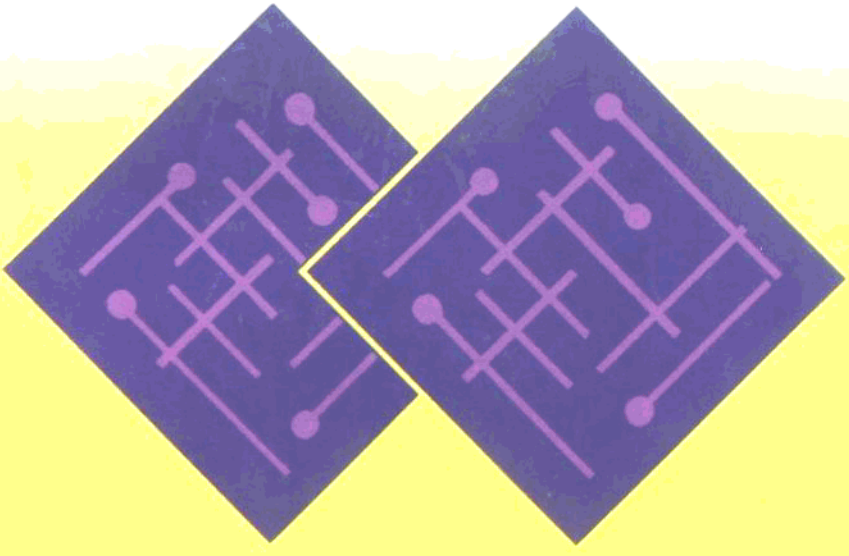
80X86 80X87汇编语言程序设计

洪志全
罗省贤 编著



电子科技大学出版社

80X86 80X87 汇编语言程序设计

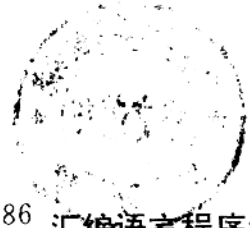
洪志全
罗省贤 编著

电子科技大学出版社
• 1996 •

内 容 提 要

汇编语言是学习计算机技术的最基本的语言之一。本书主要阐述了IBM PC 系列的计算机汇编语言程序设计的方法和技术，全书共八章：第一章到第六章介绍了 8086/8088 指令系统，以及基本的程序设计方法。第七章介绍了 80X86(80286、80386、80486)的指令系统、保护模式寻址机制、特权级、任务切换等。第八章介绍了 80X87(8087、80287、80387)的指令系统、寄存器结构，以及程序设计方法。在每一章中均附有习题，提供了大量的实际程序例程供读者学习参考。

在本书的编写和内容的安排方面，既考虑了基础内容(8086/8088)汇编语言的介绍，同时也对一些较新技术(如：VESA 标准图形接口、真彩色技术)的程序设计方法做了介绍，并把 80X86 和 80X87 的宏汇编程序设计方法融入本书中，这可以使读者全面了解 80X86 和 80X87 的汇编语言程序设计。本书既可作为初学者、计算机专业或非计算机专业的大专生、本科学生的汇编语言学习教材，也可作为研究生、广大的软件开发人员、工程技术人员的学习参考资料。



80X86 80X87 汇编语言程序设计

洪志全 罗省贤 编著

*

电子科技大学出版社出版

(成都建设北路二段四号)邮编 610054

成都理工学院印刷厂印刷

新华书店经销

*

开本 787×1092 1/16 印张 25.625 字数 610 千字

版次 1996 年 11 月第一版 印次 1996 年 11 月第一次印刷

印数 1—5000 册

中国标准书号 ISBN 7-81043-189-6/TP·185

定价：25.00 元

前　言

汇编语言是能够充分利用计算机硬件功能并可直接控制硬件的语言之一，对于编写性能要求很高的软件，汇编语言是最有效的语言，因此，汇编语言是软件设计不可缺少的重要工具。对于高等院校计算机硬件、软件专业及计算机应用专业的学生，"汇编语言程序设计"是必修的核心课程之一，学习这门课程对于训练学生掌握程序设计基本技能和调试技术十分重要。

随着微型计算机的迅猛发展，Intel 公司 86 家族的芯片 8086、80186、80286、80386、80486 以及现在已推出的 Pentium 已经成为微机中的主导处理器。80X86 是向上兼容的，功能强大，特别是 80386 以上处理器的芯片功能有了质的飞跃，例如，能寻址 4GB 字节的物理内存，具有 4 个特权级，多任务切换机制以及片内的内存管理单元等，这些功能都在保护方式下实现，可支持多用户、多任务操作系统。但如果仍沿用传统的汇编语言程序设计方法，采用实地址工作方式，则不能充分发挥 80286 以上处理器的强大功能。

为了使汇编语言程序设计教学适应当前计算机发展水平，我们编写了这本教材。本书在介绍 8086/8088 汇编语言的基础上，进一步较全面地介绍了 80286、80386 和 80486 的扩展汇编语言。全书共分八章，第一章至第四章全面介绍了 8086/8088 的基本结构、寻址方式、指令系统及编程方法；第五章主要介绍了输入输出和中断传送方法，特别还介绍了 Super VGA 编程方法、VESA 标准图形接口及真彩色技术。第六章介绍了磁盘文件操作技术，这几章都是学习汇编语言程序设计不可缺少的重要基础内容。第七章着重介绍了 80286、80386、80486 的基本结构、指令系统、任务切换以及保护方式下的编程方法。第八章介绍了 8087、80287 和 80387 协处理器结构、数据格式、指令系统及编程方法。每章后均附有习题。本书十分注重引入新知识，程序实例丰富，我们希望通过本书的学习，使读者能够提高在新型微机上的汇编语言程序设计能力。

本书适用范围较广，对于计算机应用专业的专科生或本科生，可以重点学习第一章至第六章，熟练掌握 8086/8088 汇编语言程序设计，而对后两章内容可只作一些了解。对于研究生则可重点学习第七章和第八章的内容，掌握 80286 以上处理器和协处理器的汇编语言程序设计，熟悉保护方式下的编程方法。本书除可以作为高等院校汇编语言程序设计教材外，还可以作为广大工程技术人员的参考书。

本书由洪志全和罗省贤合作编写，罗省贤编写第一、二、三、四和八章，洪志全编写第五、六和七章。全书由陈天与教授审校。

由于编者水平有限，书中可能会存在不少缺点与问题，恳请广大读者指正。

编　者

1996 年 10 月

目　录

第一章　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，汇编语言具有内存开销小，运算速度快的特点，在实时控制和实时处理领域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，因此在已有众多高级语言的今天，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。本章将主要介绍 INTEL 8086/8088 的基本结构。

Intel 8086(简称 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。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，并具有 20 条地址线，寻址能力达到 1M 字节。

1.1　8086/8088 的功能结构

中央处理机 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。8086/8088 CPU 从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。

BIU 完成 8086/8088 CPU 与存储器之间的信息传送，控制总线和 I/O 数据传送，完成逻辑地址与物理地址的转换，从存储器中取出指令，送至指令流队列中排队，并取出执行指令时所需的操作数，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。

1.2　8086/8088 的寄存器结构

8086/8088 CPU 共有 14 个 16 位寄存器，如图 1.3 所示，这些寄存器具有重要作用，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，一般可分为四类：数据寄存器、段寄存器、指针及变址寄存器和控制寄存器。

1.　数据寄存器

数据寄存器共有 4 个 16 位寄存器：AX、BX、CX 和 DX，通常用于暂存计算过程中的操作数、计算结果或其它信息。这 4 个数据寄存器既可按字(16 位)形式访问，又可按字节(8 位)形式访问，不仅有上述通用性，同时还有各自的专门用途。

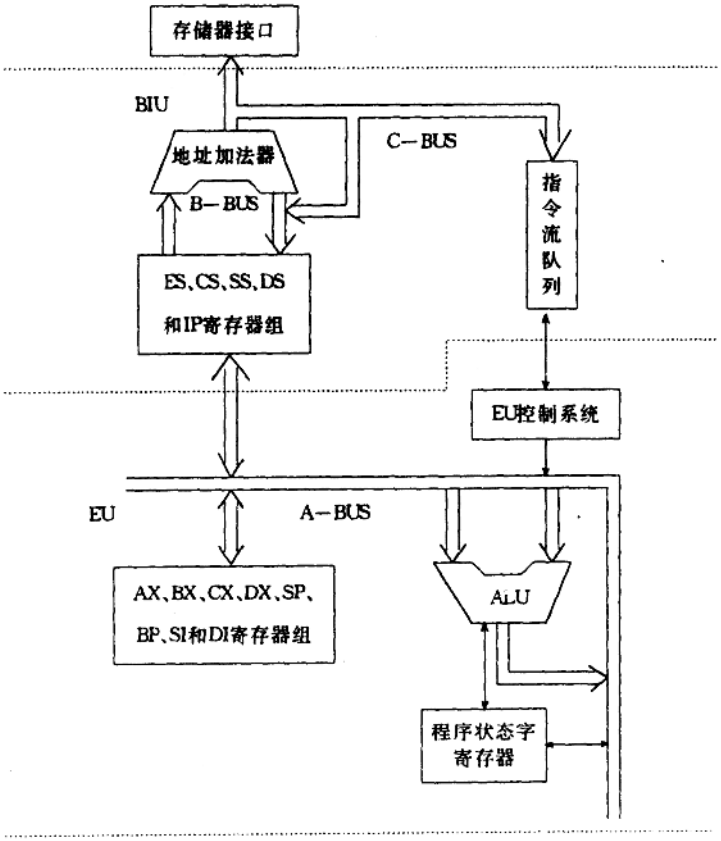


图1.1 8086/8088的功能结构

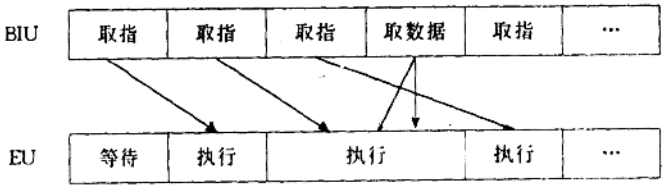


图1.2 8086/8088指令流水线结构

AX(Accumulator)——累加器,是算术运算的主要寄存器,此外还作为乘、除法运算及与外部设备传送信息的专用寄存器。

BX(Base) 基址寄存器,常用于存放内存储区的基址。

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。此外还用于存放某些I/O操作所需的I/O端口地址。

2. 段寄存器

段寄存器共有4个16位寄存器:CS、DS、SS和ES,专门用于存放段地址。

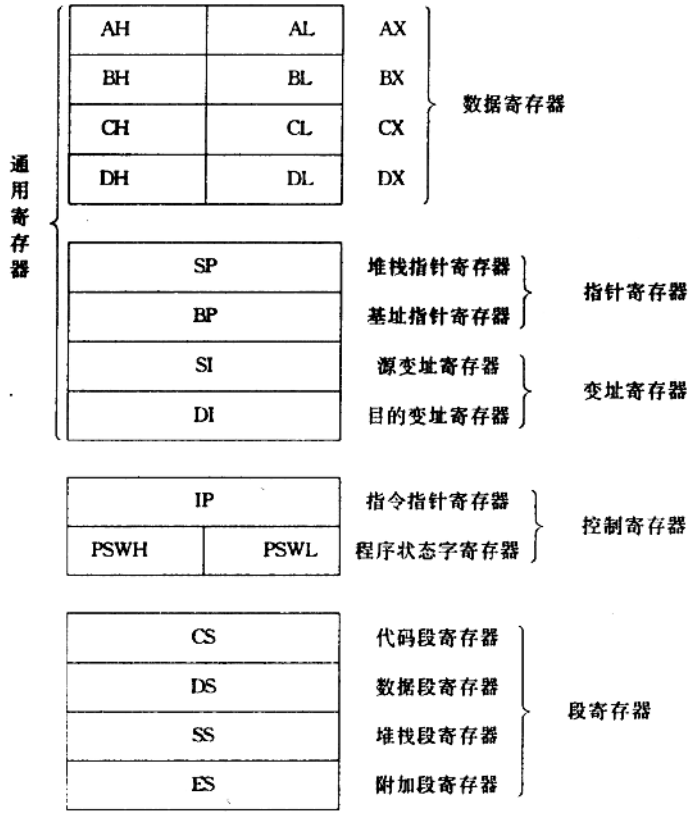


图1.3 8086/8088的寄存器结构

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，是在内存储器中开辟的一片特殊存储区。堆栈段是定义堆栈的存储区域。

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

这类寄存器共有4个16位寄存器：SP、BP、SI和DI。这几个寄存器主要用于访问存储器单元时提供16位偏移地址，其中SI、DI也可以用于暂存运算过程中的操作数。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，这一问题将在下节介绍。

SP(Stack Pointer)——堆栈指针寄存器，提供栈顶的偏移地址，与SS段寄存器联用，用于控制数据进栈和出栈。

BP(Base Pointer)——基址指针寄存器，常用于提供堆栈中某指定单元的偏移地址，与SS段寄存器联用，可访问堆栈中的任一个存储器单元。

SI(Source Index)——源变址寄存器，与DS段寄存器联用，可以访问数据段中任一个存储器单元。

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器单元。

SI、DI也常用于字串操作，并在字串操作中具有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：IP和PSW。

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，始终指向下一条要取的指令。IP与CS段寄存器联用，可以确定下一条要取指令的物理地址，因此IP是很重要的控制寄存器，用于控制程序的执行流程。

PSW(Program status word)——程序状态字寄存器，用于存放条件码标志和控制标志。PSW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。

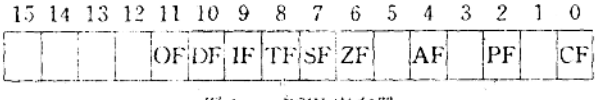


图1.4 PSW寄存器

PSW的6个条件码标志位用于记录反映程序运行结果的状态，一般由系统根据计算结果自动设置，这些标志常用作改变程序执行流程的控制条件，含意如下：

• 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，运算结果数中"1"的个数为偶数时，PF位置1，否则置0。PF可用于检查数据传送是否出错。

PSW的3个控制标志位可由程序使用专用指令置位或复位，以控制CPU的操作方式，含意如下：

• 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，有关中断原理将在第五章介绍。

·TF(Trap Flag)——陷阱标志。该标志可使CPU进入单步操作方式。TF=1,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,CPU正常工作,不产生陷阱异常。

PSW标志位的值还有一种符号表示方式。在DOS系统提供的汇编语言通用调试工具DEBUG中,可以检测PSW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表1—1所示的符号表示,有关DEBUG程序使用方法的介绍见附录二。

表1—1

标志位名称		符号	
		=1	=0
OF	溢出(是/否)	OV	NV
DF	方向(减/增)	DN	UP
IF	中断(开/关)	EI	DI
SF	符号(负/正)	NG	PL
ZF	零(是/否)	ZR	NZ
AF	辅助进位(是/否)	AC	NA
PF	奇偶(偶/奇)	PE	PO
CF	进位(是/否)	CY	NC

1.3 堆栈与存储器结构

1.3.1 堆栈

前节已提到堆栈是一种数据结构,实际上是在内存中开辟的一个一端活动,而另一端固定的数据存储区,数据的存取原则是先进后出(也可称为后进先出)。

堆栈的固定端称为栈底,活动端称为栈顶,数据的存取都在栈顶进行,堆栈指针寄存器SP始终存放栈顶的地址,即指向栈顶,如图1.5所示。

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,以及保护子程序、中断服务程序调用返回地址等有关信息。

8086/8088指令系统有两条专门指令(PUSH、POP)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个字节)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

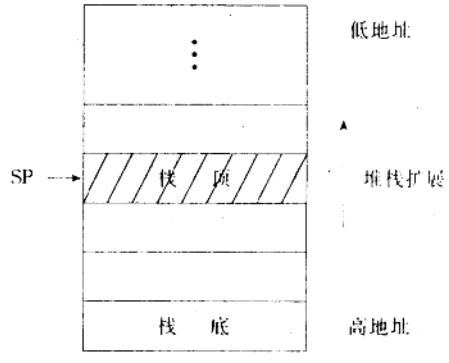


图1.5 堆栈的结构

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。

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了一个存储器地址，地址从0开始编址，顺序增1，用无符号二进制数表示，但书写时常用无符号十六进制数(数后跟H表示十六进制数)表示。64K存储器单元编址方法如图1.6所示。

一个字数据存于存储器中相邻两个字节单元，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时，应先存低字节后存高字节。

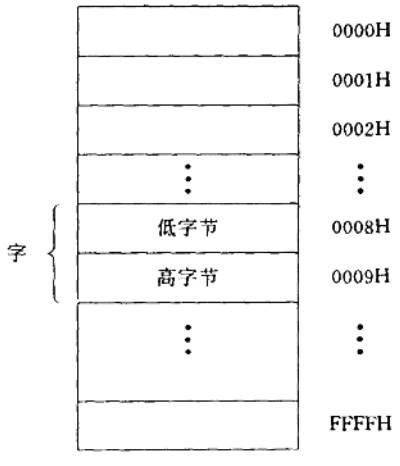


图1.6 64K存储器单元编址

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，前节已提到物理地址要由段地址和偏移地址合成。为什么要分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，16位寄存器只能表示64K的地址范围，但8086/8088有20条地址线，寻址可达1M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64K，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(可简称为段地址)，必须取为任一小段首地址。小段的划分方法是，从存储器的0地址开始，每16个字节为一小段。下面列出按20位地址对1M存储器的编址：

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

分析上面的地址，可以看出小段首址的特点。小段首地址最后一位都为0，在1M字节的寻址范围内用作段地址时最后一位0可以不用保存，并且在1M字节的寻址范围内共有64K个小段首址，从而段地址也可用16位表示。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唯一的20位地址，称为该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：

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

例如：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，而偏移地址则可由指令直接提供或由 16 位地址寄存器等方式提供。

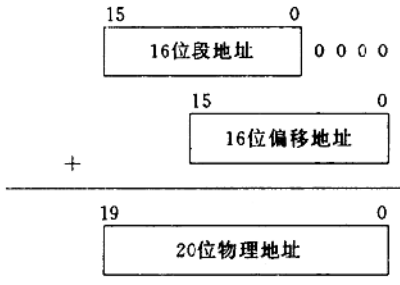


图1.7 物理地址的形成

形成各段内存储器单元物理地址时所用的寄存器如图 1.8 所示。图中用于数据段或附加段寻址的 16 位寄存器可以是基址或变址寄存器，如果指定了段跨跃前缀，也可以是 BP 寄存器。

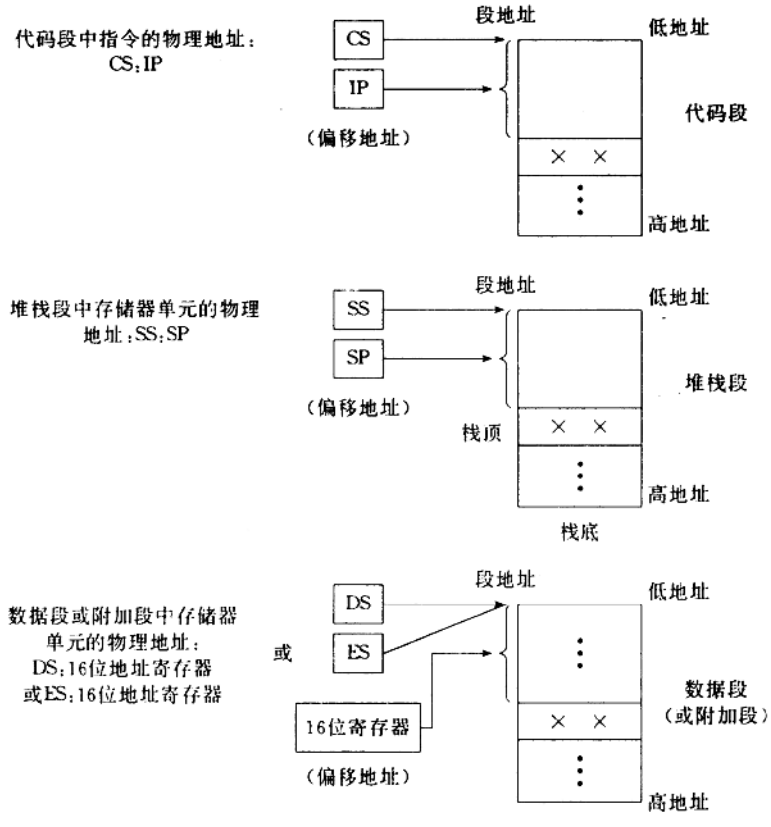


图1.8 形成各段内存储器单元物理地址所用的寄存器

在存储器中，代码段、数据段、堆栈段和附加段各段可以独立占用 64K 存储区，各段之间可间隔开或邻接，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区分配。各段存储区分配的两种方式如图

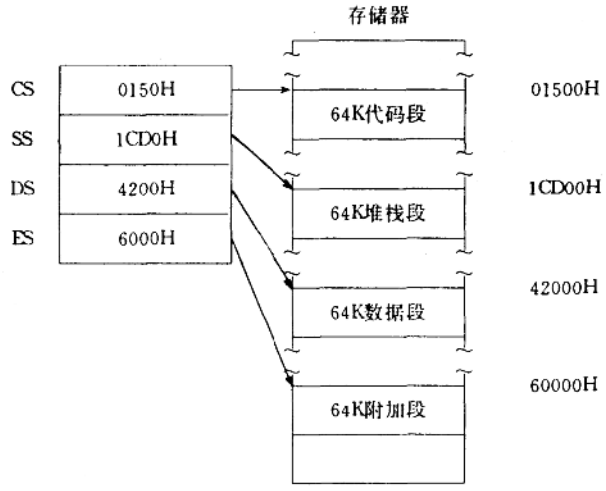


图1.9 段分配方式之一，各段独立，大小相同

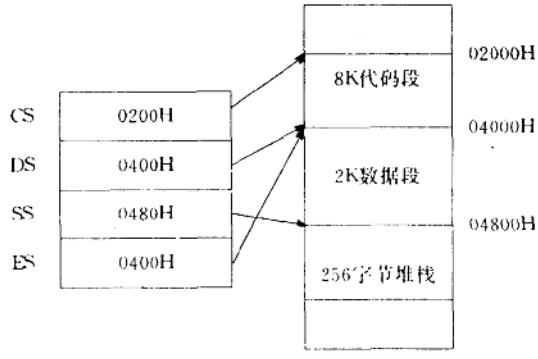


图1.10 段分配形式之二，各段大小不同，数据段和附加段的存储区重叠

1.9 和图 1.10 所示。

存储器分段法便于处理要求代码区、堆栈区和数据区处于内存不同区域时的情况，这时只需将各段寄存器置为相应地址即可。分段法也便于程序再定位的要求，只要正确设置 CS 的值，程序可装入不同存储区运行。

习 题 一

1.1 为什么要对存储器进行分段管理？