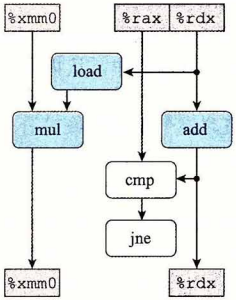
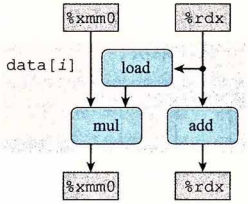
值 data+i 和 acc。

正如我们会看到的，循环寄存器之间的操作链决定了限制性能的数据相关。

图 5-14 是对图 5-13 的图形化表示的进一步改进，目标是只给出影响程序执行时间的操作和数据相关。在图 5-14a 中看到，我们重新排列了操作符，更清晰地表明了从顶部源寄存器(只读寄存器和循环寄存器)到底部目的寄存器(只写寄存器和循环寄存器)的数据流。



a) 重新排列了图5-13的操作符，更清晰地表明了数据相关



b) 操作在一次迭代中使用某些值，产生出在下一次迭代中需要的新值

图 5-14　将 combine4 的操作抽象成数据流图

在图 5-14a 中，如果操作符不属于某个循环寄存器之间的相关链，那么就把它们标识成白色。例如，比较(cmp)和分支(jne)操作不直接影响程序中的数据流。假设指令控制单元预测会选择分支，因此程序会继续循环。比较和分支操作的目的是测试分支条件，如果不选择分支的话，就通知 ICU。我们假设这个检查能够完成得足够快，不会减慢处理器的执行。

在图 5-14b 中，消除了左边标识为白色的操作符，而且只保留了循环寄存器。剩下的是一个抽象的模板，表明的是由于循环的一次迭代在循环寄存器中形成的数据相关。在这个图中可以看到，从一次迭代到下一次迭代有两个数据相关。在一边，我们看到存储在寄存器%xmm0 中的程序值 acc 的连续的值之间有相关。通过将 acc 的旧值乘以一个数据元素，循环计算出 acc 的新值，这个数据元素是由 load 操作产生的。在另一边，我们看到循环索引 i 的连续的值之间有相关。每次迭代中，i 的旧值用来计算 load 操作的地址，然后 add 操作也会增加它的值，计算出新值。

图 5-15 给出了函数 combine4 内循环的 n 次迭代的数据流表示。可以看出，简单地重

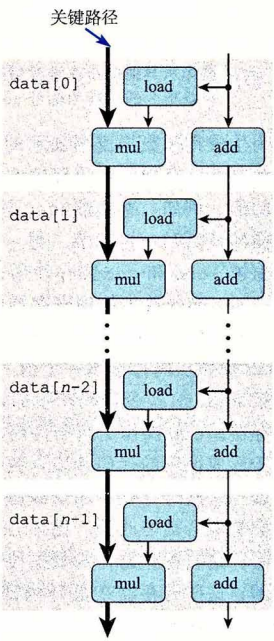


图 5-15　combine4 的内循环的 n 次迭代计算的数据流表示。乘法操作的序列形成了限制程序性能的关键路径