CHAPTER 3 PROTECTED-MODE MEMORY MANAGEMENT

本章描述Intel 64和IA-32架构的保护模式内存管理设施，包括物理内存要求、分段机制和分页机制。

另请参阅：第5章“保护”（关于处理器保护机制的描述）和第20章“8086仿真”（关于实地址和虚拟8086模式下内存寻址保护的描述）。

3.1 MEMORY MANAGEMENT OVERVIEW

IA-32架构的内存管理设施分为两部分：分段和分页。分段提供了一种隔离独立代码、数据和堆栈模块的机制，使得多个程序（或任务）可以在同一处理器上运行而互不干扰。分页提供了一种实现传统请求分页虚拟内存系统的机制，其中程序的执行环境部分会根据需要映射到物理内存中。分页也可用于提供多个任务之间的隔离。在保护模式下运行时，必须使用某种形式的分段。没有模式位可以禁用分段。然而，分页的使用是可选的。

这两种机制（分段和分页）可被配置用于支持简单的单程序（或单任务）系统、多任务系统，或采用共享内存的多处理器系统。

如图3-1所示，分段机制提供了一种将处理器的可寻址内存空间（称为线性地址空间）划分为更小的受保护地址空间（称为段）的方法。段可用于存储程序的代码、数据和堆栈，或保存系统数据结构（如TSS或LDT）。如果处理器上运行多个程序（或任务），可以为每个程序分配自己的一组段。处理器会强制维护这些段之间的边界，并确保一个程序不会通过写入另一个程序的段来干扰其执行。分段机制还允许对段进行类型划分，从而限制对特定类型段可执行的操作。

系统中的所有段都包含在处理器线性地址空间中。要定位特定段中的字节，必须提供逻辑地址（也称为远指针）。逻辑地址由段选择子和偏移量组成。段选择子是段的唯一标识符，除其他功能外，它还提供指向描述符表（如全局描述符表GDT）中称为段描述符的数据结构的偏移量。每个段都有一个段描述符，用于指定段的大小、段的访问权限和特权级、段类型以及段第一个字节在线性地址空间中的位置（称为段的基地址）。逻辑地址的偏移量部分与段的基地址相加，以定位段内的字节。因此，基地址加上偏移量形成了处理器线性地址空间中的线性地址。

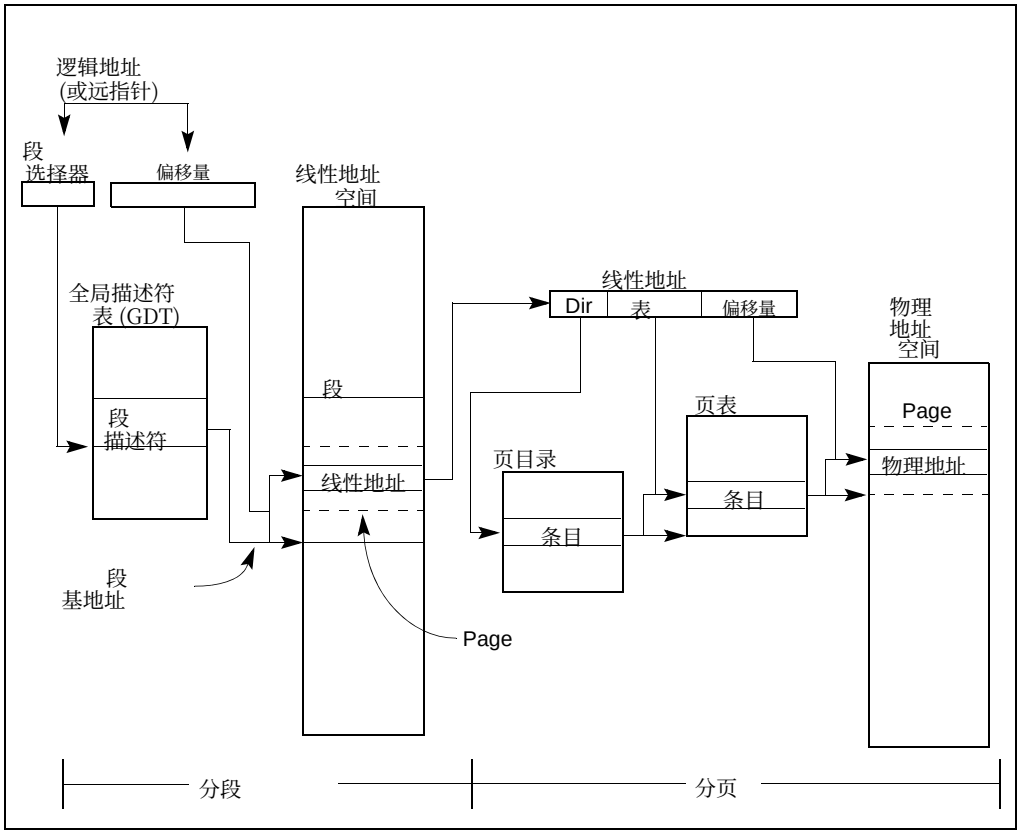

图3-1. 分段与分页

若不使用分页，处理器的线性地址空间将直接映射到处理器的物理地址空间。物理地址空间定义为处理器在其地址总线上可生成的地址范围。

由于多任务计算系统通常定义的线性地址空间远大于经济上可行的一次性装入物理内存的容量，因此需要某种“虚拟化”线性地址空间的方法。这种线性地址空间的虚拟化通过处理器的分页机制来处理。

分页支持一种“虚拟内存”环境，其中使用少量物理内存（RAM和ROM）及部分磁盘存储来模拟大型线性地址空间。使用分页时，每个段被划分为页（通常每页大小为4 KB），这些页存储在物理内存或磁盘上。操作系统或执行体维护一个页目录和一组页表来跟踪这些页。当程序（或任务）尝试访问线性地址空间中的某个地址位置时，处理器使用页目录和页表将线性地址转换为物理地址，然后对内存位置执行请求的操作（读取或写入）。

如果被访问的页当前不在物理内存中，处理器会中断程序的执行（通过产生页错误异常）。随后操作系统或执行体从磁盘将该页读入物理内存，并继续执行程序。

当在操作系统或执行体中正确实现分页时，物理内存与磁盘之间的页交换对程序的正确执行是透明的。即使是为16位IA-32处理器编写的程序，在虚拟8086模式下运行时也能（透明地）进行分页。

3.2 使用段

IA-32架构支持的分段机制可用于实现多种系统设计。这些设计范围从仅最低限度使用分段来保护

程序的平坦模型，到采用分段创建健壮操作环境的多段模型——在该环境中多个程序和任务能够可靠地执行。

以下章节将提供若干示例，说明如何在系统中运用分段机制来提升内存管理的性能与可靠性。

3.2.1 基本平坦模型

系统中最简单的内存模型是基本的“平坦模型”，在该模型中，操作系统和应用程序可访问一个连续、未分段的地址空间。这种基本平坦模型尽可能地向系统设计者和应用程序员隐藏了架构的分段机制。

要在IA-32架构中实现基本平坦内存模型，至少需要创建两个段描述符：一个用于引用代码段，另一个用于引用数据段（参见图3-2）。然而，这两个段都被映射到整个线性地址空间：即两个段描述符具有相同的基地址值0和相同的段限长4 GB。通过将段限长设置为4 GB，即使特定地址没有物理内存存在，分段机制也不会因越界内存访问而产生异常。ROM（EPROM）通常位于物理地址空间的顶部，因为处理器从FFFF_FFF0H开始执行。RAM（DRAM）则被放置在地址空间的底部，因为复位初始化后DS数据段的初始基地址为0。

3.2.2 受保护平坦模型

受保护平坦模型与基本平坦模型类似，不同之处在于段限长被设置为仅包含物理内存实际存在的地址范围（见图3-3）。任何访问不存在内存的尝试都会触发通用保护异常（#GP）。该模型提供了针对某些程序错误的最低级别硬件保护。

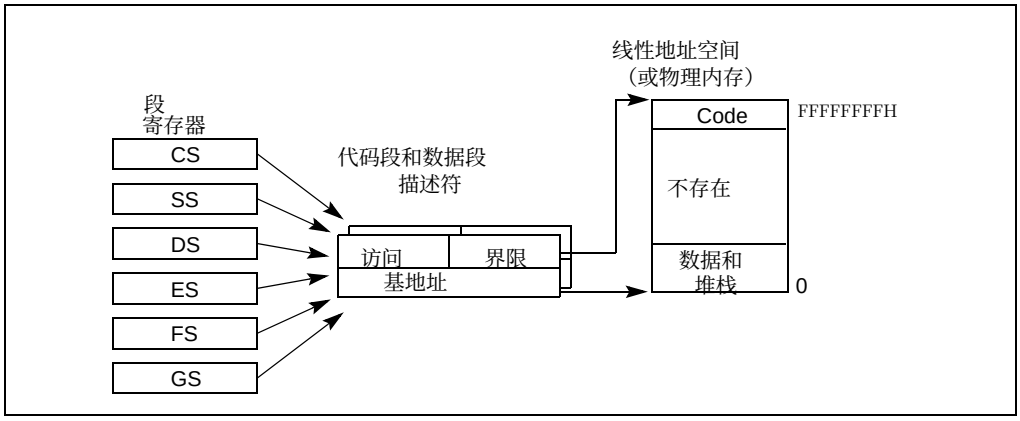

图 3-2 平坦模型

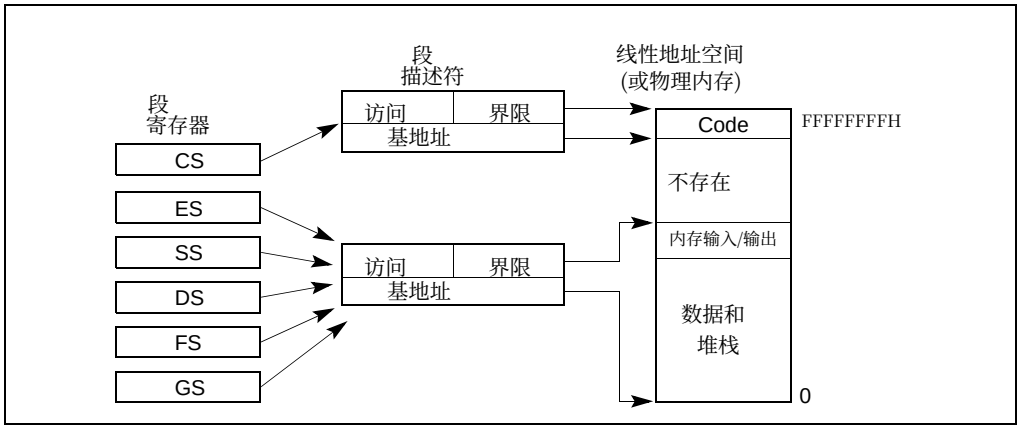

图3-3. 受保护平坦模型

可以为该受保护平坦模型增加更多复杂性以提供进一步保护。例如，若要通过分页机制实现用户与监管者代码及数据之间的隔离，需要定义四个段：特权级3的用户代码段和数据段，以及特权级0的监管者代码段和数据段。这些段通常相互重叠，并起始于线性地址空间中的地址0。这种平坦分段模型配合简单的分页结构可以保护操作系统免受应用程序影响，若为每个任务或进程添加独立的分页结构，还能实现应用程序间的相互保护。多个流行的多任务操作系统都采用了类似的设计方案。

3.2.3 多段模型

多段模型（如图3-4所示）利用分段机制的全部功能，为代码、数据结构和程序及任务提供硬件强制保护。在此模型中，每个程序（或任务）都有其自己的段描述符表和自己的段。这些段可以完全私有于其分配的程序，也可以在程序间共享。对系统中运行的所有段及单个程序执行环境的访问均由硬件控制。

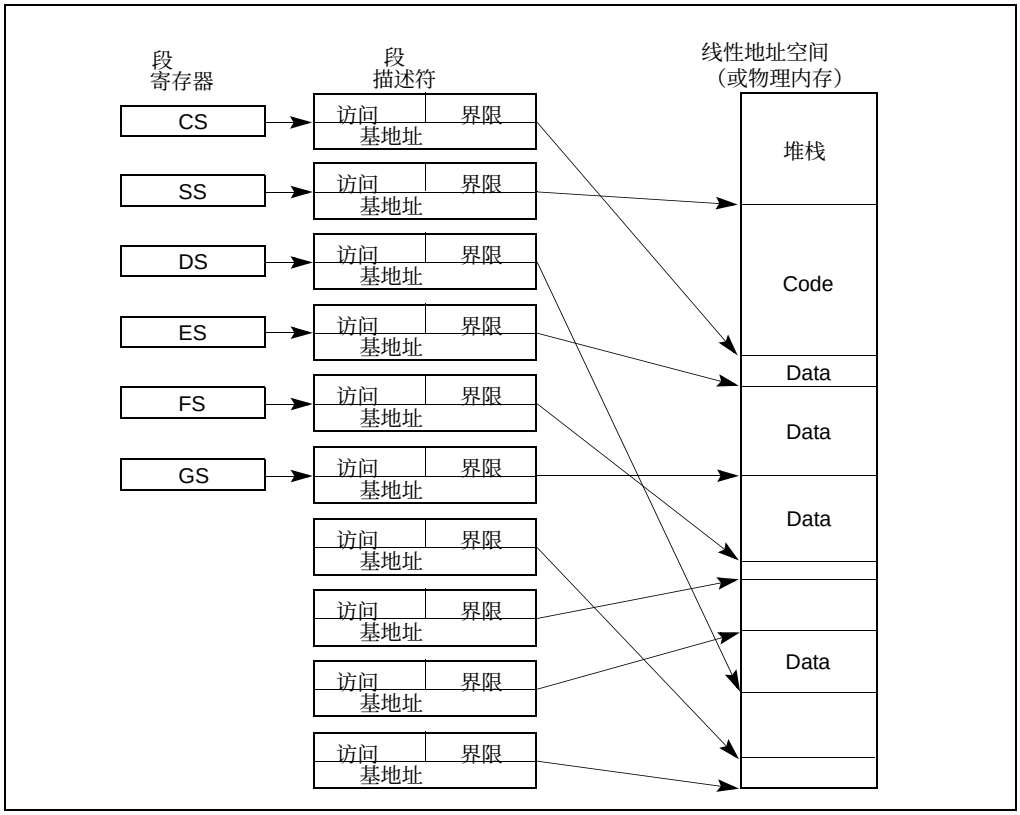

图3-4. 多段模型

访问检查不仅可用于防止引用段界限之外的地址，还可用于防止在特定段中执行不允许的操作。例如，由于代码段被指定为只读段，硬件可用于防止写入代码段。为段创建的访问权限信息也可用于设置保护环或保护级别。保护级别可用于保护操作系统过程免受应用程序的未经授权访问。

3.2.4 IA-32e模式下的分段机制

在Intel 64架构的IA-32e模式下，分段机制的效果取决于处理器是运行在兼容模式还是64位模式。在兼容模式下，分段功能与传统16位或32位保护模式语义完全相同。

在64位模式下，分段机制通常（但非完全）被禁用，从而创建一个平坦64位线性地址空间。处理器将CS、DS、ES、SS的段基址视为零，使得线性地址等于有效地址。FS和GS段属于例外情况。这些段寄存器（保存段基址）可作为线性地址计算中的附加基址寄存器使用，它们便于访问局部数据和某些操作系统数据结构。

请注意，处理器在64位模式下运行时不会执行段限检查。

3.2.5 分页与分段

分页机制可与图3-2、图3-3和图3-4所述的任何分段模型结合使用。处理器的分页机制将线性地址空间（段被映射至其中）划分为若干页（如图3-1所示）。这些线性地址空间的页随后被映射到物理地址空间中的页。分页机制提供了多种页面级保护机制，既可单独使用，也可与分段保护机制配合使用。

例如，它允许以逐页为基础实施读写保护。该分页机制还提供两级用户-监管者保护，同样可按逐页方式指定。

3.3 物理地址空间

在保护模式下，IA-32架构提供了一个常规的4 GB（2字节）物理地址空间。这是处理器能够在其地址总线上寻址的地址空间。该地址空间是平坦的（未分段），地址范围从0连续延伸到FFFFFFFFH。此物理地址空间可映射到读写存储器、只读存储器和内存映射I/O。本章描述的内存映射机制可用于将此物理内存划分为段和/或页。

从Pentium Pro处理器开始，IA-32架构还支持将物理地址空间扩展到2<style id='1'> </style>字节（64 GB），最大物理地址为FFFFFFFFFH。可通过以下两种方式之一启用此扩展功能：

- 使用 使用位于控制寄存器CR的位5中的物理地址扩展 (PAE) 标志， 4.
- 使用36位页面大小扩展 (PSE-36) 功能（在Pentium III 处理器中引入）。

物理地址支持此后已扩展到36位以上。有关36位物理寻址的更多信息，请参阅第4章“分页”。

3.3.1 Intel® 64 处理器与物理地址空间

在支持Intel 64 架构的处理器上（CPUID.80000001:EDX[29] = 1），物理地址范围的大小是具体实现相关的，并由CPUID.80000008H:EAX[的位7-0]指示。

有关EAX中返回信息的格式，请参阅《Intel® 64 和 IA-32 架构软件开发人员手册》第2A卷第3章中的“CPUID—CPU识别”。另请参阅：第4章“分页”。

3.4 逻辑地址与线性地址

在保护模式的系统架构级别，处理器使用两个阶段的地址转换来获得物理地址：逻辑地址转换和线性地址空间分页。

即使以最小化方式使用段，处理器地址空间中的每个字节都是通过逻辑地址访问的。逻辑地址由16位段选择子和32位偏移量组成（见图3-5）。段选择子用于标识字节所在的段，偏移量则指定字节在段中相对于段基地址的位置。

处理器将每个逻辑地址转换为线性地址。线性地址是处理器线性地址空间中的32位地址。与物理地址空间类似，线性地址空间是一个平坦（未分段）的2^32字节地址空间，地址范围从0到FFFFFFFFH。线性地址空间包含为系统定义的所有段和系统表。

为了将逻辑地址转换为线性地址，处理器执行以下操作：

1. 使用段选择子中的偏移量在GDT或LDT中定位段的段描述符 d
将其读取到处理器中。（仅当新的段选择子被加载到段寄存器时才需要此步骤。）2. 检查段描述符以验证段的访问权限和范围，确保该段可访问且偏移量在段界限内。3. 将段描述符中的段基地址与偏移量相加，形成线性地址。

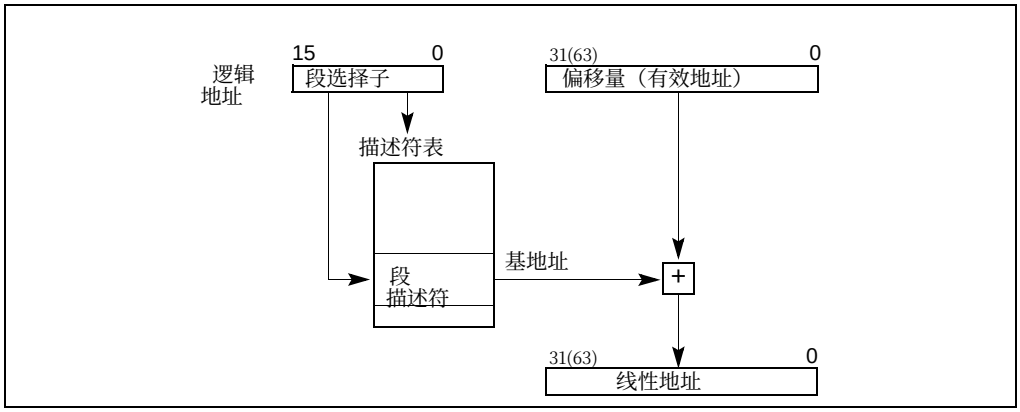

图3-5. 逻辑地址到线性地址的转换

若未启用分页机制，处理器会将线性地址直接映射为物理地址（即线性地址通过处理器的地址总线输出）。若线性地址空间启用了分页，则通过第二级地址转换将线性地址转换为物理地址。

另请参阅：第4章“分页”。

3.4.1 IA-32e模式中的逻辑地址转换

在IA-32e模式下，Intel 64处理器使用上述步骤将逻辑地址转换为线性地址。在64位模式下，段的偏移量和基地址为64位而非32位。线性地址格式也为64位宽，并需满足规范形式要求。

每个代码段描述符都提供了一个L位。该位允许代码段按代码段执行64位代码或传统32位代码。

3.4.2 段选择子

段选择子是一个16位的段标识符（参见图3-6）。它并不直接指向段，而是指向定义该段的段描述符。段选择子包含以下内容：

索引　　（位3至15）——从GDT或LDT的8192个描述符中选择一个。处理器将索引值乘以8（段描述符的字节数），并将结果与GDT或LDT的基地址（分别来自GDTR或LDTR寄存器）相加。

TI（表指示符）标志（位2）——Sp

指定要使用的描述符表：清除此标志选择GDT；设置此标志选择当前LDT。

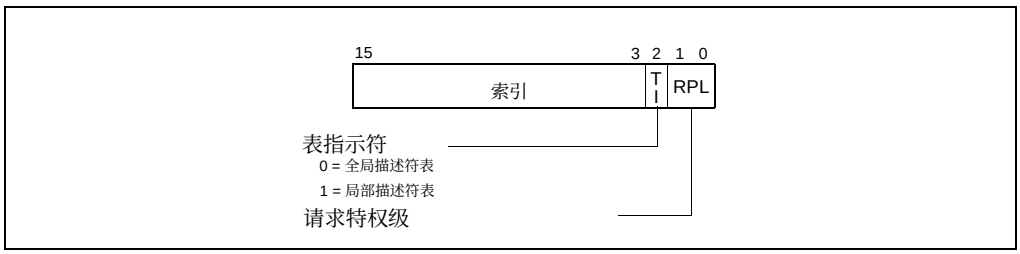

图 3-6. 段选择子

请求特权级（RPL）（位0和位1）— 规范

指定选择器的特权级。特权级范围从0到3，其中0为最高特权级。有关RPL与正在执行的程序（或任务）的CPL以及段选择子所指向的描述符的描述符特权级（DPL）之间关系的说明，请参阅第5.5节“特权级”。

GDT的第一个条目不被处理器使用。指向GDT此条目的段选择子（即索引为0且TI标志设置为0的段选择子）被用作“空段选择子”。当段寄存器（CS或SS寄存器除外）加载空选择子时，处理器不会生成异常。然而，当持有空选择子的段寄存器用于访问内存时，处理器会生成异常。空选择子可用于初始化未使用的段寄存器。用空段选择子加载CS或SS寄存器会导致生成通用保护异常（#GP）。

段选择子作为指针变量的一部分对应用程序可见，但选择子的值通常由链接编辑器或链接加载器分配或修改，而非由应用程序修改。

3.4.3 段寄存器

为减少地址转换时间和编码复杂度，处理器提供了用于保存最多6个段选择子的寄存器（见图3-7）。每个段寄存器支持特定类型的内存引用（代码、堆栈或数据）。virtually任何程序执行要发生，至少需要将代码段（CS）、数据段（DS）和堆栈段（SS）寄存器加载有效的段选择子。处理器还提供三个额外的数据段寄存器（ES、FS和GS），可用于使当前执行的程序（或任务）能够访问额外的数据段。

程序要访问某个段，该段的段选择子必须已加载到某个段寄存器中。因此，虽然系统可以定义数千个段，但只有6个能够立即使用。其他段可以通过在程序执行期间将其段选择子加载到这些寄存器中来变为可用。

图3-7 段寄存器

每个段寄存器都有一个“可见部分”和一个“隐藏部分”。（隐藏部分有时被称为“描述符缓存”或“影子寄存器”。）当段选择子被加载到段寄存器的可见部分时，处理器还会将段选择子所指向的段描述符中的基地址、段限长和访问控制信息加载到段寄存器的隐藏部分。缓存在段寄存器（可见和隐藏）中的信息使处理器能够转换地址，而无需额外的总线周期从段描述符中读取基地址和界限。在多个处理器访问同一描述符表的系统中，软件有责任在修改描述符表时重新加载段寄存器。如果不这样做，则可能在内存驻留版本的段描述符被修改后，仍使用缓存在段寄存器中的旧段描述符。

提供了两种加载指令用于加载段寄存器：

1. 直接加载指令，例如MOV指令、POP指令、LDS指令、LES指令、LSS指令、LGS指令和LFS指令。这些指令显式引用段寄存器。

2. 隐式加载指令，例如CALL、JMP和RET指令的远指针版本，SYSENTER和SYSEXIT指令，以及IRET、INTn、INTO和INT3指令。这些指令在执行过程中会附带改变CS寄存器（有时还包括其他段寄存器）的内容。

MOV指令也可用于将段寄存器的可见部分存储到通用寄存器中。

3.4.4 IA-32e模式中的段加载指令

由于ES、DS和SS段寄存器在64位模式下不被使用，它们在段描述符寄存器中的字段（基址、界限和属性）将被忽略。某些形式的段加载指令也无效（例如LDS指令、POP ES）。引用ES、DS或SS段的地址计算会被视为段基址为零来处理。

处理器会检查所有线性地址引用是否均为规范形式，而非执行界限检查。模式切换不会改变段寄存器或相关描述符寄存器的内容。这些寄存器在64位模式执行期间也不会改变，除非执行了显式的段加载操作。

为了为应用程序设置兼容模式，段加载指令（MOV到段寄存器、POP段寄存器）在64位模式下正常工作。从系统描述符表（GDT或LDT）中读取一个条目，并加载到段描述符寄存器的隐藏部分。描述符寄存器的基地址、界限和属性字段都会被加载。但是，数据和堆栈段选择器及描述符寄存器的内容将被忽略。

当在64位模式下使用FS和GS段超越时，它们各自的基地址将用于线性地址计算：(FS或GS).base + 索引 + 位移量。随后FS.base和GS.base会扩展至实现所支持的完整线性地址大小。最终的有效地址计算可跨越正负地址环绕；生成的线性地址必须是规范地址。

在64位模式下，使用FS段和GS段超越的内存访问不会检查运行时界限，也不会进行属性检查。正常的段加载（MOV到段寄存器和POP段寄存器）到FS和GS时，会在段描述符寄存器的隐藏部分加载标准的32位基址值。高于标准32位的基地址位会被清零，以确保使用少于64位的实现保持一致性。

FS.base 和 GS.base 的隐藏描述符寄存器字段被物理映射到模型特定寄存器，以便加载64位实现支持的所有地址位。具有当前特权级 = 0 （特权软件）的软件可以使用WRMSR指令将所有支持的线性地址位加载到 FS.base 或 GS.base 中。写入64位 FS.base 和 GS.base 寄存器的地址必须采用规范形式。试图向这些寄存器写入非规范地址的 WRMSR 指令将导致#GP故障。

在兼容模式下，无论加载到隐藏描述符寄存器基址字段的高32位线性地址位的值如何，FS 和 GS 覆盖操作都按照32位模式行为定义执行。兼容模式在计算有效地址时会忽略高32位。

新增的64位模式指令 SWAPGS 可用于加载 GS 基址。SWAPGS 将来自 IA32_KernelGSbase 模型特定寄存器的内核数据结构指针与 GS 基址寄存器进行交换。随后内核可在常规内存引用中使用 GS 前缀来访问内核数据结构。尝试向 IA32_KernelGSBase 模型特定寄存器写入非规范值（使用 WRMSR）将导致#GP故障。

3.4.5 段描述符

段描述符是GDT或LDT中的一种数据结构，它为处理器提供段的大小和位置，以及访问控制和状态信息。段描述符通常由编译器、链接器、加载器或操作系统/执行体创建，而非应用程序。图3-8展示了所有类型段描述符的通用描述符格式。

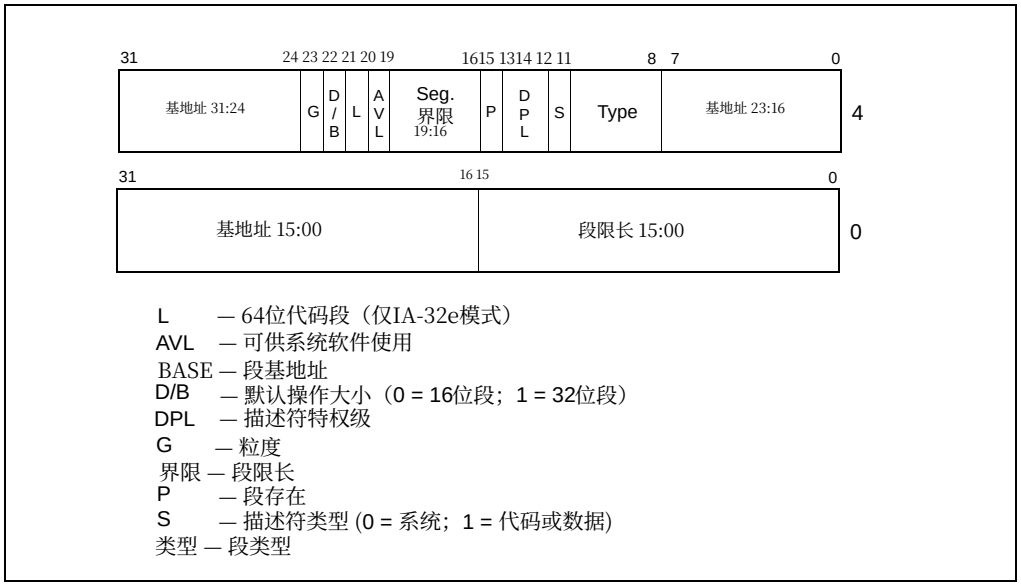

图3-8 段描述符

段描述符中的标志位和字段如下：

段限界字段说明

用于指定段的大小。处理器将两个段限界字段组合形成一个20位的值。根据G（粒度）标志的设置，处理器以两种方式之一解释段限界：

- 如果粒度标志未设置，段大小范围可从1字节到1MB，以字节为单位递增。
- 如果粒度标志已设置，段大小范围可从4KB到4GB，以4KB为单位递增。

处理器以两种不同方式使用段限界，具体取决于该段是向上扩展段还是向下扩展段。有关段类型的更多信息，请参阅第3.4.5.1节“代码段与数据段描述符类型”。对于向上扩展段，逻辑地址中的偏移量范围可从0至段限界。超过段限界的偏移量将引发通用保护异常（#GP，适用于除SS段外的所有段）或堆栈故障异常（#SS适用于SS段）。对于向下扩展段，段限界发挥反向作用：偏移量范围可从段限界加1至FFFFFFFFH或FFFFH，具体取决于B标志的设置。小于或等于段限界的偏移量会引发通用保护异常或堆栈故障异常。减小向下扩展段的段限界字段值会在段地址空间的底部（而非顶部）分配新内存。IA-32架构的堆栈始终向下增长，这使得该机制非常适合可扩展堆栈。

基地址字段定义

段内字节0在4GB线性地址空间中的位置。处理器将三个基地址字段组合形成单个32位值。段基地址应对齐到16字节边界。虽然不要求必须16字节对齐，但这种对齐方式允许程序通过将代码和数据对齐到16字节边界来最大化性能。

类型字段　指示段或门类型，并指定可对段进行的访问类型及增长方向。该字段的解释取决于描述符类型标志指定的是应用程序（代码或数据）描述符还是系统描述符。对于代码、数据和系统描述符，类型字段的编码方式各不相同（见图5-1）。关于如何使用此字段指定代码段和数据段类型的描述，请参阅第3.4.5.1节“代码段与数据段描述符类型”。

S（描述符类型）标志 指定

段描述符是用于系统段（S标志清零）还是代码段或数据段（S标志置位）。

DPL（描述符特权级）字段 指定段的特权级

。特权级范围从0到3，其中0为最高特权级。DPL用于控制对段的访问。有关DPL与执行代码段的CPL及段选择子的RPL之间关系的描述，请参见第5.5节“特权级”。

P（段存在）标志 指示

该段是否存在于内存中（置位）或不存在（清除）。如果该标志位被清除，当指向该段描述符的段选择子被加载到段寄存器时，处理器会产生一个段不存在异常（#NP）。内存管理软件可以利用此标志来控制哪些段在特定时刻实际加载到物理内存中。这为管理虚拟内存提供了除分页之外的额外控制手段。

图3-9展示了当段存在标志位被清除时段描述符的格式。当该标志位被清除时，操作系统或执行体可以自由使用标记为“可用”的位置来存储自身数据，例如关于缺失段位置的信息。

D/B（默认操作大小/默认堆栈指针大小和/或上界）标志 根据段描述符类型执行不同功能

对于可执行代码段、向下扩展数据段或堆栈段，该标志的功能各不相同。（对于32位代码和数据段，此标志应始终设置为1；对于16位代码和数据段，则应设置为0。）

- 可执行代码段。该标志称为D标志，它指示段内指令引用的有效地址和操作数的默认长度。如果该标志被设置，则假定为32位地址和32位或8位操作数；如果该标志被清除，则假定为16位地址和16位或8位操作数。指令前缀66H可用于选择非默认的操作数大小，前缀67H可用于选择非默认的地址大小。

- 堆栈段（由SS寄存器指向的数据段）。该标志称为B（大）标志，它指定用于隐式堆栈操作（如压栈、出栈和调用）的堆栈指针大小。如果该标志被设置，则使用32位堆栈指针，存储在32位ESP寄存器中；如果该标志被清除，则使用16位堆栈指针，存储在16位SP寄存器中。如果堆栈段被设置为向下扩展数据段（在下一段中描述），B标志还指定堆栈段的上界。

- 向下扩展数据段。该标志称为B标志，它指定了段的上界。如果设置了该标志，上界为FFFFFFFFH（4 GB）；如果清除了该标志，上界为FFFFH（64 KB）。

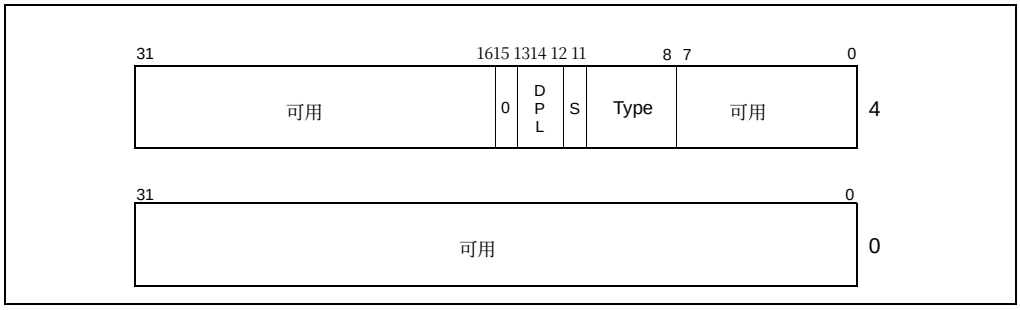

图3-9. 段存在标志位清除时的段描述符

G（粒度）标志确定

定义了段限界字段的缩放比例。当粒度标志清零时，段限长以字节单位解释；当该标志置位时，段限长以4KB单位解释。（此标志不影响基地址的粒度；基地址始终以字节粒度表示。）当粒度标志置位时，检查偏移量与段限长的匹配性时，偏移量的12个最低有效位不会被检测。

例如，当粒度标志置位时，限长为0将使得从0到4095的偏移量均有效。

L（64位代码段）标志 在IA-32e模式下，

段描述符第二个双字的第21位指示代码段是否包含原生64位代码。值为1表示此代码段中的指令在64位模式下执行。值为0表示此代码段中的指令在兼容模式下执行。如果设置了L位，则必须清除D位。当不在IA-32e模式下或对于非代码段时，第21位被保留且应始终设置为0。

可用与保留位 段描述符第二个双字的第20位

可供系统软件使用。

3.4.5.1 代码段与数据段描述符类型

当段描述符中的S（描述符类型）标志被设置时，该描述符用于代码段或数据段。此时类型字段的最高有效位（段描述符第二个双字的第11位）决定了该描述符是用于数据段（清零）还是代码段（置位）。

对于数据段，类型字段的三个低有效位（第8、9和10位）被解释为访问位 (A)、写使能 (W) 和扩展方向 (E)。有关代码段和数据段类型字段位编码的说明，请参见表3-1。根据写使能位的设置，数据段可以是只读段或读/写段。

表3-1. 代码段与数据段类型

类型字段					描述符 Type	描述
十进制	11	10 E	9 W	8 A		
0	0	0	0	0	Data	只读
1	0	0	0	1	Data	只读，已访问
2	0	0	1	0	Data	读/写
3	0	0	1	1	Data	读/写 ，已访问
4	0	1	0	0	Data	只读, 向下扩展
5	0	1	0	1	Data	只读, 向下扩展, 已访问
6	0	1	1	0	Data	读/写, 向下扩展
7	0	1	1	1	Data	读/写, 向下扩展, 已访问
		C	R	A		
8	1	0	0	0	Code	仅执行
9	1	0	0	1	Code	仅执行，已访问
10	1	0	1	0	Code	执行/读取
11	1	0	1	1	Code	执行/读取 ，已访问
12	1	1	0	0	Code	仅执行，符合性
13	1	1	0	1	Code	仅执行，符合性，已访问
14	1	1	1	0	Code	执行/读取，符合性
15	1	1	1	1	Code	执行/读取，符合性，已访问

堆栈段是必须为读/写段的数据段。使用不可写数据段的段选择子加载SS寄存器会引发通用保护异常（#GP）。如果需要动态更改堆栈段的大小，该堆栈段可以是向下扩展数据段（扩展方向标志已设置）。此时，动态更改段限长会将堆栈空间添加到

堆栈底部。如果堆栈段的大小需要保持静态，则该堆栈段可以是向上扩展或向下扩展类型。

访问位指示自上次操作系统或执行体清除该位以来，该段是否已被访问。当处理器将段的段选择子加载到段寄存器时，假设包含段描述符的内存类型支持处理器写入，便会设置该位。该位将保持设置状态，直到被显式清除。此位既可用于虚拟内存管理，也可用于调试。

对于代码段，类型字段的低三位被解释为已访问 (A)、读使能 (R) 和符合性 (C)。代码段可以是仅执行或执行/读取，具体取决于读使能位的设置。当常量或其他静态数据与指令代码一起放置在ROM中时，可能会使用执行/读取段。在这种情况下，可以通过使用带有CS覆盖前缀的指令，或者通过将代码段的段选择子加载到数据段寄存器（DS、ES、FS或GS寄存器）中，从代码段读取数据。在保护模式下，代码段不可写入。

代码段可以是符合性或非一致类型。向更高特权级的符合性段转移执行时，允许在当前特权级继续执行。而向不同特权级的非一致段转移执行将导致通用保护异常（#GP），除非使用调用门或任务门（有关符合性和非一致代码段的更多信息，请参阅第5.8.1节“直接调用或跳转至代码段”）。不访问受保护设施的系统工具以及某些异常类型（例如除法错误或溢出）的处理程序，可被加载到符合性代码段中。需要防范低特权级程序和过程访问的实用程序应置于非一致代码段中。

NOTE

无论目标段是符合性还是非一致代码段，都无法通过调用或跳转将执行权转移至更低特权级（数值上更高的特权级）的代码段。尝试此类执行转移将触发通用保护异常。

所有数据段都是非一致的，这意味着它们不能被特权级较低的程序或过程（在数值上较高特权级执行的代码）访问。然而，与代码段不同，数据段可以被特权级较高的程序或过程（在数值上较低特权级执行的代码）访问，而无需使用特殊的访问门。

如果GDT或某个LDT中的段描述符被放置在ROM中，当软件或处理器尝试更新（写入）这些基于ROM的段描述符时，处理器可能会进入无限循环。为防止此问题，应为所有放置在ROM中的段描述符设置访问位。同时，移除那些试图修改位于ROM中的段描述符的操作系统或执行体代码。

3.5 系统描述符类型

当段描述符中的S（描述符类型）标志被清除时，该描述符类型为系统描述符。处理器可识别以下类型的系统描述符：

- 局部描述符表（LDT）段描述符
- 任务状态段（TSS）描述符。
- 调用门描述符。
- 中断门描述符。
- 陷阱门描述符。
- 任务门描述符。

r.

这些描述符类型分为两类：系统段描述符和门描述符。系统段描述符指向系统段（LDT和TSS段）。门描述符本身即是“门”，它们持有指向代码段中过程入口点的指针（调用门、中断门和陷阱门），或者持有TSS的段选择子（任务门）。

表3-2展示了系统段描述符和门描述符的类型字段编码。请注意，IA-32e模式下的系统描述符为16字节而非8字节。

表 3-2. 系统段和门描述符类型

类型字段					描述	
十进制	11	10	9	8	32位模式	IA-32e模式
0	0	0	0	0	保留	16字节描述符的高8字节描述符
1	0	0	0	1	16位TSS（可用）	保留
2	0	0	1	0	LDT	LDT
3	0	0	1	1	16位TSS（繁忙）	保留
4	0	1	0	0	16位调用门	保留
5	0	1	0	1	任务门	保留
6	0	1	1	0	16位中断门	保留
7	0	1	1	1	16位陷阱门	保留
8	1	0	0	0	保留	保留
9	1	0	0	1	32位任务状态段（可用）	64位任务状态段（可用）
10	1	0	1	0	保留	保留
11	1	0	1	1	32位任务状态段（忙碌）	64位任务状态段（忙碌）
12	1	1	0	0	32位调用门	64位调用门
13	1	1	0	1	保留	保留
14	1	1	1	0	32位中断门	64位中断门
15	1	1	1	1	32位陷阱门	64位陷阱门

另请参阅：第3.5.1节“段描述符表”和第7.2.2节“TSS描述符”（有关系统段描述符的更多信息）；请参阅第5.8.3节“调用门”、第6.11节“IDT描述符”和第7.2.5节“任务门描述符”（有关门描述符的更多信息）。

3.5.1 段描述符表

段描述符表是段描述符的数组（参见图3-10）。描述符表的长度可变，最多可包含8192（213）个8字节描述符。描述符表有两种类型：

- 全局描述符表 (GDT)
- 局部描述符表 (LDT)

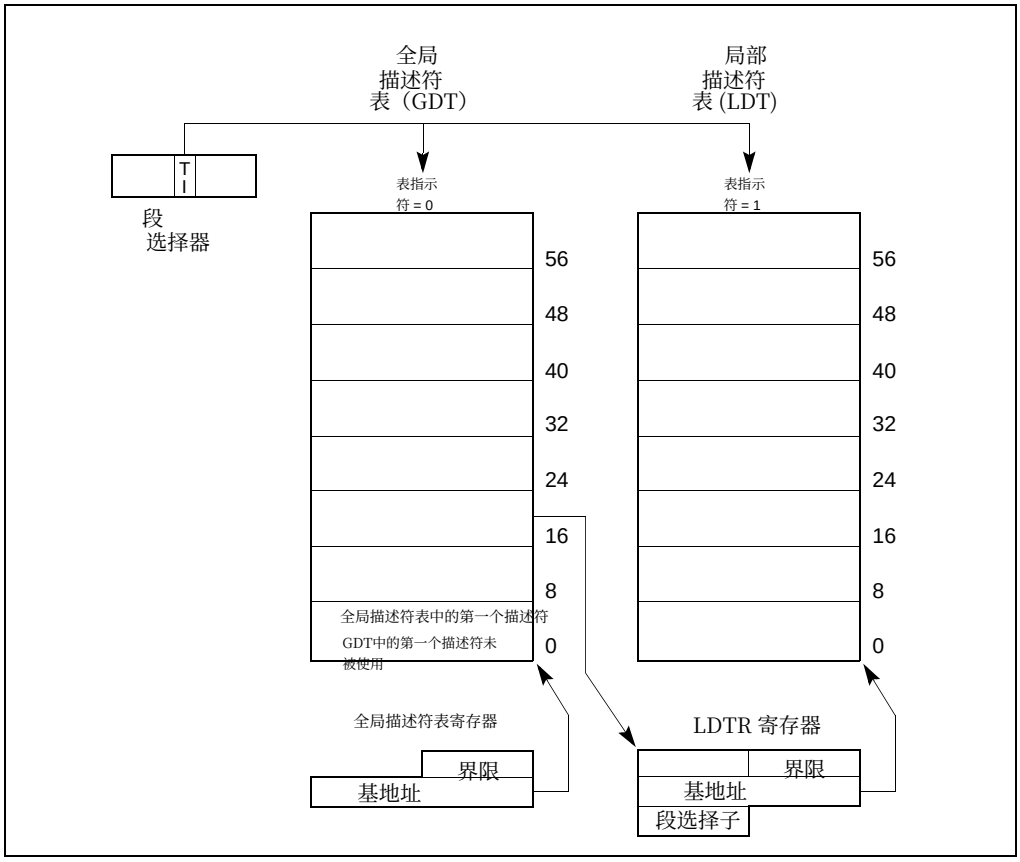

图 3-10. 全局和局部描述符表

每个系统必须定义一个GDT，可供系统中所有程序和任务使用。此外，还可以选择定义一个或多个LDT。例如，可以为每个正在运行的独立任务定义LDT，或者让部分或全部任务共享同一个LDT。

GDT本身并非一个段，而是线性地址空间中的一个数据结构。GDT的基线性地址和界限必须加载到GDTR寄存器中（参见第2.4节“内存管理寄存器”）。GDT的基地址应对齐到8字节边界以获得最佳处理器性能。GDT的界限值以字节为单位。与段类似，界限值会与基地址相加以得到最后一个有效字节的地址。界限值为0时恰好对应一个有效字节。由于段描述符始终为8字节长，GDT的界限值应始终为8的整数倍减一（即8N–1）。

GDT中的第一个描述符不被处理器使用。指向这个“空描述符”的段选择子被加载到数据段寄存器（DS、ES、FS或GS）时不会产生异常，但当尝试使用该描述符访问内存时，总是会引发通用保护异常（#GP）。通过用这个段选择子初始化段寄存器，可以确保对未使用段寄存器的意外引用必定会触发异常。

LDT 位于 LDT 类型的系统段中。GDT 必须包含 LDT 段的段描述符。如果系统支持多个 LDT，则每个 LDT 在 GDT 中都必须有独立的段选择子和段描述符。LDT 的段描述符可以位于 GDT 中的任意位置。有关 LDT 段描述符类型的信息，请参阅第 3.5 节“系统描述符类型”。

通过其段选择子来访问 LDT。为避免访问 LDT 时进行地址转换，LDT 的段选择子、基线性地址、界限和访问权限都存储在 LDTR 寄存器中（参见第 2.4 节“内存管理寄存器”）。

当存储GDTR寄存器（使用SGDT指令）时，一个48位“伪描述符”将被存入内存（参见图3-11顶部图示）。为避免用户模式（特权级3）下的对齐检查故障，该伪

描述符应位于奇数字地址（即地址MOD 4等于2）。这将使处理器存储一个对齐字，后跟一个对齐双字。用户模式程序通常不存储伪描述符，但通过以这种方式对齐伪描述符，可以避免产生对齐检查故障的可能性。在使用SIDT指令存储IDTR寄存器时，应采用相同的对齐方式。当存储LDTR或任务寄存器时（分别使用SLDT或STR指令），伪描述符应位于双字地址（即地址MOD 4等于0）。

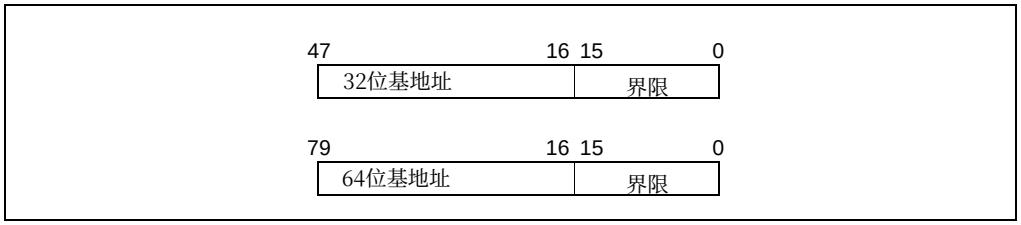

图3-11 伪描述符格式

3.5.2 IA-32e模式中的段描述符表

在IA-32e模式下，段描述符表最多可包含8192（213）个8字节描述符。段描述符表中的条目可以是8字节。系统描述符扩展为16字节（占用两个条目的空间）。

GDTR和LDTR寄存器扩展为可容纳64位基地址。相应的伪描述符为80位。（参见图3-11底部图示）。

以下系统描述符扩展为16字节：

— 调用门描述符（参见第5.8.3.1节“IA-32e模式调用门”）— IDT门描述符（参见第6.14.1节“64位模式IDT”）— LDT和TSS描述符（参见第7.2.3节“64位模式下的TSS描述符”）。