模拟器A的设计思想、特色：

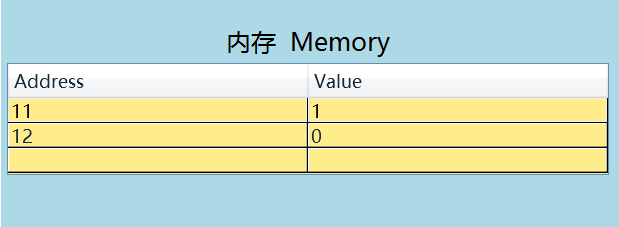
本模拟器参考借鉴了部分开源的C#项目代码，采用Tomasulo.Core作为主框架，确保主程序的健全性及功能丰富性；并利用Windows Presentation Foundation .NET Framework4编写绘制了交互式界面,方便使用者的操作；在数据表格处理方面，采用DataGrid 控件，实现指令队列、统计运行数据、寄存器、内存等部分的编辑展示。

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。

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 1, 1, 1

ADDD 4, 2, 3

ADDD 5, 3, 3

SUBD 6, 2, 3

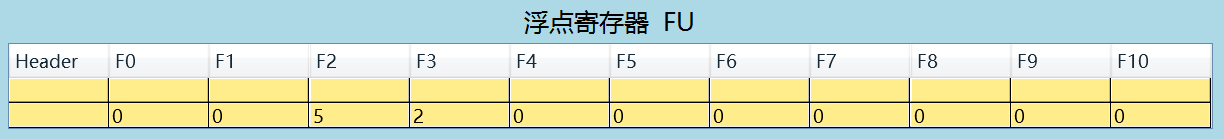
SUBD 7, 3, 2

MULD 8, 2, 3

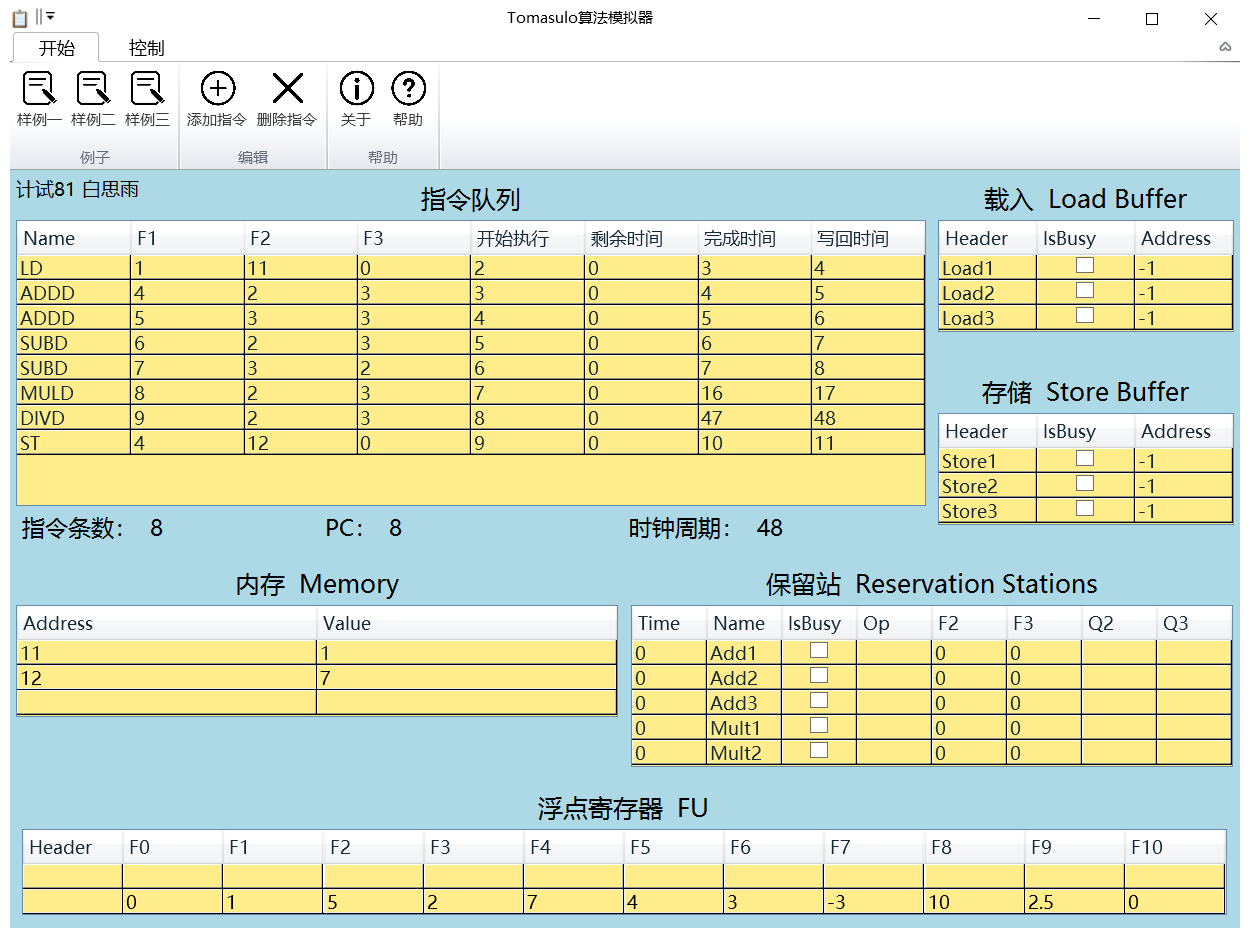
DIVD 9, 2, 3

ST 4, 12

初始寄存器：



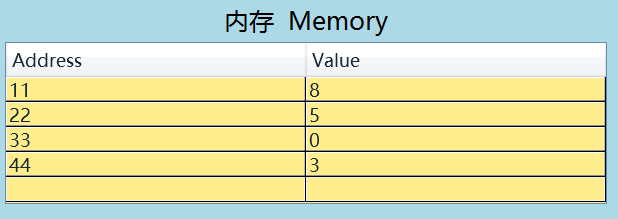
运行结果截图：



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 1, 11

LD 2, 22

ADDD 4, 1, 2

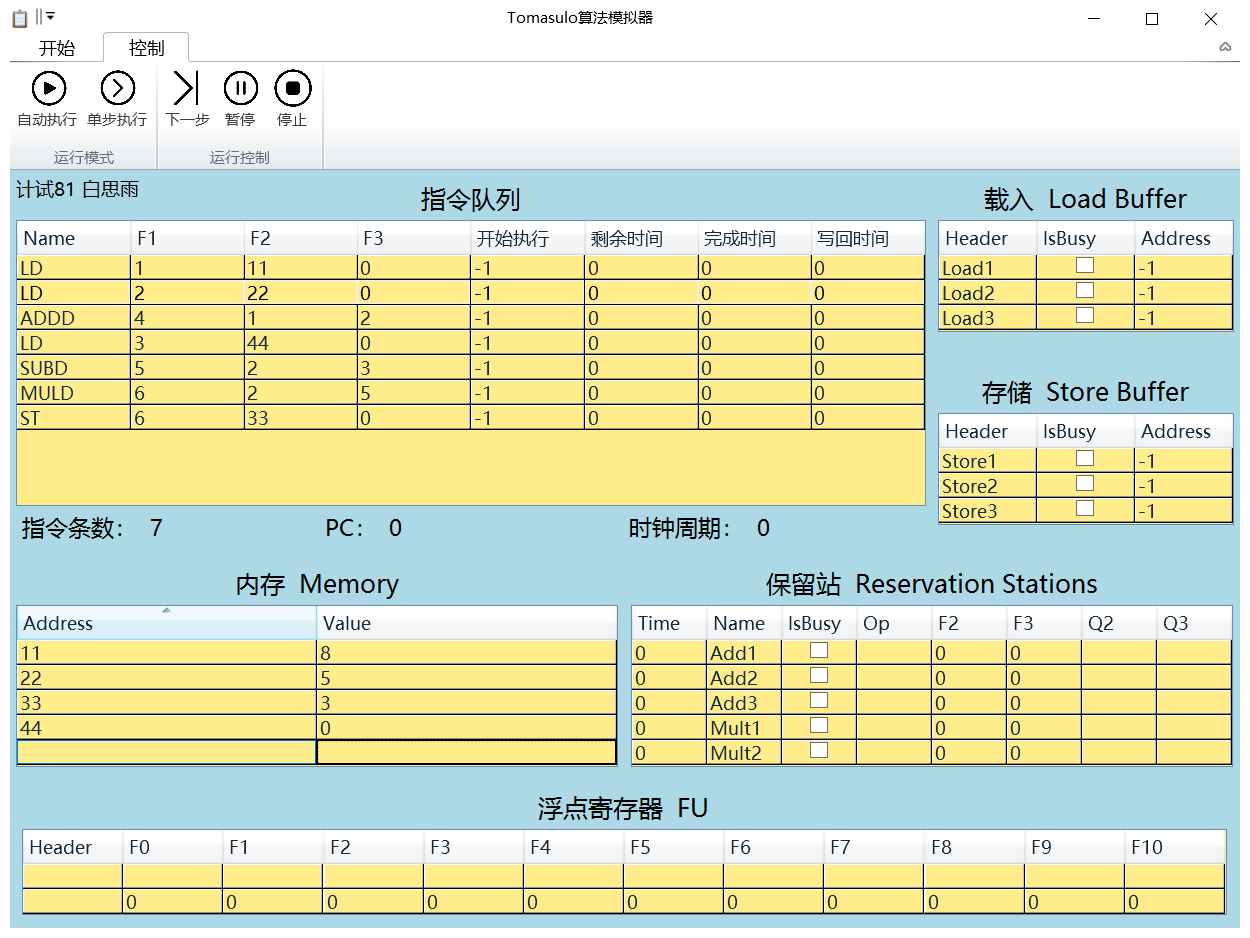
LD 3, 44

SUBD 5, 2, 3

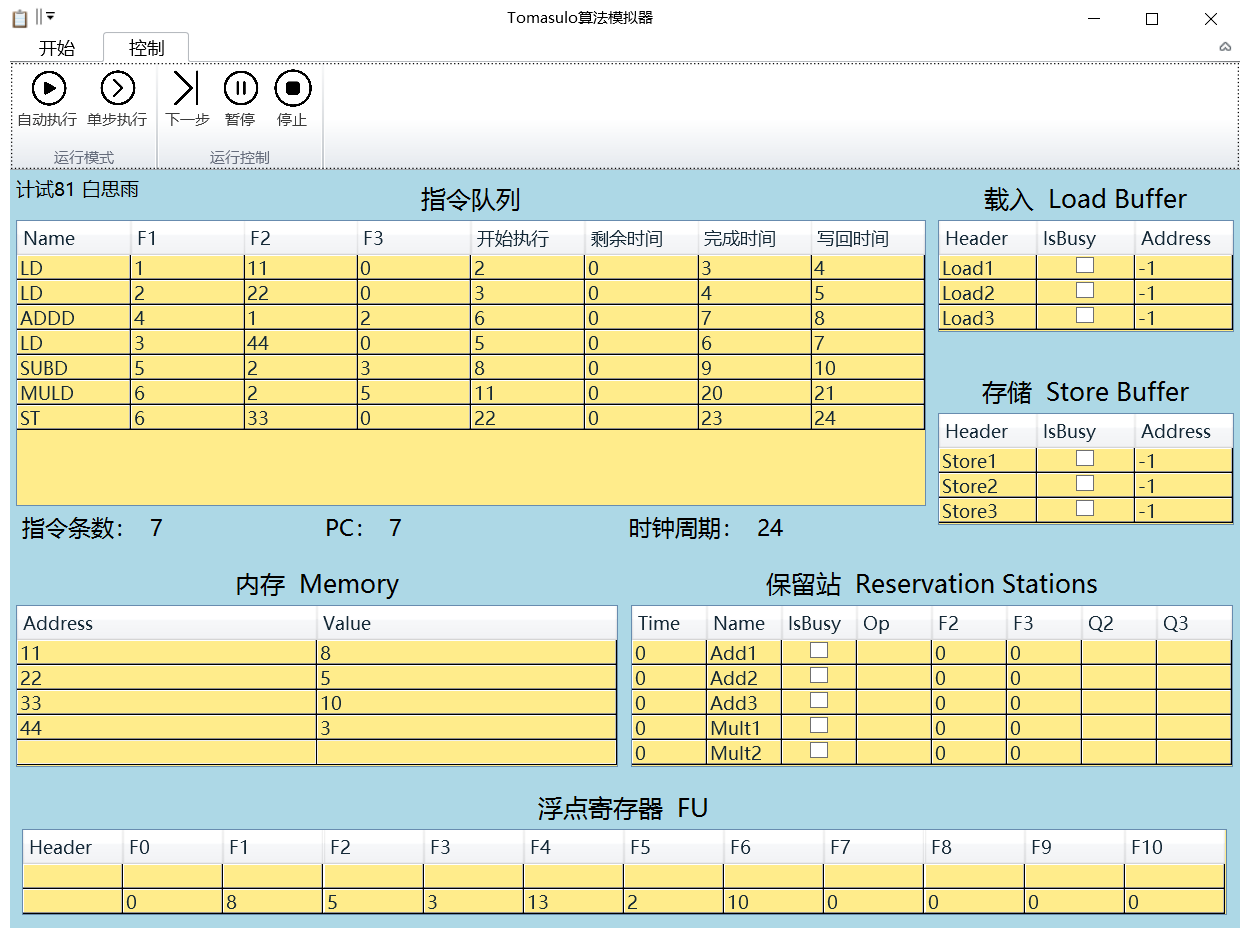
MULD 6, 2, 5

ST 6, 33

初始寄存器：

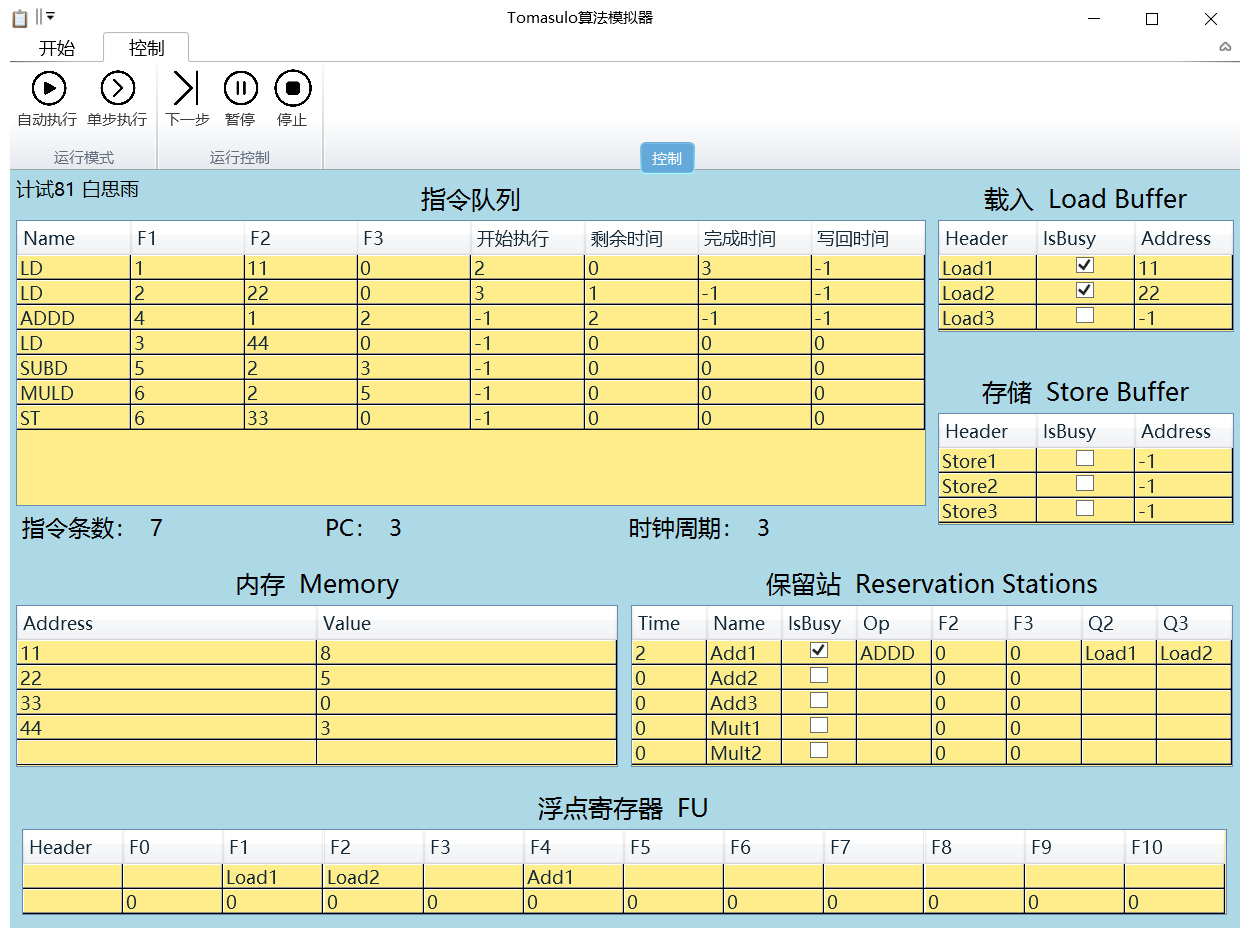


运行结果截图：

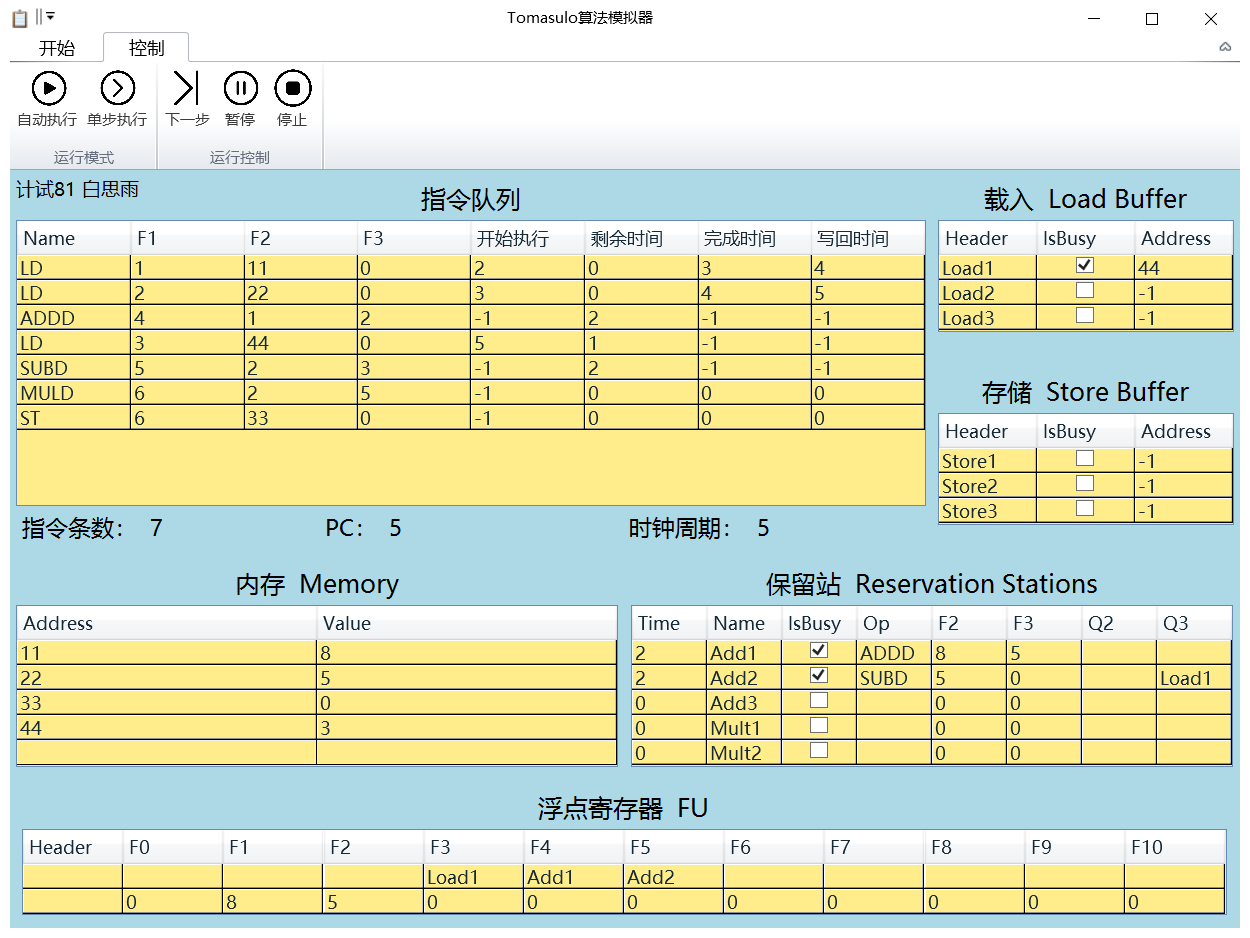


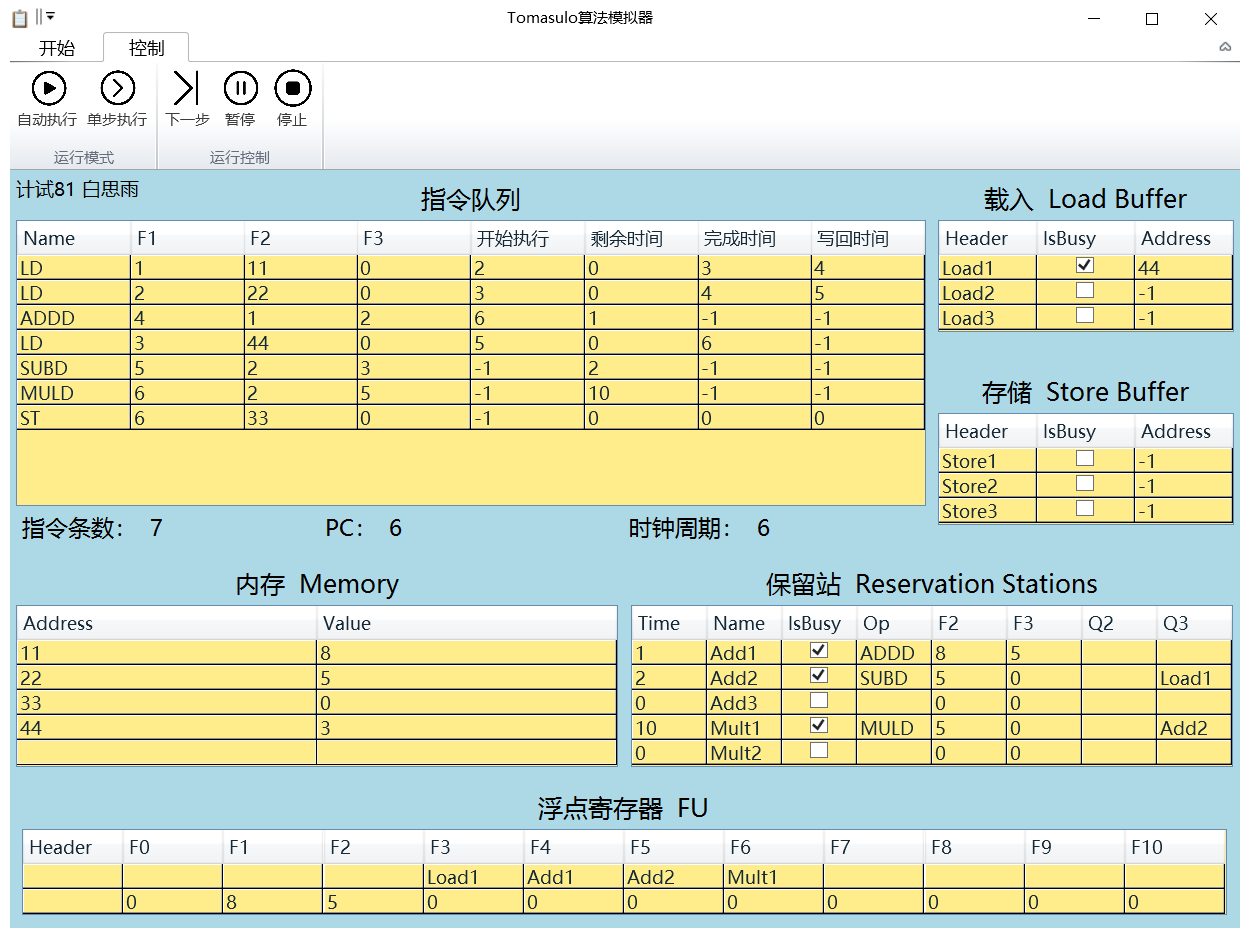
可见，第一、二步的LD 1, 11和LD 2, 22与第三步ADDD 4, 1, 2发生了RAW冲突，第三步延迟并放入保留站。第三步的开始执行时钟周期为6，而第四步开始执行时钟周期为5.

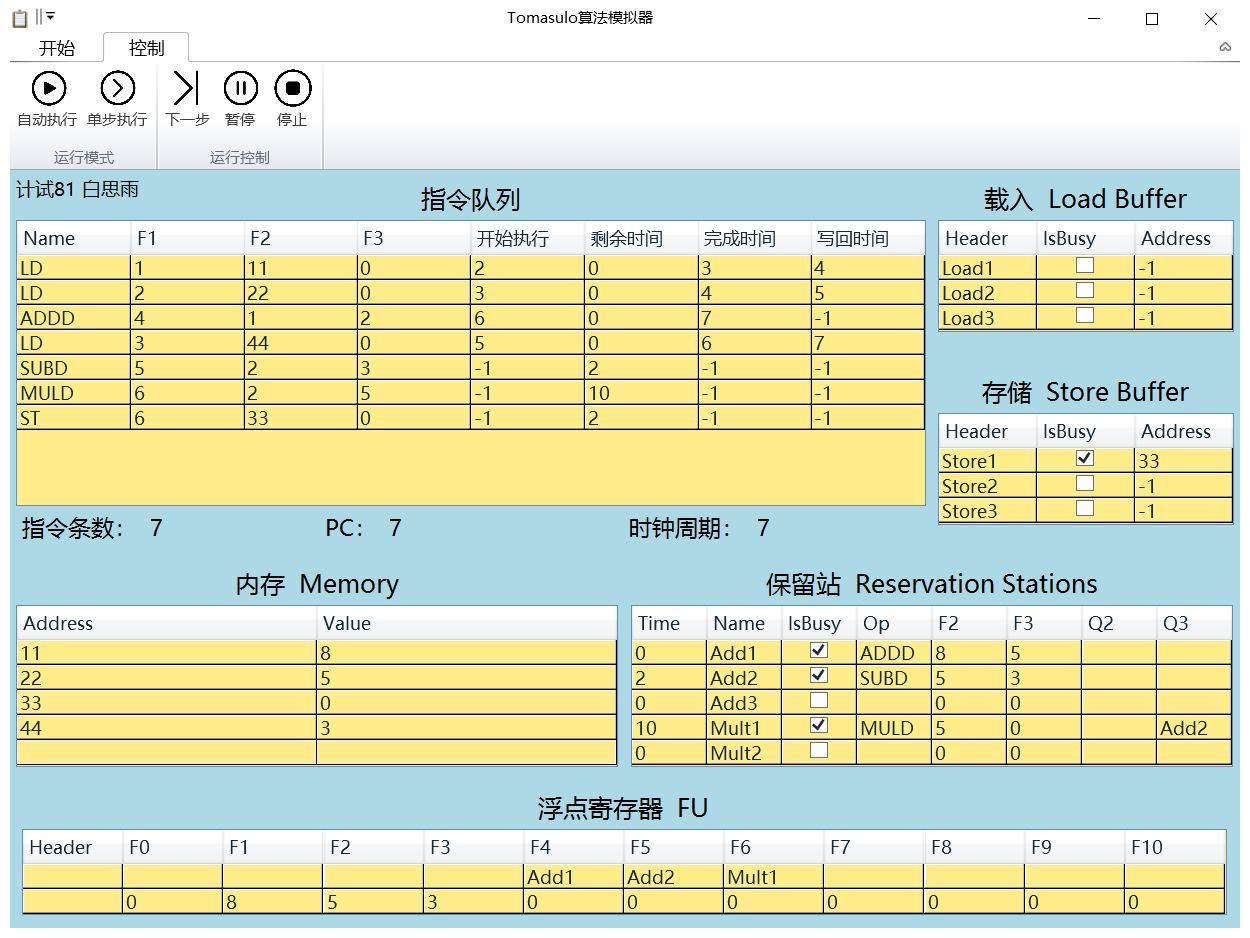
过程具体分析：



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。



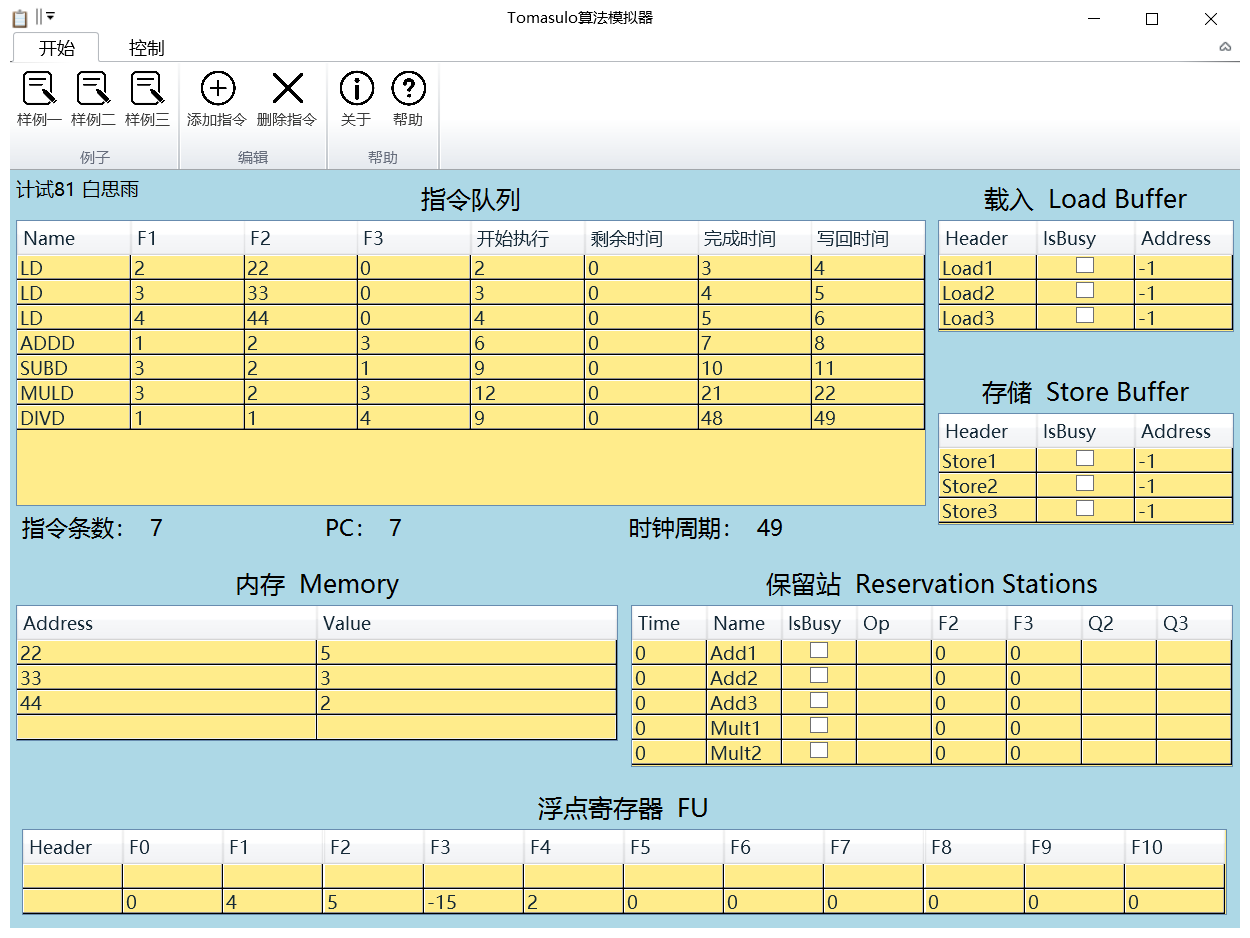




待前两步LD读取完成后，在保留站中的第三步开始执行，并写回F4寄存器，冲突解决。

1. 有至少一次的WAR冲突：

代码： 初始内存：

start:

LD 2, 22

LD 3, 33

LD 4, 44

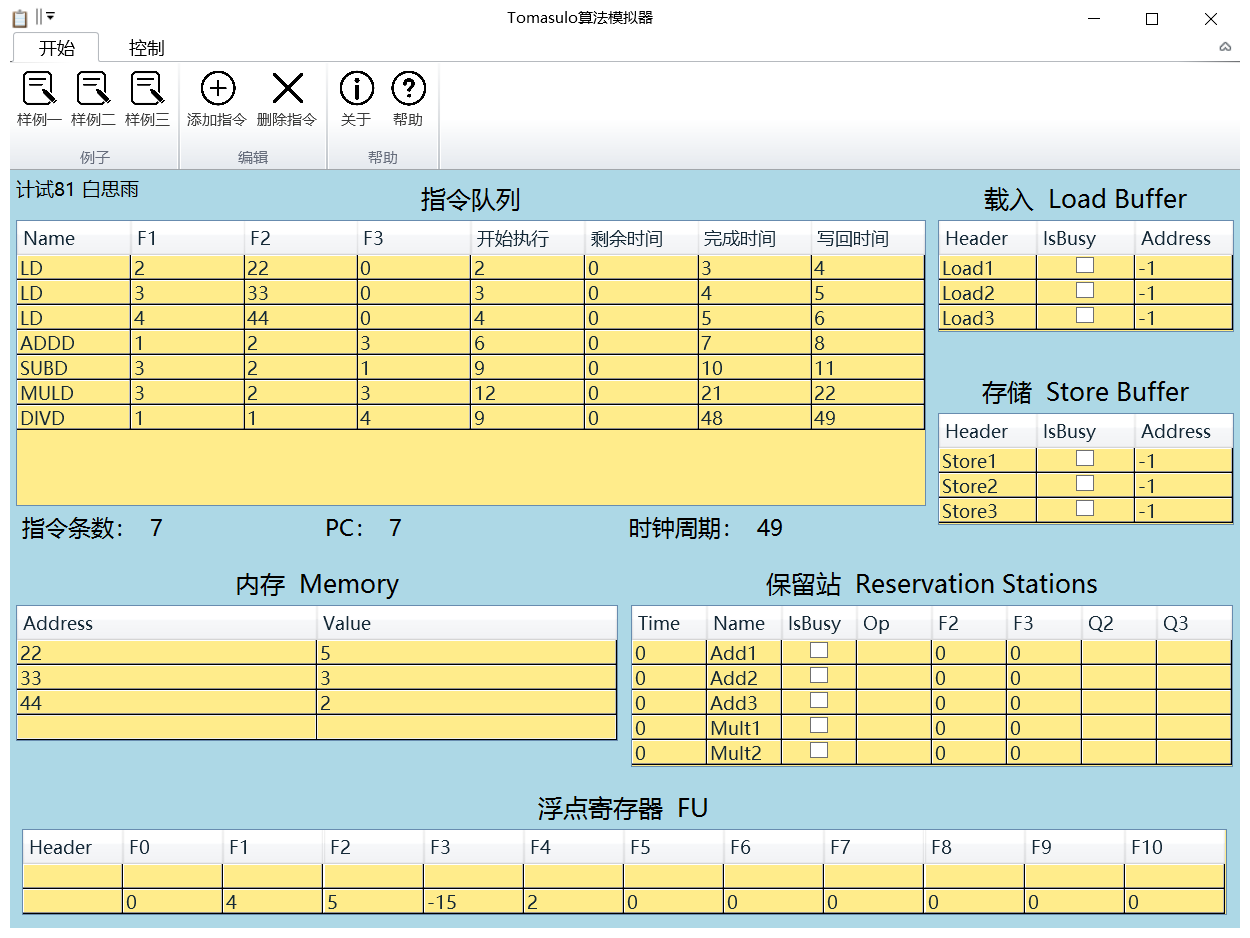
ADDD 1, 2, 3

SUBD 3, 2, 1

MULD 3, 2, 3

DIVD 1, 1, 4

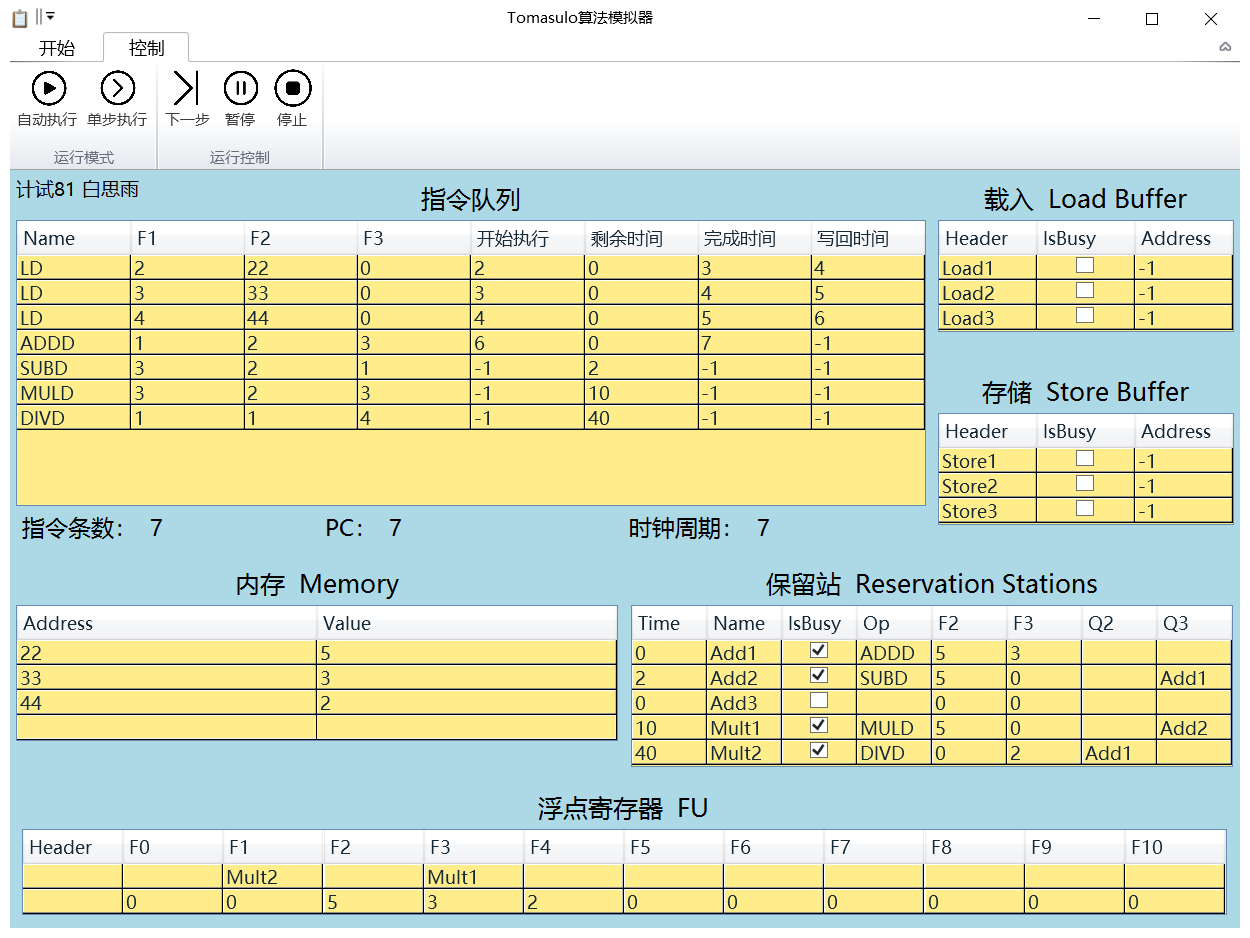
运行结果截图 ：



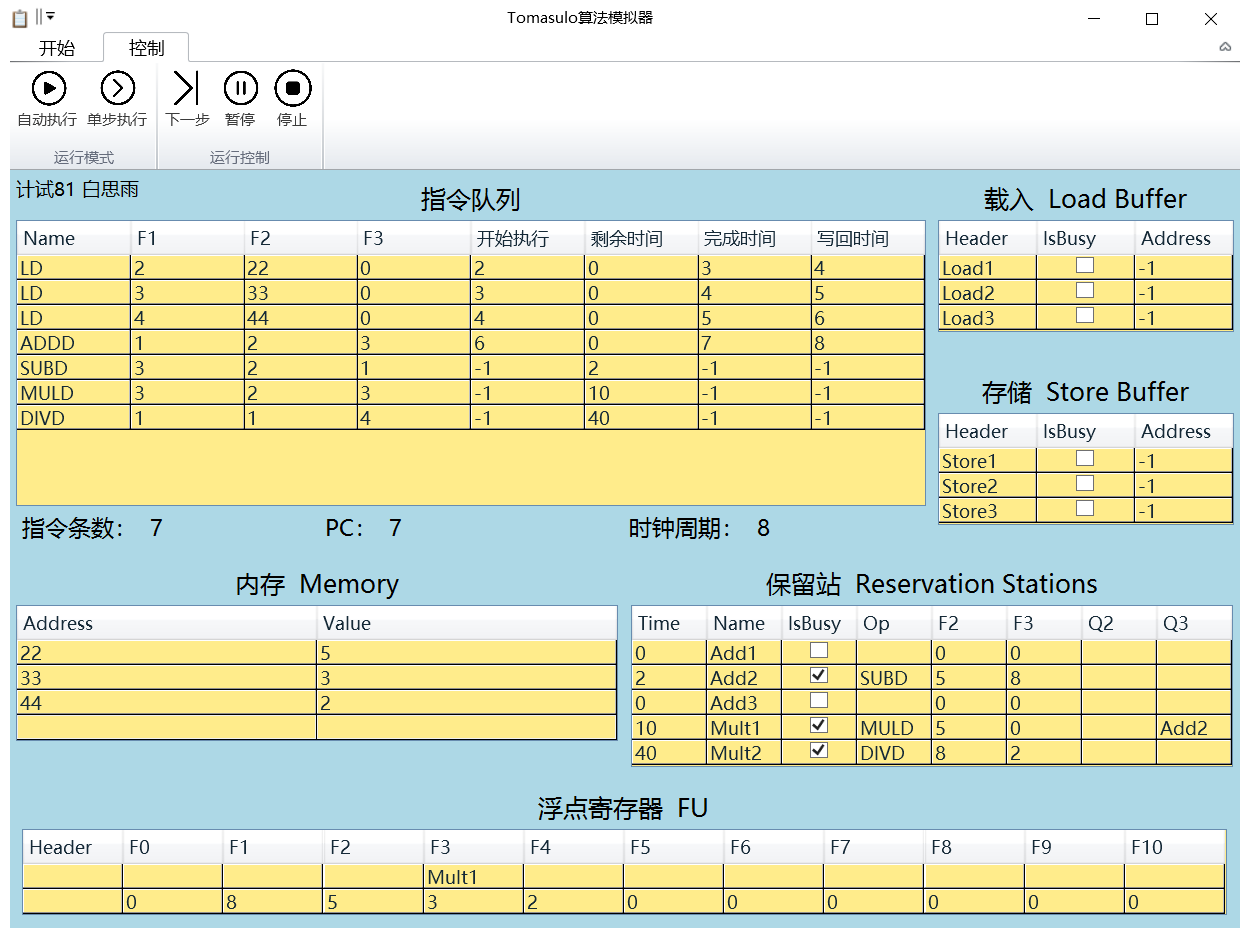
WAR冲突即为在前指令未读取寄存器数据时，后续指令先完成写寄存器操作。

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。

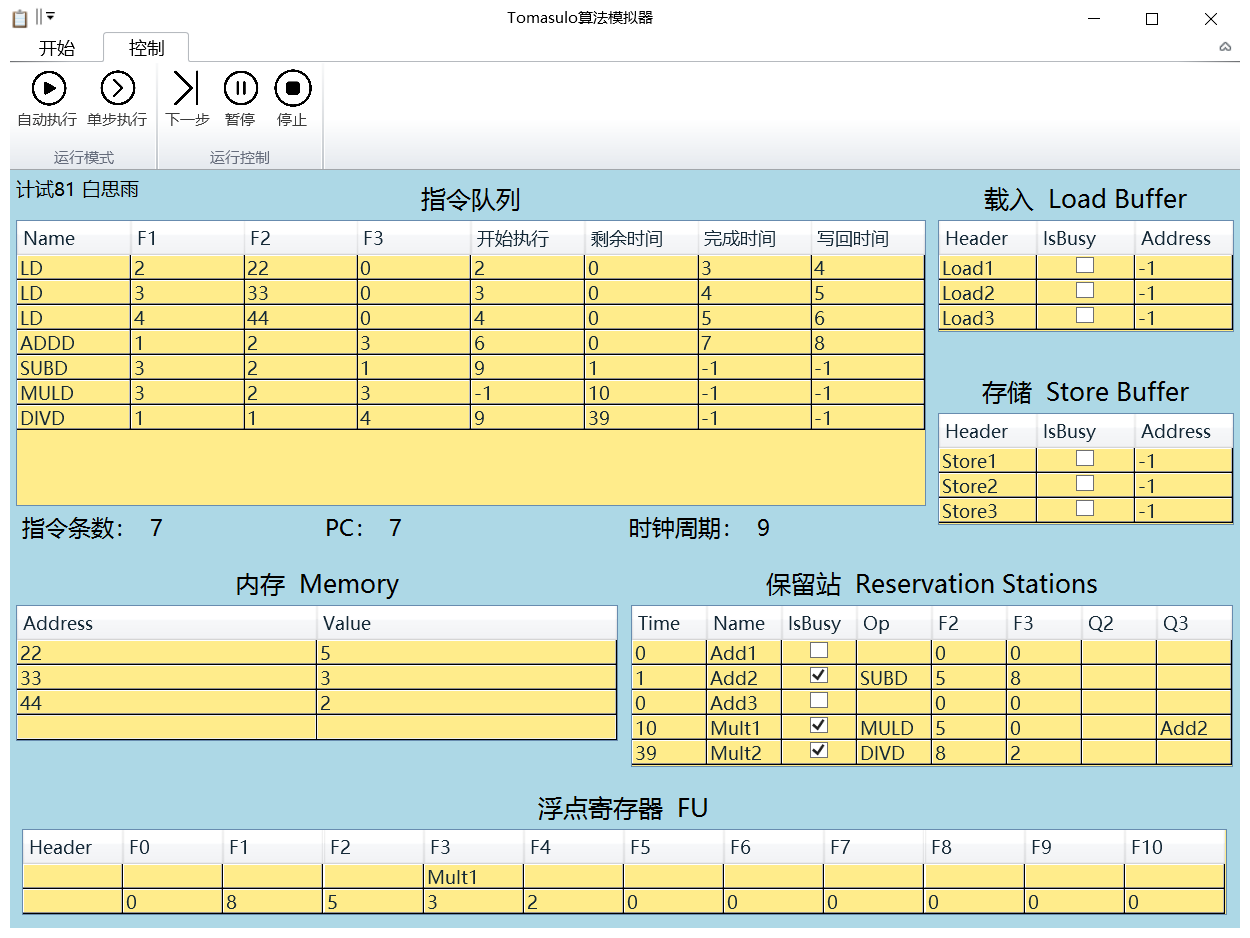
过程具体分析：



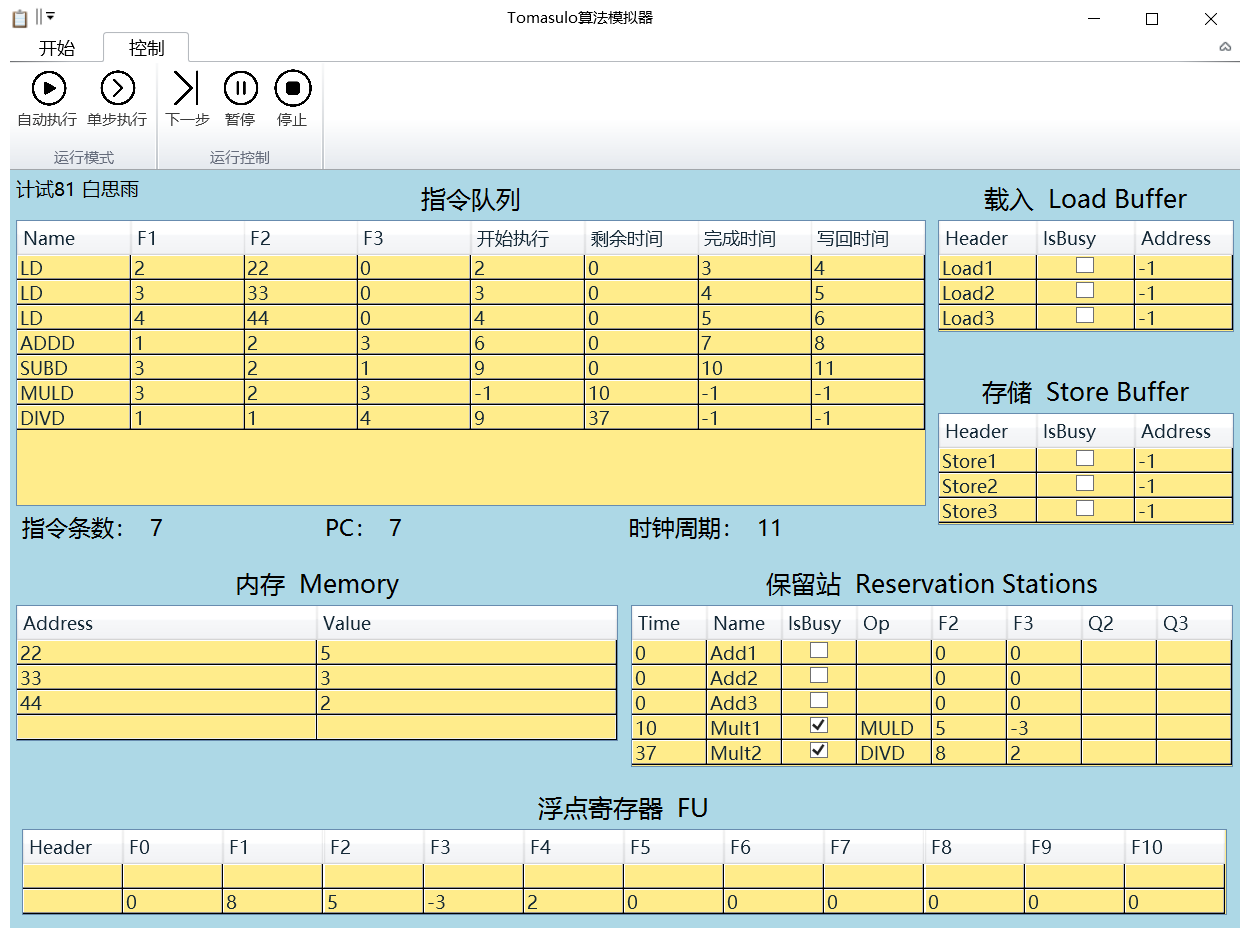
在第七时钟周期时，所有指令均已读取，并通过寄存器换名，存入保留站中。以此解决WAR冲突。



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。

实验感悟

本次实验与第一次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，比较考验学生较强的编程能力与创新思维。

通过本次实验，引用开源项目，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，受益匪浅！

十分感谢任课老师对本次实验的精心设计和安排！