第3章 微型计算机的指令系统

计算机是通过执行指令序列来解决问题的，每种计算机都有一套指令集合供用户使用。在微型计算机中，微处理器能执行的各种指令的集合称为指令系统。微处理器的主要功能是由它的指令系统来体现的。在其他条件相同的情况下，指令系统越强，机器的功能也就越强。不同的微处理器有不同的指令系统，其中每一条指令对应着处理器的一种基本操作。

计算机只能识别由二进制编码表示的指令，称为机器指令。一条机器指令应包含两部分内容，其一般格式为：

操作码	操作数

操作码部分指出此指令要完成何种操作；操作数部分则指出参与操作的对象是什么。在指令中可以直接给出操作数的值或者操作数存放在何处，操作的结果应送往何处等信息。处理器可根据指令字中给出的地址信息求出存放操作数的地址，称为有效地址（Effective Address，EA），然后对存放在有效地址中的操作数进行存取操作。

3.1 寻址方式

根据操作数的种类，8086/8088 指令系统的寻址方式分为两大类：数据寻址方式和转移地址寻址方式。

3.1.1 操作数的种类

在 8086/8088 指令系统中，操作数可分为数据操作数和转移地址操作数两大类。

1. 数据操作数

数据操作数是指指令中操作的对象是数据。数据操作数的类型有以下几种。

（1）立即数操作数：指令中要操作的数据在指令中。

（2）寄存器操作数：指令中要操作的数据存放在指定的寄存器中。

（3）存储器操作数：指令中要操作的数据存放在指定的存储单元中。

（4）I/O 操作数：指令中要操作的数据来自或送到 I/O 端口。

对于数据操作数，有的指令只有一个操作数或没有操作数。而有的指令有两个操作数，一个称为源操作数，在操作过程中保持原值不变；另一个称为目标操作数，操作后一般被操作结果所替代。

另外，还有一种隐含操作数。这种操作数从指令格式上看，好像是没有操作数，或只有

一个操作数，实际上隐含了一个或两个操作数。

2. 地址操作数

地址操作数是指指令中操作的对象是地址。其指令只有一个目标操作数，该操作数不是普通数据，而是要转移的目标地址。它也可以分为立即数操作数、寄存器操作数和存储器操作数，即要转移的目标地址包含在指令中，或存放在寄存器中，或存放在存储单元之中。

3.1.2　8086/8088 的机器代码格式

8086/8088 CPU 的机器代码格式如图 3.1 所示。

1/2 字节	0/1 字节	0/1/2 字节	0/1/2 字节
操作码	mod reg r/m	位移量	立即数

图 3.1　8086/8088 的机器代码格式

操作码占 1 或 2 字节，后面的各字节指明操作数。其中，“mod reg r/m”字节表明寻找操作数的方式（即采用的寻址方式），“位移量”字节给出某些寻址方式需要的相对基地址的偏移量，“立即数”字节给出立即寻址方式需要的数值本身。由于设计有多种寻址方式，操作数的各个字段有多种组合，表 3.1 给出了各种组合情况。

表 3.1　8086/8088 指令的寻址方式字节编码

r/m	mod					reg
	00	01	10	11		
				w=0	w=1	
000	[BX+SI]	[BX+SI+D8]	[BX+SI+D16]	AL	AX	000
001	[BX+DI]	[BX+DI+D8]	[BX+DI+D16]	CL	CX	001
010	[BP+SI]	[BP+SI+D8]	[BP+SI+D16]	DL	DX	010
011	[BP+DI]	[BP+DI+D8]	[BP+DI+D16]	BL	BX	011
100	[SI]	[SI+D8]	[SI+D16]	AH	SP	100
101	[DI]	[DI+D8]	[DI+D16]	CH	BP	101
110	[D16]	[BP+D8]	[BP+D16]	DH	SI	110
111	[BX]	[BX+D8]	[BX+D16]	BH	DI	111

8086/8088 指令最多可以有两个操作数。在“mod reg r/m”字节中，reg 字段表示一个采用寄存器寻址的操作数，reg 占用 3 位，不同编码指示 8 个 8 位（w=0）或 16 位（w=1）通用寄存器之一；mod 和 r/m 字段表示另一个操作数的寻址方式，分别占用 2 位或 3 位。

（1）mod =00 时为无位移量的存储器寻址方式。但其中，当 r/m =110 时为直接寻址方式，此时该字节后跟 16 位有效地址 D16。

（2）mod =01 时为带有 8 位位移量的存储器寻址方式。此时该字节后跟一个字节，表示 8 位位移量 D8，它是一个有符号数。

（3）mod =10 时为带有 16 位位移量的存储器寻址方式。此时该字节后跟两个字节，表示 16 位位移量 D16，它也是一个有符号数。

（4）mod =11 时为寄存器寻址方式，由 r/m 指定寄存器，此时的编码与 reg 相同。

3.1.3 与数据有关的寻址方式

指令中关于如何求出存放操作数有效地址的方法称为操作数的寻址方式。计算机按照指令给出的寻址方式求出操作数有效地址和存取操作数的过程，称为寻址操作。

在微机中，寻址方式可能有如下3种情况。

（1）操作数包含在指令中，称为立即寻址。

（2）操作数包含在CPU的内部寄存器中，称为寄存器寻址。

（3）操作数在内存的数据区中，这时指令中的操作数字段包含着此操作数的地址，称为存储器寻址。

为了更清楚地掌握寻址方式，下面以最常用的MOV指令来举例说明各种寻址方式的功能。MOV指令是一个数据传送指令，相当于高级语言的赋值语句，其格式为：

```
MOV  OPRD1,OPRD2
```

MOV指令的功能是将源操作数OPRD2传送至目的操作数OPRD1。在讲述中，假设目的操作数采用寄存器寻址方式，用源操作数来反映各种寻址方式的功能和彼此的区别。

1. 立即数寻址方式

立即数寻址方式所提供的操作数紧跟在操作码的后面，与操作码一起放在指令代码段中。立即数可以是8位数或16位数。如果是16位数，则低位字节存放在低地址中，高位字节存放在高地址中。

立即数寻址方式只能用于源操作数字段，不能用于目的操作数字段，经常用于给寄存器赋初值。

【例3.1】 将8位立即数18存入寄存器AL中。

```
MOV  AL,18
```

指令执行后，(AL)= 12H

【例3.2】 将16位立即数8090H存入寄存器AX中。

```
MOV  AX,8090H
```

图3.2给出了立即数寻址方式示意图。指令执行后的结果为：(AX)= 8090H。

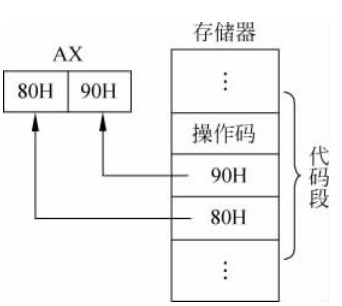

图3.2 立即数寻址方式示意图

2. 寄存器寻址方式

在寄存器寻址方式中，操作数包含于CPU的内部寄存器之中。这种寻址方式大都用于寄存器之间的数据传输。对于16位操作数，寄存器可以是AX、BX、CX、DX、SI、DI、SP和BP等；对于8位操作数，寄存器可以是AL、AH、BL、BH、CL、CH、DL和DH。这种寻址方式可以取得较高的运算速度。下列指令都属于寄存器寻址方式：

```
MOV  DS,AX
MOV  AL,CL
MOV  SI,AX
```

```
MOV  BL,AH
```

【例 3.3】　寄存器寻址方式的指令操作过程如下。

```
MOV  AX,BX
```

如果指令执行前(AX)＝ 6688H,(BX)＝ 1020H；则指令执行后,(AX)＝ 1020H,(BX)保持不变,图 3.3 给出了寄存器寻址方式示意图。

3. 存储器寻址方式

寄存器寻址虽然速度较快,但 CPU 中寄存器数目有限,不可能把所有参与运算的数据都存放在寄存器中。多数情况下,操作数还是要存储在主存中。如何寻址主存中存储的操作数称为存储器寻址方式,也称为主存寻址方式。在这种寻址方式下,指令中给出的是有关操作数的主存地址信息。由于 8086/8088 的存储器是分段管理的,因此这里给出的地址只是偏移地址(即有效地址 EA),而段地址在默认的或用段超越前缀指定的段寄存器中。

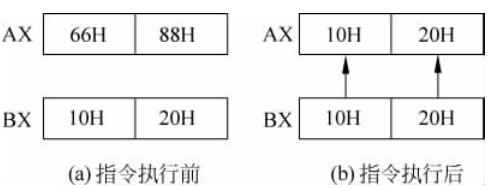

图 3.3　寄存器寻址方式示意图

为了方便各种数据结构的存取,8086/8088 设计了多种主存寻址方式。

1) 直接寻址方式

直接寻址方式是操作数地址的 16 位偏移量直接包含在指令中,和指令操作码一起放在代码段,而操作数则在数据段中。操作数的地址是数据段寄存器 DS 中的内容左移 4 位后,加上指令给定的 16 位地址偏移量。把操作数的偏移地址称为有效地址 EA,则物理地址＝16D×(DS)＋EA。如果数据存放在数据段以外的其他段中,则在计算物理地址时应使用指定的段寄存器。直接寻址方式适合于处理单个数据变量。

【例 3.4】　直接寻址方式示例如下。

```
MOV  AX,[400H]
```

如果(DS)＝ 1000H,则物理地址的计算式为：

10000H(段基地址)＋ 400H(偏移地址)＝ 10400H(物理地址)

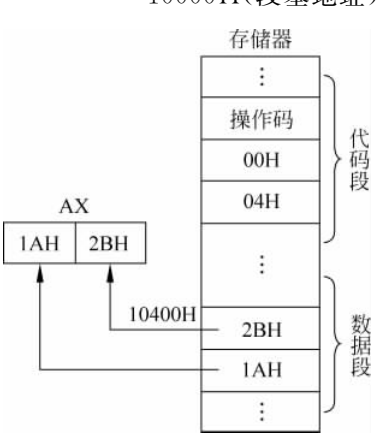

图 3.4　直接寻址方式示意图

图 3.4 给出了直接寻址方式示意图。指令执行后的结果为：(AX)＝ 1A2BH。

也可以用符号地址代替数值地址,例如：

```
MOV  AX,BUFFER
```

此时 BUFFER 为存放数据单元的符号地址。它等效于如下形式：

```
MOV  AX,[BUFFER]
```

2) 寄存器间接寻址方式

在寄存器间接寻址方式中,操作数在存储器中。操作数的有效地址由变址寄存器 SI、DI 或基址寄存器

BX、BP 提供。这又分成两种情况：

如果指令中指定的寄存器是 BX、SI、DI，则用 DS 寄存器的内容作为段地址，即操作数的物理地址为：

```
物理地址 = 10H ×(DS) + (BX、SI 或 DI)
```

如指令中用 BP 寄存器，则操作数的段地址在 SS 中，即堆栈段，所以操作数的物理地址为：

```
物理地址 = 10H ×(SS) + (BP)
```

【例 3.5】 寄存器间接寻址方式示例如下。

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

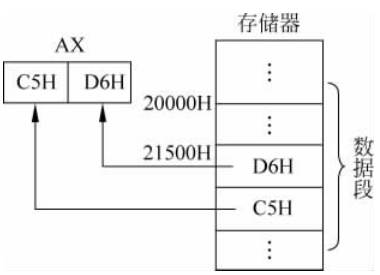

图 3.5 寄存器间接寻址方式示意图

如果(DS)＝2000H，(SI)＝1500H，则物理地址的计算式为：

20000H(段基地址)＋1500H(偏移地址)

＝21500H(物理地址)

图 3.5 给出了寄存器间接寻址方式示意图。指令执行后的结果为：(AX)＝ C5D6H。

3）寄存器相对寻址方式

该寻址方式是以指定的寄存器内容，加上指令中给出的位移量(8 位或 16 位)，并以一个段寄存器为基准，作为操作数的地址。指定的寄存器一般是一个基址寄存器或变址寄存器。即：

$$EA=\left\{\begin{matrix}(BX)\\(BP)\\(SI)\\(DI)\end{matrix}\right\}+\left\{\begin{matrix}8\text{ 位}\\16\text{ 位}\end{matrix}\quad \text{位移量}\right\}$$

与寄存器间接寻址方式类似，对于寄存器为 BX、SI、DI 的情况，段寄存器用 DS，则物理地址为：

```
物理地址 = 10H ×(DS) + (BX、SI 或 DI) + 8 位(或 16 位)位移量
```

当寄存器为 BP 时，则使用 SS 段寄存器的内容作为段地址，此时物理地址为：

```
物理地址 = 10H ×(SS) + (BP) + 8 位(或 16 位)位移量
```

【例 3.6】 寄存器相对寻址方式示例如下。

```
MOV  AX,DISP[DI]
```

也可表示为

```
MOV  AX,[DISP + DI]
```

其中 DISP 为 16 位位移量的符号地址。

如果(DS)＝ 3000H，(DI)＝ 2000H，DISP ＝ 600H，则物理地址的计算式为：

30000H(段基地址)＋ 2000H(变址)＋ 600H(位移量) ＝32600H(物理地址)

图 3.6 给出了寄存器相对寻址方式示意图,指令执行后的结果是:(AX)＝ 005AH。

4) 基址加变址寻址方式

在基址加变址寻址方式中,通常把 BX 和 BP 看作是基址寄存器,把 SI 和 DI 看作变址寄存器,可把两种方式组合起来形成一种新的寻址方式。基址加变址的寻址方式是把一个基址寄存器 BX 或 BP 的内容,加上变址寄存器 SI 或 DI 的内容,并以一个段寄存器作为地址基准,作为操作数的地址。

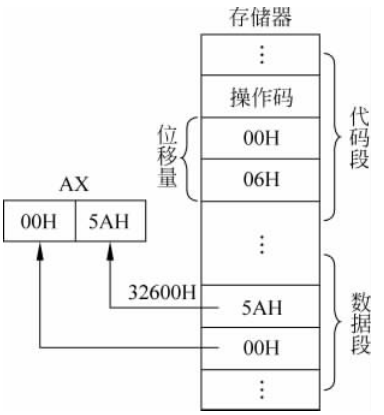

图 3.6　寄存器相对寻址方式示意图

两个寄存器均由指令指定。当基址寄存器为 BX 时,段寄存器使用 DS,则物理地址为:

```
物理地址 = 10H ×(DS) + (BX) + (SI 或 DI)
```

当基址寄存器为 BP 时,段寄存器用 SS,此时物理地址为:

```
物理地址 = 10H ×(SS) + (BP) + (SI 或 DI)
```

【例 3.7】 基址加变址寻址方式示例如下。

```
MOV  AX,[BX][DI]
```

或写为:

```
MOV  AX,[BX + DI]
```

如果(DS)＝ 1000H,(BX)＝ 2000H,(DI)＝ 3000H,则物理地址的计算式为:

10000H(段基地址)＋ 2000H(基址)＋ 3000H(变址) ＝ 15000H(物理地址)

图 3.7 给出了基址加变址寻址方式示意图。指令执行后的结果为:(AX)＝ 1288H。

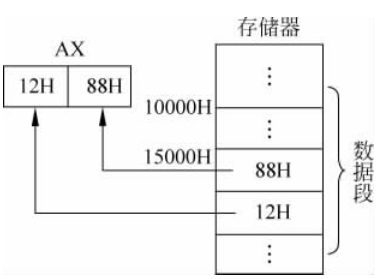

图 3.7　基址加变址寻址方式示意图

5) 相对基址变址寻址方式

在相对基址变址寻址方式中,通常把 BX 和 BP 看作是基址寄存器,把 SI 和 DI 看作变址寄存器。它是把一个基址寄存器 BX 或 BP 的内容,加上变址寄存器 SI 或 DI 的内容,再加上指令中给定的 8 位或 16 位位移量,并以一个段寄存器为地址基准,作为操作数的地址。同样,当基址寄存器为 BX 时,段寄存器使用 DS,则物理地址为:

```
物理地址 = 10H ×(DS) + (BX) + (SI 或 DI) + 8 位(或 16 位)位移量
```

当基址寄存器为 BP 时,段寄存器则用 SS。此时物理地址为:

```
物理地址 = 10H ×(SS) + (BP) + (SI 或 DI) + 8 位(或 16 位)位移量
```

【例 3.8】 相对基址变址寻址方式示例如下。

```
MOV  AX,DISP[BX][SI]
```

如果(DS)= 4000H,(BX)= 2000H,(SI)= 1000H,DISP = 800H,则物理地址的计算式为：

EA =2000H(基址)+1000H(变址)+800H(位移量)=3800H

40000H(段基地址)+ 3800H(EA)= 43800H(物理地址)

图 3.8 给出了相对基址变址寻址方式示意图。指令执行后的结果为：(AX)= EEFFH。

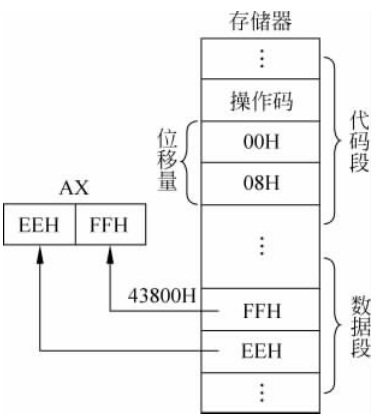

图 3.8 相对基址变址寻址方式示意图

3.1.4 与转移地址有关的寻址方式

微机指令系统中有转移指令及子程序调用等非顺序执行指令。这类指令所指出的地址是程序转移到的指定的转移地址，然后再依次顺序执行程序。这种提供转移地址的方法称为程序转移地址的寻址方式。它分为段内直接寻址、段内间接寻址、段间直接寻址和段间间接寻址 4 种情况。在 8086/8088 指令系统中，条件转移指令只能使用段内直接寻址方式，且位移量为 8 位；而非条件转移指令和子程序调用指令则可用 4 种寻址方式中的任何一种。

1. 段内直接寻址

这种寻址方式是将当前 IP 寄存器的内容和指令中指定的 8 位或 16 位位移量之和作为转向的有效地址。一般用相对于当前 IP 值的位移量来表示转向有效地址，所以它是一种相对寻址方式。当这一程序段在内存中的不同区域运行时，转移指令本身不会发生变化，这是符合程序的再定位要求的。这种寻址方式适用于条件转移及无条件转移指令，但是当它用于条件转移指令时，位移量只允许 8 位。

指令格式：

```
JMP  NEAR PTR ADDR1
JMP  SHORT ADDR2
```

其中，ADDR1 和 ADDR2 都是转向的符号地址，在机器指令中，用位移量来表示。

2. 段内间接寻址

段内间接寻址的转向有效地址是一个寄存器或一个存储单元的内容，并且可以用数据寻址方式中除立即数以外的任何一种寻址方式取得转向的有效地址。

指令格式：

```
JMP  BX
JMP  WORD PTR [BP + DISP]
```

其中，WORD PTR 为操作符，用以指出其后的寻址方式所取得的转向地址是一个字的有效地址，也就是说它是一种段内转移。

对于段内转移寻址方式，直接把求得的转移的有效地址送到 IP 寄存器就可以了。如果需要计算转移的物理地址，则应是：

$$\text{物理地址} = 10\text{H} \times (\text{CS}) + \text{EA}$$

其中，EA 为转移的有效地址。

下面的两个例子说明了在段内间接寻址方式的转移指令中有效地址的计算方法。

假设：(DS)= 2000H，(BX)= 2000H，(SI)=3000H，(25000H)=3200H。

【例 3.9】 指令如下。

```
JMP  BX
```

则执行指令后

```
(IP) = 2000H
```

【例 3.10】 指令如下。

```
JMP  [BX][SI]
```

则执行指令后

```
(IP) = (10H × (DS) + (BX) + (SI))
     = (20000H + 2000H + 3000H)
     = (25000H)
     = 3200H
```

3. 段间直接寻址

这种寻址方式直接提供转向的段地址(16 位)和偏移地址(16 位)，所以需要 32 位的地址信息。只要用指令中指定的偏移地址取代 IP 寄存器的内容，用指定的段地址取代 CS 寄存器的内容就完成了从一个段到另一个段的转移操作。

指令格式可表示为：

```
JMP  FAR PTR ANOSEG
```

其中，ANOSEG 为转向的符号地址，FAR PTR 则是表示段间转移的操作符。

4. 段间间接寻址

为了达到段间转移，这种寻址方式是用存储器中的两个相继字的内容来取代 IP 和 CS。内存单元的地址是由指令指定的除立即寻址和寄存器寻址方式以外的任何一种数据寻址方式取得。

指令格式可表示为：

```
JMP  DWORD PTR [DISP + BX]
```

其中，[DISP+BX]说明数据寻址方式为寄存器间接寻址方式，DWORD PTR 为双字操作符，以满足段间转移地址的要求。

3.2　8086/8088 指令系统

计算机的指令系统就是指该计算机能够执行的全部指令的集合。8086/8088 指令系统的指令分为 7 类，即数据传送类、算术运算类、逻辑操作类、程序控制类、数据串操作类、处理

器控制类以及输入/输出类指令。在学习指令系统过程中，以下几个方面是需要注意的。

（1）掌握指令的功能：该指令能够实现何种操作，通常指令助记符就是指令功能的英文单词或其缩写形式。

（2）分析指令支持的寻址方式：该指令中的操作数可以采用何种寻址方式。

（3）清楚指令对标志位的影响：该指令执行后是否对各个标志位有影响，以及如何影响。

（4）其他特征：如指令执行时的约定设置、必须预置的参数、隐含使用的寄存器等。

3.2.1 数据传送类指令

数据传送是计算机中最基本、最重要的一种操作。传送指令也是最常使用的一类指令，可以实现数据从一个位置到另一个位置的移动，如执行寄存器与寄存器之间、寄存器与主存单元之间的字或字节的多种传送操作。

1. 数据传送指令

指令格式：

```
MOV  OPRD1,OPRD2
```

OPRD1为目的操作数，可以是寄存器、存储器、累加器。

OPRD2为源操作数，可以是寄存器、存储器、累加器和立即数。

本指令的功能是将一个8位或16位的源操作数（字或字节）送到目的操作数中，即OPRD1←OPRD2，本指令不影响状态标志位。

MOV指令可以分为以下4种情况。

1）寄存器与寄存器之间的数据传送指令

例如：

```
MOV  AX,BX
MOV  CL,AL
MOV  DX,ES
MOV  DS,AX
MOV  BP,SI
```

代码段寄存器CS及指令指针IP不参加数的传送，其中CS可以作为源操作数参加传送，但不能作为目的操作数参加传送。

2）立即数到通用寄存器的数据传送指令

立即数只能作为源操作数使用，不能作目的操作数使用。

例如：

```
MOV  AL,25H
MOV  BX,20A0H
MOV  CH,5
MOV  SP,2F00H
```

注意：由于传送的数据可能是字节，也可能是字，源操作数与目的操作数的类型应一致，8位寄存器不能和16位寄存器之间传送数据。例如，25H送AL是允许的，2000H送AL是不允许的。在立即数参加传送的情况下，数据类型由寄存器确定。当立即数送至存储器时，

不仅其类型应该一致，而且还应当用汇编语言的指示性语句或汇编运算符加以说明。

3）寄存器与存储器之间的数据传送指令

例如：

```
MOV  AL,BUF
MOV  AX,[SI]
MOV  DISP[BX + DI],DL
MOV  SI,ES: [BP]
MOV  DS,DATA[BX + SI]
MOV  DISP[BX + DI],ES
```

第一条指令是将存储器变量 BUF 的值送至 AL 寄存器。BUF 可以看成是存储数据的单元的符号名，实际上是符号地址，因为该地址单元中的数据是可变的，用 BUF 作为变量名是很合适的。如果是对字进行操作，应将指令改写成如下的形式：

```
MOV  AX,BUF
```

这时汇编语言中在定义 BUF 变量时，要用 DW 伪指令将其定义为字类型变量，即在数据段中，用如下形式定义：

```
BUF  DW  1234H
```

这样，就可以使用“MOV　AX,BUF”指令进行数据字的传送操作。该指令将从地址 BUF 开始的两个存储单元的数据送至 AX 中，即(AX)= 1234H。在数据段中，如果用伪指令 DB 将 BUF 变量定义为字节类型，但仍要对从 BUF 开始的单元进行数据字的传送，则应将指令写成如下形式：

```
MOV  AX,WORD PTR BUF
```

其中 PTR 称为指针操作符。其作用是将 BUF 定义为新的变量类型。这个变量的起始地址没有改变，但已将 BUF 变量定义为字类型，因而指令能将一个数据字送至 AX 中。

现在再来看第二条指令“MOV　AX,[SI]”，其中[SI]属于寄存器间接寻址，即由段寄存器 DS 与 SI 一起确定源操作数的物理地址。因为是数据字传送，所以该操作数与求出的物理地址指向的两个连续存储单元有关。

在指令“MOV　DISP[BX+DI],DL”中，目的操作数是存储器操作数，寻址方式是相对基址变址寻址，也可把它写作 DISP[BX][DI]。汇编语言将对它们进行相同的处理，即有效地址 EA ＝ (BX)+(DI)+DISP。

指令“MOV　SI,ES: [BP]”有些特殊，源操作数是存储器操作数，其寻址方式是寄存器间接寻址。但没有按照事先的约定，BP 与 SS 段寄存器结合，找到实际的物理地址。而用“ES: [BP]”告诉汇编程序：BP 将与段寄存器 ES 结合，即 EA ＝(BP)。

与上述指令类似，指令“MOV　DS,DATA[BX+SI]”中的源操作数是存储器操作数，采用相对基址变址方式寻址，有效地址 EA ＝(BX)+(SI)+DATA。

“MOV　DISP[BX+DI],ES”中的目的操作数是存储器操作数，也是相对基址变址寻址，其有效地址为 EA ＝(BX)+(DI)+DISP。

由此可见，只有深刻理解指令中各操作数的寻址方式，才能正确理解指令的真正含义。

同样要注意，寄存器IP不能参加数据传送，CS寄存器不能作为目的操作数参与数据传送。

4）立即数到存储器的数据传送

立即数只能作为源操作数，且一定要使立即数与存储器变量类型一致。存储器变量的类型可以在数据定义时指定，也可以在指令中另行指定，应保证其与立即数的类型一致。否则，汇编程序在汇编时，将指出类型不一致的错误。

例如：

```
MOV  BUF,25
MOV  DS: DISP[BP],1234H
MOV  BYTE PTR [SI],40
```

以上指令如能正确执行，则BUF变量应定义为字节类型，DISP[BP]所指的存储单元应定义为字类型。第三条指令中，用指针操作符PTR重新把[SI]所指的单元定义为字节类型。在编写汇编语言程序时，用BYTR PTR[SI]能告诉汇编程序，此处按字节类型处理，生成相应的指令代码。

使用MOV指令传送数据，必须注意以下几点。

（1）立即数只能作为源操作数，不允许作目的操作数，立即数也不能送至段寄存器。

（2）通用寄存器可以与段寄存器、存储器互相传送数据，寄存器之间也可以互相传送数据。但是CS不能作为目的操作数。

（3）存储器与存储器之间不能进行数据直接传送。若要实现存储单元之间的数据传送，可以借助于通用寄存器作为中介来进行。

【例3.11】 要把DATA1单元的内容送至DATA2单元中，并假定这两个单元在同一个数据段中，可以通过以下两条指令实现：

```
MOV  AL,DATA1
MOV  DATA2,AL
```

2. 数据交换指令

数据传送指令单方向地将源操作数送至目的操作数存储单元，而数据交换指令则将两个操作数相互交换位置，例如：

```
XCHG  OPRD1,OPRD2
```

指令中的OPRD1为目的操作数，OPRD2为源操作数，该指令把源操作数OPRD2与目的操作数OPRD1交换。OPRD1及OPRD2可为通用寄存器或存储器。XCHG指令不支持两个存储器单元之间的数据交换，但通过中间寄存器，可以很容易地实现两个存储器操作数的交换。交换操作的示意图如图3.9所示。

图3.9 XCHG指令操作示意图

本指令不影响状态标志位，段寄存器内容不能用XCHG指令来交换。以下指令是合法的。

```
XCHG  AX,BX
XCHG  SI,AX
XCHG  DL,DH
XCHG  DX,BUF
```

```
XCHG  WBUF,CX
```

3. 换码指令

指令格式：

```
XLAT  TABLE
XLAT
```

以上两种格式是完全等效的。本指令的功能是把待查表格的一个字节内容送到 AL 累加器中。其中 TABLE 为一待查表格的首地址，在执行该指令前，应将 TABLE 先送至 BX 寄存器中，然后将待查字节与在表格中距表首地址位移量送 AL，即：

```
(AL)←((BX) + (AL))
```

换码指令常用于将一种代码转换为另一种代码，如扫描码转换为 ASCII 码，数字 0～9 转换为七段显示码等。使用前首先在主存中建立一个字节表格，表格的内容是要转换成的目标代码。由于 AL 的内容实际上是距离表格首地址的位移量，只有 8 位，所以表格的最大长度为 256。超过 256 的表格需要采用修改 BX 和 AL 的方法才能转换。

本指令不影响状态标志位。

XLAT 指令中没有显式指明操作数，而是默认使用 BX 和 AL 寄存器。这种采用默认操作数的方法称为隐含寻址方式，指令系统中有许多指令采用隐含寻址方式。

【例 3.12】 将首地址为 200H 的表格缓冲区中的 4 号数据取出。

```
MOV  BX,200H
MOV  AL,4
XLAT
```

4. 堆栈操作指令

堆栈被定义为一种先进后出的数据结构，即最后进栈的元素将被最先弹出来。堆栈从一个称为栈底的位置开始，数据进入堆栈的操作称为入栈(或压栈)，数据退出堆栈的操作称为出栈，每进行一次出栈操作，堆栈就减少一个元素，最后一次入栈的元素，称为栈顶元素，入栈操作和出栈操作都是对栈顶元素进行的基本操作。在计算机中，堆栈设置在一个存储区域中。8086/8088 的堆栈段的开始位置由(SS)×10H＋0000H 决定，堆栈段最大为 64KB。SP 称为堆栈指针，确切地讲是指向栈顶元素的地址指针，由 SP 的值就可以知道栈顶元素的位置。

1) 入栈操作指令

实现入栈操作的指令是 PUSH 指令，其格式为：

```
PUSH  OPRD
```

其中，OPRD 为 16 位(字)操作数，可以是寄存器或存储器操作数。PUSH 的操作过程是：

```
(SP)←(SP) - 2,((SP))←OPRD
```

即先修改堆栈指针 SP(入栈时为自动减 2)，然后，将指定的操作数送入新的栈顶位置。此处的((SP))←OPRD，也可以理解为：

```
[(SS)×10H + (SP)]←OPRD
```

或

```
[SS: SP]←OPRD
```

例如：

```
PUSH  AX
PUSH  BX
```

每条指令的操作过程分两步，首先将 SP－1→SP，把 AH 的内容送至由 SP 所指的单元。接下来再使 SP－1→SP，把 AL 的内容送到 SP 所指的单元。随着入栈内容的增加，堆栈就扩展，SP 值减少，但每次操作完，SP 总是指向堆栈的顶部。图 3.10(a)给出了入栈操作的情况。

以下入栈操作指令都是有效的。

```
PUSH  DX
PUSH  SI
PUSH  BP
PUSH  CS
PUSH  BUFFER
PUSH  DAT[BX][SI]
```

注意：每进行一次入栈操作，都压入一个字(16 位)，例中的 BUFFER，DAT[BX][SI]所指的存储器操作数应该被指定为字类型。

2) 出栈操作指令

实现出栈操作的指令是 POP 指令，其格式为：

```
POP  OPRD
```

其中，OPRD 为 16 位(字)操作数，可以是寄存器或存储器操作数。POP 指令的操作过程是：

```
OPRD←((SP)),(SP)←(SP) + 2
```

它与入栈操作相反，是先弹出栈顶的数据，然后再修改指针 SP 的内容(自动加 2)。

例如：

```
POP  BX
POP  AX
```

图 3.10(b)给出了出栈操作的情况。

以下出栈操作指令都是有效的。

```
POP  AX
POP  DS
POP  BUFFER
POP  DAT[BX][DI]
```

其中的 BUFFER 及 DAT[BX][DI]所指的存储器操作数也应被指定为字类型。

PUSH 及 POP 指令对标志位没有影响。

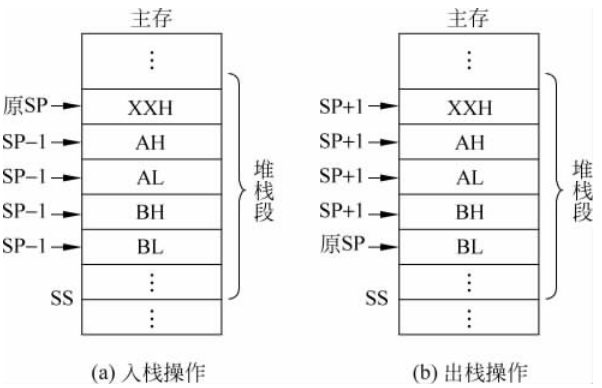

图 3.10　入栈和出栈操作

【例 3.13】 用堆栈操作指令实现两个存储器操作数 BUF1 及 BUF2 的交换。

```
PUSH   BUF1
PUSH   BUF2
POP    BUF1
POP    BUF2
```

5. 标志传送指令

1）取 FLAGS 标志寄存器低 8 位至 AH 指令

指令格式：

```
LAHF
```

指令含义为：

$(AH) \leftarrow (FLAGS)_{7\sim0}$

该指令不影响 FLAGS 的原来内容，AH 只是复制了原 FLAGS 的低 8 位内容。

2）将 AH 存至 FLAGS 低 8 位指令

指令格式：

```
SAHF
```

指令含义为：

$(FLAGS)_{7\sim0} \leftarrow (AH)$

本指令将用 AH 的内容改写 FLAGS 中的 SF、ZF、AF、PF 和 CF 标志，从而改变原来的标志位。

3）将 FLAGS 内容入栈指令

指令格式：

```
PUSHF
```

本指令可以把 FLAGS 的内容保存到堆栈中去。

4）从堆栈中弹出一个数据字送至 FLAGS 中的指令

指令格式：

```
POPF
```

本指令的功能与 PUSHF 相反，在子程序调用和中断服务程序中，往往用 PUSHF 指令保护 FLAGS 的内容，用 POPF 指令将保护的 FLAGS 内容恢复。

6. 地址传送指令

地址传送指令将存储器的逻辑地址送至指定的寄存器。

1）有效地址传送指令

指令格式：

```
LEA   OPRD1,OPRD2
```

指令中的 OPRD1 为目的操作数，可为任意一个 16 位的通用寄存器。OPRD2 为源操作数，可为变量名、标号或地址表达式。

本指令的功能是将源操作数给出的有效地址传送到指定的寄存器中。本指令对标志位无影响。

例如：

```
LEA   BX,DATA1
LEA   DX,BETA[BX + SI]
LEA   BX,[BP][DI]
```

显然，LEA BX，DATA1 的功能是将变量 DATA1 的地址送至 BX，而不是将变量 DATA1 的值送 BX。

如果要将一个存储器变量的地址取至某一个寄存器中，也可以用 OFFSET DATA1 表达式实现。

例如 MOV SI，OFFSET DATA1 指令，其功能是将存储器变量 DATA1 的段内偏移地址取至 SI 寄存器中，OFFSET DATA1 表达式的值就是取 DATA1 的段内偏移地址值。

因此，LEA BX，DATA1 指令与 MOV BX，OFFSET DATA1 指令的功能是等价的。

2）从存储器取出 32 位地址的指令

指令格式：

```
LDS   OPRD1,OPRD2
LES   OPRD1,OPRD2
```

其中，OPRD1 为任意一个 16 位的寄存器，OPRD2 指定了主存的连续 4 字节作为逻辑地址，即 32 位的地址指针。

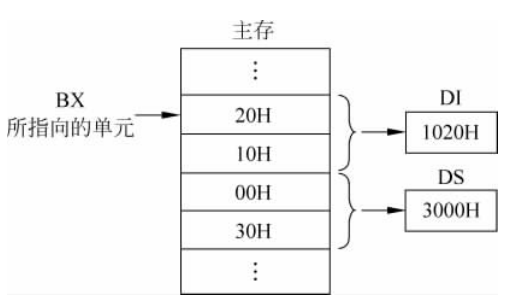

图 3.11　LDS 指令操作示意图

例如：

```
LDS   DI,[BX]
```

该指令的功能是把 BX 所指的 32 位地址指针的段地址送入 DS，偏移地址送入 DI。指令操作示意图如图 3.11 所示。

“LES DI,[BX]”指令除将地址指针的段地址送入 ES 外，其他与 LDS 类似。因此图 3.11 不仅适用于 LDS 指令，也适用于 LES

指令。

以下指令是合法的。

```
LDS  SI,ABCD
LDS  BX,FAST[SI]
LES  SI,ABCD
LES  BX,FAST[SI]
```

注意：以上指令不影响标志位。

3.2.2　算术运算类指令

算术运算类指令用来执行二进制及十进制的算术运算，主要包括加、减、乘、除指令。二进制数运算分为带符号数运算和不带符号数运算。带符号数的最高位是符号位，不带符号数所有位都是有效位。十进制数用 BCD 码表示，又分为非压缩的 BCD 码和压缩的 BCD 码两种形式。

算术运算指令会根据运算结果影响状态标志，有时要利用某些标志才能得到正确的结果。该类指令主要影响 6 个标志位，即 CF、AF、SF、ZF、PF 和 OF。

1. 加法指令

加法指令能实现字或字节的加法运算。

1）基本加法指令

指令格式：

```
ADD  OPRD1,OPRD2
```

指令含义为：

```
OPRD1 ← OPRD1 + OPRD2
```

OPRD1 为目的操作数，可以是任意一个通用寄存器，也可以是任意一个存储器操作数。在 ADD 指令执行前，OPRD1 的内容为一个加数，待 ADD 指令执行后，OPRD1 中为加法运算的结果，即和。这给程序的编写带来了很大的方便。OPRD2 为源操作数，它可以是立即数，也可以是任意一个通用寄存器或存储器操作数。立即数只能用于源操作数。

OPRD1 和 OPRD2 均为寄存器是允许的，一个为寄存器而另一个为存储器也是允许的，但不允许两个都是存储器操作数。

加法指令运算的结果对 CF、SF、OF、PF、ZF、AF 都会有影响。以上标志也称为结果标志。加法指令适用于无符号数或有符号数的加法运算。操作数可以是 8 位，也可以是 16 位。

例如：

```
ADD  AL,38H
ADD  BX,0A0AH
ADD  DX,DATA[BX]
ADD  DI,CX
ADD  BETA[BX],AX
ADD  BYTE PTR[BX],55
```

上述第一条指令及第六条指令为字节相加指令，其他四条均为字（双字节）相加指令。第三条指令中，存储器操作数是源操作数，采用寄存器相对寻址方式；第五条指令中，存储器操作数是目的操作数，采用寄存器相对寻址方式；第六条指令中，存储器操作数也是目的操作数，采用寄存器间接寻址方式，当立即数与存储器操作数做加法时，类型必须一致，故此处用 BYTE PTR[BX]，将存储器操作数的类型重新指定为字节类型，以保证两个操作数类型一致。

2）带进位加法指令

指令格式：

```
ADC  OPRD1,OPRD2
```

指令含义为：

```
OPRD1 ← OPRD1 + OPRD2 + CF
```

其中，OPRD1、OPRD2 与指令 ADD 中的含义一样。该指令对标志位的影响同 ADD 指令。

例如：

```
ADC  AL,CL
ADC  AX,SI
ADC  DX,MEMA
ADC  CL,15
ADC  WORD PTR[BX][SI],25
```

本指令适用无符号数及带符号数的 8 位或 16 位运算。

【例 3.14】 如果两个 32 位的数据已分别放在 ADNUM1 和 ADNUM2 开始的存储区中，存放时低字在前，高字在后，将两个数相加，且将和存放在 ADNUM3 开始的存储区中。

实现两个 32 位数相加的程序为：

```
MOV  AX,ADNUM1
ADD  AX,ADNUM2
MOV  ADNUM3,AX
MOV  AX,ADNUM1 + 2
ADC  AX,ADNUM2 + 2
MOV  ADNUM3 + 2,AX
```

由于两个存储器操作数不能直接相加，因而不能通过“ADD　ADNUM1，ADNUM2”指令进行。

3）加 1 指令

指令格式：

```
INC  OPRD
```

指令含义为：

```
OPRD ← OPRD + 1
```

OPRD 为寄存器或存储器操作数。这条指令的功能是对给定的除段寄存器以外的任何寄存器或存储单元内容加 1 后，再送回该操作数，可以实现字节加 1 或字加 1。本指令执行

结果影响 AF、OF、PF、SF、ZF 标志位，但不影响 CF 标志位。

例如：

```
INC  SI
INC  WORD PTR[BX]
INC  BYTE PTR[BX + DI]
INC  CL
```

上述第二条、第三条这两条指令，是对存储字及存储字节的内容加 1 以替代原来的内容。在循环程序中，常用该指令对地址指针和循环计数值进行修改。

2. 减法指令

1）基本减法指令

指令格式：

```
SUB  OPRD1,OPRD2
```

本指令的功能是进行两个操作数的相减，即：

```
OPRD1 ← OPRD1 - OPRD2
```

本指令的类型及对标志位的影响与 ADD 指令相同，注意立即数不能用于目的操作数，两个存储器操作数之间不能直接相减。操作数可为 8 位或 16 位的无符号数或带符号数。

例如：

```
SUB  DX,CX
SUB  [BX + 25],AL
SUB  DI,ALFA[SI]
SUB  CL,20
```

2）带借位减法指令

指令格式：

```
SBB  OPRD1,OPRD2
```

其中，OPRD1、OPRD2 的含义及指令对标志位的影响等均与 SUB 指令相同。完成的操作为：OPRD1←OPRD1－OPRD2－CF。

例如：

```
SBB  DX,CX
SBB  BX,2000H
SBB  AX,DATA1
SBB  ALFA[BX + SI],SI
```

3）减 1 指令

指令格式：

```
DEC  OPRD
```

其中，OPRD 的含义与 INC 指令相同，本指令的功能是 OPRD ←OPRD －1。对标志位影响同 INC 指令。

例如：

```
DEC  AX
DEC  CL
DEC  WORD PTR[DI]
DEC  ALFA[DI+BX]
```

4）取补指令

指令格式：

```
NEG  OPRD
```

OPRD为任意通用寄存器或存储器操作数。NEG指令也是一个单操作数指令，它对操作数执行求补运算，即用0减去操作数，然后将结果返回操作数。求补运算也可以表达成：将操作数按位取反后加1。NEG指令对标志位的影响与用0做减法的SUB指令一样。

例如：(AL)＝44H，取补后，(AL)＝0BCH(－44H)。

5）比较指令

指令格式：

```
CMP  OPRD1,OPRD2
```

其中，OPRD1和OPRD2可为任意通用寄存器或存储器操作数，但两者不同时为存储器操作数，立即数可用做源操作数OPRD2。本指令对标志位的影响同SUB指令，完成的操作与SUB指令类似，唯一的区别是不将OPRD1－OPRD2的结果送回OPRD1，而只是比较。因而不改变OPRD1和OPRD2的内容，该指令用于改变标志位。

例如：

```
CMP  AL,56H
CMP  DX,CX
CMP  AX,DATA1[BX]
CMP  BATE[DI],BX
```

以CMP　DX,CX为例，对标志位的影响如下：

(1) (DX)＝(CX)时，则ZF＝1。

(2) 两个无符号数比较：若(DX)＞(CX)，则CF＝0，即无借位；若(DX)＜(CX)，则CF＝1，即有借位。

(3) 两个带符号数比较：可以通过溢出标志OF及符号标志SF共同来判断两个数的大小。

当OF＝0，即无溢出时，若SF＝0，则(DX)＞(CX)；若SF＝1，则(DX)＜(CX)。

当OF＝1，即有溢出时，若SF＝1，则(DX)＞(CX)；若SF＝0，则(DX)＜(CX)。

3. 乘法指令

1）无符号数乘法指令

指令格式：

```
MUL  OPRD
```

其中，OPRD 为源操作数，即乘数。OPRD 为通用寄存器或存储器操作数。目的操作数是隐含的，即被乘数总是指定为累加器 AX 或 AL 的内容。16 位乘法时，AX 中为被乘数；8 位乘法时，AL 为被乘数。16 位乘法的 32 位乘积存于 DX 及 AX 中；8 位乘法的 16 位乘积存于 AX 中。

操作过程如下。

(1) 字节相乘：(AX)←(AL)×OPRD，当结果的高位字节(AH)≠0 时，则 CF＝1，OF＝1。

(2) 字相乘：(DX)(AX)←(AX)×OPRD，当(DX)≠0 时，则 CF＝1，OF＝1。

例如：

```
MUL   BETA[BX]
MUL   DI
MUL   BYTE PTR ALFA
```

【例 3.15】 设在 DAT1 和 DAT2 字单元中各有一个 16 位数，若求其乘积并存于 DAT3 开始的字单元中，可用以下指令组实现：

```
MOV   AX,DAT1
MUL   DAT2
MOV   DAT3,AX
MOV   DAT3 + 2,DX
```

2) 带符号数乘法指令

指令格式：

```
IMUL   OPRD
```

其中，OPRD 为任一通用寄存器或存储器操作数。隐含操作数的定义与 MUL 指令相同。本指令的功能是完成两个带符号数的相乘。

MUL 和 IMUL 指令影响标志位 CF 及 OF。

计算二进制数乘法：B4H×11H。如果把它当作无符号数，用 MUL 指令，则乘的结果为 0BF4H；如果看作有符号数，用 IMUL 指令，则乘的结果为 FAF4H。由此可见，同样的二进制数看作无符号数与有符号数相乘，即采用 MUL 与 IMUL 指令，其运算结果是不相同的。

4. 除法指令

1) 无符号数除法指令

指令格式：

```
DIV   OPRD
```

其中，OPRD 为任一通用寄存器或存储器操作数。

本指令的功能是实现两个无符号二进制除法运算。字节相除，被除数在 AX 中；字相除，被除数在 DX、AX 中，除数在 OPRD 中。

操作过程如下。

(1) 字节除法：(AL)←(AX)/OPRD，(AH)←(AX) MOD OPRD。

（2）字除法：(AX)←(DX)(AX)/OPRD,(DX)←(DX)(AX) MOD OPRD。

例如：

```
DIV  BETA[BX]
DIV  CX
DIV  BL
```

2）带符号数除法指令

指令格式：

```
IDIV  OPRD
```

其中,OPRD为任一通用寄存器或存储器操作数。隐含操作数的定义与DIV指令相同。

本指令的功能是实现两个带符号数的二进制除法运算,余数的符号与被除数符号相同。

除法指令DIV和IDIV不产生有效的标志位,但是却可能产生溢出。当被除数远大于除数时,所得的商就有可能超出它所能表达的范围。对DIV指令,除数为0,或者在字节除法时商大于255,字除法时商大于65 535,则发生除法溢出。对IDIV指令,除数为0,或者在字节除法时商不在－128～127范围内,或者在字除法时商不在－32 768～32 767范围内,则发生除法溢出。

3）字节扩展指令

符号扩展是指用一个操作数的符号位(即最高位)形成另一个操作数,后一个操作数的各位是全0(正数)或全1(负数)。符号扩展指令可用来将字节转换为字,字转换为双字。有符号数通过符号扩展加长了位数,但数据大小并没有改变。符号扩展指令不影响标志位。

指令格式：

```
CBW
```

本指令的功能是将字节扩展为字,即把AL寄存器的符号位扩展到AH中。即两个字节相除时,先使用本指令形成一个双字节长的被除数。

例如：

```
MOV  AL,34
CBW
IDIV  BYTE PTR DATA1
```

4）字扩展指令

指令格式：

```
CWD
```

本指令的功能是将字扩展为双字长,即把AX寄存器的符号位扩展到DX中。即两个字相除时,先使用本指令形成一个双字长的被除数。

符号扩展指令常用来获得除法指令所需要的被除数。例如AX＝FF00H,它表示有符号数－256；执行CWD指令后,则DX＝FFFFH,DX、AX仍表示有符号数－256。

对无符号数除法应该采用直接使用高8位或高16位清0的方法,获得倍长的被除数。这就是0位扩展。

【例 3.16】 在 DAT1、DAT2、DAT3 字节类型变量中，分别存有 8 位带符号数 a、b、c，用程序实现 (a * b+c)/a 运算。

程序如下：

```
MOV  AL,DAT1
IMUL  DAT2
MOV  CX,AX
MOV  AL,DAT3
CBW
ADD  AX,CX
IDIV  DAT1
```

5. 十进制调整指令

为了方便进行十进制的运算，8086/8088 提供了一组十进制数调整指令。这组指令对二进制运算的结果进行十进制调整，以得到十进制的运算结果。十进制数在计算机中也要用二进制编码表示，这就是二进制编码的十进制数：BCD 码。8086/8088 支持压缩 BCD 码和非压缩 BCD 码，相应地十进制调整指令分为压缩 BCD 码的调整指令和非压缩 BCD 码的调整指令。

压缩 BCD 码是通常的 8421 码，它用 4 个二进制位表示一个十进制位，一个字节可以表示两个十进制位，即 00～99。压缩 BCD 码调整指令包括加法和减法的十进制调整指令 DAA 和 DAS，它们用来对二进制加、减法指令的执行结果进行调整，得到十进制结果。

非压缩 BCD 码用 8 个二进制位表示一个十进制位，实际上只是用低 4 个二进制位表示一个十进制位 0～9，高 4 位通常默认为 0。ASCII 码中 0～9 的编码是 30H～39H，所以 0～9 的 ASCII 码(高 4 位变为 0)就可以认为是非压缩的 BCD 码。非压缩 BCD 码调整指令包括 AAA、AAS、AAM 和 AAD 四条指令，分别用于对二进制加、减、乘、除指令的结果进行调整，以得到非压缩 BCD 码表示的十进制数结果。由于只要在调整后的结果中加上 30H 就成为十进制数位的 ASCII 码，因此这组指令实际上也是针对 ASCII 码的调整指令。

在进行十进制数算术运算时，应分两步进行：先按二进制数运算规则进行运算，得到中间结果；再用十进制调整指令对中间结果进行修正，得到正确的结果。

下面通过几个例子说明 BCD 码运算为什么要调整以及怎样调整。

【例 3.17】 24+33=57。

24 的 BCD 码为 00100100B，33 的 BCD 码为 00110011B，57 的 BCD 码为 01010111B。

24+33 的二进制算式如下：

$$\begin{array}{rr} & 00100100 \\ + & 00110011 \\ \hline & 01010111 \end{array}$$

结论：结果正确，不需要调整。

【例 3.18】 27+54=81。

27 的 BCD 码为 00100111B，54 的 BCD 码为 01010100B，81 的 BCD 码为 10000001B。

27+54 的二进制算式如下：

```
  00100111
+ 01010100
----------
  01111011
```

结果不正确，因为在进行二进制加法运算时，低4位向高4位有一个进位，这个进位是按十六进制进行的，即低4位逢十六才进一，而十进制数应是逢十进一。因此，比正确结果少6，这时应在低4位上进行加6处理，调整的算式如下：

```
  00100111
+ 01010100
----------
  01111011
+ 00000110
----------
  10000001
```

调整后的结果正确。

结论：加法运算后，低4位若向高4位有进位（即AF＝1）时，应对低4位做加06H处理。

【例3.19】 97＋81＝178。

97的BCD码为10010111B，81的BCD码为10000001B，178的BCD码为000101111000B。

97＋81的二进制算式如下：

```
    10010111
+   10000001
------------
CF←100011000
```

结果不正确，因为高4位向CF的进位是按十六进制进行的，应加60H进行调整。调整的算式如下：

```
    10010111
+   10000001
------------
CF←100011000
+   01100000
------------
    01111000
```

调整后的结果正确。

结论：加法运算后，当CF＝1（有进位产生）时，应做加60H处理。

【例3.20】 64＋48＝112。

64的BCD码为01100100B，48的BCD码为01001000B，112的BCD码为000100010010B。

64＋48的二进制算式如下：

```
  01100100
+ 01001000
----------
  10101100
```

结果不正确，因为低4位大于9，且高4位也大于9。先加06H调整低4位，再加60H调整高4位。调整的算式如下：

```
        01100100
+       01001000
--------------------
        10101100
+       00000110
--------------------
        10110010
+       01100000
--------------------
   CF ← 100010010
```

调整后的结果正确。

结论：加法运算后，低 4 位大于 9 时，需做加 06H 处理；高 4 位大于 9 时，需做加 60H 处理。

下面介绍十进制数的调整指令。

1）DAA 指令

指令格式：

```
DAA
```

DAA 指令为无操作数指令。用以完成对压缩的 BCD 码加法运算进行校正。一般在 ADD 指令之后，紧接着用一条 DAA 指令加以校正，在 AL 中可以得到正确的结果。

DAA 指令的校正操作如下。

（1）若(AL∧0FH)＞9 或标志位 AF＝1，则

```
AL←(AL) + 6
AF←1
```

（2）若 AL＞9FH 或标志 CF＝1，则

```
CF←AF
AL←AL∧0FH
AL←(AL) + 60H
CF←1
```

DAA 指令影响标志位 AF、CF、PF、SF、ZF，而对 OF 未定义。

2）DAS 指令

指令格式：

```
DAS
```

DAS 指令为无操作数指令。用以完成对压缩的 BCD 码相减的结果进行校正，得到正确的压缩的十进制差。一般在 SUB 指令之后，紧接着用一条 DAS 指令加以校正，在 AL 中可以得到正确的结果。

DAS 指令校正的操作如下。

（1）若(AL∧0FH)＞9 或标志位 AF＝1，则

```
AL←(AL) - 6
AF←1
```

（2）若 AL＞9FH 或标志 CF＝1，则

```
AL←(AL) - 60H
CF←1
```

DAS 指令执行时，影响标志位 AF、CF、PF、SF、ZF，而对 OF 未定义。

3）AAA 指令

指令格式：

```
AAA
```

AAA 指令为无操作数指令。AAA 指令对在 AL 中的两个非压缩的十进制数相加后的结果进行校正。两个非压缩的十进制数可以直接用 ADD 指令相加，但要得到正确的非压缩的十进制结果，必须在 ADD 指令之后，用一条 AAA 指令加以校正，在 AX 中可以得到正确的结果。

AAA 指令进行校正的操作如下。

若(AL∧0FH)＞9 或标志位 AF＝1，则

```
AL←(AL) + 6
AH←(AH) + 1
AF←1
CF←AF
AL←AL∧0FH
```

AAA 指令对标志位 AF 和 CF 有影响，而对 OF、PF、SF、ZF 未定义。

4）AAS 指令

指令格式：

```
AAS
```

AAS 指令为无操作数指令。AAS 指令把 AL 中两个非压缩的十进制数相减后的结果进行校正，产生一个正确的非压缩的十进制数差。

AAS 指令进行校正的操作如下。

若(AL∧0FH)＞9 或标志位 AF＝1，则

```
AL←(AL) - 6
AH←(AH) - 1
AF←1
CF←AF
AL←AL∧0FH
```

AAS 指令影响标志位 AF 和 CF，而对 OF、PF、SF、ZF 位未定义。

5）AAM 指令

指令格式：

```
AAM
```

AAM 指令执行的操作为：把 AL 中的积调整到非压缩的 BCD 格式后送给 AX 寄存器。

这条指令之前必须执行 MUL 指令把两个非压缩的 BCD 码相乘(此时要求其高 4 位为 0),结果放在 AL 寄存器中。

本指令的调整方法是:把 AL 寄存器的内容除以 0AH,商放在 AH 寄存器中,余数保存在 AL 寄存器中。本指令根据 AL 寄存器的内容设置标志位 SF、ZF 和 PF,但对 OF、CF 和 AF 标志位无影响。

6) AAD 指令

指令格式:

```
AAD
```

前面所述的对非压缩 BCD 码的调整指令都是在完成相应的加法、减法及乘法运算后,再使用 AAA、AAS 及 AAM 指令来对运算结果进行十进制调整的。除法的情况却不同,它针对的情况如下所述。

如果被除数是存放在 AX 寄存器中的两位非压缩 BCD 数,AH 中存放十位数,AL 中存放个位数,而且要求 AH 和 AL 中的高 4 位均为 0。除数是一位非压缩的 BCD 数,同样要求高 4 位为 0。在把这两个数用 DIV 指令相除以前,必须先用 AAD 指令把 AX 中的被除数调整成二进制数,并存放在 AL 寄存器中。因此,AAD 指令执行的操作是:

```
10 × AH + AL→AL
0→AH
```

这条指令根据 AL 寄存器的内容设置标志位 SF、ZF 和 PF,但对 OF、CF 和 AF 标志位无影响。

3.2.3　逻辑操作类指令

逻辑操作类指令是按位操作指令,可以对 8 位或 16 位的寄存器或存储单元的内容按位操作。该类指令包括逻辑运算指令、移位指令和循环移位指令。

1. 逻辑运算指令

逻辑运算指令用来对字或字节按位进行逻辑运算,包括逻辑与 AND、逻辑或 OR、逻辑非 NOT、逻辑异或 XOR 和测试 TEST 五条指令。

1) 逻辑与运算指令

指令格式:

```
AND   OPRD1,OPRD2
```

其中,目的操作数 OPRD1 为任一通用寄存器或存储器操作数,源操作数 OPRD2 为立即数、任一通用寄存器或存储器操作数。也就是在这两个操作数中,源操作数可以是任意的寻址方式,而目的操作数只能是立即数之外的其他寻址方式,并且两个操作数不能同时为存储器寻址方式。

AND 指令实现对两个操作数按位进行逻辑与的运算,结果送至目的操作数。本指令可以进行字节或字的“与”运算。

AND 指令影响标志位 PF、SF、ZF,使 CF=0,OF=0。例如,在同一个通用寄存器自身相与时,操作数虽不变,但使 CF 置 0。本指令主要用于修改操作数或置某些位为 0。

例如：

```
AND  AL,0FH
AND  AX,BX
AND  DX,BUF1[DI+BX]
AND  BYTE[BX],00FFH
```

上例中的第一条指令，将使AL寄存器的高4位置成零保持AL低4位值不变。

2）逻辑或运算指令

指令格式：

```
OR  OPRD1,OPRD2
```

其中，OPRD1、OPRD2的含义与AND指令相同，对标志位的影响也与AND指令相同。唯一不同的地方是，OR指令完成对两个操作数按位“或”的运算，结果送至目的操作数中。本指令可以进行字节或字的“或”运算。

OR指令可用于置位某些位，而不影响其他位。这时只需将要置1的位同“1”相或，维持不变的位同“0”相或即可。

3）逻辑非运算指令

指令格式：

```
NOT  OPRD
```

其中，OPRD可为任一通用寄存器或存储器操作数。

本指令的功能是完成对操作数的按位求反运算，结果送回给原操作数，本指令可以进行字节或字的“非”运算，不影响标志位。

4）逻辑异或运算指令

指令格式：

```
XOR  OPRD1,OPRD2
```

其中，OPRD1、OPRD2的含义与AND指令相同，对标志位的影响也与AND指令相同。本指令的功能是实现两个操作数按位“异或”的运算，即相“异或”的两位不相同时，结果是1；否则，“异或”的结果为0，结果送至目的操作数中。XOR指令可以实现字节或字的“异或”运算。

XOR可以用于求反某些位，而不影响其他位。要求求反的位同“1”异或，维持不变的位同“0”异或。

例如，XOR BL,00010001B指令的功能是将BL中D_0和D_4求反，其余位不变；XOR AX,AX可以实现对AX寄存器内容清0。

5）测试指令

指令格式：

```
TEST  OPRD1,OPRD2
```

其中，OPRD1、OPRD2的含义同AND指令，对标志位的影响也与AND指令相同。该指令与AND指令一样，也是对两个操作数进行按位的“与”运算，唯一不同之处是不将相“与”的

结果送目的操作数，即本指令对两个操作数的内容均不进行修改，仅是在逻辑“与”操作后，对标志位重新置位。

TEST 指令通常用于检测一些条件是否满足，但又不希望改变原操作数的情况。这条指令之后，一般都是条件转移指令，目的是利用测试条件转向不同的程序段。

【例 3.21】 写出判断寄存器 AX 中 D_3 和 D_9 位是否为 0 的指令。

```
TEST  AX,0008H
TEST  AX,0200H
```

2. 逻辑移位指令

8086/8088 指令系统的移位指令包括逻辑左移 SHL、算术左移 SAL、逻辑右移 SHR、算术右移 SAR 等指令，其中 SHL 和 SAL 指令的操作完全相同。移位指令的操作对象可以是一个 8 位或 16 位的寄存器或存储单元。移位操作可以是向左或向右移一位，也可以移多位。当要求移多位时，指令规定移位位数(次数)必须放在 CL 寄存器中，即指令中规定的移位次数不允许是 1 以外的常数或 CL 以外的寄存器。移位指令都影响状态标志位，但影响的方式各条指令不尽相同。

1) 逻辑左移指令

指令格式：

```
SHL  OPRD1,COUNT
```

其中，OPRD1 为目的操作数，可以是通用寄存器或存储器操作数。COUNT 代表移位的次数(或位数)。移位一次，COUNT＝1，移位多于一次时，COUNT＝(CL)，(CL)中为移位的次数。

本指令的功能是对给定的目的操作数(8 位或 16 位)左移 COUNT 次，每次移位时最高位移入标志位 CF 中，最低位补 0。本指令对标志位 OF、PF、SF、ZF、CF 有影响。

2) 逻辑右移指令

指令格式：

```
SHR  OPRD1,COUNT
```

其中，OPRD1、COUNT 与指令 SHL 中意义相同。与 SHL 一样，SHR 指令也影响标志位 OF、PF、SF、ZF 和 CF。所不同的是，本指令实现由 COUNT 决定次数的逻辑右移操作，每次移位时，最高位补零，最低位移至标志位 CF 中。

例如：

```
SHL  BL,1
SHL  CX,1
SHL  ALFA[DI],1
```

或者：

```
MOV  CL,3
SHR  DX,CL
SHR  DAT[DI],CL
```

上例中前三条指令完成目的操作数逻辑左移 1 位的运算；而后两条移位指令，则实现

由 CL 内容指定的次数的右移运算，由于(CL)＝3，故分别对目的操作数逻辑右移 3 位。

3）算术左移指令

指令格式：

```
SAL  OPRD1,COUNT
```

其中，OPRD1、COUNT 与指令 SHL 中意义相同。本指令与 SHL 的功能也完全相同，这是因为逻辑左移指令与算术左移指令所要完成的操作是一样的。如果 SAL 将 OPRD1 的最高位移至 CF，改变了原来的 CF 值，则溢出标志位 OF＝1，表示移位前后的操作数不再具有倍增的关系。因而 SAL 可用于带符号数的倍增运算，SHL 只能用于无符号数的倍增运算。

4）算术右移指令

指令格式：

```
SAR  OPRD1,COUNT
```

其中，OPRD1、COUNT 与指令 SHL 中意义相同。本指令通常用于对带符号数减半的运算中，因而在每次右移时，保持最高位(符号位)不变，最低位右移至 CF 中。

图 3.12 给出了上述四条移位指令的操作示意图。

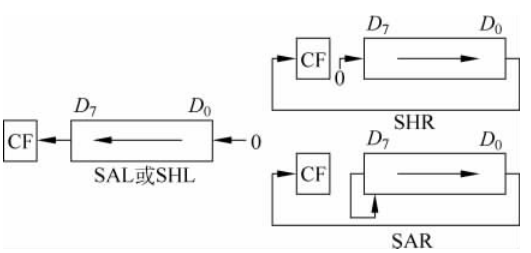

图 3.12　移位指令操作示意图

3. 循环移位指令

能实现操作数首尾相连的移位操作是循环移位指令。循环移位指令类似于移位指令，但要从一端移出的位返回到另一端形成循环。按进位标志 CF 是否参加循环移位，又可分为不带 CF 的循环移位指令和带 CF 的循环移位指令两类，每一类都可进行左移或右移，循环移位的次数由 COUNT 操作数给出。

1）不带进位循环左移指令

指令格式：

```
ROL  OPRD1,COUNT
```

2）不带进位循环右移指令

指令格式：

```
ROR  OPRD1,COUNT
```

3）带进位循环左移指令

指令格式：

```
RCL   OPRD1,COUNT
```

4）带进位循环右移指令

指令格式：

```
RCR   OPRD1,COUNT
```

循环移位指令的操作数形式与移位指令相同，如果仅移动一次，可以用 1 表示；如果需要移动多次，则需用 CL 寄存器表示移位次数。

这组指令只对标志位 CF 和 OF 有影响。CF 由移入 CF 的内容决定，OF 取决于移位一次后符号位是否改变，如改变，则 OF＝1。由于是循环移位，因此对字节移位 8 次，对字移位 16 次，就可恢复为原操作数。由于带 CF 的循环移位，可以将 CF 的内容移入，因此可以利用它实现多字节的循环。循环移位指令的操作示意图如图 3.13 所示。

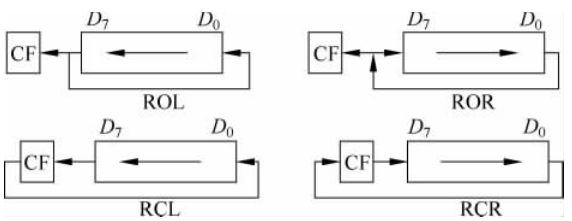

图 3.13　循环移位指令操作示意图

【例 3.22】 有两位 BCD 数存放在 BUFFER 单元，要求将其转换为 ASCII 码，存于 RESULT 开始的两个地址单元，并测试是否有字节为'0'的 ASCII 码，如有，则 CF＝1，结束操作。

程序如下：

```
      MOV   AL,BUFFER
      AND   AL,0F0H
      MOV   CL,4
      SHR   AL,CL
      OR    AL,30H
      CMP   AL,30H
      JZ    ZERO
      MOV   RESULT,AL
      MOV   AL,BUFFER
      AND   AL,0FH
      OR    AL,30H
      CMP   AL,30H
      JZ    ZERO
      MOV   RESULT+1,AL
      JMP   EXX
ZERO: STC
EXX:  HLT
```

本程序对 BUF 单元中的两位 BCD 数分离后，用 OR　AL,30H 将 AL 中的 BCD 数转换为 ASCII 码，这是因为'0'到'9'的 ASCII 码为 30H 至 39H。同理，用比较指令 CMP

AL，30H 可判断 AL 中内容是否为 30H，若是，则 ZF＝1；否则，ZF＝0。程序针对这两种情况，作两种不同的处理。

3.2.4 程序控制类指令

在 8086/8088 指令系统中，程序的执行序列是由代码段寄存器 CS 和指令指针 IP 确定的。CS 包含当前指令所在代码段的段地址，IP 则是要执行的下一条指令的偏移地址。程序的执行一般依指令序列顺序执行，但有时需用改变程序的流程。控制转移类指令通过修改 CS 和 IP 寄存器的值来改变程序的执行顺序，包括五组指令：无条件转移指令、有条件转移指令、循环指令、过程调用和返回指令以及中断指令。

利用程序控制类指令，可以实现分支、循环、子程序等程序结构。

1. 无条件转移指令

指令格式：

```
JMP  OPRD
```

其中，OPRD 为转移的目的地址。程序转移到目的地址所指向的指令后继续向下执行。

无条件转移，就是无任何先决条件就能使程序改变执行顺序。处理器只要执行无条件转移指令 JMP，就能使程序转移到指定的目标地址处。

目标地址操作数的寻址方法可以是相对寻址、直接寻址或间接寻址。相对寻址方式以当前 IP 为基础，加上位移量构成目标地址。目标地址像立即数一样，直接在指令的机器代码中就是直接寻址方式。目标地址如果在寄存器或主存单元中，就是通过寄存器或存储器的间接寻址方式。

相对寻址方式根据位移量进行转移，方便了程序段在内存中的动态加载，是最常用的目标地址寻址方式。例如，同样的一段程序，如果改变了内存地址，转移的目的地址也就改变了，但是转移指令与目的指令之间的位移并没有因此改变。JMP 指令可以将程序转移到 1MB 存储空间的任何位置。根据跳转的距离，JMP 指令分成了段内转移和段间转移。

段内转移是指在当前代码段 64KB 范围内转移，因此不需要更改 CS 段地址，只要改变 IP 偏移地址。如果转移范围用 1 字节(－128～127)表达，则可以形成所谓的“短转移 short jump”；如果地址位移用一个 16 位数表达，则形成“近转移 near jump”，它是在±32KB 范围内。

段间转移是指从当前代码段跳转到另一个代码段，此时需要更改 CS 段地址和 IP 偏移地址，这种转移也称为“远转移 far jump”。转移的目标地址必须用一个 32 位数表达，叫作 32 位远指针，它就是逻辑地址。

由此可见，JMP 指令根据目标地址不同的提供方法和内容，可以分为以下 4 种格式。

(1) 段内转移，相对寻址。

(2) 段内转移，间接寻址。

(3) 段间转移，直接寻址。

(4) 段间转移，间接寻址。

以下指令为合法的无条件转移指令：

```
JMP  SHORT TARGET
JMP  TARGET
JMP  AX
JMP  TABLE[BX]
JMP  WORD PTR [BP][DI]
JMP  FAR PTR LABLE
JMP  VAR_DOUBLEWORD
JMP  DWORD PTR [BP][DI]
```

2. 条件转移指令

指令格式：

```
JCC  OPRD
```

条件转移指令只有一个操作数 OPRD，用以指明转移的目的地址。指令助记符中的“CC”表示条件。这种指令的执行包括两个过程：第一步，测试规定的条件；第二步，如果条件满足，则转移到目标地址；否则，继续顺序执行。

条件转移指令的操作数必须是一个短标号，也就是说，所有的条件转移指令都是两字节指令，转移指令的下一条指令到目标地址之间的距离必须为－128～127。如果指令规定的条件满足，则将这个位移量加到 IP 寄存器上，以实现程序的转移。

绝大多数条件转移指令（除 JCXZ 指令外）将状态标志位的状态作为测试的条件。因此，首先应该执行影响有关的状态标志位的指令，然后才能用条件转移指令测试这些标志，以确定程序是否转移。CMP 和 TEST 指令常常与条件转移指令配合使用，因为这两条指令不改变目的操作数的内容，但可以影响状态标志位。

8086/8088 的条件转移指令非常丰富，不仅可以测试一个状态标志位的状态，而且可以综合测试几个状态标志位；不仅可以测试无符号数的高低，而且可以测试带符号数的大小等，编程时使用十分灵活、方便。所有的条件转移指令的名称、助记符及转移条件等列在表 3.2 中。其中同一行内用斜杠隔开的几个助记符，实质上代表同一条指令的几种不同的表示方法。

表 3.2　条件转移指令

指令名称	助记符	转移条件	说明
等于/零转移	JE/JZ	ZF＝1	判断单个标志位状态
不等于/非零转移	JNE/JNZ	ZF＝0	
负转移	JS	SF＝1	
正转移	JNS	SF＝0	
“1”的个数为偶转移	JP/JPE	PF＝1	
“1”的个数为奇转移	JNP/JPO	PF＝0	
溢出转移	JO	OF＝1	
不溢出转移	JNO	OF＝0	
进位转移	JC	CF＝1	
不进位转移	JNC	CF＝0	

续表

指令名称	助记符	转移条件	说明
低于/不高于或等于转移	JB/JANE	CF=1	用于无符号数的比较
高于或等于/不低于转移	JAE/JNB	CF=0	
高于/不低于或等于转移	JA/JNBE	CF=1 且 ZF=0	
低于或等于/不高于转移	JBE/JNA	CF=0 或 ZF=1	
大于/不小于或等于转移	JG/JNLE	SF=OF 且 ZF=0	用于带符号数的比较
大于或等于/不小于转移	JGE/JNL	SF=OF	
小于/不大于或等于转移	JL/JNGE	SF≠OF 且 ZF=0	
小于或等于/不大于转移	JLE/JNG	SF≠OF 或 ZF=1	
CX 等于零转移	JCXZ	(CX)=0	不用判断标志位状态

比较两个无符号数大小的指令，通常是根据一个标志位或两个标志位以确定两个数的大小。为了与带符号数的大小相区别，无符号数比较常用高于、低于来表示。带符号数比较常用大于、小于来表示。以上两种表示方法一定要能正确地区分开，否则就不能正确理解和使用这些指令。

【例 3.23】 设计一段程序实现以下功能：如果 AL 最高位为 0，则设置 AH=0；如果 AL 最高位为 1，则设置 AH=FFH。

程序段如下：

```
TEST  AL,80H
JZ  NEXT0
MOV  AH,0FFH
JMP  DONE
NEXT0: MOV  AH,0
DONE: …
```

【例 3.24】 设 X 和 Y 为存放于 X 单元和 Y 单元的 16 位操作数，计算 $|X-Y|$，并将结果存入 RESULT 单元中。

程序段如下：

```
MOV  AX,X
SUB  AX,Y
JNS  NONNEG
NEG  AX
NONNEG: MOV  RESULT,AX
```

【例 3.25】 完成下式的判定运算。

$$Y=\begin{cases}1 & X\geqslant 0\\ 0 & X<0\end{cases}$$

实现的程序如下：

```
MOV  AL,X
CMP  AL,0
```

```
        JGE   A1
        MOV   AL,0
        JMP   A2
A1:  MOV   AL,1
A2:  MOV   Y,AL
      ⋮
```

以上程序段中的 X、Y 是两个存储器变量，把 X 当成带符号数与 0 比较。

3. 循环控制指令

循环是一种特殊的转移流程，当满足（不满足）某条件时，反复执行一系列操作，直到不满足（满足）条件为止。循环流程的条件一般是循环计数，指令约定用 CX 寄存器作为计数器。在程序中用循环计数来控制循环次数。

这类指令属于段内 SHORT 短类型转移，目的地址必须距本指令在－127～128 个字节的范围内。循环指令不影响标志位。

1）循环指令

指令的一般格式为：

```
LOOP   标号
```

功能：(CX)←(CX)－1，(CX)≠0，则转移至标号处循环执行，直至(CX)＝0，继续执行后续程序。

LOOP 指令的操作是先将 CX 的内容减 1，如结果不等于 0，则转到指令中指定的短标号处；否则，顺序执行下一条指令。因此，在循环程序开始前，应将循环次数送 CX 寄存器。

2）条件循环指令

指令的一般格式为：

```
LOOPZ/LOOPE   标号
```

功能：(CX)←(CX)－1，(CX)≠0，且 ZF＝1 时，转移至标号处循环。

LOOPZ 和 LOOPE 实际上代表同一条指令。本指令的操作也是先将 CX 寄存器的内容减 1，如结果不为零，且零标志 ZF＝1，则转移到指定的短标号处。

3）条件循环指令

指令的一般格式为：

```
LOOPNZ/LOOPNE   标号
```

功能：(CX)←(CX)－1，(CX)≠0，且 ZF＝0 时，转移至标号处循环。

本指令也同样有两种表示形式。指令的操作是将 CX 寄存器的内容减 1，如结果不为 0，且零标志 ZF＝0（表示“不相等”或“不等于 0”），则转移到指定的短标号处。

4. 过程调用和返回指令

如果有一些程序段需要在不同的地方多次反复地出现，则可以将这些程序段设计成为过程（相当于子程序），每次需要时进行调用。过程结束后，再返回到原来调用的地方。采用这种方法不仅可以使源程序的总长度大大缩短，而且有利于实现模块化的程序设计，使程序的编制、阅读和修改都比较方便。

1）过程调用指令

指令格式：

```
CALL  OPRD
```

其中，OPRD为过程的目的地址。过程调用可以分为段内调用和段间调用两种。寻址方式也可以分为直接寻址和间接寻址两种。本指令不影响标志位。

（1）段内直接调用。

指令格式：

```
CALL  NEAR类型的过程名
```

每一个过程在定义时，应指定它是近类型（NEAR），还是远类型（FAR）。本指令是段内直接调用，因而过程与调用指令同处在一个代码段内。在执行该调用指令时，首先将IP的内容入栈保护，然后将指令代码给出的目的地址的段内偏移量送入IP中，从而实现过程调用，将程序转至过程入口。

（2）段内间接调用。

指令格式：

```
CALL  OPRD
```

其中，OPRD为16位通用寄存器或存储器数。本指令执行时，首先将IP的内容入栈保护，然后将目的地址在段内偏移量由指定的16位寄存器或存储器字中取至IP中，从而实现过程调用。

（3）段间直接调用。

指令格式：

```
CALL  FAR类型的过程名
```

由于是段间调用，在指令执行时，应同时将当前的CS及IP的值入栈保护，然后将FAR类型的过程名所在的段基址和段内偏移值送CS及IP，从而实现过程调用。

（4）段间间接调用。

指令格式：

```
CALL  DWORD
```

其中，DWORD为存储器操作数。段间间接调用只能通过存储器双字进行。本指令执行时，首先将当前的CS及IP的值入栈保护，然后将存储器双字操作数的第一个字的内容送IP，将第二个字的内容送CS，以实现段间调用。

2）返回指令

指令格式：

```
RET
```

本指令的作用是：当调用的过程结束后实现从过程返回至原调用程序的下一条指令。本指令不影响标志位。

由于在过程定义时，已指明其近（NEAR）或远（FAR）的属性，因此RET指令将根据段

内调用与段间调用，执行不同的操作。

对段内调用，返回时，由堆栈弹出一个字的返回地址的段内偏移量至 IP。

对段间调用，返回时，由堆栈弹出的第一个字为返回地址的段内偏移量，将其送入 IP 中，由堆栈弹出的第二个字为返回地址的段基址，将其送入 CS 中。

5. 中断指令

在程序运行时，遇到某些紧急情况或一些严重的错误（如溢出），当前程序应能够暂停，处理器中止当前程序运行，转去执行处理这些紧急情况的程序段。这种情况称为“中断”。转去执行的处理中断的子程序称为“中断服务程序”或“中断处理程序”。当前程序被中断的地方称为“断点”。中断服务程序执行完后应返回原来程序的断点，继续执行被中断的程序。中断提供了又一种改变程序执行顺序的方法。

8086/8088 具有很强的中断系统，可以处理 256 个不同方式的中断。每一个中断赋予一个中断向量码，CPU 根据向量码的不同来识别不同的中断源。8086/8088 内部中断源有除法错中断、单步中断、断点中断、溢出中断、用户自定义的软中断 5 种类型。外部中断是指来自 CPU 之外的原因引起的程序中断，分为可屏蔽中断和非屏蔽中断两种类型。

1）溢出中断指令

指令格式：

```
INTO
```

功能：本指令检测 OF 标志位，当 OF＝1 时，说明已发生溢出，立即产生一个中断类型 4 的中断，当 OF＝0 时，本指令不起作用。

2）软中断指令

指令格式：

```
INT  n
```

其中，n 为软中断的类型号。

功能：本指令将产生一个软中断，把控制转向一个类型号为 n 的软中断，该中断处理程序入口地址在中断向量表的 $n\times4$ 地址处的两个存储字（4 个单元）中。

3）中断返回指令

指令格式：

```
IRET
```

功能：用于中断处理程序中，从中断程序的断点处返回，继续执行原程序。

本指令将影响所有标志位。

无论是软中断，还是硬中断，本指令均可使其返回到中断程序的断点处继续执行原程序。

3.2.5　串操作类指令

串操作类指令是一组具有修改数据串操作指针功能的指令。数据串可以是字节串，也可以是字串，即每一个数据占用两个存储单元。数据串只能放在存储器中，对数据串的数据进行处理时，可以只对一个数据串进行，也可以对两个数据串进行，根据数据串中数据的流

动方向，可以分源数据串和目的数据串。

8086/8088 指令系统还为串操作类指令提供重复前缀，以便重复进行相同的操作。

在串操作指令中，源操作数用寄存器 SI 寻址，默认在数据段 DS 中，但允许段超越；目的操作数用寄存器 DI 寻址，默认在附加段 ES 中，不允许段超越。每执行一次串操作指令，作为源地址指针的 SI 和作为目的地址指针的 DI 将自动修改：±1（对于字节串）或±2（对于字串）。地址指针是增加还是减少则取决于方向标志 DF。在系统初始化后或执行指令 CLD 后，DF＝0，此时地址指针是增 1 或 2；在执行指令 STD 后，DF＝1，此时地址指针减 1 或 2。

1. 串传送指令

指令格式：

```
MOVS  OPRD1,OPRD2
MOVSB
MOVSW
```

其中，OPRD1 为目的串符号地址，OPRD2 为源串符号地址。

功能：OPRD1←OPRD2。

串传送指令 MOVS 将数据段主存单元的 1 字节或字，传送到附加段的主存单元中。定义数据串时，要求源串和目的串类型一致，并以其类型区别是字节或字操作。在指令中不出现操作数时，字节串传送格式为 MOVSB，字串传送格式为 MOVSW。MOVS 指令不影响标志位。

（1）对字节串操作时，若 DF＝0，则做加，即：

```
[ES: DI] ← [DS: SI],(SI)←(SI) + 1,(DI)←(DI) + 1
```

若 DF＝1，则做减，即：

```
(SI)←(SI) - 1,(DI)←(DI) - 1
```

（2）对字串操作时，若 DF＝0，则做加，即：

```
(SI)←(SI) + 2,(DI)←(DI) + 2
```

若 DF＝1，则做减，即：

```
(SI)←(SI) - 2,(DI)←(DI) - 2
```

【例 3.26】 将存储器中变量 BUFA 开始的 160 个数据串传送至 BUFB 开始的存储区，可用以下程序段实现。

```
      MOV   SI,OFFSET BUFA
      MOV   DI,OFFSET BUFB
      MOV   CX,160
      CLD
AGAIN:MOVS  BUFB,BUFA
      DEC   CX
      JNZ   AGAIN
```

2. 串比较指令

指令格式：

```
CMPS  OPRD1,OPRD2
CMPSB
CMPSW
```

其中，OPRD1 为目的串符号地址，OPRD2 为源串符号地址。

串比较指令 CMPS 将由 SI 寻址的源串中的数据与 DI 寻址的目的串中的数据（字或字节）进行比较，比较结果送标志位，而不改变操作数本身。同时，SI、DI 将自动调整。

CMPS 指令影响标志位 AF、CF、OF、SF、PF、ZF。CMPS 指令可用来检查两个字符串是否相同，可以使用循环控制方法对整串进行比较。

【例 3.27】 如对两个字节串进行比较，若一致，则 AL 内容置为 0；若不一致，则 AL 内容置为 0FFH。程序段如下：

```
      MOV  SI,OFFSET DAT1        ; DAT1 中是内存中定义的字节串 1
      MOV  DI,OFFSET DAT2        ; DAT2 中是内存中定义的字节串 2
      MOV  CX,N
      CLD
NEXT: CMPSB
      JNZ  FIN
      DEC  CX
      JNZ  NEXT
      MOV  AL,0
      JMP  EXX
FIN:  MOV  AL,0FFH
EXX:  MOV  DAT3,AL               ; DAT3 是内存中存放结果的单元
```

3. 串扫描指令

指令格式：

```
SCAS  OPRD
SCASB
SCASW
```

其中，OPRD 为目的串符号地址。

串扫描指令 SCAS 将 AL 或 AX 的内容与附加段中由 DI 寄存器寻址的目的串中的数据进行比较，根据比较结果设置标志位，但不改变操作数本身。每次比较后修改 DI 寄存器的值，使之指向下一个元素。SCAS 指令影响标志位 AF、CF、OF、SF、PF、ZF。

SCAS 指令可查找字符串中的一个关键字，只需在本指令执行前，把关键字放在 AL 或 AX 中，用重复前缀可在整串中查找。

【例 3.28】 在附加段定义了一个字符串，首地址由 STRING 指示，共有 100 个字符。在字符串中查找“空格”（ASCII 码为 20H）字符。

```
MOV  DI,OFFSET STRING
MOV  AL,20H
MOV  CX,100
CLD
```

```
AGAIN: SCASB
JZ  FOUND          ; ZF = 1,发现空格,转移到 FOUND
DEC  CX            ; 不是空格
JNZ  AGAIN         ; 搜索下一个字符
⋮                  ; 不含空格,则继续执行
FOUND: …
```

4. 串读取指令

指令格式：

```
LODS  OPRD
LODSB
LODSW
```

其中，OPRD 为源串符号地址。

串读取指令 LODS 的功能是把 SI 寻址的源串的数据字节送 AL 或数据字送 AX 中，并根据 DF 的值，地址指针 SI 进行自动调整。LODS 指令不影响标志位。

5. 字符串存储指令

指令格式：

```
STOS  OPRD
STOSB
STOSW
```

其中，OPRD 为目的串符号地址。

本指令的功能是把 AL 或 AX 中的数据存储到 DI 为目的串地址指针所寻址的存储器单元中。地址指针 DI 将根据 DF 的值进行自动调整。STOS 指令与 LODS 指令功能互逆。STOS 指令不影响标志位。

【例 3.29】 将附加段 64KB 主存区全部设置为 0。

```
MOV  AX,0
MOV  DI,0
MOV  CX,8000H          ; CX←传送次数(32×1024)
CLD                    ; 设置 DF = 0,实现地址增加
AGAIN: STOSW           ; 传送一个字
DEC  CX
JNZ  AGAIN             ; 判断传送次数 CX 是否为 0
```

在此例中，将 CLD 指令改为 STD 指令就能反向传送，实现同样功能。另外，此例中实际上只要保证 DI 为偶数即可。

【例 3.30】 数据段 DS 中有一个数据块，具有 100 个字节，起始地址为 BBUF。现在要把其中的正数、负数分开，分别存入同一个段的两个缓冲区。存放正数的起始地址为 DATAA，存放负数的起始地址为 DATAB。

```
MOV  SI,OFFSET BBUF
MOV  DI,OFFSET DATAA
MOV  BX,OFFSET DATAB
MOV  AX,DS
MOV  EX,AX             ; 所有数据都在一个段中,所以设置 ES = DS
```

```
MOV  CX,100
CLD
GOON: LODSB              ; 从 BBUF 中取出一个数据
TEST  AL,80H             ; 检测符号位,判断是正是负
JNZ  MINUS               ; 符号位为 1,是负数,转向 MINIS
STOSB                    ; 符号位为 0,是正数,存入 DATAA
JMP  AGAIN
MINUS: XCHG  BX,DI
STOSB                    ; 将负数存入 DATAB
XCHG  BX,DI
AGAIN: DEC  CX
JNZ  GOON
```

6. 重复前缀的说明

在串操作指令前加上重复前缀,可以对数据串进行重复处理。由于加上重复前缀后,对应的指令代码是不同的,因此指令的功能便具有重复处理的功能,重复的次数存放在 CX 寄存器中。

重复前缀形式有:

```
REP                      ; CX≠0,重复执行字符串指令
REPZ/REPE                ; CX≠0 且 ZF = 1,重复执行字符串指令
REPNZ/REPNE              ; CX≠0 且 ZF = 0,重复执行字符串指令
```

REP 与 MOVS 或 STOS 串操作指令结合使用,完成一组数据的传送或建立一组相同数据的数据串。

REPZ/REPE 与 CMPS 串操作指令结合使用,可以完成两组数据串的比较。当串未结束时,继续重复执行数据串指令。它可用来判定两数据串是否相同。

REPZ/REPE 与 SCAS 串操作指令结合使用,可以完成在一个数据串中搜索一个关键字。只要当数据串未结束且当关键字与元素相同时,继续重复执行串搜索指令,用于在数据串中查找与关键字不相同的数据的位置。

REPNZ/REPNE 与 CMPS 指令结合使用,表示当串未结束且当对应串元素不相同时,继续重复执行串比较指令。它可在两数据串中查找相同数据的位置。

REPNZ/REPNE 与 SCAS 指令结合使用,表示串未结束且当关键字与元素不相同时,继续重复执行串搜索指令,用于在数据串中查找与关键字相同的数据的位置。

【例 3.31】 对两个字符串 STR1 与 STR2 进行比较。

```
      MOV  SI,OFFSET STR1
      MOV  DI,OFFSET STR2
      MOV  CX,COUNT
      CLD
REPZ  CMPSB
      JNZ  NEQU
      MOV  AL,0
      JMP  OVR
NEQU: MOV  AL,0FFH
OVR:  MOV  RESULT,AL
      HLT
```

【例 3.32】 在字符串中搜索关键字，记下搜索的次数和关键字在串中的位置。

```
        CLD
        MOV   DI,OFFSET BUF
        MOV   CX,COUNT
        MOV   AL,CHAR
REPNE   SCASB
        JZ    FOUND
        MOV   DI,0
        JMP   DONE
FOUND:  DEC   DI
        MOV   BUFF,DI
        MOV   BX,OFFSET BUF
        SUB   DI,BX
        MOV   BUFF + 2,DI
DONE:   HLT
```

在本程序中，由于 DI 是自增的，若找到关键字，这时 DI 已指向关键字的下一个字符，故 DI 减 1 才是真正的关键字在字符串中的位置。用当前关键字的位置减去串首地址，即能得到搜索的次数。

3.2.6 处理器控制类指令

处理器控制指令用于控制 CPU 的动作，修改标志寄存器的状态等，实现对 CPU 的管理。

1. 标志位操作指令

标志位操作指令有 7 条，可以直接设置或清除 CF、DF 和 IF 标志位。例如，串操作中的程序，经常用 CLD 指令清方向标志使 DF＝0，在串操作指令执行时，按增量的方式修改串指针。

标志位操作指令的格式、功能等信息列于表 3.3 中。

表 3.3 标志位操作指令

指令格式	功能说明
CLC	CF＝0，进位标志位置 0
STC	CF＝1，进位标志位置 1
CMC	进位标志位求反
CLD	DF＝0，方向标志位置 0
STD	DF＝1，方向标志位置 1
CLI	IF＝0，中断标志位置 0，使 CPU 禁止响应外部中断
STI	IF＝1，中断标志位置 1，使 CPU 允许响应外部中断

这些指令仅对有关状态标志位执行操作，而对其他状态标志位则没有影响。

2. CPU 控制指令

1）处理器暂停指令

指令格式：

```
HLT
```

HLT 指令使 CPU 进入暂停状态，这时 CPU 不进行任何操作。当 CPU 发生复位(RESET)或来自外部的中断(NMI 或 INTR)时，CPU 脱离暂停状态。HLT 指令不影响标志位。

HLT 指令可用于程序中等待中断。当程序中必须等待中断时，可用 HLT，而不必用软件死循环。然后，中断使 CPU 脱离暂停状态，返回执行 HLT 的下一条指令。

注意：该指令在 PC 中将引起所谓的“死机”，一般的应用程序不要使用。

2) 处理器等待指令

指令格式：

```
WAIT
```

WAIT 指令在 8086/8088 的测试输入引脚为高电平无效时，使 CPU 进入等待状态，这时，CPU 并不做任何操作；测试为低电平有效时，CPU 脱离等待状态，继续执行 WAIT 指令后面的指令。WAIT 指令不影响标志位。

浮点指令经由 8086/8088 CPU 处理发往 8087，并与 8086/8088 本身的整数指令在同一个指令序列；而 8087 执行浮点指令较慢，所以 8086/8088 必须与 8087 保持同步。8086/8088 就是利用 WAIT 指令和测试引脚实现与 8087 同步运行的。

3) 处理器交权指令

指令格式：

```
ESC   EXTOPRD,OPRD
```

其中，EXTOPRD 为外部操作码(浮点指令的操作码)，是一个 6 位立即数；OPRD 为源操作数，可以是寄存器或内存单元。当 OPRD 为寄存器时，它的编码也作为操作码；如果为存储器操作数，CPU 读出这个操作数送给协处理器。

交权指令 ESC 把浮点指令交给浮点处理器执行。为了提高系统的浮点运算能力，8086/8088 系统中可加入浮点运算协处理器 8087。但是，8087 的浮点指令是和 8086/8088 的整数指令组合在一起的，8086/8088 主存中存储 8087 的操作码及其所需的操作数。当 8086/8088 发现是一条浮点指令时，就利用 ESC 指令将浮点指令交给 8087 执行。

ESC 指令不影响标志位。

4) 空操作指令

指令格式：

```
NOP
```

NOP 指令不执行任何有意义的操作，但占用 1 字节存储单元，空耗一个指令执行周期。该指令常用于程序调试。例如，在需要预留指令空间时用 NOP 填充，代码空间多余时也可用 NOP 填充，还可以用 NOP 指令实现软件延时。事实上，NOP 指令就是 XCHG　AX，AX 指令，它们的代码一样。NOP 指令不影响标志位。

5) 封锁总线指令

LOCK 是一个指令前缀，可放在指令的前面。这个前缀使得在当前指令执行时间内，8086/8088 处理器的封锁输出引脚有效，即把总线封锁，使别的控制器不能控制总线，直到

该指令执行完后，总线封锁解除。当CPU与其他处理器协同工作时，LOCK指令可避免破坏有用信息。

6）段超越前缀指令

```
SEG:   ; 即 CS:,SS:,DS:,ES:,取代默认段寄存器
```

在允许段超越的存储器操作数之前，使用段超越前缀指令，将不采用默认的段寄存器，而是采用指定的段寄存器寻址操作数。

3.2.7 输入/输出类指令

输入/输出指令共有两条。输入指令IN用于从外设端口接收数据，输出指令OUT则向端口发送数据。无论是接收到的数据或是准备发送的数据都必须在累加器AL(字节)或AX(字)中，所以这是两条累加器专用指令。

输入/输出指令可以分为两大类：一类是端口直接寻址的输入/输出指令；另一类是端口通过DX寄存器间接寻址的输入/输出指令。在直接寻址的指令中只能寻址256个端口(0～255)，而间接寻址的指令中可寻址64K个端口(0～65 535)。

1. 输入指令

指令格式：

```
IN  AL,n      (AL)←(n)
IN  AX,n      (AX)←(n+1),(n)
IN  AL,DX     (AL)← [(DX)]
IN  AX,DX     (AX)← [(DX+1)],[(DX)]
```

其中，n为8位的端口地址，当字节输入时，将端口地址n的内容送至AL中；当字输入时，将端口地址$n+1$的内容送至AH中，端口地址n的内容送至AL中。

端口地址也可以是16位的，但必须将16位的端口地址送入DX中。当字节寻址时，由DX内容作端口地址的内容送至AL中；当输入数据字时，[(DX+1)]送AH，[(DX)]送AL中，用符号(AX)← [(DX+1)],[(DX)]表示。

指令举例：

```
IN  AL,20H
```

或：

```
MOV  DX,0400H
IN  AL,DX
```

2. 输出指令

指令格式：

```
OUT  n,AL     (n)←(AL)
OUT  n,AX     (n+1),(n)←(AX)
OUT  DX,AL    [(DX)]←(AL)
OUT  DX,AX    [(DX+1)],[(DX)]←(AX)
```

OUT 指令中各个操作数的定义与 IN 指令相同。

指令举例：

```
MOV  AL,0FH
OUT  20H,AL
```

或：

```
MOV  DX,0400H
MOV  AL,86H
OUT  DX,AL
```

输入/输出指令对标志位不产生影响。

3.3　80x86 指令系统介绍

80x86 系列微处理器指令系统保持向上兼容，如 80486 微处理器可兼容执行 8086、80286 和 80386 的指令。在介绍了 8086/8088 指令系统的基础上，本节讲述 80286、80386、80486 和 Pentium 的新增指令以及在 8086/8088 基础上扩充的新的功能的一些指令。

3.3.1　80x86 寻址方式

由于 80x86 系列 CPU 对 8086/8088 的指令是向上兼容的，因此此前所介绍的 8086/8088 的寻址方式也适用于 80x86。下面介绍的寻址方式都是针对存储器操作数的寻址方式，它们均与比例因子有关，这些寻址方式只能用在 80386 及其后继机型中，8086/8088/80286 不支持这几种寻址方式。

在 80386 及其后续机型中，8 个 32 位的通用寄存器 EAX、EBX、ECX、EDX、ESP、EBP、ESI、EDI 既可以存放数据，也可以存放地址，也就是说，这些寄存器都可以用来提供操作数在段内的偏移地址。

1. 比例变址寻址方式

操作数的有效地址是变址寄存器的内容乘以指令中指定的比例因子再加上位移量之和，所以有效地址由 3 种成分组成。这种寻址方式与寄存器相对寻址相比，增加了比例因子，其优点在于：对于元素大小为 2、4、8 字节的数组，可以在变址寄存器中给出数组元素下标，而由寻址方式控制直接用比例因子把下标转换为变址值。

例如：

```
MOV  EAX,COUNT[ESI×4]
```

假设(DS)＝4000H，位移量为 500H。如果要求把双字数组中的元素 3 送到 EAX(EAX 为 32 位累加器)中，用这种寻址方式可以直接在 ESI 中放入 3，选择比例因子 4(数组元素为 4 字节长)就可以方便地达到目的，不必像在寄存器相对寻址方式中要把变址值直接装入寄存器中。物理地址的计算如图 3.14 所示。

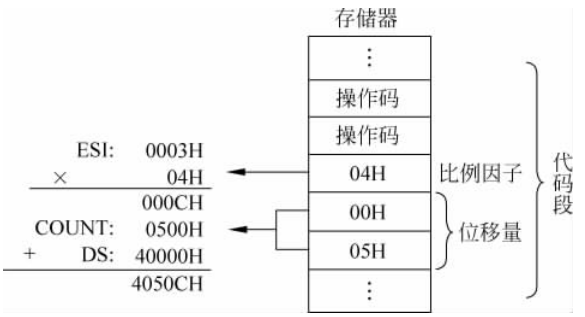

图 3.14 比例变址寻址方式

2. 基址比例变址寻址方式

操作数的有效地址是变址寄存器的内容乘以比例因子再加上基址寄存器的内容，所以有效地址由3种成分组成。这种寻址方式与基址变址寻址方式相比，增加了比例因子，其优点是很明显的。

例如：

```
MOV  ECX,[EAX][EDX×8]
```

此例中，EAX作为基址，EDX作变址，比例因子为8。

3. 相对基址比例变址寻址方式

操作数的有效地址是变址寄存器的内容乘以比例因子，加上基址寄存器的内容，再加上位移量之和，所以有效地址由4种成分组成。这种寻址方式比相对基址变址寻址方式增加了比例因子，便于对元素为2、4、8字节的二维数组的处理。

例如：

```
MOV  EAX,TABLE[EBP][EDI×4]
```

在此例中，EBP作为基址，EDI作变址，比例因子为4，位移量为TABLE。

3.3.2 80286指令系统新增指令

80286指令系统除了包括所有的8086/8088指令外，新增指令以及增强了功能的指令列于表3.4中。其中，控制保护态指令是80286工作在保护模式下的一些特权方式指令，常用于操作系统及其他控制软件中。保护模式是集实地址模式的能力、存储器管理、对虚拟存储器的支持和对地址空间的保护为一体而建立起来的一种特殊工作方式。

表 3.4 80286增强与增加的指令

类别	增强的指令	增加的指令
数据传送类	PUSH 立即数	PUSHA POPA
算术运算类	IMUL 寄存器,寄存器 IMUL 寄存器,存储器 IMUL 寄存器,立即数 IMUL 寄存器,寄存器,立即数 IMUL 寄存器,存储器,立即数	

续表

类别	增强的指令	增加的指令
逻辑运算与移位类	SHL 目的操作数,立即数(1～31) 其余 SAL、SAR、SHR、ROL、ROR、RCL、RCR 7 条移位指令同 SHL	
串操作类		[REP]INS 目的串,DX [REP]OUTS DX,源串 [REP]INSB/OUTB [REP]INSW/OUTW
高级语言		BOUND 寄存器,存储器 ENTER 立即数 16,立即数 LEAVE
保护模式的系统控制指令类	LAR(装入访问权限)、LSL(装入段界限)、LGDT(装入全局描述符表)、SGDT(存储全局描述符表)、LIDT(装入 8 字节中断描述符表)、SIDT(存储 8 字节中断描述符表)、LLDT(装入局部描述符表)、SLDT(存储局部描述符表)、LTR(装入任务寄存器)、STR(存储任务寄存器)、LMSW(装入机器状态字)、SMSW(存储机器状态字)、VERR(存储器或寄存器读校验)、VERW(存储器或寄存器写校验)、ARPL(调整已请求特权级别)、CLTS(清除任务转移标志)	

下面介绍 80286 指令系统中主要的新增及扩展指令。

1. 堆栈操作指令

指令格式:

```
PUSH  16 位立即数
```

PUSH 指令将 16 位立即数压入堆栈,如果给出的数不够 16 位,则自动扩展为 16 位后压入堆栈。该指令不影响状态标志位。在 8086/8088 指令系统中,PUSH 指令允许的操作数只能是两字节的寄存器操作数或存储器操作数。

2. 通用寄存器入栈操作指令

指令格式:

```
PUSHA
```

PUSHA 指令将所有通用寄存器 AX、CX、DX、BX、SP、BP、SI 和 DI 的内容按顺序压入堆栈,入栈的 SP 值是执行该指令之前的 SP 值,在执行完本指令后,SP 值减 16。

3. 通用寄存器内容出栈操作指令

指令格式:

```
POPA
```

POPA 指令将栈顶的内容顺序弹至 DI、SI、BP、SP、BX、DX、CX 和 AX。SP 中的值是堆栈中所有通用寄存器弹出后,堆栈指针实际指向的值(不是栈中保存的 SP 值),也即该指令执行后 SP 的值,可以通过加 16 来恢复。

4. 有符号整数乘法指令(两个操作数)

指令格式:

```
IMUL  16位寄存器,立即数
```

有符号整数乘法指令IMUL将16位通用寄存器中的有符号数作为被乘数，与有符号立即数相乘，乘积送回通用寄存器。若乘积超出有符号数的表示范围(－32 768～＋32 767)，除丢失溢出部分外，并将OF及CF置为1；否则，将OF及CF置为0。

5. 有符号整数乘法指令(3个操作数)

指令格式：

```
IMUL  16位寄存器,16位存储器,立即数
```

该指令与上一条指令功能类似，区别仅在于，将16位存储器操作数作为被乘数与立即数相乘，结果送16位寄存器。

6. 移位指令

8086/8088中有8条移位指令，移位计数使用CL或1表示，且规定当移位次数大于1时，必须使用CL。在80286中，将上述限制修改为当移位次数为1～31次时，允许使用立即数。

7. 串输入/输出指令INS/OUTS

指令格式：

```
[REP]INS  目的串,DX
[REP]OUTS  DX,源串
[REP]INSB
[REP]OUTB
[REP]INSW
[REP]OUTW
```

串输入/输出指令可以带两个操作数，也可以采用默认操作数形式，指令中不指出操作数，但要在指令助记符中用B或W指明输入/输出的数据串是字节还是字。该类指令可以实现DX指定的端口与由指定的内存地址之间的数据块传送，其类型可以是字节或字。如果加重复前缀REP，完成整个串的输入/输出操作，这时CX寄存器中为重复前缀操作的次数。

8. 内存范围检查指令

指令格式：

```
BOUND  16位寄存器,32位存储器
```

BOUND指令以32位存储器低两字节的内容为下界，高两字节的内容为上界。若16位寄存器的内容在此上、下界表示的地址范围内，程序正常执行；否则产生INT 5中断。当出现这种中断时，返回地址指向BOUND指令，而不是BOUND后面的指令，这与返回地址指向程序中下一条指令的正常中断是有区别的。

9. 设置堆栈空间指令

格式：

```
ENTER  16位立即数,8位立即数
```

在ENTER指令中，两个操作数中的16位立即数表示堆栈空间的大小，也即表示给当前过程分配多少字节的堆栈空间，8位立即数指出在高级语言内调用自身的次数，也即嵌套

层数。值得注意的是该指令使用 BP 寄存器而非 SP 作为栈基值。

10. 撤销堆栈空间指令

指令格式：

```
LEAVE
```

LEAVE 指令撤销由 ENTER 指令建立的堆栈空间。

例如：

```
TASK   PROC NEAR
       ENTER   400,8      ; 建立堆栈空间为 400 字节,允许过程嵌套 8 层
       ⋮
       LEAVE              ; 释放堆栈空间
       RET
TASK   ENDP
```

3.3.3　80386 指令系统新增指令

80386 指令系统包括了所有 80286 指令，并对 80286 的部分指令进行了功能扩充，还新增了一些指令，特别指出的是，80386 提供了 32 位寻址方式，可对 32 位数据直接操作。所有 16 位指令均可扩充为 32 位指令。表 3.5 列出了 80386 增强及增加的指令。

表 3.5　80386 增强与增加的指令

类　　别	增强的指令	增加的指令
数据传送类	PUSH 立即数 PUSHAD/POPAD PUSHFD/POPFD	MOVSX 寄存器,寄存器/存储器 MOVZX 寄存器,寄存器/存储器
算术运算类	IMUL 寄存器,寄存器/存储器 IMUL 寄存器,寄存器/存储器,立即数 CWDE CDQ	
逻辑运算与移位类		SHLD/SHRD 寄存器/存储器,寄存器,CL/立即数
串操作类	所有串操作指令后面扩展 D,如 MOVSD、OUTD 等	
位操作类		BT/BTC/BTS/BTR 寄存器/存储器,寄存器/立即数 BSF/BSR 寄存器,寄存器/存储器
条件设置类		SET 条件 寄存器/存储器

下面介绍 80386 指令系统中主要的新增及扩展指令。

1. 数据传送与扩展指令

1）MOVSX

指令格式：

```
MOVSX  寄存器,寄存器/存储器
```

MOVSX 指令将源操作数传送到目的操作数中。目的操作数可以是 16 位或 32 位寄存器；源操作数可以是寄存器或存储器操作数，其位数应小于或等于目的操作数的位数。当源操作数的位数少于目的操作数时，目的操作数的高位用源操作数的符号位填补。此指令适用于有符号数的传送与扩展。

2）MOVZX

指令格式：

```
MOVZX  寄存器,寄存器/存储器
```

MOVZX 指令与 MOVSX 功能基本相同，唯一区别的是当源操作数的位数少于目的操作数位数时，目的操作数的高位补“0”。该指令适用于无符号数的传送与扩展。

2. 堆栈操作指令

1）PUSH

指令格式：

```
PUSH  32 位立即数
```

该指令将 32 位立即数压入堆栈。该指令执行后 SP 的值将减 4。

2）PUSHAD

指令格式：

```
PUSHAD
```

该指令将所有通用寄存器 EAX、ECX、EDX、EBX、ESP、EBP、ESI 和 EDI 的内容顺序压入堆栈，其中压入堆栈的 ESP 是该指令执行前 ESP 的值。执行该指令后，ESP 的值减 32。

3）POPAD

指令格式：

```
POPAD
```

该指令将当前栈顶内容顺序弹至 EDI、ESI、EBP、ESP、EBX、EDX、ECX 和 EAX，但是最终 ESP 的值为弹出操作对堆栈指针调整后的值（而不是堆栈中保存的 ESP 的值）。即执行该指令后 ESP 的值，可以通过增加 32 来恢复。

4）PUSHFD

指令格式：

```
PUSHFD
```

该指令将 32 位标志寄存器 EFLAGS 的内容压入堆栈。

5）POPFD

指令格式：

```
POPFD
```

该指令将当前栈顶的 4 字节内容弹至 EFLAGS 寄存器。

3. 有符号数乘法指令

1）两操作数乘法指令

指令格式：

```
IMUL　寄存器,寄存器/存储器
```

该指令将 16 位或 32 位通用寄存器中的有符号数作为被乘数，相同位数通用寄存器或存储单元中的有符号数作为乘数，乘积送目的操作数。若乘积溢出，溢出位部分将丢失，且将 OF 及 CF 置 1；否则将 OF 及 CF 清 0。

2）三操作数乘法指令

指令格式：

```
IMUL　寄存器,寄存器/存储器,立即数
```

该指令与前一个指令功能基本相同，唯一区别在于，寄存器/存储器为被乘数，立即数为乘数，乘积存放在第一个操作数中。

3）符号扩展指令

该指令将 AX 中 16 位有符号数的符号位扩展到 EAX 的高 16 位中，即把 AX 的 16 位有符号数扩展为 32 位后，送 EAX。

4）符号扩展指令

该指令将 EAX 中 32 位有符号数扩展到 EDX：EAX 寄存器中，使之成为 64 位有符号数，即将 EAX 中的符号位扩展到 EDX 中。

4. 移位指令

80386 中新增加了一组移动多位的指令。它们可以把指定的一组位左移或右移到一个操作数中。

1）SHLD

指令格式：

```
SHLD　寄存器/存储器,寄存器,CL/立即数
```

该指令将第一操作数(16 位或 32 位)左移若干位(由 8 位立即数或 CL 指定)，空出位用第二操作数(与第一操作数等位宽)高位部分填补，但第二操作数的内容不变，CF 标志位中保留第一操作数最后的移出位。若仅移一位，当 CF 值与移位后的第一操作数的符号位不一致时，OF 置 1；否则 OF 清 0。SHLD 指令操作示意图如图 3.15 所示。

2）SHRD

指令格式：

```
SHRD　寄存器/存储器,寄存器,CL/立即数
```

该指令将第一操作数(16 位或 32 位)右移若干位(由 8 位立即数或 CL 指定)，空出位用第二操作数(与第一操作数等位宽)低位部分填补，指令执行后，第二操作数的内容不变，CF 标志位中保留第一操作数最后的移出位。SHRD 指令操作示意图如图 3.16 所示。

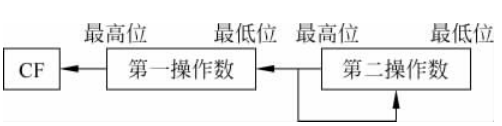

图 3.15　SHLD 指令功能示意图

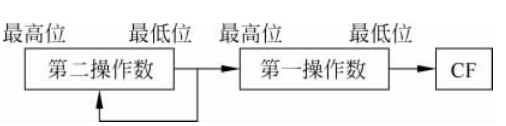

图 3.16　SHRD 指令功能示意图

5. 位操作指令

1）位测试及设置指令

测试指令可用来对指定位进行测试，因而可根据该位的值来控制程序流的执行方向，而置位指令可对指定的位进行设置。指令格式及功能如表3.6所示。

表3.6 位测试与设置指令

指令格式	功能
BT 寄存器/存储器，寄存器/立即数	第一操作数指定要测试的内容(16位或32位)，第二操作数(与第一操作数同长度的通用寄存器或8位立即数)指定要测试的位，将被测内容的指定测试位的值送CF
BTC 寄存器/存储器，寄存器/立即数	在BT指令功能的基础上，将被测试位取反
BTR 寄存器/存储器，寄存器/立即数	在BT指令功能的基础上，将被测试位清0
BTS 寄存器/存储器，寄存器/立即数	在BT指令功能的基础上，将被测试位置1

2）位扫描指令

位扫描指令用于找出寄存器或存储器地址中所存数据的第一个或最后一个是1的位。该指令可用于检查寄存器或存储单元是否为0。BSF和BSR指令格式及功能如表3.7所示。

表3.7 位扫描指令

指令格式	功能
BSF 寄存器，寄存器/存储器	对第二操作数(16位或32位通用寄存器或存储器)从最低位到最高位进行扫描，将首先扫描到“1”的位号送第一操作数，且使ZF=0。若第二操作数的各位均为0，则第一操作数的值不确定，且使ZF=1
BSR 寄存器，寄存器/存储器	与BSF指令功能基本相同，唯一区别的是该指令从最高位到最低位进行扫描

6. 条件设置指令

指令格式：

```
SET条件  寄存器/存储器
```

这是80386特有的指令，用于测试指定的标志位所处的状态，并根据测试结果，将指定的一个8位寄存器或内存单元置1或清0。它们类似于条件转移指令中的标志位测试，但前者根据测试结果将操作数置1或清0，而后者根据测试结果决定转移还是不转移。指令中的条件是指令助记符的一部分，用于指定要测试的标志位，如SETZ或SETNZ等。

3.3.4 80486指令系统新增指令

在80486微处理器中既包含有Cache存储器，又包含有浮点运算器，因此，新增了用于比较和对Cache、TLB进行操作的指令以及比较完整的浮点操作指令。全部指令可分为13类，共计200多条。表3.8列出了80486新增加的指令。

表 3.8　80486 增加的指令

类　别	指 令 格 式	功　能
数据传送类	BSWAP reg32	字节交换
算术运算类	XADD 寄存器/存储器,寄存器	交换与相加
	CMPXCHG 寄存器/存储器,寄存器	比较与交换
Cache 管理类	INVD WBINVD INVLPG	这类指令用于 80486 管理 CPU 内部的 8KB Cache

下面介绍 80486 指令系统中新增的指令。

1. 字节交换指令

指令格式：

```
BSWAP  reg32
```

该指令可以实现双字交换。使 32 位寄存器中的操作数按字节首尾交换，即 $D_{31}\sim D_{24}$ 与 $D_7\sim D_0$ 交换，$D_{23}\sim D_{16}$ 与 $D_{15}\sim D_8$ 交换。

【例 3.33】 [EAX]=01234567H，执行以下指令，并分析结果。

```
BSWAP  EAX
```

结果：[EAX]= 67452301H。

2. 算术运算指令

指令格式：

```
XADD  寄存器/存储器,寄存器
```

该指令将源操作数与目的操作数交换并相加，其中源操作数必须为寄存器，而目的操作数可以是寄存器也可以是内存单元，然后相加，其结果存放在目的操作数中。

【例 3.34】 设[EAX]= 12000001H，[EBX]= 30000002H，执行以下指令，并分析结果。

```
XADD EAX,EBX
```

结果：[EAX]= 42000003H，[EBX]=12000001H。

3. 比较与交换指令

指令格式：

```
CMPXCHG  寄存器/存储器,寄存器
```

该指令使用 3 个操作数：一个寄存器中的源操作数、一个寄存器或内存储器单元的目的操作数和一个隐含(不出现在用户所书写的指令中)的累加器(AL/AX/EAX)。如果目的操作数与累加器的值相等，源操作数送目的单元，否则将目的操作数送累加器。

【例 3.35】 设[EAX]=01010101H，[EBX]=02020202H，[ECX]=03030303H，执行以下指令，并分析结果。

```
CMPXCHG ECX,EBX
```

结果：CF=0，[EAX]=03030303H，[EBX]=02020202H，[ECX]=03030303H。

4. 作废 Cache 指令

指令格式：

```
INVD
```

INVD 指令用于将 Cache 的内容作废。其具体操作是：刷新内部 Cache，并分配一个专用总线周期刷新外部 Cache。执行该指令不会将外部 Cache 中的数据写回主存储器。

5. 写回和作废 Cache 指令

指令格式：

```
WBINVD
```

WBINVD 指令先刷新内部 Cache，并分配一个专用总线周期将外部 Cache 的内容写回主存储器，并在此后的一个总线周期将外部 Cache 刷新。

6. 作废 TLB 项指令

指令格式：

```
INVLPG
```

INVLPG 指令用于使 TLB 中的某一项作废。如果 TLB 中含有一个存储器操作数映像的有效项，则该 TLB 项被标记为无效。

3.3.5 Pentium 指令系统新增指令

与 80486 相比，Pentium 新增加了 3 条处理器专用指令和 5 条系统控制指令，但某些新增的指令是否有效与 Pentium 的型号有关，可利用处理器特征识别指令 CPUID 判别处理器是否支持某些新增指令。表 3.9 列出了 Pentium 增加的指令。由于 Pentium 系列处理器的指令集是向上兼容的，因此所有早期的软件可直接在 Pentium 机器上运行。

表 3.9 Pentium 增加的指令

类别	指令格式	含义	操作

专用指令	CMPXCHG8B 存储器	8 字节比较与交换	IF(EDX：EAX=D)
			ZF=1，D←ECX：EBX
			ELSE
			ZF=0，EDX：EAX←D

	CPUID	CPU 标识	IF(EAX=0H)
			EAX，EBX，ECX，EDX←厂商信息
			IF(EAX=1H)
			EAX，EBX，ECX，EDX←CPU 信息

	RDTSC	读时间标记计数器	EDX；EAX←时间标记计数器
系统控制指令	RDMSR	读模式专用寄存器	EDX；EAX←MSR ECX=0H，MSR 选择 MCA ECX=1H，MSR 选择 MCT
	WRMSR	写模式专用寄存器	MSR←EDX；EAX ECX=0H，MSR 选择 MCA ECX=1H，MSR 选择 MCT
	RSM	恢复系统管理模式	
	MOV CR_4，寄存器 MOV 寄存器，CR_4	与 CR_4 传输	CR4←reg32 reg32←CR_4

下面介绍 Pentium 指令系统中新增指令的功能。

1. 处理器标识指令

指令格式：

```
CPUID
```

使用该指令可以辨别微机中 Pentium 处理器的类型和特点。在执行 CPUID 指令前，EAX 寄存器必须设置为 0 或 1，根据 EAX 中设置值的不同，软件会得到不同的标志信息。

2. 8 字节比较交换指令

指令格式：

```
CMPXCHG8B  存储器
```

该指令带有一个内存储器单元操作数。它能实现将 EDX：EAX 中的 8 字节值与指定的 8 字节存储器操作数相比较，若相等，则使 ZF＝1，且将 ECX：EBX 中的值送指定的 8 字节存储单元替换原有的存储器操作数；否则使 ZF＝0，且将指定的 8 字节存储器操作数送 EDX：EAX。

【例 3.36】 设[EAX]＝01010101H，[EBX]＝02020202H，[ECX]＝03030303H，[EDX]＝04040404H，而 DS：2000H 所指的内存单元开始的 8 个连续单元的内容为 0404040401010101H。

执行以下指令，并分析结果。

```
CMPXCHG8B  [2000H]
```

结果是将 DS：2000H 所指单元开始的 8 个字节和 EDX：EAX 中的 8 个字节比较，由于[EDX：EAX]中为 0404040401010101H，而存储器 DS：[2000H]中开始的 8 个字节也是 0404040401010101H，因此 ZF＝1，并将 ECX：EBX 中的数送给 DS：2000H 开始的 8 个单元，使得 DS：[2000H]开始的 8 个字节为 0303030302020202H。

3. 读时间标记计数器指令

指令格式：

```
RDTSC
```

Pentium 处理器有一个片内 64 位计数器，称为时间标记计数器。计数器的值在每个时钟周期都递增，在 RESET 后该计数器被置 0。执行 RDTSC 指令可以将 Pentium 中的 64 位时间标记计数器的高 32 位送 EDX，低 32 位送 EAX。

一些应用软件需要确定某个事件已经执行了多少个时钟周期，在执行该事件之前和之后分别读出时钟标志计数器的值，计算两次值的差就可得出时钟周期数。

4. 读模式专用寄存器指令

指令格式：

```
RDMSR
```

该指令使软件可访问模式专用寄存器的内容，这两种模式专用寄存器是机器地址检查寄存器(MCA)和机器类型检查寄存器(MCT)。若要访问 MCA，指令执行前需将 ECX 置为 0；而为了访问 MCT，需要将 ECX 置为 1。执行指令时在访问的模式专用寄存器与寄存

器组 EDX：EAX 之间进行 64 位的操作。具体来说，是将 ECX 所指定的模式专用寄存器的内容送 EDX：EAX，高 32 位送 EDX，低 32 位送 EAX。

5. 写模式专用寄存器指令

指令格式：

```
WRMSR
```

该指令将 EDX：EAX 的内容送到由 ECX 指定的模式专用寄存器。具体来说，EDX 和 EAX 的内容分别作为高 32 位和低 32 位。若指定的模式寄存器有未定义或保留的位，则这些位的内容不变。

6. 恢复系统管理模式指令

指令格式：

```
RSM
```

Pentium 处理器有一种称为系统管理模式（SMM）的操作模式，这种模式主要用于执行系统电源管理功能。外部硬件的中断请求使系统进入 SMM 模式，执行 RSM 指令后返回原来的实模式或保护模式。

7. 32 位寄存器与 CR_4 之间的传输指令

指令格式：

```
MOV  CR4,reg32
MOV  reg32,CR4
```

该指令实现将 32 位寄存器的内容传送至控制寄存器 CR_4 或将控制寄存器 CR_4 的内容送到 32 位寄存器中。

思考与练习

1. 什么叫寻址方式？8086/8088 系统中关于存储器操作数的寻址方式有哪几类？80386 及后继处理器支持的新增的存储器寻址方式有哪几种？

2. 指令中数据操作数的种类有哪些？

3. 指出段地址、偏移地址与物理地址之间的关系。有效地址 EA 又是指什么？

4. 在 8086/8088 系统中，能用于寄存器间接寻址及相对寻址的寄存器有哪些？它们通常与哪个段寄存器配合形成物理地址？

5. 80x86 指令系统中新增加的数据传送类指令有哪些？分析它们的功能。

6. 什么是堆栈操作？以下关于堆栈操作的指令执行后，SP 的值是多少？

```
PUSH  AX
PUSH  CX
PUSH  DX
POP   AX
PUSH  BX
POP   CX
POP   DX
```

7. 用汇编语言指令实现以下操作。

(1) 将寄存器 AX、BX 和 DX 的内容相加，和放在寄存器 DX 中。

(2) 用基址变址寻址方式(BX 和 SI)实现 AL 寄存器的内容和存储器单元 BUF 中的一个字节相加的操作，和放到 AL 中。

(3) 用寄存器 BX 实现寄存器相对寻址方式(位移量为 100H)，将 DX 的内容和存储单元中的一个字相加，和放到存储单元中。

(4) 用直接寻址方式(地址为 0500H)实现将存储器中的一个字与立即数 3ABCH 相加，和放回该存储单元中。

(5) 用串操作指令实现将内存定义好的两个字节串 BUF1 和 BUF2 相加后，存放到另一个串 BUF3 中的功能。

8. 指出下列指令中，源操作数及目的操作数的寻址方式。

(1) SUB　BX,[BP+35]

(2) MOV　AX,2030H

(3) SCASB

(4) IN　AL,40H

(5) MOV　[DI+BX],AX

(6) ADD　AX,50H[DI]

(7) MOV　AL,[1300H]

(8) MUL　BL

9. 已知(DS)=1000H,(SI)=0200H,(BX)=0100H,(10100H)=11H,(10101H)=22H,(10600H)=33H,(10601H)=44H,(10300H)=55H,(10301H)=66H,(10302H)=77H,(10303H)=88H，试分析下列各条指令执行完后 AX 寄存器的内容。

(1) MOV　AX,2500H

(2) MOV　AX,500H[BX]

(3) MOV　AX,[300H]

(4) MOV　AX,[BX]

(5) MOV　AX,[BX][SI]

(6) MOV　AX,[BX+SI+2]

10. 判断下列指令是否有错，如果有错，说明理由。

(1) SUB　BL,BX

(2) MOV　BYTE PTR[BX],3456H

(3) SHL　AX,CH

(4) MOV　AH,[SI][DI]

(5) SHR　AX,4

(6) MOV　CS,BX

(7) MOV　125,CL

(8) MOV　AX,BYTE PTR[SI]

(9) MOV　[DI],[SI]

11. 设(DS)=1000H,(ES)=2000H,(SS)=3000H,(SI)=0080H,(BX)=02D0H,

(BP)＝0060H，试指出下列指令的源操作数字段是什么寻址方式？它的物理地址是多少？

(1) MOV AX,0CBH

(2) MOV AX,[100H]

(3) MOV AX,[BX]

(4) MOV AX,[BP]

(5) MOV AX,[BP＋50]

(6) MOV AX,[BX][SI]

12. 分别说明下列每组指令中的两条指令的区别。

(1) AND CL,0FH	OR CL,0FH
(2) MOV AX,BX	MOV AX,[BX]
(3) SUB BX,CX	CMP BX,CX
(4) AND AL,01H	TEST AL,01H
(5) JMP NEAR PTR NEXT	JMP SHORT NEXT
(6) ROL AX,CL	RCL AX,CL
(7) PUSH AX	POP AX

13. 试分析以下程序段执行完后 BX 的内容为何？

```
MOV  BX,1030H
MOV  CL,3
SHL  BX,CL
DEC  BX
```

14. 写出下列指令序列中每条指令的执行结果，并在 DEBUG 环境下验证，注意各标志位的变化情况。

```
MOV  BX,126BH
ADD  BL,02AH
MOV  AX,2EA5H
ADD  BH,AL
SBB  BX,AX
ADC  AX,26H
SUB  BH,－8
```

15. 编写能实现以下功能的程序段。

根据 CL 中的内容决定程序的走向，设所有的转移都是短程转移。若 D_0 位等于 1，其他位为 0，转向 LAB1；若 D_1 位等于 1，其他位为 0，转向 LAB2；若 D_2 位等于 1，其他位为 0，转向 LAB3；若 D_0、D_1、D_2 位都是 1，则顺序执行。