第3章 8086/8088 指令系统

引言：

每一系列的处理器都有自己的指令系统，可以说，指令系统功能的强弱大体上决定了计算机硬件系统功能的高低。本章以 8086/8088 CPU 指令系统为基础，介绍指令的一般概念和执行过程、CISC 和 RISC 指令的概念、寻址方式以及不同类型指令的功能。

教学目的：

(1) 了解指令的一般概念、指令的基本格式及指令的执行过程；

(2) 熟悉指令对操作数的各种寻址方式；

(3) 深入理解 8086 指令系统全部六大类指令的功能，包括指令操作码的含义、指令对操作数的要求和指令的执行结果。

3.1 概　　述

控制计算机完成指定操作并能够被计算机所识别的命令称为指令。一台计算机能够识别的所有指令的集合称为该机的指令系统。不同的计算机(或者说不同的微处理器)具有各自不同的指令系统。指令系统定义了计算机硬件所能完成的基本操作，其功能的强弱在一定程度上决定了硬件系统性能的高低。

Intel 8088/8086 CPU 指令系统也是 Intel 80x86 系列 CPU 的基本指令系统。由于 8086 和 8088 的指令系统完全相同，为叙述方便，以下统称为 8086 指令系统。

8086 CPU 与其上一代的 8 位 CPU 如 8080、8085 相比，其指令系统在指令的数量上、功能上、寻址方式的多样性上以及处理数据的能力上都有了很大的提高。例如，8086 不仅有加减法指令，还可用一条指令完成乘法或除法运算。此外它还增加了中断指令及串操作指令等。

8086/8088 CPU 的指令系统共包含 92 种基本指令，按照功能可将它们分为六大类：数据传送类、算术运算类、逻辑运算和移位、串操作、控制转移类、处理器控制。

为使读者对 8086/8088 指令系统有一个粗略的概念，表 3-1 列出了上述六大类指令中常用指令的助记符。更详细的内容将在 3.3 节中介绍。

表 3-1　8086/8088 CPU 常用指令一览表

指令类型		助　记　符
数据传送	一般数据传送	MOV、PUSH、POP、XCHG、XLAT、CBW、CWD
	输入/输出指令	IN、OUT
	地址传送指令	LEA、LDS、LES
	标志传送指令	LAHF、SAHF、PUSHF、POPF
算术运算	加法指令	ADD、ADC、INC
	减法指令	SUB、SBB、DEC、NEG、CMP
	乘法指令	MUL、IMUL
	除法指令	DIV、IDIV
	十进制调整指令	DAA、AAA、DAS、AAS、AAM、AAD
逻辑运算和移位指令		AND、OR、NOT、XOR、TEST、SHL、SAL、SHR、SAR、ROL、ROR、RCL、RCR
串操作		MOVS、CMPS、SCAS、LODS、STOS
控制转移指令		JMP、CALL、RET、LOOP、LOOPE、LOOPNE、INT、INTO、IRET 各类条件转移指令
处理器控制指令		见表 3-5

3.1.1　指令的基本构成

1. 指令的一般格式

一条指令通常由两个部分组成，如图 3-1 所示。第一部分为操作码(或称指令码)，用便于记忆的助记符表示(一般是英文单词的缩写)，用于指出指令要进行何种操作，因此是指令中必须给出的内容。另一部分是指令操作的对象，称为操作数，可根据不同的情况显式地给出或隐含存在。

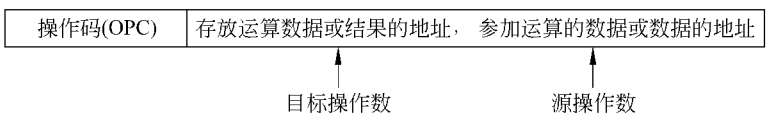

图 3-1　指令格式

指令的长度(所占的字节数)会影响指令的执行时间。8086 指令的长度在 1～7 个字节之间。操作码占用一个字节或两个字节。指令的长度主要决定于操作数的个数及所采用的寻址方式。在微处理器指令系统中，一条指令的操作数可以没有或有一个，但最多只能有两个。相应地，指令在格式上就有以下 3 种形式。

(1) 零操作数指令。指令在形式上只有操作码，操作数是隐含存在的。这类指令操作的对象通常为处理器本身。

(2) 单操作数指令。指令中仅给出一个操作数,另一个操作数隐含存在。

(3) 双操作数指令。格式如图 3-1 所示。

2. 指令中的操作数类型

8086 指令中的操作数主要有 3 种类型:立即数操作数、寄存器操作数和存储器操作数。

1) 立即数操作数

立即数是指具有固定数值的操作数,即常数,它不因指令的执行而发生变化。在 8086 系统中,立即数的字长可以是 1 字节或 2 字节;可以是无符号数或有符号数。要求数的取值范围必须符合相应字长数的规定,如果取值超出了规定的范围,就会发生错误。

在指令中,立即数操作数只能用作源操作数,而不能用作目标操作数。原因是立即数是一个常数,没有表示地址的含义。

2) 寄存器操作数

8086 CPU 的 8 个通用寄存器和 4 个段寄存器可以作为指令中的寄存器操作数,它们既可以作为源操作数,也可以用作目标操作数。

通用寄存器通常用来存放参加运算的数据或数据所在存储器单元的偏移地址。段寄存器用来存放当前操作数的段基地址。

仅有个别指令将标志寄存器 FLAGS 作为指令的操作数。

3) 存储器操作数

存储器操作数的含义是:参加运算的数据是存放在内存中的。由于 8086 指令系统中的操作数一般均为 8 位或 16 位字长,所以存储器操作数的字长也通常为字节或字,极个别的指令中有双字长的操作数。

存储器操作数在指令中既可作为源操作数,也可作为目标操作数。

第 2 章已经学习,能够唯一标识一个存储器单元的是它的物理地址,物理地址由段基地址和偏移地址两部分构成。所以,要寻找一个存储器操作数,必须首先确定操作数所在的逻辑段。一般情况下,若指令中没有明确指出操作数所在段,则 CPU 就采用默认的段寄存器来确定操作数的段基地址。各种存储器操作数所约定的默认段寄存器、段重设(即显式地指明段寄存器)所允许的段寄存器以及指令的有效地址所在寄存器请参见表 2-4。

存储器操作数的偏移地址(EA,Efficient Address)(也称有效地址)可以通过不同的寻址方式由指令给出。实际上,3.2 节中讲到的各种较复杂的寻址方式,大多都是针对存储器操作数的。

3.1.2 指令的执行时间

了解指令的执行时间,在有些时候是很重要的。例如在用软件产生定时或延时时,需要估算出一段程序的运行时间。另外,在某些实时控制要求较严或对程序运行时间要求较高的的场合,除需认真研究程序的算法外,对选择什么样的指令及采用什么样的寻址方

式也是很重要的。因为不同的指令在执行时间上有很大的差别，而不同的寻址方式其计算偏移地址所需时间也不同。由于指令的种类很多，要详细讨论各种指令的执行时间比较困难，这里只作一般的讨论。

一条指令的执行时间应包括取指令、取操作数、执行指令及传送结果几个部分，单位用时钟周期数表示。

不同指令的执行时间有较大的差别(见附录 C.1)。寄存器操作数占用的时间最短。存储器操作数的时间与采用的寻址方式有关，不同的寻址方式，计算偏移地址(EA)所需要的时间不同，其指令执行时间可能会相差很大。

在 3.1.1 节中讨论的 3 种类型的操作数中，寄存器操作数的指令执行速度最快，立即数操作数次之，存储器操作数指令的执行速度最慢。这是由于寄存器位于 CPU 的内部，执行寄存器操作数指令时，8086 的执行单元(EU)可以简捷地从 CPU 内部的寄存器中取得操作数，不需要访问内存，因此执行速度很快；立即数操作数作为指令的一部分，在取指时被 8086 总线接口单元(BIU)取出后存放在 BIU 的指令队列中，执行指令时也不需要访问内存，因而执行速度也比较快；而存储器操作数存放在内存单元中，为了取得操作数，首先要由总线接口单元计算出其所在单元的 20 位物理地址，然后再执行存储器的读写操作。所以相对前述两种操作数来说，指令的执行速度最慢。

以通用数据传送指令(MOV)为例，若 CPU 的时钟频率为 5MHz，即一个时钟周期为 0.2μs，则从寄存器到寄存器之间的传送指令的执行时间为

$$t=2\times0.2=0.4\mu s$$

立即数传送到寄存器的指令执行时间为

$$t=4\times0.2=0.8\mu s$$

而存储器到寄存器的字节传送，设存储器采用基址—变址寻址方式，则指令执行时间为

$$t=(8+EA)\times0.2=(8+8)\times0.2=3.2\mu s$$

3.1.3 CISC 和 RISC 指令系统

不同系列的 CPU 有不同的指令系统。目前，指令系统的设计有两个完全不同的方向。一个称为复杂指令系统计算机(CISC，Complex Instruction Set Computer)，另一个是 20 世纪 80 年代新发展起来的、以简化指令功能为主要目的的精简指令系统计算机(RISC，Reduced Instruction Set Computer)。

1. CISC 指令

不同系列的 CPU 有不同的指令系统，每一种 CPU 都有属于它自己的指令系统。CPU 正是通过执行一系列特定的指令来满足应用程序的特定要求的。CISC 指令的设计目标是增强指令的功能，将一些原来用软件实现的、常用的功能变成用硬件的指令系统来实现。例如，在科学计算的应用程序中，经常要计算各种各样的函数，有些计算机系统就设置了一些常用的函数运算指令，用一条指令代替软件的一个子程序来完成函数计算。

随着超大规模集成电路(VLSI)技术的发展,计算机硬件的成本不断下降,而软件成本却不断地上升,操作系统的效率和微机的性能的进一步提高促使整个指令系统在功能上有以下的改进。

(1) 在指令系统中增加更多的指令和功能更强的复杂的指令,将使用频率高的指令串用一条新的指令去取代,将使用频率高的指令用硬件加快其执行。这样就使得程序的长度和执行时间都得以缩短。

(2) 增加对高级语言和编译程序支持的指令的功能,以减少编译时间,缩短目标程序的长度,进一步降低软件成本。

(3) 尽可能缩机器语言与高级语言的差距。众所周知,编译程序的作用就是把由高级语言编写的语句翻译成一个机器指令序列,如若机器指令与高级语言的语句相类似,编译程序的任务就简单多了。这样走到极端,就是将高级语言与机器语言合二为一,构成所谓的高级语言计算机。

(4) 增加对操作系统支持的指令,以实现对操作系统的优化。有些支持操作系统的指令属于特权指令,对一般用户不公开。这类指令中,有些指令的使用频率并不高,但如果没有它们的支持,操作系统将很难实现,如处理机转换、进程切换等方面所使用的指令。

为使新的微机与其前代机在软件上兼容,指令系统只能扩充,不能减少,从而使得微机的指令系统越来越复杂。如在 Pentium 微处理机指令系统内不仅继承下它的前辈机的所有指令,而且又增加了 Cache 的指令和诸如 8 字节比较和交换等指令,指令数达 300 余条。

复杂指令难以使用这是一个不争的事实。因为编译程序必须使每一条由高级语言编写的语句经编译后,满足所生成的指令代码的长度最小、指令执行的次数最少、适合流水线操作等诸多优化所生成指令的条件。所以使用复杂指令系统是一件并不轻松的工作,尤其是非计算机专业的人士。

CISC 也有许多优点,如指令经编译后生成的指令程序较小、执行起来较快、节省硬件资源、存取指令的次数少、占用较少的存储器等。

2. RISC 指令

从计算机诞生之日起,人们就在不断地尝试着对计算机的结构和指令系统进行改进。20 世纪 70 年代,美国加州伯克利分校开始了对 CISC 指令系统合理性问题的研究,归纳出 CISC 指令系统存在以下 3 个方面的问题。

(1) "8020 规律":即在 CISC 指令系统的计算机中,20%的指令在各种应用程序中的出现频率占整个指令系统的 80%。

(2) CISC 指令系统中有大量的复杂指令,控制逻辑极不规整,给 VLSI 工艺造成很大的困难。

(3) CISC 中增加了许多复杂指令,这些指令虽然简化了目标程序、缩小了高级语言与机器语言之间的差距,但使程序总的执行时间变长、硬件的复杂度增加。

基于这些研究,人们提出了精简指令系统计算机(RISC)。RISC 目前还是一种计算机体系结构的设计思想,不是一种产品,它是近代计算机体系结构发展史中的一个里程

碑。它的核心思想是通过简化指令来使计算机的结构更加简单、合理，从而提高 CPU 的运算速度。卡内基·梅隆(Carnegie Mellon)大学对 RISC 的特点给出了一个较为明确的描述。

(1) 大多数指令在一个计算机周期内完成。所谓计算机周期，是指由寄存器取两个操作数并完成一次算术逻辑运算操作，然后再将运算结果写入寄存器所需的时间。

(2) 因为访问存储器指令需要的时间比较长，因此指令系统中应尽量减少这类指令，而采用寄存器与寄存器之间的操作。

(3) 减少寻址方式的种类。在一个 RISC 内，几乎所有的指令都使用寄存器寻址方式。其他的更为复杂的寻址方式可以通过软件的方法用这些简单的寻址方式予以合成来解决。

(4) 减少指令的种类。指令系统中的大多数指令只执行一个简单的和基本的功能。对复杂的功能，可通过软件编程的方法解决。

(5) 指令格式简单。通常 RISC 仅配备有一种或少数几种指令格式，且指令长度是固定的，并与字节的边界对准；字段位置，特别是操作码字段的位置是固定的。这样处理的好处是：对固定字段、对操作码的译码和对寄存器操作数的访问可同时进行；简化了指令的格式，也就简化了控制器；同时，以字长的单位来取指令和数据，取指令操作过程也就被优化了。

总之，RISC 的特点是简化了计算机的指令系统，进而简化了控制器。如一个 RISC 指令系统可以只有一条或两条 ADD 指令(仅有整数加、带进位加)，而 CISC 结构的 Pentium 微处理机仅加法指令就有 4 条。

虽然 RISC 指令功能简单，复杂功能需要用软件编程去实现，但经过技术测试比较，处于同样工艺水平的芯片，RISC 的运算速度要比 CISC 快 3～5 倍。

设计 RISC 类计算机的目的是提高整个系统的性能。要达到这个目的，必须要有相应的技术支持。①要求大多数操作使用寄存器操作数，从根本上提高 CPU 的运算速度；②指令采用流水线工作方式，取指令和执行指令并行执行，并通过相应的技术手段使流水线尽量不“断流”。具体地说，就是一条指令的执行是由若干个不同的功能子部件分别完成的，流水线中的若干个功能子部件按照指令的执行步骤各自完成自己的操作。如果在程序执行过程中遇到后一条指令要用到前一条指令的执行结果或程序转移情况等，可通过编译程序予以解决，即在编译程序对用高级语言编写的应用程序进行编译时，事先把机器指令的执行顺序安排好，以便最大限度地挖掘流水线的能力。

RISC 类微处理器对存储器的结构和存取速度要求很高，所以在 RISC 系统中一定要采用 Cache，以便减少争用 RISC 芯片的要求。

3.2 寻址方式

所谓寻址方式，主要是指获得操作数所在的地址的方法。在 8088/8086 系统中，一般将寻址方式分为两种不同的类型：①寻找操作数的地址；②寻找要执行的下一条指令的

地址，即程序的地址。后者主要在程序转移或过程调用时用来寻找目标地址或入口地址，这将在调用指令(CALL)和程序转移指令(JMP)中介绍。在 3.2 中，主要讨论针对操作数地址的寻址方式，并且如无特殊声明，讨论的对象主要是源操作数。

在 8086 指令系统中，说明操作数所在地址的寻址方式可分为 8 种，了解什么样的寻址方式适用于什么样的指令，对于正确理解和合理使用指令是很重要的。

3.2.1 立即寻址

立即寻址(Immediate Addressing)方式只针对源操作数。此时源操作数是一个立即数，它作为指令的一部分，紧跟在指令的操作码之后、存放于内存的代码段中，在 CPU 取指令时随指令码一起取出并直接参加运算。这里的立即数可以是 8 位或 16 位的整数。若为 16 位，则存放时低 8 位在低地址单元存放，高 8 位在高地址单元存放，如图 3-2 所示。

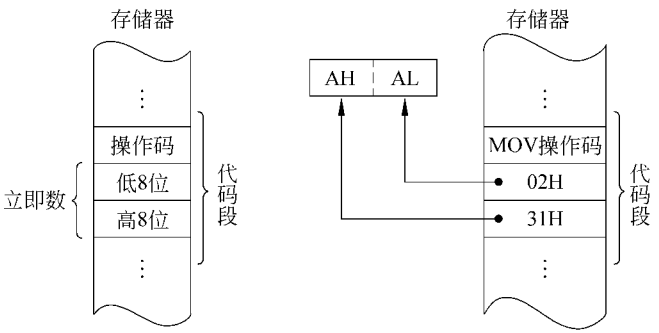

图 3-2 立即寻址方式示意图

【例 3-1】 指令“MOV AX,3102H”表示将 16 位的立即数 3102H 送入累加器 AX。指令执行后，AH＝31H，AL＝02H。

这是一条三字节指令，其执行情况示意图如图 3-2 所示。立即寻址方式主要用于给寄存器或存储单元赋初值。

3.2.2 直接寻址

直接寻址(Direct Addressing)方式表示参加运算的数据存放在内存中，存放的地址由指令直接给出，即指令中的操作数是存储器操作数。“[]”内用 16 位常数表示存放数据的偏移地址，数据的段基地址默认为数据段，可以允许段重设。

【例 3-2】 指令“MOV AX,[3102H]”表示将数据段中偏移地址为 3102H 和 3103H 两单元的内容送到 AX 中。

假设 DS＝2000H，则所寻找的操作数的物理地址为

$$20000H+3102H=23102H。$$

指令的执行情况如图 3-3 所示。

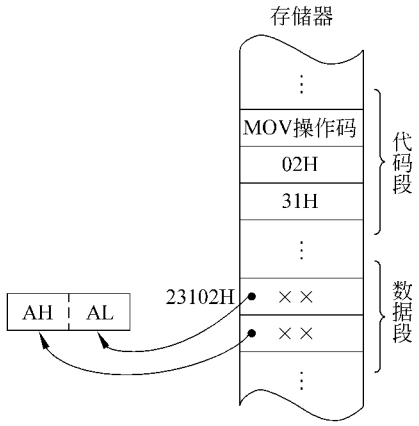

图 3-3　直接寻址方式

要注意区别直接寻址指令与前面介绍的立即寻址指令二者的不同。直接寻址指令中的数值是操作数的 16 位偏移地址，而不是数据本身。为了区分二者，指令系统规定偏移地址必须用方括号括起来。如在例 3-2 中，指令的执行不是将立即数 3102H 送到累加器 AX，而是将偏移地址为 3102H 的内存单元中的内容送到 AX。若操作数不是存放在 DS 段，则在指令中要用段重设符号加以声明。

【例 3-3】 指令“MOV BL，ES：[1200H]”表示将 ES 段中偏移地址为 1200H 单元的内容送到 BL 寄存器中。

在汇编语言中，有时也用一个符号来代替数值以表示操作数的偏移地址，通常把这个符号称为符号地址。例 3-3 中，若用 BUFFER 代替偏移地址 1200H，则指令可写成

```
MOV BL,ES:[BUFFER]
```

这两者是等效的，但 BUFFER 必须在程序的开始处予以定义，这点将在第 4 章中介绍。

3.2.3　寄存器寻址

在寄存器寻址(Register Addressing)方式下，指令的操作数为 CPU 的内部寄存器。它们可以是数据寄存器(8 位或 16 位)，也可以是地址指针、变址寄存器或段寄存器。

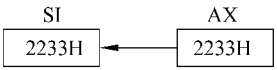

图 3-4　寄存器寻址示意图

【例 3-4】 指令“MOV SI，AX”表示将 AX 的内容送到寄存器 SI 中。若指令执行前 AX＝2233H，SI＝4455H，则指令执行后 SI＝2233H，而 AX 中的内容保持不变，如图 3-4 所示。

采用寄存器寻址方式，虽然指令操作码在代码段中，但操作数在内部寄存器中，指令执行时不必通过访问内存就可取得操作数，故执行速度较快。

3.2.4　寄存器间接寻址

寄存器间接寻址(Register Indirect Addressing)是用寄存器的内容表示操作数的偏移地址。此时寄存器中的内容不再是操作数本身，而是存放数据的偏移地址，操作数本身在内存储器中。

寄存器间接寻址方式中存放操作数偏移地址的寄存器只允许是 SI、DI、BX 和 BP，它

们可简称为间址寄存器或称为地址指针。选择不同的间址寄存器涉及的段寄存器不同。在默认情况下，选择 SI、DI、BX 作间址寄存器时，操作数在数据段，段基地址由 DS 决定；选择 BP 作间址寄存器，则操作数在堆栈段，段基地址由 SS 决定。但无论选择哪一个间址寄存器都允许段重设，可在指令中用段重设符指明当前操作数在哪一个段。

因为间址寄存器中存放的是操作数的偏移地址，所以指令中的间址寄存器必须加上方括号，以避免与寄存器寻址指令混淆。

【例 3-5】 已知 DS＝6000H，SI＝1200H，执行指令：MOV AX,[SI]。

因为指令中没有指定段重设，所以寻址时使用默认的段寄存器 DS。由已知条件可计算出操作数的物理地址＝60000H＋1200H＝61200H。指令执行情况如图 3-5 所示。

执行结果：AX＝3344H。

若操作数存放在附加段，则本例中的指令应表示成以下形式：

```
MOV AX,ES:[SI]
```

例 3-5 中，若间址寄存器采用 BP，则操作数默认存放在堆栈段。

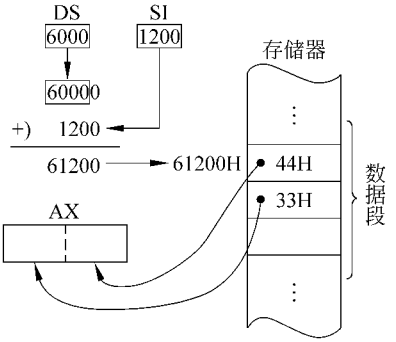

图 3-5 寄存器间接寻址示意图

【例 3-6】 若已知 SS＝8000H，BP＝0200H，指令“MOV BX,[BP]”执行后：BL＝[80200H]单元中的内容，BH＝[80201H] 单元中的内容。

有些书中又将使用 BX、BP 作为间址寄存器的寄存器寻址方式称为基址寻址方式；而将使用 SI、DI 作为间址寄存器的寄存器寻址方式称为变址寻址方式。

3.2.5 寄存器相对寻址

在寄存器相对寻址方式中，操作数在内存中的存放地址（偏移地址）由间址寄存器的内容加上指令中给出的一个 8 位或 16 位的位移量组成。操作数所在段由所使用的间址寄存器决定（规则与寄存器间接寻址方式相同）。因位移量可看作相对值，故把这种带位移量的寄存器间接寻址方式称为寄存器相对寻址。

【例 3-7】 指令 MOV AX,DATA[BX]的寻址过程示例。

设：DS＝6000H，BX＝1000H，DATA＝0008H。

则操作数所在单元的物理地址＝60000H＋1000H＋0008H＝61008H。

执行结果：AX＝5566H。

指令的执行情况如图 3-6 所示。

寄存器相对寻址常用于存取表格或一维数组中的元素——把表格的起始地址作为位移量，元素的下标值放在间址寄存器中（反过来也可以）。这样，就可存取表格中的任意一个元素。

【例 3-8】 某数据表的首地址（偏移地址）为 TABLE，要取出该表中的第 10 个字节并存放到 AL 中，可用如下指令段实现（注意位移量是从 0 开始的）：

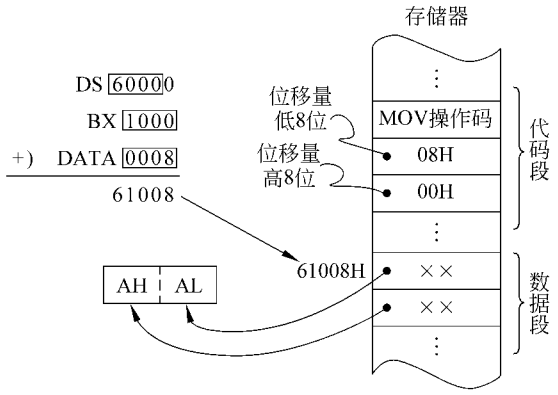

图 3-6　寄存器相对寻址示意图

```
MOV SI,9                          ;第 10 个数的位移量为 9
MOV AL,[TABLE+SI]                 ;第 10 个数的偏移地址为 TABLE+9
```

在汇编语言中，相对寻址指令的书写格式允许有几种不同的形式。例如，以下几种写法实质上是完全等价的。

```
MOV AL,DATA[SI]
MOV AL,[SI]DATA
MOV AL,DATA+[SI]
MOV AL,[SI]+DATA
MOV AL,[DATA+SI]
MOV AL,[SI+DATA]
```

3.2.6　基址—变址寻址

基址—变址寻址方式由一个基址寄存器(BX 或 BP)的内容和一个变址寄存器(SI 或 DI)的内容相加而形成操作数的偏移地址，称为基址—变址寻址。在默认的情况下，指令中若用 BX 作基址寄存器，则段地址在 DS 中；如果用 BP 作基址寄存器，则段地址在 SS 中，但允许使用段重设。

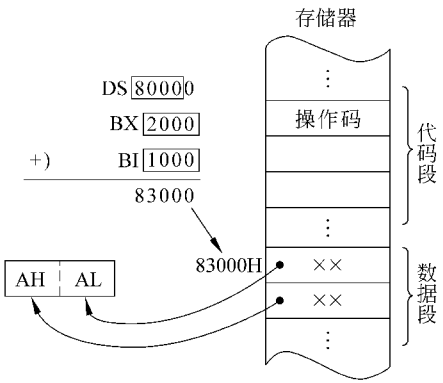

图 3-7　基址—变址寻址示意图

【例 3-9】　指令 MOV AX,[BX][SI]的寻址过程如图 3-7 所示。

设：DS=8000H,BX=2000H,SI=1000H。

则操作数的物理地址＝80000H＋2000H＋1000H＝83000H。

指令执行后：AL＝[83000H]，AH＝[83001H]。