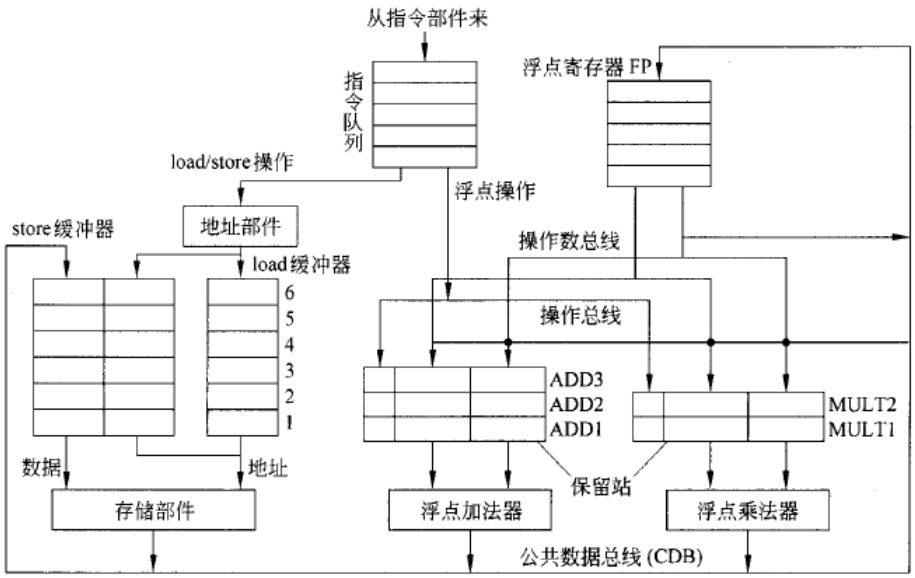
Tomasulo 算法模拟器实验报告

杨子江	2013011710	yangzj13@mails.tsinghua.edu.cn
王欣然	2014011300	wangxr1108@126.com
王倩	2014011319	q-wang14@mails.tsinghua.edu.cn

算法原理

Tomasulo 算法通过保留站的设置来记录流水中的指令的相关状态，来避免 WAR 和 WAW 冲突。对于每个保留站，当当前指令所需要的数据都已经就绪的时候就开始执行指令。每条指令执行时会影响的寄存器会记录下影响它的保留栈编号。



如图所示，浮点加法器有 3 个保留站 ADD1~3，浮点乘法器有 2 个保留站 MULT1~2，分别用于 ADDD，SUBD，MULD 和 DIVD 指令。LOAD 指令和 STORE 指令分别有一个缓冲器，和保留站有相似的作用。

Tomasulo 算法分为 3 个流水：

1. 发射(influx)

   取下一条指令，寻找对应的空闲保留站，把该指令保存到该保留站，如果其操作数在寄存器中已经就绪，就把这些操作数从寄存器中送入保留站，如果操作数还没有就绪，就把将产生该操作数的保留站的标识送入该保留站。这样，一旦该数据计算出来，这条指令便可以立即执行。如果找不到空闲的保留站则等待。

2. 运行(exec)

   监视所有保留站和缓冲区中的指令，如果符合运行的要求就放入对应的功能单元执行。

3. 写回(update)

   用计算结果更新寄存器和保留站的依赖关系以便等待的指令满足执行的条件。

算法伪代码

1. **指令流出**

   a) 浮点运算指令

   进入条件：有空闲保留站（设为 r）

   操作和状态表内容修改：

```
if (Qi[rs] != 0)
      RS[r].Qj = Qi[rs];
else {
      RS[r].Vj = Regs[rs];
      RS[r].Qj = 0;
}
if (Qi[rt] != 0)
      RS[r].Qk = Qi[rt];
else {
      RS[r].Vk = Regs[rt];
      RS[r].Qk = 0;
}
RS[r].Busy = yes;
RS[r].Op = Op;
Qi[rd] = r;
```

   b) load 和 store 指令

   进入条件：缓冲器有空闲单元

   操作和状态表内容修改：

```
if (Qi[rs] != 0)
     RS[r].Qj = Qi[rs];
else
     RS[r].Vj = Regs[rs];
RS[r].Qj = 0;
RS[r].Busy = yes;
RS[r].A = Imm;
```

   对于 load 指令：

```
Qi[rt] = rt;
```

   对于 store 指令：

```
if (Qi[rt] != 0)
     RS[r].Qk = Qi[rt];
else {
     RS[r].Vk = Regs[rt];
     RS[r].Qk = 0;
}
```

2. **执行**

   a) 浮点操作指令

   进入条件：`(RS[r].Qj == 0) && (RS[r].Qk == 0)`

   操作数和状态修改：进行计算，产生结果

   b) load/store 指令

进入条件：(RS[r].Qj == 0) 且 r 成为 load/store 缓冲队列的头部

3. **写结果**

a) 浮点指令和 load 指令

进入条件：保留站 r 执行结束，且 CDB 就绪

操作和状态表内容修改：

对任意寄存器 x，

```
if (Qi[x] == r) {
    Regs[x] = result;
    Qi[x] = 0;
}
```

对任意保留站 x，

```
if (RS[x].Qj == r) {
    RS[x].Vj = result;
    RS[x].Qj = 0;
}
if (RS[x].Qk == r) {
    RS[x].Vk = result;
    RS[x].Qk = 0;
}
RS[r].Busy = no;
```

b) store 指令

进入条件：保留站 r 执行结束，且 RS[r].Qk == 0

操作和状态表内容修改：

```
Mem[Rs[r].A] = RS[r].Vk;
RS[r].Busy = no;
```

模拟器设计与实现

本模拟器由 Java 语言写成，主要包含以下几个类：

Simulator.java：模拟器的控制类，包含算法的主要逻辑

Registers.java：寄存器组类，定义了寄存器组的接口

Memory.java：存储器类，定义了存储器的接口

ResStation.java：保留站类

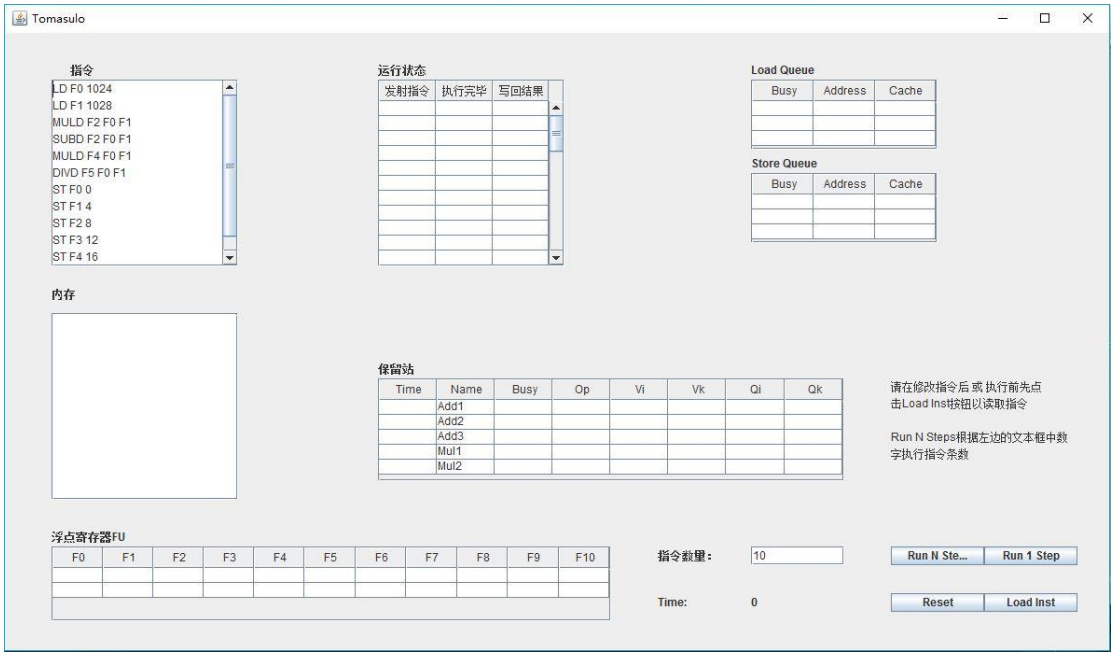
Instruction.java：指令类，定义了指令解析的接口

Adder.java：加法器类

Multiplier.java：乘法器类

Memory_unit.java：存储器驱动类

模拟器界面



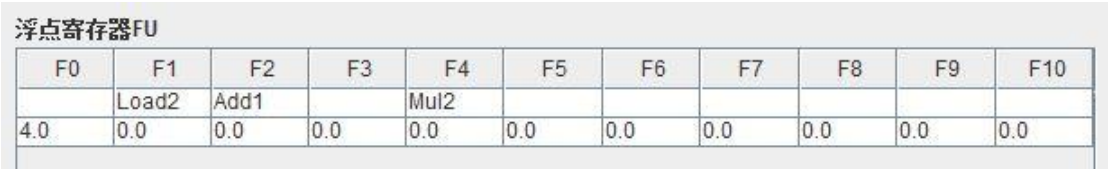
总体界面

采用文本框输入指令，输入后需要点击右下角的 Load Inst 按钮再执行

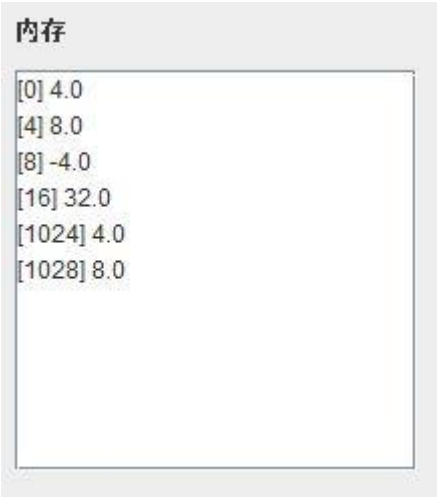
运行时可以单步，也可以多步，步数从左边输入；

Reset 可以清空，Load 为第一次运行前需要点击的（也可以以 Reset 代替）

每一条指令的运行状态，数字为时间



上图为寄存器状态，第一行为依赖的保留站名称



上图为内存状态

Load Queue

Busy	Address	Cache
Free	1024	4.0
Free	1028	8.0
Free	0	

Store Queue

Busy	Address	Cache
Busy	12	0.0
Busy	4	8.0
Busy	8	-4.0

上图为 Load/Store 缓存站

保留站

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
0	Add1	Busy	SUBD	4.0	0.0		Load2
0	Add2	Free		0.0	0.0		
0	Add3	Free		0.0	0.0		
0	Mul1	Busy	MULD	4.0	0.0		Load2
0	Mul2	Free		0.0	0.0		

上图为保留站信息

Time 为对应指令所剩余的运行时间，其他均为作业要求中的内容。