实验三 实现 Tomasulo算法模拟器

PB13011079 杨智

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。

实验环境

操作系统： mac OSX
IDE： IntelliJ 2016.1.1
编程语言： JAVA

实验要求

设计和实现一个Tomasulo算法模拟器。

1. 基本要求：
   针对程序中直线型代码，可乱序执行、乱序完成。
   a. 能够正确输出每个周期之后保留站的内容。
      保留站基本信息：

站名	状态	操作码	第一操作数值	第一操作数状态	第二操作数值	第二操作数状态

   b. 能够正确输出每个周期之后寄存器状态表的内容。
      寄存器状态表基本信息：

寄存器名

寄存器状态（0：不等待保留站的内容；n 表示等待的保留站名n>0）
寄存器内容（状态为0时，该值才有意义）

c. 能够正确输出每个周期之后指令状态表的内容（指令分为浮点运算指令和load/store指令），指令状态分为流入，执行和写回。

指令状态表基本信息

标志出每条指令流出、执行、写回这三个阶段所在的周期号

2. 较高要求：
   a. 实现带界面的模拟器，可动态输入指令并配置相关信息（e.g指令各个状态周期数，运算部件数目），并能正确输出上述基本要求中的相关信息。
   b. 实现带“再定序缓冲”的Tomasula算法模拟器，支持分支指令
3. 模拟器完成后，用你的模拟器测试以下程序并答题。

```
L.D   F6, 21（R2）
L.D   F2, 2，0（R3）
MUL.D    F0, F2, F4
SUB.D    F8, F6, F2
DIV.D    F10, F0, F6
ADD.D    F6, F8, F2
```

说明：

（1）假设浮点功能部件的延迟时间：加减法2个周期，乘法10个周期，load/store2个周期，除法40个周期。而指令的流入和写回为1个周期。

（2）第一条指令中21(R2)表示一个内存地址，将该地址的值load到F6寄存器。该内存的内容可以事先指定

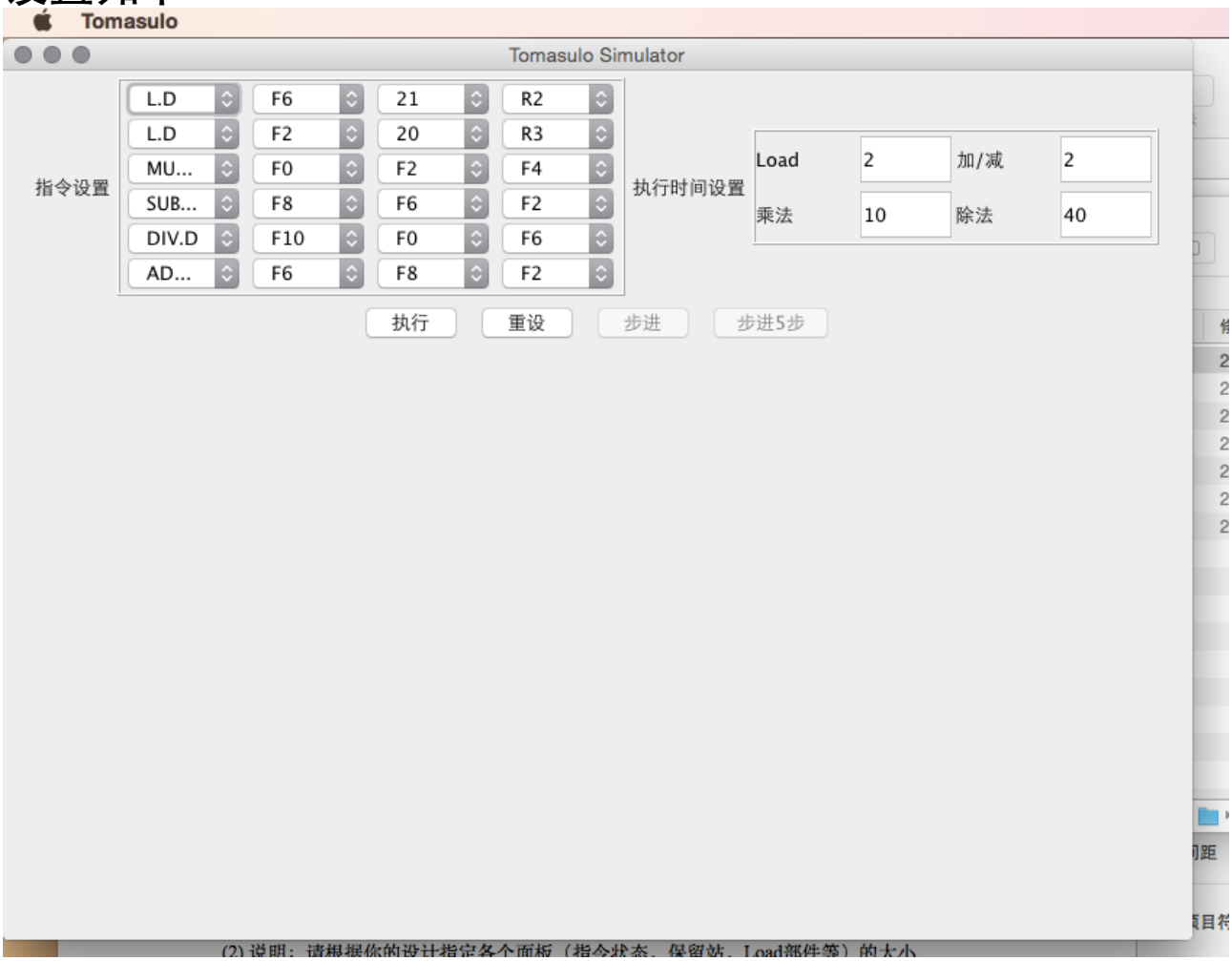
a. 给出在第5个时钟周期后，保留站的内容。
b. 给出在第10个周期后，保留站，寄存器状态表的信息。

4. 实验需要提交的内容包括：
   a. 实验源代码
   b. 实验最终的可执行文件
   c. 实验报告（包括设计思想、实验分析结论等）

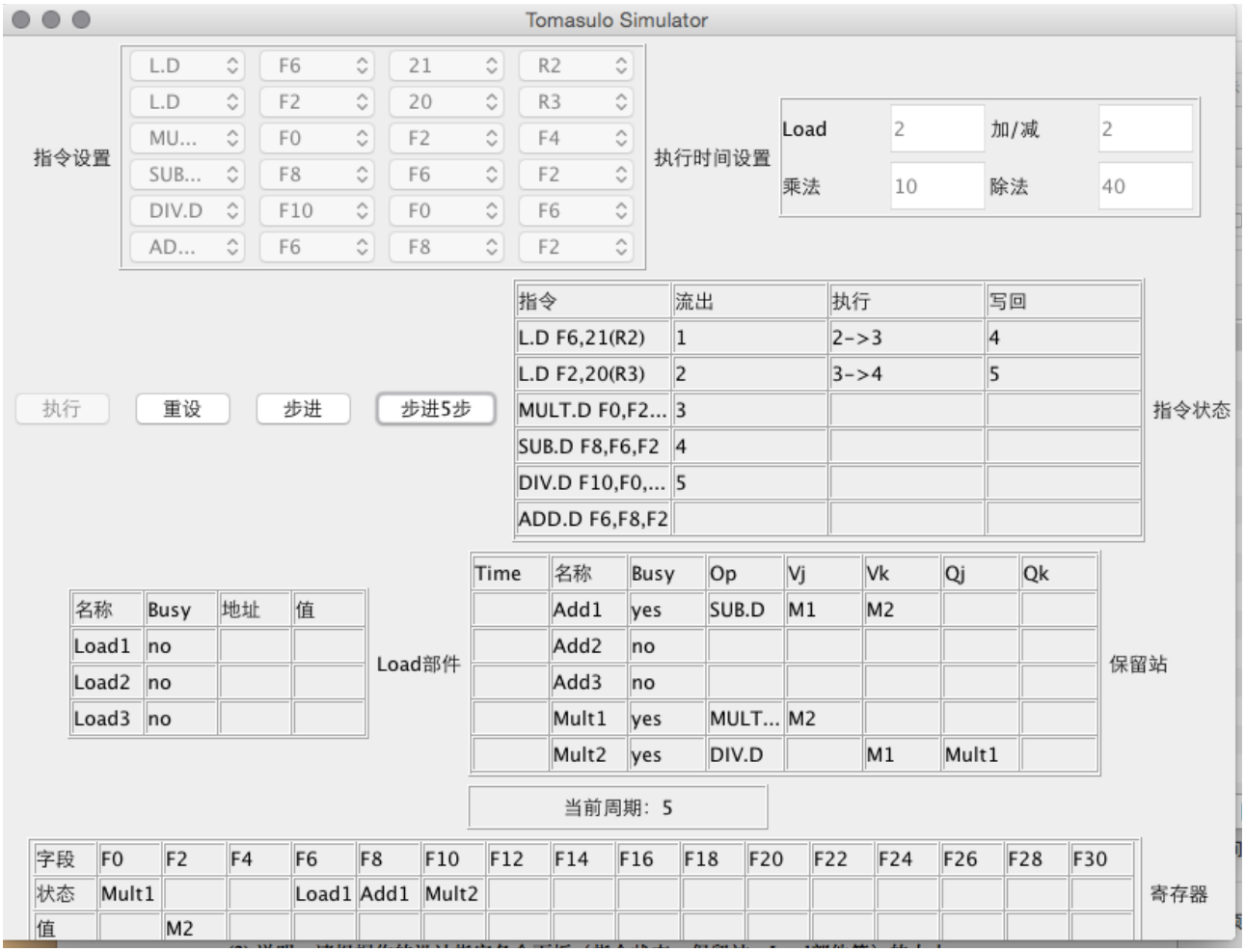
回答问题（实验要求第3点）

（第二条指令是 L.D F2, 20(R3)吧？题目写错了）

设置如下：


Tomasulo
Tomasulo Simulator
指令设置
L.D F6 21 R2
L.D F2 20 R3
MU... F0 F2 F4
SUB... F8 F6 F2
DIV.D F10 F0 F6
AD... F6 F8 F2
执行时间设置
Load 2 加/减 2
乘法 10 除法 40
执行
重设
步进
步进5步

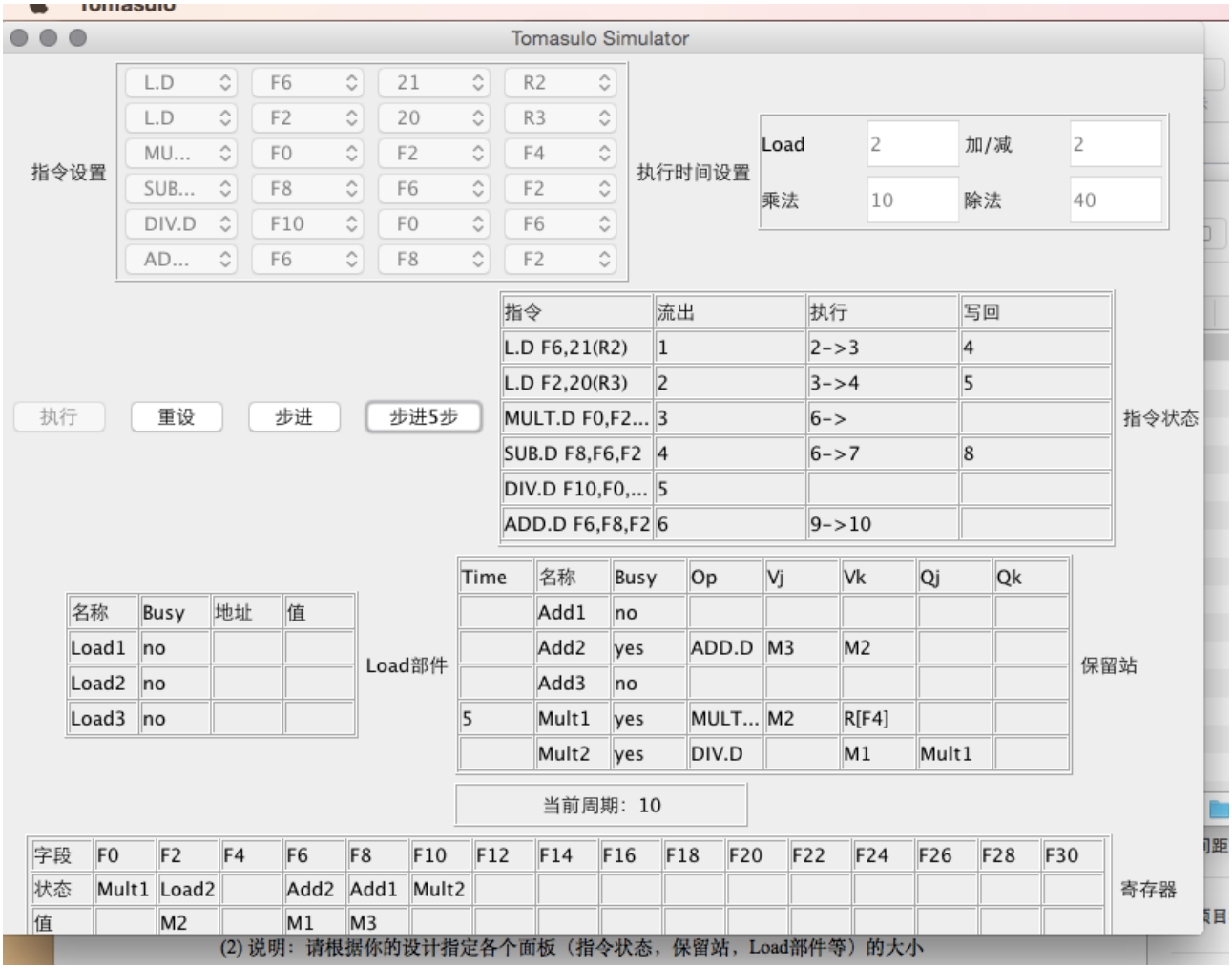
给出在第5个时钟周期后，保留站的内容。



结果分析:

按照 Tomasulo 的流程,5 个时钟周期流出了 5 条指令,其中 load 指令 2 条, mult 指令 1 条,sub 指令 1 条,divd 指令 1 条。其中 mult 和 sub 指令与 ld 产生相关,相 关寄存器编号为 F2。M1 与 M2 代表寄存器 F6 和 F2 数据已准备好。2 条 load 指令执行 完毕并且已经写回。

给出在第10个周期后，保留站，寄存器状态表的信息。



结果分析:

按照 Tomasulo 的流程,10 个时钟周期 6 条指令均已流出,且 2 条 load 指令执行完毕并且写回。sub 指令也已执行完毕并写回,目前保留站中剩下 add、mult、div3 条指令。add 指令正在执行阶段最后一个时钟周期,没有写回。div 指令等待 mult 指令 的结果。乘法指令还有 5 个时钟周期执行完毕。

设计思想

记号如下:

rs, rt : 源寄存器名；rd:目的寄存器名
RS: 保留站数据结构；r:保留站编号
RegisterStat：寄存器状态表
Reg： 寄存器堆

我写的Tomasulo算法模拟器主要是依照Tomasulo算法流水线控制的流程来设计的.

1. Issue

FP Operation:
    Wait until： Station r empty
    Action or bookkeeping:
        if(RegisterStat[rs].Qi≠0) {RS[r].Qj ← RegisterStat[rs].Qi}
        else {RS[r].Vj ←Reg[rs]; RS[r].Qj ←0 }
        if(RegisterStat[rt].Qi≠0) {RS[r].Qk ← RegisterStat[rt].Qi}
        else {RS[r].Vk ←Reg[rt]; RS[r].Qk←0 }
        RS[r].Busy ← yes; RegisterStat[rd].Qi = r;

Load or Store:
    Wait until: Buffer r empty
    Action or bookkeeping:
        if(RegisterStat[rs].Qi≠0) {RS[r].Qj ← RegisterStat[rs].Qi}
        else {RS[r].Vj ←Reg[rs]; RS[r].Qj ←0 }
        RS[r].A ←imm; RS[r].Busy ← yes;
    Load only: RegisterStat[rt].Qi = r;
    Store only:
        if(RegisterStat[rt].Qi≠0) {RS[r].Qk ← RegisterStat[rt].Qi}
        else {RS[r].Vk ←Reg[rt]; RS[r].Qk←0 }

2、Execute

    FP Operation:
        wait until: (RS[r].Qj=0) and (RS[r].Qk=0)
        Action or bookkeeping:
            computer result: Operands are in Vj and Vk
    Load-store step1:
        wait until: RS[r].Qj =0 & r is head of load-store queue
        Action or bookkeeping:
            RS[r].A ← RS[r].Vj + RS[r].A;
    Load step2:
        wait until: Load Step1 complete
        Action or bookkeeping: Read from Mem[RS[r].A]

3、Write result

FP Operation or Load:

Wait until: Execution complete at r & CDB available

Action or bookkeeping

∀x (if (RegisterStat[x].Qi=r) {Regs[x] ← result; RegisterStat[x].Qi ← 0})

∀ x (if(RS[x].Qj =r) {RS[x].Vj ←result; RS[x].Qj ←0});

∀ x (if(RS[x].Qk =r) {RS[x].Vk ←result; RS[x].Qk ←0});

RS[r].Busy ← no;

Store:

wait until: Execution complete at r & RS[r].Qk = 0

Action or bookkeeping

Mem[RS[r].A] ← RS[r].Vk;

RS[r].Busy ← no;

实验总结

Tomasulo算法则通过动态调度的方式，在不影响结果正确性的前提下，重新排列指令实际执行的顺序（乱序执行），提高时间利用效率。

该算法与之前同样用于实现指令流水线动态调度的计分板不同在于它使用了寄存器重命名机制。指令之间具有数据相关性（例如后条指令的源寄存器恰好是前条指令要写入的目标寄存器），进行动态调度时必须避免三类冒险：写后读（Read-after-Write, RAW）、写后写（Write-after-Write, WAW）、读后写（Write-after-Read, WAR）。第一种冒险也被称为真数据相关（true data dependence），而后两种冒险则并没有那么致命，它们可以由寄存器重命名来予以解决。Tomasulo算法使用了一个共享数据总线（common data bus, CDB）将已计算出的值广播给所有需要这个值作为指令源操作数的保留站。该算法尽可能降低了使用计分板技术导致的流水线停顿，从而改善了并行计算的效率。