Compiler Lab Report

刘嘉欣

2023 年 12 月 28 日

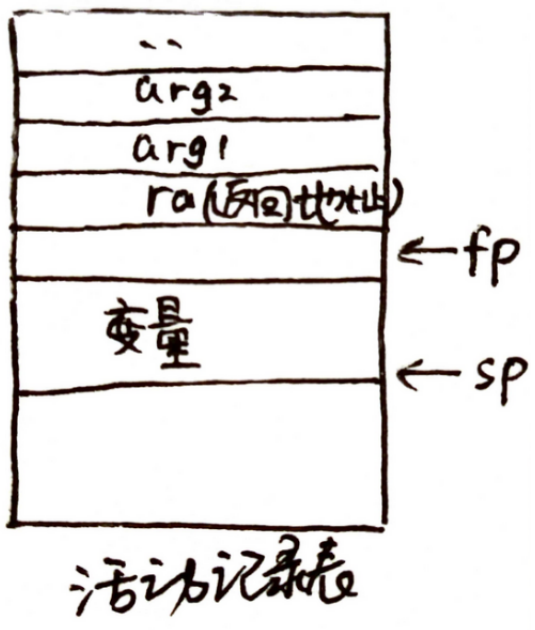
1　功能与实现

1.1　指令选择与寄存器分配

开始时无视与函数调用有关的 ARG、PARAM、RETURN 和 CALL 语句，专心处理其他的中间代码。为了简便，寄存器的分配直接采用最简单的方式：用时分配，用完 spill 到内存，这种方式虽然效率地下但也是可行的。

1.2　栈空间的分配

活动记录的布局方式如下图所示（很大程度上参考了《计算机系统基础》的课程内容）：



在函数开始时需遍历该函数的全部语句，一是要处理好传入函数的参数（这里全部放入栈中保存了）；二是统计函数用到所有的 Variable（包括变量和临时变量，还有数组），提前分配好栈空间，同时用一个 OffsetTable 记录每个 Operand 关于 fp 的偏移量，方便查找变量所在栈中的位置。

至此完成了所有内容.

2 编译方式

采用 Makefile 进行编译，直接 make 即可. 执行的命令如下：(其中，syntaxTree.c 和 syntaxTree.h 保留了实验 1 语法树的定义以及相关操作;semantic.c 和 semantic.h 为语义分析的内容, 其中符号定义和符号表的定义在 symbol.c 和 symbol.h 中，中间代码生成相关的 intercode.c 和 intercode.h 中，目标代码生成相关文件是 assemble.h 和 assemble.c)

1. flex -o lex.yy.c lexical.l 使用 Flex 编译 lexical.l 文件，生成 lex.yy.c
2. bison -o syntax.tab.c -d -v syntax.y 使用 Flex 编译 syntax.y，生成 syntax.tab.c 和 syntax.tab.h 两个文件，其中 lexical.l 引用了 syntax.tab.h
3. gcc -c syntax.tab.c -o syntax.tab.o 编译 syntax.tab.c，生成 syntax.tab.o
4. gcc -std=c99 -c -o syntaxTree.o syntaxTree.c 编译 syntaxTree.c，gcc -std=c99 -c -o main.o main.c , gcc -std=c99 -c -o symbol.o symbol.c 和 gcc -std=c99 -c -o semantic.o semantic.c 将所需的 syntaxTree.c、main.c, symbol.c 和 semantic.c 文件编译生成 syntaxTree.o、main.o、symbol.o 和 semantic.o. gcc -std=c99 -c -o intercode.o intercode.c **本次实验:** gcc -std=c99 -c -o assemble.o assemble.c
5. gcc -o ./parser assemble.o intercode.o syntax.tab.o syntaxTree.o main.o symbol.o semantic.o -lfl -ly 把 main.o 和 syntax.tab.o 和 syntaxTree.o 放到一起进行编译成最终的可执行文件 parser

测试方式：./parser [.cmm file] [output file] (&& cat [output file])