


一个高速缓存组。这种抖动导致速度下降 2 或 3 倍并不稀奇。另外，还要注意虽然我们的示例极其简单，但是对于更大、更现实的直接映射高速缓存来说，这个问题也是很真实的。

幸运的是，一旦程序员意识到了正在发生什么，就很容易修正抖动问题。一个很简单的方法是在每个数组的结尾放 B 字节的填充。例如，不是将 x 定义为 `float x[8]`，而是定义成 `float x[12]`。假设在内存中 y 紧跟在 x 后面，我们有下面这样的从数组元素到组的映射：

元素	地址	组索引	元素	地址	组索引
$x[0]$	0	0	$y[0]$	48	1
$x[1]$	4	0	$y[1]$	52	1
$x[2]$	8	0	$y[2]$	56	1
$x[3]$	12	0	$y[3]$	60	1
$x[4]$	16	1	$y[4]$	64	0
$x[5]$	20	1	$y[5]$	68	0
$x[6]$	24	1	$y[6]$	72	0
$x[7]$	28	1	$y[7]$	76	0

在 x 结尾加了填充， $x[i]$ 和 $y[i]$ 现在就映射到了不同的组，消除了抖动冲突不命中。

 **练习题 6.10** 在前面 `dotprod` 的例子中，在我们对数组 x 做了填充之后，所有对 x 和 y 的引用的命中率是多少？

旁注 为什么用中间的位来做索引

你也许会奇怪，为什么高速缓存用中间的位来作为组索引，而不是用高位。为什么用中间的位更好，是有很好的原因的。图 6-31 说明了原因。如果高位用做索引，那么一些连续的内存块就会映射到相同的高速缓存块。例如，在图中，头四个块映射到第一个高速缓存组，第二个四个块映射到第二个组，依此类推。如果一个程序有良好的空间局部性，顺序扫描一个数组的元素，那么在任意时刻，高速缓存都只保存着一个块大小

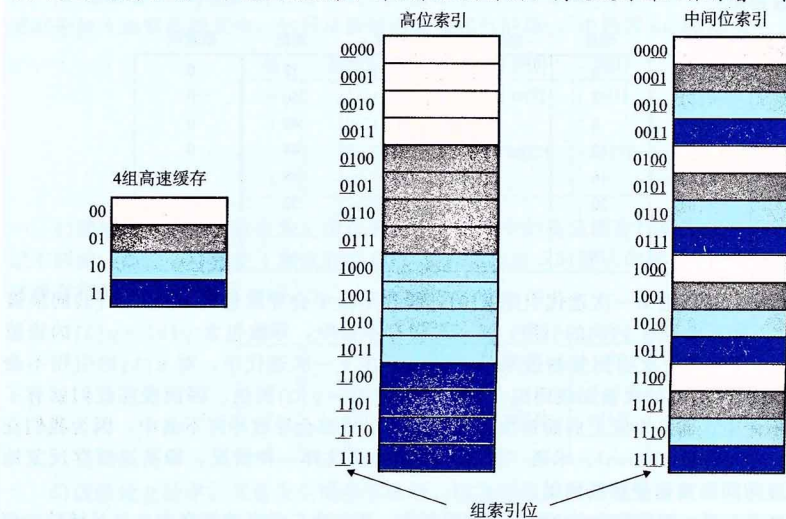


图 6-31 为什么用中间位来作为高速缓存的索引