**《计算机原理与体系结构》实验报告**

**学号：16300240023 姓名：杨芗琳 学院：计算机科学技术学院 完成时间：2018.6.1**

1. 实验名称

设计实现Tomasulo算法

1. 实验目的

2.1 熟悉指令的动态调度方法;

2.2 熟悉 Tomasulo 算法；熟悉基于 Tomasulo 算法的前瞻执行算法

1. 实验过程

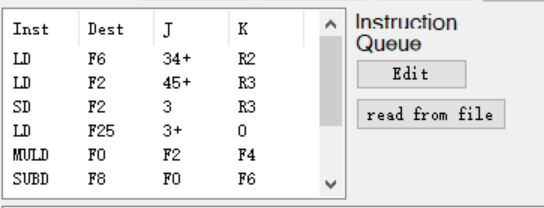
3.2 编写 Tomasulo 算法的演示程序，包括如下功能：

代码中包含分支指令，再执行分支指令时，假定使用流水线暂停技术：分支指令没有执行完毕，不允许下一条指令流出。

1. 读取以文件形式提供的指令序列，例如 program.txt

**思路**：按钮触发，一行一行读指令，以字符串的形式，每一行就新建一条指令

**截图**：



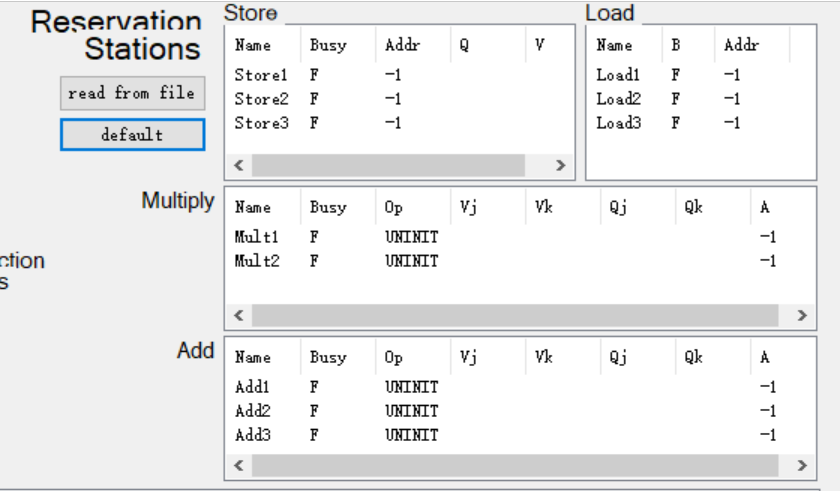
1. 读取以文件形式提供的部件配置表，例如 parts.txt

**思路**：按钮触发，以字符串的形式读入，将延迟周期和数量分别转成int类型；修改了reservation新建的传入参数，增加了延迟周期int delay，在RunExecution函数里面加在cycletocomplete体现延迟。

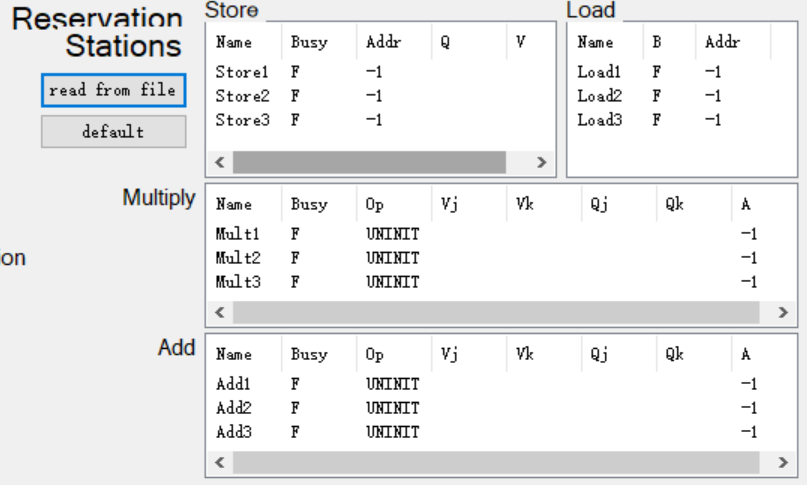
原本Reservation类里面没有Clear操作，所以重复读入会出错，所以在Reservation类里新加了一个ClearReservstion（）函数，重复读入之前先清空；

**截图**：

默认值：

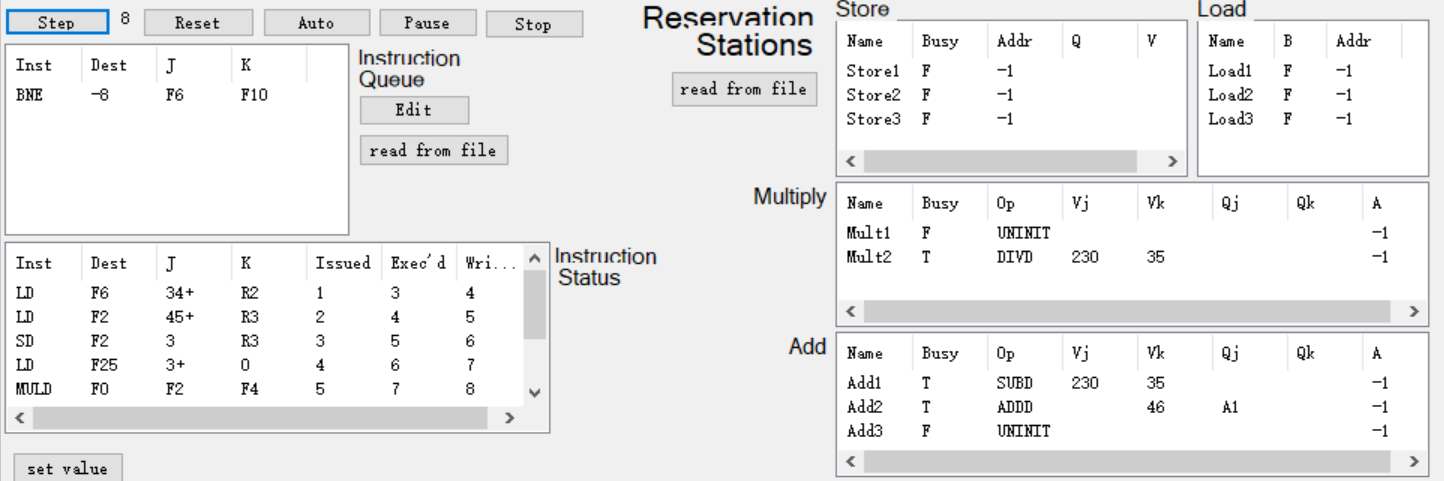


读入值：

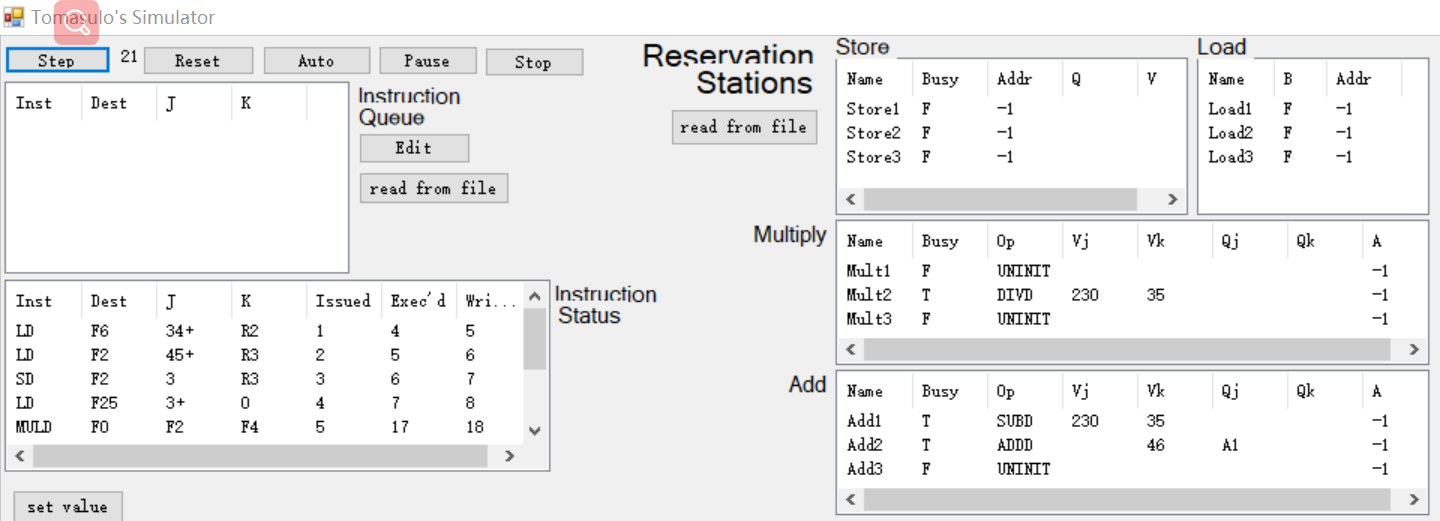


延迟实现：

默认值：



读入值：



可以看到延迟周期体现在延迟执行，满足读入配置标准。

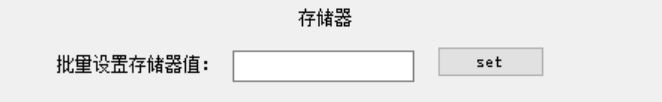
1. 设置初始存储器的值

**思路**：存储器在这个模拟器中体现为FloatingPointMemoryArray类，初始化时新建64个，并给每个存储器设置一个值。

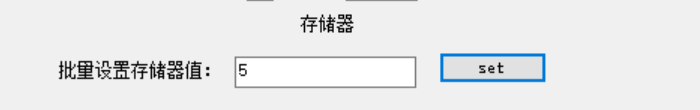
输入：新建一个Form3类，设置两个label用于提示信息输入，一个textbox用于用户输入设置的值，一个button用于触发存储器值修改的事件。当点击button时，textbox中的值作为修改值传入；

修改：在主界面Form1中加入新的函数SetFloatingPointMemory(float value),用一个for循环对每一个值进行修改，调用FloatingPointMemoryArray类的Set函数，当存储器值修改这个这个事件被触发后，调用这个函数，textbox中的值作为传入的参数；

**截图**：



效果：



1. 设置寄存器的值，可以批量设置或单个设置寄存器的值，包括浮点寄存器和整数寄存器。

**思路**：和修改存储器的想法类似。浮点寄存器体现为FloatingPointRegister类，通用寄存器体现为IntegerRegister类，在Form3界面中用label提示信息输入，textbox作为参数传入，button触发修改事件。同时在主界面中加入单个修改和批量修改的函数，都是调用类里的Set函数修改当前值。

除此之外，设置结束后需要调用update函数更新列表。

**截图：**



效果图：

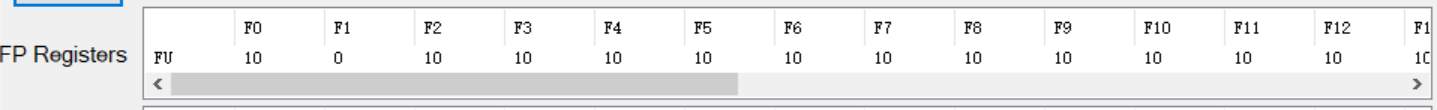
1. 批量设置浮点寄存器





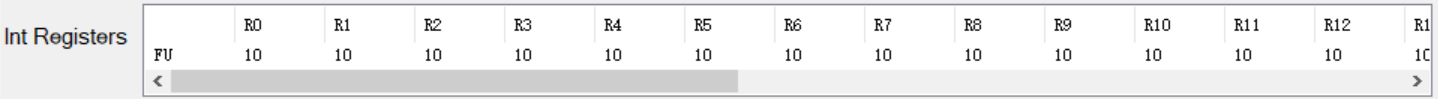
1. 单个设置浮点寄存器



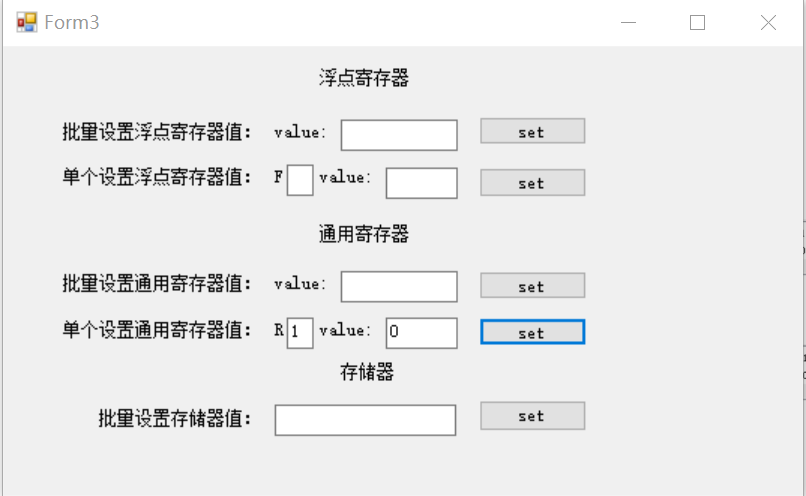


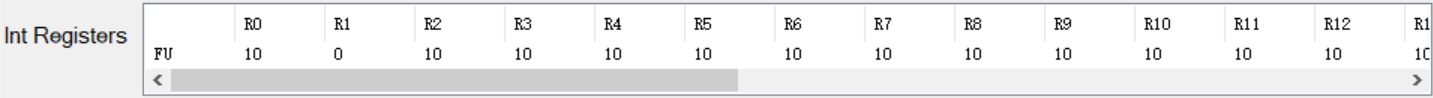
1. 批量设置通用寄存器





1. 单个设置通用寄存器





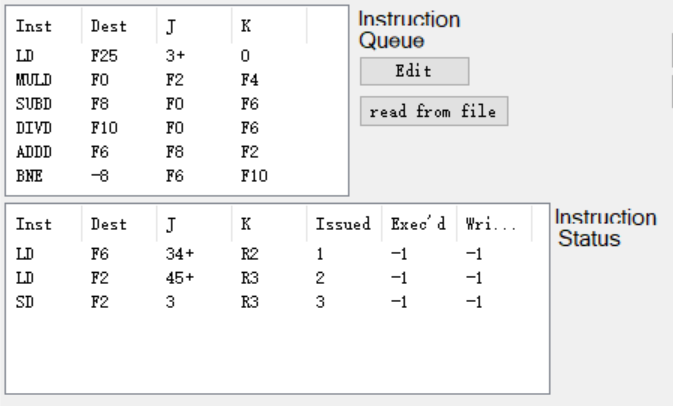
1. 界面上显示

指令队列、指令状态、保留站、寄存器状态

指令队列：已执行完毕的指令不显示。

**思路**：原有功能，无需修改。指令队列在模拟器中体现为InstructionUnit类，当流入时，把现有指令删除，同时update指令列表，从而满足已执行的指令不再出现。

**截图**：



指令状态表：显示所有已流出的指令状态。

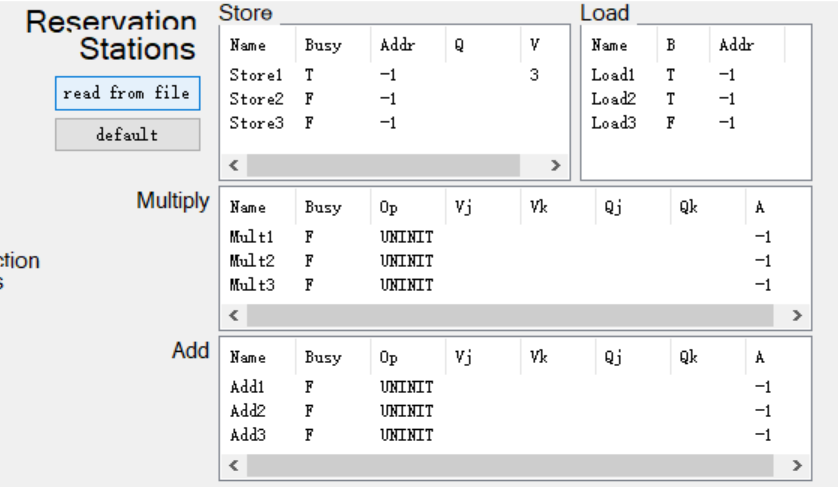
**思路**：同指令队列

**截图：**同指令队列

保留站：显示所有 Load,Store,乘法部件，加法部件的保留站信息。

**思路**：原有功能，无需修改。在模拟器中体现为Reservation类，Form1中定义了5个station，每个时钟周期下，都要update保留站状态，实现题目要求。

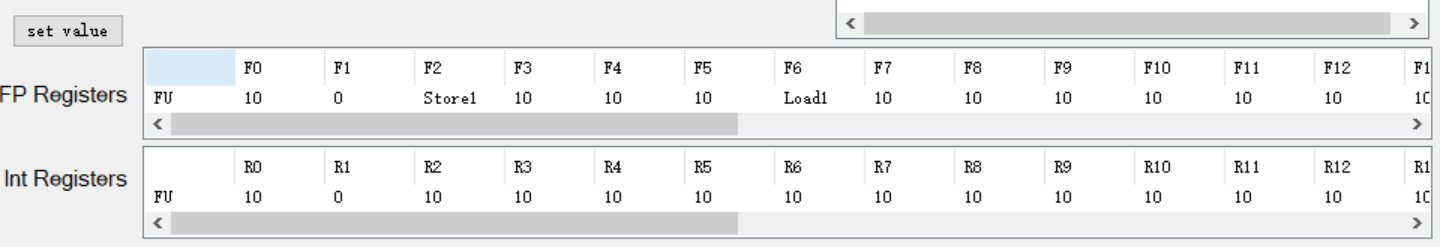
**截图：**



寄存器状态：

**思路**：已有功能，无需修改。两组寄存器在模拟器中体现为FloatingPointRegister类和IntegerRegister类，每过一个时钟周期都需要update。

**截图**：

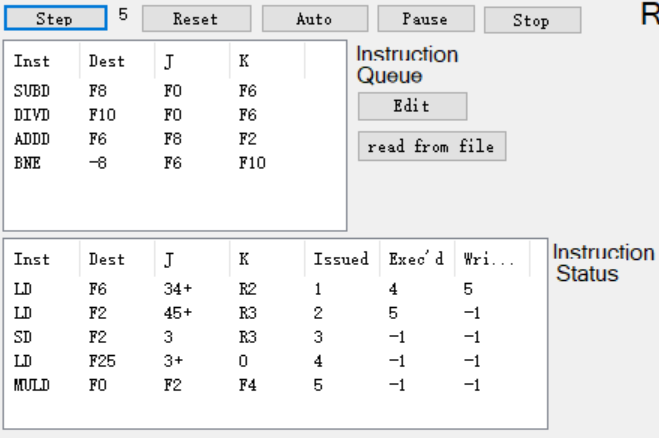


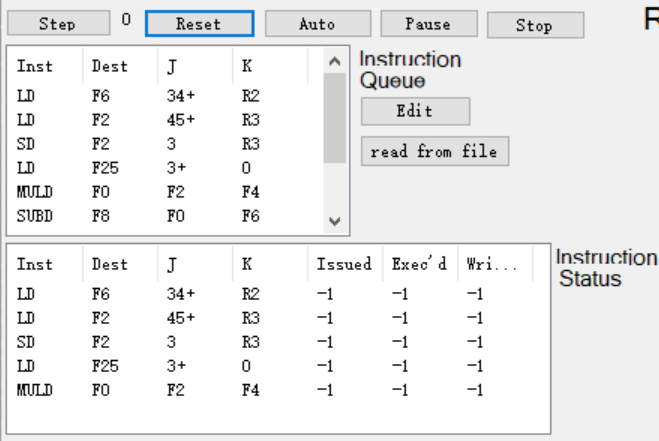
1. 界面操作功能

1.重置(reset):恢复到初始状态；

**思路**：原有功能，无需修改。恢复到初始状态意为指令队列回复到原始状态，之前的指令流出不作数。重新读入原始指令。时钟周期置零。

**截图**：

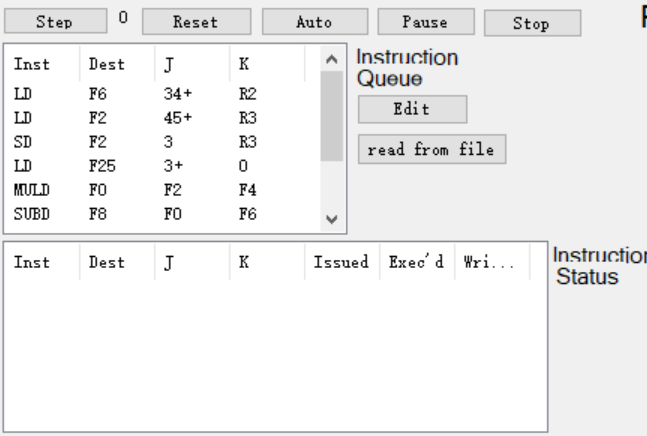


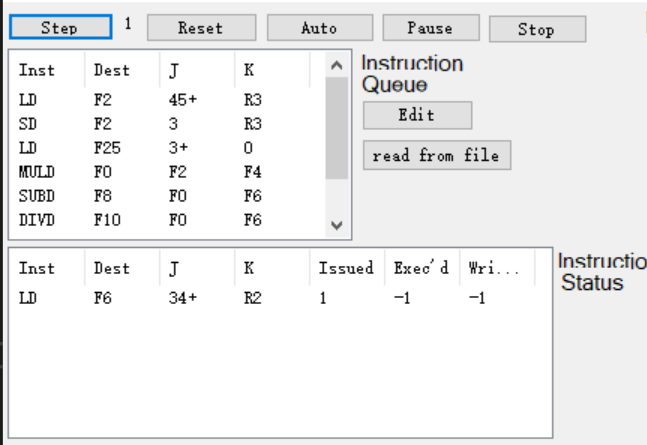


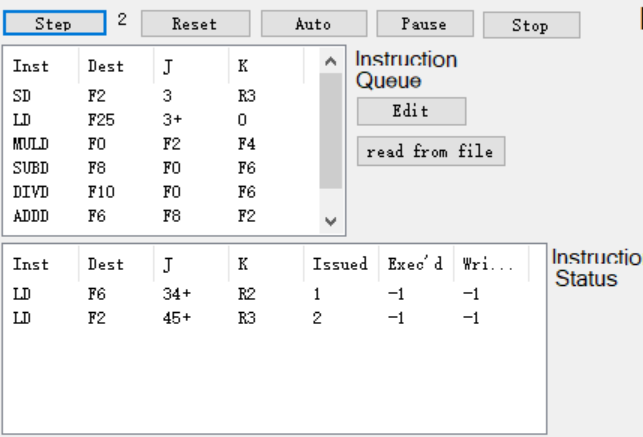
2.单步(step)：每按一下，时钟进 1，每条指令做相应的动作；

思路：原有功能，无需修改。每按一下按钮，调用RunOneCycle 函数，时钟周期+1.

截图：







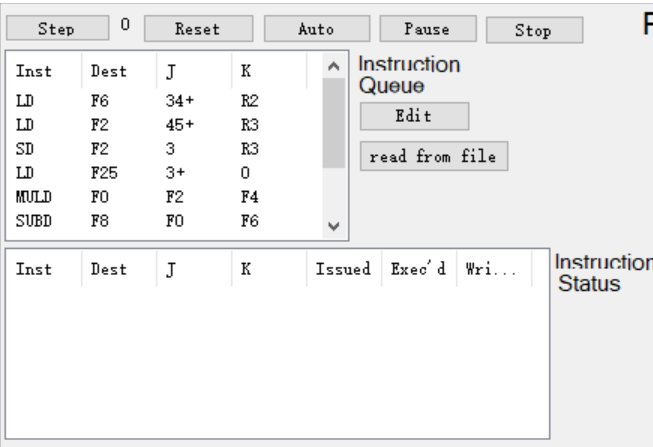
3.自动(auto)：自动执行，直到程序运行完毕，或者按下暂停(pause)键或停止 （stop）键；

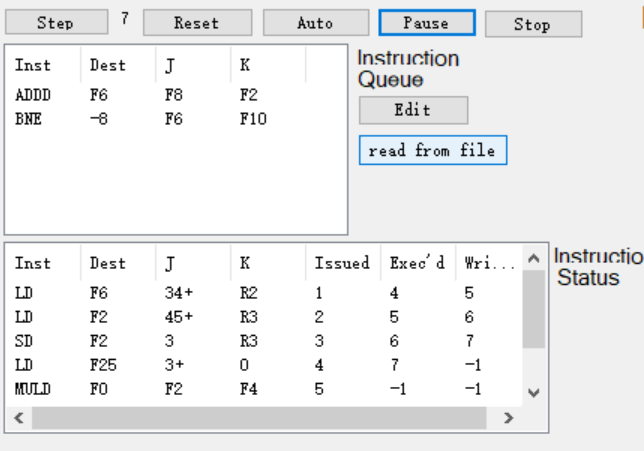
4.暂停(pause)：在自动执行时，可以暂停自动执行；再按一次，则继续自动执行。

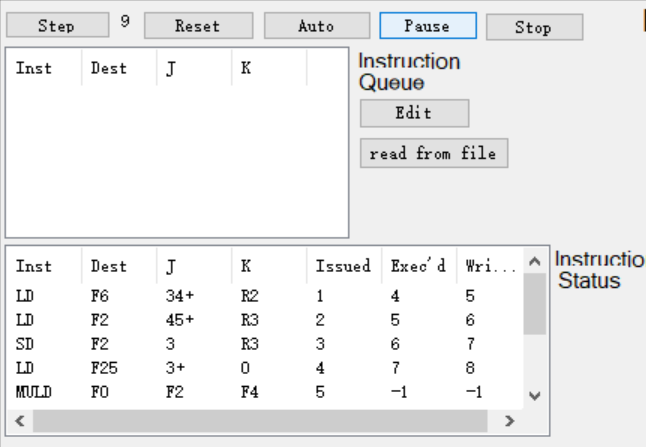
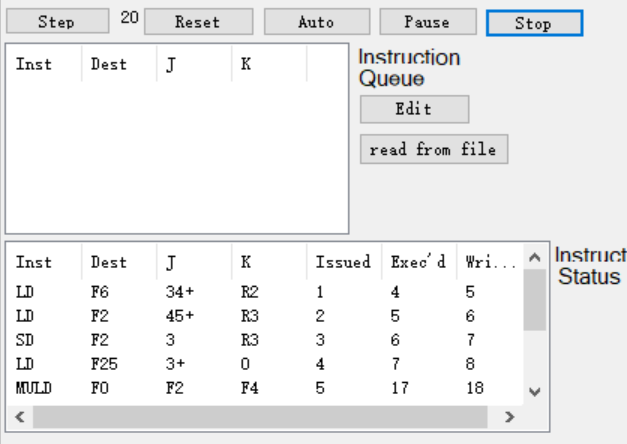
5.停止(stop):停止程序的执行

**思路**：增加了三个button，分别用于触发Auto、pause、stop事件。自动执行就是不断调用RunOneCycle（）函数，用while循环完成；这里新加入了两个bool型变量，两个变量都为循环的条件，一个用于区分Auto和Stop，如果点击Stop按钮触发该bool变量值为false，另一个用于触发pause事件，如果点击了pause，则该bool型变量取反，达到在自动执行时，可以暂停自动执行，再按一次，则继续自动执行条件；当pause按钮触发时，bool型变量取反，跳出循环，为了使循环继续，在pause事件中加入auto\_click()指令，每次触发pause事件都重新跳入该循环；整个模拟器采用单线程，所以还加入了Application.DoEvent()函数用于响应当前界面按钮；如果直接采用循环运行程序速度会非常快，所以在每个循环加入延迟1秒，更加人性化；reset事件中需要重置两个bool型变量；如果程序运行完毕需要停下，增加了一个判断所有保留站是否清空的函数，每次循环都要检查保留站是否清空，如果没有指令剩余，则停下。

**截图：**

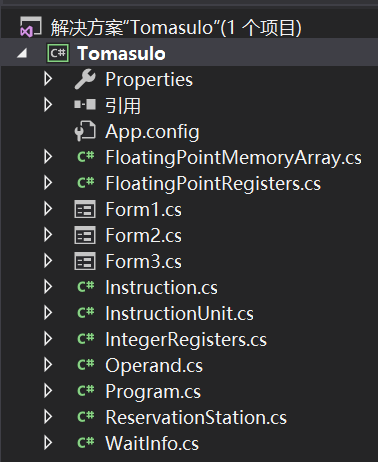




1. 对原Tomasulo模拟器的解释

4.1原有体系、功能及实现



整个模拟器所需要的功能都通过类实现，并在Form1中整合起来。

FloatingPointMemoryArray类定义了存储器

FloatingPointRegister类定义了浮点寄存器

Instruction定义指令，包括指令操作数、操作类型等

InstructionUnit类是指令列表

实现：一开始保留站初始化，传入延迟时间以及部件数量，延迟时间用一个delay数组存储；指令列表初始化后，调用Issue函数，检查结构冲突，然后放入保留站，Execute函数通过令每个保留站都调用RunExecution函数，RunExecution函数中定义了延迟时间，并且带有一个时钟周期的延迟，原因是第一次检查参数是否就绪，并且将状态改为开始执行，第二次进入将状态改成完成执行，如果加上传入的参数可以实现延迟；原有代码中load和store都有2个时钟周期的延迟，并且加上延迟参数也没有延迟，是因为第一个时钟周期只检查操作数是否就绪，将状态改为开始执行，然后第二个时钟周期不需要判断RunExecution的值就直接到下一个阶段，将addr改为ready，第三次进入将状态改为完成，其他指令判断都需要RunExecution返回值为0（finish）才进入下一个阶段，所以可以延迟执行；最后一个阶段是wb；每次执行都要更新各个存储器和保留站。

4.2存在的问题

RunOneCycle函数先调用Write然后Execute最后Issue目的是每次时钟周期只能执行一次，如果先issue再execute最后write则会发生一条指令在一个时钟周期内完成两个任务，造成混乱；但是按照算法来说，三个函数应该要同时执行，否则会出现，先wb之后，保留站free，下一条issue流入的指令本来遇到的结构冲突消失，错误。但是，由于调用函数需要有先后顺序，所以无解。

4.3改进

1.在保留站中，为了实现按照传入的参数延迟，需要在RunExecution函数返回0之后再进入下一个阶段，但是load和store都没有判断，只需要参数就绪之后，下一个循环直接进入下一个阶段，也就是没有等到RunExecution函数里面标志完成执行，所以无法按照传入的参数延迟执行；所以对load和store操作都做了修改，需要判断Run Execution函数返回值是否达到要求，只有执行完成后才能进入下一个阶段。

2.原本的延迟是默认的，load和store延2个时钟周期，add和multi延1一个时钟周期，分支延1个时钟周期，没有专门设置延迟的参数；在主界面中加入了一个delay数组存放传入的参数，实现延迟

3.原来的保留站是默认设置，因为还要读文件，所以需要重复设置保留站；加入了清空保留站的函数，每次从文件读入都需要清空

4.之前load执行过程延迟两个时钟周期，wb延迟两个时钟周期；wb阶段的延迟是在GetFloatMemory（）函数中出现的，这里设置了cycletocomplete，所以直接把这部分删除，在wb阶段直接写回，不延迟

5.对具体函数的解释和理解都标注在注释中

1. 实验源代码

Tomasulo项目，VS，debug模式