段式管理的数据结构

段式管理的数据结构

16 位 CPU

intel 处理器从 4 位机到最新的 64 位，历时发展数十年，从 8086 开始，Intel CPU 正式进入 x86 时代，而段寄存器也是这个时候诞生。

8086 处理器位数变成 16 位，但是地址总线又变成了 20 根。为了能够访问到整个地址空间，在 CPU 里添加了 4 个段寄存器，分别为 CS（代码段寄存器）DS（数据段寄存器） SS（堆栈段寄存器）ES(扩展段寄存器)。所以**段寄存器就是为了解决 CPU 位数和地址总线不同的问题而诞生的。**

那么如何用 16 位 CPU 访问 20 根地址线呢？x86 将物理内存划分很多段（所有的操作都是在真实的物理内存上，当时没有操作系统这么一说，这样就是所谓的实模式），段基地址采用两字节对齐(16 的倍数)，即 16 位的段寄存器只需要记录地址的高 16 位，ip 寄存器记录段内偏移量。因此，进程要访问的**线性地址 = (段寄存器 « 4) + (ip 寄存器(偏移量))**。

32 位 CPU

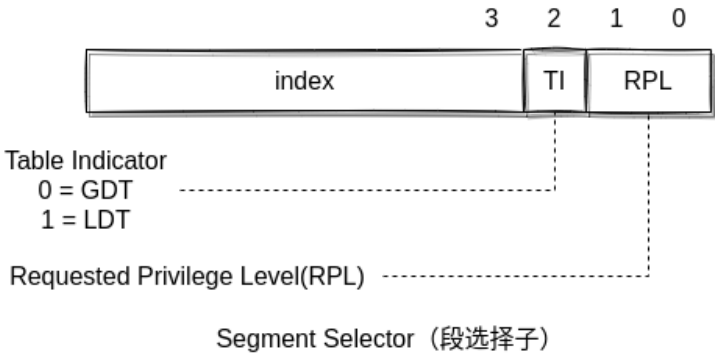
到了 80386 的时候（保护模式），这时候 cpu 是 32 位，地址总线变成了 32 根，除了先前的 4 个段寄存器，还引入了两个新的段寄存器 FS、GS（附加数据段寄存器）。但是它的寄存器大小为了兼容之前体系下的版本，寄存器依旧是 16 bits，这么说寻址能力又不能满足了。这个时候增加了两个寄存器，GDTR（全局的段的描述附表），LDTR（局部的描述附表），新增的寄存器可以不和上个版本兼容不是 16 位，是 32 位。

在 x86 保护模式下，段的信息（段基线性地址、长度、权限等）即**段描述符**占 8 个字节，段信息无法直接存放在段寄存器中（段寄存器只有 2 字节）。Intel 的设计是段描述符集中存放在 GDT(Global Descriptor Table)或 LDT(Local Descriptor Table)中，其基址分别保存在新增的寄存器 GDTR 和 LDTR 中。**同时 16 位 CPU 中的段寄存器变成段选择子，CS DS SS ES 存放的是段描述符在 GDT 或 LDT 内的索引值(index)**。然后新的段寄存器是段选择子增加了 Invisible 部分，能够存储整个短信息。

下面着重介绍 32 位段式管理的相关数据结构。

Segment Selector

Segment Selector 用于在 Descriptor Table（描述符表）中查找 descriptor（描述符），在 X86 中有三类描述符表：GDT(Global Descriptor Talbe), LDT(Local Descripotr), IDT(Interrupt Descriptor Table)。



Segment Selector（段选择子）

- RPL：当前的权限级别。CPL 的值放在 CS 寄存器的 Selector 域的 RPL，CS.Selector.RPL 与 SS 寄存器的 Selector.RPL 总是相等的，因此 SS.Selector.RPL 也是 CPL。
- TI(Table Indicator)：描述符索引位。当 TI = 0 时，从 GDT 查找；当 TI = 1 时，从 LDT 查找。
- Index(Descriptor Index)：这是 Descriptro 在 GDT/LDT 中的序号，根据 TI 的值在相应的描述符表中查找 descriptor。

例如：当有如下 selector 时，

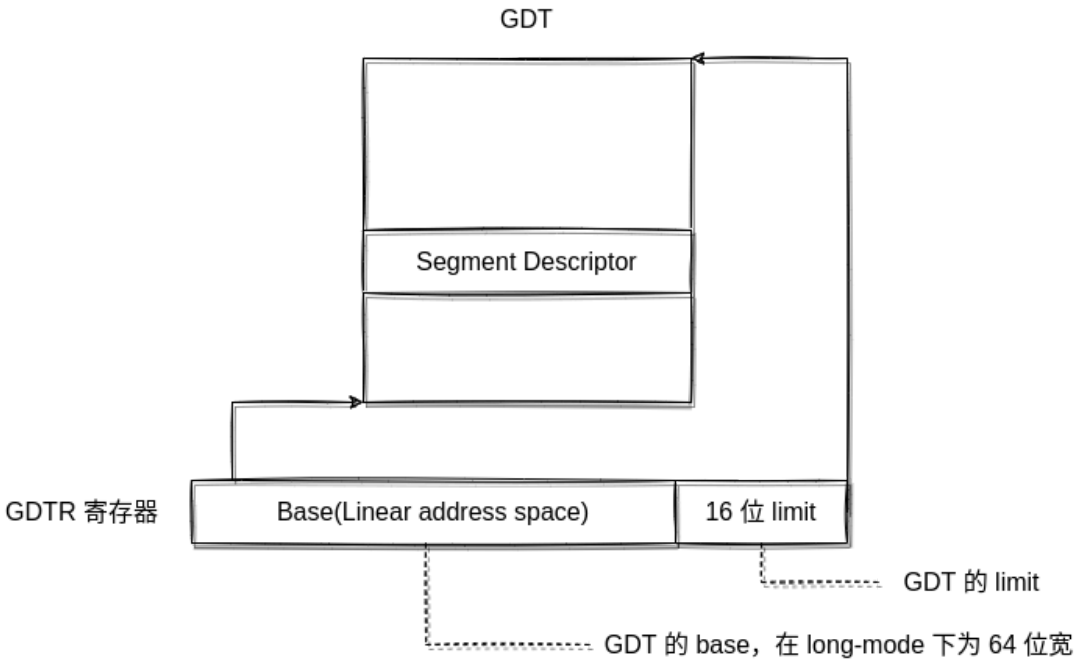
```
selector = 0008H                ;RPL = 0, TI = 0, Index = 1
```

表示在 GDT 的第 1 项得到 Descriptor，访问者使用的权限是 0 级。

GDTR

我们知道在 X86 中有三类描述符表：GDT(Global Descriptor Talbe), LDT(Local Descripotr), IDT(Interrupt Descriptor Table)。这些 descriptor table 由 descriptor table register（描述符表寄存器）进行定位，因此三种描述符表就有三种描述符表寄存器：GDTR，LDTR 和 IDTR。

GDTR 的 limit 域是 16 位值，最大的 limit 是 FFFFH，base 可以在处理器 linear address 空间的任何位值。

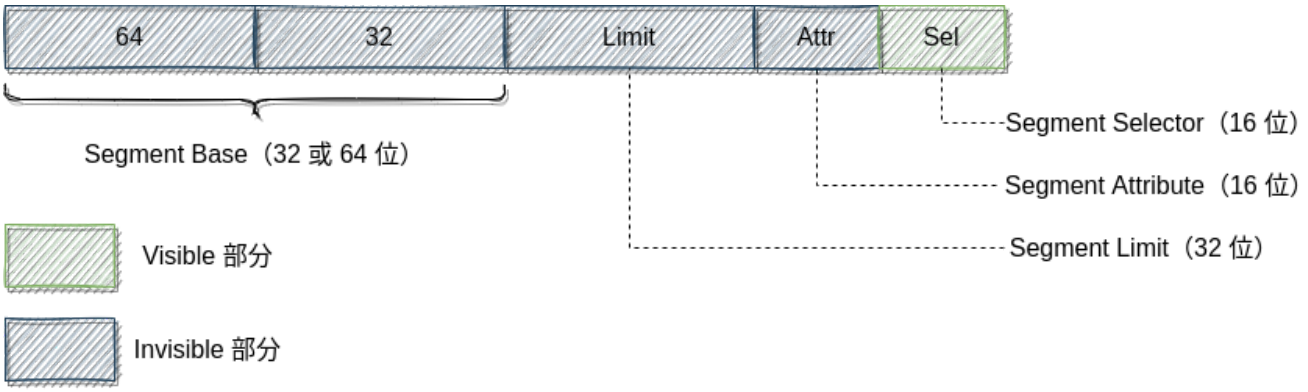


如上图所示，GDTR.base 提供 GDT 的基地值，GDTR.limit 提供 GDT 表限。使用上面的 segment selector 对 GDT 进行访问时，处理器会检查 selector 是否超出 GDT 的 limit，若 GDT 的 limit 值为 0x3FFFH，那么 GDT 内的有效范围就是 0 ~ 0x3FFH。要注意 selector 每次访问的是 8 字节。例如，当 GDT 的 limit 值为 0x0C6H 时，下列情形就超出了 limit 范围：

```
mov ax, 0xc0            ; selector 为 c0h
mov ds, ax                      ; #GP 异常
```

这个 selector 的 Index 是 0xC0H，所访问空间是 0xC0H 到 0xC7H，而 GDT 的 limit 值为 0x0C6H，这超出了 GDT 的 limit，将引发 #GP 异常。

Segment Selector Register

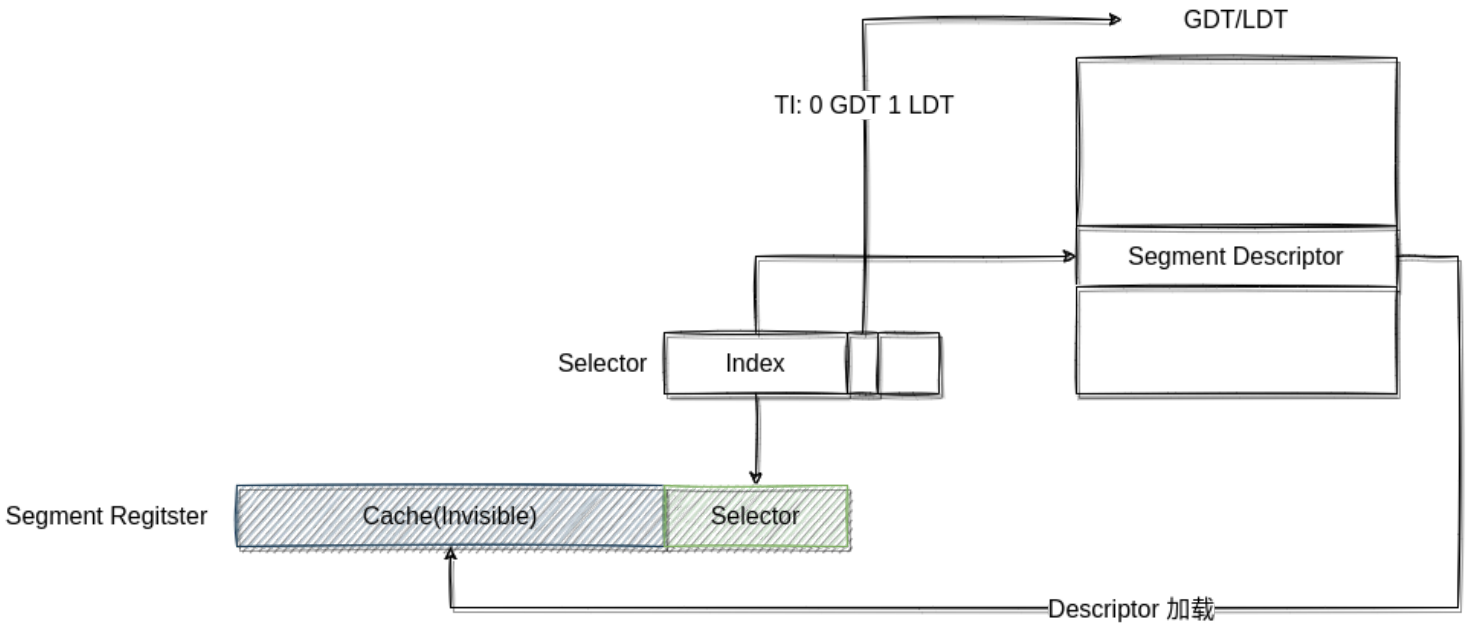


Segment Selector Register（段寄存器）

段寄存器有时被表述为段选择子寄存器，包括：Visible 和 Invisible 两部分。Invisible 部分隐藏在寄存器内部只有处理器可以用，有时也被称为 Cache 部分。段寄存器有诸多域，每个域的功能如下：

- base 域：提供段的基址；
- limit 域：提供段限，这个 32 位的段限是从 Segment descriptor 计算而来，Segment descriptor 里提供的 limit 域是 20 位宽的，加载到段寄存器后被计算成 32 位；
- attribute 域：分别由 Segment descriptor 的 Type, S, DPL, P, G, D/B, L 以及 AVL 域组合而来；
- selector 域：使用 selector 加载新的段时，selector 会被加载到段寄存器的 selector 域。

下面是一个典型的段描述符加载到段寄存器的示意图。



Load Segment Descriptor

当段寄存器发生加载时，根据 Selector 得到 segment descriptor，Selector 将加载到段寄存器的 Selector 域，同时 Segment Descriptor 也将加载到段寄存器的 Invisible 部分(Cache)。

Segment Descriptor

在整个 X86/64 体系中段寄存器和段描述符非常重要。在保护模式中，段描述符只有加载到段寄存器里才能发挥应有的作用，当一个段描述符被加载到段寄存器后，它所描述的段变成 active 状态。

段描述符可以分成以下几类：

- System descriptor
  - System Segment descriptor：包括 LDT descriptor 和 TSS descriptor。
  - Gate descriptor：包括 Call-gate, Interrupt-gate, Trap-gate 和 Task-gate descriptor。
- Non-system segment descriptor
  - Code segment descriptor
  - Data segment descriptor

64 位 CPU

在 64 位模式下：处理器把 CS/DS/ES/SS 的段基都当作 0，忽略与之关联的段描述符中的段基地址。因为在 64 位模式中，CPU 可以访问所有可寻址的内存空间。**今天大多数的 64 位 CPU 只需要访问 40 位到 48 位的物理内存，因此不再需要段寄存器去扩展。**所以这里分析段式管理只是因为之前在很多地方见到段选择子这个东西，但是不理解，故花点时间总结一下。

Reference

[1] https://zhuanlan.zhihu.com/p/324210723

[2] x86/64 体系探索及编程 邓志 电子工业出版社
