**实验报告**

计试81 白思雨 2186123935

1. 实验目的

(1)加深对指令级并行性及其开发的理解;

(2)加深对Tomasulo算法的理解;

(3)掌握Tomasulo算法在指令流出、执行、写结果各阶段对浮点操作指令以及load和store指令进行什么处理;

(4)掌握采用了Tomasulo算法的浮点处理部件的结构;

(5)掌握保留站的结构;

(6)给定被执行代码片段，对于具体某个时钟周期，能够写出保留站、指令状态表以及浮点寄存器状态表内容的变化情况。

1. 实验平台

自己设计一个流水线模拟器（简称：模拟器 A）

1. 实验内容、步骤和结果

**（1）模拟器A的设计思想、特色：**

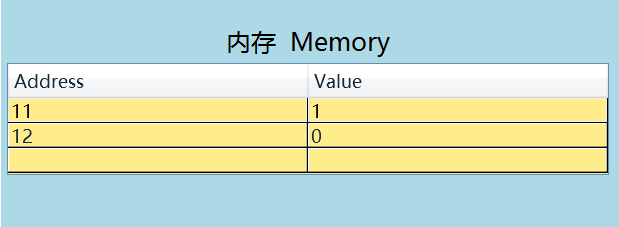
设计思想：本模拟器参考借鉴了部分开源的C#项目代码，**编写了Tomasulo.Core作为主框架**，确保主程序的健全性及功能丰富性；**并利用Windows Presentation Foundation .NET Framework4编写绘制了交互式界面**,方便使用者的操作；在数据表格处理方面，**采用DataGrid 控件，实现指令队列、统计运行数据、寄存器、内存等部分的编辑展示**。

特色：**兼容大量常用指令**，常见指令均可运行；**交互式界面**，易于观察；**可直接编辑内存、寄存器**，操作简单便捷；**可全自动、单步、单时钟周期运行**，操作性强；软件占用空间大小仅为2.36MB，易于存储、拷贝携带。

**（2）模拟器内代码测试：**

1. **没有任何冲突的流水线场景：**

start: 初始内存：

LD F1, 11

ADDD F4, F2, F3

ADDD F5, F3, F3

SUBD F6, F2, F3

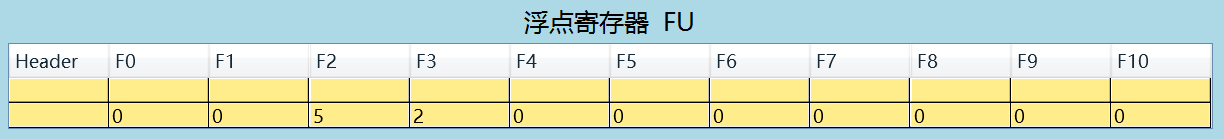
SUBD F7, F3, F2

MULD F8, F2, F3

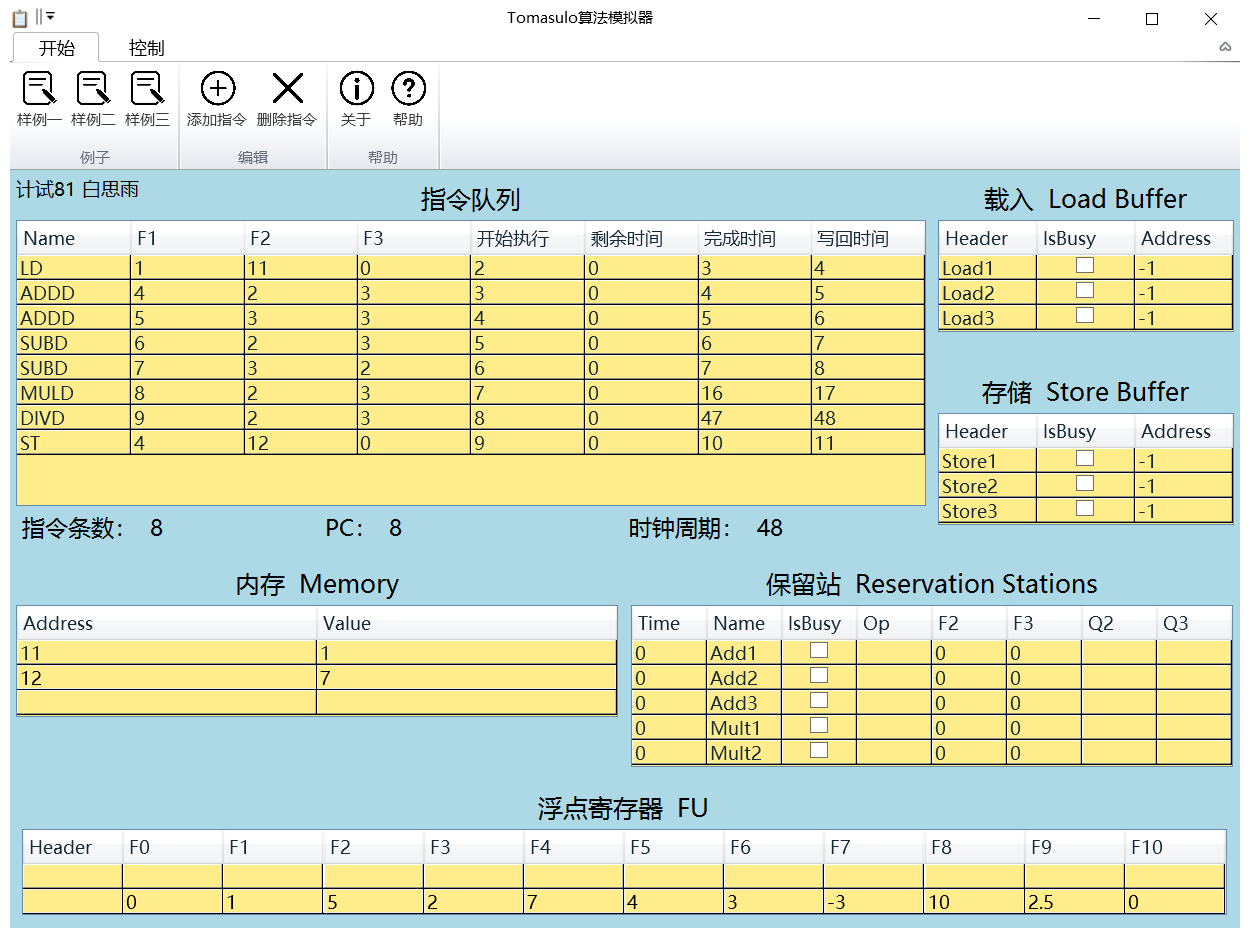
DIVD F9, F2, F3

ST F4, 12

初始寄存器：



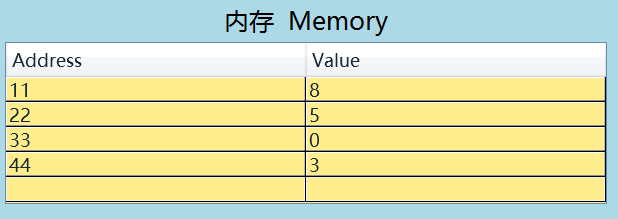
运行结果截图：



可见，每一步的时钟周期都是顺序的，没有任何冲突，一切正常执行。

1. **有至少一次的RAW冲突：**

start: 初始内存：

LD F1, 11

LD F2, 22

ADDD F4, F1, F2

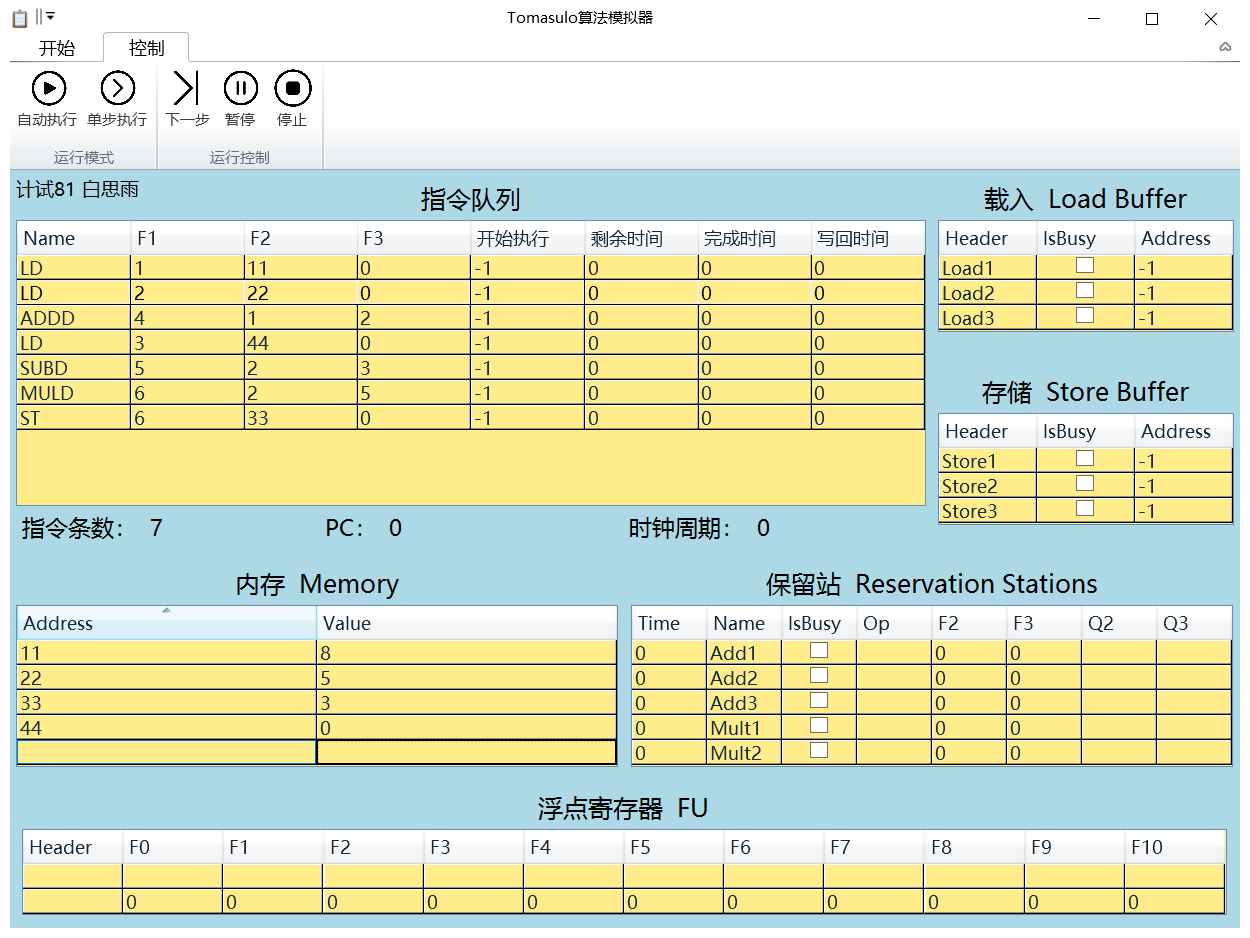
LD F3, 44

SUBD F5, F2, F3

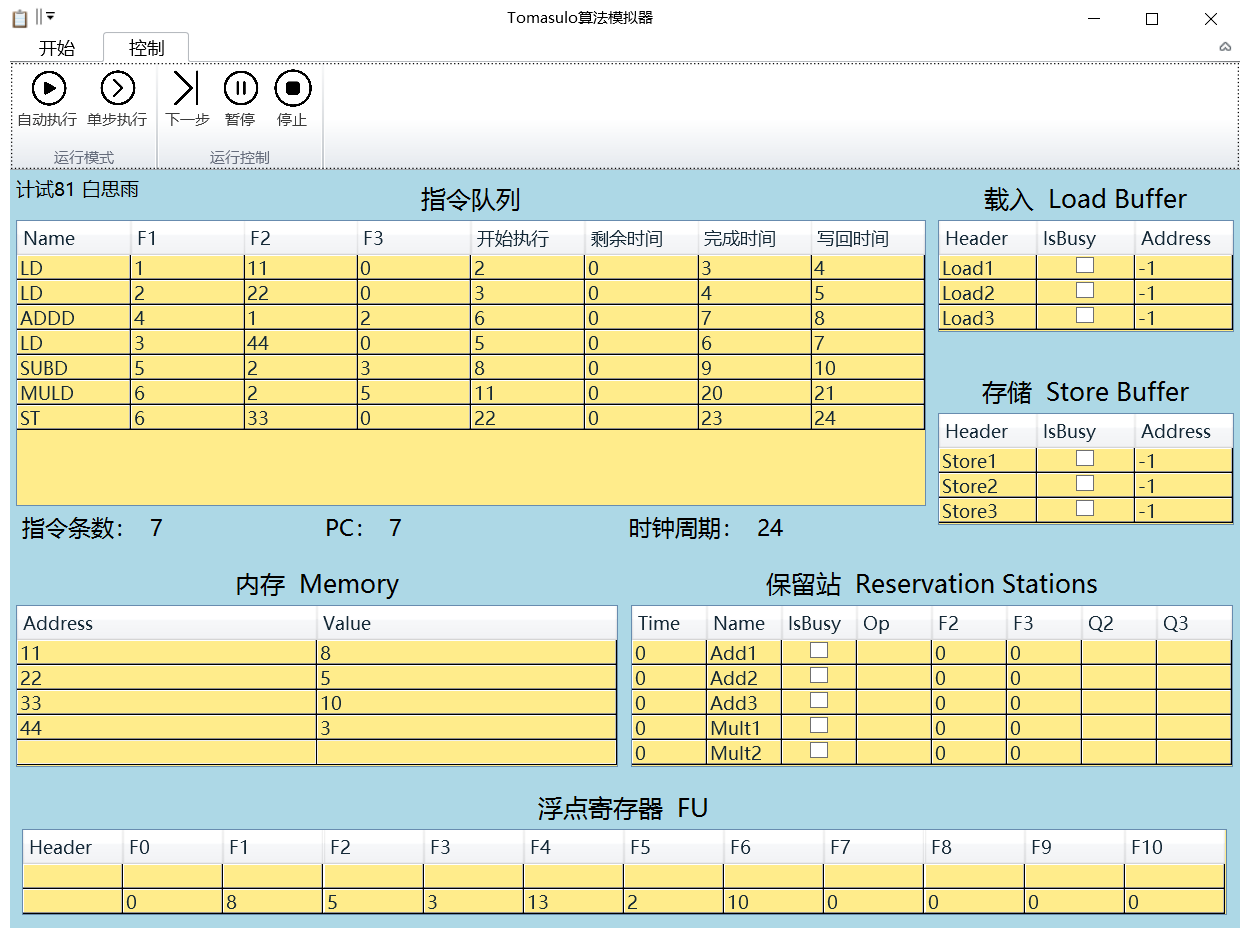
MULD F6, F2, F5

ST F6, 33

初始寄存器：

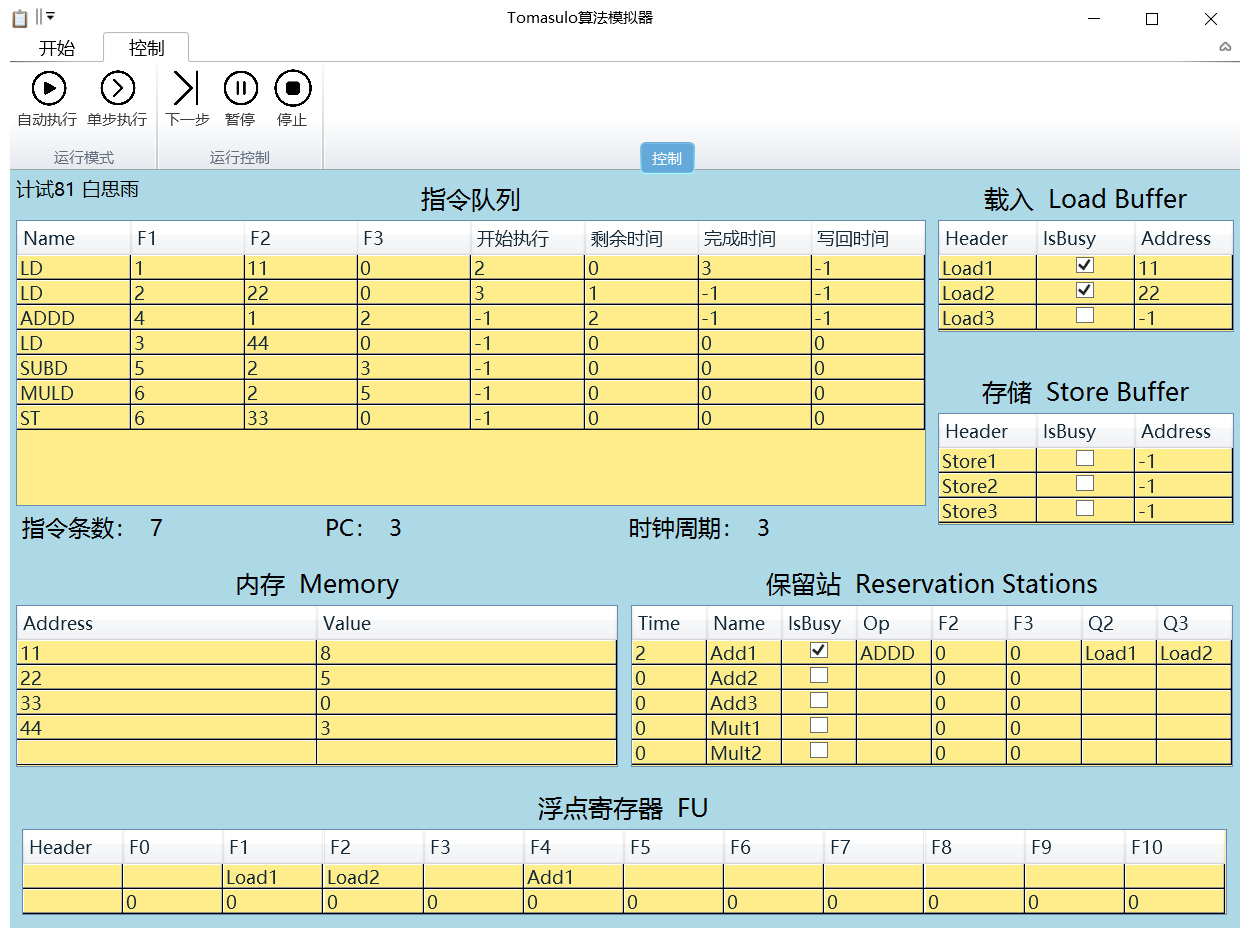


运行结果截图：

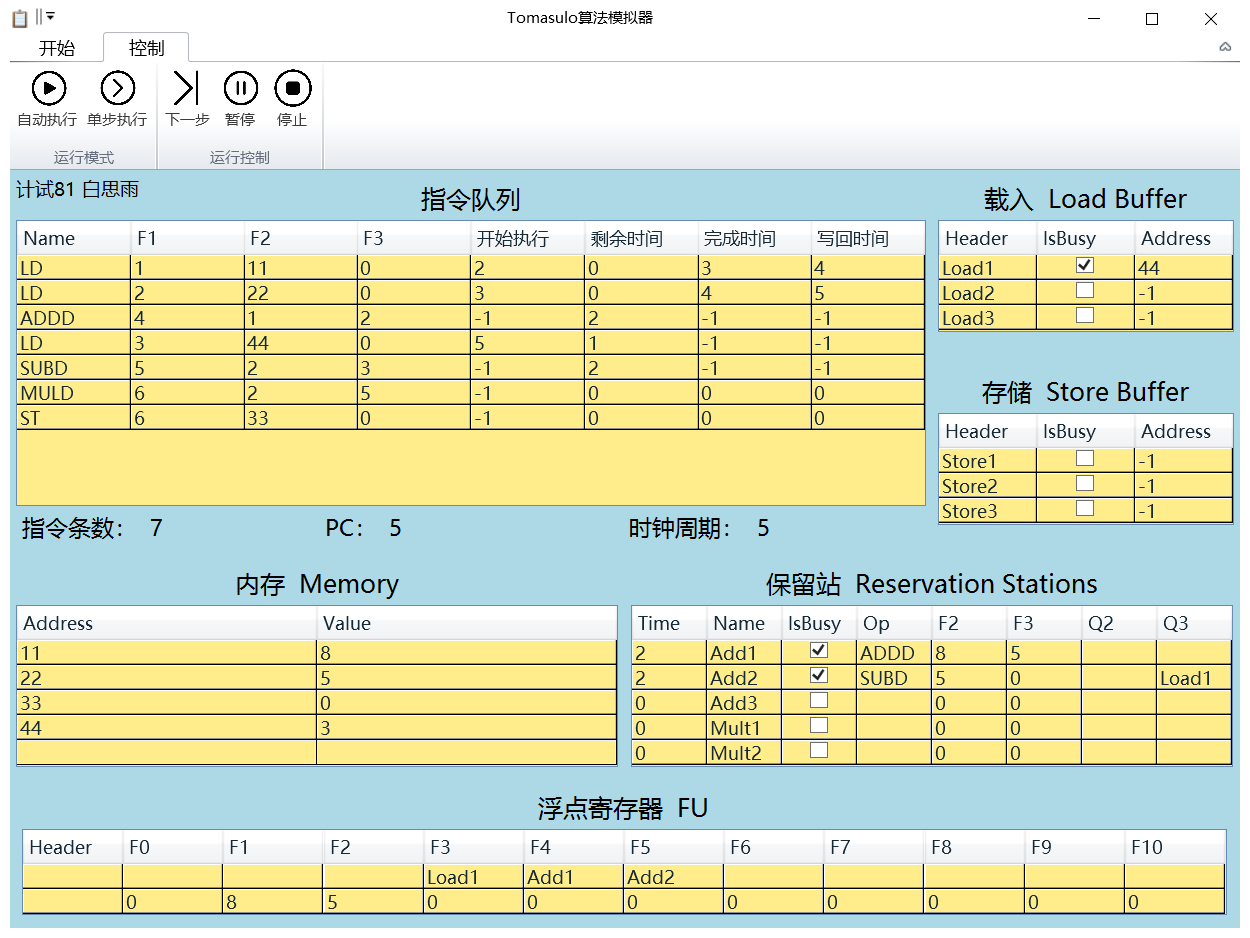


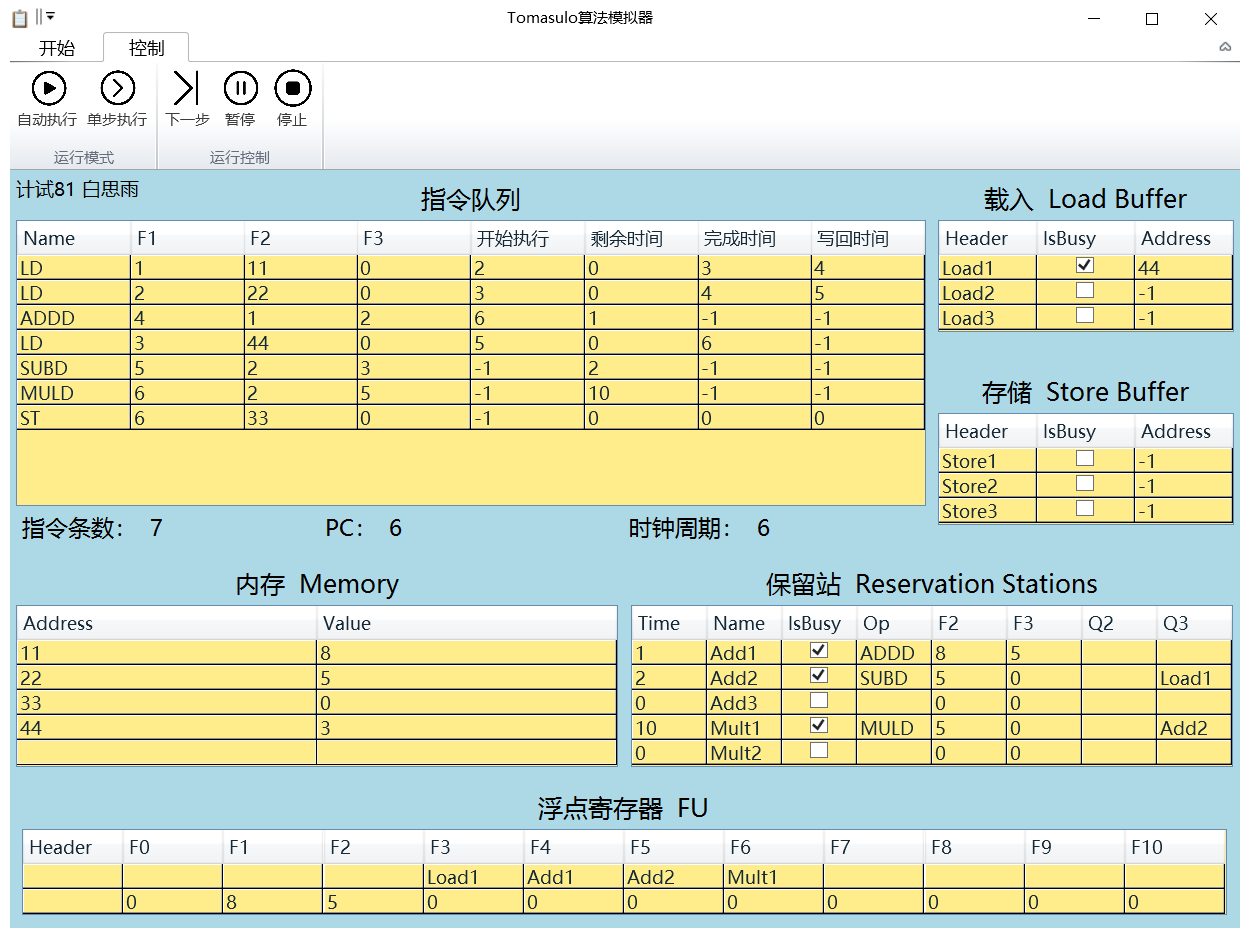
可见，第一、二步的LD F1, 11和LD F2, 22与第三步ADDD F4,F1, F2发生了RAW冲突，第三步延迟并放入保留站。第三步的开始执行时钟周期为6，而第四步开始执行时钟周期为5.

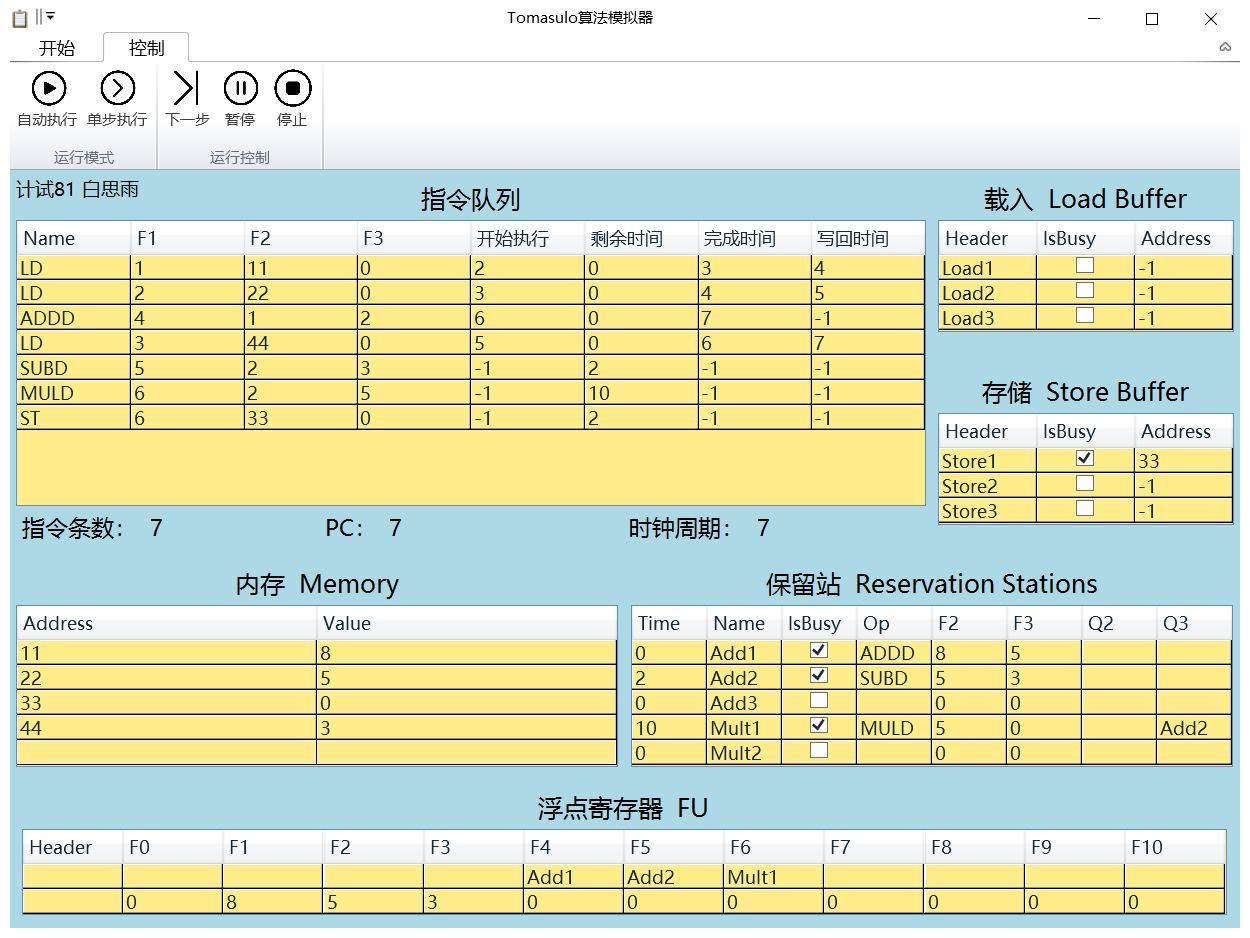
过程具体分析：



由于执行第三步时，前两步读取并未完成，存在RAW冲突。先将第三步ADDD放入保留站Add1中，并开始读取执行第四步。



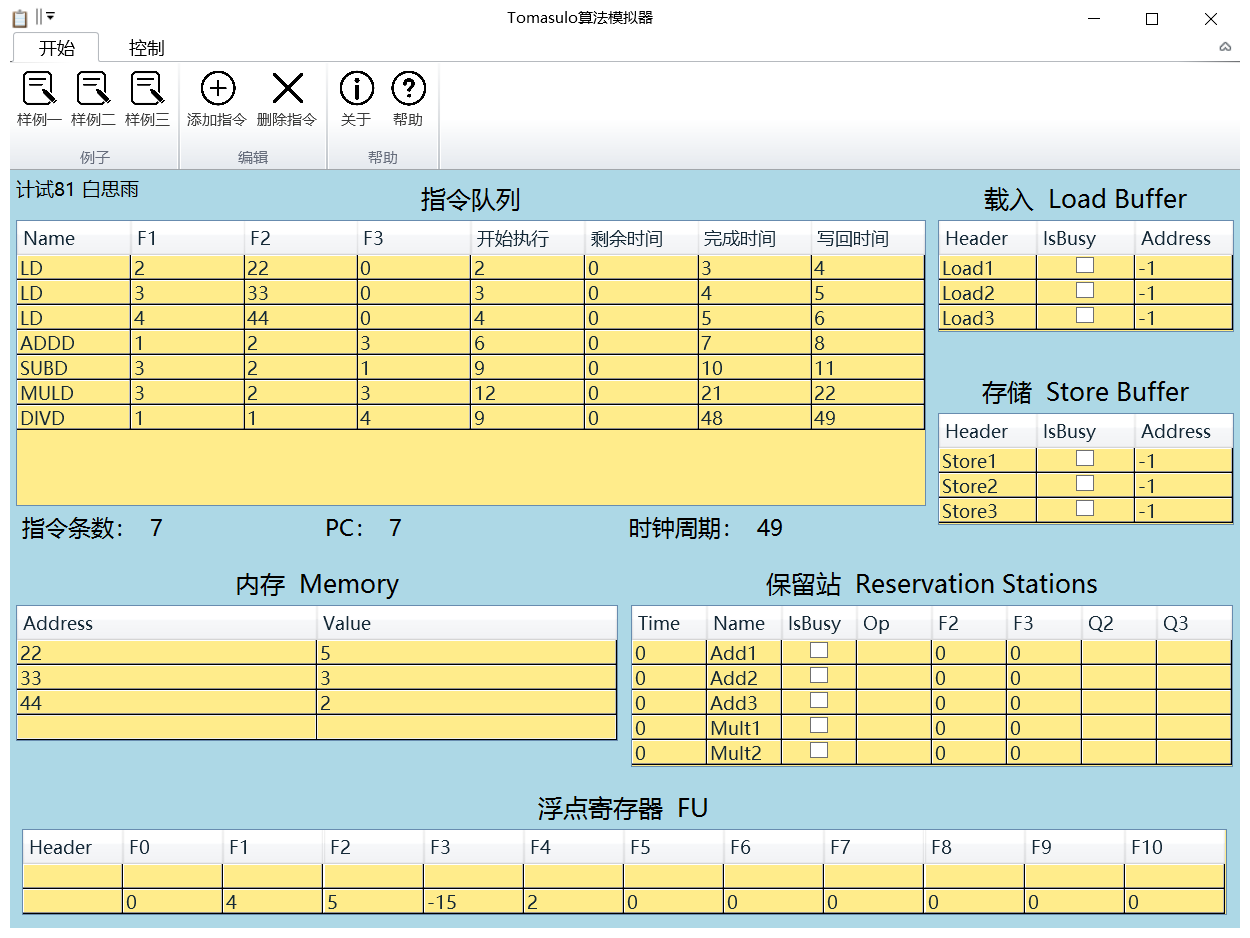




待前两步LD读取完成后，在保留站中的第三步开始执行，并写回F4寄存器，冲突解决。

1. **有至少一次的WAR冲突：**

代码： 初始内存：

start:

LD F2, 22

LD F3, 33

LD F4, 44

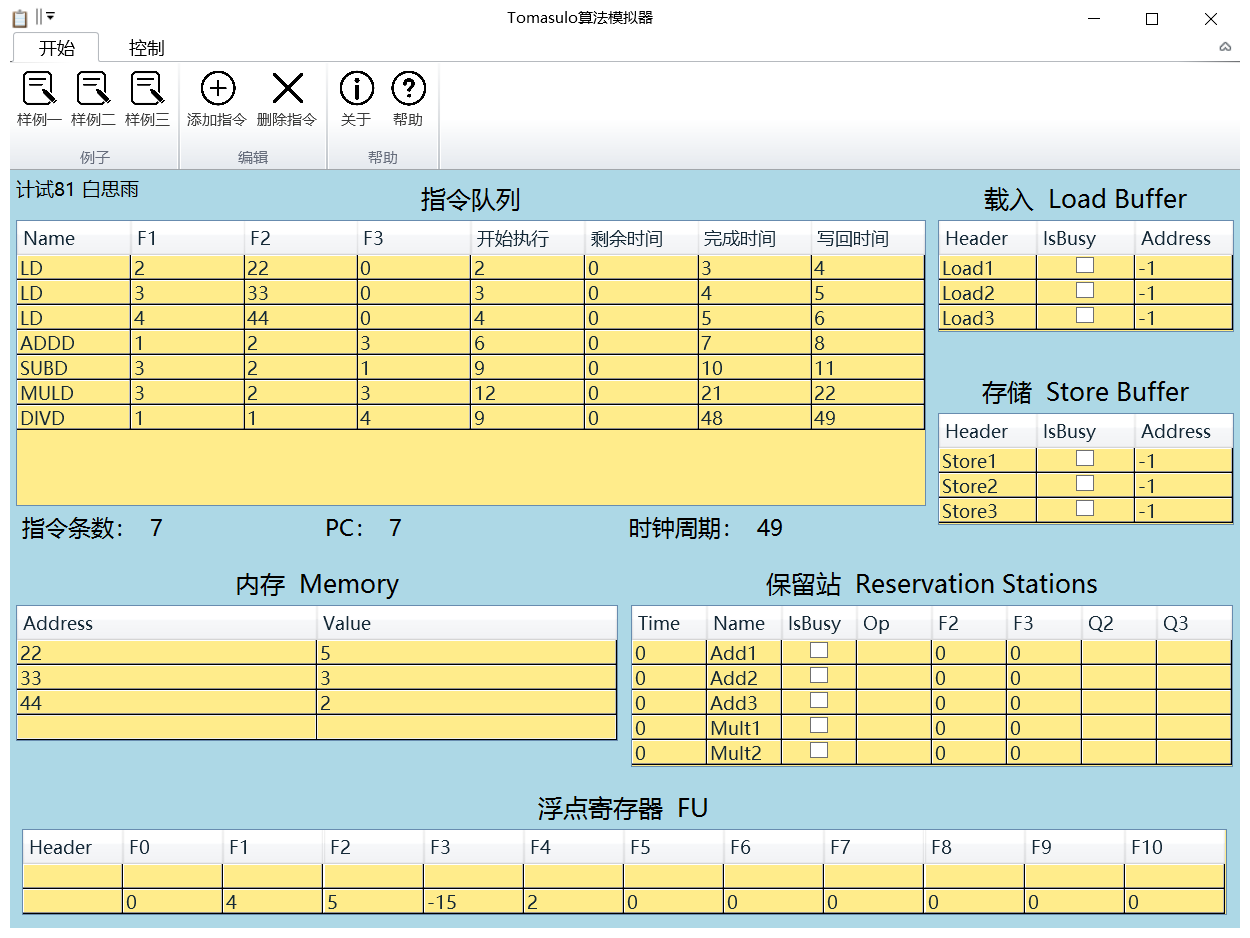
ADDD F1, F2, F3

SUBD F3, F2, F1

MULD F3, F2, F3

DIVD F1, F1, F4

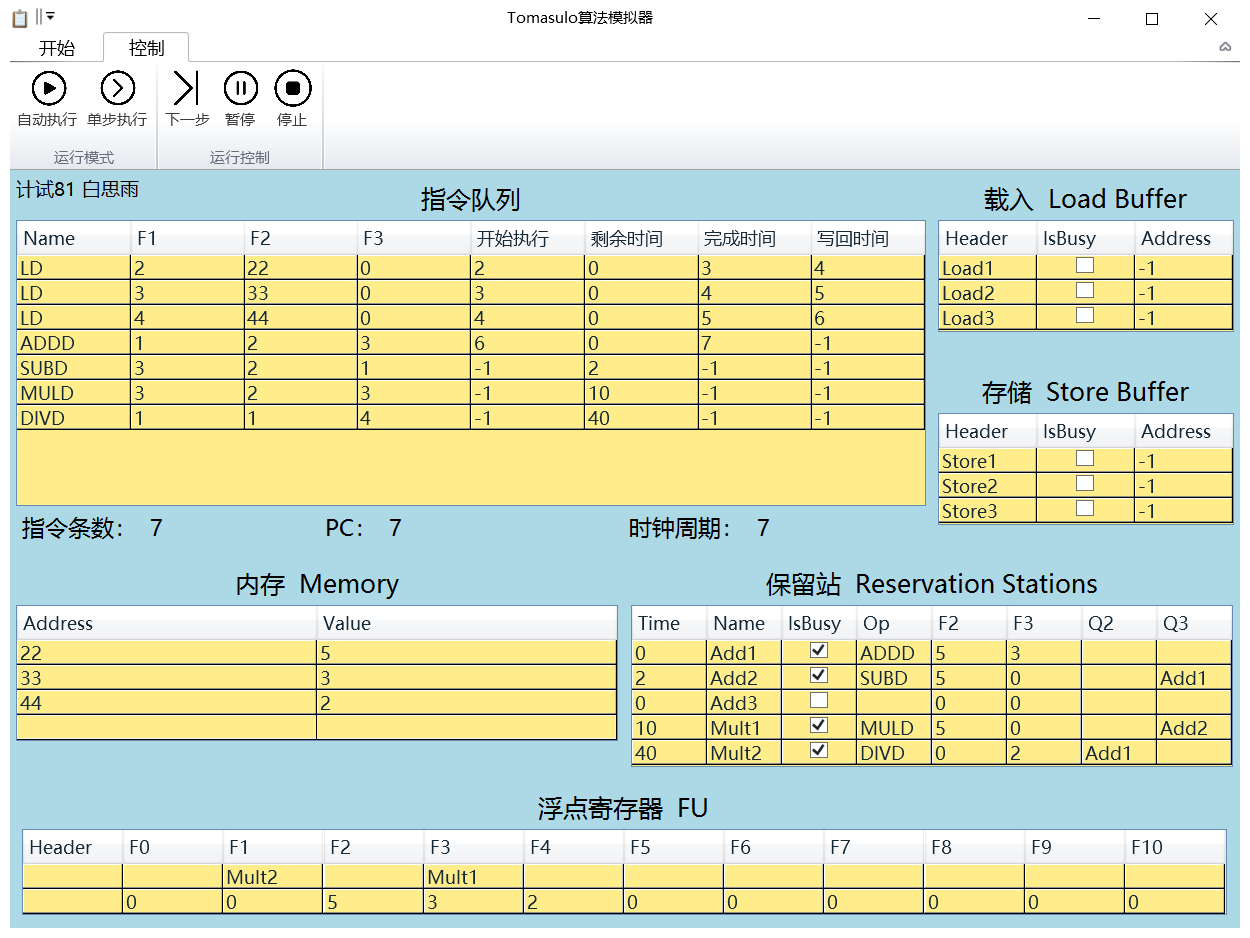
运行结果截图 ：



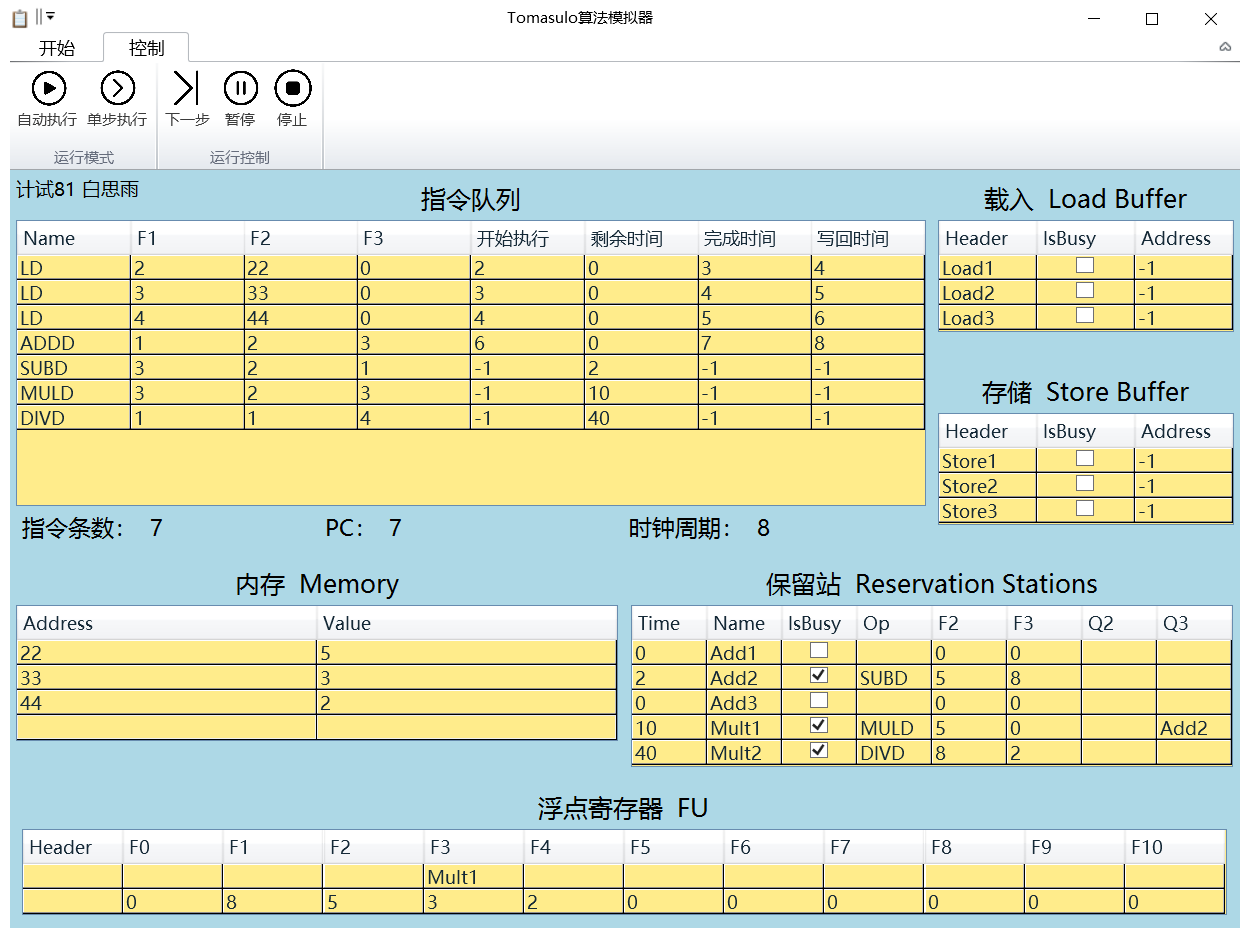
WAR冲突即为在前指令未读取寄存器数据时，后续指令先完成写寄存器操作。

在本例子中，如果第七步DIVD指令在第五步SUBD前执行完，并写入了寄存器F1，则会造成WAF冲突。而Tomasulo算法较好地解决了此问题，代码运行一切正常。

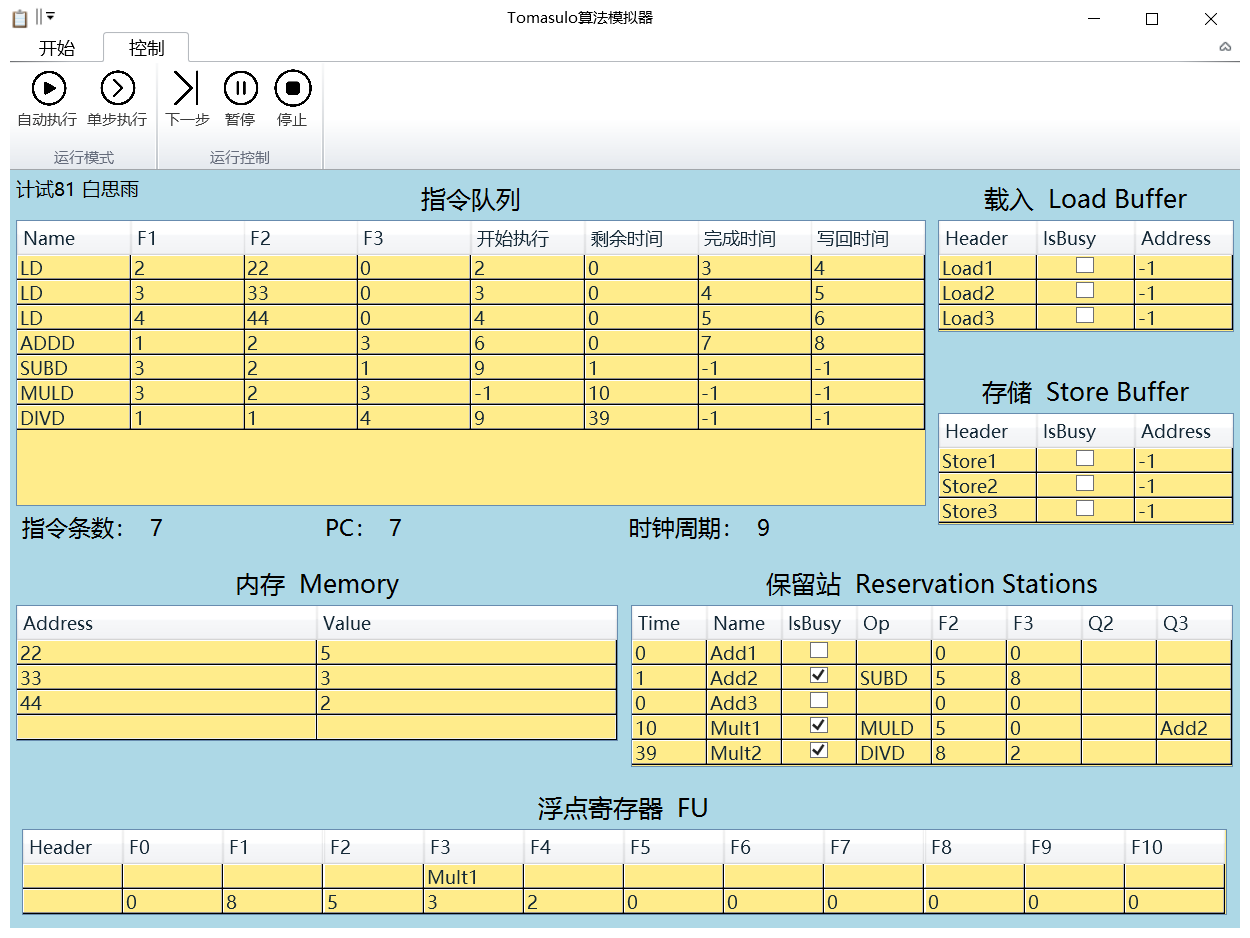
过程具体分析：



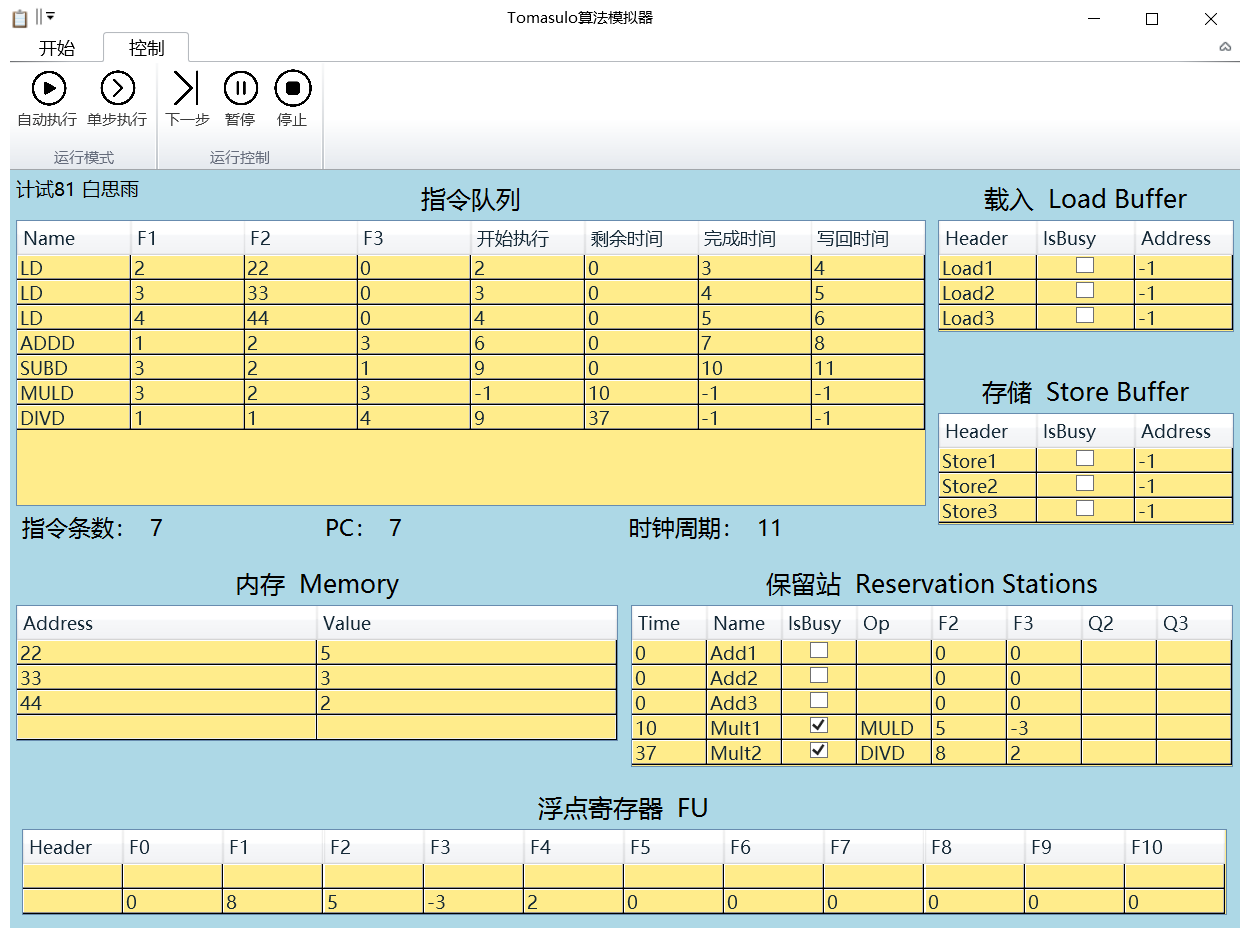
在第七时钟周期时，所有指令均已读取，并通过寄存器换名，存入保留站中。以此解决WAR冲突。



在第八时钟周期时，第四步ADDD已执行完成，Add1变为对应数值。



在第九时钟周期时，第五步SUBD和第七步DIVD开始执行。由于DIVD有40个时钟周期的延迟，暂缓执行。



在第十一时钟周期时，第五步SUBD已执行完成，Add2变为对应数值。

4.源码

见附件

5.实验感悟

本次实验与第一次的５段流水线实验比较相似，在最初着手做这个实验，搜寻网上资料时，发现大部分Tomasulo算法教程是用教材附带的模拟器实现，并不能较深入地帮助我们学习理解。而主推交互式界面的开源Tomasulo模拟器少之又少，需要我们有较强的编程能力与创新思维才能完成。

通过本次实验，引用一些开源项目作为辅助，自主编写Tomasulo模拟器程序，来学习理解Tomasulo算法，极大地锻炼了我们计算机体系结构初学者的动手编程能力，也让我们更好地学习理解了Tomasulo算法的原理以及实现过程，对指令的调度有了更好的认识，既加深了课内知识的理解，也增强了代码的编写能力，受益匪浅！

十分感谢老师对本次实验的精心设计和安排！