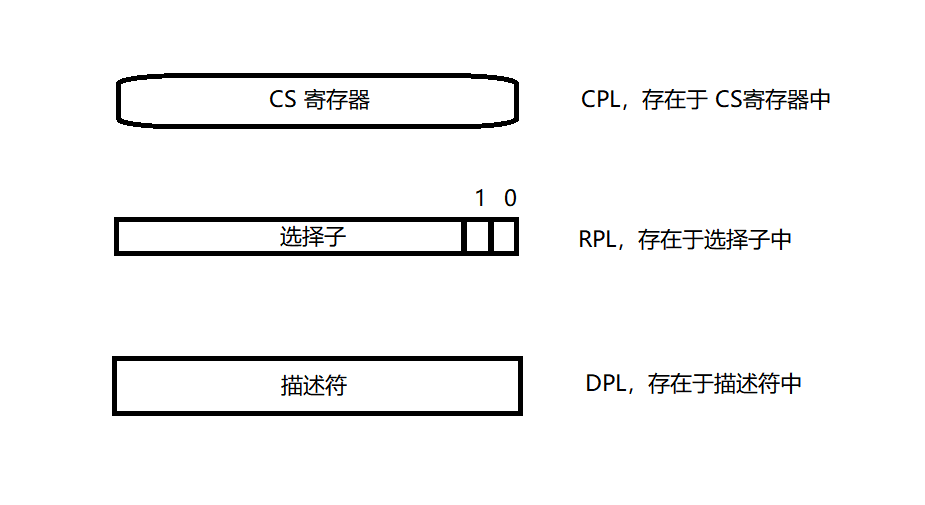
CPL是当前进程的权限级别(Current Privilege Level)，是当前正在执行的代码所在的段的特权级，存在于cs寄存器的低两位。  
  RPL说明的是进程对段访问的请求权限(Request Privilege Level)，是对于段选择子而言的，每个段选择子有自己的RPL，它说明的是进程对段访问的请求权限，有点像函数参数。而且RPL对每个段来说不是固定的，两次访问同一段时的RPL可以不同。RPL可能会削弱CPL的作用，例如当前CPL=0的进程要访问一个数据段，它把段选择符中的RPL设为3，这样虽然它对该段仍然只有特权为3的访问权限。  
 DPL存储在段描述符中，规定访问该段的权限级别(Descriptor Privilege Level)，每个段的DPL固定。  
当进程访问一个段时，需要进程特权级检查，一般要求DPL >= max {CPL, RPL}  
 下面打一个比方，中国官员分为6级国家主席1、总理2、省长3、市长4、县长5、乡长6，假设我是当前进程，级别总理（CPL=2）,我去聊城市(DPL=4)考察(呵呵),我用省长的级别(RPL=3 这样也能吓死他们:-))去访问,可以吧，如果我用县长的级别，人家就不理咱了(你看看电视上的微服私访，呵呵)，明白了吧！为什么采用RPL，是考虑到安全的问题，就好像你明明对一个文件用有写权限，为什么用只读打开它呢，还不是为了安全！



1. 对数据段和堆栈段访问时的特权级控制：  
    要求访问数据段或堆栈段的程序的CPL≤待访问的数据段或堆栈段的DPL，同时选择子的RPL≤待访问的数据段或堆栈段的DPL，即程序访问数据段或堆栈段要遵循一个准则：只有相同或更高特权级的代码才能访问相应的数据段。这里，RPL可能会削弱CPL的作用，访问数据段或堆栈段时，默认用CPU和RPL中的最小特权级去访问数据段，所以max {CPL, RPL} ≤ DPL，否则访问失败。
2. 对代码段访问的特权级控制（代码执行权的特权转移）：  
    让我们先来记一些“定律”：  
   1，所有的程序转跳，CPU都不会把段选择子的RPL赋给转跳后程序的CS.RPL. .  
   2，转跳后程序的CPL(CS.RPL)只会有下面的俩种可能  
   3，转跳后程序的CPL(CS.RPL) = 转跳前程序的CPL(CS.RPL)   
   　 或  
   　 转跳后程序的CPL(CS.RPL) =　转跳后程序的CodeDescriptor.DPL  
   以 Call 为例(只能跳到等于当前特权级或比当前特权级更高的段)：  
    怎样决定这两种选择，这就要首先知道转跳后程序的段是一致代码段还是非一致代码段.其实也很简单，规则如下：  
    1，如果能成功转跳到一致代码段, **转跳后程序的CPL(CS.RPL) = 转跳前程序的CPL(CS.RPL)**，(转跳后程序的CPL继承了转跳前程序的CPL)  
    2，如果能成功转跳到非一致代码段, **转跳后程序的CPL(CS．RPL)　＝转跳后程序的Descriptor.DPL**。(转跳后程序的CPL变成了该代码段的特权级.我在前面提到DPL是段描述符中的特权级, 它的本意是用来代表它所描述的段的特权级)怎样才能成功转跳啦?  
    这里有四个重要的概念：

==========

1)．**段的保护观念是高特权级不找低特权级办事，低特权级找高特权级帮忙，相同的一定没问题.**(这样想逻辑是没错，事实对不对就不知道.)也就是县长不找乡长，乡长不求农民，反过来农民求乡长，乡长找县长.这个概念是最重要的。

2) 一致代码段的意义: 让客人很方便的利用主人(一致代码段)的东西为自己办事.但客人这身份没有改变NewCS.RPL=OldCS.RPL所以只能帮自己办事。比方说乡长有一头牛，农民可以借来帮自己种田，但不能种别人的田.但是如果你是乡长当然可以种乡里所有的田。

3) 非一致代码段的意义：主人(非一致代码段)可以帮客人但一定是用自己的身份NewCS.RPL= DestinationDescriptorCode.DPL这里可能有安全的问题, 搞不好很容易农民变县长。主人太顽固了一定要坚持自己的身份，有什么方法变通一下，来个妥协好不好。好的，它就是RPL的用处。

4) RPL: 它让程序有需要的时候可以表示一个特权级更低的身份Max(RPL,CPL)而不会失去本身的特权级CPL(CS.RPL)，有需要的时候是指要检查身份的时候。事实上RPL跟段本身的特权级DPL和当前特权级CPL没有什么关系,因为RPL的值在成功转跳后并不赋给转跳后的CS.RPL。

==========

还是要问怎样才能成功转跳啦?这里分两种情况：

 普通转跳（没有经过Gate 这东西）：即JMP或Call后跟着48位全指针（16位段选择子+32位地址偏移），且其中的段选择子指向代码段描述符，这样的跳转称为直接（普通）跳转。**普通跳转不能使特权级发生跃迁，即不会引起CPL的变化**，看下面的详细描述：  
 《目标是一致代码段：》  
要求：**CPL(CS.RPL)>=DestinationDescriptorCode.DPL** ，其他RPL是不检查的。  
**转跳后程序的CPL(NewCS.RPL)  =  转跳前程序的CPL( OldCS.RPL)**上面的安排就是概念１，２的意思,此时，**CPL没有发生变化，纵使它执行了特权级（DPL）较高的代码**。若访问时不满足要求，则发生异常。  
 《目标是非一致代码段：》  
要求：**CPL(CS.RPL)＝DestinationDescriptorCode.DPL　AND　RPL≤CPL(CS.RPL)**转跳后程序的**CPL(NewCS.RPL) = DestinationDescriptorCode.DPL**  
上面的安排就是概念３的意思和部分１的意思----主人(一致代码段)只帮相同特权级的帮客人做事。因为前提是CPL=DPL，所以转跳后程序的CPL(NewCS.RPL) = DestinationDescriptorCode.DPL不会改变CPL的值，特权级(CPL)也没有发生变化。如果访问时不满足前提CPL=DPL，则引发异常。

 通过调用门的跳转：当段间转移指令JMP和段间转移指令CALL后跟着的目标段选择子指向一个调用门描述符时，该跳转就是利用调用门的跳转。**这时如果选择子后跟着32位的地址偏移，也不会被cpu使用，因为调用门描述符已经记录了目标代码的偏移**。使用调门进行的跳转比普通跳转多一个步骤，**即在访问调用门描述符时要将描述符当作一个数据段来检查访问权限，要求指示调用门的选择子的RPL≤门描述符DPL，同时当前代码段CPL≤门描述符DPL，就如同访问数据段一样，要求访问数据段的程序的CPL≤待访问的数据段的DPL，同时选择子的RPL≤待访问的数据段或堆栈段的DPL**。只有满足了以上条件，CPU才会进一步从调用门描述符中读取目标代码段的选择子和地址偏移，进行下一步的操作。  
 **从调用门中读取到目标代码的段选择子和地址偏移**后，我们当前掌握的信息又回到了先前，和普通跳转站在了同一条起跑线上（普通跳转一开始就得到了目标代码的段选择子和地址偏移），有所不同的是，**此时，CPU会将读到的目标代码段选择子中的RPL清0，即忽略了调用门中代码段选择子的RPL的作用**。完成这一步后，CPU开始对**当前程序的CPL**，**目标代码段选择子的RPL**（事实上它被清0后总能满足要求）以及由目标代码选择子指示的目标代码段**描述符中的DPL**进行特权级检查，并根据情况进行跳转，具体情况如下：

==========

目标是一致代码段：  
 要求：CPL(CS.RPL)≥DestinationDescriptorCode.DPL ，RPL不检查，因为RPL被清0，所以事实上永远满足RPL≤DPL，这一点与普通跳转一致，适用于JMP和CALL。  
 转跳后程序的CPL(NewCS.RPL) = 转跳前程序的CPL( OldCS.RPL)，因此特权级没有发生跃迁。  
  
目标是非一致代码段：  
 1，当用JMP指令跳转时：  
 要求：CPL(CS.RPL)＝DestinationDescriptorCode.DPL　AND　RPL<= CPL(CS.RPL)（事实上因为RPL被清0，所以RPL≤CPL总能满足，因此RPL与CPL的关系在此不检查）。若不满足要求则程序引起异常。  
 转跳后程序的CPL(NewCS.RPL) = DestinationDescriptorCode.DPL  
 因为前提是CPL=DPL，所以转跳后程序的CPL(NewCS.RPL) = DestinationDescriptorCode.DPL不会改变CPL的值，特权级也没有发生变化。如果访问时不满足前提CPL=DPL，则引发异常。  
 2，当用CALL指令跳转时：  
 要求：CPL(CS.RPL)≥DestinationDescriptorCode.DPL（RPL被清0，不检查），若不满足要求则程序引起异常。  
 转跳后程序的CPL(NewCS.RPL) = DestinationDescriptorCode.DPL  
　　当条件CPL=DPL时，程序跳转后CPL=DPL,特权级不发生跃迁；当CPL＞DPL时，程序跳转后CPL=DPL,特权级发生跃迁，这是我们当目前位置唯一见到的使程序当前执行忧先级(CPL)发生变化的跳转方法，即用CALL指令+调用门方式跳转，且目标代码段是非一致代码段。

==========

总结：以上介绍了两种情况的跳转，分别是普通跳转和使用调用门的跳转，其中又可细分为JMP跳转和CALL跳转，跳转成功已否是由CPL，RPL和DPL综合决定的。所有跳转都是从低特权级代码向同级或更高特权级（DPL）跳转，但保持当前执行特权级(CPL)不变，这里有点难于区别为什么说向高特权级跳转，又说特权级没变，这里“高特权级”是指目标代码段描述符的DPL，它规定了可以跳转到该段代码的最高特权级；而后面的CPL不变才真正说明了特权级未发生跃迁。我们可以看到，只有用CALL指令+调用门方式跳转，且目标代码段是非一致代码段时，才会引起CPL的变化，即引起代码执行特权级的跃迁！