第 3 章　8086 指令系统

本章导读

- 8086 指令的特点
- 8086 的寻址方式
- 8086 的指令格式
- 8086 的数据类型
- 8086 的指令集

计算机是通过执行指令序列来工作的，每种计算机都有一组指令集提供给用户使用，这组指令集称为该计算机的指令系统。不同 CPU 的计算机使用不同的指令系统，8086 CPU 的指令系统不但包含 8 位机的全部指令，而且增加了一些功能较强的 16 位数据处理指令，如乘法、除法指令，因而同时具有 8 位和 16 位的处理能力。

3.1　8086 指令的特点

任何一条指令都由操作码（Opcode）和操作数两部分组成。

指令中的操作码部分表明指令的操作性质，一般有 1～2 字节；操作数部分既可以表示参加操作的数，也可以表示参加操作的数所在位置。当表示参加操作的数所在位置时，操作数部分又称为地址码。操作数部分有 0～4 字节。在一条指令中，操作码部分是必需的，而操作数部分可能隐含在操作码中，或者由操作码后面的指令给出。

1．灵活的指令格式

8086 的指令中有 1 字节长，此时指令中的操作数部分隐含在操作码中，大部分对 16 位寄存器操作的指令都只有 1 字节长；当操作数在内存中，编程又需要复杂的寻址方式时，有的指令多达 6 字节长。比如，1 字节指令：

指令助记符	指令的十六进制数代码
DAA	27H

该指令是十进制数加法调整指令，这个十进制数就隐含在寄存器 AL 中。

再如，6 字节指令：

指令助记符	指令的十六进制数代码
MOV　[BX+SI+1020H], 3040H	C78020104030

表达式 BX+SI+1020H 代表一个地址，该指令是把数 3040H 送到该地址指示的内存单元中。

2．指令格式的一对多形式

用助记符编写的指令最终要翻译成二进制代码由 CPU 执行。为了方便用户理解，可用两种不

同的助记符描述同一问题，如 JE/JZ。当两个无符号数进行比较操作时，用户理解为相等则转移，也可理解为 ZF=1 则转移。虽然助记符不同，但其二进制代码是一样的。

3．较强的运算指令

当用 8 位机完成乘法运算时，只能用连加或对位权移位等方式编写一段程序实现。8086 中有乘法、除法指令，给用户提供了极大方便。

4．指令有极强的寻址能力

在微机系统中，参加操作的数据有可能在 CPU 的寄存器、内存或外部设备中。指令中如何提供数据所在位置，以便提高程序执行效率，这就需要 CPU 提供强有力的寻址能力。细分 8086 指令的寻址方式多达 9 种，特别是对内存的寻址方式十分灵活。

5．指令有处理多种数据的能力

8086 指令能够处理 8 位/16 位数、带符号/无符号数以及压缩 BCD 数/非压缩 BCD 数。带符号数和无符号数有相应的乘法、除法指令，压缩 BCD/非压缩 BCD 数有相应的调整指令。

3.2　8086 的寻址方式

寻址方式是指令中用于说明操作数所在地址的方法。可以说，寻址方式的多少也是衡量 CPU 功能的指标。8086 的寻址方式十分丰富，32 位微机的寻址方式在其基础上稍有增加。

8086 指令中的操作数有一个或两个，个别指令有三个，称为源操作数和目的操作数。除目的操作数不允许为立即数（即立即寻址）外，其余寻址方式均适合源操作数和目的操作数。

3.2.1　8086 寻址方式的说明

当操作数在 CPU 内部的寄存器中时，寻址简单且有效。遗憾的是，CPU 内部的寄存器数目有限，大量的数据以及运算结果需要存放在内存中。8086 的寻址能力是 1 MB 内存空间，指令中如何给出操作数所在地址，从而提高程序编制及指令执行效率，就在于有多种形式的寻址方式。当操作数从外部设备输入或输出时，寻址方式有其特殊性。虽然外部设备也像内存单元一样，用地址进行访问（与内存地址区别，一般称为 I/O 端口地址），但由于外设的多样性，使传输的数据宽度有 8 位和 16 位之分，同时一个微机系统配置的外部设备数目不同，I/O 端口地址的编址也就不同（如 8 位地址/16 位地址）。

8086 对外部设备的寻址方式就是针对以上需要设置的。

为了使读者便于理解 8086 的寻址方式，先说明以下两个问题。

1．有效地址（Effective Address，EA）

当操作数在内存中时，指令的地址码（操作码）给出所访问的内存单元的逻辑地址。在寻址方式中，逻辑地址由多个分量组合而成，该组合地址又叫有效地址。其有效地址组合关系如图 3-1 所示。

2．数据传输指令

为了理解寻址方式，需要以 8086 的指令为例，这里先介绍 MOV 指令。其格式为：

MOV　目的操作数，源操作数

目的操作数和源操作数均可采用不同的寻址方式，但两个操作数的类型必须一致。

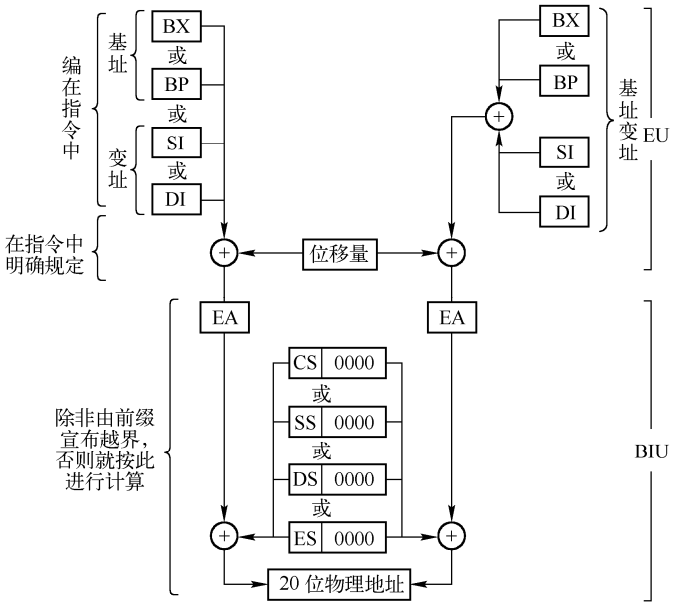

图 3-1　存储器地址

3.2.2　寻址方式介绍

1．立即寻址（immediate addressing）

操作数就在指令中，紧跟在操作码后面，作为指令的一部分存放在内存的代码段中，这种操作数称为立即数。例如：

```
MOV     AX, 34EAH
```

该指令是将一个 16 位的十六进制源操作数 34EAH 送到 8086 CPU 的 16 位寄存器 AX 中。执行后，寄存器 AX 的内容为 34EAH。又如：

```
MOV     BL, 20
```

该指令是将一个 8 位的十六进制源操作数 14H 送到 8086 CPU 的 8 位寄存器 BL 中。执行后，寄存器 BL 的内容为 14H（即 0001 0100）。

说明：当指令中的立即数后面不加字母 H 时为十进制数，汇编时该立即数由汇编程序以二进制数形式存于代码区。

2．寄存器寻址（register addressing）

若操作数在寄存器中，指令中的源操作数和目的操作数都可用这种寻址方式。对于 16 位操作数，寄存器可以是 AX、BX、CX、DX、SI、DI、SP、BP；对于 8 位操作数，寄存器可以是 AH、AL、BH、BL、CH、CL、DH、DL。下面两个例子中，目的操作数就是寄存器寻址方式。例如：

```
MOV     BP, SP
```

该指令执行后，寄存器 SP 的内容传送到了寄存器 BP。又如：

```
MOV     AX, 1234H
MOV     AL, AH
```

第一条指令执行后，寄存器 AX 的内容为 1234H，再执行下一条指令，则寄存器 AH 的内容为 12H，而寄存器 AL 的内容也为 12H。

3．直接寻址（direct addressing）

当操作数在内存单元时，在指令中必须给出被访问内存单元的逻辑地址，由 8086 CPU 的存储器管理部件计算出有效地址（按表 3-1 所示的方式），再转换成物理地址，才能对被选定的内存单元进行访问（读或写）。存储器读是指将内存单元的数据通过数据线传送给目的地，存储器写是指将数据通过数据线传送至内存单元中。

表 3-1　存储器存取时约定段和可修改段的基数

存储器存取方式	约 定 段	可超越使用的段	偏 移 量
取指令	CS	无	IP
堆栈操作	SS	无	SP
源字符串	DS	CS，ES，SS	SI
目的字符串	ES	无	DI
用 BP 做基址	SS	CS，ES，DS	有效地址
通用数据读写（BP 作为基址除外）	DS	CS，ES，SS	有效地址

当指令中的源操作数或目的操作数采用直接给出被访问内存单元的逻辑地址的方式时，这种寻址方式称为直接寻址。例如：

```
MOV     AX, [3E4CH]
```

在该指令中，源操作数用直接寻址方式，指令中由方括号内给出的是被访问内存单元的逻辑偏移地址，逻辑段地址隐含在寄存器 DS 中。指令中，EA 的内容为 3E4CH，物理地址为(DS)×10H+3E4CH。由于存储器以 8 位数据为单元组织（字节），在传输 16 位数据时，是将两个连续的内存单元组成的 16 位数据在数据线上并行传输，低地址单元的 8 位数据送到寄存器 AL 中，高地址单元的 8 位数据送到寄存器 AH 中。其他寄存器的写入类似。

又如：

```
MOV     [1234H], BL
```

在该指令中，目的操作数采用直接寻址方式，将寄存器 BL 的 8 位数据送到逻辑偏移地址为 1234H 的单元中。其中，EA 为 1234H，物理地址为(DS)×10H+1234H。

再如：

```
MOV     ES:[1234H], BL
```

该例为读者提供有关段超越的概念。第 2 章描述 8086 CPU 的寄存器时，提到过段寄存器与表示逻辑偏移地址的寄存器的隐式关系，如 CS:IP、SS:SP、DS:BX、DS:SI、DS:DI 和 DS:直接偏移地址。此例中，EA 为 1234H，而物理地址为(ES)×10H+1234H。

说明： ① 在以上例子中，指令中的直接地址是用十六进制数形式给出的。在实际编程应用中，一般使用符号地址，即由事先定义的符号表示逻辑偏移地址。例如:

```
MOV     AX, BUFF
```

② 指令中使用直接寻址方式时，在没有声明的情况下，逻辑段地址是指寄存器 DS 的内容;如果用户编程使用其他段寄存器，则需在指令中给出。例如:

```
MOV     ES:BUFF, DX
```

4．寄存器间接寻址（register indirect addressing）

当被访问的操作数在内存单元时，使用该寻址方式。与直接寻址方式不同的是，内存单元的逻辑偏移地址通过间接的方式给出，即先将被访问内存单元的逻辑偏移地址传送给寄存器，在指令中再由寄存器给出被访问的内存单元的逻辑偏移地址。例如：

```
MOV     SI, 61A8H
```

```
MOV     DX, [SI]
```

可以看出，第一条指令将偏移地址传给寄存器 SI，第二条指令采用寄存器间接方式，给出被访问内存单元的逻辑偏移地址。

说明： ① 第一条指令的源操作数是立即数，在第二条指令中将此立即数作为偏移地址使用；② 若没有声明，寄存器间接寻址方式中的逻辑段地址使用隐式用法，即 DS 与 BX、SI 和 DI 寄存器组成物理地址，SS 与 BP、SP 寄存器组成物理地址；③ 8086 中的寄存器 AX、CX、DX 一般不能在寄存器间接寻址中使用。

5．基址/变址寻址（based/indexed addressing）

这种寻址方式中提出位移量的概念，即在寄存器间接寻址给出的偏移地址上，加一个相对位移量，所以也称为相对寻址。位移量是一个带符号的 16 位十六进制数。当使用寄存器 BX 或 BP 时，称为基址寻址；使用寄存器 SI 或 DI 时，称为变址寻址。例如：

```
MOV     CX, 36H[BX]
```

其中，EA 为 36H+(BX)，物理地址为(DS) ×10H+36H+(BX)。又如：

```
MOV     -20[BP], AL
```

其中，EA 为(BP)−14H，物理地址为(SS) ×10H−14H+(BP)。

说明： ① 这种寻址方式适合以一维表格存储在内存中的操作数，用基址或变址寄存器存放表格的首地址，用位移量表示数据元素的位置，可以很方便地找到该操作数；② 这种寻址方式同样可以使用段超越。

6．基址+变址寻址（based indexed addressing）

如果用户的数据结构比较复杂，如二维表，考虑编程方便和程序执行的效率，则可以采用这种寻址方式。其有效地址 EA 由 3 部分组成：基址寄存器 BX 或 BP 的内容+变址寄存器的内容+位移量。物理地址由基址寄存器按规则选择段寄存器，也可使用段超越。例如：

```
MOV     AX, 8AH[BX][SI]
```

其中，EA 为 8AH+(BX)+(SI)，物理地址为(DS) ×10H+8AH+(BX)+(SI)。

7．串寻址（string addressing）

串寻址方式仅在 8086 的串指令中使用。串指令中的操作数由其他指令提供，且操作数大多在内存单元中。因此，规定源操作数的逻辑地址为 DS:SI，目的操作数的逻辑地址为 ES:DI。当执行串指令的重复操作时，根据设定的方向标志（DF）的内容，SI 和 DI 会自动调整。

8．I/O 端口寻址（I/O port addressing）

当操作数在外部设备时，使用 I/O 指令。微机系统中采用地址来访问不同的外部设备。为了与内存地址区别，该地址称为端口地址。由于外部设备的多样性，此时有两种不同的寻址方式访问 I/O 端口。当外部设备地址用 8 位寻址时，使用直接端口寻址方式。在这种寻址方式中，I/O 地址仅有 256 个，即 0～255（00H～FFH）。在 I/O 指令中直接给出 0～255 中被选定的地址。当外部设备地址为 16 位（或超过 8 位）寻址时，采用寄存器间接寻址方式，用寄存器 DX 作为间接寻址寄存器。此时的端口地址多达 2^{16} 个。

由于外部设备的数据宽度不同，输入指令中目的操作数可为 AL 或 AX，输出指令中源操作数可为 AL 或 AX。例如：

```
IN      AL , 25H
```

将端口地址为 25H 的输入设备中的 8 位数据送到寄存器 AL 中。又如：

```
MOV     DX, 3E4H
```

```
OUT     DX, AL
```

将 AL 寄存器的数据传输至端口地址为 3E4H 的外部设备。

3.3　8086 的指令格式及数据类型

用汇编语言编写的源程序输入计算机后，由汇编程序将它翻译成机器指令组成的机器语言程序，计算机才能识别并执行。一般来说，此翻译过程不必由人工来干预，但必要时也可由程序员来完成，称为“手工汇编”。本节简略介绍机器指令格式及组成原理。

1．8086 的操作码

指令由操作码（Opcode）和操作数（地址码）组成。8086 的指令长度是可变的，一条指令一般由 1～6 字节组成（加上前缀字节，最长可为 7 字节）。指令的操作码采用二进制代码表示本指令所执行的操作，通常用指令的第一个字节表示。有时由于用 8 位不够，因此在指令的第二个字节中还可能占 3 位。除此以外的其他字节（或位）用来表示操作数。

在多数操作码中，使用某些位来指示有关信息，常用的符号及含义如下：

- W（字位）——表示本指令是对字（W=1）还是对字节（W=0）操作。
- D（方向位）——表示参加操作的寄存器是作为目的操作数寄存器（D=1）还是作为源操作数寄存器（D=0）。
- S（符号扩展位）——用于某些立即寻址的机器指令中。如果立即数为 8 位，但要求扩展成 16 位，则 S=1；否则，S=0（所扩展的高 8 位要根据 8 位立即数的最高位来定）。

2．8086 指令中寄存器的编码

（1）指令的二进制一般形式

8086 指令的二进制一般形式如下：

操作码	d/v/s	w

其中，操作码部分占 5 位，其他位指示有关信息。例如：

```
MOD     操作码  r/m
```

MOD 为 11，表示对寄存器操作；MOD 为 0x，表示对存储器操作。

（2）编码模式

编码方式主要有以下 3 种：段寄存器编码（见表 3-2），通用寄存器编码（见表 3-3），内存寻址编码（见表 3-4）。

表 3-2　段寄存器编码

编　码	段寄存器名
00	ES
01	CS
10	SS
11	DS

表 3-3　通用寄存器编码

编　码	16 位寄存器名	8 位寄存器名
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

表 3-4　内存寻址编码

r/m 编码	基址寄存器	变址寄存器
000	BX　+	SI
001	BX　+	DI
010	BP　+	SI
011	BP　+	DI
100	×	SI
101	×	DI
110	BP	×
111	BX	×

8086 指令中通常使用一个或两个操作数，在少数指令中有隐含的第三个操作数。

3．指令中的操作数

单操作数指令中只有一个操作数，例如：

指令助记符		指令的十六进制代码
INC	AX	40H
INC	BX	43H

双操作数指令中有两个操作数，例如：

指令助记符		指令的十六进制代码
MOV	AL, 04	B004H
MOV	AX, 04	B80400
POP	DI	5FH
ADD	AX, BX	01D8H
MOV	BX, 1234H	BB3412
PUSH	AX	50H

在 8086 指令系统中，大多数指令中只有 1～2 个操作数，但也有少数指令中有 3 个操作数，不过其中 1 个操作数隐含在操作码中。例如：

```
ADC     AX, BX
```

该指令完成操作数 AX、BX 和 CF 位相加。又如：

```
LDS     SI, [BX]
```

该指令有 2 个目的操作数，将源操作数 DS:[BX]组成的物理地址的连续 4 字节单元中的内容分别送到寄存器 SI 和 DS 中。

4．指令中的数据类型

在 8086 系统中，指令中处理的数据可为 8 位和 16 位，若要求处理位数更多的数，则需要编写程序，将其按从低到高的顺序按 8 位或 16 位进行处理。

① 无符号数：没有符号的 8 位数，其值为 00H～FFH（0～255）；没有符号的 16 位数，其值为 0000H～FFFFH（0～65535）。

② 带符号数：由于符号占用 8 位中的 1 位（最高位），故 8 位中仅有 7 位表示数据，且用其补码表示，其值范围为–80H～+7FH（–128～+127）；同样，16 位中仅有 15 位表示数据，且用其补码表示，其值范围为–8000H～+7FFFH（–32 768～+32 767）。

在程序中，带符号数的符号也像数据一样被处理。对带符号数的加/减运算结果要考虑溢出问题，对乘/除运算要使用正确的指令。

③ ASCII：西文字母和一些常用符号用 ASCII 代码表示。从计算机键盘输入的数据或符号，在机器中得到的是其对应的 ASCII 代码；将数据或符号在屏幕上显示，也必须先将其转换成 ASCII 码。在程序设计中，ASCII 代码用单引号括起来。

④ BCD 数（压缩 BCD 和非压缩 BCD）：采用 BCD 数是由于十进制数符合人们的书写和阅读习惯，特别是在屏幕上显示运算结果，出现有字母的数据总是很别扭。用 4 位二进制数表示 1 位十进制数的编码，在运算时会出现一些问题，好在有相应的调整指令可以解决。

3.4　8086 的指令集

8086 指令系统按功能可分为 6 种。

① 数据传输类：其功能是将源操作数的内容传送到目的操作数中。这类指令形式较多，寻址方式也十分丰富，在程序设计中使用频率很高。

② 算术运算类：其功能是完成加、减、乘、除运算。每次运算会对 6 种状态标志产生影响。使用这类指令时，要注意指令的书写形式（源操作数和目的操作数的存放位置）以及乘/除运算中参加运算的数据类型。

③ 逻辑运算类：其功能有两种，一是完成逻辑“反”、“与”、“或”、“异或”和测试，二是对数据的移动。这类指令是按二进制位进行的，所以又称为位操作指令。这种运算也对状态标志位产生影响。

④ 串操作类：有不同的寻址方式，段寄存器和变址寄存器按 DS:SI 和 ES:DI 组合寻址。这类指令在执行前根据编程需要设置 DF 标志，在这些指令前加上指令前缀，可以完成指令的循环操作。

⑤ 程序控制类：程序中执行跳转、调子程序、中断服务等指令，都要使原来的 CS:IP 或 IP 寄存器的内容改变，使之指向下一条要执行指令的位置。程序中的分支、循环、中断等都会用到这些指令，这些指令可分为有条件（又可分为单条件和多条件）指令和无条件指令。其条件的产生是先执行运算指令，使状态标志发生变化，然后通过这些指令测试来控制程序转移。转移可在同一段中进行（仅改变 IP 内容），也可以在不同段中进行（CS 和 IP 都改变）。

⑥ 处理机控制类：提供程序控制 CPU 的各种功能，如使处理机暂停、等待、封锁总线等，还可以对 FR 寄存器中的一些标志进行置“1”或清“0”等操作。

3.4.1　数据传输指令

数据传输指令负责把数据、地址等传送到寄存器、存储单元或外部设备中，以实现存储器和寄存器、寄存器和寄存器、AX 或 AL 寄存器与 I/O 端口之间的字节型或字型数据的传输。堆栈操作、标志操作等也属于这类指令。数据传输指令又可以分成 4 种：通用数据传输，累加器专用传输（输入/输出数据传输），目的地址传输和标志寄存器传输。

指令的共同特点如下：

- ⊙ 除 POPF 和 SAHF 指令外，这类指令的操作结果不会影响 FR 寄存器中的标志。
- ⊙ 指令中有两个操作数：目的操作数和源操作数，其执行过程为：源操作数→目的操作数，当指令中仅列出一个操作数时，另一操作数为隐含。

1．通用数据传输指令

8086 有 4 个通用数据传输指令。这种指令允许以段寄存器作为操作数（除 XCHG 外），这是其他类型指令所不能实现的。

（1）传送指令 MOV

指令格式为：

MOV　　目的操作数，源操作数

由于微机系统中，源操作数和目的操作数可以存在于寄存器、存储器、外部设备中，故指令中的操作数寻址方式较多，其数据传输方向如图 3-2 所示。

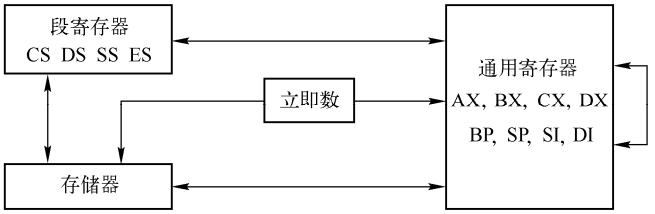

图 3-2　数据传送方向示意

MOV 指令在一个汇编语言程序中使用的频率最高，也比较容易理解，但必须注意：

⊙ 其中的寄存器不包括 IP。

⊙ 目的操作数不允许用段寄存器 CS。当指令涉及 CS 时要小心，因为执行一条改变 CS 寄存器内容的指令，将会使一个新的段成为当前代码段，而此时 IP 却仍然指示着以前代码段中的下一条指令。所以在改变 CS 值的同时，也应该给 IP 一个相应的有效值，处理机才能正常工作。

⊙ 目的操作数不允许是立即数。

⊙ 立即数不能直接送至段寄存器，要通过其他寄存器转送。

⊙ 源操作数和目的操作数的数据类型必须相同。

⊙ 不允许在两个存储单元中直接传送数据，要通过寄存器转送。

⊙ 源操作数和目的操作数不可以同时为段寄存器。

（2）进栈指令 PUSH

指令格式为：

```
PUSH    源操作数
```

堆栈操作总是对 16 位的数据进行，指令中目的操作数隐含为堆栈。进栈操作把数据传输到以 SS 为段基址、SP 为偏移地址的栈中。其操作过程如下：

<1> SP 减 1，指示堆栈中可以存放数据的位置，存源操作数的高 8 位。

<2> SP 再减 1，存源操作数的低 8 位，完成进栈操作。

（3）出栈指令 POP

指令格式为：

```
POP     目的操作数
```

指令中，源操作数隐含为堆栈，出栈操作把以 SS 为段基址、SP 为偏移地址的栈顶内容传输到目的操作数中。其操作过程如下：

<1> S 将 SS:SP 所指示的栈顶处的 2 字节的数据传输到目的操作数中。

<2> SP 加 2，指示当前栈顶位置，完成出栈操作。

（4）交换指令 XCHG

指令格式为：

```
XCHG    目的操作数, 源操作数
```

交换指令中，目的操作数和源操作数中的内容互换，因而要注意：操作数不能为立即数，源操作数和目的操作数不能同时为存储单元，段寄存器不能作为操作数。

例如，将字型数据的偏移地址为 2040H 单元和 2050H 单元的内容交换。这里给出三个方案。

方案一：用 MOV 指令

```
MOV     AX, [2040H]
```

```
MOV     BX, [2050H]
MOV     [2050H], AX
MOV     [2040H], BX
```

方案二：用 XCHG 指令

```
MOV     AX, [2040H]
XCHG    AX, [2050H]
MOV     [2040H], AX
```

方案三：用 PUSH，POP 指令

```
PUSH    [2040H]
PUSH    [2050H]
POP     [2040H]
POP     [2050H]
```

方案三巧妙地利用了堆栈区先进后出的工作方式。最后进入堆栈的数据是2050H单元的内容，然后利用指令 POP [2040H]，把此内容传输给了 2040H 单元。希望读者思考并掌握这种方法。

2．累加器专用传输指令

8086 CPU 要与外部设备交换数据，必须通过累加器 AX（16 位数据）或 AL（8 位数据）传输给 I/O 端口，外设从输出端口取数据，完成数据输出。反之，外设将数据传输到 I/O 端口，CPU 从端口中将数据取到 AX 或 AL 中，完成数据输入。

硬件系统根据外设情况，设计端口地址为 16 位（外设较多）或 8 位（外设较少），而数据的大小也会根据具体情况设计，可以是 8 位 A/D 转换器转换得到 8 位数据，也可能是 16 位 A/D 转换器转换得到 16 位数据。8086 有相应的指令处理这些情况。

（1）输入指令 IN

指令格式为：

```
IN    AL, n
IN    AX, n
IN    AL, DX
IN    AX, DX
```

说明：以上 4 种指令格式分别是 8 位/16 位数据和 8 位/16 位端口地址的组合形式；指令中 n 表示为 8 位端口地址（00H～FFH），此时地址线高 8 位默认为 0；当端口地址为 16 位时，指令中采用寄存器间接寻址，应先将 16 位端口地址传输到 DX 中，然后 DX 中的地址在 IN 指令中间接寻址。

（2）输出指令 OUT

指令格式为：

```
OUT    n, AL
OUT    n, AX
OUT    DX, AL
OUT    DX, AX
```

OUT 指令除源操作数和目的操作数与 IN 指令相反外，其他相同。

可见，无论是输入指令或输出指令，其指令中地址部分的寻址方式要么是直接寻址，要么是寄存器间接寻址。

（3）换码指令 XLAT

指令格式为：

XLAT

这条指令没有明显的操作数，其操作数需其他指令提供，所以单独执行该指令不能达到预期要求。

所谓换码，是指令能够完成 1 字节的查表转换。有时需要将一种代码转换成另一种代码，或在处理实际问题时，采用一种映射关系来完成转换。前提是正确找到映射关系，利用存储器地址的连续性和规律建立表格，然后用该指令完成转换。

执行该指令之前，需先执行两条指令：

MOV　BX, **表的偏移首地址**

MOV　AL, **被转换码**

由于该指令不能单独执行，有时称之为复合指令。

例如，建立一个 0～9 的平方表，求 5 的平方值。将 0～9 的平方表建立在偏移地址为 2000H 的内存中，如图 3-3 所示。

地址	平方值
2000H	00
2001H	01
⋮	⋮
2009H	81

图 3-3　内存中的平方表

完成求 5 的平方指令序列为：

```
MOV   BX, 2000H        ; 指向平方表的首地址
MOV   AL, 5
XLAT                   ; 执行换码指令，平方值放在 AL 中
```

以上例子完全是为说明 XLAT 指令而设计的，实际应用中求平方未必使用这种方法。

3．目标地址传输指令

目标地址传输指令是计算有效地址的指令，有效地址是指存储器地址。因此在这类指令中，源操作数必须是存储器。

（1）LEA（有效地址传输到寄存器）

指令格式为：

LEA　　16 **位寄存器，源操作数偏移地址**

例如：

LEA　　SI, [2040H]

指令执行后，SI 中的内容为 2040H。又如：

MOV　　SI, [2040H]

指令执行后，SI 中有偏移地址为 2040H 单元中的内容，而不是 2040H 这个值。

（2）LDS（装入一个新的物理地址）

指令格式为：

LDS　　16 **位寄存器，源操作数偏移地址**

该指令是三个操作数指令，其中有两个目的操作数。其功能是：将源操作数指示的偏移地址开始的 4 个连续字节中的内容传输到目的寄存器和 DS 寄存器中。

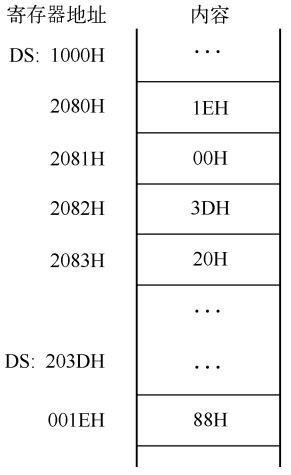

图 3-4　LDS 指令示意

根据图 3-4 执行下列指令：

```
MOV    BX, 2080H      ; 用 BX 间接寻址
LDS    SI, [BX]       ; SI←[2080H]，DS←[2082H]
MOV    AL, [SI]       ; AL=88H
```

（3）LES（装入一个新的物理地址）

这条指令除段寄存器不同外，其余与 LDS 指令类似。

4．标志寄存器传送指令

FR 寄存器是个比较特殊的寄存器，它虽然是 16 位的，但 8086 中只定义了 9 位，所定义的每一位都有自己的含义，对于它的操作跟其他寄存器不一样。其指令形式也有特点：操作数都是隐含的，其中 8 位的传输是对 SF、ZF、AF、PF 和 CF 的操作。

LAHF：FR 寄存器的低 8 位→AH。

SAHF：AH→FR 寄存器的低 8 位。

PUSHF：FR 寄存器推入堆栈中。

POPF：从栈顶中弹出的内容存入 FR 寄存器中。

3.4.2　算术运算指令

算术运算指令的共同点如下：① 运算指令影响状态标志；② 参加运算的数可以是无符号整型数、带符号整型数、压缩 BCD 数和非压缩 BCD 数；③ 乘法/除法指令中，乘数、被乘数以及除数、被除数、商和余数的存放位置有规定；④ 乘法和除法指令的书写形式有要求。

1．算术加法指令

（1）算术加法 ADD

指令功能：目的操作数←目的操作数+源操作数。

指令形式：

```
ADD     目的操作数，源操作数
```

说明：① 指令的目的操作数不能是立即寻址；② 加法操作中产生的进位影响 CF 标志；③ 带符号操作数相加要考虑溢出。例如：

```
ADD     AL, 12H
ADD     WORD PTR [BX], 5B28H ；WORD PTR 是伪指令，用以指明该存储器操作数是字型
```

（2）带进位算术加法 ADC

指令功能：目的操作数←目的操作数+源操作数+CF。

指令形式：

```
ADC     目的操作数，源操作数
```

说明：① 指令中有 3 个操作数，其中 CF 是本指令执行前的状态；② 需要完成多字节数（如 4 字节的 32 位数或更多字节）相加时使用该指令；③ 指令的目的操作数不能是立即寻址；④ 加法操作中产生的进位进入 CF 标志位；⑤ 带符号操作数相加要考虑溢出。

例如，完成无符号数 5B68F271H 和 0AC6D5698H 加法操作。由于操作数为 32 位而 8086 的寄存器只有 16 位，因此该操作要分两次进行，先对低 16 位做加法，然后对高 16 位做加法并考虑进位。

```
MOV     AX, 0F271H      ；加数的低 16 位
ADD     AX, 5698H       ；与被加数的低 16 位相加，并影响 CF
MOV     DX, 5B68H       ；加数的高 16 位
ADC     DX, 0AC6DH      ；与被加数的高 16 位和 CF 相加，运算结果在 CF、DX 和 AX 中
```

（3）加 1 指令 INC

指令功能：目的操作数←目的操作数+1。

指令形式：

```
INC     目的操作数
```

说明：操作数不能是立即寻址；该指令不影响 CF 标志；操作数为内存寻址时，须使用伪指令。

例如：

```
INC     BYTE  PTR [BX]  ; PTR 为宏汇编中的运算符，BYTE 用以指明该存储器操作数是字节型
INC     AX
```

显然，指令 INC AX 可以由指令“ADD AX,1”代替。但要注意，指令 INC AX 不影响 CF 标志，而指令 ADD AX,1 影响 CF 标志。这是 INC 指令和 ADD 指令的一个主要区别。

（4）对压缩 BCD 数加法操作的结果进行校正 DAA

指令功能：对 AL 寄存器的内容进行十进制数调整。

指令形式：

```
DAA
```

说明：① 要求参加操作的数必须是 BCD 数；② 该指令用在压缩 BCD 数加法操作后，操作数隐含在 AL 中。

DDA 指令调整方法：如果 AF 标志为 1，或 AL 寄存器的低 4 位超出 BCD 数的计数符号（即为 0AH～0FH），则 AL 寄存器的内容加 06H，且将 AF 置 1；如果 CF 标志为 1，或 AL 寄存器的高 4 位超出 BCD 数的计数符号（即为 0AH～0FH），则 AL 寄存器的内容加 60H，且将 CF 置 1。

例如：

```
MOV     AL, 85H
ADD     AL, 96H
DAA
```

在执行“ADD AL,96H”后，AL 的内容是 1BH，同时 CF 为 1，在执行 DAA 指令时，因为 AL 中的个位数是 B，同时 CF 为 1，所以要对 AL 进行加 66H 的修正，修改结果是 81H，同时 CF 中的内容不变，即为 1。

（5）对非压缩 BCD 数加法操作的结果进行校正 AAA

指令功能：对 AL 寄存器的内容进行十进制数调整。

指令形式：

```
AAA
```

说明：① 要求参加操作的数必须是非压缩 BCD 数；② 该指令用在非压缩 BCD 数加法操作后，操作数隐含在 AL 中；③ 由于该调整指令使用 AH 寄存器，故应先将 AH 内容清零。

AAA 指令调整方法：<1> 如果 AL 寄存器的低 4 位在 0～9 之间，且 AF 为 0，则跳过第<2>步执行第<3>步；<2> 如果 AL 寄存器的低 4 位在 0AH～0FH 之间或 AF 为 1，则 AL 寄存器的内容加 06H，同时 AH 寄存器内容加 1，且将 AF 置 1；<3> AL 寄存器的高 4 位清零；<4> AF 位的值送 CF。例如：

```
MOV     AX, 09              ; AH 清零，AL 中为加数
ADD     AL, 07
AAA                         ; 结果在 AX 中为 0106
```

2．算术减法指令

（1）算术减法 SUB

指令功能：目的操作数←目的操作数–源操作数。

指令形式：

```
SUB     目的操作数，源操作数
```

说明：① 指令的目的操作数不能是立即寻址；② 减法操作中产生的借位进入 CF 标志；③ 无符号操作数相减，若 CF=1，则结果为补码；④ 带符号操作数相减要考虑溢出。

例如：

```
SUB     AX, BX
SUB     BYTE PTR [SI], 3456H
```

（2）带进位算术减法 SBB

指令功能：目的操作数←目的操作数–源操作数–CF。

指令形式：

```
SBB     目的操作数, 源操作数
```

说明：① 指令中有 3 个操作数，其中 CF 是本指令执行前的状态；② 本指令在需要完成多字节数（如 4 字节的 32 位数或更多字节）相减时使用；③ 指令的目的操作数不能是立即寻址；④ 无符号操作数相减，若 CF=1，则结果为补码；⑤ 带符号操作数相减要考虑溢出。

例如，完成无符号数 5B68F271H 和 0AC6D5698H 相减操作。由于操作数为 32 位而 8086 的寄存器只有 16 位，因此该操作要分两次进行，先对低 16 位做减法，然后对高 16 位做减法并考虑借位。

```
MOV     AX ,0F271H      ; 被减数的低 16 位
SUB     AX , 5698H      ; 与减数的低 16 位相减，并影响 CF
MOV     DX , 5B68H      ; 被减数的高 16 位
SBB     DX , 0AC6DH     ; 与减数的高 16 位和低 16 位减法时产生的借位 CF 相减，运算
                        ; 结果在 CF、DX 和 AX 中
```

（3）减 1 指令 DEC

指令功能：目的操作数←目的操作数–1。

指令形式：

```
DEC     目的操作数
```

说明：① 操作数不能是立即寻址；② 该指令不影响 CF 标志；③ 操作数为内存寻址时，须使用伪指令。

例如：

```
DEC   CX
DEC   WORD PTR [BX]    ; PTR 为宏汇编中的运算符，WORD 用以指明该存储器操作数是字型
```

（4）对压缩 BCD 数减法操作的结果进行校正 DAS

指令功能：对 AL 寄存器中的内容进行十进制数调整。

指令形式：

```
DAS
```

说明：① 参加操作的数必须是压缩 BCD 数；② 该指令用在压缩 BCD 数减法操作后，操作数隐含在 AL 中。

DAS 指令调整方法：如果 AF 标志为 1，或 AL 寄存器的低 4 位超出 BCD 数的计数符号（即为 0AH～0FH），则 AL 寄存器的内容减 06H，且将 AF 置 1；如果 CF 标志为 1，或 AL 寄存器的高 4 位超出 BCD 数的计数符号（即为 0AH～0FH），则 AL 寄存器的内容减 60H，且将 CF 置 1。

例如：

```
MOV     AL, 86H
SUB     AL, 54 H
DAS
```

（5）对非压缩 BCD 数减法操作的结果进行校正 AAS

指令功能：对 AL 寄存器的内容进行十进制数调整。

指令形式：

```
AAS
```

说明： ① 参加操作的数必须是非压缩 BCD 数；② 该指令用在非压缩 BCD 数减法操作后，操作数隐含在 AL 中；③ 由于该调整指令使用到 AH 寄存器，故应先将 AH 内容清零。

AAS 指令调整方法：<1> 如果 AL 寄存器的低 4 位为 0～9，且 AF 为 0，则跳过第<2>步，执行第<3>步；<2> 如果 AL 寄存器的低 4 位为 0AH～0FH 或 AF 为 1，则 AL 寄存器的内容减 06H，同时 AH 寄存器内容减 1，且将 AF 置 1；<3> AL 寄存器的高 4 位清零；<4> AF 位的值送 CF。

例如：

```
MOV     AX, 09              ; AH 清零，AL 中为非压缩 BCD 数
SUB     AL, 07
AAS
```

（6）比较指令 CMP

指令功能：完成两个操作数相减。

指令形式：

```
CMP     目的操作数，源操作数
```

说明： ① 该指令执行“目的操作数–源操作数”，与 SUB 指令不同的是，不产生运算结果，仅影响标志；② 指令的目的操作数不能是立即寻址；③ 目的操作数和源操作数不能同时为存储器操作数。

（7）取补指令 NEG

指令功能：0–目的操作数。

指令形式：

```
NEG     目的操作数
```

说明： ① 该指令执行“0–目的操作数”，因此除目的操作数为 0 外，CF 总是被置 1；② 操作数不能是立即寻址。

例如：设计一程序段，把 DX、AX 组成的 32 位数取补，其中 DX 为高十六位。

方案一：用减法指令。

```
MOV     BX, 0
SUB     BX, AX
MOV     AX, BX
MOV     BX, 0
SBB     BX, DX
MOV     DX, BX
```

方案二：用取补和减法指令。

```
NEG     DX                  ; 0-DX→DX
NEG     AX                  ; 0-AX→AX
SBB     DX, 0               ; DX-0-CF→DX
```

注意，此例是说明取补指令的使用，并没有考虑 DXAX 中 32 位数是正或是负，也就是说，此例是对一个 32 位数的“绝对取补”。实际上，NEG 指令就是“绝对取补”的指令。

3．算术乘法指令

（1）无符号数乘法指令 MUL

指令功能：完成两个操作数相乘。

指令形式：

MUL　源操作数

说明：① 8 位×8 位⇒16 位，16 位×16 位⇒32 位；② 乘数和被乘数都不能为立即寻址；③ 乘数或被乘数必须放在 AL 或 AX 中，在指令中则隐含；④ 16 位运算结果在 AX 中，32 位运算结果在 DX 和 AX 中。

例如，完成 3EH×5DH。

```
MOV     AL, 3EH
MOV     BL, 5DH
MUL     BL                ；执行 AL×BL
```

又如：

```
MOV     AL, 3EH
MUL     5DH               ；语法错误
```

（2）带符号数乘法指令 IMUL

指令功能：完成两个操作数相乘。

指令形式：

IMUL　源操作数

说明：① 8 位 × 8 位⇒16 位，16 位 × 16 位⇒32 位；② 乘数和被乘数都不能为立即寻址；③ 乘数或被乘数必须放在 AL 或 AX 中，在指令中则隐含；④ 16 位运算结果在 AX 中，32 位运算结果在 DX 和 AX 中；⑤ 有符号数在计算机中是其补码，且符号位也参加运算，此时用 MUL 指令就得不到正确结果，IMUL 指令则会将符号部分和数值部分分别进行处理。

（3）非压缩 BCD 数乘法操作结果校正 AAM

指令功能：完成两个非压缩 BCD 数乘法结果的十进制数调整。

指令形式：

AAM

说明：① 参加操作的数必须是非压缩 BCD 数；该指令用在 MUL 指令后；② AAM 指令调整方法：因为两个非压缩 BCD 数相乘的结果为 0～81，不会到十进制数的百位，因此调整方法就是将 AL 寄存器的内容除以 0AH，商放在 AH 寄存器（与除法指令不同）中，表示转换的十位数；余数放在 AL 寄存器（与除法指令不同）中，表示转换的个位数。

例如，完成 09H×07H 运算。

```
MOV     AL, 09H
MOV     DL, 07H
MUL     DL          ; AX=003FH
AAM                 ; AX=0603H
```

说明：该指令段中若不执行 AAM 指令，则 AX 中的结果不是非压缩 BCD 数，结果是十六进制数（3FH=63）。

4．算术除法指令

（1）无符号数除法指令 DIV

指令功能：完成两个操作数相除。

指令形式：

DIV　源操作数

说明： ① 指令中使用 16 位除 8 位，32 位除 16 位的格式，被除数不够 16 位或 32 位，则需扩展；② 被除数、除数都不能为立即寻址，除数必须是寄存器或存储器寻址；③ 被除数必须放在 AX 或 DX:AX 中，在指令中则隐含；④ 16 位运算的商放在 AL 中，余数放在 AH 中；⑤ 32 位运算的商放在 AX 中，余数放在 DX 中。

例如，完成无符号数 3E2CH÷5BH 运算。

```
MOV     AX , 3E2CH
MOV     BL , 5BH
DIV     BL
```

又如，完成无符号数 8A73H÷185BH 运算。

```
MOV     AX , 8A73H
MOV     DX , 0
MOV     CL , 185BH
DIV     CL
```

（2）带符号数除法指令 IDIV

指令功能：完成两个操作数相除。

指令形式：

```
IDIV    源操作数
```

说明： ① 指令中使用 16 位除 8 位，32 位除 16 位的格式，若被除数不够 16 位或 32 位，则需用扩展指令将其扩展；② 被除数、除数都不能为立即寻址，除数必须是寄存器或存储器寻址；③ 被除数必须放在 AX 或 DX:AX 中，在指令中则隐含；④ 16 位运算的商放在 AL 中，余数放在 AH 中；⑤ 32 位运算的商放在 AX 中，余数放在 DX 中。

（3）带符号数字节扩展指令 CBW

指令功能：将 AL 中的 8 位数扩展为 16 位数。

指令形式：

```
CBW
```

说明： ① 在带符号除法指令中，被除数要扩展成 16 位时使用该指令；② 其扩展规则是根据 AL 寄存器的最高位，若 D_7 为 1，则 AH 置 FFH，若 D_7 为 0，则 AH 置 0。

例如，在偏移地址为 2340H 和 2341H 内存单元中存有两个带符号字节数，求这两个数相除的结果，并将结果分别存入存储单元 2342H 和 2343H 中。

```
MOV     AL , [2340H]
CBW
IDIV    BYTE PTR [2341H]        ；符号 BYTE PTR 说明内存单元[2341H]中的数是字节型的
MOV     [2342H], AL
MOV     [2343H], AH
```

（4）带符号数字扩展指令 CWD

指令功能：将 AX 中的 16 位数扩展为 32 位数。

指令形式：

```
CWD
```

说明： ① 在带符号除法指令中，被除数要扩展成 32 位时使用该指令；② 其扩展规则是根据 AX 寄存器的最高位，若 D_{15} 为 1，则 DX 置 FFFFH；若 D_{15} 为 0，则 DX 置 0。

阅读下列程序段，指出其执行功能和执行后结果。

```
MOV     AX, 9EB2H
CWD
XOR     AX, DX
SUB     AX, DX
```

解读：

① 第二条指令是该程序段中的关键，它将 AX 中的字型数扩展为 32 位，扩展到 DX 的内容与 AX 的 D_{15} 位有关，该例中 DX 的内容为 FFFFH。第三条指令是对原 AX 内容求反，因为 AX 中的每一位都与“1”进行异或操作。第四条指令是对 AX 求反后加 1（实为减去–1）。

② 若 AX 的 D_{15} 位的内容为 0，则 DX 为 0。此时，执行第三条指令原 AX 内容不变，因为 AX 中的每一位都与“0”进行异或操作。第四条指令是 AX 内容减 0。

综上分析，该程序段执行对带符号数求绝对值，结果是 694EH。

（5）非压缩 BCD 数除法校正 AAD

指令功能：将两个非压缩 BCD 数除法操作调整为二进制数除法操作。

指令形式：

```
AAD
```

说明： ① 该操作要满足 16 位除 8 位的除法操作要求，即非压缩 BCD 数须放在 AX 寄存器中；② 该指令用在 DIV 指令之前，先调整，后做除法操作。

AAD 指令调整方法：将 AX 中的非压缩 BCD 数转换成二进制数，因为除数也是非压缩 BCD 数，必在 0～9 之间，实质是完成二进制数除法。

例如，完成非压缩 BCD 数 0304 除 5 的运算，结果为非压缩 BCD 数。

```
MOV     AX, 0304H
MOV     BL, 05
AAD                     ; AX=0022H
DIV     BL              ; 商(AL)=06，余数(AH)=04
```

该例实质是 34 除 5，AAD 指令将非压缩的 BCD 数 0304H 转换成二进制数，然后用二进制数除法规则进行除法操作。

3.4.3　位操作指令

位操作指令的共同点如下：

⊙ 可以按二进制位进行操作。

⊙ 逻辑运算指令按逻辑门电路的运算规则。

⊙ 逻辑移位指令有左移和右移，移出的位都进入 CF 标志。

⊙ 因移空位的补充方式不同，有多种指令形式。

⊙ 逻辑移位指令中，移动超过 1 次，则用 CL 寄存器作为计数器。

⊙ 执行逻辑操作指令，CF 均被清零。

1．逻辑运算指令

（1）逻辑求反指令 NOT

指令功能：将 8 位、16 位寄存器或存储器内容求反。

指令形式：

```
NOT     目的操作数
```