第一章　从 8086 到 80486

1.1　8086 以前的时代

本世纪 50 年代，所有的电子设备（无论是收音机还是电视机）都是笨重的电子管设备，那个时代的计算机被称为第一代计算机。50 年代末期，晶体管和其它固态电路开始取代电子管，利用这种技术生产的计算机被称为第二代计算机，运算速度已达每秒几百万次，体积、重量、耗电以及造价都大为减少。

第三代是中小规模集成电路计算机。第一台这样的机器制成于 1962 年，1965 年开始批量生产。这时已有操作系统，小型机广泛应用，有了终端与网络，运算速度已达每秒千万次。

由大规模集成电路组成的计算机称为第四代。一般认为是从 1972 年开始。这时计算机的体积与成本大幅度地减少，可靠性大为提高，速度每秒已超过 1 亿次。这时，许多部件可以集成到一块很小的硅晶片上，制造微型计算机的条件终于成熟了。

早在 1969 年初，在美国加利福尼亚州的硅谷，刚刚营业几个月，只有 12 人的 Intel 公司得到一家日本计算器公司为计算器生产芯片——集成电路的委托，公司的工程师霍夫和梅泽共同设计了一组芯片，Intel 公司将这种装置称为 4004，因为 4004 是这一装置能替代的晶体管的大致数目，是其复杂程度的一种衡量标准。实际上，这就是人类历史上第一种微处理器，它能处理 4 位二进制数据。

那么，什么是微处理器呢？一个微处理器是指一个单一的硅晶片，它通常由控制部件、算术逻辑部件、寄存器、标志、输入/输出接口等部件组成，也就是通常称为 CPU (Central Processing Unit 中央处理单元）的部分，故微处理器就是微型计算机的 CPU，是微型计算机的核心部分。将微处理器加上电源部件、控制板、输入输出设备如磁盘驱动器、显示屏、键盘等等，就构成一台微型计算机了。但这只是微型计算机的硬件部分，完整的微机还应包括软件，如操作系统、各种编译系统等等。有时我们把上述硬件系统和软件系统的总和称为微型计算机系统。

1971 年，小具规模的 Intel 公司又推出 8008 微处理器。4004 和 8008 就是第一代微处理器，它们都是为专门的应用而设计的，4004 主要用于计算器，8008 则主要用于计算机终端设备。但是，Intel 公司虽然开创了微处理器的时代，却没有得到足够的重视。直到 1974 年，当 8008 成长为第二代微处理器 8080 时，才引起了计算机工业界的震动。8080 是第一个精心设计，可以广泛地在各方面应用的微处理器，可以同时处理 8 位二进制数据，它很快风靡世界。美国各地的电脑爱好者纷纷成立业余计算机小组，他们买不起昂贵的“传统的”计算机，只能买微型计算机，并为微机编制软件。当时年轻的比尔·盖茨和保罗·艾伦曾为微机编写了第一种 BASIC 语言，在业余爱好者中广为流行。许多 Intel 以外的公司开始制造 8080 片子，而且有一些公司制造出了 8080 的增强型产

品，如 Zilog 公司的 Z－80。Intel 也在 1976 年推出了增强型产品 8085。但这些微处理器的基本特点并没有多大改变。

尽管微处理器已发展到第二代，但是不少传统的计算机大公司仍然认为微处理器是不值得发展“小玩意”，计算机界的超级巨人 IBM 公司就是这么看的。所以，微型计算机工业的规模仍不大，每年只能卖出几千台微机。

但是，在 1976 年秋天，刚刚成立不久、推出几十台苹果—I 型微机的苹果电脑公司，令人震惊地推出了功能卓越、外表美观的苹果—Ⅱ型机，引起了计算机工业界的又一次极大震动。苹果—Ⅱ型电脑极为畅销，每三到四个月，产量就翻一番，1977 年一共卖出创纪录的 35000 台，超过前一年总销售量的四倍。苹果公司声名大震。

苹果公司的巨大成功，在计算机工业界引发了两件大事。第一，他们使 Intel 公司感到了强烈的竞争压力。于是，Intel 公司不久于 1978 年推出了第三代微处理器——8086，而后又推出了介于 8086 和 8080 之间的芯片——8088。第二，IBM 公司终于注意到了微型计算机广阔的发展前景，开始在这一领域倾注巨资。

1981 年 8 月，IBM 公司宣布它的第一台个人计算机问世，称为 IBM PC 机。它采用 Intel 公司主频为 4.77MHz 的 8088 作为中央处理器，64KB 的内存，以后又采用 8086 作为中央处理器，采用比尔·盖茨和保罗·艾伦成立不久的 Microsoft 软件公司研制的 MS－DOS 作为操作系统（IBM 称为 PC－DOS）。IBM PC 机的问世，使微机制造者、软件编制者、零售商以及微机购买者市场发生了彻底的、无可挽回的变化。从此，IBM 公司成为微型计算机工业的领导者，Intel 公司、Microsoft 公司则在这棵大树下迅猛成长壮大，经过十多年的发展，成为今天敢和 IBM 分庭抗礼的巨人。

以下，就三个方面较详细地讲述微型计算机的知识，使读者对微机的软、硬件及工作原理有较详细了解，为今后的学习做准备。作者假定：读者已经具有数制（二进制、八进制、十进制、十六进制）、编码（原码、反码、补码）等有关知识，对 DOS 也有初步了解，会简单使用。

1.2 8086

IBM－PC 机从它诞生的第一天起，就显示出令人激动的魅力，PC 机的问世标志着“个人计算机”时代的到来。

1.2.1 微型计算机的组成

描述一台计算机的一种方法是描述组成该计算机的功能部件，这些部件和它们之间的相互作用称为该计算机的结构。微型计算机虽然在具体的结构上比大中型计算机简单得多，但仍然是计算机，仍然是按照冯·诺伊曼提出的逻辑结构设计的。今天的微型计算机有五个关键部分：中央处理器 CPU、存储器（即内存）、输入/ 输出系统（即 Iuput/Ouput 系统，简称为 I/O 系统）、磁盘存储器和程序。微机中还有其他部件象电源、母板、总线（也是 I/O 接口的一部分）及插件框架等。图 1.1 显示了基本部分之间的逻辑关系。

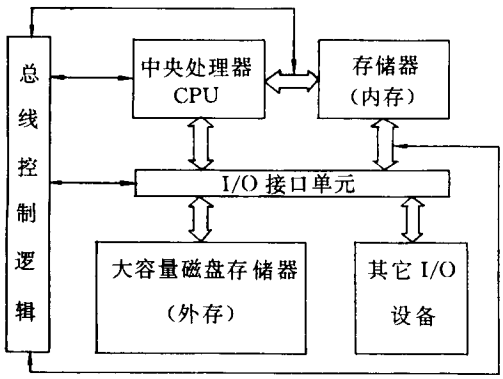

图 1.1　微机的逻辑结构

所有的微型计算机都是如此，相信读者对提到的有关概念并不陌生。我们的学习重点是 CPU 和存储器（内存）的结构，它们的联系非常紧密。要想讲述 80386/80486 的结构特性，我们必须从 8086 及有关基本概念开始，这是学习 386/486 的基础。

1.2.2　8086 成功的奥秘

8086 究竟提供了什么，使它能获得如此成功？要想了解答案，我们必须先了解 8080/8085 的不足之处。

首先，8080/8085 只能处理 8 位字长的数据，数据处理范围只有 0～255 或 - 128～127，远远不能满足需要，而要处理比 8 位字长更长的数据，必须由几个 8 位组来构成，而每一组又必须分开操作，这样就增加了处理时间。8086 则是基于十六位字长操作的芯片，数据处理范围从 0～65535（$2^{16}-1$）或 - 32768～32767，比 8080/8085 大大增加了。而且 8086 也保留了处理 8 位数据的能力，从而使短数据仍能被 8086 有效地处理，具有良好的向上兼容性（即 8080/8085 上编制的程序仍然可以在 8086 上运行）。

其次，8080/8085 的直接寻址范围是 64K 字节（1K = 1024），而 8086 的直接寻址能力达 1M（M 表示兆，$1M = 2^{20} = 1048576$）字节，这在当时是不可想象的，因为在 1980 年，一台相当贵的小型机只有 512～1024KB 内存，价值数百万美元的大型机至多只有 2MB 内存。内存的大幅度增加，有利于编制更复杂的程序。

第三，8080/8085 缺少乘法和除法指令，还缺少带符号数操作，因而使用很不方便，8086 则提供了以前缺少的这些指令。

第四，8080/8085 的寻址方式不能支持把高级语言写的程序转换成有效的 8080/8085 代码，而 8086 的寻址方式则适合于高级语言的处理。

第五，随着系统日益复杂，单靠一个处理器来执行系统全部功能的方法越来越行不通，但是，8080/8085 并不能与其他处理器合作，而 8086 则可以在多处理器环境下运行，尤其是可以与两个出色的协处理器，即数据处理器 8087 和 I/O 处理器 8089 紧密配合，大大提高了 8086 的性能，其中 8087 专门处理数据尤其是实数运算，运算速度比 8086 快一个数量级，8089 则专门用于 I/O 处理，较好地解决了输入/ 输出的“瓶颈”

问题，大大提高了 8086 的效率。

总之，8086 的设计相对于 8 位机有了很大的突破，这个大约有 29000 个晶体管的集成电路的性能较 8 位芯片大约提高了 10 倍。

1.2.3 8086 的存储器结构

8086 的存储器是一个多至 2^{20} 的 8 位数量的字节（byte）序列。每一个字节单元分配一个唯一的地址，地址用无符号数表示，二进制从 0000 0000 0000 0000 0000 到 1111 1111 1111 1111 1111，用十六进制表示即从 00000 到 0FFFFFH，如图 1.2 所示。

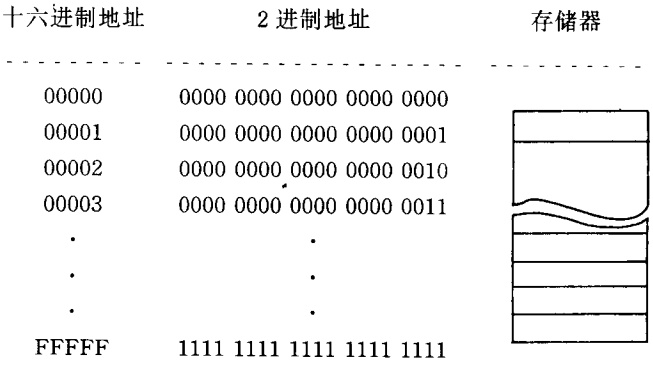

图 1.2 存储器地址

在存储器中，任何两个相邻的字节称为字（Word），任何四个相邻的四字节被称为双字（Double Word），即：一个字是 16 位，一个双字是 32 位。

在一个字中，每一个字节都有各自的地址，那么，字的地址该如何确定呢？我们规定两个字节中地址较小的一个做为该字的地址。因此，字的地址有奇地址和偶地址之分，如图 1.3 所示。

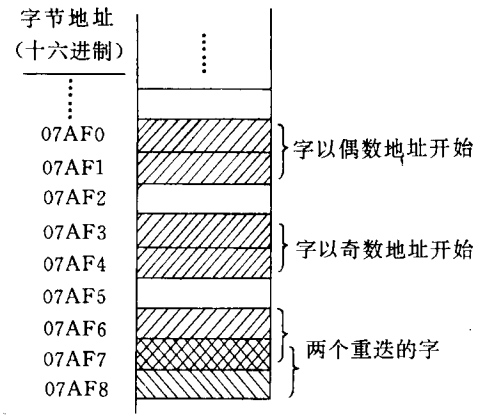

图 1.3 字在存储器中的例子

一个字有 16 位，具有较高存储器地址的字节含有该字的高 8 位，具有较低存储器地址的字节含有该字的低 8 位。比如，当把字 83A5（十六进制）存入存储器中时，先把 A5 存在较低的地址上，再把 83 存在较高的地址上。也就是先存低位字节，再存高位字节。这种情况可用图 1.4 来说明。

双字的情况与字类，如图 1.5 所示。

8086 有些指令是访问（即读或写）字节的，有些则是访问字的。在同一时间，8086 传送到存储器或从存储器中取出的信息数量总是 16 位，并且该 16 位总是在存储

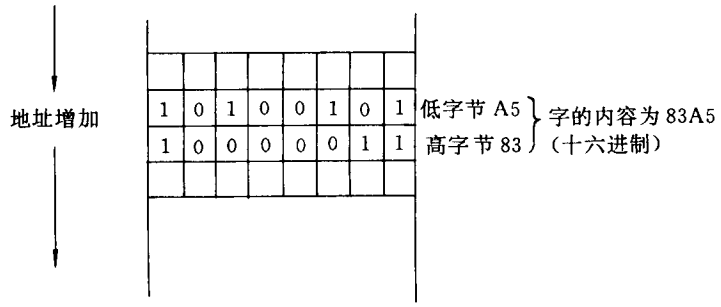

图 1.4　字的存储情况

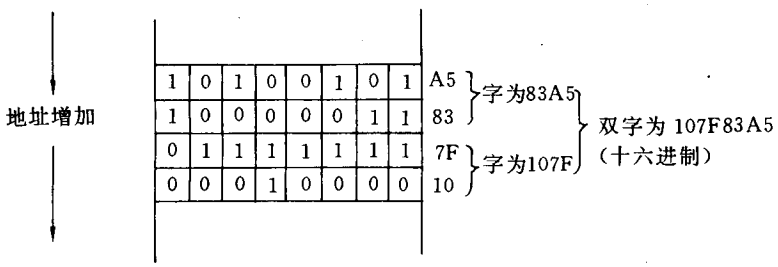

图 1.5　双字的存储情况

器中以偶地址开头的两个字节的内容。在字节指令的情况下，这些位中只有 8 位是有用的，其余 8 位被忽略。由于 8086 读写字时总是从偶地址开始，因此，将字的地址选成偶数，可以提高处理器的效率，这样读写一个偶地址的字时，可以用一个存储器访问周期来实现。否则，对开始于奇地址的字操作指令，就必须对两个连续的偶地址作两次存储器访问，忽略各自不需要的一半，对剩下的一半还要作一些字变换才能得到。各种字和字节的读出例子如图 1.6 所示。

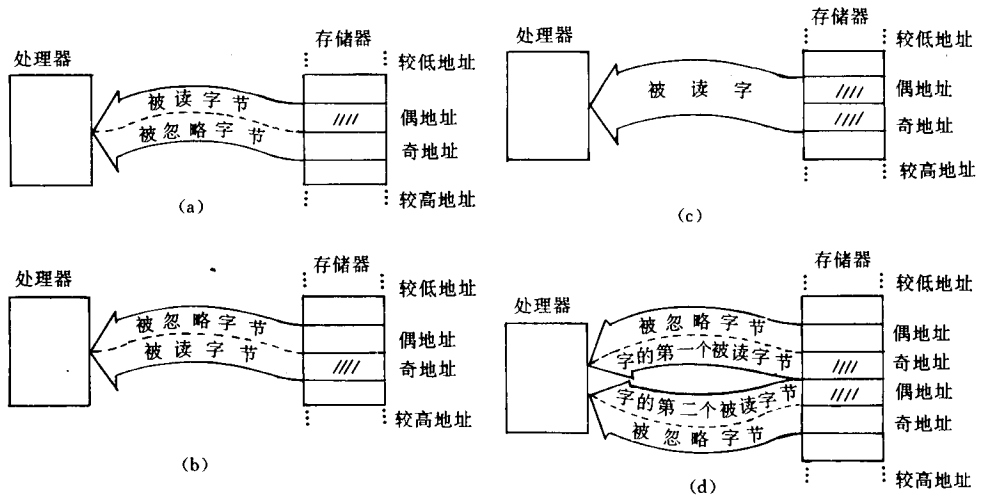

图 1.6　各种字和字节的读出例子

同样，读写双字时，双字的地址最好能被 4 整除，否则也会需要额外的总线周期。

实际上，我们编制程序时并不涉及到这些细节。这些访问，处理器都自动实现。提到这些只是为了增加读者对存储器的了解。

1.2.4 存储器的分段

由于 8086 可以寻址 2^{20}个存储器字节，所以，字节和字的地址必须表示成 20 位的二进制数。但是，8086 是设计来执行 16 位运算的，它只能处理 16 位长的字，所以，就用了一个巧妙的机构来表示地址，这就引出了存储器分段的概念。

我们可以把 1 兆字节的存储器想象为任意数量的段，其中每一段最多可含有 2^{16} (65536) 个字节，而且，每一段必须开始于一个能被 16 整除的字节地址（即该字节二进制地址的最低 4 位全是 0），这个地址被称为**段地址**。这样，一个段的段地址具有十六进制数 XXXX0 的形式。段中某一字节或字的绝对地址减去段地址，称为这一字节或字在这个段中的**偏移地址**。可以看出，段地址虽然是 20 位二进制数，但低 4 位全是 0，而偏移地址由于小于 2^{16}，所以可以用 16 位二进制数表示，这样，我们就有办法用两个 16 位二进制数来表示段和段中操作数的地址，这就又引出**寄存器**（Register）的概念。

所谓寄存器，就是这样的存储单元：和存储器（内存）不一样，它们在 CPU 中，直接参与运算，存取速度比存储器更快。CPU 在处理数据过程中往往用寄存器存放中间结果，运算的初始值也必须从内存中调入寄存器，运算完毕后再将终值返回到内存，也可以用寄存器表示系统的状态和内存中段的基地址及段中数据的偏移地址。寄存器的设置可以大大提高运算速度。

在 8086 CPU 中，有 13 个内部寄存器，这些寄存器都是 16 位的。其中有 4 个寄存器：CS、DS、SS 和 ES，用来表示段的基地址。由于 20 位段地址的低 4 位全为 0，所以把段地址的高 16 位存入它们即可。在 8086 中，有三种类型的段：用来存放程序码的代码段、用来存放程序所需有关数据的**数据段**和用来存放由于子程序调用等原因需要保存特殊数据的**堆栈段**。CS 是 Code Segment 的缩写，表示代码段；DS 是 Data Segment 的缩写；SS 则表示 Stack Segment；ES 表示 Extra Segment，即附加段。在 8086 中，CS 用于存放当前代码段的段地址，SS 则存放当前堆栈段的段地址，DS 和 ES 可以存放两个数据段的地址，一个是“数据段”，一个是“附加段”，本质上都是数据段，只是为了提高速度，为了编程需要才设立了两个数据段寄存器。

图 1.7 所示的是 4 个段寄存器分别指示着存储器中 4 个当前段的例子，在这个例子中，假定每 1 个逻辑段的长度都 64KB。例如，假定代码段寄存器 CS 中含有的十六进制数值是 D018，就表示代码段的基地址是 0D0180H（在表示十六进制数时，若第一个数字是表示十进制数 10 到 15 的 A、B、C、D、E、F，则常在它们之前加一个 0，以便把数字和字符区别开，数字末尾的 H 表示它是一个十六进制数），这样，由于该代码段的长度为 2^{16}（10000H）个字节，所以该代码段的最后一个字节的地址将是 0D0180H + 0FFFFH = 0E017FH。注意段与段之间是可以重叠的。

那么，如何通过段地址和偏移地址来寻找该段里的字节和字呢？由于 16 位的段地

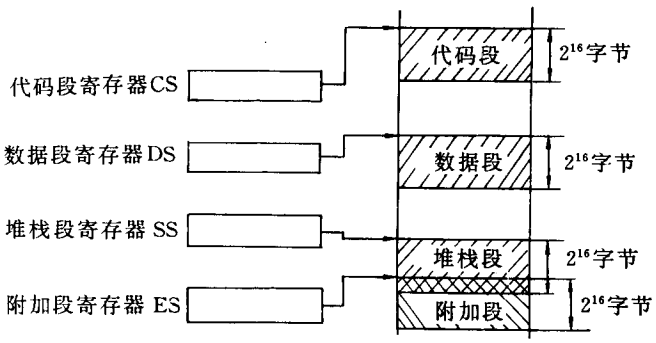

图 1.7　8086 的段寄存器

址实际上表示低 4 位是 0000 的 20 位物理地址，所以 8086 处理器实际上是把 16 位偏移地址加上附加有最低 4 位 0 的 16 位段寄存器的内容来产生 20 位的物理地址，操作的原理如图 1.8 所示。

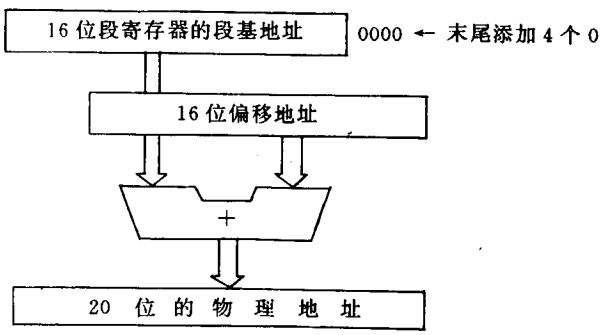

图 1.8　20 位物理地址的形成原理

例如，一个字在数据段 DS 中的偏移地址为 18H，DS 中的十六进制数是 7000H，这样，这个字的物理地址就是 70018H。数学运算如下：

$$\begin{array}{r} 7000\ 0H \\ +\quad 001\ 8H \\ \hline 7001\ 8H \end{array}$$

那么，我们如何通过计算机具体“看到”存储器中的情况呢？

打开计算机电源，计算机启动完毕之后显示提示符 C>，等待用户的下一步动作（如果使用软盘，则出现提示符 A>或 B>）。检查在 C 盘中或 C 盘的 DOS 子目录中是否有文件 debug.exe。debug 调试程序是 DOS 系统盘所提供的程序之一，假设 debug.exe 是在 C 盘的根目录下。

在 DOS 提示符后，键入“debug”，然后按回车键：

```
C:\>debug
```

这时屏幕上出现短划线“-”，它是 debug 的“提示符”，表示机器正在运行 debug，已准备好接收下一个命令。

(1)“D”命令

debug 的命令都是单个字符，可以大写也可以小写。在它后面跟一些十六进制数字来限定范围。我们现在学习第一条 debug 命令：D 命令。D 命令（Dump）就是在屏幕上显示存储器的一部分内容的命令。

在提示符下键入 d100，然后回车，即：

```
C:\>debug
-d100
1660:0100 03 E9 0E 01 2E C6 86 AB-03 01 2E 8B 8E DC 03 D1  ................
1660:0110 E9 2E 8C 86 AE 03 1E 07-BE B0 03 03 F5 BF 9F 01  ................
1660:0120 03 FD F3 A7 75 23 E9 9C-00 E9 FA 40 00 D7 04 22  ....u#.....@...″
1660:0130 19 35 93 59 57 54 80 00-00 4D 5A 6F 01 1F 00 03  .5.YWT...MZo.....
1660:0140 00 20 00 00 00 FF FF 62-03 E8 E6 00 8C C8 BF 06  .....b..........
1660:0150 00 BE D0 03 03 F5 8E C0-40 3D 00 A0 75 03 E9 B1  ........@=..u...
1660:0160 00 B9 04 00 F3 A7 75 E6-1E 2E 8E 9E AE 03 2E 8B  ......u.........
1660:0170 96 AC 03 B9 20 00 B8 01-43 CD 21 2E 8E 9E AE 03  .......C.!.......
```

这些显示内容是什么意思呢？最左边一列 1660H 表示段地址（你的计算机上显示的段地址可能与此不同），在这里是 CS 中的地址，冒号后面的 4 位数表示相对于段地址的偏移地址。再后面的 16 个两位数就是每个存储器单元中存储的十六进制数（即 8 位二进制数）。事实上，十六进制数是 debug 唯一认识的数制系统，请读者要熟悉这种表示方法，不要和十进制弄混了。

对于十进制数，在数字后面跟一个小写字母“d”，二进制数则跟“b”，上面屏幕上显示的一共 8 行，每行在十六进制中是 10 个存储单元，用十进制数表示则是 16 个，这样，屏幕上共显示了 128d 个字节。打印输出的中间有一短划线，是为了区分一行中左边与右边 8 个字节。

读者一定要熟悉用段地址和偏移地址表示地址的方法，下图更明确地说明了地址和存储器单元中的数的一一对应关系。

屏幕最右面的那些小点和字符，是存储单元中的十六进制数对应的 ASCII 码。

我们还可以随便看看存储器中其它部分的情况。如键入“d1000:0”，回车，显示出从地址 1000:0000 到 1000:007F 中的存储单元的情况。刚才第一条命令“d100”默认当前段地址是 CS 中的数，现在则完整地键入“d 段地址：偏移地址”命令。

读者还可以连续地键入“d ”，屏幕上就连续地显示存储器中的内容。

(2)“F”命令

如果要想改变显示内容，可以使用 debug 的“F”即“Fill”命令，格式为“F 起始地址 终止地址 要填入的常量”，注意它们之间的空格。这一命令是用指定的十六进制

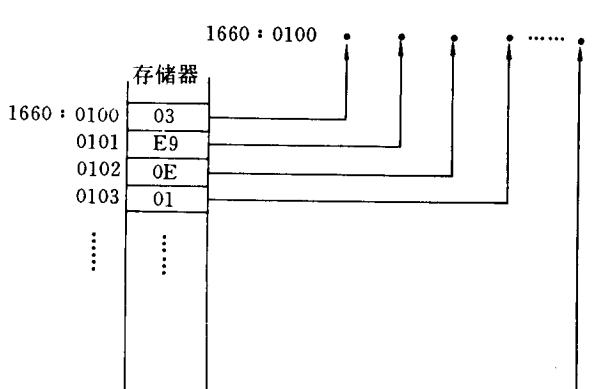

图 1.9　地址和存储器单元中的数的对应关系

数（即要填入的常量）填入从起始地址到终止地址这一部分单元，注意三个数全都是十六进制数。如果 4 位偏移地址的开始几位是 0，可以不必键入，所以有时可以键入少于 4 位数地址。如下所示：

```
-f120    14f  ff  ←——要填入的常量
 ↑       ↑
起始地址  终止地址
```

然后回车，再用 d 命令看看存储器：

```
-f120 14f ff
-d100
1660:0100 03 E9 0E 01 2E C6 86 AB - 03 01 2E 8B 8E DC 03 D1  ................
1660:0110 E9 2E 8C 86 AE 03 1E 07 - BE B0 03 03 F5 BF 9F 01  ................
1660:0120 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF  ................
1660:0130 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF  ................
1660:0140 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF  ................
1660:0150 00 BE D0 03 03 F5 8E C0 - 40 3D 00 A0 75 03 E9 B1  ........ @=.. u...
1660:0160 00 B9 04 00 F3 A7 75 E6 - 1E 2E 8E 9E AE 03 2E 8B  ...... u.........
1660:0170 96 AC 03 B9 20 00 B8 01 - 43 CD 21 2E 8E 9E AE 03  .... ... C.!.....
```

可以看出，从 120 到 14f 之间的三行全都填了 ff。

还可以键入 “ -f100 10f 61”，回车，再键入 “d100”，回车，屏幕上显示：

```
-f100 10f 61
-d100
1660:0100 61 61 61 61 61 61 61 61 - 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
1660:0110 E9 2E8C 86 AE 03 1E 07 - BE B0 03 03 F5 BF 9F 01  ................
1660:0120 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF  ................
1660:0130 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF  ................
1660:0140 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF  ................
```

```
1660:0150 00 BE D0 03 03 F5 8E C0 - 40 3D 00 A0 75 03 E9 B1   ..... @ =.. u...
1660:0160 00 B9 04 00 F3 A7 75 E6 - 1E 2E 8E 9E AE 03 2E 8B   ..... u.........
1660:0170 96 AC 03 B9 20 00 B8 01 - 43 CD 21 2E 8E 9E AE 03   .... ... C.!.....
```

注意最右边出现了一串“a”，它是十六进制数 61h 所代表的 ASCII 码，而象 00 和 0ffh 之类所代表的 ASCII 码不可打印的，所以显示一个小点“·”。

建议读者多试试这两条命令，观察显示结果及存储器的各个部分。以后，我们还会更多地使用 debug，并体会其方便性。

由于段寄存器指向 4 个当前可寻址的段（图 1.7 所示），所以可通过改变段寄存器的内容使其指向所要求的段，程序就可以对新的段中的数据和代码进行访问，这样，就可以寻址 1M 地址空间内的任何存储单元，这就是 8086 设计的精妙所在。而且，当前可寻址段已提供了广阔的工作空间：64K 字节代码段，64K 字节堆栈段和 128K 字节的数据段，这在当时已经足够用了。

有人可能会问：将程序和数据及有关的数据结构合在一起难道不好吗？这样就可以设置一个单一的段而不必要弄四个段了。是的，对于简单的程序，完全可以合成一个段（虽然在代码段中表示堆栈的数据结构复杂一些，而且容易出错），但是对于复杂的程序，分段则有巨大的优越性。8086 存储空间的分段结构，增加了程序和数据的独立性，有力地支持了模块化的软件设计，以避免搞巨大的、单块的程序。利用分段技术，程序涉及的可以只是逻辑地址（偏移地址）而不是实际地址，这样可以较大自由地指定段基地址，便于存储器资源的动态管理。所以，8086 的存储器分段，实际上为模块化、结构化的程序设计提供了硬件支持，也便于内存管理。

也有人可能会问：为什么不设立 20 位的段寄存器呢？这样不就可以直接寻址而不必要弄得那么复杂了吗？的确如此，可是计算机芯片的设计需要考虑多种因素，其中一个重要因素就是兼容性问题。由于 8080/8085 是 8 位的，所以 8086 的运算器设计成 16 位，以保持良好的兼容性，芯片的电路设计也不致于变化太大。由于地址经常需要参与运算，而 20 位的地址与 16 位的数要运算，在电路设计上十分困难，所以干脆就把存储器分段，利用基地址和偏移地址来寻址。由于都统一成 16 位，就提高了机器的速度，虽然我们学起来是困难一些。

但是，这个问题并不是没有道理的。随着计算机技术的发展、人们对数据处理要求的提高，出现了 32 位的微处理器。Intel 80386 和 80486 就是 32 位的。以下我们就可以看到，Intel 公司为了既提高芯片的性能以便和别的厂家的芯片竞争，又和 8086 保持良好的向上兼容性，费尽了心机，结构上越来越复杂，对很多问题的处理可谓苦心孤诣。

1.2.5 8086 的其它寄存器

在 8086 中，除了四个段寄存器 CS、DS、SS、ES 外，还有几个 16 位寄存器，它们是：通用寄存器 AX、BX、CX、DX，基址和变址寄存器 SP、BP、SI、DI，指令指示器 IP 和标志寄存器 FLAGS。它们表示在图 1.10 中，它们的意义、作用及使用方法，当我们讲述 386 时就会明白。

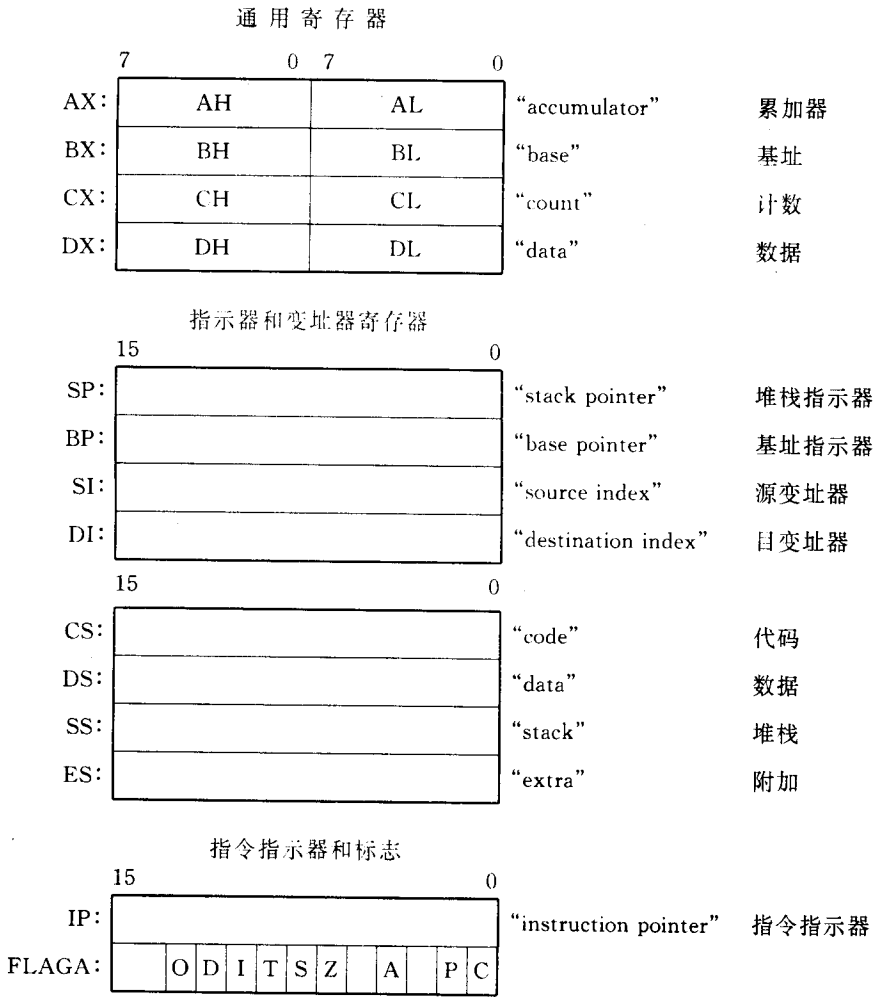

图 1.10　8086 的寄存器集

1.3　80186 和 80286

1.3.1　8086 的改进型 80186

1982 年，Intel 公司推出比 8086 功能更强的 80186 芯片，它具有一个与 8086 一样的公共的基本结构，所不同的是它加强了总线接口部件和执行部件的硬件功能。例如：8086 是用内部微程序来计算地址的，而 80186 则用专用的硬件加法器来实现。此外，80186 还提供了许多新的指令，它们能简化汇编语言的程序编制，增强实现高级语言的性能和减少目标代码的长度。

但是，80186 并不流行，其主要原因在于：就在同年，Intel 公司推出了具有划时代意义的第四代微处理器——80286。

1.3.2 第四代微处理器的先驱：80286

随着微型计算机的普及及技术的发展，8086 暴露出越来越多的缺点：

第一，速度还是不够快。

第二，随着软件规模的迅速增大，用户可用的 640KB 内存越来越显得局促。

第三，不能支持多任务处理，也就是说，在同一时间内，只能有一个程序运行。

正因为如此，Intel 公司耗费巨资设计出新一代芯片——80286。1984 年 8 月，IBM 将这种芯片用在技术先进的 AT 微机上。这种机器问世后，立即造成了极大轰动，极为畅销。

首先，286 使用了更多的内存，可达 16MB。

其次，引用了“虚拟存储器”的重要概念，使处理器利用外存来模拟内存，这样，可以利用外存模拟多达 1 千兆（GB）的虚拟存储器。

第三，计算机可以同时运行多个任务。多任务是通过硬件机构使处理器在各种任务间来回切换实现的。

形象地说，屏幕上可能有几个窗口，其中每个窗口中有一个程序，处理器在各个程序之间以极快的速度来回切换，使所有的程序看来都在同时运行，这就是“多任务”。286 是用硬件机构处理多任务的，而原来的处理器尽管可以用软件中断实现多任务，但不可靠。

第四，286 有两种工作方式：实地址方式和保护方式

第五，速度快。286 和 8086 在相同时钟频率下运行同一软件时，前者比后者要快 2．5 倍。

不过，286 的先进特点，几乎没有多少用户能利用它们，这是因为 DOS 这时已成为绝大部分微机的操作系统，而 DOS 却是以原有 8088 体系结构为基础的。DOS 程序应该在实地址方式下运行，但在 DOS 程序之间切换时又必须在保护方式下进行，又由于 286 设计上的缺陷（当然，另一方面是 DOS 的缺陷），CPU 一经转换成保护方式，DOS 程序就会失效。而且，286 使用 24 条地址线和 16 条数据线，数据处理的范围仍不是很大。

正因为如此，1985 年 10 月，Intel 公司又研制成 80386 芯片，严格来说应该是 80386DX（32 位总线），并于 1987 年正式投放市场，这再一次震惊了工业界。

1.4 成熟的第四代微处理器：80386

相比于 80286，80386 有一些无可比拟的优势：

第一，386 是真正的 32 位 CPU，寄存器经过大规模扩充，数据总线、地址总线都是 32 位，寻址能力达 4 千兆字节（2^{32} = 4 294 967 296 = 4096 兆），虚拟存储空间则达 64 兆兆（2^{46}）字节。

第二，改进了多任务处理技术，内存管理技术有很大提高，1988 年，就象由 8086 产生 8088 一样，由 80386DX 产生了 80386SX，80386SX 是一个内部 32 位总线，外部

16 位总线的微处理器。

以下，我们将详细讲述 32 位 386 即 386DX 的系统结构。

1.4.1 80386 的体系结构

在早期的 8086CPU 内部，CPU 被分成两个独立的处理单元：执行单元（Execution Unit，简写为 EU）和总线接口单元（Bus Interface Unit，简写为 BIU）。执行单元负责指令的执行，总线接口单元则负责 CPU 与内存和外部设备之间的信息传送。这种把指令运行逻辑与总线控制逻辑分割开来的方法，可异步操作以提高 CPU 的性能。

386 的体系结构要大大优于 8086。它的 CPU 被分成六个独立的处理单元：总线接口单元 BIU、代码预取单元、指令译码单元、执行单元 EU、分段单元和分页单元。如图 1.11 所示。

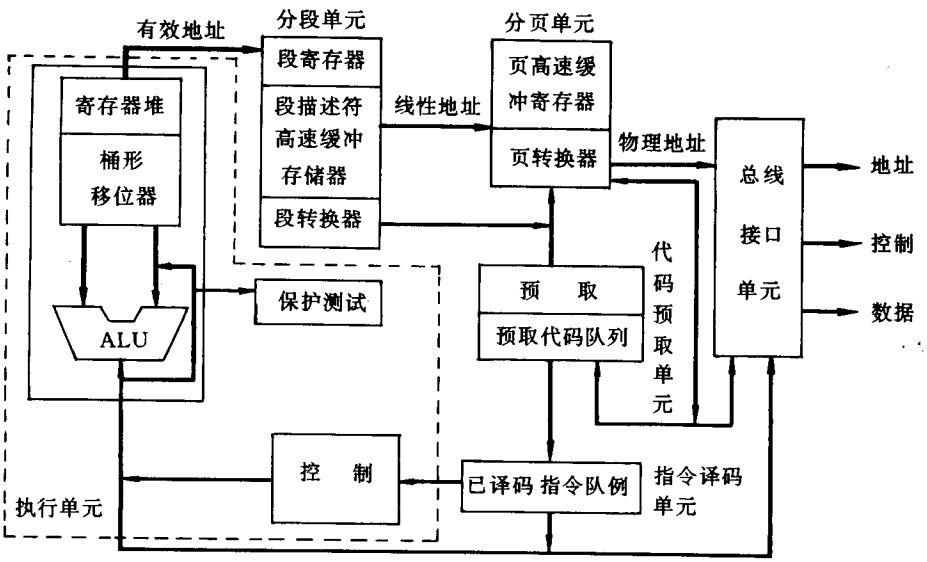

图 1.11 80386 的体系结构

总线接口单元：同 8086 一样，总线接口单元负责 CPU 与外部世界的通信。由于有可能在同一时刻有多个外部器件及 CPU 的内部单元请求访问总线，所以 BIU 必须按优先级顺序排列这些请求，这就是总线控制的作用。

代码预取单元：386 将取指令分成一个独立的处理单元，在它的预取队列中可存储 16 字节长的指令。指令一旦被预取，它就准备由指令译码单元译码。

指令译码单元：指令译码单元从代码预取单元中读取指令字节并译码，放在由执行部件使用的译码指令队列中。

执行单元：386 的执行单元又可分成三个子单元：控制子单元、数据子单元和保护测试子单元。

控制子单元的功能就是加速某些运算的实现，包括乘法、除法和有效地址的运算。

数据子单元包括算术逻辑单元 ALU 的 8 个 32 位通用寄存器，算术逻辑单元是

CPU 的真正核心，它最终实现数据的运算，64 位的桶形移位器是一个执行移位的专用硬件，用于加速移位循环及乘除法操作，使典型的 32 位乘法可在 1 微秒内完成。

保护测试子单元监视存储器访问是否超越了程序的分段。

分段单元：为了提高程序的灵活性，程序中涉及的地址一般都是逻辑地址。段单元是实现地址转换的第一阶段，它把逻辑地址转换成线性地址。段描述符高速缓冲寄存器用来加速地址转换，并在性能不受影响的情况下检查是否违反了保护条件，它可以实现任务的隔离和代码段与数据段重定位。地址转换之后，线性地址就传送给分页单元。为了使应用程序和操作系统各自得到保护，分段单元提供了四级保护机构。这种由硬件实施的保护，使各种系统的设计具有高度的完整性。

分页单元：分页单元将分段部件产生的线性地址转换成物理存储器地址，分页机构提供了对物理地址空间的管理，每一页是 4K 字节。

页高速缓冲寄存器保存最近使用的 32 个页表的入口地址。当需要存取页时，首先检查页的标记及地址是否在页高速缓冲寄存器中，这样可以提高存取的速度。

1.4.2 80386 的寄存器结构

386 一共有七类共 34 个寄存器。按其功能分为：

一、通用寄存器

一共有 8 个，如图 1.12 所示。

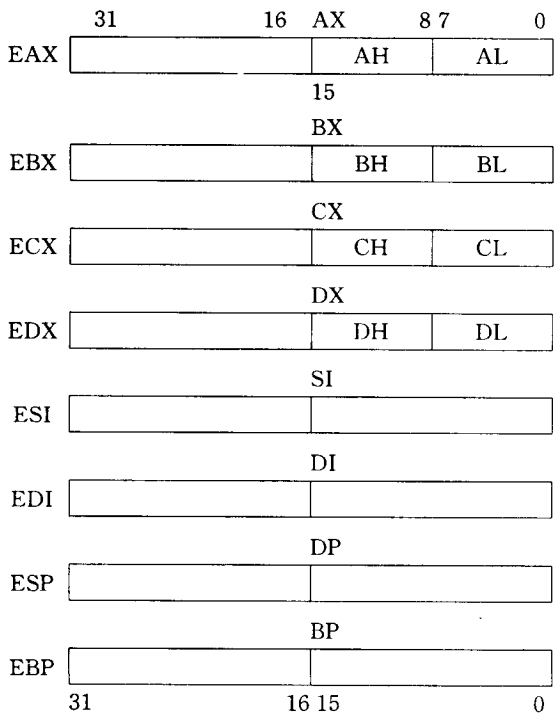

图 1.12　通用寄存器

这 8 个寄存器都是 8086 相应寄存器的扩展，低 16 位可以分别存取，AX、BX、CX、DX 又可分成高低两半共 8 个 8 位寄存器，它们也可以分别存取。这样做的原因是为了保持和 8086 的兼容性。原则上它们都可以参与数和地址的计算，但是，在汇编语言程序中，将它们作一些区分是有好处的，不仅可以使我们望文生义，而且在硬件设计上，每个寄存器也确实各有侧重。它们的通常用途为：

(E) AX 和 AL：通常用作累加器或参与乘除法运算。

(E) BX：通常用作基址寄存器。

(E) CX 和 CL：通常用来计数，比如在循环或串操作时。

(E) DX：通常参与乘除法运算或存放数据。

(E) SP：通常用作堆栈指针。

(E) BP：通常用作基址指针。

(E) SI：通常用作源变址。

(E) DI：通常用作目的变址。

大家不必死记硬背，以后多读程序，自然而然就会明白并逐渐达到熟练使用的境地。

二、段寄存器和段描述符寄存器

80386 有 6 个 16 位（不是 32 位!）段寄存器，它们的名称用途如下：

CS：代码段寄存器；

DS：数据段寄存器；

SS：堆栈段寄存器；

ES：附加（数据）段寄存器；

FS：附加（数据）段寄存器；

GS：附加（数据）段寄存器；

在实地址模式下，段的大小最大为 64K 字节。在保护模式下，段的大小可达 4G 字节，这时，段寄存器是如何表达段的基地址和段长的呢?

在实模式下，段寄存器内的数是段的实际基地址（左移 4 位即可）。但在保护模式下，段寄存器内保留的是**选择符**（Selector），这个选择符指向一个段描述符。段描述符中含有一个 32 位的段基地址、一个 32 位的段界限和其它一些必要属性，段描述符集构成描述符表。每个段寄存器都有一个相关的段描述符寄存器，对程序员来说，段描述符寄存器是不可见的。当一个选择符的值装入段寄存器时，相关的段描述符寄存器的内容就自动修改成选择符所指向的段描述符的信息，当访问存储器时，所有与其相关的段描述符寄存器自动参与这次存储器访问操作。32 位段基地址形成线性地址计算中的一个分量，32 位段界限用于界检验操作。

那么，怎么看到这些寄存器呢? debug 提供了一条命令，使我们不仅能看到 16 位寄存器，而且还能改变其内容。

“R”命令。

仍然象前一节那样装入 debug，出现提示符后，键入“r”（代表寄存器），此时，就会看到令初学者眼花潦乱的内容。

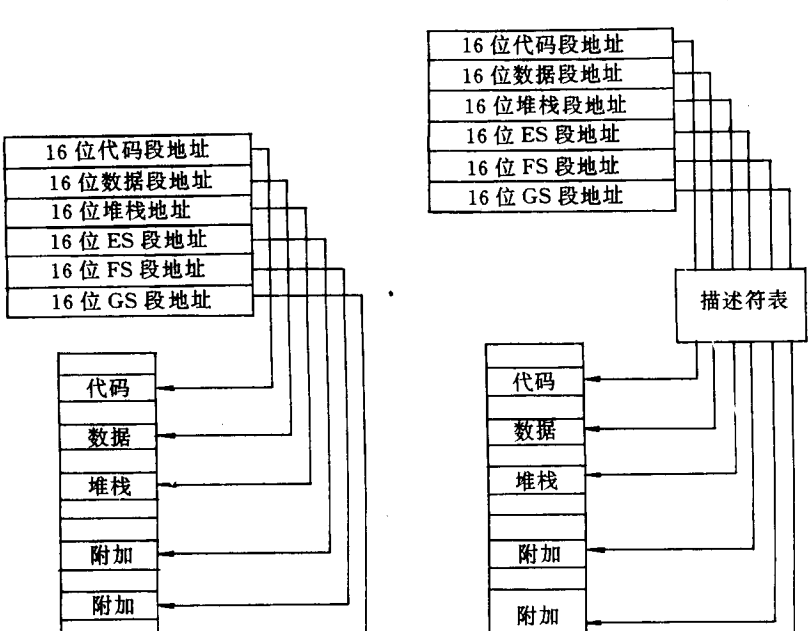

图 1.13　不同模式下的寻址方法

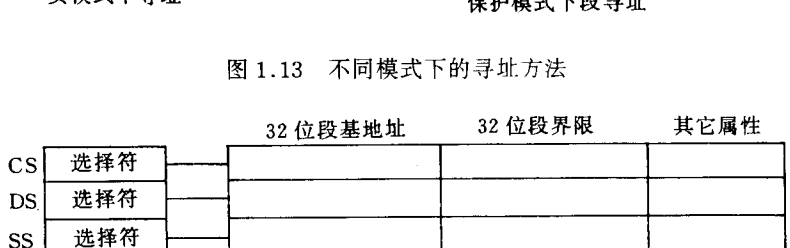

图 1.14　段描述符寄存器

```
C:\>debug
-r
AX = 0000 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000
DS = 1660 ES = 1660 SS = 1660 CS = 1660 IP = 0100 NV UP EI PL NZ NA PO NC
5309:0100 03E9            ADD         BP, CX
```

仔细看看就会明白，这告诉用户 13 个 16 位寄存器中的内容到底是什么，可以看出，AX、BX、CX、DX 四个主要寄存器的内容都是 0。DS、ES、SS、CS 四个段寄存器中的内容都相同，IP 是指令指针寄存器，下面就会讲到；IP = 0100 后的 NV……是标志寄存 FLAGS 中位的相应值，下面也会讲到。

我们可以改变寄存器中的内容。方法是：键入“r 寄存器名”，比如，要改变 AX

中的内容，就键入“rax”：

```
-rax
AX 0000
:←这里 debug 等待用户键入新值
```

假如键入 1234：

```
-rax
AX 0000
:1234
-（回到 debug 提示符）
```

这就把 1234h 送入 AX 寄存器。

为了验证确实如此，再键入“r”：

```
-r
AX = 1234 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000
DS = 1660 ES = 1660 SS = 1660 CS = 1660 IP = 0100 NV UP EI PL NZ NA PO NC
5309:0100 03E9            ADD       BP, CX
```

可见，1234H 确实已在 AX 中。

可用这个方法处理任何一个寄存器。

最后一行所表示的意思，以后再深入研究，但要注意，最后一行包含了一条指令。

很遗憾，debug 并不能显示 32 位寄存器的内容。

三、状态和控制寄存器

一共有 6 个，它们是：标志寄存器 EFLAGS、指令指针 EIP（Iustruction Pointer）和 4 个控制寄存器 CR0～CR3（Control Register），如图 1.15 所示。

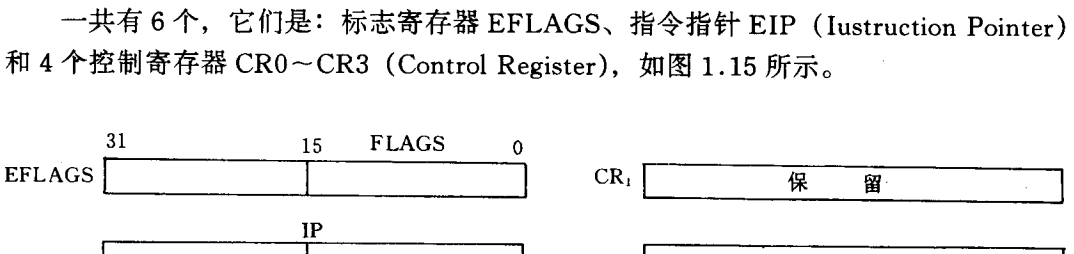

图 1.15 状态和控制寄存器

标志寄存器 EFLAGS 存放有关系统的信息，如图 1.16 所示，各位解释如下：

CF（Carry Flag）：进位标志。当算术运算产生进位或借位时，CF = 1，否则为 0。

PF（Parity Flag）：奇偶标志。若操作结果中“1”的个数为偶数，则 P = 1，否则 P = 0。这个标志可用来检查数据传送过程中是否发生错误。

AF（Auxiliary Flag）：辅助进位标志。在字节操作时，若由低半字节（一个字节