addr 指令计算存储操作的地址，在存储缓冲区创建一个条目，并且设置该条目的地址字段。s_data 操作设置该条目的数据字段。正如我们会看到的，两个计算是独立执行的，这对程序的性能来说很重要。这使得参考机中不同的功能单元来执行这些操作。

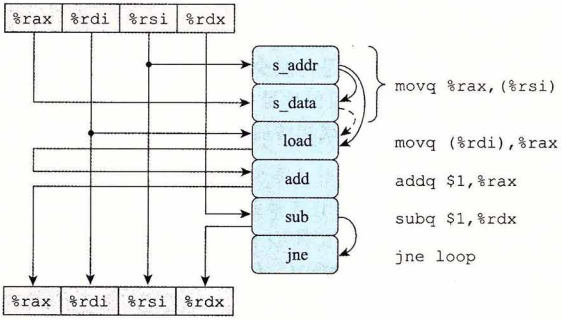

图 5-35　write_read 内循环代码的图形化表示。第一个 movl 指令被译码两个独立的操作，计算存储地址和将数据存储到内存

除了由于写和读寄存器造成的操作之间的数据相关，操作符右边的弧线表示这些操作隐含的相关。特别地，s_addr 操作的地址计算必须在 s_data 操作之前。此外，对指令 movq (%rdi),%rax 译码得到的 load 操作必须检查所有未完成的存储操作的地址，在这个操作和 s_addr 操作之间创建一个数据相关。这张图中 s_data 和 load 操作之间有虚弧线。这个数据相关是有条件的：如果两个地址相同，load 操作必须等待直到 s_data 将它的结果存放到存储缓冲区中，但是如果两个地址不同，两个操作就可以独立地进行。

图 5-36 说明了 write_read 内循环操作之间的数据相关。在图 5-36a 中，重新排列了操作，让相关显得更清楚。我们标出了三个涉及加载和存储操作的相关，希望引起大家特别的注意。标号为(1)的弧线表示存储地址必须在数据被存储之前计算出来。标号为(2)的弧线表示需要 load 操作将它的地址与所有未完成的存储操作的地址进行比较。最后，标号为(3)的虚弧线表示条件数据相关，当加载和存储地址相同时会出现。

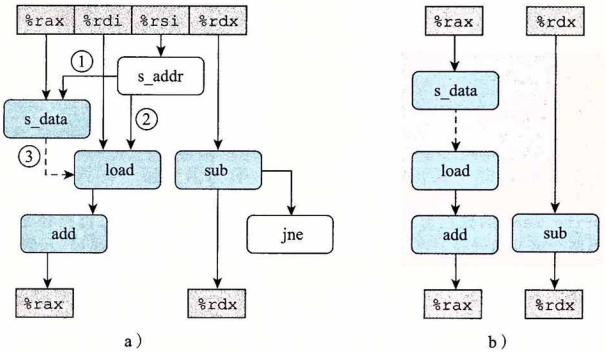

图 5-36　抽象 write_read 的操作。我们首先重新排列图 5-35 的操作(a)，然后只显示那些使用一次迭代中的值为下一次迭代产生新值的操作(b)