并且页面是按照64字节的边界对齐的（这隐含着页面地址的低6位是000000），所以在描述符中只需要18位来存储页表地址。段描述符中还包含了段大小、保护位以及其他的一些条目。图3-35b一个MULTICS段描述符的示例。段在辅助存储器中的地址不在段描述符中，而是在缺段处理程序使用的另一个表中。

每个段都是一个普通的虚拟地址空间，用与本章前面讨论过的非分段式分页存储相同的方式进行分页。一般的页面大小是1024字节（尽管有一些MULTICS自己使用的段不分页或以64字节为单元进行分页以节省内存）。

MULTICS中一个地址由两部分构成：段和段内地址。段内地址又进一步分为页号和页内的字，如图3-36所示。在进行内存访问时，执行下面的算法。

1) 根据段号找到段描述符。

2) 检查该段的页表是否在内存中。如果在，则找到它的位置，如果不在，则产生一个段错误。如果访问违反了段的保护要求就发出一个越界错误（陷阱）。

3) 检查所请求虚拟页面的页表项，如果该页面不在内存中则产生一个缺页中断，如果在内存就从页表项中取出这个页面在内存中的起始地址。

4) 把偏移量加到页面的起始地址上，得到要访问的字在内存中的地址。

5) 最后进行读或写操作。

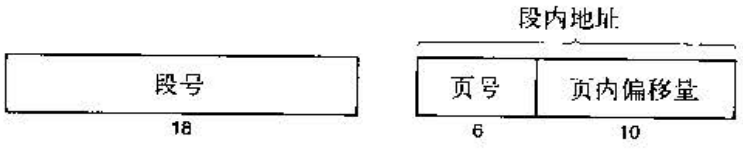

图3-36 一个34位的MULTICS虚拟地址

这个过程如图3-37所示。为了简单起见，我们忽略了描述符段自己也要分页的事实。实际的过程是通过一个寄存器（描述符基址寄存器）找到描述符段的页表，这个页表指向描述符段的页面。一旦找到了所需段的描述符，寻址过程就如图3-37所示。

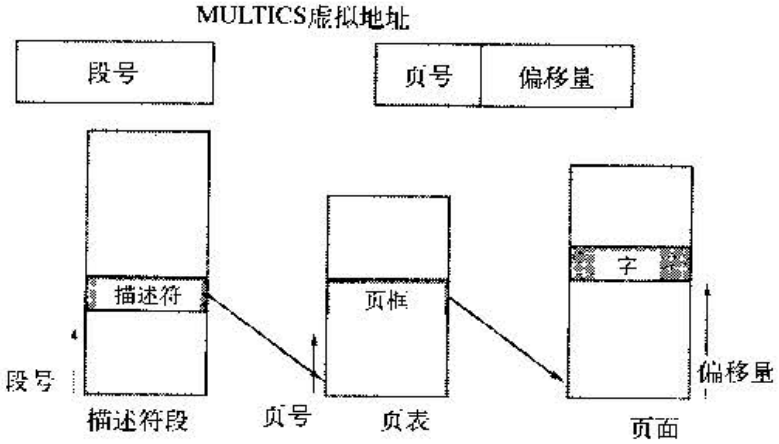

图3-37 两部分组成的MULTICS地址到内存地址的转换

正如读者所想，如果对于每条指令都由操作系统来运行上面所述的算法，那么程序就会运行得很慢。实际上，MULTICS硬件包含了16个字的高速TLB，对给定的关键字它能并行搜索所有的表项，如图3-38所示。当一个地址被送到计算机时，寻址硬件首先检查虚拟地址是不是在TLB中。如果在，就直接从TLB中取得页框号并生成要访问的字的实际地址，而不必到描述符段或页表中去查找。

TLB中保存着16个最近访问的页的地址，工作集小于TLB容量的程序将随着整个工作集的地址被装入TLB中而逐渐达到稳定，开始高效地运行。如果页面不在TLB中，才会访问描述符和页表以找出页框号，并更新TLB使它包含这个页面，最近最少使用的页面被淘汰出TLB。生存时间位跟踪哪个表项是最近最少使用的。之所以使用TLB是为了并行地比较所有表项的段号和页号。

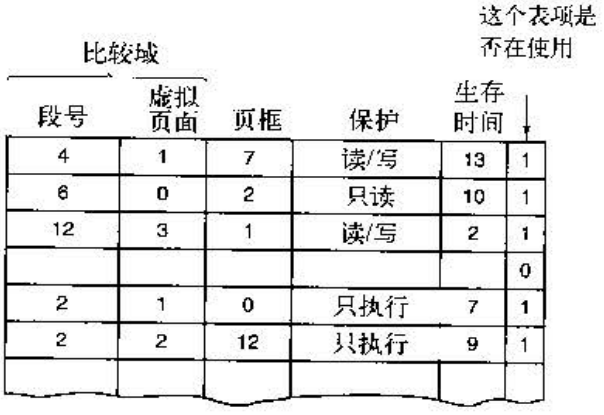

图3-38 一个简化的MULTICS的TLB，两个页面大小的存在使得实际的TLB更复杂

3.7.3 分段和分页结合：Intel Pentium

Pentium处理器的虚拟内存在许多方面都与MULTICS类似，其中包括既有分段机制又有分页机制。