第 3 章　80x86 微机系统的组成

3.1　基于 80x86 的计算机组织结构

计算机硬件按功能分为五个组成部分：控制器、运算器、存储器、输入设备和输出设备，如图 3.1 所示。各功能部件通过数据总线、地址总线和控制总线等联系，在控制器统一控制与指挥下，按照指令的功能要求协调地工作着。

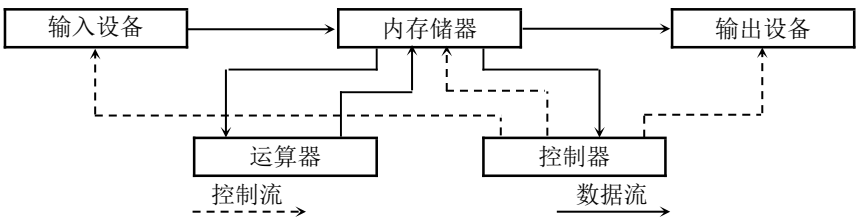

图 3.1　计算机硬件的基本组成

计算机中的所有数据和指令均以二进制形式存在内存中，存放位置由地址确定，并且所要执行的程序是预先存入内存中，以便计算机能够在控制器协调下，自动地从内存中读取指令并执行。控制器中有一个程序计数器(Program Counter，PC)，总是存放着一条待执行指令在内存中的地址，程序计数器控制着程序的执行顺序。

运算器和控制器是相对独立的两大部分，现在则常把它们组成一个整体，叫中央处理器 CPU(Central Processing Unit)，是计算机硬件中的核心部分。在微机系统中，CPU 已被集成在一片大规模或超大规模的集成电路上，所以又称为微处理器(Microprocessor)。

目前，使用最广泛的微机系列，都是以 Intel 公司的 80x86 系列微处理器或者兼容的微处理器为 CPU。其典型的系统结构如图 3.2 所示。

CPU 执行所有的算术和逻辑运算指令，以及负责全系统的控制工作；内存储器是计算机的记忆部件，所有的指令和数据均存放在这里；I/O 接口负责与外界的设备间的联系，例如，键盘、显示器、打印机等输入输出设备，以及硬盘、光盘等外部存储器设备。这些内容都是学习汇编语言必须要了解的。

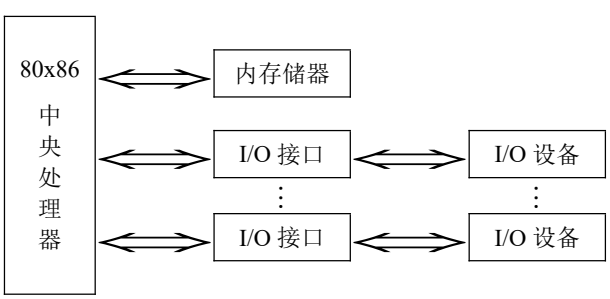

图 3.2　80x86 计算机系统结构

80x86 系列始于 1978 年 Intel 公司生产的 8086，先后有：8088, 8086, 80286, 80386, 80486 及之后的 Pentium 系列。通常所说的 80x86 系列还包括 AMD，Cyrix 等公司生产的系列芯片，它们在通用指令和体系结构方面是完全兼容。

在 80386 之前，80x86 系列是 16 位 CPU，以后的系列微处理器是 32 位的，属于 IA-32 体系结构。虽然 32 位处理器家族已经发展到目前的 Pentium-Ⅳ阶段，但从程序员角度看，除了性能有所提升外，IA-32 的体系结构没有实质性的改变。

32 位处理器提供了三种模式：实模式，即相当于高速运行的 8086 芯片，用于和 8086 等芯片兼容；保护模式；模拟 8086 模式，即在保护模式下，模拟多个 8086 处理器工作。

字长和地址总线宽度是衡量处理器的两个重要指标。其中，字长是指处理器在单位时间内一次能处理的二进制位数，是由处理器的内部结构决定的。通常寄存器的位数也就反映了机器的字长。16 位处理器字长是 16，32 位处理器的字长是 32。地址总线宽度决定了处理器的寻址空间，即可以访问内存的最大范围，如 8086 处理器有 20 根地址总线，可以访问 2^{20} (1M) 字节单元，Pentium-Ⅳ处理器有 36 根地址总线，寻址空间是 0～2^{36}-1，可以访问 2^{36} (64G) 字节单元。

3.2 CPU 资源

CPU 的主要功能是执行存放在内存储器中的指令序列。为此，除了要完成算术、逻辑操作外，还需要担负 CPU 和内存储器，以及 I/O 接口之间的数据传送任务。它的基本结构如图 3.3 所示。

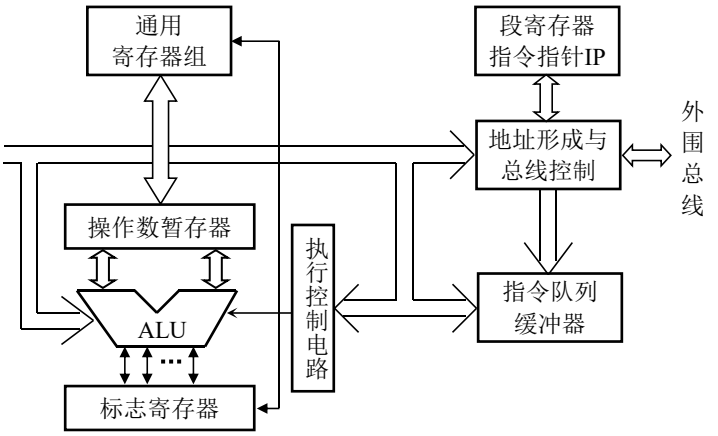

图 3.3　80x86 微处理器基本结构示意图

传统的 CPU 由运算器和控制器两大部分组成。但是随着高密度集成电路技术的发展，一些 CPU 外部的逻辑功能部件纷纷移入 CPU 内部。例如，早期的 80x86 芯片只包括运算器和控制器两大部分。从 80386 开始，为使内存储器速度能更好地与运算器的速度相匹配，使用 Cache 技术，在芯片内引入高速缓冲存储器。其后生产的 CPU 芯片随着半导体器件集成度的提高，片内高速缓冲存储器的容量也逐步扩大，但这部分器件就其功能而言还是属于内存储器的。

对于汇编语言程序设计者，CPU 中各寄存器、内存储器和 I/O 端口是他们编程的主要资源，而且大部分指令是通过寄存器来实现对操作数的预定功能。

3.2.1　控制器与运算器

1. 运算器

运算器是对数据进行加工处理的部件，它主要通过执行算术和逻辑运算操作，来完成对数据的加工和处理。参与运算的数(称为操作数)由控制器指示，从内存储器或寄存器内取到运算器，运算结果存放到寄存器或内存储器。

运算器基本都是由算术/逻辑运算单元(ALU)、累加器(ACC)、寄存器组、多路转换器和数据总线等逻辑部件组成。

2. 控制器

控制器的主要功能是从内存中取出指令，并指出下一条指令在内存中的位置。将取出的指令经指令寄存器送往指令译码器，经过对指令的分析，发出相应的控制和定时信息，控制和协调计算机的各个部件的工作，以完成指令所规定的操作。

控制器一般由指令指针寄存器(IP)、指令寄存器(IR)、指令译码器(ID)、标志寄存器、控制逻辑电路和时钟控制电路等组成.。

其中 IP(在 32 位处理器中称为 EIP)，其实就是程序计数器 PC。IP 总是指向下一条要取的指令在内存中的位置，由此实现程序的自动执行。

随着高密度、大规模集成电路技术的发展，新推出的 CPU 不断增加了新的处理和控制功能。例如，浮点运算单元(Float Point Unit，FPU)是专用于浮点运算的处理器，以前作为一种单独芯片而存在，但从 80486 开始，Intel 将 FPU 集成在 CPU 芯片中了。如今的 80x86 系列 CPU 中已陆续增加了 MMX, 3DNow, SSE, SSE2, SSE3 等指令集。

尽管 80x86 系列 CPU 的功能日益强大，指令日益丰富，但是在这本书中只介绍 CPU 的基本功能和常用的基本指令集，以此来讲述汇编语言程序设计的基本内容和基本方法。

3.2.2　80x86 寄存器组

本节介绍与汇编语言程序设计密切相关的寄存器组，这些寄存器有的属于运算器，如累加器 AX，有的分布在控制器中，如指令指针 IP，总之是包含在 80x86 处理器中的。

8086/8088、80286 这类 16 位 CPU 有 14 个基本寄存器：AX, BX, CX, DX, SP, BP, DI, SI, IP, FLAGS, CS, DS, ES, SS，这些寄存器都是 16 位的。

32 位 CPU 有 16 个基本寄存器：EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI, EIP, EFLAGS, CS, DS, ES, SS, FS, GS，多出的两个是：FS, GS。在这 16 个寄存器中，以“E”打头的寄存器都是 32 位寄存器，其余的是 16 位寄存器。所有 32 位寄存器都可用作 16 位寄存器，例如，EAX 是 32 位寄存器，可以作为 16 位寄存器 AX 来使用，对应 EAX 的低 16 位。若上述所有 32 位寄存器都作为 16 位寄存器使用，则 32 位的 CPU 与 16 位的 8086 没有什么两样。

80x86 的基本寄存器可分为三类：通用寄存器、控制寄存器和段寄存器，如图 3.4 所示。

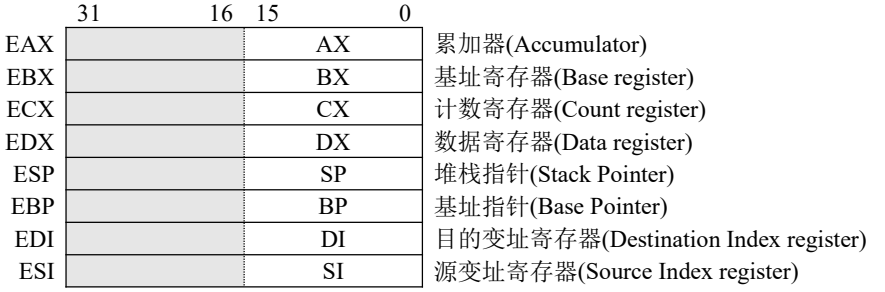

图 3.4　80x86 的基本寄存器

注：① 对于 16 位 CPU，灰色区域寄存器是不存在的。
② FS 和 GS 段寄存器没有专用名称。

下面介绍 32 位 CPU 中的基本寄存器组，关于 16 位 CPU 的基本寄存器的有关情况，可以参照此处说明。所说明的内容比较繁杂，初学者可能不好掌握，不过没有关系，在以后章节中还会进一步讲解。

1. 通用寄存器

通用寄存器(General Register)共有 8 个，主要用于算术运算、逻辑运算和数据的传送。每个寄存器都是 32 位的，但又可作为一个 16 位寄存器来使用。根据使用情况分三种：4 个数据寄存器、2 个地址指针寄存器、2 个变址寄存器。

8 个 32 位寄存器都可作为内存指针(存放的是内存地址)来使用，但对于 16 位寄存器来说，BX、BP、SI 和 DI 可用作内存指针，AX、CX、DX 和 SP 不能作为内存指针使用。

这类寄存器虽然是通用寄存器，但各自又有专用用途。

1) 数据寄存器

数据寄存器(Data Register)指的是 EAX, EBX, ECX, EDX，一般用于存放参与运算的操作数或运算结果。这四个 32 位寄存器中的低 16 位又可作为 16 位寄存器来使用，分别记作 AX, BX, CX, DX。四个 16 位寄存器中的高 8 位、低 8 位又可独立作为两个 8 位寄存器来用。高 8 位分别记作 AH, BH, CH, DH，低 8 位分别记作 AL, BL, CL, DL。如图 3.5 所示。

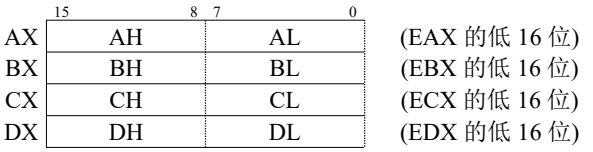

图 3.5　16 位数据寄存器的构成

(1) EAX。作为累加器使用，是算术运算所使用的主要寄存器。8 位、16 位和 32 位累加器分别对应 AL, AX 和 EAX。在乘除等指令中指定用累加器存放操作数。此外，所有的 I/O 指令都使用累加器与外设端口交换信息。

(2) EBX。可用作基址寄存器，或作为内存储器指针来使用。

(3) ECX。作为计数器使用。8 位、16 位和 32 位计数器分别对应 CL，CX 和 ECX。在循环(LOOP)、串处理及某些移位指令中用作隐含的计数器。

(4) EDX。数据寄存器。在做 16 位乘法运算时，乘法运算结果(32 位)存放在 DX:AX 中，其中 DX 存放高 16 位；在做 32 位乘法运算时，乘法运算结果(64 位)存放在 EDX:EAX 中，其中 EDX 存放高 32 位；在做 32 位除法运算时，被除操作数(32 位)存放在 DX:AX 中，其中

DX 存放高 16 位，除法运算结果的余数存放在 DX 中；在做 64 位除法运算时，被除操作数(64 位)存放在 EDX:EAX 中，其中 EDX 存放高 32 位，除法运算结果的余数存放在 EDX 中。在寄存器间接寻址的 I/O 指令中，DX 存放 I/O 端口地址。

2) 指针寄存器

指针寄存器(Pointer Register)包括 ESP 和 EBP，其对应的 SP 和 BP 寄存器可作为 16 位内存指针使用。

(1) ESP。堆栈指针。专门用以访问堆栈上数据的寄存器，一般不应该在算术运算和数据传送中使用。32 位模式下使用 ESP，16 位模式下使用 SP，其内容始终指向堆栈栈顶。从这一点上看，ESP/SP 是专用的。

(2) EBP。基址指针。可以用来存放数据，但更经常、更重要的用途是作为堆栈区的一个基地址，以便访问堆栈中的数据。EBP 或 BP 一般不应该在算术运算和数据传送中使用。

3) 变址寄存器

变址寄存器(Index Register)包括 ESI 和 EDI，其对应的 SI 和 DI 寄存器可作为 16 位内存指针使用。

(1) ESI。源变址寄存器。作为串处理指令中隐含的源变址寄存器，32 位模式下使用 ESI，16 位模式下使用 SI，指向源串在内存中的起始地址。

(2) EDI。目的变址寄存器。作为串处理指令中隐含的目的变址寄存器，32 位模式下使用 EDI，16 位模式下使用 DI，指向目的串在内存中的起始地址。

2. 控制寄存器

控制寄存器(Control Register)包括 EIP 和 EFLAGS。

EIP。用于存放下一条要执行的指令的内存地址。32 位模式下使用的是 EIP，16 位模式下使用的是 IP。在程序运行期间，CPU 会自动修改(E)IP 的内容，以使它始终指向下一条要执行指令的内存偏移地址，从而实现程序的自动执行。用户程序不能直接修改(E)IP，但是随着指令的执行，(E)IP 的内容会相应地变动。

EFLAGS。用以保存一条指令执行后，CPU 所处的状态信息及运算结果的特征。主要有：运算结果标志、状态控制标志和系统状态标志等寄存器。16 位模式下是 FLAGS。

EFLAGS 有较多的信息位，如图 3.6 所示。但作为基本编程只使用其中的 9 个标志位：6 位运算结果标志和 3 位状态控制标志。

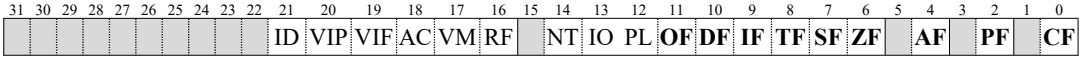

图 3.6　Pentium 处理器的标志寄存器

1) 运算结果标志

运算结果标志用来保存算术运算指令、逻辑运算指令及各类测试指令的结果状态，为后续指令的执行提供执行依据。

(1) 进位标志 CF(Carry Flag)。当进行加或减运算时，若最高位发生进位或借位，则 CF 为 1，否则为 0。该标志可用于判断无符号数运算结果是否溢出。80x86 用于设置 CF 位的指令有：STC(CF 置 1)、CLC(CF 清 0)及 CMC(CF 取反)。

(2) 奇偶标志 PF(Parity Flag)。反映运算结果中 1 的个数的奇偶性。含有偶数个 1，PF 为 1，否则为 0。

(3) 辅助进位标志 AF(Auxiliary Flag)。执行一条加法或减法运算指令时，最低 4 位向高位有进位或借位时，AF 为 1，否则为 0。

(4) 零标志 ZF(Zero Flag)。当前的运算结果为 0，则 ZF 为 1，否则为 0。

(5) 符号标志 SF(Sign Flag)。运算结果的最高位为 1，SF 为 1，否则为 0。也就是说该标志反映的是补码数正负性。

(6) 溢出标志 OF(Overflow Flag)。当运算结果超出了补码所能表示的范围，即溢出时，OF 为 1，否则为 0。用来判断有符号数运算结果是否溢出。

在应用程序中，以上介绍的 6 个标志位中，ZF, OF, CF 和 SF 经常使用，而 PF 和 AF 较少使用。

CPU 每执行一条算术运算指令或逻辑运算指令，都要根据运算的结果状态来设置各标志位。例如，执行 8 位操作：1Bh−8Dh，运算结果为 18Eh，根据此结果将各标志位设置为：CF＝1, PF＝1, AF＝1, ZF＝0, SF＝1, OF＝1；7Fh∧95h，运算结果为 15h，据此设置的各标志位为：CF＝0, PF＝0, AF＝0, ZF＝0, SF＝0, OF＝0。

2) 状态控制标志

状态控制标志用来控制 CPU 的操作，可通过专门的指令设置或清除。

(1) 陷阱标志 TF(Trap Flag)。若 TF 置 1，则 CPU 每执行完一条指令，便产生一个单步中断。这主要用于程序的调试。

(2) 中断允许标志 IF(Interrupt–enable Flag)。若 IF 置 1，则表示允许 CPU 响应外部从 INTR 引脚上发来的 I/O 中断请求；若 IF 清 0，则禁止 CPU 响应 I/O 中断请求。80x86 中专门用于将 IF 置 1 的指令是 STI，专门用于将 IF 清 0 的指令是 CLI。

(3) 方向标志 DF(Direction Flag)。若 DF 清 0，串操作指令按加方式改变相关的(E)SI 和(E)DI 的值；若将 DF 置 1，串操作指令按减方式改变相关的(E)SI 和(E)DI 的值。80x86 中专门用于将 DF 清 0 的指令是 CLD，专门用于将 DF 置 1 的指令是 STD。

3. 段寄存器

在 80x86 系统中，将一个连续字节单元的内存区域称为一个段，段寄存器(Segment Register)就是专门用于存放指示该段首地址的相关内容。

16 位 CPU 有四个段寄存器：CS, DS, SS 和 ES，32 位 CPU 又增加了两个：FS 和 GS。

在应用程序中,一般以分段方式来使用内存空间，通常内存是分为三个段(区)：代码段(区)、数据段(区)和堆栈段(区)，它们的基地址分别通过 CS, DS 和 SS 来确定，即 CS, DS 和 SS 中存放的内容分别是代码段、数据段及堆栈段的起始地址(16 位地址模式)或段选择器(32 位地址模式)。

3.3 内 存 储 器

内存储器(简称内存)主要用于存放 CPU 要执行的指令和所处理的数据，它的存取基本单位是字节(Byte)。内存中的每个字节单元都有一个唯一编号(从 0 开始顺序递增)，称为地址，CPU 等通过地址来访问内存单元。

CPU 能够访问最大的内存单元的地址范围称为寻址空间，它是由地址总线宽度决定的。8086/8088 地址总线是 20 位，可访问字节单元地址范围是 00000h～FFFFFh，共 2^{20} 字节(1MB)。

80386、80486 和 Pentium 的地址总线宽度为 32 位，相应的地址范围是 00000000h～FFFFFFFFh，共 2^{32} 字节(4GB)。而目前主流的 Pentium-Ⅳ的地址总线宽度为 36 位，相应的地址范围是 000000000h～FFFFFFFFFh，共 2^{36} 字节(64GB)。

3.3.1　内存单元与数据存放格式

内存的存取的基本单位是字节。连续若干个字节可以作为一个内存单元来使用，这样便有：1 字节、2 字节、4 字节、8 字节等不同类型的内存单元。2, 4, 8 字节单元又叫做字(Word)、双字(DWord)、四字(QWord)单元。

内存单元是以它所占用内存块的起始地址来标识，正因为如此，同一个地址可以用来存取不同类型的内存单元。如图 3.7 所示，通过 100 号地址可以访问：字节单元、2 字节单元与 4 字节单元等。虽然它们都是有 100 号地址表示，但是 100 号字节单元只占用 1 个字节空间，100 号 2 字节(字)单元表示的是首地址为 100 的 2 字节空间，4 字节(双字)单元所表示的是首地址为 100 的 4 字节空间。

单字节类型数据以字节单元为单位存放，如 ASCII 字符；多字节型数据以多字节单元作为单位存放，如 16 位有符号整数，是以 2 字节内存单元为单位存放的，32 位浮点数，则以 4 字节内存单元为单位存放。

在 80x86 系统中多字节类型数据是按“小端字节序”存放：低位数据存放在低地址单元，高位数据存放在高地址单元，所以多字节型数据的位编号是按地址从低到高的顺序进行排列的，如图 3.7 所示。

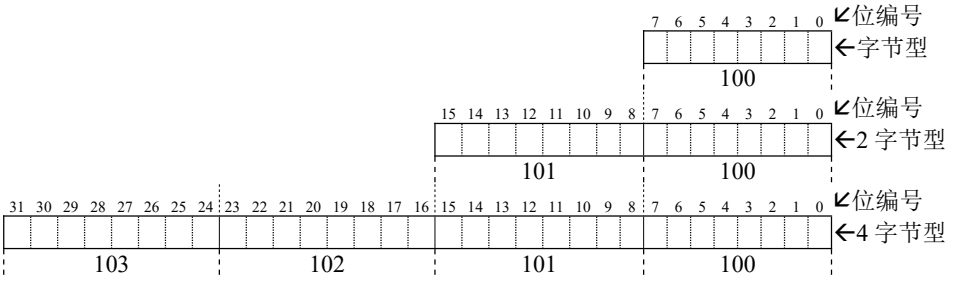

图 3.7　用同一个地址标识不同类型的内存单元示意图

例如，将 32 位 12345678h 存放到 10 号 4 字节单元中，则存放的字节顺序如图 3.8 所示。

地址	…	10	11	12	13	14	…	91	92	93	94	95	96	…
内容	…	**78**	**56**	**34**	**12**	CD	…	1B	50	70	B2	A8	00	…

图 3.8　内存数据存放格式示意图

此处，10 号地址是 4 字节(双字)单元地址。实际上 10 号地址还可以表示字节单元、2 字节(字)单元、8 字节(四字)单元。如图 3.8 中，10 号地址的字节单元内容是 78h；10 号地址的字单元内容是 5678h。

连续若干字节中的二进制代码，可以是无符号数、补码、字符编码等。例如在图 3.8 中，从 92～95 号字节中的二进制代码分别是：50h,70h,B2h,A8h，若用 16 位无符号规则，则表示的是两个数：$(7050)_{16}$、$(A8B2)_{16}$；若用 32 位无符号编码规则，则表示的是一个数：$(A8B27050)_{16}$；若用 16 位补码规则，则表示的是两个数：$(7050)_{16}$、$-(574E)_{16}$；若 32 位补码规则，则表示的

是一个数：−(574D8FB0)$_{16}$；若 ASCII 字符编码规则，则表示的是两个 ASCII 字符：'P'，'p'，…。到底是哪一种编码，这要看使用时的约定。

这里所列的例子是通过地址编号(如 10 号地址)来存取内存单元，但是在汇编语言源程序中，通常用变量名或标号等符号地址来访问内存单元。

3.3.2 内存的分段使用

80x86 系统采用内存分段(区)的方法来访问内存。

连续若干字节的内存区域称为内存段。如图 3.9 所示，内存段一般用基地址(base)和长度(length)确定，段内单元地址用偏移地址(offset，相对于 base 的偏移量)来表示。这样内存中单元的地址由两部分组成，即 base 和 offset。在 80x86 中经常用 EA(Effective Address)表示偏移地址。

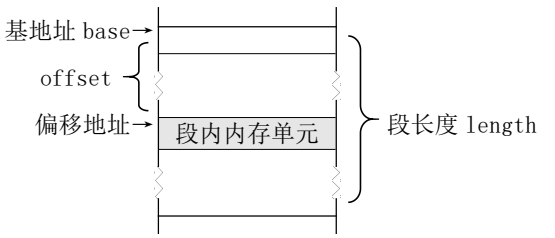

图 3.9 内存段示意图

在 80x86 系统中，内存段基地址由段寄存器来确定，所以访问内存单元的地址形式为：

段寄存器：偏移地址

段寄存器可以是 CS, DS, SS, ES, FS, GS 之一；偏移地址可以直接是地址编号，也可以是存有偏移地址的寄存器。段寄存器与偏移地址有如下的默认组合关系：

(1) 偏移地址存放在 EIP/IP 中，默认使用 CS 段寄存器；

(2) 偏移地址存放在 ESP/SP, EBP/BP 中，默认使用 SS 段寄存器；

(3) 在串处理指令中，存放在 EDI/DI 的偏移地址默认与 ES 段寄存器组合；

(4) 其他形式的偏移地址，默认使用 DS 段寄存器。

在没有明确指定段寄存器的情况下，则使用默认段寄存器。例如，地址 DS:[10]可以直接写成[10]，而地址[EBP]的默认组合是 SS:[EBP]。

一般应用程序将内存分成三种类型区域：代码段(区)、数据段(区)、堆栈段(区)。如图 3.10 所示，代码内存区用来存放程序的指令代码，其起始地址由 CS 段寄存器确定；数据内存区用来存放程序中静态数据，其起始地址由 DS 段寄存器确定；堆栈内存区为程序中的堆栈提供存储区域，它的起始地址由 SS 段寄存器确定。

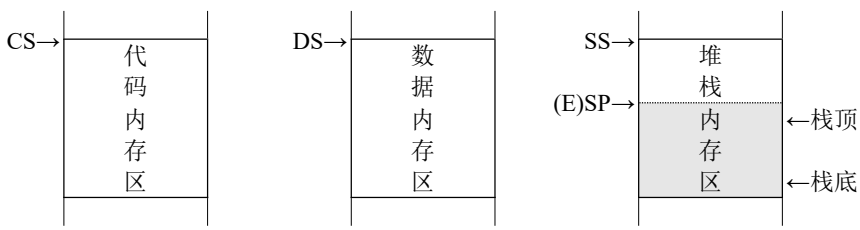

图 3.10 一般应用程序的内存分区示意图

堆栈对实现子程序的调用非常重要，下面对堆栈区作一些简要说明。

堆栈(Stack)是由连续若干个内存单元组成的、按先进后出(First In Last Out, FILO)或后进先出(Last In First Out, FILO)原则进行存取的数据结构。在堆栈结构中，最先存入数据的单元称为栈底，最后存入数据的堆栈单元称为栈顶。通常栈底是固定不变的，而栈顶却是随着数据的进栈和出栈不断变化的。在堆栈操作中，数据按顺序存入堆栈称为数据**进栈**(PUSH)或**压入**；从堆栈中按与进栈相反的顺序取出数据称为**出栈**(POP)或**弹出**。

堆栈栈顶是浮动的，为此专门设有一个指针——堆栈指针(Stack Pointer, SP)，使之始终指向栈顶。这样，进栈和出栈限定在栈顶进行，由 SP 确定存取位置。

在一般程序中专门划出一块内存区供堆栈使用，这便是堆栈区。堆栈区大小设置应该满足应用程序中堆栈使用要求，不能出现堆栈超出堆栈区的现象，否则堆栈“溢出”。

在 80x86 中，寄存器 SS 专用于确定堆栈区的起始地址，(E)SP 专门用于指示栈顶的位置，即总是指向最近进栈的数据位置。栈底固定为栈区的最高地址单元，所以，执行 PUSH 操作，(E)SP 要向低地址方向移动，执行 POP 操作，(E)SP 要向高地址方向移动。

80x86 的栈单元基本单位是 2 字节。在 16 位 CPU 中，只能以 2 字节为单位进行进栈和出栈操作，在 32 位 CPU 中，栈存取单位一般是 4 字节，但也可以按 2 字节操作。

例如，设堆栈区为从 1000h 号地址开始、连续 800h 字节的内存区，那么第一个 2 字节单元地址是 1000h，最后一个 2 字节单元(即栈底)地址是 17FEh，如图 3.11 所示。图 3.11(a)是初始状态，ESP 所指向栈单元的数据是 7050h；图 3.11(b)是在初始状态下 16 位数据 1234h 进栈后的状态，容易看出，ESP 已经向低地址方向移动 2 字节；图 3.11(c)是在初始状态下，32 位数据 12345678h 进栈后的状态，容易看出，ESP 已经向低地址方向移动了 4 字节；图 3.11(d)是在初始状态下，从堆栈中弹出一个 16 位数据后的状态，容易看出，ESP 向高地址方向移动 2 字节，弹出的数据是 7050h；图 3.11(e)是在初始状态下，从堆栈中弹出一个 32 位数据后的状态，容易看出，ESP 向高地址方向移动 4 字节，弹出的数据是 A8B27050h。(弹出数据时，原数据并没有被破坏)

在程序中，堆栈有几种重要的用途：

(1) 堆栈可用于临时保存寄存器内容，在寄存器使用完毕之后，可恢复其原始值。

(2) 在调用子程序时，CPU 用堆栈保存当前过程的返回地址。

(3) 在调用子程序时，可以通过堆栈传递参数。

(4) 过程内的局部变量在堆栈上创建，过程结束时，这些变量被丢弃。

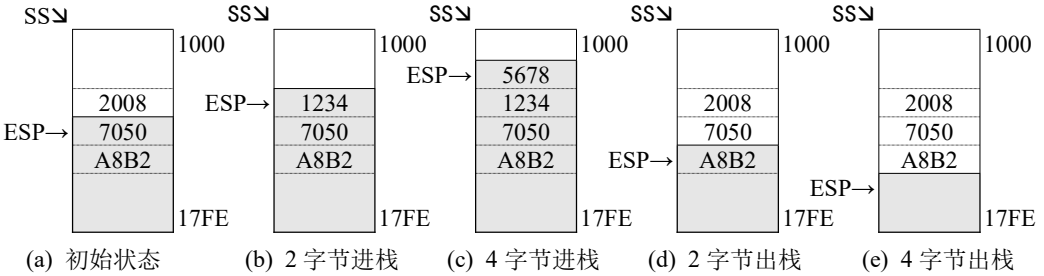

图 3.11　堆栈示意图

在 80x86 指令中，进栈指令有：PUSH, PUSHF, PUSHA, PUSHFD, PUSHAD 等，出栈指令有：POP, POPF, POPA, POPFD, POPAD 等。此外，还有一些指令也要使用堆栈，如 CALL,

RET, INT, IRET 等。这些指令的功能将在以后章节陆续介绍。

3.3.3 内存寻址

前面说过，指令以“段:偏移”这样的形式访问内存单元，这种形式的地址称为逻辑地址，那么逻辑地址是如何对应到内存单元的物理地址的呢？在 16 位地址模式下和 32 位保护模式下，CPU 采用不同的对应规则将逻辑地址转换成物理地址。下面分别说明之。

1. 16 位地址模式内存寻址

在实地址模式和虚 8086 模式下，偏移地址都是 16 位，合称为 16 位地址模式。

16 位地址模式的地址宽度是 20 位，可以表示字节单元地址的范围是 00000～FFFFFh，即寻址空间是 2^{20} 字节(1MB)。而在 16 位模式下，偏移地址是 16 位的，可表示字节单元的地址范围是 0000～FFFFh，即最大长度是 2^{16} 字节(64KB)。也就是说，16 位地址模式的最大特征是：程序的寻址空间是 1MB，每个段的最大长度是 64KB。

在 16 位地址模式下，段寄存器里存放的是 20 位地址的高 16 位，逻辑地址“段:偏移”与 20 位物理地址间的对应关系是：

20 位地址＝段地址×16+偏移地址

例 3.1 设 CS＝1000h，DS＝1200h，SS＝2000h，BP＝2000h，求下列单元的 20 位地址：(1)CS:[1234h]；(2)[1234h]；(3)[BP]；(4)DS:[BP]。

解：(1) 20 位地址＝1000h×16+1234h＝10000h+1234h＝11234h

(2) 默认的段是 DS，所以，20 位地址＝1200h×16+1234＝13234h

(3) 默认的段是 SS，所以，20 位地址＝2000h×16+2000h＝22000h

(4) 20 位地址＝1200h×16+2000h＝14000h

例 3.2 某应用程序中的代码区、数据区和堆栈区的大小分别为：8KB(2000h), 2KB(800h), 2KB(800h)。此时可按如图 3.12 所示来为各段分配内存区。注意：虽然我们可说为各段分配了各自的长度，但是在 16 位地址模式下，CPU 并不检查每个段是否越界，所以当以某一个段寄存器访问内存单元时，有可能会超出本段。例如，字节单元 CS:[3000h]，就是落在代码段之外，实际上对应的是数据段中的一个字节单元(DS:[0210h])。

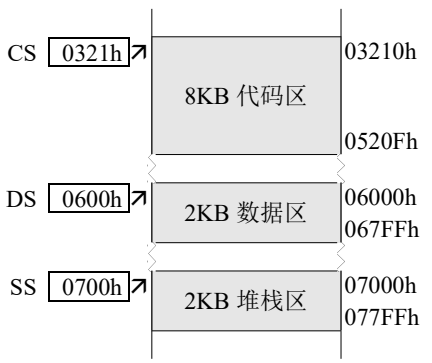

图 3.12 应用程序的各段内存分配示意图

3.4　I/O 地址空间

CPU 是通过输入输出接口(I/O 接口)与外部设备交换数据的，如图 3.14 所示。输入信息时，I/O 接口电路将输入设备的动作(如键盘敲键)转换成二进制编码，存放在自己的寄存器中，CPU 再从 I/O 接口的寄存器中读入编码，从而完成信息的输入；输出信息时，CPU 将数据送到 I/O 接口电路的寄存器，再由 I/O 接口电路将它转换成驱动外部设备的动作(如打印机打印)，从而完成信息的输出。

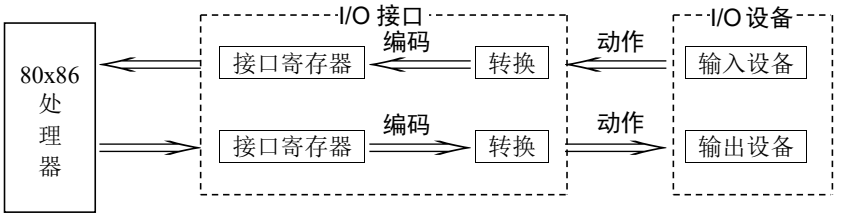

图 3.14　CPU 与外部设备交换数据示意图

这里所说的寄存器位于 I/O 接口部分，是接口寄存器，和位于 CPU 内的寄存器是完全不同的。在与外部设备交换数据时，I/O 接口寄存器主要用于暂存数据等。CPU 通过 I/O 接口寄存器与 I/O 设备交换信息。

80x86 将这些 I/O 接口寄存器称为 I/O 端口(Port)，使用 16 位地址总线来寻址这些端口，可以访问 2^{16} 字节(64KB)端口空间，其地址范围是 0000h～FFFFh，而且内存地址和 I/O 端口地址分布在两个独立的地址空间中(见图 3.2)。CPU 通过端口地址(端口号)来存取接口寄存器，从而实现与外部 I/O 设备交换信息。

80x86 系统的端口可以是 8 位或 16 位，80386 及更新的系统还可提供 32 位端口。

这 64KB 端口空间中绝大多数地址是空缺的，只有很小一部分对应有接口寄存器，这是因为系统中一般只有十几个外部设备和大容量存储设备。在不同型号 80x86 计算机，I/O 端口的编号有时不完全相同。在表 3.1 中列出了部分常用的端口地址。

表 3.1　部分常用的 I/O 端口地址

端 口 地 址	端 口 名 称	端 口 地 址	端 口 名 称
20h～23h	中断屏蔽寄存器	378h～37Fh	并行口 LPT2
40h～43h	时针/计数器	3B0h～3BBh	单色显示器端口
60h	键盘、扬声器输入端口	3BCh～3BFh	并行口 LPT1
200h～20Fh	游戏控制口	3C0h～3CFh	VGA/EGA
278h～27Fh	并行口 LPT3	3F0h～3F7h	磁盘控制器
2F8h～2FFh	串行口 COM2	3F8h～3FFh	串行口 COM1

在 80x86 系统中，专门用于端口输入的指令是：IN 及类似指令；专门用于端口输出的指令是：OUT IN 及类似指令。

程序员可以通过输入输出指令直接控制外部 I/O 设备，从而提高处理效率，具有一定的灵活性，但是实现的指令代码相当烦琐，而且要求程序员熟悉相关的硬件特性。

为了方便应用程序使用 I/O 设备，80x86 系统硬件制造商将系统的加电自检，引导装入，主要 I/O 设备的处理程序，以及接口控制等功能模块，固化在系统的 ROM 内存中，这便是通常所说的基本输入输出系统(Basic Input/Output System，BIOS)。

在 BIOS 之上，操作系统也为应用程序提供了常用的底层应用接口，例如，MS-DOS 是以 DOS 功能调用的形式提供，Windows 是以 API 的形式提供。