编译原理

Compiler Construction Principles

朱 青

信息学院计算机系，

中国人民大学，

zqruc2012@aliyun.com

第10章：目标代码的生成

- 基本问题
- 目标机器模型
- 简单的代码生成器
- 寄存器分配
- DAG的目标代码
- 窥孔优化

基本问题

- 代码生成器的输入
  - 中间代码：如后缀式、四元式、语法树、dag
  - 符号表：包含数据对象运行地址的所有信息
- 代码生成器的输出：目标代码
  - 绝对机器语言：
    - 代码存储位置及数据对象在内存中确定
    - 二进制表示，可立即执行
  - 可重定位机器语言
    - 一组目标模块，需连接装入，方可执行
  - 汇编语言
    - 需经汇编程序汇编，转换成可执行的机器代码

基本问题

- 指令选择
  - 不同的机器支持的指令系统不同
  - 用的指令不同，执行效率不同

  例如 a:=a+1, 可用 INC a 实现，或

  ```
  LD    R0,a
  ADD   R0,#1
  ST    R0,a
  ```

- 寄存器分配
  - 根据变量使用频率分配
- 计算顺序的选择
  - 不同的计算顺序，可能要求存放中间结果的寄存器数量不同

目标机器模型

- 指令形式的地址模式
  - 绝对地址型： op Ri,M　(Ri) op (M)→Ri
  - 寄存器型： op Ri, Rj　(Ri) op (Rj)→Ri
  - 变址型： op Ri,c(Rj)　(Ri) op ((Rj)+c)→Ri
  - 间接型： op Ri, * Rj　(Ri) op ((Rj))→Ri
    op Ri, * M　(Ri) op ((M))→Ri
    op Ri,*c(Rj)　(Ri) op (((Rj)+c))4→Ri
- 指令集
  - LD Ri,B ：把B单元的内容取到寄存器Ri
  - ST Ri,B：把寄存器Ri的内容存到B单元
  - J X：无条件转向X单元
  - CMP A,B：把A单元和B单元的值进行比较，置机器内部特征寄存器CT的值为0，1，或2
  - J rop X：rop的值与CT值一致时，转向X

简单的代码生成器

- 功能：把中间代码变换为目标代码
  - 充分利用寄存器
    - 要引用的变量尽可能保存在寄存器中
    - 不再引用的变量所占的寄存器尽早释放
      - 释放时机
        - 到达基本块出口
        - 寄存器必须存放别的变量：从占用的寄存器中选择一个释放，选择标准是占用的寄存器变量不再被引用，或引用点最远（由待用信息表示）
- 待用信息：
  - 待用信息：基本块B中，变量A在i点的值j点引用，并且i→j的通路上没有A的其他定值和引用，则j为i处A的下一个引用点
  - 活跃变量信息：变量在基本块出口之后是否被引用

简单的代码生成器

- 寄存器描述和地址描述
  - 寄存器描述数组RVALUE：记录寄存器的分配状况
  - 变量地址描述数组AVALUE：记录各变量现行值的存放位置
- 代码生成算法
  - 假设只有A:=B op C的四元式序列

    A． 对每个四元式i: A:=B op C，依次执行下述步骤：
    - 1．以四元式i: A:=B op C为参数，调用过程getreg(i: A:=B op C)。从getreg返回时，得到一寄存器R，用它作存放A现行值的寄存器；
    - 2、利用AVALUE[B]和AVALUE[C]，确定出B和C现行值存放位置B`和C`，如果其现行值在寄存器中，则把寄存器取作B`和C`；
    - 3．如B`≠R，则生成目标代码
      - LD　　R ，　B`
      - op　　R ，　C`

    否则，生成目标代码 op　R，C`，如B`或C`为R，则删除AVALUE[B]或AVALUE[C]中的R

简单的代码生成器

- 4．令AVALUE[A]={R}，并令RVALUE[R]={A}，以表示变量A的现行值只在R中并且R中的值只代表A的现行值；
- 5，如B或C的现行值在基本块中不再被引用，它们也不是基本块出口之后的活跃变量（由四元式i上的附加信息知道），并且其现行值在某个寄存器Rk中，则删除RVALUE[Rk]中的B或C以及AVALUE[B]或AVALUE[C]中的Rk，使该寄存器不再为B或C所占用。

B．处理完基本块中所有四元式之后，对现行值在某寄存器R中的每个变量M，若它在出口之后使活跃的，则生成ST R，M，放到主存中。

- 例题：书P316

寄存器分配

- 指令的执行代价＝访问主存单元次数＋1
- 分配方法：循环中经常使用的变量使用固定寄存器
  - 简单指令的执行代价
    - op Ri，Rj ：执行代价为1
    - op Ri，M：执行代价为2
    - op Ri，*Rj：执行代价为2
    - op Ri，*M：执行代价为3

寄存器的分配

- 固定分配寄存器的代码生成
  - 变量x如获得固定分配的寄存器R，则生成代码直接对R存取，没有分配到固定寄存器的变量，由getreg在余下寄存器中请求分配
  - 固定分配到寄存器R的变量x，如果在循环入口之前是活跃的，则在循环入口的前置结点中，生成 LD R,x的代码
  - 固定分配到寄存器R的变量x，如果在循环出口之后是活跃的，则应在循环出口之后的后置结点中生成 ST R,X
  - 凡没有固定分配到寄存器的变量，按简单代码生成算法生成代码。

DAG的目标代码

- 由DAG图生成目标代码
  - 例：书P322
- 设DAG中结点重新排序的算法
  - 尽可能使一个结点的求值紧接着它的最左变量的求值之后
  - 启发式排序算法
  - (1) while存在未列入表的内部结点do
  - (2) begin选取一个未列入表的但其全部父结点均已列
  - 入表的结点n;
  - (3) 将n列入表中;
  - (4) while n的最左子结点m不是叶结点并且其所有
  - 父结点均已列入表中do
  - (5) begin将m列入表中;
  - (6) n: =m
  - (7) end
  - (8) end

例：赋值语句T_4:=A+B-(E-(C+D))

四元式序列G

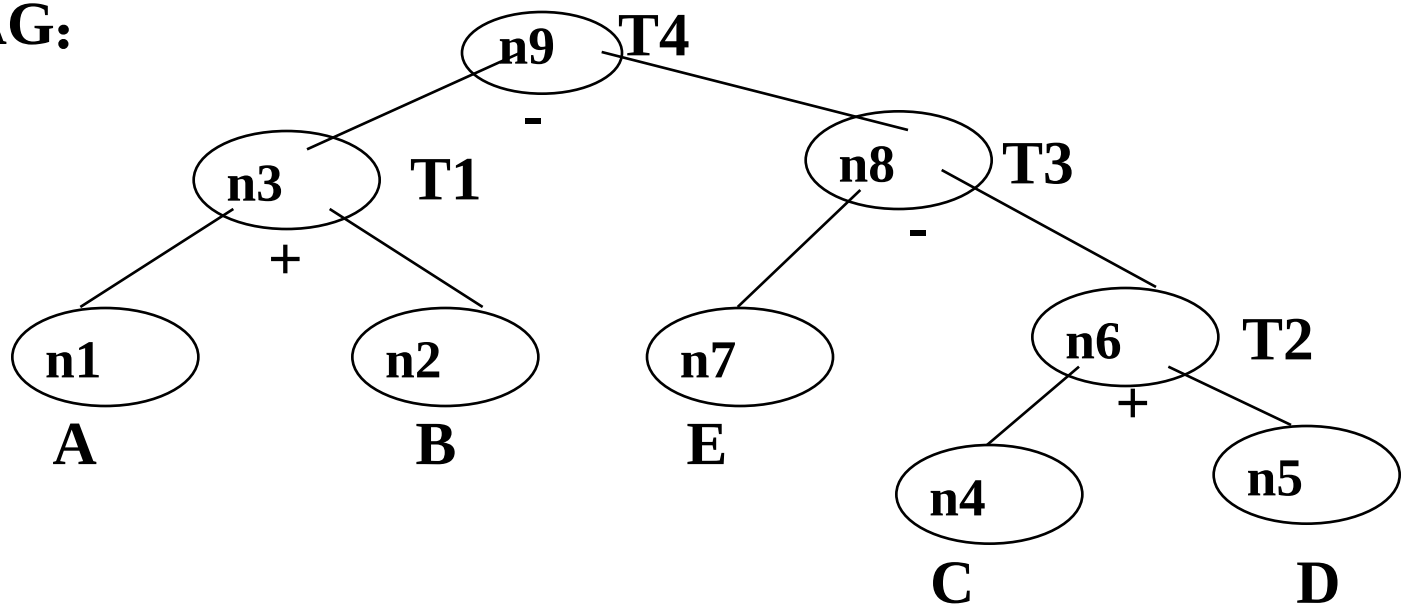
T_1：=A+B

T_2：=C+D

T_3：=E-T_2

T_4：=T_1-T_3

DAG：



T4:=A+B-(E-(C+D))

T1:= A+B	MOV A,R0
T2:=C+D	ADD B,R0
T3:=E-T2	MOV C,R1
T4:=T1-T3	ADD D,R1
	MOV R0,T1
	MOV E, R0
	SUB R1,R0
	MOV T1,R1
	SUB R0,R1
	MOV R1, T4

```
T2:=C+D          MOV  C,R0
T3:=E-T2         ADD  D,R0
T1:= A+B         MOV  E,R1
T4:=T1-T3        SUB  R0,R1
                 MOV  A,R0
                 ADD  B, R0
                 SUB  R1,R0
                 MOV  R0,T4
```

原因：T4的计算紧跟在T1之后

窥孔优化

- 在目标程序上设置一个可以移动的小窗口,称为窥孔.
  窥孔优化:主要考虑目标代码中很小范围内的指令序列的优化.
- 冗余存取
- 不可达代码
- 控制流优化
- 强度削弱
- 删除无用操作