

Lab5实验报告

PB17111623

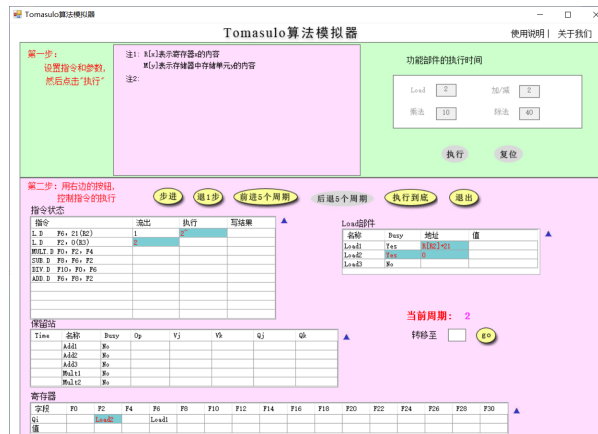
范睿

Tomasulo

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:

○

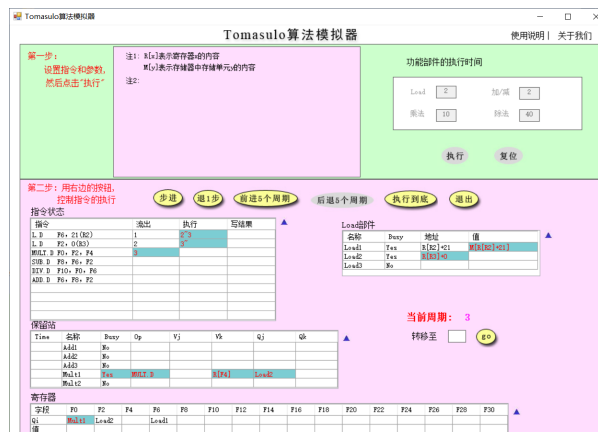


- 第二个周期，第一条load指令开始读内存，第二条load指令留出，此时两条load指令使用了两个Load部件。

- Load1部件的地址获得M[R2]+21
- Load2部件Busy变为Yes，但是还没有获取地址

- 周期3:

○

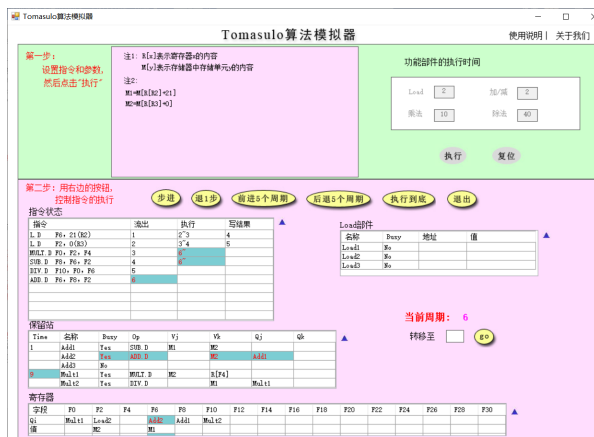


- 第三个周期，第一条load指令结果返回，第二条load指令开始读内存

- Load1部件的值位置被写入load出来的值
- Load2部件的地址为值获得M[R3]+0

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）

MUL.D刚开始执行时的系统状态为：



Mul刚开始执行的周期是第6个周期：

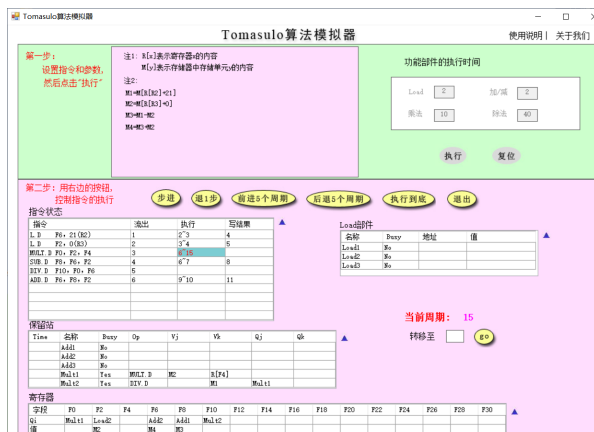
- SUB也开始执行
- ADD指令流出，占用保留张Add2资源，Add2计算资源等待Add1返回的值
- 寄存器F6的值将来自于Add2

3. 简要说明是什么相关导致MUL.D流出后没有立即执行

关于F2的写后读相关导致MUL.D流出后没有立即执行

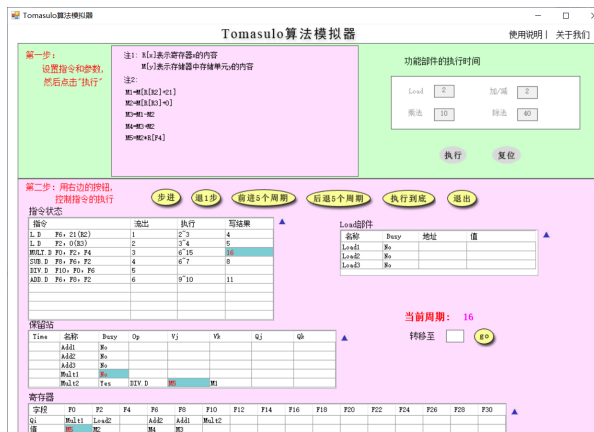
4. 请分别截图（15周期和16周期的系统状态），并分析系统发生了哪些变化

第15周期：



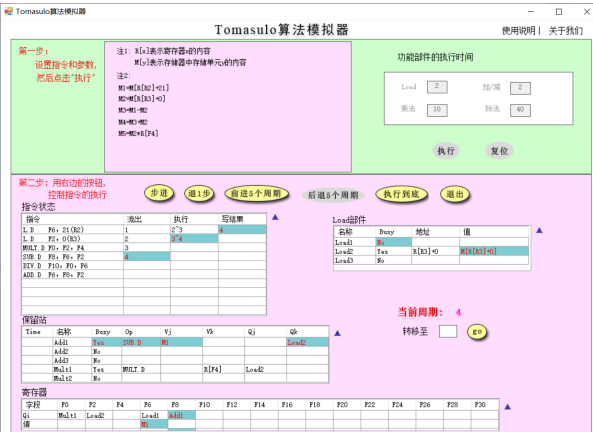
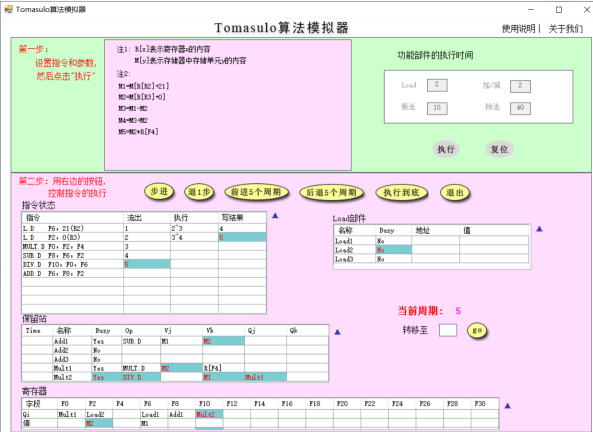
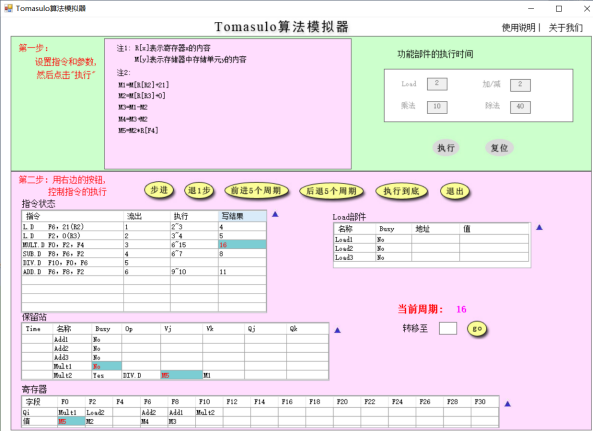
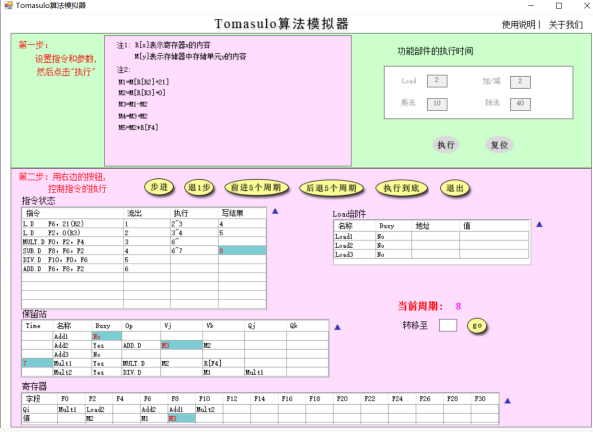
- MULT.D指令执行结束

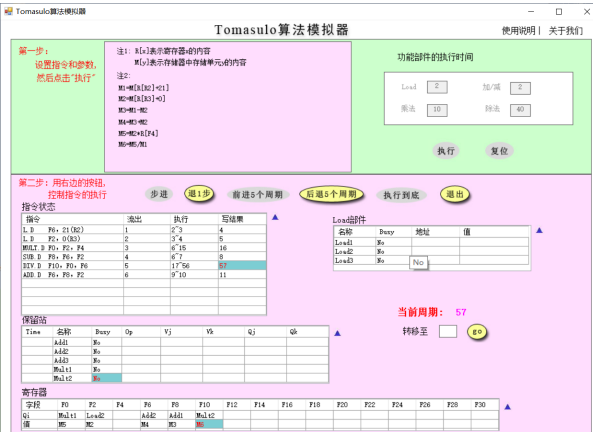
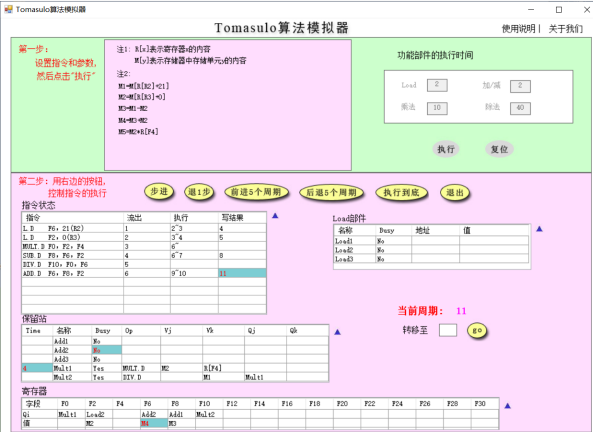
第16周期：



- MULT.D指令写结果，保留站Mult1资源被释放
- Mult2资源获得第一个操作数M5
- 寄存器F0获得值M5

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）

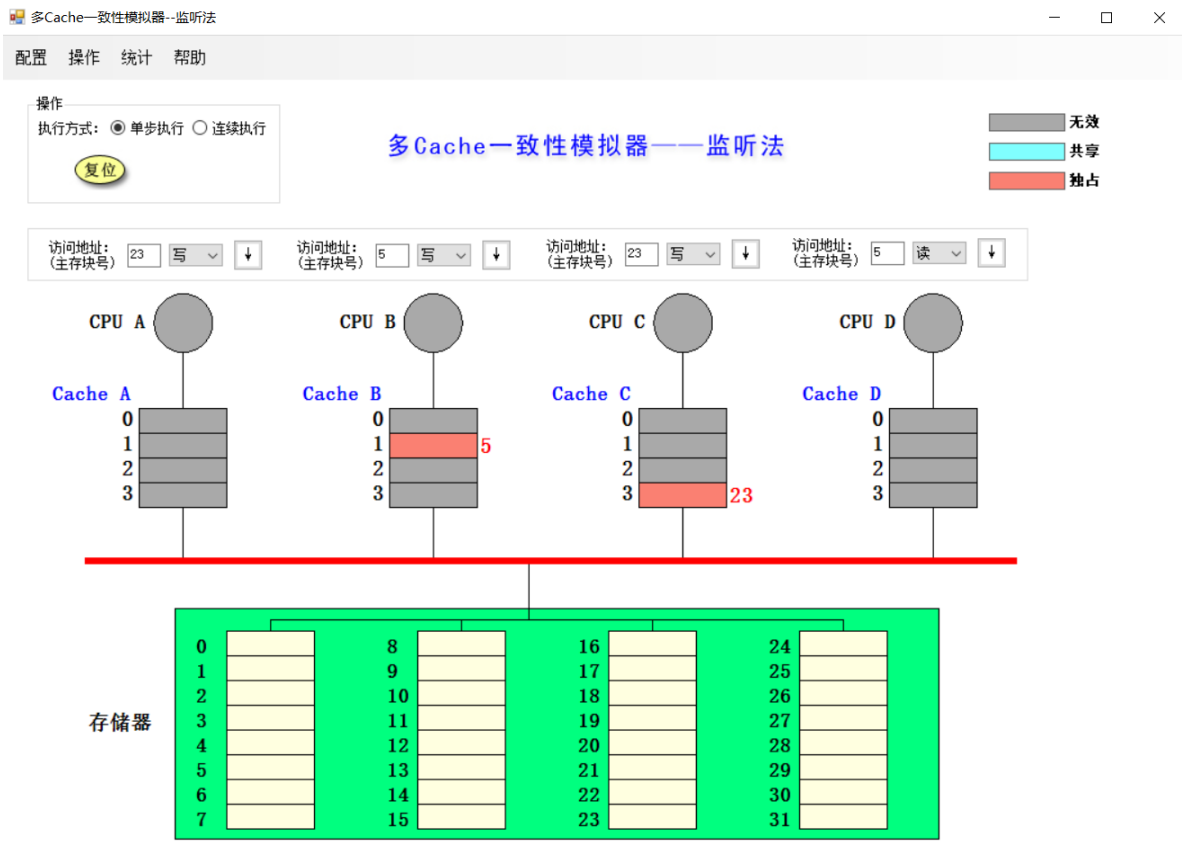
指令	结束周期	截图
L.D F6, 21 (R2)	4	
L.D F2, 0 (R3)	5	
MUL.D F0, F2, F4	14	
SUB.D F8, F6, F2	8	

指令	结束周期	截图
DIV.D F10, F0, F6	57	
ADD.D F6, F8, F2	11	

Cache一致性 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的改变
CPU A 读第5块	否	否	第5块从主存进入CPU A的缓存中，状态为共享
CPU B 读第5块	否	否	第5块从主存进入CPU B的缓存中，状态为共享
CPU C 读第5块	否	否	第5块从主存进入CPU C的缓存中，状态为共享
CPU B 写第5块	否	否	CPU B写命中，总线发出作废请求，在A和C中的第5块作废，B将新值写入第5块，状态变为独占
CPU D 读第5块	否	是	B将第5块写回存储器，状态变为共享；第5块从主存进入CPU D的缓存中，状态为共享
CPU B 写第21块	是	否	第21块从主存进入CPU B的缓存中，将第5块的位置挤占，并写入新值，状态变为独占
CPU A 写第23块	否	否	第21块从主存进入CPU A的缓存中，并写入新值，状态变为独占
CPU C 写第23块	否	是	先将23块从CPU A中被写回主存，再将23块从主存拿入CPU C，写入新值，状态变为独占
CPU B 读第29块	是	是	先将21块写回主存，再将29块拿进CPU B的缓存中，状态变为共享
CPU B 写第5块	是	否	将第5块装入CPU B的缓存中，将第29块的位置挤占，D中的第5块坐在位置变为无效，写入新值，状态变为独占

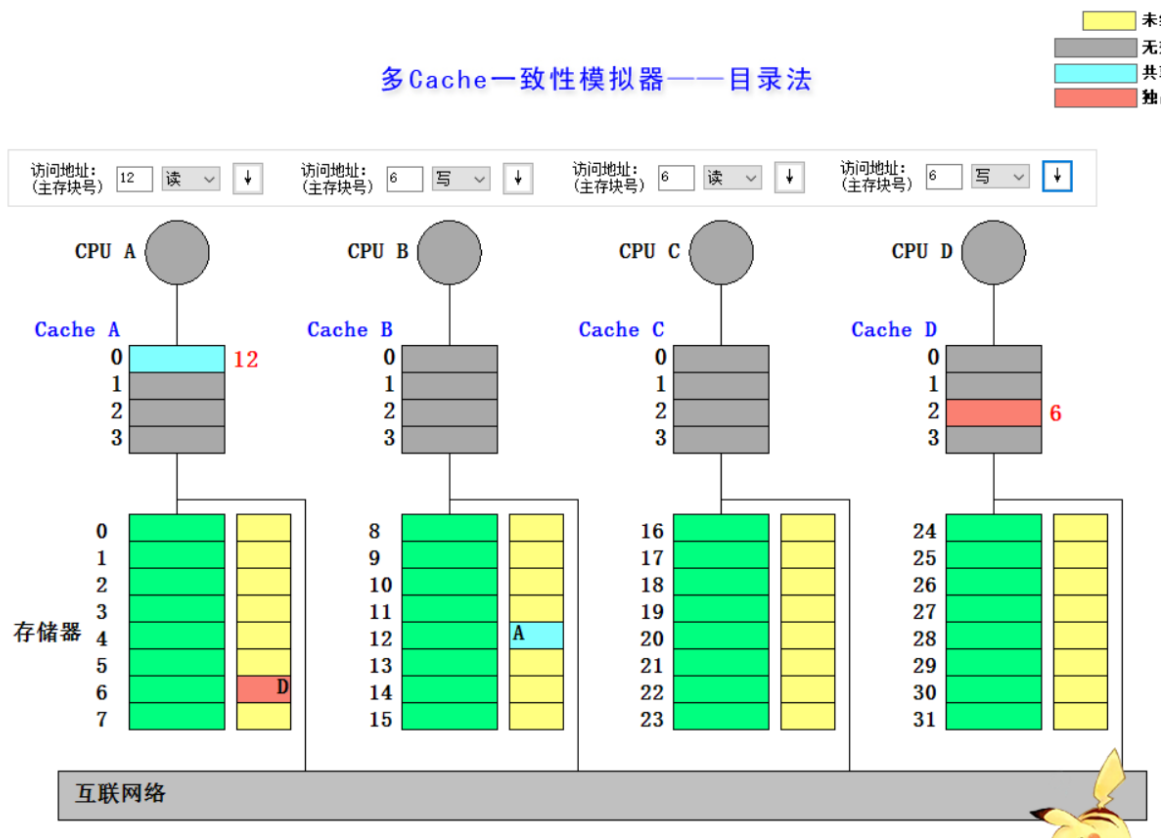
截图：



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	CPU A向宿主发送读取请求， 宿主返回数据并将共享集合设为A， 状态为共享
CPU B读第6块	CPU B通过互联网络向宿主发送读取请求， 宿主通过互联网络返回数据， 共享集合为AB， 状态为共享
CPU D 读第6块	CPU D通过互联网络向宿主发送读取请求， 宿主通过互联网络返回数据， 共享集合为ABD， 状态为共享
CPU B 写第6块	CPU B通过互联网络向宿主发送写命中请求， 宿主向所有非B的共享集合发送作废命令， AD中的第6块变为无效， 共享集合变为B， CPU B将新值写入缓存， 状态为独占
CPU C 读第6块	CPU C通过互联网络向宿主发送读失效请求， 宿主检查共享集合后发现该块在B中独占， 于是先向B发送读取请求， B中的值进入宿主， B中第6块状态变为共享， 宿主再将第6块的值返回给C， 状态为共享， 共享集合变为BC
CPU D写第20块	CPU D通过互联网络向宿主发送写不命中请求， 宿主将20块的值发送给D， 共享集合变为D， 状态变为独占， D将新值写入块20
CPU A写第20块	CPU A通过互联网络向宿主发送读失效请求， 宿主检查共享集合后， 向CPU D发送作为命令， 将D中的20块的值拿回到宿主中， 然后将20块的值发送给A， 共享集合变A， 状态为独享。A将新值写入20块
CPU D写第6块	CPU D向宿主发送写不命中请求， 宿主检查共享集合后， 将BC中的第6块作废， 然后将第6块的值发送给D， 共享集合变为D， 状态变为独占， D将新值写入缓存。
CPU A 读第12块	CPU A发现第12块应该在的位置已经有了第20块， 且状态为独占， 所以先将第20块写回主存。然后向宿主发送读不命中请求， 12块的值经过互联网络到达A中， 共享集合为A， 状态为共享， A读取。

多Cache一致性模拟器——目录法



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？

监听法：

- 优点：小型体系中，总线的交流很少，不会把太多时间浪费在不同宿主的沟通之间。
- 缺点：对于CPU数量多的架构，总线压力会很大，性能下降很快。

目录法：

- 优点：可以支持大型体系
- 缺点：宿主与缓存的沟通次数多，对于多写的操作序列不友好。

2. Tomasulo算法相比Score Board算法有什么异同？

不同之处：

- 解决的WAW相关的方式不同
 - Tomasulo通过在Issue阶段的寄存器换名来解决
 - ScoreBoard通过在Issue阶段检测没有其他活动的指令使用相同目的寄存器时才发射指令来解决
- 写结果的方式不同
 - Tomasulo可以直接将结果写入保留站中
 - ScoreBoard只能将结果写入寄存器，再从寄存器中读入数据

相同之处：

- 两者都通过动态调度来消除RAW相关

Tomasulo：分布式；解决了结构相关、RAW、WAR、WAW

ScoreBoard：集中式；解决了结构相关、RAW、WAW、WAR

3. Tomasulo算法是如何解决结构、RAW、WAR和WAW相关的？

- 结构相关：只有当计算资源可用时才将指令发射
- RAW：Execution阶段中，如果操作数没有准备好则检测Common Data Bus直到等到操作数，避免RAW相关
- WAR和WAW相关：寄存器改名