据段。例如：

```
MOV     AL,[2 * EBX+10]    ;从数据段偏移地址为 2×EBX+10 的单元中取数→AL
MOV     AH,[SI+5]          ;从数据段偏移地址为 SI+5 单元中取数→AH
```

前者是有比例因子的变址寻址，后者是没有比例因子的变址寻址。

5. 基址加变址寻址

基址加变址寻址是基址和变址两种寻址方式的组合，它也有两种格式，区别仅在于地址表达式中是否含有比例因子。

（1）有比例因子的基址加变址寻址，在这种格式中：

存储单元的偏移地址＝基址寄存器内容＋比例因子×变址寄存器的内容＋位移量

指令格式中，完整的逻辑地址表达式写成如下形式：

```
段寄存器：[基址寄存器＋比例因子 * 变址寄存器＋位移量]
```

或者

段寄存器：位移量[基址寄存器＋比例因子＊变址寄存器]

或者

段寄存器：位移量[基址寄存器][比例因子＊变址寄存器]

访问约定的逻辑段，段超越前缀可以省略。

注意：在这种方式中基址寄存器和变址寄存器都必须是规定的32位寄存器。

(2) 没有比例因子的基址加变址寻址，在这种格式中：

存储单元的偏移地址＝基址寄存器内容＋变址寄存器的内容＋位移量

指令格式中，完整的逻辑地址表达式写成如下形式：

段寄存器：[基址寄存器＋变址寄存器＋位移量]

或者

段寄存器：位移量[基址寄存器＋变址寄存器]

或者

段寄存器：位移量[基址寄存器][变址寄存器]

访问约定的逻辑段，段超越前缀可以省略。在这种方式中基址寄存器和变址寄存器都必须是规定的16位寄存器。

【小结】

① 在基址、变址、基址加变址这3种寻址方式中，偏移地址表达式中的位移量是无符号整数。

② 带有比例因子的变址寻址，这种寻址方式常用于检索一维数组元素，当数组元素都是2字节时，比例因子取2，同理当一维数组元素都由4字节长或8字节长的元素组成时，比例因子应选取4或8。

③ 带有比例因子的基址加变址，这种寻址方式常用于检索二维数组元素。

④ 在间址、基址、变址、基址加变址这4种寻址方式中，程序员可以使用16位的寄存器寻址，也可以使用32位的寄存器寻址，但请注意一个问题：当CPU工作在实地址模式的时候，段长度最大为64K，不论你采用16位寄存器寻址还是32位寄存器寻址都必须保证CPU最终算出的偏移地址不超过FFFFH，而且操作数最高字节单元的偏移地址也不能超过FFFFH，否则执行寻址操作时系统将要瘫痪。例如：

```
MOV EBX,10000H      ;EBX 中偏移地址大于 FFFFH
MOV AL,[EBX]        ;执行该指令，系统瘫痪

MOV SI,0FFFFH       ;虽然 SI 中偏移地址不大于 FFFFH
MOV AX,[SI]         ;但[SI]寻址的是双字节数，高字节偏移地址为
                    ;10000H，超出了 FFFFH，执行该指令系统死机
```

3.2.4 80486寻址方式的段约定和段超越

表3.2给出了存储器寻址时规定使用的寄存器，以及约定访问的逻辑段，由于基址加变址寻址是基址寻址和变址寻址的组合，所以表中没有列出基址加变址寻址时规定使用的寄存器。

表3.2 存储器寻址规定使用的寄存器与段约定

间接寻址寄存器	基址寻址寄存器	变址寻址寄存器	约定访问的逻辑段
BP	BP		堆栈段
BX,SI,DI	BX	SI,DI	数据段
EBP,ESP	EBP,ESP	EBP	堆栈段
EAX,EBX,ECX EDX,ESI,EDI	EAX,EBX,ECX EDX,ESI,EDI	EAX,EBX,ECX EDX, ESI,EDI	数据段

表中指出：在用间址、基址、变址、基址加变址访问存储器时，如果使用BP或者EBP、ESP参与寻址，则CPU自动认为这是访问堆栈段，为此在指令的逻辑地址表达式部分，段超越前缀“SS：”可以省略。反之，如果使用BP或者EBP、ESP参与寻址，而程序员真正想访问的是堆栈段之外的其他逻辑段，那么逻辑地址表达式中必须明确写出相关逻辑段的段超越前缀（否则将出现寻址错误），为了简单起见，以间址为例：

```
MOV     AL,DS:[BP]          ;用BP间址访问数据段
MOV     AL,ES:[BP]          ;用BP间址访问ES附加段
MOV     AL,FS:[EBP]         ;用EBP间址访问FS附加段
```

表中还指出，间址寻址使用BX、SI、DI，基址寻址使用BX，变址寻址使用SI、DI，以及使用EAX～EDX、ESI、EDI参与寻址，CPU自动认为这是访问数据段，所以指令的逻辑地址表达式部分，段超越前缀“DS：”可以省略，反之，如果用这些寄存器参与寻址数据段以外的其他逻辑段，必须明确写出相关逻辑段的段超越前缀，例如：

```
MOV     AL,CS:[BX]          ;用BX间址访问代码段
MOV     AL,ES:[SI+5]        ;用SI变址，访问ES附加段
MOV     AL,GS:[EAX+10]      ;用EAX基址寻址，访问GS附加段
```

由于源程序中，不允许出现重复定义的变量名，因此使用变量名直接寻址访问存储器，不需要另加段超越前缀。

此外，执行堆栈操作指令（PUSH，POP，RET…），CPU自动寻址堆栈段，并且自动使用SP（或ESP）的当前值作为偏移地址完成压入或弹出操作，程序员无法用段超越前缀进行干预。

最后要指出的是，绝大多数程序都使用段约定访问存储器，读者应努力使自己习惯这种编程风格，提高程序的可读性。

3.3 80486 标志寄存器

标志寄存器含有两类标志，即状态标志和控制标志。状态标志记录了当前指令执行后的一些状态信息，控制标志控制微处理器操作。

80486 标志寄存器为 32 位结构，实际使用 15 位，共 14 个标志。80486 标志寄存器的位结构如图 3.1 所示。

图 3.1 80486 SX 标志寄存器

下面分别介绍各标志位的功能，以及在一般情况下标志位的置 0/置 1 条件：

1. C 标志(进位/借位标志)

C 标志记录了加/减运算之后，最高位的进位值/借位值。

字节加/减、字加/减、双字加/减时，最高位产生进位/借位，则 C 标志置 1，否则 C 标志置 0，移位和循环移位指令也影响 C 标志。

2. A 标志(辅助进位/借位标志)

A 标志又称半进位/半借位标志。

加/减运算时，D_3 位向 D_4 位有进位/借位，则 A 标志置 1，否则置 0。

3. S 标志(符号标志)

S 标志记录了运算结果的最高位位值。字节运算后结果的 D_7 位为 1，字运算后结果的 D_{15} 位为 1，双字运算后结果的 D_{31} 位为 1 均使 S 标志置 1，否则 S 标志置 0。如果是有符号数运算，S 标志为 0 表示结果为正数，S 标志为 1，表示结果为负数。

4. Z 标志(全零标志)

当运算结果为全零时 Z 标志置 1，否则置 0。

5. P 标志(奇偶性标志)

运算结果的低 8 位中，1 的个数为偶数个则 P 标志置 1，否则 P 标志置 0。在数据通信中，该标志是校验数据传送正确性的一种手段。

6. O 标志(溢出标志)

运算结果产生溢出则 O 标志置 1，否则 O 标志置 0。

溢出标志是很重要的标志，只有正确理解有关溢出的基本概念，才能正确地使用溢

出标志。

（1）什么是溢出

运算结果超出了目标寄存器或目标单元所能表示的范围就称为溢出，例如运算结果出现以下情况之一，都称为溢出。

① 两个 n 位的无符号二进制数相加，结果大于 2^n-1。

② 两个 n 位的有符号二进制数相加，结果大于 $2^{n-1}-1$ 或小于 -2^{n-1}，其中 n 为字长，即 n 为 8、16 或 32。

（2）CPU 如何判断溢出

从以上分析可以看出，溢出和操作数的性质有关。但是操作数的性质是由程序员定义的，计算机硬件无法知道参与运算的操作数是有符号数还是无符号数，因此硬件只能默认一种选择，即参与运算的操作数都是有符号数。做了这样的硬性规定之后，当两数的符号位相同而且和结果的符号位相异时，硬件就自动使 O 标志置 1，否则使 O 标志置 0。

搞清楚溢出标志的置位条件之后，程序员怎样判断溢出呢？在程序设计时，如果程序员定义操作数是有符号数，则运算之后应测试 O 标志，O 标志为 1 表示溢出，如果程序员定义操作数是无符号数，则运算之后应测试 C 标志，C 标志为 1 表示溢出。

7. D 标志（方向标志）

执行 STD 指令则 D 标志置 1，执行 CLD 指令 D 标志置 0。方向标志的作用是：CPU 在执行串操作指令时，控制字符串指针的调整方向，D 标志为 0，进行增址型调整，D 标志为 1，进行减址型调整。

8. I 标志（中断允许标志）

I 标志控制 CPU 是否响应来自引脚 INTR 的可屏蔽中断请求。执行 STI 指令 I 标志置 1，CPU 响应可屏蔽中断；执行 CLI 指令 I 标志置 0，CPU 不响应可屏蔽中断。

9. T 标志（陷阱标志）

T 标志置 1 后，CPU 将进入单步执行方式，即每执行一条指令后都产生一次“单步中断”，若该标志为 0 则 CPU 连续执行指令。指令系统中没有设置使 T 标志置 0/置 1 的指令，需要时可编写一段程序使 T 标志置 0/置 1。

10. IOPL 标志（I/O 特权级标志）

IOPL 标志占两位，表示 0～3 级 4 个 I/O 特权级，0 为最高级，3 为最低级，该标志用于保护模式下的输入输出操作，只有当任务的特权级高于或等于 IOPL 时，执行 I/O 指令才能保证不产生异常。

11. NT 标志（任务嵌套标志）

NT 标志仅用于保护模式。在保护模式下，如果当前执行的 A 任务是嵌套于 B 任务

之中的，则 NT 标志置 1，从而指示 CPU A 任务执行完毕要返回到 B 任务之中。

12. R 标志（恢复标志）

R 标志与调试寄存器配合使用，当 CPU 响应“断点异常中断”时 R 标志置 1，然后标志寄存器压栈再转入相应的断点处理程序，此时若遇到调试故障也不再产生异常中断，断点处理程序结束后返回断点指令。

13. VM 标志（虚拟标志）

如果 CPU 工作在保护模式而 VM 又被置成 1，则 CPU 就转换成虚拟 86 操作模式。

14. AC 标志（对准检查标志）

若 AC 标志为 1，且 CR0 寄存器的 AM 位也为 1，则进行字、双字或四字的对准检查，若要访问的内存操作数未按边界对准则发出异常中断。什么是边界？CPU 规定，访问字操作数应从偶地址开始，访问双字操作数应从 4 的整数倍地址开始，访问四字操作数应从 8 的整数倍地址开始，不符合上述规定就是越界。

在上述 14 个标志当中，C、P、A、Z、S、O 为状态标志，它们记录了当前指令执行后的一些特征信息（并非所有指令对它们都发生影响）为程序转移提供测试条件；D 标志是控制标志，用于串操作指令中控制字符串指针的调整方向；其余标志均属于系统标志。前 9 种是 8086/8088 的标志位，IOPL、NT 是 80286 开始增加的标志位；R、VM 是 80386 开始增加的标志位；AC 为 80486SX 增加的标志位。

3.4 汇编语言基本语法

3.4.1 汇编源程序的语句类型

汇编程序对源程序是以语句为单位进行汇编的，一个完整的汇编源程序至少应包含两类语句：

（1）指令性语句，即通常所说的符号指令；

（2）指示性语句，即伪指令。

指令和伪指令是两种不同的概念。指令是通知 CPU 进行某种操作的命令，由硬件完成其功能。伪指令为汇编程序提供汇编信息，为链接程序提供链接信息，显然伪指令的功能是由相应的软件完成的。

指令性语句的书写格式如下：

```
标号：符号指令；注释
```

它由 3 部分组成：标号及其后跟的冒号、符号指令、分号及其后跟的注释。标号不是必需的，只有当某条指令作为转移指令的目标时，该条指令才需加上标号，标号和指令之间必须用冒号间隔。注释可有可无，CPU 不执行，只是在列出程序清单时，照原样显示，

便于阅读。

指示性语句的书写格式如下：

变量名 伪指令;注释

变量名也不是必需的，一条指示性语句，如果有变量名的话，则变量名与伪指令之间用“空格”做间隔符，这是与指令性语句在书写格式上的区别。

3.4.2 标号、变量和常量

1. 标号和变量

标号和变量的命名规则是：以字母或者下划线开头，后跟字母、数字、下划线，长度不超过31个字符，但系统保留字不能作为标号和变量名。保留字是系统专用的有特殊意义的名字，例如，指令的操作码助记符、运算符、寄存器名称、伪指令助记符等。

标号代表指令地址，它为转移指令提供了转移目标。变量代表内存操作数的存储地址，或者说变量名就代表某个单元。由于标号和变量是用一串字符命名的，从这个意义上讲标号和变量又称为符号地址。

标号被定义在代码段，变量通常被定义在数据段、附加段或堆栈段，故而标号和变量都有3个属性。

(1) 段属性：即标号或变量所在段的段基址，用SEG运算符可以算出。

(2) 偏移属性：即标号或变量所代表的单元，相对于段首址之间的地址偏移量(又称有效地址)，用OFFSET运算符可以求出。

(3) 类型属性。

① 变量的类型有字节型、字型、双字型、四字型等。

用DB伪指令定义的变量，其所属的单元均为字节型。用DW、DD、DQ伪指令定义的变量，其所属的单元分别有字型、双字型和四字型属性。

了解变量的类型是十分重要的，汇编语言规定：读写内存操作数时源目操作数的类型必须一致。变量的类型可以用PTR运算符做临时性的修改。

② 标号的类型有FAR(远)、NEAR(近)两种属性。

如果某个标号是段内转移指令的目标地址，那么这个标号是近程标号，它的类型属性为NEAR(近)，如果某个标号是其他代码段转移指令的目标地址，那么这个标号的类型属性为FAR(远)。

2. 常量

常量包括立即数、字符串常数和符号常数。

(1) 立即数

例如：12,0A8H,10101010B,34Q,－2。

立即数必须以数字开头，以A～F开头的十六进制数必须加前缀数字0。立即数的数制用后缀表示，后缀D(或者省略)表示十进制数，后缀H表示十六进制数，后缀B表示

二进制数，后缀 Q 表示八进制数。程序员可以按自己的习惯书写立即数，经过汇编之后，汇编程序将把各种进制表示的立即数统统转换成等值的二进制数，负数转换成补码。

(2) 字符串常数

用单引号括起来的字符串称为字符串常数。如'A'、'P10'等，经过汇编之后，单引号中的每个字符转换成相应的 ASCII 码，可以像使用立即数一样使用它们，例如：

```
MOV     DL,'1'     ;31H →DL
```

(3) 符号常数

符号常数用伪指令 EQU 或者伪指令＝定义，使用符号常数有利于程序调试，增加程序的可读性，符号常数经过定义之后，也可以像立即数一样使用，例如：

```
COUNT       EQU   55
POINTER      =    2F8H
             ⋮
             MOV   CL,COUNT          ;CL=55
             MOV   DX,POINTER        ;DX=2F8H
```

3.4.3 运算符

1. 数值运算符

(1) 算术运算符有：＋(加)、－(减)、*(乘)、/(除)、MOD(模除)5 种。其中，模除的概念是：做除法取其余数，例如：

```
123     MOD     45     ;结果是 33
```

算术表达式可以作为立即数，它们是在汇编时由汇编程序完成运算的，如果表达式的值超出范围，汇编时给出错误信息，例如：

```
MOV     AX,22*3        ;正确
MOV     AL,22*33       ;错误
```

(2) 逻辑运算符有：NOT(非)、AND(与)、OR(或)、XOR(异或)、HIGH、LOW。其中 HIGH 和 LOW 为分离运算符，HIGH 截取操作数的高 8 位，LOW 截取操作数的低 8 位，例如：

```
X          EQU     11110000B
           ⋮
MOV        AL,X AND 01010101B     ;AL=50H
MOV        AX,HIGH 5566H          ;AX=55H
```

(3) 关系运算符有：EQ(等于)、NE(不等于)、GT(大于)、LT(小于)、GE(大于等于)、LE(小于等于)。

2. 修改属性的运算符

前文讲过，标号和变量都有类型属性，后面还要讲到用 DB 伪指令定义的变量是字节

型变量，用 DW、DD 定义的变量分别是字型和双字型变量。在程序设计中经常遇到这样的问题：某个变量的类型已经定义了，在访问这些变量的时候，为了保证操作数类型匹配，需要改变变量的类型。为此汇编语言提供了 PTR 运算符。PTR 运算符的使用格式如下：

类型说明符　PTR　地址表达式

其中类型说明符有：BYTE（字节）、WORD（字）、DWORD（双字）、FAR（远）、NEAR（近）。地址表达式可以是转移地址标号、过程名或内存操作数的 5 种寻址方式之一。

PTR 运算符只能出现在指令的操作数部分，它的功能是在本条指令中临时修改由地址表达式指向的内存单元、转移地址标号或子程序的属性。

书写指令时，遇到下列情况之一，必须用 PTR 运算符修改或确认内存操作数的类型：

（1）在双操作数指令中（如 MOV、ADD、CMP、AND 等），源为立即数，目标为直接寻址的内存操作数，但源、目类型不一致；源为单字节或双字节立即数，目标是采用间址、基址、变址或基址加变址寻址的内存操作数，这两种情况必须用 PTR 运算符临时修改内存操作数的类型。

（2）在单操作数指令中（如 INC、DEC、NEG 等），操作数为非直接寻址的内存操作数，则内存操作数必须用 PTR 运算符明确说明其类型。

3. 返回属性或数值的运算符

汇编语言中有一些单目运算符，它们加在运算对象之前，可求出对象的某个参数。常用的有以下几个：

（1）SEG 运算符

格式：

```
SEG     段名
```

功能：计算某个逻辑段的段基址。

例如：

```
MOV     AX,SEG DATA
MOV     DS,AX
```

算出段名为 DATA 的逻辑段段基址并赋给 AX，再转赋给 DS 寄存器。用 SEG 运算符也可以求出段中某个变量或标号的段基址。

（2）OFFSET 运算符

格式：

```
OFFSET  变量名或标号名
```

功能：算出变量名（或标号名）所在单元的偏移地址。

例如，假设数据段：

```
BUF     DB      12,34,56
        ⋮
        MOV     AX,SEG 数据段段名
        MOV     DS,AX
        MOV     BX,OFFSET BUF
        MOV     AL,[BX]
        ⋮
```

算出变量 BUF 所在单元的偏移地址并赋给 BX,对数据段用 BX 间址取数送 AL,所以 AL=12。

(3) TYPE 运算符

格式:

```
TYPE  变量名或标号
```

功能:算出变量或标号的类型,对于字节型变量,返回值为 1,字型变量返回值为 2,双字型变量返回值为 4。

(4) $ 运算符

$ 运算符返回汇编计数器的当前值。具体用法见“字节定义伪指令”。

4. 方括号运算符和地址表达式

用方括号括起来的地址表达式,是访问内存操作数常用的寻址方式。方括号的另一个用途是表示数组的下标,下标可以是常数、算术表达式、16 位或 32 位的寻址方式表达式。用下标访问数组元素属于直接寻址范畴。

3.4.4 数据定义伪指令

伪指令、伪语句是汇编语言提供的指示性语句,它为汇编程序和链接程序提供信息。伪指令、伪语句本身不占用内存单元,它们的功能是在汇编和链接时由相应的软件完成的。本节仅介绍最常用的伪指令。

1. 等值伪指令

格式:

```
符号常数  EQU  表达式
```

功能:将表达式的值赋给一个符号常数

例如:

```
XX      EQU   12
COUNT   EQU   55
```

2. 等号伪指令

格式:

符号常数＝表达式

功能：将表达式的值赋给一个符号常数。

例如：

```
XX      ＝12
COUNT＝55
```

说明：用 EQU 伪指令定义的符号常数，其值在后继语句中不能更改，用等号伪指令定义的符号常数，在后继语句中可以重新定义。符号常数一经定义，在后继语句中就可以像立即数一样使用它们。定义语句可以放在任何逻辑段。

3. 字节定义伪指令

格式：

```
变量名  DB  一串用逗号间隔的单字节数
```

例如：

```
BUF1      DB      22H,5＊6,10101010B
          DB      120,－5,0A6H,'HELLO'
COUNT     EQU     $－BUF1
BUF2      DB      ?,?,10 DUP('A')
```

功能与说明：

① DB 是 Define Byte 的缩写，？代表随机数。

② DUP 是 Duplicate 的缩写，译为“重复”，左边是重复系数，右边圆括号中为需要重复设置的数据，10 DUP('A')等价于 10 个用逗号间隔的 41H。

③ $ 代表汇编计数器的当前值，最常见的用法是和等值、等号伪指令配合，紧跟在 DB、DW 等伪指令之后，统计分配给某个变量的单元数，本例 COUNT 等于 11。

④ DB 伪指令的功能是：通知汇编程序，把所定义的单字节数转换成二进制数，并从指定的变量单元开始依次存放。用 DB 伪指令定义的这些单元的属性为“字节型”。

4. 字定义伪指令

格式：

```
变量名    DW  一串用逗号间隔的双字节数
```

例如：

```
WNUM    DW  1234H,56,'AB','C',?
```

功能与说明：

① DW 是 Define Word 的缩写，用 DW 定义的问号为双字节随机数。

② 出现在 DW 伪指令中的字符串常数，单引号中只能是一个或两个字符。

③ DW 伪指令通知汇编程序，把所定义的双字节数从指定变量开始依次存放，每一

个双字节数的存放规律是：低位字节存入低地址单元(i单元)，高位字节存入高地址单元(i+1单元)。

上例经汇编之后，数的存放规律是：WNUM～WNUM+9单元依次为34H、12H、38H、00H、42H、41H、43H、00H、××、××。

④ 被DW定义的这些单元的属性为“字型”。

5. 双字定义伪指令

格式：

变量名　　DD　一串用逗号间隔的4字节数

例如：

DNUM　　DD　?,12345678H

功能：通知汇编程序，把DD定义的数，从指定的变量名开始依次存放，每一个数占4个单元，每一个数的存放规律也是低位字节存入低地址单元，较高字节存入较高的地址单元。被DD定义的这些单元都有“双字型”属性。

6. 多字节定义伪指令

格式：

变量名　DF/DQ/DT　一串用逗号间隔的多字节数

说明：这是高版本汇编程序新增的伪指令，伪指令助记符分别为DF、DQ、DT，它们分别为所定义的每一个数，分配6个、8个、10个单元。

3.5　80486基本集指令

80486指令系统是在8086/8088、80286、80386指令基础上发展形成的。和80286比较，增加了32位操作和访问存储器的32位寻址方式。

80486可以工作在实模式、保护模式和虚拟86模式下，为了支持系统工作模式，指令系统中设计了系统管理指令、保护模式控制指令以及高级语言支持指令等。作为汇编语言程序设计的基础，本书仅介绍80486的基本集指令，有关中断和输入输出的指令在相关章节中介绍。

3.5.1　传送类指令

传送类指令执行后，源操作数不变，不影响状态标志。(标志寄存器传送指令除外)这类指令又可以分为通用传送指令、堆栈操作指令和输入输出指令三类。其中输入输出指令在第7章介绍。

1. 通用传送指令

(1) 数据传送指令

格式：

```
MOV     目标操作数,源操作数
```

功能：把源操作数传送到目标寄存器或目标单元，源操作数不变，不影响状态标志。

说明：

① 源操作数可以是8、16或32位的立即数、寄存器、段寄存器或内存操作数。目标是与源等长的寄存器、段寄存器(CS除外)或内存单元。源、目不能同为内存操作数。

② 源、目操作数类型必须匹配，如果目标是用间址、基址、变址或基址加变址寻址的内存单元，而源是单字节或双字节立即数，则必须用PTR运算符说明目标操作数的属性，例如：

```
MOV     BYTE PTR [BX],12H          ;12H→BX间址的字节型单元
MOV     WORD PTR [SI+5],1234H      ;1234H→SI变址的字型单元
```

③ 不能向段寄存器写入立即数，对段寄存器初始化应借用一个16位的寄存器过渡，例如：

```
MOV     AX,DATA     ;DATA为数据段段名，编译后即为DATA段的段基址
MOV     DS,AX
```

④ 以CS为目标的一切传送指令都是非法的。

(2) 符号扩展传送指令

格式：

```
MOVSX   目标寄存器,源操作数
```

功能与说明：目标为16或32位的寄存器，源是小于(或等于)目标字长的寄存器或内存操作数。

该指令将源操作数的符号位向高位扩展使其与目标字长相同，然后再传送到目标操作数，而源操作数不变，例如：

```
MOV     DL,-16          ;DL=F0H
MOVSX   BX,DL           ;BX=FFF0H,而DH,DL不变
```

(3) 零扩展传送指令

格式：

```
MOVZX   目标寄存器,源操作数
```

功能：与MOVSX类似，只是将源操作数高位用零补足16或32位，然后再传送到目标寄存器。

(4) 偏移地址传送指令

格式：

```
LEA  目标寄存器,源操作数
```

功能与说明：目标为 16 或 32 位的寄存器，源为内存操作数。该指令将内存单元的偏移地址(而不是该单元的内容)传送到目标寄存器中，例如：

```
LEA    BX,BUF        ；将 BUF 单元的偏移地址→BX
LEA    EAX,[SI+5]    ；将数据段采用 SI+5 变址寻址的那个单元的偏移地址→EAX
```

偏移地址传送还可以用 MOV 指令完成，例如：

```
MOV    BX,0FFSET BUF ;BUF 单元的偏移地址→BX
```

OFFSET 是汇编语言提供的运算符，在源程序汇编时完成偏移地址计算，执行该指令时完成赋值。

(5) 指针传送指令

格式：

```
操作码助记符  目标寄存器,源操作数
```

功能与说明：

① 操作码助记符有 LDS、LES、LFS、LGS、LSS，其后两位字母代表段寄存器，它们是隐含的目标寄存器，共有 5 条地址指针传送指令。

② 如果目标是 16 位通用寄存器，则源操作数应是 32 位内存操作数，指令执行后，内存操作数高 16 位→隐含的段寄存器，低 16 位→目标指定的通用寄存器。

③ 如果目标是 32 位通用寄存器，则源操作数应是 48 位内存操作数，指令执行后，内存操作数高 16 位→操作码指定的段寄存器，低 32 位→目标指定的通用寄存器。例如：

设数据段：

```
ADDR1    DF    1234567890ABH
ADDR2    DD    1A2B3C4DH
```

代码段：

```
对 DS 初始化
LES    EBX,ADDR1   ;ES=1234H,EBX=567890ABH
LDS    SI, ADDR2   ;DS=1A2BH,SI=3C4DH
```

(6) 标志寄存器传送指令

格式：

```
LAHF
SAHF
```

功能：LAHF 将标志寄存器低 8 位传送到 AH 寄存器中，SAHF 将 AH 寄存器的内容传送到标志寄存器低 8 位。

（7）交换指令

格式：

XCHG　目标操作数,源操作数

功能与说明：源和目标是等长的寄存器操作数、内存操作数，但不能同为内存操作数，该指令完成源、目标操作数互换。

（8）字节交换指令

格式：

BSWAP　32位寄存器

功能：将32位通用寄存器的位31～位24与位7～位0交换，位23～位16与位15～位8交换。

（9）查表指令

格式：

XLAT　表头变量名

功能：取出DS:[BX+AL]中的一个字节→AL，或者取出DS:[EBX+AL]中的一个字节→AL。

说明：若干单字节数的集合称为字节表，存放第一个表项的内存变量名称为“表头”，该指令查找存放在数据段中的字节表。指令执行前应做两项准备工作：表头偏移地址送BX或EBX；要查找的表项相对于表头的地址位移量送AL，则指令执行时CPU以BX或EBX为基址，以AL值为位移量进行基址寻址取出相应的表元素送AL。

设某数码管显示电路，其八段数码管的字形编码如表3.3所示，要求查找与NUM单元的数对应的字形编码。

表3.3　数码管字形编码表

字形	0	1	2	3	4	5	6	7	8	9	A	b	C	d	E	F
编码(H)	3F	06	5B	4F	66	6D	7D	07	7F	6F	77	7C	39	5E	79	71

首先应在数据段按字形0～F的顺序设置一张字形编码表：

```
TAB    DB      3FH,06H,5BH,4FH,66H,6DH,7DH,07H
       DB      7FH,6FH,77H,7CH,39H,5EH,79H,71H
NUM    DB      ××        ;0～15中的任一数
```

代码段设置如下指令，即可查出和NUM单元数对应的字形编码：

```
:
MOV      BX,OFFSET TAB
MOV      AL,NUM
XLAT     TAB                ;AL=相应的字形编码
```