


参数	值
高速缓存块偏移 (CO)	0x_____
高速缓存组索引 (CI)	0x_____
高速缓存标记 (CT)	0x_____
高速缓存命中? (是/否)	
返回的高速缓存字节	0x_____


 **练习题 6.15** 对于内存地址 0x1FE4，再做一遍练习题 6.13。

A. 地址格式(每个小方框一个位):

12	11	10	9	8	7	6	5	4	3	2	1	0
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

B. 内存引用:

参数	值
高速缓存块偏移 (CO)	0x_____
高速缓存组索引 (CI)	0x_____
高速缓存标记 (CT)	0x_____
高速缓存命中? (是/否)	
返回的高速缓存字节	0x_____

 **练习题 6.16** 对于练习题 6.12 中的高速缓存，列出所有的在组 3 中会命中的十六进制内存地址。

6.4.5 有关写的问题

正如我们看到的，高速缓存关于读的操作非常简单。首先，在高速缓存中查找所需字 w 的副本。如果命中，立即返回字 w 给 CPU。如果不命中，从存储器层次结构中较低层中取出包含字 w 的块，将这个块存储到某个高速缓存行中(可能会驱逐一个有效的行)，然后返回字 w 。

写的情况就要复杂一些了。假设我们要写一个已经缓存了的字 w (写命中，write hit)。在高速缓存更新了它的 w 的副本之后，怎么更新 w 在层次结构中紧接着低一层中的副本呢？最简单的方法，称为直写(write-through)，就是立即将 w 的高速缓存块写回到紧接着的低一层中。虽然简单，但是直写的缺点是每次写都会引起总线流量。另一种方法，称为写回(write-back)，尽可能地推迟更新，只有当替换算法要驱逐这个更新过的块时，才把它写到紧接着的低一层中。由于局部性，写回能显著地减少总线流量，但是它的缺点是增加了复杂性。高速缓存必须为每个高速缓存行维护一个额外的修改位(dirty bit)，表明这个高速缓存块是否被修改过。

另一个问题是如何处理写不命中。一种方法，称为写分配(write-allocate)，加载相应的低一层中的块到高速缓存中，然后更新这个高速缓存块。写分配试图利用写的空间局部性，但是缺点是每次不命中都会导致一个块从低一层传送到高速缓存。另一种方法，称为非写分配(not-write-allocate)，避开高速缓存，直接把这个字写到低一层中。直写高速缓存通常是非写分配的。写回高速缓存通常是写分配的。

为写操作优化高速缓存是一个细致而困难的问题，在此我们只略讲皮毛。细节随系统的不同而不同，而且通常是私有的，文档记录不详细。对于试图编写高速缓存比较友好的