寄存器分配

为了将程序的语法树转化为汇编代码主要需要经过这样一些环节。以处理 while+db 语言为例：

- 拆分复合表达式
- 生成基本块
- 分配寄存器
- 生成汇编代码

1 寄存器分配之前的步骤

1.1 拆分复合表达式

- 关键点 1：每个赋值语句至多进行一步计算

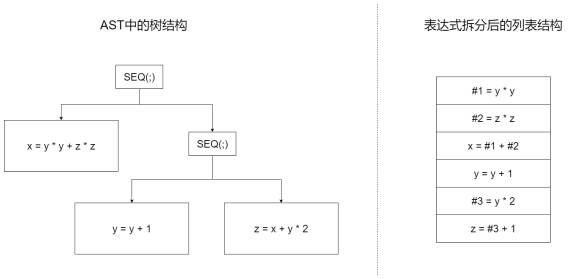
  拆分前

  ```
  x = (y * y + z * z) * 4
  ```

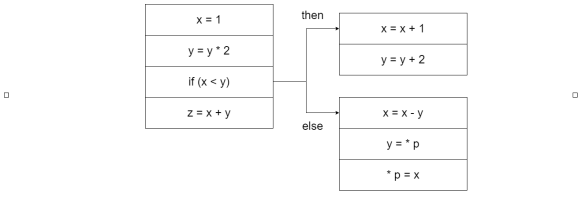
  拆分后

  ```
  #0 = y * y;
  #1 = z * z;
  #2 = #0 + #1
  x = #2 * 4
  ```

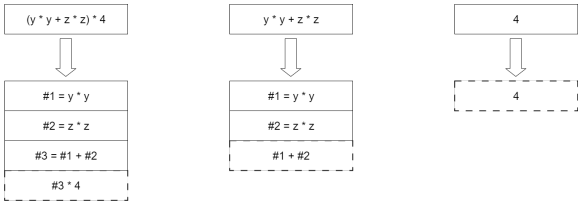
- 关键点 2：转化后的程序仅在必要处保留树结构



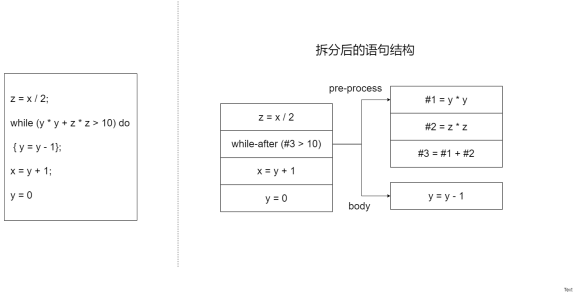
语句层面，if 语句和循环语句依旧保留树结构，但是顺序执行不再保留树结构。



递归拆分复杂表达式，拆分的结果是一个简单赋值语句序列，和一个简单表达式。



- 关键点 3：要妥善处理 while 循环条件的计算语句



- 关键点 4：要妥善处理短路求值

  由于短路求值的存在，and 与 or 必须转化为 if 语句。例如：

  拆分前

```
if (p && * p != 0)
then { ... }
else { ... }
```

  拆分后

```
if (p)
then { #2 = * p;
       #1 = (#2 != 0) }
else { #1 = 0 };
if (#1)
then { ... }
else { ... }
```

1.2 生成基本块

2 寄存器分配

2.1 活性分析

- 计算方法：

$$in(u) = (out(u) \backslash def(u)) \cup use(u)$$

$$out(u) = \bigcup_{u \to v} in(v)$$

- 生成 interference graph：如果存在一个 u 使得 $x, y \in in(u)$，那么就在 x 与 y 之间连一条边，表示它们不能分配同一个寄存器。

2.2 简单寄存器分配算法

本课程只考虑最简单的寄存器分配，即每个变量对应一个寄存器，多个变量可能对应相同的寄存器。较为前沿高效的寄存器分配算法，有时会对同一寄存器的不同使用位置分配不同的寄存器。

- 假设共有 K 个寄存器；
- 步骤一，Simplify：在 interference graph 中删除度数不超过 $K-1$ 的节点，删除的节点用栈记录；
- 步骤二，Spill：如果无法 Simplify，就任意删除一个节点，再回到前面步骤一；
- 步骤三，Select：按照删除节点的倒序为所有变量分配寄存器；

  注：Simplify 删除的节点能够保证被分配到与 interference graph 上相邻节点不冲突的寄存器，Spill 变量可能可以分配到寄存器（假 Spill），也可能无法分配到寄存器（真 Spill），此时应当存储在内存中；
- 步骤四，Start over：如果有至少一个节点无法分配到寄存器，则改写相关代码并重做 liveness 分析与上述所有步骤。
- 如果 3 号变量无法分配寄存器，并且存储在 `%rbp - 16` 地址，那么：

  ```
  #2 = #3 + 1
  ```

  ```
  #3 = * #4
  ```

  分别会被改写为：

  ```
  #3 = * (%rbp - 16)
  #2 = #3 + 1
  ```

  ```
  #3 = * #4
  * (%rbp - 16) = #3
  ```

  这样，两次使用 `#3` 变量之间的代码 `#3` 都不再是 live 的了。
- 如果所有变量都分配到了寄存器，那么寄存器分配过程结束；如果有至少一个节点无法分配到寄存器，那么在执行完上述程序改写后，变量的 live 区域会大大缩小，此时应当重新进行 liveness 分析，并回到步骤一，从头开始重新分配寄存器，这个过程称为 Start over。往往经过至多 2-3 次 Start over 之后，寄存器分配算法就能顺利结束。

2.3 寄存器分配算法的优化

- 改进思路：如果有 move 指令（形如 `x = y` 的指令）且两个变量的 live 区域不重叠，那么就可以合并这两个变量，并删除 move 指令，这个过程成为合并（Coalesce）；
- 由于合并操作可能导致 interference graph 变稠密，从而导致原本可以不需要 Spill 的情形变得需要 Spill，得不偿失，因此一般只采用保守合并策略:；

  保守合并策略

  - 如果 `x` 与 `y` 两个变量的所有邻居中度数 $\geqslant K$ 的数量 $\leqslant K-1$，那么可以合并 `x` 与 `y` ；
  - 如果 `x` 的每个邻居要么也是 `y` 的邻居，要么度数 $\leqslant K-1$，那么可以合并 `x` 与 `y` ；
- 步骤一，Simplify：只删除 move 无关的度数 $\leqslant K-1$ 的节点；

- 步骤二，Coalesce，进行保守合并；
- 步骤三，Freeze：如果没有可行的 Simplify 和 Coalesce 操作可以继续进行，那么就选择一条 move 指令，放弃对他进行合并，之后回到步骤一与步骤二；
- 上述三类操作都无法进行时则采用 Spill，最后 Select 与 Start Over 的流程与前述简化版本的算法一致。