A.2 实验二 流水线及流水线中的冲突

**A.2.1 实验目的**

1. 加深对计算机流水线基本概念的理解。
2. 理解MIPS结构如何用5段流水线来实现，理解各段的功能和基本操作。
3. **加深对**数据冲突、结构冲突**的理解，理解**这两类冲突对CPU性能的影响。
4. 进一步理解解决数据冲突的方法，**掌握如何应用定向技术来减少数据冲突引起的停顿。**

**A.2.2 实验平台**

实验平台采用指令级和流水线操作级模拟器MIPSsim。

环境的建立：见A.0。

**A.2.3 实验内容和步骤**

首先要掌握MIPSsim模拟器的使用方法。详见附录B。

1. 启动MIPSsim。

2．根据预备知识中关于流水线各段操作的描述，进一步理解流水线窗口中各段的功能，掌握各流水寄存器的含义。（用鼠标双击各段，就可以看到各流水寄存器的内容）

3. 熟悉MIPSsim模拟器的操作和使用方法。

可以先载入一个样例程序（在本模拟器所在的文件夹下的“样例程序”文件夹中），然后分别以单步执行一个周期、执行多个周期、连续执行、设置断点等的方式运行程序，观察程序的执行情况，观察CPU中寄存器和存储器的内容的变化，特别是流水寄存器内容的变化。

4. 选择配置菜单中的“流水方式”，使模拟器工作于流水方式下。

5．观察程序在流水线中的执行情况，步骤如下：

1. 选择MIPSsim的“文件”→“载入程序”选项来加载pipeline.s（在模拟器所在文件夹下的“样例程序”文件夹中）。
2. 关闭定向功能。这是通过在“配置”→“定向”（使该项前面没有“√”号）来实现的。
3. 用单步执行一周期的方式（“执行”菜单中）或用F7执行该程序，观察每一周期中，各段流水寄存器内容的变化、指令的执行情况（“代码”窗口）以及时钟周期图。
4. 当执行到第13个时钟周期时，各段分别正在处理的指令是：

**（解释每条指令的含义）**

IF： **LW $r4,60($r6)**

载入指令，从内存地址中（r6+60）处读取内容，将其存储到寄存器r4中。

ID： **ADDI $r3,$r0,25**

加法指令，将寄存器r0的值加上立即数25，并将结果存储到寄存器r3。

EX： **ADDI $r1,$r1,-1**

带立即数的加法指令，将寄存器r1的值减去1，并将结果存储回寄存器r1。

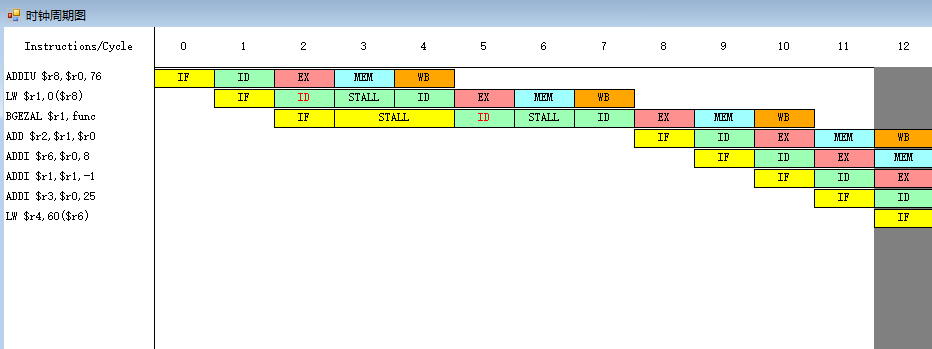
MEM： **ADDI $r6,$r0,8**

带立即数的加法指令，将寄存器r0的值加上8，并将结果存储到寄存器r6。

WB： **ADD $r2,$r1,0**

带立即数的加法指令。将 r1 和 0相加，然后将结果存储到 r2 寄存器中。

画出这时的时钟周期图。（解释插入几个暂停的原因）



6. 这时各流水寄存器中的内容为：

指令依次使用资源，有些指令会实现RAW冲突。为一些指令暂停执行,为消除结构冲突而插入流水线气泡。

（解释下面几个流水寄存器的含义）

IF/ID.IR： 0x8CC4003C

IF/ID.NPC： 0x030

ID/EX.A： 0x00

ID/EX.B： 0x00

ID/EX.Imm： 0x019

ID/EX.IR： 0x20030019

EX/MEM.ALUo： 0x004

EX/MEM.IR： 0x2021FFFF

MEM/WB.LMD： 0x00

MEM/WB.ALUo： 0x008

MEM/WB.IR： 0x20060008

（解释下面几个流水寄存器的含义）

**IF/ID.IR**： 保存从指令存储器中读取的当前指令的内容。

**IF/ID.NPC**： 这是下一个程序计数器（PC）的值，即指令地址。

**ID/EX.A**： 这是在ID阶段计算得到的第一个源操作数的值。

**ID/EX.B**： 这是在ID阶段计算得到的第二个源操作数的值。

**ID/EX.Imm**： 这是从指令中提取的立即数。

**ID/EX.IR**： 这是指令存储寄存器。

**EX/MEM.ALUo**： 这是在EX阶段计算得到的ALU（算术逻辑单元）的输出结果。在EX阶段的末尾，ALU的输出结果被传递到EX/MEM.ALUo寄存器。

**EX/MEM.IR**： 这是指令存储寄存器。

**MEM/WB.LMD**： 这是从内存中读取的数据（通常是加载操作的结果）。

**MEM/WB.ALUo**： 这是在MEM阶段计算得到的ALU的输出结果。

**MEM/WB.IR**： 这是在MEM阶段计算得到的ALU的输出结果。

7. 观察和分析结构冲突对CPU性能的影响，步骤如下：

（1）加载structure\_hz.s（在模拟器所在文件夹下的“样例程序”文件夹中）。

（2）执行该程序，找出存在结构冲突的指令对以及导致结构冲突的部件。**（写出存在结构冲突的指令并画出流水线的时空图）**

**存在结构冲突的指令:**

ADD.D $f3,$f0,$f1和ADD.D $f2,$f0,$f1结构冲突，导致结构冲突的部件：浮点加法器

ADD.D $f4,$f0,$f1和ADD.D $f3,$f0,$f1结构冲突，导致结构冲突的部件：浮点加法器

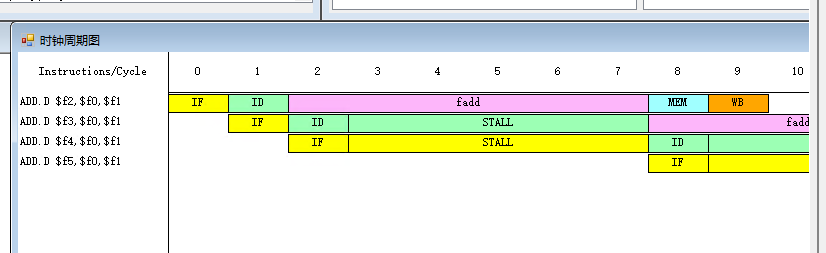
ADD.D $f5,$f0,$f1和ADD.D $f4,$f0,$f1结构冲突，导致结构冲突的部件：浮点加法器

ADD.D $f6,$f0,$f1和ADD.D $f5,$f0,$f1结构冲突，导致结构冲突的部件：浮点加法器

ADD.D $f7,$f0,$f1和ADD.D $f6,$f0,$f1结构冲突，导致结构冲突的部件：浮点加法器

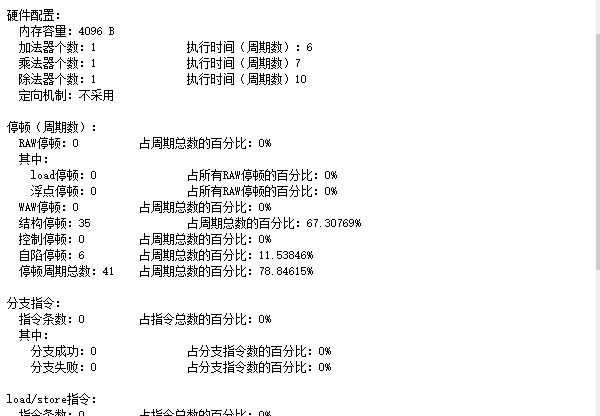
ADD.D $f8,$f0,$f1和ADD.D $f7,$f0,$f1结构冲突，导致结构冲突的部件：浮点加法器

ADD.D $f9,$f0,$f1和ADD.D $f8,$f0,$f1结构冲突，导致结构冲突的部件：浮点加法器



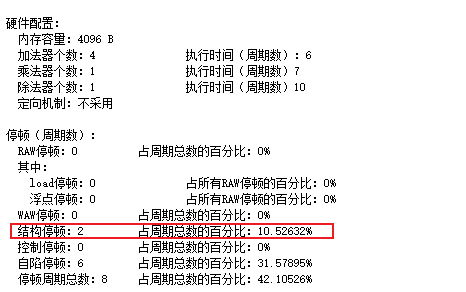
记录由结构冲突引起的停顿时钟周期数，计算停顿时钟周期数占总执行周期数的百分比；

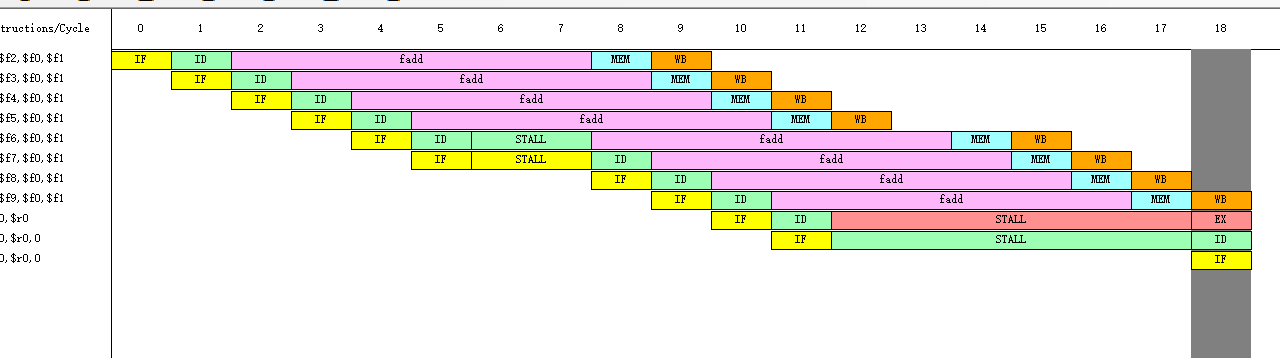
结构停顿：35 占周期总数的百分比：67.30769%



（4）把浮点加法器的个数改为4个。

（5）再次重复上述（1）～（3）的工作。**（画出改进后的流水线的时空图）**





（6）分析结构冲突对CPU性能的影响，讨论解决结构冲突的方法。

**对CPU性能的影响:**让程序的流水运行出现结构停顿，需要等待其他指令执行，大大降低程序执行速度，降低CPU性能。

**解决方法:**

①　插入气泡

②　数据旁路

③　换名

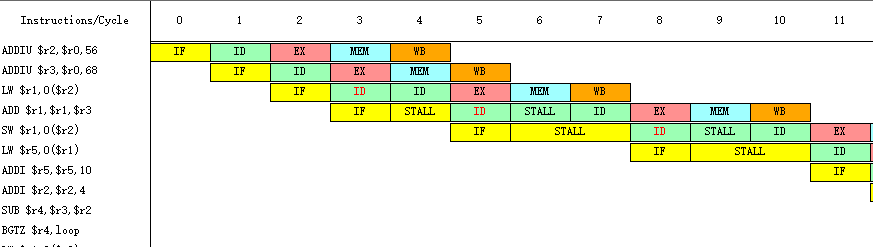
8. 观察数据冲突并用定向技术来减少停顿，步骤如下：

（1）全部复位。

（2）加载data\_hz.s（在模拟器所在文件夹下的“样例程序”文件夹中）。

（3）关闭定向功能。这是通过在“配置”→“定向”（使该项前面没有“√”号）来实现的。

（4）用单步执行一个周期的方式（F7）执行该程序，同时查看时钟周期图，列出在什么时刻发生了RAW（先写后读）冲突。（解释前8条指令之间存在的相关性，画出流水线的时空图，解释其中暂停的含义）



**前8条指令之间存在的相关性:**

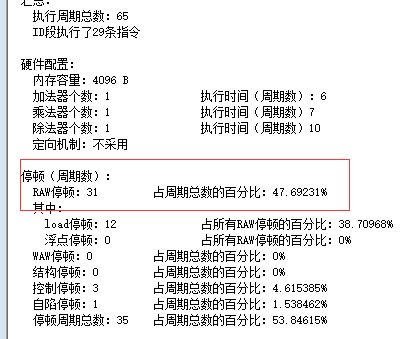
ADDIU $r2, $r0, A 和 LW $r1, 0($r2)：数据依赖关系。后一条指令依赖于前一条指令的结果，即r2 寄存器中的值。

其他指令类似。

**暂停的含义：**

防止出现RAW错误。

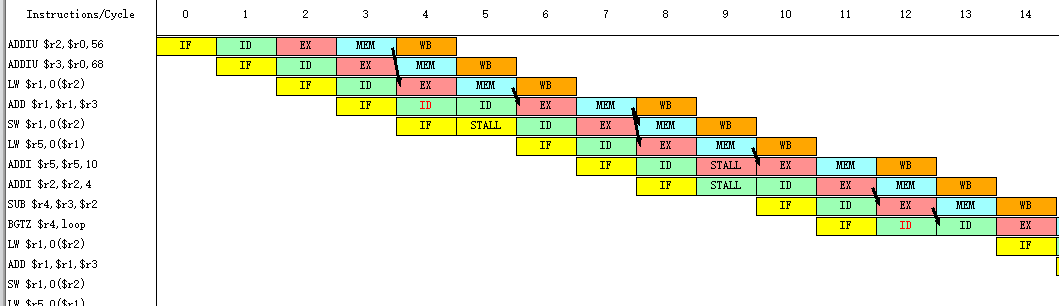
1. 记录数据冲突引起的停顿时钟周期数以及程序执行的总时钟周期数，计算停顿时钟周期数占总执行周期数的百分比。



（6）复位CPU。

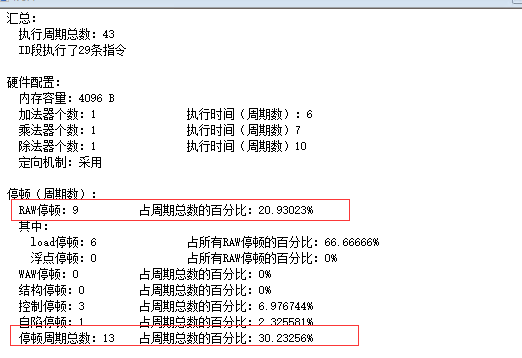
（7）打开定向功能。这是通过在“配置”→“定向”（使该项前面有一个“√”号）来实现的。

（8）用单步执行一周期的方式（F7）执行该程序，同时查看时钟周期图，列出在什么时刻发生了RAW（先写后读）冲突，并与（3）的结果进行比较；



（9）记录数据冲突引起的停顿时钟周期数以及程序执行的总时钟周期数。计算采用定向技术后性能提高的倍数。

停顿（周期数）：9 停顿时钟周期数占总执行周期数的百分比为20.93023%



采用定向技停顿（周期数）：9 停顿时钟周期数占总执行周期数的百分比为20.93023%

术后性能提高的倍数为比原来提高 (65-43)/65=0.34倍

**总结：**

验证了流水线中的冲突和看到IF，ID,EX,MEM,WB这经典的五段流水线具体的执行方式。

验证了并行流水线的执行方式。明白了如何减少结构冲突。