图 5-36b 说明了当移走那些不直接影响迭代与迭代之间数据流的操作之后，会发生什么。这个数据流图给出两个相关链：左边的一条，存储、加载和增加数据值(只对地址相同的情况有效)，右边的一条，减小变量 cnt。

现在我们可以理解函数 write_read 的性能特征了。图 5-37 说明的是内循环的多次迭代形成的数据相关。对于图 5-33 示例 A 的情况，有不同的源和目的地址，加载和存储操作可以独立进行，因此唯一的关键路径是由减少变量 cnt 形成的，这使得 CPE 等于 1.0。对于图 5-33 示例 B 的情况，源地址和目的地址相同，s_data 和 load 指令之间的数据相关使得关键路径的形成包括了存储、加载和增加数据。我们发现顺序执行这三个操作一共需要 7 个时钟周期。

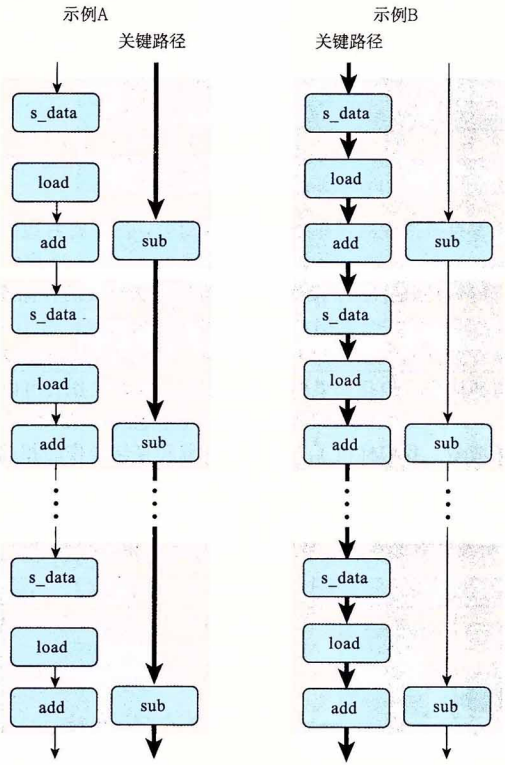

图 5-37　函数 write_read 的数据流表示。当两个地址不同时，唯一的关键路径是减少 cnt(示例 A)。当两个地址相同时，存储、加载和增加数据的链形成了关键路径(示例 B)

这两个例子说明，内存操作的实现包括许多细微之处。对于寄存器操作，在指令被译码成操作的时候，处理器就可以确定哪些指令会影响其他哪些指令。另一方面，对于内存操作，只有到计算出加载和存储的地址被计算出来以后，处理器才能确定哪些指令会影响其他的哪些。高效地处理内存操作对许多程序的性能来说至关重要。内存子系统使用了很多优化，例如当操作可以独立地进行时，就利用这种潜在的并行性。