Tomasulo 算法设计文档

姜仲禹　邵韵秋　黄家晖

2017 年 6 月

目录

1 算法及程序设计说明

1.1 概述

我们按照作业要求设计了流水线，支持 16 个浮点数寄存器进行计算（分别为 F0、F2 到 F30），8 个整数寄存器（R0 至 R7）用于间接寻址，4096 个内存单元。保留站的数目也按要求完成，3 个加减法保留站，2 个乘除法保留站，3 个 load 保留站和 3 个 store 保留站。

项目使用 Java 编写完成，UI 的编写使用了 JavaFX 以构建更加美观的界面。

1.2 设计思路

由于电路中很多步骤都是并行完成，而我们模拟流水线运行的算法将会是线性执行的，并且电路中寄存器的值都是在电压跳变后才会发生改变，所以在当周期计算的过程中，其实是使用了上个周期的计算结果，这就是实验设计所需要面对的第一个难关。

为了解决这个问题，在一个周期中，我们按照 WriteResult（写回）、Exe（执行）、Issue（发射）的顺序执行，就避免了流水线选择不应该选择的指令执行或者结果还没有写回的问题。

第二个问题是读写指令，由于书上和 PPT 中都没有明确的指出这个问题，所以一开始我们认为 load 和 store 指令是分别拥有一个队列，分别顺序执行，然而在编写代码的过程中，我们发现这样的执行顺序会导致内存中的 WAR，WAW 类似问题。为了解决这个问题，在查阅了原始论文和其他资料后，我们了解到 Tomasulo 算法的 load 和 store 指令其实是共用同一个指令队列，必须完全按照顺序执行，于是我们就按照这样的思路实现了算法。

1.3 代码结构

我们将这个流水线封装为一个类，这样就方便了 UI 对流水线的调用，并且定义了保留站类方便流水线对于保留站的管理。

UI 通过对接口的调用，可以控制流水线的行为，并观测流水线中各指令的实时状态。

2 UI 使用说明

我们设计的 Tomasulo 算法模拟器界面如图 1所示。

图 1: Tomasulo 算法模拟器运行示意图

界面的左侧从上到下依次显示所有指令执行情况、加（减）法和乘（除）法部件的保留栈以及相应保留栈的标识位（各个标识位的含义与书本相同）、Load-Store 队列。关于 Load-Store 队列需要再作一点说明：在我们的假设中，存储器同一时刻仅能进行一个地址的读操作或写操作，因此为了展示更清晰，在 UI 中将读写队列合为一个，能够更加直观观察 LD 和 ST 指令执行顺序的意义。对于 LD 指令，由于不需要等待任何操作数就绪，所以其 Vj 和 Qj 列没有意义；而对于 ST 指令，其存储的内容需要等待相应的寄存器的值，Vj 代表操作数寄存器的值，Qj 代表将产生所需值的保留栈名称。

界面的右侧显示浮点寄存器的值以及通用寄存器的值，以及内存中含有非 0 值的地址偏移和数值。内存和寄存器的值可以在模拟之前由用户设定，但模拟过程中不能改变，模拟一轮结束之后寄存器和内存的值都会自动清零。

2.1 输入代码与设置初始状态

输入代码 点击窗口左侧的“输入指令”功能即可输入测试代码，代码的输入请严格遵循指令格式，如果出现语法错误或未知错误，请检查是否出现了下面的问题：

- 确认操作数的个数正确；
- 寄存器的逗号后不应该有空格：`ADDD F0,F2,F4` 正确，`ADDD F0, F2, F4` 错误；

另外，我们也提供了若干个示例进行测试，详情请参阅测试说明一节。

设置初始寄存器和内存 点击窗口菜单栏的“更改”按钮，可以选择更改寄存器或是内存的值，如图 2所示。

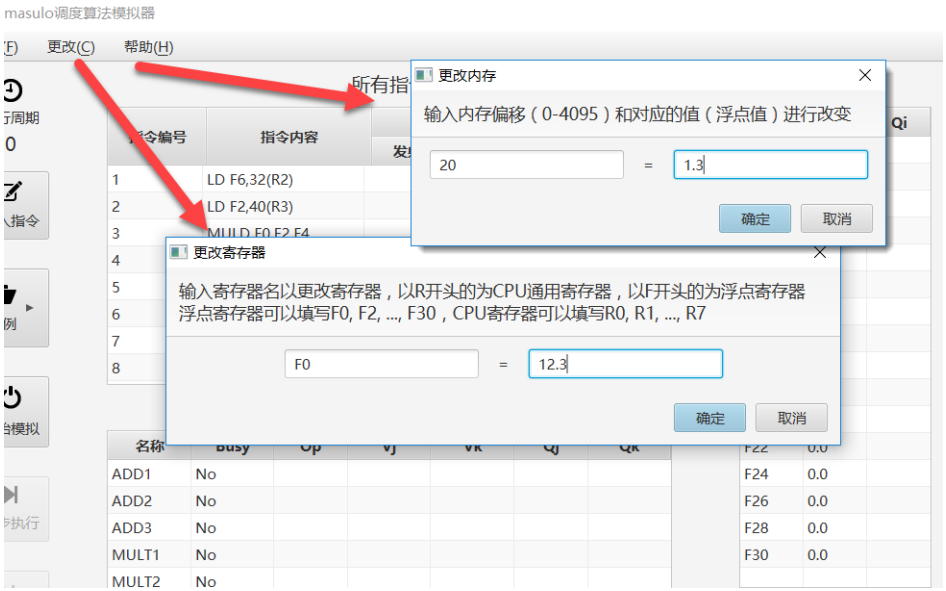

图 2: 模拟前设置内存和寄存器的初始值

2.2 单步调试与连续模拟

单步/多步调试 点击“开始模拟”即可进入模拟状态，此时可以选择单步前进或是多步前进，如图 3所示。当所有指令执行完毕的时候，模拟器自动停止。

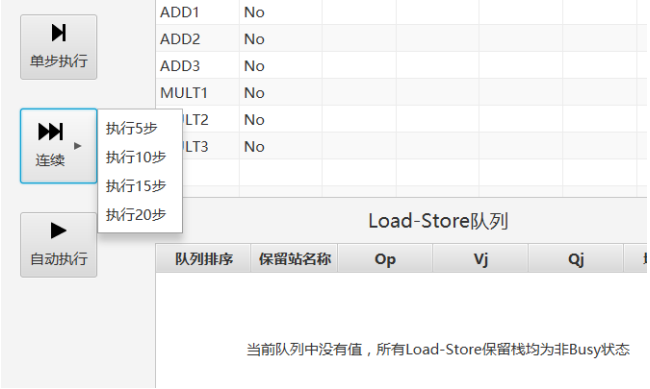

图 3: 使用模拟器进行单步或多步调试

连续模拟 点击“自动执行”按钮，模拟器即会每隔 1 秒自动前进一个时钟周期并刷新各个表格的状态。当所有指令执行完毕的时候，模拟器自动停止，并在状态栏给出相关的提示。

3 测试说明

3.1 概述

Tomasulo 算法的核心思想的记录和检测指令相关的操作数一旦就绪，立即执行，即顺序发射乱序执行，通过寄存器换名的方式解决 WAW 和 WAR 冲突。此外，还需要通过访问存储器队列解决读写访问内存的冲突。

此次算法模拟器的测试就着重对访问内存冲突，WAW、WAR 冲突，以及队列和保留站满等情况进行检测。

本次所用到的测例均在 samples 文件夹下，共有 s1-s7 七个测例，其主要测试的部分如表 1所示：（注：定义了内存初始化函数 meminit(), 可在 pipline.java 的 run() 函数中使用）

文件	测试说明
s1.txt	课件中的例子
s2.txt	load/store 冲突检测
s3.txt	综合检测
s4.txt	综合检测
s5.txt	综合检测
s6.txt	load/store 冲突检测
s7.txt	WAW/WAR 冲突检测，保留站满溢检测

表 1: 测试文件功能说明

3.2 WAR/WAW 冲突处理测试

s7.txt 即为 WAR/WAW 冲突处理的测试文件，且会出现保留站满的情况。具体可能出现的冲突如表 2所示：

序号	命令	冲突说明
1	SUBD F0,F2,F4	
2	ADDD F2,F4,F8	F2 WAR
3	MULD F2,F2,F8	F2 WAR,WAW
4	DIVD F8,F8,F6	F8 WAR
5	ST F2,0(R0)	
6	MULD F2,F4,F8	F2 WAW; F8 WAR; 且缓冲区可能满了
7	MULD F6,F0,F2	F2,F0 WAR；且缓冲区可能满了

表 2: WAR/WAW 冲突处理测试文件说明

3.3 load/store 冲突处理测试

s6.txt 和 s2.txt 即为 load/store 冲突处理的测试文件。以 s6.txt 中的测试为例，可能出现的冲突如表 3所示：

序号	命令	冲突说明
1	LD F0,0(R0)	
2	ST F0,4(R0)	RAW, 等待 (1)LD 指令写寄存器完成
3	LD F2,8(R0)	
4	ST F2,12(R2)	RAW, 等待 (3)LD 指令写寄存器完成
5	LD F0,4(R2)	
6	ST F0,16(R2)	RAW, 等待 (5)LD 指令写寄存器完成
7	LD F2,16(R2)	等待 (6)ST 指令写内存完成
8	ST F2,0(R3)	RAW, 等待 (7)LD 指令写寄存器完成

表 3: s6 可能出现的冲突

3.4 综合测试

s1.txt, s3.txt, s4.txt, s5.txt 中均为综合测试的文件，测试内容包括访存和浮点数运算，可能出现的冲突基本都有覆盖到，下面以 s5.txt 中的测试为例做出说明，如表 4所示。

序号	命令	冲突说明
1	LD F2,8(R4)	
2	LD F4,12(R4)	
3	LD F8,8(R4)	
4	DIVD F0,F2,F4	F2,F4 RAW, 需等 1,2 访存指令完成
5	ADDD F6,F0,F8	F0,F8 RAW, 需等 3,4 指令执行完毕
6	ST F6,0(R4)	F6 RAW, 需等 5 指令执行完毕
7	SUBD F8,F10,F14	F8, WAR, 与指令 5 相关
8	MULD F6,F10,F8	F6 WAW, 与指令 5 相关；F8 RAW, 需等待指令 7 执行完毕
9	ST F6,0(R4)	F6 RAW, 需等待指令 6 和 8 执行完毕

表 4: s5 可能出现的冲突

3.5 测试结果

经测试，该模拟程序均能正确通过这七个测试，寄存器和内存中的值也符合预期。(注：目前 ui 文件中的样例 1-7 即为测试所用文件。)