**目标代码生成**

姓名：李敏婕

学号：201511210131

1. **实验目的**

（1）基本任务：

将得到的中间代码转换为MIPS32汇编代码（15分）并完成实验报告（5分）

MIPS32汇编代码任务：

1. 通过书上的例4.1和例4.2各4分，共8分

2. 7个testcase每个1分，共7分

（2）附加任务：

对中间代码进行优化后再转换为目标代码，需完成单独的实验报告，根据优化的难易程度和效果对实验总分加2-5分

（3）输入格式：

程序输入是一个包含C--源代码的文本文件；

程序需要能够接收一个输入文件名作为参数

（4）输出格式：

要求程序将运行结果输出到文件。对每个输入文件，程序应当输出相应的MIPS32汇编代码。

1. **实验环境**

编程环境为ubuntu上的gedit

1. **实验设计**

Test.l是词法分析文件

Syntax.y是语法分析文件

Symbol.h是包括符号表，中间代码四元式的结构体定义

Symbol.c是包括符号表建立和中间代码、目标代码输出的函数的文件

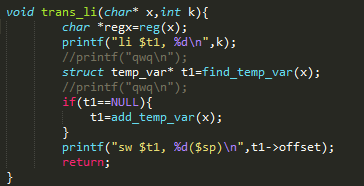
本次目标代码生成主要是在symbol.h和symbol.c文件中添加了对目标代码输出的功能，其中主要分为三个步骤：

1. 指令选择

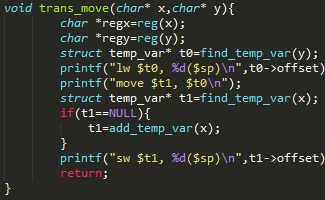
采取书中的中间代码与MIPS32指令对应的示例，在生成中间代码的同时，把它逐字逐句地翻译成目标代码

下面给出每种情况的目标代码翻译函数：

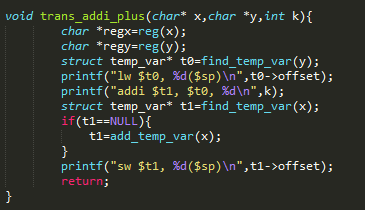
X:=#k



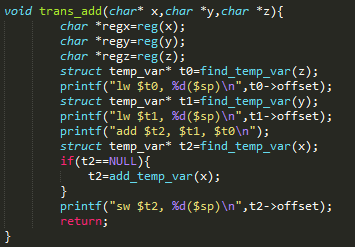
X:=y



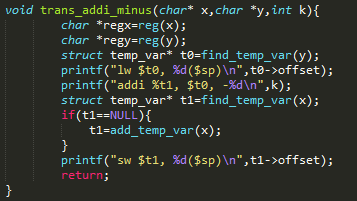
X:=y+#k;



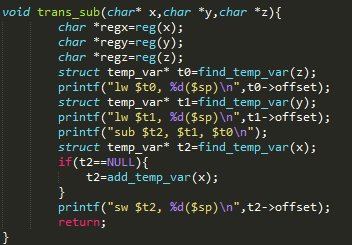
X:=y+z



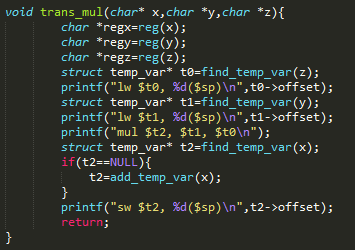
X:=y-#k



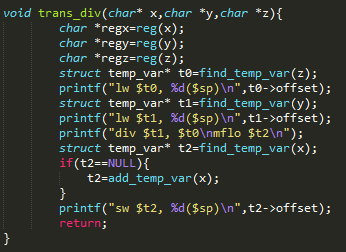
X:=y-z



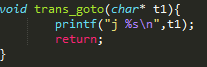
X:=y\*z



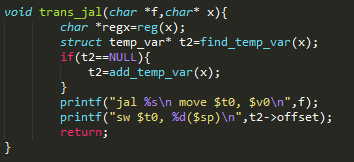
X:= y/z



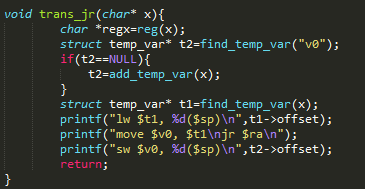
GOTO x



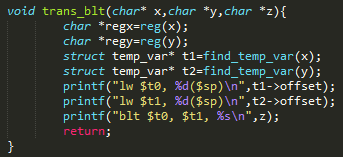
X := CALL f



RETURN x:



IF X relop Y GOTO Z

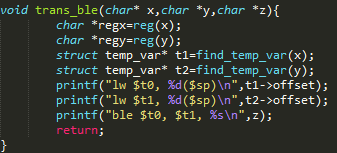


1. 寄存器分配

这里采取的是**朴素寄存器分配**，即除了存放从内存中取出来的变量值的寄存器（$t0-$t2），不使用其他寄存器，因为指令中最多需要用三个寄存器，即 add reg(x), reg(y), reg(z),所以$t0、$t1、$t2这三个临时寄存器就足以满足我们的需要

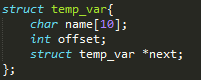
还需要注意，这时我们每次在输出目标代码的时候都需要先从内存中取出变量对应的值，并且在输出目标代码之后，把变量的值放回内存，这两个操作分别可以用lw，和sw指令实现

比如输出ble指令的目标代码的函数如下：

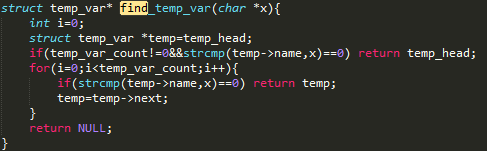


从图中可以看出此时为了实现从内存中取出对应的变量，我们需要通过中间代码中对应变量的名字来找到其在内存中的位置

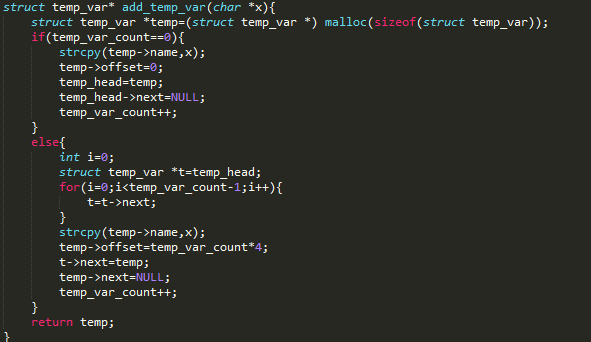
这就可以通过建立一个中间变量表（temp\_var）来实现，其中name代表中间变量在中间代码中的名字，offset表示中间变量在内存中的偏移量



我们每次从内存中取出变量时，都先在中间变量表中找它的位置（用find\_temp\_var函数）

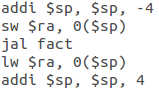


如果找不到说明它是一个没有出现过的中间变量，那么用add\_temp\_var函数新建一个名为它的中间变量

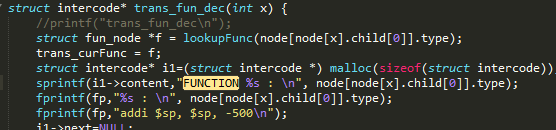


1. 栈的管理

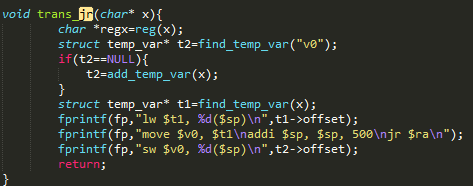
从2的思路来看，我们是通过$sp的偏移量来记录变量在内存中的位置的，但是这种记录方式在函数调用的时候可能会出现问题，这是因为每次我们在调用函数的时候都会生成如下汇编代码：



可以看出，$sp的位置不是固定的，在调用函数的时候它会向下-4，这时候就必须手动对函数中的变量位置进行更改了，我的思路是事先遍历一遍该函数所需要用到的所有变量，记下他们的总数n，在函数运行的时候先将栈往下偏移4\*n的距离



然后再函数返回的时候（jr $ra）之前，把栈恢复到原来的位置



1. **实验结果**

***（其中黑体字部分为编译程序输入和输出的流程）***

第一个样例：

输入文件为test.cmm

**命令行依次输入：**

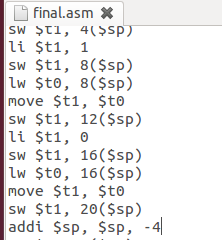
**flex test.l**

**bison bison -d syntax.y**

**gcc syntax.tab.c -lfl -ly -o par**

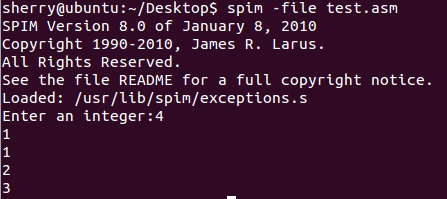
**./par test.cmm**

**可得输出文件final.asm，内容如下：**



**命令行输入spim -file final.asm可得**

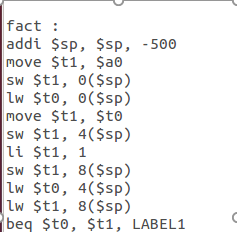
这时候我们输入一个整数4，就能输出斐波那契数列的前四个数



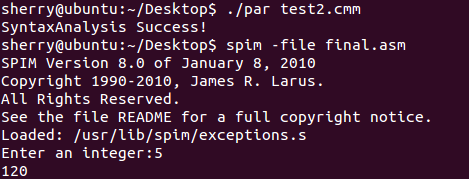
第二个样例：

同第一个样例一样，输入文件变为test2.cmm

依次输入命令行语句，可以得到输出的汇编文件final.asm



这时候我们输入一个整数5，就能输出5！=120



1. **实验总结**

这学期的编译实验非常考验我们书写工程代码的能力，虽然大部分内容都可以借鉴实验书，但是由于之前符号表的健壮性不够好，从而中间代码和目标代码的输出有问题。又因为我重写了之前的代码，这次的目标代码没能按时写完，感觉还是自己在写编译器的时候没有一个好的架构思路，导致出现后面用到的东西前面没有实现或者实现的有问题等等情况。总体来说我感觉这学期的编译原理实验虽然非常的难，但是对于提高我们书写工程性代码的能力是十分必要的。