

# lab5 Tomasulo 和 cache 一致性

PB18111793 裴启智

## 实验目的

- 熟悉Tomasulo模拟器和cache一致性模拟器（监听法和目录法）的使用
- 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
- 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。
- 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
- 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

## 实验内容

### 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

L.D F6, 21 (R2)

L.D F2, 0 (R3)

MUL.D F0, F2, F4

SUB.D F8, F6, F2

DIV.D F10, F0, F6

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/store 2 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
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）
3. 简要说明是什么相关导致 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
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）

- 指令流

L.D F6, 21 (R2)

L.D F2, 0 (R3)

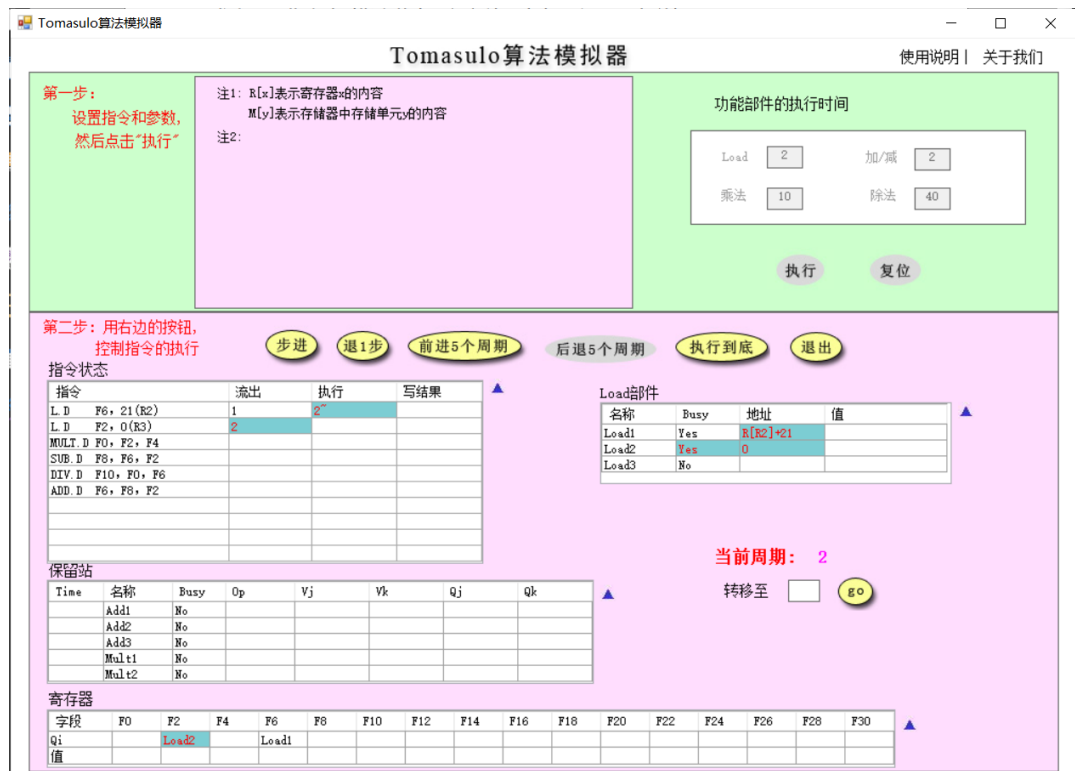
MUL.D F0, F2, F4

SUB.D F8, F6, F2

DIV.D F10, F0, F6

ADD.D F6, F8, F2

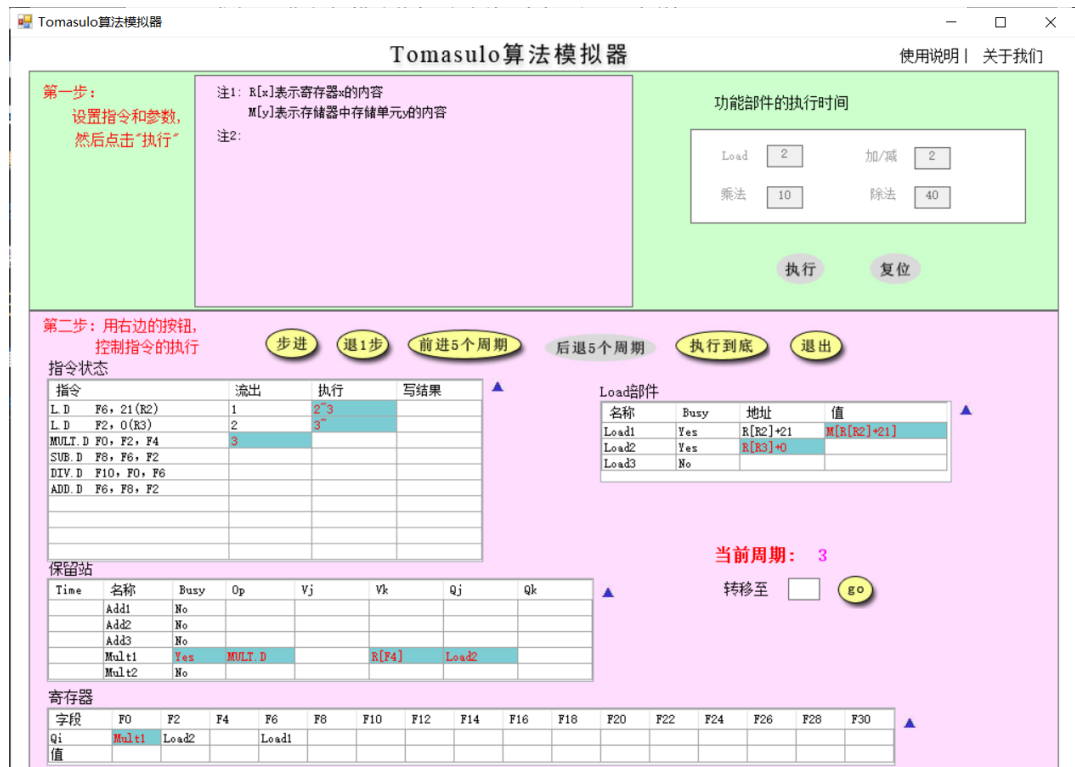
1. 分别截图（当前周期2和当前周期3），请简要说明load部件做了什么改动
  - 当前周期2



第二条 L.D 指令占用部件 Load2, 被置为 Busy 状态

第一条 L.D 指令计算访存地址  $R[R2] + 21$ , 并存入 Load1 部件的地址寄存器

- 当前周期3

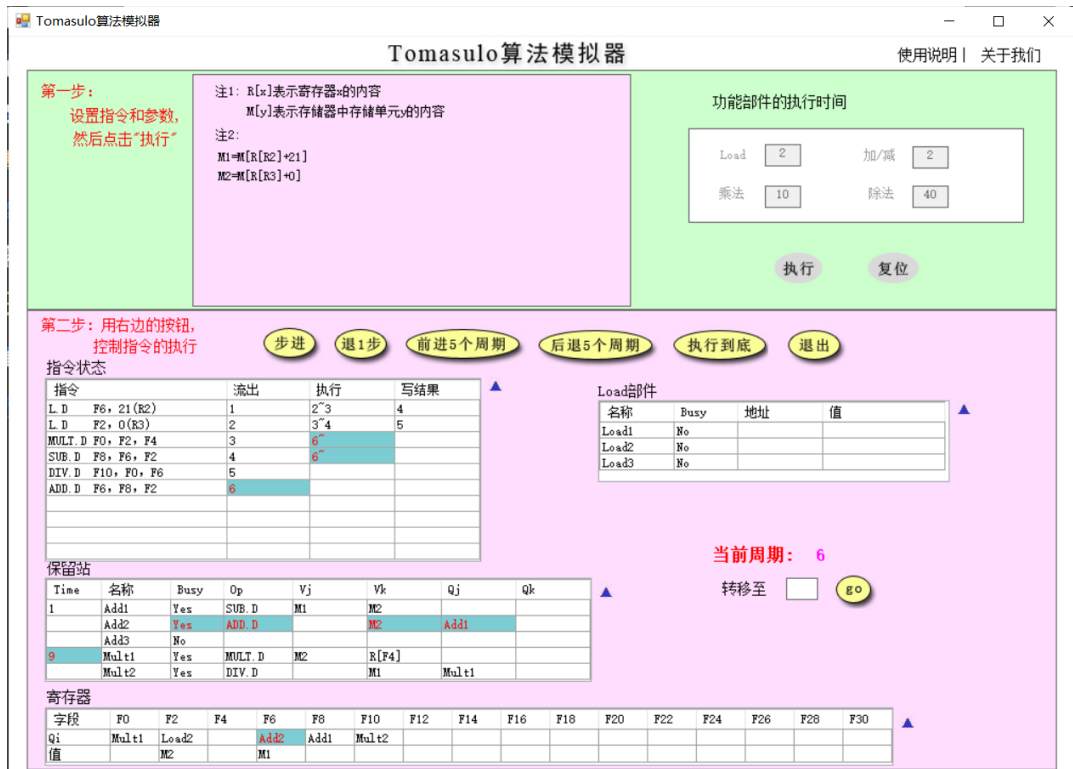


Load1 部件将存从存储器读到的值保存到 Load1 部件的值寄存器中

第二条 L.D 指令计算访存地址  $R[R3] + 0$ , 并存入 Load2 部件的地址寄存器

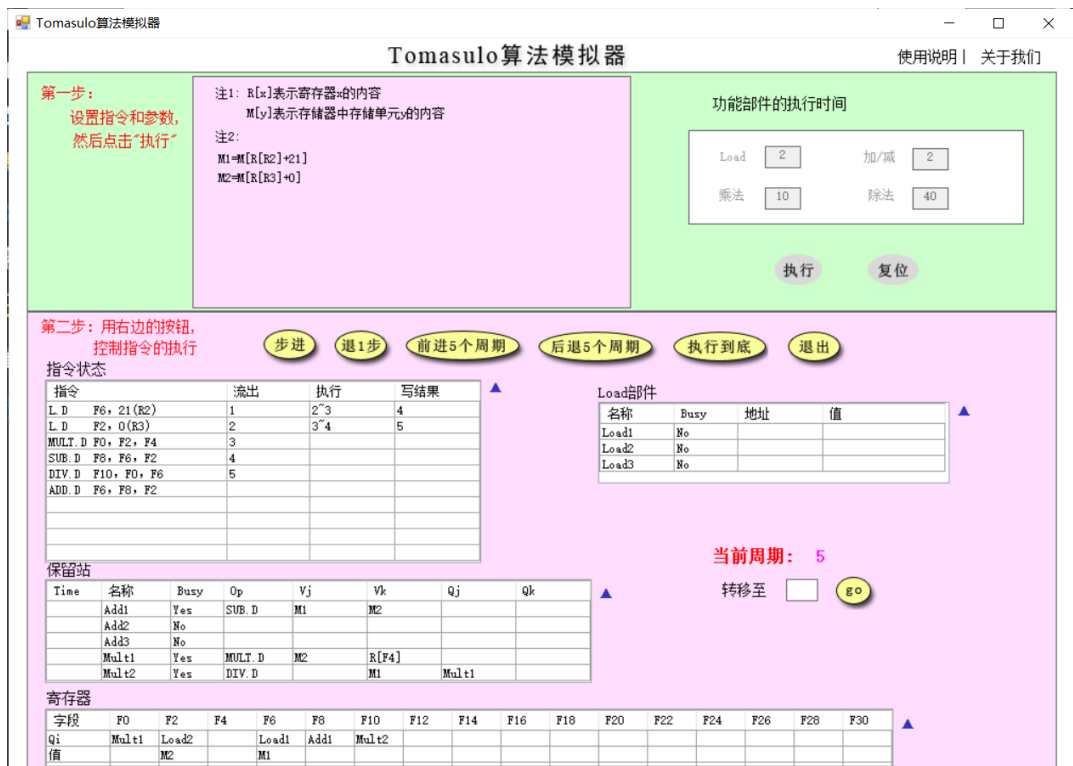
- 请截图 (MULT.D 刚开始执行时系统状态), 并说明该周期相比上一周期整个系统发生了哪些改动 (指令状态、保留站、寄存器和 Load 部件)

- MULT.D 刚开始指令是的系统状态 (第 6 周期)



- 该周期相比上一周期整个系统发生的改动

上一周期状态如下



- 指令状态: 流出第 6 条指令, 第 3、4 条指令进入执行阶段
- 保留站
  - 流出的第 6 条指令(ADD.D) 占用 Add2 的保留站
  - 进入执行状态的第 3 条指令(MULT.D)和第 4 条指令(SUB.D)开始执行完成倒计时
- 寄存器: 流出的第 6 条指令(ADD.D)将会写回 F6 寄存器, 即 F6 寄存器等待部件 Add2 的结果
- Load 部件: 无变化

### 3. 简要说明是什么相关导致MUL.D流出后没有立即执行

- 关于操作数 F2 的 RAW 相关。MUL.D 需要等待第 2 条 L.D 指令写结果

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

**Tomasulo算法模拟器** 使用说明 | 关于我们

**第一步：**  
设置指令和参数，  
然后点击“执行”

注1: R[x]表示寄存器x的内容  
M[y]表示存储器中存储单元y的内容

注2:  
M1=M[R[R2]\*21]  
M2=M[R[R3]\*0]  
M3=M1-M2  
M4=M3+M2

功能部件的执行时间

Load	2	加/减	2
乘法	10	除法	40

执行 复位

**第二步：用右边的按钮，  
控制指令的执行**

步进 退1步 前进5个周期 后退5个周期 执行到底 退出

指令状态

指令	流出	执行	写结果
L D F6, 21(R2)	1	2~3	4
L D F2, 0(R3)	2	3~4	5
MULT.D F0, F2, F4	3	6~15	
SUB.D F8, F6, F2	4	6~7	8
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9~10	11

Load部件

名称	Busy	地址	值
Load1	No		
Load2	No		
Load3	No		

当前周期: 15  
转移至  go

保留站

Time	名称	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULT.D	M2		R[F4]	
	Mult2	Yes	DIV.D		M1	Mult1	

寄存器

字段	F0	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30
Qi	Mult1	Load2		Add2	Add1	Mult2										
值	M2			M4	M3											

○ 第 16 周期

**Tomasulo算法模拟器** 使用说明 | 关于我们

**第一步：**  
设置指令和参数，  
然后点击“执行”

注1: R[x]表示寄存器x的内容  
M[y]表示存储器中存储单元y的内容

注2:  
M1=M[R[R2]\*21]  
M2=M[R[R3]\*0]  
M3=M1-M2  
M4=M3+M2  
M5=M2\*R[F4]

功能部件的执行时间

Load	2	加/减	2
乘法	10	除法	40

执行 复位

**第二步：用右边的按钮，  
控制指令的执行**

步进 退1步 前进5个周期 后退5个周期 执行到底 退出

指令状态

指令	流出	执行	写结果
L D F6, 21(R2)	1	2~3	4
L D F2, 0(R3)	2	3~4	5
MULT.D F0, F2, F4	3	6~15	16
SUB.D F8, F6, F2	4	6~7	8
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9~10	11

Load部件

名称	Busy	地址	值
Load1	No		
Load2	No		
Load3	No		

当前周期: 16  
转移至  go

保留站

Time	名称	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	Yes	DIV.D	M5	M1		

寄存器

字段	F0	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30
Qi	Mult1	Load2		Add2	Add1	Mult2										
值	M5	M2		M4	M3											

○ 变化

- 第 15 周期 MULT.D 指令执行结束
- 第 16 周期 MULT.D 指令开始写入结果，保留站中的 Mult1 部件被释放，其 Busy 被置为 No，将结果写回寄存器 F0。保留站中的 Mult2 部件（对应执行 DIV.D 指令）从寄存器 F0 中读到第一个操作数 M5，DIV 指令操作数已经准备完成，即将进入执行状态。

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）

第 57 周期

Tomasulo算法模拟器

使用说明 | 关于我们

第一步：

设置指令和参数，  
然后点击“执行”

注1: R[x]表示寄存器x的内容  
M[y]表示存储器中存储单元y的内容

注2:  
 $M1 = M[R[R2] * 21]$   
 $M2 = M[R[R3] * 0]$   
 $M3 = M1 - M2$   
 $M4 = M3 + M2$   
 $M5 = M2 * R[F4]$   
 $M6 = M5 / M1$

功能部件的执行时间

Load 2

加/减 2

乘法 10

除法 40

执行

复位

第二步：用右边的按钮，  
控制指令的执行

步进

退1步

前进5个周期

后退5个周期

执行到底

退出

指令状态

指令	流出	执行	写结果
L.D F6, 21(R2)	1	2~3	4
L.D F2, 0(R3)	2	3~4	5
MULT.D F0, F2, F4	3	6~15	16
SUB.D F8, F6, F2	4	6~7	8
DIV.D F10, F0, F6	5	17~56	57
ADD.D F6, F8, F2	6	9~10	11

Load部件

名称	Busy	地址	值
Load1	No		
Load2	No		
Load3	No		

保留站

Time	名称	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

寄存器

字段	F0	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30
Qi	Mult1	Load2		Add2	Add1	Mult2										
值	M5	M2		M4	M3	M6										

当前周期: 57

转移至

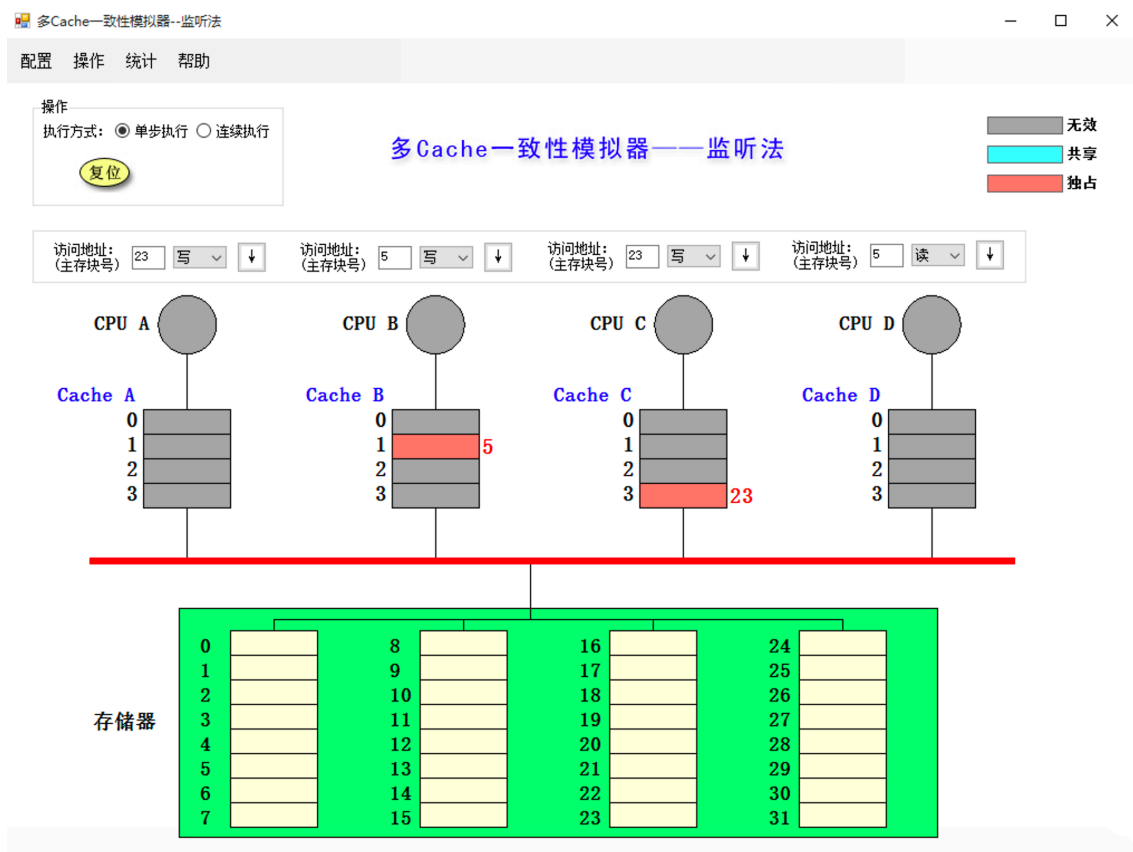
go

## 多 cache 一致性算法-监听法

- 利用模拟器进行下述操作，并填写下表

所进行的访问	是否替换	是否写回	监听协议进行的操作与块状态改变
CPUA 读第5块	是, 替换 Cach A 的块1	否	CacheA发送Read Miss, 存储器传输第5块到CacheA CacheA的块1状态变为共享
CPUB 读第5块	是, 替换 Cach B 的块1	否	CacheB发送Read Miss, 存储器传输第5块到CacheB CacheB的块1状态变为共享
CPUC 读第5块	是, 替换 CacheC 的块1	否	CacheC发送Read Miss, 存储器传输第5块到CacheC CacheC的块1状态变为共享
CPUB 写第5块	否	否	CacheB发送Invalidation CacheB第1块状态为独占 CacheA和CacheC第1块状态变为无效
CPUD 读第5块	是, 替换 CacheD 的块1	是	CacheD发送Read Miss CacheB写回第5块 存储器传输第5块到CacheD CacheB的块1状态变为共享 CacheD的块1状态变为共享
CPUB 写第21块	是, 替换 CacheB 的块1	否	Cache B发送Write Miss 存储器传输第21块到CacheB Cache B块1状态变为独占
CPUA 写第23块	是, 替换 CacheA 的块3	否	CacheA发送Write Miss 存储器传输第23块到CacheA CacheA的块3状态变为独占
CPUC 写第23块	是, 替换 CacheC 的块3	是	CacheC发送Write Miss CacheA写回第23块 存储器传输第23块到CacheC CacheA的块3状态变成无效 CacheC的块3状态变成独占
CPUB 读第29块	是, 替换 CacheB 的块1	是	CacheB写回第21块, CacheB发送Read Miss 存储器传输第29块到CacheB CacheB的块1状态变为共享
CPUB 写第5块	是, 替换 CacheB的块1	否	CacheB发送Write Miss 存储器传输第5块到Cache B CacheB的块1状态变为独占 CacheD的块1状态变为无效

- 请截图, 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

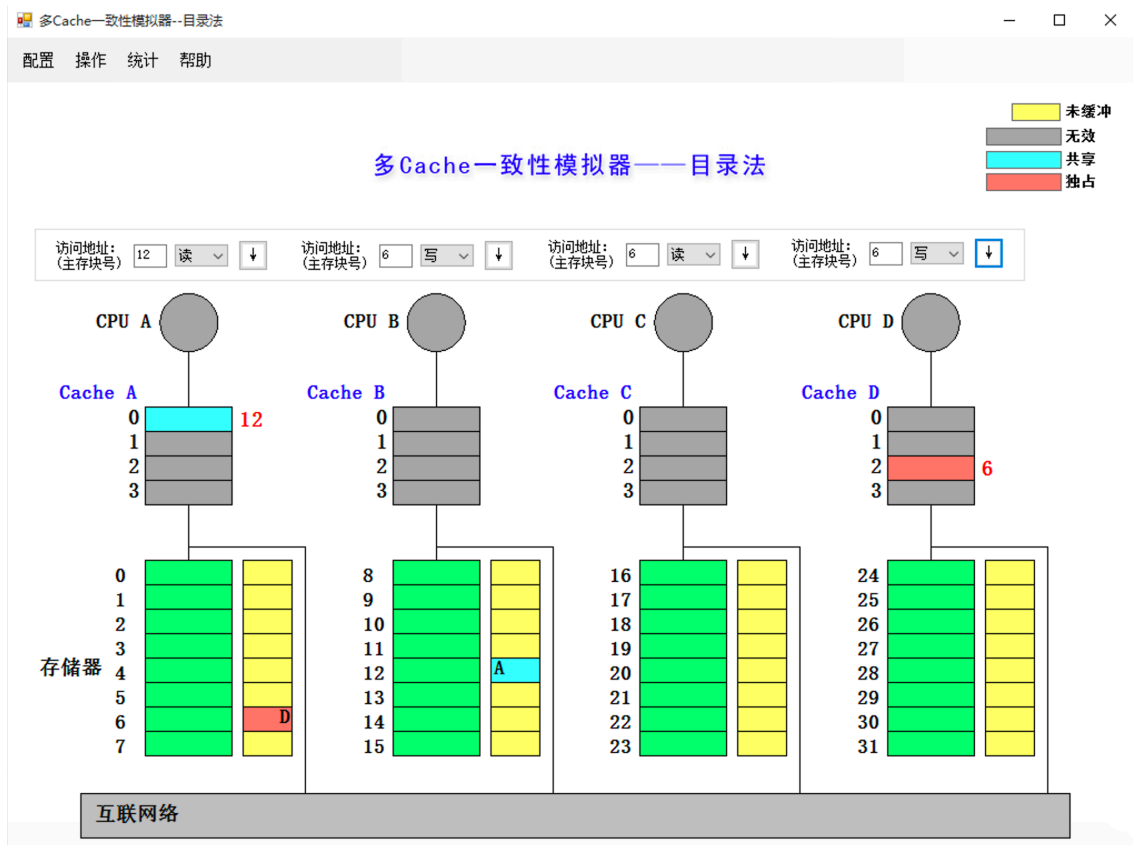
- 利用模拟器进行下述操作，并填写下表

所进行的访问	监听协议进行的操作与块状态的改变
CPUA 读第6块	CacheA发送Read Miss(A,6)到存储器 存储器传输第6块到CacheA CacheA的块2状态变为共享 存储器目录的块6状态变为共享 共享集合{A}
CPUB 读第6块	CacheB发送Read Miss(B,6)到存储器 存储器传输第6块到CacheB CacheB的块2状态变为共享 共享集合{A, B}
CPUD 读第6块	CacheD发送Read Miss(D,6)到存储器 存储器传输第6块到CacheD CacheD的块2状态变为共享 共享集合{A, B, D}
CPUB 写第6块	CacheB发送Write Hit(B,6)到存储器 存储器发送Invalidate(6)到Cache A、D CacheA和CacheD的块2状态变为无效 CacheB的块2状态变为独占 共享集合{B}
CPUC 读第6块	CacheC发送Read Miss(C,6)到存储器 存储器发送Fetch(6)到CacheB CacheB传输第6块到存储器 CacheB的块2状态变为共享 存储器传输第6块到CacheC CacheC的块2状态变为共享 共享集合{B, C}
CPUD写第20块	CacheD发送Write Miss(D,20)到存储器 存储器传输第20块到CacheD CacheD的块0状态变为独占 存储器目录的块20状态变为独占 共享集合{D}
CPUA写第20块	CacheA发送Write Miss(A,20)到存储器 存储器发送Fetch和Invalidate(20)到CacheD CacheD写回第20块 CacheD的块0状态变为无效 存储器传输第20块到CacheA CacheA的块0状态变为独占 共享集合{A}
CPUD写第6块	CacheD发送Write Miss(D,6)到存储器 存储器发送Invalidate(6)到CacheB、CacheC CacheB、CacheC的块2状态变为无效 存储器传输第6块到CacheD CacheD的块2状态变为独占 存储器目录的块6状态变为独占 共享集合{D}



所进行的访问	监听协议进行的操作与块状态的改变
CPUA 读第12块	CacheA发送Write Back(A,20)到存储器 CacheA的块0状态变为无效 CacheA发送Read Miss(A,12)到存储器 存储器传输第12块到CacheA CacheA的块0状态变为共享 存储器目录的块12状态变为共享 共享集合{A}

- 请截图，展示执行完以上操作后整个cache系统的状态。



## 综合问答

- 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：实现相对简单，但对总线要求较高，可扩展性较差
  - 目录法：减少了对总线的使用（不需要通过总线进行广播），但需要专门的目录空间，空间代价较大，且实现较复杂
- 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）
  - 相关
    - Tomasulo和Score Board都解决了结构相关和数据相关（包括RAW、WAW、RAR）：对于结构相关都是用有结构相关不发射的方法，对RAW都使用动态调度的方法，检测到没有冲突再读寄存器、开始执行
    - Tomasulo使用寄存器重命名的方法消除WAR和WAW相关，而Score Board是检测WAR和WAW相关，并用插入停顿来解决这两种相关
  - Tomasulo是分布式方法，Score Board是集中式方法
- 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：通过保留站，当所需功能部件被占用时（即有结构冲突时）不发射

- WAR,WAW：在指令发射时，通过保留站中重命名寄存器来解决
- RAW：在保留站中记录每个功能部件所需的操作数，如果操作数还未得到，则推迟执行相应的指令。寄存器均就绪后，读取寄存器，开始执行

## 实验总结

---

本次实验不用写代码，主要是对算法的理解和应用。通过助教提供的模拟器，对Tomasulo、多 cache 一致性算法都有了更深刻的理解，收获颇丰。