第3章

8086/8088 指令系统

指令系统是微处理器的基本功能描述，是各种程序设计语言的基础。8086/8088 指令系统是 Intel 80X86 系列微处理器的基本指令集，包括数据传送指令、算术运算指令、逻辑运算指令和移位指令、控制转移指令和处理器控制指令。这些指令是汇编语言程序设计的基础。

3.1 概　　述

程序是人操纵驾驭计算机的工具，程序设计语言是编制程序的工具。程序设计语言主要分为机器语言、汇编语言和高级语言。这些语言的使用环境不同，操纵计算机的方法不同，各有优缺点。

3.1.1 机器语言与汇编语言

机器语言能被计算机硬件直接识别并执行，它由二进制代码组成。机器语言中的每一条称为指令，计算机能够识别的所有指令的集合称为指令系统。指令是计算机能够执行的最小功能单位，机器语言程序就是由一条条的指令按一定顺序组织起来的指令序列。计算机的 CPU 不同，指令系统也不同。Intel 公司的 80X86 系列 CPU，因其硬件结构设计上的包容性，指令系统具有兼容性，用 8088/8086 CPU 的指令系统设计的程序可以在 80X86 系列的 CPU 上执行。8088/8086 CPU 的指令系统常被称为 80X86 系列 CPU 的基础指令。本章以 8086 CPU 为主，介绍常用计算机指令的格式、寻址方式和用法。

一条指令一般由操作码和操作数两部分组成。操作码详细地说明指令要执行的操作，操作数是指令执行时需要的数据。机器语言中操作码和操作数都是二进制代码，因而难于记忆、书写和输入，即使对于计算机的设计者，也一样难于使用。因此，指令中的操作码和操作数用便于记忆的符号代替，编程语言因而有了第一次发展，由机器语言进化到汇编语言。符号化的操作码称为指令助记符，操作数称为操作数助记符。本章以汇编语言指令的形式，介绍 8086CPU 指令集中常用的指令格式、寻址方式和用法。

3.1.2 指令的基本构成

1. 指令的一般格式

一条指令包含操作码和操作数两部分。任何指令都含有操作码，操作数可以有一个，也可以有两个，还可以没有。只有一个操作数的指令常称为单操作数指令，有两个操作数的指令常称为双操作数指令。形式上无操作数的指令，通常操作数是隐含的。操作数有源操作数和目的操作数之分。

如图 3-1 所示，8086 CPU 指令由 1～7 字节组成。操作码占 1～2 字节，操作数占 2～5 字节。操作码的长度取决于指令系统的规模大小。操作数的长度与指令的寻址方式有关。

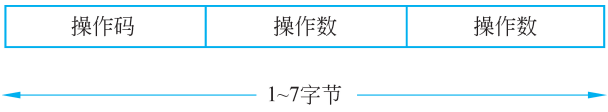


图 3-1 指令的基本组成

2. 操作数类型

8086/8088 CPU 的算术运算单元只能处理整数，不能处理小数，因此本章中所有指令的操作数都是整数，没有小数。8086 CPU 指令的操作数有三种类型：立即数操作数、寄存器操作数和存储器操作数。

- 立即数操作数又称为常数，可以是数值型常数，也可以是字符型常数。数值型常数可以是字节或字类型的整数，可以无符号或有符号。立即数在指令中只能作为源操作数，不能作为目的操作数。例如：

```
MOV   AX, 2023H       ;2023H 是字类型的无符号整数,AX=2023H
MOV   AX,-1423H       ;-1423H 是字类型的带符号整数,AX=EBDDH(补码)
MOV   AH,'A'          ;'A'是字符型常数,将'A'的 ASCII 值 41H 赋给 AH 寄存器
MOV   AL, 6           ;6 是数值型操作数,默认为十进制,指令执行后 AL=6
```

- 寄存器操作数，8086CPU 的 8 个 16 位的通用数据寄存器 AX、BX、CX、DX、SP、BP、SI、DI 和 4 个段寄存器 CS、DS、SS、ES 可作为 16 位寄存器操作数，AH、BH、CH、DH 和 AL、BL、CL、DL 可作为 8 位寄存器操作数。控制寄存器 IP、Flags 只在特定指令中作为操作数。寄存器操作数在指令中可作为源操作数，也可作为目的操作数，段寄存器 CS 除外，它只能作为源操作数。
- 存储器操作数，就是用内存单元中的数据作为操作数。存储器操作数既可以作为源操作数，也可以作为目的操作数，但多数指令要求源和目的操作数不能同时为存储器操作数。指令中的操作数如果是存储器操作数，指令中通常给出的是存储单元的地址或用某种方式指明存储单元的地址，指令执行时 CPU 根据这个地址从存储单元中取出操作数，然后执行指令。存储器操作数的长度类型可以是 1 字节、2 字节(字)或者 4 字节(双字)。寄存器操作数和存储器操作数相对于立即数又称为变数。

数据在内存中以“高高低低”的原则存放，低字节存于低地址内存中，高字节存于高地址内存中。存储器操作数如果是多字节，指令中指明的存储单元地址通常是它的低地址，或称为首地址。如寄存器 AX 中的内容为 6E53H，将它存入 20000H 中，结果如图 3-2 所示。

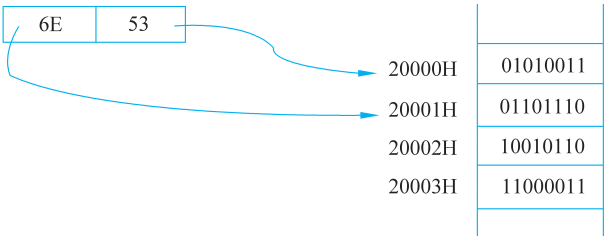


图 3-2　数据存放

3. 指令的书写格式

指令的书写格式如下：

标号：操作码助记符　目的操作数助记符，源操作数助记符；注释

例如：GOON:　　MOV　AX,BX　;数据传送

- 标号是字母、数字组合的符号，代表指令，是指令的地址——用符号表示的地址。标号后跟冒号“：”作为间隔符。不是每条指令都有标号，标号由程序员根据编程需要设定。标号一般由字母开头的字母、数字组成，长度不超过 31 个字符。不允许使用汇编语言中的保留字作标号。
- 操作码助记符与操作数助记符之间至少应有一个空格作为间隔符。如果指令有两个操作数，操作数之间以逗号“,”作为间隔符。紧挨操作码助记符的操作数为目的操作数，另一个为源操作数。
- 操作数助记符与注释之间用分号“;”作为间隔符。注释部分可有可无，可以跟在指令的后面，也可以单独一行，若注释超过一行，则新行以分号“;”开头。简明扼要的注释可以增加程序的可读性，汇编语言程序的可读性很差，应尽量多写注释。
- 指令中的标点符号应为 ACSII 字符。

3.2　寻 址 方 式

8088/8086 CPU 对内存采用分段管理，内存单元地址由段基址和段内偏移地址组成，段基址和段内偏移地址称为内存单元的逻辑地址。指令中存储器操作数的内存单元地址一般给出的是段内偏移地址，段基址以默认方式指明。段内偏移地址，在多数时候指令也不是直接给出，而是给出一种计算偏移地址的公式或方法。这么做的原因一方面是编程需要，如循环程序每次循环处理不同的数据，数据的地址就应该以某种规律变化；另一方面，与数据在内存中的存放形式有关，如一维数组、二维数据等。当然，也与程序员的个人编程习惯有关。实际上，寻址方式就是 CPU 为方便编程而设置的存取存储器操作数的方式方法。

寻址方式，即获得地址的方法，主要指获得段内偏移地址的方法，段基址常采用默认方

式获得。指令的操作数有三种类型：立即数、寄存器操作数和存储器操作数。立即数作为指令的一部分出现在指令中，随着 CPU 取指令的动作进入 CPU 内，不需要再寻址；寄存器操作数本就在 CPU 内部，寄存器的符号就是地址；所以，寻址方式主要解决的是存储器操作数的段内偏移地址的获得方法。操作数的段基址由相应的段寄存器 DS、ES 或 SS 提供，指令中以默认的方式表达。立即数、寄存器操作数，它们不需要寻址，但是，为了理论的统一性，也给它们命名了寻址方式。下面分别介绍 8086 CPU 指令中操作数的寻址方式，共有 8 种。

3.2.1 立即寻址

操作数是立即数，例如：

```
MOV  AL, 76          ;把十进制数 76 传送给 AL,字节(8 位)数据传送
MOV  AX, 2000H       ;把 2000H 传送给 AX,字(16 位)传送
MOV  BX, 34H         ;把 0034H 传送给 BX,字(16 位)传送
MOV  SI, 0           ;把 0000H 传送给 SI,字(16 位)传送
MOV  AH, 'A'         ;把 A 的 ASCII 41H 传送给 AH,字节传送
MOV  AX, 'BA'        ;把 BA 的 ASCII 传送给 AX,AX=4241H,字传送
MOV  CL, 10010010B   ;把二进制数 10010010 传送给 CL,字节传送
ADD  AL, 6           ;寄存器 AL+6,结果回送到 AL 中
```

立即数作为操作数，这个操作数的寻址方式称为立即寻址，其实它不用寻址。立即数操作数只能作源操作数，不能作目的操作数。立即数可以以十进制形式书写，而且默认是十进制，书写时不用加后缀。立即数如果是十六进制，则需要加后缀 H，而且如果是字母开头的十六进制数，汇编语言要求前面加 0，如 MOV　BX,0FE60H 。立即数如果是二进制形式，则后缀要加 B。

3.2.2 直接寻址

直接寻址是存取存储器操作数的方法之一。数据段内存储单元中的数据作为操作数，指令中直接给出操作数所在的内存单元的偏移地址。偏移地址可以是数值形式的地址，也可以是用符号表示的地址，用符号表示的地址称为符号地址。例如，内存单元偏移地址 2000H 中存有数据 55H，在指令中这个操作数的形式为[2000H]，符号[] 表示地址。例如：

```
MOV  BL, [2000H]     ;将偏移地址为 2000H 的存储单元中的数据传送给 BL
MOV  BX, [3200H]     ;将偏移地址 3200H 的连续两个存储单元中的数据传送给 BX
MOV  AL, BUFF        ;将数据段中 BUFF 单元存储的数据传送给 AL
MOV  AX, STR         ;将数据段中 STR 和 STR+1 单元存储的数据传送给 AX
MOV  THERE, BX       ;将 BX 中的数据传送给以 THERE 开始的连续两个单元中
```

操作数[2000H]的寻址方式为直接寻址，如果 2000H 单元存储的数据为 55H，MOV BL，[2000H] 指令执行后 BL＝55H。

```
MOV  BX, [3200H]
```

指令中操作数[3200H]的寻址方式为直接寻址。将以偏移地址3200H为首地址的连续两个内存单元的内容传送给BX寄存器。如图3-3所示，3200H单元存储的数据为F8H，传送给BL寄存器，3201H单元存储的数据为03H，传送给BH寄存器。上面指令执行后BX=03F8H。

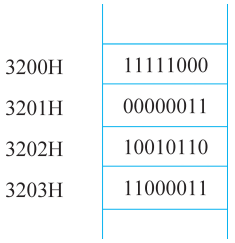


图3-3 直接寻址

注意： BX为16位的寄存器，决定了这条指令为16位的数据传送指令。

上面指令示例中的BUFF、STR、THERE都是存储单元的偏移地址，也称为符号地址，它们应该在程序开始处予以定义，否则不能使用。它们的寻址方式为直接寻址，可以书写为

```
MOV  AL, [BUFF]    ;将 BUFF 存储单元中的数据传送给 AL
MOV  AX, [STR]     ;STR、STR+1 存储单元的数据分别传送给 AL、AH
MOV  [THERE], BX   ;BL、BH 中的数据分别传送给存储单元 THERE、THERE+1
```

更常见的写法为

```
MOV  AL, BUFF
MOV  AX, STR
MOV  THERE, BX
```

8088/8086 CPU对内存采用分段管理，汇编指令中存储器操作数的地址包括段基址和段内偏移地址两部分。上面指令中的[2000H]、[3200H]、BUFF、STR、THERE，都是段内偏移地址，它们的段基址由DS指明。通常情况下，存储器操作数默认在数据段，段基址在DS。在特殊说明的情况下，存储器操作数的段基址也可以替换为CS、ES或SS。表3-1说明了8088/8086 CPU系统中逻辑地址的来源。

表3-1 8088/8086 CPU系统中逻辑地址的来源

操作类型	默认段	可替换的段	偏移地址
取指令	CS	无	IP
堆栈操作	SS	无	SP
BP作基地址	SS	CS,DS,ES	由寻址方式决定
BX作基地址	DS	CS,ES,SS	由寻址方式决定
一般数据读/写	DS	CS,ES,SS	由寻址方式决定
串操作中的源串	DS	CS,ES,SS	SI
串操作中的目的串	ES	无	DI

CPU执行指令时，在BIU中按照段基址x10H+段内偏移地址的方法，计算产生20位的物理地址并送到地址总线上，然后BIU对选中的存储单元进行数据存取。例如：

```
MOV  BL, [2000H]
```

设DS=1200H，指令在执行过程中，首先BIU在地址加法器中进行计算：

```
1200H×10H+2000H=14000H
```

然后从 14000H 内存单元取出 55H 传送给 BL 寄存器。

如果操作数的段基址不是 DS 段，指令要特别说明。例如，在 ES 段，指令应书写为

```
MOV  BL, ES:[2000H]
```

这种用法称为段超越，“:”为段超越符，物理地址的计算方法为 ES×10H＋2000H。

3.2.3 寄存器寻址

操作数在 CPU 内部的寄存器中，即寄存器操作数。8086/8088CPU 的 8 个 16 位的通用数据寄存器 AX、BX、CX、DX、SP、BP、SI、DI 和 4 个段寄存器 CS、DS、SS、ES 可以作为 16 位寄存器操作数，AH、BH、CH、DH 和 AL、BL、CL、DL 可以作为 8 位寄存器操作数。寄存器操作数在指令中可以作为源操作数也可以作为目的操作数，通常源和目的操作数的长度应该保持一致，可以都是字节类型，也可以都是字类型。下面示例中源和目的操作数的寻址方式都是寄存器寻址。

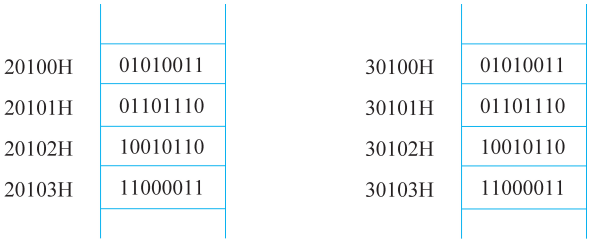
```
MOV  AL, BL    ;把 BL 的内容复制到 AL 中，字节传送
MOV  AH, BH    ;把 BH 的内容复制到 AH 中，字节传送
MOV  CH, CL    ;把 CL 的内容复制到 CH 中，字节传送
MOV  BH, AL    ;把 AL 的内容复制到 BH 中，字节传送
MOV  AX, BX    ;把 BX 的内容复制到 AX 中，字传送
MOV  DX, CX    ;把 CX 的内容复制到 DX 中，字传送
MOV  SP, BP    ;把 BP 的内容复制到 SP 中，字传送
MOV  SI, DI    ;把 DI 的内容复制到 SI 中，字传送
MOV  DI, AX    ;把 AX 的内容复制到 DI 中，字传送
MOV  BX, ES    ;把 ES 的内容复制到 BX 中，字传送
ADD  AX, BX    ;AX+BX 结果保存在 AX 中，16 位加法运算
```

3.2.4 寄存器间接寻址

寄存器间接寻址可以表示任何存储单元的数据。内存单元的偏移地址存放在寄存器中，可以是 SI、DI、SP、BX 寄存器中的任何一个，但不能是其他寄存器。例如，如果偏移地址为 0100H 的内存单元中的数据作为操作数，而且这个地址已经存储在 SI 寄存器中（SI＝0100H），指令中我们可以使用［ SI ］形式表示这个操作数，[ ]表示这是地址。

```
MOV   AL, [SI]     ; 将[SI]表示的存储单元中的数据传送给 AL
```

操作数[SI]的寻址方式为寄存器间接寻址。上述指令的功能为：SI 的内容为内存单元的偏移地址，DS 为段基址（默认），指令执行时 CPU 中的 BIU 按照公式 DS×10H＋SI 计算出存储单元的物理地址并送到地址总线上。参照图 3-4，如果 DS＝2000H，则 DS×10H＋SI＝2000H×10H＋0100H＝20100H，BIU 从物理地址为 20100H 的内存单元中取出数据 53H 送给 EU，EU 将 53H 传送给 AL，指令执行结束。



(a) 数据段中的数据　　(b) 附加段中的数据

图 3.4　段中的数据

注意，[SI]表示存储区域的首地址。因此[SI]可以表示一个内存单元，也可以表示 2 个连续的内存单元，甚至是 4 个连续的内存单元。这一点与直接寻址 [1000H] 的用法一致。

再如：

```
MOV  AX, [SI]    ;将 DS:[SI]指明的连续两个内存单元中的数据传送到 AX
MOV  DX, [DI]    ;将 DS:[DI]指明的连续两个内存单元中的数据传送到 DX
MOV  [BX], AX    ;AX 的内容传送到 DS:[BX]指明的连续两个内存单元中
MOV  CX, [BP]    ;将 SS:[BP]指明的连续两个内存单元中的数据传送到 CX
```

操作数[SI]、[DI]、[BX]、[BP]的寻址方式都是寄存器间接寻址。8086CPU 中能作为寄存器间接寻址方式使用的寄存器只有 4 个：BX、BP、SI、DI。这 4 个寄存器作为间接寻址使用时，要用[ ]申明，这时常称它们为地址指针寄存器或间址寄存器。BP 作为间址寄存器时，段基址默认为 SS，其他 3 个的默认段基址为 DS，但都可以段超越。例如：

```
MOV  DX, ES:[DI]    ;将 ES:[DI]指明的连续两个内存单元中的数据传送到 DX
```

如果 ES＝3000H，DI＝0102H，则 ES×10H＋DI＝3000H×10H＋0102H＝30102H，上述指令的具体功能为：以 30102H 为首地址，从内存中取出连续两个内存单元中的数据传送给 DX，指令执行的结果为：DX＝C396H。

如果某种情况下要求间接寻址操作数必须具有确定的数据长度，要求汇编语言在其前边增加属性操作符 PTR，例如，指令 MOV　[SI]，4 中的两个操作数的长度都不是确定的，CPU 执行指令时是进行字节传送，还是字传送？存在不确定性，这时需要使用 BYTE PTR、WORD PTR、DWORD PTR 这几种伪指令操作符去限定数据类型。例如：

```
MOV  BYTE PTR[SI], 4    ;将 04H 赋值给 SI 表示的内存单元
MOV  WORD PTR[DI], 4    ;将 0004H 赋值给以 DI 为首地址的连续两个内存单元
MOV  AL, 4              ;将 4 赋值给 AL，字节传送，因为 AL 具有确定的数据长度
```

3.2.5　寄存器相对寻址

寄存器相对寻址是定位存储器操作数的另一种方法，是在寄存器间接寻址的基础上，存取跟它有一定距离（位移量）的内存单元中的数据所使用的寻址方式。例如，存取数据表中的数据，可以用间址寄存器存放数据表首地址，用位移量指明要存取表中的哪一个数据。寄存器相对寻址表示的内存单元，它的偏移地址由两部分组成：一部分由间接寻址寄存器提

供;另一部分是指令给定的 8 位或 16 位的地址位移量。二者相加形成操作数的有效地址。例如:

```
MOV  AX, [BX+DATA]          ;将以 BX+DATA 为首地址的连续两个内存单元中的数据
                            ;传送给 AX
MOV  AL, [SI+20H]           ;将以 SI+20H 为地址的内存单元中的数据传送给 AL
MOV  CX, [DI+DATA]          ;将以 DI+DATA 为首地址的连续两个内存单元中的数据传送给 CX
MOV  DX, [BP+DATA]          ;将堆栈段中以 BP+DATA 为首地址的连续两个内存单元中的数据
                            ;传送给 DX
```

上述指令的书写格式很灵活,也可以书写成如下形式:

```
MOV  AX, [BX]+DATA
MOV  AL, 20H [SI]
MOV  CX, DATA [DI]
MOV  DX, DATA+[BP]
```

寄存器相对寻址方式对寄存器的要求与寄存器间接寻址一样,只有 4 个寄存器:BX、BP、SI、DI 可以使用,并且要用[ ]申明。BP 在寄存器相对寻址时,段基址默认为 SS,其他 3 种情况,段基址默认为 DS。

3.2.6 基址变址寻址

基址变址寻址也是以寄存器间接寻址为基础,寻址数组中的数据。它由一个基址寄存器(BX 或 BP)加上一个变址寄存器(SI 或 DI),作为操作数的偏移地址。例如:

```
MOV  AX, [BX][SI]    ;将以 BX+SI 为首地址的连续两个内存单元中的数据送给 AX
MOV  CX, [BP][DI]    ;将以 BP+DI 为首地址的连续两个内存单元中的数据送给 CX
MOV  [BX][DI], AL    ;将 AL 中的数据存入 BX+DI 表示的内存单元中
MOV  [BP][SI], DX    ;将 DX 中的数据存入以 BP+SI 为首地址的连续两个内存单元中
```

8086CPU 中,寄存器 BX 和 BP 为基址寄存器,SI 和 DI 为变址寄存器。这种寻址方式中,一个基址寄存器加一个变址寄存器构成操作数,操作数的形式只有 4 种:

```
[BX][SI]
[BX][DI]
[BP][SI]
[BP][DI]
```

这种寻址方式的段基址是 DS 还是 SS 呢? 8086 汇编规定以基址为主,如果基址寄存器为 BP,则操作数的段基址默认由 SS 提供;若 BX 为基址寄存器,则段基址默认由 DS 提供。例如:

```
MOV   AX, [BX][DI]    ;源操作数的物理地址为 DS×10H+BX+DI
MOV   [BP][DI], DX    ;目的操作数的物理地址为 SS×10H+BP+DI
```

基址变址寻址方式要求必须一个基址和一个变址组合,不允许两个基址或两个变址组合。下面指令是错误的。

```
MOV   AX, [BX][BP]
```

基址变址寻址可以寻址存储器数组中的任何元素。将数组的首地址装入基址寄存器，将要存取元素的序号存入变址寄存器，通过修改变址寄存器中的内容访问数组中的各个元素。

3.2.7 基址变址相对寻址

基址变址相对寻址是在基址变址寻址的基础上增加一个位移量，这种寻址方式常常用来寻址存储器中的二维数组。操作数的地址由基址寄存器加上变址寄存器再加上地址位移量构成。如果基址寄存器为 BP，则段基址默认由 SS 提供；若基址寄存器为 BX，则段基址默认由 DS 提供。例如：

```
MOV   AX, BUFF[BX][SI]          ;将 BUFF+BX+SI 作为偏移地址的连续两个内存单元
                                ;的数据传送给 AX
MOV   STRING[BX][DI], DX        ;将 DX 的内容传送到以 STRING+BX+DI 为偏移地址的
                                ;连续两个内存单元
MOV   CX, [BP+DI+20H]           ;将 BP+DI+20H 作为偏移地址的连续两个内存单元的
                                ;数据传送给 CX
MOV   FILE[BP][SI], AL          ;将 AL 的内容传送到以 FILE+BP+SI 为偏移地址的内存
                                ;单元
```

指令也可以书写为

```
MOV   AX, [BUFF+BX+SI]
MOV   [STRING+BX+DI], DX
MOV   [FILE+BP+SI], AL
```

基址变址相对寻址方式要求必须一个基址和一个变址组合，不允许两个基址或两个变址组合。

这种寻址方式主要用于寻址二维数组中的数据。如果程序中定义了 4 行 10 列的二维数组 FILE，FILE 作为数组首地址，基址寄存器 BX 寻址行，变址寄存器 SI 寻址列，则可以很方便地实现数据阵列检索。

3.2.8 隐含寻址

操作码隐含地指明操作数，例如乘法指令、字符串操作指令。

```
MUL BL     ;AL 乘以 BL，结果存在 AX 中，隐含乘数 AL 和乘积 AX
MOVSB      ;把 DS:SI 指明的内存单元的数据传送到 ES:DI 指明的内存单元
```

3.3 8086 CPU 指令系统

8088 CPU 和 8086 CPU 的指令系统完全相同，共有 133 条基本指令，本节将这些指令

分为 6 个功能组进行介绍。

- 数据传送指令
- 算术运算指令
- 逻辑运算与移位指令
- 串操作指令
- 程序控制指令
- 处理器控制指令

汇编语言的基本语法规则及基本概念，如变量、标号、表达式及运算符等，将在介绍指令的过程中讲解，不单独讲解。在介绍具体指令之前，先介绍一些符号约定：

```
OPRD    各种类型的操作数
src     源操作数
dst     目的操作数
acc     累加器 AX 或 AL
port    输入/输出端口
count   计数器
```

3.3.1 数据传送指令

数据传送指令是汇编语言中使用最频繁的指令，通常细分为通用数据传送指令、输入/输出指令、地址传送指令和标志寄存器传送指令。

1. 通用数据传送指令 MOV，PUSH，POP，XCHG，XLAT

1）MOV

指令格式：

```
MOV  dst,src
```

功能：数据传送，把 src 的内容传送到 dst 中。

说明：把源操作数复制到目标操作数中，它可以实现：

（1）立即数到通用寄存器的数据传送。

例如：

```
MOV  AL,4          ;AL=4
MOV  AX,1000H      ;AX=1000H
MOV  SI,037BH      ;SI=037BH
```

但是

```
MOV  DS,2000H      ;语法错误,不能用立即数给段寄存器赋值
```

应该为

```
MOV  AX,2000
MOV  DS,AX
```

(2) 立即数到存储单元的数据传送。

例如：

```
MOV  WORD PTR[DI], 2000H
```

将立即数 2000H 传送到内存单元，内存单元的地址以间接寻址的方式由 DI 提供。设 DS=3000H，DI=1500H，目的操作数的物理首地址为 31500H，指令执行后如图 3-5 所示。

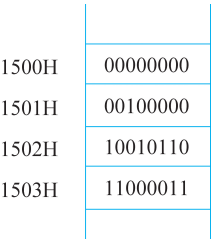


图 3-5　数据传送

PTR 是属性运算符，功能为修改操作数的类型。WORD PTR 的作用是将操作数的类型设置为字类型，BYTE PTR 的作用是将操作数的类型设置为字节类型。例如：

```
MOV  BYTE PTR[SI], 4AH
```

将立即数 4AH 传送到内存单元，内存单元的地址以间接寻址的方式由 SI 提供，传送一字节。

```
MOV  [DI],04AH          ;语法错误：源和目的操作数的类型都不确定
```

这条指令不可用，因为源和目的操作数的类型都不确定，指令执行结果也不确定，这种情况称为二异性。指令执行时可能传送一字节，将立即数 4AH 传送给[DI]指明的内存单元，也可能传送两字节，4A 赋给内存单元[DI]，0 赋给内存单元[DI+1]。

MOV 指令中的两个操作数的类型必须至少有一个是确定的，另一个依附这一个。属性运算符 PTR 帮助确定存储器操作数的类型。

(3) CPU 内部寄存器之间的数据传送。

例如：

```
MOV  AL,BL          ;BL 传送给 AL,传送一字节
MOV  AX,BX          ;BX 传送给 AX,传送一个字
MOV  DS,AX          ;给数据段寄存器赋值
MOV  SI,BP          ;BP 传送给 SI,传送一个字
```

(4) 寄存器与存储单元之间的数据传送。

例如：

```
MOV  AL,[2000H]       ;将 2000H 单元的内容传送给 AL,传送一字节
MOV  AX,[SI]          ;将以 SI 为首地址的连续两个内存单元中的数据传送给 AX
MOV  [3200H], CX      ;将 CL 存入 3200H 单元,将 CH 存入 3201H 单元
MOV  ARRY[DI], DL     ;将 DL 的内容存入 ARRY+DI 的内存单元中
MOV  DL,[BX][SI]      ;将 BX+SI 内存单元中的数据传送给 DL
```

(5) 存储单元之间的数据传送。

不能用一条 MOV 指令实现内存单元之间的数据传送。8086 汇编语法规定 MOV 指令中两个操作数不能同时为存储器操作数。要实现存储单元之间的数据传送，需要两条指令。例如：

```
MOV  [DI], [SI]       ;语法错误
```

应该为

```
MOV  AX,[SI]
MOV  [DI], AX
```

使用 MOV 指令时注意：

(1) MOV 指令不影响标志寄存器的任何标志位。

(2) 源和目的操作数不可同时为存储器操作数。

(3) 源和目的操作数必须等长，即同时为字节类型或字类型。

(4) 不能用立即数给段寄存器赋值。

(5) 不允许给 CS 赋值。

(6) MOV 指令不能访问 IP 和 FLAGS。

2) PUSH，POP

PUSH 和 POP 是堆栈操作指令助记符。堆栈是程序在内存中开辟的一个数据存储区，用于保存函数调用或者中断处理过程中的程序返回地址，或者是程序暂时不用而又必须保存的数据。程序中，堆栈是用段定义语句在内存中定义的一个段，段基址存放在 SS 寄存器，段内偏移地址存放在 SP 寄存器。堆栈段的起始单元称为栈底，堆栈指针 SP 常称为栈顶。

堆栈是一种线性表，只在栈顶一端进行数据输入/输出操作。堆栈操作采用先进后出(或后进先出)的存取方法。数据存入堆栈称为压入堆栈，使用 PUSH 指令。从堆栈中取出数据称为弹出堆栈，使用 POP 指令。程序中，堆栈初始化时，堆栈段起始物理地址为 SS * 10H，段的最大长度为 FFFFH，堆栈段的末地址最大为 SS * 10H＋FFFFH，堆栈指针默认 SP＝0。假设 SS＝2000H，堆栈段如图 3-6 所示。

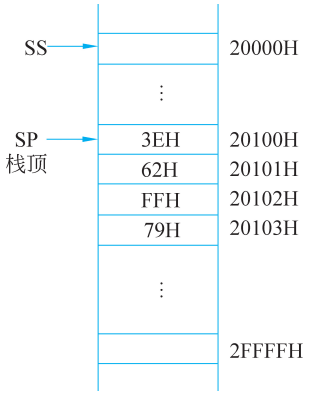


图 3-6　堆栈段

堆栈指令的操作数必须是 16 位的字类型操作数。所以，PUSH 指令执行时，首先将操作数的高字节数据存入 SP-1 单元中，低字节数据存入 SP-2 单元中，然后执行 SP-2 修改堆栈指针。SP 总是指向堆栈中最上层包含数据的存储单元，它称为栈顶。如果连续向堆栈中压入数据，SP 的值越来越小，代表堆栈中可用的存储区域越来越少，栈顶向堆栈段的起始单元方向移动。

图 3-6 中，第一次向堆栈压入数据时，SP＝0000H，假设 DX＝C56EH，执行 PUSH DX 指令，高字节 C5H 存入 SP－1＝FFFFH 单元中，低字节 6EH 存入 SP－2＝ FFFEH 单元中。然后执行操作 SP－2＝SP，即 0000H－2＝FFFEH。堆栈段起始单元物理地址为 20000H，第一次压入的数据存储在堆栈段的末端，物理地址为 2FFFEH 和 2FFFFH 单元，堆栈指针 SP＝FFFEH。

从堆栈中弹出数据时，首先弹出 SP 中的数据(16 位数据中的低 8 位)，然后弹出 SP＋1 中的数据(16 位数据中的高 8 位)，接下来执行 SP＋2 调整指针。数据的弹出过程与压入过程相反，堆栈指针 SP 增大，向堆栈段的末端方向移动。

指令格式：

```
PUSH  src     ;压栈指令
POP   dst     ;出栈指令
```

PUSH 指令把操作数压入堆栈，执行过程为

(1) src 的高 8 位存入[SP-1]，src 的低 8 位存入[SP-2]；

(2) SP-2 送 SP。

例如：

设 SS=2000H，SP=102H，AX=623EH，执行下面的指令后：

```
PUSH  AX
```

AX 的数据 62H 存入 20101H 单元，3EH 存入 20100H 单元，SP=0100H。

POP 指令把操作数弹出到 dst 中，执行过程为

(1) 把[SP]的内容弹出到 dst 的低 8 位，把[SP+1]的内容弹出到 dst 的高 8 位；

(2) SP+2 送 SP。

例如：设 SS=2000H，SP=100H，执行下面的指令后：

```
POP  BX
```

堆栈中 20100H 单元的数据传送给 BL，20101H 单元的数据传送给 BH，BX=623EH，SP=102H。

堆栈操作指令属于单操作数指令，操作数可以是寄存器，也可以是存储器。堆栈指令的操作数必须是字类型，可以是 16 位的通用寄存器或段寄存器，也可以是两个连续的内存单元，可以采用任何寻址方式。8086CPU 不允许立即数作为堆栈操作的操作数，CS 不能作为出栈指令的操作数。堆栈指令不影响任何标志位。

例如：

```
PUSH   DI                 ;将 DI 中的数据压入堆栈
PUSH   SI                 ;将 SI 中的数据压入堆栈
PUSH   DX                 ;将 DX 中的数据压入堆栈
PUSH   DS                 ;将 DS 中的数据压入堆栈
PUSH   CS                 ;将 CS 中的数据压入堆栈
PUSH   WORD PTR[1000H]    ;将以 1000H 单元开始的连续两个单元的内容压入堆栈
PUSH   WORD PTR[SI]       ;将以[SI]单元开始的连续两个单元的内容压入堆栈
PUSH   WORD PTR[BP+6]     ;将以[BP+6]单元开始的连续两个单元的内容压入堆栈
POP    DI                 ;从栈顶中弹出两个单元的数据送给 DI
POP    SI                 ;从栈顶中弹出两个单元的数据送给 SI
POP    DX                 ;从栈顶中弹出两个单元的数据送给 DX
POP    DS                 ;从栈顶中弹出两个单元的数据送给 DS
POP    WORD PTR[1000H]    ;将 SP 单元的数据弹出送给 1000H 单元，SP+1 单
                          ;元的数据弹出送给 1001H 单元
POP    WORD PTR[SI]       ;将 SP 单元的数据弹出送给[SI]单元，SP+1 单元的
                          ;数据弹出送给[SI+1]单元
POP    WORD PTR[BX+DI]    ;将 SP 单元的数据弹出送给[BX+DI]单元，SP+1 单
                          ;元的数据弹出送给[BX+DI+1]单元
```

注意：程序中，压栈操作和出栈操作通常成对出现，以保持堆栈的平衡。

3）交换指令 XCHG

指令格式：XCHG OPRD1,OPRD2

功能：两个操作数交换。

说明：操作数可以是通用寄存器，不能是段寄存器；可以是存储器单元，但两个操作数不能同时为存储器操作数。两个操作数的字长必须相等。

例如：

```
XCHG  AX, BX
XCHG  AL, [SI]
XCHG  [BX+DI], CX
```

4）字节转换指令 XLAT

XLAT 主要用于查表转换和隐含寻址。该指令的功能是将内存单元[BX＋AL]的单字节内容传送给 AL 寄存器，指令执行前后 AL 的内容发生转换。使用 XLAT 指令前，需要预置 DS：BX 指向一张表，BX 作为表的首地址。如果查表中第 9 字节的内容，需要预置 AL 为 9。

指令格式：XLAT

【例 3-1】 DS 数据段中存放有 7 段 LED 显示器表，如图 3-7 所示。TABLE 为表首地址，查表取出 LED 显示器数字 3 的显示字型码。

```
MOV   BX, OFFSET TABLE
MOV   AL, 3
XLAT                ;AL=4FH
OUT   88H, AL       ;LED 显示数字 3
```

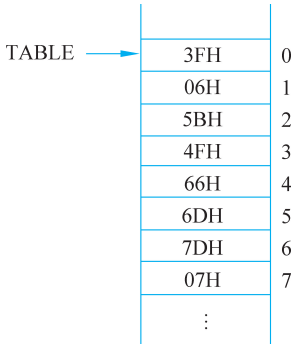


图 3-7　码值表

2. 输入/输出指令 IN/OUT

8086 CPU 对所有输入/输出端口统一管理，提供了一个与内存储器地址空间分开的、完全独立的地址空间，I/O 端口的地址有 8 位和 16 位两种形式。8086 CPU 对 I/O 端口的管理与对内存储器的管理不同，输入/输出指令中操作数的寻址方式也不同，是输入/输出指令特有的。

- 直接端口寻址方式：当端口地址是 8 位的二进制数时，可以在指令中直接使用该地址。
- 寄存器间接寻址方式：当端口地址为 16 位时，不能直接使用，需要预先将其传送到 DX 寄存器中，并且只能是 DX 作为间接寻址寄存器。

8086 CPU 无论从端口输出数据还是输入数据，都要通过累加器 AL 或 AX，所以输入/输出指令又称为累加器专用传送指令。

1）输入指令 IN

格式：

```
IN   ACC, port          ;直接寻址，8 位 port 为立即数端口地址
```

或

```
IN  ACC, DX              ;间接寻址,DX 存有 16 位端口地址
```

例如：

```
IN  AL, 60H              ;从 60H 端口输入一字节
IN  AX, 90H              ;从 90H 端口输入一个字
IN  AL, DX               ;从 DX 端口输入一字节
IN  AX, DX               ;从 DX 端口输入一个字
```

2）输出指令 OUT

格式：

```
OUT  port, ACC           ;直接寻址,port 为 8 位立即数端口地址
```

或

```
OUT  DX,ACC              ;间接寻址,DX 存有 16 位端口地址
```

例如：

```
OUT  60H, AL             ;AL 从 60H 端口输出
OUT  90H, AX             ;AX 从 90H 端口输出
OUT  DX, AL              ;AL 从 DX 端口输出
OUT  DX, AX              ;AX 从 DX 端口输出
```

【例 3-2】 从并行口 0378H 输出一个字符'A'。

```
MOV  DX,0378H
MOV  AL,'A'
OUT  DX,AL
```

3. 地址传送指令 LEA，LDS，LES

地址传送指令共 3 条。

1）取有效地址指令 LEA

指令格式：LEA dst,src

功能：把源操作数的地址偏移量传送到目标操作数。

说明：源操作数必须是存储器操作数，目标操作数必须是 16 位寄存器。

例如：

```
LEA  BX, TABLE           ;TABLE 的偏移地址传送给 BX,TABLE 为符号变量
LEA  SI, DATA[BX]        ;SI=BX+DATA
```

设 BUFF 为符号变量，比较下面两条指令的功能：

```
MOV  DI, OFFSET BUFF
LEA  DI, BUFF
```

上面两条指令的功能完全相同，但 LEA 指令更简洁。OFFSET 为取值运算符，又称为数值返回运算符，用于求变量或标号的属性值。常见的取值运算符还有 SEG。

- SEG 运算符用于求变量或标号所在段的段基址。例如：

```
MOV  AX, SEG BUFF          ;BUFF 的段基址送 AX
```

- OFFSET 用于求变量或标号的偏移地址。例如：

```
MOV  AX, OFFSET BUFF       ;BUFF 的偏移地址送 AX
```

2）取地址指针指令 LDS

指令格式：LDS dst, src

功能：将段基址传送给 DS 寄存器，偏移地址传送给 16 位的地址指针寄存器。

说明：双字操作。从源操作数中连续取出 4 字节。

例如：

```
LDS  BX,ES: [1000H]
```

设从 ES:[1000H]单元起顺序存放 4 字节，如图 3-8 所示，上面指令的执行将 0378H 作为偏移量传送给 BX，将 2000H 作为段基址传送给 DS。

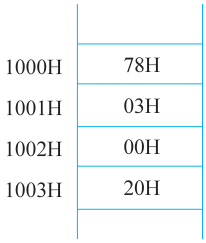


图 3-8　LDS 指令

3）取地址指针指令 LES

与 LDS 指令功能相似，只是段基址装入 ES，而不是 DS。例如：

```
LES  SI,  [1000H]
```

4. 标志寄存器传送指令 LAHF，SAHF，PUSHF，POPF

这 4 条指令都是隐含寻址方式。

1）LAHF 指令

指令格式：LAHF

功能：把标志寄存器 FLAGS 的低 8 位装入 AH 寄存器。

2）SAHF 指令

指令格式：SAHF

功能：把 AH 传送到 FLAGS 的低 8 位，指令影响标志位。

3）PUSHF 指令

指令格式：PUSHF

功能：把 FLAGS 压入堆栈。

4）POPF 指令

指令格式：POPF

功能：从堆栈中弹出两字节传送给 FLAGS，指令影响标志位。

3.3.2 算术运算指令

8088/8086 CPU 提供了加、减、乘、除 4 种基本算术运算指令，可以实现二进制数的运算，也可以实现十进制数的运算，可以实现字节运算，也可以实现字运算，可以进行有符号数运算，也可以进行无符号数运算，有符号数运算以补码形式进行。这 4 种算术运算指令都对标志位产生影响。算术运算指令应尽量使用累加器作操作数。

1. 加法运算指令和调整指令 ADD，ADC，INC，AAA，DAA

1）不带进位的加法运算指令 ADD

ADD 指令完成两个操作数相加，并将结果保存在目的操作数中。

指令格式：ADD OPRD1, OPRD2

功能：操作数 OPRD1 与 OPRD2 相加，结果保存在 OPRD1 中。

说明：操作数 OPRD1 可以是累加器 AL 或 AX，也可以是其他通用寄存器或存储器操作数，OPRD2 可以是累加器、其他通用寄存器或存储器操作数，还可以是立即数。OPRD1 和 OPRD2 不能同时为存储器操作数，不能为段寄存器。ADD 指令的执行对 6 个状态标志位都产生影响。

例如：

```
ADD  AL, BL                ;AL+BL 结果存回 AL 中
ADD  AX, SI                ;AX+SI 结果存回 AX 中
ADD  BX, 3DFH              ;BX+03DFH 结果存回 BX 中
ADD  DX, DATA[BP+SI]       ;DX 与内存单元相加，结果存回 DX 中
ADD  BYTE PTR[DI], 30H     ;内存单元与 30H 相加，结果存回内存单元中
ADD  [BX], AX              ;内存单元[BX]与 AX 相加，结果存回[BX]中
ADD  [BX+SI], AL           ;内存单元与 AL 相加，结果存回内存单元中
```

【例 3-3】 求 D9H 与 6EH 的和，并注明受影响的标志位状态。

```
MOV  AL, 0D9H
MOV  BL, 6EH
ADD  AL, BL
```

$$\begin{array}{r} 11011001 \\ +\ 01101110 \\ \hline 101000111 \end{array}$$

结果 AL＝47H，标志位 CF＝1，PF＝1，AF＝1，ZF＝0，SF＝0，OF＝0。

2）带进位的加法运算指令 ADC

ADC 指令主要用于多字节数的加法运算，以保证低位向高位的进位被正确接收。ADC 指令完成两个操作数相加之后，再加上 Flags 的进位标志 CF 的值，CF 的值可能为 1 或 0。

指令格式：

```
ADC  OPRD1, OPRD2
```

功能：操作数 OPRD1 与 OPRD2 相加后，再加上 CF 的值，结果保存在 OPRD1 中。

说明：对操作数的要求与 ADD 指令一样。

例如：

```
ADC  AL, BL        ;AL+BL+CF 结果存回 AL
ADC  AX, BX        ;AX+BX+CF 结果存回 AX
ADC  [DI], 30H     ;[DI]+30H+CF 结果存回[DI]单元中
```

【例 3-4】 求 E583AD9FH 与 C725BC6EH 的和，结果存放在 DX:AX 中。

```
MOV  AX, 0AD9FH    ;AX=AD9FH
MOV  BX, 0BC6EH    ;BX=BC6EH
ADD  AX, BX        ;AX=6A0DH,CF=1
MOV  DX, 0E583H
MOV  BX, 0C725H
ADC  DX, BX        ;DX=ACA9H,DX:AX=ACA96A0DH,CF=1
```

在多字节数的加法运算中，首先进行低位数相加，再进行高位数相加。最低位相加可以用 ADD 指令，是因为不需要加进位 CF。高位数相加需要使用 ADC 指令，如果低位的加法运算使 CF=1，说明刚才的加法运算有进位，这个进位必须送到高位数中，否则运算结果是错误的。

3）加 1 指令 INC

加 1 指令 INC 又称增量指令，该指令不影响 CF 标志位。

指令格式：INC OPRD

功能：OPRD 加 1 后送回 OPRD。

说明：操作数 OPRD 可以是寄存器或存储器操作数，指令可以完成字节或字的加 1 操作。

例如：

```
INC  AL
INC  AX
INC  BYTE PTR[SI]
INC  WORD PTR[BX+DI]
```

4）十进制数加法调整指令 AAA、DAA

ADD 和 ADC 指令允许 BCD 数作为操作数进行加法运算，按照十进制数的方式完成加法运算。但是，CPU 在完成运算时依然按照二进制数进行，所以在 ADD 或 ADC 指令之后，应进行十进制的调整。

- 指令格式：AAA

功能：将 AL 中的数进行十进制调整，结果保存在 AX 中。

说明：之前的加法指令必须是两个非压缩 BCD 码相加，结果在 AL 中。AAA 指令隐含操作数 AL 和 AH。指令执行时：

（1）若 AL 的低 4 位值大于 9 或辅助进位 AF=1，则将 AL 加 6，将 AH 加 1，并将 AF 和 CF 标志位均置 1。

（2）AL 高 4 位清 0。

- 指令格式：DAA

功能：将 AL 中的数进行十进制调整，结果保存在 AX 中。

说明：之前的加法指令必须是两个压缩 BCD 码相加，结果在 AL 中。AAA 指令隐含

操作数 AL 和 AH。指令执行时：

① 若 AL 的低 4 位值大于 9 或辅助进位 AF=1，则将 AL 加 6，AF 置 1。

② 若 AL 的值大于 9FH 或进位 CF=1，则将 AL 加 60H，CF 置 1。

2. 减法运算指令 SUB，SBB，DEC，NEG，CMP，AAS，DAS

1）不带借位 CF 的减法指令 SUB

指令格式：SUB　OPRD1，OPRD2

功能：操作数 OPRD1 减去 OPRD2，结果保存在 OPRD1 中。

说明：操作数 OPRD1 可以是累加器 AL 或 AX，也可以是其他通用寄存器或存储器操作数，OPRD2 可以是累加器、其他通用寄存器或存储器操作数，还可以是立即数。OPRD1 和 OPRD2 不能同时为存储器操作数，不能为段寄存器。SUB 指令的执行对 6 个状态标志位都产生影响。

例如：

```
SUB  AL, BL                  ;AL-BL,结果存回 AL
SUB  CX, BX                  ;CX-BX,结果存回 CX
SUB  DX, [SI]                ;DX 与[SI]内存单元相减,结果存回 DX
SUB  DATA[BX], CL            ;内存单元的数减去 CL,结果存回内存单元
SUB  BL, 2                   ;BL-2,结果在 BL 中
SUB  WORD PTR[BP+SI],100H    ;内存单元减去 100H,结果存回内存
```

【例 3-5】　D9H 与 6EH 相减，并注明受影响的标志位状态。

```
MOV AL, 0D9H
MOV BL, 6EH
SUB AL, BL
```

```
  1 1 0 1 1 0 0 1
- 0 1 1 0 1 1 1 0
-----------------
  0 1 1 0 1 0 1 1
```

结果 AL=6BH，标志位 CF=0，PF=0，AF=1，ZF=0，SF=0，OF=1。

2）带借位 CF 的减法指令 SBB

指令格式：SBB　OPRD1，OPRD2

功能：操作数 OPRD1 减去 OPRD2 再减去 CF 的值，结果保存在 OPRD1 中。

说明：与 SUB 指令相同，常用于多字节数减法，对 6 个状态标志位都产生影响。

例如：

```
SBB  AL, 30H              ;AL-30H-CF,结果存回 AL
SBB  AX, BX               ;AX-BX-CF,结果存回 AX
SBB  [DI], AH             ;[DI]-AH-CF,结果存回内存单元[DI]中
```

3）减 1 指令 DEC

DEC 指令又称为减量指令，该指令不影响 CF 标志位，对其他 5 个状态标志位产生影响。

指令格式：DEC　OPRD

功能：操作数 OPRD 减 1 后回送 OPRD。

说明：操作数 OPRD 可以是寄存器或存储器操作数，指令可以完成字节或字的减 1

操作。

例如：

```
DEC  CX
DEC  CL
DEC  BYTE PTR [ARRAY+SI]
```

4）操作数求补指令 NEG

指令格式：NEG OPRD

功能：(0－OPRD)结果送回 OPRD，即对 OPRD 包括符号位在内逐位取反后加 1，结果回送到 OPRD。

说明：OPRD 可以是寄存器或存储器操作数。如果操作数非 0，则指令的执行使 CF＝1，否则 CF＝0。NEG 指令的执行对 6 个状态标志位都产生影响。

【例 3-6】
```
MOV AL,31H
NEG AL;     AL=CFH,标志位 CF=1,PF=1,AF=1,ZF=0,SF=1,OF=0
```

5）比较指令 CMP

比较指令 CMP 用于比较两个数的大小，不外乎大于、小于、等于、大于或等于、小于或等于几种情况。指令执行结束后，通过检测状态标志位的值，就可以判定是哪一种情况，程序可以据此设定执行方向，因此，比较指令后面常跟有条件转移指令。

指令格式：

```
CMP   OPRD1, OPRD2
```

功能：OPRD1 减去 OPRD2，结果并不回送给 OPRD1。CMP 指令的执行影响 6 个状态标志位。

说明：指令的执行不影响两个操作数，操作数不变，但影响 6 个状态标志位。OPRD1 可以是寄存器或存储器操作数，OPRD2 可以是立即数、寄存器或存储器操作数。

例如：

```
CMP  AL, AH                ;AL 与 AH 比较
CMP  AX, BX                ;AX 与 BX 比较
CMP  [SI+DATA], AX         ;[SI+DATA]连续两个存储单元的数与 AX 比较
CMP  CL, 8                 ;CL 与 8 比较
CMP  POINTER[BX], 100H     ;[BX+POINTER]连续两个存储单元的数与 100H 比较
```

【例 3-7】
```
        CMP  AL, 0          ;AL 与 0 比较
        JE   EQUA           ;若相等,则转移到 EQUA 处继续执行,否则顺序执行下一条指令
        …
EQUA:   CMP  BL, CL         ;BL 与 CL 比较
        JA   GOON           ;若 BL 大于 CL,则转移到 GOON 处执行,否则顺序执行下一条指令
        …
```

如果希望实现 BL 小于 CL，则转移，可以使用 JB 条件转移指令。如果希望实现 BL 大于或等于 CL，则转移，可以使用 JAE 条件转移指令。如果希望实现 BL 小于或等于 CL，则