# 一、笔记

### 1.概述

除10-10-12外，还有2-9-9-12，9-9-9-9-12等分页机制，要弄清这些机制，最好的方法不是记公式，而是逆向分析操作系统是如何解析线性地址的。

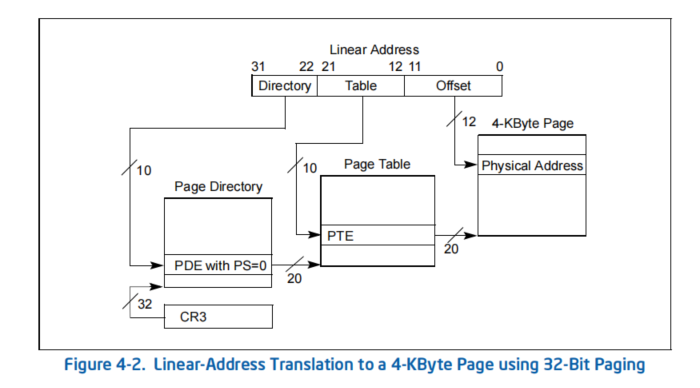
这节课火哥带着我们逆向了win7的101012分页内核的 \_MmIsAddressValid 函数。该函数检测一个线性地址是否有效，它做的事情跟我们前几节课拆线性地址原理上是一样的，也是获取PDE,PTE并检查属性。

作业是自己逆一遍 MmIsAddressValid ，不过在此之前，还是先根据白皮书上的资料回顾一下前两次课学习的内容。

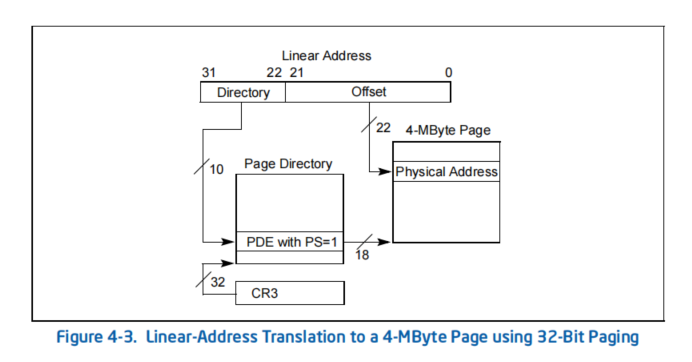
### 2.101012分页机制梳理

101012分页机制下，4KB小页线性地址按照10位-10位-12位的方式分成三部分：Directory, Table, Offset。如图4-2所示。高10位 Directory是页目录Page Directory的下标，每项是4字节，所以寻址的时候 Directory要乘以4。页目录的基址，也就是CR3，加上4\*Directory就找到了页目录项PDE。

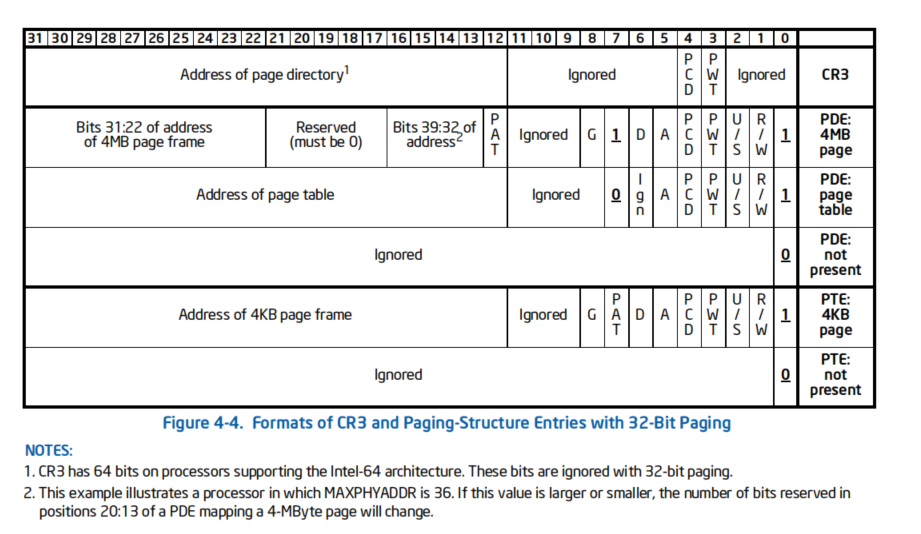
前面讲过，PDE的低12位是属性，低12位清零后就是页表的基址。线性地址的Table是页表下标，同样要乘以4。 页表基址+4\*Table 就找到了页表项PTE。PTE低12位同样是属性，清零后就是线性地址对应的4KB物理页基址，最后加上页内偏移Offset就找到了线性地址对应的物理地址。



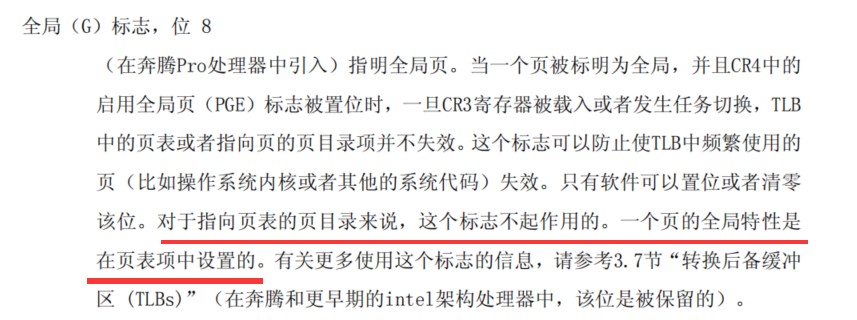
除了4KB小页之外，还有4MB大页。如果PDE.PS=1，那这个线性地址就是大页，它就没有PTE。高10位仍然是页目录下标，低22位全部是页内偏移。



我发现白皮书中译版和英文版还是有点区别，可能英文版比较新，英文版这里给了一张图我觉得挺好的，给贴出来：

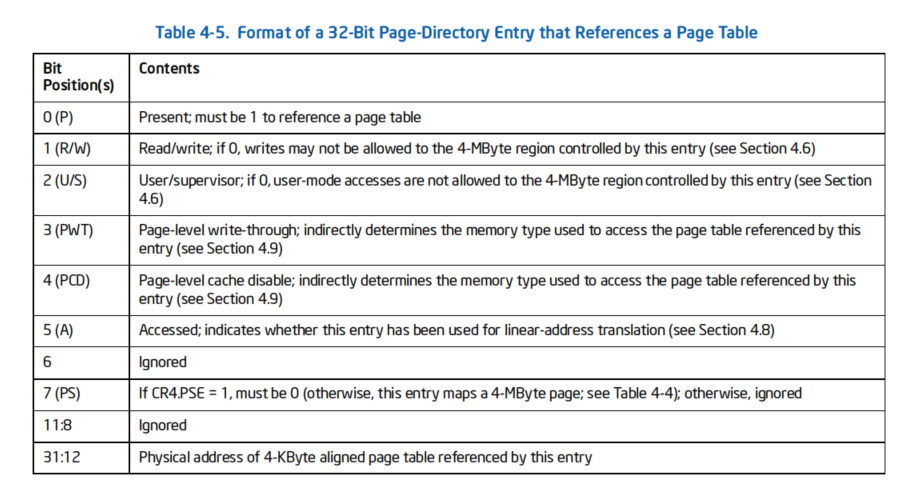


可以看到页目录基址CR3的低12位也是属性，寻址的时候也要清零的。P=0就是没挂页，PDE.PS在第7位，PDE.PS=1就是4MB大页，此时G位是有用的；PDE.PS=0就是小页，此时G位被忽略了。这里可以从白皮书中找到解释：

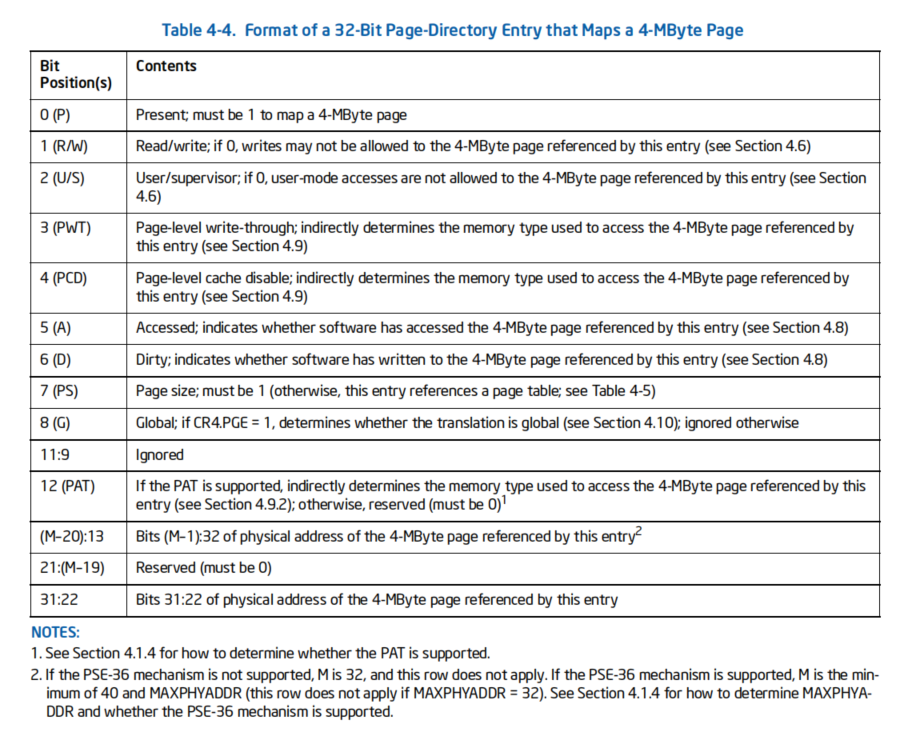


这就是说啊，指向页表的PDE，也就是小页，它的全局属性是由PTE设置的。只有大页才可以设置全局属性。

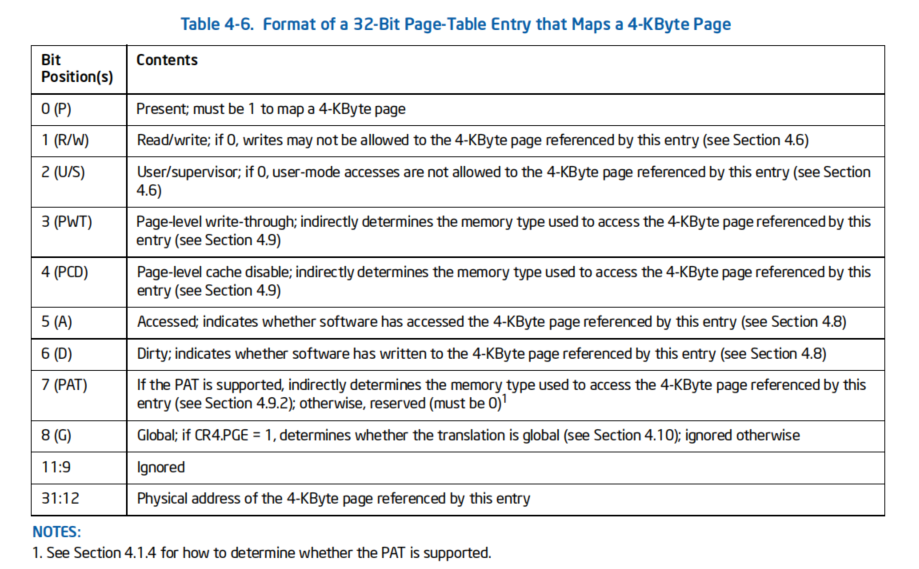
### 3.附：PDE, PTE属性位说明



指向页表的PDE（小页）



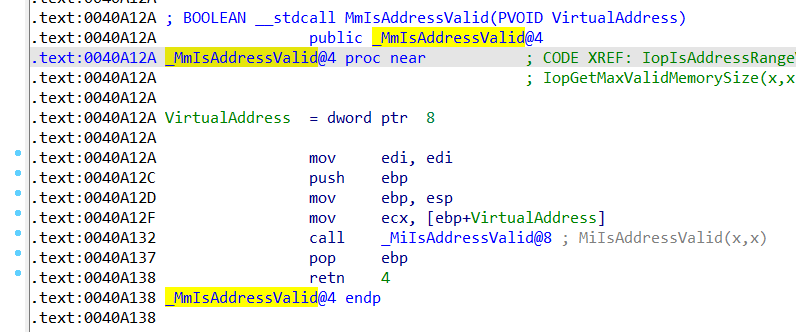
指向大页的PDE



小页PTE

# 二、作业

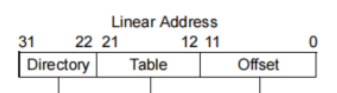
### 1.MmIsAddressValid 分析此函数



\_MmIsAddressValid 本身啥也不干，就调用一下 \_MiIsAddressValid



MiIsAddressValid 函数的逆向结果如上图，计算PDE, PTE时的移位和逻辑与运算是编译器优化，和我们手动拆结果是一样的。



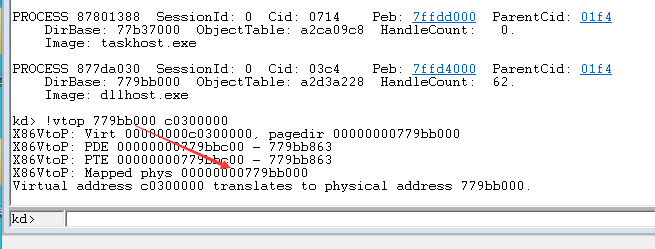
我们手动拆的方法是基于Intel给的定义：

&PDE = CR3.Address + Directory \* 4

&PTE = PDE.Address + Table \* 4

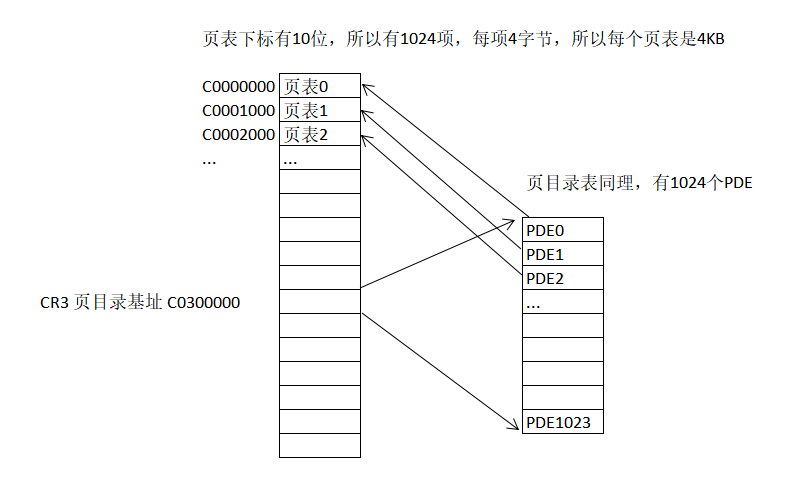
物理地址 = PTE.Address + Offset

而操作系统是无法访问物理地址的，操作系统和这些页目录表，页表打交道用的是线性地址。页目录表基址是0xC0300000，这个其实就是CR3的线性地址。



简单解释一下MiIsAddressValid的算法，比如计算PDE，我们知道右移22位是啥意思，就是取线性地址的Directory部分，而它这里是右移20位，右边低地址还多出两位，然后与上0xFFC就是把低2位清零。所以现在的结构就是 Directory(10位) + 00(2位)，我们知道乘以4就是左移2位，所以这里就明白了，原来是 Directory \* 4的意思，最后加上页目录表基址的线性地址 0xC0300000，就找到了PDE。

计算PTE就比较抽象了，它通过一个右移和一个与运算直接从线性地址计算得到了PTE。这里可以画一个图来帮助理解：



&PDE = 0xC0300000 + Directory \* 4

&PTE = 0xC0000000 + Directory \* 4KB + Table \* 4

Directory表示第几个PDE，也就是第几个页表，所以这里用0xC0000000首地址加上Directory \* 4kb就能找到页表基址，再加上 Table \* 4 就能找到PTE了。

所以我们就可以理解MiIsAddressValid为啥要这么算了

((VirtualAddress >> 10) & 0x3FFFFC) + 0xC0000000

Directory | Table | Offset

10 | 10 | 12

右移10位，就是把Offset扔掉，再左移2位

Directory | Table | Offset高2位

10 | 10 | 2

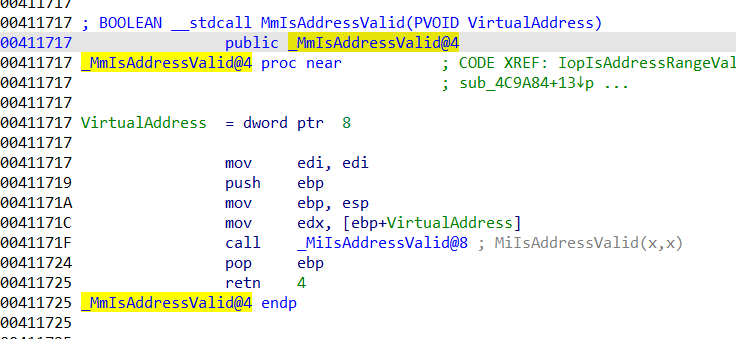
& 0011 1111 1111 1111 1111 1100

Directory \* 4KB | Table \* 4

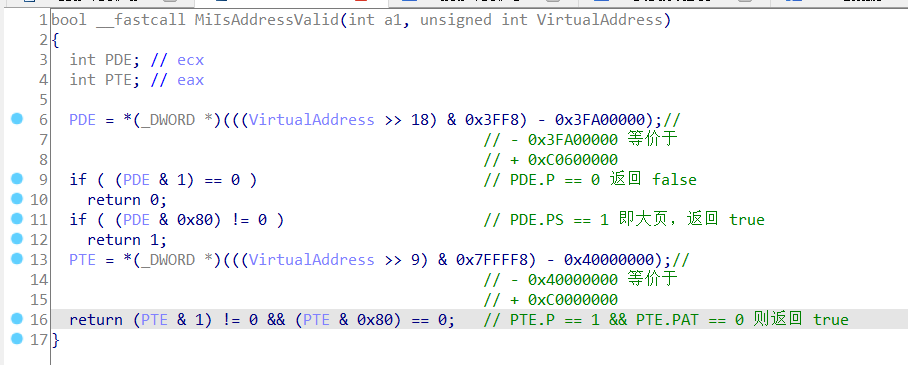
Directory右移了12位，就是乘以4KB，Table右移了2位，就是乘以4.

最后加上页表基址0xC0000000，就得到了PTE的地址。

### 分析一下 29912 内核下的这个函数



同样是调用\_MiIsAddressValid



忽略计算PDE,PTE的过程，其实和101012几乎没有区别，除了页目录基址变成了 0xC0600000

### MmIsAddressValid缺陷

Win7这个函数的逻辑有漏洞，可以被利用。下次课细说。

### 周作业预告：所有PDE PTE US位改成1 实现R3直接访问R0