编译原理实验四报告

胡俊豪 181240020@smail.nju.edu.cn

1 功能

本实验完成了实验手册中所描述的所有功能，并通过了所有测试样例。

1.1 测试说明

2 如何编译

本实验的编写测试环境，文件夹结构，Makefile 内容均与手册上一致，只需要在 Code 文件夹下使用命令 make，就可以得到一个名为 parser 的可执行文件。输入 "./parser testfile.cmm out.s" 即可完成对文件的解析，得到的 MIPS 代码保存在 out.s 文件中。

3 设计特色

本次实验采取的一个最大的理念就是极简主义。下面，我们分别从三个方面来探讨本实验中的极简主义。

3.1 指令选择

在上一个实验中，我们对生成的中间代码进行了非常强的规约，即，保证所有的操作都是对中间变量本身。比如，我们不允许 $t1 = *t2 + \#8$，只允许 $t2 = *t2, t3 = \#8, t1 = t2 + t3$。再例如，我们不允许 $t1 = \&v + \#8$，只允许 $t2 = \&v, t3 = \#8, t1 = t2 + t3$。这样的话，在生成最终的机器代码的时候，我们只需要两 "款" 指令。第一款是赋值指令，用于把所有带 $*\#\&$ 等这些符号的运算先算好并放在一个寄存器 t 中，在把 t 拿来作后续的运算。第二款指令就是只包含寄存器的任意运算（加减乘除等等）。除此之外的其他 "复合" 运算，比如包含立即数的寻址运算就不会再出现了，这极大地简化了代码生成的难度，当然也极大地增大了代码的运算开销。

3.2 寄存器选择

在该实验中，为了体现极简主义，我们只是用两到三个寄存器（比如 $t0-t2$），把所有需要用到的值都存在栈里，要用到的时候立刻 load，不用的时候立刻 store。

3.3 栈管理

由于我们只用到两三个特殊的寄存器，且每次用完都 store 回活动记录中，我们并不需要保存寄存器。除此之外，我们做的更绝一点，约定不使用函数的那四个默认参数传递寄存器，全部使用栈来传递参数，进一步简化我们的实现复杂度。

3.3.1 数组

数组的首地址被存在某一个临时变量中，所以首地址存放在栈上属于局部变量的位置，但是数组体被存放在栈中属于数组的部分，动态开辟。这样一来，数组首地址所在的变量可以像其他任何变量一样，通过相对于 $fp 的偏移量来定位，定位到数组首地址之后，才在栈上继续定位到某一个特定的数组元素。

3.3.2 函数参数

我们在进行函数调用的时候，在栈上存放好函数的所有参数（如之前所说，我们不使用寄存器 a0-a3）。

3.3.3 栈的总结

综上所述，在栈上，每一个活动记录中，我们只存储以下信息：返回地址，$fp，函数参数（防止递归调用的时候函数参数被冲掉），所有该函数内的临时变量，以及数组体。

3.4 代码优化

我们采用以上极简主义的代码生成方式，势必会带来极高的代码冗余度。为了消除这些冗余度，我们专门开辟一个篇章来实现代码优化，将代码生成和代码优化隔离开，已实现模块化。但由于这是最后一个实验，没有性能要求，正好也赶上期末，我们的代码优化模块可能需要等到以后有机会，再完成了。

3.5 代码结构

在转换代码的过程中，每次调用函数之前和之后都有一些相同的工作要做，比如栈指针的变化，在栈上存放一些变量等等，这些动作被封装成函数（比如 before_funcall 和 af-

ter_funcall)，每次函数调用的时候，都用这些函数把函数调用语句包围起来，以达到简洁明了的目的。

4 致谢

感谢刘春旭和张思拓两位同学，课余饭后的讨论，使得在实验过程中累积的疑惑与不解得到逐一解决。