Tomasulo 模拟器实验报告

王 琛　2016011360　计 65

June 7, 2019

1 实验原理

本次实验要求完成 Tomasulo 算法模拟器。Tomasulo 算法以硬件的方式实现寄存器重命名，允许指令顺序发射，乱序执行，能够有效提高流水线的吞吐率和运行效率。

2 实验要求

实验要求实现的模拟器应当能够接受 NEL 语言作为输入，并且给出每个周期结束时相应部件的运行状态。NEL 语言只支持三类指令，运算指令、装载指令和控制指令。另外，对于一些细节的要求如下：

- 在某个指令的执行完成周期，该指令所占有的 FU 就应当被释放。其他就绪指令可以在此周期进入 FU，下个周期即能开始执行。
- 在某个指令的写回周期，该指令占用的保留站被释放。在此周期被 issue 的指令可以进入保留站。
- 当除法中除数为 0 时，直接写入被除数的值，只需要执行一个周期。在记录每条指令的执行和写回周期时，由于循环的出现，同一条 DIV 指令可能被执行多次。如果后执行的指令由于除 0 只需要一个周期，可能提前完成。由于我们要求记录指令第一次发射的执行完成和写回周期，因此有可能记录的是后完成的指令周期，从而导致错误的结果。对此，我的处理方法是，保留站中记录指令的发射周期，在记录执行完成和写回周期时，判断保留站的指令发射周期和对应的指令发射周期是否相同，如果相同则为第一次发射，应当记录。否则，此次发射不是指令第一次执行，不记录。

3 实验设计

3.1 设计思路

按照实验要求，代码中模拟了对应的功能部件、保留站和 32 个寄存器。项目使用 Java 编写。

Tomasulo 算法的核心有三个过程，发射、执行和写回。在硬件上，这三个过程是并行完成的，但是程序中不得不串行执行。为了保证本周期释放的保留站能够被本周期发射的指令使用，我将发射放在了每个周期的最后完成。

3.2 具体设计

程序主要分为指令模块、寄存器模块、保留站模块、运算部件模块、主程序模块以及用户界面模块。下面具体介绍：

3.2.1 指令模块：Instr.java & InstrLoader.java

Instr.java 中定义了指令的基类，包含指令各阶段的时间信息，指令执行所需周期，指令的种类信息。

```
    abstract class Instr {
    int issue, exec, write; //issue, exec and write time
    int ready;
    int latency;
    String instrStr;
    InstrType type;
    Instr (InstrType type, String instrStr, int latency) {
        this.latency = latency;
        this.type = type;
        this.issue = -1;
        this.exec = -1;
        this.write = -1;
        this.ready = -1;
        this.instrStr = instrStr;
    }
}
```

而 ArithInstr, JPInstr, LDInstr 则继承了 Instr 类，分别代表算术指令、跳转指令和加载指令，每个子类有自己的操作数等独特属性。

InstrLoader 类则是从 Nel 文件中加载指令，将每条指令存放为 Instr 类，最终将所有指令按顺序放入链表中。

3.2.2 寄存器模块：Register.java

寄存器模块主要是存放每个寄存器的信息。valid 表示其中的值是否有效，如果有效，value 则为寄存器的值；如果无效，说明该寄存器正在等待某个保留栈，此时 rs 存放了对应的保留站名称。

```
class Register {
    int value;
    String rs; // the name of waiting rs
    boolean valid; // invalid if it is waiting
    Register() {
        value = 0;
        rs = null;
        valid = true;
    }
    void writeValue(int res) {
        value = res;
        valid = true;
        rs = null;
    }
    void clear() {
        value = 0;
        rs = null;
        valid = true;
    }
}
```

3.2.3 保留站模块：Reserv.java

保留站模块是对保留站进行的抽象，记录了保留站对应指令的序号，发射、执行、写回时间，指令的操作类型、指令的结果。另外，busy 表示保留站是否被占用，而 exec 表示保留站对应的指令是否在功能单元执行。exec 的时间是 busy 的子集，引入 exec 的主要目的在于选择功能单元时区分已经进入和未进入功能单元的保留站。由于 Load 指令使用的保留站的其他指令稍有不同，代码中做了相应的区分，将其设置为 LoadBuffer 类。

3.2.4 功能单元模块：FU.java

功能单元记录了其是否被占用、剩余执行时间以及执行的指令。基类是 FU，子类有 ArithFU 和 LoadFU。

```
class FU {
    int runtimeLeft;
    boolean busy;
    // int res;
    Instr instr;
    String name;
    FU (String name) {
```

```
            this.name = name;
            this.busy = false;
        }
    }
```

3.2.5 主程序模块：Tomasulo.java

此模块完成了对每个功能部件的初始化以及指令的执行。下面对各个函数做具体的介绍。

issue issue 函数根据不同的指令类型检查相对应的保留站是否有空余，如果有则发射指令，占用相对应的保留站。另外，由于没有做分支预测，如果当前程序中 JUMP 指令尚未返回结果，则直接无法发射指令。

exec 对应算法中的执行阶段。遍历所有的功能部件，查看是否有指令在执行，如果有，则将其执行的剩余时间减 1。如果剩余时间为 0，则释放此功能部件，并且将其对应的指令的写回时间设置为下个周期。

writeBack 对应算法的写回阶段。遍历所有的保留站，利用 exec 阶段中记录的写回时间，如果写回时间正好等于当前周期，则进行写回操作。具体由 checkAndWrite 函数完成，该函数以某个保留站的名字做为参数，遍历其余的保留站和寄存器，如果有等待传入保留站的数据，则将数据写入，这也是 Tomasulo 算法的核心之处，相当于硬件中在 Common Data Bus 中传送数据。

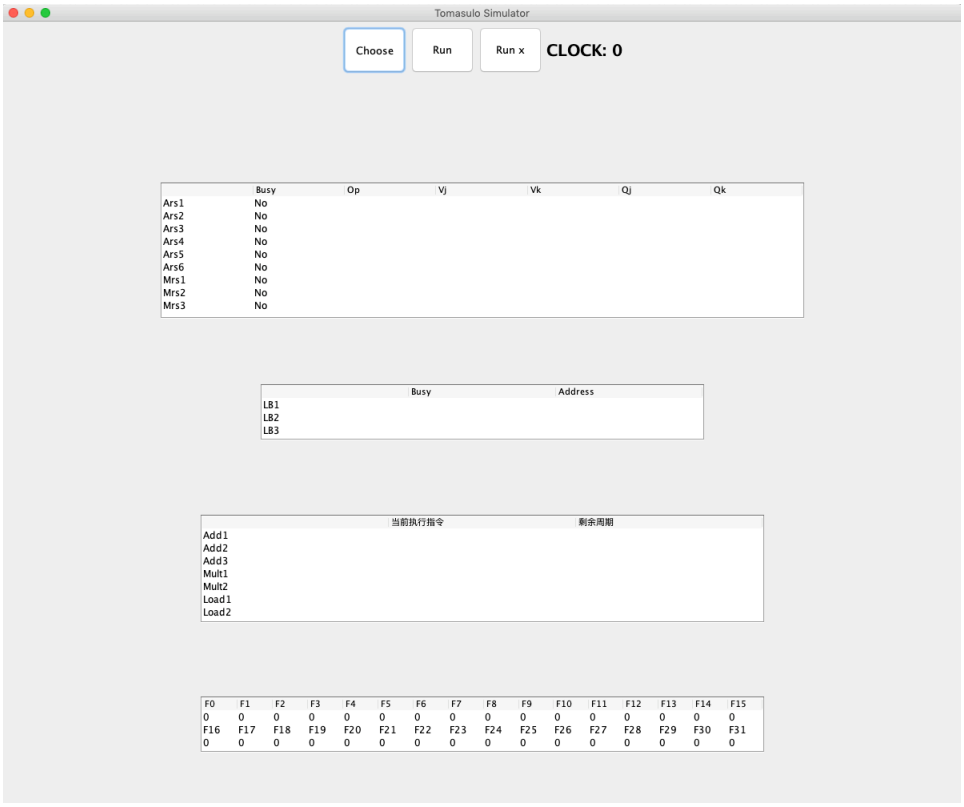
checkFU 除了以上三个算法中的核心函数，模拟器中在每个周期还要执行 checkFU 函数。此函数的主要目的是检查是否有空闲的功能部件，以使就绪指令能够及时进入功能部件，下周期开始执行。

executeByStep 执行算法的一个周期。首先判断是否执行完成，如果没有完成，依次执行 exec, writeBack, issue, checkFU 函数。将 issue 放在 exec 和 writeBack 的主要目的是能够使用 writeBack 刚释放的保留站。

3.2.6 UI 模块：Widgets.java & MainWin.java

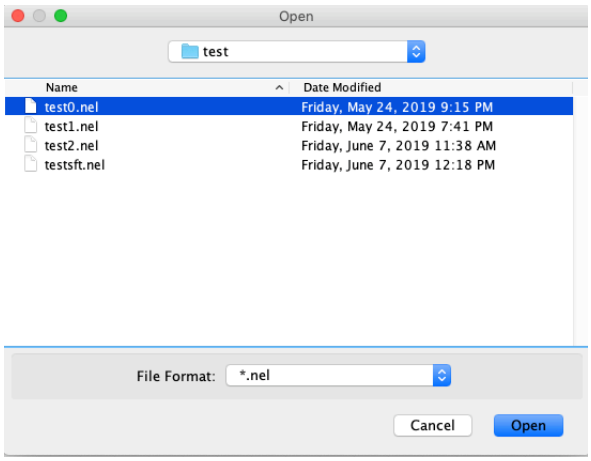
Widgets.java 定义了寄存器、保留站和功能部件的显示方式。MainWin 对此进行初始化并且显示。

主界面如下：



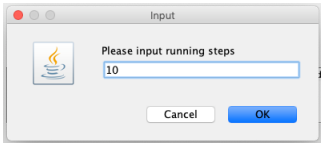
每个按钮的功能为：

Choose　选择一个 nel 文件将其加载到模拟器中。

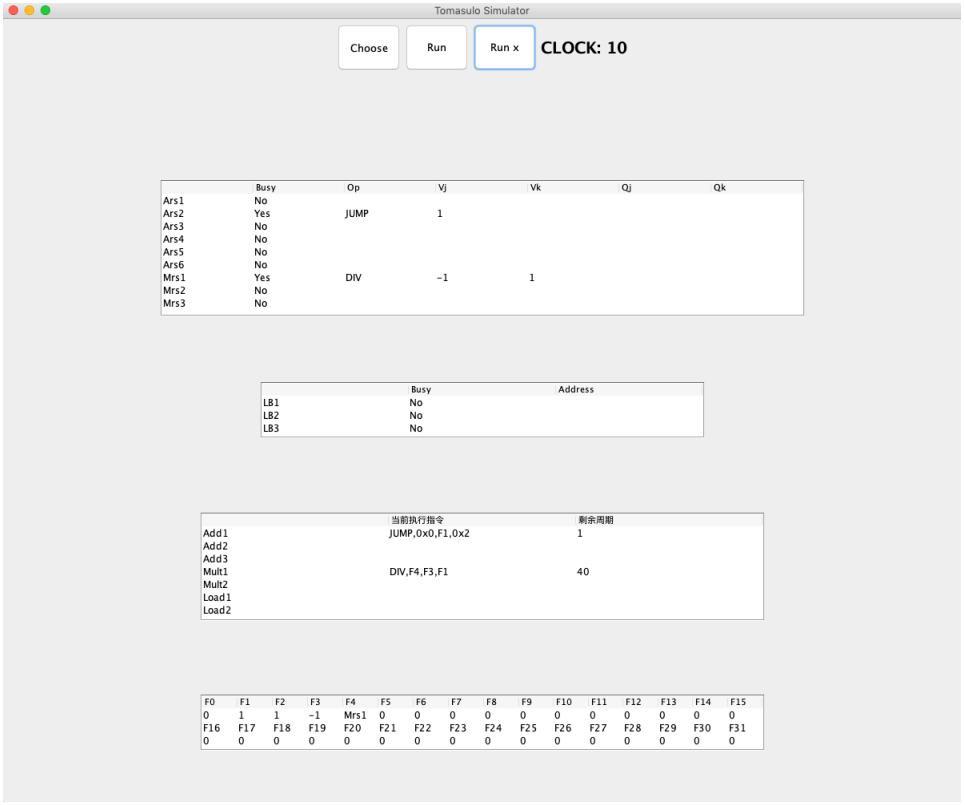


Run　调用 executeByStep 函数，执行一个周期。

RunX　弹出输入框，输入一个数字，执行相应数目的周期。



执行了 10 个周期的显示：



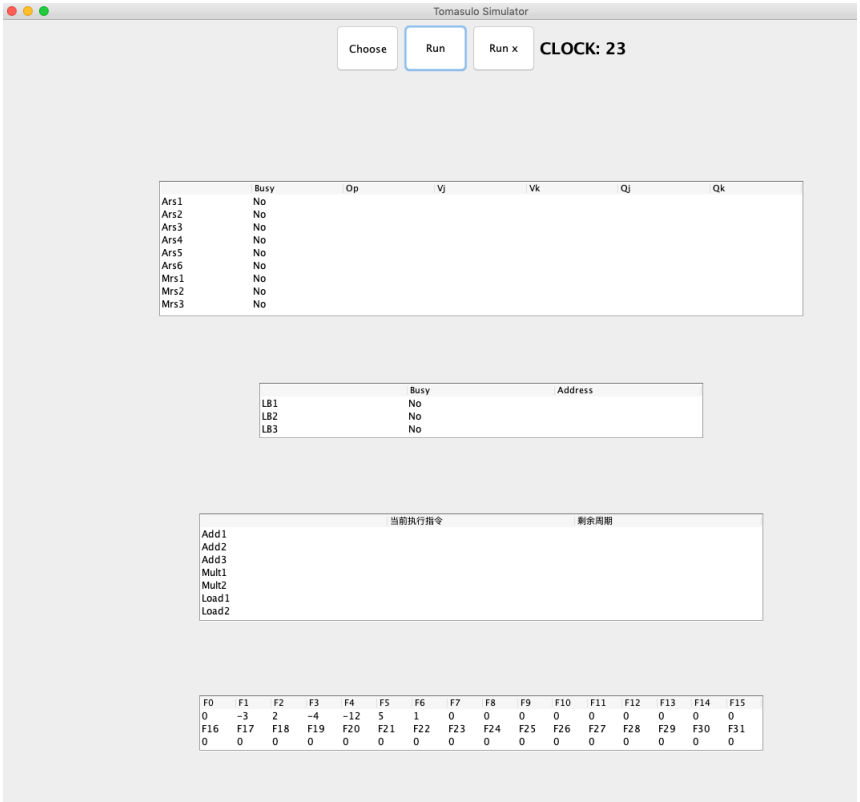
3.3 扩展指令

实现中，我扩展了 nel 指令集，加入了 SAL 和 SAR，对应了算术左移和算术右移。格式为 SAR, F1, F2, F3，含义是将 F2 寄存器的值右移 F3 位放入 F1 中，执行周期为 1，使用 Ars 作为保留站，Add 作为功能单元。

给出的测试用例如下：(见 testsft.nel)

```
LD,F1,0x3
LD,F5,0x5
LD,F6,0x1
SAR,F2,F5,F6
ADD,F5,F1,F2
LD,F3,0xFFFFFFFC
MUL,F4,F1,F3
LD,F1,0x4
LD,F2,0x2
SAL,F1,F1,F2
JUMP,0x10,F1,0x2
DIV,F2,F1,F3
SUB,F1,F2,F5
```

包含了 SAR 和 SAL 指令，JUMP 中 F1 等于 16，满足条件应该直接跳转到最后一条指令，最终的执行结果如下图：



经过验证，执行结果与人工计算完全相同。

3.4 补充测例

test3.nel 中对于各种冲突进行了测试：

```
LD,F2,0x4
LD,F4,0x1
LD,F8,0x10
LD,F6,0x2
SUB,F0,F2,F4
ADD,F2,F4,F8       F2 WAR冲突
MUL,F2,F2,F8       F2 RAW, WAW冲突
DIV,F8,F8,F6       F8 WAR冲突
MUL,F2,F4,F8       F2 WAW, F8 WAR冲突, 保留站满
MUL,F6,F0,F2       F2, F0 WAR冲突, 保留站满
```

经过测试，程序运行正确。

3.5 代码编译

代码使用的 javac 和 java 版本均为 1.8.0_191，所有源代码均在 src/tomasulo 文件夹中，做为一个 package 的形式存在。编译时进入 src 文件夹，执行 javac tomasulo/*.java

编译，使用 java tomasulo/MainWin 执行，也可用 java tomasulo/Main 执行命令行程序。

或者直接执行 run.sh 文件，会将编译的 class 文件存放入 src 同级的 bin 文件夹下，之后直接执行。

4 实验总结

这次实验是实现 Tomasulo 算法模拟器。实验中让我体会最深的是，课堂上讲的、作业做的算法，似乎都很简单，但是实验中却有数不清的细节要注意。由于代码封装做的不好，我写了很多重复的代码，也因此出现了很多可笑的 bug。代码中有很多细节需要注意，需要想的很清楚才能写下去。最终我还是顺利完成了实验，感谢老师和助教的付出！