目标代码生成评审材料

所有功能都已实现

1.输出目标代码

（1）对于语义正确的输入串，输出正确的目标代码（汇编代码）。

【测试用例】

calc.gpl

func.gpl

if.gpl

while.gpl

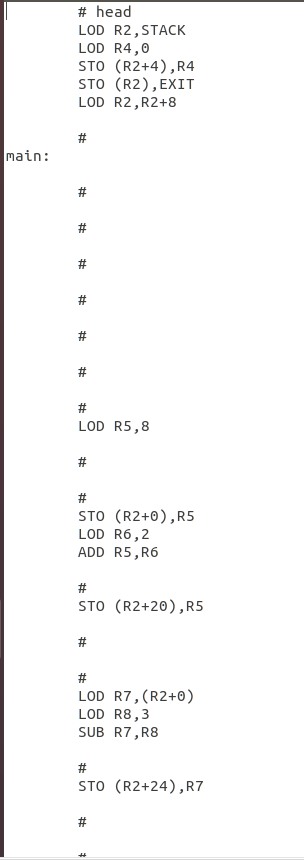
for.gpl

init.gpl

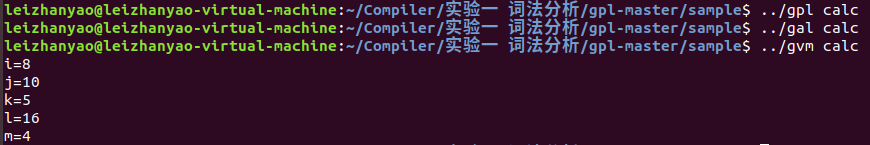
共六个

执行结果分别为

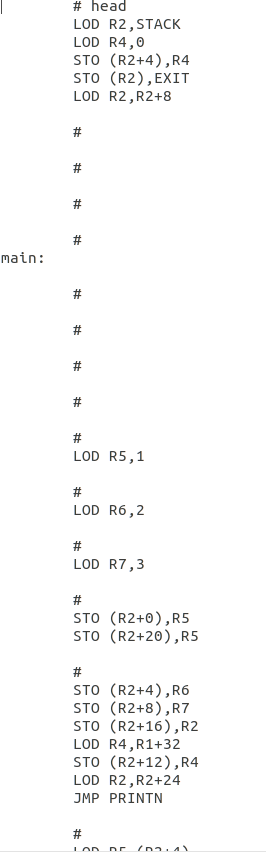
1）calc.gal为：



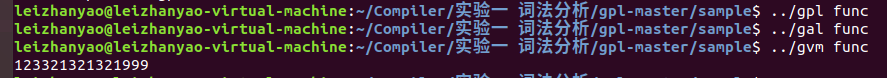
程序执行结果为



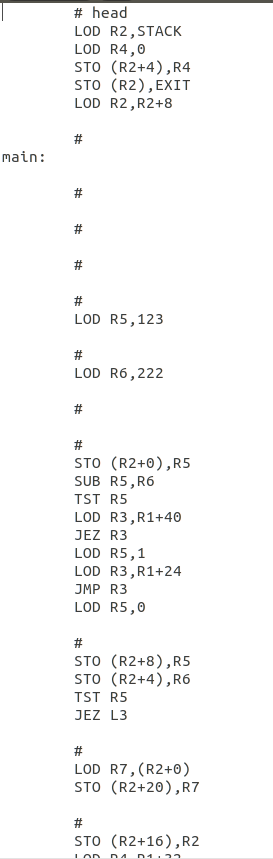
2）func.gal为：



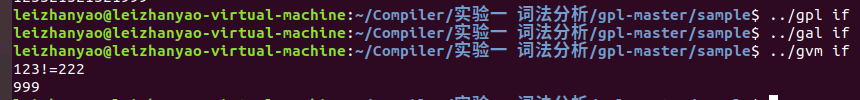
程序执行结果为



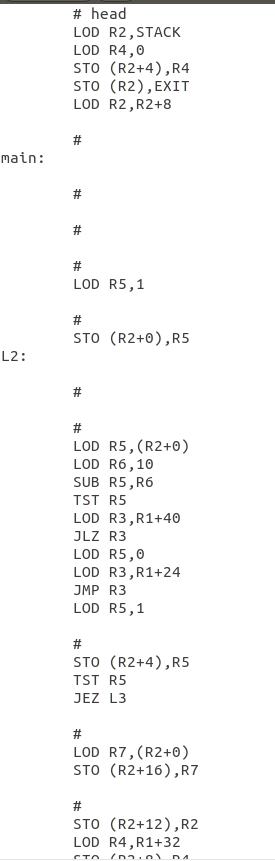
3）if.gal为



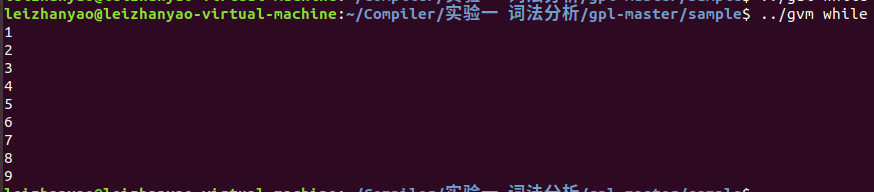
程序执行结果为



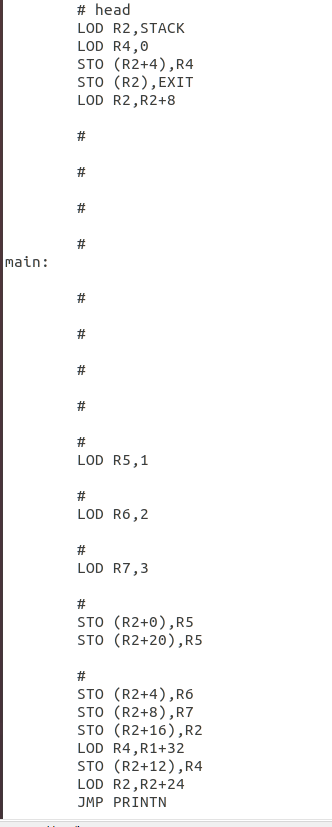
4）while.gal为



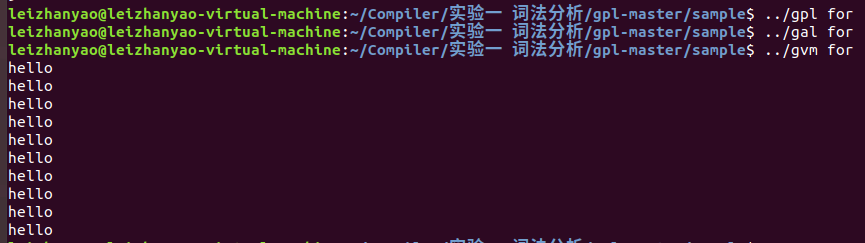
程序执行结果为



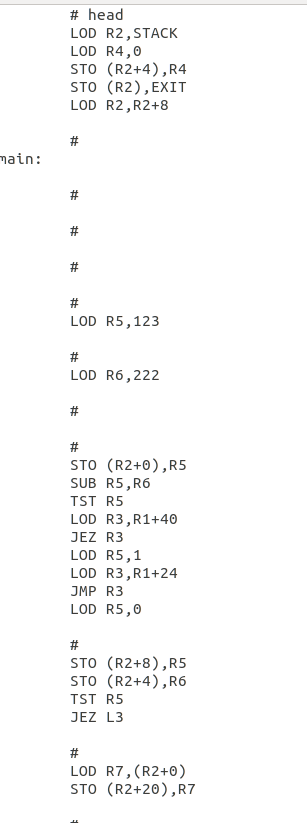
5）for.gal为



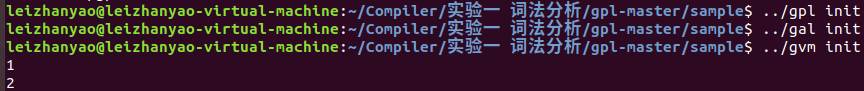
程序执行结果为



6）init.gal为



程序执行结果为



\*注：所有测试用例的完整版在评审材料的“测试用例”中有。

根据以上结果，证明目标代码生成正确，程序可以正常运行，实验成功。

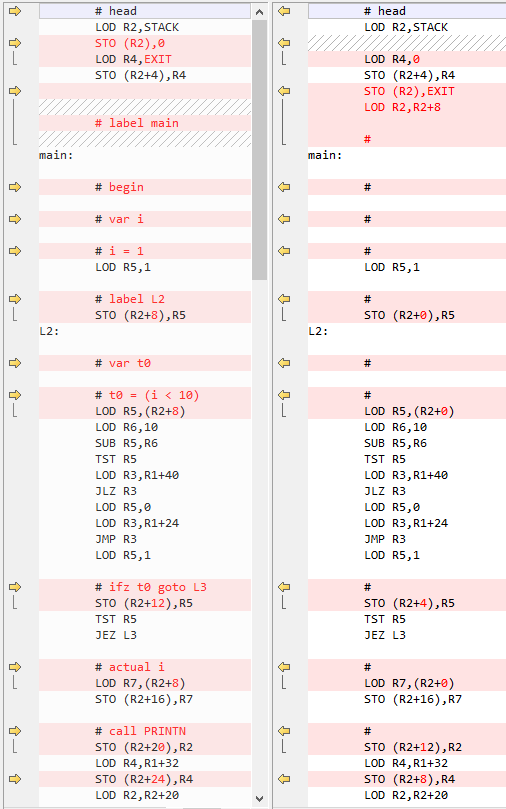
【代码简述】老师说这个已经实现了，只要前面实验三的中间代码生成正确，这里就一定正确。

（2）按照活动记录中各个字段的先后顺序按照课堂上的顺序排列。

【测试用例】

以while.gal为例

【执行结果】



对比前后生成结果，左边是原本的gal，右边是新的gal。可以看出，程序的执行结果不变，而只是LOD和STO指令中存储器操作数的一些偏移量发生了变化。这里实际上就是内存中他们的存放位置不同了。这就是活动记录顺序的改变。实验成功。

【代码简述】

主要是在1348行开始的cg\_code函数以及下面和它有关的一些列函数中改的。老师给的代码的顺序是“参数区、动态连接、返回地址、变量区”，我们需要改成“返回地址、动态连接、参数区、变量区”。只需要在上述函数里面找到相应内容的位置，调换他们的顺序即可。

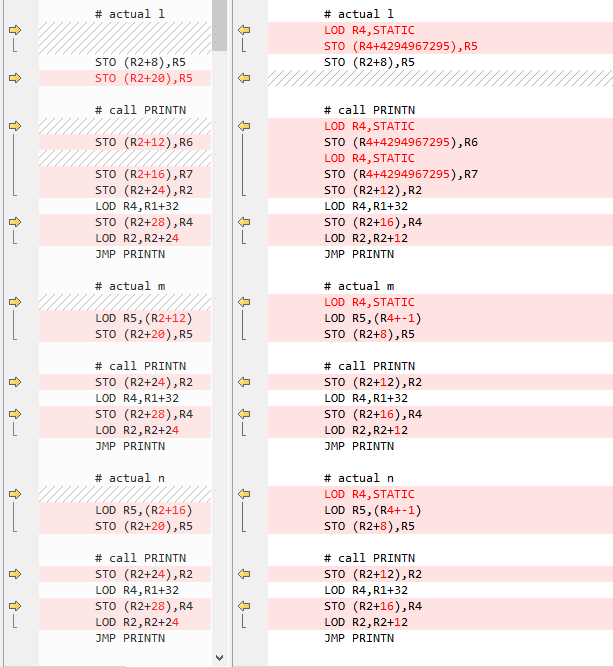
具体可见代码的1348~1628行。

（3）按照源代码中顺序传递实参的值

【测试用例】

以func.gal为例

【执行结果】



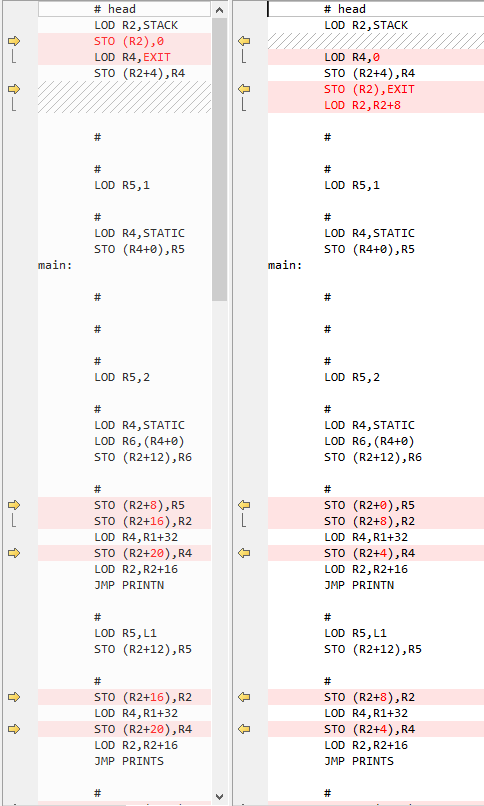
通过对比前后代码发现，参数传递顺序确实发生了改变，而程序执行结果一样，成功。

【代码简述】实际上，只要在实验三里面实现了改变传参顺序，这里的目标代码的传参顺序就一起改变了，因为目标代码就是直接按照中间代码翻译的。因为我实验三中，中间代码的顺序改好了，所以这里就直接是对的，没有什么需要修改代码的地方。

（4）支持变量（包括局部变量、全局变量）的初始化操作

【测试用例】以init.gal为例

【结果描述】



通过前后对比，发现最上面那里，是变量的初始化操作，gal代码发生了一些变化。结合程序执行和gal代码的逻辑，说明结果正确，实验成功。

【代码简述】这个也是需要一点代码修改的。具体也是1348~1628行中改的。

3.问题一：你的目标代码是如何访问参数、局部变量和全局变量的。

我们知道函数的活动记录中有“参数区”和“变量区”这两个区域，而且“变量区”实际上还分为局部变量区和全局变量区的。所以只需要在这里面找就可以了。具体来说，参数是通过bp+偏移量找到的，局部变量=局部变量区首地址+偏移量找到的，全局变量=全局变量区首地址+偏移量找到的。

4.问题二：是否提供了足够的测试用例（至少5个）证明你的程序实现了相关功能？

提供了足够的用例。

输出目标代码中提供了calc.gpl、func.gpl、if.gpl、while.gpl、for.gpl、init.gpl共六个用例

字段顺序、传参顺序、变量初始化又分别以while、func、init三个用例进行详解，同时还使用了以前代码生成的gal文件进行对比。

5.问题三：是否能够详细说明你的代码中最具特色或个性化的功能的实现方法？

有特色我觉得还是支持了for语句吧。因为我在实验三中加入了支持for语句的功能，就相当于重新按照lex的产生式的编写方法实现了for的tac，从而进一步实现for语句的支持。这个功能实现的重点就在于一定要了解lex特有部分的编写方法才能写得出来。这也加深了我对编译原理的理论知识的理解。