记分牌算法仿真器的实现

一、项目背景

计算机系统结构课程实验选题之算法仿真器，通过实现记分牌算法仿真器，理解 CPU 的底层运作，进一步学习指令调度、数据冲突、乱序执行等概念的原理和方法，并为其他人学习记分牌算法提供仿真器。

二、记分牌算法原理

在记分牌中指令按顺序解码，每条指令的执行过程分为 4 段：流出、读操作数、执行和写结果：

（一）流出

Algorithm 1 Issue

Input: Operation to perform in the unit *Op*, Destination register number *D* Source-register numbers *S*1, *S*2, Functional Unit *FU*

```
1: function issue(Op, D, S1, S2)
2:     wait until (!Busy[FU] AND !Result[D]);
3:     Busy[FU] ← yes;
4:     Op[FU] ← op;
5:     Fi[FU] ← D;
6:     Fj[FU] ← S1;
7:     Fk[FU] ← S2;
8:     Qj[FU] ← Result[S1];
9:     Qk[FU] ← Result[S2];
10:    Rj[FU] ← Qj[FU] == 0;
11:    Rk[FU] ← Qk[FU] == 0;
12:    Result[D] ← FU;
13: end function
```

（二）读操作数

Algorithm 2 Read operands

Input: Functional Unit *FU*

```
1: function read_operands(FU)
2:     wait until (Rj[FU] AND Rk[FU]);
3:     Rj[FU] ← no;
4:     Rk[FU] ← no;
5:     Qj[FU] ← 0;
6:     Qk[FU] ← 0;
7: end function
```

（三）执行

Algorithm 3 Execute

Input: Functional Unit *FU*

```
1: function execute(FU)
2:     Execute whatever FU must do
3: end function
```

（四）写结果

Algorithm 4 Write Result

Input: Functional Unit *FU*

```
1: function write_back(FU)
2:     wait until (∀f((Fj[f]≠Fi[FU] OR Rj[f]=no) & (Fk[f]≠Fi[FU] OR
   Rk[f]=no)))
3:     ∀f(if Qj[F]=FU then Rj[f] ← yes);
4:     ∀f(if Qk[F]=FU then Rk[f] ← yes);
5:     Result[D] ← 0;
6:     Busy[FU] ← no;
7: end function
```

记分牌中记录的信息包括指令状态表、功能部件状态表以及结果寄存器状态表三个部分：

（一）指令状态表：记录正在执行的各条指令已经进入到了哪一段。

指令	指令状态表			
	流出	读操作数	执行	写结果

（二）功能部件状态表：记录各个功能部件的状态。每个功能部件有一项，每一项由以下 9 个字段组成:

Busy：忙标志，指出该功能部件正在使用。初值为 no；

Op：该功能部件正在执行或将要执行的操作；

Fi：目的寄存器编号；

Fj，Fk：源寄存器编号；

Qj，Qk：指出向源寄存器 Fj、Fk 写数据的功能部件

Rj、Rk：标志位，为 yes 表示 Fj、Fk 中的操作数就绪且还未被取走。

部件名称	功能部件状态表								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk

（三）结果寄存器状态表：每个寄存器在该表中有一项，用于指出哪个功能部件将把结果写入该寄存器。

	结果寄存器状态表									
	F0	F2	F4	F6	F8	F10	F12	F14	...	F30
部件名称										

记分牌算法基本思想：当某条指令被暂停时，不相关指令跨越流出与执行；记分牌全面负责和管理这些指令的流出和执行；记分牌算法实现按序流出、乱序执行、乱序完成，消除了许多等待，提高处理器的性能。

三、设计与实现

模拟器总共包括两个部分，输入和输出。输入包括浮点流水线中各部件的延迟（加法、乘法与除法部件）、指令序列的指令数、指令序列。输出包括每一周期的记分牌记录的三个表格。

（一）实体类

包括 Instruction 和 FU 两个实体类，分别对应指令和功能部件。

```
1. public class FU {
2.     //功能部件状态表的九个字段
3.     private boolean busy=false;
4.     private String op=null;
```

```
    private String Fi=null;
    private String Fj=null;
    private String Fk=null;
    private String Qj=null;
    private String Qk=null;
     private boolean Rj=false;
     private boolean Rk=false;
     //功能部件的名字
     private String name;
     //操作延迟周期
     private int time;
}
public class Instruction {
     //对输入进行解析成以下几部分
     private String inst;
     private String op;
     private String Fi;
     private String Fj;
     private String Fk;
     //指令状态表对应的字段
     private int issue=0;
     private int read=0;
     private int exe=0;
     private int writeback=0;
     //指令是否完成执行
     private boolean finish=false;
     //指令完成的周期
     private int exeCycle=0;
     private FU fu;
}
```

（二）数据转换处理

1. 根据指令序列的指令数新建指令数个指令对象，将其解析为包括操作码、目的寄存器、操作数几部分。

```
public Instruction(String input) {
        String[] parts = input.split("[,\\s()]");
        if (parts.length >= 4) {
            this.op = parts[0].trim();
            this.Fi = parts[1].trim();
            this.Fj = parts[2].trim();
            this.Fk = parts[3].trim();
        } else {
            System.out.println("Invalid input: " + input);
         }
    }
```

2. 新建功能部件对象（记分牌中共有五个功能部件），并对对应的功能部件赋值输入的各功能部件延迟，Integer 功能部件的延迟为 0。

```
//新建五个 FunctionalUnit 功能部件对象
        FU[] fus=new FU[5];
        fus[0]=new FU("Integer",0);
        fus[1]=new FU("Add",addTime);
        fus[2]=new FU("Mult1",multTime);
        fus[3]=new FU("Mult2",multTime);
        fus[4]=new FU("Div",divTime);
```

3. 初始化记分牌存储的三个表格，对每个表格的内容进行填充，对于指令状态表的初始值均为空（不可为 0）。

```
String[][] datas1 = new String[instructions.length][5];
            for (int i = 0; i < instructions.length; i++) {
                datas1[i][0]=instructions[i].getInst();
                for (int j = 1; j < 5; j++) {
                    datas1[i][j] = " "; // 使用空格填充
                }
            }
            String[] titles1 = {"Instruction", "Issue", "Oprand", "Execution", "Write"};
            String[][] datas2 = new String[5][10];
             datas2[0][0] = "Integer";
             datas2[1][0] = "Add";
             datas2[2][0] = "Mult1";
             datas2[3][0] = "Mult2";
             datas2[4][0] = "Divide";
             for (int i = 0; i < 5; i++) {
                 for (int j = 1; j < 10; j++) {
                     datas2[i][j] = " ";
                 }
             }
             String[] titles2 = {"FUname", "Busy", "Op", "Fi", "Fj", "Fk", "Qj", "Qk", "Rj", "Rk"};
             String[][] datas3 = new String[1][17];
             String[] titles3 = {"", "F0", "F2", "F4", "F6", "F8", "F10", "F12", "F14", "F16", "F18", "F20", "F22", "F24", "F26", "F28", "F30"};
             for (int i = 0; i < 1; i++) {
                 datas3[i][0] = "Name";
                 for (int j = 1; j < 17; j++) {
                     datas3[i][j] = " ";
                 }
             }
```

（三）记分牌算法具体实现

1. 判断指令是否能够流出，先读取指令的操作码，如果其所需要的

功能部件并不处于 Busy 状态并且不存在 WAW 冲突，流出该指令，并更新记分牌的指令状态表中该指令流出对应的周期以及记分牌中功能部件状态表中对应功能部件的值，如果不能够流出，则退出指令流出的判断。对于乘法指令，由于有 mult1 和 mult2 两个功能部件，所以先判断 mult1，如果 mult1 不满足条件再判断 mult2。

```
//判断每一条指令是否能够流出，若某条指令流出，从指令队列中删除
            for (i = 0; i < instructions.length; i++) {
                if ("LD".equals(instructions[i].getOp())) {
                    if (instructions[i].getIssue() == 0) {
                        ////判断指令所需功能部件是否Busy以及是否存在WAW冲突
                        if (!fus[0].getbusy() && result.get(instructions[i].getFi()) == null) {
                            issue(fus[0], instructions[i], result);
                            instructions[i].setIssue(cycle);
                            break;
                        } else {
                            break;
                        }
                    }
                } else if ("ADDD".equals(instructions[i].getOp()) || "SUBD".equals(instructions[i].getOp())) {
                    if (instructions[i].getIssue() == 0) {
                        if (!fus[1].getbusy() && result.get(instructions[i].getFi()) == null) {
                            issue(fus[1], instructions[i], result);
                            instructions[i].setIssue(cycle);
                            break;
                        } else {
                            break;
                        }
                    }
                } else if ("MULTD".equals(instructions[i].getOp())) {
                    if (instructions[i].getIssue() == 0) {
                        ////对于乘法指令，有mult1和mult2两个功能部件，故先判断mult1，若不满足条件则判断mult2
                        for(int j=2;j<4;j++){
                            if(!fus[j].getbusy()&&result.get(instructions[i].getFi())==null){
                                instructions[i].setFu(fus[j]);
                                issue(fus[j], instructions[i], result);
                                instructions[i].setIssue(cycle);
                                break;
                            }
```

```
                    }
                    break;
//                    if (!fus[2].getbusy() && result.get(instructions[i].getFi()) == null) {
//                        instructions[i].setFu(fus[2]);
//                        issue(fus[2], instructions[i], result);
//                        instructions[i].setIssue(cycle);
//                        break;
//                    } else if (!fus[3].getbusy() && result.get(instructions[i].getFi()) == null) {
//                        instructions[i].setFu(fus[3]);
//                        issue(fus[3], instructions[i], result);
//                        instructions[i].setIssue(cycle);
//                        break;
//                    }
                }
            } else if ("DIVD".equals(instructions[i].getOp())) {
                if (instructions[i].getIssue() == 0) {
                    if (!fus[4].getbusy() && result.get(instructions[i].getFi()) == null) {
                        issue(fus[4], instructions[i], result);
                        instructions[i].setIssue(cycle);
                        break;
                    }
                }
            }
        }
```

2. 判断已流出的指令是否能够读操作数。如果指令已经流出并且还没有读操作数，那么判断该指令源操作数是否可用（为避免 RAW 冲突），即 Rj、Rk 是否为真，如果可用，则该指令读操作数，更新记分牌中的数据。记录该指令的结束周期（针对不同的功能部件延迟）。

```
//判断每一条指令是否能够读操作数
        for (i = 0; i < instructions.length; i++){
            if ("LD".equals(instructions[i].getOp())){
                ////判断指令是否已经流出并且还未读操作数
                if (instructions[i].getIssue()!=cycle&&instructions[i].getIssue()!=0&&instructions[i].getRead() == 0) {
                    ////判断源操作数的可用性，避免 RAW 冲突
                    if (fus[0].getRj() && fus[0].getRk()) {
                        readOprand(fus[0]);
                        instructions[i].setRead(cycle);
                    }
                }
```

```
                }else if("ADDD".equals(instructions[i].getOp())||"SUBD".equals(instructions[i].getOp())){
                    if (instructions[i].getIssue()!=cycle&&instructions[i].getIssue()!=0&&instructions[i].getRead() == 0){
                        if (fus[1].getRj() && fus[1].getRk()) {
                            readOprand(fus[1]);
                            instructions[i].setRead(cycle);
                            ////对于需要不同执行周期的指令，保存需要执行的周期
                            instructions[i].setExeCycle(cycle + fus[1].getTime());
                        }
                    }
                } else if ("MULTD".equals(instructions[i].getOp())) {
                    if (instructions[i].getIssue()!=cycle&&instructions[i].getIssue()!=0&&instructions[i].getRead() == 0){
                        if (instructions[i].getFu().getRj() && instructions[i].getFu().getRk()) {
                            readOprand(instructions[i].getFu());
                            instructions[i].setRead(cycle);
                            instructions[i].setExeCycle(cycle + instructions[i].getFu().getTime());
                        }
                    }
                } else if ("DIVD".equals(instructions[i].getOp())) {
                    if (instructions[i].getIssue()!=cycle&&instructions[i].getIssue()!=0&&instructions[i].getRead() == 0) {
                        if (fus[4].getRj() && fus[4].getRk()) {
                            readOprand(fus[4]);
                            instructions[i].setRead(cycle);
                            instructions[i].setExeCycle(cycle + fus[4].getTime());
                        }
                    }
                }
            }
```

3. 对每一条已经读操作数但还未执行结束的指令，检测是否已经执行完毕（已经到达执行结束的周期），如果执行完毕则更新记分牌中指令状态表中执行的周期。

```
else if ("ADDD".equals(instructions[i].getOp())||"SUBD".equals(instructions[i].getOp())) {
                    if (instructions[i].getRead()!=cycle&&instructions[i].getRead() != 0 && instructions[i].getExe() == 0) {
                        execute(fus[1]);
```

```
                    //判断其是否已经执行完毕
                    if (cycle == instructions[i].getExeCycle()) {
                        instructions[i].setExe(cycle);
                    }
                }
            }
```

4. 对于已经执行完毕且还未写结果的指令，判断是否有正在执行的指令需要读当前指令的目的寄存器的值（是否存在 WAR 冲突），如果不存在，则允许其写结果，并记录该记分牌已经完成执行

```
if(instructions[i].getExe()!=cycle&&instructions[i].getExe()!=0&&instructions[i].getWriteback() == 0) {
                    boolean flag = true;
                    ////判断是否有正在执行的指令需要读当前指令的目的寄存器的值
                    for (FU fu : fus) {
                        if (fus[0].getFi().equals(fu.getFj()) && fu.getRj()) {
                            flag = false;
                        }
                        if (fus[0].getFi().equals(fu.getFk()) && fu.getRk()) {
                            flag = false;
                         }
                     }
                    ////若没有，则不存在WAR冲突，进行写结果操作，并记录该指令已完成执行
                    if (flag) {
                        writeback(fus, fus[0], result);
                        instructions[i].setWriteback(cycle);
                        instructions[i].setFinish(true);
                    }
                }
```

5、在每一个周期开始，先判断是否所有的指令都已经完成执行，如果均已经完成，程序结束。

```
//遍历指令队列
        for (i = 0; i < instructions.length; i++) {
            if (!instructions[i].getFinish()) {
                break;
            }
        }
        if (i >= instructions.length) {
            break;
        }
```

（四）输出数据

在每一个周期结束，使用 draw1()或 draw2()方法输出记分牌中三张表的信息，该方法获取每个单元格所需的信息，更新表格。以下是 draw1()中的代码：

```
for (int i = 0; i < instructions.length; i++) {
            table1.setValueAt(String.valueOf(instructions[i].getIssue()>0?instructions[i].getIssue():""), i, 1);
            table1.setValueAt(String.valueOf(instructions[i].getRead()>0?instructions[i].getRead():""), i, 2);
            table1.setValueAt(String.valueOf(instructions[i].getExe()>0?instructions[i].getExe():""), i, 3);
            table1.setValueAt(String.valueOf(instructions[i].getWriteback()>0?instructions[i].getWriteback():""), i, 4);
        }
        for (int i = 0; i < 5; i++) {
            table2.setValueAt(String.valueOf(fus[i].getbusy()), i, 1);
            table2.setValueAt(fus[i].getOp(), i, 2);
             table2.setValueAt(fus[i].getFi(), i, 3);
             table2.setValueAt(fus[i].getFj(), i, 4);
             table2.setValueAt(fus[i].getFk(), i, 5);
             table2.setValueAt(fus[i].getQj(), i, 6);
             table2.setValueAt(fus[i].getQk(), i, 7);
             table2.setValueAt(String.valueOf(fus[i].getRj()), i, 8);
             table2.setValueAt(String.valueOf(fus[i].getRk()), i, 9);
        }
        for (int i = 0; i < 1; i++) {
            for (int j = 1; j < 17; j++) {
                table3.setValueAt(result.get("F" + (2 * j - 2)), 0, j);
            }
        }
```

四、Console 和 GUI 版本

该项目的输出分为 Console 控制台输出和 GUI 输出两个版本，可根据不同需求设置不同输出，在代码中分别由 init2()和 init1()方法初始化，并由 draw2()和 draw1()方法实现。

（一）Console 版本

注：该部分格式和可变元素调用 Table 文件夹中的 ConsoleTable.java 实现。关于该部分的出处见参考资料 3。

运行指令为：

```
java ScoreBoard.java ConsolePrint
```

运行效果如下：

```
C:\zlr-Scoreboard\src>java -classpath "C:\zlr-Scoreboard\src\\";. ScoreBoard ConsolePrint
请分别输入各部件的延迟，加法部件延迟：2
乘法部件延迟：10
除法部件延迟：40
请输入指令的数量：6
请输入指令：
LD F6,34(R2)
LD F2,45(R3)
MULTD F0,F2,F4
SUBD F8,F6,F2
DIVD F10,F0,F6
ADDD F6,F8,F2
```

Cycle 1
指令状态表

Instruction	Issue	ReadOprand	Execution	WriteBack
LD F6,34(R2)	1			
LD F2,45(R3)				
MULTD F0,F2,F4				
SUBD F8,F6,F2				
DIVD F10,F0,F6				
ADDD F6,F8,F2				

功能部件状态表

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	true	LD	F6		R2			true	true
Add	false							false	false
Mult1	false							false	false
Mult2	false							false	false
Div	false							false	false

结果寄存器状态表

F0	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30
			Integer												

按回车键继续

Cycle 14
指令状态表

Instruction	Issue	ReadOprand	Execution	WriteBack
LD F6,34(R2)	1	2	3	4
LD F2,45(R3)	5	6	7	8
MULTD F0,F2,F4	6	9		
SUBD F8,F6,F2	7	9	11	12
DIVD F10,F0,F6	8			
ADDD F6,F8,F2	13	14		

功能部件状态表

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	false							false	false
Add	true	ADDD	F6	F8	F2			true	true
Mult1	true	MULTD	F0	F2	F4			false	false
Mult2	false							false	false
Div	true	DIVD	F10	F0	F6	Mult1		false	true

结果寄存器状态表

F0	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30
Mult1			Add		Div										

按回车键继续

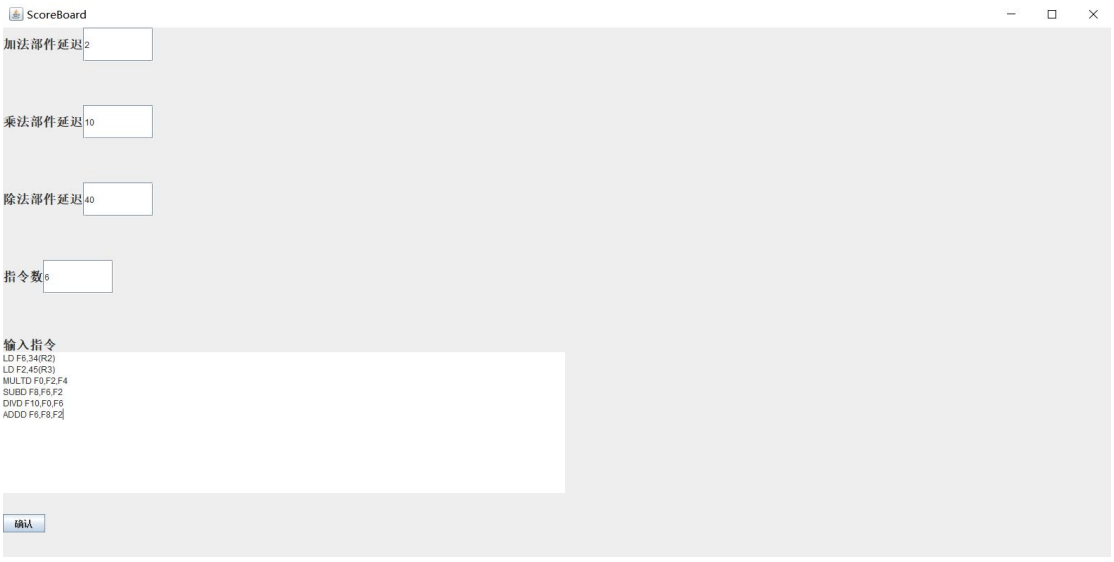
```
Cycle 62
指令状态表
+----------------+-------+------------+-----------+-----------+
| Instruction    | Issue | ReadOprand | Execution | WriteBack |
+----------------+-------+------------+-----------+-----------+
| LD F6,34(R2)   | 1     | 2          | 3         | 4         |
| LD F2,45(R3)   | 5     | 6          | 7         | 8         |
| MULTD F0,F2,F4 | 6     | 9          | 19        | 20        |
| SUBD F8,F6,F2  | 7     | 9          | 11        | 12        |
| DIVD F10,F0,F6 | 8     | 21         | 61        | 62        |
| ADDD F6,F8,F2  | 13    | 14         | 16        | 22        |
+----------------+-------+------------+-----------+-----------+
功能部件状态表
+---------+-------+----+----+----+----+----+----+-------+-------+
| Name    | Busy  | Op | Fi | Fj | Fk | Qj | Qk | Rj    | Rk    |
+---------+-------+----+----+----+----+----+----+-------+-------+
| Integer | false |    |    |    |    |    |    | false | false |
| Add     | false |    |    |    |    |    |    | false | false |
| Mult1   | false |    |    |    |    |    |    | false | false |
| Mult2   | false |    |    |    |    |    |    | false | false |
| Div     | false |    |    |    |    |    |    | false | false |
+---------+-------+----+----+----+----+----+----+-------+-------+
结果寄存器状态表
+----+----+----+----+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| F0 | F2 | F4 | F6 | F8 | F10 | F12 | F14 | F16 | F18 | F20 | F22 | F24 | F26 | F28 | F30 |
+----+----+----+----+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|    |    |    |    |    |     |     |     |     |     |     |     |     |     |     |     |
+----+----+----+----+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
按回车键继续
```

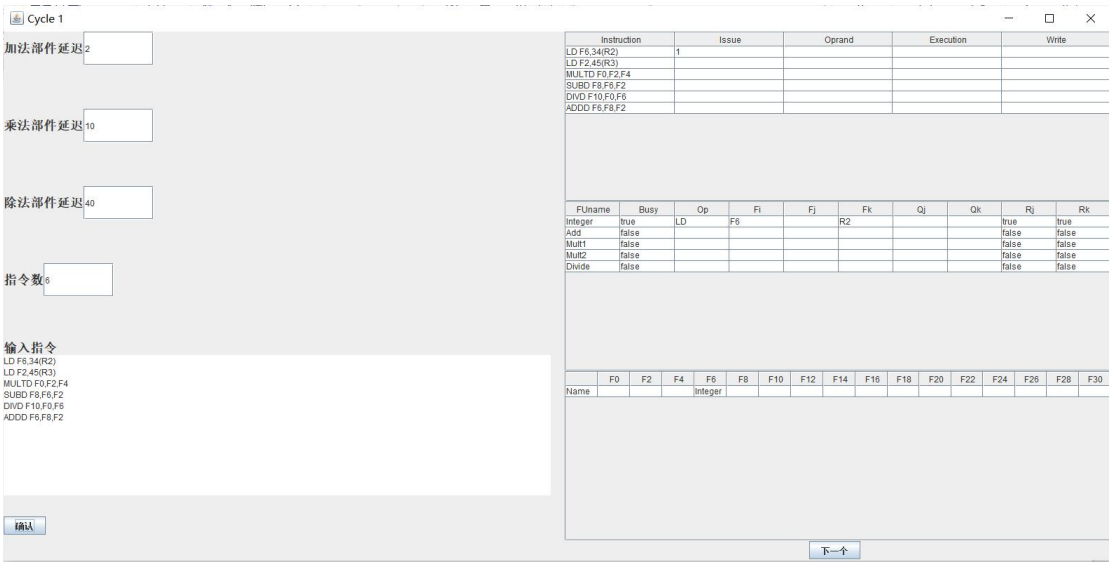
（二）GUI 版本

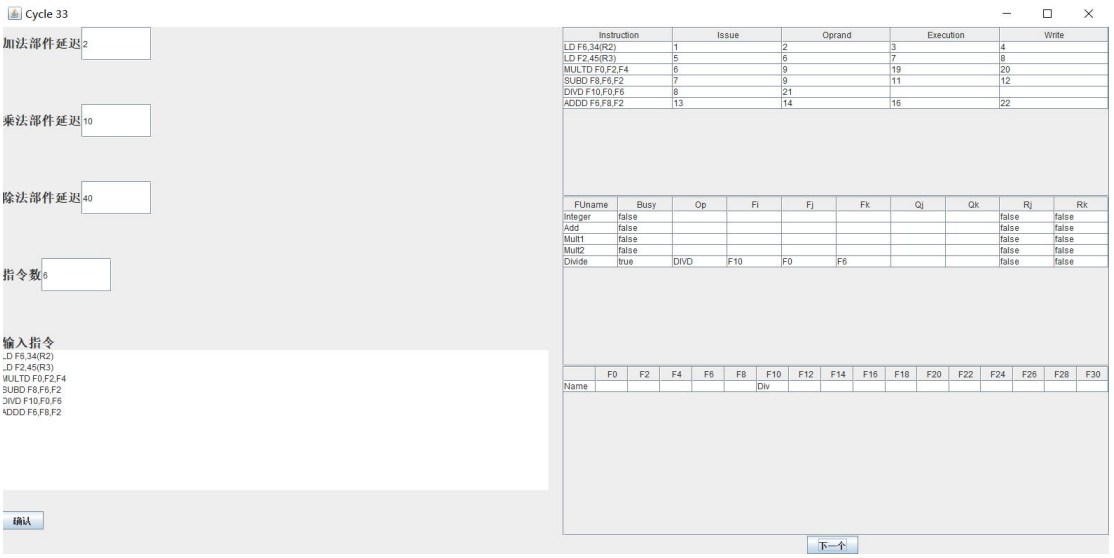
运行指令为：

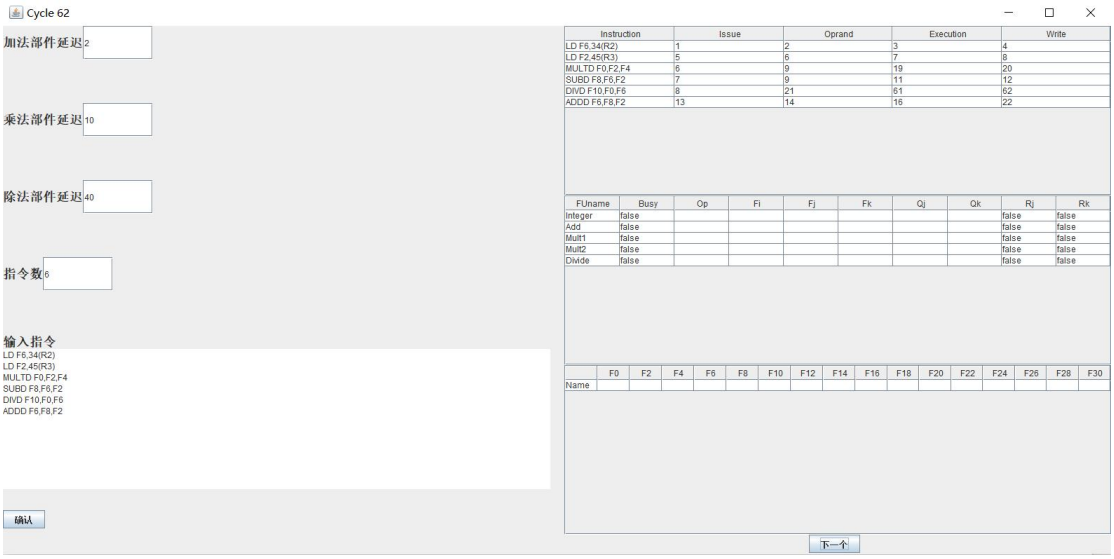
```
java ScoreBoard.java GUIPrint
```

运行效果如下：









五、测试用例

测试用例 TestExamples.txt 列举了包括存在 WAW、WAR、RAW 三种冲突

的实例，通过测试对比了该记分牌仿真器和其他仿真器的输出值，并验证了结果的正确性。

六、参考资料

1、记分牌原理：

《计算机系统结构·量化研究方法（第五版）》 [美] John L. Hennessy / [美] David A. Patterson

2、Scoreboading 算法：

https://en.wikipedia.org/wiki/Scoreboarding

3、Table 文件夹：

https://github.com/clyoudu/clyoudu-util/tree/master/src/main/java/github/clyoudu/consoletable