编译原理 Project4 实验报告

姓名:万子文

学号:151220102

邮箱:540304594@qq.com

如果测试过程中出现问题，希望助教可以通过邮箱联系我，避免因为gcc 版本或者其他原因产生问题。

一．实验完成情况

1. 在实验 3 的基础上实现了机器指令的
2. 采用朴素的寄存器分配方法，将实验 3 的中间代码转为机器代码打印到制定文件，并通过了所有必选测试样例以及自己产生的测试样例的测试。能够的代码条件与实验 3 相同
3. 实验代码在目录 spim 文件夹下，包括 spim.c,spim.h

二．实验说明

编译步骤(*在工程文件下的 lexical 子目录下)

1. flex lexical.l(产生词法分析代码)
2. bison –d syntax.y (产生语法分析代码)
3. gcc main.c syntax.tab.c parsertree.c ../semantic/symbolTable.c ../semantic/covert.c ../intercode/intercode.c ../intercode/translate.c ../spim/spim.c -lfl -ly -o parser
   将上述所有代码编译，链接，最终产生的目标文件为 parser
4. 执行: ./parser example.c out.s
   将编译 example.c 的代码，并将产生的机器代码存放到 out.s 中

三．实验细节

1. 寄存器的选择和分配

采用了最简单的朴素寄存器分配方法，将更计算完成的内容立即装载回内存,下图以 assign 为例子

```
    if (left->attr==IVARIABLE){

        fprintf(stream, "   li $t0, %d\n", right->ICons);
        fprintf(stream, "   sw $t0, %d($fp)\n", leftAddr);
    }
```

这是 Assign 左侧为普通变量，右侧为立即数时的翻译

2. 实指令选择问题比较简单，实验报告里就不做过多描述
3. 关于栈的管理

```
while (temp_ice->cur->kind==IARG){
    Operand op=temp_ice->cur->ARG.argument;
    int argAddr=GetVTAddrRelFP(op);
    fprintf(stream,"   lw $t0, %d($fp)\n",argAddr);
    fprintf(stream,"   sw $t0, %d($sp)\n",4*(num-i-1));
    temp_ice=temp_ice->next;
    i++;
}

CALLGenerator(temp_ice->cur,stream);
fprintf(stream,"   lw $ra, -4($fp)\n");
fprintf(stream,"   addi $sp, $sp, %d\n",4*num);
```

这是准备调用函数对应的机器指令
while 循环依次压入所有的参数，
然后执行 Call 指令

```
  fprintf(stream,"   subu $sp, $sp, 8\n");
  fprintf(stream,"   sw $ra, 4($sp)\n");
  fprintf(stream,"   sw $fp, 0($sp)\n");
  fprintf(stream,"   addi $fp, $sp, 8\n");
  fprintf(stream, "   subu $sp, $sp, %d\n", (VSize + TSize)*4);
```

这是函数开头调用的机器指令

```
/* now in xxx function:
 *                          high address
 *          --------------
 *          |            |
 *          | parameters |
 *          |            |
 *          --------------  <- $fp
 *          |  ret addr  |
 *          --------------
 *          |   old fp   |
 *          --------------
 *          |   T data   |
 *          --------------
 *          |   V data   |
 *          -------------- <- $sp
 *                          low address
```

栈中从上向下依次是 */

将中间代码 V 变量和 T 变量全部压入栈中。

四．测试样例试验结果

通过了所有的必选测试样例以及自己制作的样例