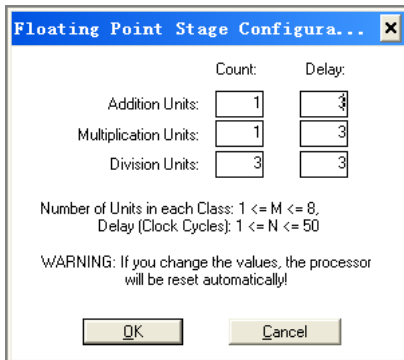
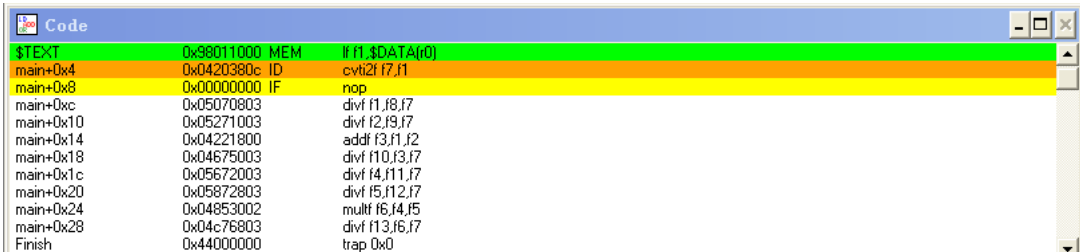
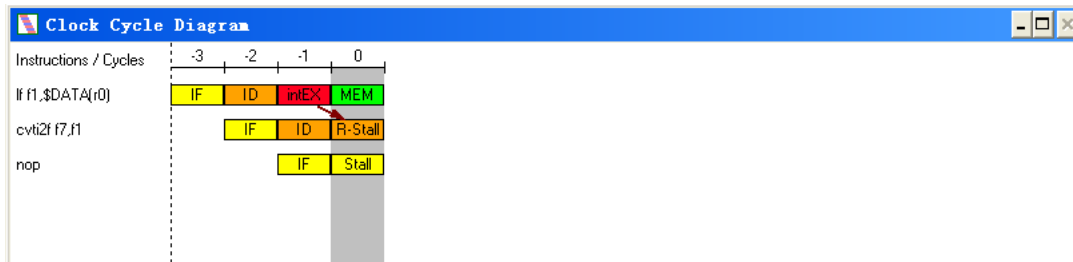


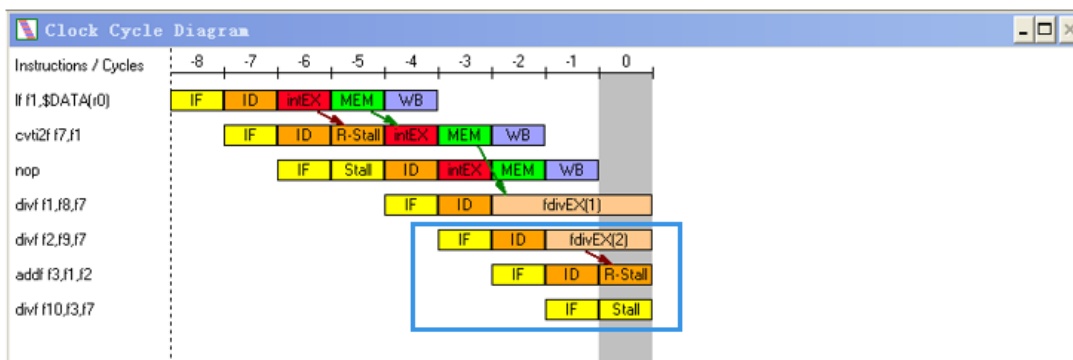
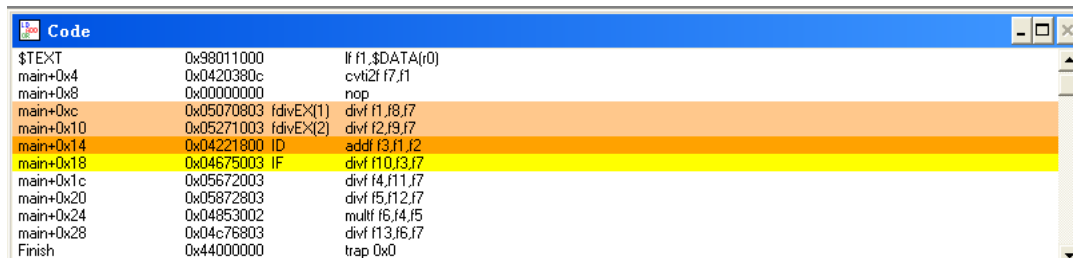
山东大学计算机科学与技术学院

计算机体系结构课程实验报告

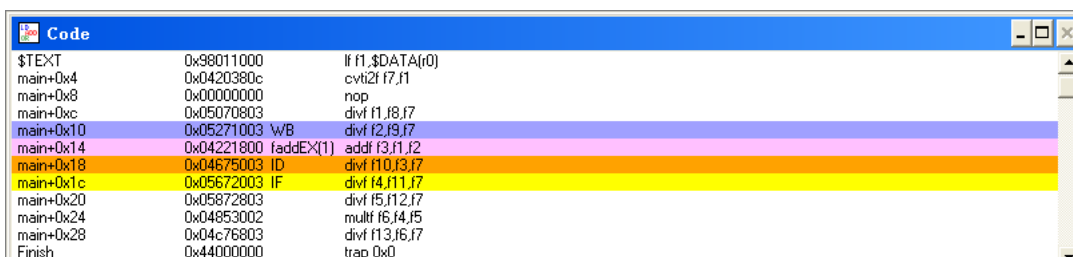
学号：201800130031	姓名：来苑	班级：计科 1 班
实验题目：实验六 指令调度		
实验学时：2 学时	实验日期：2021. 6. 14	
实验目的： 通过本实验，加深对指令调度的理解，了解指令调度技术对 CPU 性能改进的好处。		
硬件环境： 机房		
软件环境： WindowsXp		
实验步骤与内容： 1. 通过 <i>Configuration</i> 菜单中的“ <i>Floating point stages</i> ”选项，把除法单元数设置为 3，把加法、乘法、除法的延迟设置为 3 个时钟周期。  2. 用 WinDLX 模拟器运行调度前的程序 sch-before.s 。记录程序执行过程中各种相关发生的次数以及程序执行的总时钟周期数。 单步执行，至此发生了第一次数据相关，以及相应产生的第一次数据相关。由于指令 If f1,ONE 与指令 cvti2f f7,f1 关于寄存器 f1 产生了数据相关，所以产生了一个 R-Stall。而由于指令 cvti2f f7,f1 暂停在译码阶段 ID，所以与下一条指令 nop 产生了结构相关。		
		

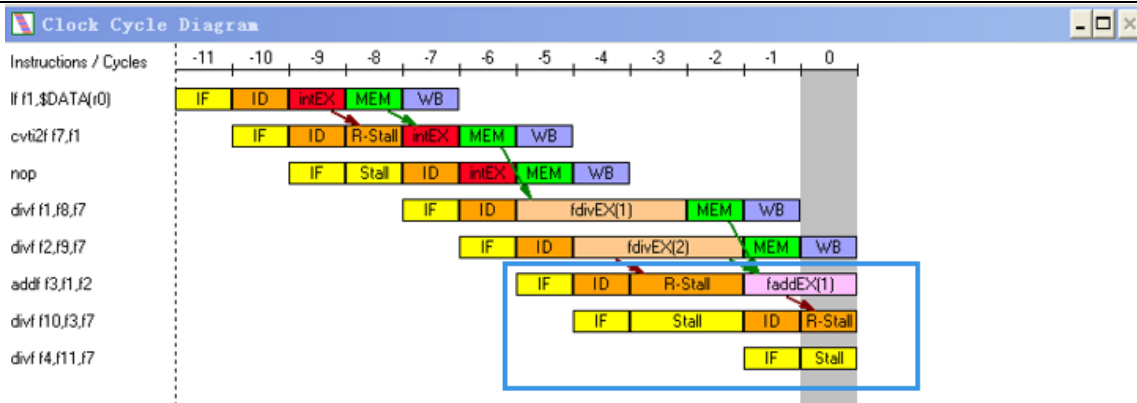


继续单步执行，此处发生了第二次数据相关，以及相应产生的第二次结构相关。由于指令 `divf f2,f9,f7` 与指令 `addf f3,f1,f2` 关于寄存器 `f2` 产生了数据相关，所以产生了一个 R-Stall。而由于指令 `addf f3,f1,f2` 暂停在译码阶段 ID，所以与下一条指令 `divf f10,f3,f7` 产生了结构相关。

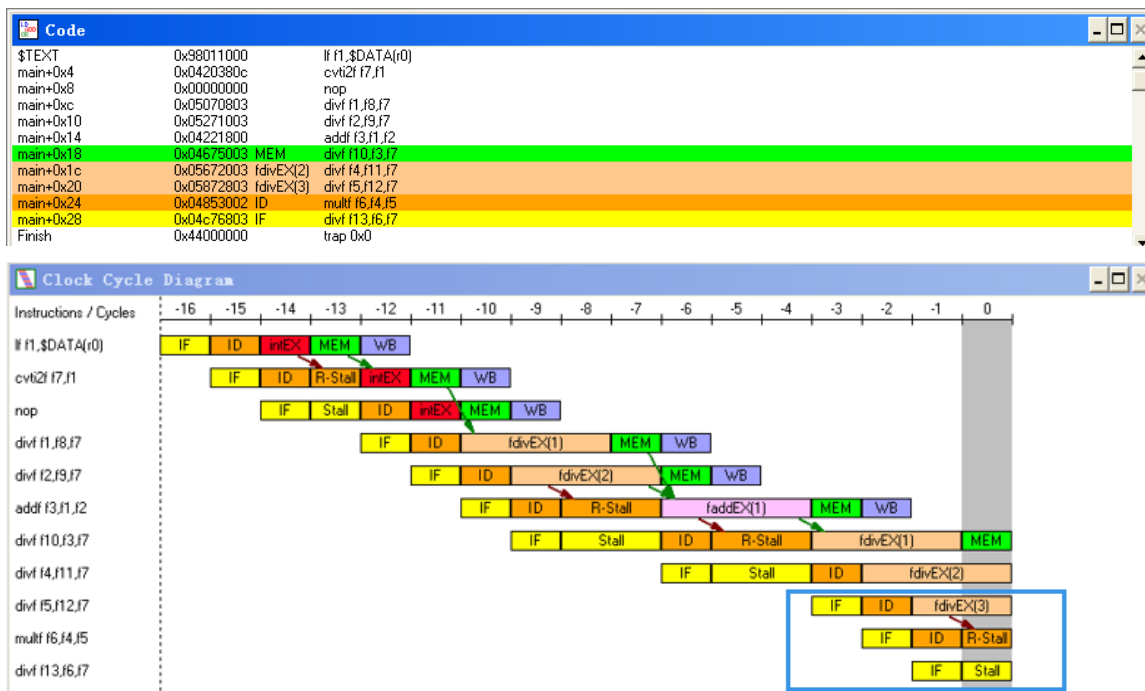


继续单步执行，此处发生了第三次数据相关，以及相应产生的第三次结构相关。由于指令 `divf f10,f3,f7` 与指令 `addf f3,f1,f2` 关于寄存器 `f3` 产生了数据相关，所以产生了一个 R-Stall。而由于指令 `divf f10,f3,f7` 暂停在译码阶段 ID，所以与下一条指令 `divf f4,f11,f7` 产生了结构相关。



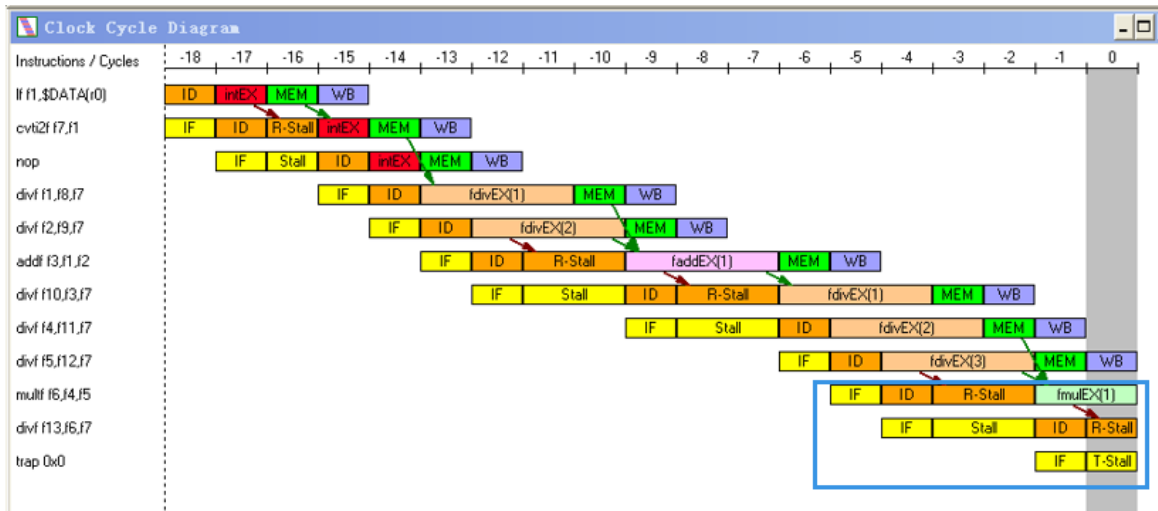


继续单步执行，此处发生了第四次数据相关，以及相应产生的第四次结构相关。由于指令 **multf f6,f4,f5** 与指令 **divf f5,f12,f7** 关于寄存器 f5 产生了数据相关，所以产生了一个 R-Stall。而由于指令 **divf f5,f12,f7** 暂停在译码阶段 ID，所以与下一条指令 **divf f13,f6,f7** 产生了结构相关。

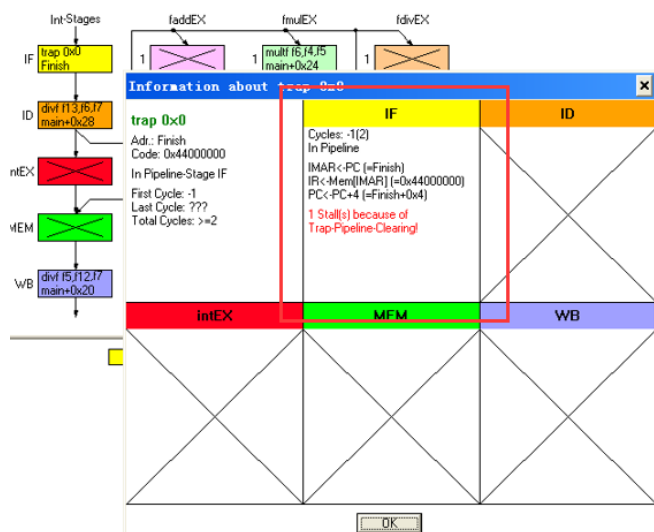


继续单步执行，此处发生了第五次数据相关。由于指令 **multf f6,f4,f5** 与指令 **divf f13,f6,f7** 关于寄存器 f6 产生了数据相关，所以产生了一个 R-Stall。而由于指令 **divf f13,f6,f7** 暂停在译码阶段 ID，所以与下一条指令 **trap 0** 也产生了冲突，为了等待指令 **divf f13,f6,f7** 结束，这里产生了 T-Stall。

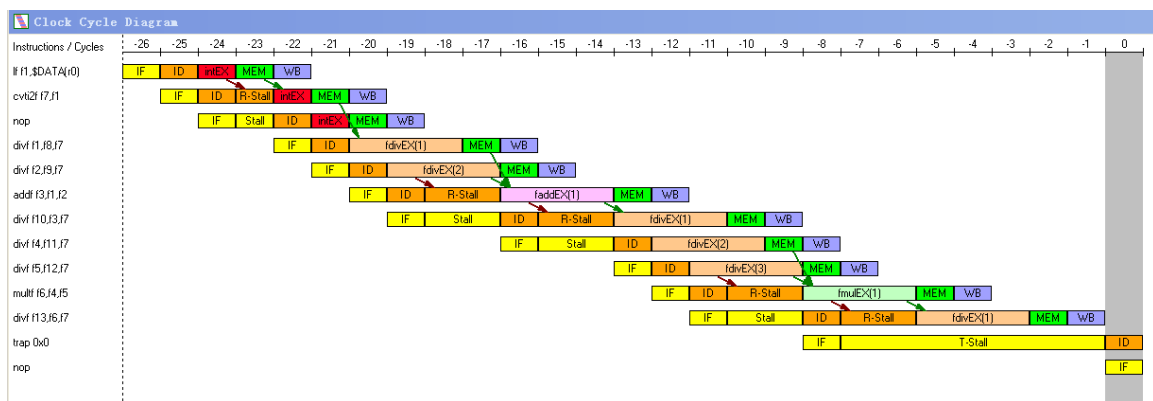




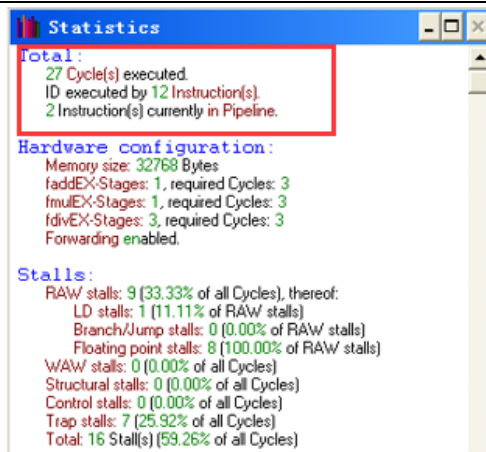
T-Stall:



程序完整 Clock 图如下:

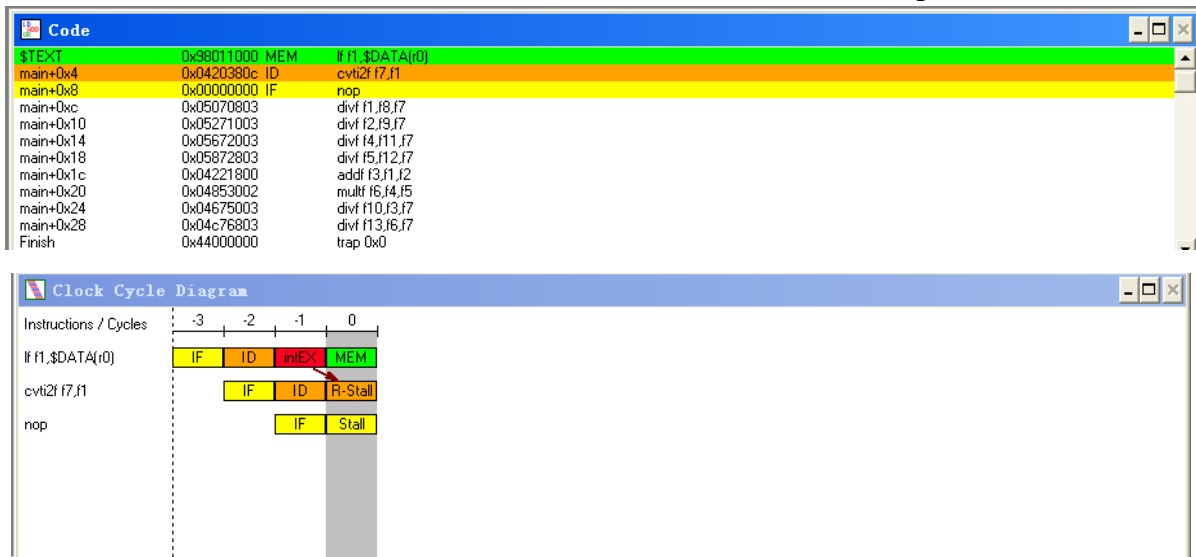


根据记录，程序执行过程中，一共发生了数据相关 5 次、结构相关 4 次、控制相关 1 次。

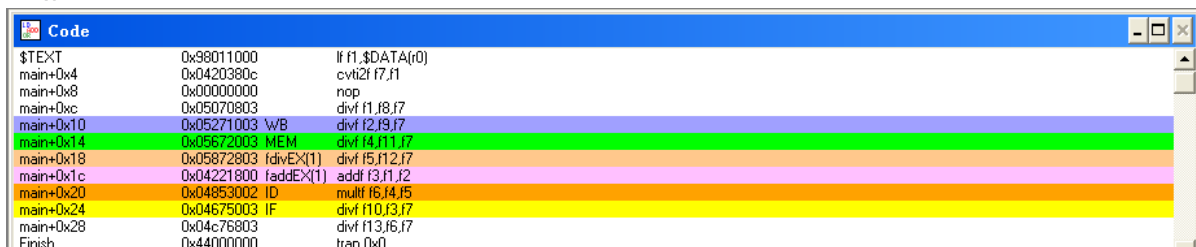


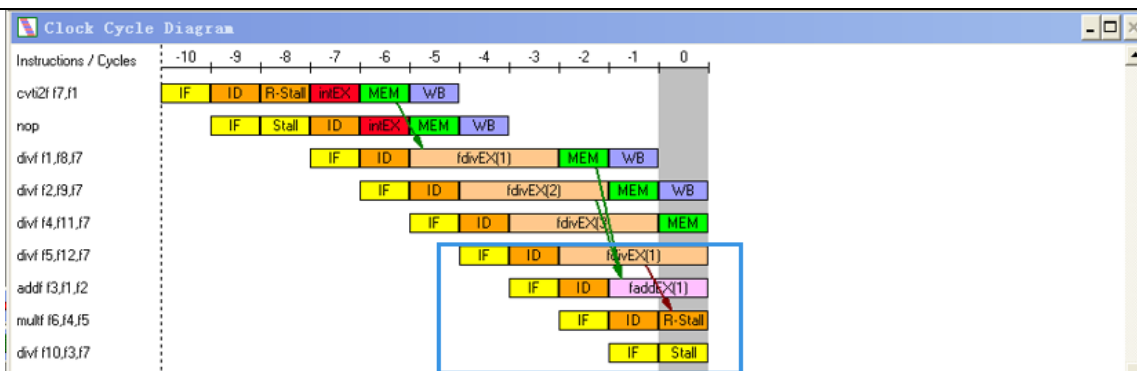
3. 用 WinDLX 模拟器运行调度后的程序 sch-after.s，记录程序执行过程中各种相关发生的次数以及程序执行的总时钟周期数。

单步执行，至此发生了第一次数据相关，以及相应产生的第一次数据相关。由于指令 **If f1,ONE** 与指令 **cvti2f f7,f1** 关于寄存器 f1 产生了数据相关，所以产生了一个 R-Stall。而由于指令 **cvti2f f7,f1** 暂停在译码阶段 ID，所以与下一条指令 **nop** 产生了结构相关。

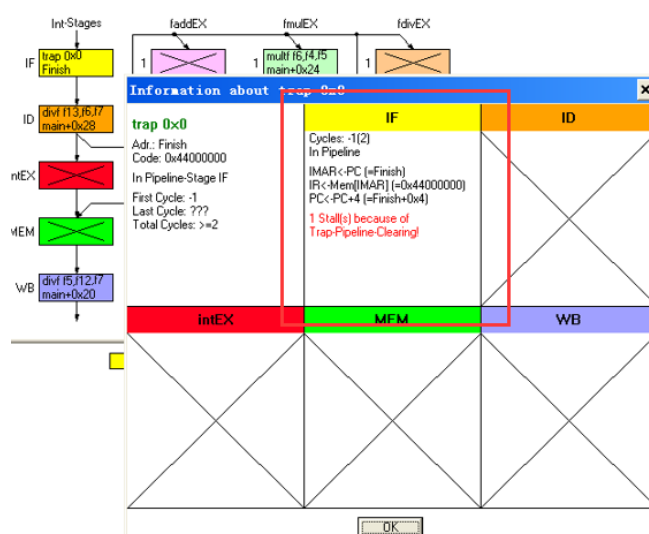
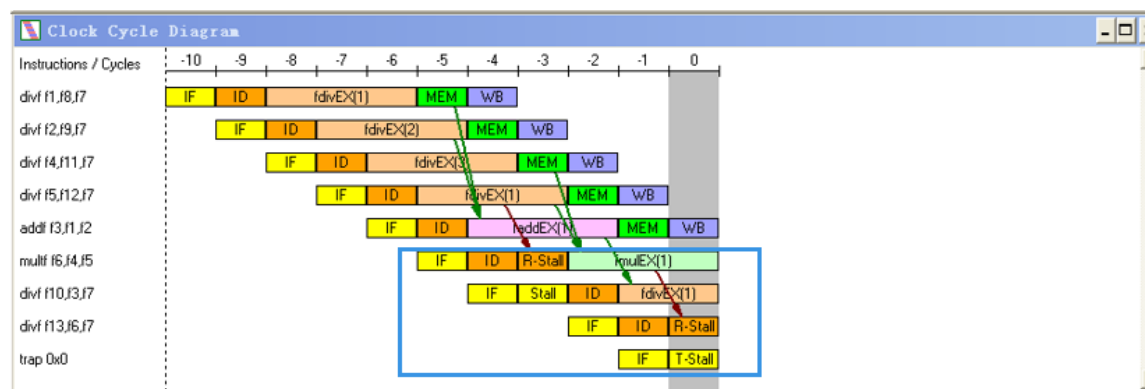
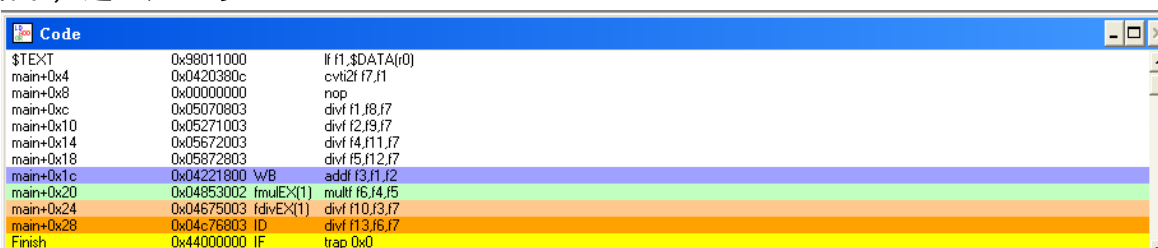


继续单步执行，此处发生了第二次数据相关，以及相应产生的第二次结构相关。由于指令 **divf f5,f12,f7** 与指令 **multf f6,f4,f5** 关于寄存器 f2 产生了数据相关（因为除法所需的时钟周期较长，所以其后三条指令都是可能产生冲突的指令），所以产生了一个 R-Stall。而由于指令 **multf f6,f4,f5** 暂停在译码阶段 ID，所以与下一条指令 **divf f10,f3,f7** 产生了结构相关。

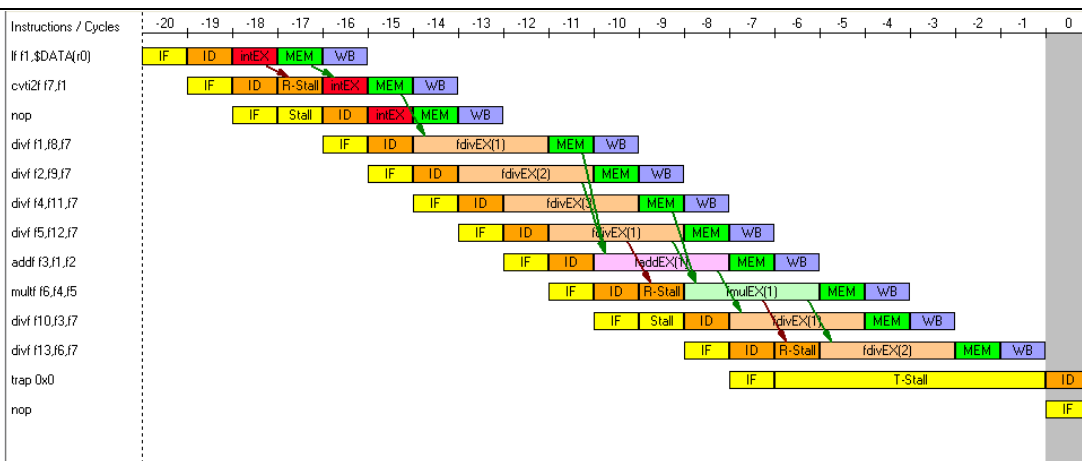




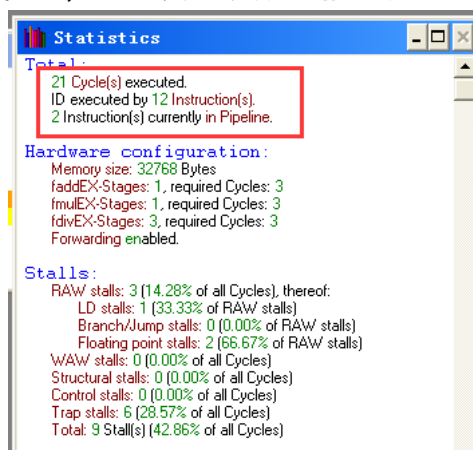
继续单步执行，此处发生了第三次数据相关。由于指令 **multf f6,f4,f5** 与指令 **divf f13,f6,f7** 关于寄存器 f6 产生了数据相关，所以产生了一个 R-Stall。而由于指令 **divf f13,f6,f7** 暂停在译码阶段 ID，所以与下一条指令 **trap 0** 也产生了冲突，为了等待指令 **divf f13,f6,f7** 结束，这里产生了 T-Stall。



程序完整 Clock 图如下：



根据记录，程序执行过程中，一共发生了数据相关 3 次、结构相关 2 次、控制相关 1 次。



4. 根据记录结果，比较调度前和调度后的性能。

调度前所需时钟周期数为 27，调度后所需时钟周期数为 21，性能提升到原来的 $27/21=1.286$ 倍。

5. 论述指令调度对于提高 CPU

调度后程序与调度前程序的差别：

```

nop                                ;floating-point format
divf      f1,f8,f7                  ;move Y=(f8) into f1 (f1)=(f8)/(f7)
divf      f2,f9,f7                  ;move Z=(f9) into f2 (f2)=(f9)/(f7)
addf      f3,f1,f2                  ;(f3)=(f1)+(f2) 数据相关
divf      f10,f3,f7                 ;move f3 into X=(f10) (f10)=(f3)/(f7)
divf      f4,f11,f7                 ;move B=(f11) into f4 (f4)=(f11)/(f7)
divf      f5,f12,f7                 ;move C=(f12) into f5 (f5)=(f12)/(f7)
multf     f6,f4,f5                  ;(f6)=(f4)*(f5) 数据相关
divf      f13,f6,f7                 ;move f6 into A=(f13) (f13)=(f6)/(f7)

```



```

nop                                ;floating-point format
divf      f1,f8,f7                  ;move Y=(f8) into f1
divf      f2,f9,f7                  ;move Z=(f9) into f2
divf      f4,f11,f7                 ;move B=(f11) into f4
divf      f5,f12,f7                 ;move C=(f12) into f5
addf      f3,f1,f2
multf     f6,f4,f5
divf      f10,f3,f7                 ;move f3 into X=(f10)
divf      f13,f6,f7                 ;move f6 into A=(f13)

```

这样调度主要是为了解决乘除法指令产生的数据相关。在会产生数据相关的乘除法指令间尽可能多的放置只与其他寄存器相关的指令，从而在不影响程序正确性的情况下提高

性能。

合适的指令调度可以避免数据冲突，从而提高 CPU 性能。

结论分析与体会：

通过本次实验，我对流水线的五个阶段有了更深的理解，也加深了对数据相关、结构相关、控制相关以及其解决方法的认识。同时，我也进一步体会到了良好的指令调度可以大量避免数据相关，从而提高性能。