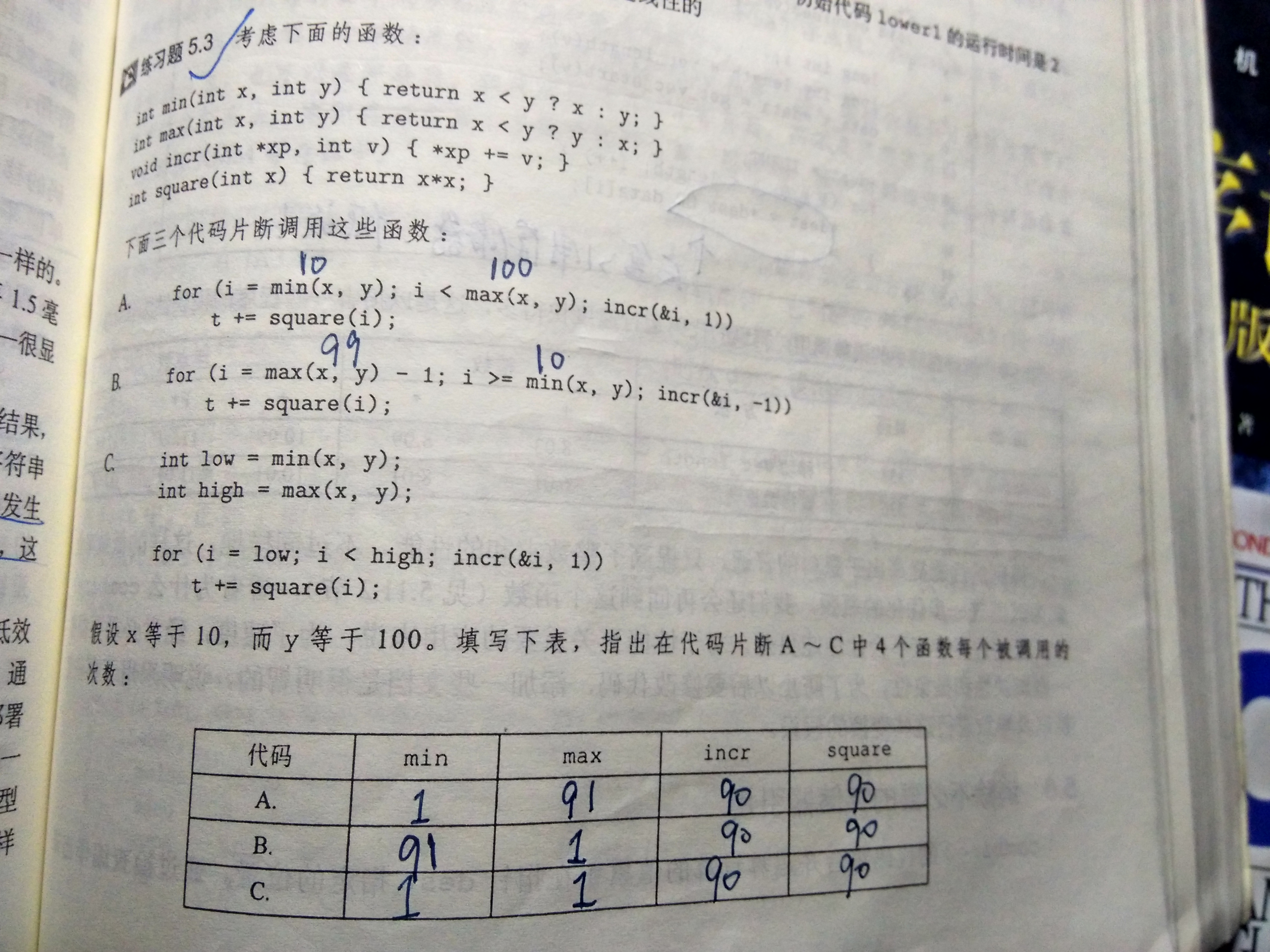
**5.3**



**5.4**

A．在没经过优化的代码中，寄存器%xmm0简单地被用作临时值，每次循坏迭代中都会被设置和使用。在经过更多优化的代码中，它被使用的方式更像是combine4中的变量x，积累向量元素的乘积。不过，与combine4的区别主要在于每次迭代第二条movss指令都会更新dest。

B．两程序拥有一样的行为，包括具有存储器别名使用的时候。

C. 这个变换可以不改变程序的行为，因为除了第一次迭代，每次迭代开始的时从dest中读出的值和前一次迭代最后写入到这个寄存器中的值是相同的。因此，合并指令可以简单地使用在循坏开始时就已经在%xmm0中的值。

**5.10**

A. 对于0 <= i <= 998，它将每个元素a[i]设置为i+1

B. 对于1 <= i <= 999，它将每个元素a[i]设置为0

C. 在第2种情况中，每次迭代的加载都依赖于前一次迭代的存储结果。因此，在连续的迭代之间有写/读相关。有一个很有趣的现象值得注意，它的CPE等于5.00，比对函数write\_read的示例B测量到的CPE小1。这是由于write\_read在存储这个值之前先增加了它，这需要一个时钟周期。

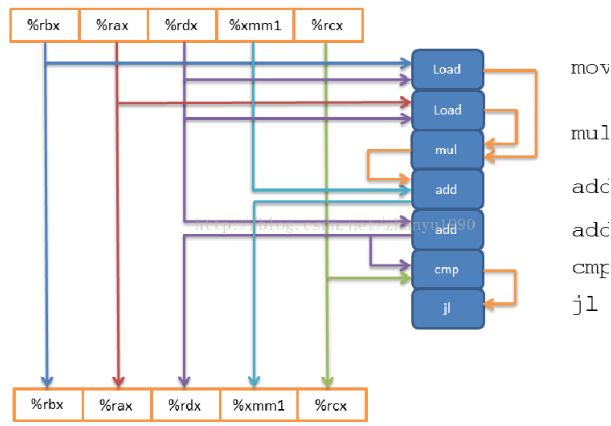
D. 得到的CPE等于2.00，与示例A相同，这是由于 存储和后续的加载之间没有相关。

**5.11**

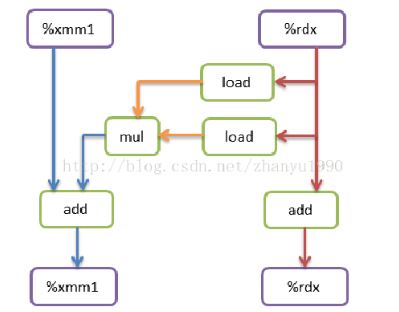
我们看到，这个函数在连续的迭代之间有写/读相关——一次迭代的目的值p[i]与下一次的源值p[i-1]相同。

**5.15**

A.



关键路径是%xmm1更新路径上的加法。



B. CPE下界是浮点加法的延迟。

C. 两个load操作的吞吐量界限。（我觉得是2.00）

D. 因为乘法不在关键路径上，乘法也是流水线执行的，其限制因素为吞吐量界限。整个程序的限制因素为最后的浮点数加法的延迟，这个延迟对float和double都是3.00。