编译原理实验 4 实验报告

--151220097 孙旭东

0.实验环境

本次的实验环境具体为：

系统：ubuntu 16.04　　GCC：version 5.4.0

Flex：version 2.6.0　　Bison：version 3.0.4

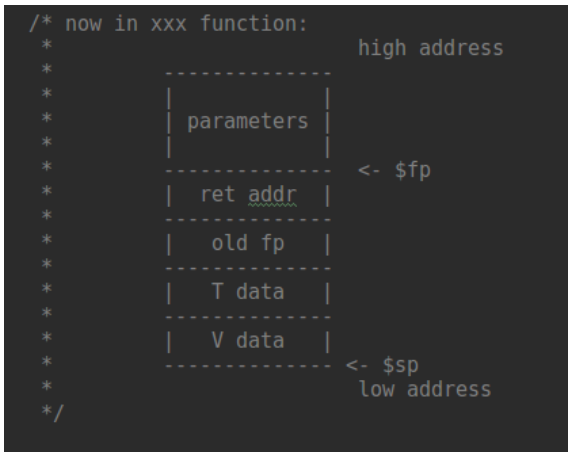
我的邮箱地址是 **248381185@qq.com**，如果检查中出现问题请助教联系我。

1. 实现功能

实验四的所有功能均已实现，至此，**编译原理大实验全部完成**！

本次实验的具体功能是在实验三的中间代码的基础上生成机器代码（spim 代码），并成功让机器代码运行，最终完成编译原理实验的所有内容。本次实验的主要流程是：逐条读取中间代码，并将其翻译成为对应的机器代码的内容，由与 spim 代码和中间代码十分相似，所以大致的翻译过程难度不大，**本次实验的亮点**，也就是**关键内容**，在于如何实现**寄存器的调度**和如何在过程调用时**管理栈帧**。我所采用的寄存器的调度算法是**朴素算法**，即所有变量总是存储在内存中，需要使用时载入到寄存器中，修改之后立即存储到内存中。这样做的好处是无需管理寄存器的分配，缺点是程序运行的效率较低，会有很多冗余的 load 和 save 操作。我选择将变量全部**存储在栈中**，而不是存储在静态数据区中，这么做可能会繁琐一点，但是是有巨大的好处的。

从之前的实验来看，我们在实验四中遇到的所有变量都不是全局变量，全部是局部变量，如果把他们存储在静态数据区中，当出现函数的递归调用时，很有可能被调用函数的变量会冲刷掉调用者的同名变量，导致不可预知的错误，所以我选择将他们存储在栈中。具体的**栈帧的组织方式**如下：



fp 指向的是参数的位置，从参数往下依次是返回地址，旧 fp，以及局部数据。被调用函数在执行时会从 T data 和 V data 中载入，存储数据，函数执行结束后会首先恢复旧的 fp，然后从 ret addr 中取出返回地址返回到调用者。而作为调用者的函数，在调用之前会把被调用函数的参数依次压入栈，然后跳转到被调用者执行。

2.编译方法

由于涉及的文件较多，所以编写了 makefile 来帮助进行程序编译。

make compile　　--编译生成 parser 程序

make clean　　--删除程序

具体的，如果有一个文件 test.cmm 希望进行编译，那么步骤为：

（1） **make compile**　　//删除旧程序，编译生成新程序

（2） **./compiler test.cmm** //对 test.cmm 进行分析

对 test.cmm 分析完毕之后生成的机器代码会保存为文件 test.cmm.s 中，也就是说**生成的代码文件和测试文件会在同一文件夹中**。

3.完成度与亮点

完成了要求的所有内容，亮点体现在**寄存器分配和栈帧管理**，具体可以参考上文中的内容。

书中的给出的测试用例全部通过。

4.运行演示

```
sunxudong@ubuntu:~/CLionProjects/compiler$ make compile
rm -f compiler
cd lexical_syntax && bison -d -v syntax.y
cd lexical_syntax && flex --header-file=lex.yy.h lexical.l
gcc main.c lexical_syntax/syntax.tab.c lexical_syntax/lex.yy.c lexical_syntax/Parsing
Node.c lexical_syntax/text.c \
semantic/semantic.c semantic/list.c semantic/check.c intercode/ICTable.c intercode/In
terCode.c intercode/optimization.c -g -O0 -std=c11 -Wall -Wextra -Wpedantic -Werror -
Wno-unused-function -Wno-sign-compare -D_GNU_SOURCE -lfl -ly -o compiler \
SPIM/spim.c
sunxudong@ubuntu:~/CLionProjects/compiler$ ./compiler test/lab4/1.cmm
Parsing test/lab4/1.cmm begin...
Parsing test/lab4/1.cmm over(with no error).
Semantic analysis test/lab4/1.cmm begin...
Semantic analysis test/lab4/1.cmm over(with no error).
InterCode generation of test/lab4/1.cmm begin...
Intermediate code generation of test/lab4/1.cmm over(with no error).
Machine Code generation of test/lab4/1.cmm begin...
Machine code generation of test/lab4/1.cmm over.

sunxudong@ubuntu:~/CLionProjects/compiler$
```

```
sunxudong@ubuntu:~/CLionProjects/compiler/test/lab4$ spim -file 1.cmm.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter an integer:7
1
1
2
3
5
8
13
sunxudong@ubuntu:~/CLionProjects/compiler/test/lab4$
```