5.进程切换

在x86架构中，**TSS（Task State Segment）** 是一个特定的内存段，用于保存和恢复的上下文信息。
x86架构提供了硬件支持的任务切换机制。在任务切换过程中，CPU会自动保存和加载TSS中的信息，从而实现从一个任务到另一个任务的切换。

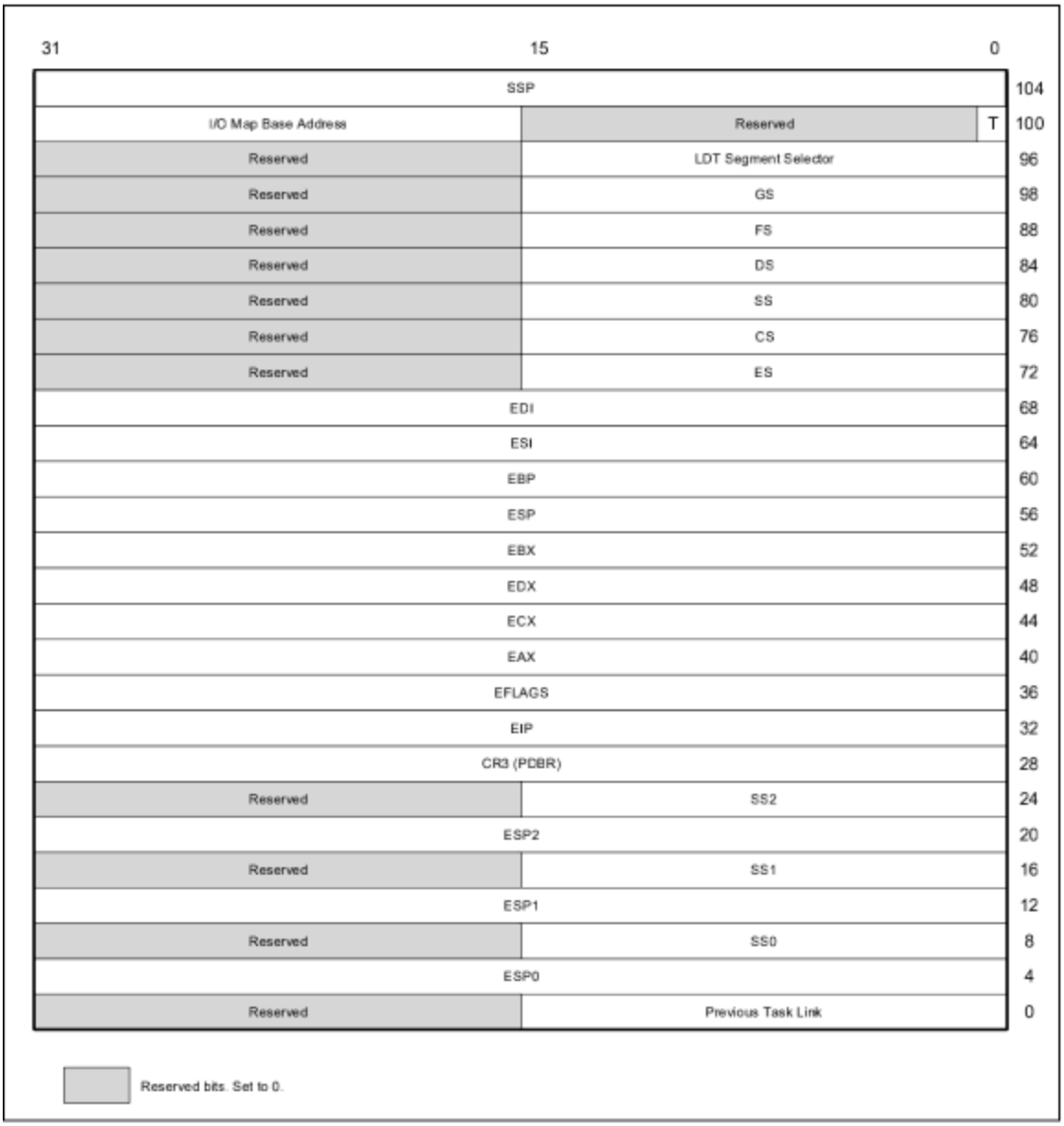


Figure 7-2. 32-Bit Task-State Segment (TSS)

软硬件任务切换的区别

- **硬件任务切换（硬切换）：** 在x86处理器中，当发生硬件任务切换时，CPU会自动加载TSS信息，快速恢复任务的执行状态，这种切换通常不需要操作系统的干预。

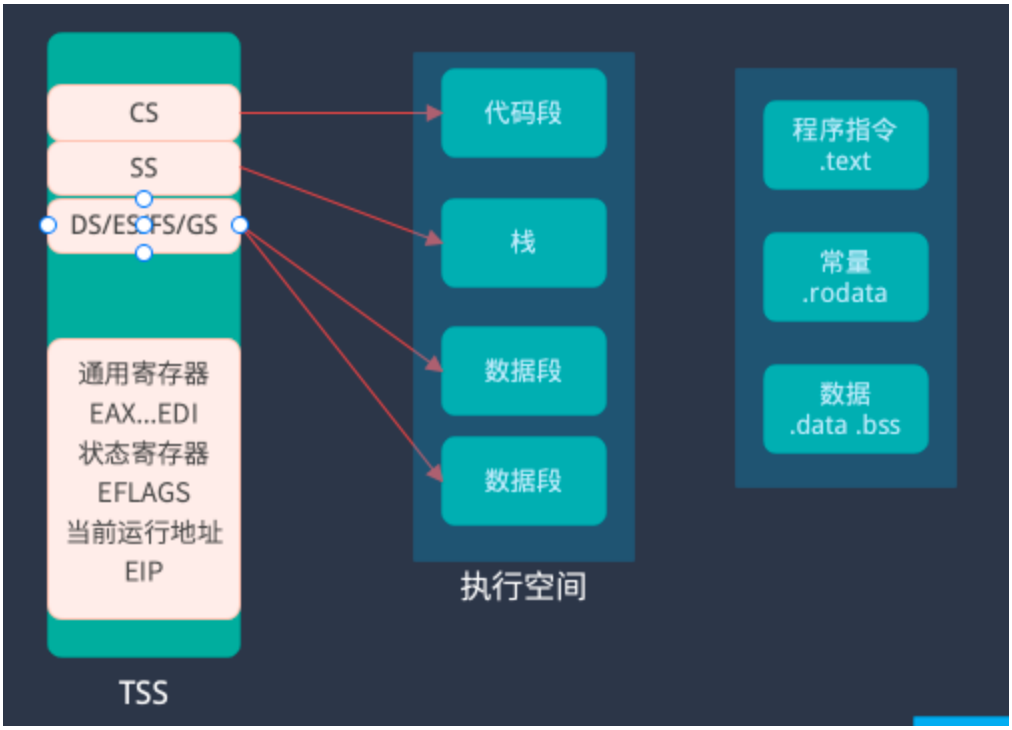
- **软件任务切换（软切换）：** 在现代操作系统中，许多任务切换是由操作系统内核通过软件控制的，操作系统会手动保存和恢复任务状态，不一定依赖硬件自动完成任务切换。

在现代操作系统（如Linux或Windows）中，TSS的作用有所减少，因为大多数操作系统采用**上下文切换**（Context Switch）和**虚拟内存**技术来进行任务调度，而不是依赖硬件任务切换。

初始化任务

通过TSS结构，我们就能够知道一个程序当前的运行状态。**在进行不同程序运行的切换时，需要将前一个程序完整的运行状态全部保存到TSS，这样当后续该程序需要再次运行时，再将该状态进行从TSS中恢复，从而就像能完整地继续从上次切换的位置继续往下运行，好像什么都没发生**

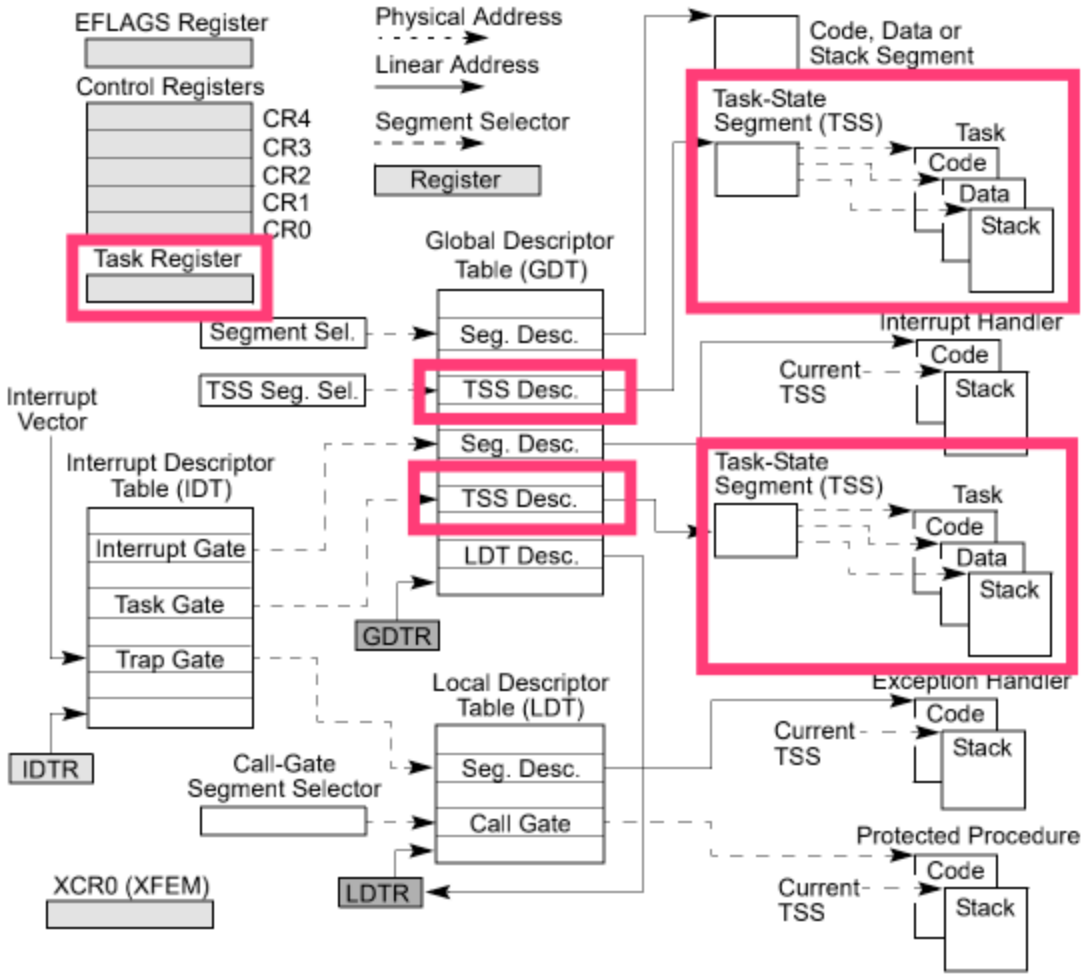
一样。



目前只需要在切换时保存好当前的段寄存器、通用寄存器、EIP、EFLAGS中的内容到当前程序的TSS中，然后再加载下一个将要运行程序的TSS中段寄存器、通用寄存器、EIP、EFLAGS值进行恢复。

简单双任务相互切换

为任务配置好TSS之后，还需要将其加入到CPU的硬件数据配置中，添加方法是在GDT表中添加一个专门的TSS描述符指向该结构，如下图所示。
当前执行哪个任务，则由Task Register指向相应的描述符。
在系统初始化时，必须向Task Register写入一个有效的TSS描述符对应的选择子。



intel编程文档卷3 第62页

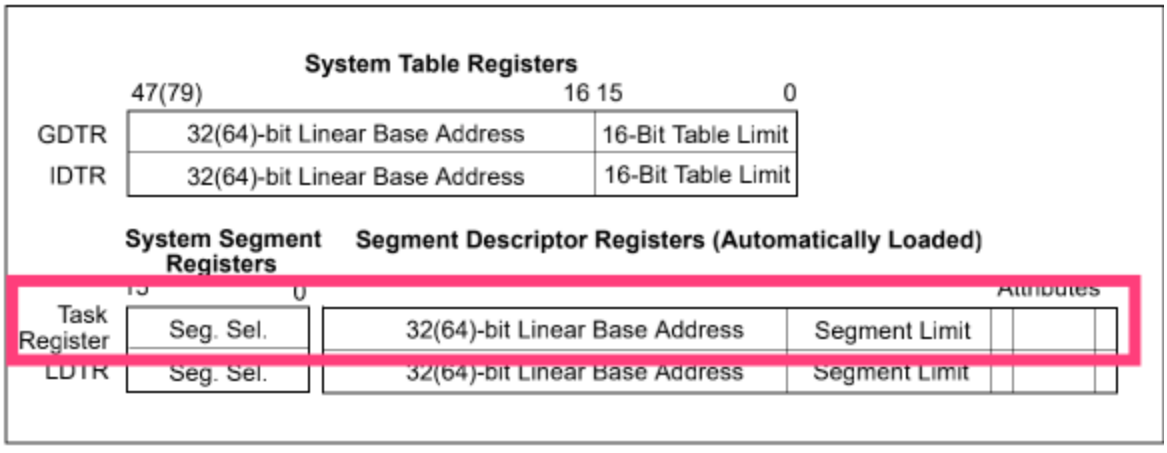


Figure 2-6. Memory Management Registers

intel编程文档卷3 第72页

TSS描述符的内容与代码段、数据段描述符等结构类似，主要区别在于Type字段。在任务初始化时，从GDT表中分配了相应的空闲描述符，然后按TSS描述符格式进行初始化。

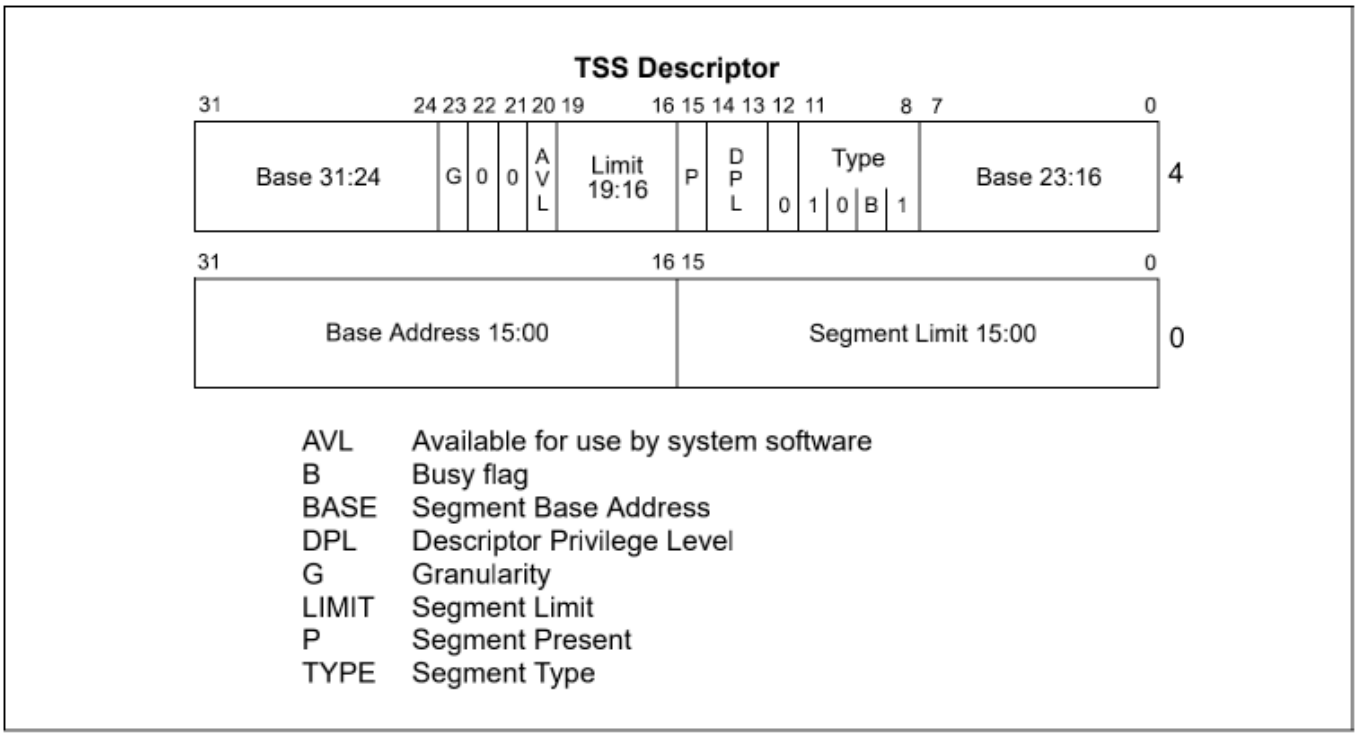


Figure 7-3. TSS Descriptor

intel编程文档卷3 第256页

任务切换具体流程

配置完所有的描述符之后，就可以通过远跳转far_jump()跳转到另一个任务运行。

```
static inline void far_jump(uint32_t selector, uint32_t offset) {
    uint32_t addr[] = {offset, selector };
    __asm__ __volatile__("ljmpl *(%[a])"::[a]"r"(addr));
}
```

偏移量为0为什么?
CPU 会使用 `ljmpl` 指令（或类似的指令）跳转到新的任务。此时，偏移量为 `0` 是为了确保 CPU 跳转到目标 TSS 段的起始位置。
选择子通过 GDT 查找目标 TSS 段的描述符，TSS 段的描述符基址被设置为：TSS 段的起始位置，TSS 段的起始位置+offset(0)->定位到TSS 段的起始位置。

Task Register (TR) 的使用:
TSS 段通过 **Task Register (TR)** 来指定，TR 存储的是当前任务的 TSS 选择符。
CPU 会将当前任务的 TSS 保存在 TSS 段（通过TR获得）中，并加载新任务的 TSS，更新 **TR** 寄存器，使其指向新的任务的 TSS。

手动任务切换方法

根据函数的调用原理分析出其中一些寄存器不需要保存。

存在当前任务的栈中。选用任务栈进行保存，我们就可以简单的使用一些PUSH机令，就可以完成这些计算器的保存。

任务切换这个过程就变成了将一些寄存器保存在当前任务的栈中，并且从下一个任务的栈中提取出相应寄存器值再恢复到 CPU的内核寄存器中。