编译原理课程Lab4

班级	队伍	组长	组员	联系邮箱
普通班	2人队	181870290 周心瑜	181860127 姚逸斐	181870290@smail.nju.edu.cn

实现功能

本次实验在词法分析、语法分析、语义分析和中间代码生成程序的基础上，将C--源代码翻译为MIPS32指令序列，并在SPIM Simulator上运行，相当于编写了一个可以实际运行的编译器。

设计思路

本次实验的设计主要在于MIPS32.h与MIPS32.c中。

指令选择

在实验三中，中间代码采用链表结构实现，故本次实验中将中间代码逐条翻译成目标代码。在MIPS32.c，`MIPS_translater`调用`__initialize__`初始化输出文件，`initialize_regs`初始化寄存器，接着逐条将中间代码对应到目标代码上。这过程中涉及寄存器分配，构造活动记录等，具体实现见下。

寄存器分配

本实验中寄存器分配采用朴素寄存器分配算法，$t0~$t7寄存器可供分配。

寄存器的数据结构是一个32个`Register`结构体组成的数组，`Register`包含名字，是否被使用的标志位和一个Var_类型的变量指针。Var_类型包括变量名称，偏移量和寄存器编号。

`assign_op2reg`负责为变量和常数分配寄存器，`int assign_reg2memeory`负责将寄存器的值写入内存。每翻译一条中间代码时，调用`get_free_reg`为所用到的变量寻找空闲的寄存器分配，之后再调用`free_regs`释放寄存器。

栈管理

`assign_stack`负责分配栈。`translate_FUNCTION_ENTRY`负责构造函数的活动记录，将sp减去栈帧的大小，将返回地址、fp旧值依次压栈。接着将参数压栈，将溢出的变量，数组和结构体分配到栈上。最后还原栈帧。

`translate_CALL`负责处理函数调用，将寄存器和参数存入栈。在函数调用结束后，依次将之前保存的内容从栈中恢复出来。`translate_RETURN`负责处理函数返回，将函数开头保存过的寄存器恢复出来，然后将栈恢复原样。

编译方法

实验编写环境为：

Ubuntu	GCC	Flex	Bison
16.04	5.4.0	2.6.0	3.0.4

使用makefile进行编译，在命令行输入`make`编译；输入`make test`对Test目录中的文件进行测试，也可以使用`./parser ../Test/input.cmm ../Test/output.s`对某一指定文件生成目标代码，可以使用SPIM Simulator来进行检查。

实验感悟

- 实验中遇到的问题与解决方法
  - 注意 `Operand x->u.assign.left` 指针因先前实验编写的判断机制，运行时可能为空。所以在本次实验中需要检查指针是否为空，防止内存访问错误。
  - 在编写程序，需要注意一些细节。如：给定的指令代码中的寄存器 $sp 的 '$' 曾经被遗漏。
  - 需要给中间代码临时变量分配寄存器。此时用到`assign_op2reg`建立连接。之前的思路错误为给临时变量选择空闲寄存器。导致之后使用变量时未能再次连接到寄存器，出现错误。
- 感谢https://github.com/massimodong/compilers-tests提供的测试数据