编译原理实验 4
机器代码生成

黄真川（101220042）　　元玉慧（101220151）

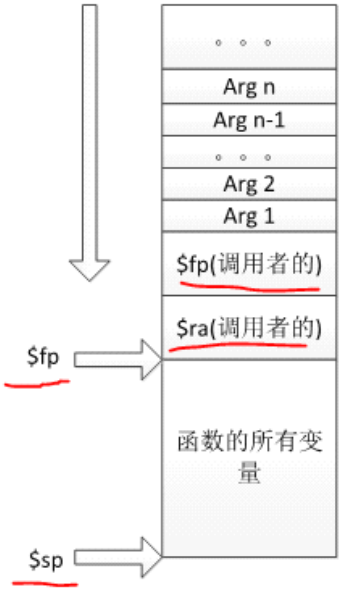
一　实验完成情况

由于时间的原因，实验设计中采用了最简单的寄存器分配策略同时选用了$t0\ $t1\ $t2 这几个寄存器，每次都是用寄存器 load 数据，然后把寄存器中的数据存到内存中。同时实现了实验的要求。

对实验 3 中的所有不包含结构体、多维数组的所有测试样例以及实验 4 要求中的 2 个测试进行了测试，都是正确的。

二　实验关键部分

-1- 活动记录设计以及内存分配



我们的活动记录方式是采用了栈帧$fp 指针保存函数中的基地址，类似于 x86 中的%ebp 寄存器，采用$ra 保存函数的返回地址。

函数调用的时候，要传给函数的实参存放在$fp 上面，被调用函数 param 变量以及函数内部的变量是存放在$fp 下面，访问的时候相对地址要存储在我们设计的一个结构体内。

```
struct MIPSnode{
    char var[20];
    int addr;
    MIPSnode *next;
};
```

其中 var 用来记录变量的名字，addr 用来记录其相对于自身所在函数的$fp 的偏移量。
我们的内存分配时在遍历中间代码的时候遇到 DEF_FUNCTION 的中间代码后，直接对函数内的所有变量分配空间，并且记录每个变量的相对偏移量到结构体中。

-2-MIPS 指令翻译设计

在上次实验中我们使用了线形 IR，最简单的指令选择方式是逐条将中间代码进行对应到目标代码上。所以本次实习的中间代码直接借鉴了指导攻略中的翻译方案。需要注意的地方就是函数调用时 fp，sp 寄存器的保存，以及数组翻译时对地址的处理。
如对数组元素进行赋值，*x= y 的翻译模式如下：
先将右侧变量 y 存入$t2,将左侧地址 x 存入$t0,然后通过 sw $t2, 0($t0)指令即可修改数组元素值。

```
else if(code->u.assign.left->kind == ADDRESS){
    load("$t2", code->u.assign.right);
    load("$t0", code->u.assign.left);
    fprintf(fout, "   sw $t2, 0($t0)\n");
}
```

函数调用的翻译模式：
先将$fp,$ra 压栈，jal 执行函数调用，恢复$fp, $ra,然后修改$sp 以便传入参数，保存结果$v0

```
case _CALL:
    stack_push("$fp");
    stack_push("$ra");
    fprintf(fout, "   jal %s\n", code->u.assign.right->u.func_name);//
    stack_pop("$ra");
    stack_pop("$fp");
    fprintf(fout, "   addi $sp, $sp, %d\n", arg_count * 4);
    store("$v0", code->u.assign.left);//******************
    arg_count = 0;
    break;
    //***************
```

函数返回的翻译模式：
将结果压入寄存器$t2，然后传入$v0，修改$SP，执行 jr 返回。

```
case _RETURN:
    load("$t2", code->u.uniop.op);
    fprintf(fout, "   move $v0, $t2\n");
    fprintf(fout, "   move $sp, $fp\n");
    fprintf(fout, "   jr $ra\n");
    break;
```

-3-对中间代码的修改

由于我们实验 3 中设计中间代码的时候，对于数组元素的访问操作类似于

```
a[1] = a[2] + a[3]
```

对应产生的中间代码类似于

```
*t11 = *t12 + *t13
```

然后在翻译成 MIPS 指令的时候就相当于同时处理 load 和 store 指令，还有指令相加，与窥孔优化技术相反，我们需要将这条中间代码翻译成多条机器代码，以避免处理时出现问题，然后我们分别在之前的 ir_translate.c 中的多处，添加了临时的 TEMP 变量来存储*t 类的变量来避免上述复杂的处理情况。

另外对中间代码中的数据结构里的名称进行了修改，比如参数名称存的不是源代码中的名称而是 IR 输出的名称，因为对于机器代码来讲真实名称毫无作用，这样在翻译成机器代码时可消除不一致性。

三　实验测试方法

源代码必须在含 glibc，gcc，flex，bison 的系统中编译运行。源码目录中 include 为头文件夹。注意若要运行汇编代码，还需要 SPIM 模拟器

本实验采用 Makefile 管理编译，在终端输入 make 命令，生成可执行文件 cmm_parser,输入./cmm_parser test.c test.s（前者可以是你想测试的满足实验约定文件，后者是 mips 汇编文件）即可运行。注意为简单起见 cmm_parser 一次只允许编译一个源文件。