第1章 80386基本体系结构

微处理机的不同应用需要不同的结构支持。对于某些应用(例如,在 Berkely UNIX 环境下运行的各种应用)可能需要线性地址空间,而对那些要管理大批动态数据结构的应用可能要求有由硬件实施法则,以保护其动态生成目标的可见性。80386 的体系结构为用户提供一组 32 位通用寄存器。它们在使用时不受任何限制,既可用来进行数值计算,又可用它形成存储器地址。80386 体系结构还给用户提供几种存储器管理和寻址方式,以满足不同的要求。此外,80386 还提供多种寻址方式、数据类型、指令以及某些特殊的结构,便于高级语言实施。

80386 基本体系结构包括寄存器模型、数据类型、寻址方式和指令系统,它形成了高级语言编译代码的生成基础,同时也是汇编语言应用程序的设计基础。

1.1 寄 存 器

80386 总共有 34 个寄存器,按其功能可分成以下几类:

通用寄存器(General-Purpose Register)

段寄存器(Segment Register)

状态和控制寄存器(Status and Control Register)

系统地址寄存器(System Address Register)

调试寄存器(Debug Register)

测试寄存器(Test Register)

1.1.1 通用寄存器(General-Purpose Register)

8 个通用寄存器是 8086 和 80286 寄存器的超集。它们的名字和用途分别为:

EAX 通常用作累加寄存器(Accumulator)

EBX 通常用作基址寄存器(Base)

ECX 通常用来记数(Count)

EDX 通常用来存放数据(Data)

ESP 通常用作堆栈指针(Stack Pointer)

EBP 通常用作基址指针(Base Pointer)

ESI 通常用作源变址(Source Index)

EDI 通常用作目标变址(Destination Index)

8 个通用寄存器中通常保存 32 位数据,但为进行 16 位的操作并提供与 Intel 系列 16

位机兼容，它们的低位部分被当成 8 个 16 位的寄存器。为了支持 8 位的操作还可进一步把 EAX、EBX、ECX 和 EDX 寄存器低位部分的 16 位，再分成 8 位一组的高位字节和低位字节两部分(见图 1.1)，作为 8 个 8 位寄存器。这 8 个寄存器分别被命名为 AH、BH、CH、DH 和 AL、BL、CL、DL。对 8 位或 16 位寄存器的操作只影响相应的寄存器。例如，在做 8 位加法运算时，位 7 的进位并不传送给目的寄存器的位 9，而且把在标志寄存器中的进位标志(CF)相应地置数。总之，这 8 个通用寄存器既可以支持 1 位、8 位、16 位和 32 位数据运算，也支持 16 位和 32 位存储器寻址。

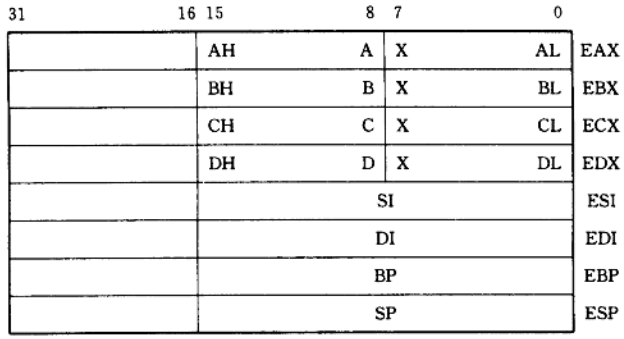

图 1.1　80386 的通用寄存器

1.1.2　段寄存器(Segment register)

80386 有 6 个 16 位段寄存器(见图 1.2)，也称选择符(selector)，它们的名字和用途如下：

CS　代码段寄存器(Code Segment)
DS　数据段寄存器(Data Segment)
SS　堆栈段寄存器(Stack Segment)
ES　附加数据段寄存器(Extra Data segment)
FS　附加数据段寄存器(Extra Data segment)
GS　附加数据段寄存器(Extra Data segment)

(程序员还可给 FS 和 GS 寄存器起恰当的缩写名)

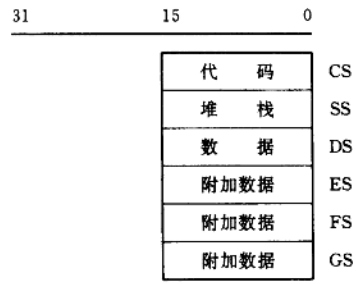

图 1.2　80386 段寄存器

6 个 16 位的段寄存器实现存储空间的分段。所谓段即是把 64T(兆兆)字节存储空间分成各自独立的逻辑地址空间。这样每个程序同时可有 6 个逻辑地址空间供交换用，其中包括 4 个独立的数据空间。

6 个 16 位的段寄存器选择各自的存储区域，其中 CS、DS 和 SS 这 3 个段寄存器给当前的代码、数据和堆栈段寻址。剩下的 3 个段寄存器给用户定义的数据区域寻址。所以在段寄存器内给当前可访问存储单元保存选择符的值。对于实地址方式，段的大小是从 1 个字节到 64K 字节。而对于保护方式寻址，可允许段的大小从 1 个字节到 4G(千兆)字节。

在实方式时，80386 段寄存器内有指定段的实际基地址(见图 1.3)。在保护方式下，段寄存器内保存着 16 位的地址。这个 16 位地址就是通常所说的选择符(selector)。由它指示描述符表中的某一项。实际的段基地址被插在描述符表中，如图 1.4 所示。

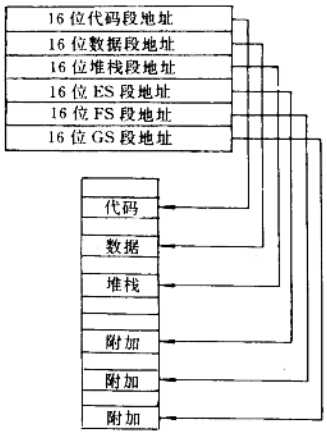

图 1.3　实方式下段寻址

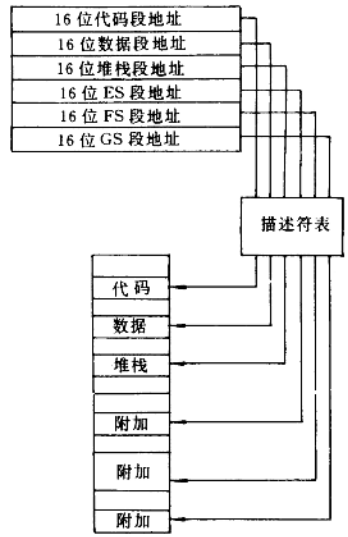

图 1.4　保护方式下段寻址

正在执行的程序代码都驻留在存储器内，而由代码段寄存器 CS 对程序代码进行寻址。现役数据段的基地址由数据段寄存器寻址。在堆栈段寄存器 SS 内保存现役堆栈段的当前基地址。因为通常都是用堆栈存放中间结果，在调用子程序时，还要给出它们自己的存储器段，所以在堆栈段寄存器 SS 内一定要保存好现役堆栈段当前基地址。当然，程序设计人员借助于附加数据段寄存器 ES，还可以访问其他段。通过 ES、FS 和 GS 这 3 个段寄存器，80386 可以同时对 3 个现役数据段寻址。

1.1.3　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，然而了解它的存在和作用却非常有益。在 80386 的内部，描述符寄存器与可见的各个段寄存器是相联的，如图 1.5 所示。每个描述符寄存器中保存 32 位的段基地址、32 位的段界限和其他属性。

当一个选择符的值装入段寄存器时，相关描述符寄存器的内容就自动地修改成正确的信息。在实地址方式中，只有基地址被直接修改(把选择符的值左移 4 位即可)，因为在实方式中最大段限和段属性是固定的。在保护方式中，基地址、段界限和属性全部按照每个选择符指引的段描述符内容进行修改。

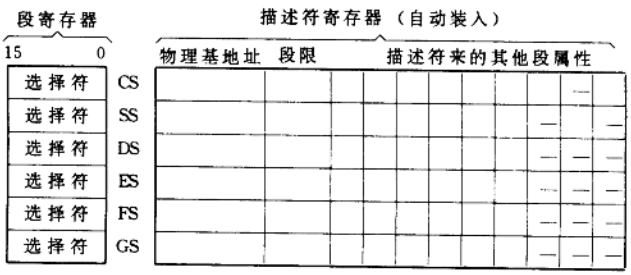

图 1.5 80386段寄存器和有关描述符寄存器

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该次存储器的访问操作。32 位段基地址成为线性地址计算中的一个分量，32 位界限用于界检验操作，对照要求的存储器访问类型检验属性。

1.1.4 状态和控制寄存器(status and coutrol register)

状态和控制寄存器是由标志寄存器 EFLAGS (Flag Register)、指令指针 EIP (instuction pointer)和 4 个控制寄存器 CR0～CR3 (Control Register)组成，如图 1.6 所示。

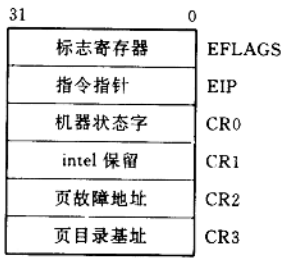

图 1.6 状态和控制寄存器

指令指针寄存器 EIP 中存放下一条要执行指令的偏移量(Offset)，这个偏移量是相对目前正在运行的代码段寄存器 CS 而言的。偏移量加上当前段的地址，形成了下一条指令的地址。EIP 中的低 16 位可以分开来进行访问，给它起名叫作指令指针 IP (Instruction Pointer)寄存器，用于 16 位寻址。

标志寄存器 EFLAGS 存放有关微处理机的信息(如图 1.7 所示)。标志寄存器中的第 1、3、5、15 位及 18～31 位都没有定义。但是，当使用 SAHF 指令或 PUSHF 指令时，或当出现中断处理时，除位 1 外，其余位全部是 0。

第 0 位 CF(Carry Flag)是进位标志位，在算术运算时如果产生进位或借位，则把 CF 位置 1，否则把 CF 位置 0。在执行移位和环移指令时也要用到 CF。CF 内保存从寄存器移出或环移出的那一位。

第 2 位 PF (Parity Flag)是奇偶校验标志位，主要是用在数据通信上。如果是奇数奇偶校验，把 PF 置 1；如果是偶数奇偶校验把 PF 置 0。

第 4 位 AF (Auxiliary Carry Flag)是辅助进位标志。在用 BCD(二—十进制)进行算

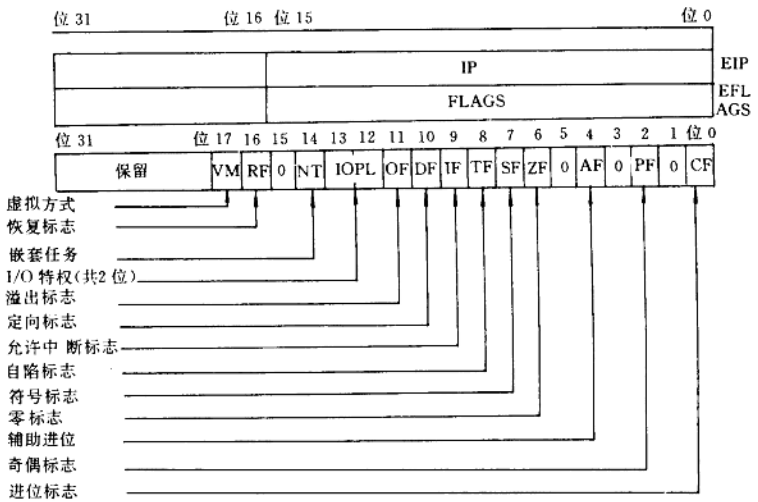

图 1.7　80386 指令指针寄存器 EIP 和标志寄存器 EFLAGS

注：0 表示没有使用。

术运算时，用第 4 位 AF 表示进位输出是否进入到 BCD 最低四位，或者是否到 BCD 最低四位借位。

第 6 位 ZF(Zero Flag)是零标志位，当结果为零时则将其置成 1，否则将其置成零。

第 7 位 SF(Sign Flag)是符号标志位。当结果为负时则将其置成 1，结果为正时则将其置成 0。

第 8 位 TF(Trap Flag)是自陷标志位，当将其置成 1 时则可以进行单步执行。当指令执行完后，就有可能生成异常事故 1 的自陷。也就是说，在程序执行过程中，每执行完一条指令，都要由异常事故 1 处理程序进行检验。当将第 8 位 TF 清 0 后，且当将断点地址装入调试寄存器 DR0—DR3 时，才会产生异常事故 1 的自陷。

第 9 位 IF(Interrupt Flag)是中断标志位，是用来表示允许或者禁止某些外部中断。若第 9 位 IF 被置成 1，则允许 CPU 认定外部中断请求信号；若将 IF 位清成 0，则表示禁止外部中断请求。只有当第 12、13 位(输入输出特权位)指出当前最高特权值 CPL，才允许将新值置入标志寄存器 EFLAGS 时再改变 IF 位的值。

第 10 位 DF(Direction Flag)是定向标志。DF 位规定了在执行串操作过程中，对源变址寄存器 ESI 或目标变址寄存器 EDI 是增值还是减值。如果 DF 值为 1，则寄存器减值；若 DF 值为 0，则寄存器值增加。

第 11 位 OF(Overflow Flag)是溢出标志位，用它表示运算时出现的进位进入了结果的高序位，可高序位却没有进位输出，或是高序位并没有接收进位输入却产生了进位输

出。对带符号的操作数进行运算时，若运算结果一旦超出了数可表示的范围，OF 位就被置成 1。否则将其清成 0。

第 12、13 位 IOPL 是输入输出特权级位，在保护方式下常要使用这两位标志。由于输入输出特权级标志共两位，它的取值范围只可能是 0、1、2 和 3 共 4 个值，恰好与输入输出特权级 0～3 级相对应。在当前任务的特权级 CPL 高于或等于输入输出特权级时，就可以执行像 IN、OUT、INS、OUTS、STI、CLI 和 LOCK 等指令而不会产生异常事故 13(中断号 13)。在当前任务特权级 CPL＝0 时，上托标志出栈指令 POPF 或中断返回指令 IRET 可以改变 IOPL 字段的值。当新的标志信息进入任务的 TSS 时，任务切换操作也可以改变 IOPL 的值。

第 14 位 NT 是嵌套任务标志位。在保护方式下常使用这个标志。当 80386 在发生中断时和执行 CALL 指令时就有可能会引起任务切换。若是由于中断或由于执行 CALL 指令而出现了任务切换，则将嵌套任务标志位 NT 置 1。若没有出现任务切换，则将 NT 位清 0。很明显，若将 NT 位置成 1，则说明欲执行的任务是嵌套在前一项任务之中的。同时还必须指出当前嵌套任务的任务状态段 TSS 和有一条返回前一项任务的 TSS 的有效途径。标志寄存器 EFLAGS 的嵌套任务标志位 NT 的值由中断返回指令 IRET 给以测定，以决定是进行同任务内的返回还是不同任务间的返回。上托标志出栈指令 POPF 或中断返回指令 IRET 会根据从堆栈弹出来的信息，在任何特权级下都能给 NT 位置值。

第 16 位 RF(Resume Flag)是恢复标志位，也是 80386 新增添的一位标志位。当每条指令成功地执行完后，都会自动地将 RF 位置成 0，但中断返回指令 IREF、上托标志出栈指令 POPF 和实现任务转换的指令除外。RF 标志与调试寄存器一起用于断点和单步操作，当 RF 置成 1 时，下一条指令的所有调试故障全被忽略。

第 17 位 VM(Virtual 8086 Mode Flag)是虚拟 8086 方式标志位，是 80386 新设置的一个标志位。表示 80386 微处理机是在虚拟的 8086 环境中运行。如果 80386 微处理机是在保护方式下，而 VM 位又被置成 1，这时 80386 就转换成虚拟 8086 操作方式，使全部段执行操作就像是在 8086 微处理机上运行一样。VM 位只能由两种方式中的一种方式给以设置，或者是在保护方式下，由最高特权级(0)级代码段的中断返回指令 IRET 设置，或者是由任务转换进行设置。

80386 还定义了由 4 个 32 位的控制寄存器 CR0—CR3 组成一个寄存器组，如图 1.8 所示。

在这几个寄存器中保存全局性和任务无关的机器状态，这些寄存器连同后面将要说明的系统地址寄存器(System Address Register)一起，保存影响系统中所有任务的机器状态。访问控制寄存器时，要用存、取这一类传送指令，例如 MOV CR0 指令。

控制寄存器 CR0 包含有 6 个预定义标志，表示和控制机器的状态。其中 0—15 位叫作机器状态字 MSW(Machine Status Word)，这使得 80386 在保护方式下能与 80286 兼容。

控制寄存器 CR0 的 0 位是允许保护位 PE (Protection Enable)，用于启动微处理机的保护方式。如果 PE 置 1，保护方式启动；如果 PE 置 0，则微处理机在实地址方式下运行。

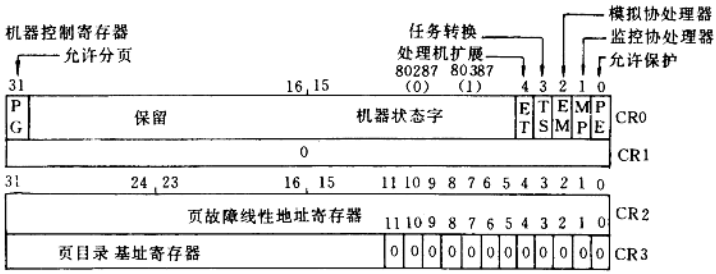

图 1.8 80386 的控制寄存器组

CR0 中的第 1 位是监控协处理位 MP(Monitor Coprocessor)。它与 TS 位一起决定：当 TS=1 时操作码 WAIT 是否产生一个“协处理器不能使用”的出错信息。

CR0 中的第 2 位是模拟协处理器位 EM(Emulate Coprocessor)。如果 EM=1，则所有协处理器的操作码都将产生一个“协处理器不能使用”的出错误信号；如果 EM=0，则所有协处理器的操作码都能在实际的 80287 或 80387 协处理器上执行。

CR0 中的第 3 位是任务转换位 TS(Task Switched)。当一个任务转换完成之后自动把它置 1。随着 TS 置 1，协处理器的操作码将会引起一个协处理器不能使用的陷阱。

CR0 中的第 4 位是微处理机的扩充类型位 ET(Processor Extension Type)，其内保存着微处理机扩充类型的信息。如果 ET=0，表示系统内用的是 80287 浮点协处理器。如果 ET=1，表示系统内用的是 80387 浮点协处理器。这样，设计人员可选择使用 80287 和 80387。

CR0 的第 31 位是允许分页位 PG(Paging Enable)。它表示芯片上的分页部件是否允许 PG 位和 PE 位配对，给 80386 提供选择操作环境的 4 种组合方式。由 PG 位和 PE 位定义的操作方式由表 1.1 说明。

表 1.1 PG、PE 位的 4 种组合

PG	PE	方　式
0	0	“实方式”，8086 操作
0	1	“保护方式”，确切的 80286 的语义加上 32 位扩充(在扩充的 80286/80386 保护方式范围内，还定义了一个支持虚拟 8086 的子环境)
1	0	未被定义。如果试图利用这一组合则将发出一个一般的保护故障
1	1	“分页保护”环境，允许保护方式下的所有设施都能分页

CR1 是未定义的控制寄存器，用作备用。

CR2 是页故障线性地址寄存器，保存最后出现页故障的全 32 位的线性地址。

CR3 是页目录基址寄存器，保存页目录表的物理基地址。因为 80386 的页目录表是按页排列的，所以低 12 位不起作用，即使写上了内容，也不被理会。

1.1.5 系统地址寄存器(System Address Register)

80386 有 4 个系统地址寄存器，如图 1.9 所示，它保存操作系统需要的保护信息和地址转换表信息。

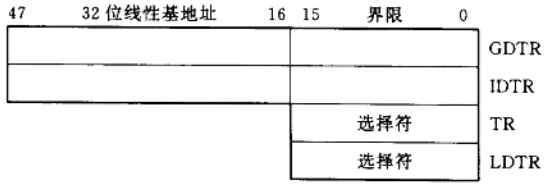

图 1.9 80386 系统地址寄存器

这 4 个专用的寄存器用于引用 80386 在保护方式下所需要的表或段。这 4 个系统地址寄存器的名字和作用如下：

全局描述符表寄存器 GDTR(Global Descriptor Table Register)，是 32 位寄存器，用来保存全局描述符表(GDT)的 32 位线性基址和 16 位 GDT 的界限；

中断描述符表寄存器 IDTR(Interrupt Descriptor Table Register)，是 32 位寄存器，用来保存中断描述符表(IDT)的 32 位线性地址和 IDT 的 16 位界限；

局部描述符表寄存器 LDTR(Local Descriptor Table Register)，是 16 位寄存器，其内保存着 16 位的局部描述符表 LDT 段的选择符；

任务状态寄存器 TR(Task State Register)是 16 位寄存器，其内保存任务状态段 TSS 段的 16 位选择符。

用以上 4 个寄存器给目前正在执行的任务定义任务环境、地址空间和中断向量空间，这些寄存器的语义和控制与 80286 兼容。

1.1.6 调试寄存器(Debug Register)

80386 为调试提供了硬件支持。在 80386 芯片内有 8 个调试寄存器 DR0—DR7，如图 1.10 所示。

这些寄存器可以使系统程序设计人员定义 4 个断点，用它们可以规定指令执行和数据读写的任何组合。DR0—DR3 是线性断点地址寄存器，其中保存着 4 个断点地址。DR4、DR5 是两个备用的调试寄存器，目前尚无定义。DR6 是断点状态寄存器(Breakpoint Status Register)，其低序位是指示符位，当允许事故调试并检测出事故而进入调试事故处理程序时，由硬件把指示符位置 1，调试处理程序在退出之前必须把这几位清

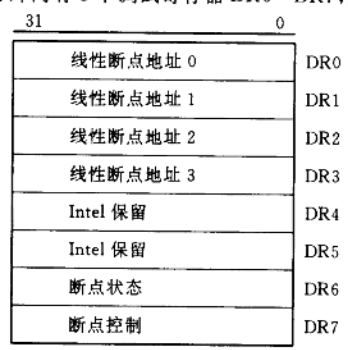

图 1.10 80386 的调试寄存器

0。DR7 是断点控制寄存器(Breakpoint Control Register),它的高序半个字又被分成 4 个字段,用来规定断点字段的长度是 1 个字节、2 个字节还是 4 个字节,和规定将引起断点的访问类型。低序半个字的位字段用于"允许"断点和"允许"所选择的调试条件。

1.1.7 测试寄存器(Test Register)

80386 有两个 32 位的测试寄存器 TR6 和 TR7,如图 1.11 所示。

TR6 和 TR7 是两个 32 位寄存器,用于在转换旁视缓冲器 TLB(Translation Lookaside Buffer)中测试随机存储器(RAM)和相联存储器(CAM)。TR6 是测试命令寄存器,其内存放测试控制命令。TR7 是数据寄存器,其内保存转换旁视缓冲器测试的数据。

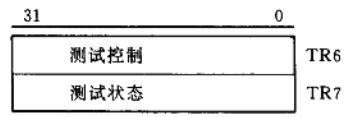

图 1.11 80386 的测试寄存器

1.1.8 寄存器的可访问性

在实方式和保护方式下,这些寄存器的可访问性有所不同,表 1.2 示出不同点。详细说明请参阅第 4 章。

表 1.2 寄存器使用方法表

寄存器	实方式		保护方式		虚拟方式	
	装入	存储	装入	存储	装入	存储
通用寄存器	可	可	可	可	可	可
段寄存器	可	可	可	可	可	可
标志寄存器	可	可	可	可	IOPL	IOPL
控制寄存器	可	可	PL=0	PL=0	不	可
GDTR	可	可	PL=0	可	不	可
IDTR	可	可	PL=0	可	不	可
LDTR	不可	不可	PL=0	可	不	不
TR	不可	不可	PL=0	可	不	不
调试控制	可	可	PL=0	PL=0	不	不
测试寄存器	可	PL=0	PL=0	PL=0	不	不

说明:1. PL=0,仅当现行特权级为 0 时,才能访问该寄存器。

2. IOPL,在虚拟 8086 方式中,PUSHF 和 POPF 指令对 I/O 特权级敏感。

1.2 寻址操作

80386 有许多寄存器,而且又提供了功能很强的指令系统,因而,80386 能够进行各种

不同数据类型的操作和寻址。80386 的寻址方式是先前 Intel 系列机寻址方式的扩充。由于增加了几个地址寄存器，所以 80386 的寻址方式又得到发展，显得更加灵活。

1.2.1 80386 的几种寻址方式

80386 为指令指定操作数提供了 11 种寻址方式，这些寻址方式可以有效地实现 PASCAL、FORTRAN、BASIC 等高级程序设计语言所需的绝大多数数据的访问。可把 80386 的寻址方式分成寄存器寻址、立即寻址与存储器寻址 3 种寻址方式。

1. 寄存器方式和立即方式

这两种寻址方式专供处理寄存器操作数和立即操作数的指令使用。

a. 寄存器操作数方式

操作数可放在 8 位、16 位或 32 位通用寄存器中。

b. 立即操作数方式

这种方式很直观，操作数作为操作码的一个组成部分含在指令中。

2. 存储器寻址方式

存储器寻址方式有 9 种。存储器寻址方式提供一种指定操作数有效地址的手段、途径。线性地址由两部分组成：段基地址和有效地址。而有效地址由 4 种地址分量任意组合而成。这 4 种地址分量如下。

a. 位移

跟在指令后面的 8 位或 32 位的立即值(在指令中加上地址前缀可使用 16 位位移)。

b. 基地址

任一通用寄存器的内容。这个基地址寄存器常常由编译程序用来指向局部变量区域的起始点。

c. 变址

除 ESP 以外的任一通用寄存器的内容。通常使用变址寄存器访问某一数组元素或某一字符串。

d. 比例常数

变址寄存器的值可以乘上一个比例常数(1、2、4 或 8)。乘上比例常数的变址方式对于访问数组或数据特别有效。

以上 4 种分量任意组合构成 9 种寻址方式。由于有效地址的计算与其他指令的执行是按流水线方式进行的，因此使用其中任何一种寻址方式组合都不会降低性能。但同时使用基地址、变址和位移分量的情况例外，因为此时要增加一个额外的时钟周期。由 4 种分量构成的 9 种寻址方式介绍如下：

① 直接方式

操作数的偏移以 8 位、16 位或 32 位位移量的形式作为指令的组成部分。

例：INC WORD PTR[500]

② 寄存器间接方式

在某个基地址寄存器或变址寄存器中含有该操作数的地址。

例：MOV [ECX], EDX

③ 基地址方式

某个基地址寄存器的内容与一个位移量相加，构成操作数的偏移量。

例：MOV ECX, [EAX+24]

④ 变址方式

某个变址寄存器的内容与一个位移量相加构成操作数偏移量。

例：ADD EAX, TABLE[ESI]

⑤ 比例变址方式

变址寄存器的内容乘以比例常数，再加上位移量，构成操作数的偏移量。

例：IMUL EBX,TABLE[ESI * 4],7

⑥ 基地址变址方式

基地址寄存器的内容与变址寄存器的内容相加构成操作数的有效地址。

例：MOV EAX,[ESI][EBX]

⑦ 基地址比例变址方式

变址寄存器的内容与比例常数相乘，其结果再与基地址寄存器的内容相加得出操作数的偏移量。

例：MOV ECX, [EDX * 8][EAX]

⑧ 基地址变址位移方式

变址寄存器的内容、基地址寄存器的内容和位移量相加构成操作数的偏移量。

例：ADD EDX,[ESI][EBP+C0FFFFF0]

⑨ 基地址比例变址位移方式

寄存器的内容乘以比例常数，其结果再与基地址寄存器内容和位移相加构成操作数的偏移量。

例：MOV EAX,LOCALTABLE[EDI * 4][EBP+80]

3. 80386 寻址方式综述

如上所述，80386 的操作数可以放在寄存器内，也可以放在主存储器内，也可以放在 I/O 的地址空间中，还可以把操作数隐含在指令中，而作为指令的一个组成部分。

操作数存入寄存器是最简单的一种，也是处理数据最快的一种方法。如完成下例计算：

$$Power = Voltage \times Current$$

假设 Current 值已存放在寄存器 EBX 中，Voltage 值已存放在寄存器 ESI 中，就可用下面指令序列

```
MOV EDI,ESI          ;Voltage 的值存在 EDI 中
IMUL EDI,EBX         ;EDI×Current 的乘积存在 EDI 中
```

完成计算，其结果(Power)存到 EDI 中。

任何算术运算或逻辑运算符都可用 80386 的 8 个通用寄存器的内容进行操作。另外

8 位、16 位、32 位的常数都可以直接嵌在一条指令之中做为立即数。16 位和 32 位操作可以规定 8 位符号扩展或零扩展的立即数。把那些用寄存器和立即数作为操作数的有代表性的典型指令列于表 1.3。

通常。80386 的寄存器对寄存器的操作都在两个时钟内完成。在时钟频率为 16MHz 时，每执行一次操作约用 125ns 时间。

大多数操作数都存放在主存储器中，在 80386 中表示存储器的地址是指针(Pointer)。指针又分近指针和远指针。近指针是有效地址的另一种称呼。远指针由两部分构成，其一是 16 位的段选择符(Segment Selector)，用它选择一个特定的段，其二是 32 位的偏移量(Offset)，它是段的基址和要寻找项间的距离，如图 1.12 所示。

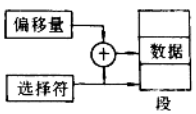

图 1.12 地址信息

表 1.3 80386 操作数寻址举例

指　令	时钟	语　义
INC EAX	2	EAX 的内容加 1
IMUL EBX,−3	9	EBX 中的整数乘−3
CMP CX,0	2	CX 的内容与 0 比较，并置条件码
MOVSX EAX,SI	3	把 16 位寄存器 SI 内容中的符号扩展，一同送入 EAX
MOV DWORD PTR[56],−12445654	2	把地址为 56 的 32 位整数赋值为−12445654
JMP Jump Table[EBX * 4]	10	跳转到跳转表入口 EBX 处存放的地址
SUB DX,WORD PTR[EBP+EDI * 2−10]	7	从 DX 减去在地址[EBP+EDI * 2−10]处的 16 位量

一个完全的 80386 寻址结构由 5 部分组成，即段址＋基址(段内)＋变址×比例常数＋位移量。段址即当前段的基址，它保留在 6 个 16 位的段寄存器内。存放基址的寄存器可以是 8 个通用寄存器中的任一个，也可以没有基址。存放变址的寄存器可以是除 ESP 寄存器之外的 7 个通用寄存器中的任一个，当然也可以没有变址。比例常数可以是 1、2、4 或 8。位移量可以是 8 位、16 位或 32 位，也可以没有位移量。

由此可见，80386 的整个指针是 16＋32＝48 位。又因为段选择符的低序位二位(位 1 位 0)实际上用于保护目的，因此，整个指针还有 46 位的有效地址。这样就能满足微处理机对 64T(2^{46})字节虚拟存储器的访问。这样大的地址空间一般能满足最大的程序或最大数据结构的需要。同时，80386 也支持短的 32 位指针，也就是说，偏移量限在当前段内。因此，也只能访问 4G(2^{32})字节地址空间。

80386 有一整套地址生成机构，可生成操作数的有效地址，供访问操作数用。这些机构(以后详述)是为适应目前流行的高级语言中的存储范例设置的。

因为大多数操作数都存在主存储器内，所以操作数如没有驻留在寄存器中，则必须到存储器把操作数预先取出放到寄存器内。最简单的形式就是在指令中直接给出操作数的有效地址(Effective Address)。然而，在程序真正执行之前，这一特定的操作数的存储器地址往往是不知道的。在这种情况下，有效地址的计算就要用到上述方法。把计算公式一般化，就可把有效地址的寻址方式写成：

[基址寄存器]+[变址寄存器]*[比例常数]+[位移量]

[基址寄存器]表示基址寄存器的内容,[变址寄存器]表示变址寄存器的内容。如前所述,基址寄存器可以是任一通用寄存器,而变址寄存器可以是除 ESP 以外的任一通用寄存器。因为 ESP 总是指着当前堆栈,所以用它对堆栈进行相对寻址。比例常数可以是 1、2、4 或 8,其值一经规定,就要与变址寄存器的值相乘。这样作的目的是简化对多字节元素的数组的查找。位移量也是一个常数,其值范围从 -2^{31} 到 $2^{31}-1$。为更简单明确地说明 80386 的存储器寻址方式,把 80386 的存储器寻址方式列在表 1.4 中。

基址、变址、位移量和比例常数都是任意的。按照组合,80386 处理机共有 11 种不同的 32 位寻址方式。这些方式概括了几种比较好的常用高级语言的编译程序最常用的那些方式。为使读者更加清楚地认识这 11 种不同的寻址方式,现列于表 1.5 中。表中还包括立即寻址和寄存器寻址。

表 1.4　32 位存储器寻址方式

基址 (base)		变址 (index)		比例常数 (scale)		位移量 (displacement)
EAX	+	EAX	*	1	+	0
EBX		EBX				
ECX		ECX		2		
EDX		EDX				8 位位移量
ESP		—		4		
EBP		EBP				
ESI		ESI				
EDI		EDI		8		16 位位移量

表 1.5　80386 的 11 种寻址方式

地址成分			寻址方式
		位移量	仅位移量
基址			仅基址
基址		+位移量	基址+位移量
	变址		仅变址
	变址×比例常数		变址×比例常数
	变址	+位移量	变址+位移量
	变址×比例常数	+位移量	变址×比例常数+位移量
基址	+变址		基址+变址
基址	+变址×比例常数		基址+变址×比例常数
基址	+变址	+位移量	基址+变址+位移量
基址	+变址×比例常数	+位移量	基址+变址×比例常数+位移量
寄存器			寄存器寻址
立即数			立即寻址

下面把 80386 对高级语言的存储器寻址的支持列入表 1.6。

表 1.6 80386 对高级语言存储器寻址的支持

存储类型	类型说明	寻 址 方 式
静态	标 量	[位移量]
	结 构	[位移量]
	标量数组	[位移量+变址]
	结构数组	[位移量+变址]
自动	标 量	[基址+位移量]
	结 构	[基址+位移量]
	标量数组	[基址+位移量+变址]
	结构数组	[基址+位移量+变址]
堆栈	标 量	[基址]
	结 构	[基址+位移量]
	标量数组	[基址+变址]
	结构数组	[基址+位移量+比例常数]

1.2.2 基址位移量

下面具体说明一种 80386 寻址方式。

设想它通过压栈把参数传送到子程序中去。通过使用基址位移量方式，用下一指令在两个时钟周期内可从寄存器 EAX 内获得最后一个参数：

```
MOV DWORD PTR[ESP+4],EAX;
```

假设指针 NextRecord 存在寄存器 EAX 中，使用下面这条指令，在 4 个时钟周期内可把它联向一个数据，并不需要额外的内务操作：

```
MOV EAX,DWORD PTR[EAX+Linkoffset];
```

最后，假设 V 指向一个动态分配的整型向量，要求把项 V[i+2]预置成 5。如果把 V 送入寄存器 ESI，并且把它作为一个寄存器变量再存到 EDI，则下面这条语句在 3 个时钟周期内能完成所需的操作。实现这个操作用的是基址+变址×比例常数+位移量这种寻址方式，MOV DWORD PTR[ESI+EDI * 4+8]，使用比例常数是为了在一个 4 字节整型数组中实现变址，免得再用其它指令。另外，再强调指出：80386 支持先前所有的 Intel 系列机的寻址方式，以确保 80386 与 Intel 系列机兼容。

以上每条指令所需时钟个数是很少的，这保证了 80386 高速运行。高速的关键在于重叠操作：在执行内代码(In-line Code)时，在前一条指令的最后一个时钟就开始形成本条指令的有效地址。因为形成一个有效地址又需一个时钟(除去规定了变址以外)，所以有效地址的形成时间几乎都隐藏在一个指令执行时间之内。

因为把 80386 的存储管理部件 MMU 也配置在芯片之内，所以在有效地址的形成时间之内也包括把逻辑地址转换成物理地址所需的时间。因此，在 80386 内实际上并不存在地址的生成和转换的延迟问题。只当使用变址时才出现例外，这时有效地址生成需 2 个时钟，其中有一个时钟的有效延迟。

1.2.3 位和位串的操作

80386 支持单个位和位串操作。一个位串可达 4G 位(2^{32})长。操作时是按位串的位顺序进行的。一个位字段(Bit Field)可实现 32 位操作。

1. 单个的位操作

80386 提供了一套完整的位操作指令。这些指令的操作数可以选择单独一位,也可以从存储器中 4G 位位串内选择从 1 到 32 位的一个位段,也可以选择 8 位、16 位或 32 位的寄存器。而单个位的地址或位字段的起始地址都是通过功能位[基址,偏移量]指定的。如图 1.13 所示。图中基址可使用 80386 的任一种寻址方式指定寄存器中的一个字节地址,或指定存储器中的一个字节地址,而偏移量给出了距这个字节边界的位偏移量(Bit Offset)。因为偏移量是 32 位的,因此它能指定从−2 千兆位到+2 千兆位的范围。位操作指令的操作数并不限定在一个字节或一个字的边界上,这些操作数的地址是位地址。

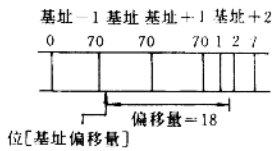

图 1.13 位寻址

在系统软件中,经常访问各独立的位。例如实现操作系统中常用的信号量(semaphore)的具体操作,就是通过在一个叫作 sema 字节第二位上完成原子的(被封锁的)测试和置位,并且使它处于等待状态,直到把这位清 0:

```
Waitlp: Look BTS sema, 2
    JC          Waitlp
```

现再举一例,研究下面一个 C 程序段。这个程序段定义了变量 Access Right 的位字段:

```
Struct (present: 1;
        DPL          :   0
        Seg          :   1;
        ECRA         :   4;
        ) Access Right;
        reg Var=Access Right.DPL;
```

假设 reg Var 是保存在 ESI 中的寄存器变量,其代码直接转换成了处理机的指令序列:

```
MOV CL,   2     ;建立字段长度
MOV EAX,  5     ;形成字段偏移量(1+4)
XBTS ES1,accessRight,EAX,CL
```

为便于浮点操作,80386可与80287或80387数值协同处理器配合。使用这两种芯片增加了处理机处理的数据类型,使处理机能完成64位字整数、18位 BCD 整数、32位中的单独一位实数值、64位双精度实型数、80位扩展实型数的操作。数值协同处理器还提供全部的适用于这些数据类型的指令集。

为适应BCD数值需要，提供了一些原语以帮助设置压缩的(packed)和非压缩的(unpacked)十进制运算。此外，数值协同处理器还使80位压缩的十进制算术运算得以实现。

80386还把64位运算当成一个目标，像32位整数相乘产生的64位乘积和除法时64位的被除数，都是64位运算例证。而且借助于带进位的加位(add-with-carry)和带借位的减法(Subtract-with-carry)很容易实现双精度加减运算。

2. 位串操作

80386可以对每个长达31位的连续位序列进行取数和存数操作。而这些位字段(bit field)本身存放在最长可达4G 位的位串之内。当然也可以测试和修改每一位的值，并且可正向扫描位字段和反向扫描位字段，从而找出第一个置1的位。这一特点在PASCAL 语言中可用来实现集合(set)类型。例如，如果 COL 是 set of colon 类型中的一个目标，那么PASCAL 的一段语句

```
while C in Col do
```

可以翻译成

```
BSF EAE,Col;   找出集合第一个元素，存入 EAX
JZ Loop Exit;  作完后退出
```

1.2.4 16位地址和32位地址的区别

为实现与80286和8088的软件兼容，80386可以用实方式和保护方式执行16位指令。处理机通过检查段描述符中的 D 位确定该处理机正在执行的指令长度。若 D=0，则表示所有操作数长度和有效地址都设定为16位；若 D=1，则操作数和地址的缺省为32位。在实方式中，操作数和地址的缺省长度为16位。

不管操作数和地址缺省长度如何，80386均可能执行16位或32位指令，这是通过使用强制性前缀指定的。操作数长度前缀和地址长度前缀逐条指令地强行取代 D 位的值，这两种前缀均由 Intel 汇编程序自动加在指令前。

例如，处理机正以实方式执行时，程序设计人员需要访问 EAX 寄存器。实现这一功能的汇编程序代码可以是 MOV EAX,32bit MEMORYOP，那么 ASM386就自动确定需要某一操作数长度前缀，并将生成该前缀。

若 D 位为0，此时程序员要想用比例变址寻址方式去访问一个数组，地址长度前缀允许用 MOV DX, TABLE[ESI * 2]。汇编程序可以使用某个地址长度前缀。因为当 D=0 时，其缺省寻址方式为16位。

若 D 位为1，程序员想要存储16位的量，可用操作数长度前缀指定一个仅为16位的值：MOV MEM16,DX。

操作数长度前缀和地址长度前缀可以单独使用或结合起来用于任何一条指令。在实方式中，地址长度前缀不允许访问超过64K 字节的地址。超过 FFFFH 的有效地址会引起“一般保护故障”。地址长度前缀只允许使用存储器寻址方式中的9种寻址方式。

执行32位代码时，80386使用8位或32位位移量，并且可把任一寄存器当成基地址寄存器或变址寄存器。执行16位代码时，位移为8位和16位，且基地址寄存器与变址寄存器也符

合80286的要求。表1.7示出这些区别。

表1.7 基地址寄存器和变址寄存器

	16位寻址	32位寻址
基地址寄存器	BX,BP	任一通用32位寄存器
变址寄存器	SI,DI	除 ESP 外的任一通用32位寄存器
比例常数	无	1,2,4,8
位移量	0,8,16位	0,8,32位

1.3 数据类型

80386支持 Intel 系列机的全部数据类型。又因为80386具有功能很强的指令系统、丰富的数据类型和灵活的寻址方式，所以给编译程序设计人员提供了极大的方便。80386能支持大多数高级语言具有的基本数据类型。表1.8示出80386为这些数据类型提供的基本操作。

表1.8 80386支持的数据类型

操作	数据类型					
	整数	序数	位串	字符串	浮点	BCD
传送：取/存存储器转换精度	支持	支持	支持	支持	支持	支持②
算术运算：加、减、乘、除、求反	支持	支持	不	不	支持①	支持②
逻辑运算："与"、"或"、"异或"、移位	支持	支持	不	不	不	不
比较	支持	支持	支持	支持	支持①	支持②
超越函数	不	不	不	不	支持①	不

① 用80287或80387数值协同处理器可直接获得。

② 具有 ASCII 码和十进制校正指令用来实现有效循环。BCD 通过使用80287或80387数值协处理器可直接支持 BCD。

80386也支持带符号的8位和16位整数、32位带符号的和不带符号的整数。当使用双精度带符号的乘和除指令时，也支持64位带符号的整数，还支持从1位到32位的位字段。通过80287或80387数值协同处理器还支持64位带符号的整数。更详细的说明请参阅图1.14。

使用寄存器或立即操作数时，大多数操作都在两个时钟周期内完成。另外，又因为80386具有流水线以及两个时钟的存储器总线操作，所以写入存储器操作也必须在两个时钟内完成。

存储器的基本单位是字节，16位称为一个字，而32位称为双字。80386把字定为16位，目的在于在符记方法上与以前的 Intel 系列机兼容。

80386一个字包含存储器中相邻的两个字节，低地址的是低位字节。双字包含了存储器中4个相邻的字节。一个字的地址或一个双字的地址就是其低字节的地址。

下面详细介绍80386支持的数据类型。