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：

```
        # head
        LOD R2,STACK
        LOD R4,0
        STO (R2+4),R4
        STO (R2),EXIT
        LOD R2,R2+8

        #
main:

        #

        #

        #

        #

        #

        #

        #
        LOD R5,8

        #

        #
        STO (R2+0),R5
        LOD R6,2
        ADD R5,R6

        #
        STO (R2+20),R5

        #

        #
        LOD R7,(R2+0)
        LOD R8,3
        SUB R7,R8

        #
        STO (R2+24),R7

        #

        #
```

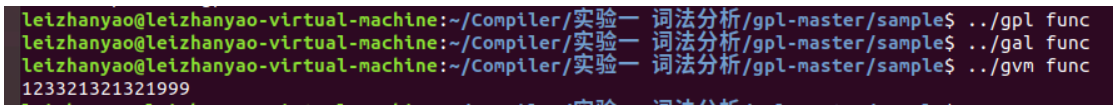
程序执行结果为

```
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gpl calc
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gal calc
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gvm calc
i=8
j=10
k=5
l=16
m=4
```

2）func.gal 为：

```
|       # head
        LOD R2,STACK
        LOD R4,0
        STO (R2+4),R4
        STO (R2),EXIT
        LOD R2,R2+8

        #

        #

        #

        #
main:

        #

        #

        #

        #

        #
        LOD R5,1

        #
        LOD R6,2

        #
        LOD R7,3

        #
        STO (R2+0),R5
        STO (R2+20),R5

        #
        STO (R2+4),R6
        STO (R2+8),R7
        STO (R2+16),R2
        LOD R4,R1+32
        STO (R2+12),R4
        LOD R2,R2+24
        JMP PRINTN

        #
        LOD R5,(R2+4)
```

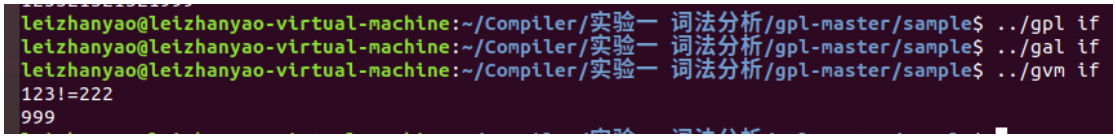
程序执行结果为


leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gpl func
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gal func
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gvm func
123321321321999

3）if.gal 为

```
        # head
        LOD R2,STACK
        LOD R4,0
        STO (R2+4),R4
        STO (R2),EXIT
        LOD R2,R2+8

        #
main:

        #

        #

        #

        #
        LOD R5,123

        #
        LOD R6,222

        #

        #
        STO (R2+0),R5
        SUB R5,R6
        TST R5
        LOD R3,R1+40
        JEZ R3
        LOD R5,1
        LOD R3,R1+24
        JMP R3
        LOD R5,0

        #
        STO (R2+8),R5
        STO (R2+4),R6
        TST R5
        JEZ L3

        #
        LOD R7,(R2+0)
        STO (R2+20),R7

        #
        STO (R2+16),R2
```

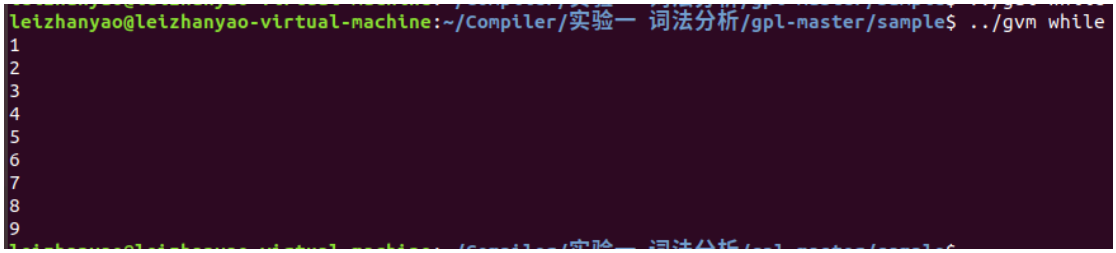
程序执行结果为


leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gpl if
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gal if
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gvm if
123!=222
999

4）while.gal 为

```
        # head
        LOD R2,STACK
        LOD R4,0
        STO (R2+4),R4
        STO (R2),EXIT
        LOD R2,R2+8

        #
main:

        #

        #

        #
        LOD R5,1

        #
        STO (R2+0),R5
L2:

        #

        #
        LOD R5,(R2+0)
        LOD R6,10
        SUB R5,R6
        TST R5
        LOD R3,R1+40
        JLZ R3
        LOD R5,0
        LOD R3,R1+24
        JMP R3
        LOD R5,1

        #
        STO (R2+4),R5
        TST R5
        JEZ L3

        #
        LOD R7,(R2+0)
        STO (R2+16),R7

        #
        STO (R2+12),R2
        LOD R4,R1+32
        STO (R2+8),R4
```

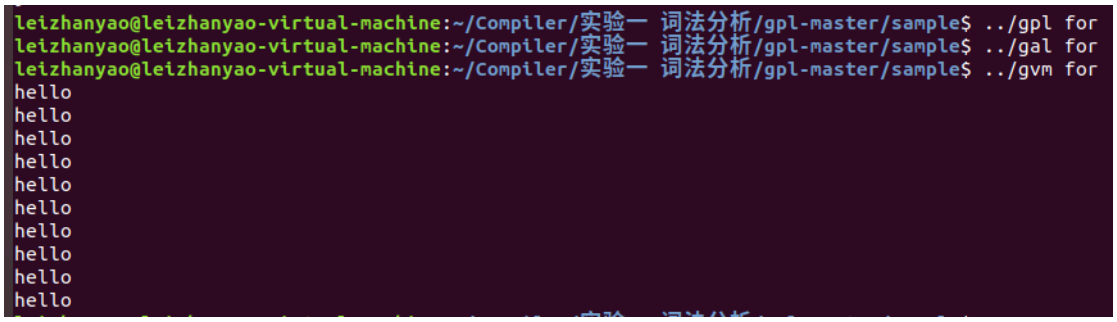
程序执行结果为


leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gvm while
1
2
3
4
5
6
7
8
9

5）for.gal 为

```
        # head
        LOD R2,STACK
        LOD R4,0
        STO (R2+4),R4
        STO (R2),EXIT
        LOD R2,R2+8

        #

        #

        #

        #
main:

        #

        #

        #

        #

        #
        LOD R5,1

        #
        LOD R6,2

        #
        LOD R7,3

        #
        STO (R2+0),R5
        STO (R2+20),R5

        #
        STO (R2+4),R6
        STO (R2+8),R7
        STO (R2+16),R2
        LOD R4,R1+32
        STO (R2+12),R4
        LOD R2,R2+24
        JMP PRINTN
```

程序执行结果为


leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gpl for
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gal for
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gvm for
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello

6）init.gal 为

```
        # head
        LOD R2,STACK
        LOD R4,0
        STO (R2+4),R4
        STO (R2),EXIT
        LOD R2,R2+8

        #
main:

        #

        #

        #

        #
        LOD R5,123

        #
        LOD R6,222

        #

        #
        STO (R2+0),R5
        SUB R5,R6
        TST R5
        LOD R3,R1+40
        JEZ R3
        LOD R5,1
        LOD R3,R1+24
        JMP R3
        LOD R5,0

        #
        STO (R2+8),R5
        STO (R2+4),R6
        TST R5
        JEZ L3

        #
        LOD R7,(R2+0)
        STO (R2+20),R7

        #
```

程序执行结果为

```
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gpl init
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gal init
leizhanyao@leizhanyao-virtual-machine:~/Compiler/实验一 词法分析/gpl-master/sample$ ../gvm init
1
2
```

*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。

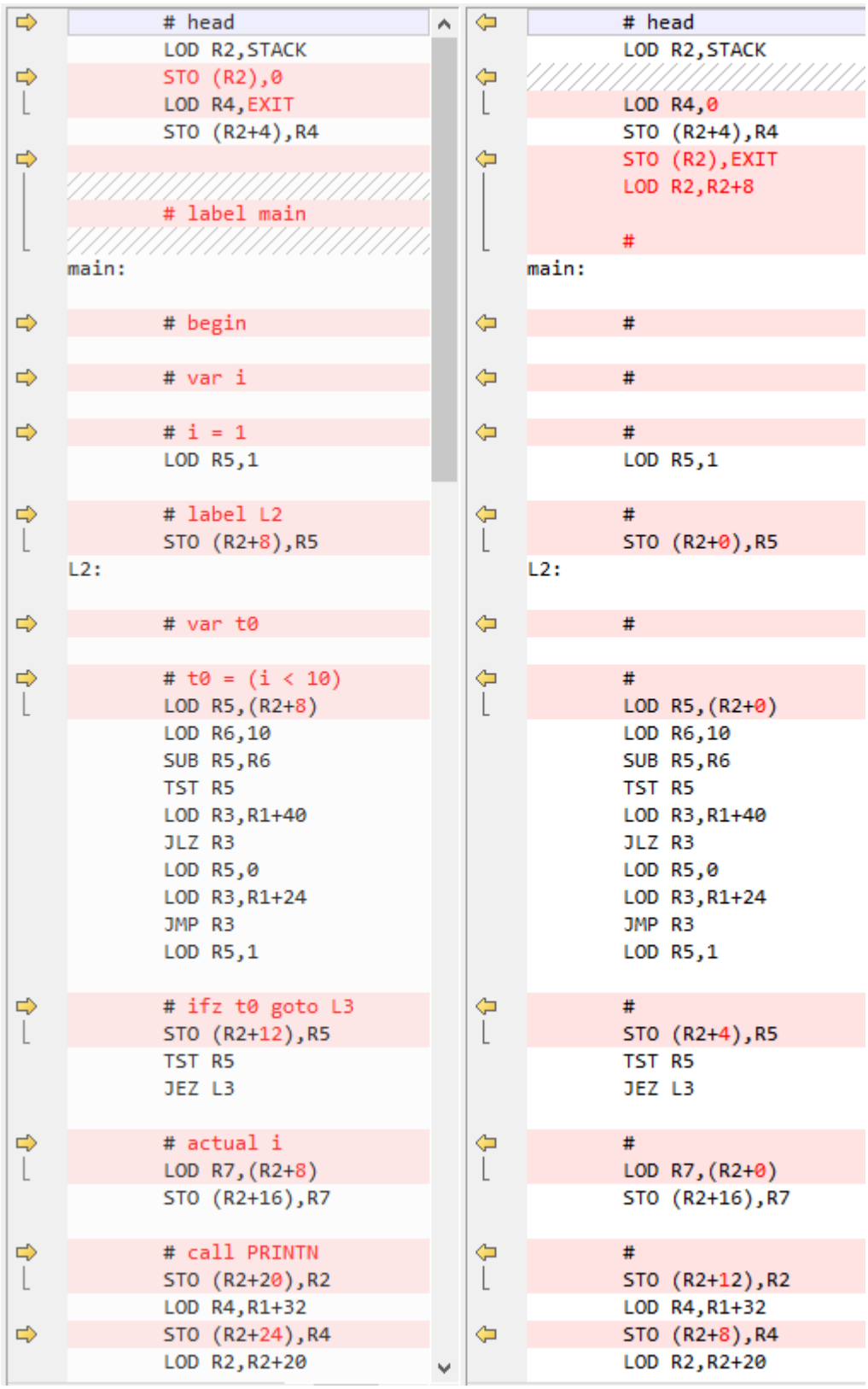
【代码简述】老师说这个已经实现了，只要前面实验三的中间代码生成正确，这里就一定正确。

（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】



```
        # head
        LOD R2,STACK
        STO (R2),0
        LOD R4,EXIT
        STO (R2+4),R4

        # label main
main:

        # begin

        # var i

        # i = 1
        LOD R5,1

        # label L2
        STO (R2+8),R5
L2:

        # var t0

        # t0 = (i < 10)
        LOD R5,(R2+8)
        LOD R6,10
        SUB R5,R6
        TST R5
        LOD R3,R1+40
        JLZ R3
        LOD R5,0
        LOD R3,R1+24
        JMP R3
        LOD R5,1

        # ifz t0 goto L3
        STO (R2+12),R5
        TST R5
        JEZ L3

        # actual i
        LOD R7,(R2+8)
        STO (R2+16),R7

        # call PRINTN
        STO (R2+20),R2
        LOD R4,R1+32
        STO (R2+24),R4
        LOD R2,R2+20
```

```
        # head
        LOD R2,STACK

        LOD R4,0
        STO (R2+4),R4
        STO (R2),EXIT
        LOD R2,R2+8

        #
main:

        #

        #

        #
        LOD R5,1

        #
        STO (R2+0),R5
L2:

        #

        #
        LOD R5,(R2+0)
        LOD R6,10
        SUB R5,R6
        TST R5
        LOD R3,R1+40
        JLZ R3
        LOD R5,0
        LOD R3,R1+24
        JMP R3
        LOD R5,1

        #
        STO (R2+4),R5
        TST R5
        JEZ L3

        #
        LOD R7,(R2+0)
        STO (R2+16),R7

        #
        STO (R2+12),R2
        LOD R4,R1+32
        STO (R2+8),R4
        LOD R2,R2+20
```

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】

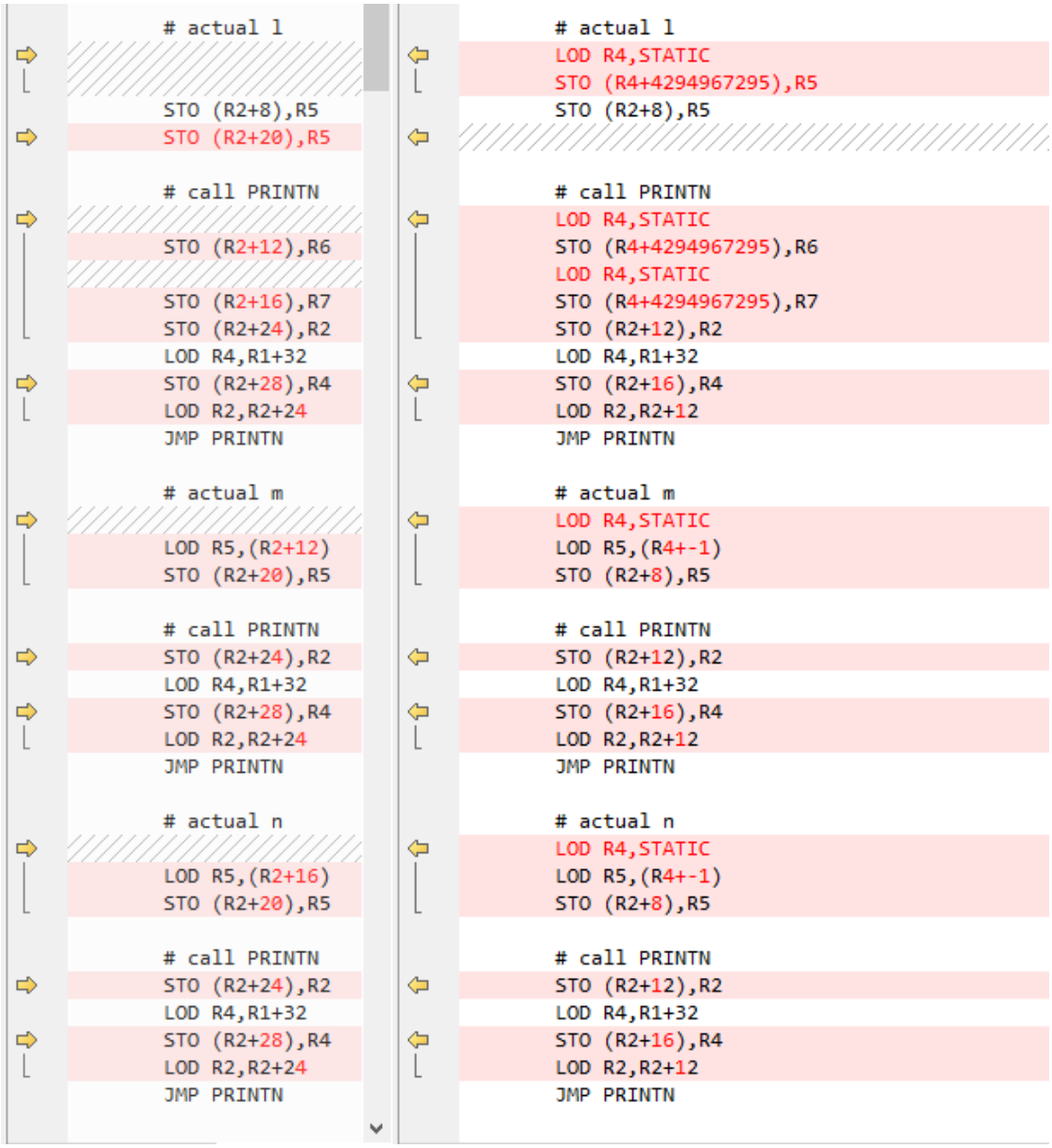
主要是在 1348 行开始的 cg_code 函数以及下面和它有关的一些列函数中改的。老师给的代码的顺序是“参数区、动态连接、返回地址、变量区”，我们需要改成“返回地址、动态连接、参数区、变量区”。只需要在上述函数里面找到相应内容的位置，调换他们的顺序即可。具体可见代码的 1348~1628 行。

（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】



```
# actual l
STO (R2+8),R5
STO (R2+20),R5

# call PRINTN
STO (R2+12),R6

STO (R2+16),R7
STO (R2+24),R2
LOD R4,R1+32
STO (R2+28),R4
LOD R2,R2+24
JMP PRINTN

# actual m
LOD R5,(R2+12)
STO (R2+20),R5

# call PRINTN
STO (R2+24),R2
LOD R4,R1+32
STO (R2+28),R4
LOD R2,R2+24
JMP PRINTN

# actual n
LOD R5,(R2+16)
STO (R2+20),R5

# call PRINTN
STO (R2+24),R2
LOD R4,R1+32
STO (R2+28),R4
LOD R2,R2+24
JMP PRINTN
```

```
# actual l
LOD R4,STATIC
STO (R4+4294967295),R5
STO (R2+8),R5

# call PRINTN
LOD R4,STATIC
STO (R4+4294967295),R6
LOD R4,STATIC
STO (R4+4294967295),R7
STO (R2+12),R2
LOD R4,R1+32
STO (R2+16),R4
LOD R2,R2+12
JMP PRINTN

# actual m
LOD R4,STATIC
LOD R5,(R4+-1)
STO (R2+8),R5

# call PRINTN
STO (R2+12),R2
LOD R4,R1+32
STO (R2+16),R4
LOD R2,R2+12
JMP PRINTN

# actual n
LOD R4,STATIC
LOD R5,(R4+-1)
STO (R2+8),R5

# call PRINTN
STO (R2+12),R2
LOD R4,R1+32
STO (R2+16),R4
LOD R2,R2+12
JMP PRINTN
```

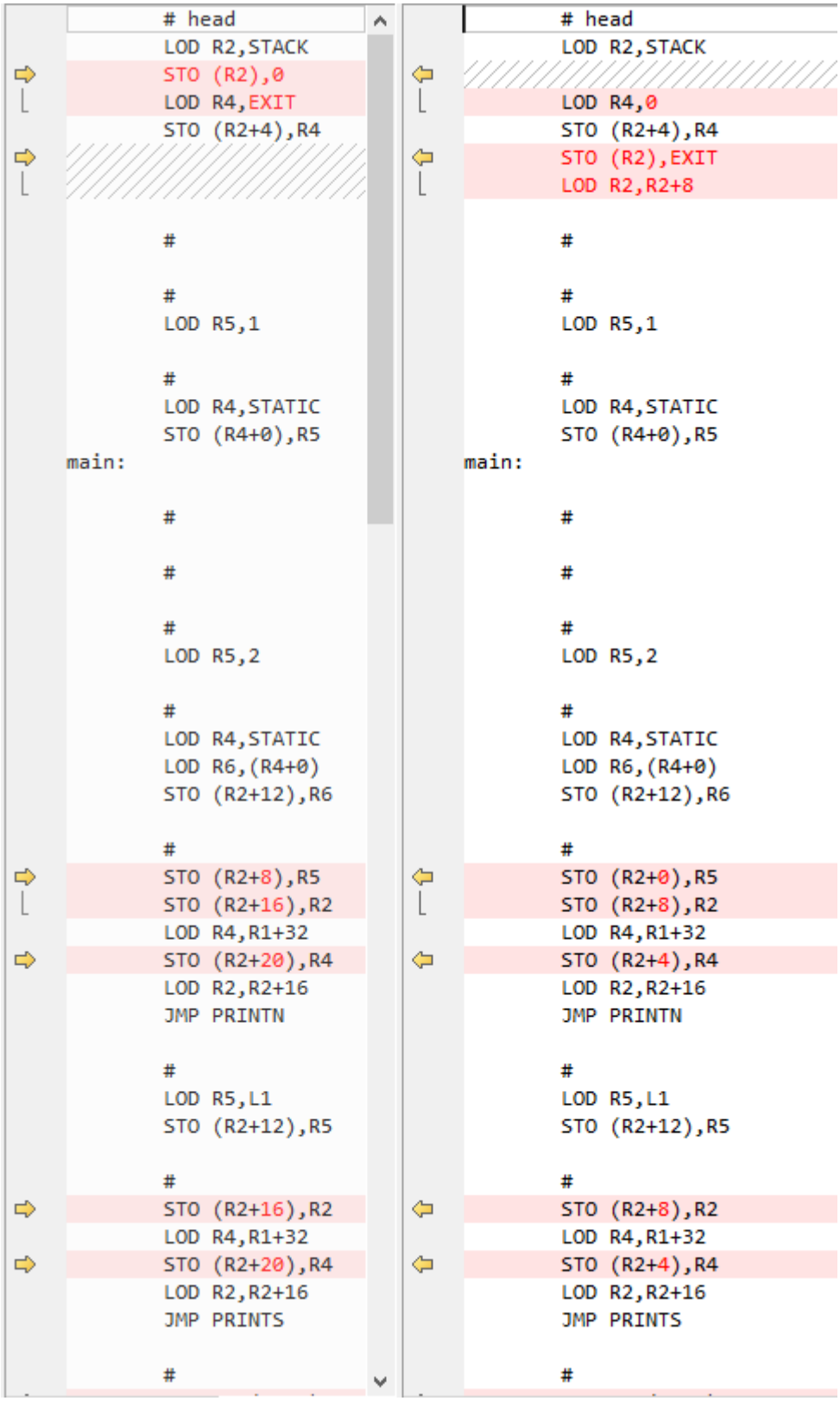
通过对比前后代码发现，参数传递顺序确实发生了改变，而程序执行结果一样，成功。

【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】



```
        # head                          |        # head
        LOD R2,STACK                    |        LOD R2,STACK
        STO (R2),0                      |
        LOD R4,EXIT                     |        LOD R4,0
        STO (R2+4),R4                   |        STO (R2+4),R4
                                        |        STO (R2),EXIT
                                        |        LOD R2,R2+8
        #                               |        #
        #                               |        #
        LOD R5,1                        |        LOD R5,1
        #                               |        #
        LOD R4,STATIC                   |        LOD R4,STATIC
        STO (R4+0),R5                   |        STO (R4+0),R5
main:                                   |main:
        #                               |        #
        #                               |        #
        #                               |        #
        LOD R5,2                        |        LOD R5,2
        #                               |        #
        LOD R4,STATIC                   |        LOD R4,STATIC
        LOD R6,(R4+0)                   |        LOD R6,(R4+0)
        STO (R2+12),R6                  |        STO (R2+12),R6
        #                               |        #
        STO (R2+8),R5                   |        STO (R2+0),R5
        STO (R2+16),R2                  |        STO (R2+8),R2
        LOD R4,R1+32                    |        LOD R4,R1+32
        STO (R2+20),R4                  |        STO (R2+4),R4
        LOD R2,R2+16                    |        LOD R2,R2+16
        JMP PRINTN                      |        JMP PRINTN
        #                               |        #
        LOD R5,L1                       |        LOD R5,L1
        STO (R2+12),R5                  |        STO (R2+12),R5
        #                               |        #
        STO (R2+16),R2                  |        STO (R2+8),R2
        LOD R4,R1+32                    |        LOD R4,R1+32
        STO (R2+20),R4                  |        STO (R2+4),R4
        LOD R2,R2+16                    |        LOD R2,R2+16
        JMP PRINTS                      |        JMP PRINTS
        #                               |        #
```

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。