第 3 章　Pentium 的指令系统

微处理器的指令系统与其性能密切相关，CPU 设计中千方百计采用各种先进技术，最终都集中体现在通过性能优越的指令系统更快更好地运行各种程序，实现更多更强的性能。

Pentium 的指令系统也称为 80×86 指令系统，这是在 16 位的 8086 指令系统基础上，随着 80386/80486/Pentium 的相继推出而不断扩展而来的，它兼容了低档 CPU 的全部指令。在功能上，这是一个设计得非常成功的指令系统。

本章讲述 Pentium 的寻址方式，并讲述 Pentium 指令系统指令的含义、设计考虑和使用方法。为便于读者编程时参考，在《微型计算机技术及应用——习题、实验题和综合训练题集》的附录中列出了更详细的说明和使用例子。

3.1　Pentium 的寻址方式

对于一条汇编语言指令来说，有两个问题要解决：首先，需要指出进行什么操作，这由指令操作符来表明；其次，需要指出大多数指令涉及的操作数和操作结果放在何处。为了使指令形式比较简单，一般情况下，约定将操作结果送到原来放操作数的地方，这样，第二个问题就归结为指出操作数的来源，这就是操作数的寻址方式。

指令系统中有一类指令叫转移指令，还有一类叫调用指令，这两类指令涉及转移地址或调用地址的提供方式，一般也称为指令地址的寻址方式。

所以，实际上在两种情况下涉及寻址方式：一种是对操作数进行寻址；另一种是对转移地址和调用地址进行寻址。下面所讨论的寻址方式都是针对操作数的，关于指令地址的寻址，将在讲述转移指令和调用指令时作具体说明。

对于操作数来说，实际上，可以有四种来源：直接由指令本身提供；由寄存器提供；由存储器提供；或者由输入端口提供。反过来，操作结果可以有三种去向：送到寄存器、存储器或输出端口。这样，综合起来就有四类寻址方式：立即数寻址、寄存器寻址、输入/输出端口寻址和存储器寻址。前面三种寻址方式都比较简单，但是，存储器的地址可以用多种方法提供，所以对应存储器有多种寻址方式。其实，前两种情况下，并不需要寻找操作数的地址，但是，习惯上也称为寻址。

3.1.1　立即数寻址

Pentium 指令系统中，有一部分指令所用的操作数就在指令中提供，这种方式叫立即数寻址。比如：

```
MOV     AL,80H              ;将 80H 送入 AL
MOV     AX,1090H            ;将 1090H 送 AX,AH 中为 10H,AL 中为 90H
MOV     EAX,10002000H       ;将 10002000H 送 EAX
```

注意：

① 立即数只能作为源操作数，不能作为目的操作数。

② 立即数寻址一般用于对寄存器赋值。

③ 因为操作数可从指令中直接取得，不需要总线周期，所以，立即数寻址方式的显著特点就是速度快。

3.1.2 寄存器寻址

如果操作数在 CPU 的内部寄存器中，那么，寄存器名可在指令中指出，此为寄存器寻址方式。

对 8 位操作数来说，寄存器可为 AH、AL、BH、BL、CH、CL、DH、DL；对 16 位操作数来说，寄存器可为 AX、BX、CX、DX、SI、DI、SP 或 BP；而对 32 位操作数来说，寄存器可为 EAX、EBX、ECX、EDX、ESI、EDI、ESP 或 EBP。比如：

```
INC     CX          ;将 CX 的内容加 1
ROL     AH,1        ;将 AH 中的内容循环左移一位
MOV     ECX ,EAX    ;将 EAX 中的 32 位数送 ECX
```

注意：

① 寄存器寻址的指令执行时，操作就在 CPU 内部进行，不需要使用总线周期，因此，执行速度快。

② 可对源操作数用寄存器寻址，也可对目的操作数用寄存器寻址，还可两者都用寄存器寻址。

3.1.3 输入/输出端口寻址

CPU 和外部设备都是通过输入/输出(I/O)端口来传输数据。在此过程中，涉及直接寻址和间接寻址两种方式。

1. I/O 直接寻址

用这种方式时，I/O 端口的地址直接在指令中提供。比如：

```
IN      AL,82H      ;82H 端口中的字节输入到 AL
OUT     80H,AX      ;将 AL 中的数据送 80H 端口，将 AH 中的数据送 81H 端口
IN      EAX,80H     ;将 80H～83H 端口中的 4 个字节由低到高送 EAX 中
```

2. I/O 间接寻址

用这种方式时，先在 DX 寄存器中设置好 I/O 端口的地址，I/O 指令用 DX 进行寄存器间接寻址。比如，事先在 DX 中设置端口号 80H，那么，下列指令完成注释中对应的功能。

```
IN      AL,DX       ;将端口 80H 中的字节读入 AL
OUT     DX,AX       ;将 AL 中内容输出到端口 80H，将 AH 中的内容输出到端口 81H
OUT     DX,EAX      ;将 EAX 中的 4 个字节由低到高分别送 80H～83H 这 4 个端口
```

注意：

① I/O 直接寻址时，寻址范围为 0～255，即最大端口号为 FFH。

② I/O 间接寻址时，只能用 DX 寄存器，寻址范围为 0～65 535，即最大端口号为 FFFFH。

3.1.4 存储器寻址

程序运行时，大多数情况下，操作数在存储器中，为此，Pentium 提供了多种存储器寻址方式。存储单元的地址由段基址和偏移量组成，但通常一个程序涉及的数据都放在某一个段中，段基址确定以后并不需要经常改变。所以，对存储单元的寻址实际上是确定偏移量即有效地址 EA(effective address)。

EA 的完整表达式为：

EA＝基址＋变址×比例因子＋位移量

从上式可见，有效地址由基址、变址、位移量三个分量计算得到，而变址分量还可以带一个比例因子。

① **基址**　任何通用寄存器都可作为基址寄存器，其内容即为基址。注意，这里的基址不是段基址，而只是一个延续下来的习惯叫法，实际上是指有效地址的一个基础量。

② **位移量**　这是指令操作码后面的 32 位、16 位或 8 位的数。

③ **变址**　除了 ESP 寄存器外，其他通用寄存器都可作为变址寄存器，但常用的变址寄存器为 DI、SI、EDI 和 ESI。

④ **比例因子**　变址寄存器的值可乘以一个比例因子，比例因子可为 1、2、4 或 8。

至于段基址，在保护方式，是指段寄存器中的选择子所指的相应段描述符中的段基址，在实地址方式和模拟 8086 方式，就是段寄存器中的值左移 4 位得到的值。

图 3.1 表示了这种寻址计算方法。

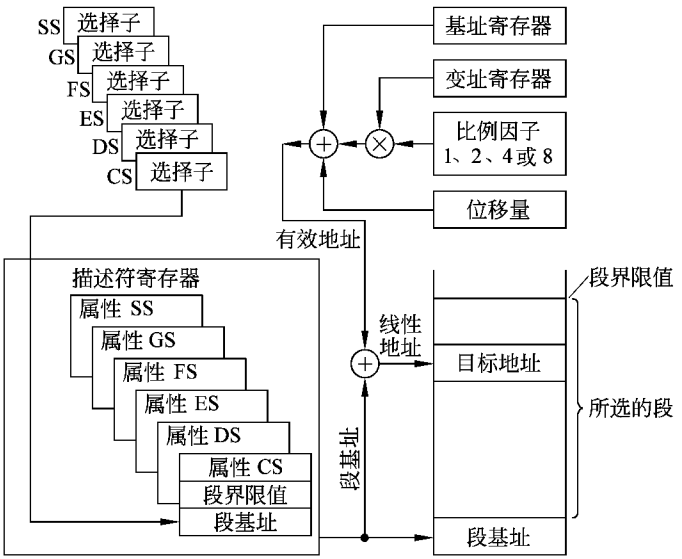

图 3.1　寻址计算图解

按照上述表达式，可以有如下八种存储器的寻址方式，但只有前五种是常用的。

1. 直接寻址

这种方式中，存储单元的有效地址由指令直接指出，所以，直接寻址是对存储器访问时可采用的最简单方式。比如：

```
MOV     AX,[1070H]          ;将 DS 段的 1070H 和 1071H 两单元的内容取到 AX 中
MOV     EAX,ES:[1000H ]     ;将 ES 段 1000H 开始的 4 个字节的数据送到 EAX 中
```

2. 寄存器间接寻址

这种方式中,存储单元的有效地址由寄存器指出。可用 16 位寄存器 AX、BX、CX、DX、SI、DI、BP 或 SP,也可用 32 位寄存器 EAX、EBX、ECX、EDX、ESI、EDI、EBP 和 ESP。

要注意的是:

① 用 BP、SP、EBP 和 ESP 进行间接寻址时,默认段为 SS,用其他寄存器进行间接寻址时,默认段为 DS。比如:

```
MOV     AX,[BX]             ;将 DS 段中由 BX 所指地址开始的两单元的内容送 AX
MOV     EAX,[EBX]           ;将 DS 段中由 EBX 所指地址开始的 4 个单元的内容
                            ;送 EAX
MOV     EAX,[BP]            ;对 SS 段由 BP 所指单元开始的 4 个字节送 EAX
```

② 如果对非默认段进行寻址,则必须在指令前用前缀指出段寄存器名。比如:

```
MOV     CX,ES:[BX]          ;将 ES 段由 BX 所指单元开始的 2 个字节送 CX
```

SI、DI、EDI 和 ESI 称为变址寄存器,所以用这 4 个寄存器间接寻址也叫变址寻址。变址寻址通常用于对数组元素操作,另外,后面还将讲到串操作指令要求用固定的变址寄存器对操作数寻址,操作过程中,指令会自动修改变址寄存器中的地址,以指向下一个操作数。

3. 寄存器相对寻址

用这种方式时,EA 为寄存器中内容和指令中给出的位移量的和。位移量可正可负,被看成是对寄存器所指地址的一个相对值,所以,称为寄存器相对寻址,也称为带位移量的寄存器间接寻址。位移量可以是 8 位、16 位或 32 位。比如:

```
MOV     AX,[SI+100H]        ;如 SI=2000H,则将 DS 段 2100H~2101H 中内容送 AX
```

寄存器相对寻址常常用于表格处理,此时,将表格的首地址作为位移量,通过修改寄存器的内容指向表格中的某一项。

4. 基址加变址的寻址

通常将 EBX、EBP、BX 和 BP 称为基址寄存器,将 ESI、EDI、SI 和 DI 称为变址寄存器,把基址寄存器和变址寄存器组合起来可构成一种新的寻址方式,即基址加变址的寻址。用这种方式时,操作数的有效地址为基址寄存器的内容加上变址寄存器的内容。比如:

```
MOV     AX,[BX+SI]          ;将 BX 和 SI 中的内容之和所指单元开始的 2 字节送 AX
MOV     EDX,[EBX+ESI]       ;将 EBX 和 ESI 共同指出的地址开始的 4 字节送 EDX
MOV     EDX,[EBX][ESI]      ;和上面的指令功能相同
```

注意:

① 在基址加变址的寻址方式中,有一个对段寄存器的约定规则,即如果将 EBP 或 BP 作为基址寄存器,则默认段为 SS,在其他情况下,默认段为 DS。

② 如果操作数不在默认段,则要用前缀指出相应的段寄存器名。

由于基址加变址的寻址方式中,允许两个地址分量分别改变,所以,使用起来很灵活,特别是为访问堆栈中的数组提供了极大的方便,所以,常常用于数组或表格处理,可用基址寄

存器放数组或表格首地址，用变址寄存器指向数组或表格的某一个项。

5. 相对的基址加变址寻址

用基址加变址来指出存储单元地址时，也允许带一个位移量，成为相对的基址加变址寻址。对这种寻址方式的约定和使用场合同于基址加变址寻址方式。比如：

```
MOV     AX,[BP+SI+0050]          ;将 ES 段由 BP 和 SI 中的内容与 0050 相加作为有效
                                 ;地址
```

如图 3.2 所示，在访问堆栈数组时，可在 BP(或 EBP)中存放栈顶的地址，用位移量表示数组第一个元素到栈顶的距离，变址寄存器 SI(也可为 DI、EDI 或 ESI)指向数组元素。

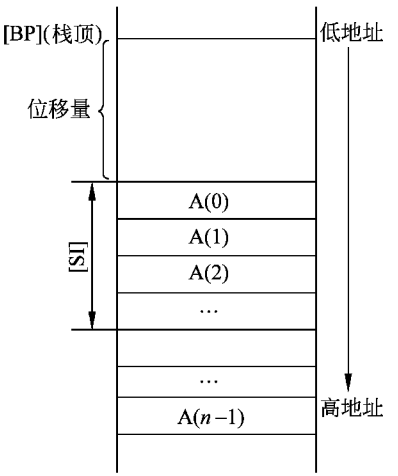

图 3.2　对堆栈中数组的访问

6. 相对的带比例因子的变址寻址

用这种方式时，EA 为变址寄存器中的值乘以比例因子再加位移量，比例因子可为且只可为 1、2、4、8，这是为了便于处理元素长度为 1、2、4、8 字节的数组。比如：

```
IMUL    EBX,[ESI*4+7]    ;ESI 的内容乘以 4 再
                         ;加 7 形成有效地址
```

7. 基址加比例因子的变址寻址

这种方式中，EA 为基址寄存器的内容与比例因子和变址寄存器内容的乘积之和，比例因子可为且只可为 1、2、4、8。比如：

```
MOV     EAX,[EBX][ESI*4]         ;将 DS 段由 EBX+ESI×4 所指单元开始的 4 字节送 EAX
MOV     ECX,[EDI*8][EAX]         ;EDI 内容乘以 8 再加 EAX 内容即为有效地址
```

8. 相对的基址加比例因子的变址寻址

这是计算有效地址的最完整的表达方式，有效地址为基址寄存器的内容、比例因子与变址寄存器内容的乘积、以及位移量三者之和。比如：

```
MOV     EAX,[EDI*4][EBP+80]      ;EDI 的内容乘 4，加 EBP 的内容，再加 80 即为有效地址
```

3.2　Pentium 的指令系统

Pentium 的指令分为如下几类：

- 传送指令；
- 算术运算指令；
- 逻辑运算和移位指令；
- 串操作指令；
- 调用/转移/循环控制/中断指令；
- 标志操作和处理器控制指令；
- 条件测试和字节设置指令；

- 位处理指令；
- 系统管理指令；
- 支持高级语言的指令。

3.2.1 传送指令

Pentium 有七组传送指令，用来实现 CPU 的内部寄存器之间、CPU 和存储器之间、CPU 和 I/O 端口之间的数据传送。这七组指令是：通用传送指令、堆栈操作指令、交换指令、输入/输出指令、换码指令、地址传送指令和标志传送指令。

1. 通用传送指令 MOV 和 MOVZX/MOVSX

(1) 两个操作数的位数相同的传送指令 MOV

MOV 指令是用得最多的指令，它可实现 CPU 内部寄存器之间的数据传送、寄存器和内存之间的数据传送，还可把一个立即数送给内部寄存器或内存单元。比如：

```
MOV     AL,BL                   ;BL 中的 8 位数据送 AL
MOV     ES,DX                   ;DX 中 16 位数据送 ES
MOV     AX,[BX]                 ;BX 和 BX+1 所指的两个内存单元的内容送 AX
MOV     [DI],AX                 ;累加器的内容送 DI 和 DI+1 所指的两个单元
MOV     CX,[1000]               ;将 1000 和 1001 两单元的内容送 CX
MOV     WORD PTR [SI],6070H     ;将 6070H 送 SI 和 SI+1 所指出的两单元
MOV     DX,5040H                ;立即数 5040H 送 DX
MOV     EAX,[EBX+ECX*2+1000H]   ;将 DS 段由 EBX 的内容加 ECX 内容乘以 2,再加上
                                ;1000H 三者之和所指单元开始的 4 个字节送 EAX
MOV     CRn,EAX                 ;往控制寄存器 CRn 中设置一个 32 位值,其中
                                ;CRn 可为 CR0、CR2、CR3 或 CR4
MOV     DRn,EAX                 ;往调试寄存器 DRn 设置一个初值,DRn 可为
                                ;DR0～DR3、DR6、DR7
```

(2) 两个操作数的位数不相同的传送指令 MOVZX/MOVSX

当传送的源操作数和目的操作数位数不相同时，就要用到 MOVZX 和 MOVSX 指令，前者在传送时进行零扩展，后者在传送时进行符号扩展。比如：设 BL 中为 80H，下面指令分别实现对应注释中的功能。

```
MOVZX   EAX,BL                  ;80H 被零扩展为 0000 0080H 送 EAX 中
MOVSX   EAX,BL                  ;80H 被符号扩展为 FFFF FF80H 送 EAX 中
```

对于通用传送指令的使用，有以下几点需要注意：

① 通用传送指令可传送 8 位、16 位或 32 位数据，具体取决于指令中涉及的寄存器是 8 位、16 位还是 32 位，也取决于立即数的形式。

② 通用传送指令中总是既含源操作数，又含目的操作数，两者之中至少有一个是用寄存器指出的，这可减少指令长度。

③ 不能在两个内存单元之间直接传送数据。

④ 不能从一个段寄存器向另一个段寄存器传送数据。

⑤ 在通用传送指令中，寄存器既可作为源操作数，也可作为目的操作数，但 CS、IP、EIP

寄存器不能作为目的操作数，即这些寄存器的值不能随意修改。

⑥ 用 EBX、ESI、EDI、BX、SI、DI 来间接寻址时，默认的段寄存器为 DS，而用 EBP、ESP、BP 或 SP 来间接寻址时，默认的段寄存器为 SS。

⑦ 当执行给 SS 寄存器赋值的传送指令时，系统会自动禁止外部中断，等到本条指令和下条指令执行之后，又自动恢复对 SS 寄存器赋值前的中断开放情况。这样做是为了允许程序连续用两条指令分别对 SS 和 SP(或 ESP)寄存器赋值，同时又防止堆栈空间变动过程中出现中断。了解这点后，就应该注意在修改 SS 和 SP(或 ESP)的指令之间不要插入其他指令。

⑧ 所有的通用传送指令都不改变标志。

2. 堆栈操作指令 PUSH/POP、PUSHA/POPA 和 PUSHAD/POPAD

计算机运行中需要设置堆栈，堆栈是内存中的特殊存储区，按照“先进后出、后进先出”的规则存取数据。在子程序调用和转向中断处理程序时，分别要保存返回地址或断点地址；在进入子程序和中断处理程序后，还需要保留通用寄存器的值；子程序返回和中断处理程序返回时，则要恢复通用寄存器的值，并分别将返回地址或断点地址恢复到指令指针寄存器中。这些功能都要通过堆栈来实现，其中寄存器值的保存和恢复需要由堆栈操作指令来完成。

Pentium 指令系统中提供了两类堆栈操作指令。一类是普通的堆栈操作指令，一类是堆栈成组操作指令。

(1) 普通堆栈操作指令 PUSH/POP

PUSH 指令把一个 2 字节或 4 字节的数据推入堆栈，操作数既可为立即数，也可为寄存器和存储器中的数。比如：

```
PUSH    EAX                     ;将 EAX 的内容推入堆栈，栈顶为低位字节，栈指针减 4
PUSH    BX                      ;将 BX 的内容推入堆栈，堆栈指针减 2
PUSH    [BX+DI]                 ;将 BX+DI 和 BX+DI+1 所指两单元的内容推入堆栈
PUSH    0870H                   ;将立即数 0870H 推入堆栈，栈顶单元中为 70H
PUSH    DWORD PTR [EBX+ESI]     ;将 DS 段中由 EBX 和 ESI 的内容所指单元开始的 4 个
                                ;字节推入堆栈，栈顶地址减 4
```

同样，还有一系列弹出指令。比如：

```
POP     BX                      ;将栈顶两单元弹出送 BX，栈顶地址加 2
POP     ES                      ;将栈顶两单元弹出送 ES，栈顶地址加 2
POP     EAX                     ;将栈顶 4 个单元弹出送 EAX，栈顶地址加 4
```

(2) 堆栈成组操作指令 PUSHA/POPA 和 PUSHAD/POPAD

堆栈成组操作指令有很强的功能，用一条 PUSHA 指令就可将 8 个 16 位通用寄存器的内容按 AX、CX、DX、BX、SP、BP、SI、DI 的次序推入堆栈，堆栈指针减 16，栈顶单元为 DI 的低 8 位，其中进栈的 SP 内容是推入堆栈操作前的值。而用 PUSHAD 指令则可将 8 个 32 位寄存器推入堆栈，堆栈指针减 32。PUSHAD 推入堆栈的次序为：EAX、ECX、EDX、EBX、ESP、EBP、ESI、EDI，栈顶单元为 EDI 的低 8 位，进栈的 ESP 内容为推入堆栈操作前的值。

POPA/POPAD 指令和 PUSHA/PUSHAD 进行相反的操作，即从堆栈栈顶弹出一系列数据送到 8 个 16 位或 32 位寄存器，栈顶指针分别加 16 或 32。

堆栈操作指令的形式很简单，但使用时，仍有几点必须注意：

① 堆栈操作总是按字或双字进行的，不能按字节进行，也就是说，没有 PUSH AH、POP BL 这样的字节操作指令。

② 每推入一个字，堆栈地址指针减 2，推入堆栈的数据放在栈顶，低位字节放在较低地址单元(真正的栈顶单元)，高位字节放在较高地址单元。执行弹出指令时，正好相反，每弹出 1 个字，栈顶指针加 2。推入或弹出双字，则堆栈指针减 4 或加 4。堆栈成组操作指令则指针按 16 或 32 改变，但推入指令总是使指针的值减小，弹出指令总是使指针的值加大。

③ CS 寄存器的值可推入堆栈，但反过来，不能从堆栈中弹出 1 个值到 CS 寄存器。

④ 堆栈中的内容是按后进先出的次序进行传送的。因此，保存寄存器和恢复寄存器的内容时，要按照对称的次序执行一系列推入指令和弹出指令。即如果一个子程序开头这样保存寄存器的值：

```
PUSH    EAX
PUSH    EBX
PUSH    EDI
PUSH    ESI
```

则子程序返回前，应如下恢复寄存器的值：

```
POP     ESI
POP     EDI
POP     EBX
POP     EAX
```

3. 交换指令 XCHG/BSWAP

(1) 字节、字和双字交换指令 XCHG

XCHG 指令可实现字节交换、字交换或双字交换。交换过程可以在通用寄存器之间进行，也可在通用寄存器和存储单元之间进行，但不能在存储单元之间进行。一条 XCHG 指令相当于三条 MOV 指令。比如：

```
XCHG    AL,BL          ;AL 和 BL 之间进行字节交换
XCHG    BX,CX          ;BX 和 CX 之间进行字交换
XCHG    [2530],CX      ;CX 中的内容和 2530、2531 两单元的内容交换
XCHG    EAX,EDI        ;寄存器和寄存器进行双字交换
XCHG    ESI,[EBX]      ;寄存器和内存进行双字交换
```

使用交换指令时，要注意以下两点：

① 目的操作数和源操作数不能均为内存单元。

② 段寄存器和 IP、EIP 寄存器不能作为交换指令的操作数。

(2) 寄存器内部字节交换指令 BSWAP

BSWAP 指令将指定的 32 位寄存器中的 4 个字节通过两两交换实现反序排列，即双字

的第 31～24 位与第 7～0 位交换，第 23～16 位与第 15～8 位交换，使数据次序按字节为单位反过来，以此改变数据的存放方式。

比如：[EAX]＝0123 4567H，则执行指令：

```
BSWAP     EAX          ;使[EAX]＝6745 2301H
```

4. 输入/输出指令 IN/OUT

在主机和外部设备之间传送信息时，用输入/输出指令。对输入/输出端口的寻址有两种方式：直接寻址和利用 DX 寄存器的间接寻址。

执行输入指令时，CPU 可从 1 个 8 位端口读入 1 个字节到 AL 中，也可从 2 个连续的端口读 1 个字到 AX 中，或者从连续的 4 个 8 位端口读 1 个双字到 EAX 中。

执行输出指令时，CPU 可将 AL 中的 1 个字节写到 1 个 8 位端口，也可将 AX 中的 1 个字写到 2 个连续的 8 位端口中，或将 EAX 中的双字写到连续的 4 个端口中。

输入/输出指令即 I/O 指令可分为两大类：一类是直接的 I/O 指令；另一类是间接的 I/O 指令。

(1) 直接的 I/O 指令

这些指令中直接提供了 I/O 端口号。比如：

```
IN        AL,50H       ;将 50H 端口的字节读入 AL
IN        AX,70H       ;将 70H、71H 两端口的值读入AX，70H 端口的值读入 AL，71H 端
                       ;口的值读入 AH
IN        EAX,70H      ;将 70H～73H 共 4 个端口的值读入 EAX
OUT       80H,AX       ;将 AX 中的内容输出到 80H、81H 端口
```

(2) 间接的 I/O 指令

间接 I/O 指令执行前，必须在 DX 寄存器中设置好端口号。假设现在 DX 寄存器中为 80H，则下面指令的功能如注释所示。

```
IN        AL,DX        ;从 DX 所指的端口 80H 中读取 1 个字节
IN        AX,DX        ;从两个端口中读取 1 个字送到 AX 中，80H 中值送 AL，81H 中值送 AH
IN        EAX,DX       ;从 80H～83H 共 4 个端口读取 1 个双字送到 EAX 中，83H 中读取的
                       ;值为 EAX 的最高 8 位
OUT       DX,AX        ;将 AX 中的字输出到 80H、81H 端口
OUT       DX,EAX       ;将 EAX 中的双字输出到 80H～83H 共 4 个端口
```

使用 I/O 指令时，要注意以下几点：

① 只能用累加器作为执行 I/O 过程的机构，不能用其他寄存器代替。

② 用直接 I/O 指令时，寻址范围为 0～255，即 FFH 是直接 I/O 指令中允许使用的最大端口号。而 Pentium 最多可有 65 535 个外设端口，对应的端口地址为 0～FFFFH，当端口号大于 255 时，只能用间接的 I/O 指令。当然，凡是能用直接 I/O 指令的地方都可用间接 I/O 指令来代替。间接 I/O 指令的寻址范围为 0～65 535。不过用间接 I/O 指令前，要在 DX 寄存器中设置好端口号。

③ 用间接 I/O 指令时，特别要注意，只能用 DX 寄存器，而不能用别的寄存器，甚至不能用 EDX 寄存器。

④ 每次 I/O 指令传输的字节数决定于累加器，如用 AL，则 I/O 操作对应一个字节，如用 AX，则 I/O 操作对应一个字，如用 EAX，则 I/O 操作对应一个双字。

⑤ I/O 指令不影响标志位。

5. 换码指令 XLAT/XLATB

XLAT 和 XLATB 指令将累加器中的一个值变换为内存表格中的某一个值，一般用来实现编码制的转换，换码指令的名称也正是由此而来。

使用换码指令时，要求用 BX 或 EBX 寄存器指向表的首地址，AL 中为表内某一项的地址与表格首地址之间的位移量，指令执行时，会将 BX 或 EBX 中的值和 AL 中的值相加作为有效地址，然后将此地址所指单元中的值取到 AL 中。如以 BX 为基址寄存器，则指令形式为 XLAT；如以 EBX 为基址寄存器，则指令形式为 XLATB。XLATB 和 XLAT 指令功能相同，只是 XLATB 以 EBX 为基址，这样，就可对更大的数据组进行换码操作。

图 3.3 表示了以 BX 为基址寄存器时换码指令的功能。

换码指令对一些不规则代码的转换非常方便。比如，通信系统中用到这样一种代码，称为格雷码，其中每个码由两个 1、3 个 0 组成，这种代码比较容易检错和纠错。具体编码规则如下：

0——11000	1——00011	2——00101	3——00110
4——01001	5——01010	6——01100	7——10001
8——10010	9——10100		

从二进制数的角度看这些编码，它们之间没有明显的规律，这时就可用换码指令来实现转换，即在内存中存放一个代码表，如图 3.4 所示。

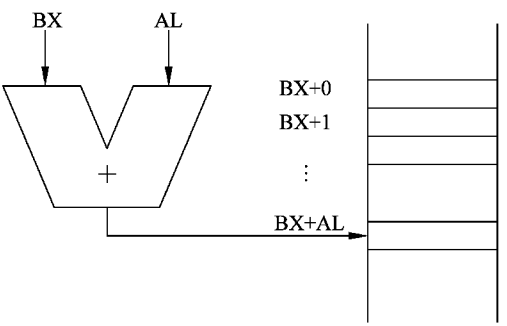

图 3.3 换码指令的功能

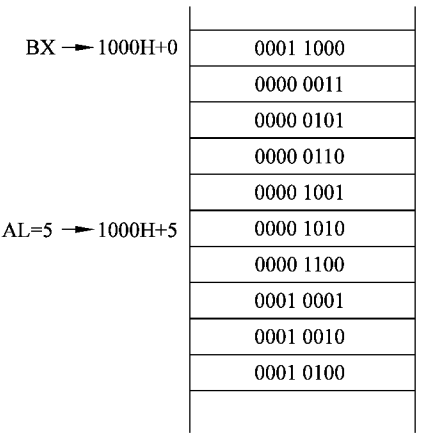

图 3.4 XLAT 指令对表格的访问

于是，要将二进制数字（比如 5）转换成对应代码时，只要将此二进制数送 AL 寄存器，而表的首地址（这里为 1000H）送到 BX，之后，再执行 XLAT 指令就行了。

6. 地址传送指令 LEA 和 LDS/LES/LSS/LFS/LGS

Pentium 指令系统中，有 6 条专用于传送地址的指令，即 LEA、LDS、LES、LSS、LFS 和 LGS。