况最好用一个循环来表示，所以对 $k=2$ 的情况，我们同样也采用这个编程惯例。我们称这种变换为“$k\times1$ 循环展开”，因为循环展开因子为 k，而累积值只在单个变量 acc 中。

```
/* 2 x 1 loop unrolling */
void combine5(vec_ptr v, data_t *dest)
{
    long i;
    long length = vec_length(v);
    long limit = length-1;
    data_t *data = get_vec_start(v);
    data_t acc = IDENT;

    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
        acc = (acc OP data[i]) OP data[i+1];
    }

    /* Finish any remaining elements */
    for (; i < length; i++) {
        acc = acc OP data[i];
    }
    *dest = acc;
}
```

图 5-16 使用 2×1 循环展开。这种变换能减小循环开销的影响

练习题 5.7 修改 combine5 的代码，展开循环 $k=5$ 次。

当测量展开次数 $k=2$(combine5)和 $k=3$ 的展开代码的性能时，得到下面的结果：

函数	方法	整数		浮点数	
		+	*	+	*
combine4	无展开	1.27	3.01	3.01	5.01
combine5	2×1 展开	1.01	3.01	3.01	5.01
	3×1 展开	1.01	3.01	3.01	5.01
延迟界限		1.00	3.00	3.00	5.00
吞吐量界限		0.50	1.00	1.00	0.50

我们看到对于整数加法，CPE 有所改进，得到的延迟界限为 1.00。会有这样的结果是得益于减少了循环开销操作。相对于计算向量和所需要的加法数量，降低开销操作的数量，此时，整数加法的一个周期的延迟成为了限制性能的因素。另一方面，其他情况并没有性能提高——它们已经达到了其延迟界限。图 5-17 给出了当循环展开到 10 次时的 CPE 测量值。对于展开 2 次和 3 次时观察到的趋势还在继续——没有一个低于其延迟界限。

要理解为什么 $k\times1$ 循环展开不能将性能改进到超过延迟界限，让我们来查看一下 $k=2$ 时，combine5 内循环的机器级代码。当类型 data_t 为 double，操作为乘法时，生成如下代码：

```
Inner loop of combine5. data_t = double, OP = *
i in %rdx, data %rax, limit in %rbp, acc in %xmm0
.L35:                                       loop:
  vmulsd  (%rax,%rdx,8), %xmm0, %xmm0         Multiply acc by data[i]
  vmulsd  8(%rax,%rdx,8), %xmm0, %xmm0        Multiply acc by data[i+1]
```