编译原理实验4：目标代码生成

实验报告

吴澄杰　　151220122　　wuchengjie@smail.nju.edu.cn　　2018年7月4日

一、实验进度

此次实验，基于之前实验中实现的语法树、符号表管理、中间代码生成等内容，我进一步实现了目标代码生成，可以将符合实验合法要求的 C 语言源代码翻译为可以被 SPIM 这一 MIPS32 指令模拟器接受并运行的正确汇编代码。我实现了寄存器的动态分配和函数栈帧的分配、管理、回收，可以进行函数的递归。我的程序通过了所有 2 个必做样例。

二、编译与实验

在 main.c 中设置宏变量 LAB 值为 4，然后在工程目录下，输入 make 命令，即可完成编译，生成实验四的可执行程序 parser。工程目录下的 example4 文件夹中有实验 4 的 2 个必做样例的源代码、必做样例的示例输出、和我的程序可输出的中间代码和汇编代码。若要测试程序在样例上的运行，只需将样例文件名和输出汇编代码结果的文件名作为参数给到 parser 运行即可。比如，./parser ./example4/example1 ./example4/example1_sol.s 就可以得到程序在第一个测试样例上给出的汇编代码，并写入文件 ./example4/example1_sol.s 中。

三、具体实现描述

有了中间代码，可以比较方便地生成目标汇编代码。大多数情况下，只要对中间代码逐条翻译即可。需要考虑的问题主要是汇编代码的格式，寄存器的分配、使用和栈内存空间的管理。下面将分点叙述这些主要实现问题。

1. 汇编代码合法布局

一个合法的可供 SPIM 模拟器执行的 MIPS32 代码包含数据段 .data，代码段 .text 和用 .globl main 标明的程序入口。在本实验中，read 和 write 函数是程序与控制台交互的接口，是每一份 C 语言源代码中不会定义但可使用的函数。因此，在翻译成汇编代码的时候，我将 read 和 write 函数的汇编代码直接拷贝到每一份汇编代码中，并且以单独定义的方法处理 read 和 write 函数的调用。这样就可以使得 read 和 write 函数总是可以被使用的。

需要注意的是，为了实现函数的递归，我将所有临时变量都储存在了函数的栈帧中。由于实验要求中提到不考虑全局变量，因此数据段中始终只会存有 read 和 write 函数需要使用到的两个全局字符串常量。

2. 寄存器的分配

在本实验中，我使用了动态的寄存器分配方法。我使用了 t0-t9 10个通用寄存器用以保存程序运行中需要存储的变量。每当需要使用寄存器时，首先会检查该变量是否已经被加载到寄存器中，若是则直接使用该寄存器，否则则寻找一个空闲寄存器加载该变量。若所有寄存器都已经被使用了，则选择一个最早被使用了的寄存器强制使用。

我定义了 Register 结构体来管理寄存器的使用信息。

```
struct Register
{
    bool free;
    bool value;
    struct Symbol_t *sym;
};
struct Register regs[MAX_REGS];
```

其中，free 记录该寄存器是否空闲，value 记录该寄存器存的是否是常数，sym 记录在该寄存器保存的是变量的值的时候指向该变量的指针。

平时，寄存器中的值并不会立即被拷贝回内存。只有当寄存器需要被新的需求使用，或者当切换基本块时，寄存器中变量的值才会统一被拷贝回内存。我使用 FUNCTION 和 LABEL 作为标志来划分基本块。基本块之间的转移可以通过顺序执行、GOTO 和条件跳转、函数调用和返回等方式。

在代码生成过程中，我会对寄存器中的变量值维护一个 dirty 脏位信息。若变量值被在寄存器中被改变、被赋值，则脏位为 true，反之为 false。在拷贝寄存器中的值回内存时，只有被修改的变量的值，即脏位为 true 的，才会被拷回。

3. 栈帧管理

为了能够实现函数的递归，将所有变量作为全局变量存储是不可行的，必须将临时变量恰当地存储在栈帧中。我用 $sp 寄存器保存栈帧的顶部，用 $fp 寄存器保存栈帧的底部。所有的临时变量都依靠 $fp 寄存器加上地址的偏移量进行访问。

在进行函数调用时，将所有参数逆序存在栈的顶部，然后保存好返回地址后使用 jal 命令进行函数调用。函数返回后，从栈中读出原返回地址并保存到相应寄存器中，然后恢复栈顶寄存器的位置。

在函数内部，首先要将原栈底寄存器 $fp 压栈，然后将 $fp 的值设为现 $sp 的值。在函数返回前，要恢复 $fp 的原值。这一点十分类似 i386 的标准。

常见的函数开头处的命令序列为：

```
addi $sp, $sp, -4
sw $fp, 0($sp)
move $fp, $sp
```

```
addi $sp, $sp, -68      % 开辟空间存储临时变量
```

函数结尾处的命令序列为：

```
move $sp, $fp
lw $fp, 0($sp)
addi $sp, $sp, 4
jr $ra
```

函数返回值统一在 $v0 寄存器中保存和获得。$ra 存储的是函数的返回地址。

四、实验总结

此次实验我实现了一个 C 语言代码的 MIPS32 汇编代码生成软件，可以在之前实验的基础之上，依据符号表、抽象语法树、中间代码生成相应的汇编代码。代码生成过程使用了动态寄存器分配，这可以在相当大的程度上增加对寄存器的使用效率，减少生成代码的长度。我还对函数栈帧进行管理，使得可以正确处理包含递归的程序。