HUIBIAN YUYAN CHENGXU SHEJI

汇编语言程序设计

主编 鹿玲杰
主审 马瑞民

哈尔滨工程大学出版社

汇编语言程序设计

主编 鹿玲杰
主审 马瑞民
编委 鹿玲杰 唐国维 王东

哈尔滨工程大学出版社

内 容 简 介

本书介绍了 Intel 80386 CPU 指令系统及其程序设计方法。全书共分五章。第一章介绍了汇编语言的基本概念以及 80X86 系列 CPU 的体系结构。第二章详细介绍了 80386 的指令系统，并给出了各种指令在程序中使用的例子。第三章介绍了汇编语言源程序的基本结构和常用的伪指令以及宏汇编。第四章主要介绍了汇编程序设计的基本方法和技巧，并通过各种实例充分给予解释。第五章介绍 DOS 和 BIOS 系统功能调用的使用方法和有关的程序设计。

本书注重基本概念、方法和技巧，内容丰富，实用性强，可作为高等院校计算机专业及有关专业的教材，或作为技术人员的培训教材，也可作为广大从事微型计算机科研、生产、教学和应用开发的科技人员的自学参考书。

汇编语言程序设计
主　　编 鹿玲杰
责任编辑 张植扑

哈尔滨工程大学出版社出版发行
（哈尔滨市文庙街 11 号楼　邮编 150001）
新 华 书 店 经 销
大庆石油学院印刷厂印刷

开本 787mm×1092mm　1/16　印张 15.875 字数 373 千字
1997 年 12 月第 1 版　　1997 年 12 月第 1 次印刷
印数：1～500 册
ISBN 7－81007－828－3
TP・70　　定价：18.00 元

前　言

近年来，广为流行与普及的微型计算机，多数是以 80386/80486 甚至是 80586 为 CPU 的。在这样的 CPU 上，虽然可以用高级语言来编制任意的程序，但是为了更充分地利用计算机的硬件特性，更多的直接控制计算机的基本资源，编制占用内存少、执行速度快、效率高的程序，就必须使用汇编语言。

汇编语言程序设计是计算机专业一门重要的专业课程。鉴于目前介绍 80386 汇编语言程序设计方面的书籍较少，致使许多程序设计至今仍停留在 8086/8088 汇编语言的基础上。为此，我们组织编写了此书，目的是帮助读者学习和掌握 80386 汇编语言程序设计，为开发高质量的实用程序奠定基础。

本书注重实用性，力图做到通俗易懂、概念清楚。通过各种程序设计实例向读者介绍了汇编语言程序设计的思想、方法和技巧，并结合编者多年积累的教学经验，对难于理解、易出差错、容易混淆的内容和问题给予了明确的说明和解释。书中的所有实例都是精心挑选并经过上机验证的，读者可通过阅读它们来加深自己对一些概念的理解。

全书共分五章。第一章介绍了汇编语言的基本概念以及 80X86 系列 CPU 的体系结构。第二章详细介绍了 80386/80286 的指令系统，并给出了各种指令在程序中使用的例子。第三章介绍了汇编语言源程序的基本结构和常用的伪指令以及宏汇编。第四章主要介绍了汇编语言程序设计的基本方法和技巧，并通过各种实例充分给予解释。第五章介绍了 DOS 和 BIOS 系统功能调用的使用方法和有关的程序设计。

参加本书编写的人员，均是多年从事《汇编语言程序设计》课程的教师。本书第一、二章由唐国维编写，第三、四章由鹿玲杰编写，第五章由王东编写，全书由鹿玲杰统稿，马瑞民主审。由于计算机技术发展迅速，编者水平有限，经验不足，书中的错误在所难免，敬请读者指正，以利改进。

编　者

1997 年 8 月

目　　录

1 概 述

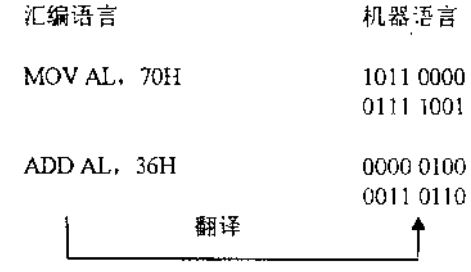
本章首先介绍了汇编语言的基本概念和微型计算机的基本结构，使读者对汇编语言有一个初步的认识，并概括地介绍了 8086/8088、80286、80386 微处理器结构，以及存储器管理等问题，这些内容是进行汇编语言程序设计的基础。

1.1 汇编语言的基本概念

1.1.1 什么是汇编语言

在计算机发展的初期，人们是用机器指令码(二进制编码)来编写程序的，这就称为机器语言。但是机器语言无明显的特征，不好理解和记忆，也不便于学习，在编制程序时容易出错。所以人们就用助记符代替操作码，用符号来代替地址，这就是汇编语言阶段。因此汇编语言指令是指用助记符表示相应机器的指令的操作码和操作数，按照一定的格式书写的一种面向机器的指令形式，又称符号指令。而由汇编指令按照一定的语法规则书写的程序则称汇编语言源程序，又称汇编源程序。

汇编语言使指令易于理解和记忆，便于交流，大大前进了一步。但是，机器还是只认得机器码，所以用汇编语言写的源程序在机器中还必须经过翻译，变成用机器码表示的程序(称为目标程序，Object Program)，机器才能识别和执行。具有这样功能的程序称为汇编程序(Assembler)。这一过程说明如下：



汇编语言程序(Assembler)

在汇编源程序中，除了上述汇编指令外，还有必要的数据及其结构的描述，以及源程序向汇编程序(即翻译程序)提供的一些必要的信息。这些是通过称作伪汇编指令即伪指令实现的。伪指令是相对汇编指令而言的，一般说它并不像汇编指令那样一对一的被汇编(翻译)成代码指令。关于一个完整的汇编程序的构成，以及对它的深入理解，这正是本课程的宗旨，因此是不能够在这里一下子说清楚的。一个简单的汇编语言程序如下。

例1.1 输出字符"3"的汇编语言程序。

```
MOV DL, '3'
MOV AH, 2
INT 21H
INT 20H
```

1.1.2 为什么要用汇编语言

汇编语言是一种面向机器的程序设计语言，是一种低级语言。它直接利用机器提供的指令系统编写程序，与具体的计算机硬件有着密切的关系，用它们编写出的程序只适用于某一系列的计算机，可移植性差。但因汇编语言指令与机器语言指令一一对应，所以汇编语言可直接利用机器硬件系统的许多特性，如寄存器、标志位以及一些特殊指令等，执行速度快，占用内存少。也就是说，汇编语言是计算机提供给用户的最快而又最有效的语言，是能够利用计算机所有硬件特性并能直接控制硬件的唯一语言，因而在程序的空间和时间要求很高的场合，汇编语言是必不可少的。至于对很多需要直接控制硬件的应用场合，则更是非用汇编语言不可。

而高级语言(如 Pascal、Fortran、Basic 等)是面向问题的，它与机器的硬件无关，可以在各种不同的计算机上运行，因此可移植性好。但是用高级语言编写的程序是不能直接执行的，需要由编译程序或解释程序将它们翻译成对应的目标程序，机器才能接受。由于高级语言指令与机器语言指令不是一一对应的，往往一条高级语言指令要对应着多条机器语言指令，因此这个翻译过程要比将汇编源程序翻译成目标程序花费的时间要长的多，产生的目标程序也较冗长，占用存储空间大，执行速度也较汇编语言慢。虽然采用高级语言编写程序可以节省软件的开发时间，但它不允许程序员直接利用寄存器、标志位等这些计算机硬件特性，因而影响了许多程序设计技巧的发挥。用汇编语言编写程序可以充分发挥机器硬件的特性，提高编程质量和运行速度。但由于汇编语言要求对计算机的组成部分进行操作，因此学习汇编语言程序设计除了要掌握机器的指令系统外，还要熟悉机器的内部结构，特别是中央处理器(CPU)和存储器的结构，以及与编程有关的其它部分(芯片)的结构，如中断系统、视频显示、键盘接收、定时功能、通信等。

是否采用汇编语言编写程序，要视具体的应用场合而定。一般来说，某些对时间和存储器容量要求较高的程序用汇编语言来书写，如系统软件、实时控制系统、智能化仪器仪表软件等。

用汇编语言编程和调试的时间都比较长，程序设计的技巧性和灵活性也比较高，但作为一个软件工作者，特别是系统软件和实时控制的软件设计者，应能熟练的使用汇编语言进行编程和调试。

1.2 微型计算机的基本结构

微型计算机与传统的大、中、小型计算机一样，是由五个基本部分组成的，它们是：输入设备、输出设备、运算器、控制器和存储器。

1. 输入和输出设备

输入和输出设备是实现人与计算机之间信息交换所必须的部件。输入设备负责将用

户要处理的数据及程序等信息输入到计算机存储器中。常用的输入设备有终端键盘、光电扫描仪等。输出设备负责将处理后的结果或其它信息进行输出显示。常用的输出设备有屏幕终端、打印机、绘图仪等。

2. 存储器

存储器是用来存储计算机要处理的程序和数据，并将运算的中间结果以及处理后的结果存储起来。在计算机系统中，存储器能存储的全部信息称为存储容量。从使用者的观点来看，存储器的容量越大越好，存取信息的时间越短越好。80286 CPU 能提供 24 位的地址，所以它可配置容量最多达 16 兆(2^{24})字节的存储器。80386 CPU 提供了 32 位的地址，它可配置容量最大为 4 千兆(2^{32})字节的存储器。

存储器分为随机存储器(RAM)和只读存储器(ROM)两种。RAM 既可以写入数据也可以从中读出数据，但断电后 RAM 不保留原有的信息。RAM 主要用来存放用户数据和程序。ROM 每个单元的信息是固化了的，用户可从中读出信息，但一般不能改变其信息。这种存储器中的信息在断电后不会丢失，只要再通电就又可读出原来的信息。ROM 主要用来存放计算机自身管理的系统程序，它分为两种：PROM 和 EPROM。前者是可编程的只读存储器，后者是可擦除的可编程只读存储器，但这种擦除必须使用特殊工具，在特殊情况下才能将原来的信息更改和擦除。

3. 运算器

运算器简称为 ALU，它是按指令对数据进行算术运算和逻辑运算的处理部件。ALU 通常由累加器和各种寄存器组成，用以暂存数据和完成各种最基本的算术运算和逻辑运算操作，同时也可在控制器的控制下，根据指令要求将处理结果送到存储器或输入/输出设备。

4. 控制器

控制器通常由指令寄存器、指令计数器、译码器和各种控制线路组成。控制器完成所有的输入/输出操作，以及对运算器的控制。它从存储器中读取程序指令，经过译码向计算机各部分发出执行微操作的控制信号，使指令得以执行。因此，控制器可实现对输入/输出设备、存储器和运算器的控制和管理，它将原始数据从输入/输出设备或存储器中取出后送到 ALU，又将 ALU 加工处理的结果送到存储器或输入/输出设备中去。

1.3 8086/8088 微处理器

8088 是一种准 16 位微处理器，它是在 Intel 公司的 8 位微处理器 8080/8085 的基础上发展起来的一种准 16 位微处理器，它的内部结构是 16 位的，但数据总线是 8 条，它能处理 16 位数据，也能处理 8 位数据。它能执行整套 8080/8085 的指令，所以它在汇编语言上与 8080/8085 是兼容的，又增加了许多 16 位操作指令。其主要的性能是：系统时钟是 4.77MHz，基本指令 99 条；机内字长 16 位，数据线则为 8 根；地址线有 20 根，提供了 1M 字节的寻址能力；平均运算速度为 0.65MIPS(百万条指令/秒)。

1.3.1 8086/8088 CPU的组成及功能

CPU 的任务是执行存放在存储器里的指令序列。它由运算器和控制器两部分组成，可简单地用图 1-1 来表示。

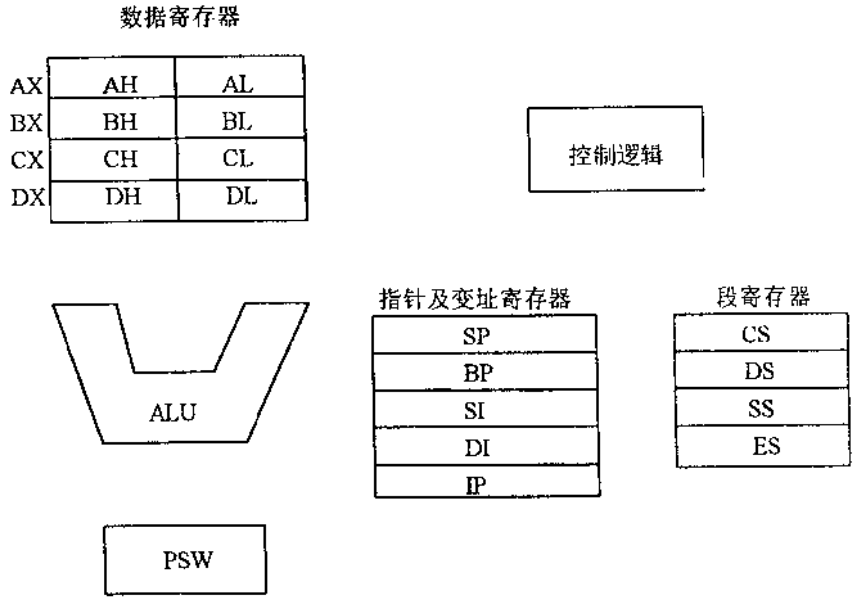


图1-1 8086/8088 CPU组成

从图中可以看出，CPU 由三部分组成：

1. 算术逻辑部件 ALU (Arithmetic Logic Unit)用来进行算术和逻辑运算。

2. 控制逻辑负责对全机的控制工作，包括从存储器取出指令、对指令进行译码分析、从存储器取得操作数、发出执行指令的所有命令、把结果存入存储器，以及对总线及 I/O 传送的控制等。

3. 工作寄存器在计算机中起着重要的作用，每一个寄存器相当于运算器中的一个存储单元，但它的存取速度比存储器要快的多。它是用来存放计算过程中所需要的或所得到的各种信息，包括操作数地址、操作数及运算的中间结果等。

8088 CPU 从功能上来说分成两大部分：总线接口单元 BIU (Bus Interface Unit)和执行单元 EU (Execution Unit)。

BIU 负责与存储器接口，即 8088 CPU 与存储器之间的信息传送，都是由 BIU 进行的。具体的说，即 BIU 负责从内存的指定部分取出指令，送至指令流队列中排队；在执行指令时所需的操作数，也由 BIU 从内存的指定区域取出，传送给 EU 部分去执行。

EU 部分负责指令的执行，因为取指部分与执行部分是分开的，所以在一条指令的执行过程中，就可以取出下一条(或多条)指令在指令流队列中排队。在一条指令执行完以后可以立即执行下一条指令，减少了 CPU 为取指令而等待的时间，提高了 CPU 的利用率和整个运行速度。

如前所述，在 8080/8085 以及标准的 8 位微处理器中，程序的执行是由取指和执行指

令的循环来完成的，即执行的顺序为取第一条指令，执行第一条指令；取第二条指令，执行第二条指令……直至取最后一条指令，执行最后一条指令。这样，在每一条指令执行完以后，CPU 必须等待，到下一条指令取出来以后才能执行。所以，它的工作顺序如图 1-2 所示。

但在 8088 中，因为 BIU 和 EU 分开，所以取指和执行可以重叠。它的执行顺序如图 1-3 所示。这就大大减少了等待取指所需的时间，提高了 CPU 的利用率。这样一方面可以提高整个执行速度，另一方面又降低了对与之相配的存储器的存取速度的要求。这种重叠的操作技术，过去只在大型机中才使用。

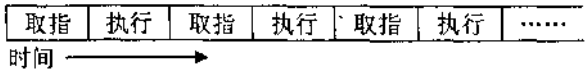


图1-2 一般8位机的执行方式

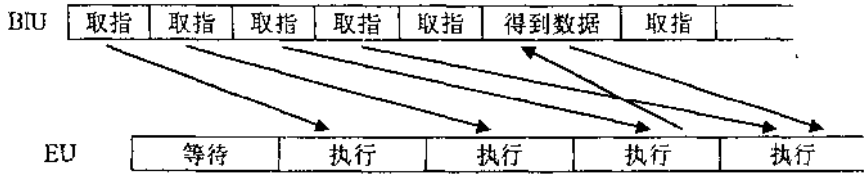


图1-3 8088执行方式

1.3.2 8086/8088 CPU 的寄存器组

1. 数据寄存器

包括 AX、BX、CX、DX 四个通用寄存器，用来暂时存放计算过程中所用到的操作数、结果或其它信息。它们都可以以字(16 位)的形式访问，或者也可以以字节(8 位)的形式访问，例如对 AX 可以分别访问高位字节 AH 或低位字节 AL。这四个寄存器都是通用寄存器，但它们又可以用于各自的专用目的。

AX(Accumulator)作为累加器用，所以它是算术运算的主要寄存器。另外，所有的 I/O 指令都使用这一寄存器与外部设备传送信息。

BX(Base)可以作为通用寄存器使用，此外在计算存储器地址时，它经常用作基址寄存器。

CX(Count)可以作为通用寄存器使用，此外在循环(LOOP)和串处理指令中用作隐含的计数器。

DX(Data)可以作为通用寄存器使用。一般在作双字长运算时，把 DX 和 AX 组合在一起存放一个双字长数，DX 用来存放高位字。此外，对某些 I/O 操作，DX 可用来存放 I/O 的端口地址。

2. 指针及变址寄存器

包括 SP、BP、SI、DI 四个 16 位寄存器。它们可以像数据寄存器一样在运算过程中存放操作数，但它们只能以字(16 位)为单位使用。此外，它们更经常的用途是在段内寻址时

提供偏移地址。其中 SP(Stack Pointer)称为堆栈指针寄存器,BP(Base Pointer)称为基址指针寄存器,它们都可以与 SS 寄存器联用来确定堆栈段中某一存储单元的地址。SP 用来指示栈顶的偏移地址,BP 可以作为栈区中的一个基地址,以便访问堆栈中的其它信息。SI(Source Index)源变址寄存器和 DI(Destination Index)目的变址寄存器一般与 DS 联用,用来确定数据段中某一存储单元的地址。这两个变址寄存器有自动增量和自动减量的功能,所以用于变址是很方便的。在串处理指令中,SI 和 DI 作为隐含的源变址和目的变址寄存器,此时 SI 和 DS 联用,DI 和 ES 联用,分别达到在数据段和附加段中寻址的目的。

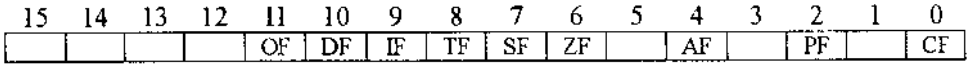
3. 段寄存器

包括代码段寄存器 CS(Code Segment)、数据段寄存器 DS(Data Segment)、堆栈段寄存器 SS(Stack Segment)和附加段寄存器 ES(Extra Segment)四个段寄存器。每个段寄存器可以确定一个段的起始地址,而这些段则各有各的用途。代码段存放当前正在运行的程序。数据段存放当前运行程序的数据,如果程序中使用了串处理指令,则其源操作数也存放在数据段中。堆栈段定义了堆栈的所在区域,堆栈是一种数据结构,它开辟了一个比较特殊的存储区,并以后进先出的方式来访问这一区域。附加段是附加的数据段,它是一个辅助的数据区,也是串处理指令的目的操作数存放区。

4. 控制寄存器

包括 IP 和 PSW 两个 16 位寄存器。IP(Instruction Pointer)为指令指针寄存器,它用来存放代码段中的偏移地址。在程序运行的过程中,它始终指向下一条指令的首地址,它与 CS 寄存器联用确定下一条指令的物理地址。当这一地址送到存储器后,控制器可以取得下一条要执行的指令,而控制器一旦取得这条指令就马上修改 IP 的内容,使它指向下一条指令的首地址。可见,计算机就是用 IP 寄存器来控制指令序列的执行流程的,因此 IP 寄存器是计算机中很重要的一个寄存器。

PSW(Program Status Word)为程序状态字寄存器,这是一个 16 位寄存器,由条件码标志(Flag)和控制标志构成,如下所示:



其中条件码标志用来记录程序中运行结果的状态信息。因为这些状态信息往往是后续条件转移指令的转移控制条件,所以称为条件码。它包括以下 6 位:

OF(Overflow Flag)溢出标志。在运算过程中,如操作数超出了机器能表示的范围则称为溢出,此时 OF 为 1,否则置 0。

SF(Sign Flag)符号标志,记录运算结果的符号,结果为负时置 1,否则置 0。

ZF(Zero Flag)零标志,运算结果为 0 时 ZF 位置 1,否则置 0。

CF(Carry Flag)进位标志,记录运算时从最高有效位产生的进位(借位)值。例如执行加法指令时,最高有效位有进位时置 1,否则置 0。

AF(Auxiliary carry Flag)辅助进位标志,记录运算时第 3 位(半个字节)产生的进位(借位)值。例如执行加法指令时第 3 位有进位时置 1,否则置 0。

PF(Parity Flag)奇偶标志，用来为机器中传送信息时可能产生的代码出错情况提供检验条件。当结果操作数中 1 的个数为偶数时置 1，否则置 0。

控制标志有三个：

DF(Direction Flag)方向标志，在串处理指令中控制处理信息的方向用。当 DF 位为 1 时，每次操作后使变址寄存器 SI 和 DI 减量，这样就使串处理从高地址向低地址方向处理。当 DF 为 0 时，则使 SI 和 DI 增量，使串处理从低地址向高地址方向处理。

IF(Interrupt Flag)中断标志，当 IF 为 1 时，允许中断，否则关闭中断。

TF(Trap Flag)陷井标志，用于单步方式操作。当 TF 位为 1 时，每条指令执行完后产生陷井，由系统控制计算机；当 TF 位为 0 时，CPU 正常工作不产生陷井。

以上就是 PSW 中各种状态和控制信息的含义。其中控制信息是由系统程序或用户程序根据需要用指令来设置的，而状态信息是由中央处理机根据计算的结果自动设置的。为方便起见，机器提供了设置状态信息的指令，必要时，程序员可以使用这些指令来建立状态信息。

在调试程序 DEBUG 中提供了测试标志位的手段，它用符号表示标志位的值。表 1-1 说明每种标志位(因 DEBUG 不提供 TF 的符号，所以表格中未包括 TF 位在内)的符号表示。

表1-1 PSW中标志位的符号表示

标 志 名	标志为1	标志为0
OF 溢出(是 / 否)	OV	NV
DF 方向(减量 / 增量)	DN	UP
IF 中断(允许 / 关闭)	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

1.3.3 存储器的分段管理

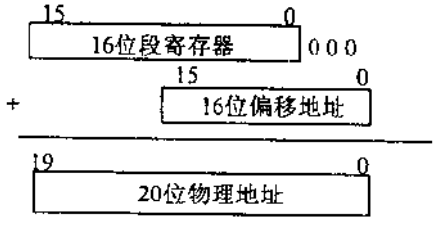
IBM PC 机的字长为 16 位，可表示的地址范围为 0000H～0FFFFH，即 64K 字节。而 8086/8088 有 20 条地址线，直接寻址能力为 $2^{20}=1M$ 字节，所以在一个 8086/8088 组成的系统中，可以有多达 1M 字节的存储器。这 1M 字节逻辑上可以组成一个线性矩阵，地址从 00000H～0FFFFFH。给定一个 20 位的地址，就可以从这 1M 字节中取出所需要的指令或操作数。那么在 16 位字长的机器里，用什么方法来提供 20 位地址，以寻址 1M 内存空间呢？

由于 8086/8088 内部的 ALU 进行的是 16 位运算，有关地址寄存器如 SP、IP，以及 BP、SI、DI 等也都是 16 位的，因而 8086/8088 对地址的运算也只能是 16 位。这就是说，对于 8086/8088 来说，寻找操作数的范围最多只能是 64K 字节。为此，提出了用存储器分段的办法来解决寻址 1M 内存空间的问题，即把 1M 的内存空间划分成若干不超过 64K 的

区段，简称为段(这样，在编制程序时，要把程序划分成不同的段，如数据段、代码段等)，每段最多为64K，从而段内地址可以用16位表示。而每一个段都有一个起始地址，称为段基地址或段地址。对段地址要有所限制，段地址不能起始于任意地址，必须从称作小段或节(Paragraph)的首地址开始。而小段或节是指从0地址开始的每16个字节，如下所列的每一行就是一个小段：

```
00000,00001,00002,...,0000E,0000F;
00010,00011,00012,...,0001E,0001F;
00020,00021,00022,...,0002E,0002F;
...
FFFF0,FFFF1,FFFF2,…,FFFFE,FFFFF;
```

其中第一列就是每个小段的首地址。其特征是在16进制表示的地址中，最低位为0，即20位地址的低4位为0(在1M字节的地址空间里，共有64K个这样的小段首地址，它们均可作为段起始地址)。这样，就可以规定段地址只取段地址的高16位值，其低4位的0值为隐含，从而可用16位的寄存器或存储单元来保存段地址。为此，20位的物理地址便可由16位的段地址和16位的段内偏移地址(指在段内相对于段起始地址的偏移值)组成，即用两个16位组来表示一个20位的物理地址，从而解决了寻址1M内存的问题。物理地址的计算方法可以表示如下：



也就是说，把段地址左移4位，再加上偏移地址就形成物理地址，即

16D×段地址＋偏移地址＝物理地址

显然，每个存储单元只有唯一的物理地址，但它却可由不同的段地址和偏移地址组成。

值得一提的是，存储器段的划分不是固定的，即其大小可以是64K范围内的任意长度，其位置在整个可用存储空间中由系统按某种规则随机分配(通常用户不必关心)。

另外，存储器分段是对程序而言的，即用户在编程序时，要把程序的不同部分分别安排在不同的段当中(用伪指令实现)。如代码部分放在代码段(CS)，数据部分放在数据段(DS)，此外还可能用到堆栈段(SS)和附加段(ES)。这样，当一个程序编好之后，经汇编、连接形成可执行文件之后，便可调入内存，由系统为每一个段分配一块实际的物理空间，进而可以执行。比如，一个程序包括8K的代码段、2K的数据段和256字节的堆栈段，那么，一种可能的分配方案如图1-4所示。其中代码段只需8K，地址范围是02000H～11FFFH，所以代码段结束后的第一个存储单元地址就是一个小段的首地址，它可直接作为数据段的起始地址。同样，堆栈段也可紧接在数据段之后。

通过存储器分段，虽然限定每个程序段的空间不能超过64K，但一个程序可包括一个

以上的代码段、数据段、堆栈段和附加段，比如说一个程序可能包括两个代码段，三个数据段和一个堆栈段。可见程序的空间是可以任意长的，并没有因段区的划分而受到限制。

对于较短的程序，四个段可能均在 64K 的范围之内，而程序运行时所需要的信息也都在本程序所规定的段之内，那么只需要在程序的开头一次设定各段寄存器的值就可以了。如果程序较长，如代码部分或数据部分超过了 64K，在编程序时就要考虑把它们分成若干个都小于 64K 的段，而在程序中要多次设定各段寄存器的值，从而确保在程序运行过程中动态地修改段寄存器的内容，从而确保所获得的信息的正确性。此外，若在程序中还要访问除本身四个段以外的其它段区的信息，那么在程序中也必须动态的修改段寄存器的内容。一个程序也并不一定要包括所有的四个段，即除代码段外其它段都不是必须的，所以极端情况是一个程序仅由一个代码段构成。

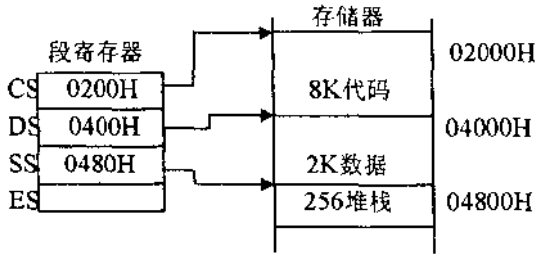


图1-4 存储器的段分配

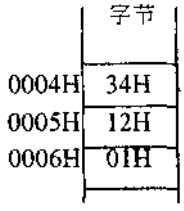


图1-5 存储单元的地址和内容

存储器分段的方法给程序设计带来一定的麻烦，但却很好的解决了寻址 1M 内存的问题，扩大了存储空间，而且对程序的再定位也是方便的。

顺便指出，在存储器里是以字节为单位存储信息的，即每一个字节单元占有一个地址。但机器字长是 16 位的，大部分数据都是以字为单位表示的，所以一个字存入存储器要占有相继的两个字节，低位字节存入低地址，高位字节存入高地址。这样两个字节单元就构成了一个字单元。字单元的地址采用它的低地址表示，如图 1-5 中 4 号字单元的内容为 1234H，表示为(0004H)＝1234H，而对于 4 号字节单元来讲，其内容为 34H，表示为(0004H)＝34H。同样，5 号字单元为(0005H)＝0112H，5 号字节单元为(0005H)＝01H。可见，同一地址既可看作字节单元的地址，也可以看作字单元的地址，这要依据使用情况确定。可以看出，字单元的地址可以是偶数，也可以是奇数。但是，在机器里，访问存储器(要求取数或存数)都是以字为单位进行的，也就是说，机器是以偶地址访问存储器的。这样，对于奇地址的字单元，要取一个字需要访问两次存储器，当然这样做要花费较多的时间。

1.4 80286 微处理器

80286 CPU 是 Intel 公司于 1984 年推出的新一代高性能的微处理器，它不论是在速度上还是在功能上都比 8088 CPU 更加强大，是 IBM AT 计算机的 CPU 芯片。80286 有

两种工作方式，一种称为实地址方式，一种称为保护虚地址方式(或保护方式)。在实地址方式下，80286 相当于一个增强了的 8088，它支持 8088 的所有指令。实地址方式是 AT 机的默认方式，即计算机开机以后的方式，除非一个程序有意地将方式改变为保护方式，否则将一直保持在这一方式下。

在保护方式下，80286 支持任何在实地址方式下的操作，并为数据保护和内存管理提供了许多功能，这就为多用户和多任务提供了硬件支持。80286 可以管理两种内存，一种是物理地址空间，另一种是虚地址空间，物理地址空间是 80286 当时使用的内存，而虚地址空间是 80286 可以使用到的内存空间。两个地址空间的大小不一样，物理空间可用到 16MB，而虚地址空间则可用到 1GB(2^{30}字节)。所谓虚地址空间，是指一个程序需要访问的内存部分不在物理空间中，此时操作系统可以用 80286 的内存管理功能，将所需虚存部分转换为物理地址空间。

1.4.1 80286 CPU 的内部结构

80286 有四个独立的处理部件，分别是执行部件 EU (Execution Unit)、总线部件 BU (Bus Unit)、指令部件 IU (Instruction Unit)和地址部件 AU (Address Unit)。80286 采用流水线作业形式，使各个部件能同时工作。

1. 总线部件 BU

总线部件由地址锁存器、驱动器、处理机扩展接口、总线控制、数据收发、预取和 6 字节的预取队列组成。总线部件把 CPU 接到外部系统总线，是 CPU 与系统之间的一个高速接口。控制地址、数据和控制信号从 CPU 输出或向 CPU 输入，要在取代码、数据等期间内有效的满足 CPU 对外部总线的传送要求，以最高的速度传送数据，即在两个处理机时钟周期内传送一个字节，实现零等待状态，完成对存储器的读写。总线部件还必须预取指令，即在当前指令执行之前把它从存储器内取出。总线部件中的预取部件从存储器中取出可达 6 个字节的指令，并把它们暂存在预取指令队列中，这个队列是预取部件和指令译码器之间的一个缓冲。指令译码器对指令进行准备和译码，然后送到执行部件去执行。

2. 指令部件 IU

这个部件用来对指令译码，并作好供执行部件执行的准备。当指令部件把来自 6 字节预取队列的指令译码后，就把它们放到已译好码的指令队列中准备执行。译好码的指令包含执行部件 EU 执行时需要的所有指领域(不需要再进一步译码)。指令部件的引入进一步改善了流水线操作，改变了 8086 要由执行部件译码的局面。指令部件以每个时钟周期一个字节的速率接收数据，它的译码部件连续译码，使得已经译好码的队列总是有几条译好码的指令字节等待执行，与此同时执行部件 EU 执行的总是事先译好码的指令，使得译码部件和执行部件并行操作，大大提高了 80286 的速度。

3. 执行部件 EU

执行部件由寄存器、控制部件、算术及逻辑运算部件 ALU 和微代码只读存储器构成，它负责执行指令。为此，它使用自己的资源，同时也与执行指令所必须的其它逻辑部件交换控制信息和时序信息。微代码只读存储器 ROM 规定了指令执行的内部微指令序列，指令在内部微指令序列的控制下执行。当一条指令的微指令序列快要完成时，ROM 就发出信号，让执行部件从指令队列里再取出下一个 ROM 地址。这项技术的采用，就使得执

行部件总是处于忙碌状态。

4. 地址部件 AU

地址部件由偏移量加法器、段界限检查器、段基地址、段大小和物理地址加法器等部件构成。它实施存储器管理及保护功能，计算出操作数的物理地址。同时检查保护权限，在保护方式下，地址部件提供完全的存储管理、保护和虚拟存储支持。为此，地址部件在存储器中建立操作系统控制表，以描述机器的全部存储器，然后由硬件如实的执行表中的信息。地址部件在检查访问权的同时完成地址转换，在这个部件内有一个高速缓冲寄存器，高速缓冲寄存器内保存段的基地址、段界限和当前选中的正在执行任务所用的全部虚拟存储器的访问权。为使从存储器中读出的信息减至最小，高速缓冲寄存器允许地址部件在一个时钟周期内完成它的功能。

四个内部部件并行操作，这使 80286 能够支持虚拟存储管理，并且对整个存储器提供保护而又不降低系统的操作速度。

1.4.2 80286 CPU 的寄存器及功能

1. 与 8088 相同的寄存器

80286 CPU 的寄存器包含了 8088 CPU 所有的寄存器，如通用寄存器 AX、BX、CX 和 DX，指针寄存器 BP、SP、SI 和 DI，段寄存器 CS、SS、DS 和 ES，指令指针寄存器 IP，标志寄存器 FLAGS。其中标志寄存器中的标志位比 8088 的标志位增加了两个标志，如图 1-6 所示。

15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
×	NT	IOPL	O	D	I	T	S	Z	×	A	×	P	×	C

图 1-6 80286 标志寄存器

标志寄存器中的 CF、PF、AF、ZF、SF、TF、IF、DF 和 OF 各位的含义与 8088 相同，新增加的两个标志 IOPL 和 NT 用于 80286 的保护方式。其中 IOPL 标志是输入/输出特权标志，说明输入/输出操作处于 0～3 的哪一级；NT 标志是嵌套进程标志，表示当前执行的进程是否嵌套于另一进程中，若是则 NT＝1，否则 NT＝0。

2. 80286 新增加的寄存器

(1) 机器状态字寄存器

除以上各寄存器外，80286 增加了一个机器状态字寄存器 MSW。机器状态字寄存器只用了最低四位，如图 1-7 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
×	×	×	×	×	×	×	×	×	×	×	×	TS	EM	MP	PE

图 1-7 80286 机器状态字

MSW 中各位的含义是：

PE 是保护允许位，当 PE 为 1 时表示 80286 处于保护方式，否则表示 80286 处于实