

80386.486

微机系统分析

朱传乃 编著



北京希望电脑公司

316
21

微机系统分析

朱传乃 编著

北京希望电脑公司

1992.6

版权所有
不许翻印
违者必究

维 修 服 务 消 息

希望公司长期以来作为AST, Houston Instrument等公司的特别代表, 已成为国内最大的销售、维修服务中心。为感谢广大用户的支持, 为广大用户服务, 我公司维修部决定对外开放, 为各界用户的下列产品提供专修特别服务。

- 一、AST公司的P286、386、486等各种微机及显示器。
- 二、HI公司的DmP-42、52、56、61、62等各种绘图机。
- 三、HI公司的TG1017、TG8024、TG8036等各种数字化仪。
- 四、各种20"大屏幕彩色显示器CD1920、MX-200、MX-210、NEC-5D等。

北京希望公司维修部

经理: 陈泽安

电 话: 2567818

BB机: 126—61134

★北京市新闻出版局

准印证号: 3156-90156

★订购单位: 北京8721信箱资料部

★邮 码: 100080

★电 话: 2562329

★传 真: 01—2561057

★乘 车: 320、302、332、路车
到海淀黄庄下车

★办公地点: 希望公司大楼一楼往里走
101房间

前 言

INTEL公司于1985年和1989年分别推出了80386和80486 CPU,这两种CPU同8086系列(简称86系列)的CPU在目标代码一级保持了向上的兼容性,都是高性能的第四代32位CPU。它们的共同特点是:①软件向上兼容。②面向多任务处理。③通用寄存器从16位扩充到32位。④运算速度大大提高。⑤具有自动数据总线大小功能。⑥地址总线扩充到32位。另外,80486同80386相比又有几点不同:①80486采用单倍的时钟频率,即在80486的CLK端加入的时钟频率就是其内部的处理器时钟频率。②80486内部包含有8K字节的数据和指令混合型高速缓存器,用于为频繁访问的指令和数据提供快速的局部存储。③在80486的内部包含有浮点运算部件(FPU),它实际上是增强型的80387,但其处理速度同80387相比提高了3~5倍。④同80386相比,80486的内部数据总线的宽度由32位扩充为64位,使指令周期加速。

1987年IBM公司推出了采用80386 CPU的PS/2系列微型计算机(采用80386 CPU的微型计算机均简称为386微机),其后,由不同厂家生产的386微机逐年增多,486微机系统也在不断涌现。相比之下,对386和486微机系统从总体上进行系统而全面介绍的资料却很少,这是广大从事系统开发和系统维修的用户所普遍感到的问题。编写本书的目的就是希望能为他们提供一点有益的帮助。本书主要围绕以下三个方面对386(486)微机系统进行分析:①80386(486)CPU的体系结构,工作原理,性能特点和用法。②386(486)微机系统的总体体系结构。③386(486)微机系统的主存储器的构成。在编写本书的过程中,李秋实,吴寒星,朱晓松,董彬彬等许多同志给予了大力的帮助,在此向他们表示诚挚的感谢!书中若有不当之处敬请读者批评指出。

编 者

目 录

第一章 概述	(1)
1.1 80386 CPU	(1)
1.1.1 80386 CPU的内部结构.....	(1)
1.1.2 80386 CPU的内部寄存器.....	(3)
1.2 协处理器80387.....	(5)
1.2.1 80387的内部寄存器	(5)
1.2.2 80386和80387.....	(6)
1.3 80486 CPU	(8)
1.3.1 80486 CPU的特点	(8)
1.3.2 80486 CPU的内部结构和内部寄存器.....	(9)
1.4 标志寄存器.....	(11)
1.5 控制寄存器.....	(13)
1.6 工作模式.....	(16)
第二章 存储器管理功能	(17)
2.1 虚拟存储器和物理存储器.....	(17)
2.2 描述符表和段寄存器.....	(19)
2.2.1 描述符表.....	(19)
2.2.2 段寄存器.....	(21)
2.2.3 对存储器的访问.....	(22)
2.3 描述符.....	(23)
2.3.1 数据段和代码段描述符.....	(23)
2.3.2 堆栈段描述符.....	(25)
2.4 别名 (ALIAS).....	(26)
第三章 保护功能	(29)
3.1 存储器保护功能.....	(29)
3.1.1 存储器的访问.....	(29)
3.1.2 修改段寄存器时的保护.....	(29)
3.1.3 对虚拟地址进行变换时的保护.....	(31)
3.1.4 因保护功能而产生的异常中断.....	(32)
3.2 特权级保护功能.....	(33)
3.2.1 OS和应用程序.....	(33)
3.2.2 修改DS, ES, FS和GS时的保护功能	(36)
3.2.3 修改SS时的保护功能.....	(37)
3.2.4 修改CS时的保护功能.....	(38)
3.2.5 特权级保护例外的代码段.....	(44)
第四章 中断和异常中断的处理	(46)

4.1 中断和异常中断	(46)
4.1.1 中断和异常中断的原因	(46)
4.1.2 中断描述符表IDT	(46)
4.2 保护模式下的中断处理	(47)
4.2.1 通过中断门和陷阱门的中断处理	(47)
4.2.2 错误代码	(48)
4.2.3 特权级保护	(50)
4.2.4 通过任务门的中断处理	(51)
4.3 中断向量的分配	(53)
第五章 多任务/多用户系统	(58)
5.1 多任务系统	(58)
5.1.1 多道程序	(58)
5.1.2 LDT和GDT	(60)
5.2 任务及任务的描述符	(62)
5.2.1 任务及其LDT	(62)
5.2.2 任务及其TSS	(62)
5.2.3 系统地址寄存器	(64)
5.3 任务转换	(65)
5.3.1 任务的设置	(65)
5.3.2 任务转换的过程	(65)
5.3.3 任务转换的方法	(68)
5.3.4 任务门	(69)
5.3.5 任务转换时的B位, NT位和反向链的变化	(70)
5.3.6 IRET/IRETD指令	(72)
5.3.7 任务转换时的特权级保护	(72)
5.3.8 描述符表的项目分类	(73)
第六章 80386/80486的指令	(75)
6.1 具有保护功能的指令	(75)
6.1.1 ARPL指令	(75)
6.1.2 LGDT, LIDT, SGDT和SIDT指令	(77)
6.1.3 LLDT, LTR, SLDT和STR指令	(77)
6.1.4 VERR和VERW指令	(78)
6.1.5 LAR和LSL指令	(78)
6.1.6 只有在特权级0才可以执行的指令	(79)
6.1.7 和IOPL有关的指令	(79)
6.2 新增加的指令	(81)
6.2.1 ENTER和LEAVE指令	(82)
6.2.2 ENTER指令的算法	(85)
6.3 其它的指令	(86)

6.3.1 CALL, JMP 和 中断指令	(86)
6.3.2 由保护模式向实模式的转移步骤	(89)
6.3.3 ASM 386和ASM 86的不同点	(91)
6.3.4 指令与标志的关系	(92)
第七章 页	(95)
7.1 P位和A位	(95)
7.2 线性地址	(96)
7.2.1 线性地址与物理地址的关系	(96)
7.2.2 从线性地址变换为物理地址的例子	(97)
7.3 目录项和页表项	(98)
7.3.1 目录项	(98)
7.3.2 页表项	(99)
7.4 页的保护功能	(99)
7.5 TLB	(101)
7.5.1 TLB的功能	(101)
7.5.2 TLB的测试	(103)
第八章 80286的软件和虚拟8086模式	(106)
8.1 80286的软件和80386	(106)
8.1.1 互换性	(106)
8.1.2 16位模块和32位模块	(107)
8.1.3 16位代码和32位代码的共存	(109)
8.1.4 通过门的控制转移	(111)
8.2 8086的软件和80386	(113)
8.2.1 80386的实模式	(113)
8.2.2 虚拟86模式	(114)
8.2.3 保护模式和虚拟86模式之间的转换	(115)
8.2.4 虚拟86模式下的中断和页功能	(117)
8.2.5 8086和80386的OS	(120)
第九章 调试寄存器和描述符中的D位	(122)
9.1 调试寄存器	(122)
9.1.1 线性地址调试寄存器 (DR0—DR3)	(122)
9.1.2 指定断点条件的调试寄存器DR7	(123)
9.1.3 断点状态调试寄存器DR6	(124)
9.2 指令的断点和RF标志	(125)
9.3 描述符中的D位	(125)
9.3.1 堆栈段描述符中的D位	(125)
9.3.2 代码段描述符中的D位	(126)
9.3.3 确定操作数大小的前缀指令66H	(127)
9.4 系统的初始化	(128)

9.4.1 CPU复位后的初始状态.....	(128)
9.4.2 向保护模式的转移.....	(129)
第十章 标准总线的体系结构.....	(131)
10.1 标准总线.....	(133)
10.2 ISA (PC AT) 总线.....	(134)
10.2.1 总线周期.....	(135)
10.2.2 ISA总线信号的特性和功能.....	(139)
10.3 MCA.....	(143)
10.3.1 MCA的连接器的.....	(143)
10.3.2 MCA信号的功能.....	(146)
第十一章 系统存储器.....	(152)
11.1 动态随机存取存储器 (DRAM) 片子的性能和特点.....	(152)
11.1.1 DRAM片子的存取方式.....	(152)
11.1.2 DRAM片子的刷新操作.....	(155)
11.2 系统存储器的构成.....	(159)
11.2.1 80386 CPU的引脚及其功能.....	(159)
11.2.2 80386的总线周期.....	(161)
11.2.3 DRAM控制器.....	(165)
11.2.4 SRAM和高速缓存器 (cache).....	(172)

第一章 概 述

INTEL公司于1985年和1989年分别推出了80386和80486 CPU, 这两种CPU和8086系列的CPU在目标代码一级完全保持了向上的兼容性, 都是高性能的第四代32位处理器。概括说来它们具有的特点如下:

① 已有的8086/88, 80186/188和80286的软件可以照样在80386和80486系列的微机上运行。

② 多任务处理容易。另外, 80386和80486系列的微机可以照样运行UNIX, XENIX, PC-DOS和MS-DOS等不同的操作系统。

③ 各种寄存器的容量从16位扩充到32位, 可以执行64位的数据运算, 32位 $\times$ 32位的乘法运算, 64位/32位的除法运算。并且, 进一步增强了位处理指令的功能。

④ 运算速度大大地提高。80386 CPU的时钟频率有12MHZ和16MHZ等多种, 在16MHZ时钟下, 执行速度达3~4MIPS (1MIPS=100万条指令/秒), 和CPU之间的数据传送速度为32M字节/秒。80486CPU的执行速度更快, 在相同的时钟频率下, 是80386的2~4倍。80486CPU在25MHZ时钟频率时, 执行速度高达20MIPS, 在33MHZ时钟频率时达27MIPS。

⑤ 具有自动数据总线大小 (automatic data bus sizing) 功能, CPU读/写数据的宽度不止有32位, 可以在32位和16位之间自由地进行转换。

⑥ 地址信号线扩充到32位, 可以处理 2^{32} 字节 (4G字节) 的物理存储器空间。如果利用虚拟存储器, 其存储空间达 2^{40} 字节 (64T字节)*。

⑦ 作为80386CPU的协处理器, 可以使用80287, 也可以使用高性能的具有32位数据总线的80387。80387的运算速度是80287的6~8倍。80486和80386不同, 在片子内部包含有和80387功能相同的FPU (浮点运算部件), 具有更高的运算速度。

本章将从总体上对80386CPU, 80387协处理器和80486CPU进行概括地介绍。

1.1 80386CPU

1.1.1 80386CPU的内部结构

大家知道, 8086/88CPU的内部由执行部件 (EU) 和总线接口部件 (BIU) 两个部分组成, 80286CPU的内部由总线部件 (BU), 指令部件 (IU), 执行部件 (EU) 和地址部件 (AU) 四部分组成。而80386CPU的内部结构如图1.1所示, 主要由六个部件组成, 这个六个部件是:

- ① 总线接口部件 (BIU: Bus Interface Unit)。
- ② 指令译码部件 (IDU: Instruction Decode Unit)。

* 1K字节=1024字节
1M字节=1024K字节
1G字节=1024M字节
1T字节=1024G字节

- ③ 指令予取部件 (CPU: Code prefetch Unit)。
- ④ 执行部件 (EU: Execution Unit)。
- ⑤ 段管理部件 (SU: Segment Unit)。
- ⑥ 页管理部件 (PU: Paging Unit)。

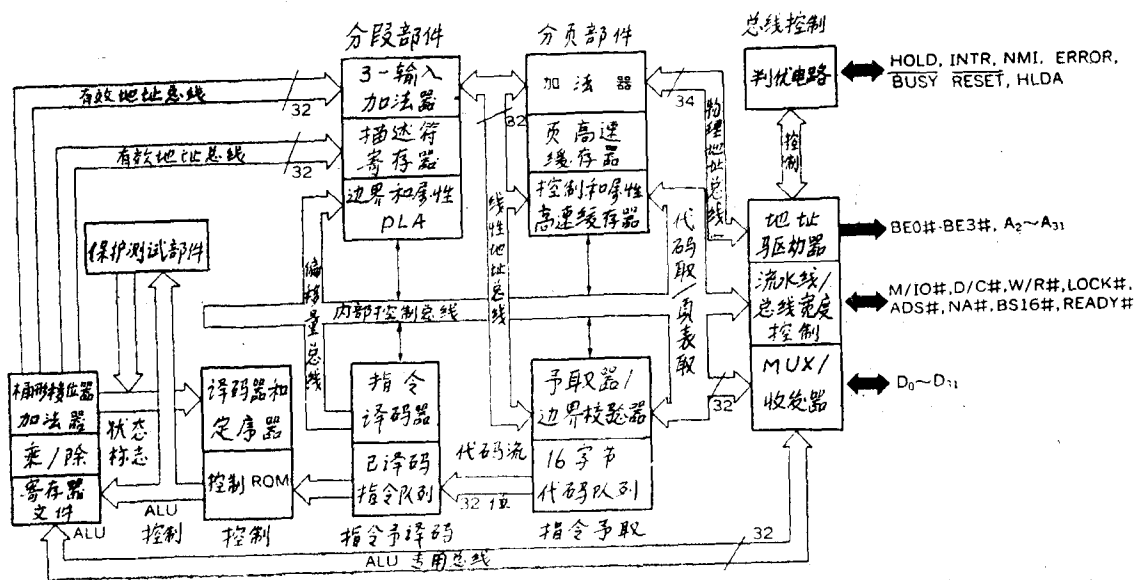


图1.1 80386CPU的内部结构

可以看到, 80386CPU 的内部结构和 80286CPU 基本相同, 主要增加了页管理部件。80386CPU 的这些部件在内部分别进行同步, 独立并行地进行操作, 实现了流水线化作业, 避免了顺序处理, 最大限度地发挥处理器的性能, 使总线的利用效率达到最佳状态。

80386的总线接口部件主要用来产生访问外部存储器和I/O口所需要的地址、数据和命令信号。另外,总线接口部件还控制同80387(或80287)协处理器的接口。大家知道,在8086/88和8087系统中,8087本身可以对存储器进行访问。而在80286和80287系统中,80287对存储器的访问,全部是通过80286总线部件内的协处理器接口进行的。即80287和存储器之间交换数据时,80286的作用相当于DMA控制器。在80386和80387(包括80287)系统中,80386总线接口部件和80286总线部件的功能相同。

80386的指令予取部件中包含有16字节的予取队列寄存器,当总线空闲时,由存储器读取指令存放到指令予取队列寄存器。80386的指令平均长度为3.5字节(24位~28位),所以,予取队列寄存器大约可以存放5条指令。

段管理部件用来把由指令指定的逻辑地址变换成为线性地址。如以后所述，逻辑地址是在程序中指定的虚拟地址。页管理部件的功能是把线性地址换算成物理地址。页管理部件是在80386CPU中新增加的部件，它是80386的一大特长。页管理部件同时又是可选择部件，如果不使用这个部件，80386的线性地址就是物理地址。

对80386来说,段管理部件和页管理部件一起都是存储器管理部件,理解和掌握这两个部件的工作原理对分析80386微机系统是最为重要的。

1.1.2 80386CPU的内部寄存器

如图1.2所示，80386CPU内部包含有如下的寄存器：

- ① 通用寄存器
- ② 标志寄存器
- ③ 指令指示器
- ④ 段寄存器
- ⑤ 系统地址寄存器
- ⑥ 调试寄存器
- ⑦ 控制寄存器
- ⑧ 测试寄存器

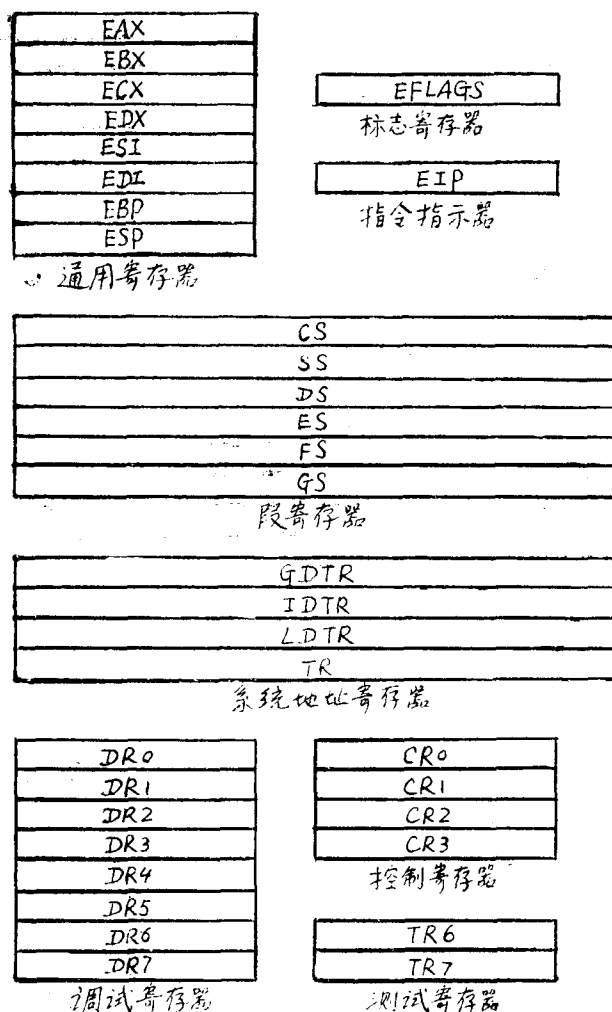


图1.2 80386CPU的寄存器

如果对图1.2的寄存器作进一步说明，则可以如图1.3和图1.4所示，分为扩充的8086寄存器和新增的寄存器。通用寄存器是32位的寄存器，在寄存器名字的前面附加字母E。这

些寄存器可以象8086（包括8088）的8位或16位寄存器一样进行使用。例如，AX是EAX的低16位，AH是AX的高8位等。

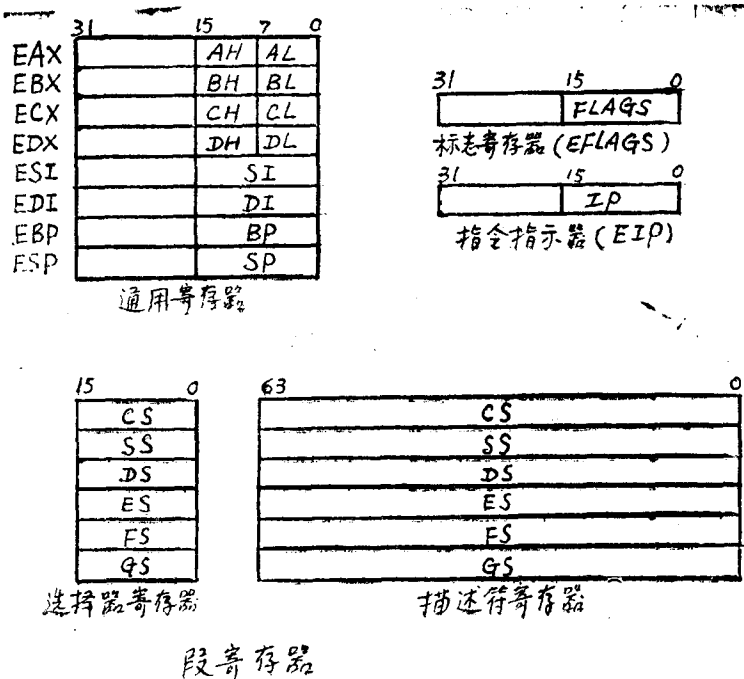


图1.3 扩充的8086寄存器

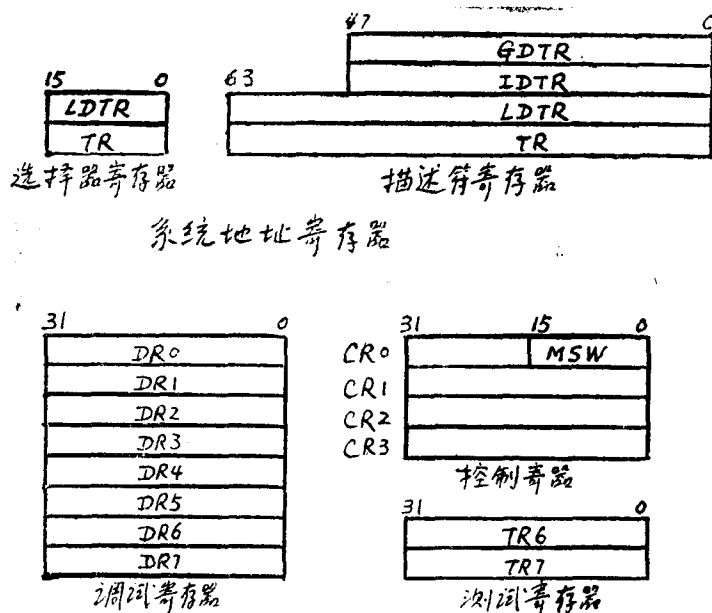


图1.4 新增加的寄存器

标志寄存器 (EFLAGS) 和指令指示器 (EIP) 都是32位的寄存器，按照使用方法，CPU也可以只使用这些寄存器的低16位 (FLAGS和IP)。

和8086相比,除CS, SS, DS, ES之外,80386又增加了FS和GS这两个新的段寄存器,以便减轻对ES和DS段寄存器的压力。80386中的段寄存器和8086中的段寄存器有所不同,这些段寄存器是由16位的选择器寄存器和64位的描述符寄存器构成。80386中的段选择器寄存器即相当于8086中的16位段寄存器。关于这些段寄存器的作用,在说明段管理部件的工作原理时,再作进一步介绍。

图1.4所示为新增加的寄存器,8086没有这些寄存器。系统地址寄存器也称作系统表寄存器,主要用来在保护模式下管理4个系统表,这4个系统表是:

- ① GDT (Global Descriptor Table; 全局描述符表)
- ② IDT (Interrupt Descriptor Table; 中断描述符表)
- ③ LDT (Local Descriptor Table; 局部描述符表)
- ④ TSS (Task State Segment; 任务状态段)

GDT, IDT和LDT这三个描述符表分别由GDTR (GDT寄存器), IDTR (IDT寄存器)和LDTR (LDT寄存器)进行指定。另外,存放任务环境的称作TSS的表由TR (任务寄存器)进行指定。GDTR和IDTR是由32位基地址字段和16位边界字段构成的48位寄存器,即使在实模式下,为了对保护模式进行初始化设置,也可以对GDTR和IDTR进行访问。另外,LDTR和TR与段寄存器的构成相同,由16位的选择器寄存器和64位的描述符寄存器组成。这两个寄存器只能在保护模式下使用,由程序可访问的部分只限于选择器寄存器。通过指令(LLDT, LTR)或任务转换,可以把由各自的选择器寄存器所指定的LDT描述符或TSS描述符自动地加载到LDTR或TR的描述符寄存器。

80386CPU有4个控制寄存器CR0~CR3, CR1未用, CR3用于提供页目录的基地址, CR2用于提供页故障线性地址。CR0寄存器的低16位可以作为机器状态字处理。CR0寄存器各控制位的定义在本章介绍了80486CPU之后再作进一步说明。CR0~CR3都是与80386的页功能有关的寄存器,在介绍80386的页功能时,还要说明它们的功用。

调试寄存器DR0~DR7主要用来设置程序的断点,测试寄存器TR6和TR7也是与页功能有关的寄存器,在后面的章节里要专门介绍调试寄存器和测试寄存器的功用。

1.2 协处理器80387

协处理器80387又称数值运算处理器,它和80287一样在内部包含有8个80位的运算寄存器。增加数值运算处理器是为了从体系结构上扩充80386的功能。

1.2.1 80387的内部寄存器

80387的内部寄存器如图1.5所示。运算寄存器用于存放进行运算的数据,由8个80位的堆栈寄存器组成。进出堆栈寄存器的数据和进出存储器堆栈的数据按同样的方式进行。图1.5的状态寄存器中的TOP字段(位11~13)指示现在的堆栈寄存器的顶是哪个寄存器。用FLD指令等向堆栈寄存器加载数据时, TOP被减1,把数据送到由TOP字段指示的新的堆栈寄存器。反之,用FST指令等把堆栈寄存器中的数据保存到存储器时,是把现在由TOP字段指示的堆栈寄存器中的内容保存到存储器,然后再把TOP字段加1。另外,在进行运算时,把TOP字段指示的堆栈寄存器用作累加器,运算结果是存入到由TOP字段指示的堆栈寄存器中。控制寄存器是控制80387运算的寄存器,由控制寄存器的位决定运算错误的处理,运算的精度,数值的舍入,以及无限大的处理方法等。

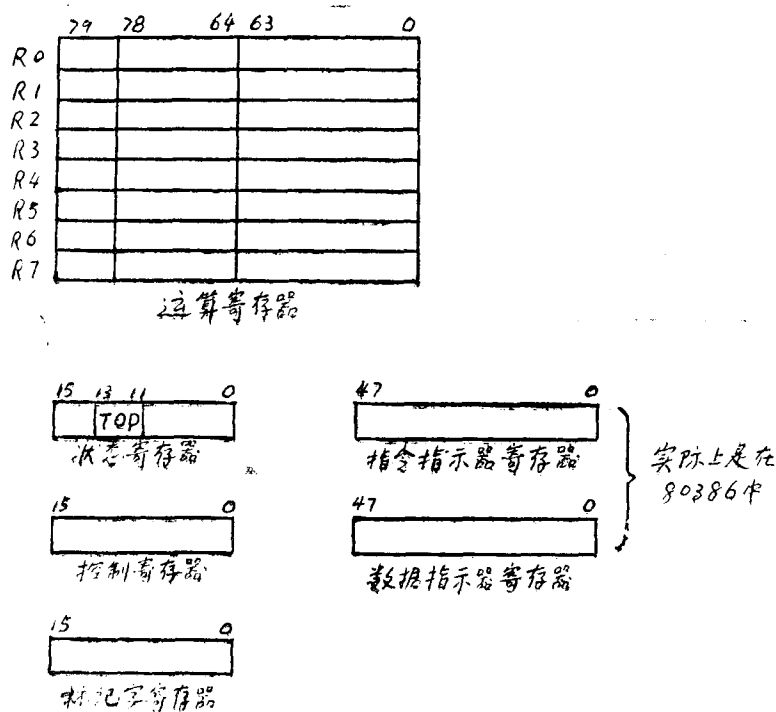


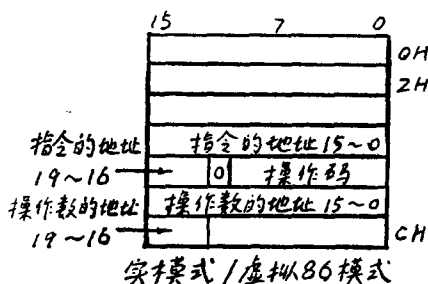
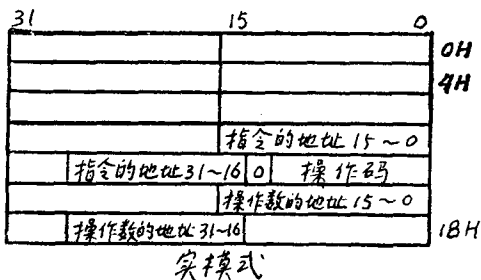
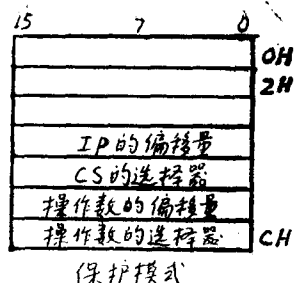
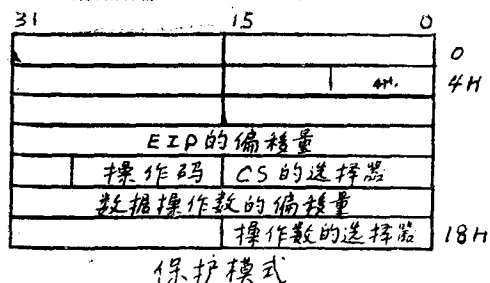
图1.5 80387的内部寄存器

状态寄存器除TOP字段以外，还有表示运算结果的状态位（运算结果是否为0，是否产生溢出/下溢等）。

标记（tag）寄存器表示R0~R7各堆栈寄存器的内容（是否输入正确数值的数据，是否处于空闲状态等）。指令指示器寄存器表示80387指令的地址和操作码。当80387指令的操作数是存储器操作数时，数据指示器寄存器表示操作数据的地址。用FLDENV, FSETENV, FSAVE和FRSTOR指令，可以把指令指示器寄存器和数据指示器寄存器中的内容保存到存储器。按照80386是工作在实模式，还是保护模式，或是虚拟86模式，以及按照操作数的属性是32位，还是16位，被保存内容的格式会有所不同。图1.6所示为指令指示器寄存器和数据指示器寄存器中的内容被保存到存储器中的格式。80386在保护模式下操作数的属性是32位时，指令和数据的地址是用选择器和32位的偏移量所表示的逻辑地址，操作码也保存在存储器中。80386在保护模式下操作数的属性是16位时，指令和数据的地址是用选择器和16位的偏移量表示的逻辑地址，操作码不保存在存储器中，和80287在80286的保护模式下具有相同的格式。80386在实模式下操作数的属性是32位时，指令和数据的地址不是逻辑地址，而是32位的物理地址。另外，操作码也保存在存储器中。80386在实模式下操作数的属性是16位时，指令和数据的地址是20位的物理地址，操作码同样是保存在存储器中。在虚拟86模式下，除了指令和数据的地址是线性地址以外，和实模式下操作数的属性是16位时的情况相同。

1.2.2 80386和80387

指令指示器寄存器和数据指示器寄存器不是在80387中，实际上是在80386中。但是，这



(a) 32 位的环境

(b) 16 位的环境

图1.6 指令指示器寄存器和数据指示器寄存器

些寄存器能够用ESC（80387指令的总称）指令进行访问，如同是80387中的寄存器一样进行处理。

80387的指令由80386进行读取，再通过I/O口地址传送给80387。80387访问存储器中的数据，也是由80386生成地址，进行读/写，再通过I/O口对80387进行数据的输入/输出。

为了在80386和80387之间进行数据的输入/输出，80386把800000F8H~800000FFH的I/O口地址分配给80387。

由80386向80387传送指令以后，80387执行该条指令，这时80386和80387可以独立地对指令进行加工。但是，80387在执行指令时，80386不能向80387传送新的80387的指令，这就要求80386要与80387进行同步。为此，80386使用了BUSY引脚。80387在执行ESC指令期间向80386的BUSY引脚供给低电平。当80386取到ESC指令时，要对BUSY引脚上的信号进行检查。如果该信号为低电平，则80386不向80387传送指令，等待它变成高电平。BUSY信号变成高电平之后，80386可以传送指令，而80386在传送指令之前还要对ERROR引脚进行检查，如果ERROR引脚的信号为低电平，则产生协处理器异常中断（中断类型码16）。但是，也有一部分访问存储器的ESC指令（如FNSAVE指令等），不对BUSY和ERROR引脚进行检查。

WAIT指令不是ESC指令，而是如同ESC指令一样，是用来检查BUSY和ERROR引脚信号的指令。如果执行WAIT指令，BUSY引脚是低电平时，80386在BUSY引脚变为高电平之前，不执行下一条指令。如果BUSY变为高电平，则对ERROR引脚进行检查，如果ERROR引脚为低电平，则产生协处理器异常中断。如果ERROR引脚为高电平，则执行WAIT指令的下一条指令。

1.3 80486CPU

1.3.1 80486CPU的特点

INTEL公司于1989年4月推出了32位的80486CPU，和80386相比，在相同的工作频率下，其处理速度提高了2~4倍。因此，人们把80486称作超级32位CPU。它的主要特点如下：

① 通过采用RISC (Reduced Instruction Set Computer; 缩减指令系统计算机) 技术，使芯片上的不规则的控制部分减少，指令能以较短的周期执行。同时，把以前的微代码控制改为用布线逻辑直接控制，进一步缩短可变长指令的译码时间，达到基本的指令可以用一个时钟周期完成，性能大大提高。

另外，80386在结构上要求外部时钟必须是CPU时钟频率（或称处理器时钟频率）的两倍。以目前推出频率为40MHZ的80386CPU为例，这就是说在它的CLK2端必须加入一个由80MHZ的振荡器提供的时钟频率。这么高的频率无疑会增加电路设计的困难。而80486采用的是单倍的时钟频率，即在80486的CLK端加入的时钟频率，就是它内部的处理器时钟的频率，因此可以大大增加电路的稳定性。

② 80486芯片内部包含有8K字节数据和指令混合型的高速缓存器，用于为频繁访问的指令和数据提供快速的局部存储。高速缓存器系统截取80486对内存的访问，对所需要的数据是否驻留在高速缓存器进行查询。如果数据或指令在高速缓存器中，这就称作“命中 (HIT)”。每当“命中”发生，则可以不插入等待状态就把数据或指令取回。如果“未命中 (MISS)”，则把内存询问返回给系统，并从主存读取数据或指令以进行弥补。一个设

表1.1 80486和80386CPU基本指令执行的时钟数

	80486		80386
	外速缓存器 命中	高速缓存器 未命中	零等待
MOV reg, mem	1	3	4
MOV mem, reg	1	1	2
MOV reg, reg	1	—	2
PUSH reg	1	1	2
POP reg	1	3	4
条件转移(未转移)	1	1	3
条件转移(转移)	3	5	9.25*
无条件转移	3	5	9.25*
CALL	3	5	9.25*
RET	5	7	12.25*
ALU reg, reg	1	—	2
ALU reg, mem	2	4	6
ALU mem, reg	3	5	7
Shift/Rot reg	2	—	3
Shift/Rot mem	4	6	7

* 平均时钟数

计成功的高速缓存器系统将会产生很高的“命中”率。由于“未命中”的比率很低，这实际上就把系统的等待状态减少到零。高的“命中”率，也将导致一个很低的系统总线占用率，这就使系统总线有更多的时间用于其它控制。

从技术的观点来看，对80486这种超级CPU最好配备64K~256K字节的高速缓存器。但是，实践表明，当高速缓存器的容量超过32K字节时，其命中率增加缓慢。在80486的芯片上高速缓存器采用4路成组相关方式，内部总线的宽度为16字节（128位），所以效能特别强。因此，再考虑对80486系统外加高速缓存器，其效率增加并不明显。表1.1列出了80486和80386CPU基本指令执行的时钟数，以示比较。

③ 在80486芯片的内部包含了增强型80387协处理器，在80486内部称作浮点运算部件（FPU）。FPU和80387完全兼容。由于FPU功能扩充，并且是在80486芯片的内部，使引线缩短，因此其处理速度和80387相比提高了 $\varepsilon \sim 5$ 倍。表1.2列出了80486和80386加80387执行ESC指令的时钟数，以示比较。

表1.2 80486CPU的FPU和80386CPU加80387执行ESC指令的时钟数

ESC	80486	80386+80387
	高速缓存器命中	零等待
FLD 32位	3	20
	64位	25
FST 32位	7	44
	64位	45
FADD 32位	7*	24—32
	64位	24—37
FSUB 32位	7*	24—32
	64位	29—37
FMUL 32位	8*	27—57
	64位	32—57
FDIV 32位	70*	89
	64位	94

* 和CPU核心部分并行动作时的情况

④ 和80386相比，80486的内部数据总线的宽度为64位，而80386内部数据总线的宽度为32位，这也是80486缩短指令执行周期的一个原因。

⑤ 其它。80486的集成度为120万晶体管，芯片大小为 $16 \times 11 \text{mm}^2$ ，采用CMOS工艺。168个引脚，PGA封装。工作时钟频率有25MHZ，33MHZ。寄存器之间的加法时间为1个时钟周期等。

1.3.2 80486CPU的内部结构和内部寄存器

图1.7所示为80486CPU的内部结构框图。可以看到，除了高速缓存器管理部件和浮点运算部件以外，和80386的内部结构基本相同，还包含如下七个部件：

- ① 总线接口部件
- ② 指令予取部件
- ③ 指令译码器