第11章 代码生成

重点： 代码生成器设计中的问题,目标语言，一个简单的代码生成器,寄存器的分配和指派

难点： 寄存器的分配和指派

第11章 代码生成

11.1 代码生成器设计中的问题

11.2 目标语言

11.3 一个简单的代码生成器

11.4 窥孔优化

11.5 寄存器分配与指派

11.6 本章小结

第11章 代码生成

- 代码生成是编译的最后一个阶段，由代码生成器完成。其任务是把中间代码转换为等价的、具有较高质量的目标代码，以充分利用目标机器的资源。当然，代码生成器本身也必须具有较高的运行效率。
- 目标代码可以是绝对地址的机器代码，或相对地址的机器代码，也可以是汇编代码。
- 本章用微型机的汇编指令来表示目标代码。

11.1 代码生成器设计中的问题

- 虽然代码生成器的具体实现依赖于目标机器的体系结构、指令系统和操作系统，但存储管理、指令选择、寄存器分配和计算顺序等问题却是设计各种代码生成器都要考虑的问题，本节讨论这类共性问题。

11.1.1 代码生成器的输入

- 代码生成器的输入包括中间代码和符号表信息，符号表信息主要用来确定中间代码中的变量所代表的数据对象的运行时地址。
- 假设在代码生成前，编译器的前端已经将源程序扫描、分析和翻译成为足够详细的中间代码，其中变量的值已经可以表示为目标机器能够直接操作的量(位、整数、实数、指针等)；
- 已经完成了必要的类型检查；
- 在需要的地方已经插入了类型转换符；明显的语义错误(如试图把浮点数作为数组下标)也都已经被检测出来了。

11.1.2 目标代码的形式

- 代码生成器的输出是目标代码。目标代码的形式主要有如下3种：
  - **绝对机器语言代码**。所有地址均已定位，可以立即被执行。适于小程序的编译，因为它们可以迅速地被执行。
  - **可重定位的机器语言代码**。允许分别将子程序编译成一组可重定位模块，再由连接装配器将它们和某些运行程序连接起来，转换成能执行的机器语言程序。好处是比较灵活，并能利用已有的程序资源，代价是增加了连接和装配的开销。
  - **汇编语言代码**。生成汇编语言代码后还需要经过汇编程序汇编成可执行的机器语言代码，但其好处是简化了代码生成过程并增加了可读性。

11.1.3 指令选择

- 所谓**指令选择**是指寻找一个合适的机器指令序列来实现给定的中间代码。
- 目标机器指令系统的性质决定了指令选择的难易程度
  - 指令系统的一致性和完备性是两个重要的因素
  - 特殊机器指令的使用和指令速度是另一些重要的因素

11.1.3 指令选择

- 若不考虑目标程序的效率，指令的选择将非常简单：
  - 如：三地址语句x := y + z翻译成如下代码序列：（x，y和z都是静态分配）
  - MOV y, R_0 /* 把y装入寄存器R_0 */
  - ADD z, R_0 /* z加到R_0上 */
  - MOV R_0, x /* 把R_0存入x中 */
- 逐个语句地产生代码，常常得到低质量的代码

11.1.3 指令选择

语句序列 a := b + c

d := a + e

的代码如下

```
MOV     b,    R0
ADD     c,    R0
MOV     R0,   a     -- 若a不再使用，第三条也多余
MOV     a,    R0    -- 多余的指令
ADD     e,    R0
MOV     R0,   d
```

11.1.3 指令选择

- 如果目标机器有加1指令(INC)，则a : = a+1的最有效实现是:

  INC a

而不是

  MOV a, R_0

  ADD #1, R_0

  MOV R_0, a

11.1.4 寄存器分配

- 将运算对象放在寄存器中的指令通常要比将运算对象放在内存中的指令快且短，因此，要想生成高质量的目标代码，必须充分使用目标机器的寄存器
- 寄存器的使用包括：
  - 寄存器分配：为程序的某一点选择驻留在寄存器的一组变量
  - 寄存器指派：确定变量将要驻留的具体寄存器

11.1.4 寄存器分配

- 选择最优的寄存器指派方案是一个NP完全问题，如果考虑到目标机器的硬件和(或)操作系统对寄存器的使用约束，该问题还会进一步复杂，详见11.5节。

11.1.5 计算顺序选择

- 计算执行的顺序同样会影响目标代码的效率。某些计算顺序比其它顺序需要较少的寄存器来保存中间结果，因而其目标代码的效率也要高。
- 选择最佳计算顺序也是一个NP完全问题。为简单起见，只讨论如何按给定的三地址码的顺序生成目标代码。

11.4 窥孔优化

- 窥孔优化是一种简单有效的局部优化方法，它通过检查目标指令中称为窥孔的短序列，用更小更短的指令序列进行等价代替，以此来提高目标代码的质量。
- 窥孔是放在目标程序上的一个移动的小窗口。孔中的代码不需要是连续的。
- 该技术的特点是每次优化后的结果可能又为进一步的优化带来了机会。所以有时会对目标代码重复进行多遍优化。下面介绍几种典型的窥孔优化技术。

11.4 窥孔优化

- 11.4.1 冗余指令消除
- 如果遇到如下的指令序列：

(1) MOV R_0, a

(2) MOV a, R_0 (11.1)

则可以删除指令(2)。但是，如果(2)带有标号，通常是不能删除的。

11.4 窥孔优化

- 11.4.2 不可达代码消除
- 删除紧跟在无条件跳转指令后的无标号指令称为不可达代码删除。

11.4 窥孔优化

- 11.4.3 强度削弱
- 强度削弱是指在目标机器上用时间开销小的等价操作代替时间开销大的操作。例如用x*x实现x^2要比调用一个指数过程快很多。用移位操作实现乘以2或除以2的定点运算要更快一些。用乘法实现(近似)浮点除法也可能会更快一些。

11.4 窥孔优化

- 11.4.4 特殊机器指令的使用
- 为了提高效率，目标机器有时会使用一些硬件指令来实现某些特定的操作。例如，有一些机器具有自动加1和自动减1的寻址模式。这些模式的运用可以大大提高参数传递过程中压栈和出栈的代码质量，它们还可以用在形如i:=i+1的语句的代码中。

11.4 窥孔优化

- 11.4.5 其他处理
- 也可以利用其他一些途经进行窥孔优化。例如，通过删除那些不必要的连续转移；利用代数恒等性质删除形如x := x+0或x := x*1的代码的代数化简。

11.5 寄存器分配与指派

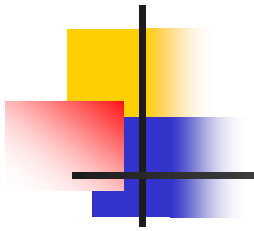
- 由于只涉及寄存器运算对象的指令要比那些涉及内存运算对象的指令短且快，因此有效地利用寄存器非常重要。
- 寄存器分配的任务是为程序的某一点选择应该驻留在寄存器中的一组变量
- 寄存器指派则负责挑出变量将要驻留的具体寄存器。

本章小结

1. 目标代码的生成需要尽力开发利用机器提供的资源，特别是根据开销选用恰当的指令和寄存器，以提高其执行效率；
2. 稀缺资源寄存器的有效利用涉及到后续引用问题，寄存器描述符用来记录每个寄存器当前的内容；地址描述符记录运行时存放变量当前值的一个或多个位置，用来确定对变量的存取方式；
3. 使用引用计数能够较好地实现寄存器的分配和指派；
4. 不同形式的三地址码对应不同的目标代码，且具有不同的执行代价；
5. 不可达和冗余指令删除、控制流优化、强度削弱、代数化简、特殊指令使用等都是有效的窥孔优化方法；

考试

- 时间：16周周四第三大节
- 地点：m102/m104
- 题型：概念、分析、设计
- 分数分布
  - 第一章：6分
  - 第二章：3分
  - 第三章：18分
  - 第四章：12分
  - 第五六章：36分
  - 第七章：16分
  - 第八九十章：9分



预祝各位前程似锦