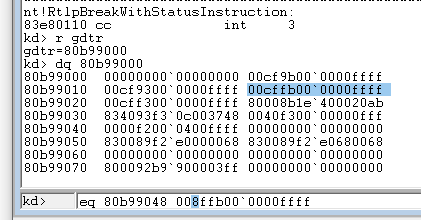
今天讲了D/B位和TYPE位。

# D/B位

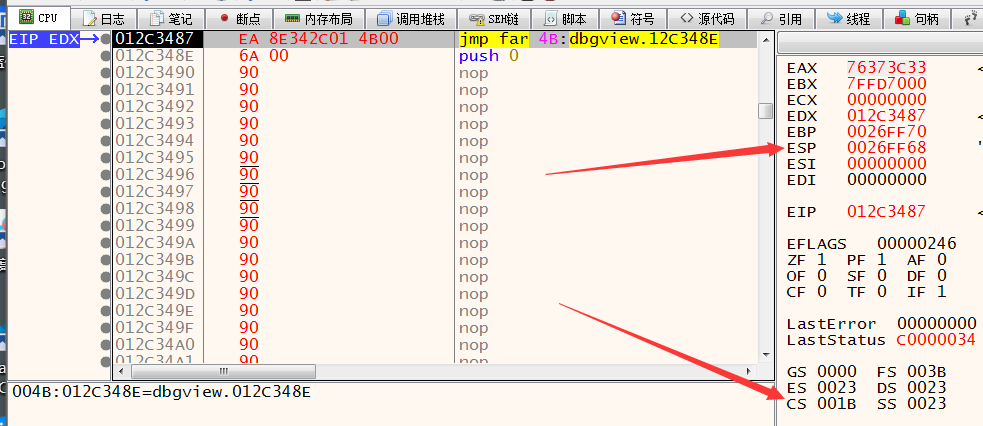
首先在做D/B位的实验。D/B位会影响栈操作和代码操作时，默认寄存器的位数。D/B=1，默认操作位数是32位；D/B=0，默认操作位数是16位。

第一个实验，把CS的段描述符复制到GDT表偏移48的位置，然后把CS改成4B，然后观察现象。（为什么是4B而不是48，是因为RPL要等于3）

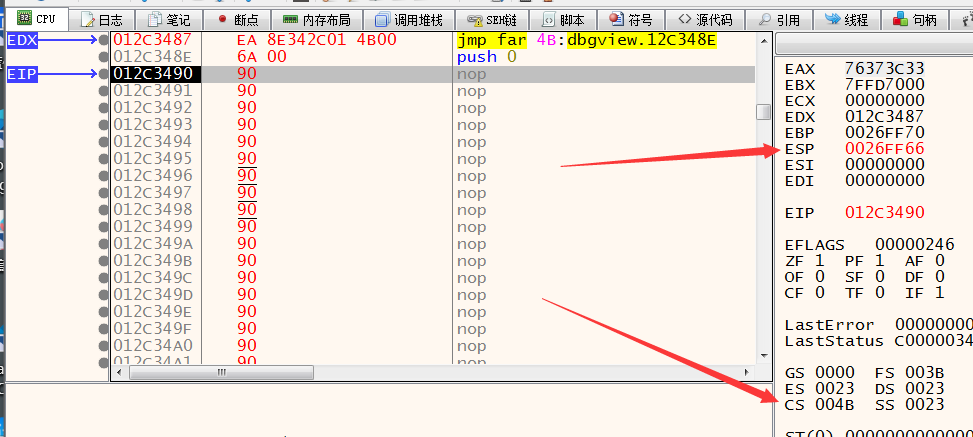


拷贝CS到48处，改D/B为0，此时默认操作数大小为16位。

由于CS不能MOV修改是，我们用跨段跳转不提权的方式将CS改为4B，然后观察push指令对esp的影响：



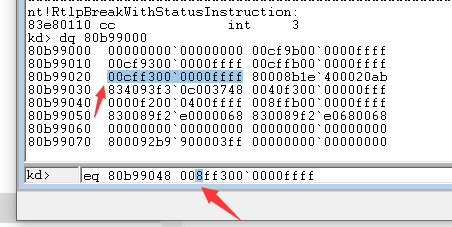
执行指令前



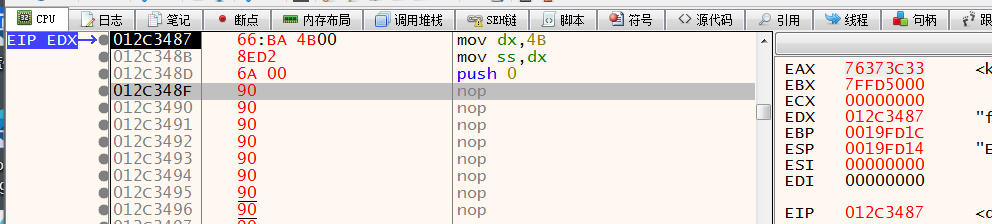
执行指令后

可以看到，CS改为4B后，push 0 把esp的值减少了2。

下一个实验，看看D/B对栈的影响，我们把SS(0x23)拷贝到0x48偏移的位置，同时把D/B改成0：



看看现在操作栈会是什么结果



结果是push直接报异常



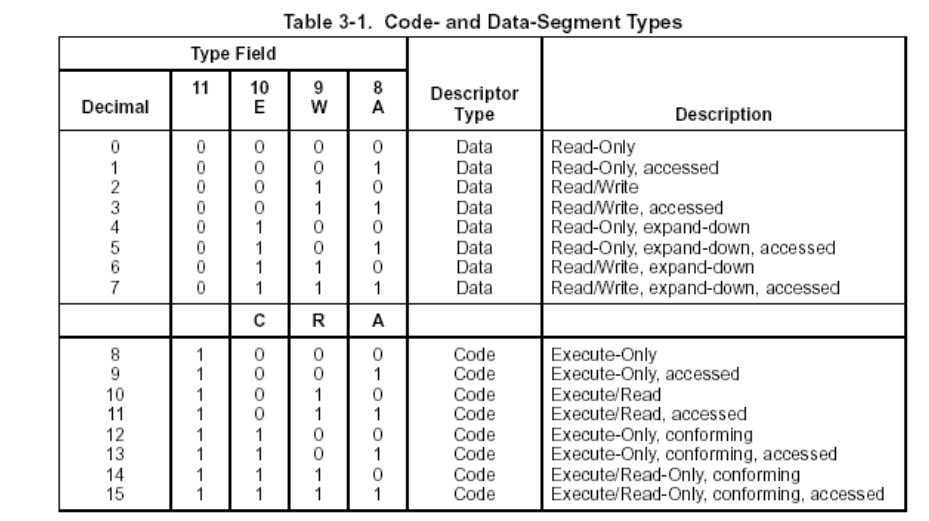


原因也很好解释，ESP的地址（19FD14）已经超过16位SP可以表示的范围（FFFF），所以报错。

上面两个实验介绍了D/B位对SS,CS的影响，至于DS，则是没影响的。这个我的理解是，形如 mov dword ptr ds:[0x12345678], eax这种指令，指令本身已经说明目标操作数是dword大小了，所以不受影响。

# TYPE位

接下来是TYPE位，TYPE实际上是4个位组成的，当S=1时，TYPE表示数据段代码段的type。注意，当S=0时，就不查这个表了。（现在还没讲到，是那一堆门相关的知识）

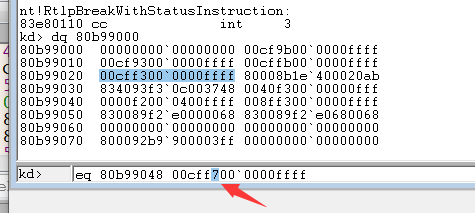


这张表有两个点需要注意，一个是内存的生长方向，一个是一致代码段和非一致代码段。这个一致非一致没什么卵用，是纯段模式下的东西，没法做试验。着重关注生长方向。

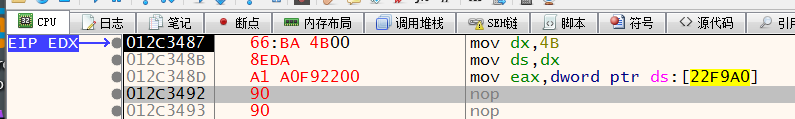
这个内存生长方向比较搞笑，type=[0,1,2,3]时，地址范围是 [base, base+limit]。举个例子，base=0,limit=fffff,G=1,type=3，表示的是数据段，范围是0-FFFFFFFF。这种情况很常见。

在这种情况下，如果将type改成7，那就是可读写向下生长，那么地址有效范围就是除了0-FFFFFFFF以外的区间，也就是所有地址都无效了。此时只要进行寻址，就会报异常。

那么在此基础上，把limit改成0，即有效地址范围是 0-0 以外的空间，那么整个4GB就又有效了，此时寻址就不会报异常了。



复制DS，改type为7，向下生长。Base=0，limit=FFFFF，G=1





执行这段代码，改DS后，有效地址范围是0-0，只要进行ds寻址，就会报错。

在刚才的基础上，如果将limit改为0，就不会报错了：



