

北京邮电大学



实验二： 流水线及流水线中的冲突

学院： 计算机学院（国家示范性软件学院）

专业： 计算机科学与技术

班级： 2022211305

学号： 2022211683

姓名： 张晨阳

2025 年 4 月 11 号

目录

1. 实验目的	1
2. 实验平台	1
3. 实验内容和步骤	1
4. 实验总结	15

1. 实验目的

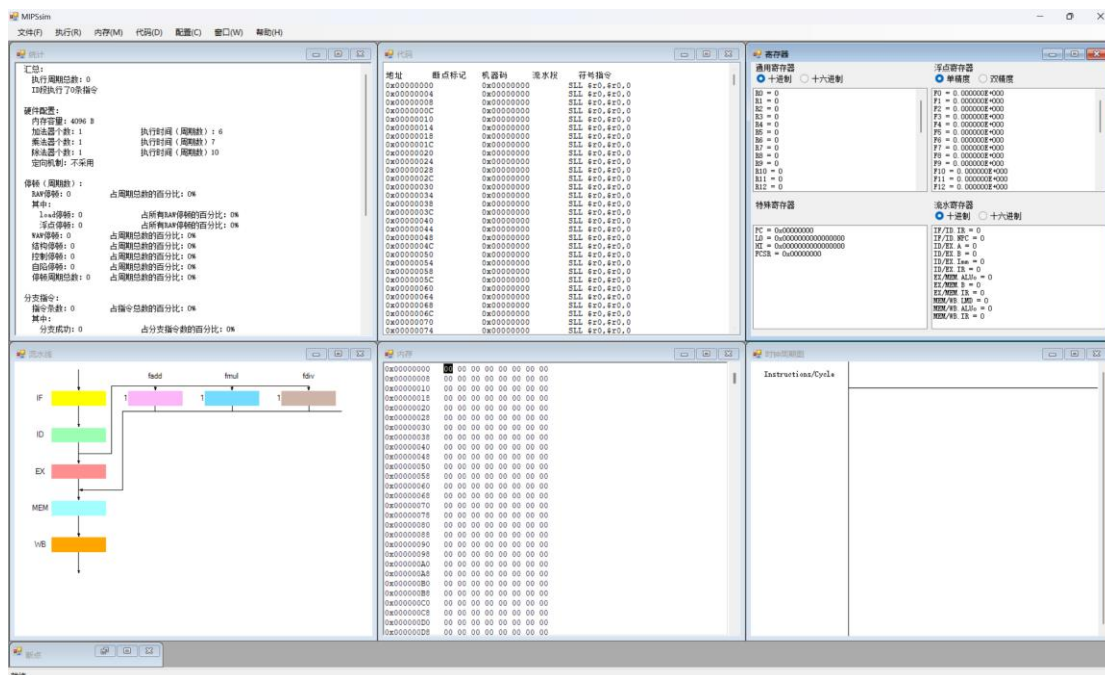
- (1) 加深对计算机流水线基本概念的理解。
- (2) 理解 MIPS 结构如何用 5 段流水线来实现，理解各段的功能和基本操作。
- (3) 加深对数据冲突和资源冲突的理解，理解这两类冲突对 CPU 性能的影响。
- (4) 进一步理解解决数据冲突的方法，掌握如何应用定向技术来减少数据冲突引起的停顿。

2. 实验平台

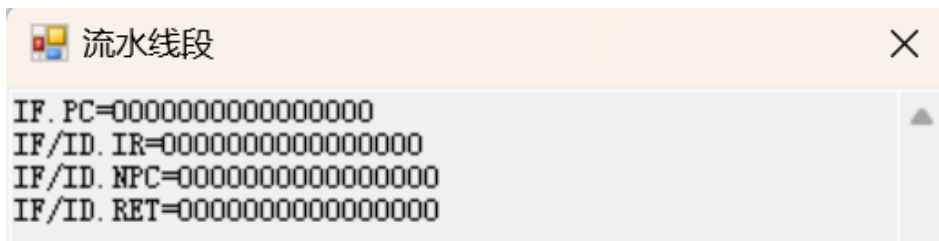
指令级和流水线操作级模拟器 MIPSsim。

3. 实验内容和步骤

- (1) 启动 MIPSsim。

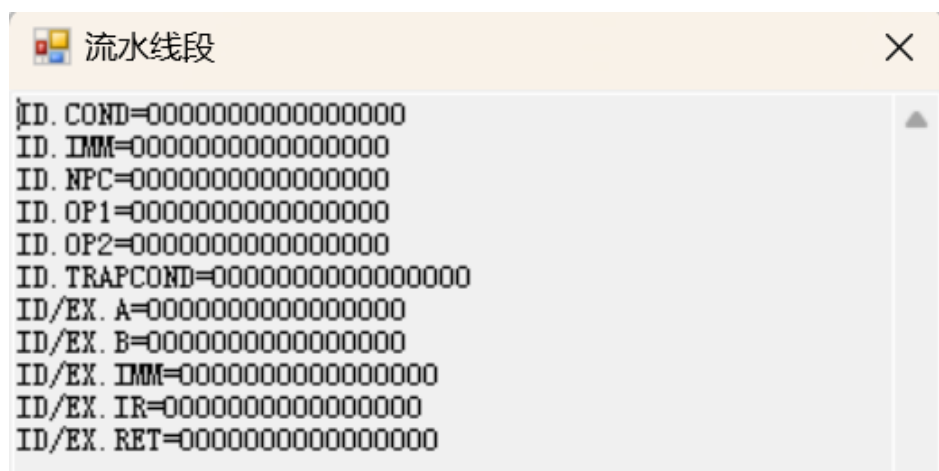


- (2) 进一步理解流水线窗口中各段的功能，掌握各流水寄存器的含义。（鼠标双击各段，即可看到各流水寄存器的内容）



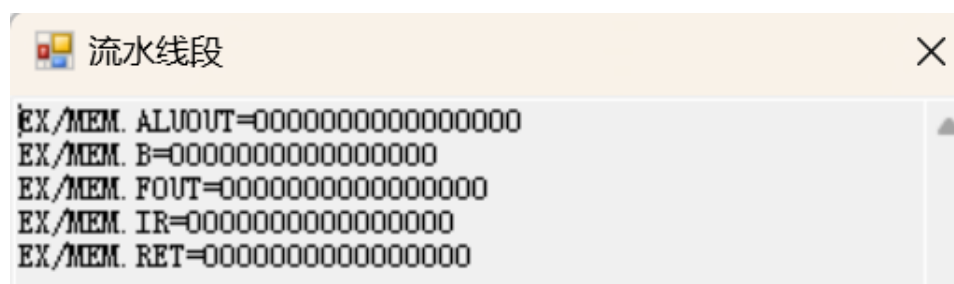
如上图所示，**IF/ID** 段共有三个流水寄存器，分别为 IR、NPC、RET。

其中 IR 为指令寄存器，NPC 为下一程序计数器，RET 为返回地址寄存器。



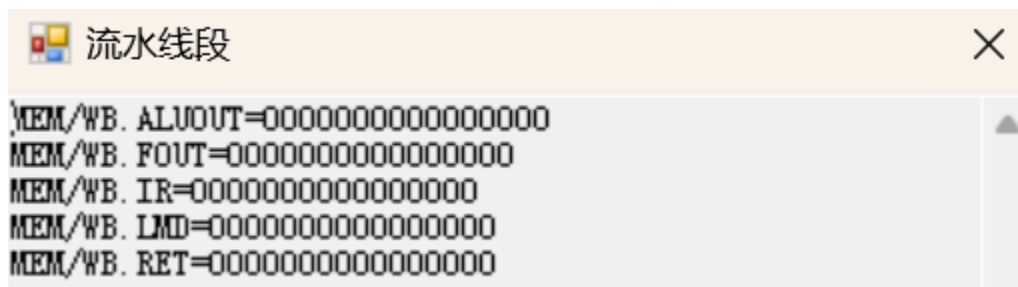
如上图所示，**ID/EX** 段共有五个流水寄存器，分别为 A、B、IMM、IR、RET。

其中 A 为第一操作数寄存器，B 为第二操作数寄存器，IMM 为立即数寄存器，IR 和 RET 的功能与上一段相同。



如上图所示，**EX/MEM** 段共有五个流水寄存器，分别为 ALUOUT、B、FOUT、IR、RET。

其中 ALUOUT 用于存储 ALU 计算结果，FOUT 用于保存浮点运算结果或标志位。



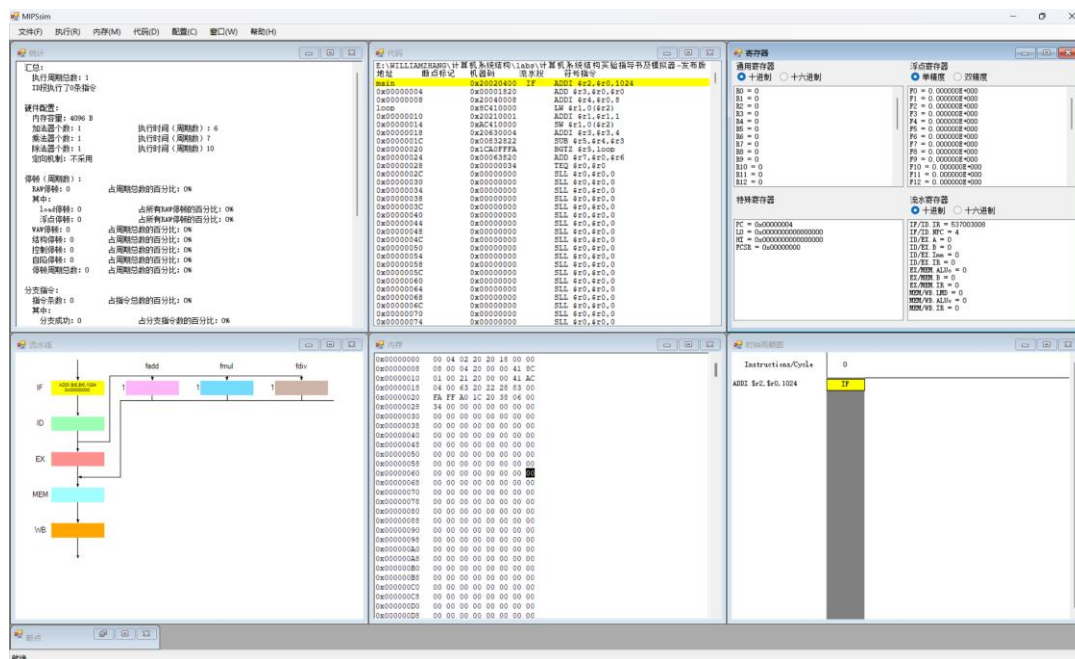
如上图所示，**MEM/WB** 段共有五个流水寄存器，分别为 ALUOUT、FOUT、IR、LMD、RET。

其中 LMD 用于存放从存储器读出的数据。

(3) 载入一个样例程序（在本模拟器所在文件夹下的“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 中寄存器和存储器内容的变化，特别是流水寄存器内容的变化。

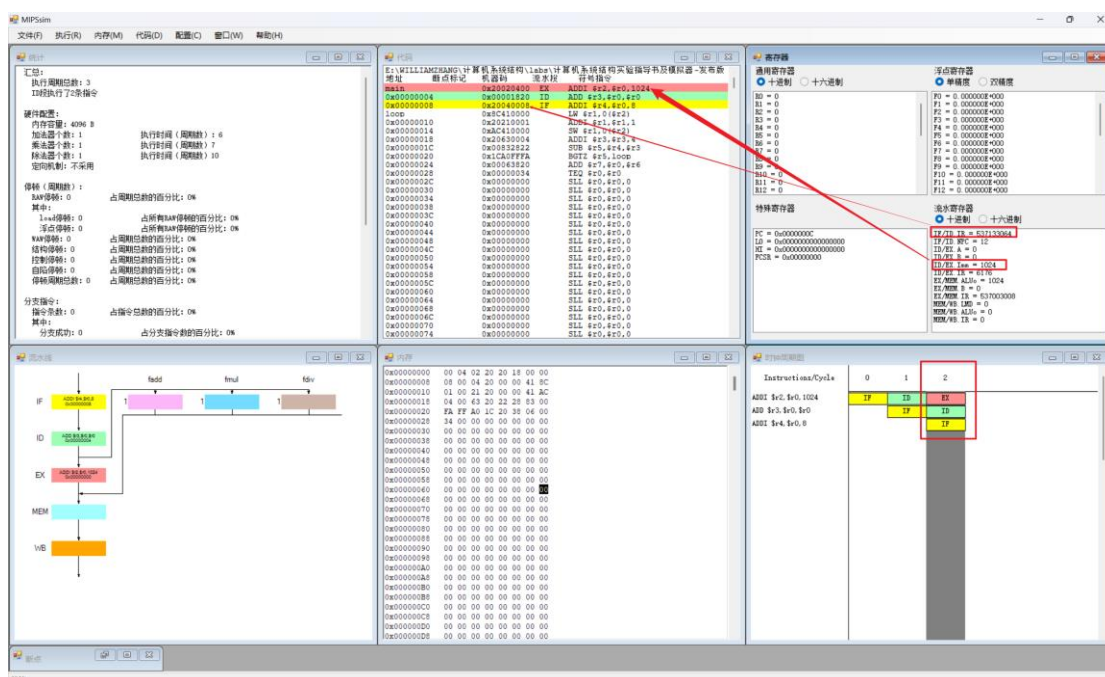
此处选择载入 branch.s:

- 单步执行第一个周期



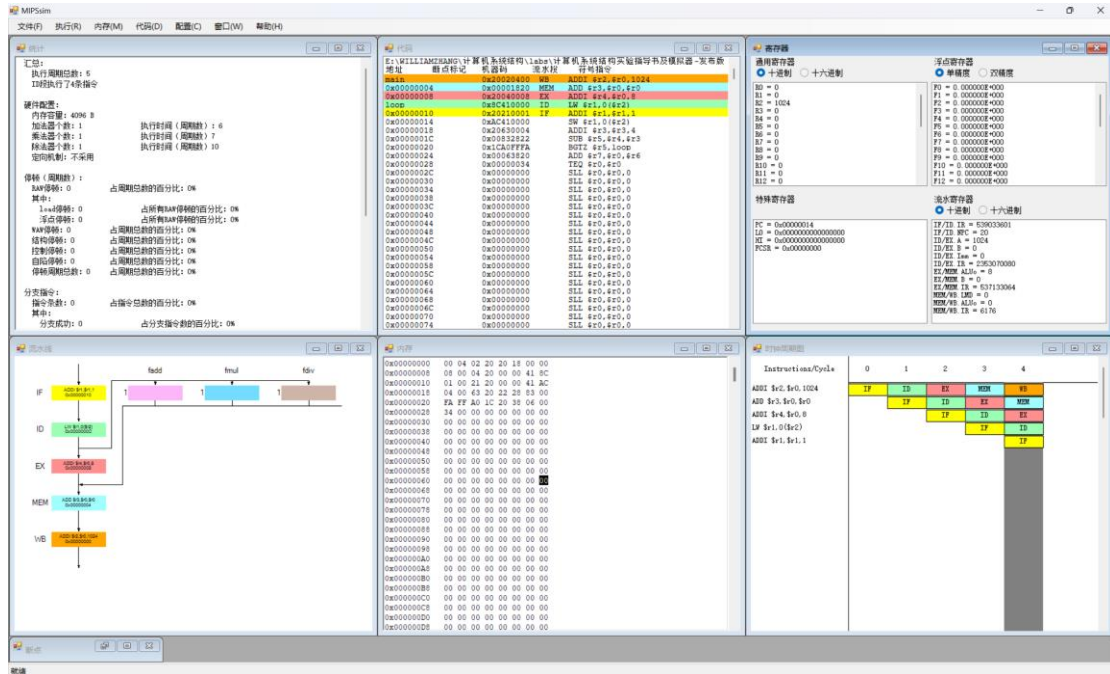
此时读入第一条指令。

- 单步执行第三个周期



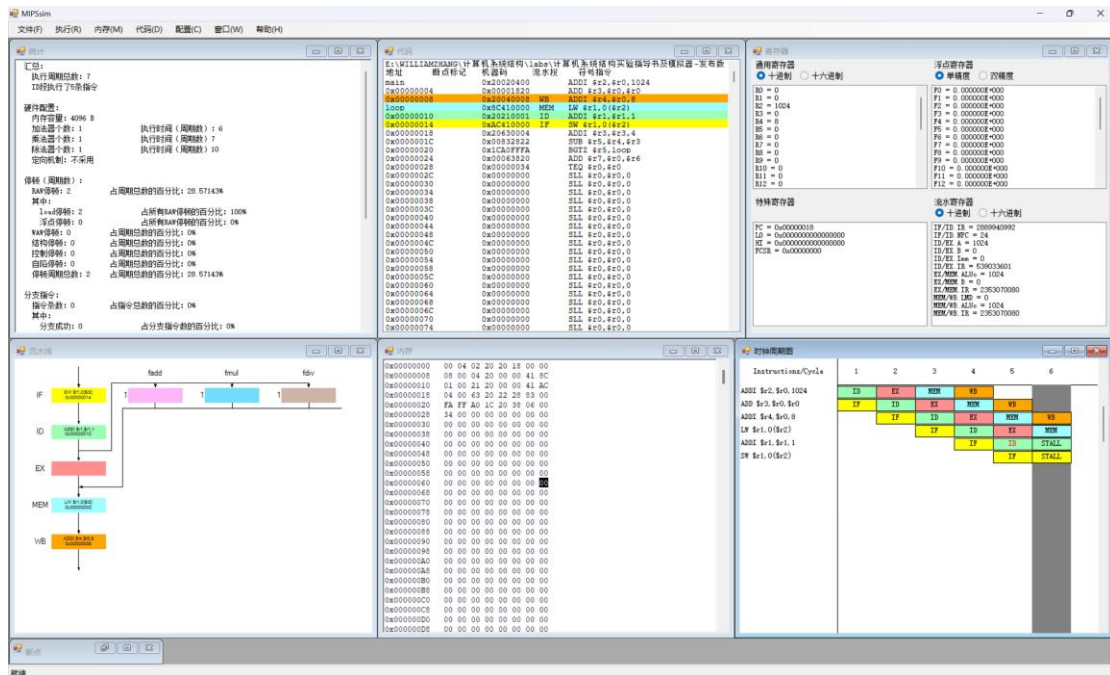
如上图，此时，存入第三条指令，IR 中存放第三条指令的机器码；
此时第一条指令是 EX 阶段，立即数是 1024，故 Imm 中存入的是 1024。

• 单步执行第五个周期



此时将计算结果写回至存储器中。

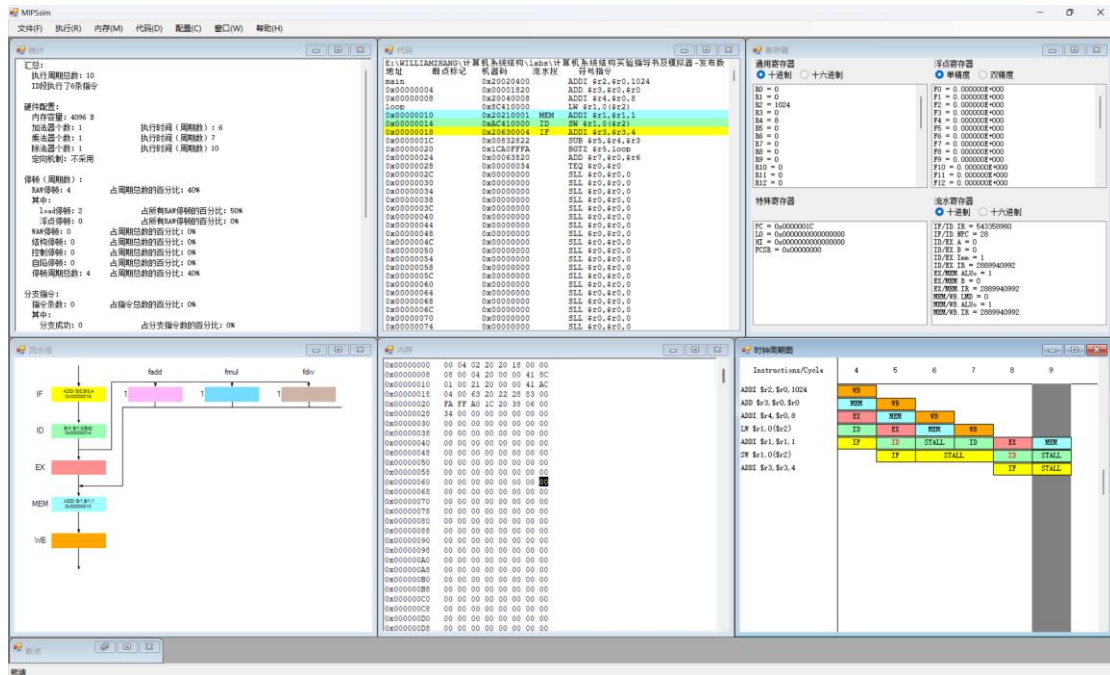
• 单步执行第七个周期



如图所示：由于第四条指令 `LW $r1, 0($r2)` 仍未完成 WB 阶段，所以 `ADDI $r1, $r1, 1` 与 `SW $r1, 0($r2)` 指令需要等待该指令 WB 阶段完成才能够进行读取并

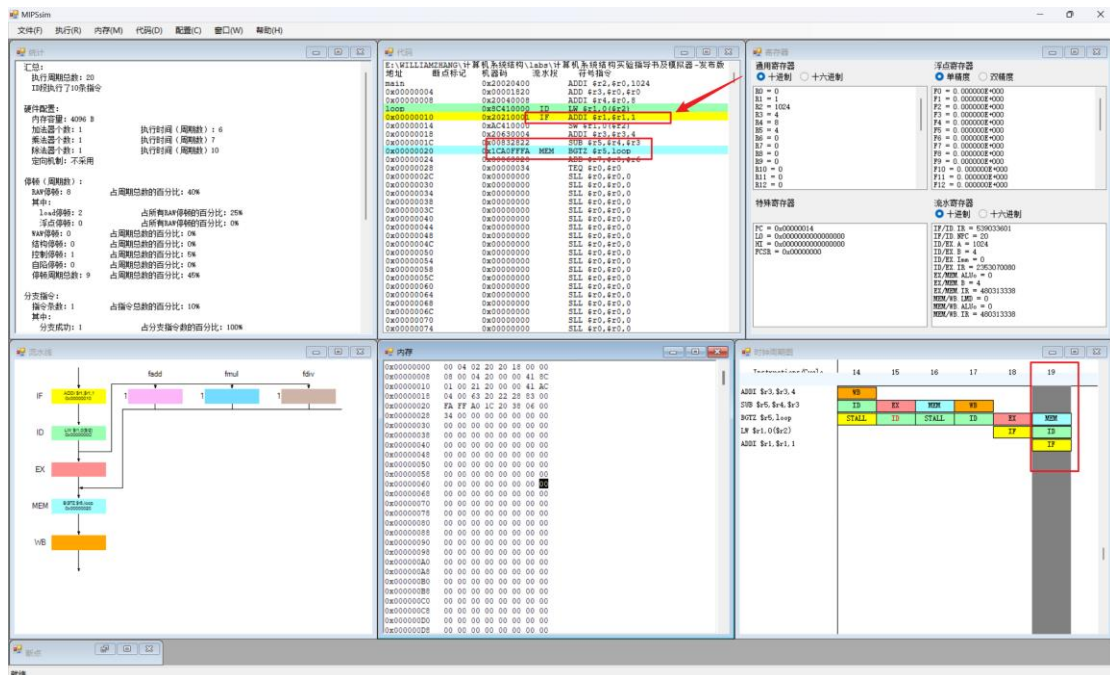
执行。

- 单步执行第十个周期



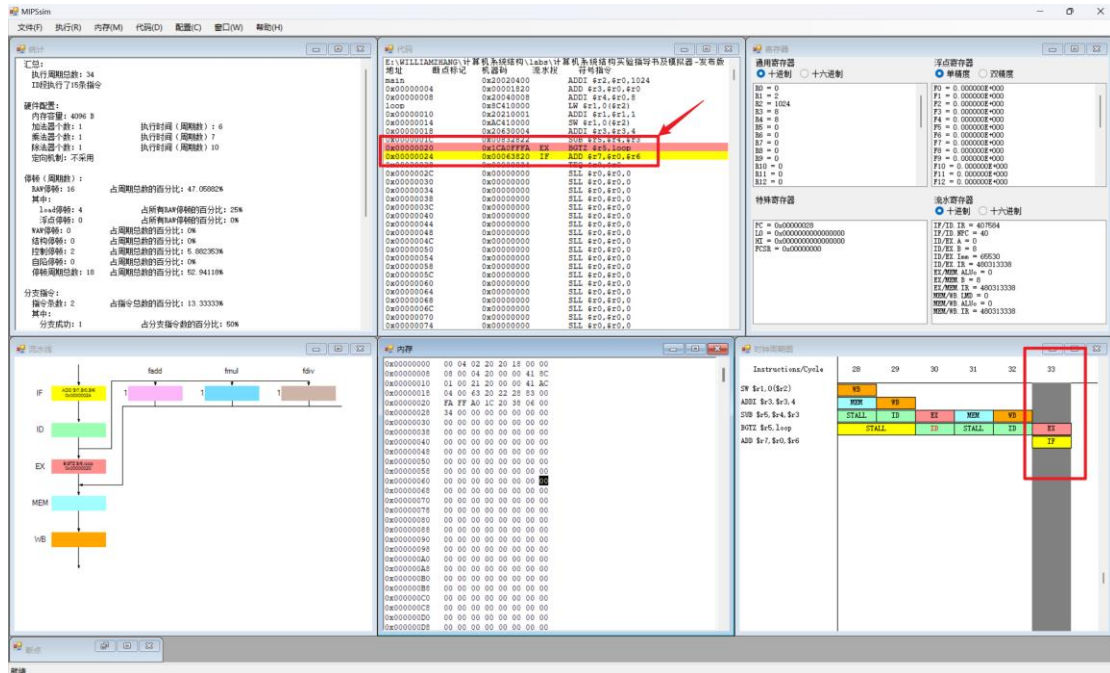
如上图所示：指令 `SW $r1, 0($r2)` 需要使用修改后的 `r1`，而 `ADDI` 指令还没有完成 `WB` 阶段，所以此时处于 `STALL` 状态；同时 `SW` 指令处于 `ID` 阶段，故下一个 `ADDI` 指令也处于 `STALL` 状态，无法从 `IF` 移动到 `ID` 阶段。

- 执行多个周期至单步执行第二十个周期



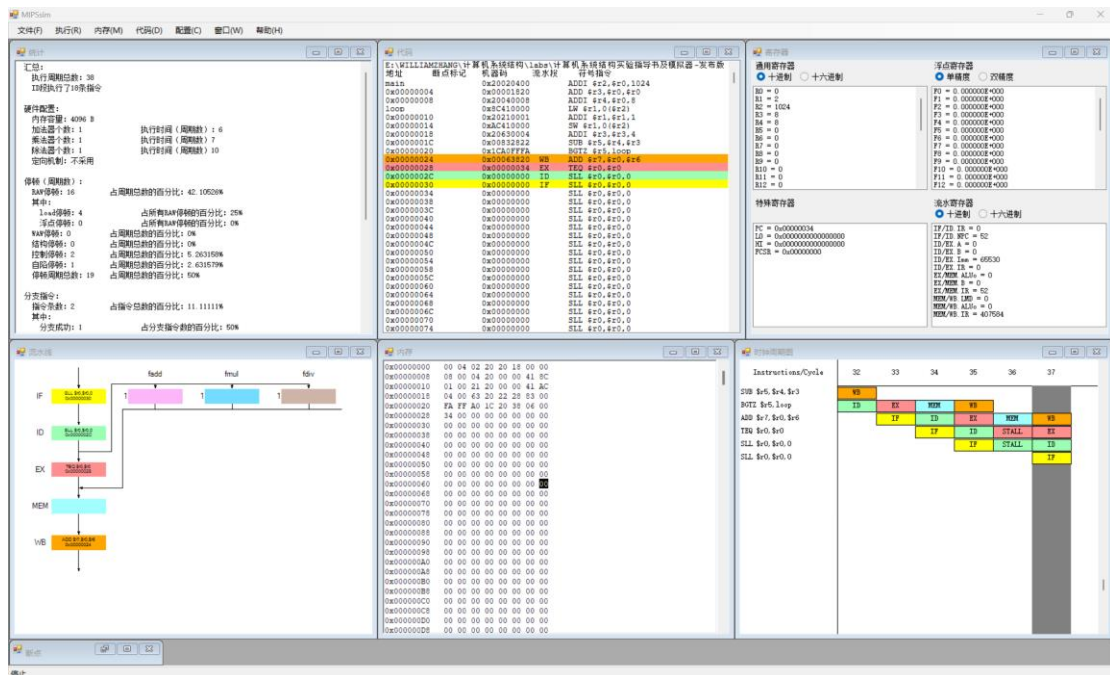
此时循环的结束的条件并未达到，所以跳转至 `loop` 指向的位置执行循环。

- 执行多个周期至第三十四个周期



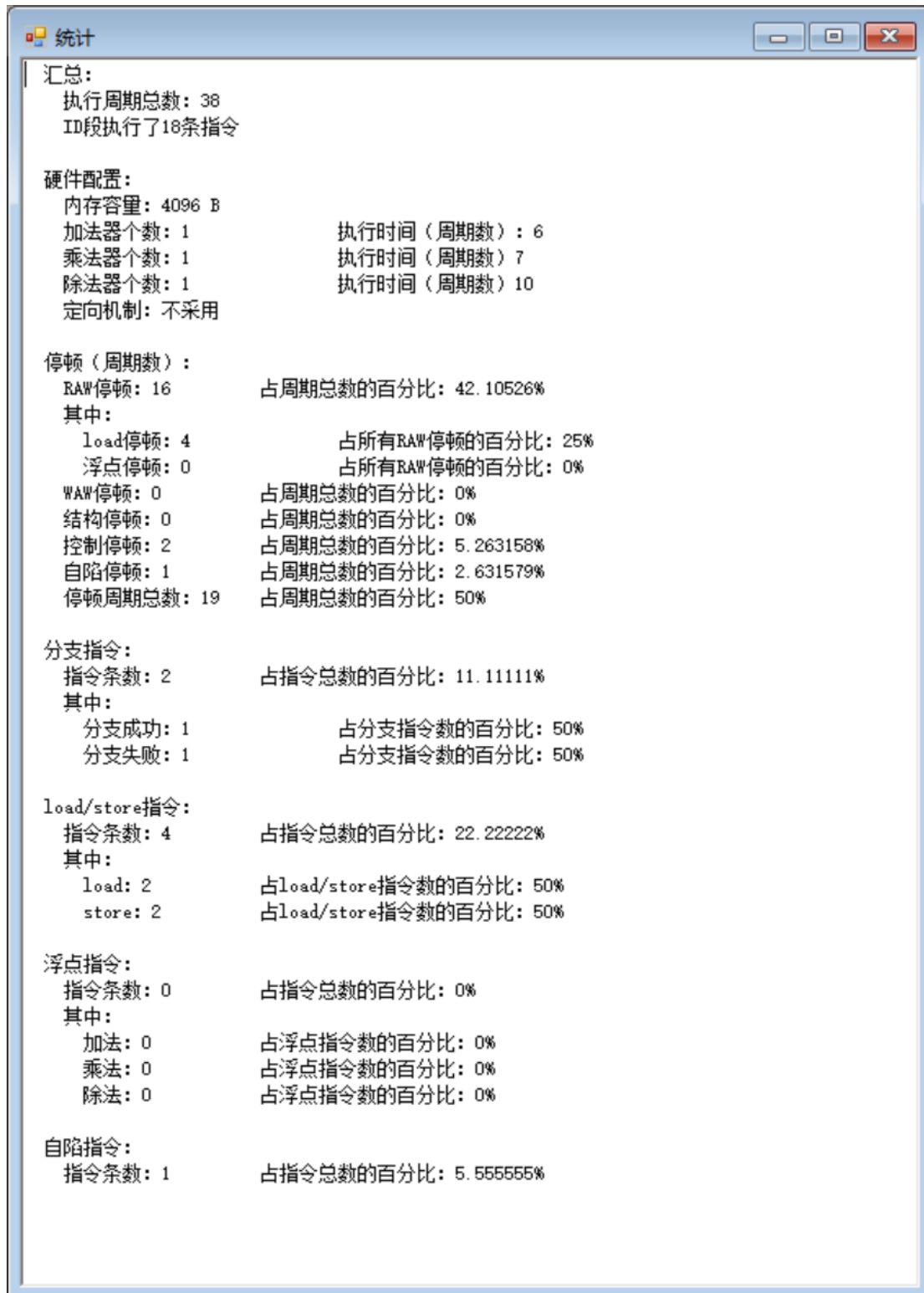
此时 r5 中的值大于 0，故循环结束。

- 执行多个周期至第三十八个周期

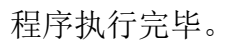
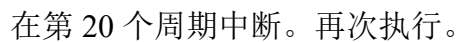


程序执行完毕。

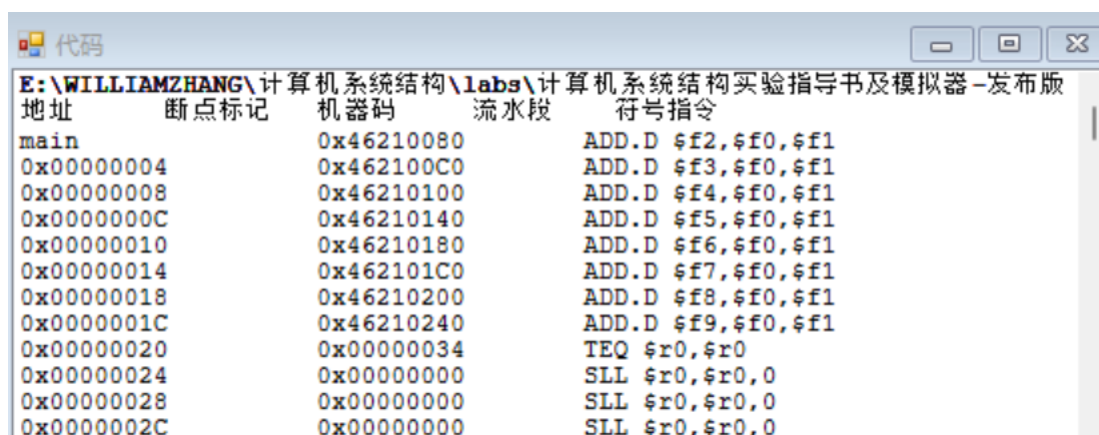
- 连续执行



如上图所示, 共发生了 16 次 RAW 停顿, 其中 4 次是 load 停顿, 大多数停顿的发生是由于数据相关。



- (4) 选择配置菜单中的“流水方式”选项，使模拟器工作于流水方式下。
- (5) 观察程序在流水方式下的执行情况。
- (6) 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。
 - 3) 记录由结构冲突引起的停顿周期数，计算停顿周期数占总执行周期数的百分比。
 - 4) 把浮点加法器的个数改为 4 个。
 - 5) 再重复 1-3 的步骤。
 - 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。



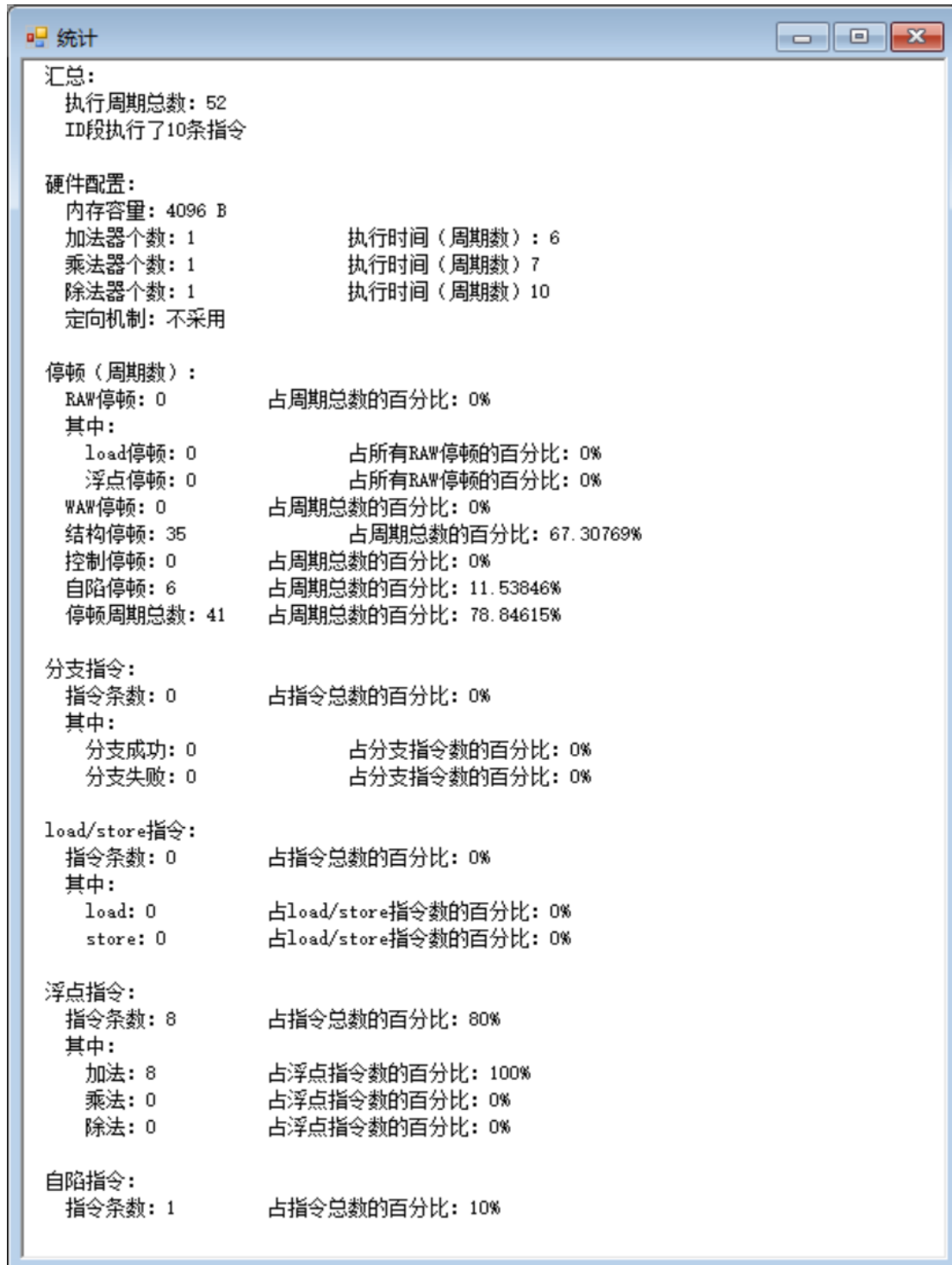
The screenshot shows a code editor window titled "代码" (Code). The file path is `E:\WILLIAMZHANG\计算机系统结构\labs\计算机系统结构实验指导书及模拟器-发布版`. The code is assembly for a program named `main`. It lists addresses, branch markers, machine codes, pipeline stages, and symbolic instructions.

地址	断点标记	机器码	流水段	符号指令
<code>main</code>		<code>0x46210080</code>		<code>ADD.D \$f2,\$f0,\$f1</code>
<code>0x00000004</code>		<code>0x462100C0</code>		<code>ADD.D \$f3,\$f0,\$f1</code>
<code>0x00000008</code>		<code>0x46210100</code>		<code>ADD.D \$f4,\$f0,\$f1</code>
<code>0x0000000C</code>		<code>0x46210140</code>		<code>ADD.D \$f5,\$f0,\$f1</code>
<code>0x00000010</code>		<code>0x46210180</code>		<code>ADD.D \$f6,\$f0,\$f1</code>
<code>0x00000014</code>		<code>0x462101C0</code>		<code>ADD.D \$f7,\$f0,\$f1</code>
<code>0x00000018</code>		<code>0x46210200</code>		<code>ADD.D \$f8,\$f0,\$f1</code>
<code>0x0000001C</code>		<code>0x46210240</code>		<code>ADD.D \$f9,\$f0,\$f1</code>
<code>0x00000020</code>		<code>0x00000034</code>		<code>TEQ \$r0,\$r0</code>
<code>0x00000024</code>		<code>0x00000000</code>		<code>SLL \$r0,\$r0,0</code>
<code>0x00000028</code>		<code>0x00000000</code>		<code>SLL \$r0,\$r0,0</code>
<code>0x0000002C</code>		<code>0x00000000</code>		<code>SLL \$r0,\$r0,0</code>

由此处可以发现：存在冲突的部件为加法部件。

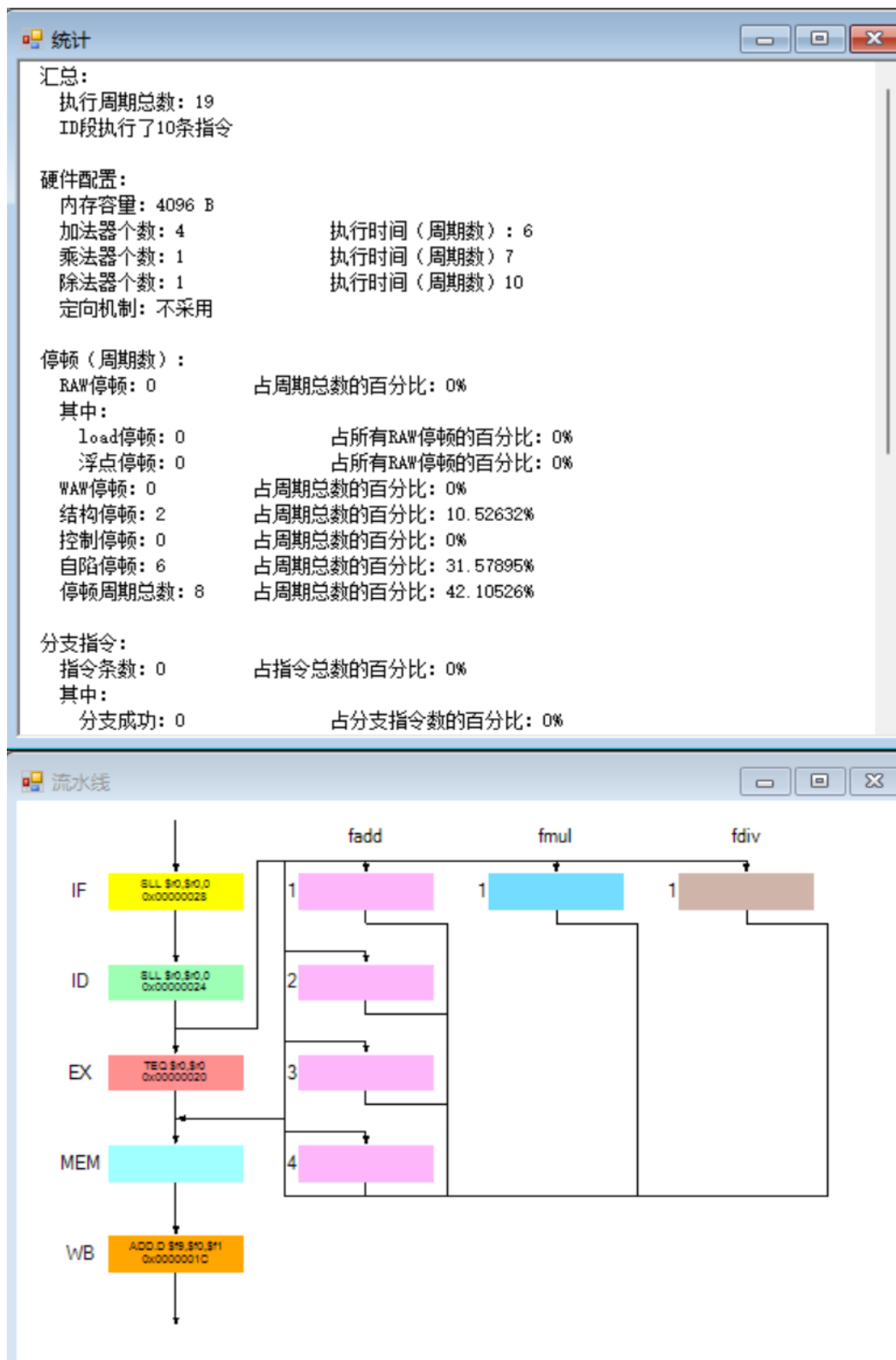
且任何两条加法指令之间都存在冲突。

执行程序结果如下：



如上所示, 总执行周期为 52, 结构冲突导致的停顿周期数为 35, 约占总执行周期数的 67.3%。

我们将浮点加法器的数量改为 4 个, 重复上述步骤, 得到如下结果:



总执行周期为 19，结构冲突导致的停顿周期数下降为 2，约占总执行周期数的 10.5%。

(7) 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（在“配置”菜单下选择取消“定向”）。
- 4) 用单步执行一个周期的方式执行该程序，观察时钟周期图，列出什么时刻发生了 RAW 冲突：4, 6, 7, 9, 10, 13, 14, 17, 18, 20, 21, 25, 26, 28, 29, 32, 33, 36, 37, 39, 40, 44, 45, 47, 48, 51, 52, 55, 56, 58, 59
- 5) 记录数据冲突引起的停顿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。

汇总：

执行周期总数：65
ID段执行了29条指令

硬件配置：

内存容量：4096 B	
加法器个数：1	执行时间（周期数）：6
乘法器个数：1	执行时间（周期数）7
除法器个数：1	执行时间（周期数）10
定向机制：不采用	

停顿（周期数）：

RAW停顿：31	占周期总数的百分比：47.69231%
其中：	
load停顿：12	占有所有RAW停顿的百分比：38.70968%
浮点停顿：0	占有所有RAW停顿的百分比：0%
WAW停顿：0	占周期总数的百分比：0%
结构停顿：0	占周期总数的百分比：0%
控制停顿：3	占周期总数的百分比：4.615385%
自陷停顿：1	占周期总数的百分比：1.538462%
停顿周期总数：35	占周期总数的百分比：53.84615%

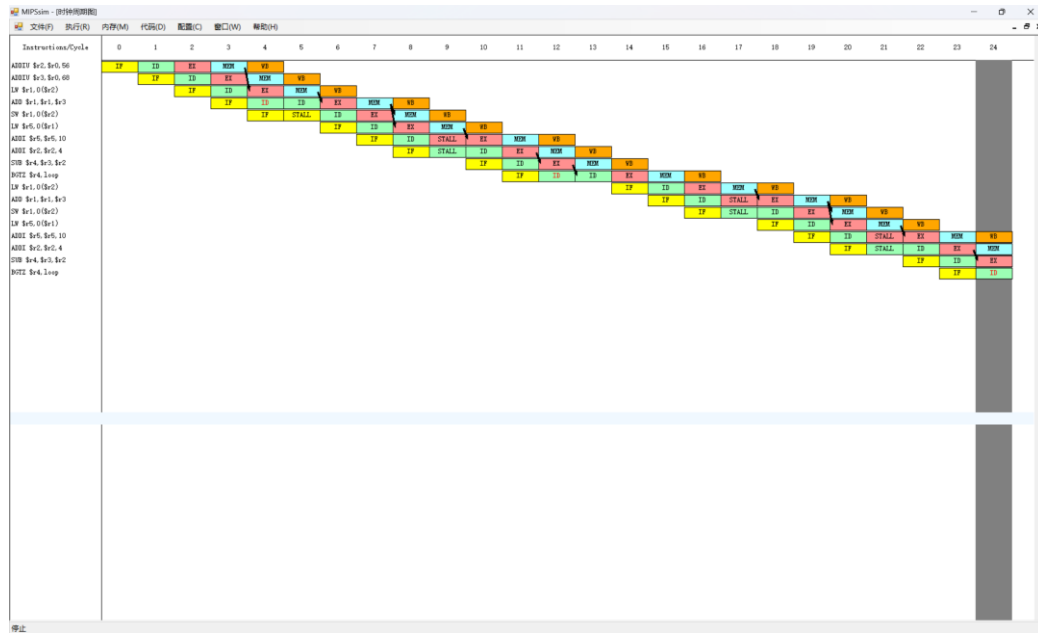
分支指令：

指令条数：3	占指令总数的百分比：10.34483%
其中：	
分支成功：2	占分支指令数的百分比：66.66666%

总周期数 65 个，由于数据冲突引起的停顿周期数 31 个，占比 47.7%

- 6) 复位 CPU。
- 7) 打开定向功能。
- 8) 用单步执行一个周期的方式执行该程序，查看时钟周期图，列出什么时刻发生了 RAW 冲突，并与步骤 3) 的结果比较：5, 10, 13, 18, 22, 25, 30, 34, 37

时钟周期图为:



- 9) 记录数据冲突引起的停顿周期数以及程序执行的总周期数。计算采用定向以后性能比原来提高多少。

汇总:

执行周期总数: 43

ID段执行了29条指令

硬件配置：

内存容量: 4096 B

加法器个数: 1

乘法器个数: 1

除法器个数: 1

定向机制：采用

执行时间(周期数): 6

执行时间(周期数) 7

执行时间(周期数) 10

停顿（周期数）：

RAW 停顿: 9

占周期总数的百分比: 20.93023%

其中：

load停顿: 6

占有所有RAW停顿的百分比: 66.66666%

浮点停顿: 0

占所有RAW停顿的百分比: 0%

WAW停顿: 0

占周期总数的百分比: 0%

结构停顿: 0

占周期总数的百分比: 0%

控制停顿: 3

占周期总数的百分比: 6.976744%

自陷停顿: 1

占周期总数的百分比: 2.325581%

停顿周期总类

占周期总数的百分比: 30.23256%

[illegible][illegible]

分支指令：

指令条数: 3

占指令总数的百分比: 10.34483%

其中：

分支成功: 2

占分支指令数的百分比: 66.66666%

总周期数 43 个，由数据冲突引起的停顿周期数 9 个，占比 20.9%。

采用定向技术后，性能提高了 $65 \div 43 = 1.51$ 倍。

4. 实验总结

通过本次实验，我深入理解了流水线技术的工作原理及冲突处理方法，在 MIPSsim 模拟器中观察了 5 段流水线的执行过程，发现结构冲突会因资源竞争导致显著性能下降，而数据冲突则可通过定向技术有效缓解。