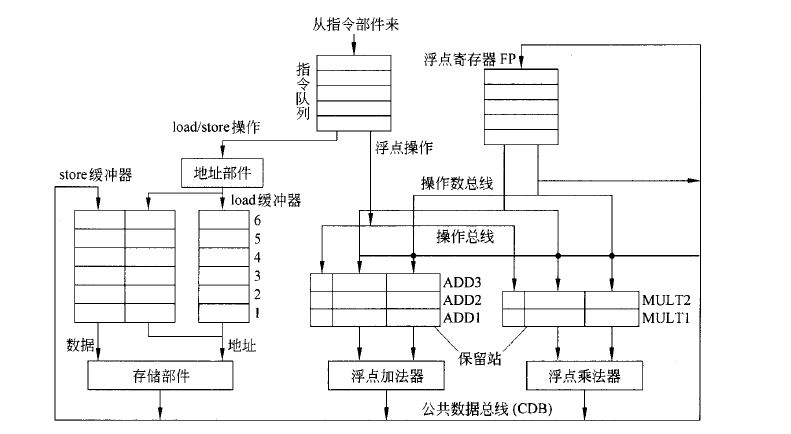
**Tomasulo算法模拟器实验报告**

|  |  |  |
| --- | --- | --- |
| 杨子江 | 2013011710 | [yangzj13@mails.tsinghua.edu.cn](mailto:yangzj13@mails.tsinghua.edu.cn) |
| 王欣然 | 2014011300 | [wangxr1108@126.com](mailto:wangxr1108@126.com) |
| 王倩 | 2014011319 | [q-wang14@mails.tsinghua.edu.cn](mailto: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。

1. 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。

1. 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**
2. 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;

1. 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;

}

1. **执行**
2. 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

1. 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

1. **写结果**
2. 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;

1. 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**