编译原理lab5实验报告

姓名：张玄逸	学号：201220194
日期：2023.1.5	邮箱：1822771416@qq.com

实验内容

- 使用了201220107贺柄毓同学提供的框架代码，实现了复制传播，常量传播，公共表达式消除，死代码消除，得分98。

测试方法

代码可使用Makefile进行编译，得到可执行文件parser。

在文件根目录下输入 ./parser [文件名1] [文件名2]即可执行，如 ./parser Test/test1.ir out.ir，前一个作为输入文件，后一个作为输出文件。

对框架代码的理解

- 预处理
  - 地址

    中间代码中，变量前面可以加取地址符&，而在lab5的预处理中直接去掉了对&的存储。因为取地址实际上只会对函数中申请的数组，结构体使用，所以只需要将所有的DEC x [size]改为DEC x [size]； y := &x ; 即可。这样后续使用&x的地方都直接替换为y，免去了&的表示。
  - if 语句的跳转

    ```
    if x relop y goto label1
    goto label2
    label1  一类的语句
    改为if x !relop y goto label2
    ```

- 数据流分析
  - 各种数据流的分析过程非常类似，结合面向对象编程的特性可以构建一个宏观的框架，将前向or后向，传递函数，控制流约束函数等抽象成虚函数，最上层的分析只用实现迭代流程，具体的函数放在每个具体的数据流分析中去实现。
  - 首先将边界（exit或entry）初始化，再初始化其它所有块。
  - 然后开始循环迭代，一直持续除非IN/OUT没有更新，否则将更新的块的前驱/后继加入列表进行下一次迭代。
- 优化过程
  - 全局公共表达式消除
    - 常量传播
    - 可用表达式分析
    - 复制传播
  - 常量传播（第二次）

- 活跃变量分析（死代码消除，while循环反复迭代）

实验思路

- 常量传播
  - 此模块将某些确定是定值的变量修改为常量。
  - 在最开始，函数参数初始化为UNDEF。根据传递函数和控制流约束方程，我们能完成程序在赋值语句和块间跳转时的分析。传递函数的种类：
    - z := x：z赋为x的类型
    - z := x op y ： 9种情况分类讨论
    - CALL LOAD READ：z赋为UNDEF

    控制流约束与x op y规则类似，只不过当两者是不同常量时赋为NAC。
  - ```
    if (stmt->stmt_type == IR_ASSIGN_STMT)
        {
            IR_assign_stmt *assign_stmt = (IR_assign_stmt *)stmt;
            IR_var def = assign_stmt->rd;
            CPValue use_val = Fact_get_value_from_IR_val(fact, assign_stmt-
    >rs);
            Fact_update_value(fact, def, use_val);
        }
    ......
    ```
- 可用表达式分析
  - 此模块进行公共子表达式的替换。它分为三个步骤：常量折叠，公共表达式分析和复制传播。
  - 公共表达式分析中，首先进行简单表达式替换，也就是x+0, x*1之类的代数恒等式。接下来开始分析，对于每个新的表达式，都杀死受影响的旧表达式并将自己添加进集合中。
- 活跃变量分析
  - 此模块分析活跃变量并清除死代码。
  - 传递函数先执行kill后执行gen，特别注意此分析是后向分析，需要倒序遍历块内的语句。
  - ```
    // def:
    if (def != IR_VAR_NONE)
    { // 生成新def
        VCALL(*fact, delete, def);
    }

    // use:
    for (unsigned i = 0; i < use.use_cnt; i++)
    {
        IR_val use_val = use.use_vec[i];
        if (!use_val.is_const)
        {
            IR_var use = use_val.var;
            VCALL(*fact, insert, use);
        }
    }
    ```
- 复制传播
  - 此模块利用y := x，在后续对y的使用中尽量替换为x。

- 建立def_to_use/use_to_def链，将使用关系对应起来。对每个赋值语句，用新的def终结旧的对应，将自己添加进去。

总结与反思

- 要充分理解框架

  在写常量传播模块时，因为对传播所用的update和meet操作没有认识到位而导致混淆，造成误用的情况。实际上meet用于数据流控制函数中，作用是整合不同路线得到的数据类型，而update是直接更换数据类型，用于在块内部为ASSIGN，READ等语句的左值类型做更改。

  以及在复制传播模块中，一开始不理解def_to_use/use_to_def链就按自己的理解来写。后来查阅百科得知，def_to_use链用于将某个变量对应到之后可以使用它的值的其它变量，use_to_def链反之。之后按照正确的定义修改了代码。