实验报告（四）

- 曹洋笛 161220004 2904428882@qq.com

一、实现功能

- 将实验三中得到的中间代码经过指令选择、寄存器选择以及栈管理之后，转换为MIPS32汇编代码

二、亮点/实现

1. 良好的代码结构：

   封装了多个实用函数，包括寻找可用寄存器的allocate函数、把变量从内存加载到寄存器的ensure函数、把变量从寄存器溢出到内存的spillFromReg函数等等。

2. 使用帧指针 $fp (ebp)：

   每个函数在栈上占据一块空间（活动记录），使用 $fp 保存这个活动记录的底部，如图：

函数调用时的栈空间	指针
Argument 5	
Argument 6	
ebp Address	= $fp
Local Var 1	
…	
Local Var n	= $sp

   这样就可以使用 $fp 来描述变量的位置，比如上表中的 Local Var 1 的位置就是 “**-4($fp)**”。

3. 内存变量对应表：

   该表的每个元素的结构请见下一节数据结构介绍的 “struct MemElem”。

   它记录了中间代码中出现的每个变量和临时变量所对应的在栈中应存储的位置（记录为属性值 offsetFP，即相对于帧指针 ebp($fp) 的偏移量），在每个变量第一次出现时，栈都为其预留一个空位（即 “addi $sp, $sp, -4”），并把此时的偏移量（$sp - $fp）保存到 offsetFP，这个位置不会改变（一个萝卜一个坑），这样，在寄存器溢出时就能根据这个属性直接找到该寄存器内的变量应该保存到的位置。

4. 数组在栈上存储：

当使用 DEC 为数组 array 声明一个长为 size 的空间时，在栈上为其预留出长 size 的空间来保存数组的值，假设当前栈指针 esp、帧指针 ebp，且 esp - ebp = -x，则声明数组空间就是令当前栈指针 esp = esp - size，数组地址范围是 [ebp - x - size, ebp - x) 的区间，对于内存变量对应表中的元素 array，它的属性 offsetFP 的值就是数组首地址相对于帧指针 $fp 的偏移。

在加载数组array到寄存器时（即中间代码的 **&array**），直接加载首地址而不是数组元素，例如加载 &array 到寄存器 $t0，读出 array 的属性值 offsetFP，则代码为：**addi $t0, $fp, offsetFP**，这样寄存器里就是数组首地址，可以直接通过加一个数来得到数组某一个下标的地址。

在从数组的某个元素位置 addr 取值时（即中间代码的 ***addr**），使用 “sw” 和 “lw”，例如当前寄存器 $t0 中是变量 m，寄存器 $t1 中是数组的某个元素的地址的临时变量 t9，则对于中间代码 “m = *t9”，应该翻译成：**lw $t0, 0($t1)**，而对于中间代码 “*t9 = m”，应该翻译成：**sw $t0, 0($t1)**。

三、数据结构

- Reg：表示一个 $t0 ~ $t7 的寄存器

```
struct Reg {
  RegName name; // 编号，例如 $t0编号8, $t1编号9, ...
  char* str; // 编号对应的string，例如 "$t0", "$t1", ...
  bool isEmpty; // 是否为空
  bool isConst; // 是否是常数
  union {
    char* var; // 寄存器里是变量或临时变量或数组
    int val; // 寄存器里是常数
  };
  int useBit; // 使用位，基本块内下一次使用的位置，用于判定清空哪个寄存器
};
```

- MemElem：内存变量对应表（数组）的元素

```
struct MemElem {
  bool isNull;
  char* name; // 变量名或临时变量名或数组名
  int offsetFP; // 该变量地址或数组首地址相对于帧指针$fp的偏移
  bool isArray; // 是否是数组
};
```

四、编译

使用助教的Makefile可以成功编译。

运行时，可以选择：

- `./parser xxx.cmm`：翻译成MIPS代码并打印在控制台

- `./parser xxx.cmm out.s`： 翻译成MIPS代码并保存在.s 文件中