Lab5 实验报告

PB18051098 徐碧涵

实验目标

- 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

实验环境

Tomasulo 模拟器，cache 一致性模拟器

实验过程

一、Tomasulo 算法模拟器

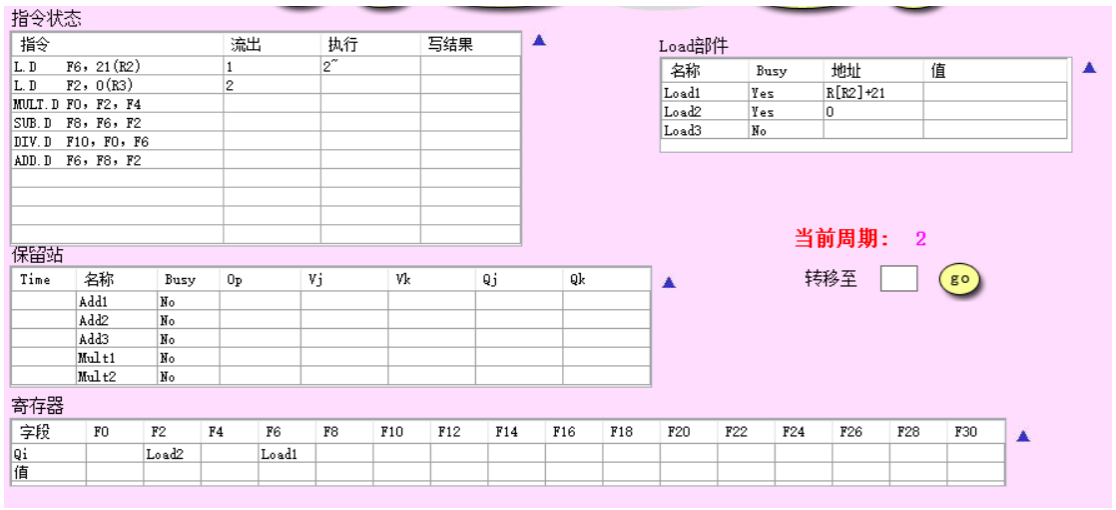
使用模拟器进行以下指令流的执行并对模拟器截图、回答问题

```
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
```

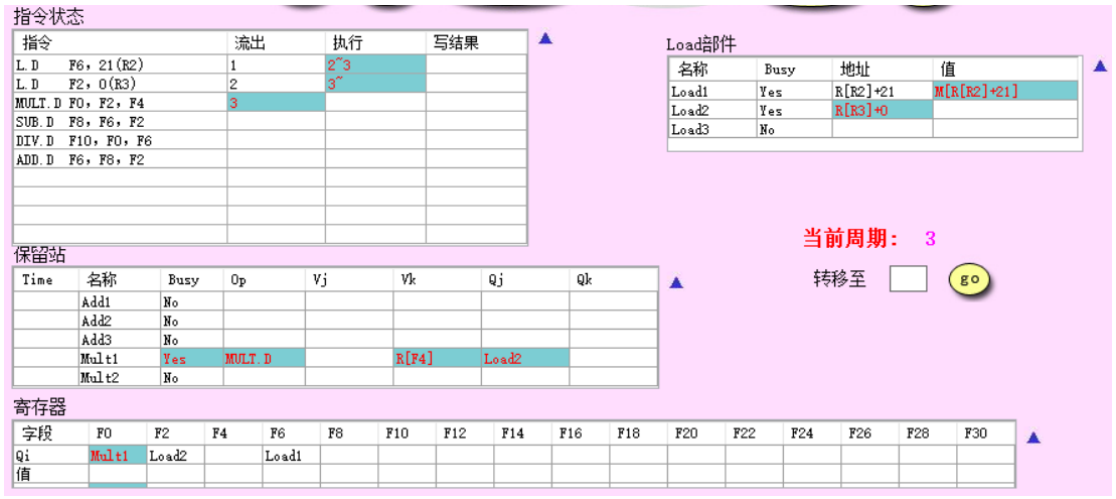
假设浮点功能部件的延迟时间：加减法 2 个周期，乘法 10 个周期，load/store2 个周期，除法 40 个周期。

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 截图：



周期 3 截图：



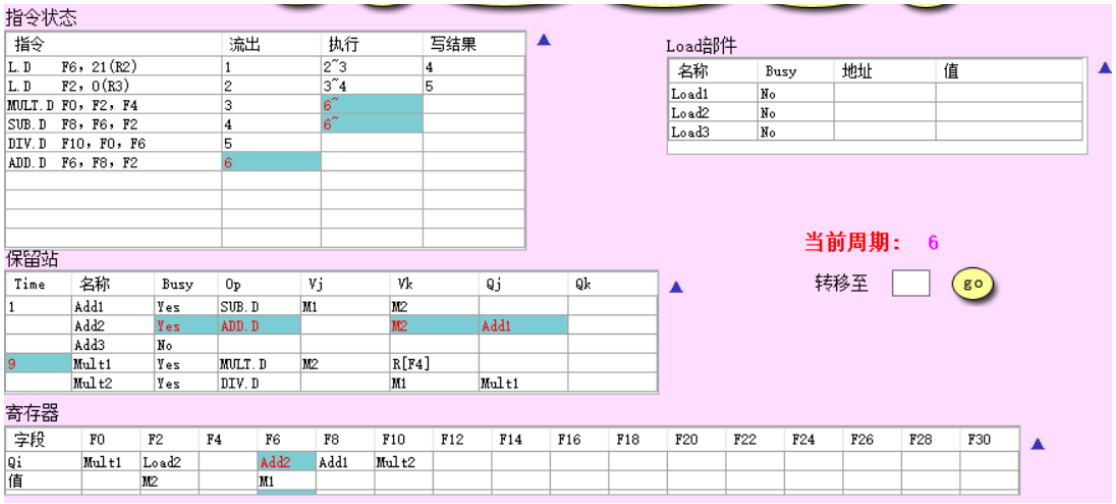
Load 部件的改动：

Load1 更新了访存取出的值

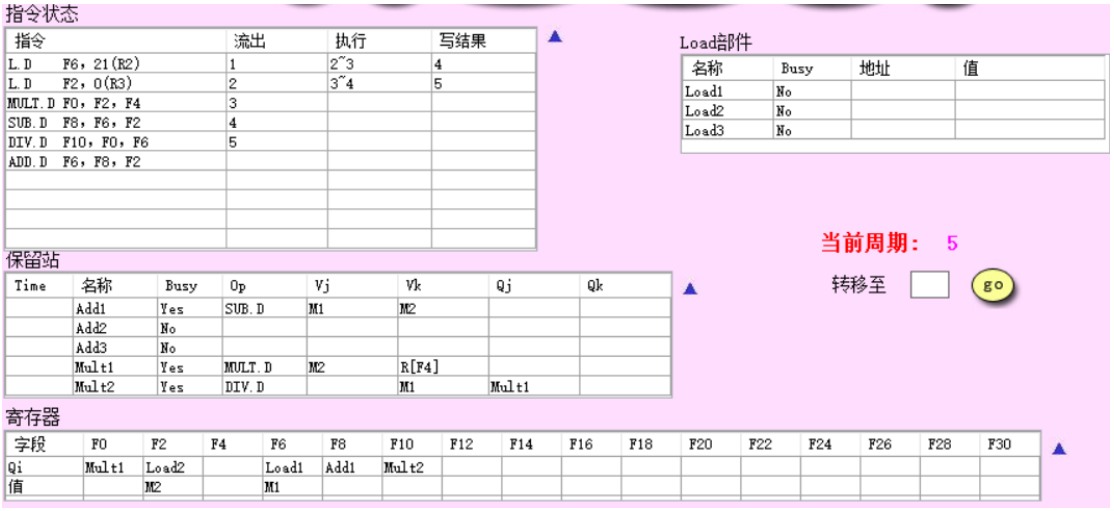
Load2 更新了取值的地址

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 刚开始执行时系统状态如下：



上一周期系统状态如下：



可见该周期相比于上一周期系统发生的变化如下;

指令状态：

指令 L.D F2, 0(R3) 写结果结束

指令 MULT.D F0, F2, F4 开始执行

指令 SUB.D F8, F6, F2 开始执行

指令 ADD.D F6, F8, F2 流出

保留站：

Mult1 部件的 Time 变为 9

Add2 部件 Busy 变成 Yes，Op 为 ADD.D，Vk 为 M2，Qj 为 Add1

寄存器：

F6 的 Qi 从 Load1 变成 Add2

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

L.D F2, 0（R3）与 MUL.D F0, F2, F4 两条指令之间存在 RAW 相关导致在该 load 指令未执行完毕的情况下 MUL.D 指令无法执行

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

15 周期系统状态如下：

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

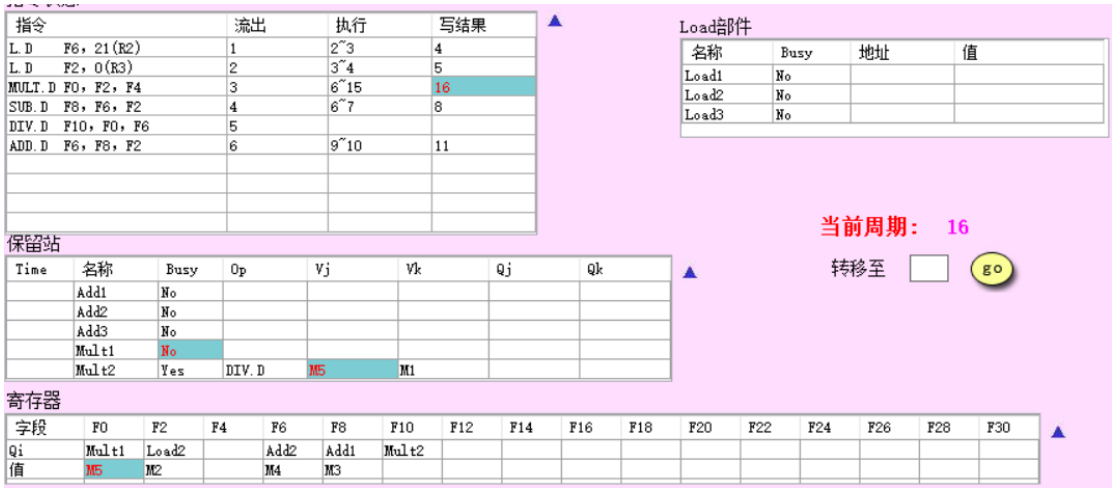
保留站

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											

16 周期系统状态如下：



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											

系统发生变化如下：

指令状态：

MULT.D 指令执行阶段完毕进入写结果阶段

保留站：

Mult1 部件 Busy 由 yes 变为 no

Mult2 部件 Vj 变为 M5

寄存器：

F0 寄存器值变为 M5

Load 部件：

无变化

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）

可见所有指令刚刚执行完毕是第 57 周期。

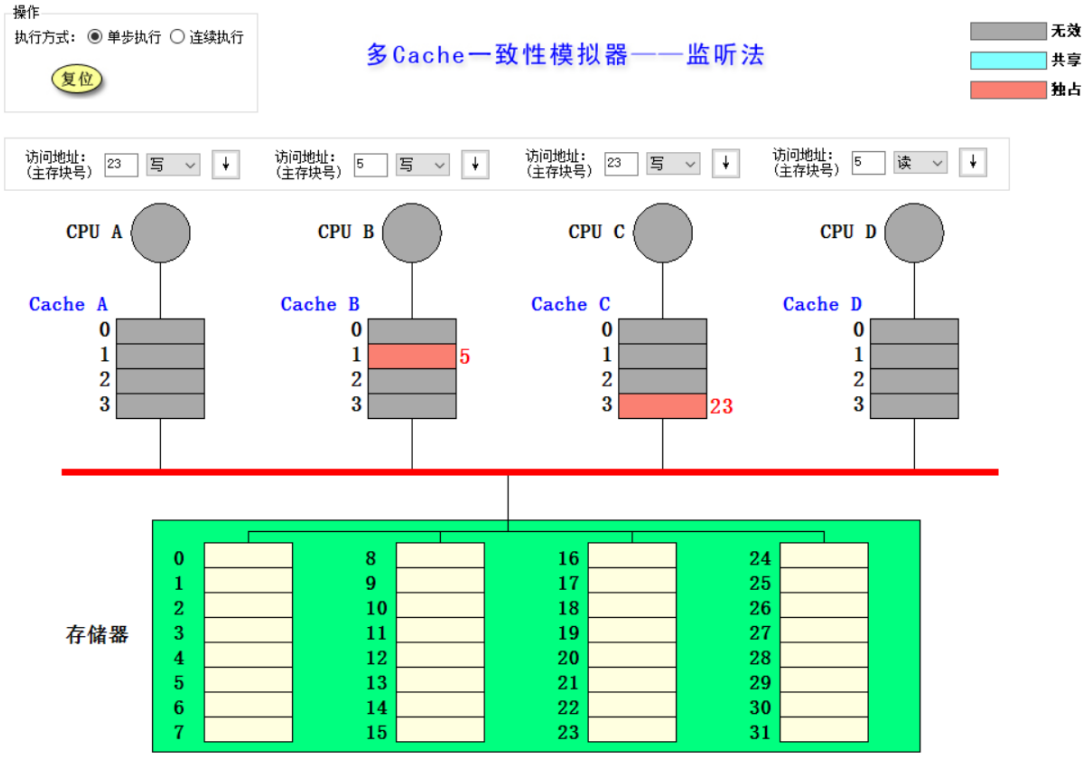
二、多 cache 一致性算法-监听法

1.利用模拟器进行下述操作，并填写下表

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	否	否	存储器第 5 块放入 cache A 块地址 1 中，将该块设为共享态，并发送给 CPU A
CPU B 读第 5 块	否	否	存储器第 5 块放入 cache B 块地址 1 中，将该块设为共享态，并发送给 CPU B
CPU C 读第 5 块	否	否	存储器第 5 块放入 cache C 块地址 1 中，将该块设为共享态，并发送给 CPU C
CPU B 写第 5 块	否	否	写命中 Cache B，将 cache A 与 cache C 中存储的第 5 块作废，并将 cache B 中该块的状态设为独占，CPU B 写 cache
CPU D 读第 5 块	否	是	CPU D 读不命中，cache B 将其存储的第 5 块副本写回至 memory，并将块状态由独占变为共享态，之后第五块放入 cache D 块地址 1 中，将其设为共享态并发送给 CPU D
CPU B 写第 21 块	是	否	CPU B 写不命中，将存储器第 21 块替换 cache B 中存储的第 5 块，块状态设为独占，CPU B 写 cache
CPU A 写第 23 块	否	否	写不命中，将存储器第 23 块放入 cache A 块地址 3 中，将该块设为独占态，CPU A 写 cache
CPU C 写第 23 块	否	是	CPU C 写不命中，cache A 将其存储的第 23 块副本写回至 memory，之后第 23 块放入 cache C 块地址 3

			中，将其设为独占态，CPU C 写 cache
CPU B 读第 29 块	是	是	CPU B 读不命中，将其存储在 cache B 地址 1 中的存储器第 21 块副本写回至 memory 中，将存储器第 29 块读入 cache B 块地址 1 中，设为共享态并发送给 CPU B
CPU B 写第 5 块	是	否	将存储器第 5 块替换 Cache B 块地址 1 中存储的存储器第 29 块，作废 cache D 中存储的第 5 块 ，并将 cache B 中的该块设为独占态， CPU B 写 cache

2. 请截图，展示执行完以上操作后整个 cache 系统的状态



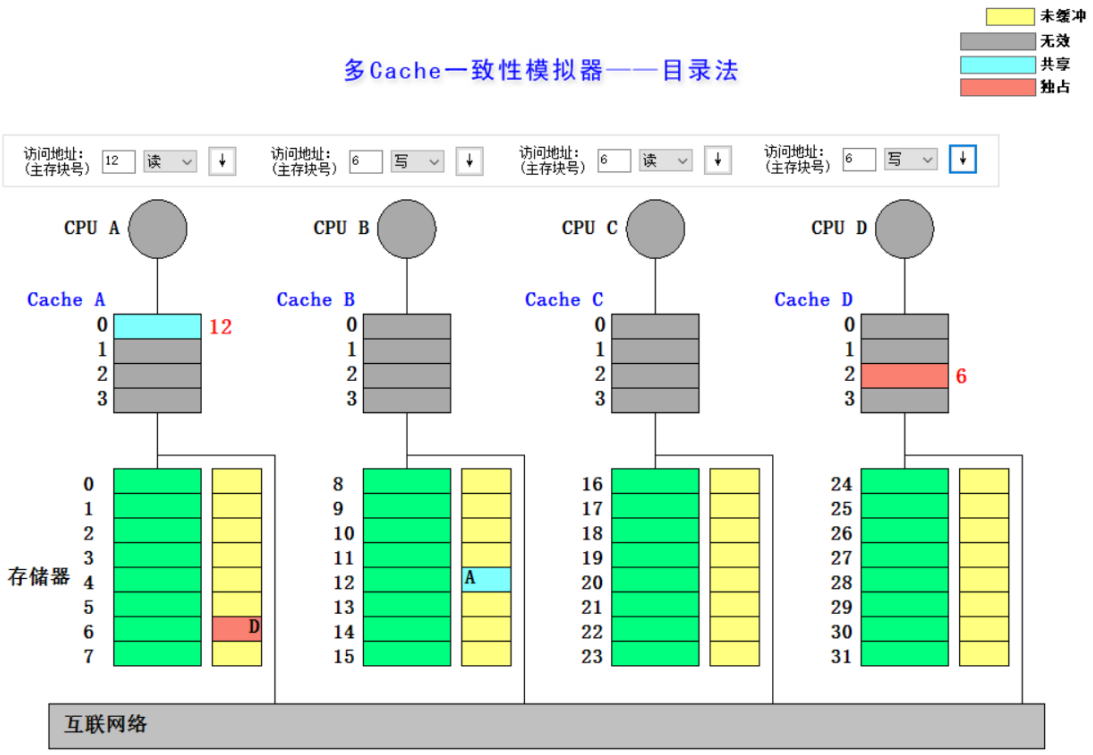
三、多 cache 一致性算法-目录法

1.利用模拟器进行下述操作，并填写下表

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 读不命中，本地向宿主结点发送读不命中(A, 6)消息，宿主把数据块第六块传给本地节点，放入 cache A 块地址 2 中，设为共享态，共享集合设为 {A}，块被送到 CPU A
CPU B 读第 6 块	CPU B 读不命中，本地向宿主结点发送读不命中(B, 6)消息，宿主把数据块第六块传给本地节点，放入 cache B 块地址 2 中，设为共享态，共享集合设为 {A，B}，块被送到 CPU B
CPU D 读第 6 块	CPU D 读不命中，本地向宿主结点发送读不命中(D, 6)消息，宿主把数

	据块第六块传给本地节点，放入 cache D 块地址 2 中，设为共享态，共享集合设为 {A，B，D}，块被送到 CPU D
CPU B 写第 6 块	CPU B 写命中，本地向宿主结点发送写命中（B，6）消息，宿主向远程节点 A 发送作废（6）消息，宿主向远程节点 D 发送作废（6）消息，共享集合为{B}，将其设为独占态，CPU B 写 cache
CPU C 读第 6 块	CPU C 读不命中，本地向宿主结点发送读不命中(C, 6)消息，宿主给远程节点 B 发取数据块（6）的消息，远程节点 B 把数据块发送给宿主节点，宿主节点把数据块发送给本地节点 C，共享集合为{B，C}，为共享态，块送到 CPU C
CPU D 写第 20 块	CPU D 写不命中，向宿主节点发写不命中（B，20）消息，宿主把数据块发送给本地节点，存放在 cache D 地址 0 处，共享集合为{D}，设为独占态，CPU D 写 cache
CPU A 写第 20 块	CPU A 写不命中，本地结点向宿主节点发送写不命中（A，20）消息宿主给远程结点 D 发送取且作废（20）的消息，远程节点 D 把数据块送给宿主节点并把 cache 中的该块作废，宿主把数据块送给本地结点 A，存放在 cache A 地址 0 处，共享集合为{A}设为独占态，CPU A 写 cache
CPU D 写第 6 块	CPU D 写不命中，向宿主节点发送写不命中（D，6）消息，宿主向远程节点 B 发送作废（6）消息，向远程节点 C 发送作废（6）消息，宿主把数据块发送给本地节点，存放在地址 2 处，第六块共享集合为{D}设为独占态，CPU D 写 cache
CPU A 读第 12 块	CPU A 写不命中，向被替换块的宿主节点发送写回并修改共享集（A，20）消息，块 20 共享集为{}，本地向宿主节点发送都不命中（A，12）消息，块 12 宿主节点把数据块发送给本地节点，存放在地址 0 处，块 12 共享集为{A}，块送到 CPU A

2.请截图，展示执行完以上操作后整个 cache 系统的状态



四、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）

监听法在 CPU 数较少时，使用总线通讯快，开销小，效果好，但是当 CPU 数增多时，总线压力增大，冲突增多使得性能下降。

目录法使用集中目录来记录块的状态，不会受到总线的限制，但是随着核数增加时目录的开销随之变大且通讯代价更高。

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）

（1）两者都解决了 RAW，WAR，WAW 三种相关，两种算法都是通过动态调度的方式来解决 RAW 冲突，在解决 WAR 和 WAW 时，Tomasulo 通过重命名技术，Score Board 使用等待的方式直到相关解除（2）Tomasulo 是分布式，Score Board 是集中式

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：只有在 RS 空闲时才控制发射指令

RAW：跟踪每个源操作数，仅当所有源操作数就绪时才发射指令，通过推迟指令执行避免 RAW 相关。

WAR，WAW：使用寄存器重命名技术解决 WAR 和 WAW