系统结构实验二实验报告

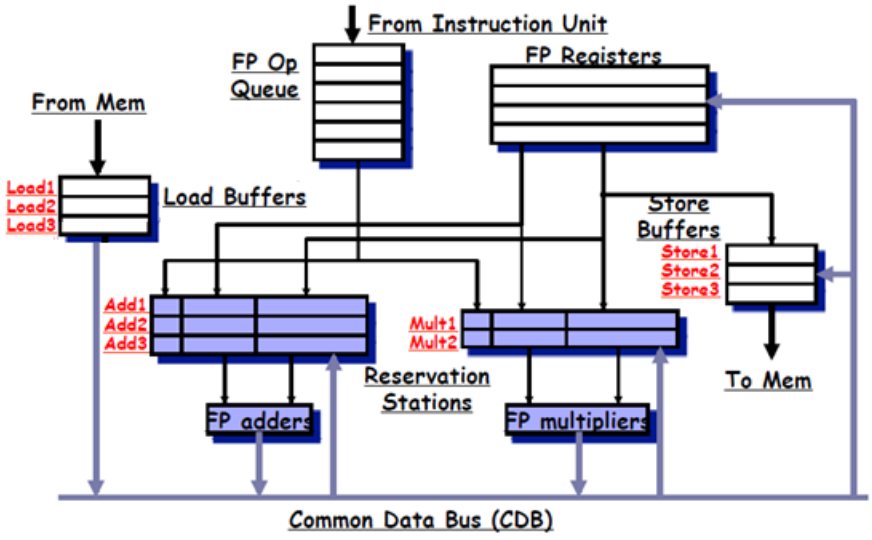
小组成员：闫吉宇 2014011435，魏佳夫 2014011437，陈明豪 2014011443

实验原理

Tomasulo 算法

Tomasulo 算法以硬件方式实现了寄存器重命名，允许指令乱序执行，是提高流水线的吞吐率和效率的一种有效方式。该算法首先出现在 IBM360/91 处理机的浮点处理部件中，后广泛应用于现代处理器设计中。

假设浮点处理部件结构如下图所示。浮点处理部件从取指单元接收指令，存入浮点操作队列。浮点操作队列每拍最多发射 1 条指令给浮点加法器或浮点乘除法器。浮点处理部件包含一个浮点加法器和一个浮点乘除法器。浮点加法器为两段流水线，输入端有三个保留站 A1、A2、A3，浮点乘除法器为六段流水线，输入端有两个保留站 M1，M2。当任意一个保留站中的两个源操作数到齐后，如果对应的操作部件空闲，可以把两个操作数立即送到浮点操作部件中执行。Load Buffer 和 Store Buffer 各缓存三条访存操作。



实验要求

设计实现 Tomasulo 算法模拟器

Tomasulo 算法模拟器能够执行浮点加、减、乘、除运算及 LOAD 和 STORE 操作，为了简化起见，我们在下表中给出了各种操作的具体描述。

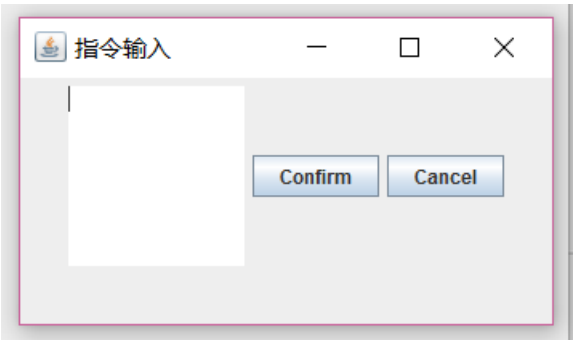
指令格式	指令说明	指令周期	保留站/缓冲队列项数
ADDD F1,F2,F3	F1, F2, F3为浮点寄存器 寄存器至少支持（F0~F10）	2个周期	3
SUBD F1, F2, F3	同上	2个周期	
MULD F1, F2, F3	同上	10个周期	2
DIVD F1, F2, F3	同上	40个周期	
LD F1, ADDR	F1为寄存器，ADDR为地址， 0<=ADDR<4096	2个周期	3
ST F1, ADDR	同上	2个周期	3

- 支持单步执行及连续执行（n条指令），实时显示算法的运行状况，包括各条指令的运行状态、各寄存器以及内存的值、保留站（Reservation Stations）状态、Load Buffer和Store Buffer缓存的值等；
- 程序执行完毕后，能够显示指令执行周期数等信息；
- 为了简化设计，建议模拟器提供编辑内存值功能，以便实现数据输入；浮点除法可不做除0判断；
- 能够以文本方式输入指令序列。

界面简介

全局来看，运行后界面如下

1. **指令输入按钮**可以提供文本方式输入指令功能，右端复选框表示的是即将要执行的代码，上图表示的是 default 情形。点击指令输入按钮后弹出如下界面



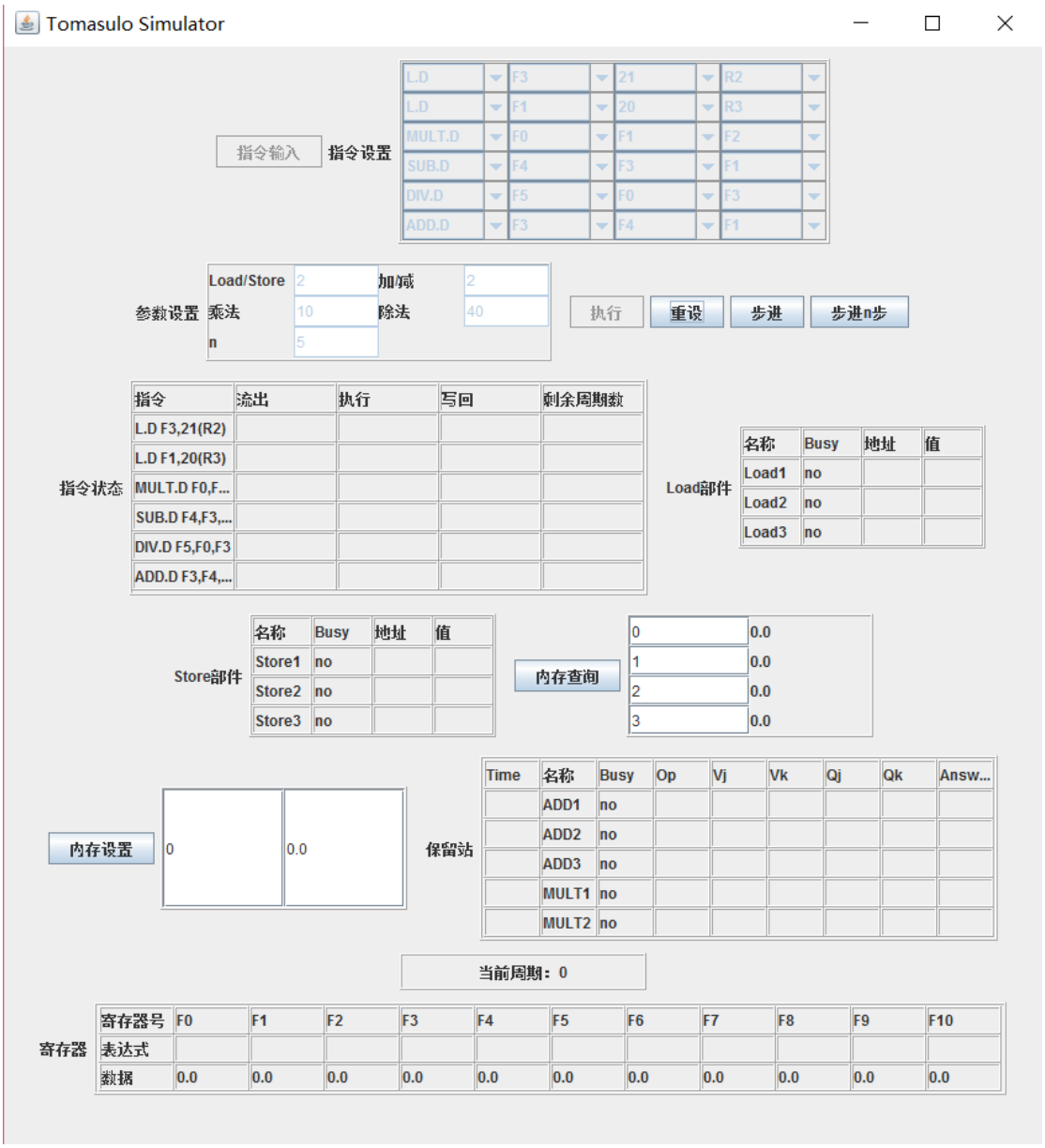
可在文本输入区域按格式输入多行指令，点击 confirm 后该窗体关闭，主窗体指令设置部分将变为输入的指令，不再是 default 情形。

还可以通过主窗体指令设置复选框的点击进行指令的输入，对每条指令的格式限制已经集成在程序中了，可以保证该方法写入的指令格式正确。

2. 自此，执行指令序列的输入已经结束。

参数设置框有 5 种参数可供设置：不同指令形式需要的周期数，以及执行后步进 n 步的 n 值设置。

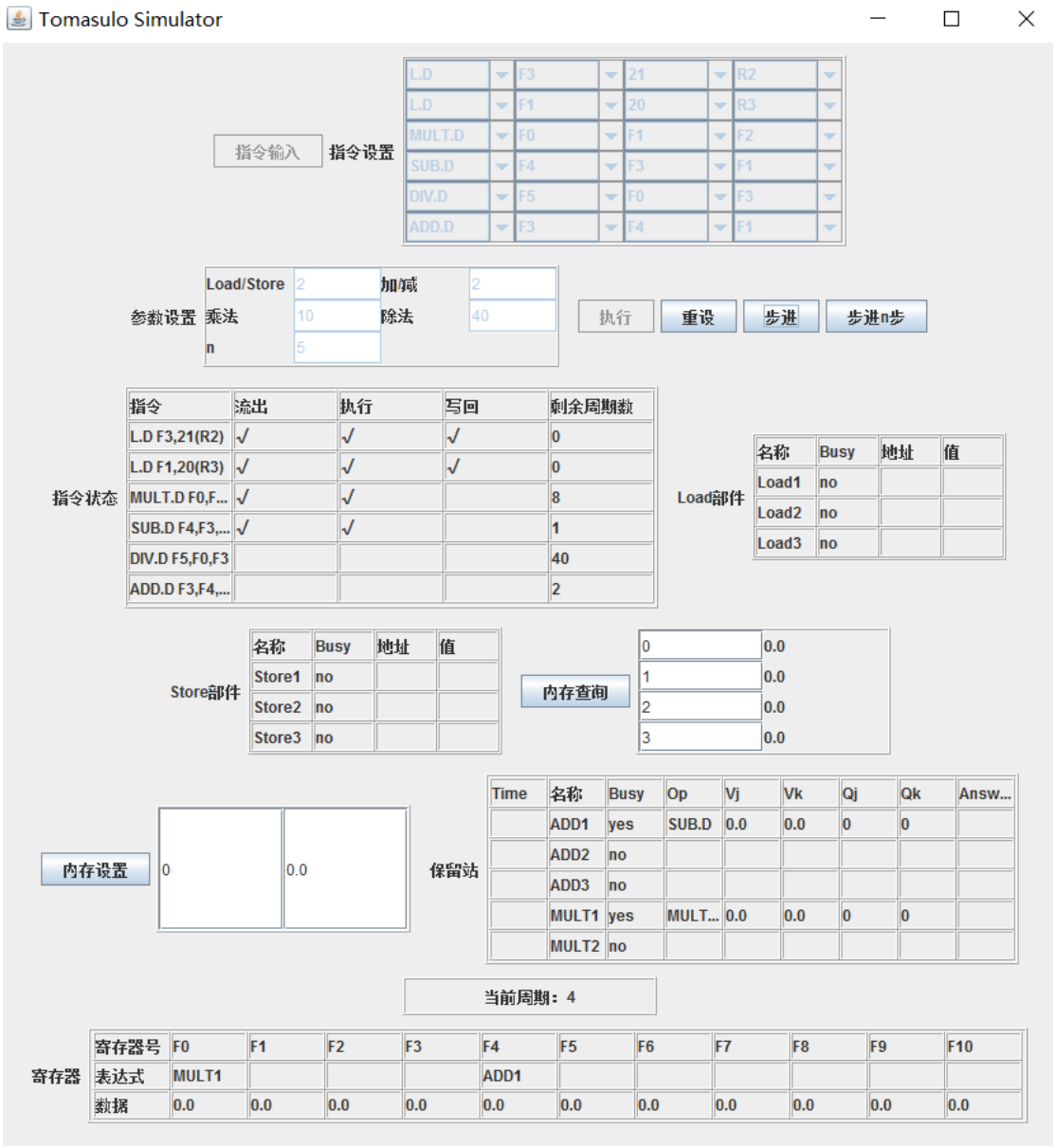
3. 点击执行按钮后，指令序列将开始执行，界面如下



有关指令设置和参数设置的功能都已关闭，可点击的按钮如下：

1. 步进，步进 n 步；分别使流水线向前走 1 个周期和 n 个周期。
2. 内存查询，先在文本框中输入想要查询的地址，点击内存查询内存中的值后会显示在后面。
3. 内存设置，第一个文本框输入地址，第二个文本框输入数值，点击内存设置按钮将对应地址的值设为对应数值。
4. 重设按钮，可以重新初始化整个系统，回到指令设置界面

除按钮外，其他部件的显示都会随按钮的点击与当前指令执行状态实时显示对应模拟器件的状态。类似于：



算法具体实现

类定义说明：

1.寄存器状态类

```java
public class RegisterStation{
    /**
     * 存放以该寄存器为目的寄存器的保留站名称
     */
    String Qi;
    /**
     * 该寄存器的值
     */
    String value;
    /**
     * 寄存器此时的状态
     */
    String state;
}
```

2.指令类，表示单条指令

```java
public class Instruction{
    /**
     * 指令名称
     */
    String name;
    /**
     * 目的操作数
     */
    String opr1;
    /**
     * 源操作数1
     */
    String opr2;
    /**
     * 源操作数2
     */
    String opr3;

}
```

3.指令执行状态类

```java
public class InstructionStation{
    /**
     * 当前指令所在保留站名称
     */
    String Qi;
    /**
     * 当前指令所处的状态
     */
    int state;
    /**
     * 指令执行所需要的时间
     */
    int excutetime;
    /**
     * 该指令状态所对应的指令
     */
    Instruction instruction;

    boolean out,exec,wb;

}
```

4.保留站状态类

```java
public class ReservationStation{
    /**
     * 设置Qi便于对寄存器Qi进行赋值，Qi为该保留站的名称，用以告知指定寄存器结果来源
     * */
    String Qi;
    /**
     * 指令第一个操作数：op2
     * */
    String Vj;
    /**
     * 指令第二个操作数：op3
     * */
    String Vk;
    /**
     * 产生指令第一个操作数值的保留站
     * */
    String Qj;
    /**
     * 产生指令第二个操作数值的保留站
     * */
    String Qk;
    /**
     * 指令操作的类型
     * */
    String Op;
    /**
     * 保留站工作状态
     */
    String Busy;
    /**
     * 保留站结果
     */
    String Answer;
}
```

5.load/store 缓存类

```
public class LoadStoreStation{
    /**
     * 保留站名称
     */
    String Qi;
    /**
     * Load组件状态
     */
    String Busy;
    /**
     * Load组件访存地址
     */
    String Addr;
    /**
     * 访存值
     */
    String value;

    boolean ready;
}
```

大部分状态都用String存储，便于前端显示

6.主类：Tomasolu类，过长不再贴图

主要方法有：

Tomasulo()，界面初始化

init()，重置初始化

getTimeForEX(Instruction) 获取指令需要的执行周期

display() 前后端交互

actionPerformed(ActionEvent) 按钮事件

core() 步进一步

guangbo(String, String) 广播，模拟公共数据总线

execute(String) 执行，调用于core（）

checkrs(String) 检查，调用于core（）

send_out_inst() 流出，调用与core（）

main(String[]) 函数入口

算法核心：

算法核心在 Tomasolu 中的 core（ ）函数中，步进按钮会调用该函数，对 Tomasolu 算法的模拟实现也在该函数中，前后端接口是 display 函数。

core()函数分两部分：

1. 步进部分

```
public void core(){
    send_out_inst();
    for(int i=0; i<IS.length;i++){
        if(IS[i].out && !IS[i].wb){
            if(checkrs(IS[i].Qi)){
                IS[i].exec = true;
                IS[i].excutetime -= 1;
                if(IS[i].excutetime == 0){
                    if(!execute(IS[i].Qi)){
                        System.out.printf("wrong : %s",IS[i].Qi);
                    }
                    IS[i].wb = true;
                }
            }
        }
    }
```

先尝试发送指令（send_out_inst()），再尝试执行（checkrs() and execute()）已被发送过还未完成的指令。

2. 前端交互部分

```
for(int i = 0;i<RegS.length;i++){
    regst[1][i+1] = RegS[i].Qi;
    regst[2][i+1] = RegS[i].value;
}
for(int i = 0;i<LS.length;i++){
    ldst[i+1][1] = LS[i].Busy;
    ldst[i+1][2] = LS[i].Addr;
    ldst[i+1][3] = LS[i].value;
}
for(int i = 0;i<SS.length;i++){
    stst[i+1][1] = SS[i].Busy;
    stst[i+1][2] = SS[i].Addr;
    stst[i+1][3] = SS[i].value;
}
for(int i = 0;i<RS.length;i++){
    resst[i+1][1] = RS[i].Qi;
    resst[i+1][2] = RS[i].Busy;
    resst[i+1][3] = RS[i].Op;
    resst[i+1][4] = RS[i].Vj;
    resst[i+1][5] = RS[i].Vk;
    resst[i+1][6] = RS[i].Qj;
    resst[i+1][7] = RS[i].Qk;
    resst[i+1][8] = RS[i].Answer;
}
for(int i=0;i<IS.length;i++){
    if(IS[i].out){
        instst[i+1][1] = "√";
    }
    if(IS[i].exec){
        instst[i+1][2] = "√";
    }
    if(IS[i].wb){
        instst[i+1][3] = "√";
    }
    instst[i+1][4] = Integer.toString(IS[i].excutetime);
}
```

将当前系统状态输出到 String 数组中，由 display 函数在界面上显示出来。

send_out_inst 函数，checkrs 函数，execute 函数，guangbo 函数等等具体不再细表，大体是根据 Tomasolu 算法的流程走了下来，具体见代码。

遇到的问题：

1. 处理 load/store 指令时需要建立缓冲队列 ,这两种指令结束的顺序应该是先进先出的，不然可能发生 Bug。
2. Mem 的设置应该为 float 类型，该实验涉及的运算指令都是浮点数类型。
3. 指令的多行输入需要用 JTextArea，不能用 JTextField。
4. 已有指令执行与发射新指令的前后顺序
5. Load 指令更改目的寄存器 Qi 的时机。

实验感想

本次实验动手比较晚，最终也没能按时完成，真是很抱歉，给助教带来很多不便。

本次实验前端框架来自于网上的一个样例，与本实验还是有较大差别，我们做了很多更改以符合我们实验的要求。算法实现方面则是完全自己重写了一遍。

本次实验自己动手对于算法的理解还是很有帮助的 ,对于流水线动态调度问题也加深了理解。只是小组内负责前端的同学好像没有太涉及算法的内容，感觉他好像吃亏了，O(∩_∩)O~~。