第 3 章　微处理器指令系统

指令系统是微处理器所能执行指令的集合，这组指令集定义了计算机硬件所能完成的基本操作。不同系列的微处理器由于其内部结构不同而有不同的指令系统，随着处理器的升级指令系统也会不断扩充。CISC 处理器的指令系统比较丰富完善，且有专用指令实现特定的功能。RISC 处理器指令系统的特色是常用功能的指令简单高效，处理特殊任务则要通过组合指令来完成。

3.1　指令格式

指令格式体现了指令系统的设计思想。要掌握指令系统，先了解指令格式是必要的。

各种微处理器的指令格式都是类似的，主要由操作码和操作数两部分组成。操作码指示CPU执行哪种操作，是数据传送、加减乘除，还是逻辑操作等，是指令中不可缺少的部分。操作数指示指令执行过程中所需要的数据，如加法指令中的加数被加数等。指令操作数的给出方式比较复杂，可以是参加操作的数本身（立即操作数），也可以是存放操作数的寄存器（寄存器操作数）或存储单元地址（存储器操作数）。如何寻找（规定）操作数是寻址方式问题，将在3.2节介绍。微处理器汇编语言指令的一般格式如下：

```
指令助记符　　操作数列表　　；注释
```

指令助记符是指令的名称，对应机器指令中的操作码部分。操作数可以有多个，因指令不同而不同，按序排列并用“,”分开。“;”后是注释部分，不影响指令的执行。

80x86/Pentium CPU汇编语言指令的一般格式如图3-1所示，占1～16字节。一条指令在被汇编翻译成机器指令后，对应若干字节的机器码。

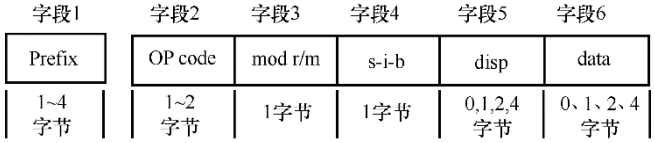


图 3-1　80x86/Pentium 指令的一般格式

在一条指令的 6 个字段中：2～6 为基本字段，1 是附加字段，简明意义如下。

（1）OP code 操作码字段。它规定指令的操作类型，说明指令所要完成的操作，同时指出操作数类型（字节/字/双字）、操作数传送方向、寄存器编码或操作数的符号扩展等。

（2）mod r/m 和 s-i-b 寻址方式字段。该字段规定寄存器/存储器操作数的寻址方式。mod r/m 为主寻址字节，规定操作数存放的位置（r/m）、存储器操作数有效地址 EA 的计算方法等。s-i-b 为“比例-变址-基址”寻址字节。一般访问存储器的指令中都含有主寻址字节，而“比例-变址-基址”寻址字节是否需要由主寻址字节的编码决定。

（3）disp 位移量字段。位移量是存储器操作数段内偏移地址的一部分。该字段指出位移量的大小，其长度为 0、1、2、4 字节。

（4）data 立即数字段。该字段指明立即操作数的数值大小，其长度也是 0、1、2、4 字节。8 位立即数与 16/32 位操作数一起使用时，汇编程序将它扩展至符号相同的 16/32 位数。同理，也可将 16 位立即数扩展至 32 位。

（5）Prefix 前缀字段。用于修改指令操作的某些属性。常用前缀有 5 类，每个前缀的编码为 1 字节。在一条指令前可同时使用多个指令前缀，不同前缀的前后顺序无关紧要。

① 段超越前缀：用于将前缀中指明的段寄存器取代指令中默认的段寄存器。

② 操作数宽度前缀：用于改变当前操作数宽度的默认值。

③ 地址宽度前缀：用于改变当前地址宽度的默认值。

④ 重复前缀：用于重复串的基本操作，以提高 CPU 处理串数据的速度。

⑤ 总线锁定前缀 LOCK：用于产生 LOCK 信号，以防止其他总线主控设备中断 CPU 在总线上的传输操作。

在指令中附加操作数宽度前缀或地址宽度前缀，可以保证 80386 及其以后的 CPU 在实模式和保护模式下均可执行 16 位和 32 位指令，实现 80x86/Pentium 系列 CPU 的兼容性。

在机器指令的 6 个字段中，只有操作码字段是必要的，其他字段是否需要，具体取决于所涉及的特定操作。

RISC 处理器，如 MIPS 或 ARM，指令格式的最大特征是机器码位数固定、编码紧凑。

MIPS处理器的汇编语言指令格式同80x86/Pentium，指令集采用32位编码，其3种机器码格式如图3-2所示，分别是寄存器类（R型）、立即数类（I型）和转移类（J型）。其中，SA（Special Area）字段比较复杂，可以由特定的CPU自定义，用来实现指令集的扩充。

	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
R 类型	操作码 OP	RS1	RS2	RD	SA	辅助操作码 OPX
I 类型	操作码 OP	源操作数寄存器 RS	目标操作数寄存器 RD	立即数 Immediate（16 bit）		
J 类型	操作码 OP	直接转移指令的目标地址 Target（26 bit）				

图 3-2　MIPS 的指令编码格式

ARM 体系结构从 ARM v1 发展到 ARM v8，指令集包含有 32 位和 16 位（Thumb）指令两类，指令功能不断扩大并向下兼容，ARM v8 还扩展了对 64 位寄存器的操作。ARM 汇编语言指令的一般格式如下：

```
指令助记符 条件 S      操作数列表          ; 注释
```

与 80x86/Pentium 指令格式略有不同，附着于助记符后的条件为指令执行的条件域，如 EQ、NE 等，是可选项；S 是指令的执行结果是否影响程序状态寄存器的后缀，有则影响。ARM 32 位或 16 位指令的机器码一般格式如图 3-3 和图 3-4 所示。

4 bit	3～4 bit	4～5 bit	4 bit	4 bit	11 bit
条件码	指令类型	操作码、寻址方式、操作是否影响 CPSR 值	源操作数寄存器	目标操作数寄存器	第二个源操作数、寄存器列表

图 3-2　ARM 32 位指令的一般格式

3～6 bit	1～3 bit	3～5 bit	3～8 bit	3 bit
指令类型	操作码	目标操作数寄存器、源操作数寄存器、立即数	源操作数寄存器、立即数、寄存器列表	第二个目标操作数寄存器

图 3-3　ARM Thumb 指令的一般格式

3.2 寻址方式

3.2.1 寻址方式与有效地址 EA 的概念

寻址方式就是指令中用来说明操作数怎样存放以及如何寻找操作数的方式。CPU 据此可以方便地访问各类操作数。通常，指令中的操作数有 3 种可能的存放位置：

- 操作数包含在指令中，即指令的操作数部分就是操作数本身。这种操作数称为立即数，对应的指令寻址方式称为立即寻址。
- 操作数存放在 CPU 的某个内部寄存器中，称为寄存器操作数。这时指令的操作数部分是 CPU 内部寄存器的编码，对应的寻址方式称为寄存器寻址。
- 操作数在内存的数据区中，称为存储器操作数。这时指令的操作数部分给出操作数的存储地址，指令执行时需要根据该地址找到需要的操作数。这种寻址方式称为存储器寻址。

CPU 指令中给出的存储器地址都是逻辑地址，逻辑地址与实际物理地址之间的映射由内存管理单元 MMU 来完成。以 Intel 80x86/Pentium 系列 CPU 指令为例，若内存中某一单元的逻辑地址用 DS:TABLE 来表示。在实地址模式下，DS 中是段基地址的高 16 位，称为段地址；对于保护模式，DS 中存放的是指向段描述符的 16 位段选择符，通过它可以得到 32 位的段基地址。TABLE 为段内偏移量，有 16 位（实模式下）和 32 位（保护模式下）两种。存储器寻址时，指令的操作数部分给出的地址就是这个段内偏移量。为了在程序设计时能够适应各种数据结构的需要，80x86/Pentium 系列 CPU 规定这个段内偏移量可以由几个基本部分组合而成，所以也把它称为有效地址 EA。

组成有效地址 EA 的基本部分包括：基址寄存器内容，变址寄存器内容，位移量，比例因子。这四个基本部分称为有效地址四分量。其中，基址、变址寄存器中通常为某局部存储区的起点；位移量是指令中的 disp 字段，是一个具体数值，也可以是符号常量或变量；比例因子是 32 位寻址方式中特有的部分。它们的组合情况和计算方法如下：

EA=(基址寄存器)+(变址寄存器×比例因子)+位移量

对于 16 位寻址和 32 位寻址方式，有效地址 4 种分量的使用规则有所不同，如表 3-1 所示。由这 4 种分量可组合出多种存储器寻址方式。

表 3-1 有效地址 4 种分量的使用规则

有效地址分量	16 位寻址	32 位寻址
基址寄存器	BX，BP	任何 32 位通用寄存器（EAX/EBX/ECX/EDX/ESI/EDI/EBP/ESP）
变址寄存器	SI，DI	除 ESP 外的任何 32 位通用寄存器（EAX/EBX/ECX/EDX/ESI/EDI/EBP）
比例因子	1	1，2，4，8
位移量	0，8，16（位）	0，8，32（位）

3.2.2 80x86/Pentium 各种寻址方式

80x86/Pentium CPU 指令系统的寻址方式有 11 种，对 8086/8088/80286（16 位寻址）来说只有 8 种，80386/80486 及 Pentium 系列 CPU 除了可 16 位寻址，还支持 32 位寻址方式。以下介绍中均以指令的源操作数为对象进行分析。

1．立即寻址

立即寻址方式下，操作数作为立即数直接包含在指令中，紧跟在操作码之后与其一起存放在代码段区域。因此，立即数总是和操作码一起被存入 CPU 的指令队列，在指令执行时不需再访问存储器。

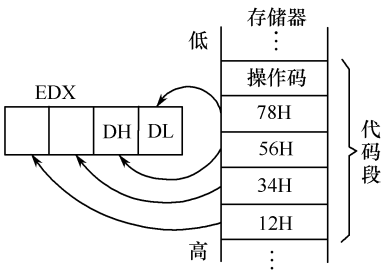


图 3-2　立即寻址方式

立即操作数可以是 8 位、16 位或 32 位。若是 16 位，则低位字节存放在相邻 2 字节存储单元的低地址单元中；若是 32 位，则低位字存放在相邻 2 字存储单元的低地址单元中。立即寻址方式仅用于源操作数，常用来给寄存器赋初值。例如：

```
MOV     BL，12H
MOV     AX，1234H
MOV     EDX，12345678H
```

其中，MOV　EDX, 12345678H 的执行过程如图 3-2 所示。

2．寄存器寻址方式

寄存器寻址方式的操作数存放在指令规定的 8 位、16 位或 32 位 CPU 内部寄存器中。例如：

```
INC     CL                  ; CL←CL+1
MOV     DS，AX              ; DS←AX
MOV     ECX，EAX            ; ECX←EAX
```

显然，这是 CPU 内部操作，不需要使用访问总线周期，所以指令的执行速度较快。

3．存储器寻址方式

存储器寻址时，操作数是在存储区中，指令的操作数部分指出此操作数的有效地址 EA，而段地址在默认的或段超越前缀指定的段寄存器中。根据 EA 的不同组合方式，存储器寻址方式又可分为以下 9 种。

（1）直接寻址

直接寻址是一种最简单的存储器寻址方式。在这种寻址方式下，指令中操作数部分直接给出操作数的有效地址 EA，它是 16 位或 32 位的位移量数据，与操作码一起放在代码段中。操作数一般在数据段（DS）中，这是一种默认方式。如果要对除 DS 段之外的其他段（CS、ES、SS、FS、GS）中的数据寻址，应在指令中增加段超越前缀指出段寄存器名。例如：

```
MOV     AX，[2000H]         ; 将 DS 段中 2000H 和 2001H 单元内容分别送 AL 和 AH
MOV     AX，ES:[2000H]      ; 将 ES 段中 2000H 和 2001H 单元内容分别送 AL 和 AH
```

汇编语言程序中，直接寻址中的有效地址 EA 也可用变量名形式给出。例如：

```
VALUE   DB   12H            ; 定义变量 VALUE 并赋初值
MOV     AL，VALUE           ; 将变量 VALUE 所指的字节单元内容 12H 存入 AL
```

图 3-3 是直接寻址方式的示意。

（2）寄存器间接寻址

操作数在存储区内，操作数的有效地址 EA 在指令指定的寄存器中，即 EA=（寄存器）。寄存器的使用规定在 16 位寻址和 32 位寻址时不一样。

① 16 位寻址时，EA 放在 SI、DI、BP、BX 中。这时又有两种段的默认情况。

若以 SI、DI、BX 间接寻址，则默认操作数在数据段中。例如：

```
MOV     AX, [SI]      ; 默认 DS 为段地址
```

若以 BP 间接寻址，则默认操作数在堆栈段中，提供了一种随机访问堆栈数据的方法。例如：

```
MOV     AX, [BP]      ; 默认 SS 为段地址
```

若操作数不在上述规定的默认段，则必须在指令的相应操作数前加上段超越前缀。例如：

```
MOV     CX, DS:[BP]
; “DS:”是该指令的段超越前缀
```

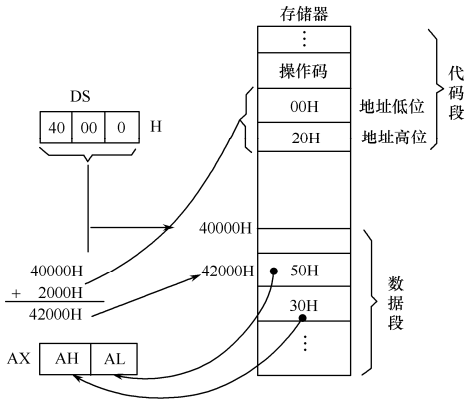


图 3-3　直接寻址方式

② 32 位寻址时，8 个 32 位通用寄存器均可作为寄存器间接寻址。例如：

```
MOV     CH, [EAX]
MOV     DX, [EBP]
```

这时，除了 EBP、ESP 默认段寄存器为 SS，其余 6 个通用寄存器均默认段寄存器为 DS。同样可以采用加段超越前缀的方法对其他段进行寻址。

由此可见，直接寻址中有效地址 EA 来自指令自身，相当于一个常量。寄存器间接寻址中有效地址 EA 来自寄存器，寄存器的内容由前面的指令确定，相当于一个变量。

（3）基址寻址

在这种寻址方式下，操作数的有效地址 EA=(基址寄存器)+位移量。式中，位移量在指令中给出并与操作码一起存放在代码段中。

① 16 位寻址时，BP 和 BX 作为基址寄存器。默认情况下，BX 以 DS 作为段寄存器，BP 以 SS 作为段寄存器，可以段超越。位移量是 8 位或 16 位。

② 32 位寻址时，8 个 32 位通用寄存器均可作为基址寄存器。其中 ESP、EBP 默认段寄存器为 SS，其余 6 个寄存器均默认段寄存器为 DS，可以段超越。位移量是 8 位或 32 位。例如：

```
MOV     AX, [BX+24]       ; 也可写成 MOV  AX, 24[BX]
MOV     ECX, [EBP+50]     ; 也可写成 MOV  ECX, 50[EBP]
MOV     DX, [EAX+TABLE]   ; 或写成 MOV  DX, TABLE[EAX]，TABLE 是符号常量或变量
```

（4）变址寻址

变址寻址方式下，有效地址 EA=(变址寄存器)+位移量。它的指令格式及寻址过程与基址寻址相同，区别仅在于将基址寄存器改成变址寄存器。

① 16 位寻址时，SI 和 DI 作为变址寄存器，且默认 DS 作为段寄存器。例如：

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

② 32 位寻址时，除 ESP 外的任何 32 位通用寄存器均可作为变址寄存器，且 EBP 以 SS 为默认段寄存器，其余以 DS 为默认段寄存器。例如：

```
MOV     EAX, 5[EBP]
MOV     ECX, DATA[EAX]
```

基址或变址寻址非常适合对一维数组元素进行检索操作，常用位移量表示数组起始地址，基址或变址寄存器表示数组元素的可变下标，通过修改基址或变址寄存器的内容可方便地访问数组中的任意元素。

（5）比例变址寻址

比例变址寻址只适合 32 位寻址的情况。有效地址 EA=(变址寄存器)×比例因子+位移量，乘以比例因子的操作是在 CPU 内部靠硬件完成的。例如：

```
MOV     EAX, TABLE[ESI*4]
```

比例变址寻址的作用和基址或变址寻址的作用相似，但更适合对多字节一维数组元素进行检索，当数组元素大小为 2、4、8 字节时，可取比例因子 2、4、8，访问更方便、高效。

（6）基址加变址寻址

基址加变址寻址方式中，操作数的有效地址 EA=(基址寄存器)+(变址寄存器)。它也有 16 位寻址和 32 位寻址两种情况。每种情况下基址、变址寄存器的使用规定和段寄存器的默认规定与前面所述相同。注意，当一种寻址方式中基址、变址寄存器默认的段寄存器不同时，一般由基址寄存器来决定默认哪一个段寄存器作为段地址。若在指令中规定了段超越，则可用其他段寄存器作为段地址。例如：

```
MOV     AX, [BX+SI]       ; 或写成 MOV  AX, [BX][SI]，BX 决定默认 DS 为段寄存器
MOV     EAX, [EDX][EBP]   ; 由 EBP 决定默认 SS 为段寄存器
```

基址加变址寻址适合检索二维数组元素和双重循环等操作。

（7）基址加比例变址寻址

这种寻址方式只适合 32 位寻址的情况。有效地址 EA=(基址寄存器)+(变址寄存器)×比例因子。例如：

```
MOV     ECX, [EDX*4][EAX]         ; 或  MOV    ECX, [EDX*4+EAX]
MOV     AX, [EBX*8][ESI]          ; 或  MOV    AX, [EBX*8+ESI]
```

基址加比例变址寻址适合检索多字节二维数组元素等操作。

（8）带位移量的基址加变址寻址

带位移量的基址加变址寻址方式下，有效地址 EA=(基址寄存器)+(变址寄存器)+位移量。它也分 16 位寻址和 32 位寻址两种情况。基址、变址寄存器的使用规定和对段寄存器的默认规定与前面所述相同。例如：

```
MOV     AX, [BX+SI+MASK]          ; 或  MOV    AX, MASK[BX][SI]
```

带位移量的基址加变址寻址也适合对二维数组操作，位移量即为数组起始地址。

（9）带位移量的基址加比例变址寻址

这种寻址方式用到有效地址的全部 4 个分量，EA=(基址寄存器)+(变址寄存器)×比例因子+位移量。与另外两种含比例因子的寻址方式一样，它只有 32 位寻址一种情况。各种规定和默认情况同前所述。寻址过程中，变址寄存器内容乘以比例因子的操作在 CPU 内部由硬件完成。例如：

```
MOV     AX, [EDI*8+ECX+40]        ; 或  MOV    AX, [EDI*8][ECX+40]
```

带位移量的基址加比例变址寻址适合起始地址不为零的多字节二维数组元素的检索操作。

3.2.3 80x86/Pentium 存储器寻址的段约定

进行存储器操作数访问时，指令中一般不特别指出段寄存器，这是因为对各种类型的存储器寻址，在 80x86/Pentium 中关于段和偏移地址寄存器的基本默认约定如表 3-2 所示。只要在指令中不特别说明要超越这个约定，则按基本约定来寻找操作数。

表 3-2　存储器操作数访问中关于段和偏移地址寄存器的约定

访问存储器类型	默认段寄存器	允许超越的段寄存器	偏移地址寄存器
取指令	CS	无	(E)IP
堆栈操作	SS	无	(E)SP
源串数据访问	DS	CS、ES、SS（FS、GS）	(E)SI
目的串数据访问	ES	无	(E)DI
通用数据访问	DS	CS、ES、SS（FS、GS）	偏移地址 EA
以(E)BP、(E)SP 间接寻址的指令	SS	CS、DS、ES（FS、GS）	偏移地址 EA

表 3-2 说明，除了程序只能在代码段、堆栈操作数只能在堆栈段、目的串操作数只能在附加数据段，其他约定都是允许段超越的。

3.2.4　几种处理器寻址方式比较

总结 80x86/Pentium CPU 指令系统的寻址方式，立即寻址与寄存器寻址是最好理解的，存储器寻址分为两大类，即指令中给出的是存储器操作数的绝对地址还是相对地址。

MIPS 或 ARM 等 RISC 处理器的指令条数和寻址方式少，因而指令的译码速度快，控制逻辑规整简单。MIPS 及 ARM 与 80x86/Pentium 指令在寻址方式上的重要不同就是存储器寻址，由于前者存储器操作数的地址分量少，所以它们的寻址方式较为简洁。表 3-3 给出了它们的寻址方式比较。

表 3-3　各种处理器寻址方式比较

		80x86/Pentium	MIPS	ARM
立即寻址		√	√	√
寄存器寻址		√	√	√
存储器寻址方式	直接寻址	√	√	√
	寄存器间接寻址	√		√
	基址寻址	√	√	√
	变址寻址	√		
	比例变址寻址	√		
	基址加变址寻址	√	√	
	基址加比例变址寻址	√		
	带位移量的基址加变址寻址	√		
	带位移量的基址加比例变址寻址	√		
	多寄存器寻址			√
	寄存器移位寻址			√

在 ARM 中，定义了不同工作模式下的多组通用寄存器 R0-R15，它们都可作为基址寄存器。除了与 80x86/Pentium 的一些相同寻址方式，ARM 增加了专有的多寄存器和寄存器移位寻址方式。MIPS 的所有通用寄存器都可以定义为基址寄存器或变址寄存器。例如，ARM 多寄存器寻址：

```
STMIA     R0!,{R1-R4}  ; [R0]←R1, [R0+4]←R2, [R0+8]←R3, [R0+12]←R4
                       ; 将 R1-R4 的内容送到以 R0 为首地址的连续内存区域并更新 R0
```

ARM 寄存器移位寻址：

```
ADD       R0,R1,R2,LSL #1  ; R0←R1+(R2<<1)，将 R2 左移一位后与 R1 相加，结果
```

存 R0

MIPS 基址加变址寻址：

```
ADD     $1,($2+$3)   ; R1←R1+[R2+R3]，将内存地址 R2+R3 中的值与寄存器 R1
                     ; 相加，结果保存到 R1
```

3.3 8086/8088 CPU 指令系统

Intel 8086/8088 CPU 指令系统是 80x86/Pentium 系列 CPU 的基本指令集。指令的操作数宽度是 8 位或 16 位，偏移地址宽度是 16 位。可以分成 6 个功能组：数据传送类、算术运算类、逻辑运算与移位类、串操作类、控制转移类和处理器控制类。本节着重介绍这些指令的功能与应用。为方便理解指令，需简要说明指令操作数符号，如表 3-4 所示。

表 3-4　指令操作数符号说明

符　号	意　义
OPRD，OPRD*N*	操作数；第 *N* 个操作数，如 OPRD1、OPRD2、OPRD3 等
DEST，SRC	目的操作数，源操作数
reg	通用寄存器，长度可以是 8 位或 16 位
reg8	8 位通用寄存器 AH/AL/BH/BL/CH/CL/DH/DL
reg16	16 位通用寄存器 AX/BX/CX/DX/SI/DI/BP/SP
Sreg	段寄存器 DS/CS/SS/ES（/FS/GS）
mem	存储器操作数，长度可以是 8 位或 16 位
mem8	8 位存储器操作数
mem16	16 位存储器操作数
imm	立即数，长度可以是 8 位或 16 位
imm8	8 位立即数
imm16	16 位立即数

3.3.1 数据传送类指令

数据传送是计算机中最基本最重要的一类操作，在程序中使用的比例最高。它们用于实现寄存器与寄存器、寄存器与存储器、累加器（AL/AX）与I/O端口之间的字节或字的传送，也可将立即数传送到寄存器或存储器单元中。为此，指令中必须指明操作数的源位置和目标位置。数据传送类指令有14条，分为4组，如表3-5所示。除了SAHF和POPF指令，其余指令执行后对标志位都没有影响。

1．通用数据传送指令

通用数据传送指令共5条，除了XCHG，是唯一允许以段寄存器作为操作数的指令。

（1）MOV 指令

指令格式：MOV　　OPRD1, OPRD2

功能：将源操作数传送给目标操作数，即 OPRD2→OPRD1，源操作数保持不变。OPRD1 和 OPRD2 可以是字节或字，但两者必须等长（同为字节传送或字传送）。

MOV 指令有如下一些形式：

```
MOV   reg/Sreg/mem, reg          ; reg/Sreg/mem←reg
```

表 3-5　数据传送类指令

类　别	指令功能	指令书写格式（助记符）
通用数据传送	字节或字传送	MOV　目标, 源
	字压入堆栈	PUSH　源
	字弹出堆栈	POP　目标
	字节或字交换	XCHG　目标, 源
	字节翻译	XLAT
地址传送	装入有效地址	LEA　目标, 源
	装入逻辑地址	LDS　目标, 源
	装入逻辑地址	LES　目标, 源
标志位传送	将标志寄存器 FR 低字节装入 AH	LAHF
	将 AH 内容装入 FR 低字节	SAHF
	将 FR 内容压入堆栈	PUSHF
	从堆栈弹出 FR 内容	POPF
I/O 数据传送	输入字节或字	IN　累加器, 端口
	输出字节或字	OUT　端口, 累加器

```
MOV   reg/mem, Sreg              ; reg/mem←Sreg
MOV   reg/Sreg, mem              ; reg/Sreg←mem
MOV   reg/mem, imm               ; reg/mem←imm
```

源操作数可以是通用寄存器、段寄存器、存储器以及立即操作数；目标操作数可以是通用寄存器、段寄存器（CS 除外）或存储器。各种数据传送关系如图 3-4 所示。例如：

```
MOV   AL, CH                ; 通用寄存器之间传送字节数据
MOV   DS, AX                ; 通用寄存器与段寄存器（CS 不能是目标）之间传送数据
MOV   AX, 0FF3BH            ; 立即数 FF3BH 传送到通用寄存器
MOV   AL,BUFFER             ; 存储器与通用寄存器之间传送数据，BUFFER 为字节存储单元名
MOV   DAT[BP+DI], ES        ; 段寄存器与存储器之间传送数据，DAT 为常量或字存储单元名
```

使用 MOV 指令传送数据时应注意：① 立即数和段寄存器 CS 不能作为目标操作数；② 立即数不能直接传送到段寄存器；③ 不允许在两个存储单元之间或在两个段寄存器之间直接传送数据；④ 将立即数传送到存储单元时，必须显式说明存储器操作数的宽度类型。

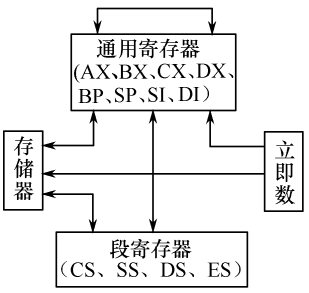


图 3-4　数据传送关系

（2）堆栈操作指令 PUSH/POP

堆栈是内存的一个数据区，按照“后进先出”原则组织的一段内存区域。80x86/Pentium CPU 规定，堆栈设置在堆栈段（SS）内，向下生长，堆栈指针 SP 始终指向堆栈的顶部。在子程序（过程）调用或处理中断时，堆栈用于保存当前的断点地址（在 8086/8088 中为 CS 和 IP）和现场数据，以便程序正确返回。断点地址的保存是由子程序调用指令或中断响应来完成的，现场数据保存可通过堆栈操作指令来实现。

堆栈操作指令有两条：入栈指令 PUSH 和出栈指令 POP。当需要在一个操作数和堆栈之间传送数据时，常使用它们。

指令格式：PUSH　　OPRD2

　　POP　　OPRD1

功能：PUSH 指令使 SP−2→SP，然后将 16 位源操作数压入堆栈。入栈时，高字节存放在高地址，低字节存放在低地址。源操作数可以是通用寄存器、段寄存器和存储器。

出栈指令 POP 的执行过程与 PUSH 相反，它从栈顶弹出 16 位操作数到目标操作数。弹出时，低地址字节送低字节，高地址字节送高字节，同时修改 SP+2→SP，使 SP 指向新的栈顶。目标操作数可以是通用寄存器、段寄存器（CS 除外）或存储器。

对 8086/8088 来说，堆栈操作总是按字（而不是字节）存取，具体入栈/出栈指令如下：

```
PUSH  reg16  ; SP=SP-2,[SP]←reg16     POP  reg16   ; reg16←[SP]，SP=SP+2
PUSH  Sreg   ; SP=SP-2,[SP]←Sreg      POP  Sreg    ; Sreg←[SP]，SP=SP+2
PUSH  mem16  ; SP=SP-2,[SP]←mem16     POP  mem16   ; mem16←[SP]，SP=SP+2
```

图 3-5 是指令 PUSH　AX 和 POP　AX 的执行示例。

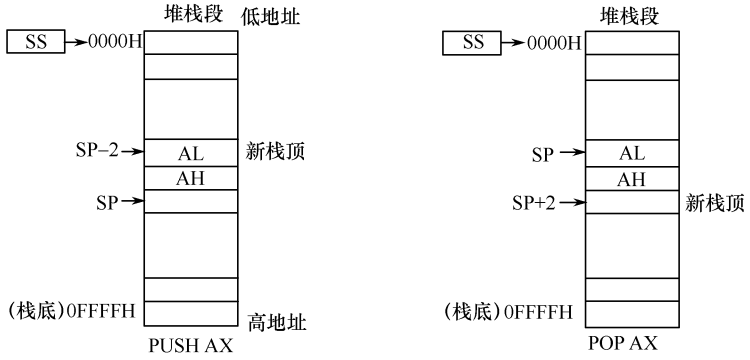


图 3-5　堆栈操作

（3）交换指令 XCHG

指令格式：XCHG　　OPRD1, OPRD2

功能：将长度相同（同为字节或同为字）的源操作数与目标操作数进行交换。

具体指令如下：

```
XCHG  reg/mem, reg        ; reg/mem←→reg
XCHG  reg, mem            ; reg←→mem
```

例如：

```
MOV   AX, 2244H           ; AX=2244H
MOV   BX, 3366H           ; BX=3366H
XCHG  AX, BX              ; 交换后，AX=3366H，BX=2244H
```

注意：① 段寄存器和立即数不能作为操作数；② 不能在两个存储单元之间直接交换数据。

（4）查表转换指令 XLAT（或称为换码指令）

指令格式：XLAT　或　XLAT　　OPRD　　; AL←[BX+AL]

功能：完成 1 字节的查表转换，将数据段中偏移地址为(BX+AL)的存储单元的内容送入 AL 寄存器，即 DS:[BX+AL]→AL。

XLAT 指令对于一些无规律的代码转换（换码）特别方便。使用时，先在数据段建立一个长度小于 256 字节的表格，表的首地址置于 BX 中，AL 中存放查找对象在表中的下标。指令执行后，所查找的对象存于 AL 中，BX 内容保持不变。

例如，将数字 0～9 的 BCD 码转换为 7 段 LED 显示器的显示代码。

数字 0～9 的 BCD 码对应的 7 段 LED 显示代码为 40H，79H，24H，30H，19H，12H，

02H，78H，00H，18H，将它们依次存放在数据段中偏移地址为 0800H 开始的区域。若待转换的 BCD 码为 0100B，则实现代码转换的程序段为：

```
MOV   BX, 0800H
MOV   AL, 4
XLAT                        ; AL 中为 0100B 对应的 7 段代码 19H
```

XLAT 指令的操作数是隐含的，汇编语言程序中可指定表的首地址以方便阅读。表的首地址名称仅供汇编程序使用，对指令的执行没有任何影响。该指令还允许段超越，如 ES:XLAT。

2．地址传送指令

地址传送指令用于传送存储器的逻辑地址信息（见表 3-5），指令中的源操作数只能是存储器操作数。

（1）有效地址传送指令 LEA

指令格式：LEA　　OPRD1, OPRD2

功能：将源操作数在当前段内的有效地址（即偏移地址）传送至目标操作数。

具体指令为：

```
LEA     reg16, mem          ; reg16←Addr(mem)
```

注意：这条指令同 MOV 指令的区别是，MOV 指令传送操作数的内容，而 LEA 传送的是操作数的存储地址。例如：

```
MOV     DI, TABLE           ; 将变量 TABLE 的内容传送至 DI
LEA     DI, TABLE           ; 将变量 TABLE 的偏移地址传送至 DI
LEA     DX, [1000H]         ; 将 1000H 单元的偏移地址→DX，DX=1000H
```

（2）地址指针传送指令 LDS 和 LES

指令格式：LDS/LES　　OPRD1, OPRD2

这两条指令的功能类似，都是将由源操作数有效地址决定的双字存储单元中的第一个字的内容送入指令指定的 16 位通用寄存器，第二个字的内容传送给段寄存器 DS 或 ES。通常，双字存储单元中是一个 32 位的地址指针。

具体指令为：

```
LDS（LES）   reg16, mem      ; reg16←[EA]，DS(ES)←[EA+2]
```

例如：

```
TABLE   DD  12345678H
LDS     BX, TABLE           ; BX←5678H，DS←1234H
```

3．标志位传送指令

CPU 通过这类指令对标志寄存器（FR）进行保护与更新。指令的操作数用隐含方式规定，指令共 4 条，见表 3-5。

（1）标志寄存器读/写指令 LAHF/SAHF

指令 LAHF 用于将标志寄存器的低字节（含 SF、ZF、AF、PF、CF）读出后传送给 AH 寄存器。这条指令本身不影响标志位。

标志寄存器写指令 SAHF 与 LAHF 的操作相反。它把寄存器 AH 中的内容写入标志寄存器的低字节，取代某些标志位（SF、ZF、AF、PF、CF）的原来状态。

（2）标志寄存器入栈/出栈指令 PUSHF/POPF

PUSHF 指令的执行过程同 PUSH 指令，只是将标志寄存器 FR 的内容压入堆栈。POPF 则将当前栈顶的一个字传送给标志寄存器 FR，同时修改栈顶指针 SP。

在子程序调用和中断服务中，常用这两条指令保护和恢复需要的标志位。利用它们还可以方便地改变标志寄存器中任一位的状态。例如，8086/8088 指令系统中没有直接修改 TF 标志位的指令，可用以下程序段来实现 TF 的改变：

```
PUSHF
POP     AX                  ; 标志寄存器的内容送 AX
OR      AH, 01H             ; 将 TF 位置 1（TF 在标志寄存器的 b8 位）
PUSH    AX
POPF                        ; AX 的内容送标志寄存器
```

4．输入/输出数据传送指令 IN/OUT

指令格式：IN　　OPRD1, OPRD2

　　OUT　　OPRD1, OPRD2

这组指令专门用于在 AL 或 AX 寄存器与 I/O 端口之间传送数据。

一台计算机可以配接许多外部设备，每个外部设备与 CPU 之间要交换数据、状态信息和控制命令，每一种这样的信息交换都要通过一个端口来进行。计算机系统中的各端口也像存储器那样用地址来区分。80x86/Pentium CPU 中有 16 条 I/O 地址线，最多可提供 64K 个传送 8 位数据的端口地址或 32K 个传送 16 位数据的端口地址。当端口地址小于 256 时，采用直接寻址方式，在指令中直接指定端口地址；当端口地址大于或等于 256 时，采用间接寻址方式，端口地址放在 DX 中。

具体指令如下：

```
IN    AL, imm8   ; AL←[端口 imm8](其他同)   OUT  imm8, AL  ; [imm8] ←AL
IN    AX, imm8   ; AX←[imm8+1], [imm8]     OUT  imm8, AX  ; [imm8+1], [imm8] ←AX
IN    AL, DX     ; AL←[DX]                 OUT  DX,AL     ; [DX]←AL
IN    AX, DX     ; AX←[DX+1], [DX]         OUT  DX,AX     ; [DX+1], [DX]←AX
```

例如：

```
IN    AX, 20H             ; 从端口 20H 和 21H 输入 16 位数据到 AL 和 AH(AX)
MOV   DX, 3F0H
IN    AL, DX              ; 从端口 03F0H 输入 8 位数据到 AL
OUT   27H, AL             ; 将 8 位数据从 AL 输出到端口 27H
OUT   DX, AX              ; 将 16 位数据从 AX 输出到 DX 和 DX+1 指定的端口
```

3.3.2　算术运算类指令

算术运算类指令支持加、减、乘、除等基本算术运算，操作对象可以是字节或字的无符号和有符号的二进制整数，也可以是无符号的压缩、非压缩 BCD 数。非压缩 BCD 数的高 4 位在进行乘/除运算时必须全为 0，进行加/减运算时可以是任何值。

算术运算类指令共 20 条，如表 3-6 所示。除了数据宽度变换指令（CBW，CWD），其余指令的执行结果都影响标志位。这些标志位中的绝大多数可供条件转移指令进行测试，以改变程序的流程。掌握指令执行结果对标志位的影响，对编程有重要的作用。

1．二进制算术运算指令

（1）加/减法类指令

① 加/减法指令 ADD/SUB

指令格式：ADD　　OPRD1, OPRD2

　　SUB　　OPRD1, OPRD2

表 3-6　算术运算类指令

类　别	指令功能	指令书写格式（助记符）	状态标志位					
			OF	SF	ZF	AF	PF	CF
加法	加法	ADD　目标，源	↕	↕	↕	↕	↕	↕
	带进位加法	ADC　目标，源	↕	↕	↕	↕	↕	↕
	加 1	INC　目标	↕	↕	↕	↕	↕	—
减法	减法	SUB　目标，源	↕	↕	↕	↕	↕	↕
	带借位减法	SBB　目标，源	↕	↕	↕	↕	↕	↕
	减 1	DEC　目标	↕	↕	↕	↕	↕	—
	取负	NEG　目标	↕	↕	↕	↕	↕	①
	比较	CMP　目标，源	↕	↕	↕	↕	↕	↕
乘法	不带符号数乘法	MUL　源	↕	*	*	*	*	↕
	带符号数乘法	IMUL　源	↕	*	*	*	*	↕
除法	不带符号数除法	DIV　源	*	*	*	*	*	*
	带符号数除法	IDIV　源	*	*	*	*	*	*
	字节转换成字	CBW	—	—	—	—	—	—
	字转换成双字	CWD	—	—	—	—	—	—
十进制调整	非压缩 BCD 数的加法调整	AAA	*	*	*	↕	*	①
	压缩 BCD 数的加法调整	DAA	*	↕	↕	↕	↕	↕
	非压缩 BCD 数的减法调整	AAS	*	*	*	↕	*	↕
	压缩 BCD 数的减法调整	DAS	↕	↕	↕	↕	↕	↕
	非压缩 BCD 数的乘法调整	AAM	*	↕	↕	*	↕	*
	非压缩 BCD 数的除法调整	AAD	*	↕	↕	—	↕	*

注：↕ 运算结果影响标志位；* 标志位为任意值；— 运算结果不影响标志位；① 标志位置 1。

功能：完成两个操作数的加/减运算，结果送入目标操作数，即 OPRD1±OPRD2→OPRD1。

具体指令如下：

```
ADD   reg, reg/mem/imm        ; reg←reg + (reg/mem/imm)
ADD   mem, reg/imm            ; mem←mem + (reg/imm)
SUB   reg, reg/mem/imm        ; reg←reg – (reg/mem/imm)
SUB   mem, reg/imm            ; mem←mem – (reg/imm)
```

要求源操作数和目标操作数同为带符号数或同为无符号数，且长度相等。源操作数可以是寄存器、存储器或立即数，目标操作数只能是寄存器或存储器，且两个操作数不能同时为存储器。运算影响全部状态标志位。

例如，设 AX=65A0H，BX=B79EH，则指令 ADD　　BX, AX 执行后，BX=1D3EH，CF=1，SF=0，OF=0，AF=0，PF=0，ZF=0。

结果表明，若 BX 和 AX 中是无符号数，则运算结果大于 65535（CF=1）；若 BX 和 AX 中是有符号数，则运算结果在 16 位补码所能表示的范围内（–32768～+32767），运算结果正确（OF=0，SF=0）。

若 AX、BX 中存放的数和上例相同，则指令 SUB　AX, BX 执行后，AX=AE02H，CF=1，SF=1，OF=1，AF=1，PF=0，ZF=0。

结果表明，若两个操作数是无符号数，CF=1 表示不够减，运算结果是以 2^{16} 为模的差值的补码；若两个操作数是有符号数，则运算结果超出 16 位补码所能表示的范围（OF=1），AX 中存放的数（AE02H）并不是正确的运算结果。

② 带进位、借位的加/减法指令 ADC/SBB

指令格式：ADC　OPRD1, OPRD2

SBB　OPRD1, OPRD2

ADC/SBB 指令常与 ADD/SUB 配合，用于多精度数间的运算。除了加法操作时要加上进位位或减法操作时再减去借位位，它们同 ADD/SUB 指令的使用注意事项和对标志位的影响情况相同。

具体指令如下：

```
ADC  reg, reg/mem/imm              ; reg←reg + (reg/mem/imm) + CF
ADC  mem, reg/imm                  ; mem←mem + (reg/imm) + CF
SBB  reg, reg/mem/imm              ; reg←reg – (reg/mem/imm) – CF
SBB  mem, reg/imm                  ; mem←mem – (reg/imm) – CF
```

例如，有两个 32 位数（多精度数）分别存放在自 FIRST 和 SECOND 开始的存储区中，变量定义如下（DW 是汇编语言伪指令，它为变量 FIRST、SECOND 按字分配存储单元）：

```
FIRST    DW  2211H, 4433H          ; 定义变量 FIRST=44332211H
SECOND   DW  6655H, 8877H          ; 定义变量 SECOND=88776655H
```

用字相加指令实现两个变量的和，并将和存放于 FIRST 变量的程序段为：

```
MOV  AX, SECOND            ; 取第二个加数的低 16 位
ADD  FIRST, AX             ; 与第一个加数的低 16 位相加并存回和，如有进位，则 CF=1
MOV  AX, SECOND+2          ; 取 32 位数据 SECOND 的高 16 位
ADC  FIRST+2, AX           ; 将两个数的高 16 位相加并加上 CF，和存回 FIRST 的高 16 位
```

③ 增量和减量指令 INC/DEC

指令格式：INC　OPRD

DEC　OPRD

功能：INC/DEC 是单操作数指令，完成对指定操作数的加 1/减 1，然后返回运算结果。

具体指令如下：

```
INC   reg        ; reg←reg+1          DEC   reg        ; reg←reg-1
INC   mem        ; mem←mem+1          DEC   mem        ; mem←mem-1
```

设计这两条指令的主要目的是调整地址指针和计数器（循环程序的循环次数），所以它们不影响 CF 标志（即保持此指令之前的值）。对其他 5 个标志位的影响与前述加/减法指令一样。指令中的操作数可以在通用寄存器或内存中。例如：

```
INC   CX                           ; CX 寄存器中的内容加 1
DEC   BYTE  PTR [BX]               ; 将数据段中以 BX 指向的字节存储单元内容减 1
DEC   WORD  PTR [BX]               ; 将数据段中以 BX 指向的字存储单元的内容减 1
```

④ 求补指令 NEG

指令格式：NEG　OPRD

功能：对操作数求补，即用零减去操作数，再把结果送回操作数。求补运算也可以表达成：将操作数按位求反加 1。指令执行的效果是改变操作数的符号，但绝对值不变，所以

又称为取负指令。具体指令如下：

```
NEG   reg                    ; reg←0-reg
NEG   mem                    ; mem←0-mem
```

NEG 指令影响所有状态标志位，不过对 CF 标志的影响为：若操作数不是 0，则总是使 CF 置 1；否则将 CF 清 0。若在字节操作时对−128、或在字操作时对−32768 求补，则操作数没有变化，但溢出标志 OF 置位。例如：

```
NEG   AL                     ; 这两条指令实现(100-AL)运算
ADD   AL, 100
```

⑤ 比较指令 CMP

指令格式：CMP　　OPRD1, OPRD2

比较指令 CMP 计算 OPRD1−OPRD2，但运算结果不送回 OPRD1，即 CMP 指令与减法指令 SUB 执行同样的操作，同样影响标志位，只是不改变操作数本身。参加比较的源操作数可以是立即数、寄存器或存储器操作数，目标操作数只能是寄存器或存储器操作数。

具体指令为：

```
CMP   reg, reg/mem/imm       ; reg-(reg/mem/imm)
CMP   mem, reg/imm           ; mem-(reg/imm)
```

比较指令用于比较两个数的大小关系，执行比较指令后，可以根据受影响的标志位状态来判断两个操作数是否相等、谁大、谁小。所以 CMP 指令后面常跟条件转移指令，根据比较结果产生不同的分支转移。判断无符号数和有符号数大小的条件有所不同，列举如下。

- 如果是两个无符号数比较，可根据比较结果的进位标志 CF 的状态来判断：若 CF=1，则 OPRD1<OPRD2；若 CF=0，则 OPRD1>OPRD2。
- 如果是两个有符号数比较，则根据 SF 和 OF 两个标志位的关系来判断：若 SF⊕OF=1，则 OPRD1<OPRD2；若 SF⊕OF=0，则 OPRD1>OPRD2。
- 不管操作数有无符号，若在比较指令后，ZF=1，则 OPRD1=OPRD2，否则不等。

所以，条件转移指令也按无符号数和有符号数分为两类（详见表 3-11）。

（2）乘/除法指令

① 乘法指令 MUL/IMUL

指令格式：MUL　　OPRD

　　IMUL　　OPRD

MUL 和 IMUL 指令分别用于实现无符号数的乘法和有符号数的乘法运算。它们都只有一个源操作数，源操作数可以是寄存器或存储器，而目标操作数隐含规定在累加器中。

具体的指令如下：

```
MUL   reg    或   IMUL reg      ; AX←AL × reg8（或 DX,AX←AX × reg16）
MUL   mem    或   IMUL mem      ; AX←AL × mem8（或 DX,AX←AX × mem16）
```

MUL 指令用于无符号二进制数乘法运算。若为字节（OPRD 为字节）乘法，则 AL×OPRD，乘积存于 AX 中；若为字（OPRD 为字）乘法，则 AX×OPRD，乘积的高 16 位存放在 DX 中、低 16 位存放在 AX 中。当乘积的高半部分（字节相乘时为 AH，字相乘时为 DX）不为 0，则标志 CF=OF=1，表示在 AH 或 DX 中存有乘积的有效数值；否则，CF=OF=0。而其他状态标志位的内容不确定。

IMUL 指令除了完成两个有符号数的相乘，其他与 MUL 完全类似。若乘积的高半部分（字节相乘时为 AH，字相乘时为 DX）存有积的有效数值（不光是符号部分），则 CF=OF=1；

否则 CF=OF=0，表示高半部分全部是低半部分符号位的扩展。

由于乘法指令为乘积保留了 2 倍于原来操作数的存储空间，因而不会出现溢出现象。

② 除法指令 DIV/IDIV

指令格式：DIV　　OPRD

　　IDIV　　OPRD

DIV 和 IDIV 分别用来实现无符号数的除法和有符号数的除法运算。指令中对操作数的规定同乘法指令，除法指令执行后，所有的状态标志位都不确定。

具体的指令如下：

```
DIV   reg    或   IDIV   reg        ; AH,AL←AX/reg8（或 DX,AX←DX:AX/reg16）
DIV   mem    或   IDIV   mem        ; AH,AL←AX/mem8（或 DX,AX←DX:AX/mem16）
```

DIV 用于无符号数的除法运算。对字节除法，被除数在 AX 中，除数即 OPRD（字节），商在 AL 中，余数在 AH 中；对字除法，被除数高位在 DX 中、低位在 AX 中，除数即 OPRD（字），商在 AX 中，余数在 DX 中。若商超过存放它的寄存器的容量（字节相除时为 FFH，字相除时为 FFFFH），则引起 0 型中断（即除法出错中断），且商和余数为不定值。

IDIV 除了完成有符号数相除，其余与 DIV 完全类似。商的符号根据代数符号的规则确定，余数的符号与被除数的相同。在字节相除时，最大的商是+127（7FH），最小的负数商是−127（81H）；在字相除时，最大的商为 32767（7FFFH），最小的负数商是−32767（8001H）。若相除以后，商超出上述讨论值的范围，则商和余数的值不能确定，并产生一个 0 型中断。

注意：在除法运算中，要求被除数的长度为除数长度的 2 倍。若被除数和除数是具有相同位数的字节或字，这时需要先将被除数扩展为字或双字，且大小和符号不变，再相除。

2．符号扩展指令 CBW/CWD

指令格式：CBW

　　CWD

功能：CBW 和 CWD 分别用于将有符号操作数从字节转换成字或从字转换为双字，操作数隐含规定在累加器中。CBW 将 AL 中的字节符号数的符号扩展到 AH 中，CWD 将 AX 中的字符号数的符号扩展到 DX 中。CBW 和 CWD 指令都不影响任何标志位。

扩展规则如下：① 若 AL<80H，则 AH←0；若 AL≥80H，则 AH←FFH；② 若 AX<8000H，则 DX←0；若 AX≥8000H，则 DX←FFFFH。

例如，求 0BF4H÷0100H（有符号数相除）。由于除数为字，则必须将被除数进行符号扩展后才能相除。程序段如下：

```
MOV    AX, 0BF4H
CWD                                ; 被除数扩展为 DX:AX=0000 0BF4H
MOV    BX, 0100H
IDIV   BX                          ; AX←商 0BH，DX←余数 F4H
```

注意：对无符号数，可采用高位直接置“0”的方法进行扩展。

3．BCD（十进制）运算调整指令

CPU 中的 ALU 单元只能执行二进制数的算术操作。对于 BCD 数的算术运算，可以直接使用二进制数的算术运算指令，但必须在二进制算术运算指令后，紧接着用一条调整指令来加以校正调整。所有调整指令都隐含操作数在累加器中，对标志位的影响如表 3-6 所示。

（1）非压缩 BCD 数加/减法调整指令 AAA/AAS

AAA/AAS 用于将 AL 中的由两个非压缩 BCD 数相加/减后的结果进行调整，以得到正

确的非压缩 BCD 和/差。程序中，它们紧跟在 ADD（或 ADC）/SUB（或 SBB）指令后。

AAA/AAS 指令完成如下调整操作：若(AL&0FH)>9 或标志 AF=1，则 AL←AL±6，AH←AH±1，AF←1，CF←AF，AL←AL&0FH（“-”对指令 AAS 而言）。

（2）压缩 BCD 数加/减法调整指令 DAA/DAS

DAA/DAS 用于将 AL 中两个压缩 BCD 数相加/减的结果进行校正，以得到正确的压缩 BCD 和/差。程序中，它们紧跟在 ADD（或 ADC）/SUB（或 SBB）指令后。

DAA/DAS 指令完成如下调整操作：若(AL&0FH)>9 或 AF=1，则 AL←AL±6，AF←1（“-”对 DAS 而言）；若 AL>9FH 或 CF=1，则 AL←AL±60H，CF←1（“-”对 DAS 而言）。

例如，AL=47H（十进制数 47 的 BCD 码），BH=25H（十进制数 25 的 BCD 码），用下列指令可得到这两个数和的正确结果。

```
ADD   AL，BH              ；AL←AL+BH
DAA                       ；将 AL 的内容调整为 72H
```

注意，所有调整指令仅用来对一次加/减运算的结果 AL 进行调整。因而对多位压缩或非压缩的 BCD 数的运算，必须由低位向高位逐个进行调整。

【例 3-1】 计算$(8576)_{10}+(2695)_{10}$。假设两个操作数分别存于 AX、BX，即 AX=8576H，BX=2695H，则下列程序段将实现两个数的求和运算，结果放在 AX 中。

```
ADD   AL，BL              ；低位相加
DAA                       ；结果调整并存于 AL 中
MOV   DL，AL              ；暂存低字节压缩 BCD 数在 DL 中
MOV   AL，AH
ADC   AL，BH              ；高位相加
DAA                       ；结果调整并存于 AL 中
MOV   AH，AL              ；高字节压缩 BCD 数在 AH 中
MOV   AL，DL              ；压缩 BCD 数的和在 AX 中
```

（3）非压缩 BCD 数乘/除法调整指令 AAM/AAD

AAM 紧跟在 MUL 指令后，将 AX 中两个 1 位非压缩 BCD 数相乘的结果进行调整，得到正确的非压缩 BCD 的乘积（高位在 AH 中，低位在 AL 中）。调整操作为：

```
AH ← AX/0AH               ；AX 被 0AH 除的商→AH
AL ← AX % 0AH             ；AX 被 0AH 除的余数→AL
```

例如：

```
MOV   AL，07H
MOV   BL，09H
MUL   BL                  ；执行乘法指令后 AX=003FH
AAM                       ；执行乘法调整指令后 AX=0603H
```

AAD 指令用来在进行两个非压缩 BCD 数的除法运算之前，先调整 AL 和 AH 中的内容，再用二进制除法指令 DIV 相除。相除以后，以非压缩 BCD 数表示的商在 AL 中，相应的余数在 AH 中。调整操作为：

```
AL←AH×0AH+AL，AH←0
```

例如：设 AX=0103H，BL=06H（分别表示非压缩 BCD 数 13 和 6）。

```
AAD                       ；先调整得 AX=000DH
DIV   BL                  ；执行除法指令后，商 AL=02H，余数 AH=1
```

3.3.3 逻辑运算与移位指令

为了处理字节或字中各位的信息，8086/8088 CPU 提供了一组位处理指令，包括逻辑运算、移位和循环移位指令三部分。

1．逻辑运算指令

逻辑运算指令列于表 3-7 中。所有的指令都对操作数进行按位操作，操作数可以是字节或字。目标操作数不能是立即数；当有两个操作数时，不能两个操作数都为存储器操作数；无论是目标操作数还是源操作数都不能是段寄存器。

表 3-7　逻辑运算类指令

类　别	指令功能	指令书写格式（助记符）	状态标志位					
			OF	SF	ZF	AF	PF	CF
逻辑运算	非（字/字节）	NOT　目标	—	—	—	—	—	—
	与（字/字节）	AND　目标，源	0	↕	↕	*	↕	0
	或（字/字节）	OR　目标，源	0	↕	↕	*	↕	0
	异或（字/字节）	XOR　目标，源	0	↕	↕	*	↕	0
	测试（字/字节）	TEST　目标，源	0	↕	↕	*	↕	0

注：↕ 运算结果影响标志位；— 运算结果不影响标志位；0 标志位置‘0’；* 标志位为任意值。

逻辑操作指令主要用于，根据源操作数中的位组合格式有选择地使目标操作数的某些位置位、复位，或对其进行测试等。

在这组指令中，仅 NOT 指令不影响标志位。其他指令执行后，除 AF 状态不确定外，总是使 OF=CF=0。ZF、PF、SF 根据运算结果被置位或复位，以反映操作结果的特征。

（1）逻辑非指令 NOT

指令格式：NOT　　OPRD

功能：将操作数“按位取反”，然后送回原处。

具体指令为：

```
NOT    reg/mem
```

例如：

```
NOT    AX                    ; 对 AX 内容按位取反并送回 AX
```

（2）逻辑与/或/异或指令 AND/OR/XOR

指令格式：AND/OR/XOR　　OPRD1, OPRD2

功能：AND、OR、XOR 指令执行按位逻辑“与”“或”和“异或”操作。它们均为双操作数指令，两个操作数宽度必须相等（同为字节或字）。执行结果存入 OPRD1 中。

具体指令为：

```
AND/OR/XOR     reg, reg/mem/imm
AND/OR/XOR     mem, reg/imm
```

AND 指令用于对某些位清 0，其余位保持不变（要清 0 的位和 0“与”）；OR 指令用于对某些位置 1，其余位保持不变（要置 1 的位和 1“或”）；XOR 指令使某些位取反，其余位保持不变（要取反的位与 1“异或”）。

特别地，当某个操作数自己和自己相“与”、自己和自己相“或”时，可使 CF 清 0 而

操作数保持不变；当操作数自己与自己相“异或”时，可以将操作数和 CF 同时清 0。例如：

```
        XOR   AL, AL          ; 清 AL 寄存器和 CF 位
        MOV   AL, '8'         ; 8 的 ASCII 值送 AL，AL=38H
        AND   AL, 0FH         ; 执行后，AL=08H（AL 的高 4 位被屏蔽，低 4 位被析取）
```

（3）测试指令 TEST

指令格式：TEST　OPRD1, OPRD2

若希望在不改变原有操作数的情况下检测操作数的某一位或某几位是 1 还是 0，可使用 TEST 指令。TEST 指令的功能与 AND 指令相同，但操作结果不送回目标操作数，只根据结果设置标志位。这条指令之后一般都是条件转移指令，利用测试结果转向不同的程序段。

具体指令为：

```
        TEST  reg, reg/mem/imm
        TEST  mem, reg/imm
```

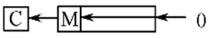
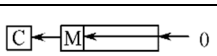
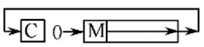
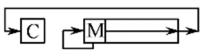
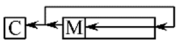
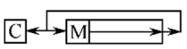
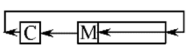
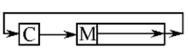
例如，检测 AL 中最低位是否为 1，为 1 则转移至 THERE，可使用以下指令。

```
        TEST  AL, 01H         ; 将源操作数设置为一个立即数，其中需要测试的位置 1，其余位为 0
        JNZ   THERE           ; ZF=0，则 AL 寄存器的第 0 位为 1，转移到 THERE
        …                     ; ZF=1，表明 AL 寄存器的第 0 位为 0，顺序执行
THERE:  …
```

2．移位和循环移位指令

移位和循环移位指令可以对字节或字中的各位进行算术移位、逻辑移位或循环移位，指令及其对应的操作功能如表 3-8 所示。

表 3-8　移位与循环移位指令

类别	指令功能	指令书写格式（助记符）	操作功能	状态标志位					
				OF	SF	ZF	AF	PF	CF
移位	逻辑左移	SHL　目标，计数值		↕	↕	↕	*	↕	↕
	算术左移	SAL　目标，计数值		↕	↕	↕	*	↕	↕
	逻辑右移	SHR　目标，计数值		↕	↕	↕	*	↕	↕
	算术右移	SAR　目标，计数值		↕	↕	↕	*	↕	↕
循环移位	循环左移	ROL　目标，计数值		↕	—	—	*	—	↕
	循环右移	ROR　目标，计数值		↕	—	—	*	—	↕
	带进位循环左移	RCL　目标，计数值		↕	—	—	*	—	↕
	带进位循环右移	RCR　目标，计数值		↕	—	—	*	—	↕

指令中的目标操作数只能是寄存器或存储器操作数，可以是字节也可以是字。指令中的计数值决定移位或循环移位的次数，计数值或者是 1，或者是 CL 中设定的次数。

（1）移位指令 SHL/SAL/SHR/SAR

移位指令包括逻辑移位与算术移位。从表 3-8 可以看出，逻辑左移与算术左移操作在物理上完全相同，每移位 1 次目标操作数按位依次左移 1 位，最高位进入 CF，右端补 0。逻

辑右移 1 次使操作数的左端移入 0，算术右移 1 次则将最高位复制 1 位以保持符号位不变，移出的位都进入 CF。

移位指令影响 PF、SF、ZF、CF、OF 这 5 个标志位。其中，CF 总是等于目标操作数最后移出的那一位。若移位计数值=1，且执行结果使目标操作数的符号位发生变化，则 OF=1，否则 OF=0；若移位计数值>1，则 OF 状态不确定。移位操作对 AF 状态的影响不确定。

具体指令为：

```
SHL/SAL/SHR/SAR    reg，1/CL
SHL/SAL/SHR/SAR    mem，1/CL
```

移位指令用于将字节或字的某些位分离出来，或用来对二进制数进行 2 的方幂运算。左移 1 次，只要左移之后的数值未超出 1 字节或字的表达范围，则原数的每 1 位的权增加一倍，相当于原数乘以 2；右移 1 位相当于除以 2。若需要将有符号数乘以或除以 2^n，则使用算术移位指令；若需要将无符号数乘以或除以 2^n 时，则使用逻辑移位指令。

【例 3-2】 将 AL 中的一个 8 位有符号数乘以 8（2^3），结果存入 AX 中，指令序列为：

```
CBW                                ; 将字节 AL 扩展到字 AX
MOV   CL，3                        ; 移动 3 次，需要将 3 放入 CL
SAL   AX，CL（或 SHL   AX，CL）    ; AX←AX×8
```

【例 3-3】 下列指令将存于 AH 和 AL 中的非压缩 BCD 数转换成压缩的 BCD 数：

```
MOV   CL，4
SHL   AL，CL
SHR   AX，CL                       ; 转换结果在 AL 寄存器中
```

（2）循环移位指令 ROL/ROR/RCL/RCR

循环移位指令有带进位位、不带进位位两种。带进位位的循环移位指令把 CF 标志作为目标操作数的扩展，参与循环操作。与移位指令不同的是，从操作数一端移出来的位“循环地”进入该操作数的另一端。循环移位指令只影响 CF 和 OF 两个标志位，CF 只存放最后一次循环移出的那一位的值，OF 状态的变化规则同移位指令。

具体指令为：

```
ROL/ROR/RCL/RCR    reg，1/CL
ROL/ROR/RCL/RCR    mem，1/CL
```

【例 3-4】 有一个 4 字节（32 位）的数，它们或是存放在两个寄存器中（如 DX 和 AX 中），或是存放在连续的内存单元中，用下列指令可实现这个 4 字节数整体左移 1 位。

```
SAL  AX，1       或       SAL  FIRST_WORD，1
RCL  DX，1       或       RCL  SECOND_WORD，1
```

【例 3-5】 内存中 4 位十进制数以压缩 BCD 码形式存放在 DA1 开始的两个单元，以下程序段实现它们乘以 10 的运算，结果存放在 DA2 开始的内存单元。

```
        MOV   AX，DA1                  ; 取 4 位十进制数→AX
        XOR   DL，DL                   ; 乘积的最高位 DL 清 0
        MOV   CX，4                  ; 4 位十进制数乘 10 就是左移 4 次，CX 中为移位次数
LP:     SHL   AX，1                  ; DL、AX 整体左移 4 次
        RCL   DL，1
        DEC   CX                     ; 移位次数-1
        JNZ   LP                     ; CX≠0，转移到 LP 处继续移位；CX=0，移位结束
        MOV   DA2，AX                ; 保存结果
```