编译原理 MiniDecaf 编译器实验
Step 5 实验报告

李祥泽
2018011331
lixiangz18@mails.tsinghua.edu.cn

实验内容

实现 main 函数中的多条语句

变更是很少且显而易见的.

实现变量的声明和赋值

使用一个 `HashMap<String, Symbol>` 实现符号表, 其中 `Symbol` 是我所定义的符号类. 为了在 IR 中同时包含 IR 代码和局部变量计数, 将 IR 也单独成类, 其中两个字段即为 IR 代码和计数.

在涉及变量的地方使用 `get` 方法查符号表. 由于 Java 很友好地规定用 `HashMap` 中不存在的 key 查表将返回 null, 可以通过查表结果是否为 null 判断符号的存在性, 而无需专门做 `containsKey` 查找.

栈帧参考实验指导书实现.

思考题

描述程序运行过程中函数栈帧的构成, 分成哪几个部分? 每个部分所用空间最少是多少?

如下所示, 左侧是该函数栈帧的开头, 右侧是低地址.

```
[ old ra | old fp | localVar0 | ... | localVarX | stackMachine >
```

依次表示: 保存的返回地址 (4 Bytes), 保存的栈帧基址指针 (4 Bytes), 声明的局部变量 (每个 4 Bytes), Stack Machine 所用的计算栈 (随时变化).

如果 MiniDecaf 也允许多次定义同名变量, 并规定新的定义会覆盖之前的同名定义, 请问在你的实现中, 需要对定义变量和查找变量的逻辑做怎样的修改?

在定义变量时仍然做符号名存在性检查. 如果不存在则照常. 如果存在, **不**报错, 而是进下一步计算初始化表达式 (在初始化表达式中仍可能调用这个变量, 所以暂时还不能覆盖); 计算完成后, 将结果存入该变量原先占有的内存空间 (相当于一次赋值).

查找变量的逻辑应该无需修改.

Honor Code

主要参考实验指导书实现.