第3章 80X86的指令系统和寻址方式

CHAPTER

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。

对于一台计算机，它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：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部分；另一部分是要给出参与操作的对象是什么，此部分称为操作数部分，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。

本章主要介绍80X86的指令系统和寻址方式，在介绍指令系统和寻址方式前，首先介绍80X86指令系统中常用的数据类型。

本章的主要内容如下：

- 数据类型；
- 80X86的寻址方式；
- 80X86的指令系统。

3.1 数据类型

计算机执行指令过程中需要处理各种类型的数据，80X86微机在其内部定点处理单元微处理器和浮点单元FPU的支持下，可处理以下7种类型的数据。

1. 无符号二进制数

指不带符号位的一个字节、字或双字的二进制数。这类数由微处理器支持。

2. 带符号二进制数

此类数有正、负之分,均以补码表示。有8位数(字节)、16位数(字)、32位数(双字)、64位数(4字)四种。微处理器仅支持8位、16位和32位带符号整数。

3. 浮点数

80X86中的浮点数由符号位、尾数和阶码(即指数部分)三个字段组成。浮点数由FPU支持,分为单精度(32位)、双精度(64位)和扩展精度(80位)三种形式。

4. BCD码

有非压缩BCD码和压缩BCD码两种,前者用一个字节表示一个0～9的十进制数,这些数为无符号数并按字节存放,数的大小由低半字节确定;后者用一个字节来表示两个0～9十进制数。高半字节为高位,每半个字节中的取值只能是0～9。微处理器只支持8位非压缩BCD码和压缩BCD码,FPU只支持压缩BCD码,且最大长度为80位(10个字节),最多可处理20位BCD码。

5. 串数据

- 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。

32位微处理机中可处理的串数据最大长度可达$2^{32}-1$字节。

6. ASCII码数据

包括ASCII码字符串和ASCII码数串(0～F)两种。

7. 指针类数据

包括近程指针和远程指针。前者为32位的偏移量,用于段内的寻址;后者由一个16位段选择符值和一个32位的偏移量组成,用于段间数据访问和段间转移。

另外,在80X86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,字不必定位于偶数地址,双字也不一定定位于能被4整除的地址,即允许不按2的整次边界对齐。但是,当使用有32位总线的系统配置时,处理器和存储器之间的数据传输的最大长度为双字。对于对齐的双字可以一次传送,而未对齐的双字则需要两次传送。为了获得最佳性能,可将字操作对齐于偶地址,双字操作对齐于能被4整除的地址。

3.2 80X86的寻址方式

80X86微机中,存储器中既存放数据也存放程序,也就是指令。80X86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程序中出现转移和调用时的程序定位方式。

3.2.1　数据寻址方式

80X86 指令中所需的操作数来自以下几个方面。

(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,这种操作数被称为立即数。

(2) 操作数包含在微处理器的某个内部寄存器中。

(3) 操作数包含在存储器中。

在 80X86 微机系统中,任何内存单元的地址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,而偏移地址由以下四个基本部分组合而成。

① 基址寄存器;

② 变址(间址)寄存器;

③ 比例因子;

④ 位移量。

这四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 EA＝基址＋变址×比例因子＋位移量

按照第 2 章所介绍的 80X86 系统的存储器组织方式,指令中不能使用实际地址,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 EA 后,可由有效地址和段首址形成逻辑地址。

当采用 16 位寻址方式时, 80X86 使用 8 位或 16 位的位移量,用 BX 或 BP 作基址寄存器,SI 和 DI 变址(间址)寄存器,比例因子为 1;当采用 32 位寻址方式时, 80X86 使用 8 位或 32 位的位移量,所有 32 位通用寄存器都可以用作基址寄存器,除 ESP 以外的所有通用寄存器都可以用作变址(间址)寄存器,并且可采用 2、4 或 8 几种不同的比例因子。

这四种元素可优化组合出 9 种存储器寻址方式。这种寻址计算方法如图 3.1 所示。下面我们结合 80X86 的指令简要介绍相应的寻址方式。

1. 直接寻址

这种寻址方式的位移量就是操作数的有效地址,位移量直接包含在指令中,它与操作码一起存放在代码段区域。例如:

```
INC WORD PTR [500];
```

该指令的有效地址为 500,它的线性地址＝数据段基地址＋500。

直接寻址一般多用于存取某个存储器单元中的操作数,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器单元。

2. 寄存器间接寻址

与直接寻址方式的区别是,这种寻址方式不直接给出操作数的有效地址,而是由寄存

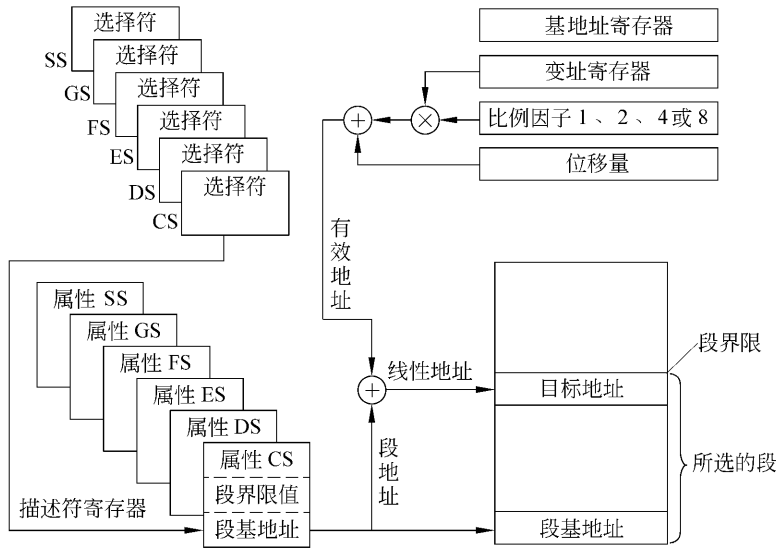

图 3.1 寻址计算图解

器给出有效地址的指针,即有效地址是基址或变址寄存器中的内容。例如:

```
MOV [ECX],EDX;ECX 中的内容为操作数的有效地址
```

操作数的逻辑地址(即线性地址)=数据段基地址+ECX 中的内容。

3. 基址寻址

基址寄存器的内容,加上指令格式中的位移量而形成操作数的有效地址。例如:

```
MOV ECX,[EAX+24];由 EAX 中内容加位移量 24 组成操作数的有效地址
```

操作数的逻辑地址(即线性地址)=数据段基地址+这个有效地址。

4. 变址寻址

变址寻址方式与基址寻址方式很相似,其有效地址的形成是变址寄存器的内容加上指令格式中的位移量。例如:

```
ADD EAX,TABLE[ESI];ESI 中的内容加 TABLE 变量的地址组成操作数的有效地址
```

操作数的逻辑地址(即线性地址)=数据段基地址+这个有效地址。

5. 带比例因子的变址寻址

是变址寻址方式的另一种寻址方式,指操作数的有效地址等于变址寄存器内容乘以比例因子再加上指令格式中的位移量。例如:

```
IMUL EBX,TABLE [ESI×4],7;ESI 中的内容乘以 4 再加 TABLE 变量的地址形成有效地址
```

操作数的逻辑地址(即线性地址)=数据段基地址+这个有效地址。

6. 基址变址寻址

操作数的有效地址等于基址寄存器的内容加变址寄存器的内容。例如：

```
MOV EAX,[ESI][EBX];EBX 中的内容加 ESI 中的内容为操作数的有效地址
```

操作数的逻辑地址(即线性地址)＝数据段基地址＋这个有效地址。

7. 基址、带比例因子的变址寻址

操作数的有效地址等于变址寄存器的内容乘比例因子再加基址寄存器的内容。例如：

```
MOV ECX,[EDI×4][EAX];EDI 中的内容乘以 4 再加 EAX 中的内容形成操作数的有效地址
```

操作数的逻辑地址(即线性地址)＝数据段基地址＋这个有效地址。

8. 带位移量的基址变址寻址

操作数的有效地址等于基址寄存器的内容加位移量后再与变址寄存器的内容相加所形成的地址。例如：

```
ADD EDX,[ESI][EBP+00FFFFF0H]   ; ESI 的内容加 EBP 的内容再加 00FFFFF0H,即为操作数的
                               ; 有效地址
```

操作数的逻辑地址(即线性地址)＝数据段基地址＋这个有效地址。

9. 带位移量、带比例因子变址基址寻址

指操作数的有效地址等于变址寄存器内容乘以比例因子,加基址寄存器内容,再加位移量。例如：

```
MOV EAX,[EDI×4][EBP+80];EDI 的内容乘 4,加 EBP 的内容,再加 80 即为操作数的有效地址
```

操作数的逻辑地址(即线性地址)＝数据段基地址＋这个有效地址。

除了上述 9 种存储器寻址方式外,还有两种非存储器寻址的简单寻址方式：

立即寻址：操作数以立即数形式出现在指令中。

寄存器寻址：指操作数存放在 32 位、16 位或 8 位的通用寄存器中。

32 位寻址方式中要注意在以 BP、EBP 或 ESP 作为基址寄存器访问存储器数据时,默认的段寄存器是 SS;当 EBP 作为变址寄存器使用时,不影响默认段寄存器的选择。所有其他寻址方式下的存储器数据访问都使用 DS 作为默认段寄存器,包括没有基址寄存器的情况。此外,可以显式地在指令中指定 CS/SS/ES/FS/GS 作为访问存储器数据时所引用的段寄存器,以改变默认的段寄存器。

3.2.2　程序寻址方式

程序在存储器中的寻址方式有如下四种。

1. 段内直接寻址方式

段内直接寻址方式也称为相对寻址方式。指把指令本身提供的位移量加到指令指针寄存器中去形成目标有效地址的寻址方式。例如：

```
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上一节所述数据存储器寻址方式取得，所得到的转移的有效地址用来更新IP的内容。由于此寻址方式仅修改IP的内容，所以这种寻址方式只能在段内进行程序转移。例如：

```
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。例如：

```
JMP   2500H:3600H       ;转移的段地址和偏移地址都在指令中给出
CALL  2600H:3800H       ;调用的段地址和偏移地址都在指令中给出
```

4. 段间间接寻址方式

这种寻址方式是由指令中给出的存储器数据寻址方式包括存放转移地址偏移量和段地址。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。例如：

```
JMP    DWORD PTR[DI]        ;转移地址在DI,DI+1,DI+2,DI+3所
                            ;指的内存单元中,前2个字节为偏移
                            ;量,后2个字节为段地址
CALL   DWORD PTR[DI]        ;调用地址在DI,DI+1,DI+2,DI+3所
                            ;指的内存单元中,前2个字节为偏移
                            ;量,后2个字节为段地址
```

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一个段。

3.2.3 操作数宽度和寻址宽度的确定

这里的操作数宽度和寻址宽度也可称为操作尺寸和寻址尺寸，实际上指的是操作数二进位数和可寻址的二进位数。8086和80286的操作数宽度和寻址宽度都是16位的。

而 32 位处理器的操作数宽度和寻址宽度可以是 16 位的，也可以是 32 位的。

操作数宽度和寻址宽度的确定涉及到 W 域、默认段属性及指令前缀等概念。

1. W 域与操作数宽度

在 8086 的指令格式中，操作码的末位称为 W 域。当 W＝0 时，表示操作数宽度为 8 位，即以字节为单位；当 W＝1 时，表示操作数宽度是 16 位。

与 8086 类似，在其他 80X86 中，也有 W 域。当 W＝0 时，也表示操作数宽度是字节；当 W＝1 时，有两种操作数宽度：在 16 位模式(实地址模式、虚拟 8086 模式、286 兼容保护模式)中，表示默认操作数宽度是 16 位的；在 32 位模式中，表示默认操作数宽度是 32 位的。

2. 默认的操作数宽度和寻址宽度

在保护模式下，可执行段描述符中的 D 位确定了默认操作数宽度和寻址宽度的属性。这个属性适用于在无操作数宽度前缀和无寻址宽度前缀的条件下，段内所有指令的执行。当 D 位为 0 时，表示默认的段操作数宽度和寻址宽度是 16 位的，为 1 时，则表示是 32 位的。

在实地址模式、虚拟 8086 模式和 286 兼容保护模式下，运行程序的默认操作数宽度和寻址宽度是 16 位的。

3. 指令的操作数宽度和寻址宽度前缀

一条指令的内部编码可以包含两种宽度前缀：操作数宽度前缀(66H)和寻址宽度前缀(67H)。它们可以单独出现，也可以同时出现在指令前面。

操作数宽度前缀使处理器在默认的 16 位模式下进行 32 位(或 8 位)的操作，或者在默认的 32 位模式下进行 16 位(或 8 位)的操作。

寻址宽度前缀使处理器在默认的 16 位模式下进行 32 位寻址，或者在默认的 32 位模式下进行 16 位寻址。

当指令前无宽度前缀时，指令的操作数宽度和寻址宽度按默认的属性确定；当加上操作数宽度或/和寻址宽度前缀时，则实际的尺寸属性改变成与默认属性不同，下面给出几个例子加以说明。

例 3.1　在实地址模式下，程序员欲访问 32 位的 EAX 寄存器，可以使用指令：

```
MOV   EAX,32 位操作数
```

ASM 386 或 ASM 586 在汇编时自动地在该指令代码前加上操作数宽度前缀，从而将本来为 16 位的(默认)操作数宽度属性改变成了 32 位属性。

例 3.2　在 16 位默认属性的代码段中，程序员可以(汇编时加寻址宽度前缀)使用指令：

```
MOV   DX,TABLE[ESI×2]
```

这条指令中，为了便于对数组的访问，用了 32 位寻址的比例因子。

例 3.3　在 D＝1(默认段属性为 32 位)的代码段中，程序员可以使用指令：

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

进行16位数据的操作。汇编时,会加上操作数宽度前缀。

需要指出,在16位默认地址模式下,使用寻址宽度前缀不能使偏移量范围超过64KB,否则将会导致一般保护异常处理。使用寻址宽度前缀只意味着在实地址模式下可以使用32位寻址方式,但不能超过它所规定的限制。

3.2.4 I/O地址空间

80X86提供一个区别于物理存储器的独立的I/O地址空间,其中含有64K个可单独寻址的8位端口。两个相邻的8位端口可以构成一个16位端口;4个相邻的8位端口可以构成一个32位的端口。16位端口应对齐于偶数地址,在一次总线访问中传送16位信息;32位端口对齐于能被4整除的地址,在一次总线访问中传送32位信息。8位端口的地址可以任意确定。

I/O地址空间的寻址方式很简单,仅有两种:直接寻址和DX间接寻址。

1. 直接寻址

指由指令直接给出端口号,端口号可以为0～255。例如:

```
IN AL,32H;
```

32H为8位端口号。

2. DX间接寻址

由DX寄存器指出端口号,这种方式给出的端口号可为0～65 535。例如:

```
IN AL,DX;
```

DX寄存器的内容为端口号。

注意:无论程序工作在16位还是32位模式,都用DX作为I/O空间的间址寄存器,这是因为I/O地址空间仅有64KB的缘故。

3.2.5 段寄存器的确定

为了简化指令系统,80X86的指令在形式上只给出了地址偏移值(有效地址),来指明当前段寄存器是哪一个。通常用表3.1给出默认规定来隐含指出所选用的段寄存器。

表3.1 段寄存器选择规定

访存类型	默认寄存器	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(E)BP、(E)SP间址的指令	SS	可用CS、DS、ES、PS、GS
其他	DS	可用CS、SS、ES、FS、GS

这种隐含选择规定可以被段超越前缀改变。当在一条指令前面加上段超越前缀时，则由段超越前缀所指定的段寄存器取代默认的段寄存器。例如：

```
MOV EBX,ES:[EDI];
```

源操作数在 ES 指定的段中。

3.3　80X86 的指令系统

80386 对 8086/8088 和 80286 指令系统进行了扩充，这种扩充不仅体现在增加了指令的种类，也体现在提供了 32 位寻址方式和 32 位操作方式。80486 在 80386 体系结构的基础上增加了 6 条新指令，并扩充了数字指令集。Pentium 在此基础上增加了 6 条指令，并且扩充了一些指令的功能。指令机器码格式如下：

前缀	操作码	寻址方式	位移量	立即数
0~ 3 字节	1~ 2 字节	0~ 2 字节	0~ 4 字节	0～4 字节

指令的最大长度为 15 字节。前缀部分依次有 LOCK 前缀、寻址宽度前缀、操作数宽度前缀、重复前缀和段超越前缀。

本节对 80X86 的指令系统分类进行综述，只讨论指令的外部功能，而不涉及指令内的保护性检查。有关保护的内容和有关指令的细节请参阅有关参考文献。80X86 的指令多为双操作数指令，在介绍指令时，所指的第一操作数、第二操作数是根据它们在指令中出现的顺序而言的，不包含指令中隐含的操作数。对于双操作数指令：第一操作数通常用作目的操作数，第二操作数通常用作源操作数，两个操作数运算后的结果放入目的操作数中，这样目的操作数中的原数据要丢失，如果运算中还会用到这个数的话，应预先把它保存起来。80X86 的指令也有三操作数指令，它除给出两个操作数参加运算外，另外还指出运算结果的存放地址，如指令 IMUL EBX,TABLE [ESI×4],7 就是三操作数指令。另外，有些指令只有一个操作数，如 INC CX 指令；甚至可能没有操作数，如 NOP 指令。

3.3.1　传送类指令

这类指令按字节、字或双字在存储器与存储器之间，寄存器与寄存器之间以及存储器与寄存器之间进行传送。

1. 传送指令 MOV

格式：

```
MOV dest,src; dest←src
```

这里 dest 为目的操作数，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（如图 3.2 所示）。

例如：

```
MOV  EAX,EBX        ;EAX←(EBX)(32位双字)
MOV  AX,BX          ;AX←(BX) (16位字)
MOV  AL,BL          ;AL←(BL) (8位字节)
MOV  EAX,25         ;EAX ←25 (32位立即数)
MOV  EAX,[500]      ;EAX←DS:500H 单元中双字
```

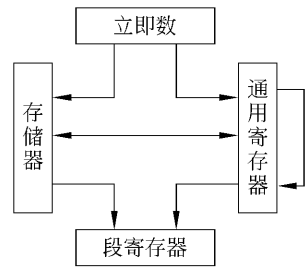

图 3.2 MOV 指令所允许传送示意图

注意：使用 MOV 指令时，不能往段寄存器送立即数。如要往段寄存器设置新值，必须先将立即数送通用寄存器或存储单元，再从寄存器或存储单元送段寄存器。但是，不允许往 CS 寄存器或(E)IP 寄存器进行任何形式的数据传送。另外，传送指令的两个操作数的位数必须相同。

2. 交换指令 XCHG

格式：

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

要求：dest 为 8 位、16 位或 32 位通用寄存器或存储单元，src 为 8 位、16 位或 32 位通用寄存器或存储单元。

功能：XCHG 指令用于交换两个操作数。这条指令实际上起到了三条 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。当一个操作数是存储器操作数时，XCHG 指令会自动地激活 LOCK 信号，这一特性在多处理器访问共享临界资源时非常有用。

例如：几个处理器共享打印机，因打印机不能为几个处理器交替打印，为此设置一个信号灯单元 Semaphore。当每个处理器欲使用打印机时，都先去查看信号灯单元，在确信打印机空闲时才能使用，否则只好等待到打印机释放。

与使用打印机有关的程序段如下：

```
TEST: MOV   AL,0FFH
      XCHG  AL,Semaphore
      AND   AL,AL                ;产生条件码
      JNZ   TEST                 ;查看 Semaphore 内容,不为 0 时等
                                 ;待(忙时等待)
        .
        .   处理器使用打印机
        .
      MOV   AL,0
      MOV   Semaphore,AL         ;使用完,置空闲标志
```

XCHG 指令的执行需两个总线周期。前一周期，将 Semaphore 内容传给 CPU 内部的暂存器，后一周期，再将置定的 AL 内容送 Semaphore 单元，并将暂存器内容送 AL。