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。

2．实验平台

自己设计一个流水线模拟器（简称：模拟器 A）

3．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:

```
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
```

初始内存：

内存 Memory

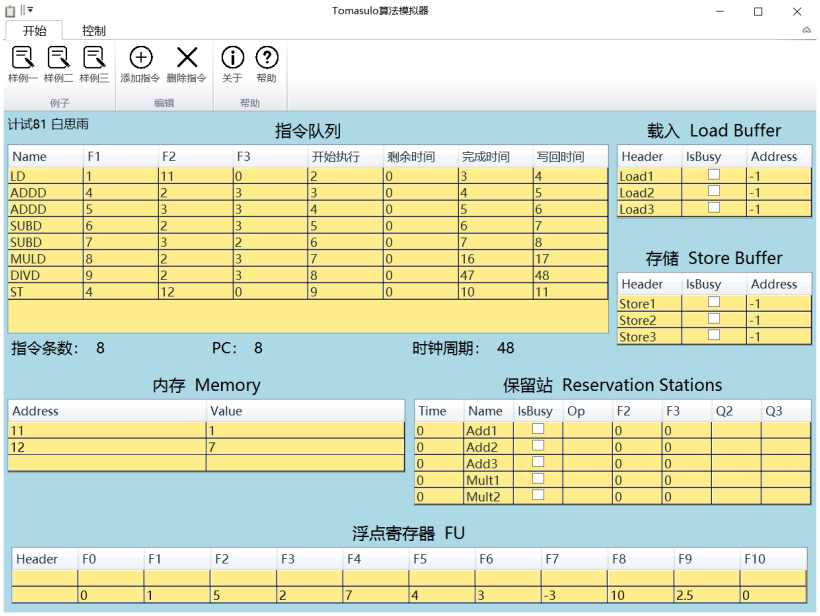
Address	Value
11	1
12	0

初始寄存器：

浮点寄存器 FU

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
	0	0	5	2	0	0	0	0	0	0	0

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。

② 有至少一次的 RAW 冲突：

start:

```
LD      F1,  11
LD      F2,  22
ADDD    F4,  F1,  F2
LD      F3,  44
SUBD    F5,  F2,  F3
MULD    F6,  F2,  F5
ST      F6,  33
```

初始内存：

内存 Memory

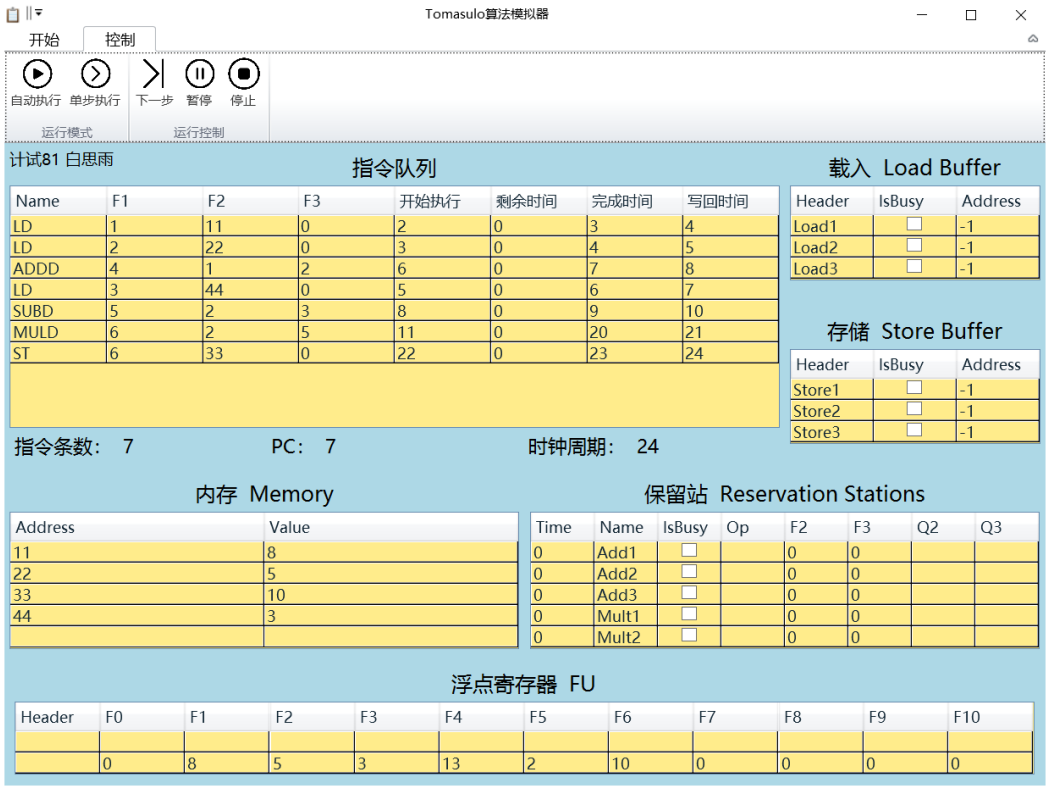
Address	Value
11	8
22	5
33	0
44	3

初始寄存器：

浮点寄存器 FU

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
	0	0	0	0	0	0	0	0	0	0	0

运行结果截图：



Name	F1	F2	F3	开始执行	剩余时间	完成时间	写回时间
LD	1	11	0	2	0	3	4
LD	2	22	0	3	0	4	5
ADDD	4	1	2	6	0	7	8
LD	3	44	0	5	0	6	7
SUBD	5	2	3	8	0	9	10
MULD	6	2	5	11	0	20	21
ST	6	33	0	22	0	23	24

Header	IsBusy	Address
Load1	☐	-1
Load2	☐	-1
Load3	☐	-1

Header	IsBusy	Address
Store1	☐	-1
Store2	☐	-1
Store3	☐	-1

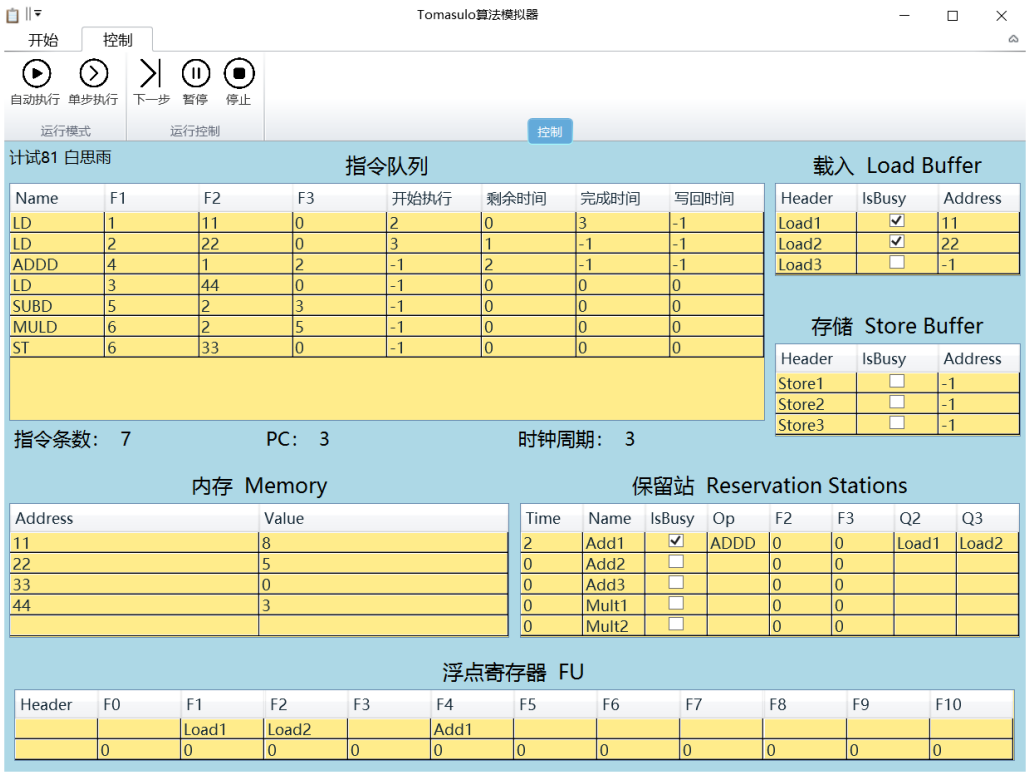
Address	Value
11	8
22	5
33	10
44	3

Time	Name	IsBusy	Op	F2	F3	Q2	Q3
0	Add1	☐		0	0		
0	Add2	☐		0	0		
0	Add3	☐		0	0		
0	Mult1	☐		0	0		
0	Mult2	☐		0	0		

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
	0	8	5	3	13	2	10	0	0	0	0

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：



Name	F1	F2	F3	开始执行	剩余时间	完成时间	写回时间
LD	1	11	0	2	0	3	-1
LD	2	22	0	3	1	-1	-1
ADDD	4	1	2	-1	2	-1	-1
LD	3	44	0	-1	0	0	0
SUBD	5	2	3	-1	0	0	0
MULD	6	2	5	-1	0	0	0
ST	6	33	0	-1	0	0	0

Header	IsBusy	Address
Load1	☑	11
Load2	☑	22
Load3	☐	-1

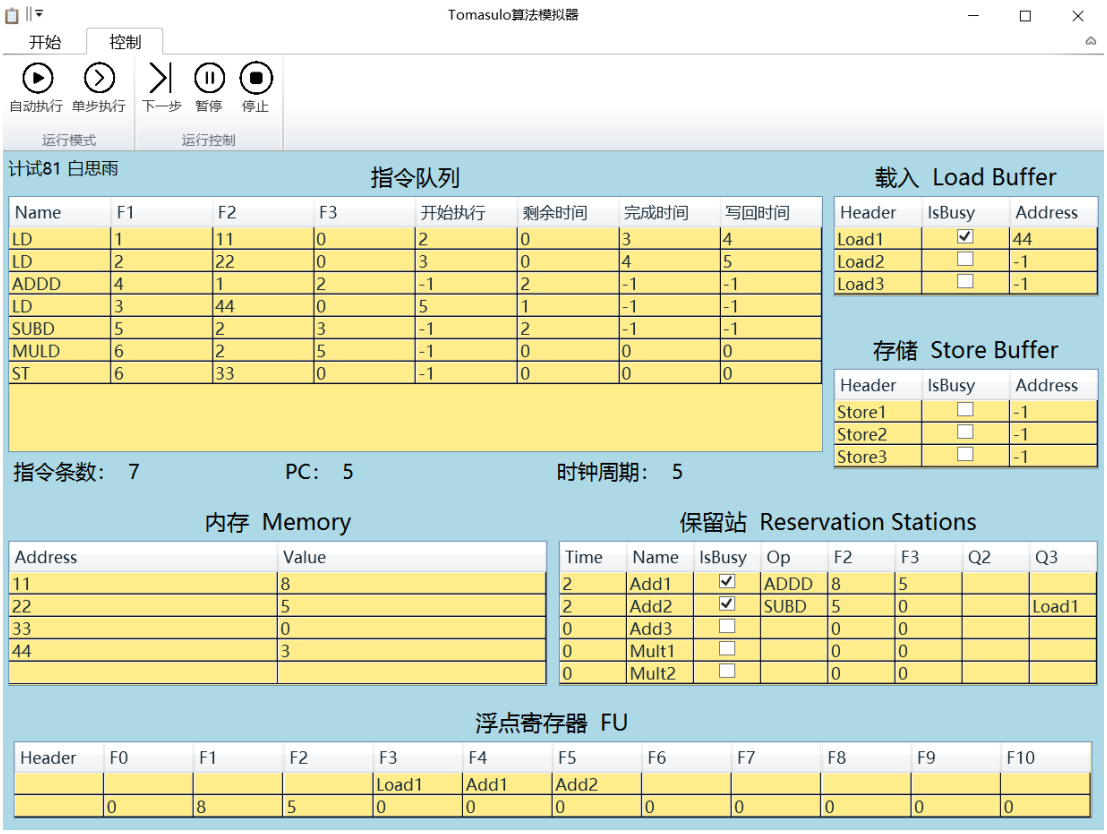
Header	IsBusy	Address
Store1	☐	-1
Store2	☐	-1
Store3	☐	-1

Address	Value
11	8
22	5
33	0
44	3

Time	Name	IsBusy	Op	F2	F3	Q2	Q3
2	Add1	☑	ADDD	0	0	Load1	Load2
0	Add2	☐		0	0		
0	Add3	☐		0	0		
0	Mult1	☐		0	0		
0	Mult2	☐		0	0		

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
		Load1	Load2		Add1						
	0	0	0	0	0	0	0	0	0	0	0

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。



Name	F1	F2	F3	开始执行	剩余时间	完成时间	写回时间
LD	1	11	0	2	0	3	4
LD	2	22	0	3	0	4	5
ADDD	4	1	2	-1	2	-1	-1
LD	3	44	0	5	1	-1	-1
SUBD	5	2	3	-1	2	-1	-1
MULD	6	2	5	-1	0	0	0
ST	6	33	0	-1	0	0	0

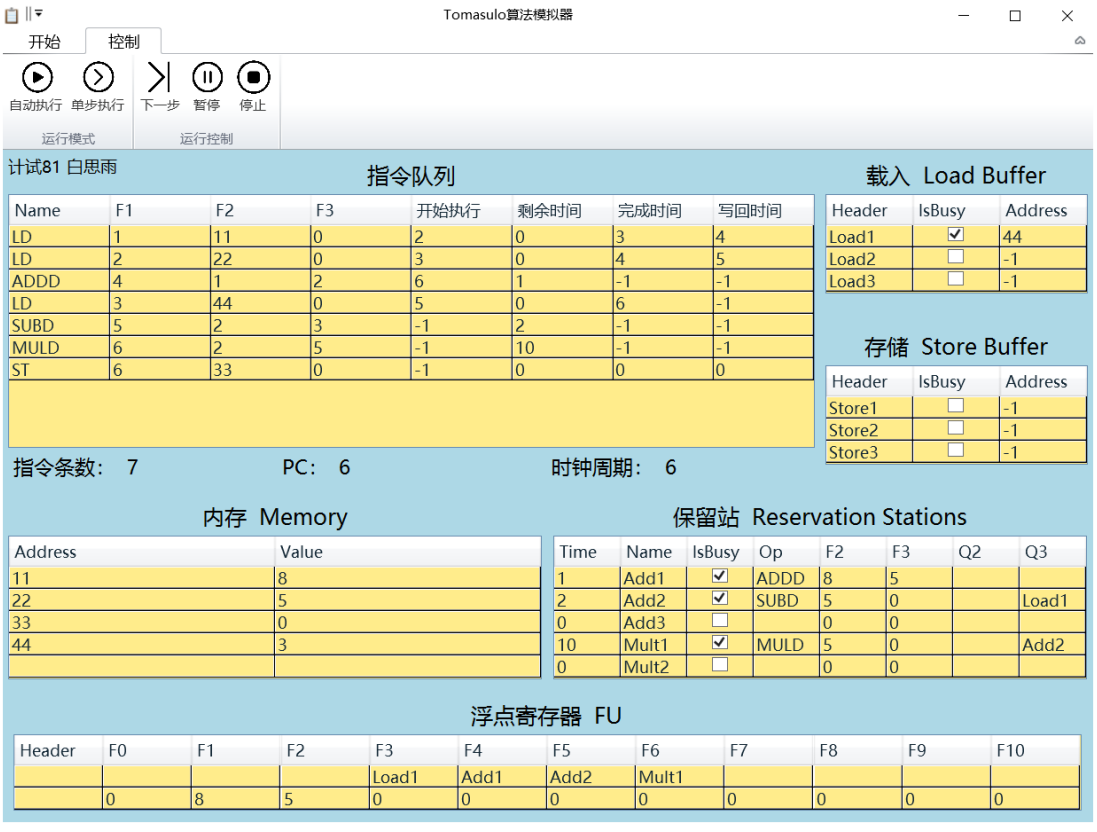
Header	IsBusy	Address
Load1	☑	44
Load2	☐	-1
Load3	☐	-1

Header	IsBusy	Address
Store1	☐	-1
Store2	☐	-1
Store3	☐	-1

Address	Value
11	8
22	5
33	0
44	3

Time	Name	IsBusy	Op	F2	F3	Q2	Q3
2	Add1	☑	ADDD	8	5		
2	Add2	☑	SUBD	5	0		Load1
0	Add3	☐		0	0		
0	Mult1	☐		0	0		
0	Mult2	☐		0	0		

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
				Load1	Add1	Add2					
	0	8	5	0	0	0	0	0	0	0	0



Name	F1	F2	F3	开始执行	剩余时间	完成时间	写回时间
LD	1	11	0	2	0	3	4
LD	2	22	0	3	0	4	5
ADDD	4	1	2	6	1	-1	-1
LD	3	44	0	5	0	6	-1
SUBD	5	2	3	-1	2	-1	-1
MULD	6	2	5	-1	10	-1	-1
ST	6	33	0	-1	0	0	0

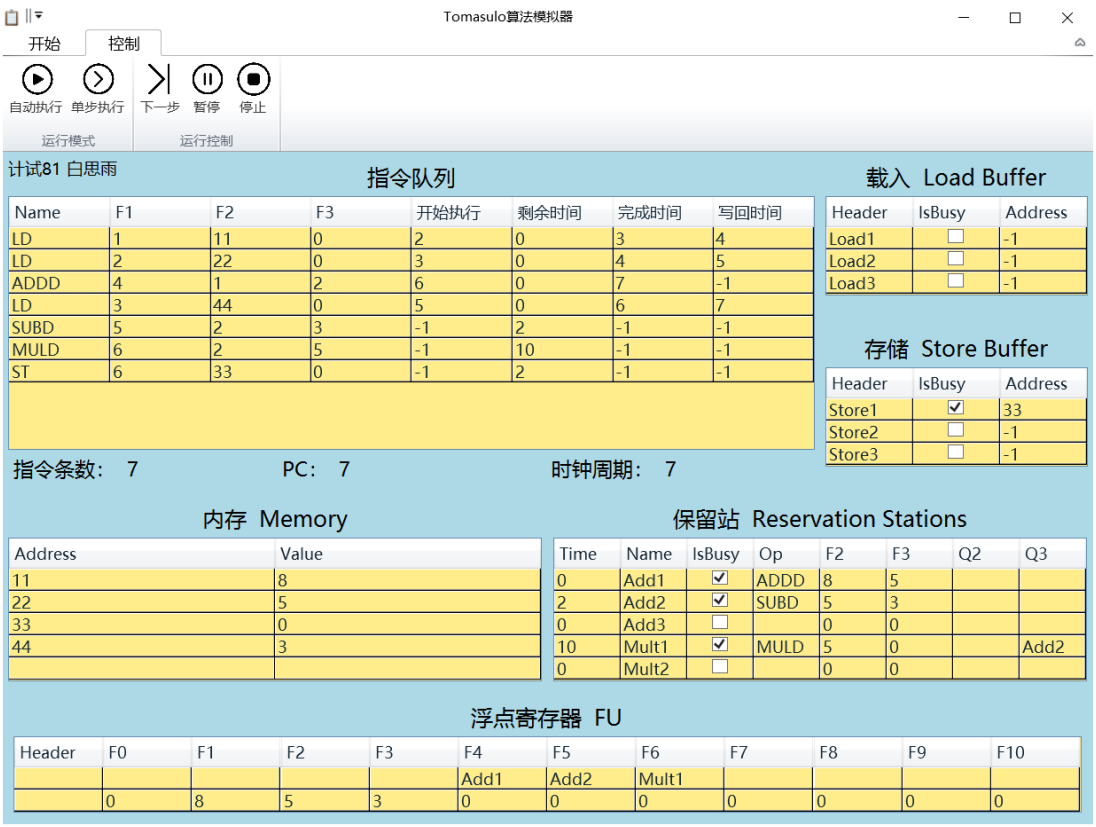
Header	IsBusy	Address
Load1	☑	44
Load2	☐	-1
Load3	☐	-1

Header	IsBusy	Address
Store1	☐	-1
Store2	☐	-1
Store3	☐	-1

Address	Value
11	8
22	5
33	0
44	3

Time	Name	IsBusy	Op	F2	F3	Q2	Q3
1	Add1	☑	ADDD	8	5		
2	Add2	☑	SUBD	5	0		Load1
0	Add3	☐		0	0		
10	Mult1	☑	MULD	5	0		Add2
0	Mult2	☐		0	0		

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
				Load1	Add1	Add2	Mult1				
	0	8	5	0	0	0	0	0	0	0	0



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。

③ 有至少一次的 WAR 冲突：

代码：

start:

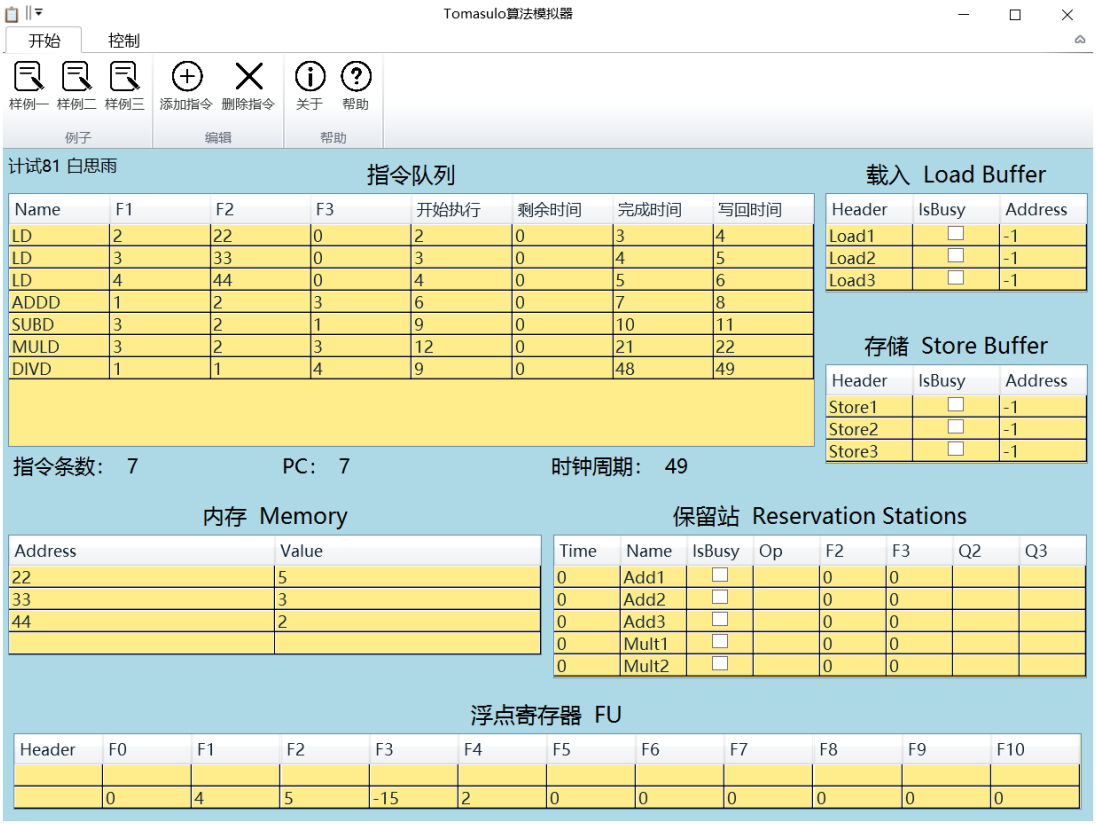
```
LD      F2,  22
LD      F3,  33
LD      F4,  44
ADDD    F1,  F2,  F3
SUBD    F3,  F2,  F1
MULD    F3,  F2,  F3
DIVD    F1,  F1,  F4
```

初始内存：

内存 Memory

Address	Value
22	5
33	3
44	2

运行结果截图 ：



Name	F1	F2	F3	开始执行	剩余时间	完成时间	写回时间
LD	2	22	0	2	0	3	4
LD	3	33	0	3	0	4	5
LD	4	44	0	4	0	5	6
ADDD	1	2	3	6	0	7	8
SUBD	3	2	1	9	0	10	11
MULD	3	2	3	12	0	21	22
DIVD	1	1	4	9	0	48	49

Header	IsBusy	Address
Load1	☐	-1
Load2	☐	-1
Load3	☐	-1

Header	IsBusy	Address
Store1	☐	-1
Store2	☐	-1
Store3	☐	-1

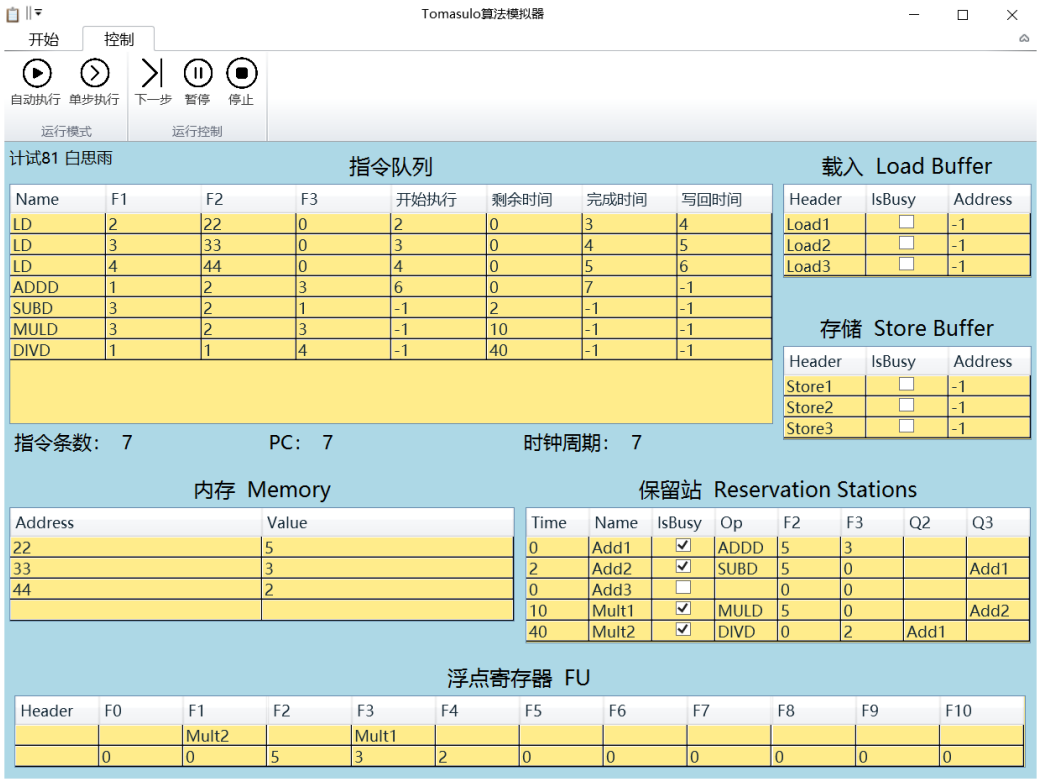
Address	Value
22	5
33	3
44	2

Time	Name	IsBusy	Op	F2	F3	Q2	Q3
0	Add1	☐		0	0		
0	Add2	☐		0	0		
0	Add3	☐		0	0		
0	Mult1	☐		0	0		
0	Mult2	☐		0	0		

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
	0	4	5	-15	2	0	0	0	0	0	0

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:



Name	F1	F2	F3	开始执行	剩余时间	完成时间	写回时间
LD	2	22	0	2	0	3	4
LD	3	33	0	3	0	4	5
LD	4	44	0	4	0	5	6
ADDD	1	2	3	6	0	7	-1
SUBD	3	2	1	-1	2	-1	-1
MULD	3	2	3	-1	10	-1	-1
DIVD	1	1	4	-1	40	-1	-1

Header	IsBusy	Address
Load1	☐	-1
Load2	☐	-1
Load3	☐	-1

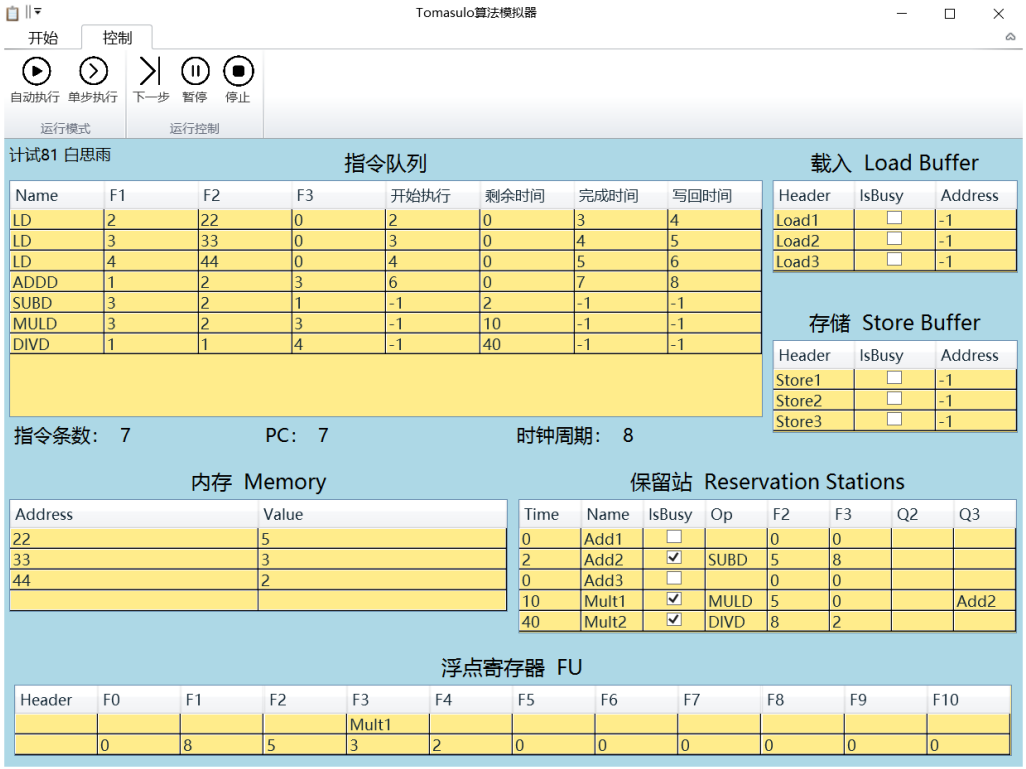
Header	IsBusy	Address
Store1	☐	-1
Store2	☐	-1
Store3	☐	-1

Address	Value
22	5
33	3
44	2

Time	Name	IsBusy	Op	F2	F3	Q2	Q3
0	Add1	☑	ADDD	5	3		
2	Add2	☑	SUBD	5	0		Add1
0	Add3	☐		0	0		
10	Mult1	☑	MULD	5	0		Add2
40	Mult2	☑	DIVD	0	2	Add1	

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
		Mult2		Mult1							
	0	0	5	3	2	0	0	0	0	0	0

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。



Name	F1	F2	F3	开始执行	剩余时间	完成时间	写回时间
LD	2	22	0	2	0	3	4
LD	3	33	0	3	0	4	5
LD	4	44	0	4	0	5	6
ADDD	1	2	3	6	0	7	8
SUBD	3	2	1	-1	2	-1	-1
MULD	3	2	3	-1	10	-1	-1
DIVD	1	1	4	-1	40	-1	-1

Header	IsBusy	Address
Load1	☐	-1
Load2	☐	-1
Load3	☐	-1

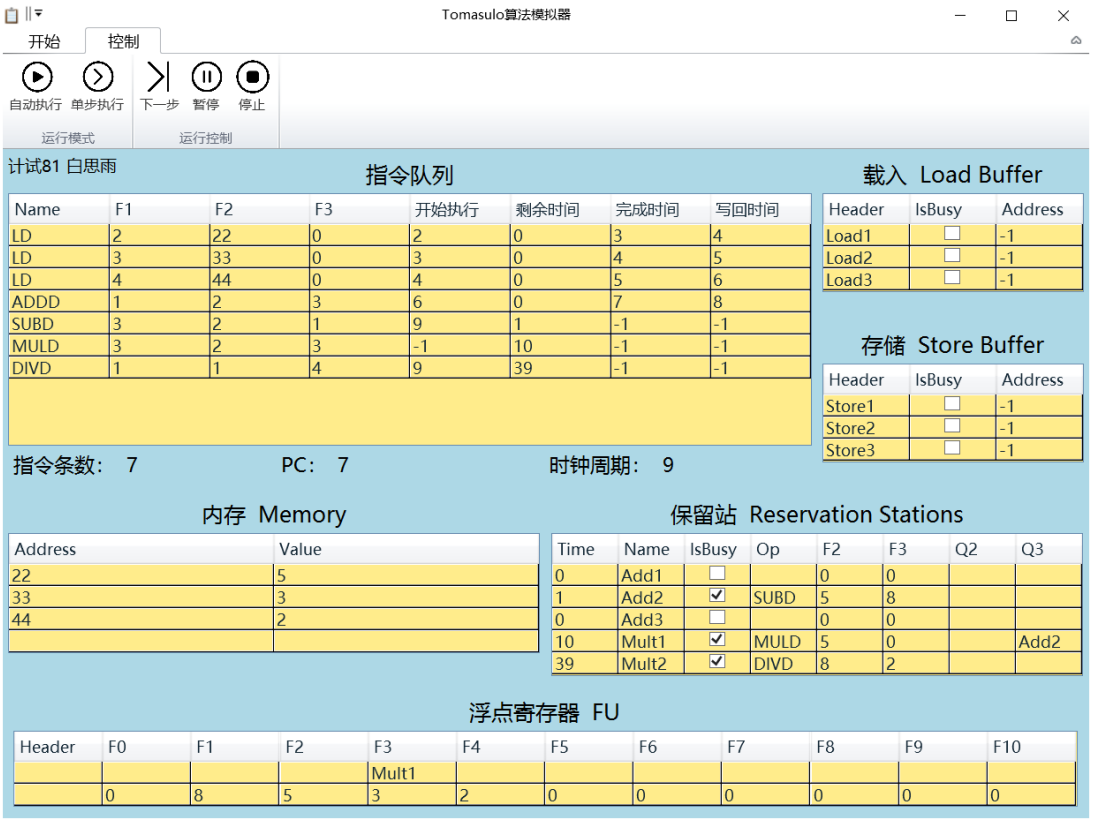
Header	IsBusy	Address
Store1	☐	-1
Store2	☐	-1
Store3	☐	-1

Address	Value
22	5
33	3
44	2

Time	Name	IsBusy	Op	F2	F3	Q2	Q3
0	Add1	☐		0	0		
2	Add2	☑	SUBD	5	8		
0	Add3	☐		0	0		
10	Mult1	☑	MULD	5	0		Add2
40	Mult2	☑	DIVD	8	2		

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
				Mult1							
	0	8	5	3	2	0	0	0	0	0	0

在第八时钟周期时，第四步 ADDD 已执行完成，Add1 变为对应数值。



Name	F1	F2	F3	开始执行	剩余时间	完成时间	写回时间
LD	2	22	0	2	0	3	4
LD	3	33	0	3	0	4	5
LD	4	44	0	4	0	5	6
ADDD	1	2	3	6	0	7	8
SUBD	3	2	1	9	1	-1	-1
MULD	3	2	3	-1	10	-1	-1
DIVD	1	1	4	9	39	-1	-1

Header	IsBusy	Address
Load1	☐	-1
Load2	☐	-1
Load3	☐	-1

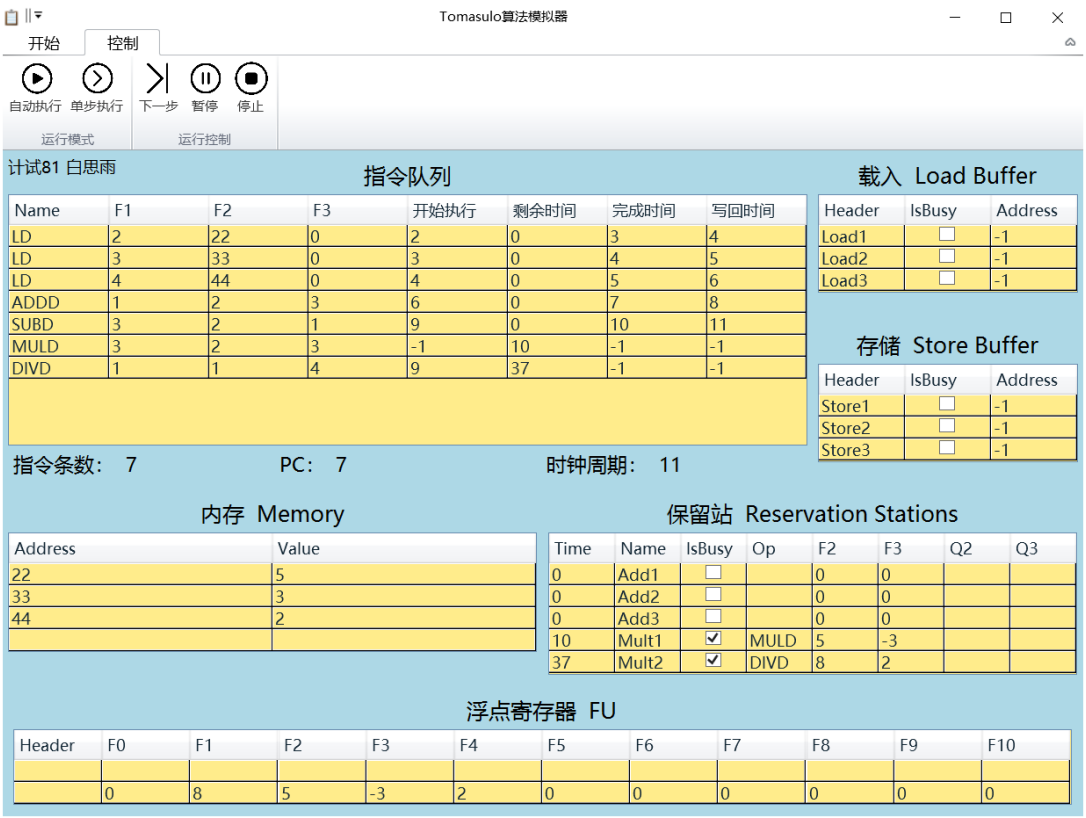
Header	IsBusy	Address
Store1	☐	-1
Store2	☐	-1
Store3	☐	-1

Address	Value
22	5
33	3
44	2

Time	Name	IsBusy	Op	F2	F3	Q2	Q3
0	Add1	☐		0	0		
1	Add2	☑	SUBD	5	8		
0	Add3	☐		0	0		
10	Mult1	☑	MULD	5	0		Add2
39	Mult2	☑	DIVD	8	2		

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
				Mult1							
	0	8	5	3	2	0	0	0	0	0	0

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。



Name	F1	F2	F3	开始执行	剩余时间	完成时间	写回时间
LD	2	22	0	2	0	3	4
LD	3	33	0	3	0	4	5
LD	4	44	0	4	0	5	6
ADDD	1	2	3	6	0	7	8
SUBD	3	2	1	9	0	10	11
MULD	3	2	3	-1	10	-1	-1
DIVD	1	1	4	9	37	-1	-1

Header	IsBusy	Address
Load1	☐	-1
Load2	☐	-1
Load3	☐	-1

Header	IsBusy	Address
Store1	☐	-1
Store2	☐	-1
Store3	☐	-1

Address	Value
22	5
33	3
44	2

Time	Name	IsBusy	Op	F2	F3	Q2	Q3
0	Add1	☐		0	0		
0	Add2	☐		0	0		
0	Add3	☐		0	0		
10	Mult1	☑	MULD	5	-3		
37	Mult2	☑	DIVD	8	2		

Header	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
	0	8	5	-3	2	0	0	0	0	0	0

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！