第 1 章　Intel 8086/8088 的基本结构

汇编语言是计算机软件设计的重要工具，在系统软件开发、实时控制和实时处理领域中有着不可替代的地位。用汇编语言编程可以充分发挥计算机硬件的功能，进行高质量的程序设计，开发出的软件具有内存开销小，运算速度快的特点。因此，在已经有众多高级语言和可视化集成开发环境工具的今天，汇编语言仍是一门不可缺少的有效的程序设计语言。

由于汇编语言密切依赖计算机硬件系统，所以要掌握汇编语言程序设计，必须首先熟悉计算机的内部结构以及与机器结构紧密相关的指令系统。本章将先介绍 Intel 8086/8088 的基本结构，为下章学习指令系统打下基础。Intel 8086 是美国电子器件公司（Intel 公司）1978 年最早推出的 16 位微处理器，其内部结构和数据总线都是 16 位。20 世纪 70 年代末，由于 8 位机系统的外部设备和各种 8 位接口部件已被普遍采用，Intel 公司又推出了 16 位内部结构，8 位外部数据总线的准 16 位微处理器 8088。8086 和 8088 指令系统相同，支持 16 位算术运算逻辑，具有 20 位的地址总线，寻址空间为 1MB。

1.1　8086/8088 的功能结构

中央处理部件 CPU 的任务是执行存放在存储器里的指令序列。CPU 一般由运算器和控制器两部分组成，在微机中也常称为微处理器。8086/8088 CPU 按功能可分为两个独立的部分：总线接口单元 BIU（Bus Interface Unit）和执行单元 EU（Execution Unit），其功能结构如图 1.1 所示。

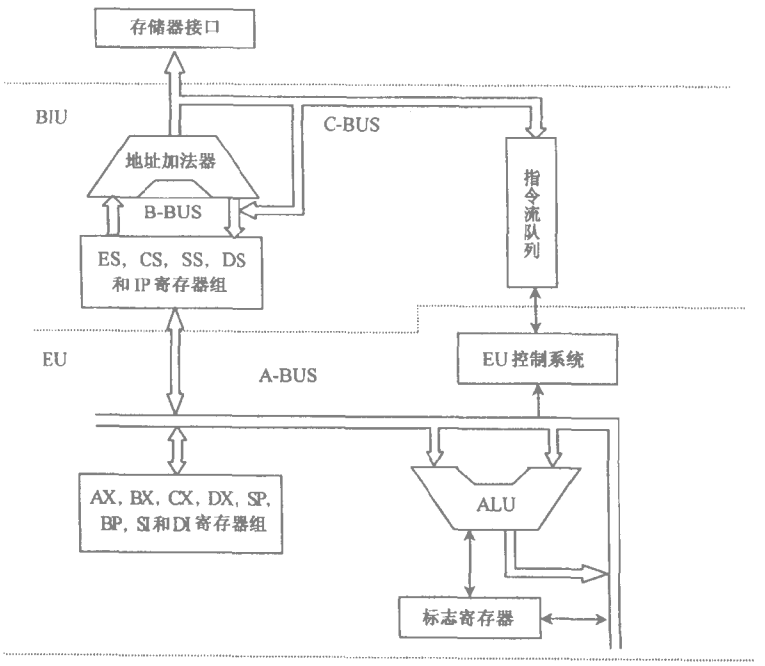


图 1.1　8086/8088 的功能结构

BIU 完成 8086/8088 与存储器之间的信息传送、总线控制和 I/O 数据传送，还包括逻辑地址与物理地址的转换；从存储器中取指令送指令流队列排队，取出执行指令时所需的操作数，并传送给 EU 完成运算和操作。

EU 负责对来自指令流队列中的指令译码并执行，实施算术逻辑运算操作。

由于 BIU 和 EU 是两个相对独立的部件，因此取指令和执行指令可以并行完成，形成指令流水线结构，如图 1.2 所示。指令流水线结构大大减少了 CPU 等待取指令的时间，提高了 CPU 的利用率和系统运行速度。

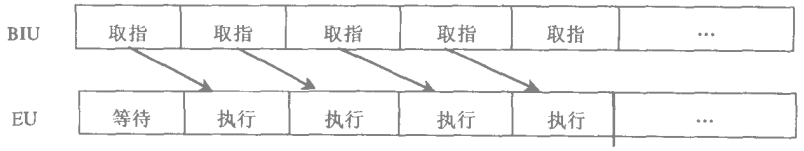


图 1.2 8086/8088 指令流水线结构

1.2 8086/8088 的寄存器结构

8086/8088 共有 14 个 16 位寄存器，一般可分为 4 类：数据寄存器、段寄存器、控制寄存器、指针及变址寄存器，如图 1.3 所示。这些寄存器具有重要的作用，专门用于存放指令执行时需要的各种信息，如操作数、操作数地址以及中间计算结果等。

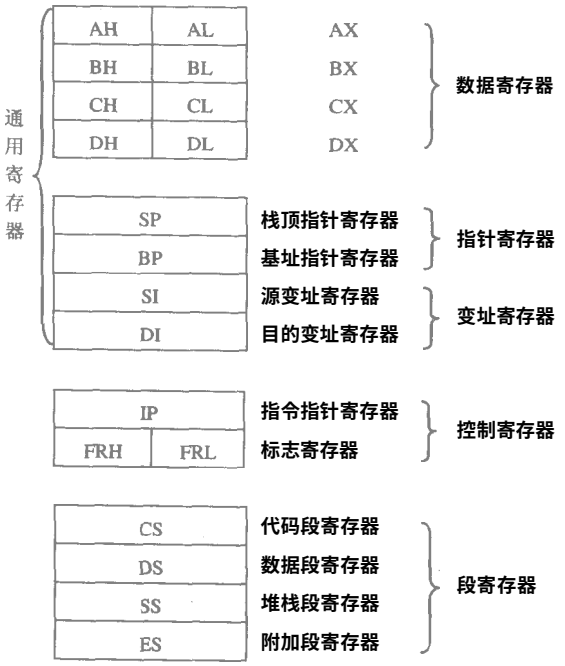


图 1.3 8086/8088 寄存器结构

1. 数据寄存器

数据寄存器共有 4 个 16 位寄存器，即 AX、BX、CX 和 DX，通常用于暂存计算过程中的操作数、计算结果或其他信息。数据寄存器既可按字类型（16 位）访问，又可按字节类型（8 位）访问，不仅有上述一般用途，同时还有各自的专门用途。

AX（Accumulator）：累加器，是算术运算的主要寄存器，此外还作为乘、除法运算及 I/O 操作的专用寄存器。

BX（Base）：基址寄存器，常用于存放存储区的起始地址。

CX（Count）：计数寄存器，常用于循环操作和字串处理的计数控制。

DX（Data）：常与 AX 共同用于双字长运算，DX 存放高位字，AX 存放低位字。此外还用于在 I/O 操作时提供外部设备接口的端口地址。

2. 段寄存器

段寄存器共有 4 个 16 位寄存器，即 CS、DS、SS 和 ES，专门用于存放段地址。

CS（Code Segment）：代码段地址寄存器，存放代码段的起始地址。代码段用于存放当前正在运行的程序。

DS（Data Segment）：数据段地址寄存器，存放数据段的起始地址。数据段用于存放当前正运行的程序所用的数据。

SS（Stack Segment）：堆栈段地址寄存器，存放堆栈段的起始地址。堆栈是一种以后进先出方式访问的数据结构，堆栈段是定义堆栈的存储区。

ES（Extra Segment）：附加段地址寄存器，存放附加段的起始地址。附加段是附加的数据段，作为辅助数据区存放当前正运行程序所用的数据。

3. 指针及变址寄存器

指针及变址寄存器共有 4 个 16 位寄存器，即 SP、BP、SI 和 DI，主要用于在访问存储器单元时提供 16 位偏移地址，其中 SI、DI 也可以作为数据寄存器使用。（一个存储器单元的物理地址（实际地址）由段地址和偏移地址两部分构成，这一问题将在下节讨论。）

SP（Stack Pointer）：栈顶指针寄存器，提供堆栈栈顶单元的偏移地址，与 SS 段寄存器联用，以控制数据进栈和出栈。

BP（Base Pointer）：基址指针寄存器，常用于提供堆栈内某个单元的偏移地址，与 SS 段寄存器联用，可访问堆栈中的任一个存储单元。

SI（Source Index）：源变址寄存器，与 DS 段寄存器联用，可以访问数据段中的任一个存储单元。

DI（Destination Index）：目的变址寄存器，与 ES 段寄存器联用，可访问附加段中的任一个存储单元。

SI、DI 也常用于在字串操作中提供偏移地址，并具有地址自动增量或减量的功能。

4. 控制寄存器

控制寄存器共有 2 个 16 位寄存器，即 IP 和 FR。

IP（Instruction Pointer）：指令指针寄存器，存放代码段中指令的偏移地址。在程序执行过程中，始终自动给出下一条要取的指令的偏移地址。IP 与 CS 段寄存器联用，可以确定下一条要取的指令的物理地址，因此 IP 是很重要的控制寄存器，用于控制程序的执行流程。

FR（Flags Register）标志寄存器，用于存放反映处理器和运行程序执行结果状态的控制标志和条件码标志。FR 中共有 9 个标志位，其中 6 个是条件码标志位，3 个是控制标志位，如图 1.4 所示。FR 中的内容也可称为程序状态字（PSW, Program Status Word）。

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

图 1.4 标志寄存器

FR 中的 6个条件码标志位用于记录程序运行结果的状态，一般由系统根据计算结果自动设置。这些标志常用做改变程序执行流程的控制条件，其含义如下。

OF（Overflow Flag）：溢出标志。在运算过程中，如果操作数超过了机器字长所限的数据表示范围，则称为发生溢出，此时 OF 位置 1，否则置为 0。

SF（Sign Flag）：符号标志。运算结果为负时，SF 位置 1，否则置 0。

ZF（Zero Flag）：零标志。运算结果为零时，ZF 位置 1，否则置 0.

CF（Carry Flag）：进位／借位标志。反映运算过程中数据最高有效位是否发生了进位或借位，若发生进位或借位，CF 位置 1，否则置 0。

AF（Auxiliary Carry Flag）：辅助进位／借位标志。运算过程中半字节有进位或借位时，AF 位置 1，否则置 0。

PF（Parity Flag）：奇偶标志。运算结果数的低 8 位中“ 1 ”的个数为偶数时，PF 位置 1，否则置 0。

FR 的 3 个控制标志位可在程序中使用指令将其置 1 或置 0,以控制 CPU 的操作方式，其含义如下。

DF（Direction Flag）：方向标志。用于在串操作中控制地址的变化方向。当 DF=1 时，每次操作后 SI、DI 中的地址自减，即串操作从高地址处理到低地址；当 DF=0 时，每次操作后 SI、DI 中的地址自增，串操作从低地址处理到高地址。

IF（Interrupt Flag）：中断标志。当 IF=1 时，允许中断；当 IF=0 时，关闭中断。有关中断系统的原理将在第 5 章介绍。

TF（Trap Flag）：单步标志（也称陷阱标志）。该标志控制 CPU 是否工作在单步操作方式下。TF=1 时，每执行一条指令后，CPU 产生陷阱异常，暂停执行正运行的程序。这种方式为程序员调试程序提供了方便，程序员可在每条指令执行后，查看 CPU 状态及指令执行结果。TF=0 时，CPU 正常工作，不产生陷阱异常。

FR 寄存器的标志位值还有一种符号表示方式。在 DOS 系统提供的汇编语言通信调试工具 DEBUG 中，可以显示 FR 的标志位。为了提高标志位值的可读性，DEBUG 中标志位的值采用了符号表示（见表 1-1），附录二给出了有关 DEBUG 程序使用方法的介绍。

表 1-1 FR 中标志位的符号表示

标志位	标志位名称	符号表示	
		=1	=0
OF	溢出标志（是/否）	OV	NV
DF	方向标志（减/增）	DN	UP
IF	中断标志（开/关）	EI	DI
SF	符号标志（负/正）	NG	PL
ZF	零标志（是/否）	ZR	NZ
AF	辅助进位标志（是/否）	AC	NA
PF	奇偶标志（偶/奇）	PE	PO
CF	进位标志（是/否）	CY	NC

1.3 堆栈与存储器结构

存储器是计算机存储信息的主要部件，程序代码、数据以及程序的运算结果都保存在存

储器中。堆栈则是存储器中的一块专用存储区，在操作系统和应用程序中都具有特殊的重要作用。本节介绍堆栈和存储器的结构及数据存取方式。

1.3.1 堆栈

堆栈是一种数据结构，实际上是在存储器中开辟的一端活动、另一端固定的数据存储区，数据的存取原则是先进后出。

堆栈的固定端称为栈底，活动端称为找顶，数据的存取都在栈顶进行，堆栈栈顶指针寄存器 SP 始终存放栈顶单元的地址，即 SP 始终指向栈顶，跟踪栈顶的变化，如图 1.5 所示。

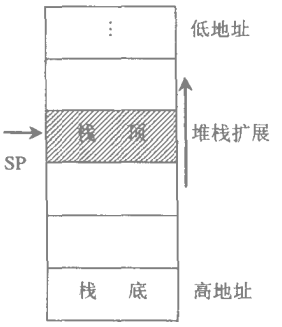


图 1.5 堆栈的结构

堆栈有着极其重要的作用，常用于保存计算和操作过程中的现场数据，保护子程序和中断服务程序返回地址等有关信息。

8086/8088 指令系统有两条指令（PUSH、POP）专门完成数据进栈和出栈操作，具体操作过程将在下一节指令系统中介绍。

注意，堆栈存取必须以字（2 字节）为单位，每次操作（数据进栈或出栈）后，SP 总是指向新的栈顶。随着进栈数据增多，堆栈将不断扩展，栈顶地址（SP）不断减小，因此堆栈是从高地址向低地址方向扩展的。

1.3.2 存储器结构

在存储器中，信息是以字节（8 个二进制位）为最小单位存储的，每一个字节单元分配一个惟一的存储器地址。地址从 0 开始编址，顺序增 1，用无符号二进制数表示，但为了方便书写，常用无符号十六进制数（数后加基数说明符 H 表示十六进制数）表示。例如，64KB 存储器空间的单元编址方法如图 1.6 所示。

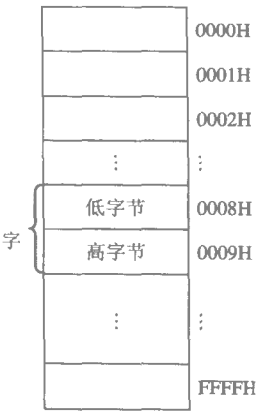


图 1.6 64KB 存储器单元编址

一个字数据在存储器中存于相邻的两个字节单元，数据的低字节存入低地址单元，高字节存入高地址单元，所以按地址递增方式存储一个字数据时，应先存低字节，后存高字节。

要访问一个存储器单元，必须先得到这个存储器单元的物理地址。物理地址要由段地址和偏移地址合成。为什么要段地址和偏移地址呢？这是由于 8086/8088 用于寻址的寄存器都是 16 位，而 16 位寄存器只能寻址 64KB 的地址范围，但 8086/8088 有 20 条地址线，寻址可达 1MB 地址范围，即可在 00000H～FFFFFH 范围内寻址。为了能在 16 位字长的机器中提供 20 位地址，8086/8088 采用了存储器分段管理的方法。

分段的方法是将存储器划分成段，每段最大不超过 64KB，这样段内地址可用 16 位表示，称为偏移地址。段的起始地址称为段基址（常简称为段地址），必须为任一小段的起始地址。小段的划分方法是：从存储器的 0 地址单元开始，每 16 个字节单元为一小段。表 1-2 列出了 1MB 存储器的编址。

表 1-2　1MB 存储器的编址

00000H	00001H	00002H	……	0000EH	0000FH
00010H	00011H	00012H	……	0001EH	0001FH
00020H	00021H	00022H	……	0002EH	0002FH
⋮	⋮	⋮	⋮	⋮	⋮
FFFE0H	FFFE1H	FFFE2H	……	FFFEEH	FFFEFH
FFFF0H	FFFF1H	FFFF2H	……	FFFFEH	FFFFFH

分析表 1-2 中的地址，可以看出表中第 1 列小段起始地址的特点，即小段起始地址最后一位都为 0，在 1M 字节的寻址范围内用做段地址时，最后一位 0 可以不用保存，并且在 1M 字节的寻址范围内正好共有 64KB 个小段起始地址，从而段地址也可以用 16 位来表示。这样用一个 16 位段地址和一个 16 位段内偏移地址表示一个存储器单元，就可以使得每一个存储器单元都有一个惟一的 20 位地址，称为存储器单元的物理地址。因此，根据上述分析， 20 位物理地址可按图 1.7 所示的方法形成。

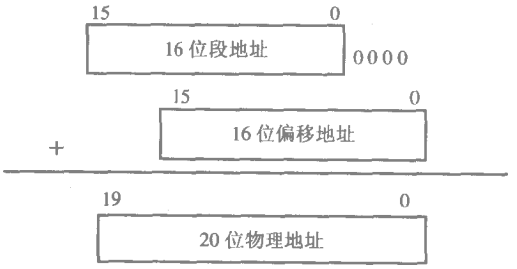


图 1.7　物理地址的形成

20 位物理地址=16 位段地址左移 4 位（即段地址乘以 16）＋ 16 位偏移地址

例 1-1　已知一个存储器单元的段地址为 3200H，偏移地址为 1210H，则该存储器单元的物理地址为：

3200H×16+1210H=33210H

每当 CPU 需要访问一个存储器单元时，就需要产生一个 20 位物理地址，这时会有一个段寄存器（CS、DS、SS 或 ES）被自动选中用于提供段地址，而偏移地址则由指令直接提供或由 16位可提供地址的寄存器给出。

形成各段内存储单元物理地址时所用的寄存器如图 1.8 所示。

在默认情况下，用于提供数据段或附加段偏移地址的 16 位寄存器可以是基址寄存器或变址寄存器。如果在指令中指定了由段寄存器 DS 或 ES 提供段地址，也可以用 BP 寄存器提供偏移地址。

在存储器中，代码段、数据段、堆栈段和附加段可以各段独立占用 64KB 存储区。分配各段存储空间时，各段之间可以间隔开或邻接，也可以部分重叠或完全重叠共享一段存储区。此外各段长度也可以不同，总之可以根据需要对各段进行不同形式的存储空间分配。各段存储区分配的两种典型方式如图 1.9 和图 1.10 所示。

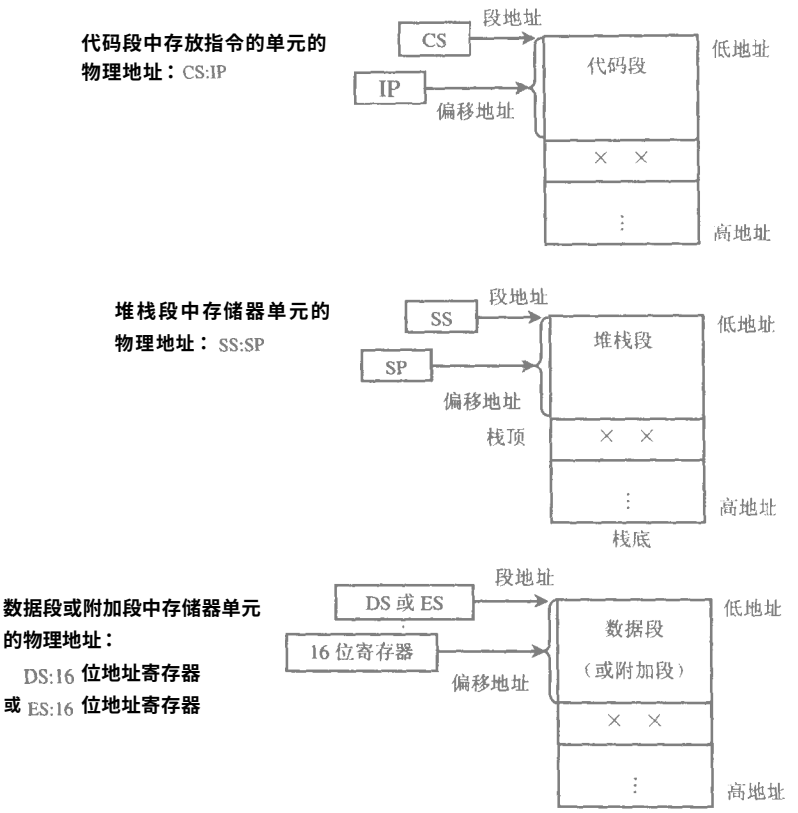


图 1.8　形成各段内存储器单元物理地址所用的寄存器

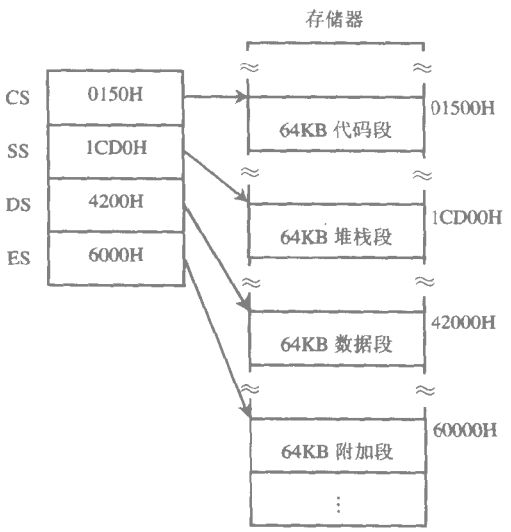


图 1.9　段分配方式之一（各段大小相等）

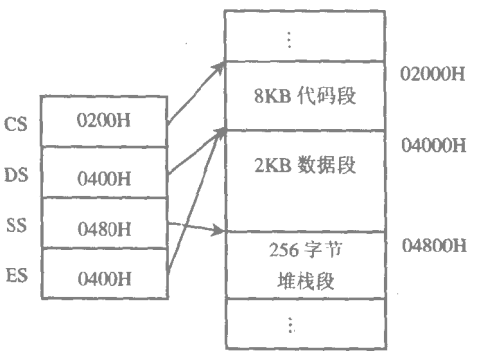


图 1.10　段分配形式之二（各段大小不等）

存储器分段管理法便于将代码区、堆栈区和数据区分配在不同存储区域，这时只需要将各个段寄存器置为相应的段地址就能保证正确地访问各个段。分段法也适应程序装入重定位的要求，只要正确设置 CS 中的段地址，就具备了程序装入不同存储区正常运行的基础。

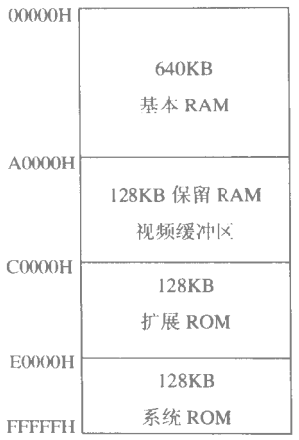


图 1.11 存储器的 1MB 地址空间分配

存储器 1MB 地址空间的分配映像基本上如图 1.11 所示，整个 1MB 地址空间从低地址端到高低址端可分为以下 4 个区段。

1. 基本 RAM 区（00000H~9FFFFH）

该区占 640KB，由 DOS 系统管理。操作系统装入后将占用低地址端一小部分空间（存放中断向量表等），其余空间由用户使用。

2. 保留 RAM 区（A0000H~BFFFFH）

该区占 128KB，用做视频缓冲区，是显示卡 RAM 的地址映射空间，用于存放屏幕显示信息，这部分地址空间实际上未全用。

3. 扩展 ROM 区（C0000H~DFFFFH）

该区占 128KB，由 I/O 接口卡的 ROM 提供支持，用于存放系统不直接支持的外设驱动程序。

4. 系统 ROM 区（E0000H~FFFFFH）

该区占 128KB，由系统使用，提供基本输入／输出系统（Basic Input/Output System）ROM-BIOS 程序，用于驱动输入／输出设备、系统加电自检、从磁盘引导操作系统等；此外，还提供了 CMOS 设置程序以及字符／图形符号点阵等信息。

本章介绍了 8086/8088 的功能结构、寄存器结构和用途、堆栈操作的特点以及存储器地址分配和分段管理方法，这些内容是学习下一章要介绍的寻址方式和指令系统的基础。

习题一

1.1 为什么要对存储器进行分段管理？存储器分段管理有什么优点？

1.2 寄存器可以分为哪几组，各组的主要用途是什么？

1.3 堆栈的存取方式有什么特点？

1.4 用连线连接左列概念与右列的解释：

（1）存储器	A. 控制 CPU 操作方式的标志位，共有 3 位：DF、IF、TF。
（2）堆栈	B. 表示存储器字节单元在所在段内位置的偏移量。
（3）段寄存器	C. 记录指令执行结果的标志位，共有 6位：OF、SF、ZF、AF、PF、CF。
（4）标志寄存器	D. 计算机存储程序和数据等信息的记忆部件。
（5）SP	E. 惟一地代表存储器中每一个字节单元的地址。
（6）IP	F. 以先进后出方式存取数据的一段存储区。
（7）物理地址	G. 保存各段起始地址的寄存器，共有 4 个：CS、DS、ES和 SS。
（8）偏移地址	H. 存放下一条要取指令的单元偏移地址的寄存器。
（9）条件码标志位	I. 保存堆栈当前栈顶地址的寄存器。
（10）控制标志位	J. 保存条件码标志和控制标志的 16位寄存器。

1.5 在偏移地址为 00A0H 和 00A4H 的两个单元中，分别存放了两个字数据 1FA0H 和 4B3FH，画图表示出这两个字数据在存储器单元中的存放情况。

1.6 某存储器单元的段地址和偏移地址为 4514H:017BH，其物理地址是多少？

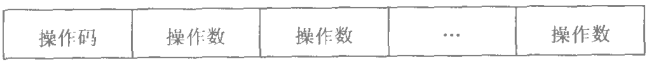
1.7 已知一条指令执行前，(CS) = 03FAH，(IP) = 45A0H，该指令的第 1 字节的物理地址是多少？

1.8 如果堆栈栈顶指针寄存器(SP) = 01FAH，进 栈 4 个数据后(SP) = ？，然后再出栈 3 个数据，(SP) = ？

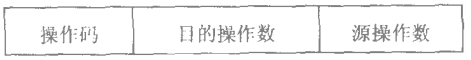
第 2 章　8086/8088 的寻址方式和指令系统

各种计算机提供给用户使用的指令集称为指令系统。计算机通过执行指令系统所支持的指令序列，可以完成各种复杂的运算和操作。本章主要介绍 8086/8088 指令系统以及 CPU 执行指令时获取操作数所使用的寻址方式，这些知识是掌握汇编语言程序设计方法的基础。

指令系统中的指令格式通常如图 2.1 所示。



（a）基本格式



b）双操作数指令格式

图 2.1　指令格式

图中，操作码指定了完成什么运算和操作，操作数则指定了运算对象。操作数字段可以有多个，但现代计算机大多数采用双操作数指令，两个操作数分别称为源操作数和目的操作数，源操作数一般不受指令执行结果的影响，而目的操作数将被运算结果所替代。双操作数指令格式如图 2.1（b）所示。

操作数字段可以是操作数本身，也可以是操作数的地址或地址的一部分，还可以是有关操作数地址的信息。因而要取得一个操作数，首先要确定获得该操作数地址的方法，即确定寻址方式。合理地选择寻址方式是一个重要问题，因为即使执行的指令相同，如果使用的寻址方式不同，指令执行时间的差别也是很大的，会直接影响程序运行的速度。一般来说，复杂的寻址方式所需的指令执行时间更长。

2.1　寻址方式

寻址方式可分为两大类，第 1 类是与数据有关的寻址方式，第 2 类是与转移地址有关的寻址方式。本节先介绍第 1 类寻址方式，第 2 类寻址方式将在第 4 章介绍。与数据有关的寻址方式有 7 种，下面通过对源操作数使用不同的寻址方式来介绍 7 种寻址方式的特点和用途。

为简便起见，本节以常用的数据传送指令 MOV 为例介绍寻址方式，该指令的格式为：

MOV　目的操作数，源操作数

指令的功能是完成源操作数到目的操作数的数据传送。

1. 立即寻址方式（Immediate Addressing）

在立即寻址方式下，操作数直接包含于指令中，即由操作码和 8 位或 16 位操作数形成一条指令，指令存放在代码段中，这种操作数称为立即数。立即数寻址方式只能用于源操作数字段。

例 2-1　MOV　AL，5

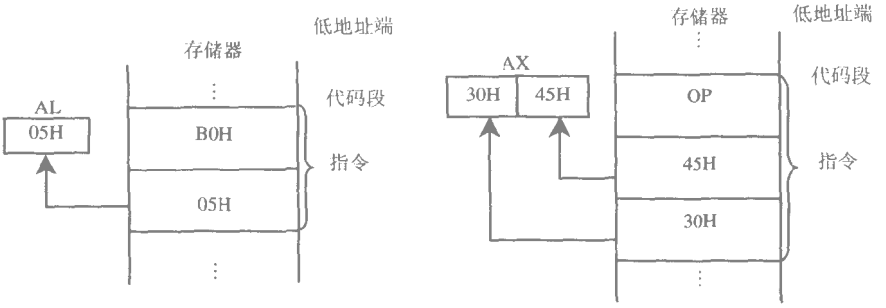
在本例中，无论执行指令前 AL 中是何值，执行指令后，立即数 5 就赋给了 AL 寄存器，

使（AL）=05H，这种寻址方式如图 2.2（a）所示。

例 2-2　MOV　AX，3045H

指令执行后，（AX）=3045H，操作如图 2.2（b）所示，图中 OP 代表操作码。

立即寻址方式的用途：立即数主要用于表示常数，立即寻址方式常用于给寄存器或存储器单元赋值。



（a）8 位立即寻址方式　　（b）16 位立即寻址方式

图 2.2　立即寻址方式

2. 寄存器寻址方式（Register Addressing）

在这种寻址方式下，操作数存放在 CPU 的寄存器中，由指令指出存放该操作数的寄存器。操作数可以是 8 位或 16 位的，相应地存放在 8 位或 16 位寄存器中。

例 2-3　MOV　AX，BX

指令执行前：（AX）=3045H，（BX）=4000H

指令执行后：（AX）=4000H，（BX）=4000H

这种寻址方式如图 2.3 所示。

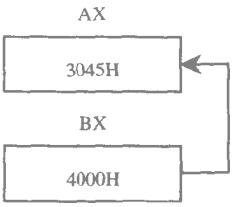


图 2.3　寄存器寻址方式

寄存器寻址方式的用途：用寄存器提供操作数时存取速度快，不需要访问存储器，可提高运算速度。

3. 直接寻址方式（Direct Addressing）

在这种寻址方式下，操作数的偏移地址包含于指令中，且位于操作码之后，而操作数本身一般存放在数据段中，也可存放在其他段（如附加段）中。如果操作数存在其他段，则必须在指令中指出操作数所在段使用的段寄存器。

例 2-4　设（DS）=3000H，指令为：

MOV　AX，[2000H]

因此，要访问的存储单元物理地址为：

30000H+2000H=32000H

如果该存储单元的值为 3050H，即有

（32000H）=3050H

指令执行结果是将 32000H 单元中的数 3050H 送 AX 中：（AX）=3050H，操作如图 2.4 所示。

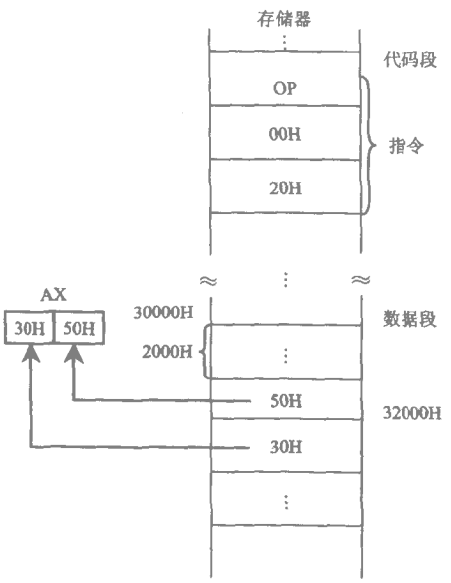


图 2.4　直接寻址方式

如果操作数存放在附加段中，指令应写为：MOV AX，ES:[2000H]

其中“ES:”即“段寄存器名:”,称为段跨越前缀。当未指定段跨越前缀时,系统默认操作数存放于数据段。

在上例中，也可用符号地址表示 2000H 单元,如果 2000H 单元的符号地址已定义为 NUM（有关数据单元的符号地址定义将在第 3 章介绍),则指令可写为：MOV AX,NUM 或 MOV AX，[NUM]，两条指令执行结果是相同的。

直接寻址方式主要用于存取某个存储器单元中的操作数，即从一个存储器单元中取操作数，或将一个操作数存入某个存储器单元。

4. 寄存器间接寻址方式（Register Indirect Addressing）

在这种寻址方式下，操作数据的偏移地址存于基址寄存器 BX、BP 或变址寄存器 SI、DI 中，而操作数则存于存储器中。

当指令指定的寄存器为 BX、SI 或 DI 时，操作数应存放于数据段中，由 DS 段寄存器提供段地址，按下式计算操作数的物理地址：

$$\text{操作数物理地址}=16\times(DS)+\begin{cases}(BX)\\(SI)\\(DI)\end{cases}$$

如果指令中指定的寄存器为 BP，则操作数应存放于堆栈段中，由 SS 段寄存器提供段地址，按下式计算操作数的物理地址：

操作数物理地址=16×(SS) +(BP)

如果指令中指定了段跨越前缀，则物理地址根据指定的段寄存器来计算，从而可取得其他段的操作数。

例 2-5 设（DS）=2000H，（BX）=1000H ，指令为：

MOV AX，[BX]

因此要访问的存储器单元的物理地址为：

16×(DS) + (BX) = 20000H+1000H = 21000H

如果（21000H）=50A0H，指令执行结果是从 21000H 单元取操作数送 AX: (AX) =50A0H。这种寻址方式如图 2.5 所示。由图可见，寄存器 BX 给出的是要访问的存储单元的偏移地址。

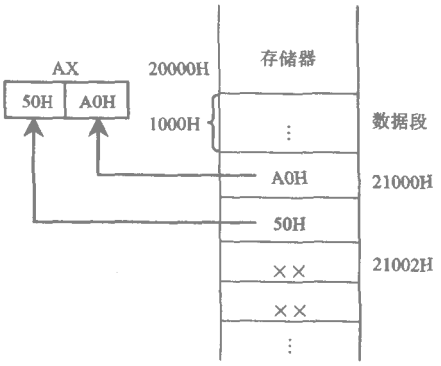


图 2.5 寄存器间接寻址方式

如果操作数存于附加段，指令应写为：MOV AX，ES:[BX]，该操作数的物理地址为 16×(ES) +(BX)。

寄存器间接寻址方式可用于处理数组。例如将一组数连续存放于存储器单元中，并且在 BX 中存放该数组的起始地址，执行一次指令后，增大或减小 BX 中的地址，就可对数组中的相邻项数据进行操作。在上例中，如果将（BX）加 2 后再执行一次指令，就可以从 21002H 地址开始的两个字节单元中取得新的字操作数。

5. 变址寻址方式（Indexed Addressing）

在这种寻址方式下，操作数的偏移地址由基址寄存器（或变址寄存器）中的值与包含于指令中的 8 位或 16 位位移量之和形成。这种寻址方式也称为相对寄存器寻址方式。

如果指令中指定的寄存器是 BX、SI 或 DI,段寄存器使用 DS，操作数的物理地址按下式计算：

$$\text{操作数物理地址}=16\times(\text{DS})+\begin{cases}(\text{BX})\\(\text{SI})\\(\text{DI})\end{cases}+\begin{cases}8\text{位位移量}\\16\text{位位移量}\end{cases}$$

如果指令中指定的寄存器是 BP，段寄存器使用 SS，操作数的物理地址按下式计算：

$$\text{操作数的物理地址}=16\times(\text{SS})+(\text{BP})+\begin{cases}8\text{位位移量}\\16\text{位位移量}\end{cases}$$

如果指令指定了段跨越前缀，计算物理地址时，则由指定的段寄存器提供段地址。

例 2-6 设（DS）=3000H，（SI）=2000H，ARRAY=4000H ，指令为：

```
MOV  AX，ARRAY [SI]
```

存源操作数单元的物理地址=16×(DS) + (SI) +ARRAY=30000H+2000H+4000H=36000H

如果（36000H) = 1234H，指令执行结果为：（AX）=1234H

该指令的寻址方式如图 2.6 所示。由图可见,位移量 ARRAY 实际上就是数组的起始地址，(SI) =2000H 是要访问的单元在数组内的位置偏移量。

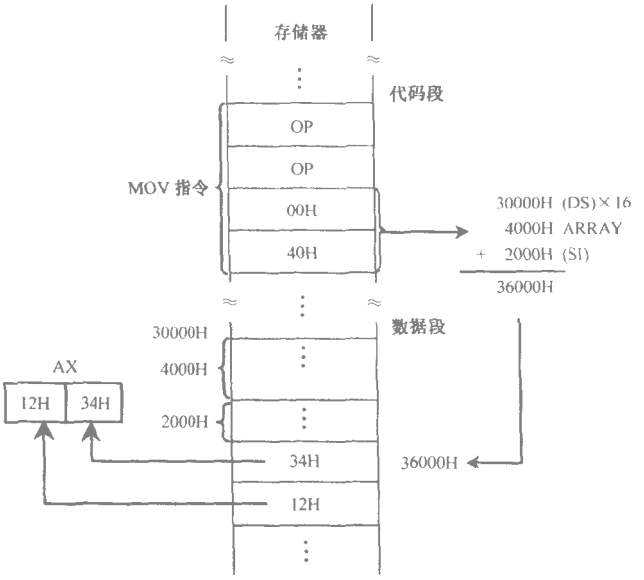


图 2.6 变址寻址方式

上述指令也可写成：MOV AX，[ARRAY+SI]，两条指令执行结果相同。

变址寻址方式常用于处理数组。例如数组起始地址为 ARRAY ，每执行一次指令后，修改基址寄存器或变址寄存器（此例为 SI）的内容，就可获得数组中的下一项数据。

6. 基址变址寻址方式（Based Indexed Addressing）

在这种寻址方式下，操作数的偏移地址是基址寄存器和变址寄存器中的值之和，具体使用的两个寄存器都在指令中指定。

如果指令中指定使用 BX 寄存器，将由 DS 提供段地址，操作数的物理地址按下式计算；

$$\text{操作数的物理地址}=16\times(\text{DS})+(\text{BX})+\begin{cases}(\text{SI})\\(\text{DI})\end{cases}$$

如果指令中指定使用 BP 寄存器，则由 SS 提供段地址，操作数的物理地址按下式计算：

$$\text{操作数的物理地址} = 16\times(SS) + (BP) + \begin{cases}(SI)\\(DI)\end{cases}$$

例 2-7　设（DS）=2100H，（BX）=0158H，（DI）=1000H　，指令为：

```
MOV  AX，[BX][DI]
```

存源操作数的单元物理地址=16×(DS) + (BX) + (DI) =21000H+0158H+1000H=22158H

如果(22158H) =1234H，指令执行结果为：（AX）=1234H。指令的寻址方式如图 2.7 所示。由图可见，BX 中是数组的起始地址，(DI) =1000H 是要访问的单元在数组内的位置偏移量。

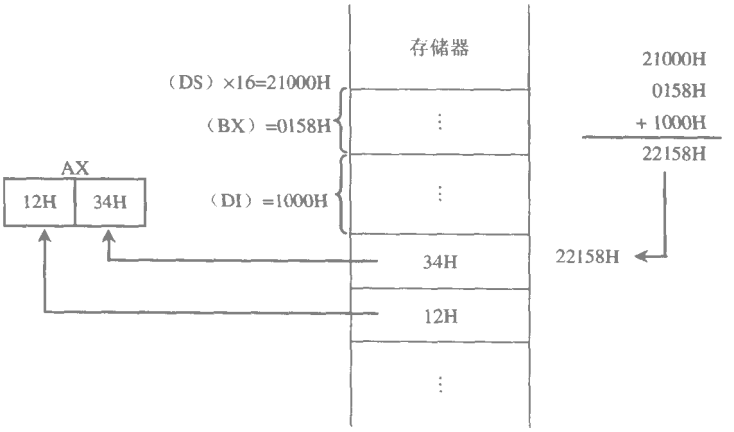


图 2.7　基址变址寻址方式

上述指令也可以写成如下的等价形式：

```
MOV  AX，[BX+DI]
```

基址变址寻址方式常用于处理数组。数组起始地址可存于基址寄存器 BX 中，再利用变址寄存器（SI 或 DI）指定要访问的单元在数组中的位置偏移量。这种寻址方式比前两种处理数组的寻址方式更灵活，可以修改两个寄存器的内容（根据需要修改数组起始地址或数组元素的位置偏移量）。

7. 相对基址变址寻址方式（Relative Based Indexed Addressing）

在这种寻址方式下，操作数的偏移地址是基址寄存器、变址寄存器中的值与 8 位或 16 位位移量之和，其中位移量包含于指令中。

这种寻址方式与前一种基址变址寻址方式只有一点差别，就是形成物理地址时增加了一个 8 位或 16 位的位移量。类似地，操作数的物理地址计算式为：

$$\text{操作数的物理地址} = 16\times(DS) + (BX) + \begin{cases}(SI)\\(DI)\end{cases} + \begin{cases}8\text{位位移量}\\16\text{位位移量}\end{cases}$$

或

$$\text{操作数的物理地址} = 16\times(SS) + (BP) + \begin{cases}(SI)\\(DI)\end{cases} + \begin{cases}8\text{ 位位移量}\\16\text{ 位位移量}\end{cases}$$

例 2-8　设（SS）=3000H，（BP）=2000H，（SI）=1000H，ARRAY=0250H，　指令为：

```
MOV  AX，ARRAY [BP][SI]
```

存源操作数的单元物理地址:=16×(SS) + (BP) + (SI)+ARRAY

=30000H+2000H+1000H+0250H

=33250H

如果(33250H) =1234H，指令执行结果为：(AX) =1234H。

指令的寻址方式如图 2.8 所示。由图可见，BP 给出栈顶地址，位移量 ARRAY 是堆栈内数组的起始地址，(SI) 是要访问的单元在数组内的位置偏移量，指令完成的操作就是从堆栈内数组 ARRAY 中的 1000H 单元（物理地址为 33250H）取出数 1234H 送 AX 寄存器。

相对基址变址寻址方式的用途是便于访问堆栈中的数组。一般用 BP 指向栈顶,用位移量表示从栈顶到数组起点的偏移量（即堆栈内数组的起始地址），再由变址寄存器指定要访问单元在数组中的位置偏移量。采用这种寻址方式可以访问堆栈中的任一个存储器单元。

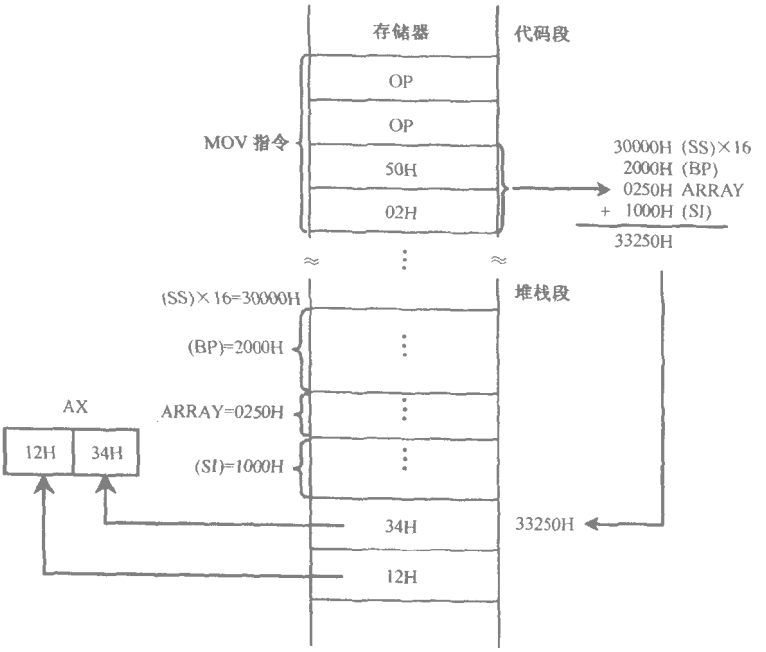


图 2.8 相对基址变址寻址方式

2.2 指令系统

8086/8088 指令系统中的指令按功能可分为下列 6 类：

①数据传送指令　　②算术运算指令

③逻辑运算与移位指令　　④串操作指令

④控制转移指令　　⑥ 处理机控制指令

本节将分类介绍指令系统的指令，其中少数指令（例如转移指令、循环控制指令及输入/输出指令等）将在后续的章节中结合相关内容介绍。

2.2.1 数据传送指令

数据传送指令的功能是把数据或地址传送到寄存器或存储器单元中。

2.2.1.1 通信传送指令

1. 传送指令 MOV

指令格式：MOV dst，src

执行操作：（dst）←（src）

功能：在 CPU 与存储器之间或 CPU 内部传送数据。

标志位：不受影响。

指令中 dst 表示目的操作数，src 表示源操作数。

例 2-9 MOV AL，'E'

指令执行后，字符'E'的 ASCII 码值送入 AL 寄存器中。

MOV 指令可以有下面几种形式；

① MOV mem/reg1，reg2

② MOV reg1，mem/reg2

③ MOV mem/reg，imm

其中，mem 表示存储器单元，reg 表示寄存器，imm 表示立即数，reg/mem 表示 dst 或 src 分别可为寄存器或存储器单元。8086/8088 指令不允许两个操作数都是存储器单元，除了源操作数为立即数的情况外，两个操作数中必须有一个是寄存器寻址方式，并且目的操作数不允许是立即数或 CS 段寄存器。

例 2-10 下列指令都是合法指令：

```
MOV    DS，AX        ; 寄存器传送到段寄存器
MOV    AX，BX        ; 寄存器传送到寄存器
MOV    VAL，CX       ; 寄存器传送到存储器单元
MOV    CX，504AH     ; 立即数传送到寄存器
MOV    VAL，3047H    ; 立即数传送到存储器单元
MOV    DX，VAL       ; 存储器单元传送到寄存器
```

下列指令是非法指令：

```
MOV    45，DL
MOV    VAL1 . VAL2
MOV    CS，AX
```

2. 入栈指令 PUSH

指令格式：PUSH src

执行操作：（SP）←（SP）-2，((SP) +1，(SP)) ←（src）

功能：将源操作数指定的字数据压入堆栈栈顶。

标志位：不受影响。

指令有下列形式：

PUSH reg/mem

例 2-11 PUSH AX

设（AX）= 3104H，指令执行情况如图 2.9 所示。

指令执行的过程是首先将堆栈栈顶指针寄存器 SP 的值减 2，使 SP 指向新的栈顶，然后将 AX 中的值送入新栈顶。

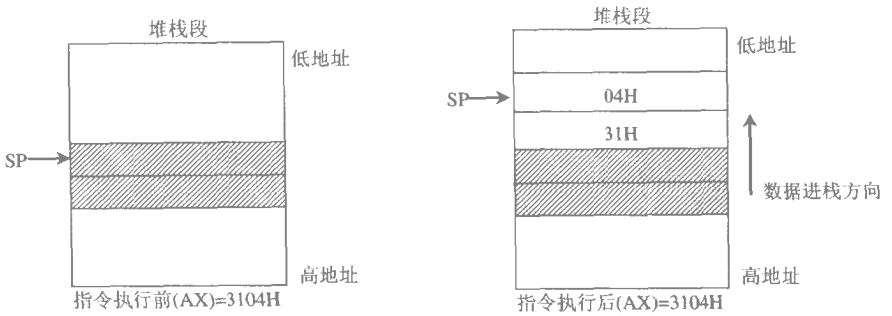


图 2.9 PUSH 指令执行情况

3. 出栈指令 POP

指令格式： POP dst

执行操作：（dst）←（(SP) +1，(SP)），(SP) ←（SP）+2

功能：将栈顶的一个字数据出栈，送到目的操作数指定的存储器单元或寄存器中。

标志位：不受影响。

指令形式有下列两种：

① POP reg/mem

② POP segreg

例 2-12 POP BX

设（BX）=1000H **，指令执行情况如图** 2.10 **所示。**

指令执行时，首先将栈顶的字数据送入 BX，**再将** SP **加** 2，**使** SP **指向数据出栈后的新栈顶。注意** POP **指令的目的操作数不能是** CS **段寄存器。**

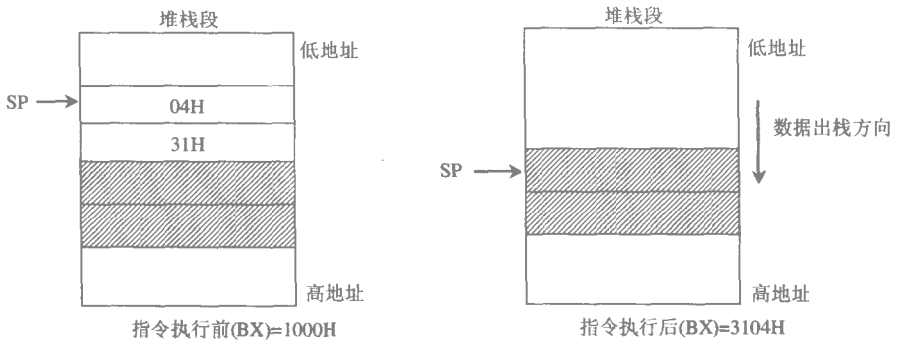


图 2.10 POP 指令执行情况

PUSH、POP **指令都只能做字操作，因为数据进栈或出栈以字为单位。这两条指令常用于将工作寄存器的内容、子程序或中断服务程序的返回地址等现场信息入栈保护或出栈恢复。**

4. 交换指令 XCHG

指令格式： XCHG oprl，opr2

执行操作：（oprl）↔（opr2）

功能：将两个操作数互换。

标志位：不受影响。

该指令有下列形式：