中国科学技术大学计算机学院
《计算机体系结构》实验报告

2021.06.07

实验题目：Tomasulo 软件模拟器和多 Cache 一致性软件模拟器

学生姓名：胡毅翔

学生学号：PB18000290

计算机实验教学中心制

2019 年 9 月

1 实验目的

1. 熟悉 Tomasulo 模拟器和 cache 一致性模拟器（监听法和目录法）的使用。
2. 加深对 Tomasulo 算法的理解，从而理解指令级并行的一种方式-动态指令调度。
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；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。
4. 理解监听法和目录法的基本思想，加深对多 cache 一致性的理解。
5. 做到给出指定的读写序列，可以模拟出读写过程中发生的替换、换出等操作，同时模拟出 cache 块的无效、共享和独占态的相互切换。

2 实验环境

1. PC 一台
2. Windows 10 操作系统
3. Vivado 2019.1
4. Visual Studio Code 1.56.2
5. Tomasulo 模拟器
6. 多 Cache 一致性模拟器

3 Tomasulo 算法模拟器

使用模拟器进行以下指令流的执行并对模拟器截图、回答问题。

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

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。

1. 分别截图（当前周期 2 和当前周期 3），请简要说明 load 部件做了什么改动。

   周期 2：Load 部件：占用 Load2 部件，Busy 置位；R2 就绪，将地址保存在 Load1 部件的地址寄存器。

图 1: 周期 2

   周期 3：Load 部件：Load1 部件将从存储器读到的值保存在 Load1 部件寄存器；R3 就绪，将地址保存在 Load2 部件地址寄存器。

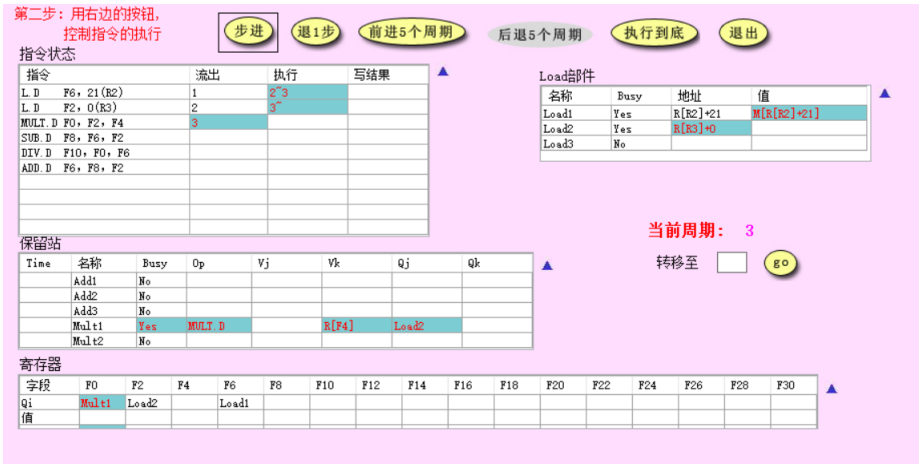

图 2: 周期 3

2. 请截图（MUL.D 刚开始执行时系统状态），并说明该周期相比上一周期整个系统发生了哪些改动（指令状态、保留站、寄存器和 Load 部件）。

   相比上一周期系统发生的改变：

   - 指令状态：发射第 6 条指令；第三条、第四条指令进入执行状态。
   - Load 部件：无变化。
   - 保留站：新发射的 ADD.D 指令占用 Add2 保留站，进入执行的指令 MUL.D 和 SUB.D 开始执行，时间开始倒计时。

- 寄存器：新发射的指令 ADD.D 指令等待 F8 寄存器。

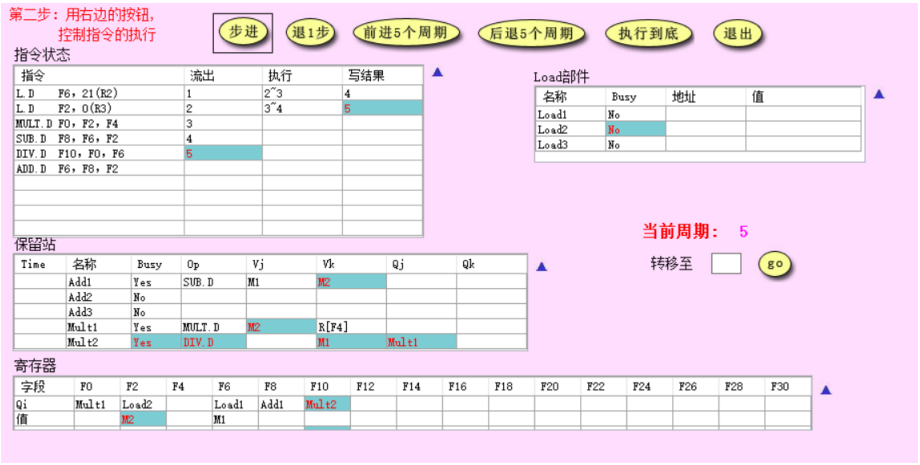

图 3: 周期 5（上一周期）

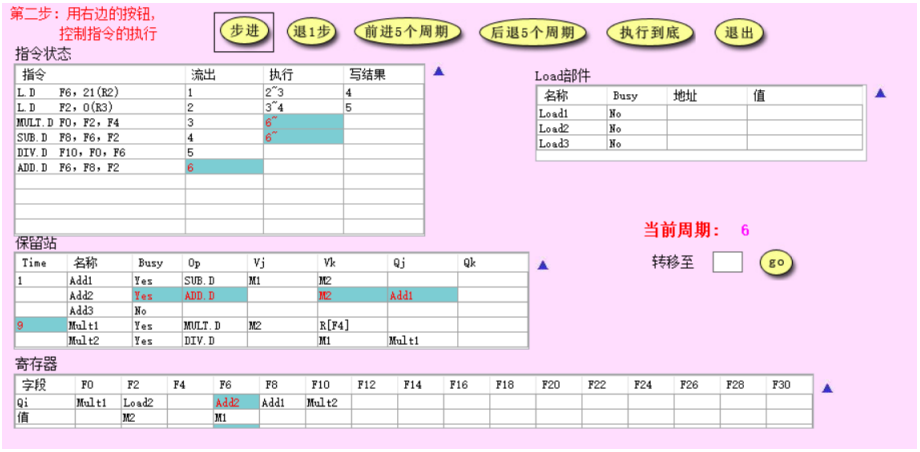

图 4: 周期 6（MUL.D 刚开始执行）

3. 简要说明是什么相关导致 MUL.D 流出后没有立即执行。

   源操作数 F2 为写回，直到第五周期 M2 写入后才就绪。

4. 请分别截图（15 周期和 16 周期的系统状态），并分析系统发生了哪些变化。

   周期 15：指令 ADD.D 和指令 SUB.D 在周期 7 15 周期内执行完毕，将结果写回，释放相应的保留站和寄存器；此时 MULT.D 指令执行了 10 个周期。

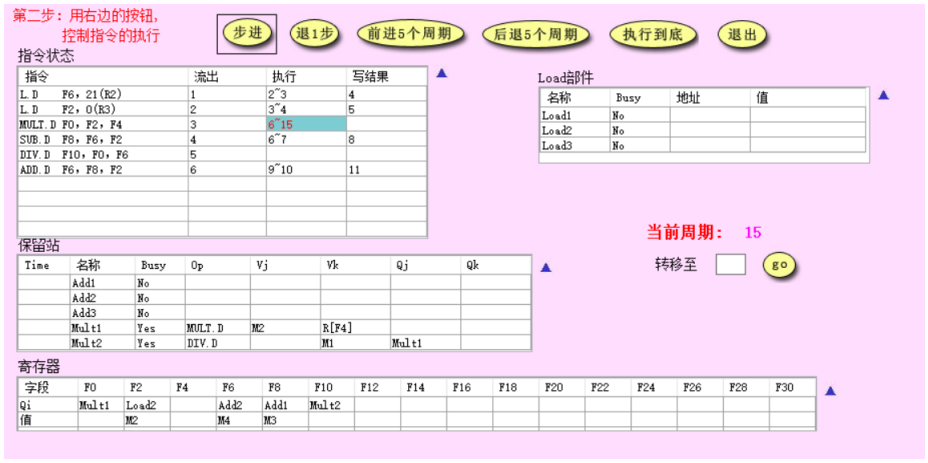

图 5: 周期 15

周期 16：指令 MUL.D 写回结果，释放保留站 CBD 将结果广播到寄存器和指令 DIV.D 对应的保留站。

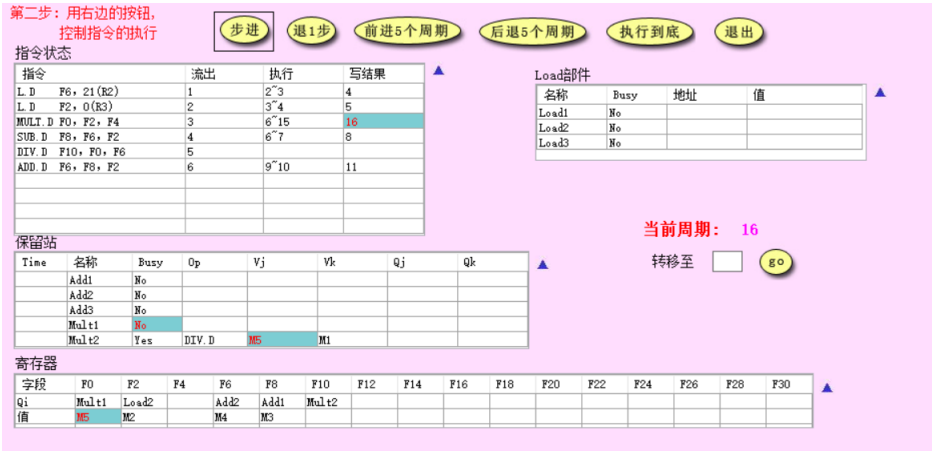

图 6: 周期 16

5. 回答所有指令刚刚执行完毕时是第多少周期，同时请截图（最后一条指令写 CBD 时认为指令流执行结束）。

图 7: 周期 57

所有指令执行完毕是第 57 个周期。

4 多 cache 一致性算法-监听法

1. 利用模拟器进行下述操作，并填写下表。

表 1: 监听法

所进行的访问	是否发生了替换?	是否发生了写回?	监听协议进行的操作与块状态改变
CPU A 读第 5 块	替换 Cache A 的块 1	0	CacheA 发射 Read Miss， 存储器传输第 5 块到 CacheB， CacheA 的块 1 从状态 I 转为 S
CPU B 读第 5 块	替换 CacheB 的块 1	0	CacheB 发射 Read Miss， 存储器传输第五块到 CacheB， CacheB 的块 1 从状态 I 转为 S
CPU C 读第 5 块	替换 CacheC 的块 1	0	CacheC 发射 Read Miss， 存储器传输第 5 块到 CacheC， CacheC 的块 1 从状态 I 转换为 S
CPU B 写第 5 块	0	0	CacheB 发射 Invalidate， CacheA 的块 1 从状态 S 转换到 I， CacheC 的块 1 从状态 S 转换到 I， CacheB 的块 1 从 S 转换到 M

Continued on next page

Continued

CPU D 读第 5 块	替换 CacheD 的块 1	CacheB 的块 1 写回	CacheD 发射 Read Miss， CacheB 写回第 5 块， 存储器传输第 5 块到 CacheD， CacheB 的块 1 从状态 M 转换到 S， CacheD 的块从状态 I 转换到 S
CPU B 写第 21 块	替换 CacheB 的块 1	0	CacheB 发射 Write Miss， 存储器传输第 21 块到 CacheB， CacheB 的块 1 从状态 S 转换为 M
CPU A 写第 23 块	替换 CacheA 的块 3	0	CacheA 发射 Write Miss， 存储器传输第 23 块到 CacheA， CacheA 的块 1 从状态 I 转换到 M
CPU C 写第 23 块	替换 CacheC 的块 3	CacheA 的块 3 写回	CacheC 发射 Read Miss， CacheA 写回第 23 块， 存储器传输第 23 块到 CacheC， CacheA 的块 3 从状态 M 转换到 I， CacheC 的块 3 从状态 I 转换到 M
CPU B 读第 29 块	替换 CacheB 的块 1	CacheB 的块 1 写回	CacheB 写回第 21 块， CacheB 发射 Read Miss， 存储器传输第 29 块到 CacheB， CacheB 的块 1 从状态 M 转换到 S
CPU B 写第 5 块	替换 CacheB 的块 1	0	CacheB 发射 Write Miss， 存储器传输第 5 块到 CacheB， CacheB 的块 1 从状态 S 转换到 M， CacheD 的块 1 从状态 S 转换带 I

2. 请截图，展示执行完以上操作后整个 cache 系统的状态。

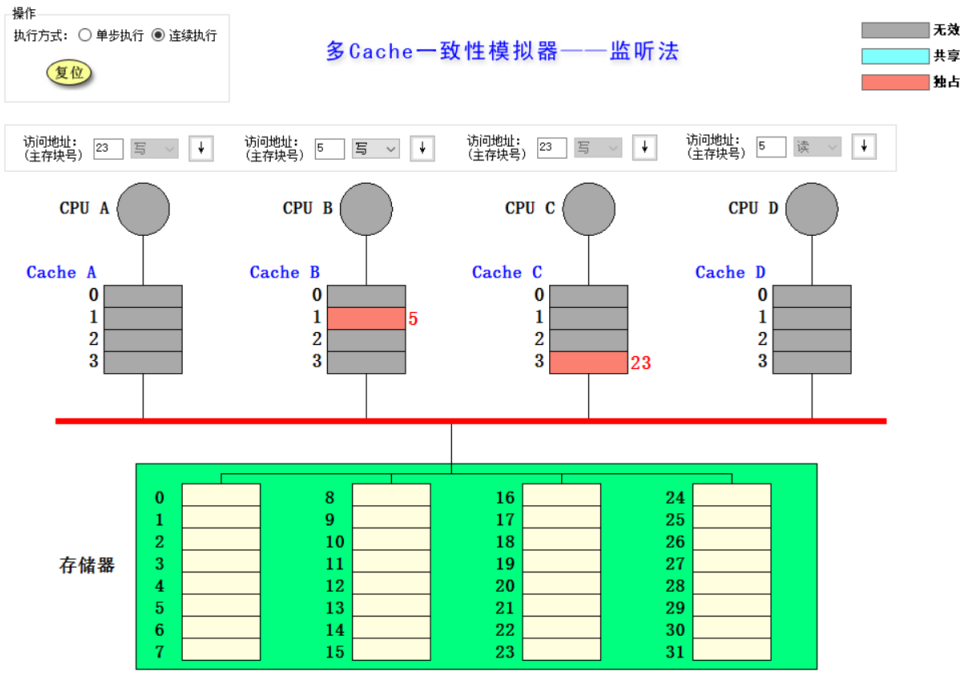

图 8: 执行完毕 cache 系统状态

5 多 cache 一致性算法-目录法

1. 利用模拟器进行下述操作，并填写下表。

表 2: 目录法

所进行的访问	监听协议进行的操作	块状态改变
CPU A 读第 6 块	Cache A 发送 Read Miss 到 Memory A， Memory A 传输第 6 块到 Cache A， Cache A 的块 2 从状态 I 转换为 S	Memory A 的块 6， 状态：U->S， Presence bits：0000->0001， 共享集合 {A}
CPU B 读第 6 块	Cache B 发送 Read Miss 到 Memory A， Memory A 传输第 6 块到 CacheB， Cache B 的块 2 从状态 2 转为 S	Memory A 的块 6， 状态：S->S， Presence bits：0001->0011， 共享集合 {A, B}
CPU D 读第 6 块	Cache D 发送 Read Miss 到 Memory A， Memory A 传输第 6 块到 Cache D， Cache D 的块 2 从状态 I 转为 S	Memory A 的块 6， 状态：S->S， Presence bits: 0011->1011， 共享集合 {A,B,D}

Continued on next page

Continued

CPU B 写第 6 块	Cache B 发送 Write Hit 到 Memory A， Memory A 发送 Invalidate(6) 到 Cache A， Cache A 的块 2 从状态 S 转为 I， Memory A 发送 Invalidate(6) 到到 Cache D， Cache D 的块 2 从状态 S 转换到 I， Cache B 的块 2 从状态 S 转换到 M	Memory A 到块 6， 状态：S->M， Presence bits：1011->0010， 共享集合 {B}
CPU C 读第 6 块	Cache C 发送 Read Miss 到 Memory A， Memory A 发送 Fetch(6) 到 Cache B， Cache B 传输第 6 块到 Memory A， Cache B 的块 2 从状态 M 转为 S， Memory A 传输第 6 块到 Cache C， Cache C 的块 2 从状态 I 转为 S	Memory A 的块 6， 状态：M->S， Presence bits：0010->0110， 共享集合：{B,C}
CPU D 写第 20 块	Cache D 发送 Write Miss 到 Memory C， Memory C 传输第 20 块到 Cache D， Cache D 的块 0 从状态 I 转为 M	Memory C 的块 20， 状态：U->M， Presence bits：0000->1000， 共享集合 {D}
CPU A 写第 20 块	Cache A 发送 Write Miss 到 Memory C， Memory C 发送 Fetch &Invalidate(20) 到 Cache D， Cache D 传输第 20 块到 Memory C， Cache D 的块 0 从状态 M 转为 I， Memory C 传输第 20 块到 Cache A， Cache A 的块 0 从状态 I 转换到 M	Memory C 的块 20， 状态：M->M， Presence bits：1000->0001， 共享集合 {A}
CPU D 写第 6 块	Cache D 发送 Write Miss 到 Memory A， Memory A 发送 Invalidate(6) 到 Cache B， Cache B 的块 2 从状态 S 转为 I， Memory A 传输第 6 块到 Cache D， Cache D 的块 2 从状态 I 转为 M	Memory A 的块 6， 状态：S->M， Presence bits：0110->1000， 共享集合 {D}

Continued on next page

Continued

CPU A 读第 12 块	Cache A 发送 Write Back 到 Memory C， Cache A 的块 0 从状态 M 转为 I， Cache A 发送 Read Miss 到 Memory B， Memory B 传输第 12 块到 Cache A， Cache A 的块 0 从状态 I 转为 S	Memory C 的块 20， 状态：M->U， Presence bits：0001->0000， 共享集合 {}； Memory B 的块 12， 状态：U->S， Presence bits：0000->0001， 共享集合：{A}

2. 请截图，展示执行完以上操作后整个 cache 系统的状态。

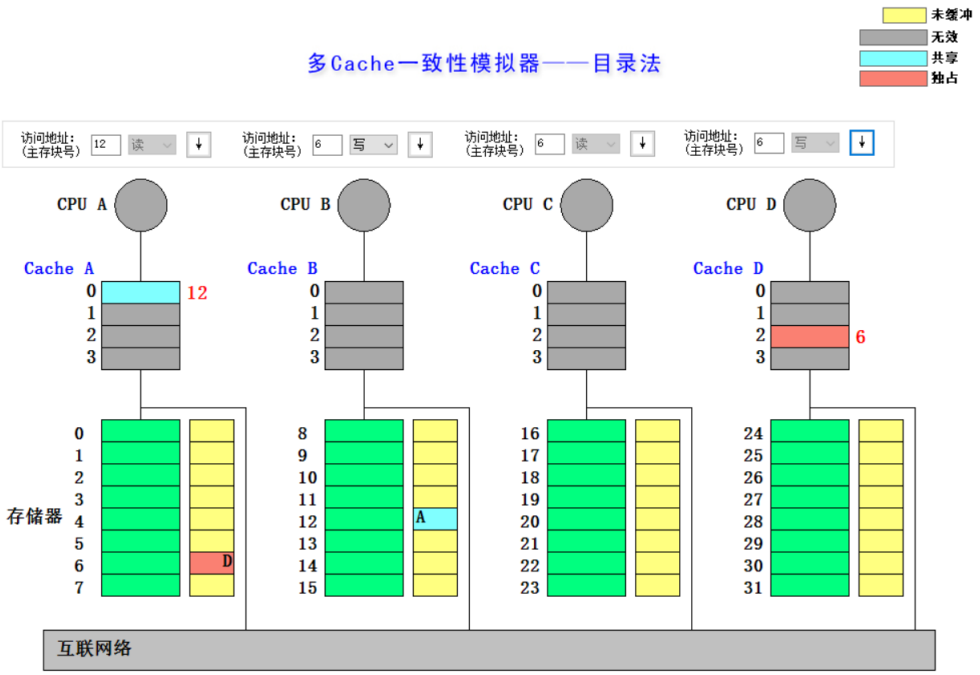

图 9: 执行完毕 cache 系统状态

6 综合问答

1. 目录法和监听法分别是集中式和基于总线，两者优劣是什么？（言之有理即可）

   监听法：

   优点：保证了 Cache 的一致性，实现了写互斥和写串行

   缺点：

   - 扩展性差，总线上能够连接的处理器数目有限

- 存在总线竞争问题
- 总线的带宽会带来一些限制
- 在非总线和或环形网络上监听困难
- 总线事务多，通信开销大

目录法：

优点：

- 拓展性强，可以连接的处理器数目更多
- 降低了对于总线带宽的占用
- 可以有效地适应交换网络进行通信

缺点：

- 需要额外的空间来存储 Presence Bits，当处理器数目较多的时候会有很大的存储开销
- 总线竞争
- 存储器接口通信压力大，存储器速度成为限制

2. Tomasulo 算法相比 Score Board 算法有什么异同？(简要回答两点：1. 分别解决了什么相关，2. 分别是分布式还是集中式) (参考第五版教材)

Tomasulo 算法

解决的相关：

- WAR 相关：使用 RS 的寄存器或指向 RS 的指针代替指令中的寄存器-寄存器重命名
- WAW 相关：使用 RS 中的寄存器值或指向 RS 的指针代替指令中的寄存器
- RAW 相关：检测到寄存器就绪即没有冲突再读取操作数，进入执行阶段
- 结构相关：有结构冲突不发射
- 结果 Forward：从 FU 广播结果到 RS 和寄存器

特点: 分布式；指令状态、相关控制和操作数缓存分布在各个部件中 (保留站)

Score Board 算法

- WAR 相关：对操作排队，仅在读操作数阶段读寄存器
- WAW 相关：检测到相关后，停止发射前一条指令，直到前一条指令完成
- RAW 相关：检测到没有冲突 (寄存器就绪) 再读取操作数，进入执行阶段
- 结构相关：有结构相关不发射
- 结果 Forward：写回寄存器接触等待

3. Tomasulo 算法是如何解决结构、RAW、WAR 和 WAW 相关的？（参考第五版教材）

   结构相关：有结构冲突不发射

   RAW 相关：检测到没有冲突，即存储器就绪再读取操作操作数，进入执行阶段

   WAW 相关：使用 RS 中的寄存器值或指向 RS 的指针代替指令中的寄存器-寄存器重命名

   WAR 相关：使用 RS 中的寄存器值或指向 RS 的指针代替指令中的寄存器-寄存器重命名