汇编语言百日通

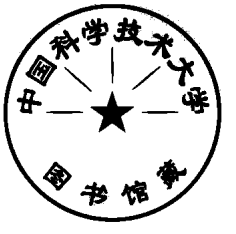
HUI BIAN YU YAN BAI RI TONG

陈国章 王剑君 蔡兆海 编著

天津科学技术出版社

汇编语言百日通

陈国章　王剑君　蔡兆海编著



天津科学技术出版社

责任编辑：徐彤

汇编语言百日通

陈国章　王剑君　蔡兆海　编著

*

天津科学技术出版社出版

天津市张自忠路189号　邮编300020

河北省迁安县印刷厂印刷

新华书店天津发行所发行

*

开本 787×1092毫米　1/16　印张 21.25　字数 509 000

1996年12月第1版

1996年12月第1次印刷

印数：1—3 000

ISBN 7-5308-2004-4

TP·88　定价：24.00元

内 容 提 要

本书是为了学习汇编语言的读者而编写的。该书内容丰富、全面，共分十章。具体包括：预备知识；汇编语言介绍；指令系统；汇编语言程序设计；中断系统；输入和输出；宏汇编；基本程序设计技术；高级程序设计技巧和汇编语言程序上机指导等。

本书可作为计算机专业学生的学习参考书，也可供从事微型计算机应用的科技人员阅读和使用。

前　言

汇编语言是计算机专业中一门重要的技术基础课。为了帮助所有想学习汇编语言的读者尽快地学好和掌握汇编语言，我们编写了《为汇编语言百日通》一书。

汇编语言是一种强有力的语言，它能够使程序设计人员完全控制计算机的操作。它不同于一般的计算机高级语言（如：BASIC 和 FORTRAN 语言等），当程序设计人员采用高级语言设计程序时，只能依赖相应于该高级语言的解释程序或编译程序，当这些解释程序或编释程序不支持某种特性时，则程序设计人员也就束手无策了。遇到这种情况，程序设计人员可以采用汇编语言，自己编写子程序或者直接利用功能极强的 BIOS 子程序来满足程序设计要求。可见，汇编语言也已看成是对各种高级语言的补充。

本书的特点是：合理选材，条理清楚，通俗易懂，内容充实，实用性强。从内容上看，全书共分十章。第一章，重点介绍 80286 微处理器和汇编语言的数据表示。第二章，主要介绍组成汇编语言的基本元素和汇编语句。第三章，重点介绍 80286 指令系统。第四章，重点介绍汇编语言程序设计。第五章，着重介绍中断系统以及利用中断的应用实例。第六章，介绍汇编语言的输入和输出。第七章，介绍了宏汇编的相关内容。第八章，介绍基本程序设计技术，其中包括代码转换和表格的应用等。第九章，通过实例来介绍高级程序设计技巧。第十章，为帮助读者学好汇编语言，提供了必要的上机指导。最后，我们还给了几个附录以便读者在编写程序时查阅和使用。

本书在编写的过程得到了许多专家的支持和鼓励，其中有：南开大学教授齐寅峰；河北农业大学教授张德培和陈文光等，编者对此表示深深的谢意。

希望读者提出宝贵意见。

编者

1996.1

目　录

第一章 预备知识

§1.1 为何需要汇编语言

我相信，许多使用过计算机的人都用某种高级语言编写过计算机程序，例如 Pascal，C，Fortran，尤其是 Basic。Basic 学起来容易，用起来方便，用它编写程序可以很快完成大多数计算任务。既然如此，为何人们还要用到其它语言呢？原因之一就是 Basic 适用范围有限，例如进行科学计算，用 Fortran 更适宜，而编写系统程序等用 C 语言更显得得心应手。可见，每种高级语言各有自己的特点。然而，这些高级语言又有一个共同的特点是离硬件较远，都属于面向问题的语言，那就是用高级语言编写的程序，机器不能直接执行，必须由编译程序或解释程序将它翻译成对应的机器语言程序，机器才能接受。但是，通过编译程序或解释程序生成的机器语言程序往往比较冗长，占用的存贮空间也较大，因而执行速度较慢。这对于处理大量信息及图形显示，或是要考虑发生的情况较多等，会使计算机用较多的时间用于翻译程序，而没有用于执行程序，显然高级语言执行速度慢的特点不能满足用户的需要。另外，高级语言的程序员也无法直接利用机器硬件系统的许多特性，例如寄存器、标志以及一些特殊指令等等，影响了许多程序设计技巧的发挥。为解决这些矛盾，有必要使用占用空间少，而执行速度快的汇编语言。

汇编语言（assembler languecge 或 assembly language）是一种面向机器的程序设计语言，是一种初级语言。该语言的基本内容是机器语言符号化的描述。虽然，汇编语言也和其它高级语言一样，是一些字符的集合，告诉计算机做什么，但是汇编语言指令集中的这些字符直接涉及到计算机的组成部分，它是直接利用机器提供的指令系统编写的程序，它给出的是具体命令，而非高级语言只给出一般性命令。通常，汇编语言的执行语句与机器语言的指令是一一对应的关系。也就是说汇编语言的一个执行语句对应一条机器语言指令。

汇编语言提供给编程者对计算机的更大的控制权。但这需要更多更细致更不方便的工作。高级语言对于大多数编程者在大多数情况下是很好用的，但对于那些要求计算机最大工作性能的人来说，汇编语言是必须的。换句话说，汇编语言使程序员能够完全控制计算机的操作。高级语言使程序员只能依赖于解释程序或编译程序编写者的思想。如果编译程序的编写者支持某种特性（例如查询游戏适配器），那自然好。但是如果他不支持，我们就无能为力了。如果用汇编语言，我们就可以编写自己的子程序，或者利用计算机制造商提供的功能很强的 BIOS 子程序来完成特定的任务。事实上，绝大多数的汇编语言程序都是对各种高级语言的补充，加上了许多原编写者遗漏了的特性。我们可以看到，大多数复杂的游戏，图形程序和大的商业程序至少有一部分是用汇编语言编写的。

可见，用汇编语言编写的程序比用高级语言编写的程序执行的速度快，占用的内存少，使用灵活，但这是用比高级语言的编写程序更费时费力的代价换取的。那么是否采用汇编语言编写程序，要看具体的应用情况，在软件的开发时间及软件的质量方面进行权衡和选择。一般来说，对于执行时间和存贮器容量要求较高的程序采用汇编语言编写，如实时控

制系统、智能化仪器仪表及高性能软件等方面。

用汇编语言编程可以充分发挥机器硬件的功能并提高编程的质量。为此，学习汇编语言程序设计首先应该熟悉机器的指令系统。而指令系统又是与具体机器的内部结构密切相关的，因此我们要熟悉计算机的内部结构，尤其是中央处理器（CPU）和存贮器的结构。另外我们还应熟悉机器中与编程有关的其它部分的结构，例如中断系统等。此外，我们还必须知道系统为我们提供的软件环境，如存在磁盘上的操作系统、汇编程序、调试程序等。我们可以充分利用它们的支持，直接调用其中的子程序。

因为汇编语言与机器关系很密切，而机器因机型不同而异，所以通常它是为特定的计算或特定的计算机系列设计的，因而就出现了 Z80 汇编程序，8086/8088 汇编程序以及我们下面所要介绍的 80286 汇编程序。

§1.2 80286 的由来

最早的微处理器是 4 位的，也就是说，它一次只能传送 4 位信息。为了传输多于 4 位的信息，只好分几次传送，所以速度很慢。

1972 年推出的第一个商用 8 位微处理器，每次可以传送 8 位信息，被认为是第一代 8 位微处理器，它是 Intel 公司推出的 8008。它是按类似计算器的体系结构设计的，包括一个累加器，六个便签寄存器，一个栈指针寄存器，八个地址寄存器，还有一些特殊的输入输出指令。1973 年 Intel 公司又推出了第二代的 8008 称为 8080。

8080 可以说是升级了的 8008，它有更强的寻址和输入输出能力，更多的指令，执行起来更快。虽然在 8080 的设计中保留了 8008 的全部设计思想，但 8080 的内部组织更合理。

到了 1976 年，技术的进步使得 Intel 公司推出了更高档的 8080，它被叫做 8085。8085 基本上是重新包装的 8080，它增加了如下一些功能：加电初始化、向量中断、串行输入输出口和一个单正 5V 的电源支持。

到 8085 时，Intel 公司在微处理器上遇到了强有力的竞争对手，Zilog 公司的 8080 增强型 Z80 出现了，同 8080 设计完全不同的 Motorola 6800 和 MOS 技术公司的 6502 也出现了。Intel 公司没有继续在 8 位微处理器上纠缠，而是做了一个很大的跳越，在 1978 年推出了 8086。它是一种 16 位微处理器，它比 8080 处理数据的时间提高了 10 倍。而 8086 与 8080 软件是在汇编语言上兼容的，这表明一些少量的传输，8080 的现有程序可以在 8086 上反汇编和运行，而 8080 的指令集成了 8086 指令的子集。

利用了这种兼容性，Intel 公司又推出了 8086 的另一种芯片，它内部有 16 条数据线，但片外只有 8 条数据总线，起名为 8088。8088 被广泛用于 IBM PC，PC XT 和便携机上。1982 年，Intel 公司又推出了 80186，该片子集成了 8086 的处理器能力并且又加上了 15 个其他的支持电路，这个所谓“片子上的计算机”包含二个独立的 DMA 通道（直接访问内存通道，direct memsry access）用来为高速外设、磁盘服务，还包含一个可编程中断，如控制器。在软件方面它提供了 8086 的指令集，还有附加的指令，支持人们设计高级语言翻译器或编译器。

同年，Intel 公司推出了 80286 微处理器来控制 IBM PC AT。80286（简称 286）是 80186 的增强型。它提供了内存管理和保护的特殊功能，这些功能对于多用户应用是很重要的。

§1.3 80286CPU 的功能结构

80286 微处理器具有许多先进的特性，这些特性是为了实现高性能和满足多用户和多任务系统的需要而设计的。

许多微处理器的执行顺序是从内存读取第一条指令，解释第“一”条指令，执行第“一”条指令……直至执行最后一条指令。这样，在每一条指令执行完以后，CPU 必须等到下一条指令取出来后才能执行。它的工作顺序如下图：

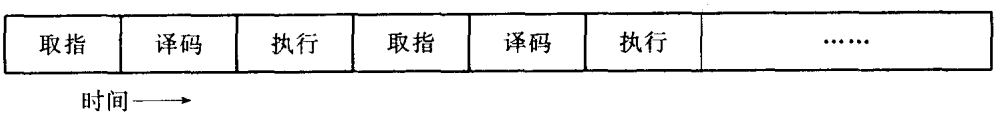


图 1-1

这样拖沓的一次一条指令减慢了处理器的速度。80286 解决了这些问题，在芯片上读取，解释和执行指令分别由芯片中专用的部分组成。就功能而言，80286 这种专用部分成两大块：总线接口单元 BIU（Bus Interface Unit）和执行单元 EU（Execution Unit）。其中 BIU 又分为三部分：总线部分 BU（Bns Unt），指令部分 IU（Instruction Unit）和地址部分 AU（Address Unit）。CPU 的功能结构可参见下图 1—2：

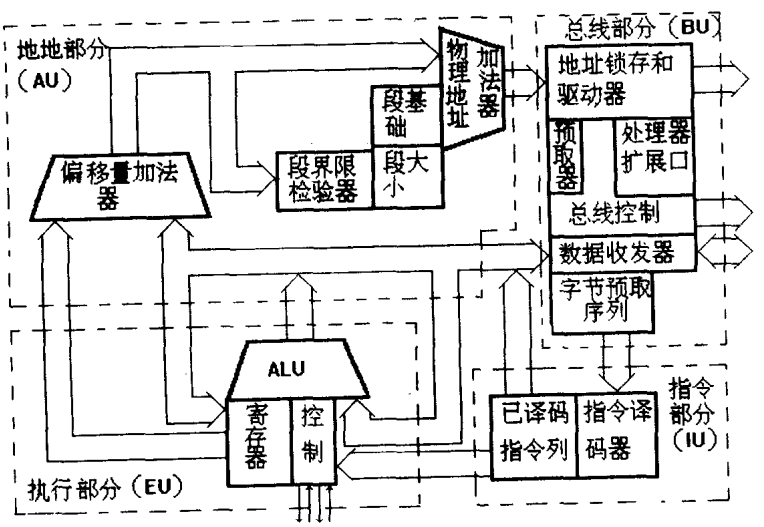


图 1-2 80286 内部结构

1. 总线部分（BUS Unit，BU）

总线部分是 80286 的传递载体，它的任务是从内存读出指令，在处理器和“外部”之间传输数据，总线部分将每条新指令放入一个代码队列，片子的指令部分可以访问这个队列。

2. 指令部分（Instruction Unit，IU）

指令部分从 BU 的代码队列中取出指令，解释它们，将代码存入指令队列或管道上(pipeline)，它相当于一个电子售货机。管道能存三个被解释了的指令。

3. 执行部分（Execution Unit，EU）

执行部分在内部 ROM 中的微码的控制下执行指令。当它快执行完当前指令时，ROM 发信号经 EU 去从指令队列中取下一条指令。

请注意在 80286 中，总线部分、指令部分、执行部分它们是彼此独立的并行操作，即 BU 和 IU 在 EU 执行前一条指令时也在工作。所以当 EU 需要另一个程序指令时，它一般在管道中取，它们之间的关系请见下图

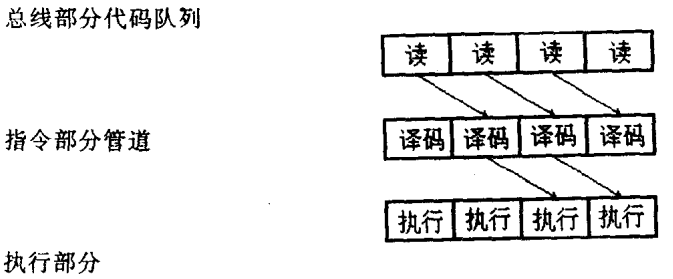


图 1-3　80286 中三部分的并行操作

4. 地址部分（Address unit，AU）

地址部分通过将虚地址转换成实地址，检查保护权限，起内存管理和保护作用。

由上述 80286CPU 的功能结构可以看出，当它的一条指令执行完以后就可以执行下一条指令，这样减少了 CPU 为取指令而耗费的等待时间，提高了 CPU 的利用率，加快了系统的运行速度。另一方面又降低了对与之配合的存贮器的存取速度的要求。

§1.4　8086 微处理器概况

现在让我们从总体上来了解一下 8086 微处理器。

1.4.1　操作方式

8086 有两种操作方式，其一是实地址方式，其二是虚地址方式（或叫保护方式）。

· 实地址方式

在实地址方式下，80286 操作基本上象一个增强了的 8086，它支持 8086 的所有指令，而它运行程序的速度却相当于 8086 的五倍。当我们打开计算机的电源开关，80286 开始在实地址方式下启动，并一直保持在这一方式下（除非程序有意地将方式改变为保护方式）。

· 保护方式

在保护方式下，80286 支持所有在实地址方式下的操作，并为数据保护和内存管理提供

了多方面的功能，最突出的是 80286 可以用虚地址技术访问很大的内存空间。它可以管理二种内存，一种是真正的物理地址空间，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 可以使用到的空间。这两个空间大小不一样，物理地址空间可以用到 16MB（16×10^6 字节），相当于 2^{24}字节，而虚地址空间可用到 2^{30}字节。我们说虚地址的概念是这样的：当一个程序需要访问的内存部分不在物理空间时，操作系统可以用 80286 的内存管理功能，将所需虚存部分换入物理地址空间。这种存贮交换编程者是看不见的，是由操作系统安排的，可以访问全部的 2^{30}字节地址空间。在 DOS3.X 中，IBM 提供了大约一兆字节可使用的内存，用户可将这一部分用做虚磁盘（或称为 RAM 磁盘）。要访问这部分，计算机可以立即切换到保护方式，传输数据，然后再变换到平时的实地址方式。这一存贮空间比硬盘快得多，而且也可以使用户得到相当大的存贮空间。

1.4.2 内部寄存器

在 80286 的内部是用一组 16 位的寄存器来存贮信息的，一共有 15 个，其中有 12 个是数据和地址寄存器，另外还有一个是指令指针寄存器，一个标志寄存器和一个机器状态字寄存器。我们通常把 12 个数据和地址寄存器分为三组，每组四个寄存器，它们是数据寄存器，指针寄存器，索引寄存器和段寄存器。

请见图 1-4

1. 数据寄存器

我们可以把数据寄存器用做 16 位或 8 位寄存器。当我们要处理 16 位数据时，它们分别被称为 AX、BX、CX、DX。如果我们要处理 8 位数据时，它们分别补称为 AH、BH、CH、DH 和 AL、BL、CL、DL，这里 L 和 H 分别代表 16 位寄存器中的低 8 位和高 8 位。

这里所有的寄存器都可作为通用寄存器使用，例如：

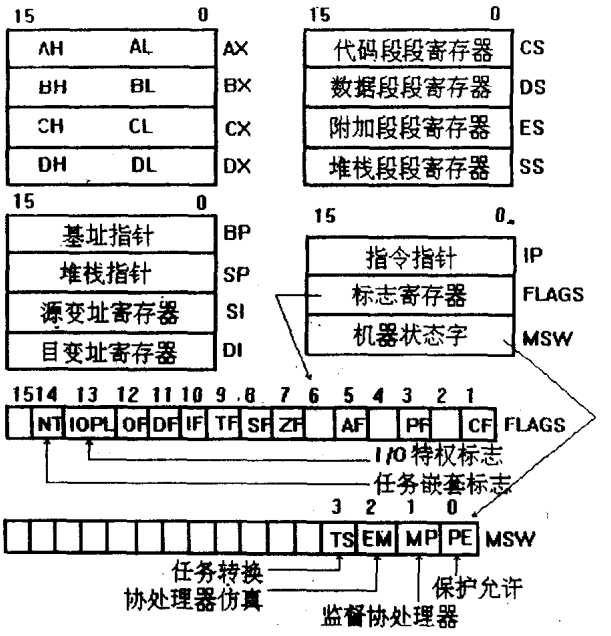


图 1-4 内部寄存器

（1）AX——累加器。在乘除、I/O 操作和串操作时使用。

AL 寄存器在这些同样操作的字节匹配和翻译及十进制运算操作时使用。

AH 寄存器在字节乘除时使用。

（2）BX——基址寄存器，通常用于内存地址数据。

（3）CX——计数寄存器。当处理循环操作时，用作重复计数器，当处理串操作时起基本计数器作用。CL 对于多位移动和循环操作起移位计数器作用。

（4）DX——数据寄存器。用于字乘、除操作。在输入输出操作中它提供端口号。

说明：

以上所介绍的数据寄存器是唯一可用作 8 位或 16 位寄存器，而其它寄存器只用作 16

位。

2. 段寄存器

80286 支持同时存取 4 个段的操作码模块。这 4 个段均为 16 位的寄存器，它们分别存放着 80286 的 4 个段的首地址。它们是：

· 代码段（CS）寄存器，指示出驻留在存贮器中的当前正在执行的程序段，80286 将（CS 乘 16）内容和指令指针内容相加计算出下一条指令的内存地址。

· 栈段（SS）寄存器，指示出当前栈段。栈通常用于保存中间结果以及子程序的调用。

· 数据段（DS）寄存器，指示出当前的数据段，它一般用来保存变量。

· 附加段（ES）寄存器，指示出当前的附加段，附加段一般用于串操作，也可以存取第二个并行操作的数据段。

在所选择的段内寻址一个单元是这样完成的：首先，在 CS、DS、SS 或 ES 这 4 个段寄存器中选择一个有效的段，然后提供一个 16 位的基地址偏移量。16 位的段地址和 16 位的偏移量组合起来形成 32 位虚拟地址指针的高半部和低半部。一旦选择了一个段，就只需要在指令中确定低 16 位的偏移量地址了。

根据 80286 正在操作的方式，对段地址有不同的解释。在实地址方式中，段寄存器包含了实际的物理地址。在保护方式中，段寄存器包含了虚拟存贮器的地址，而且需要把逻辑地址转换为物理存贮器地址。

3. 指针和索引寄存器

正如我们前面讲过的，在所选择的段内，任何给定单元的物理地址都可以通过将段地址和偏移量组合而获得。这个偏移量可以包含在任何指针、基址和变址寄存器中。我们可以通过组合 CS 中的段号和 IP 中的位移，计算出一条指令的地址。同理，也可以通过组合一个段寄存器中的段号和另一个寄存器的偏移量来访问其他段的数据。如要访问数据段，它从 DS 得到段号并从 BX 或一个索引寄存器得到位移；要访问栈段，它从 SS 中得到段号，从指针寄存器中（SP 或 BP）取得位移，它也可以通过 ES 中的一个段号访问附加段。

所以说，用堆栈段选择器（SS）和堆栈指针（SP）或基址指针（BP）这一对寄存器进行堆栈操作是非常容易的。从基址寄存器（BX）可以得到进入数据段（DS）和（ES）的偏移量，借助于源变址（SI）和目的变址（DI）寄存器与当前有效的数据段一起使用，可以进行更复杂的数据操作。

4. 指令指针

因为总线部分不知道程序执行的顺序，它总是从连续的内存地址中取指令，因此只有在程序执行传输一个新的不按顺序的地址时，EU 才必须等待指令从内存读出。此时，EU 必须等待 BU 清除代码队列及取出下一条指令。只有在这儿以后 80286 才同许多其他的微处理器一样等待每一条指令被读出。

正因为 80286 的独特的工作方式，Intel 公司的设计者们有意将它们的“下一个执行地址”寄存器同其他厂商的“下一次取址”寄存器区别开来，他们称之为指令指针（IP）而不称之为程序计数器（PC）。IP 中永远包含 80286 将要执行的下一条指令的位移。因为 IP 有这一功能，不要求对其内容进行算术运算。

5. 标志

我们常常希望程序可以在 80286 执行指令的基础上对一些结果进行判断。例如，当运

算结果为零时进行一种操作，当结果不为零时进行另一种操作等等。

16 位的标志寄存器指示了不同状态的情况，它们可以帮助我们进行判断。其中 6 位指示状态，3 位用来控制程序，另二位与保护状态有关。请见下示：

位15								位7	位6				位2		位0	
//	NT	IO	PL	OF	DF	IF	TF	SF	ZF	//	AF	//	PF	//	CF	标志
																指令指针
											ET	TS	EM	MP	PE	机器状态字

80286 的状态字和控制寄存器

说明：

（1）第 0 位，进位标志（CF）。当用 8 位或 16 位操作数完成的算术操作（如加法或减法）产生进位或借位时，进位标志（CF）置为 1，否则为 0。CF 也用于移位和循环指令，并包含了移出或循环移出寄存器的位。

（2）第 2 位，奇偶标志（PF）。主要用于数据通讯应用程序中，如果操作结果中有偶数个“1”，此位为 1，否则为 0。

（3）第 4 位，辅助进位标志（AF）。用于 BCD（压缩十进制数）算术运算，表示最低有效的 4 位 BCD 数值位是否产生了进位或借位，若有进位或借位，此位为 1，否则为 0。

（4）第 6 位，零标志（ZF）。如果运算结果为 0，此位为 1，否则为 0。

（5）第 7 位，符号标志（SF）。只有在对符号数操作时才有意义。如果算术、逻辑移位或循环操作产生负的结果，SF 为 1，否则为 0。也就是说，无论 8 位还是 16 位结果，SF 反映了结果最高一位的情况。

（6）第 8 位，陷井标志（TF）。若（TF）置位时，使 80286 处理器置为单步方式，从而允许对程序进行调试。

（7）第 9 位，中断允许标志（IF）。当（IF）置 1 时，允许外部中断；当它复位时，禁止外部中断。

（8）第 10 位，方向标志（DF）。如果（DF）为 0，SI 和/或 DI 自动增量，即 80286 向前处理一个字符串（向高地址，或从左向右）。如果（DF）为 1，SI 和/或 DI 自动减量，即 80286 向后处理一个字符串（向低地址，或从右向左）。

（9）第 11 位，溢出标志（OF）。在对符号数操作时是一个基本的错误指示器，它表示操作数是否产生了一个进位进入结果的最高有效位，但是并没有从最高有效位产生出一个进位，或者从最高有效位产生一个进位，但是并没有一个进位进入最高有效位。

例如，两个同符号数相加或两个异号数相减产生的结果在操作数中容纳不下时，OF 为 1，否则为 0，如果操作数最高位（符号）在一算术移位操作时改变则 OF 为 1，否则为 0。

OF 标志与 CF 标志相结合也可以指示乘法结果的长度，如果积的上半部非 0，OF 和 CF 为 1，否则它们全为 0。还有当一个除法操作产生的商溢出结果寄存器时，OF 为 1。

（10）第 12 位和第 13 位，特权标志（IOPL）。它是以前的微处理没有提供的新的标志，并且只有当微处理器处于保护方式时使用，两位的输入/输出特权标示（IOPL）用于保证指令只完成那些允许完成的操作。特权层共有四层，即 00 层 01 层，10 层和 11 层。00 层为