1　80x86 的结构

1.1　80x86 的功能结构

8086、80286、80386 和 80486 的功能结构分别如图 1-1、1-2、1-3 和 1-4 所示。

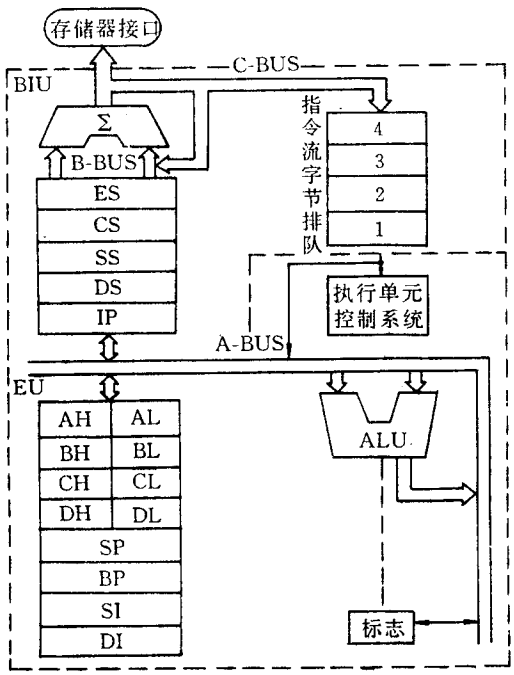


图 1-1　8086 的功能结构

8086 已经在 8 位微处理器的基础上前进了一步，把执行单元 EU 与总线接口单元 BIU 分开，从而在执行指令的同时(若不取操作数的话)可以取下一条指令，具有了指令流水线的功能，如图 1-5 所示。

80286 比 8086 前进了一大步，除了保留执行单元 EU 和总线接口单元 BIU 外，由于要支持保护虚地址方式下的 16M 内 存寻址功能，支持描述符表示的寻址机制，就扩展了一个地址单元 AU；同时在 8086 的指令流水线的基础上，增加了指令预取队列(3 条指令)，扩展为指令单元 IU。

80386 除了把指令预取队列扩展为 16 个字节，把地址单元的地址扩展为 32 位，可直接寻址 4G 字节外，最重要的扩展是增加了片内的存储管理单元 MMU，它能实现分页机制，把逻辑地址和物理地址分为大小相等(每页 4K 字节)的若干页，通过两级页表(页目录表和

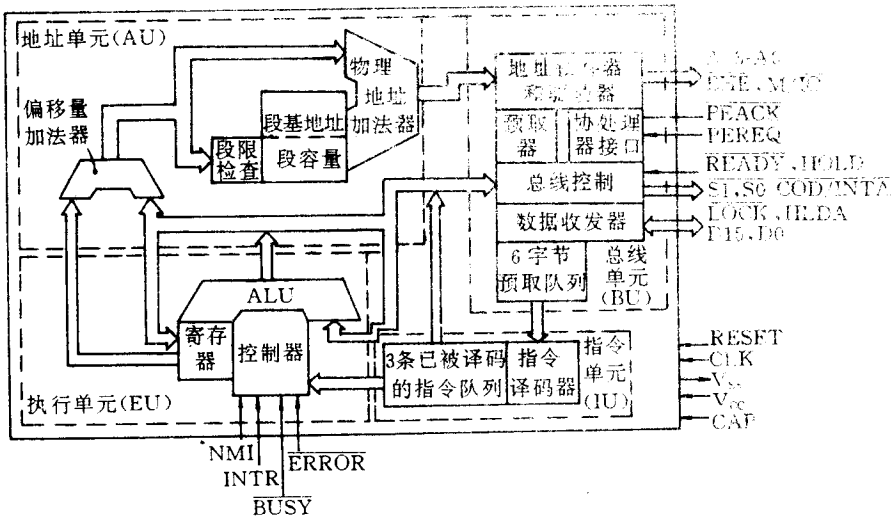


图 1-2　80286 的功能结构

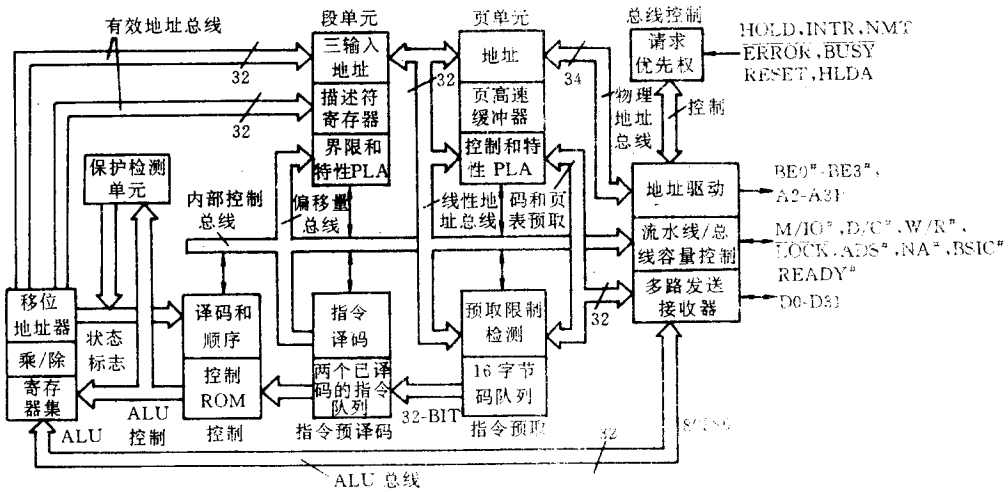


图 1-3　80386 的功能结构

页表)实现逻辑地址与物理地址的映射,能实现缺页中断,从而实现了虚拟存储器管理。

80486 的结构在多个方面有很大的发展,但最突出的是,它是 80386 和 80387(浮点运算协处理器)和 8K 字节的 Cache(指令与数据 cache)的结合。从而在功能上有了极大的提高。80486 的主 CPU 功能与 80386 一致,有关协处理器的内容我们在第二章中介绍。

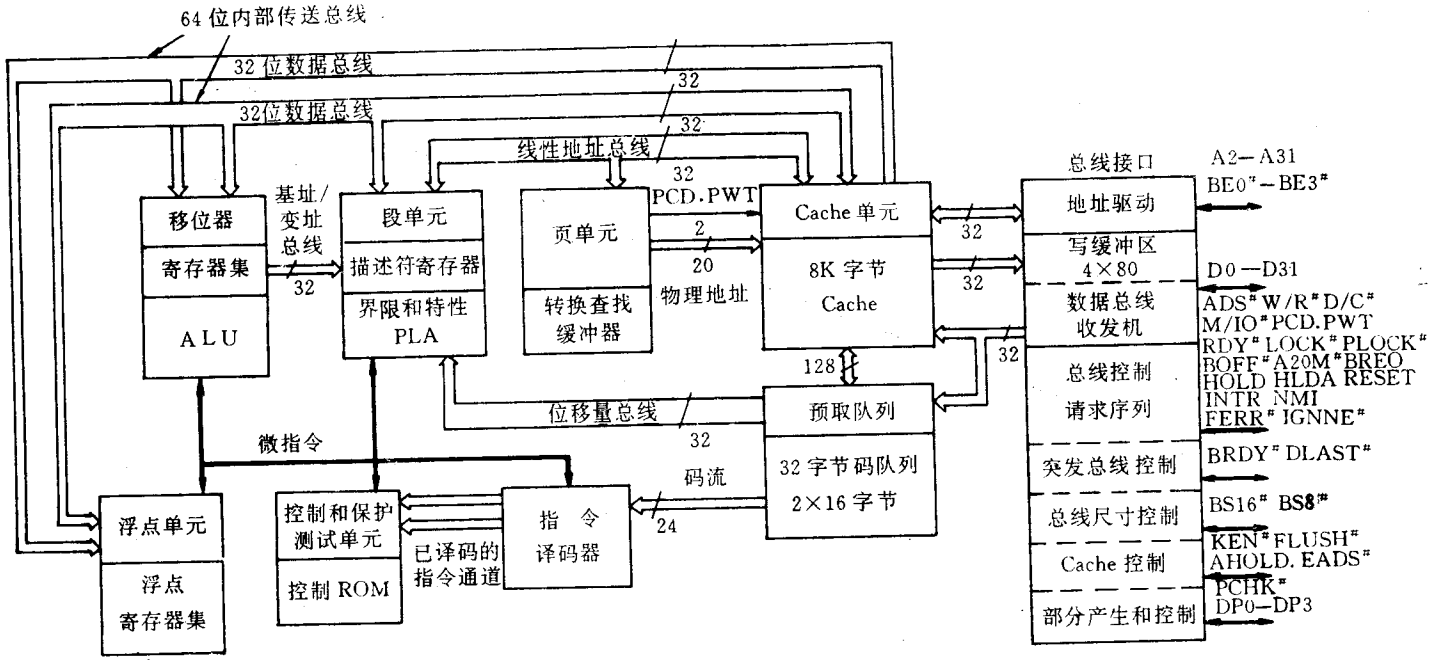


图 1-4　80486 的功能结构

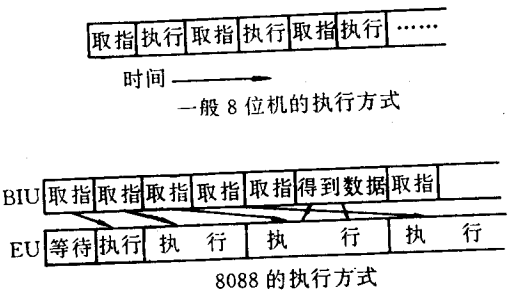


图 1-5　8086 中的指令流水线

1.2　80x86 的寄存器结构

8086、80286、80386 和 80486 的寄存器结构分别如图 1-6、1-7、1-8 所示。

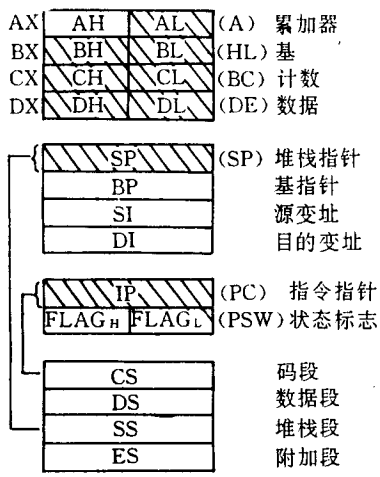


图 1-6　8086 的寄存器结构

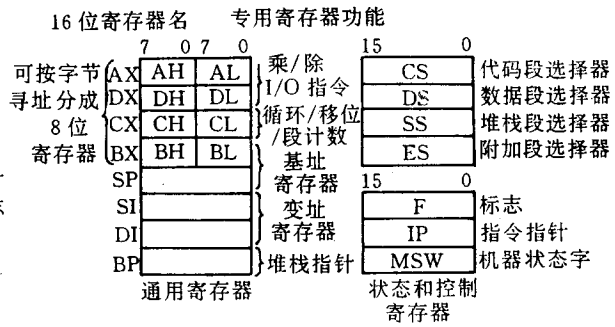


图 1-7　80286 的寄存器结构

8086 和 80286 的通用寄存器 AX、DX、CX、BX、SI、DI、BP 和 SP 是 16 位的，其中 AX、DX、CX 和 BX 都可以分为两个 8 位寄存器 AH、AL，DH、DL，CH、CL，BH、BL 使用。

8086 和 80286 中，指令指针 IP 和标志寄存器 FLAG 也都是 16 位的。

8086 和 80286 中，都有 4 个 16 位的段寄存器 CS、DS、SS 和 ES。

80286 比 8086 多了一个 16 位的机器状态字 MSW。

在 80386 和 80486 中，通用寄存器都扩展为 32 位，成为 EAX、EDX、ECX、EBX、ESI、EDI、EBP 和 ESP。它们的低 16 位分别为 8086 和 80286 的通用寄存器 AX、DX、CX、BX、SI、DI、BP 和 SP，可以单独使用。而且 AX、DX、CX 和 BX 都可以分为两个 8 位的寄存器使用。

在 80386 和 80486 中，指令指针和标志寄存器也都扩展为 32 位的 EIP 和 EFLAG，但它们的低 16 位仍可单独使用。

80386 和 80486 的段寄存器仍是 16 位的，但增加了两个数据段寄存器 FS 和 GS。

在 80386 和 80486 中，已经把 80286 中的机器状态字 MSW，扩展为三个 32 位的控制寄存器 CR0、CR2 和 CR3。此外，在 80386 和 80486 中，还有 6 个排错寄存器 DR0、DR1、DR2、

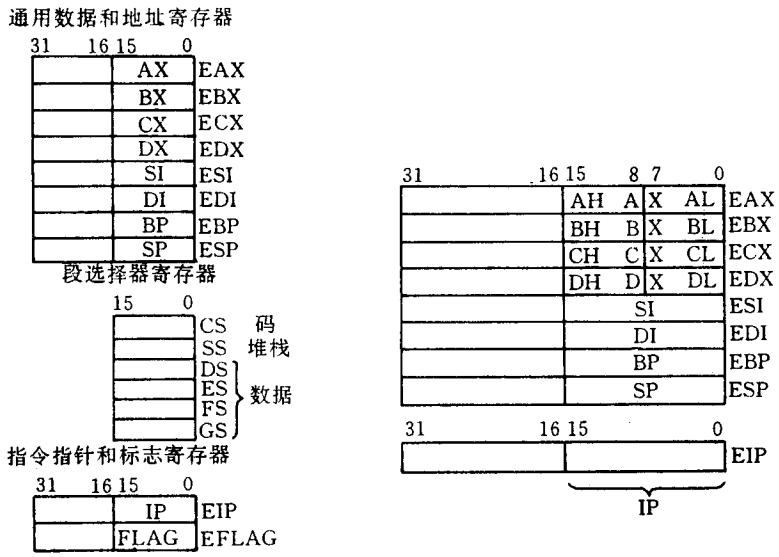


图 1-8　80386 的寄存器结构

DR3、DR6 和 DR7；80386 中有两个测试寄存器 TR6、TR7；在 80486 中有 5 个测试寄存器 TR3、TR4、TR5、TR6 和 TR7。

一、标志寄存器

8086、80286、80386 和 80486 的标志寄存器如图 1-9、1-10、1-11 和 1-12 所示。

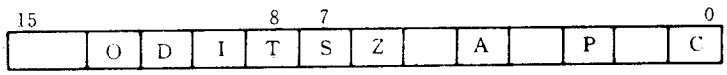


图 1-9　8086 的标志寄存器

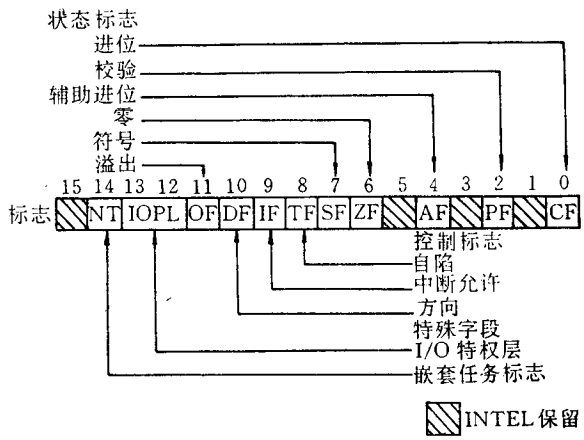


图 1-10　80286 的标志寄存器

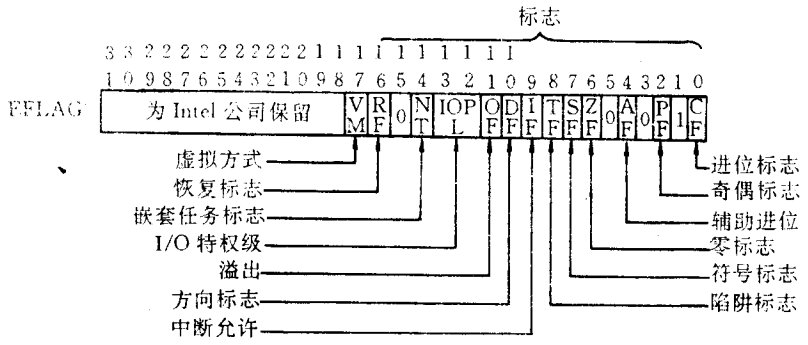


图 1-11　80386 的标志寄存器

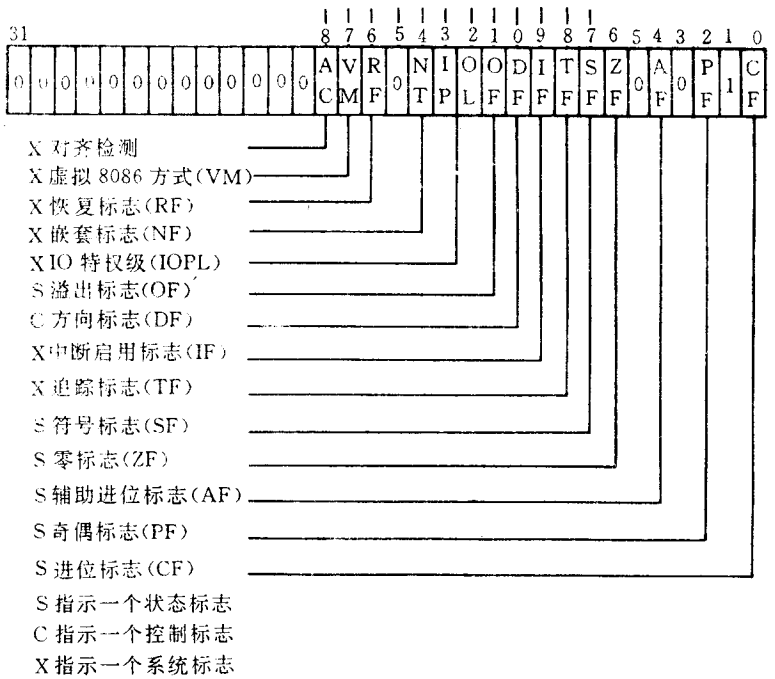


图 1-12　80486 的标志寄存器

在 8086 中共有 9 个标志，其中

1. 进位标志 C(Carry Flag)

当结果的最高位(字节操作时的 D_7 或字操作时的 D_{15})产生一个进位或借位，则 C=1；否则为 0。这个标志主要用于多字节的加、减法运算。移位和循环指令也能够把操作数(在寄存器或某一存储单元中)的最高位(左移时)或最低位(右移时)放入标志 C 中。

2. 奇偶标志 P(Parity Flag)

若操作结果中"1"的个数为偶数，则 P=1；否则 P=0。这个标志可用于检查在数据传送过程中是否出现错误。特别是在串行通信中，为了提高传送的可靠性，常采用奇偶校验。利用 P 标志可进行奇偶校验，或产生奇偶校验位。

3. 辅助进位标志 A(Auxiliary Carry Flag)

在字节操作时，由低半字节(一个字节的低 4 位)向高半字节进位或借位，或在字操作时，低位字节向高位字节进位或借位，则 A=1；否则为 0。这个标志用于十进制算术运算指令中。

4. 零标志 Z(Zero Flag)

若运算的结果为 0，则 Z=1；否则 Z=0。

5. 符号标志 S(Sign Flag)

S 的值与运算结果的最高位相同。即结果的最高位(字节操作时为 D_7，字操作时为 D_{15})为 1，则 S=1；否则 S=0。

由于在 80c6 中，符号数是用补码表示的，所以 S 标志反映了结果的符号：0=正，1=负。

6. 溢出标志 O(Overflow Flag)

在算术运算中，带符号数的运算结果，超出了 8 位或 16 位带符号数能表达的范围，即在字节运算时大于+127 或小于-128；在字运算时大于+32767 或小于-32768，则此标志置位；否则复位。INTO 一个任选的溢出中断指令，在溢出情况下能产生中断。

以上 6 个标志为状态标志，反映了操作结果的状态。

以下三个标志为控制标志。

7. 方向标志 D(Direction Flag)

若用指令置 D=1，则串操作指令就成为自动减量指令，也就是从高地址到低地址或是从"右到左"来处理串；若使 D=0，则串指令就成为自动增量指令。

8. 中断允许标志 I(Interrupt-enable Flag)

若用指令使 I=1，则允许 CPU 响应外部的可屏蔽中断请求；若使 I=0，则屏蔽上述的中断请求。但 I 标志位状态，对于外部的非屏蔽中断请求，或内部产生的中断不起作用。

9. 追踪标志 T(Trap Flag)

若置 T=1，则使 CPU 进入单步方式，以便于调试。在这种方式中，CPU 在每条指令执行以后，产生一个内部的中断，允许程序在每条指令执行完以后进行检查。若置 T=0，则 CPU 执行指令后不产生内部中断。

80286 中标志寄存器仍是 16 位，它保留了 8086 的所有 9 个标志位，还增加了两种(3 位)系统标志。

I/O 特权标志 IOPL(两位)

它规定了能使用 I/O 敏感指令的特权级，在 80286 以上的处理器中，有一部分指令如 CLI(关中断指令)、STI(开中断指令)、IN(输入)、INS(输入串)、OUT(输出)和 OUTS(输出串)为 I/O 敏感指令。IOPL 的值(可为 0—3)规定了能执行这些指令的特权级(在 80286 以上的处理器中有 0—3 共 4 个特权等级，0 级特权最高，3 级特权最低)。在 IOPL 的值确定以后，特权高于 IOPL 的程序能执行 I/O 敏感指令；而特权低于 IOPL 的程序，若企图执行 I/O 敏感指令，则会引起异常。

嵌套标志 NT

在 80286 以上的处理器上，允许执行多任务，故任务可以切换也可以嵌套。NT=1，表示当前执行的任务嵌套于另一个任务中，执行完该任务后，可以用 IRET 指令返回到原来的任务；在 NT=0 时，则不能。详见后面任务切换部分的说明。

在 80386 中，除了保留 80286 的所有标志位外又增加了两个标志位，而且标志寄存器扩展为 32 位。扩展的两个系统标志为：

虚拟 8086 方式标志 VM

在 80386 的保护虚地址方式下，为了能在多任务系统中执行 8086 的任务，设置了虚拟 8086 方式。在 80386 处于保护虚地址方式时，若使 VM 位置位，则 80386 就进入了虚拟 8086 方式。VM 位只能在保护方式下由 IRET 指令(若当前的特权级＝0)或在任何特 权级下由任务切换设置。VM 位不受 POPF 指令的影响，PUSHF 指令总是使此位清零。

恢复标志 RF

此标志是与调试寄存器的断点或单步操作一起使用的。在断点处理之前，在两条指令之间对该位进行检查。若 RF 位置位，则在下一条指令执行期间不处理任何调试故障。

在 80486 中，包括了 80386 的所有标志位，还增加了一个地址对齐检测标志。

地址对齐检测标志 AC。

当置 AC 位为 1 时，若发现地址不对齐就会产生异常。地址不对齐是指：若访问一个字时，地址为奇地址；若访问双字时，地址不处在双字边界上；若访问 8 字节操作数时，地址不对齐在 64 位的边界上。但不对齐故障只是在特权级 3 的程序运行时才会产生。在特权级 0、1、2 运行的程序，忽略 AC 标志的设置。

在 80486 中，各种数据类型地址对齐的要求如表 1-1 所示。

表 1-1　数据类型对齐要求

存　储　器　访　问	对齐(字节边界)
字	2
双字	4
单精度实数	4
双精度实数	8
扩展精度实数	8
选择子	2
48-位段指针	4
32-位段指针	2
32-位浮点指针	4
48-位“伪描述符”	4
FSTENV/FLDENV 保存区	4/2(操作数大小)
FSAVE/FRSTOR 保存区	4/2(操作数大小)
位串	4

二、控制寄存器

80286 比 8086 多了一个控制寄存器——16 位的机器状态字寄存器，如图 1-13 所示。

其中各位功能如下：

PE(第 0 位)：保护方式允许位。当 PE＝0 时，CPU 处在实地址方式；当 PE＝1 时，CPU 处在保护虚地址方式。在 CPU 复位时，PE＝0。

MP(第 1 位)：监督协处理器位，它用于与 TS 位一起决定 WAIT 指令是否引起异常。若

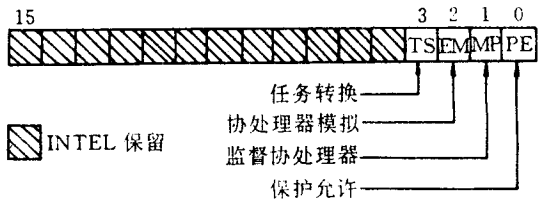


图 1-13 80286 中的机器状态字

TS=1,MP=1,则遇到 WAIT 指令就引起异常 7。TS 位是在任务切换时自动设置的。MP 位对协处理器指令不起作用。

EM(第 2 位):模拟协处理器位,决定协处理器指令能否被执行。EM=0,由协处理器执行协处理器指令;EM=1,则协处理器指令将引起异常。显然,系统中有协处理器时应置 EM=0。

TS(第 3 位):任务切换位。凡发生任务切换,则 TS=1,此后若执行协处理器指令就引起设备不可用(DNA)异常(异常号 7)。若 TS=1 又 MP=1,则 WAIT 指令也将引起 DNA 异常。

TS 位的存在可以把保护协处理器状态的操作,延迟到执行协处理器指令的时刻。在任务切换时,正常的任务状态段不保留协处理器的状态,因为在一个进程中可能不执行协处理器指令,所以没有必要在任务切换时都保护协处理器的状态,而只是置 TS=1。而当新任务要执行协处理器指令时,才产生异常 7,就可以在异常 7 的处理程序中再保护协处理器的状态。设置 MP=1 的目的是,在 TS=1 时,不只协处理器指令会引起异常,WAIT 指令也能引起异常。

在 80386 中,有一组控制寄存器:CR0、CR2 和 CR3(CR1 保留),如图 1-14 所示。

其中,CR0 是 80286 的机器状态字 MSW 的扩展,为一个 32 位的寄存器。其中低 4 位的意义与 80286 中相同。新增了 ET 和 PG 位。

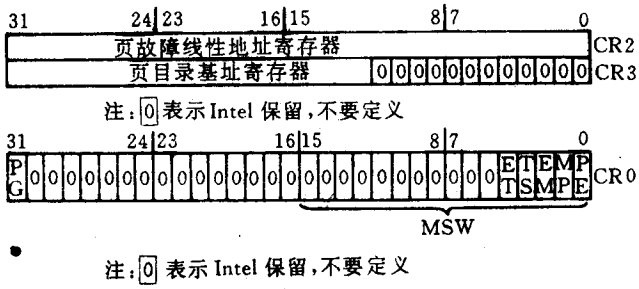


图 1-14 80386 中的控制寄存器

ET(第 4 位):处理器扩展类型位:ET 位反映了所扩展的协处理器的类型,ET=0 为 80287,ET=1 为 80387。所扩展的协处理器的类型,也可以在 80386 复位后,从 ERROR# 引脚的电平来确定。在复位后此信号有效(低电平),则表示存在 80387;否则,表示存在 80287。

为了严格保持与 80286 的兼容性,ET 位并不受 LMSW 指令的影响。但当存储 MSW 或 CR0 时,位 4 准确地反映 ET 位的当前状态。

PG(第 31 位):分页允许位。若 PG=1,则允许分页部件工作(即片内的 MMU 部件工

作，把线性地址经过两级页表转换为物理地址）；PG＝0，禁止分页部件工作，线性地址即为物理地址。

PG＝1，分页部件工作的前提是 PE＝1，80386 工作在保护虚地址方式，它们的关系如表 1-2 所示。

表 1-2　由 PE 和 PG 决定的处理器工作方式

PG	PE	工作方式
0	0	实地址方式
0	1	保护虚地址方式，不允许分页
1	0	非法组合，不能这样使用
1	1	保护虚地址方式，允许分页

只有在使用分页机制的情况下，CR2 和 CR3 才起作用。

CR2 包含的是出现页故障时的线性地址，以便于页故障处理程序使用。

CR3 包含的是第一级页表即页目录表的地址。

80486 的 CR0 如图 1-15 所示。它除了包含 80286 中的低 4 位（因 80486 片内包含了协处理器，故不需要 80386 中的 ET 位）外还扩展了若干位，分别叙述如下：

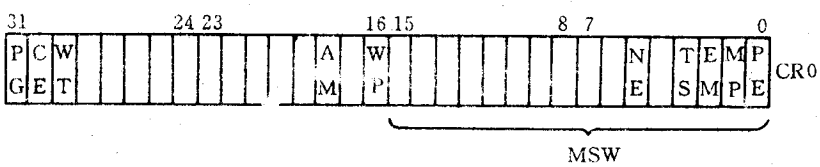


图 1-15　80486 中的 CR0

NE（第 5 位）：数字处理器异常位。此位控制对未屏蔽的浮点异常（UFPE）的处理。若 NE＝1，则未屏蔽的浮点异常为中断向量 16；若 NE＝0，则未屏蔽的浮点异常是通过一个外部中断来处理。后一种是为了与 DOS 操作系统兼容而设置的。在 DOS 系统中，数字协处理器 8087，80287 和 80387 的未屏蔽异常是通过外部中断向量 13 来报告的。

WP（第 16 位）：写保护位。在 80386 中，在页的保护特性中有只读（不允许写）保护，但这样的保护只对处在特权级 3 的用户程序有效；工作在特权级 0、1 和 2 的系统程序可以对只读页进行写入。在 80486 中增加了此位。当 WP＝1 时，禁止任何特权的程序对只读页进行写入，从而支持了操作系统中的 Copy-on-Write 功能。

AM（第 18 位）：对齐屏蔽位。此位能控制在标志寄存器（EFLAGS）中的对齐检测位 AC 是否起作用。AM＝0，禁止 AC 位；AM＝1，则启用 AC 位。AM＝0 是 80386 的兼容模式，因为 80386 允许使用地址未对齐在操作边界上的数据。

WT（第 29 位）：写透明位；CE（第 30 位）：cache 启用位。这两位用于控制 80486 片内 cache 的工作方式。本书不涉及 80486 的片内 cache，故不作进一步说明。

2 80x87的结构

2.1 概 述

16位微处理器之前的CPU，是不适合于数值计算的。因为它们的字长太短，能表达的数值范围太小；若用多字节表示，则计算的速度太慢。而且，没有乘除法指令，使用很不方便。16位甚至32位微处理器虽然有了较强的功能和较大的数值表示范围，但是作数值运算仍是十分困难的。为此，在16位微处理器的基础上设计了与之相配合的专门用于数值计算的协处理器。例如Intel公司开发的与8086(8088)相配合的8087，与80286相配合的80287，与80386相配合的80387，以及在80486中含于片内的协处理器(相当于80387)。我们统称为80x87。它们在许多主要方面是相同的。

数值计算最主要有两个要求：

(1) 计算精度高

(2) 计算速度快

为了满足这样的要求，80x87中设置了8个80位的寄存器。这样的80位的寄存器可用于以下七种数据类型：

(1) 整数字(16位)

(2) 短整数(32位)

(3) 长整数(64位)

(4) 短实数(32位：1个符号位，8位阶，23位尾数)相当于单精度数

(5) 长实数(64位：1个符号位，11位阶，52位尾数)相当于双精度数

(6) 组合的十进制数(80位：1个符号位，18位BCD数)

(7) 临时实数(80位：1个符号位，15位阶，64位尾数)

在80x87的内部是用80位的临时实数表示的，比一般高级语言中用的双精度数还长得多。其可表达的数值范围达到：$3.19 * 10^{-4932} \leqslant |X| \leqslant 1.2 * 10^{4932}$，是一个相当大的数值范围，可以达到很高的精度。

在80x87中设计了有很强数值计算能力的指令系统，其主要的指令种类如表2-1所示。

80x87的指令是与80x86的指令混合编制在一个完整的程序中的，即程序中有一些指令由80x87执行，而另一些指令由80x86执行。指令这样分配是自动实现的。而且在80x87执行数值运算指令时，80x86可以继续执行自己的指令，做到两个处理器的并发执行，从而大大提高了系统的能力。80x86的所有寻址方式都可以用于寻址80x87所需的存储器操作数，因而80x87可以方便地处理数值数组、结构、各种变量。80x87与80x86的密切配合可以使数值运算(特别是浮点运算)的速度提高约100倍。8086与8087结合的一些典型运算指令的执行时间如表2-2所示。

表 2-1 80x87 的指令种类

分　类	指　令
数据传送	取数、存数、交换
算术运算	+、−、*、/、$\sqrt{\ }$、反向减、反向除、换算、求余数、取整、改变符号、求绝对值。
比　较	比较、测试、检验
超越函数	tg、arctg、2^x-1、$y\cdot\log_2(x+1)$、$y\cdot\log_2 x$
取常数	0.0、1.0、π、$\log_2 10$、$\log_2 e$、$\log_{10} 2$、$\log_e 2$
处理器控制	初始化、中断控制、访问控制字、状态/环境的保护/恢复、清除事故

表 2-2 8086 和 8087 的一些运算指令的执行时间

指令类型		处理器类型	
		8086+8087(5MHz)	8086(5MHz)
		执行时间（单位：μs）	
浮点运算指令	加/减	17	1600
	单精度乘法	19	1600
	双精度乘法	27	2100
	除	39	3200
	比较	9	1300
	取双精度数	10	1700
	存双精度数	21	1200
	求平方根	36	19600
	求正切	90	13000
	指数运算	100	17100
18 位 BCD 指令	加/减	127	12040
	乘	297	22990
	除	323	26560
	比较	150	20250

特别要指出的是，80x87 的浮点运算符合 IEEE 的浮点标准。

由于 80x87 能不带舍入地处理 18 位十进制数，所处理的整数长度可达 64 位（$\pm 10^{18}$），因而在事务处理、商业部门和计算机辅助设计(CAD)等方面大有用武之地。

2.2 80x87 的数字系统

当人用纸进行数值计算时，从理论上讲可以是完全精确而不带任何误差的。也即人所使用的实数系统是连续的，任意大小与精度的数值都可以表示。对任何一个实数值来说，总存在着无穷多个更大的数值和无穷多个更小的数值；同时，在任何两个实数之间，也存在着

无穷多个实数。例如在 1.5 与 1.6 之间，就存在着 1.51、1.52、1.555、1.599999、…1.59999999…等无穷多个实数。显然，计算机是不可能工作在整个实数轴上的。无论计算机的规模有多大，它的寄存器和存储器的长度总是有限的，这就使得计算机所能表示的数值大小(范围)及数值精度受到限制 。所以，实际上计算机所能表示的实数系统是一组离散的、有限的数值，它仅仅是实数集的一个子集，是实数系统的一种近似。计算机的位数越多，可表达的数值范围越大，能表示的数值的精确度也越高。

80x87 中的实数，是一种双精度的长实数，它的表示范围大约为 $\pm 4.19\times10^{-307}$— $\pm 1.67\times^{308}$。这是一个相当大的数值范围，在实际应用中，需处理的数据和最终结果超出这一范围的情况是相当罕见的。可见，虽然 80x87 是微型计算机中的协处理器，但已为实际应用提供了一个“足够大”的取值范围。而 80x87 中的临时实数的取值范围更大，如表 2-3 所示。

表 2-3　80x87 中的实数范围

数　据　类　型	近似的取值范围
短实数(单精度)	$8.43*10^{-37}\leqslant\|x\|\leqslant3.37*10^{36}$
长实数(双精度)	$4.19*10^{-307}\leqslant\|x\|\leqslant 1.67*10^{306}$
临时实数	$3.19*10^{-4932}\leqslant\|x\|\leqslant1.2*10^{4932}$

把 8087 的基本实数系统投影到实数轴上，其中的实点(·)表示 80x87 所能精确表示的实数，如图 2-1 所示。

80x87 所能精确表示的两个相邻的实数之间总有一个间隔。如果某次运算的结果正好是 80x87 所能表示的某个实数值，那么 80x87 就精确地表示它；但经常出现结果值落在两个相邻的实数之间，此时，就要根据舍入规则，将该结果舍入成为它所能表示的值。这就有一个精度问题。但在 80x87 中，大部分应用场合精确度是足够的。

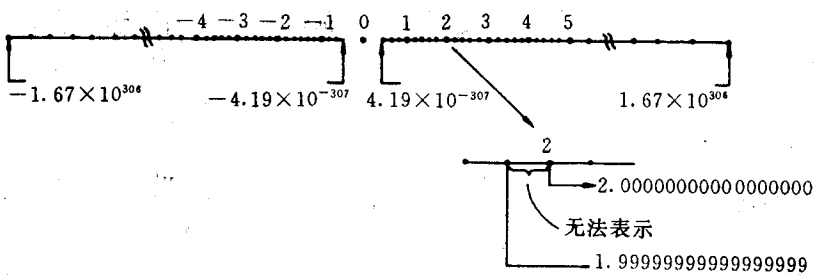
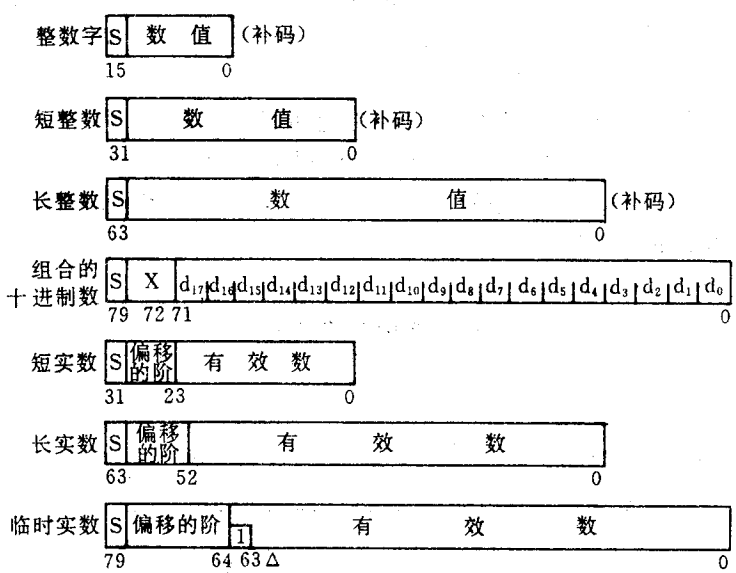


图 2-1　80x87 的实数系统

从图 2-1 还可以看到，80x87 所能表示的实数不是均匀地分布在实数轴上的，在任意的 2 的连续的幂次方之间，80x87 所能表示的实数个数是相等的(因为它的位数是固定的)，即在 2^{16}(65536)和 2^{17}(131072)之间存在着的可表示的实数个数与 2^{1}(2)和 2^{2}(4)之间是相同的。因此，可表示的两个相邻实数之间的间隔是随着数值的增大而增大的。

一般来说，双精度数的实数集已经足够大(表达范围)、足够密(精确度)的了。在 80x87 中所以还要有 80 位的临时实数是想给常数和中间结果以更大的范围和更高的精度，以保证

最后结果的精确度。所以，在运算过程中，应尽可能把中间结果保存在 80x87 的寄存器堆栈中，而不要以结果的形式存放在存储器中。80x87 的 7 种数据类型的格式如图 2-2 所示。所能表达的数值范围如表 2-4 所示。



注：S＝符号位("0"为正数，"1"为负数)
d_n＝10 进制数(2 位/字节)
X＝无效位
Δ＝隐含的 2 进制小数的位置。对临时实数是被存储的，而在短和长实数中是隐含的。指数的偏移：短实数为 127(7FH)，长实数为 1023(3FFH)，临时实数为 16383(3FFFH)

图 2-2　80x87 的数据类型格式

表 2-4　各种数的数值范围

数 据 类 型	位　数	有效数位(10 进制)	近似范围(十进制)
整数字	16	4	$-32767 \leqslant x \leqslant 32767$
短整数	32	9	$-2*10^9 \leqslant x \leqslant 2*10^9$
长整数	64	18	$-9*10^{19} \leqslant x \leqslant 9*10^{19}$
组合的十进制数	80	18	$\underbrace{-99\cdots99}_{18\text{位}} \leqslant x \leqslant \underbrace{99\cdots99}_{18\text{位}}$
短实数	32	6～7	$8.43*10^{-37} \leqslant \lvert x\rvert \leqslant 3.37*10^{36}$
长实数	64	15～16	$4.19*10^{-307} \leqslant \lvert x\rvert \leqslant 1.67*10^{306}$
临时实数	80	19	$3.19*10^{-4932} \leqslant \lvert x\rvert \leqslant 1.2*10^{4932}$

一、二进制整数

80x87 中有三种二进制整数类型，但它们的格式实际上是相同的，只是位数不同。最高位(最左面的位)是符号位，"0"表示正，"1"表示负。负数用补码表示。值得注意的是，在 80x87 中只有二进制整数是用补码表示的，其他数据类型均采用原码表示——正负数只是

符号不同，数值位是相同的。

二、十进制整数

在 80x87 中的十进制整数是用组合的 BCD 码表示，共用 10 个字节即 80 位。最高字节的最高位为符号位，其余位无用，后面 9 个字节，每个字节为两位 BCD 数，故总共为 18 位十进制数。

三、二进制实数

80x87 的二进制实数都采用科学记数法表示，分为阶和尾数，在计算机中则用浮点表示，符合 IEEE 标准。每个数由三部分组成：符号字段、阶码字段和有效数字段。符号字段规定数的正负；有效数字段用于存放数值的有效数字(尾数)；阶码字段用于调整二进制小数点的位置，它也决定了数值的大小。

80x87 中通常是以规格化的格式来表示其有效数字的，即以 1_{Δ}fff…ff 的格式表示有效数字的。其中“Δ”表示一个假设的小数点，故有效数字由一位整数及一个由多位数字组成的小数部分组成。其中小数部分的位数取决于实数的类型，短实数为 23 位，长实数为 52 位，而临时实数为 63 位。在实数的这种规格化表示中，整数位取值总是“1”，这样就消除了“小”的数值的前面的那些“0”，从而使得有效位字段中所表示的有效数位的数目达到最大值。但 80x87 的短实数和长实数中的整数位是一个隐含位，即它实际上并没有真正出现在实数格式当中，而只有在临时实数格式当中，才真正有这个整数位。

为了确定数值大小，还必须考虑指数部分(阶码)。尾数的规格化处理提高了数值的精度，而引入阶码就扩大了数值的表达范围。阶码是为了把二进制小数点定位到有效数字中，它与科学计算中所采用的十进制指数类似。指数(阶码)为正，表示小数点应向右移；指数为负，小数点应向左移。为了省去指数中的符号和便于实现实数之间的比较，80x87 中以偏移的形式来存放指数，即在原指数上加上了一个常数——偏移基数。这个偏移值对于不同的数据类型是不相同的。对于短实数，偏移值为 127＝7FH；对于长实数偏移值为 1023＝3FFH；对于临时实数，偏移值则为 16383＝3FFFH。选择这样的偏移值是为了使阶码总是为正。这样做的好处是，两个实数可以像两个不带符号的二进制整数一样进行比较，一旦发现某个对应位不同时，就可以确定数的大小，对后面的各位就没有必要再进行比较了。采用了偏移指数以后，实现的真实指数可以由阶码段的值减去相应的偏移基数来求得。

若有一个用 16 进制数表示的短实数为：

BE580000

展开成二进制为：1011 1110 0101 1000 0000 0000 0000 0000

把它的符号位、阶码字段和有效位字段分开为：

1 0111 1100 1011 000 0000 0000 0000 0000

符号位为“1”，则此数为负数。

阶码部分转换的十进制值为 124，则实际的指数为：

$$124 - 127 = -3$$

有效数为：

1.1011 000 0000 0000 0000 0000

把这个二进制数字转换为十进制格式得到：

1.0
.5
.125
.0625

1.6875

把这三部分综合起来，可得短实数为：

$$x = -1.6875 \times 2^{-3}$$
$$= -1.6875 \times (0.125)$$
$$= -0.2109375$$

又例如有用16进制表示的长实数为：

406CD25179FCED82

展开为二进制形式：

0100 0000 0110 1100 1101 0010 0101 0001 0111 1001 1111 1100 1110 1101 1000 0010

按符号、阶、有效位分开为：

0 100 0000 0110 1100 1101 0010 0101 0001 0111 1001 1111 1100 1110 1101 1000 0000

符号位为0，则是正数。

阶码为1000 0000 0110，转换为十进制数为1030，减去偏移基数1023，实际指数为7。

有效数字为：

1.1100 1101 0010 0101 0001 0111 1001 1111 1100 1110 1101 1000 0000

可以把小数点后的每一位对应的十进制值算出来（第一位为1/2，第二位为1/4，第三位为1/8。即$1/2^m$，m是位数），然后把它们加起来就可以得到有效数。

1.0
.5
.25
.03125
.015625
.00390625
.00048828125
.00006103515625
.……

1.80133056640625

这是计算了前8位的值，可以根据精度的要求算下去，最后把三部分综合起来，则实数

$$x = +1.80133056640625 \times 2^7$$
$$= +1.80133056640625 \times 128$$
$$= +230.5703124$$

2.3 80x87 的结构

8087、80287、80387 的结构，分别如图 2-3、2-4、2-5 所示。

8087 的结构分成两大部分：控制单元(CU)和数值处理单元(NEU)。数值处理单元负责

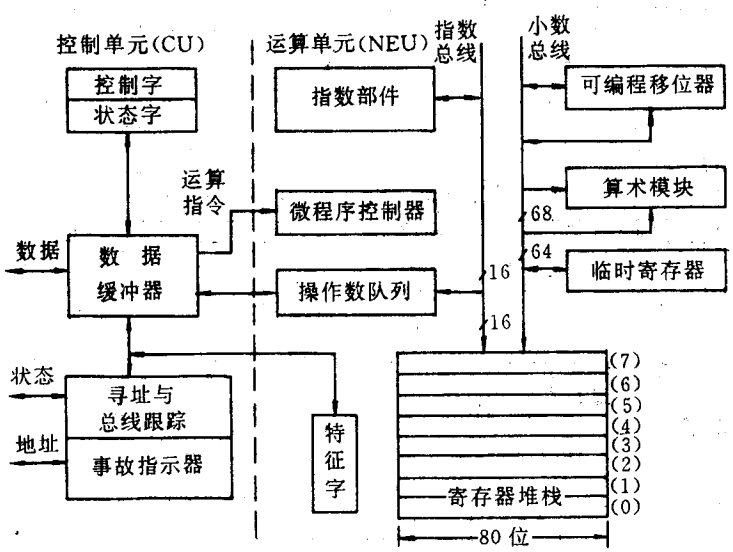


图 2-3 8087 的结构

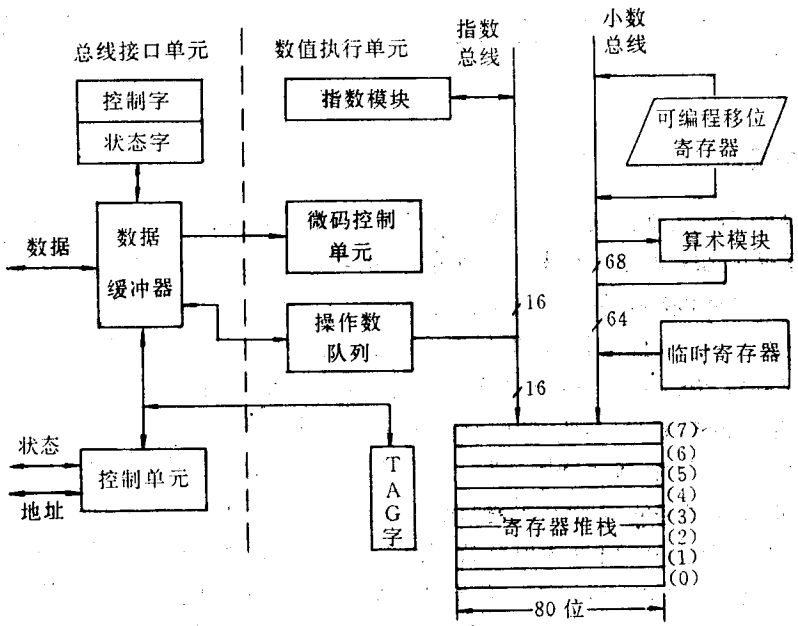


图 2-4 80287 的结构