计算机体系结构



**Tomasulo算法实验报告**

|  |  |
| --- | --- |
| 学 院： | 计算机与信息技术学院 |
| 专 业： | 计算机科学与技术 |
| 指导教师： | 吴为民 |
| 班 级： | 计科1602 |
| 姓 名： | 麻锦涛 |
| 学 号： | 16281262 |

2019年 5月 11日

1. **实验目的：**

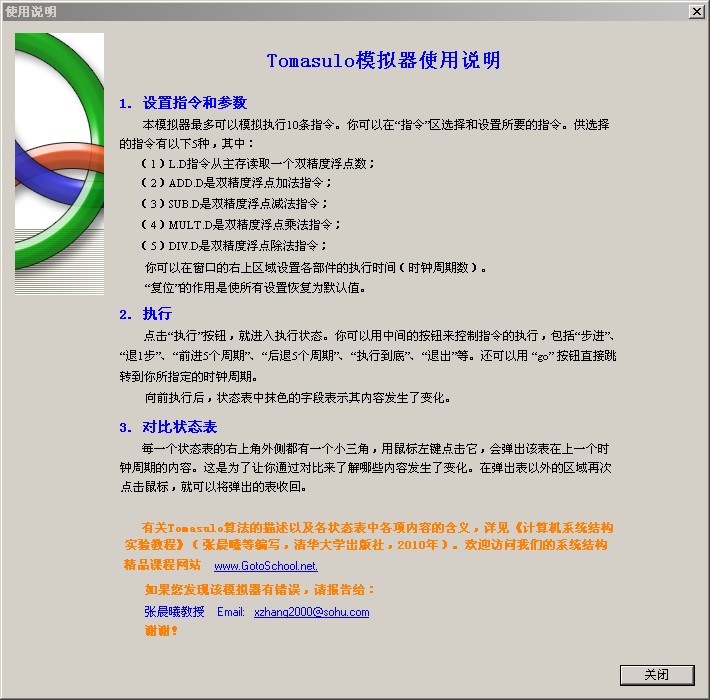
**熟悉 Tomasulo 算法。**

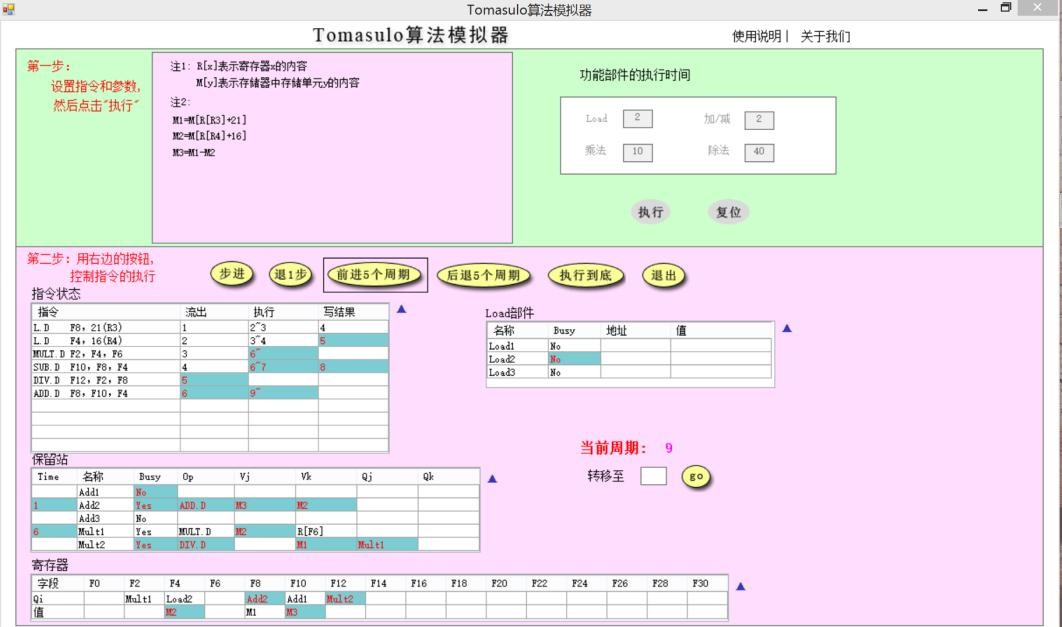
**二、实验内容**

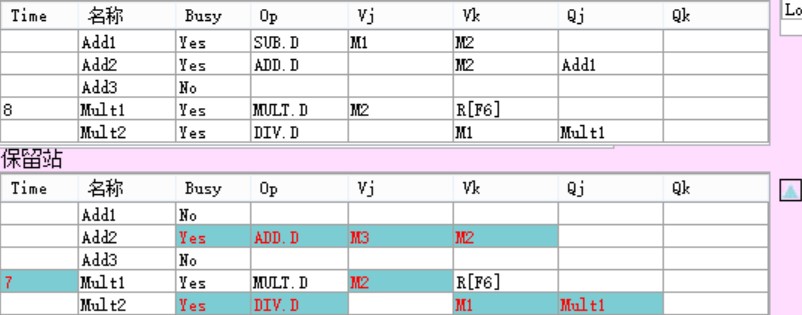
**1. 了解Tomasulo算法的模拟软件**

**运行Tomasulo算法的模拟软件，了解其使用说明。了解 Tomasulo 算法的模拟软件**

下图为 Tomasulo 模拟器的使用说明：

 按照软件自动给出的 6 条指令，学习使用 Tomasulo 算法模拟器过程截图如下：

 使用模拟器提供的对比状态表在前进 5 个周期后的保留站的前后对比图如下：

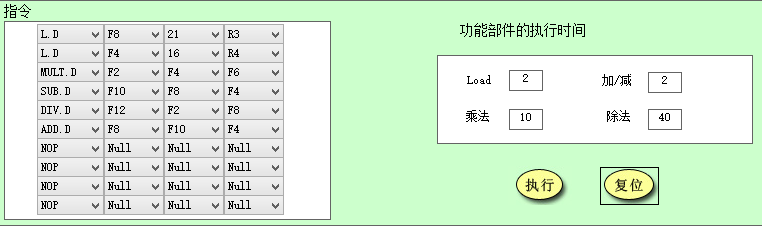


**2. 掌握Tomasulo算法的运行过程及其原理**

**可任意设置一系列指令（当然，按软件自动给出的6条指令也可以），然后，按步进方式运行。每运行一步，均对保留站的状态变化进行解释（如果状态无变化，即只有时钟改变时，可不予解释），直至运行结束。解释的内容包括：**

1. **指明哪条指令从一种状态变到另一种状态。状态包括：流出(我们称为发射)、执行、写结果。**
2. **对于指令的状态变化，保留站（当然也可以包括寄存器、load部件）发生了哪些变化。**

答：按照软件自动给出的6条指令执行，具体指令及功能部件执行时间如下：

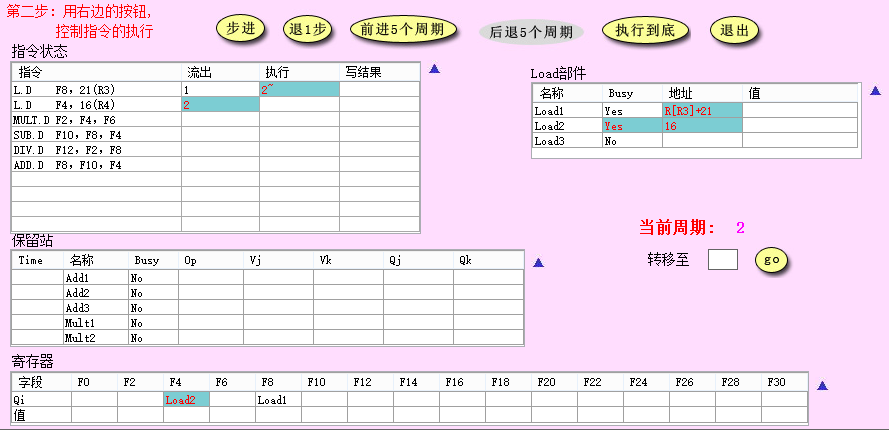


1. 第一个周期，截图如下：



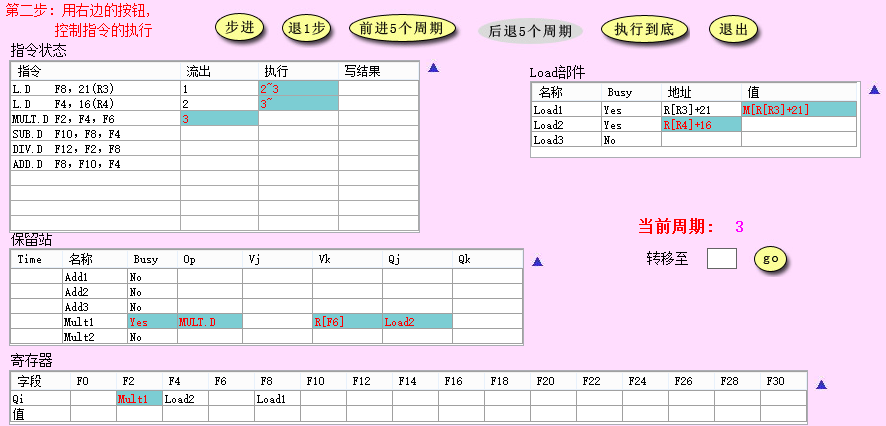
在第一个周期中，由于第一条指令“LD, F8, 21(R3)”为LOAD指令，且LOAD缓冲器有空间，因此该指令进入发射阶段，并在LOAD缓冲器中将其重命名为“Load1”，其对应存储器的地址初始填入第一个操作数21，此外，在寄存器中将F8对应的Qi域置为Load1。

1. 第二个周期，截图如下：



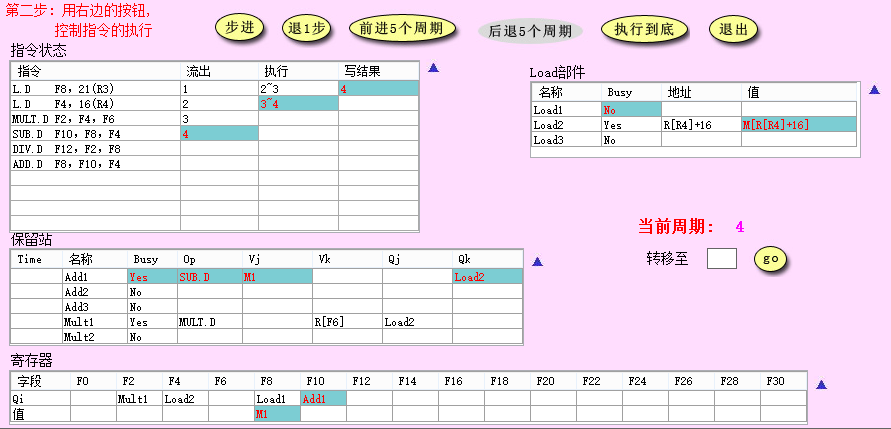
在第二个周期中，指令“LD, F8, 21(R3)”从发射阶段进入执行阶段，计算得到其对应存储器的地址为R[R3]+21。由于第二条指令“LD, F4, 16(R4)”为LOAD指令，且LOAD缓冲器中有空间，因此该指令进入发射阶段，并在LOAD缓冲器中将其重命名为Load2，其对应存储器的地址初始填入第一个操作数16，此外，在寄存器中将F4对应的Qi域置为Load2。

1. 第三个周期，截图如下：



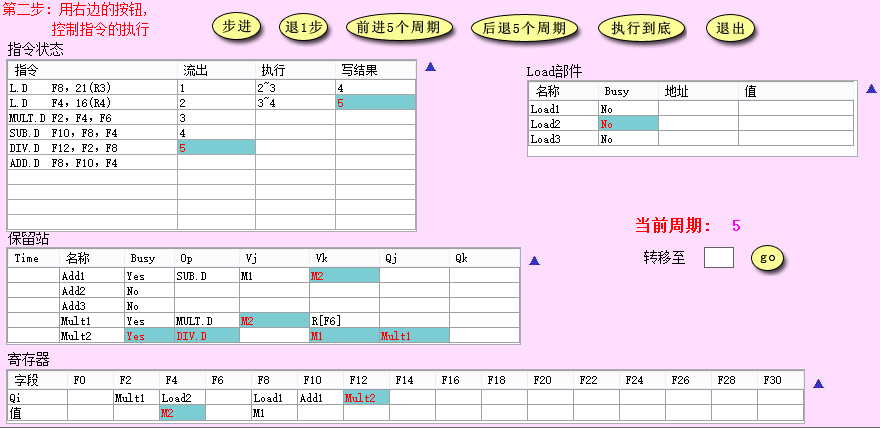
在第三个周期中，指令“LD, F8, 21(R3)”仍处于执行阶段，取出其对应存储器地址中存放的值为M[R[R3]+21]。指令“LD, F4, 16(R4)”从发射阶段进入执行阶段，计算得到其对应存储器的地址为R[R4]+16。由于第三条指令“MULTD F2, F4, F6”执行FP操作，且保留站中有空间，因此该指令进入发射阶段，并在保留站中将其重命名为Mult1，其对应的操作为MULTD，由于第一个操作数F4仍未算出，因此保留站中的Vj域为空，Qj域存放计算F4的LOAD缓冲器，即Load2；由于第二个操作数F6已在寄存器中，因此保留站中的Vk域为R[F6]；此外，在寄存器中将F2对应的Qi域置为Mult1。

1. 第四个周期，截图如下：



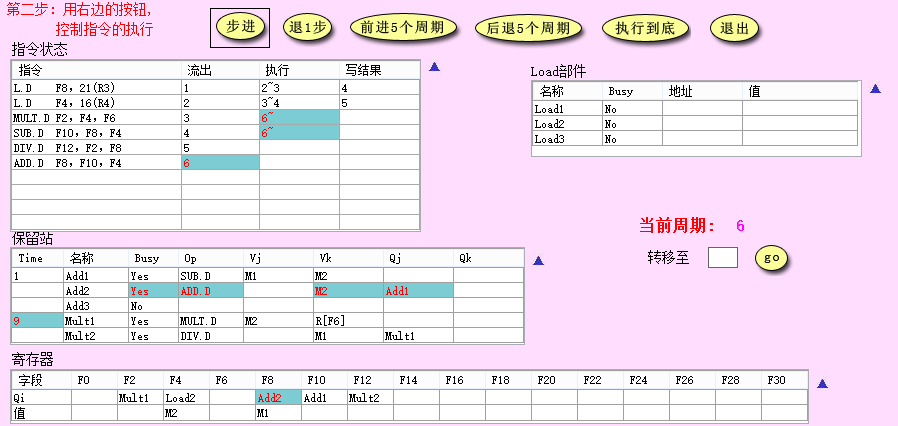
在第四个周期中，指令“LD, F8, 21(R3)”从执行阶段进入写回阶段，将LOAD缓冲器中存放的值记为M1并送入寄存器中，同时释放该指令占用的LOAD缓冲器空间。指令“LD, F4, 16(R4)”仍处于执行阶段，取出其对应存储器地址中存放的值为M[R[R4]+16]。由于Load2还未执行，指令“MULTD F2, F4, F6”无法获得操作数F4的值，因此仍处于发射阶段。由于第四条指令“SUBD F10, F8, F4”执行FP操作，且保留站中有空间，因此该指令进入发射阶段，并在保留站中将其重命名为Add1，其对应的操作为SUBD，由于第一个操作数F8已在寄存器中，因此保留站中的Vj域为M1；由于第二个操作数F4仍未算出，因此保留站中的Vk域为空，Qk域存放计算F4的LOAD缓冲器，即Load2；此外，在寄存器中将F10对应的Qi域置为Add1。

1. 第五个周期，截图如下：



在第五个周期中，指令“LD, F4, 16(R4)”从执行阶段进入写回阶段，将LOAD缓冲器中存放的值记为M2并送入寄存器中，同时释放该指令占用的LOAD缓冲器空间。由于Load2已执行完毕，因此将保留站中Add1和Mult1对应的操作数V域均更新为M2。由于第五条指令“DIVD F12, F2, F8”执行FP操作，且保留站中有空间，因此该指令进入发射阶段，并在保留站中将其重命名为Mult2，其对应的操作为DIVD，由于第一个操作数F2仍未算出，因此保留站中的Vj域为空，Qj域存放计算F2的保留站，即Mult1；由于第二个操作数F8已在寄存器中，因此保留站中的Vk域为M1；此外，在寄存器中将F12对应的Qi域置为Mult2。

1. 第六个周期，截图如下：

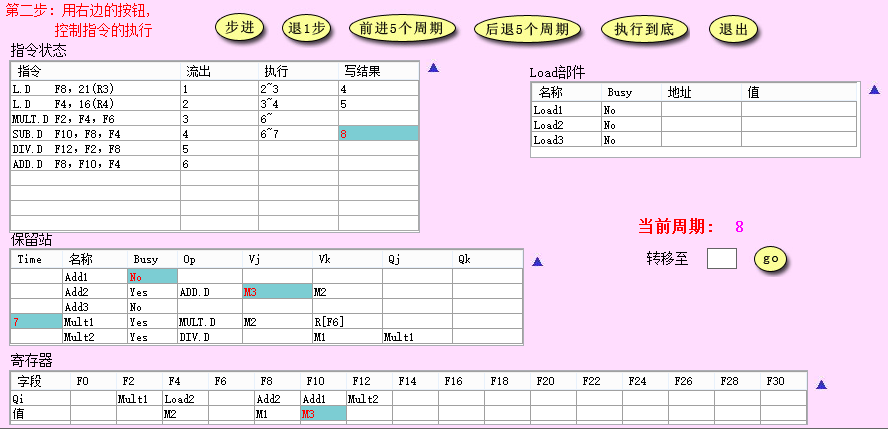


由于第五个周期中指令“MULTD F2, F4, F6”和指令“SUBD F10, F8, F4”均获得了全部操作数的值，因此在第六个周期中，两条指令均进入执行阶段。由于F2还未计算出来，因此指令“DIVD F12, F2, F8”仍处于发射阶段。由于第六条指令“ADDD F8, F10, F4”执行FP操作，且保留站中有空间，因此该指令进入发射阶段，并在保留站中将其重命名为Add2，其对应的操作为ADDD，由于第一个操作数F10仍未算出，因此保留站中的Vj域为空，Qj域存放计算F10的保留站，即Add1；由于第二个操作数F4已在寄存器中，因此保留站中的Vk域为M2；此外，在寄存器中将F8对应的Qi域置为Add2。

1. 第七个周期：

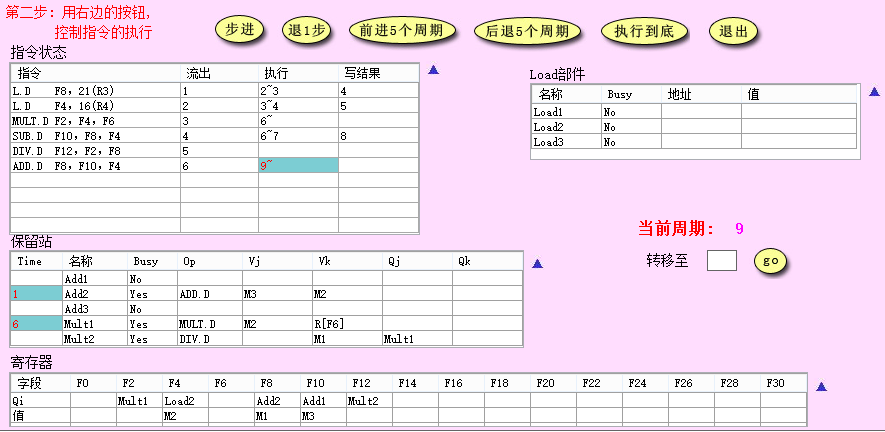
在第七个周期中，指令“MULTD F2, F4, F6”和“SUBD F10, F8, F4”均仍处于执行状态。指令“DIVD F12, F2, F8”和“ADDD F8, F10, F4”均由于未获得全部的操作数而处于发射状态。保留站、寄存器及LOAD缓冲器中的状态均无变化。

1. 第八个周期，截图如下：



在第八个周期中，指令“MULTD F2, F4, F6”仍处于执行状态。指令“SUBD F10, F8, F4”从执行阶段进入写回阶段，将得到的结果记为M3并送入寄存器中，同时释放该指令占用的保留站空间。由于Add1已执行完毕，因此将保留站中Add2对应的操作数V域更新为M3。

1. 第九个周期，截图如下：

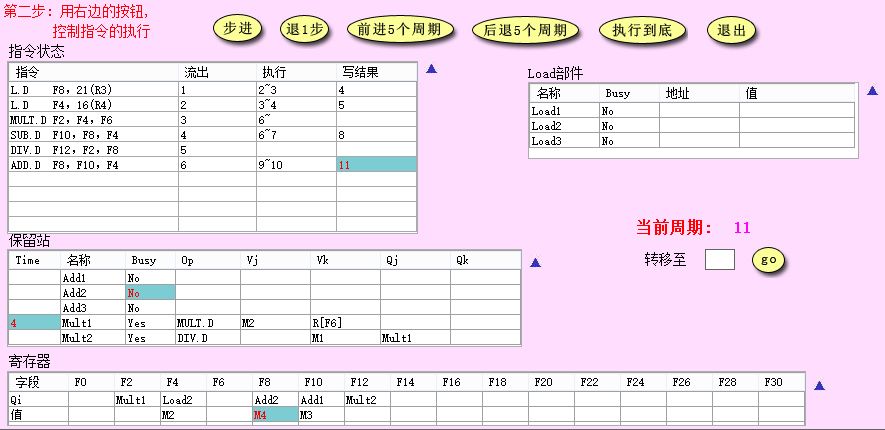


在第九个周期中，指令“MULTD F2, F4, F6”仍处于执行状态。由于F2还未计算出来，因此指令“DIVD F12, F2, F8”仍处于发射阶段。由于第八个周期中指令“ADDD F8, F10, F4”获得了全部操作数的值，因此在第九个周期中，该指令进入执行阶段。

1. 第十个周期：

在第十个周期中，指令“MULTD F2, F4, F6”和“ADDD F8, F10, F4”均仍处于执行状态。指令“DIVD F12, F2, F8”由于未获得全部的操作数而处于发射状态。保留站、寄存器及LOAD缓冲器中的状态均无变化。

1. 第十一个周期，截图如下：

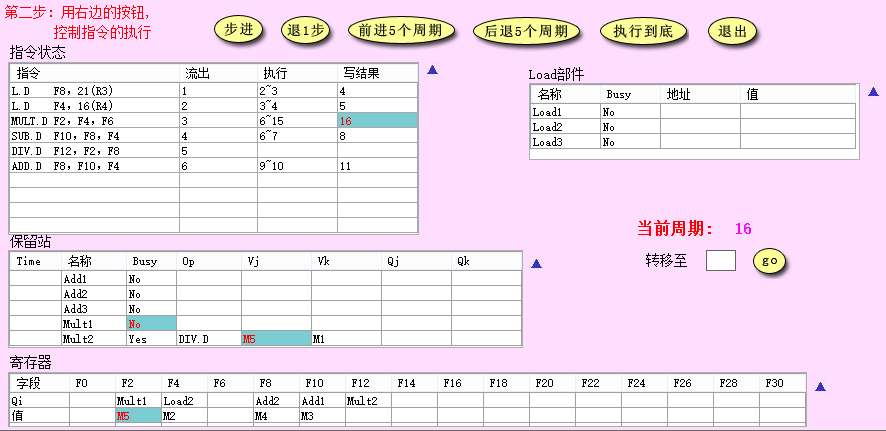


在第十一个周期中，指令“MULTD F2, F4, F6”仍处于执行状态。指令“ADDD F8, F10, F4”从执行阶段进入写回阶段，将得到的结果记为M4并送入寄存器中，同时释放该指令占用的保留站空间。指令“DIVD F12, F2, F8”由于未获得全部的操作数而处于发射状态。

1. 第十二个周期~第十五个周期：

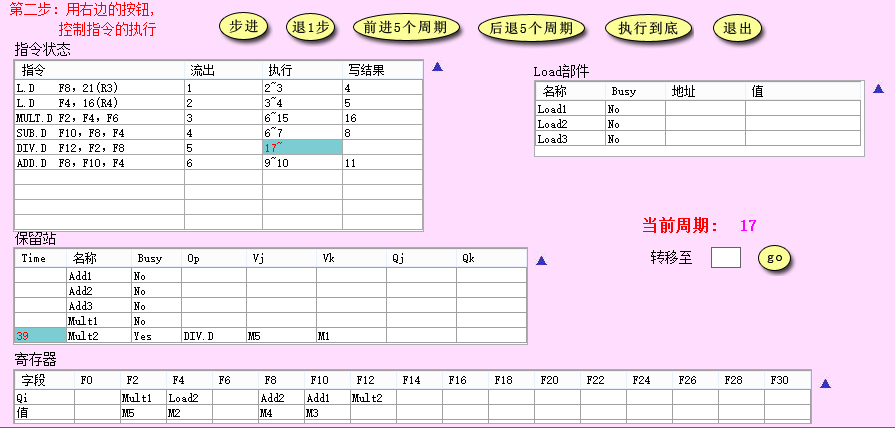
在第十二个周期到第十五个周期中，指令“MULTD F2, F4, F6”始终处于执行状态。指令“DIVD F12, F2, F8”由于未获得全部的操作数而处于发射状态。保留站、寄存器及LOAD缓冲器中的状态均无变化。

1. 第十六个周期，截图如下：



在第十六个周期中，指令“MULTD F2, F4, F6”从执行阶段进入写回阶段，将得到的结果记为M5并送入寄存器中，同时释放该指令占用的保留站空间。指令“DIVD F12, F2, F8”由于未获得全部的操作数而处于发射状态。由于Mult1已执行完毕，因此将保留站中Mult2对应的操作数V域更新为M5。

1. 第十七个周期，截图如下：

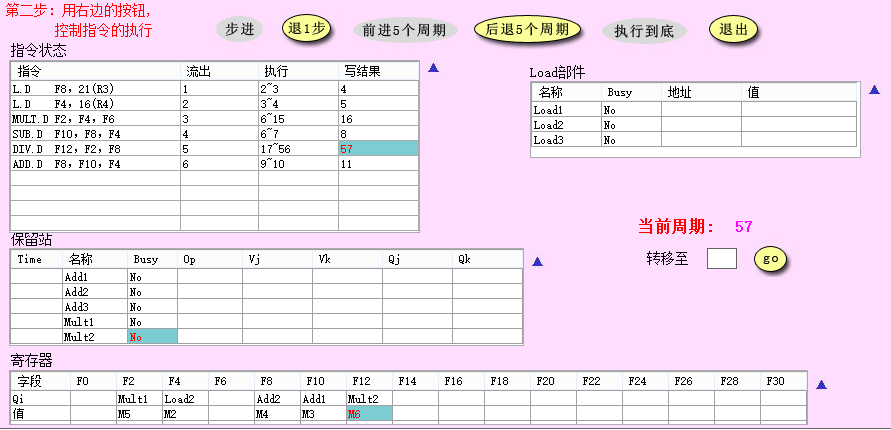


由于第十六个周期中指令“DIVD F12, F2, F8”获得了全部操作数的值，因此在第十七个周期中，该指令均进入执行阶段。

1. 第十八个周期~第五十六个周期：

在第十八个周期到第五十六个周期中，指令“DIVD F12, F2, F8”始终处于执行状态。保留站、寄存器及LOAD缓冲器中的状态均无变化。

1. 第五十七个周期，截图如下：



在第五十七个周期中，指令“DIVD F12, F2, F8”从执行阶段进入写回阶段，将得到的结果记为M6并送入寄存器中，同时释放该指令占用的保留站空间。

**3. 举例解释为什么Tomasulo算法消除了这些指令中的WAR和WAW冒险。**

**WAR 和 WAW 实质是命名冲突，而不存在数据依赖。Tomasulo 算法通过寄存器。换名来消除 WAR 和 WAW 冒险。寄存器换名是通过保留站来实现的。**

1. WAR冒险

存在 WAR 冒险的语句组：

S1：MULTD F8，F2，F2

S2：ADDD F2， F0， F4

看起来S2必须要等待S1从F2 中读取数据后，才可以向F2中写数据， （WAR）

但是如果将 S2 中的目标操作数换为 F10 就完全不存在冒险。因此在 Tomasulo

算法中额外设置寄存器，也就是保留站，它作为 S2 的“临时目标寄存器”，

并且把这个“临时目标寄存器”和真正的目标寄存器 F2 关联起来，就是寄存

器的重命名，这样在执行 S2 语句时并不是将结果写入寄存器 F2 中，而是写

入与之关联的保留站中，这样就消除了 WAR 冒险。

1. WAW冒险

存在 WAW 冒险的语句组：

S1：ADDD F0， F2， F4

S2：MULTD F2， F6， F8

S3：MULTD F10，F0， F2

S4：ADDD F0， F12， F14

中间 S3 将 F0 作为源操作数，S1，S4 都写入 F0 中，当结果计算出来之后，

将其先写入 CDB（公共数据总线），而不是直接写入寄存器 F0，之后从 CDB

写入目的寄存器以及等待此结果的保留站，当连续写同一寄存器时，只有最

后一次才能写入，从而消除了 WAW 冒险