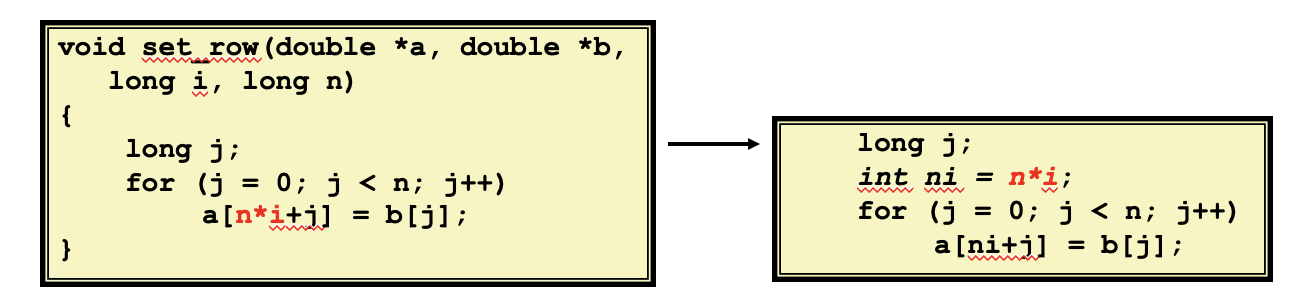
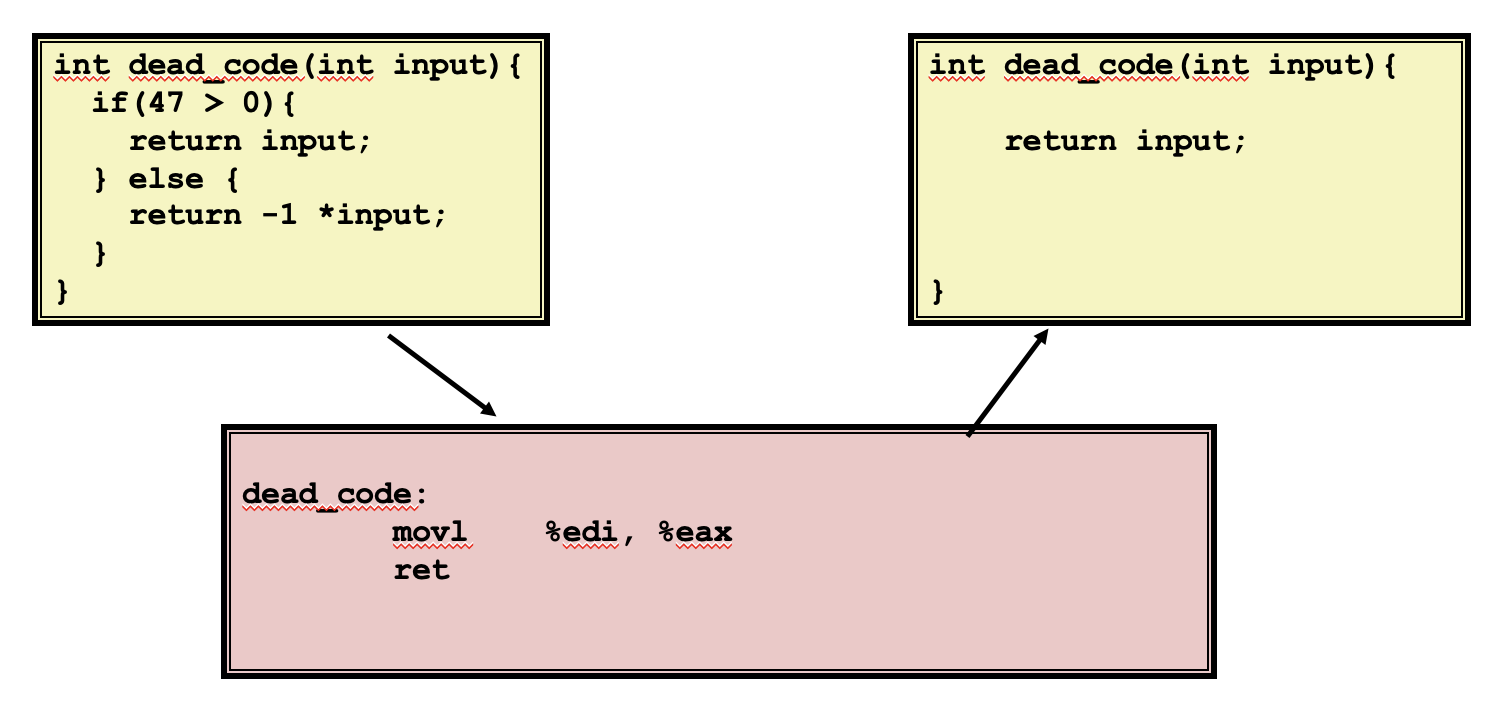
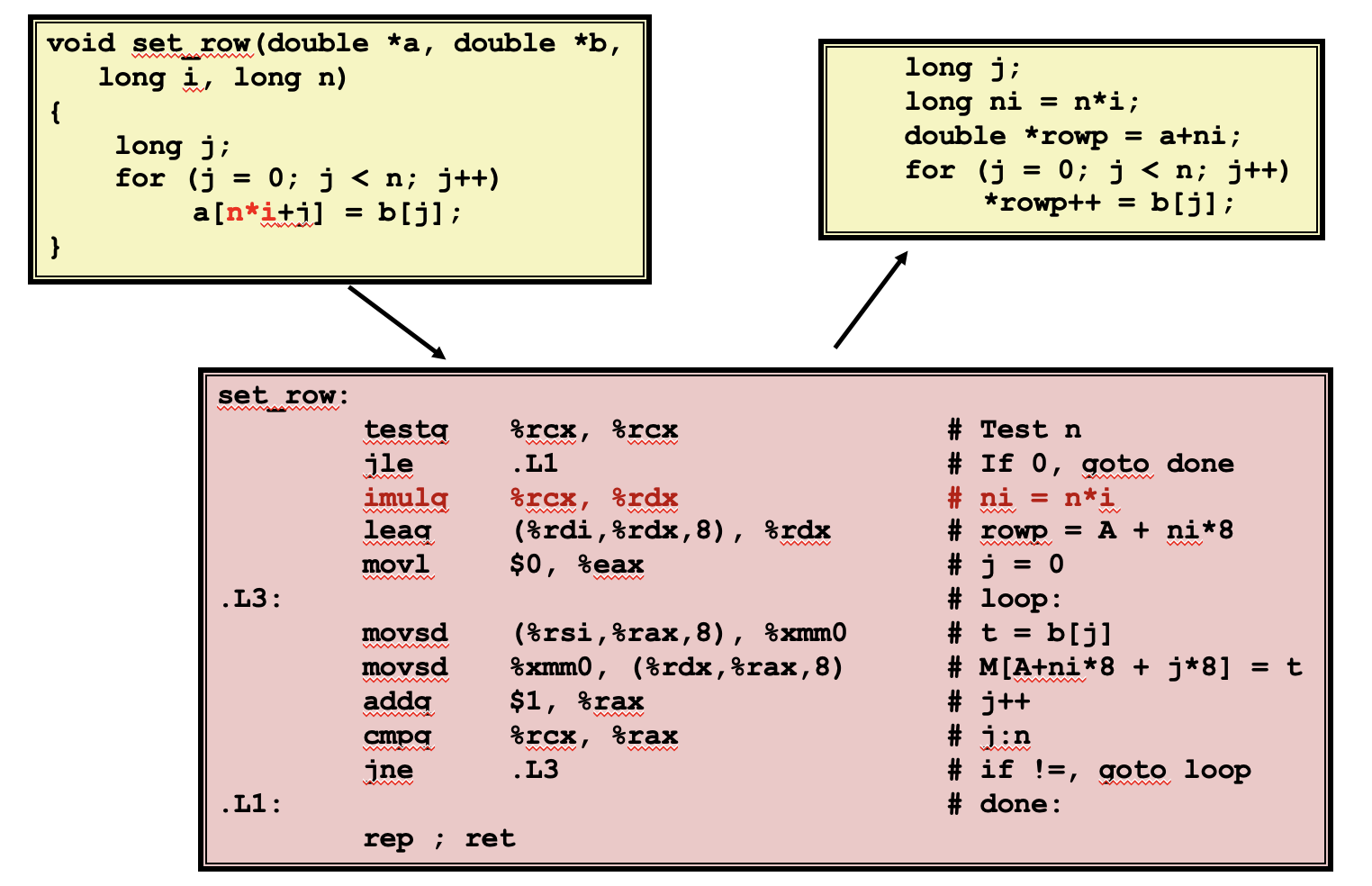
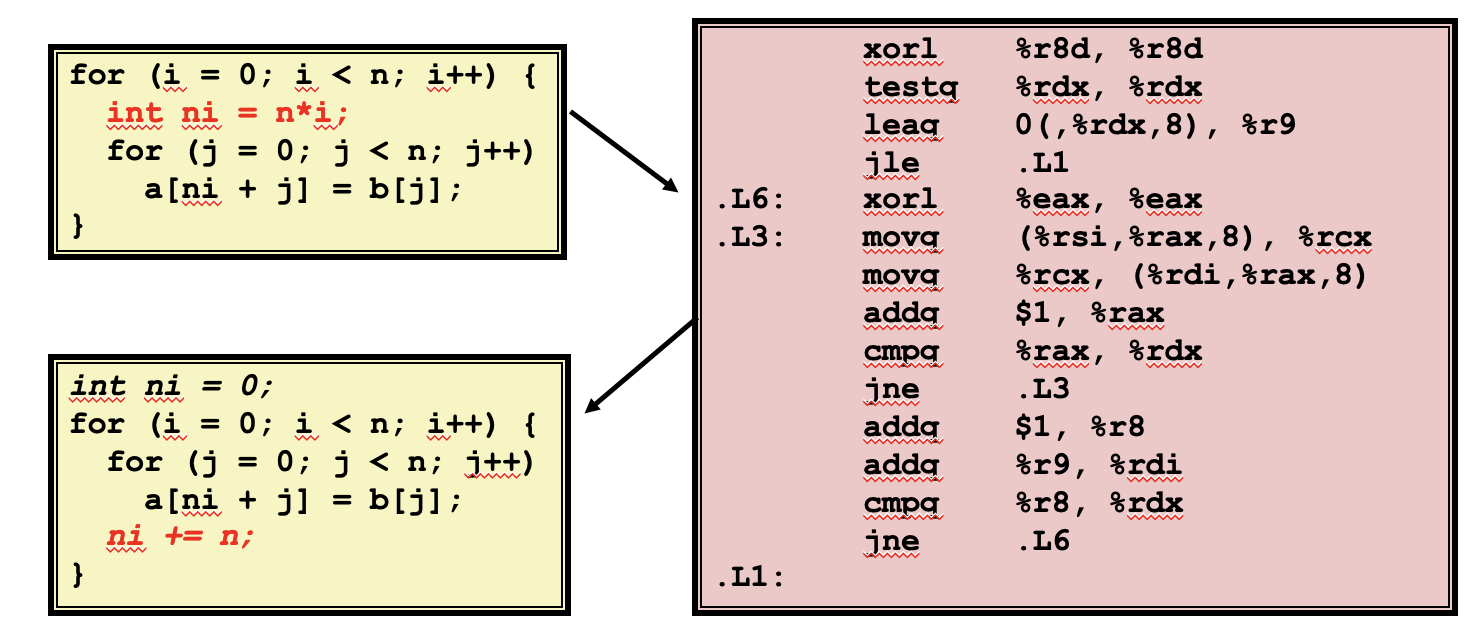
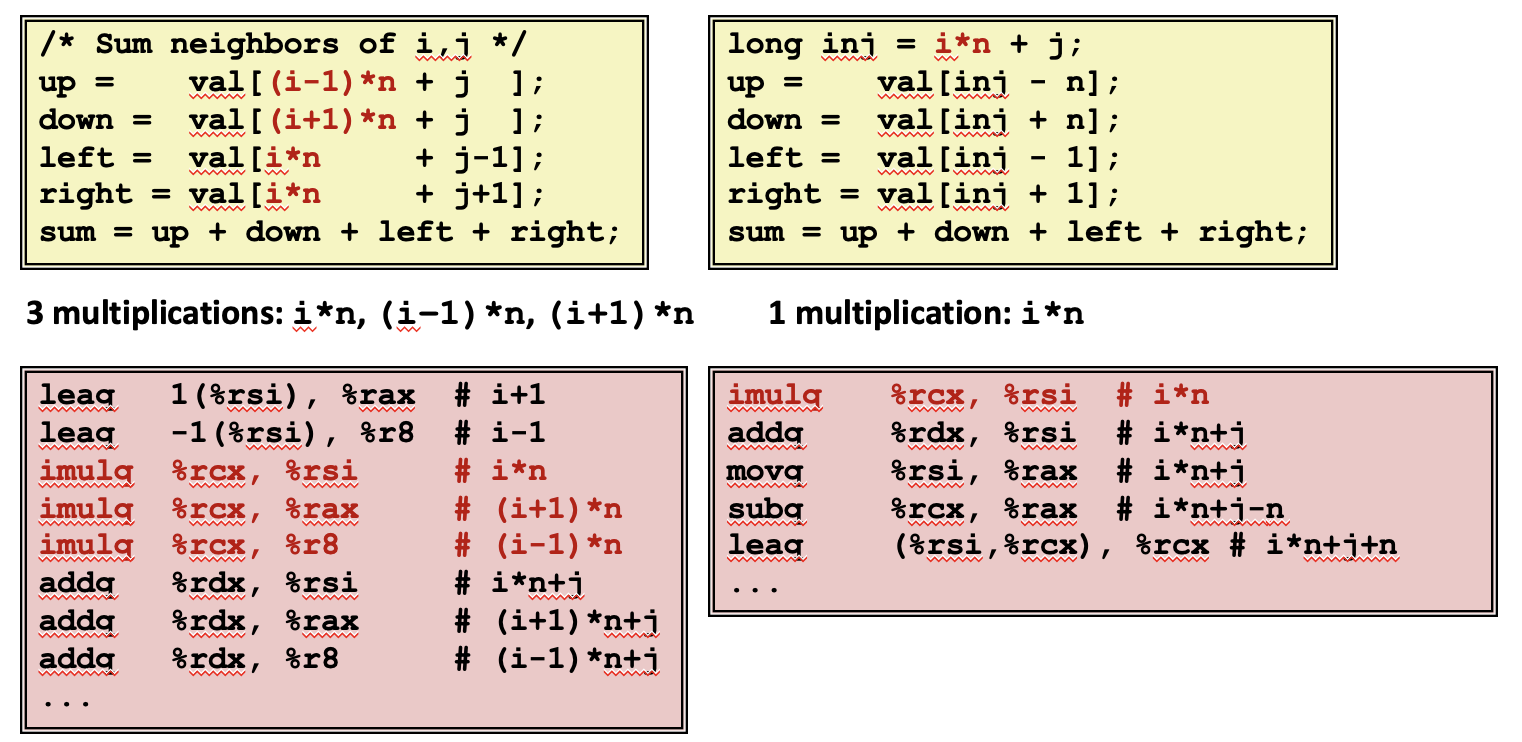
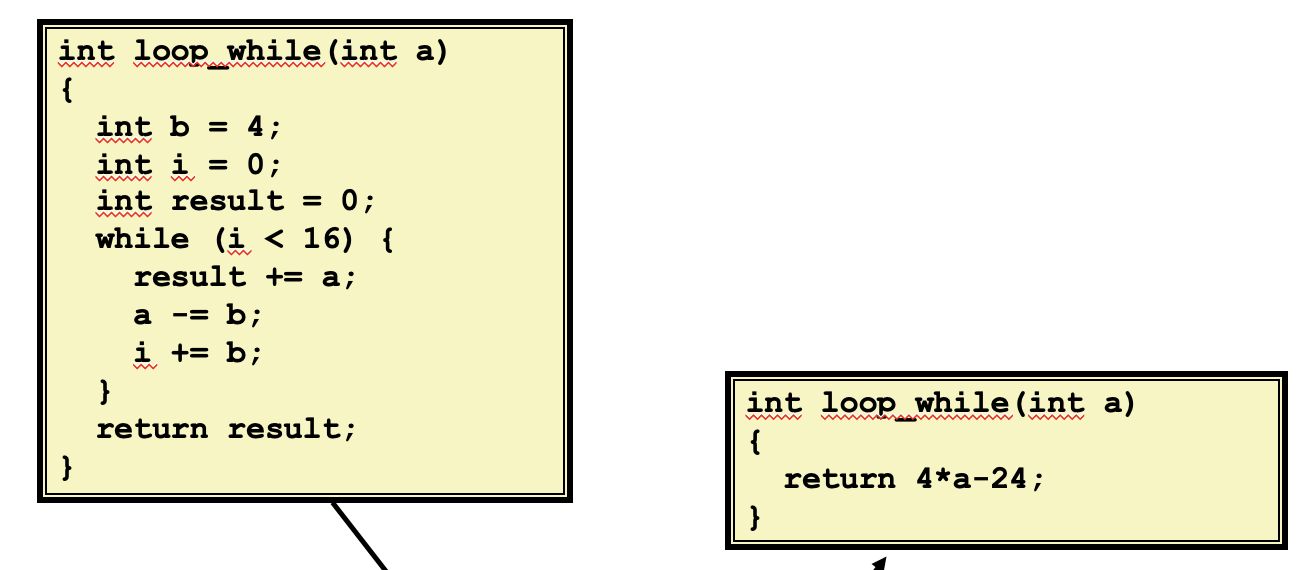
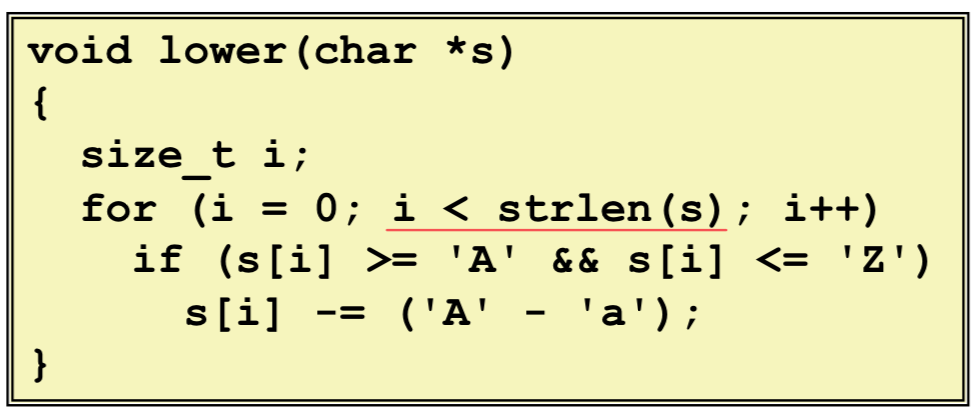
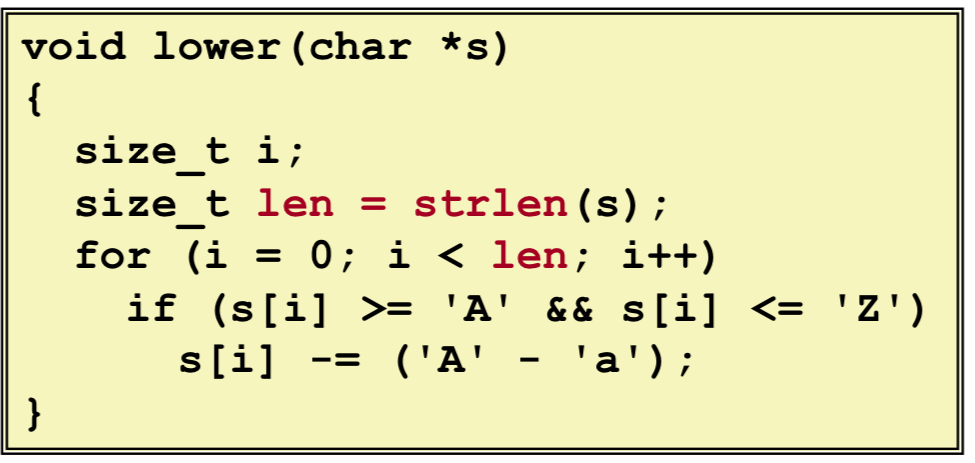
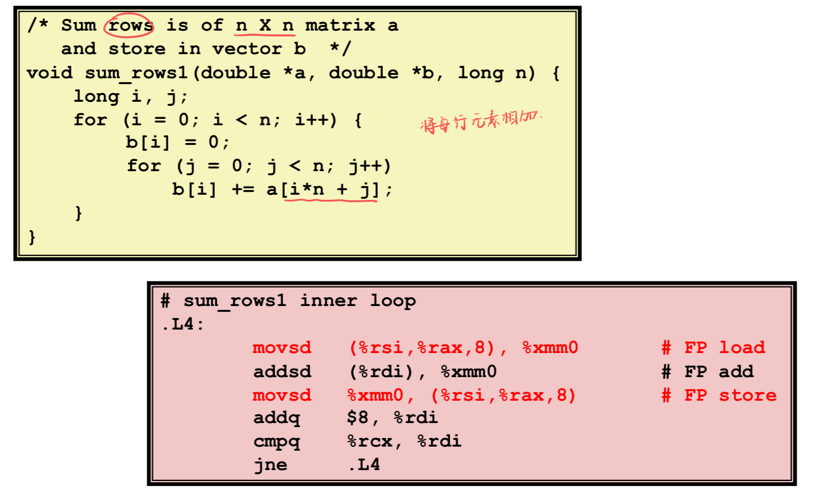
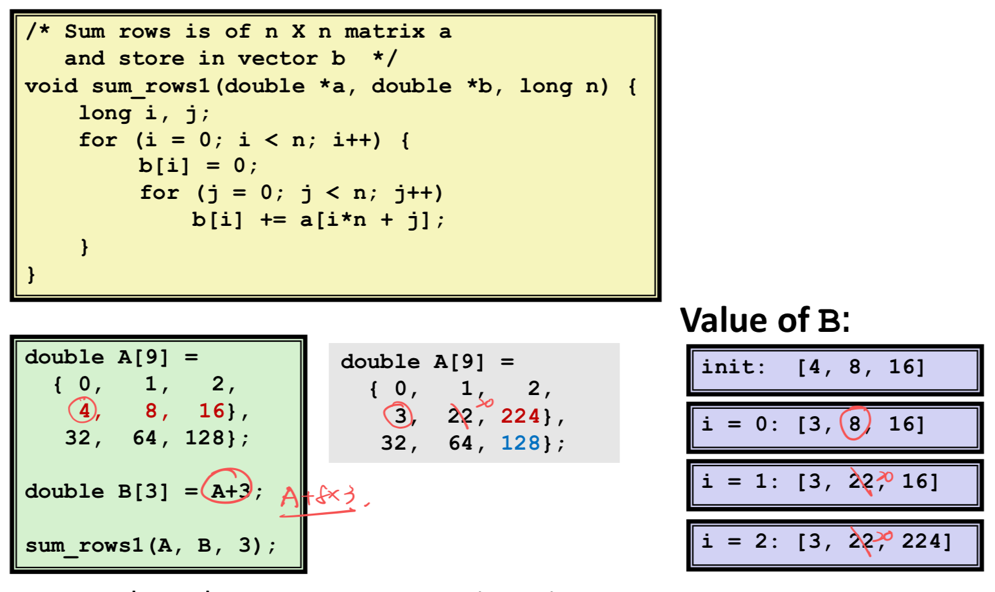
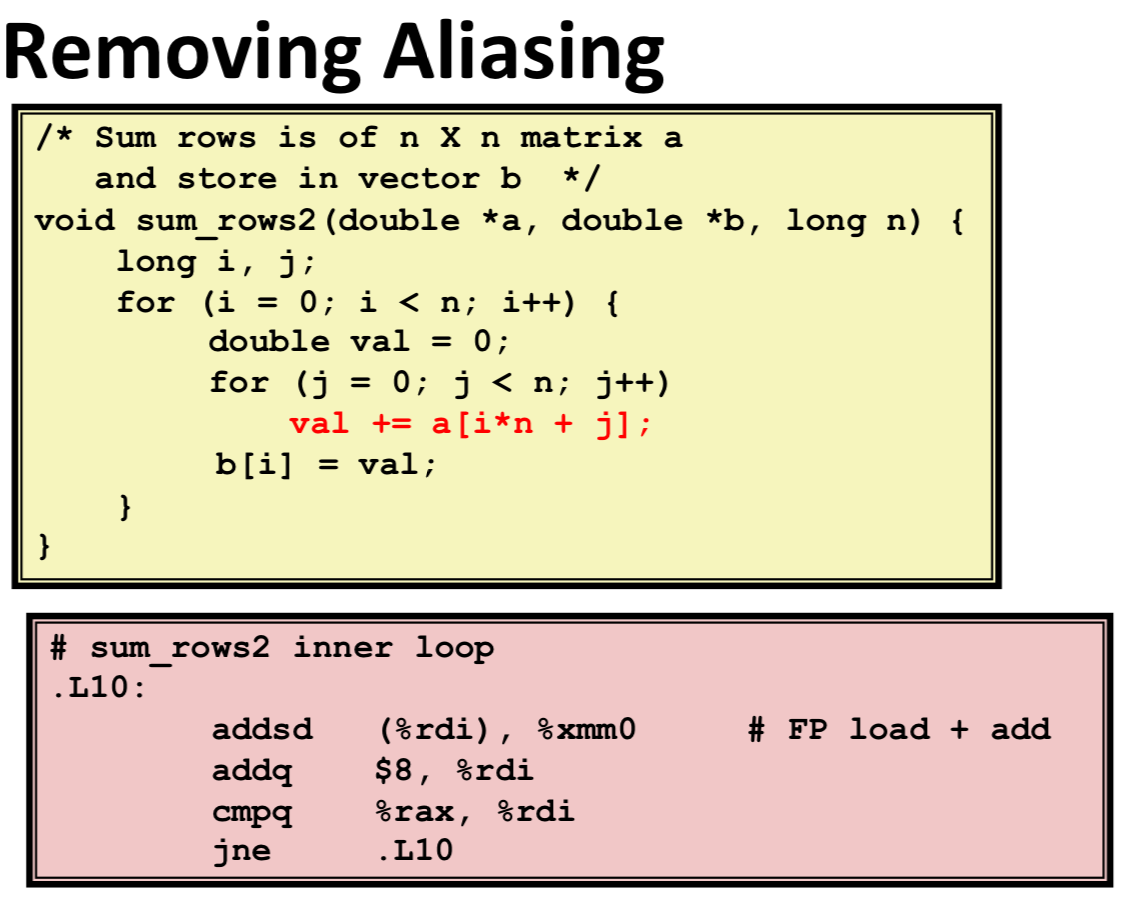
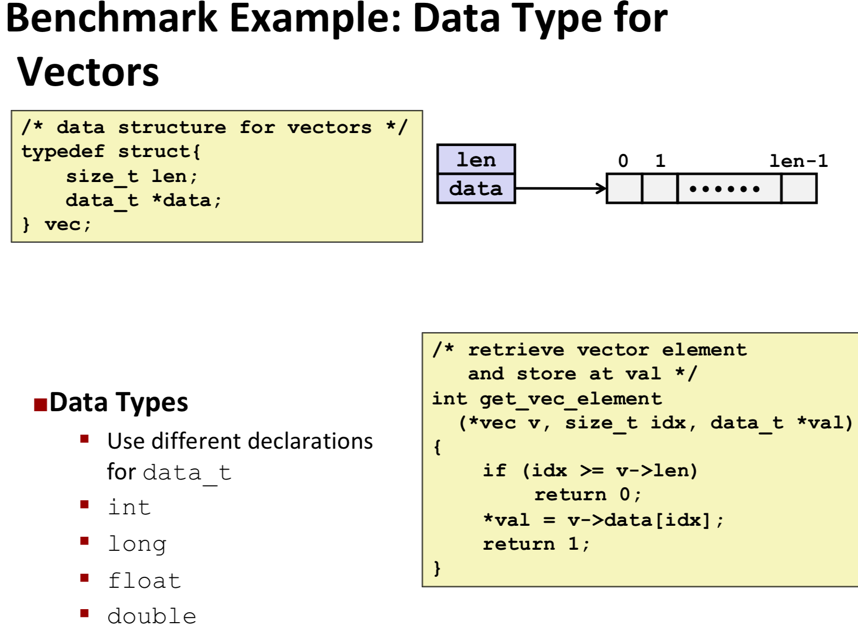
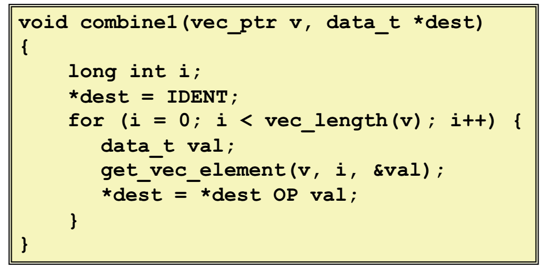
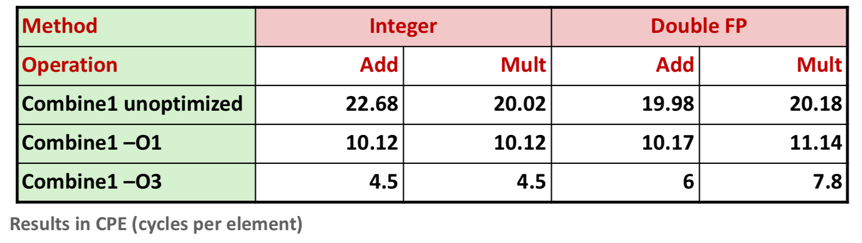
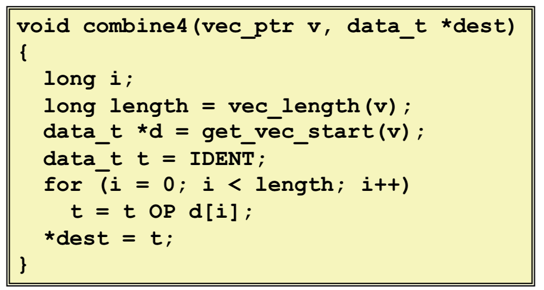
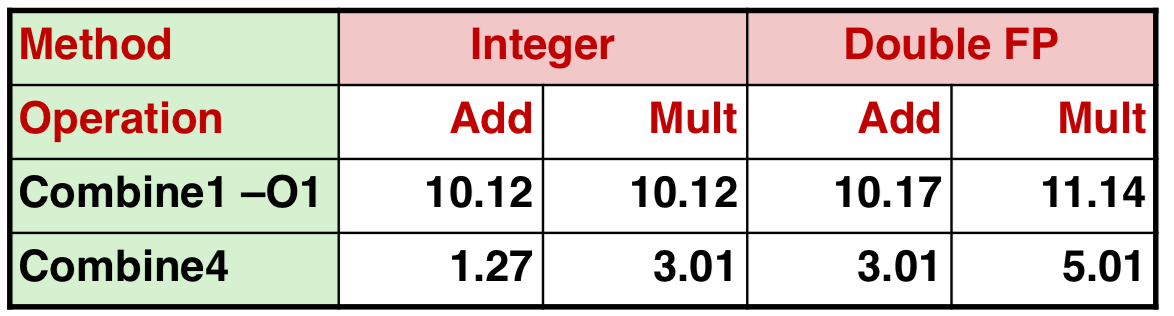
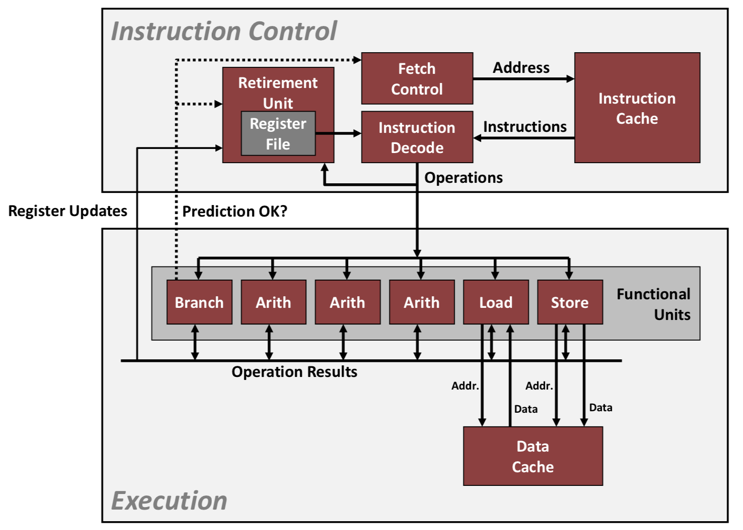
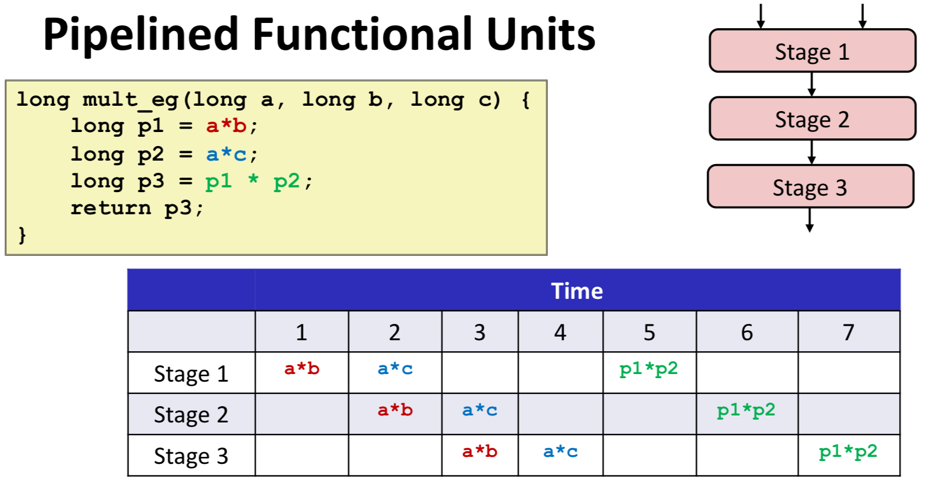
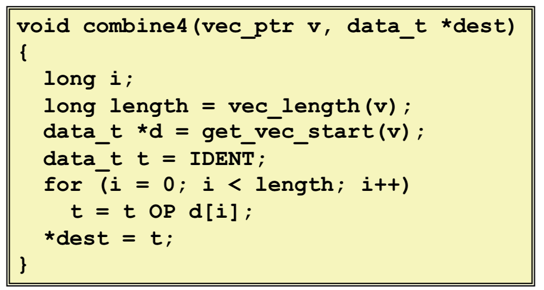
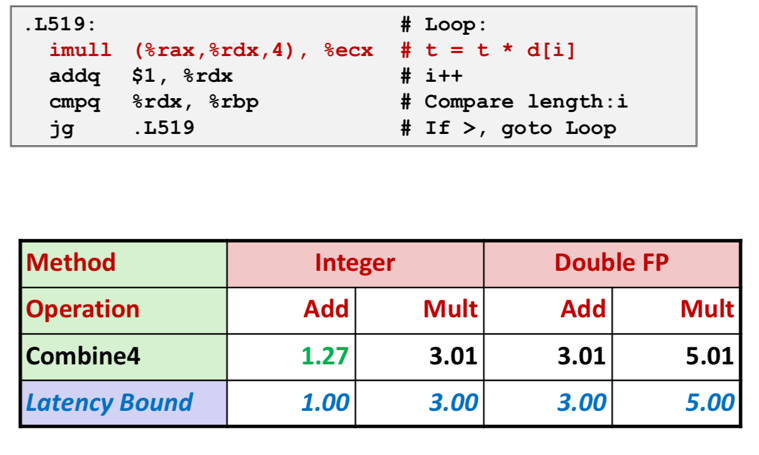
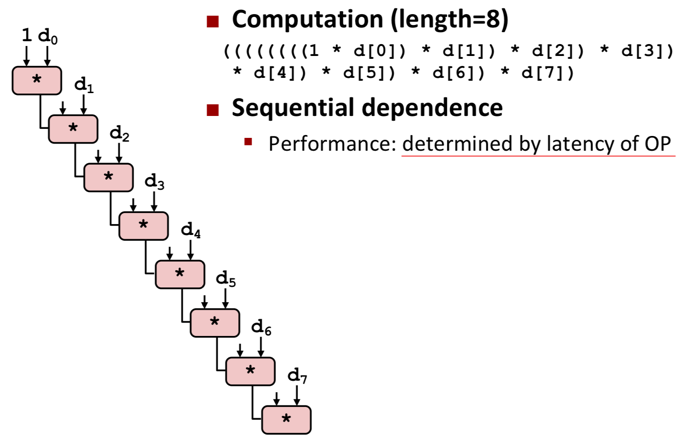
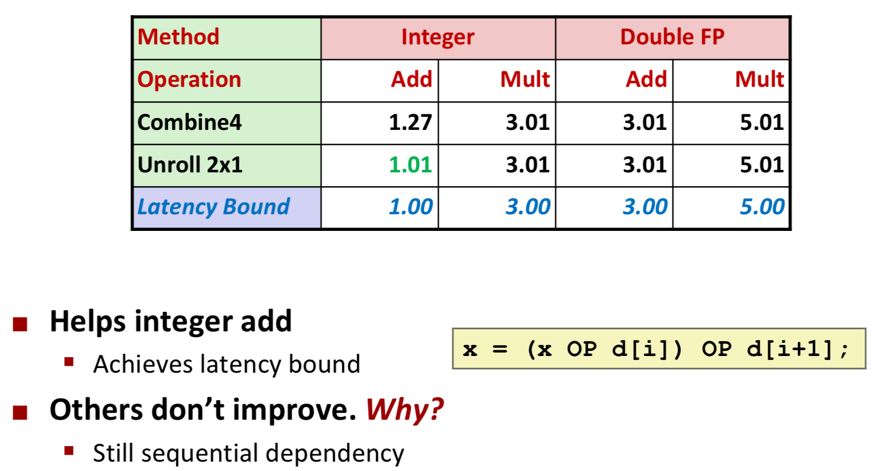
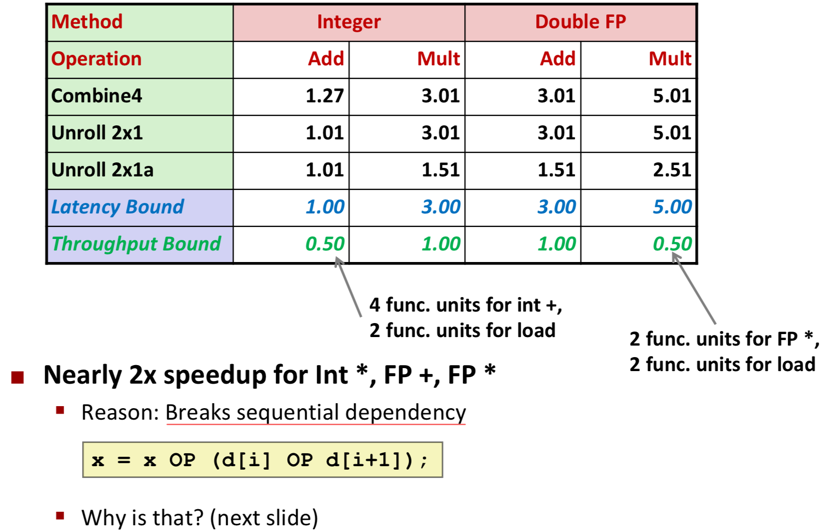
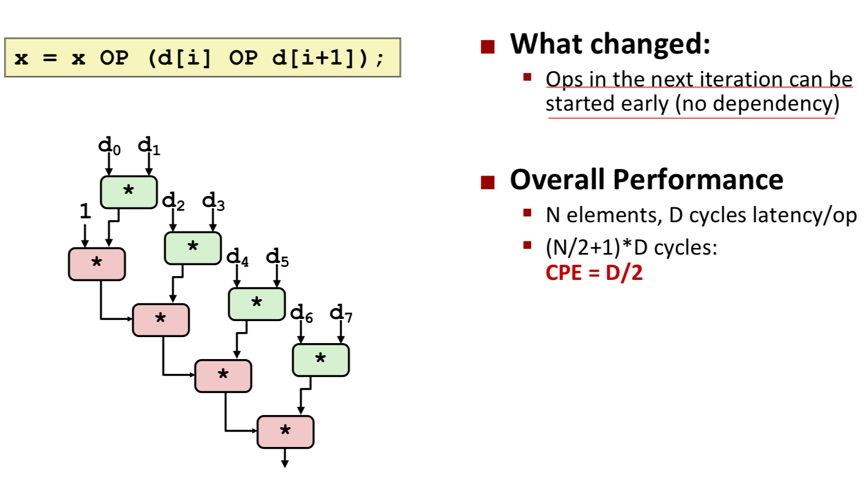
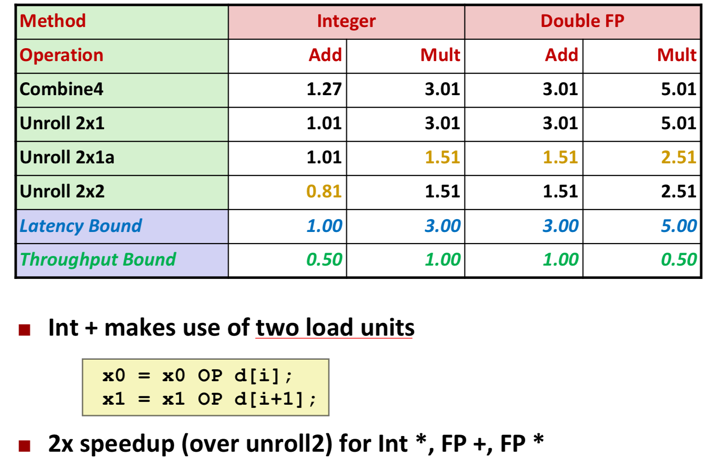
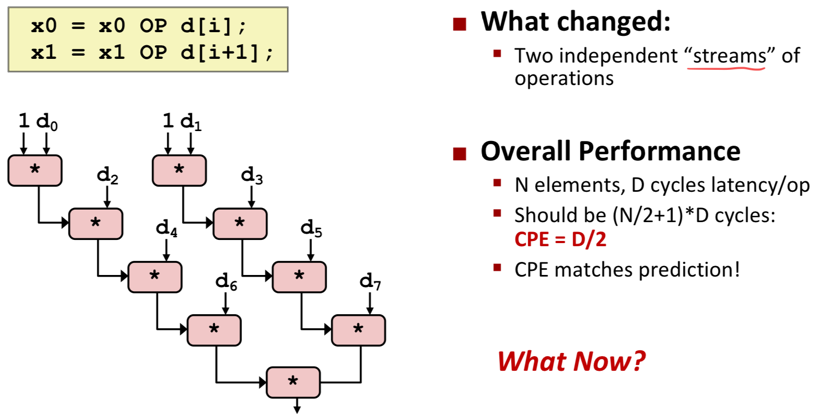
# 第5章 优化程序性能

* **简介**
  + **程序性能**
    - 渐进复杂度
    - 常数
  + **优化可用的方法**
    - 使用更好的数据结构与算法
    - 编译为高效的二进制代码
      * 熟悉编译器的标志和配置优化
    - 编写编译器可以有效优化的源代码，以高效执行代码
      * 了解优化编译器的功能和局限性
    - 让指令并行执行
      * 将任务分为可并行执行的部分
      * 利用指令级并行性、线程级并行性和任务级并行性
  + **编译器的自动优化**
    - 提供程序到机器的高效映射
      * 寄存器分配
      * 代码选择和排序（调度）
      * 消除轻微的低效率
    - 一般不会改进算法的渐进复杂度
      * 由程序员选择最佳算法
      * 在算法和数据结构的设计中，通常更重要的是优化时间复杂度的阶数（通常用大O表示），而不是关注常数因子的微小优化（但是常数优化也很重要）
    - 难以克服“优化障碍”
      * 潜在的内存别名（两个或多个不同的指针引用了相同的内存位置或地址）
      * 潜在的过程副作用
* **通用的优化策略**
  + 不依赖于特定的处理器架构或编译器
  + **代码移动（-O1）**
  + **清除无用的代码（-O0）**
  + **用耗时短的指令替代耗时长的（-O2）**
  + **共享公共子表达式（-O1）**
  + **消除循环（-O1）**
* **优化障碍**
  + **编译器优化的局限性**
    - 在基本约束下优化
      * 不能产生任何程序行为的改变
      * 哪怕只有在很特殊的情况下可能改变程序行为，编译器也不会选择优化
    - 对程序员来说可能是显而易见的行为可能会被语言和编码风格所混淆
      * 数据范围可能比变量类型所建议的更有限
    - 大多数分析仅在过程内进行
      * 在大多数情况下，全程序分析成本太高
      * 较新版本的 GCC 在单个文件中执行过程间分析，但不是在不同文件中的代码之间
    - 大多数分析仅基于静态信息
      * 编译器难以预测运行时输入
    - 当有疑问时，编译器必须保守
  + **障碍1:过程调用**
    - 类似于代码移动，但是编译器不会自动执行优化
      * 过程可能有副作用，每次调用时可能更改全局状态
      * 对于给定的参数，优化前后函数可能不会返回相同的值

取决于全局状态的其他部分（不能保证lower没有改变s长度）

lower 可能与 strlen 相互作用

* 编译器将过程调用视为黑盒
  + 不会管过程具体有没有副作用
  + 过程附近的优化一般都很弱
* 优化措施
  + 使用内联函数（-O1）
  + 自行移动
* **障碍2:可能的内存别名**
  + 内存别名
    - 两个不同的内存引用指定同一位置
    - 在C中很容易发生
    - 养成引入局部变量的习惯——告诉编译器不要检查内存别名的方法
* **利用指令级并行**
  + **简介**
    - 需要对现代处理器设计有大致的了解
      * 硬件可以并行执行多条指令
    - 性能受数据相关限制
    - 简单的转换可以显著提高性能
      * 编译器通常无法进行这些转换
      * 浮点运算中缺乏关联性和分布性
*  combine1->combine4
  + 将vec\_length移动到循环外
  + 避免每次循环时的边界检查（get\_vec\_element中）
  + 用临时变量存储累计值
  + 消除了一些开销的来源
* 现代处理器设计
* 超标量处理器
  + 定义：超标量处理器可以在一个周期内发出和执行多个指令。指令是从顺序指令流中检索的，通常是动态调度的
  + 优势：无需编程工作，超标量处理器可以利用大多数程序所具有的指令级并行性
* 流水线化的功能单元
* 将计算划分为阶段
* 将部分计算从一个阶段传递到另一个阶段
* 一旦值传递给 i+1，阶段 i 就可以开始新的计算
* 例如，程序用7个周期完成3次乘法，即使每次乘法需要3个周期
* **循环展开**
* 2\*1
* 2\*1a
* 2\*2
* 总结
  + 思路

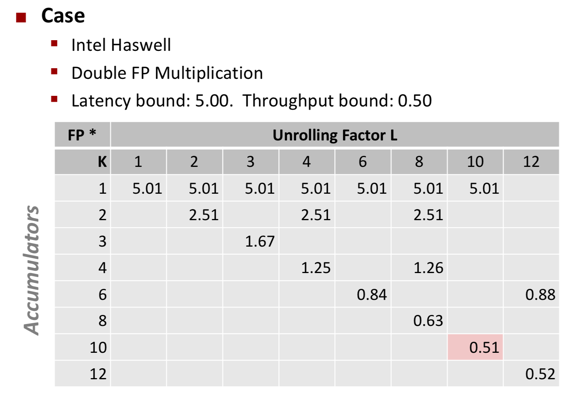
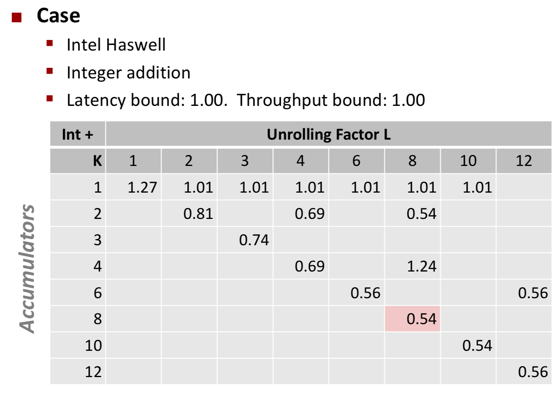
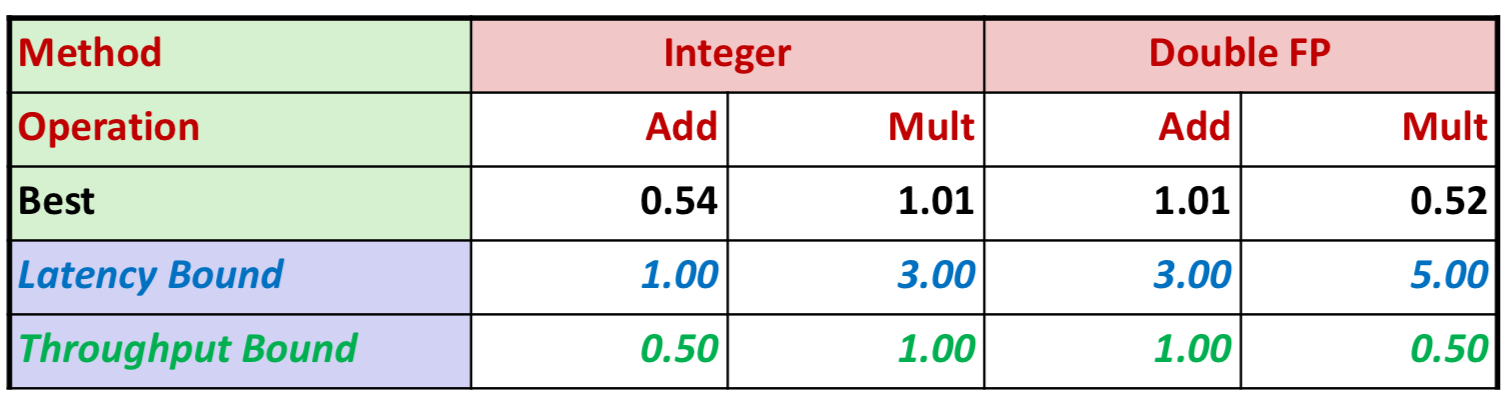
可以展开到任意次L

可以并行计算K个结果

L必须是K的倍数

* 局限性

不能超越执行单元的吞吐量限制

短长度开销大（顺序执行即可）

* **处理条件分支**
  + **难点**
    - 指令控制单元必须在执行单元之前很好地工作，以产生足够的操作来保持执行单元维持流水线工作状态
    - 遇到条件分支时，无法可靠地确定继续获取指令编码的位置
      * Branch Taken & Branch Not-Taken
    - 在分支/整数单元确定结果之前不能修改程序员可见状态
  + **分支预测逻辑**
    - 思路
      * 猜测是否跳转
      * 开始在预测位置执行指令，但不能真的修改寄存器文件和数据内存
      * 预测错了要取消后续指令，重新装载流水线

现代处理器上需要多个时钟周期

可成为限制程序性能的主要因素