CHAPTER 3

第3章

8086微处理器及其寻址方式

由于汇编语言是面向机器的语言，它和高级语言相比，最大的优越性在于操作灵活，即可以直接作用到硬件的最底层，这是高级语言所不能实现的。但同时也带来许多新问题，即在进行汇编语言程序设计时，一个程序设计者必须对一个微处理器的结构有比较全面和十分明确的了解。从汇编语言的角度看，实模式下 80x86 和 8086 没有多大差异，且 Intel 的 80x86 指令系统向上兼容。本章对 8086 作较为详细的介绍。

3.1 8086CPU 的基本逻辑结构

3.1.1 8086 的基本逻辑结构

8086CPU 的结构是一次划时代的革命，其基础的逻辑结构一直沿用至今，就算是最新型的酷睿多核 CPU，使用的也是地址、数据及控制三总线的协同工作方法。不管哪种 CPU，在工作时都是围绕一个核心，即对某个指定的“地址”进行操作，存取数或者运算。这个地址可以是内存的某个地址、某个外设端口，甚至是某个通用寄存器。关于通用寄存器，例如 AX、BX，我们可以理解为在 CPU 内部的一个可唯一寻址的地址。在这种意义下，“地址”当然也可以是某个 CPU 内部的寄存器。

“对地址中数的操作是计算机工作的核心。”随着大家对汇编语言的深入学习，特别是将来对微机原理系统地学习，理解并且在脑海中建立了计算机体系结构的整体框架后，会更加深入地理解这句话。

图 3-1 是 8086CPU 的基本逻辑结构。只有理解了这个逻辑结构，才能得心应手地用汇编语言灵活地编制各种程序。

8086 微处理器从功能上来说可以分为并行工作的两大部分：执行单元（Execution Unit，EU）和总线接口单元（Bus Interface Unit，BIU）。总线接口部件的功能是负责与存储器、I/O 端口传送数据。具体来说，总线接口部件要从内存取指令送到指令队列，CPU 执行指令时，总线接口部件要配合执行部件从指定的内存单元或者外设端口中取数据，将数据传送给执行部件，或者把执行部件的操作结果传送到指定的内存单元或外设端口中。执行部件的功能是负责指令的执行。

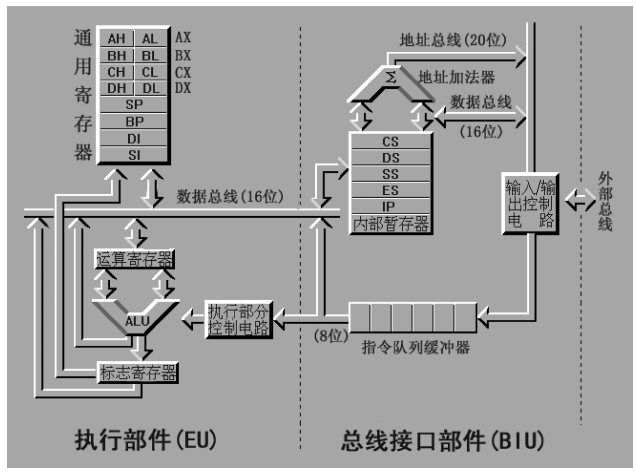


图 3-1 8086 微处理器功能结构示意图

EU 从 BIU 的指令队列中获取指令，译码后开始执行，当指令要求访问内存或端口时，EU 请求 BIU 存取数据。这时 EU 算出的地址是相对段基的 16 位位移量，BIU 根据指令要求选定相应的段寄存器得到段基，然后再由段基和位移得出操作数的 20 位物理地址。

EU 执行指令时，BIU 从存储器中取出后继指令并送入指令队列，该队列同时能存放 6 字节指令。在绝大多数情况下，这一指令队列能保证 EU 从该队列中取出马上要执行的指令，而不必到内存中直接取指令。

3.1.2 理解并运用基本逻辑结构图

结合这个逻辑结构来理解性地记忆计算机的指令，可以帮助我们理解指令中操作数与被操作数的限制。

比如，指令“MOV DS，200H”为什么是错误的呢？而指令“MOV AX，200H”和“MOV DS，AX”为什么就可以呢？尽管这是由硬件工程师设计的 CPU 的结构决定的，要追根究底是非常复杂的。不过，通过图 3-1，我们还是可以寻找到一些规律，达到理解的目的。200H 是立即数，也就是说包含在指令中的数据，仅能被直接送到 AX、BX、CX、DX 等通用寄存器，而想要改变 DS 的值必须通过连接总线接口单元(BIU)及执行单元(EU)的内部数据总线，或是连接总线接口单元及外部总线之间的数据总线来进行。所以，想要改变 DS 的值，必须由 AX(通常就用 AX)等通用寄存器来中转。

3.1.3 8086CPU 的运行特点

8086CPU 运行时有以下特点：

(1) 内存分段组织，并处于一个大的线性地址空间内被随机访问。

(2) 程序指令必须先被调入内存才能得以执行。

(3) 指令代码与程序中的数据混合存储，靠 CS 及 DS 等段寄存器标明指令及数据性质。

(4) 指令在 CPU 内串行执行,每一个瞬间,CPU 指向并执行一条指令。

(5) CPU 靠地址总线、数据总线及控制总线协调各部件的工作。

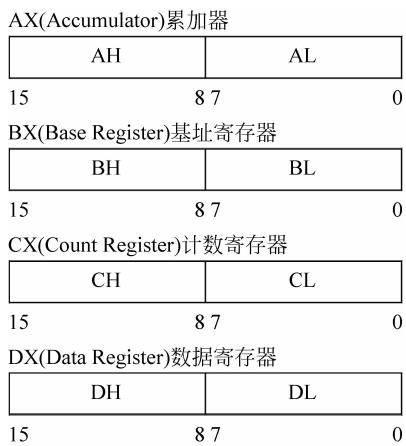
3.1.4 8086 寄存器组

寄存器是 CPU 内部存放操作数的地方,它的操作速度比内存操作速度快。从图 3-1 可以看出 8086 共有 14 个寄存器,这些寄存器都是 16 位的,可以存放 16 位二进制数。8086 的 14 个寄存器依功能分为三大组:第一组为通用寄存器,共 8 个;第二组为段寄存器,共 4 个;第三组为控制寄存器,共两个。

1. 通用寄存器

通用寄存器包括数据寄存器、指针寄存器和变址寄存器。

1) 数据寄存器



这 4 个数据寄存器中,每一个可单独作为 16 位的寄存器 AX、BX、CX、DX 使用,也可作为 8 位寄存器使用。作为 8 位寄存器使用时,每一个又可作为两个寄存器。例如,AX 的高 8 位称做 AH 寄存器,而低 8 位称做 AL 寄存器,类似地,有 BH、BL、CH、CL、DH、DL。其中作为累加器的是 AX(16 位)和 AL(8 位),其他用以存放操作数和中间结果,一般均可参与算术及逻辑运算,而且可以互换。

如果把一个字(Word)存入一个 16 位的寄存器,那么这个字的高位字节和低位字节会分别存放在这个寄存器的高 8 位寄存器和低 8 位寄存器中。例如,将一个字 0001101000001010B(6666D)存入 AX 寄存器,那么 AH 中存储了它的高 8 位 00011010B(26D),AL 中存储了它的低 8 位 00001010B(10D)。

在 8086 汇编语言中,一些汇编指令虽然在书写指令时没有写相应寄存器的名字,但该指令本身就决定了使用某一特定的寄存器,这时称其为隐含使用某某寄存器。例如,乘法指令“MUL BL”就隐含使用 AL 或 AX 寄存器。寄存器作为一些特殊用途使用时,则必须使用其中某些特定的寄存器,详见表 3-1。

表 3-1 数据寄存器的隐含使用及特殊用途

寄存器	用　途
AX	乘法指令作累加器,I/O 指令作数据寄存器 由累加器作立即数运算,比其他指令少占字节
AL	BCD 码、ASCII 码运算的累加器,查表指令 XLAT 的累加器 字节乘法,字节除法,字节 I/O,十进制算术运算
AH	作为取指令标志 LAHF 的目的寄存器,字节乘法,字节除法
BX	间接寻址作基址寄存器,XLAT 指令中作基址寄存器
CX	串操作或循环控制中的计数器,指令执行后内容会自动变化
CL	移位,循环计数,指令执行后内容不变
DX	字乘法,字除法,I/O 指令中作端口间接寻址
SI	存储器指针(串操作中的源指针)
DI	存储器指针(串操作中的目的指针)
BP	存储器指针(存取堆栈的指针)
SP	堆栈指针

2) 指针和变址寄存器

指针和变址寄存器都是 16 位的寄存器,且只能作为 16 位寄存器使用。其中,指针寄存器有两个:BP(Base Pointer Register,基址指针寄存器)和 SP(Stack Pointer Register,堆栈指针寄存器);变址寄存器也有两个:SI(Source Index Register,源变址寄存器)和 DI(Destination Index Register,目标变址寄存器)。

2. 段寄存器

8086 在取指令和寻找内存操作数的地址时,采用了分段寻找的方式,在同一时刻可将内存分成 4 个物理段:代码段、数据段、堆栈段和特别数据段,段与段允许有交。4 个段寄存器分别是 CS(Code Segment Register,代码分段寄存器)、DS(Data Segment Register,数据分段寄存器)、SS(Stack Segment Register,堆栈分段寄存器)和 ES(Extra Segment Register,特别分段寄存器)。这 4 个寄存器都只能作为 16 位寄存器使用。

3. 控制寄存器

控制寄存器有两个:一个是指令指针寄存器(IP Register,IP);另一个是状态标志寄存器(FLAG Register,FLAG)。

IP 和 CS 用作指令寻址,由 CS 和 IP 共同决定当前要取指令的地址,而且每执行一条指令,CS 和 IP 会自动指向下一条指令。

FLAG 的各位用来存放各种不同的标志,如图 3-2 所示。算术运算指令和逻辑运算指令的运行结果都将定性地反映在不同的标志位上,以便后续的条件判断指令根据这些标志实现判断转移,判断转移的实质就是修改 CS 和 IP。这也正是计算机能够实现判断转移的底层原理。

				OF	DF	IF	TF	SF	ZF		AF		PF		CF
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

图 3-2 标志寄存器

标志寄存器的每一位可以理解为CPU内部的一盏灯，而这些灯的亮与灭由运算结果决定，因此后续指令可以根据灯的状态来实现判断转移。

标志寄存器各位的状态表示如下。

CF：进位标志，在进行字/字节运算产生进位或借位时置1，否则置0。

PF：奇偶性标志，结果有偶数位个1时置1，否则置0，用以检验传送结果是否有误。

AF：辅助进位标志，当进行字节运算由低4位向高4位进位或借位时置1，否则置0。在作BCD码运算时常常使用。

ZF：零标志，当运算结果为0时置1，否则置0。

SF：符号标志，当运算结果为负，即结果最高位为1时置1，否则置0。

TF：陷阱标志，若TF=1，则在执行指令时产生单步中断。

IF：中断标志，若IF=1开中断，响应可屏蔽中断；IF=0，关中断。

DF：方向标志，DF置1引起串操作指令的变址寄存器自动减值，DF置0引起串操作指令的变址寄存器自动增值。

OF：溢出标志，运算溢出时自动置1，当它为1时可用溢出中断指令产生中断。

*3.2 指令与数据

根据冯·诺依曼原理，计算机的特点之一就是一切以内存为中心。指令与数据存放在同样的内存空间。这种计算机的体系结构一直沿用至今，即使使用最新型的多核CPU的计算机也依然如此。

那么计算机是靠什么来区分指令与数据的呢？

例如，某个内存单元的状态为01000011B，即存放着二进制数01000011B。如果将其理解为八进制数，就是103Q；如果将其理解为十进制数，就是67；如果将其理解为一个ASCII字符，那就代表字符C；如果将其发送到某个外设端口，可能就是某个外部设备的控制字，或者是某种设备的数据。可见，同样是一个地址存放的01000011B二进制数，在不同的应用场合，它代表着完全不同的含义。作为数据是这样，作为指令也是这样。还是这个二进制数01000011B，如果将其作为指令送入CPU的指令队列，进一步送入译码器去译码并且执行，那它就是寄存器BX加一指令INC BX。表3-2是某个内存单元存放的01000011B的不同含义。

表3-2 一个内存单元状态01000011B的不同含义

状态、数据、指令	具体表示	含义
内存字节状态	01000011	一个字节的内存单元存放的内容
ASCII字符	'C'	表示大写字母C
十六进制	43H	表示十六进制的数值43
八进制	103Q	表示八进制的数值103
十进制	67D	表示十进制的数值67
二进制	01000011B	表示二进制数01000011B
开关量	01000011	表示一系统开关量：关开关关关关开开，可用来实现控制
机器指令	INC BX	机器指令，表示BX的内容加1结果放回BX

指令与数据存放在同样的内存空间中，没有额外的标记来指明这些是指令还是数据。内存单元中存放的究竟是指令还是数据，不是由存储单元的性质决定的，而是由 CPU 的寻址决定的。当通过段寄存器 CS 与指令指针寄存器 IP 指向该内存地址时，CPU 将载入以该地址为起始存放单元的一条指令来执行，这时该地址存放的内容就是这条指令，或者是这条指令的一部分。所以说是一条指令或指令的一部分，是因为 80x86 的指令是不等长指令，根据指令的功能不同，从一个字节到 6 个字节不等。同样的内存单元，当通过内存操作数的寻址方式寻找到该地址时，这个单元的内容就将作为一个数据来使用，至于是何种形式的数据，完全以程序设计者的安排进行解释。至于深入内涵的理解，需要通过寻址规则的掌握和大量程序设计的过程来加深理解。

如果将存放数据的地址用作程序来执行会出现什么结果？答案应该是明确的。尽管内存的存储单元既可以存放数据也可以存放程序指令，可是我们必须保证被代码段 CS 及指令指针寄存器 IP 指向的单元所存放的应该是有意义的 CPU 指令，否则 CPU 在载入这些二进制代码解释执行时，必然出现无法预料的结果：死机、重启或其他。

3.3 8086 的存储器分段结构

Intel 公司从 8086 开始采用分段的方式管理存储器。它把内存和程序分成若干个段，每个段的起点用一个段寄存器来记忆。所以，学习汇编语言必须清楚地理解存储器的分段含义、存储单元的逻辑地址和其物理地址之间的转换关系。

3.3.1 8086 的存储器分段

计算机的内存单元是以“字节”为最小单位进行线性编址的。为了标识每个存储单元，就给每个存储单元规定一个编号，此编号就是该存储单元的物理地址。这个编号通常用十六进制来表示。

1. 物理段

8086 内部有 20 根地址线，其编码区间为 00000H～0FFFFFH，所以，在同一时刻，它可以直接访问的物理空间为 1M(2^{20})B。而 CPU 内部存放存储单元偏移量的寄存器(如 IP、SP、BP、SI、DI 和 BX 等)都是 16 位，它们的编码范围仅为 0000H～0FFFFH。这样，如果用 16 位寄存器来访问内存的话，则只能访问内存的最低端的 64KB，其他内存单元将无法访问。为了能用 16 位寄存器来有效地访问 1MB 的存储空间，8086 采用了内存分段的管理模式，并引用段寄存器的概念。

存储器分段就是 CPU 把内存空间划分成若干个段，由于寄存器字长为 16 位，所以规定每个段的最大容量为 64KB。这样每个存储单元就可以用“段地址：偏移地址”的逻辑地址来表示其准确的物理位置，而且段地址和偏移地址都可以用 16 位寄存器表达。

在程序执行的瞬间，内存分为 4 个物理段：代码段、数据段、堆栈段和特别数据段。这 4 个段的段地址分别由段寄存器 CS、DS、SS、ES 给出，4 个物理段可以彼此相交、重叠或分离，

如图 3-3 所示。

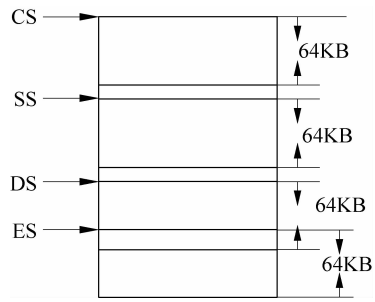


图 3-3　当前物理段示意图

段地址的计算方法是：

段寄存器内容 × 10H

例如，我们用(CS)表示 CS 的内容，当(CS)＝0A001H 时，代码段的段地址为：

0A001H × 10H ＝ 0A0010H

同样道理，如果执行程序的瞬间(CS)＝0A001H，(DS)＝200AH，(SS)＝0B20EH，(ES)＝3000H，我们可以说这时代码段、数据段、堆栈段、特别数据段的段地址分别为 0A0010H、200A0H、0B20E0H、30000H。可见段寄存器的内容唯一决定了当前该段的地址，因此在不引起混淆的情况下，也往往将段寄存器的内容称做段地址。

于是可以得到两个结论。结论一：在 00000H～0FFFFFH 的字节空间中，能够作为段地址的物理地址一定是能够被 16 整除的地址，这是由段地址的形成规则，即计算方法所决定的。结论二：在不同时刻，只要修改段寄存器的内容，段地址就可以指向 00000H～0FFFFFH 中任何一个能够被 16 整除的位置。

2. 物理地址的形成方式

从图 3-1 中的总线接口单元部分，我们可以看到有一个地址加法器的部件，正是这个部件使得 8086 系统的 16 位的寄存器可以寻址 20 位的内存地址空间，那么具体是怎么实现的呢？

为了加深理解 8086 内存分段的管理模式和了解物理地址的形成方式，在此举一个形象点的例子。假如有一幢 10 层大楼，如果每一层有 64 个房间，楼层号和每一层的房间号用两位数字表示，如表 3-3 所示。

表 3-3　房间编号

楼层号	房间号		房间号		房间号
01	01	…	27	…	64
02	01	…	27	…	64
03	01	…	27	…	64
⋮	⋮	⋮	⋮	⋮	⋮
10	01	…	27	…	64

如果要找 3 楼的 27 号房间，也就是 0327 房间，只需要上到 3 楼，并找到房间号为 27 的那个房间就可以了(从 01 号房间顺序查找)。在 0327 这个房间号信息表达的过程中，我们发现，层号 03 自动地左移了两位，也就是说自动地乘上了 100，变成了 0300，然后再加上房间号就构成了 0327。这样，用楼层表示的两位数和层内房间号的两位数，解决了一个 4 位编解码的问题。

段寄存器的内容就相当于楼层号，段内位移就相当于楼层内的房间号。8086 的寻址问题，就是用两个 16 位解决 20 位地址的问题。解决的办法是首先将段寄存器的内容由 16 位

变成 20 位，末尾 4 位为零，然后加上段内 16 位位移量构成。

例如，(CS)＝0A001H＝1010，0000，0000，0001B，变成 20 位：1010，0000，0000，0001，0000B，即：

$$(CS) \times 10H = 0A0010H$$

如果这时：

$$(IP) = 102H$$

则当前指令的物理地址为：

$$(CS) \times 10H + (IP) = 0A0112H$$

我们通常把内存操作数所在存储单元的实际地址与其所在段的段地址之间的距离称做段内位移量，也称做有效地址(Effective Address，EA)或偏移量(Offset)等。有了段地址和偏移量，就能唯一地确定某一内存单元在存储器内的具体位置。所谓寻址，除指令所在地址的寻址(指令寻址用 CS 和 IP 得到)、堆栈寻址(堆栈寻址用 SS 和 SP 得到)外，主要是指内存操作数的寻址，其实质就是内存操作数段地址的确定和有效地址(EA)的计算方法。

段地址也称做段基地址或段基址。

由此可见，内存操作数的物理地址分为两部分：段地址和偏移量，其计算方法如下：

物理地址 ＝ 段寄存器内容 × 10H ＋ 偏移量 ＝ 段地址 ＋ 偏移量

无论是指令、堆栈，还是内存操作数，其所在物理地址的计算都可以用“左移 4 位”和“加”运算来实现。图 3-4 和图 3-5 是物理地址的计算示意图。

图 3-4　物理地址的形成(1)

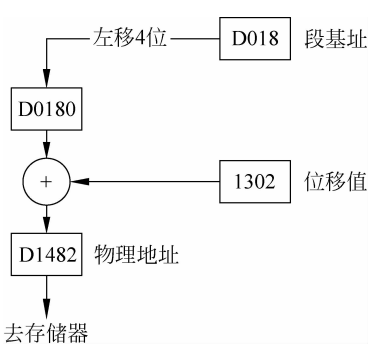


图 3-5　物理地址的形成(2)

对物理地址来说，当段地址变化时，只要对其偏移量进行相应的调整就可对应同一个物理地址。所以，同一个物理地址可有多个逻辑地址，如图 3-6 所示。

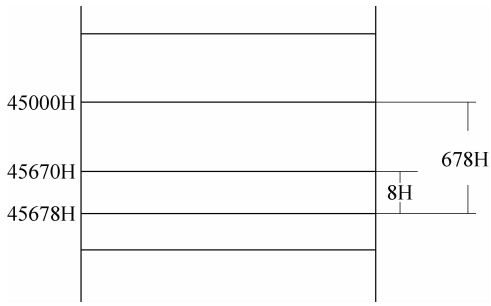


图 3-6　段地址和偏移量

当段寄存器内容为 4500H 时，内存地址 45678H 可以由 678H 这个偏移量具体而准确的表示。当段寄存器内容为 4567H 时，内存地址 45678H 可以由 8H 这个偏移量具体而准确的表示。在程序运行时，这种地址的生成，都是通过指令由 CPU 自动完成的。

对上述问题的理解，有助于我们在将来的学习中理解 8086 程序中的 64KB 限制的本质意义及其解决方法；有助于理解转移指令中的短跳转、近跳转、远跳转的实质；有助于理解过程调用机制和中断机制；有助于对现场保护、现场恢复的理解。

3. 逻辑段

程序执行时由 4 个段寄存器给出的段为物理段，而源程序中由 SEGMENT/ENDS 伪指令定义的段，或简化段定义中的.CODE、.DATA、.STACK 定义的段，为逻辑段。程序中逻辑段的组织允许有一个或多个代码段、数据段、堆栈段，特别是数据段。这些逻辑段可以定义在一个或多个不同的源程序文件中，参见第 4 章。但在汇编、连接之后变成的执行文件中，在程序执行的瞬间，统统都对应着以上所述的物理段，只不过不同时刻，所指向的段地址可能不同而已。程序中逻辑段的分配与组织方式，以及与物理段的对应关系如图 3-7 所示。

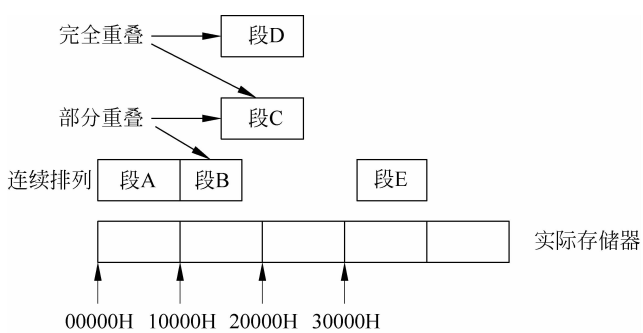


图 3-7　程序中逻辑段的分配和组织方式

3.3.2　段寄存器的引用

由于 8086 内部有 4 个段寄存器，所以程序执行时可同时访问 4 个不同含义的段。但这 4 个段的分工有所不同。

取指令时，段寄存器 CS 指向指令存放的内存段，即代码段，IP 指向下一条待执行指令在该段的偏移量，二者共同决定了待取指令的地址（如表 3-4 第 1 行所示）：

$$(CS) \times 10H + (IP)$$

涉及堆栈操作时，例如压栈指令 PUSH 和退栈指令 POP，段寄存器 SS 指向用于堆栈的内存段，即堆栈段，SP 指向该堆栈的栈顶，二者共同决定了堆栈操作的地址（如表 3-4 第 2 行所示）：

$$(SS) \times 10H + (SP)$$

由于堆栈操作的特殊性，压栈和退栈时，对 SP 的引用有特定的处理过程。压栈前需要(SP)－2 送 SP，而退栈后则需要(SP)＋2 送 SP。详见 3.4 节堆栈。压栈和退栈过程中 SP 的内容是动态变化的，这种变化的目的是为了有效地利用存储空间，不用时就释放。

另外，为了通过堆栈在不同函数间及汇编所谓的过程间传递数据，就需要提供一种机制。这种机制既要访问堆栈，又不改变 SP。显然压栈和退栈是不能够满足这种要求的，原因是压栈和退栈都会改变 SP。解决的办法是：当用 BP 寄存器构造位移量，即有效地址 EA 访问内存时，80x86 设计为相对 SS 的寻址。于是在这种情况下内存操作数的地址为：

$$(SS) \times 10H + \text{由 BP 构造的 EA}$$

即以 BP 间接寻址时，其默认指定的段寄存器是 SS，如表 3-4 中最后一行所示。这样的访问形式，既访问了堆栈的单元，又不会改变堆栈指针 SP。

如果指令中的内存操作数不是以 BP 内容构造有效地址 EA，而是用其他寻址方式构造有效地址 EA，这时内存操作数的地址总是相对 DS 段，如表 3-4 中第 3 行所示。

在进行串操作时，源串段地址由 DS 指向，位移由 SI 指出；目标串段地址由 ES 指向，位移由 DI 指出，如表 3-4 中第 4 行和第 5 行所示。汇编语言中的串操作是指成片单元的移动、比较、搜索等。

由表 3-4 可以看出，取指令总是相对 CS 段，不可能是其他段，位移也只能由 IP 指定，不可能是其他。堆栈操作也只能是相对 SS 段，位移由 SP 指定，不可能是其他。串操作的目标串也只能是相对 ES 段，位移由 DI 指定，不可能是其他。

表 3-4 中第 2 列指出在默认指定的情况下，相应操作所使用的段寄存器；第 3 列说明在显式指定的情况下，可以选择的段，详见 3.5 节寻址方式。

表 3-4　段寄存器引用规定

访问存储器涉及的方式	正常使用的段寄存器	可选用的段寄存器	偏　　移
取指令	CS	无	IP
堆栈操作	SS	无	SP
变量（以下情况除外）	DS	CS、ES、SS	有效地址（EA）
源数据串	DS	CS、ES、SS	SI
目的数据串	ES	无	DI
BP 作为指针寄存器使用	SS	CS、DS、ES	有效地址（EA）

正是由于段寄存器和位移的不同分工，使得我们可以在相关操作时设置对应的寄存器，以便让 CPU 按照我们希望的方式进行寻址。

3.4 堆 栈

3.4.1 什么是栈

栈也称做堆栈(Stack),是一种先进后出(Last In First Out,LIFO)的线性表,简称LIFO表。栈允许插入和删除的一端称做栈顶,另外一段称做栈底。我们在汇编语言中所说的堆栈就是指栈。堆栈有两个基本操作:入栈(PUSH)和出栈(POP)。入栈就是将一个新的元素放入栈顶,这一个元素只能是字,不能是字节。入栈也称做压栈。出栈则是从栈顶取出一个元素。其中,栈顶的元素总是最后入栈、最先出栈。出栈也称做退栈或弹出。

3.4.2 8086的栈机制

8086CPU中提供了栈的机制,这就使得我们可以在基于8086CPU编程的时候将一段内存当作栈来使用,并通过8086的相关指令以栈的方式访问内存。

8086提供了与栈相关的多条指令,其中最常用的有入栈(PUSH)和出栈(POP)两个指令,两种操作都是以字为单位进行的。入栈时栈顶从高地址向低地址方向增长。

例如:

PUSH AX:表示将寄存器AX的内容送入栈中。

POP AX:表示将栈顶的内容弹出并送入AX寄存器中。

我们知道,栈操作过程中,段寄存器SS指向用于堆栈的内存段,SP指向该堆栈的栈顶,把它们合在一起就可以访问栈顶单元。现在,我们就可以完整地描述栈操作的执行过程了。

PUSH AX分两步执行:

(1) SP的内容减2送SP,即(SP)=(SP)-2,使栈顶指针指向当前栈顶的前一个元素的位置。

(2) 将AX中的内容送入栈顶(SS:SP)指向的内存单元,即由(SS)×10H+(SP)指向的元素单元。

POP AX的执行过程与PUSH相反,如下所示:

(1) 将栈顶(SS:SP)指向的内存单元数据送入AX,即由(SS)×10H+(SP)指向的元素送AX。

(2) SP内容加2送SP,即(SP)=(SP)+2,使栈顶指针指向当前栈顶的后一个元素。

例3-1 堆栈操作实例。

源程序ex3-1.asm如下:

```
.MODEL SMALL
.STACK 4H
.CODE
GO:    MOV AH, 'A'
       MOV AL,'B'
       MOV BH,'C'
       MOV BL,'D'
       PUSH AX
```

```
PUSH BX
POP AX
POP BX
MOV AH,4CH
INT 21H
END GO
```

本例源程序经汇编连接后生成执行文件，在某台机器上用 EMU88086 4.05 版仿真装入该执行文件，单条指令运行，得到的堆栈段情况为：

```
(SS) = 0711H
(SP) = 0004H
```

图 3-8 用来说明程序执行过程中堆栈的变化情况。其中，图 3-8(a)是压栈前的堆栈状态，可以看出：由于段寄存器 SS 内容为 0711H，SP 内容为 04H，因此堆栈段段地址为 07110H。执行 PUSH AX 指令时，首先 SP 的内容减 2 送 SP，SP 指向地址 07112H，然后将 AX 的内容压入当前栈顶，即地址 07112H 和 07113H，于是这两个单元的内容变为 42H 和 41H，如图 3-8(b)所示。

然后执行指令 PUSH BX，首先 SP 的内容减 2 送 SP，SP 指向 07110H，然后将 AX 的内容压入当前栈顶，即地址 07110H 和 07111H，于是这两个单元的内容变为 44H 和 43H，如图 3-8(c)所示。

接着执行指令 POP AX，首先将当前栈顶 07110H 的字 4344H 弹出到 AX，然后 SP 内容加 2 送 SP，SP 指向 07112H，如图 3-8(d)所示。

再执行指令 POP BX，首先将当前栈顶 07112H 的字 4142H 弹出到 AX，然后将 SP 的内容加 2 送 SP，SP 指向 07114H，如图 3-8(e)所示。

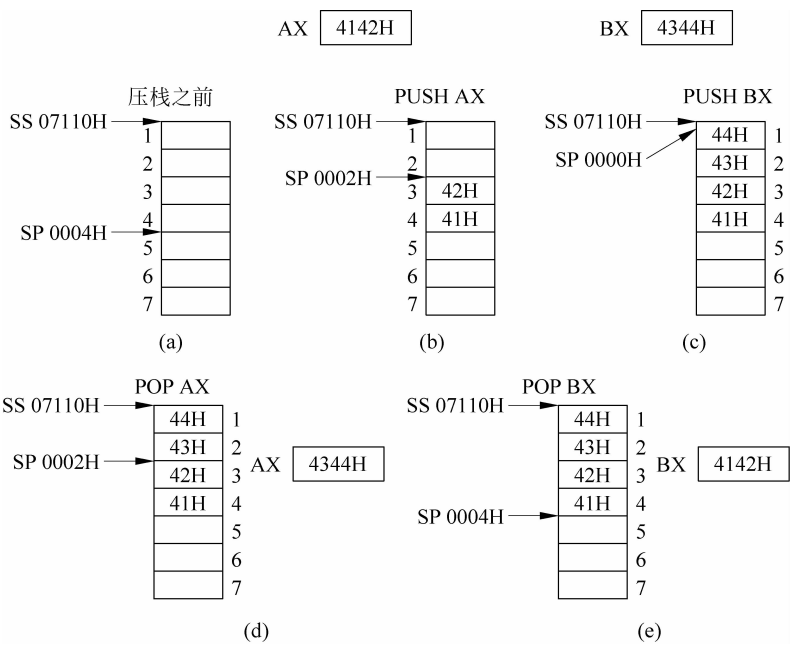


图 3-8 压栈和退栈

从本程序运行过程可以看出：

(1) 以上程序运行结果验证了堆栈的先进后出，指令序列：

```
PUSH AX
PUSH BX
POP AX
POP BX
```

运行后交换了 AX 和 BX 的内容。

(2) 本程序栈长为 4 个字节，最多容许两个元素，即两个字进栈。“PUSH AX”，AX 进栈前 SP 值为 4，进栈时首先 SP 内容减 2，然后 AX 压栈。因此对于栈长为 N 个元素的堆栈，在栈为空的情况下，SP 的内容应该是 $2N$。

(3) 本程序执行“PUSH BX”后，栈已经到底，这时的 SP 值为 0，栈顶也是栈底。因此对于任何一个堆栈，当堆栈满了的时候 SP 的值为 0，并指向堆栈段的段首。

(4) 压栈是用 PUSH 指令源操作数的一个字替换掉栈顶的一个字，源操作数不变。弹出是将栈顶的字替换掉目标操作数中的字，而堆栈内存中的内容并没有改变，改变的仅仅是堆栈指针，堆栈指针的改变就足以释放堆栈中的字空间，这就意味着这个字空间的内容虽然没有改变，但允许新的字压栈替换它。可见堆栈空间是动态管理的，一旦不用就释放。

3.5 寻 址 方 式

考虑到读者最终是要用汇编指令来书写程序而不是用代码指令来书写程序，因此本节将从汇编指令角度来阐述 8086 指令的寻址方式。寻址方式就是寻找指令中的操作数的方式，寻址主要是指寻找内存数据的地址。

3.5.1 汇编指令

1. 汇编指令的书写形式

一条汇编指令通常可以写成如下形式：

```
[标号:] 操作码 [目标操作数][, 源操作数] [; 注释]
```

说明：

(1) 汇编指令中“[ ]”中的内容可有可无。

(2) 标号必须是用字母打头的字母或数字组成的字符串。标号供转移指令作为转移的目标。一般情况下都没有标号，如果有标号，则其在程序单位中必须是唯一的。

(3) 一条能汇编成机器代码指令的汇编指令必须有唯一的操作码。操作码是汇编指令的关键字，它指出该指令做什么。

(4) 一条汇编指令中的目标操作数用来指出指令的处理结果置于何处。在许多指令中目标操作数既表示处理的对象之一来自何处，又指出处理的结果置于何处。加之它总是紧接着操作码出现在源操作数的左边，因此有的书上也称目标操作数为左源。

(5) 一条汇编指令中的源操作数用来指出指令处理的对象来自何处。

(6) 汇编指令末尾的分号表示由“;”起直至<ENTER>前均为注释部分,汇编程序在其译成机器代码指令时将无视这一部分,而仅在输出源程序清单时才被原样输出。在输入源程序时,每一条汇编指令末尾必须输入换行键<ENTER>,<ENTER>表示本指令的结束,下一指令的开始。

2. 汇编指令的分类

指令行中可以没有源操作数和目标操作数。在这种情况下通常是对某一固定的或称做隐含的操作数的操作,这种指令被称为无操作数指令。

指令行中可以没有源操作数而仅含目标操作数。这时目标操作数既指出处理对象来自何处,又指出处理结果置于何处。这种指令被称为单操作数指令。

指令行中既有源操作数又有目标操作数的指令称做双操作数指令。双操作数指令中目标操作数一定出现在源操作数的左边。

例如:

```
LOOP1: AAA              ;带标号,固定操作数 AL<ENTER>
       MOV AX,53H       ;双操作数指令<ENTER>
       INC DH           ;单操作数指令<ENTER>
```

其中,第一条汇编指令前的 LOOP1 是一标号,AAA 是操作码,这一指令隐含使用操作数 AL,“;”后直到<ENTER>前是注释部分,在 DOS 下运行汇编程序,注释部分只能用英文和 ASCII 码符号书写,在中文操作系统下则可用中文书写,<ENTER>是指换行键。

第二条指令中 MOV 是操作码,AX 是目标操作数,指出要将结果存入寄存器 AX 中,53H 是源操作数,指出指令处理的数来自何处,这里就是十六进制数 53H 本身,该指令执行结果就是将 53H 送入 AX。

第三条指令中 INC 是操作码,DH 是目标操作数。指令功能是将 DH 寄存器内容加 1 后将结果送入 DH。

由以上 3 条指令可以看出,这里所说的操作码通常是指指令功能的助记符,它给出了指令功能便于记忆的形式,例如 MOV 是 Move 的缩写等。

3.5.2 指令中的操作数

指令中的操作数有 3 种,分别是立即数操作数、寄存器操作数和内存操作数。

单操作数指令的操作数只能是寄存器操作数或内存操作数。

双操作数指令的目标操作数只能是寄存器操作数或内存操作数,而源操作数可以是三者之一,但是两操作数不能同时为内存操作数。

1. 立即数操作数

立即数操作数作为代码指令的一个字段出现在双操作数指令中。除了乘法、除法和字符串操作指令之外,立即数操作数均可作为源操作数。立即数操作数在汇编指令中可以用十六进制或八进制形式书写,例如 0F0H、777Q;也可以用二进制形式书写,例如 101B;还可以用十进制形式书写,例如 99D。但要注意,在用十六进制书写时第一个字符非数

字 0～9时，前面一定要补一个 0，例如 FAH 应记为 0FAH。另外，汇编指令中立即数操作数还可以以一个表达式的形式出现，此时该立即数就是表达式的值。

例 3-2　通过 MOV AX,3456H 和 MOV DL,41H 看立即数。

在 DEBUG 调试程序下，我们可以得到其代码，如图 3-9 所示。

```
-a
0C48:0100 mov ax,3456
0C48:0103 mov dl,41
0C48:0105
-d
0C48:0100  B8 56 34 B2 41 B4 40 CD-21 80 3E E3 99 00 74 08
0C48:0110  41 BA E3 99 B4 40 CD 21-B4 3E CD 21 34 00 37 0C
0C48:0120  3E CD 21 BA BC 98 B4 41-CD 21 C6 06 D1 96 00 E9
0C48:0130  57 FA E9 AE FE 24 0C 7A-10 50 24 08 A2 D4 99 58
0C48:0140  24 04 A2 D8 99 08 06 D2-99 A0 D8 99 0A C0 C3 80
0C48:0150  3E B7 98 FF 75 0C BF E1-94 BD B7 98 E8 11 01 E8
0C48:0160  74 E4 32 C0 86 06 16 99-0A C0 75 03 E9 8C 00 8B
0C48:0170  36 B9 98 BF 76 97 B8 00-29 CD 21 80 3C 00 74 06
-
```

图 3-9　含有立即数的指令

从图 3-9 可以看出，“MOV AX,3456H”对应的代码指令是 B85634H，这是一个 3 字节的指令，B8 是操作码，5634H 是操作数，显然立即数操作数 3456H 是作为指令的一部分出现在指令中，而且高位在高地址，低位在低地址。同理，指令“MOV DL,41H”的代码指令是 B241H，其中，41H 是立即数，它同样是作为指令的一部分出现在代码指令中。因此不难看出立即数作为代码指令的一个字段出现在指令中。因此它是在代码段，而不是在数据段或其他什么地方。

注意：立即数操作数不能作为目标操作数，道理是显然的，因为立即数是代码指令的一部分，不可能在指令执行过程中将一个常数置入代码指令的某个字段。

2. 寄存器操作数

寄存器操作数是指寄存器的内容参加运算，或用寄存器存放结果。例 3-2 中的 AX 和 DL 就是寄存器操作数。

段寄存器的内容指出当前 4 个段基址，这些寄存器不能用一般传送指令将立即数送入。如果需要将立即数置入段寄存器中的 DS、ES 或 SS，则首先应将该值送入 AX 或其他通用寄存器，然后再由 AX 传送给 DS、ES 或 SS。至于 CS，因其与指令地址有关，一般无须用户干预。

标志寄存器 FLAG 的各位在执行算术运算和逻辑运算指令后一般均被修改，至于修改成什么形式，将依指令及执行结果而定，它可以反映出当时处理器和累加器所检测到的结果。详见这一指令的说明。标志寄存器一般不能作为操作数，但可用标志指令或 INC、DEC、ADD、MUL、DIV 等来处理。

通用寄存器(AX、BX、CX、DX、SP、BP、SI、DI、AH、AL、BH、BL、CH、CL、DH、DL)和指针及变址寄存器(BX、BP、SI、DI)均可参加算术和逻辑运算操作。通用寄存器在单操作数指令中可作目标操作数，在双操作数指令中既可作源操作数也可作目标操作数。

一些汇编指令中虽然没有显式地写有寄存器，但是它却隐含着使用所指定的通用寄存器，详见表 3-5。

表 3-5 寄存器作为隐含操作数

指 令	隐 含 应 用
AAA、AAD、AAM、AAS	AL、AH
CBW、CWD	AL、AH 或 AX、DX
DAA、DAS	AL
IN、OUT	AL 或 AX
MUL、IMUL、DIV、IDIV	AL、AX 或 AX、DX
LAHF、SAHF	AH
LES	ES
LDS	DS
循环或移位指令	CL
字符串操作指令	CX、SI、DI
XLAT	AL、BX

注意：在双操作数指令中，允许目标操作数和源操作数都是寄存器，但类型必须一致，要么都是字节寄存器，要么都是字寄存器。无论何种允许的操作数，双操作数源与目标存储类型必须一致，要么都是字节，要么都是字。

3. 内存操作数

内存操作数又称存储器操作数，是指把内存某地址存放的字节、字作为指令的处理对象。这时要将该字节、字作为源操作数或目标操作数，当其作为源操作数时从内存中取出，或送到某个寄存器，或参加运算等；当其作为目标操作数时，则是将操作的结果置入该内存单元。无论是何种内存操作数，关键是必须找到其所在地址，即必须指出其所在段和相对于段首的位移(即有效地址 EA)，才能如图 3-4 和图 3-5 那样确定其物理地址。内存操作数地址的确定是寻址规则的重点。

内存操作数所在段的段寄存器的名字在汇编指令中一般是不写的，它遵循着如表 3-6 所示的隐含原则，根据内存操作数类型的不同相应使用不同的段寄存器。

注意：在双操作数指令中，两个操作数不能同时是内存操作数，这是由其体系结构和代码指令的编码规则所决定的。

3.5.3 寻址规则

寻址规则是指寻找指令中操作数的规则。根据操作数类型的不同，寻址方式也各不相同，但不外乎固定寻址、立即寻址、直接寻址和间接寻址几种方式。其中，固定寻址和立即数寻址比较简单，重点和难点是内存操作数的直接寻址和间接寻址。因此有时提到寻址规则，也往往指内存操作数的寻址规则。

1. 固定寻址(Fixed Addressing)

汇编指令中隐含对固定目标的操作称做固定寻址。例如非压缩型 BCD 码校正指令，或 ASCII 码加法校正指令 AAA，表面看只有操作码没有操作数，但其隐含着使用固定操作数

AL 和 AH 寄存器，被调整的数位于 AL 中，调整的结果在 AL 和 AH 中。8086 中这一类指令多为对应的单字节指令，详见表 3-5。

2. 立即数寻址(Immediate Addressing)

立即数可以是字节常数或字常数，它只能作为源操作数出现在指令中。

例如：

```
MOV AX,0FFH     ;字立即数
MOV AL,53H      ;字节立即数
MOV AH,53       ;十进制书写的立即数
```

3. 寄存器直接寻址(Register Addressing)

通用寄存器、指针及变址寄存器均可作源操作数和目标操作数，用于寄存器直接寻址。这时寄存器的内容就是指令运算和操作的数。

在汇编指令书写时，凡寄存器直接寻址仅需在“源操作数”或“目标操作数”位置写上该寄存器名即可。

例如：

```
MOV AX,53H      ;立即数 53H 送寄存器 AX,AX 是寄存器操作数
ADD AX,BX       ;AX←(AX) + (BX) 即 AX 内容加 BX 内容后将结果送 AX
```

在汇编指令的讲解过程中，通用寄存器操作数用 r 表示。例如：

```
MOV r,im
```

表示 MOV 指令中 im 所在位置，可以是任一与目标操作数 r 数据类型一致的立即数。

数据寄存器虽然都可以参与算术及逻辑运算，但是如果用 AX、AL 作累加器存放结果，则指令码会更短。

4. 内存储器直接寻址(Direct Addressing)

内存储器直接寻址也称内存直接寻址，是指有效地址 EA 直接可由代码指令中某一字段得到，它可以是以字节或字表示的位移(Disp)，当为字节时由该字节的符号扩展 8 位二进制数形成 16 位的字，该字即为有效地址(EA)。

在汇编指令的讲解过程中我们用 disp 表示存储器直接寻址的位移。例如：

```
MOV disp, AX
```

表示该指令中目标操作数可以是存储器直接寻址，至于实际程序中的书写形式，其中的 disp 应根据具体程序决定。

存储器直接寻址时物理地址的构成规则为：

$$(DS) * 10H + disp$$

例如：

```
MOV AL,DS:[4]
```

其中，DS:[4]就是内存直接寻址，该指令的功能是将数据段位移为 4 的那个内存单元存放的一个字节送 AL 寄存器。

例 3-3 内存直接寻址实例一。

源程序 ex3-3.asm 如下：

```
DATA SEGMENT
    DB 41H
    DB 42H
DATA ENDS
CODE SEGMENT
    ASSUME CS: CODE, DS: DATA
GO: MOV AX, DATA
    MOV DS, AX
    MOV DL,DS:[0]     ;内存直接寻址,相对 DS 段,EA 就是直接位移量 0
    MOV AH, 2
    INT 21H
    MOV DL,DS:[1]     ;内存直接寻址,相对 DS 段,EA 就是直接位移量 1
    INT 21H
    MOV AH, 4CH
    INT 21H
CODE ENDS
    END GO
```

例 3-4 内存直接寻址实例二。

源程序 ex3-4.asm 如下：

```
DATA SEGMENT
    A1 DB 41H
    B1 DB 42H
DATA ENDS
CODE SEGMENT
    ASSUME CS: CODE, DS: DATA
GO: MOV AX, DATA
    MOV DS, AX
    MOV DL,A1   ;内存直接寻址,相对 DS 段,变量 A1 相对数据段的直接位移量为 0,有效地址 EA 就是
                ;直接位移量 0
    MOV AH, 2
    INT 21H
    MOV DL,B1   ;内存直接寻址,相对 DS 段,变量 B1 相对数据段的直接位移量为 1,有效地址 EA 就是
                ;直接位移量 1
    INT 21H
    MOV AH, 4CH
    INT 21H
CODE ENDS
    END GO
```

以上两个例题的执行结果，都是在显示器输出字符 AB。

注意：直接寻址在汇编语言的书写形式上类似于直接位移量和立即数寻址，必须从整

个程序上下文分析中才能得出正确结论。例 3-4 中汇编指令“MOV AX,DATA”中的 DATA 和“MOV AH,2”中的 2 是立即数,而“MOV DL,A1”中的 A1 和“MOV DL,B1”中的 B1 则是直接位移量。这个结论可以通过例 3-4 和例 3-3 的对比得出,也可以在程序执行时利用调试手段看清楚。

关于本例中的 DATA,它是一个逻辑数据段的段名,这个段名经过汇编、连接和操作系统装入后就是一个具体的段地址常数,因此它是一个立即数。立即数是不能够直接送段寄存器的,必须首先将该段值送至一个 16 位的通用寄存器,这里是 AX,然后通过 AX 送 DS。这就是本例中装填段寄存器为什么要通过下面两条指令的原因:

```
MOV AX, DATA
MOV DS, AX
```

例 3-5　假设(DS)=3000H,而直接位移量为 2100H,AX 内容为 3047H。执行指令 MOV DS:[2100H],AX,存储器直接寻址的寻址过程如图 3-10 所示。

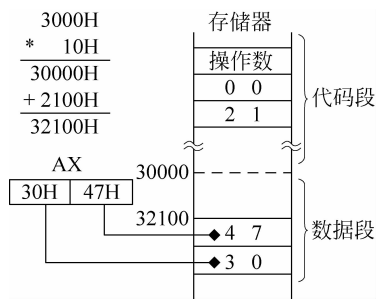


图 3-10　存储器直接寻址

存储器直接寻址方式隐含使用的段寄存器同其他存储器寻址方式一样。8086 允许段超越,这种情况下汇编指令中需在 disp 前根据需要写上表 3-4 中允许选择的段寄存器的名字,然后加一冒号,此时在形成物理地址时将使用相应的段寄存器作基址以代替隐含的段寄存器。

例如:

```
MOV AX, ES:[SI + 5]
```

这里“ES:”称做段前缀操作,“ES:[SI+5]”中如果没有“ES:”段前缀指定,则是相对 DS 的寻址,而这里用“ES:”作段前缀指定就改成了相对 ES 寻址。于是内存操作数的地址计算变为:

$$(ES) \times 10H + (SI) + 5$$

同理:

```
MOV DS:[BP + 2],AL
```

这里“DS:”称做段前缀操作,“DS:[BP+2]”中如果没有“DS:”段前缀指定,则是相对 SS 的寻址,而这里用“DS:”作段前缀指定就改成了相对 DS 寻址。于是内存操作数的地址计算变为:

$$(DS) \times 10H + (BP) + 2$$

究竟在哪些情况下允许使用段前缀,参见表 3-4 和以下间接寻址规则。

5. 寄存器间接寻址(Register Index Addressing)

寄存器的内容作为有效地址 EA 的存储器寻址方式称为寄存器间接寻址。此时寄存器可以是 SI、DI、BP、BX 之一,在书写汇编指令时相应寄存器名用[ ]括起来,表示该寄存器的内容为 EA。此地址形成规则如表 3-6 表示。

表 3-6 寄存器间接寻址的地址形成

变 址 器	汇编书写形式	有效地址(EA)	段 寄 存 器	物 理 地 址
SI	[SI]	(SI)	DS	(DS) * 10H+(SI)
DI	[DI]	(DI)	DS	(DS) * 10H+(DI)
BX	[BX]	(BX)	DS	(DS) * 10H+(BX)
BP	[BP]	(BP)	SS	(SS) * 10H+(BP)

例如：

```
MOV AX,[BP]    ;AX←((SS) * 10H + (BP))
```

表示 SS 内容乘以 10H，加上 BP 的内容构成了物理地址，取该地址的内容送 AX 寄存器。段前缀操作同“4. 内存储器直接寻址”。

6. 变址寻址(Index Addressing)

变址寻址是指使用 SI、DI、BX、BP 与直接位移量 disp 构成的存储器寻址方式。此时 EA 的构成规则是在寄存器间接寻址中 EA 的计算后再加上 disp。

变址寻址方式的操作数在汇编指令中书写时可以是下述形式之一：

```
[BX] + disp
disp[BX]
[BX + disp]
```

其中，BX 位置处也可以是 SI、DI、BP。

例如：

```
MOV AX, [BP] + VARA
MOV AX, VARA[BP]
MOV AX, [BP + VARA]
```

都是等同的，其中，VARA 是变量名，关于段前缀的使用同“5. 寄存器间接寻址”，其地址形成规则见表 3-7。

注意：寄存器间接寻址和存储器直接寻址是变址寻址的特例。在变址寻址中如果不含变址寄存器，就是存储器直接寻址；如果没有直接位移量，仅仅有变址寄存器，就是寄存器间接寻址。

表 3-7 变址寻址的地址形成

变 址 器	汇编书写形式	有 效 地 址	段 寄 存 器	物 理 地 址
SI	[SI+disp]	(SI)+disp	DS	(DS) * 10H+(SI)+disp
DI	[DI+disp]	(DI)+disp	DS	(DS) * 10H+(DI)+disp
BX	[BX+disp]	(BX)+disp	DS	(DS) * 10H+(BX)+disp
BP	[BP+disp]	(BP)+disp	SS	(SS) * 10H+(BP)+disp

7. 基址加变址的寻址方式

以 BX 和 BP 为基址寄存器，而以 SI、DI 为变址寄存器构成的存储器寻址方式称为基址

加变址的寻址方式，其有效地址的构成为：EA＝基址寄存器的内容＋变址寄存器的内容，或者是上述形式构成的值再加上 disp 形成 EA，在汇编指令中这种操作数可以是下述形式之一：

```
[BX + SI + disp],表示 EA = (BX) + (SI) + disp
[BX + DI + disp],表示 EA = (BX) + (DI) + disp
[BP + SI + disp],表示 EA = (BP) + (SI) + disp
[BP + DI + disp],表示 EA = (BP) + (DI) + disp
```

其中，disp 可以是 0，这种情况下将省略 disp。在汇编源程序中以上书写形式与下面的书写形式是等效的：

```
[BX][SI] + disp
disp[BX][SI]
[BX + SI] + disp
[BX + SI + disp]
```

其地址形成规则详见表 3-8。表 3-8 的组合关系可以用图 3-11 表示。

表 3-8　基址加变址的地址形成

基址寄存器	变址寄存器	汇编书写形式	有 效 地 址	段寄存器	物 理 地 址
BX	SI	[BX＋SI＋disp]	(BX)＋(SI)＋disp	DS	(DS)＊10H＋EA
BX	DI	[BX＋DI＋disp]	(BX)＋(DI)＋disp	DS	(DS)＊10H＋EA
BP	SI	[BP＋SI＋disp]	(BP)＋(SI)＋disp	SS	(SS)＊10H＋EA
BP	DI	[BP＋DI＋disp]	(BP)＋(DI)＋disp	SS	(SS)＊10H＋EA

关于使用段前缀指令改变段基址的方式同“5. 寄存器间接寻址”。

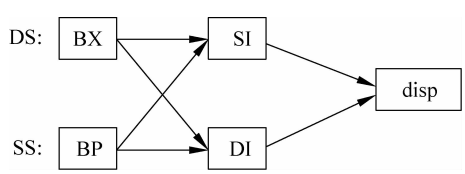


图 3-11　基址加变址的地址形成规则

注意：在存储器各种间接寻址方式中只要含 BP，通常是对堆栈段的寻址。当基址加变址的寻址中，如果仅仅含一个寄存器，就是前面所述的变址寻址。以上寻址规则中除固定寻址、立即数寻址外，其他情况统统可以看作基址加变址的特例。因此只要掌握了图 3-11 的基址加变址的地址组合关系和段前缀的指定规则，就完全掌握了 8086 的内存操作数，或称做存储器操作数的寻址规则。

8. I/O 端口寻址(I/O PORT Addressing)

对分配在 I/O 空间中的端口有两种寻址方式：一种是直接端口寻址方式，端口号可以是 8 位二进制数(0～255)；另一种是间接端口寻址方式，当端口号大于 255 且小于等于 65535 时必须使用间接端口寻址，间接端口寻址时端口号应事先存放在 DX 中，间接寻址的端口地址可以是 0～65535 的口地址。

直接端口寻址举例如下：

```
IN   AL, 63H   ;从 63H 口读信息到 AL 寄存器
OUT 65, AL     ;将 AL 内容送 65 端口，口地址可以用汇编允许的任何进制数字
```

间接端口寻址举例如下：

```
MOV DX, 63H
IN AL, DX       ;以上两条指令完成从 63H 口读信息到 AL 寄存器
```

又如：

```
MOV DX, 256     ;端口号 256 -> DX
OUT DX, AX      ;AX 内容送 256 号端口
```

间接端口寻址可以通过对 DX 的修改实现对一片端口的寻址。

注意：在进行端口寻址时，当端口为 8 位口时只能通过 AL 寄存器与这些口进行信息交换，当端口为 16 位口时只能通过 AX 寄存器与这些口进行信息交换。这就是说，80x86 微处理器与 I/O 设备的通信只能通过累加器 AX 或 AL 进行，而不能是其他任何寄存器或内存。

端口间接寻址的寄存器只能是地址寄存器 DX，而不能是其他任何寄存器，而且在这种情况下的间接寻址，不像前面存储器间接寻址一样，汇编指令的书写形式不允许在 DX 外写方括号。

习 题 3

1. 8086 的寄存器分为几组？每组都是哪些寄存器？每一个寄存器都有哪些特殊用途？

2. 已知(BH)＝35H，(BL)＝23H，能不能说(BX)＝3523H？SI 寄存器可分为 SH 和 SL 吗？

3. 解释以下术语：操作码、操作数、立即操作数、寄存器操作数、内存操作数、段地址、物理地址、有效地址。

4. 写出以下汇编指令中内存操作数所在的物理地址表达式。

```
MOV AL,[BX + 5]
MOV [BP + 3 ],AX
INC BYTE PTR[SI + 3]
MOV BX,[BX + SI + 2]
MOV DL,ES:[BX + DI]
```

5. 用两种方法写出从 61H 端口读入信息的汇编指令。

6. 判断以下指令书写形式的正确性：

```
MOV AL,BX
MOV AL,CL
INC [BX]
MOV 5,AL
MOV [BX],[SI]
MOV BL, F5H
```

7. 假设 DS＝3000H，SS＝4000H，SI＝200H，BX＝500H，BP＝700H，计算以下各种内存操作数的地址：

```
DS: [1]
DS: [BP + 4]
[BP]
[SI + 5]
[BX + SI + 3]
[BP][SI] + 7
[BX + SI] + 6
```

8. 深入理解栈的概念。用 EMU8086 等仿真软件单步执行以下程序，观察栈空间值的变化以及 SS、SP 等寄存器的值的变化。

```
.MODEL SMALL
.STACK 4H
.CODE
GO: MOV AX, 1234H
    PUSH AX
    POP AX
    MOV AH, 4CH
    INT 21H
    END GO
```

9. 正在执行的程序在哪个物理段？当前要执行的指令由哪两个寄存器指出？其物理地址如何计算？

10. 以例 3-1 为例，说明汇编过程中汇编程序如何知道源程序已经结束，汇编、连接后形成的执行程序文件从何处开始执行，如何实现自动执行下一条指令，又如何知道程序结束返回操作系统。

11. 指令中的立即数存放在指令中还是指令之外？举例说明。

12. 指令中的内存操作数，其有效地址的构成由哪三部分组成？分别指什么？举例说明。

13. 能够用作指令中的内存操作数的基址寄存器有哪几个？能够用作变址寄存器的有哪几个？