

**《计算机体系结构》课程**

**实验报告**

|  |  |
| --- | --- |
| **实验名称：** | **实验3-2 Tomasulo算法实验** |
| **姓 名：** | **江家玮** |
| **学 号：** | **22281188** |
| **班 级：** | **计科2204班** |
| **日 期：** | **2024.12.05** |

**目录**

**[1 实验目的 1](#_Toc17142)**

**[2 实验内容 1](#_Toc17493)**

**[3 实验流程 1](#_Toc19925)**

[3.1 模拟器使用说明 1](#_Toc9455)

[3.2 默认指令执行过程分析 3](#_Toc12161)

[3.3 Tomasulo算法处理RAW相关说明 7](#_Toc24604)

[3.4 Tomasulo算法实现寄存器重命名说明 8](#_Toc7556)

[3.5 Tomasulo算法避免数据相关说明 9](#_Toc30480)

**1 实验目的**

本次实验的主要目的为熟悉Tomasulo算法的实现流程，掌握指令流水化过程中乱序执行和寄存器重命名原理。

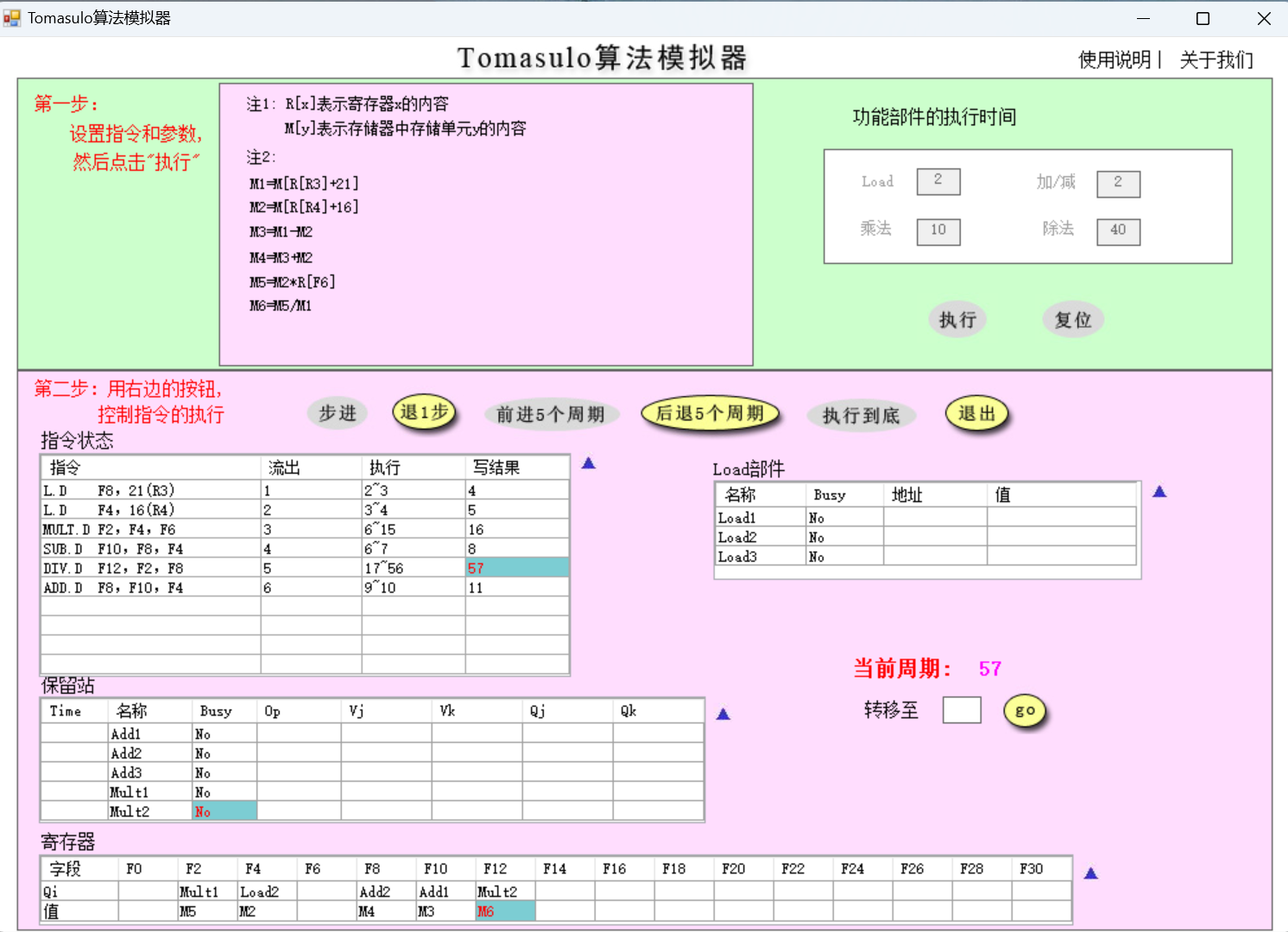
**2 实验内容**

1. 仔细阅读给出的Tomasulo模拟器的使用说明，熟悉该模拟器的使用。
2. 分析该模拟器中默认程序的指令乱序执行和寄存器重命名过程。观察并分析在程序执行过程中，指令状态、保留站、load部件、寄存器的状态变化。（可以选取几个cycle来说明）
3. 结合相关指令举例说明Tomasulo算法是如何处理RAW相关的？
4. Tomasulo算法是如何实现寄存器重命名的？举例说明。
5. Tomasulo算法能避免哪些数据相关？结合相关指令举例说明Tomasulo算法是如何消除这些数据相关的？（请结合指令状态、保留站、load部件、寄存器的状态变化进行详细分析。可自行对指令进行修改。）

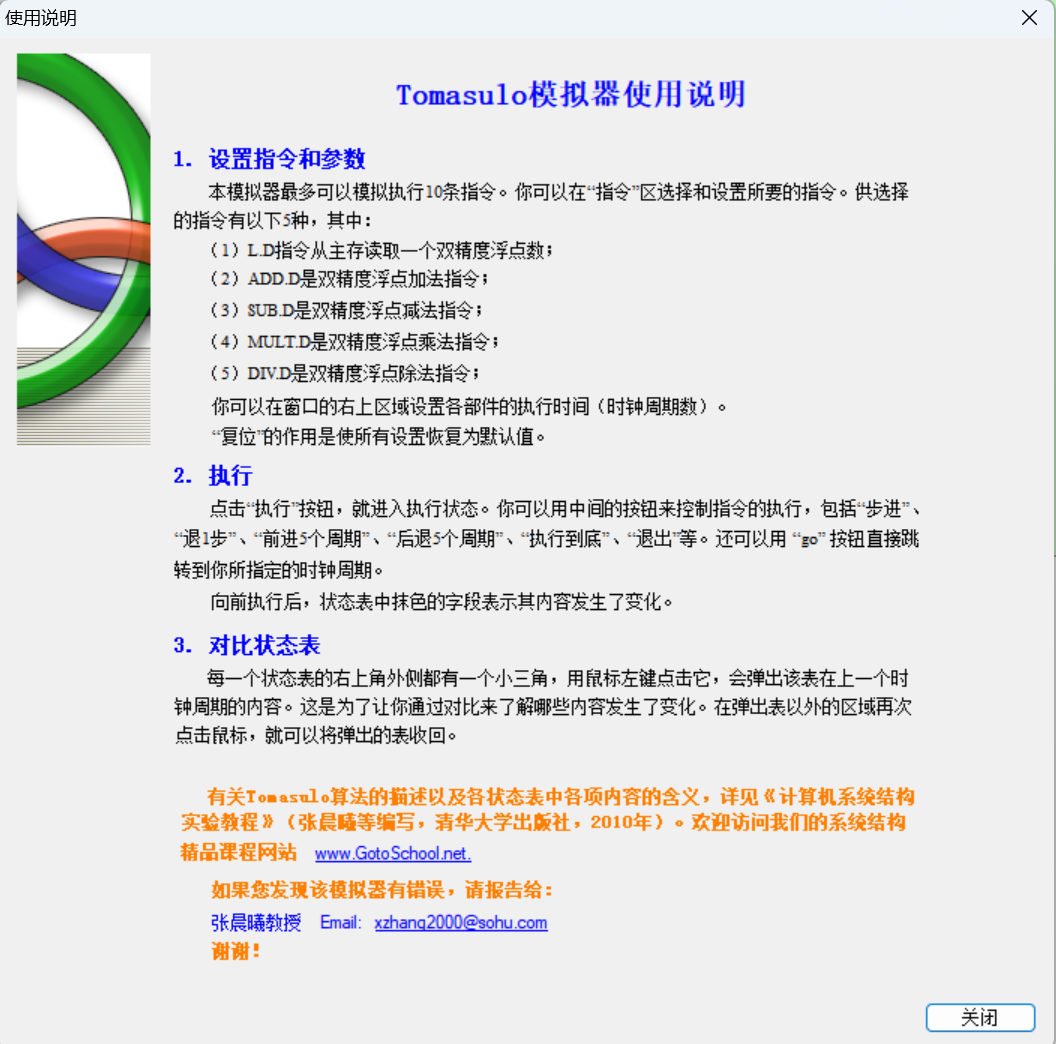
**3 实验流程**

**3.1 模拟器使用说明**

Tomasulo模拟器的使用如图3-1-1和3-1-2所示，下载完成后即可使用。

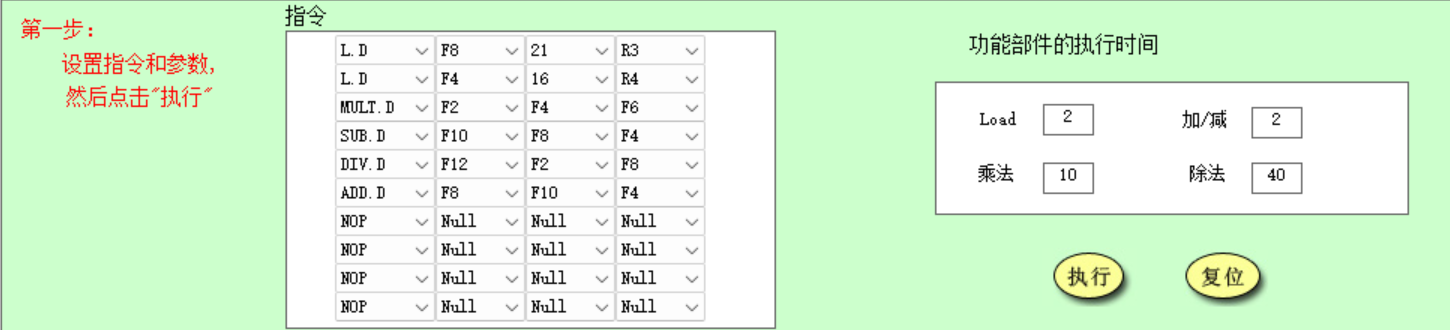


**图3-1-1 Tomasulo模拟器页面展示**

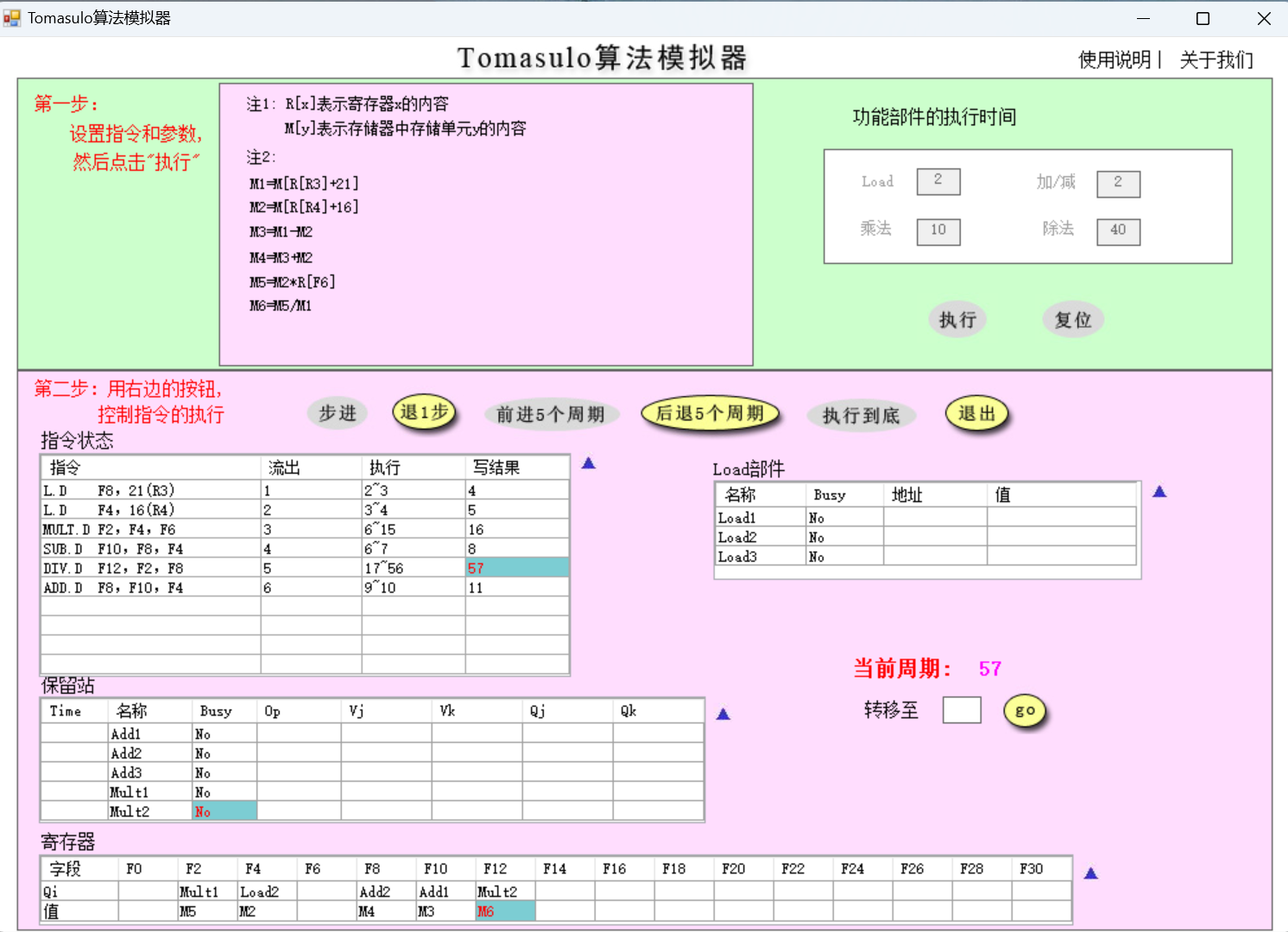


**图3-1-2 Tomasulo模拟器使用说明**

从以下图片展示页面中即可调整指令、寄存器以及参数：

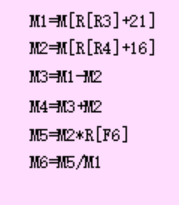
****

**图3-1-3 指令展示**

****

**图3-1-4 指令执行后页面展示**

由上图可知，整个tomasulo模拟器实现的是如下图所示的代码部分：

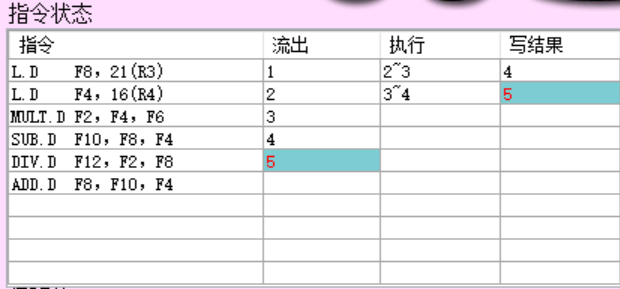


**图3-1-5 运行代码展示**

**3.2 默认指令执行过程分析**

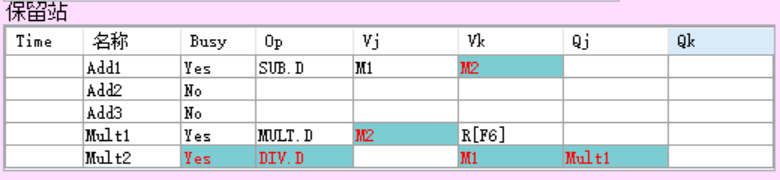
**以第5条指令为例**

指令状态：此时，第二条load指令刚刚完成执行，写回结果F4，因为同周期都是先读后写，乘法指令和减法指令要用到F4，存在数据相关，所以在这个周期依旧没有执行，此时除法指令流出，进入保留站。

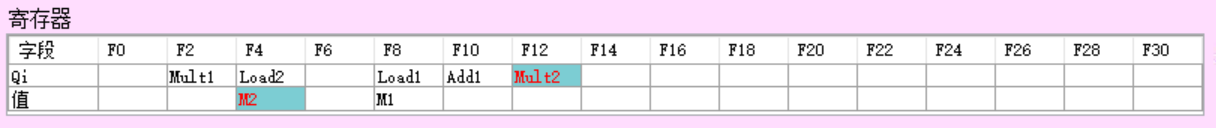


**图3-2-1 第五条指令指令状态展示**

保留站：此时保留站保存有乘法运算，减法运算和除法指令三条指令，在上一个周期（第四个周期）中因为源操作数中都有F4的操作字段是通过load指令获得，而load指令还在执行中，所以乘法和减法运算目前都不能执行，而在这个周期，我们可以看到指令都已经获得好了操作数，下一个周期（第六个周期）就可以执行了。对于除法指令，因为要用到的F2是乘法指令运算的结果，所以Qj出直接标记了对应的指令保留站编号mult1，存在数据相关，所以也不会执行。

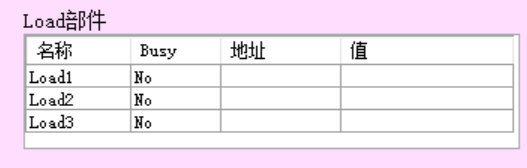


**图3-2-2 第五条指令保留站展示**

寄存器：此时前两条load指令已经完成了计算，所以F8，F4都可以直接标记数值为M1和M2，其他寄存器类似F2,F10,F12是其余指令的目标寄存器，因为还没有完成指令，所以只存有Qi标记，告知后续使用的指令需要等待的指令编号。

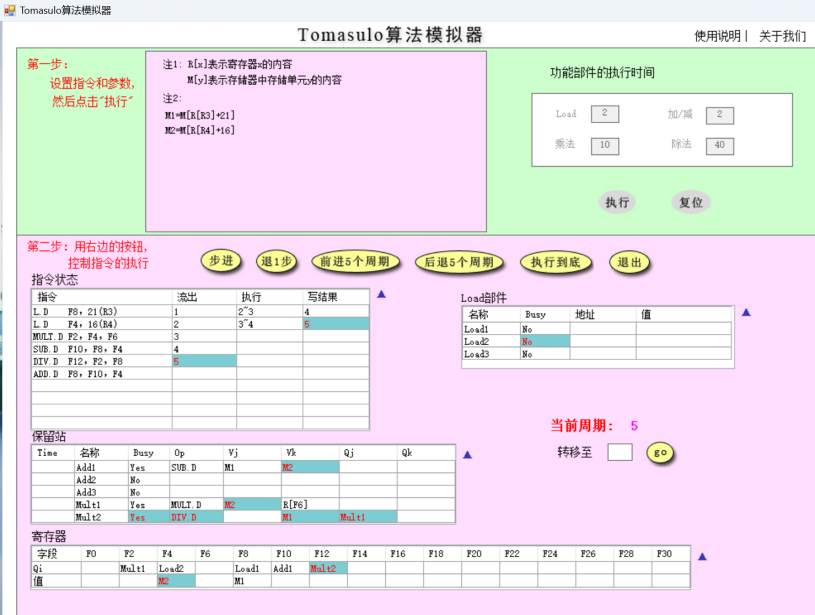
**图3-2-3 第五条指令寄存器展示**

Load部件：前两条load指令都已经完成，并且此时没有新的load指令流出，所以此时load部件中为空。



**图3-2-4 第五条指令Load部件展示**

寄存器重命名：在除法指令中，其中因为F2对应的乘法指令还没有完成，所以可以再保留站中看到对应的位置标记的是Mult2，也就是产生F2的指令编号，这里就实现了寄存器的重命名。



**图3-2-5 第五条指令展示**

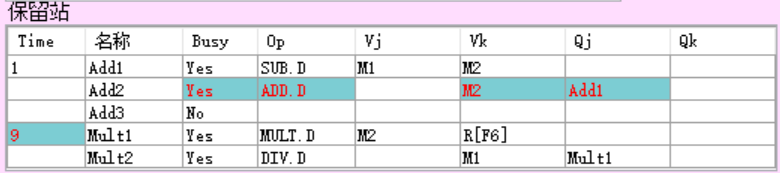
**以第6条指令为例**

指令状态：此时，第二条load指令刚刚完成执行，写回结果，第三条、第四条load指令正在执行，第五条除法指令因为数据相关，操作数F2还是load2的形式，所以没有开始执行，第四六指令流出，进入保留站。



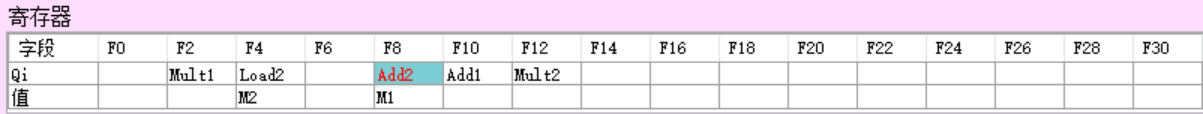
**图3-2-6 第六条指令指令状态展示**

保留站：此时保留站保存有加法运算、乘法运算、减法运算和除法指令四条指令，在第五个周期中因为源操作数中有F2的操作字段是通过Mult指令获得，而Mult指令还在执行中，所以除法指令运算目前不能执行，而在这个周期，我们可以看到指令都已经获得好了操作数，下一个周期（第七个周期）就可以执行了。



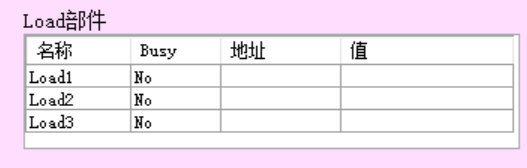
**图3-2-7 第六条指令保留站展示**

寄存器：此时前两条load指令已经完成了计算，所以F8，F4都可以直接标记数值为M1和M2，其他寄存器类似F2、F10、F12是其余指令的目标寄存器，因为还没有完成指令，所以只存有Qi的标记，告知后续使用的指令需要等待的指令编号。



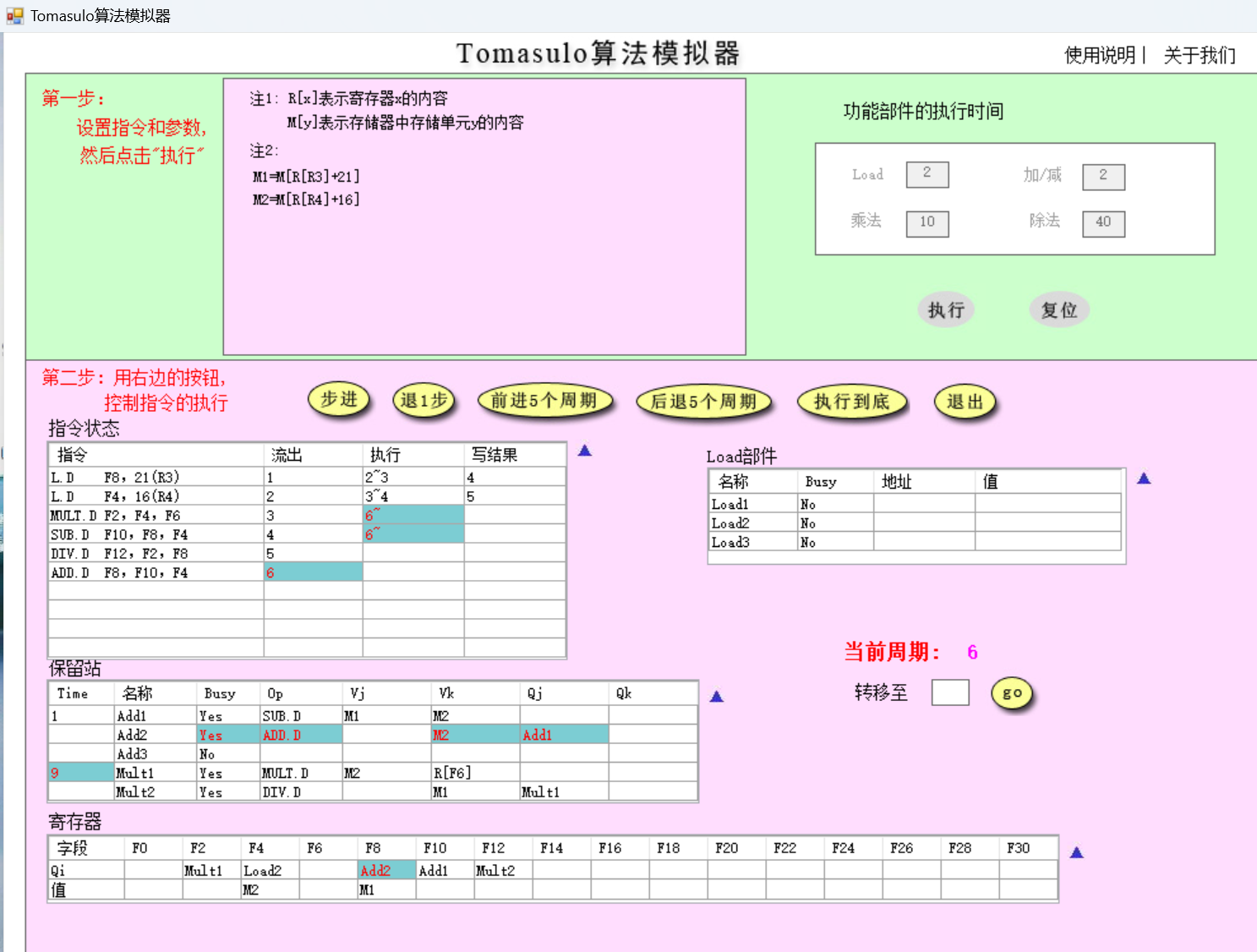
**图3-2-8 第六条指令寄存器展示**

Load部件：前两条load指令都已经完成，并且此时没有新的load指令流出，所以此时load部件中为空。



**图3-2-9 第六条指令Load部件展示**

寄存器重命名：在加法指令中，其中因为F10对应的减法指令还没有完成，所以可以再保留站中看到对应的位置标记的是Add1，也就是产生F10的指令编号，这里就实现了寄存器的重命名。

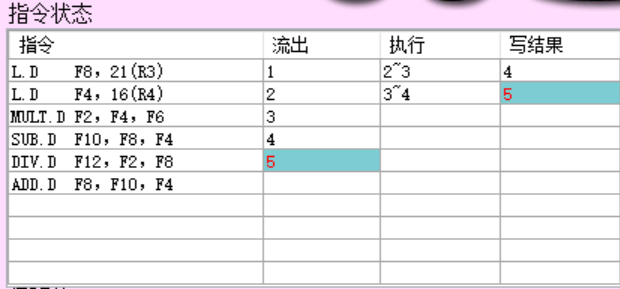


**图3-2-10 第六条指令展示**

**3.3 Tomasulo算法处理RAW相关说明**

Tomasulo算法对于RAW的处理思路就是等待相关指令完成源寄存器数据的计算之后，将数据传到公共数据总线CDB 上并告知所有功能单元，各自功能单元自行判断，需要单元的从CDB上读取数据。

结合指令进行说明，在上面说明的第五周期，此时已经流出了乘法指令，减法指令和除法指令，但是因为前两个指令要用的数据在这个周期才完成load的写结果，除法指令更是要用到乘法指令的结果，所以此时指令虽然流出并且在保留站中存储，但是并没有开始执行。



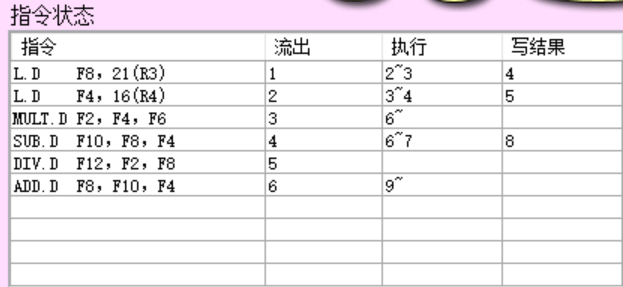
**图3-3-1 第五个周期指令状态展示**

直到第六个周期的时候，F4完成load指令，这时乘法和减法指令才开始执行，中间的等待就是tomasulo算法处理RAW相关的方式。



**图3-3-2 第六个周期指令状态展示**

同样的例子在第九个周期的时候也可以看到，因为加法指令的F10数据之前没有完成计算，所以指令只流出，不执行，直到第八个周期完成了减法指令的写结果后，第九个周期的加法指令才开始执行。



**图3-3-3 第九个周期指令状态展示**

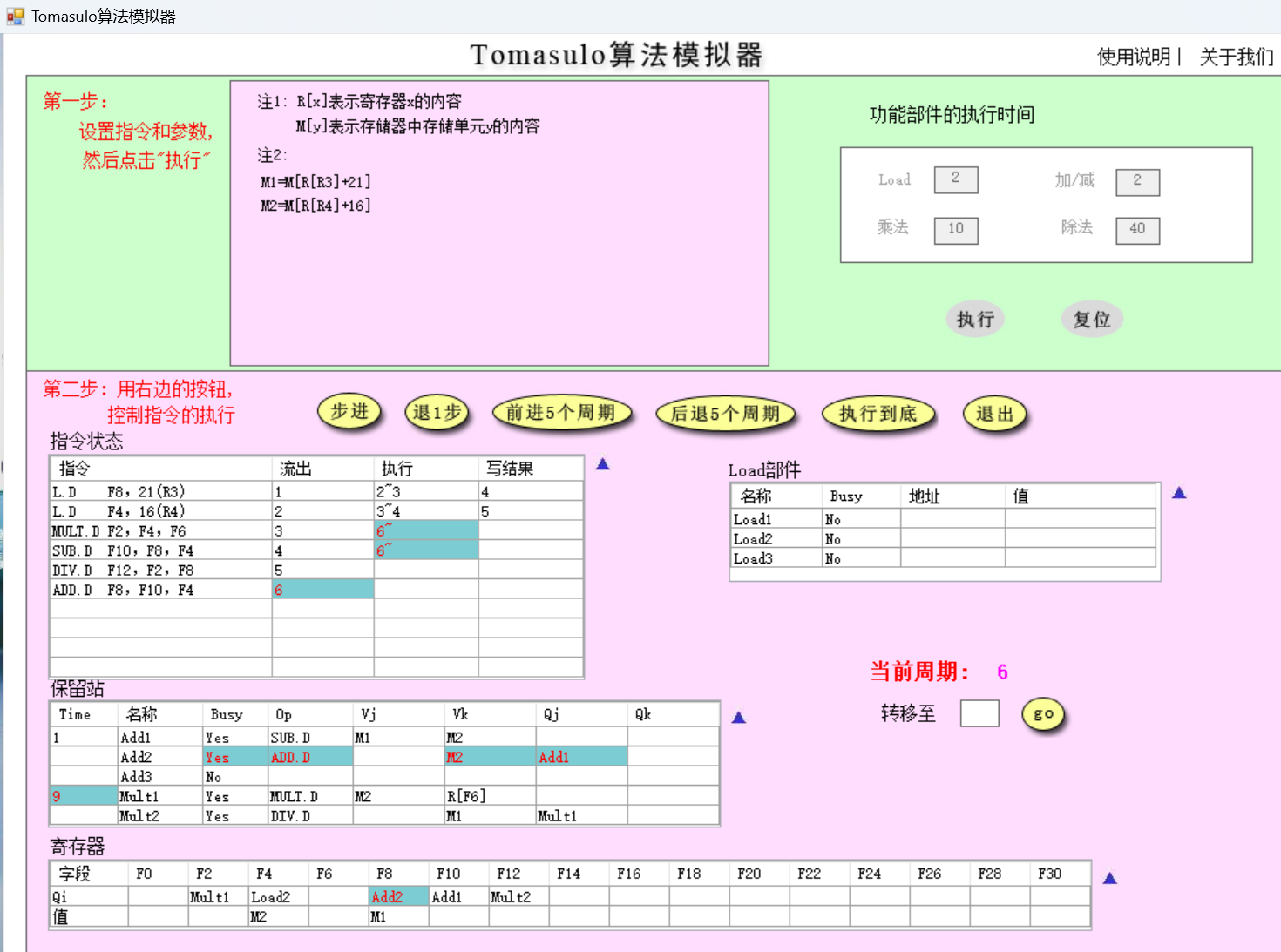
**3.4 Tomasulo算法实现寄存器重命名说明**

对寄存器进行重命名是Tomasulo算法的核心操作，因为采用乱序执行会导致名相关，也就是因为访问寄存器名一直所引起的相关，但是这是可以通过修改寄存器名避免的，而Tomasulo算法就是执行了这一点。

一个新的指令流出，进入保留站之后，它首先就要检查操作数，在寄存器中查看对应的操作数有没有，如果有的话自然万事大吉直接读取就可以，如果是基于其他运算指令得到的，并且还没有完成，寄存器中此时就没有数值，此时对这个操作数进行标记，标记这个计算指令对应的保留站编号，对于源操作数，也需要改写成这个指令的保留站编号，这时就完成了寄存器的重命名。

举例说明：在第六个周期中，加法指令流出，进入保留站。此时需要用到的操作数为F10和F4，F4在load指令完成之后，在寄存器处已经有保存，可以直接使用，但是F10是正在执行的减法指令的目的寄存器，这时，将源操作数标记为这个指令的保留站编号Add1，就是完成了寄存器的重命名。

同时，这里对F8的目的寄存器也进行了修改，本来已经完成了load操作，但是因为成为了新的目的寄存器，所以需要重新设置Qi。



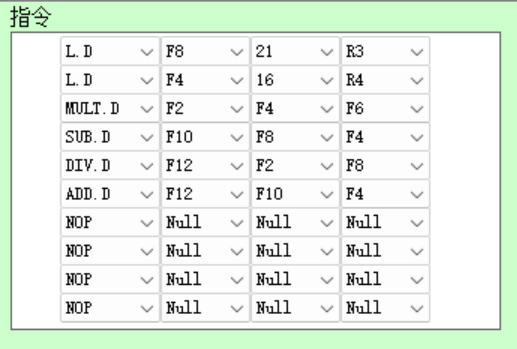
**图3-4-1 第六周期运行展示**

**3.5 Tomasulo算法避免数据相关说明**

tomasulo算法可以避免RAW相关，这部分在前面第三部分的时候已经解释过了。

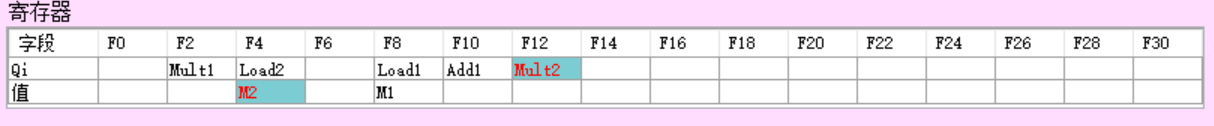
tomasulo算法也可以避免WAW相关，原理是前一个指令先进功能单元，紧接着对应的目标寄存器的标识就修改为了这个保留站的编号，接下来，第二个指令进入功能单元，目的寄存器的标识再次修改成第二个，这时即便是第二个指令先完成并写回，前一个指令再算好了也不会改变寄存器了，因为在CDB通知各个功能单元的时候，对应的寄存器只会响应第二个指令完成时传递发出的CDB信号。

举例说明：修改模拟器中的默认指令，将最后的加法指令的目的寄存器调整为F12，此时最后两条指令之间就产生了WAW相关，因为除法指令的执行时间会更长，如果不进行处理的话，就会出现除法指令执行——加法指令执行——加法指令完成写回——除法指令完成写回的情况。



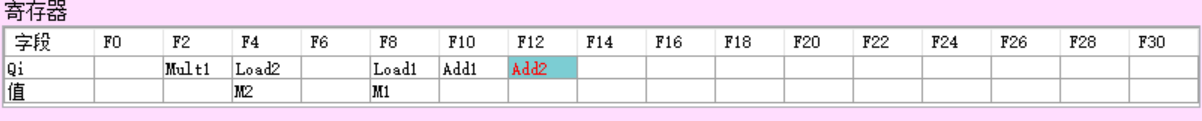
**图3-5-1 指令设置**

而通过使用tomasulo算法，我们可以看到在第五个周期的时候，除法指令流出，进入保留站，F12寄存器保留了除法指令的保留站编号：



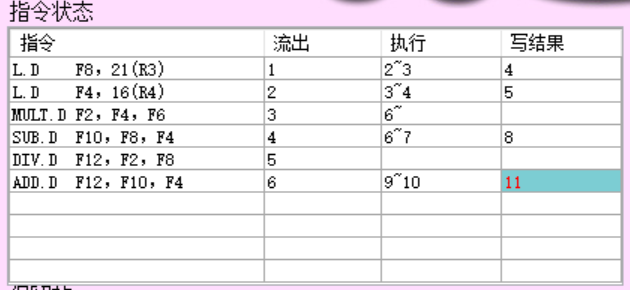
**图3-5-2 第五周期寄存器展示**

在第六个周期的时候，加法指令流出，此时F12的寄存器更改标识为加法指令对应的保留站编号：

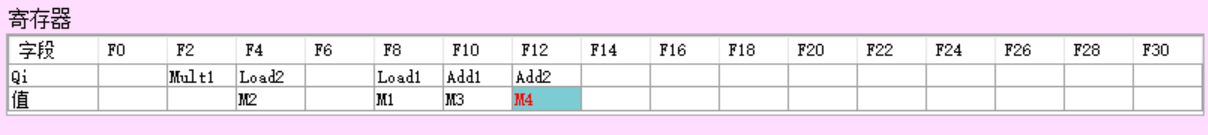


**图3-5-3 第六周期寄存器展示**

那么此时第一条指令什么时候完成就已经不重要了，F12对应的寄存器只会在加法指令完成之后，响应CDB信号读取数据，可以看到在加法指令完成的第十一个周期，寄存器F12中就获取了数据：



**图3-5-4 第十一周期指令状态展示**



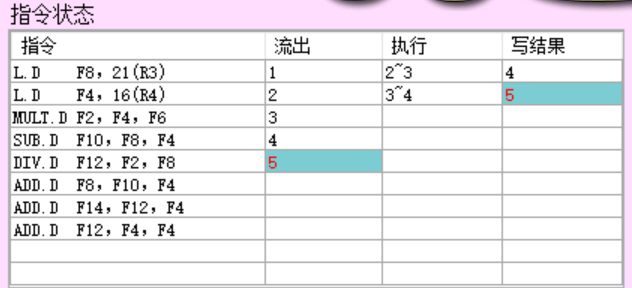
**图3-5-5 第十一周期寄存器状态展示**

而继续执行程序到最后除法指令完成，可以看到最后的除法运算完成之后，F12的寄存器数据也没有改变，说明tomasulo算法可以处理WAW相关。

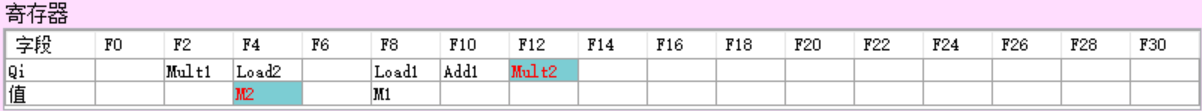
Tomasulo算法也可以避免WAR相关，因为前一个读指令先进保留站，有数据的话是直接从寄存器取，没有的话就是标记上对应的指令的保留站编号，这样一来，后续的写操作的目的寄存器和前一个读的已经不同名了，自然没有影响。

举例说明：添加了两个加法指令，其中第一个指令要读F12寄存器，第二个指令要写F12寄存器，如果没有处理WAR相关，指令乱序执行，就可能会出现因为加法指令完成更快，写回到F12的位置，第一个加法指令读到的是第二个指令的结果，但是实际指令的目的是让它读除法指令的结果，导致WAR相关。

而在tomasulo算法中，我们可以看到在第五个周期的时候，除法指令流出，目的寄存器F12此时标记的就是这个除法指令的保留站编号，在这里实现了寄存器的重命名：

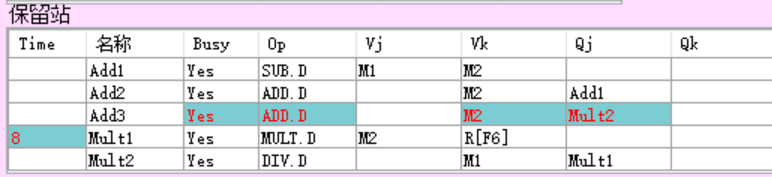


**图3-5-6 第五周期指令状态展示**



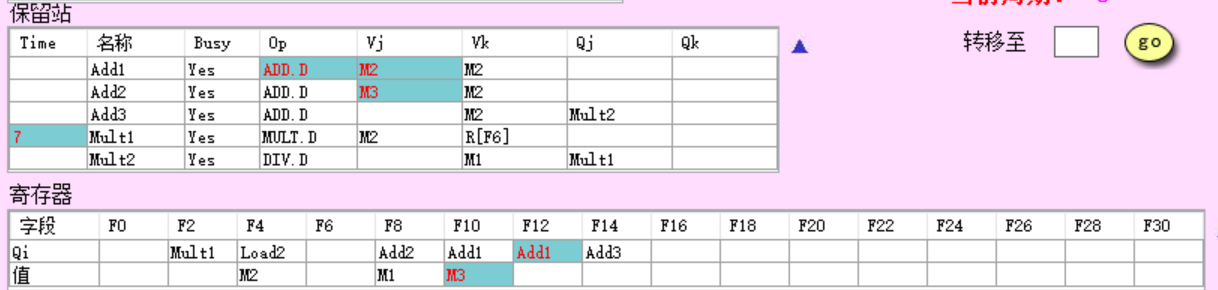
**图3-5-7 第五周期寄存器状态展示**

接下来第七个周期的时候，添加的第一个加法指令流出，此时检查操作数，F4在寄存器中已经有了，所以可以直接读取，F12此时还没有，所以读取的是对应指令的保留站编号并存在保留站中：



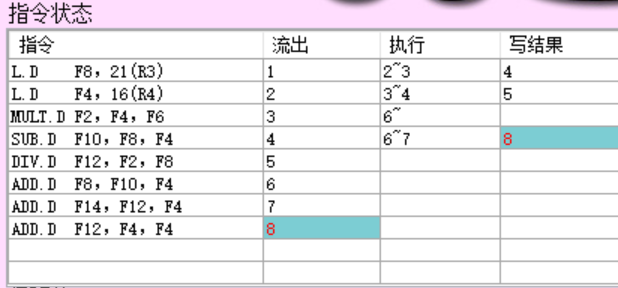
**图3-5-8 第七周期保留站状态展示**

在之后是第八个周期，添加的第二个写F12的指令流出，此时因为刚好第四个指令sub指令完成，所以不会存在结构相关的问题，指令直接存放在保留站的第一个位置，并且此时，F12的对应的保留站指令修改为添加的第二个指令：



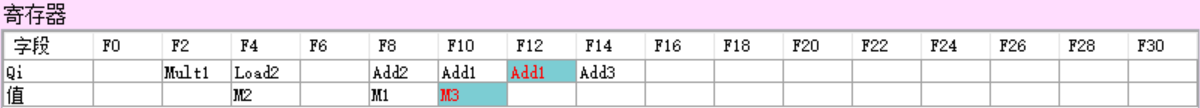
**图3-5-9 第八周期保留站、寄存器状态展示**

此时，继续执行到添加的F12写回指令完成：



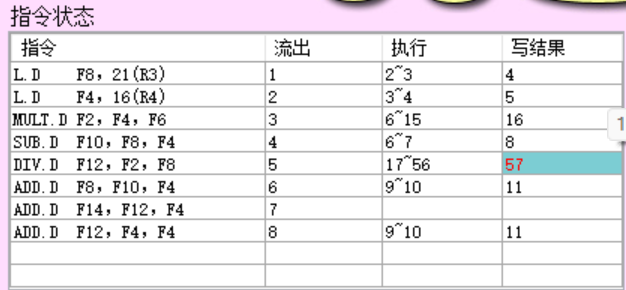
**图3-5-10 第八周期指令状态展示**

此时F12也已经有了自己的数据了：



**图3-5-11 第八周期寄存器状态展示**

如果没有使用tomasulo算法，那么可能这个时候添加的第一条指令就会读取走这个F12的数据并且开始执行，但是因为tomasulo算法的寄存器重命名规则，他的保留站里存的用的操作数是保留站编号为Mult2的指令的结果数据，所以此时不执行，继续等待。  
继续执行到除法指令完成：



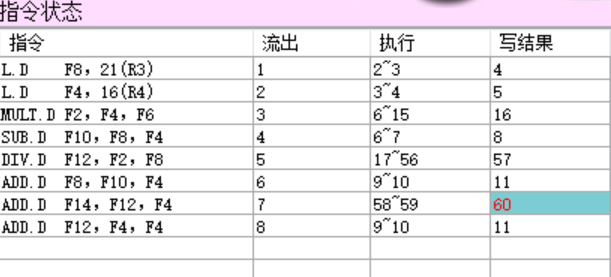
**图3-5-12 指令状态展示**

此时查看保留站，可以看到对应的保留站编号域变更为具体数值域：



**图3-5-13 保留站状态展示**

接下来开始执行



**图3-5-14 指令状态展示**

这里后面的写指令没有对前面的读指令造成影响，说明tomasulo算法可以处理WAR相关。