Verilog 实验 lab6 实验报告

PB19071405 王昊元

2022 年 06 月 04 日

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 实验要求

Tomasulo 算法模拟器 使用模拟器进行指令流的执行并对模拟器截图、回答问题

多 cache 一致性算法-监听法 利用模拟器进行下述操作，填写下表，并截图，展示执行完以上操作后整个 cache 系统的状态。

多 cache 一致性算法-目录法 利用模拟器进行下述操作，填写下表，并截图，展示执行完以上操作后整个 cache 系统的状态。

3 实验环境

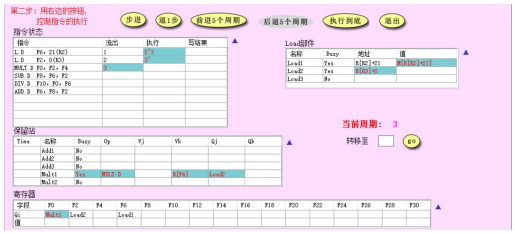
- 电三楼 406 机房电脑
- Windows 7

4 实验结果及问题回答

4.1 Tomasulo 算法模拟器

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

(a) 当前周期 2

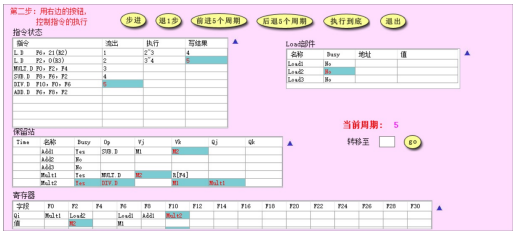


(b) 当前周期 3

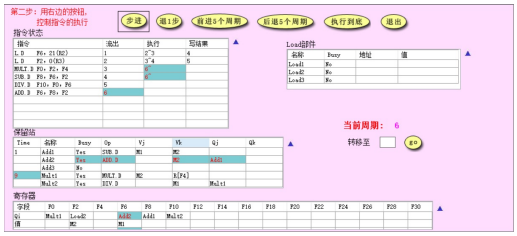
(a) 对于当前周期 2，Load2 部件被占用，其 Busy 置为有效；存储器 R2 就绪，将偏移运算后的目标地址存于 Load1 部件的地址寄存器中。

(b) 对于当前周期 3，Load1 部件根据地址寄存器中地址，从存储器 R2 中对应的地址读取值至 Load1 部件的值寄存器中；存储器 R3 就绪，将偏移运算后的目标地址存于 Load2 部件的地址寄存器中。

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）



(c) MUL.D 上一周期



(d) MUL.D 刚开始执行时的周期

- 指令状态：第 6 条指令流出，第 3、4 条指令开始执行
- 保留站：新流出的第 6 条 ADD.D 类型的指令占用保留站 Add2，第 3 条指令 MUL.D 和第 4 条指令 SUB.D 开始执行，同时二者各自的执行时间开始倒计时
- 寄存器：新发射的第 6 条 ADD.D 类型的指令等待 F8 寄存器就绪，准备向 F6 寄存器中写入值
- Load 部件：无改动

3. 简要说明是什么相关导致 MUL.D 流出后没有立即执行
从第 2 条指令（`L.D F2, 0(R3)`）到第 3 条指令（`MUL.D F0, F2, F4`）间存在的关于 F2 的 RAW 相关导致 MUL.D 流出后没有立即执行，MUL.D 需要等 M2 写入寄存器 F2 后才能执行。

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

(e) 15 周期

(f) 16 周期

(a) 第 15 周期：MULT.D 执行完毕

(b) 第 16 周期：释放保留站 Mult1，其 Busy 位置为无效；将指令 MULT.D 的结果 M5 写 CBD，同时广播至寄存器 F0 和指令 DIV.D 对应的保留站 Mult2

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）

所有指令刚刚执行完毕时是第 57 个周期。

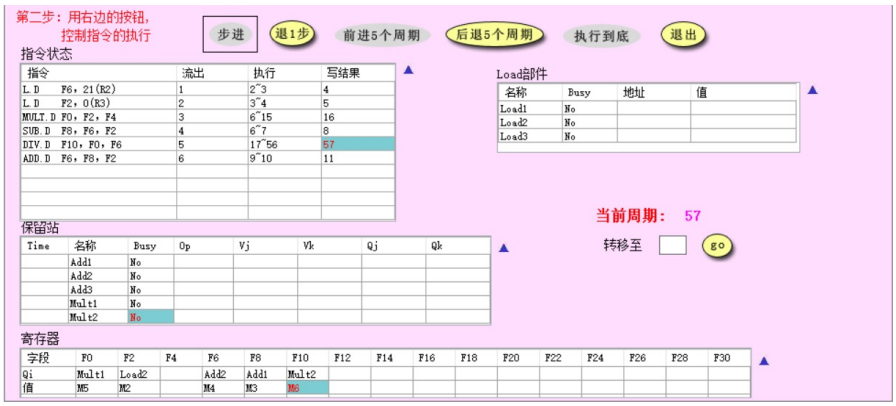


图 1: 所有指令刚刚执行完毕

4.2 多 cache 一致性算法-监听法

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	是	否	Cache A 发射 Read Miss，存储器传输第 5 块到 Cache A， 替换 Cache A 的块 1，Cache A 的块 1 从状态 I 转为 S 对 Cache A 的块 1 进行读操作
CPU B 读第 5 块	是	否	Cache B 发射 Read Miss，存储器传输第 5 块到 Cache B， 替换 Cache B 的块 1，Cache B 的块 1 从状态 I 转为 S 对 Cache B 的块 1 进行读操作
CPU C 读第 5 块	是	否	Cache C 发射 Read Miss，存储器传输第 5 块到 Cache C， 替换 Cache C 的块 1，Cache C 的块 1 从状态 I 转为 S 对 Cache C 的块 1 进行读操作
CPU B 读第 5 块	否	否	Cache B 发射 Invalidate，Cache A 的块 1 从状态 S 转换到 I， Cache C 的块 1 从状态 S 转换到 I，对 Cache B 的块 1 进行写操作，Cache B 的块 1 从状态 S 转换到 M
CPU D 读第 5 块	是	是	Cache D 发射 Read Miss，写回 Cache B 的块 1 至存储器的第 5 块， Cache B 的块 1 从状态 M 转为 S，存储器传输第 5 块到 Cache D， 替换 Cache D 的块 1，Cache D 的块 1 从状态 I 转为 S 对 Cache D 的块 1 进行读操作
CPU B 读第 21 块	是	否	Cache B 发射 Write Miss，存储器传输第 21 块到 Cache B， 替换 Cache B 的块 1，Cache B 的块 1 从状态 S 转换为 M，对 Cache B 的块 1 进行写操作
CPU A 读第 23 块	是	否	Cache A 发射 Write Miss，存储器传输第 23 块到 Cache A， 替换 Cache A 的块 3，Cache A 的块 3 从状态 I 转换为 M，对 Cache A 的块 3 进行写操作
CPU C 读第 23 块	是	是	Cache C 发射 Write Miss，写回 Cache A 的块 3 至存储器的第 23 块， Cache A 的块 3 从状态 M 转为 I，存储器传输第 23 块到 Cache C， 替换 Cache C 的块 3，Cache C 的块 3 从状态 I 转为 M，对 Cache C 的块 3 进行写操作
CPU B 读第 29 块	是	是	写回 Cache B 的块 1 至存储器的第 21 块，Cache B 的块 1 从状态 M 转换为 I， Cache B 发射 Read Miss，存储器传输第 29 块到 Cache B， 替换 Cache B 的块 1，Cache B 的块 1 从状态 I 转换为 S 对 Cache B 的块 1 进行读操作
CPU B 读第 5 块	是	否	Cache B 发射 Write Miss，存储器传输第 5 块到 Cache B，替换 Cache B 的块 1， Cache B 的块 1 从状态 S 转换为 M，Cache D 的块 1 从状态 S 转换为 I，对 Cache B 的块 1 进行写操作

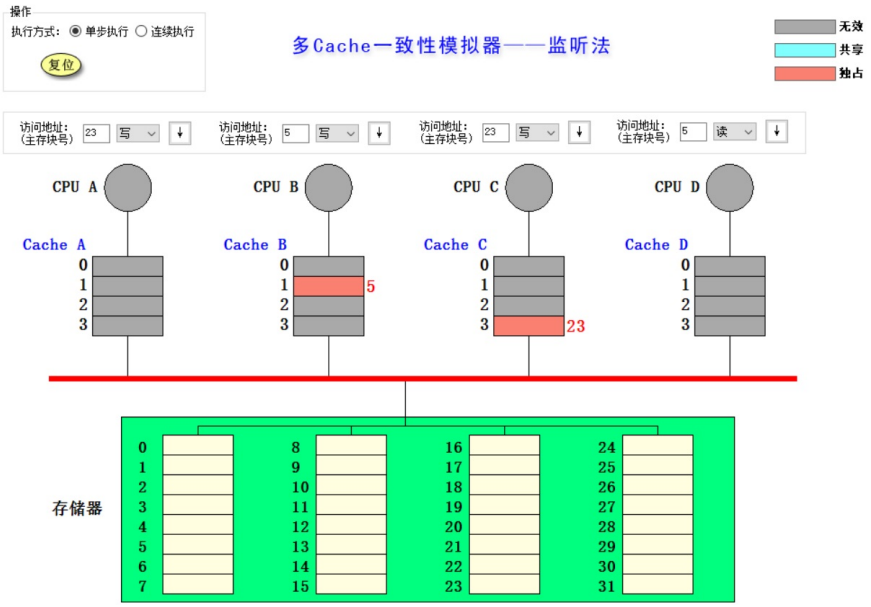


图 2: 监听法执行完后的状态

4.3 多 cache 一致性算法-目录法

表 2: 目录法

所进行的访问	监听协议进行的操作与块状态改变
CPU A 读第 6 块	Cache A 发射 Read Miss 到 Memory A，Memory A 传输第 6 块到 Cache A，替换 Cache A 的块 2， Cache A 的块 2 从状态 I 转为 S； Memory A 的块 6 由状态 U 转为 S，其 Presence bits 由 0000 变为 0001，其共享集合变为 {A}； 对 Cache A 的块 2 进行读操作
CPU B 读第 6 块	Cache B 发射 Read Miss 到 Memory A，Memory A 传输第 6 块到 Cache B，替换 Cache B 的块 2， Cache B 的块 2 从状态 I 转为 S； Memory A 的块 6 状态仍为 S，Presence bits 由 0001 变为 0011，共享集合变为 {A,B}； 对 Cache B 的块 2 进行读操作
CPU D 读第 6 块	Cache D 发射 Read Miss 到 Memory A，Memory A 传输第 6 块到 Cache D，替换 Cache D 的块 2， Cache D 的块 2 从状态 I 转为 S； Memory A 的块 6 状态仍为 S，Presence bits 由 0011 变为 1011，共享集合变为 {A,B,D}； 对 Cache D 的块 2 进行读操作
CPU B 读第 6 块	Cache B 发射 Write Hit 到 Memory A，Memory A 发射 Invalidate(6) 到 Cache A 和 Cache D， Cache A 的块 2 从状态 S 转为 I，Cache D 的块 2 从状态 S 转为 I，Cache B 的块 2 从状态 S 转为 M； Memory A 的块 6 由状态 S 转为 M，Presence bits 由 1011 变为 0010，共享集合变为 {B}； 对 Cache B 的块 2 进行写入
CPU C 读第 6 块	Cache C 发射 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，替换 Cache C 的块 2， Cache C 的块 2 从状态 I 转为 S； Memory A 的块 6 由状态 M 转为 S，Presence bits 由 0010 变为 0110，共享集合变为 {B,C}； 对 Cache C 的块 2 进行读操作
CPU D 读第 20 块	Cache D 发射 Write Miss 到 Memory C， Memory C 传输第 20 块到 Cache D，替换 Cache D 的块 0，Cache D 的块 0 从状态 I 转为 M； Memory C 的块 20 由状态 U 转为 M，Presence bits 由 0000 变为 1000，共享集合变为 {D}； 对 Cache D 的块 0 进行写入
CPU A 读第 20 块	Cache A 发射 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，替换 Cache A 的块 0，Cache A 的块 0 从状态 I 转为 M； Memory C 的块 20 状态仍为 M，Presence bits 由 1000 变为 0001，共享集合变为 {A}； 对 Cache A 的块 0 进行写入
CPU D 读第 6 块	Cache D 发射 Write Miss 到 Memory A，Memory A 发射 Invalidate(6) 到 Cache B 和 Cache C， Cache B 的块 2 从状态 S 转为 I，Cache C 的块 2 从状态 S 转为 I， Memory A 传输第 6 块到 Cache D，替换 Cache D 的块 2，Cache D 的块 2 从状态 I 转为 M； Memory A 的块 6 由状态 S 转为 M，Presence bits 由 0110 变为 1000，共享集合变为 {D}； 对 Cache D 的块 2 进行写入
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 发射 Read Miss 到 Memory B， Memory B 传输第 12 块到 Cache A，替换 Cache A 的块 0，Cache A 的块 0 从状态 I 转为 S； Memory C 的块 20 由状态 M 转为 U，其 Presence bits 由 0001 变为 0000，其共享集合变为 $\emptyset$； Memory B 的块 12 由状态 U 转为 S，其 Presence bits 由 0000 变为 0001，其共享集合变为 {A}； 对 Cache A 的块 0 进行读操作

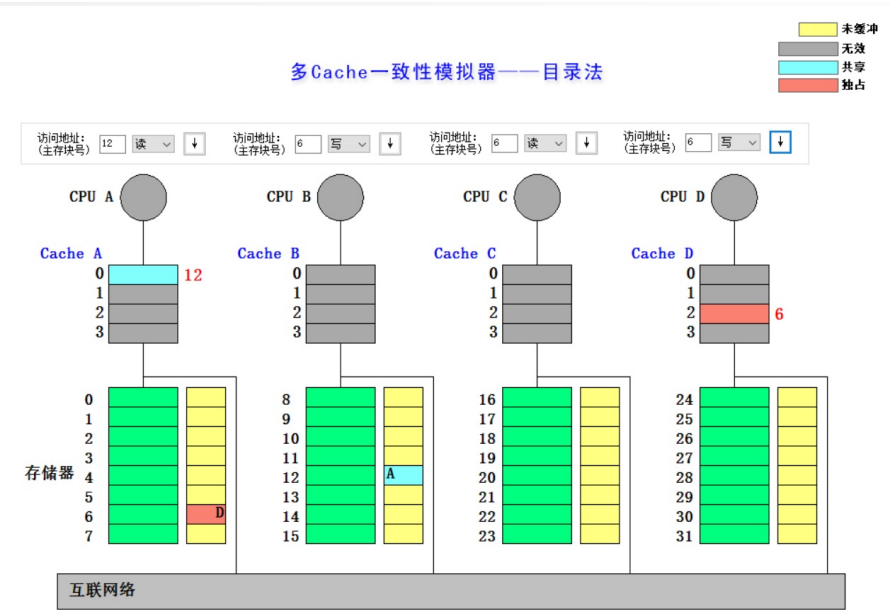


图 3: 目录法执行完后的状态

5 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）

   (a) 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
     - 存在总线竞争问题
     - 总线事务多，通信开销大
     - 总线的带宽受限
     - 扩展性差，总线上能够连接的处理器数目有限
     - 在非总线和或环形网络上监听困难

   (b) 目录法

   - 优点
     - 降低了对于总线带宽的占用
     - 拓展性强，可以连接的处理器数目更多
     - 可以有效地适应交换网络进行通信
   - 缺点
     - 存在总线竞争问题
     - 需要额外的空间来存储 Presence Bits，当处理器数目较多的时，存储开销较大
     - 存储器接口通信压力大，存储器速度限制传输速度

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 算法为分布式；其指令状态、相关控制和操作数缓存分布在各个部件中 (保留站)。其核心思想是：记录和检测指令相关，操作数一旦就绪就立即执行。把发生 RAW(写后读) 冲突的可能性减少到最少；通过寄存器换名来消除 WAR(读后写) 和 WAW(写后写) 冲突；指令若存在结构冲突，则不予以发射。
   - Score Board 算法为集中式；其不能解决 WAR 和 WAW 相关，只能检测到 WAR 和 WAW 相关，它是通过将后一条指令暂停来克服此类问题的。

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：等待至写入完成，待读取的寄存器中的数据就绪，冲突消除后，再进入读指令执行阶段
   - 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