第 3 章　80x86 指令系统和寻址方式

指令是计算机能接受的软件工作者命令的最小工作单元，它最终是由计算机内的电器元件状态来体现的，这一状态为译码器所识别，并经一定时间周期付诸实施，这种指令称为机器指令。一条机器指令应包含两部分内容：一是要给出此指令要完成何种操作，这部分称为指令操作码；二是要给出参与操作的对象是什么，此部分称为操作数，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，然后对存放在有效地址中的操作数进行存取操作。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。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。

本章首先介绍 16 位微处理器 8086/8088 的常用指令和寻址方式，在此基础上以 80386 为对象讲述 32 位微处理器的指令系统和寻址方式。对 80x86 的控制转移类指令、串操作与重复前缀指令、子程序调用与返回指令以及中断调用与返回指令的讲解将穿插在第 4 章的汇编语言程序设计以及第 8 章的中断和 DMA 技术中介绍，其目的就是为了使学习者更好地理解和应用这些指令。

本章的主要内容如下：

- 8086 指令系统概述。
- 8086 的寻址方式和常用指令。
- 80386 的寻址方式和指令系统。
- 80486/Pentium 微处理器新增指令。

3.1　8086 指令系统概述

3.1.1　数据类型

计算机中的数是用二进制表示的，数的符号也是用二进制表示的。把一个数连同其符号在机器中的表示加以数值化称为机器数，实际上机器数是将符号“数字化”所得到的数。计算机执行指令过程中需要处理各种类型的机器数，可处理的数据类型有 7 种。

1. 无符号二进制数

无符号二进制数是指不带符号位的一个字节、字或双字的二进制数。

2. 带符号二进制数

带符号二进制数有正、负之分,均以补码形式表示。补码由符号位加数值两部分组成。带符号数的最高位表示符号位,最高位为0表示为正,最高位为1表示为负。正数的补码和原码相同,负数的补码是符号位保持不变(即为1),其余各位按位取反,再在最末位加1。

3. BCD码

BCD码是用二进制数的形式表示的十进制数。计算机中BCD码有两种,分别是非压缩BCD码和压缩BCD码两种。前者用8位二进制数表示一位十进制数,那么一个字节能表示的十进制数的范围是0～9;后者用4位二进制数表示一位十进制数,那么一个字节能表示的十进制数的范围是0～99。例如,17的非压缩BCD码是0000000100000111B,即0107H;压缩BCD码是00010111B,即17H。

4. 定点数和浮点数

在计算机中,针对小数点的处理有两种表示方法,分别是定点数表示法和浮点数表示法。

1) 定点数表示法

定点数表示法就是小数点固定在某个位置上。在定点计算机中,一般将小数点定在最高位(即纯小数)或将小数点定在最低位(即纯整数)。

2) 浮点数表示法

浮点表示法就是小数点的位置不固定。浮点数在计算机中通常的表示形式为"2的正/负阶码次方×尾数",其中阶码是一个正整数,尾数是一个小数,尾数的区间为[0.5,1],如果尾数不在此区间,则要进行规格化。规格化通过尾数左右移、阶码加减完成,其规则是:尾数左移一次,阶码减1,尾数右移一次,阶码加1。

5. 串数据

计算机可以处理的串数据有以下几种:

- 位串,一串连续的二进制数。
- 字节串,一串连续的字节。
- 字串,一串连续的字。
- 双字串,一串连续的双字。

6. ASCII码数据

ASCII(American Standard Code for Information Interchange,美国信息交换标准码)是一种字符数据的常用表示方法。包括ASCII码字符、ASCII码数和ASCII控制符等。

7. 指针类数据

指针类数据包括近程指针和远程指针。前者为16位的偏移量,用于段内寻址、段内数据访问或转移;后者由16位段值和16位的偏移量组成,用于段间寻址、段间数据访问

或段间转移。

另外，在 8086/8088 处理机中，为了达到数据结构的最大灵活性和最有效地使用内存，字数据不必定位于偶数地址，即允许不按 2 的倍数边界对齐。但是，对齐的字数据可以一次传送，而未对齐的字数据则需要两次传送。为了获得最佳性能，一般将字操作对齐于偶地址。

3.1.2 80x86 指令的基本组成

80x86 的汇编指令由操作码和操作数两部分组成。操作码是指令的操作命令，操作数是指令的操作对象。汇编指令最后要翻译成机器指令。

1. 指令的操作码

80x86 的机器指令的操作码(OP)采用二进制代码表示本指令所执行的操作。在大多数指令的操作码中，常用某位指示某些具体操作信息。图 3.1 所示的 8086 操作码含有 3 个特征位，分别为 W 位、D 位和 S 位。它们的含义如下：

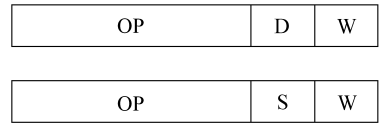


图 3.1 8086 操作码格式

(1) W 位是字操作标志位。当 W＝1 时，表示当前指令进行字操作；当 W＝0 时，表示当前指令进行字节操作。

(2) D 位是对目标操作数进行寄存器寻址的标志。对于双操作数指令(立即数指令和串操作指令除外)，其中一个操作数必定由寄存器指出。这时，寄存器名通过后面的 REG 域指出，此外，要用 D 位指出寄存器所寻址的是源操作数还是目标操作数。如 D＝1，则寄存器所寻址的是目标操作数；如 D＝0，则寄存器所寻址的是源操作数。

(3) S 位是符号扩展位。一个 8 位补码可以扩展为 16 位补码。使所有高位等于低位字节的最高有效位(即符号位)，这种扩展叫符号扩展，在加法指令、减法指令和比较指令中，S 位和 W 位结合起来指出各种情况。例如，S＝0，W＝0 时，为 8 位操作数；S＝0，W＝1 时，为 16 位操作数。

2. 机器指令格式

80x86 的机器指令为 1～6B。一般用指令的第一个字节或者头两个字节表示指令的操作码和寻址方式，通常称为操作码域。操作码指出了执行这条指令时 CPU 要做什么操作，寻址方式则表示执行指令时所用的操作数的来源。图 3.2 给出了 8086 指令格式。

紧随在操作码域后的字节一般称操作数域。至于操作数域属于图 3.2 中的哪一种情况，要由指令前半部分的操作码和寻址方式确定。这里有几点需要指出：

(1) 一条指令可以包含一个操作数，也可以包含一个以上的操作数。具有一个操作数的指令称为单操作数指令，单操作数指令中的操作数可能由指令本身提供，也可能由指令隐含地指出。

(2) 若位移量或立即数为 16 位，那么在指令代码中，将低位字节放在前面，高位字节

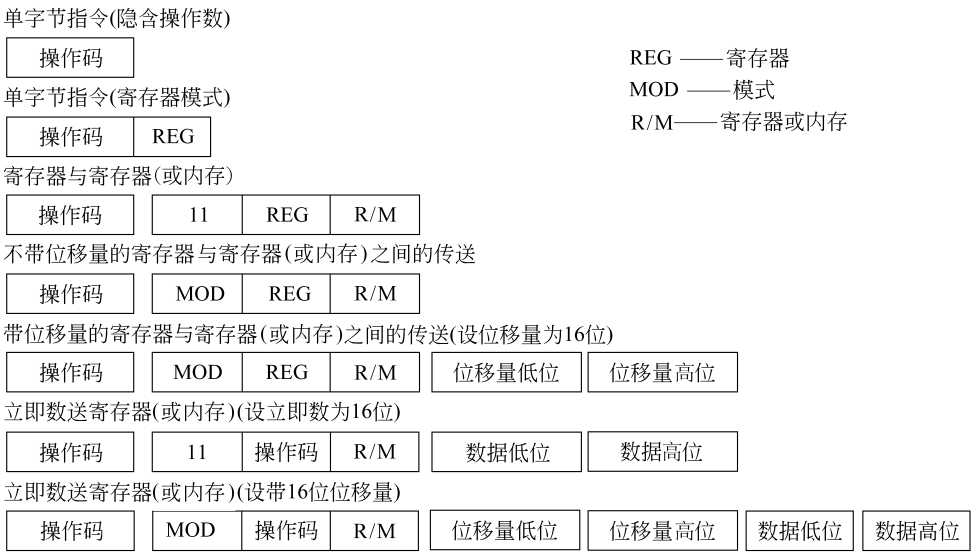


图 3.2　8086 指令格式

放在后面。

(3) 80x86 指令系统中大多数指令的操作码安排在第一个字节,但有几条指令是特殊的,其指令中的第一个字节不但包含操作码成分,而且还隐含地指出了寄存器名,从而整个指令只占一个字节,成为单字节指令。这些指令字节数最少,执行速度最快,用得也最频繁。

3. 指令的执行时间

指令执行时间取决于时钟周期长短和执行指令所需要的时钟周期数。如果涉及内存操作,那么执行一条指令的时间为基本执行时间加上计算有效地址所需要的时间。

8086 的总线部件和执行部件是并行工作的,因此在计算指令的基本执行时间时都假定要执行的指令已经预先放在指令队列中,也就是说,没有计算取指时间。有些指令在执行过程中要多次访问内存,因此,访问内存的次数也是考虑指令执行总时间的重要因素。

可见,执行一条指令所需的总时间为基本执行时间、计算有效地址的时间和为了读取操作数和存放操作结果需访问内存的时间之和。

3.2　8086 / 8088 的寻址方式和指令系统

3.2.1　8086/8088 的寻址方式

在计算机的存储器系统中存储着两类信息,一是数据,二是程序。这里所说的程序就是指令序列。8086/8088 的寻址方式包括程序寻址方式和数据寻址方式。数据寻址方式是指获取指令所需的操作数或操作数地址的方式。程序的寻址方式是指在程序中出现转移和调用时的程序定位方式。

1. 数据寻址方式

数据寻址方式就是寻找参加运算的操作数的方式。80x86 指令中所需的操作数来自以下 3 个方面。

(1) 操作数包含在指令中。在取指令的同时,操作数也随之得到,这种操作数称为立即数。

(2) 操作数包含在 CPU 的某个内部寄存器中,这种操作数称为寄存器操作数。

(3) 操作数包含在存储器中,这种操作数称为存储器操作数。

存储器操作数存放在内存单元中,在 80x86 微机系统中,任何内存单元的地址均由段基址和偏移地址(又称偏移量)组成,段基址由段寄存器提供,而偏移地址由以下 4 个基本部分组成。

(1) 基址,8086/8088 系统的基址寄存器为 BX 和 BP。

(2) 变址,8086/8088 系统的变址寄存器为 SI 和 DI。

(3) 比例因子,采用 1、2、4 或 8 几种不同的比例因子。8086/8088 系统中比例因子为 1。

(4) 位移量,即相对于某个存储单元的偏移量。

这 4 个部分称为偏移地址的四元素。一般将这四元素按某种计算方法组合形成的偏移地址称为有效地址(Effective Address,EA)。它们的组合和计算方法为

有效地址=基址+变址×比例因子+位移量

根据 80x86 系统的存储器组织方式,指令中不能使用实际地址(即物理地址),而只使用逻辑地址或称为操作数的线性地址,因此在求得有效地址后,可由有效地址和段首址形成逻辑地址。这 4 种元素可优化组合出 9 种存储器寻址方式。图 3.3 给出这 4 种元素的优化组合的寻址计算方法。

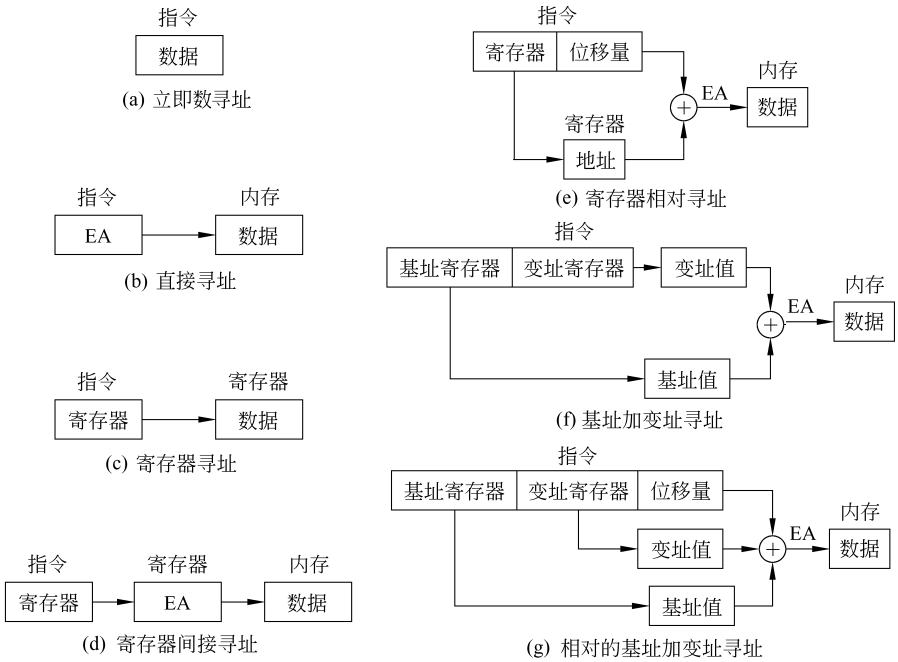


图 3.3 8086/8088 的操作数寻址方式

1) 立即寻址,操作数为立即数

8086/8088指令系统中,有一部分指令所用的8位或16位操作数由指令机器码提供,这种方式称为立即寻址方式,即操作数直接包含在指令机器码中,它紧跟在操作码的后面,与操作码一起放在代码区域中。

例 3.1

```
MOV  AX,0A7FH              ;AX←0A7FH,执行后,AH=0AH,AL=7FH
MOV  AL,5H                 ;AX←5H
```

立即寻址方式主要用于给寄存器或存储单元赋初值。因为操作数可以从指令中直接取得,不需要运行总线周期,所以立即数寻址方式的显著特点是执行速度快。

2) 寄存器寻址,操作数为寄存器操作数

寄存器寻址是指操作数包含在CPU内部的寄存器中。对于16位操作数,寄存器可以是AX、BX、CX、DX、SI、DI、SP、BP,而对于8位操作数,寄存器可以为AH、AL、BH、BL、CH、CL、DH、DL。寄存器寻址的指令本身存放在存储器的代码段,而操作数则在CPU寄存器中。由于指令执行过程中不用访问存储器,因此执行速度很快。

寄存器寻址可以进行16位操作数操作,也可进行8位操作数操作。

例 3.2

```
MOV  AX,BX
MOV  AH,AL
```

3) 存储器寻址,操作数为存储器操作数

(1) 直接寻址。

直接寻址方式是在指令的操作码后面直接给出操作数的16位偏移地址,这个偏移地址也称有效地址(EA)。该有效地址与指令的操作码一起存放在内存的代码段,也是低8位在前,高8位在后。操作数隐含数据段操作,存放在内存的数据段(DS)区域中,在给定的16位有效地址的地方。

例 3.3

```
MOV  AX,[2000H]
```

直接寻址指令中,表示有效地址的16位数必须加上方括号。指令的功能不是将立即数2000H传送到累加器AX中,而是将一个内存单元的内容传送到AX中,该内存单元地址是2000H。

假设该数据段寄存器DS=3000H,则内存单元的物理地址是DS的内容左移4位后再加上指令中的16位有效地址,形成有效的访问内存的物理地址,即

$$3000H \times 10H + 2000H = 32000H$$

直接寻址一般多用于存取某个存储单元中的操作数,例如从一个存储单元取操作数,或者将一个操作数存入某个存储单元。

(2) 寄存器间接寻址。

寄存器间接寻址方式是指令中的操作数存放在存储器中,存储单元的有效地址由寄

存器指出，这些寄存器可以是 BX、BP、SI、DI 之一，即有效地址 EA＝基址寄存器的内容或变址寄存器的内容，对寄存器指向的存储单元进行数据操作。这些用来存放存储器操作数偏移地址的寄存器称为地址指针。

下面分别以 BX、SI、DI 进行寄存器间接寻址（BX、SI、DI 作为地址指针）的方式与以 BP 进行寄存器间接寻址（BP 作为地址指针）的方式为例进行说明。在以 BX、SI、DI 进行寄存器间接寻址时，隐含的数据段寄存器为 DS；而在以 BP 进行寄存器间接寻址时，隐含的数据段寄存器为 SS。

例 3.4

```
MOV  AX,[BX]                    ;物理地址=DS×16+BX
MOV  BX,[SI]                    ;物理地址=DS×16+SI
MOV  [DI],DX                    ;物理地址=DS×16+DI
```

例 3.5

```
MOV  [BP],BX                    ;物理地址=SS×16+BP
```

无论用 BX、SI、DI 或者 BP 作为间接寄存器，都允许段超越，即也可以使用上面所提到的约定情况以外的其他段寄存器。

例 3.6

```
MOV  AX,ES:[BX]                 ;物理地址=ES×16+BX
MOV  DS:[BP],DX                 ;物理地址=DS×16+BP
```

（3）相对寄存器寻址。

相对寄存器寻址的有效地址 EA＝基址寄存器的内容或变址寄存器的内容±16 位或 8 位位移量。以 BX、SI、DI 进行寄存器间接寻址（BX、SI、DI 作为地址指针）的方式，隐含的段寄存器为数据段寄存器 DS；而以 BP 进行寄存器间接寻址（BP 作为地址指针）的方式，隐含的段寄存器为堆栈段寄存器 SS。

例 3.7

```
MOV  AX,3003H[SI]
```

假设 DS＝3000H，SI＝2000H，指令中的 3003H 即为位移量 DISP。指令操作的存储器物理地址＝3000H×10H＋EA＝30000H＋2000H＋3003H＝35003H。

例 3.8

```
MOV  SI,08H[BX]                 ;物理地址= DS×16+BX+08H
MOV  AX,[BX+100H]               ;物理地址= DS×16+BX+100H
MOV  AL,[BP+08H]                ;物理地址= SS×16+BP+08H
MOV  0200H[BP],AX               ;物理地址= SS×16+BP+0200H
```

（4）基址、变址寻址。

将基址寻址方式和变址寻址方式联合起来的寻址方式称为基址、变址寻址方式。有效地址 EA＝基址寄存器的内容＋变址寄存器的内容。

例 3.9

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

假设 BX=1500H,SI=2000H,DS=8000H。

物理地址=DS×10H+EA=8000H×10H+1500H+2000H=83500H。

(5) 相对的基址、变址寻址。

这种寻址方式是在基址变址寻址方式的基础上再加上或减去 16 位或 8 位位移量。有效地址 EA=基址寄存器的内容+变址寄存器的内容±16 位或 8 位位移。

例 3.10

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

假设 MASK=64H,BX=A500H,SI=2200H,DS=6000H。

物理地址=DS×10H+EA=6000H×10H+A500H+2200H+64H=6C764H。

2. 程序寻址方式

程序寻址方式就是寻找程序转移地址的方式。程序在存储器中的寻址方式分为段内转移和段间转移。段内转移指转移地址与转移指令地址在同一段中,段间转移指目标地址与转移指令地址不在同一个段中。

1) 直接寻址方式

段内直接寻址方式也称为相对寻址方式,用指令本身提供的位移量修改指令指针寄存器,形成有效目标地址的寻址方式。

例 3.11

```
JMP   1000H                    ;转移地址在指令中给出
CALL  1000H                    ;调用地址在指令中给出
```

2) 段内间接寻址方式

程序转移的地址存放在寄存器或存储单元中,这个寄存器或存储单元内容可用以上所述寄存器寻址或存储器寻址方式取得,所得到的转移有效地址用来更新 IP 的内容。由于此寻址方式仅修改 IP 的内容,所以只能在段内进行程序转移。

例 3.12

```
JMP   BX                       ;转移地址由 BX 给出
CALL  AX                       ;调用地址由 AX 给出
JMP   WORD PTR [BP+TABLE]      ;转移地址由 BP+TABLE 所指的存储单元给出
```

3) 段间直接寻址方式

这种寻址方式是在指令中直接给出 16 位的段基值和 16 位的偏移地址,用来更新当前的 CS 和 IP 的内容。

例 3.13

```
JMP   2500H:3600H              ;转移的段地址和偏移地址在指令中给出
```

```
CALL  2600H:3800H                ;调用程序的段地址和偏移地址在指令中给出
```

4）段间间接寻址方式

这种寻址方式是寻找指令中给出的存储器数据的地址，包括存放转移地址的偏移量和段地址。其低位字地址单元存放的是偏移地址，高位字地址单元中存放的是转移段基值。这样既更新了 IP 内容又更新了 CS 的内容，故称为段间间接寻址。

例 3.14

```
JMP  WORD PTR[BX]                ;转移到当前代码位置内
                                 ;有效地址存放在 BX 寻址的单元中
```

3. 段寄存器的确定

为了简化指令系统，8086/8088 的指令在形式上只给出了地址偏移值（有效地址），但隐含着对某段寄存器的操作，表 3.1 给出的默认规定指出了所选用的隐含段寄存器。

表 3.1 段寄存器选择规定

访 存 类 型	默认段寄存器	段超越前缀的可用性
代码	CS	不可用
PUSH、POP 类代码	SS	不可用
串操作的目标地址	ES	不可用
以 BP、SP 间址的指令	SS	可用 CS、DS、ES
其他	DS	可用 CS、SS、ES

这种隐含选择规定可以被段超越运算符“:”改变。当在一条指令前面加上段超越运算符时，则由段超越运算符所指定的段寄存器取代默认的段寄存器。

例 3.15

```
MOV  BX,ES:[DI]                  ;源操作数在 ES 指定的段中
```

3.2.2 8086/8088 的常用指令

8086/8088 的指令分为数据传送指令、算术运算指令、逻辑运算和移位指令、控制转移指令、串操作指令、程序控制指令和处理器控制指令等。本节主要介绍 8086/8088 常用的指令，对于控制转移指令、串操作指令、程序控制指令、I/O 操作指令以及中断指令等将在后续章节中陆续讲解。

1. 数据传送指令

数据传送指令用于实现 CPU 的内部寄存器之间、CPU 内部寄存器和存储器之间、CPU 累加器 AX 或 AL 和 I/O 端口之间的数据传送，此类指令除了 SAHF 和 POPF 指令外均不影响标志寄存器的内容。需要注意的是，在数据传送指令中，源操作数和目标操作

数的数据长度必须一致。

1) MOV 指令

格式:

```
MOV  dest,src                    ;dest←src
```

这里,dest 为目标操作数,src 为源操作数,以下类同。

功能:MOV 指令用于将一个操作数从存储器传送到寄存器,或从寄存器传送到存储器,或从寄存器传送到寄存器,也可以将一个立即数存入寄存器或存储单元,但不能用于存储器与存储器之间以及段寄存器之间的数据传送。MOV 指令传送关系如图 3.4 所示。

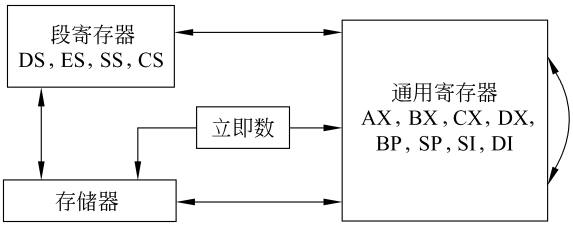


图 3.4 MOV 指令的传送关系

MOV 指令有 6 种数据传输格式。

(1) CPU 的通用寄存器之间的数据传输。

例 3.16

```
MOV  AL,BL                       ;BL 寄存器的 8 位数送到 AL 寄存器
MOV  SI,BX                       ;BX 寄存器的 16 位数送到 SI 寄存器
```

(2) 立即数(常数)到存储单元的数据传输。

例 3.17

```
MOV  MEM_BYTE,20H                ;将立即数 20H 送到 MEM_BYTE 存储单元
MOV  DS:[0005H],4500H            ;将立即数 4500H 送到 DS:0005H 所指的两个
                                 ;相邻存储单元中
```

(3) 立即数到通用寄存器的数据传输。

例 3.18

```
MOV  AL,20H                      ;将立即数 20H 送到 AL 寄存器
MOV  SP,2000H                    ;将立即数 2000H 送到 SP 寄存器
```

(4) 通用寄存器和存储单元之间的数据传输。

例 3.19

```
MOV  AL,DS:[1000H]               ;将地址 DS:1000H 存储单元的内容送到 AL
MOV  ES:[0002H],BX               ;将 BX 中的 16 位数据传输到地址 ES:0002H
                                 ;所指的两个相邻存储单元中
```

(5) 段寄存器和存储单元之间的数据传输。

例 3.20

```
MOV  ES,[BX]              ;BX 寄存器所指内容传送到 ES
MOV  [1000H],CS           ;将 CS 内容传送到地址为 1000H 所指的两个相邻存储器单元
```

(6) 通用寄存器和段寄存器之间的数据传输。

例 3.21

```
MOV  AX,ES                ;将 ES 段地址传送到 DS 寄存器
MOV  DS,AX
```

使用 MOV 指令时,必须注意以下几点:

- MOV 指令可以传 8 位数据,也可以传 16 位数据,这取决于寄存器是 8 位还是 16 位,或立即数是 8 位还是 16 位。
- MOV 指令中的目标操作数和源操作数不能都是存储器操作数,即不允许用 MOV 实现两个存储单元间的数据传输。
- 不能用 CS 和 IP 作为目标操作数,即这两个寄存器的内容不能随意改变。
- 不允许在段寄存器之间传输数据,例如 MOV DS,ES 是错误的。
- 不允许用立即数作为目标操作数。
- 不能向段寄存器传送立即数。如果需要对段寄存器赋值,可以通过 CPU 的通用寄存器 AX 完成。

例 3.22

```
MOV  AX,1000H             ;将数据段首地址 1000H 通过 AX 传入 DS 寄存器
MOV  DS,AX
```

2) 交换指令 XCHG

格式:

```
XCHG  dest,src            ;dest←→src
```

功能:XCHG 指令用于交换两个操作数。这条指令实际上起到了 3 条 MOV 指令的作用,指令中的两个操作数可以是两个寄存器操作数或一个寄存器与一个存储器操作数。

交换指令可实现通用寄存器之间、通用寄存器与存储单元之间的数据(字节或字)交换。

例 3.23

```
XCHG  AL,BL               ;AL 与 BL 寄存器的内容进行交换
XCHG  BX,CX               ;BX 与 CX 寄存器的内容进行交换
XCHG  DS: [2200H],DX      ;DL 寄存器与地址 DS:2200H 的内容进行交换
                          ;DH 寄存器与地址 DS:2201H 的内容进行交换
```

使用交换指令时应注意以下两点:

- 两个操作数不能同时为存储器操作数。
- 任一个操作数都不能使用段寄存器,也不能使用立即数。

3) 取有效地址指令 LEA

格式:

```
LEA  r16,mem
```

要求:r16 为一个 16 位通用寄存器,mem 为存储单元。

功能:LEA 指令的作用是将有效地址(在这里指地址偏移值)送通用寄存器,而不是将存储单元的内容送通用寄存器。

例 3.24

```
LEA  BX,DS:1000H          ;将地址 1000H 送到 BX,即(BX)=1000H
LEA  BX,BUFFER            ;将存储变量 BUFFER 的变量地址传送到 BX
LEA  BX,[2728]            ;将 2728 偏移地址送 BX
LEA  SP,[BP][DI]          ;将[BP+DI]寻址方式的偏移地址送 SP
```

在很多情况下,LEA 指令可以用相应的 MOV 指令代替。

例 3.25

```
LEA  BX,VARWORD
MOV  BX,OFFSET VARWORD
```

这两条指令的执行效果是完全一样的。区别在于后者用伪指令 OFFSET,由编译程序在编译时赋值;而前者在执行时赋值。

注意:

- 两个操作数不能同时为存储器操作数。
- 任一个操作数都不能使用段寄存器,也不能使用立即数。

4) 装入地址指令

格式:

```
LDS dest,src              ;将内存中连续 4B 内容送到 DS 和指定的通用寄存器
LES dest,src              ;将内存中连续 4B 内容送到 ES 和指定的通用寄存器
```

功能:LDS 指令把 src 操作数所指的内存中连续 4B 单元内容的低 16 位数据存入 dest 指定的通用寄存器中,高 16 位存入 DS 中;LES 指令把 src 操作数所指的内存中连续 4B 单元内容的低 16 位数据存入 dest 指定的通用寄存器中,高 16 位存入 ES 中。

一个内存单元的逻辑地址用段基值和偏移量两个部分来描述,每个部分均为 16 位二进制数,即 2B。因而一个内存单元的逻辑地址需要 4B 描述,这 4B 存放内存中,顺序为:低 2B 为偏移量,高 2B 为段基值。因此,本指令可以看成是把内存中连续 4 个单元内存放的地址取出来,存入 DS(或 ES)和指定的通用寄存器中。

注意:dest 必须是通用寄存器之一,src 必须是内存操作数。

例 3.26

```
LDS  DI,[2130H]           ;将 2130H 和 2131H 单元的内容送 DI
                          ;将 2132H 和 2133H 单元的内容送 DS
```

本例的操作如图 3.5 所示。

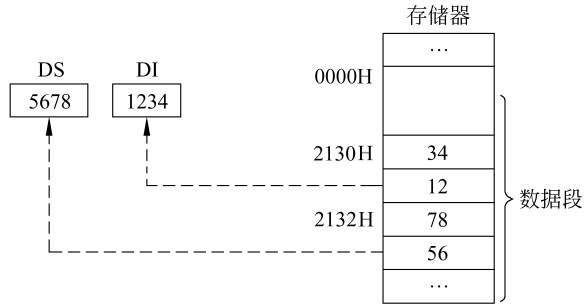


图 3.5 LDS 指令示意图

例 3.27

```
EXDWORD DD 12345678H
LDS  SI,EXDWORD              ;指令执行后,将 1234H 送 DS,5678H 送 SI
```

5) 标志传送指令

格式：

```
LAHF
SAHF
```

功能：标志传送指令有取标志指令 LAHF 和存标志指令 SAHF，LAHF 指令用于将标志寄存器的低 8 位送入 AH，SAHF 指令用于将 AH 的内容送入标志寄存器的低 8 位。

6) 表转换指令 XLAT

格式：

```
XLAT label
XLAT                         ;AL←DS:((BX)+(AL))
```

功能：XLAT 指令以转换表中的一个字节来替换 AL 寄存器中的内容，可用于码的转换。其中 DS:BX 指向转换表的首址，转换前 AL 内容为序号，转换后 AL 内容为对应的码。例如，将 ASCII 码转换为 EBCDIC 码，也可用于代码加密。

例 3.28 将 AL 中的一个数字字符的 ASCII 码，经变换加密后从 42H 端口输出。表 3.2 给出了密码转换表。

表 3.2 密码转换表

明文码	加密码	明文码	加密码
0	'5'	5	'6'
1	'7'	6	'8'
2	'9'	7	'0'
3	'1'	8	'2'
4	'3'	9	'4'

有关程序段如下：

```
SUB         AL,'0'              ;将 AL 中 ASCII 字符转换成表 XMIT_TABLE 中的序号
LDS         BX,TAB_POINT        ;将表头地址指针送 DS:BX
XLAT        XMIT_TABLE          ;查转换表,将对应的加密码放入 AL
OUT         42H,AL              ;从 42H 端口输出加密后的 ASCII 码
TAB_POINT   DD XMIT_TABLE
XMIT_TABLE  DB '5791368024'
```

表 3.2 是程序中所用的转换表，若原 AL 的内容为字符'3'，则执行此段程序后，AL 中的内容为字符'1'。

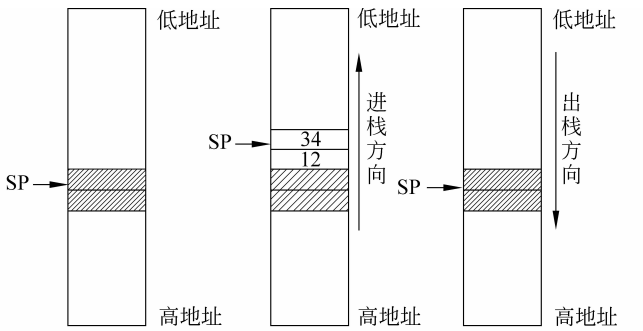
7) 堆栈操作指令

堆栈的特性在第 2 章已经详细介绍过。堆栈是向下生成的，也就是说，压栈操作是先将 SP 减 2，再将数据压入 SS:SP 指向的单元；弹出操作则先将 SS:SP 指向的数据弹出，再将 SP 加 2。

格式：

```
PUSH  src
POP   dest
```

功能：PUSH 指令用于压入存储器操作数，寄存器操作数或立即数。注意，src 和 dest 为 16 位的寄存器或存储单元，图 3.6 给出了进栈和出栈操作示意图。



(a) 进栈前堆栈段SF　(b) 进栈后堆栈段TF　(c) 出栈后堆栈段IF

图 3.6　进栈和出栈操作示意图

例 3.29

```
PUSH  AX                  ;将 AX 的内容压入堆栈
PUSH  [2000H]             ;将 DS 段逻辑地址为 2000H 的内容压入堆栈
```

POP 指令与 PUSH 指令相反，将栈顶的数据弹出到通用寄存器或存储器中。

例 3.30

```
POP  AX                   ;将栈顶的内容弹出到 AX 寄存器
POP  M_ADD                ;将栈顶的内容弹出到 M_ADD 指定的存储器中
```

注意：

- PUSH 和 POP 指令只允许按字访问堆栈，即两类指令的操作数必须是 16 位寄存器或存储单元的操作数。
- CS 不能作为目标操作数。

8）符号扩展指令

（1）字节扩展指令。

格式：

```
CBW
```

功能：将 AL 中的单字节数的符号扩展到 AH 中。若 AX＜80H，则 0→AH；若 AL≥80H，则 FFH→AH。

例 3.31

```
MOV  AL,6FH              ;AL=01101111B
CBW                      ;AH=00000000B,AL 内容不变
MOV  AL,0AFH             ;AL=10101111B
CBW                      ;AH=11111111B,AL 内容不变
```

（2）字扩展双字指令。

格式：

```
CWD
```

功能：将 AX 中的单字节数的符号扩展到 DX 中。若 AX＜8000H，则 0→DX；若 AL≥8000H，则 FFFFH→DX。

例 3.32

```
MOV  AX,4F0AH            ;AX=4F0AH
CWD                      ;DX=0000H,AX 内容不变
MOV  AX,0EF0AH           ;AX=0EF0AH;
CWD                      ;DX=0FFFFH,AX 内容不变
```

注意：这两条符号扩展指令在带符号数的乘法和除法运算中十分有用，对标志位没有影响。

2. 算术运算指令

8086 的算术运算指令包括加、减、乘、除 4 种基本运算指令，以及为适应进行 BCD 码十进制运算而设置的各种校正指令，共 20 条。在基本算术运算指令中，除加 1、减 1 指令外，均为双操作数指令。双操作数指令的两个操作数必须有一个在寄存器中，双操作数指令的目标操作数不允许使用立即数，单操作数指令也不允许使用立即数。

算术运算指令涉及的操作数从数据形式来讲有两种，即 8 位的操作数和 16 位的操作数，这些操作数可分为两类，即无符号数和带符号数。无符号数把所有的数位都当成数值位，因此 8 位无符号数表示的范围为 0～255（或 0～0FFH）；16 位无符号数表示的范围为

0～65 535(或0～0FFFFH)。带符号数的最高位作为符号位：0表示正号，1表示负号。计算机中的带符号数通常用补码表示，这样，8位带符号数表示的范围－128～＋127(或80H～7FH)；16位带符号数表示的范围为－32 768～＋32 767(8000H～7FFFH)。

1) 加法类指令

(1) 不带进位的加法指令。

格式：

```
ADD  dest,src            ;dest←(dest)+(src)
```

功能：ADD指令用来对源操作数和目标操作数字节或字相加，结果放在目标操作数中。

说明：

- ADD指令不允许两个存储器单元内容相加，两个操作数不能同时为存储器操作数。
- ADD指令也不允许在两个段寄存器之间相加。
- 对标志位有影响，主要是CF、ZF、OF、SF标志位。

例 3.33

```
ADD  AX,BX               ;AX←(AX)+(BX)
```

例 3.34

```
ADD  AX,0F0F0H
```

设指令执行前(AX)＝5463H，执行后，将得到结果(AX)＝4553H，且CF＝1，ZF＝0，SF＝0，OF＝0。

```
          (AX)＝0101    0100    0110    0011
    ＋           1111    0000    1111    0000
    ------------------------------------------
          [1] ←  0100    0101    0101    0011→AX
          CF
```

(2) 带进位标志的加法指令。

格式：

```
ADC dest,src                ;dest←(dest)+(src)+(CF)
```

功能：ADC指令和ADD指令的功能类似，区别在于ADC在完成两个字或两个字节数相加的同时，还要考虑进位标志CF的值，若进位位CF为1，则将结果加1。

注意：src(源操作数)和dest(目标操作数)不能同时为存储单元。段寄存器不能进行算术运算。

例 3.35

```
ADC  AX,BX                  ;AX←(AX)+(BX)+(CF)
ADC  AL,4                   ;AL←(AL)+4+(CF)
ADC  EXTMEM,AX              ;EXTMEM←(EXTMEM)+(AX)+(CF)
ADC  EXTMEM,23              ;EXTMEM←(EXTMEM)+23+(CF)
```

例 3.36 有两个 4 字节数分别放在 FIRST 和 SECOND 开始的存储区，低字节在低地址处，编写程序将两数相加，并将结果存入 FIRST 开始的存储区。

```
MOV  AX,FIRST                ;第一个数的低 16 位→AX
ADD  AX,SECOND               ;两数的低 16 位相加→AX
MOV  FIRST,AX                ;低 16 位相加结果存入 FIRST 及 FIRST+1 单元
MOV  AX,FIRST+2              ;第一个数的高 16 位→AX
ADC  AX,SECOND+2             ;两数的高 16 位连同低 16 位进位相加→AX
MOV  FIRST+2,AX              ;高 16 位相加的结果存入 FIRST+2 及 FIRST+3 单元
```

(3) 加 1 指令。

格式：

```
INC  reg/mem            ;reg/mem←(reg)/(mem)+1
```

要求：reg 为 8 位或 16 位通用寄存器，mem 为 8 位或 16 位存储单元。

功能：将源操作数加 1，再送回该操作数。这条指令一般用于循环程序的指针修改，INC 指令只有一个操作数。

说明：

- 操作数可以是寄存器或存储单元，但不能为立即数。
- INC 指令影响标志位 AF、OF、PF、SF 和 ZF，但不影响 CF 位。
- INC 指令将操作数视为无符号数。

例 3.37

```
INC  AL                 ;AL←(AL)+1
INC  BX                 ;BX←(BX)+1
INC  BYTE PTR MEMLOC    ;MEMLOC←(MEMLOC)+1
```

2) 减法类指令

(1) 不带借位的减法指令。

格式：

```
SUB  dest,src           ;dest←(dest)-(src)
```

功能：将目标操作数减去源操作数，结果送到目标操作数，并根据结果设置标志。

说明：

- SUB 指令不允许两个存储器单元内容相减，两个操作数不能同时为存储器操作数。
- SUB 指令也不允许在两个段寄存器之间相减。
- 对标志位有影响，主要影响 CF、ZF、OF、SF 标志位。

例 3.38

```
SUB  AX,BX              ;AX←(AX)-(BX)
SUB  SI,DS:[100H]       ;SI 的内容减去地址 DS:[100H]和 DS:[101H]所指内容→SI
SUB  AL,30H             ;AL←(AL)-30H
```

```
SUB  DS:[20H],1000H        ;地址 DS:[20H]所指的字单元的 16 位
                           ;减去立即数 1000H→DS:[20H]字单元
SUB  SS:[1000H+2],CL       ;SS 段偏移地址 1000H+2 所指的字节单元
                           ;减去 CL→1000H+2 字节单元
```

(2) 带借位的减法指令。

格式:

```
SBB  dest,src              ;dest←(dest)-(src)-(CF)
```

SBB 指令与 SUB 指令的功能相似,区别是 SBB 在完成字节或字相减的同时还要减去借位 CF。

说明:

- src(源操作数)和 dest(目标操作数)不能同时为存储单元。
- 段寄存器不能进行算术运算。

例 3.39

```
SBB  AX,BX                 ;(AX)←(AX)-(BX)-(CF)
SBB  AX,2010H              ;(AX)←(AX)-2010H-(CF)
```

(3) 减 1 指令。

格式:

```
DEC  reg/mem               ;reg/mem←(reg)/(mem)-1
```

减 1 指令只有一个操作数。操作数可以为寄存器或者存储单元,不能为立即数。该指令实现将操作数中的内容减 1,又叫减量指令。

例 3.40

```
DEC  CX                ;(CX)←(CX)-1
DEC  DS:[100H+2]       ;将数据段 DS 偏移地址 100H+2 所指单元内容减 1,结果送回该单元
```

注意:DEC 指令和 INC 指令一样,执行后对 CF 不产生影响。

(4) 取补指令。

格式:

```
NEG  reg/mem           ;reg/mem←0-(reg)/(mem)
```

功能:将操作数取补后送回源操作数,即将操作数连同符号逐位取反,然后在末位加 1,这适用于操作数在机器内用补码表示的场合。

NEG 指令的操作数可以是 8 位或 16 位通用寄存器和存储器操作数,不能为立即数。

例 3.41

```
MOV  AL,00000001B          ;AL=00000001B
NEG  AL                    ;将 AL 中的数取补,AL=11111111B
NEG  WORD PTR [SI+1]       ;将 SI+1、SI+2 单元中的内容取补
```

注意:因为执行 NEG 指令时使用 0 减去操作数,只有操作数为 0 时才使 CF=0,否

则使 CF＝1。如果操作数的值为－128(80H)或－32 768(8000H),执行取补指令之后结果没有变化,即取补后送回的新值仍为 80H 或者 8000H,但此时的溢出标志 OF＝1。

(5) 比较指令。

格式:

```
CMP  dest,src                ;(dest)-(src)
```

功能:将目标操作数与源操作数相减,不送回结果,只根据结果置标志位。

例 3.42

```
CMP  AX,BX                   ;将 AX-BX 后,置标志位
CMP  AL,20H                  ;将 AX-20H 后,置标志位
```

说明:本指令通过比较(相减)结果置标志位,表示两个操作数的关系。比较以下几种情况,以 CMP A,B 为例说明。

① 判断两个操作数是否相等。可根据 ZF 标志位判断:若 ZF＝1,说明 A＝B;若 ZF＝0,说明 A≠B。

② 判断两个操作数的大小。

- 判断两个无符号操作数的大小。可根据 CF 标志位判断:若 CF＝1,说明 A＜B;若 CF＝0,说明 A≥B。
- 判断两个带符号操作数的大小。可根据 SF 及 OF 标志判断:若 SF⊕OF＝1,即 SF 与 OF 不同时,说明 A＜B;若 SF⊕OF＝0,即 SF 与 OF 相同时,则 A≥B。

例 3.43

```
CMP  EXTMEM,61FAH            ;将存储器 EXTMEM 数与立即数相比,置标志位
CMP  AX,0FF00H               ;将 AX 寄存器与立即数相比,置标志位
```

一般情况下,CMP 指令后经常会有一条条件转移指令,用来检查标志位的状态是否满足某种关系。

例 3.44

```
    CMP  AL,5
    JZ   ABC                 ;若 AL=5 则转 ABC,否则顺序执行
    ⋮
ABC:
    ⋮
```

3) 乘法指令

(1) 无符号数乘法指令。

格式:

```
MUL  reg/mem                 ;dest←隐含被乘数 AL/AX 乘以乘数 reg/mem
```

功能:完成两个不带符号的 8 位或 16 位二进制数的乘法计算。乘积存放在 AH、AL 或 DX、AX 中。

例 3.45

```
MOV  AL,NUMBER1
MUL  NUMBER2              ;将 AL 的内容乘以 NUMBER2,乘积存放在 AX 中
```

例 3.46

```
MOV  AX,NUMBER1
MUL  NUMBER2              ;将 AX 的内容乘以 NUMBER2,乘积的高 16 位
                          ;放在 DX 中,低 16 位放在 AX 中
```

(2) 带符号乘法指令。

格式:

```
IMUL  reg/mem
```

功能:完成两个带符号的 8 位或 16 位二进制数乘法计算。乘积存放在 AH、AL 或 DX、AX 中。

例 3.47

```
MOV   AX,1234H
IMUL  NUMBER              ;乘积高 16 位放在 DX 中,低 16 位放在 AX 中
```

4) 除法指令

(1) 无符号数除法指令。

格式:

```
DIV  src                  ;隐含被除数 AX/DX 或 AX 除以除数(src)
```

功能:当用 DIV 指令进行无符号数的字或字节相除时,所得的商和余数均为无符号数,分别放在 AL 和 AH 中;若进行无符号数的双字或字相除时,所得的商和余数也是无符号数,分别放在 AX 和 DX 中。

注意:除法有一种特殊情况。例如被除数为 1000,放在 AX 中,除以 2,商为 500,应放在 AL 中,余数为 0,应放在 AH 中。此时 500 超过了 AL 的最大范围 256,会产生 0 号中断。

例 3.48

```
MOV  AX,NUMBER
DIV  DIVSR                ;商在 AL 中,余数在 AH 中
```

(2) 带符号除法指令。

格式:

```
IDIV  src
```

功能:IDIV 指令用于两个带符号数相除,其功能和对操作数长度的要求与 DIV 指令类似,本指令执行时,将被除数、除数都作为带符号数,其相除操作和 DIV 相同。