将虚拟地址0送到MMU。MMU看到虚拟地址落在页面0（0~4095），根据其映射结果，这一页面对应的是页框2（8192~12 287），因此MMU把地址变换为8192，并把地址8192送到总线上。内存对MMU一无所知，它只看到一个读或写地址8192的请求并执行它。MMU从而有效地把所有从0~4095的虚拟地址映射到了8192~12 287的物理地址。

同样地，指令

```
MOV REG, 8192
```

被有效地转换为：

```
MOV REG, 24576
```

因为虚拟地址8192（在虚拟页面2中）被映射到物理地址24 567（在物理页框6中）上。第三个例子，虚拟地址20 500在距虚拟页面5（虚拟地址20 480~24 575）起始地址20字节处，并且被映射到物理地址12 288+20=12 308。

通过恰当地设置MMU，可以把16个虚拟页面映射到8个页框中的任何一个。但是这并没有解决虚拟地址空间比物理内存大的问题。在图3-9中只有8个物理页框，于是只有8个虚拟页面被映射到了物理内存中，在图3-9中用叉号表示的其他页并没有被映射。在实际的硬件中，用一个“在/不在”位（present/absent bit）记录页面在内存中的实际存在情况。

当程序访问了一个未映射的页面，例如执行指令

```
MOV REG, 32780
```

将会发生什么情况呢？虚拟页面8（从32 768开始）的第12个字节所对应的物理地址是什么呢？MMU注意到该页面没有被映射（在图中用叉号表示），于是使CPU陷入到操作系统，这个陷阱称为缺页中断（page fault）。操作系统找到一个很少使用的页框且把它的内容写入磁盘（如果它不在磁盘上）。随后把需要访问的页面读到刚才回收的页框中，修改映射关系，然后重新启动引起陷阱的指令。

例如，如果操作系统决定放弃页框1，那么它将把虚拟页面8装入物理地址8192，并对MMU映射做两处修改。首先，它要标记虚拟页面1表项为未映射，使以后任何对虚拟地址4096~8191的访问都导致陷阱。随后把虚拟页面8的表项的叉号改为1，因此在引起陷阱的指令重新启动时，它将把虚拟地址32780映射为物理地址4108（4096+12）。

下面查看一下MMU的内部结构以便了解它是怎么工作的，以及了解为什么我们选用的页面大小都是2的整数次幂。在图3-10中可以看到一个虚拟地址的例子，虚拟地址8196（二进制是0010000000000100）用图3-9所示的MMU映射机制进行映射，输入的16位虚拟地址被分为4位的页号和12位的偏移量。4位的页号可以表示16个页面，12位的偏移可以为一页内的全部4096个字节编址。

可用页号作为页表（page table）的索引，以得出对应于该虚拟页面的页框号。如果“在/不在”位是0，则将引起一个操作系统陷阱。如果该位是1，则将在页表中查到的页框号复制到输出寄存器的高3位中，再加上输入虚拟地址中的低12位偏移量。如此就构成了15位的物理地址。输出寄存器的内容随即被作为物理地址送到内存总线。

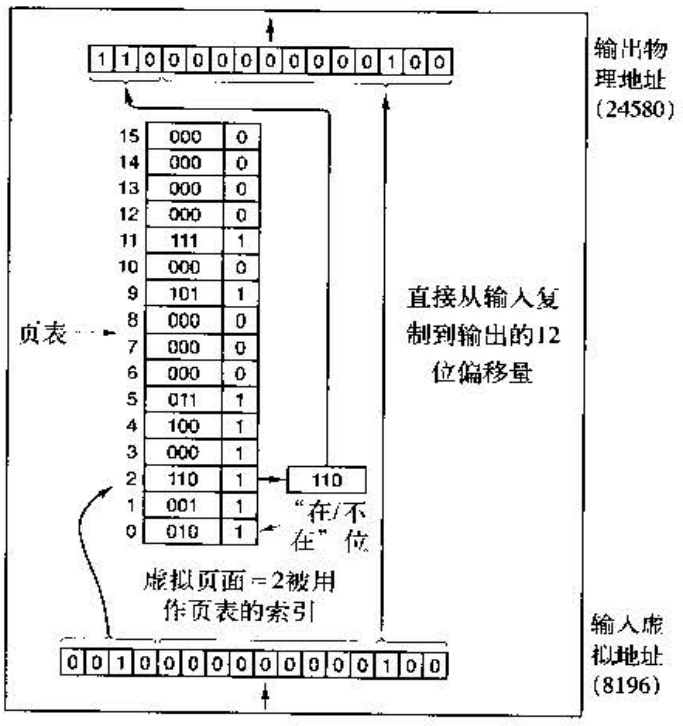

图3-10 在16个4KB页面情况下MMU的内部操作

3.3.2 页表

作为一种最简单的实现，虚拟地址到物理地址的映射可以概括如下：虚拟地址被分成虚拟页号（高