《计算机系统结构》

指令调度与延迟分支

学院：计算机学院（国家示范性软件学院）

班级：2018211314

姓名：李志毅

学号：2018211582

实验五 指令调度与延迟分支

一、实验目的

（1）加深对指令调度技术的理解。

（2）加深对延迟分支技术的理解。

（3）熟练账务用指令调度技术解决流水线中的数据冲突的方法。

（4）进一步理解指令调度技术对 CPU 性能的改进。

（5） 进一步理解延迟分支技术对 CPU 性能的改进。

二、实验环境

指令级和和流水线操作级模拟器 MIPSsim

三、实验步骤

1. 启动 MIPSsim

2. 根据实验 2 的相关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，掌握各流水线寄存器的含义

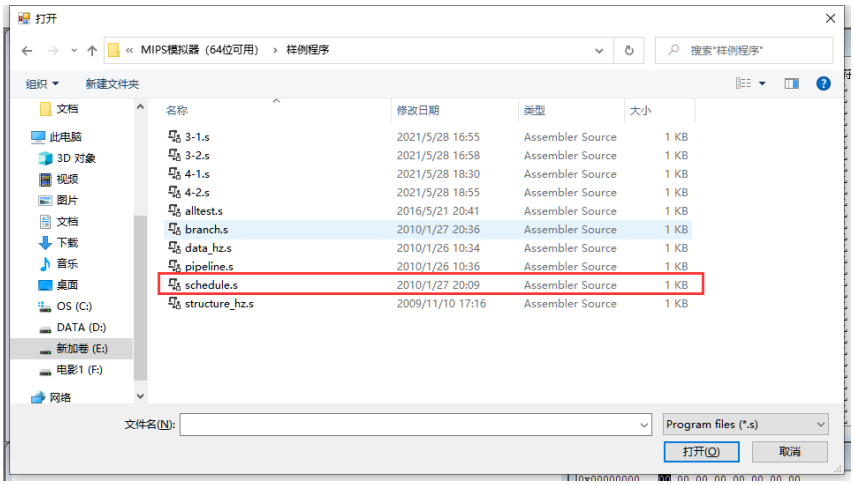
实验 2 中已经很详细的分析了每个寄存器的内容，这里不再赘述。

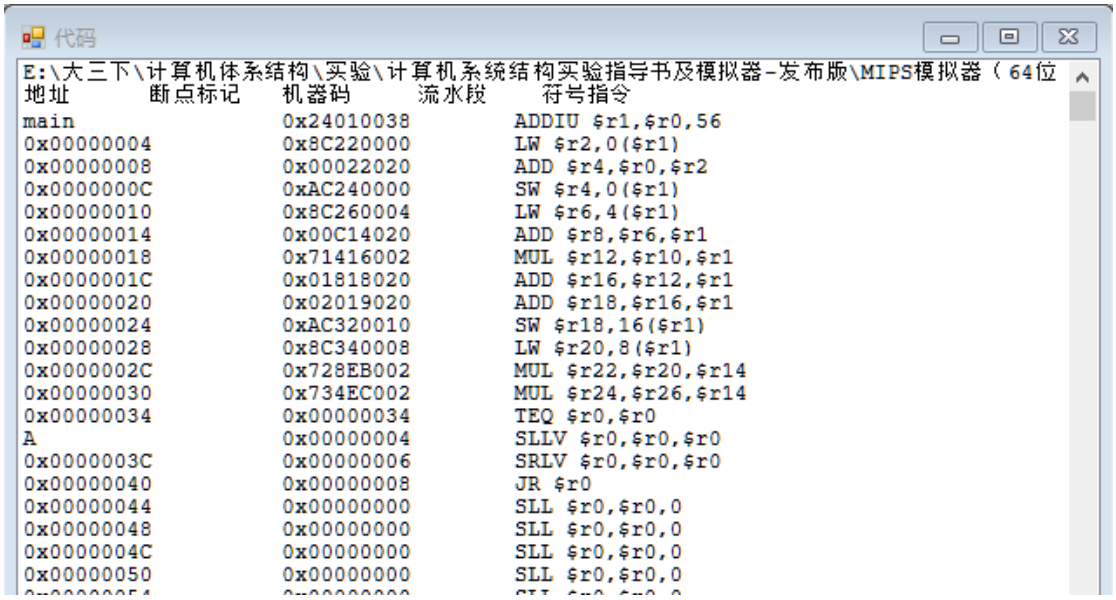
3. 选择“配置”→“流水方式”选项，使模拟器工作在流水方式下

4. 用指定调度技术解决流水线中的结构冲突与数据冲突

4.1 启动 MIPSsim

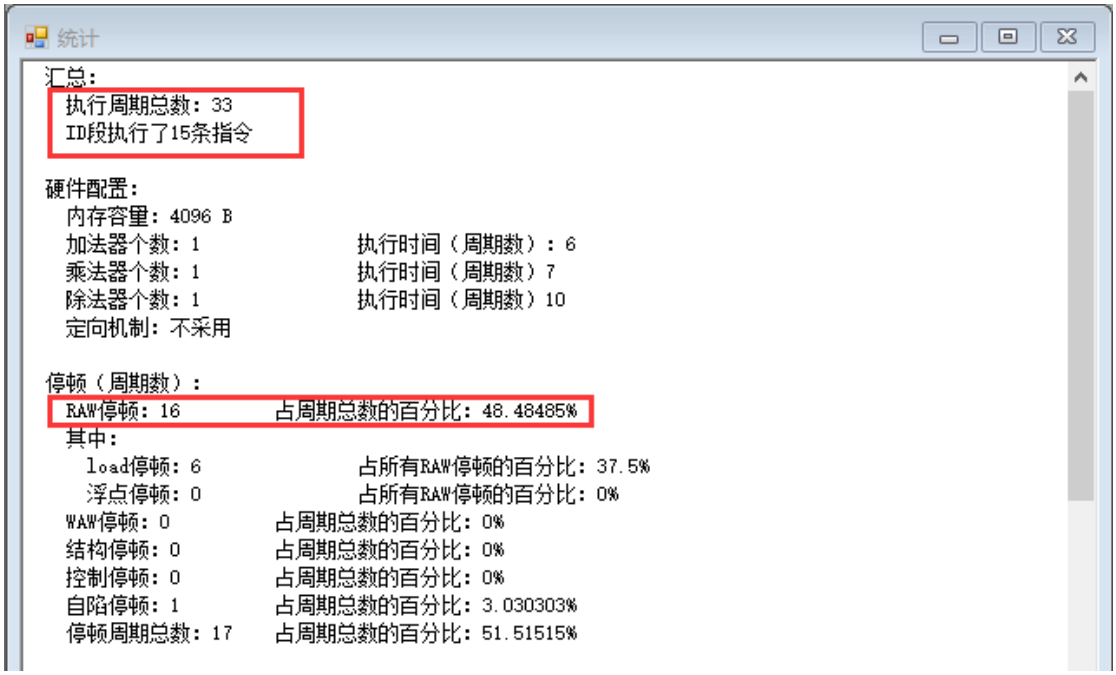
4.2 载入 schedule.s 程序





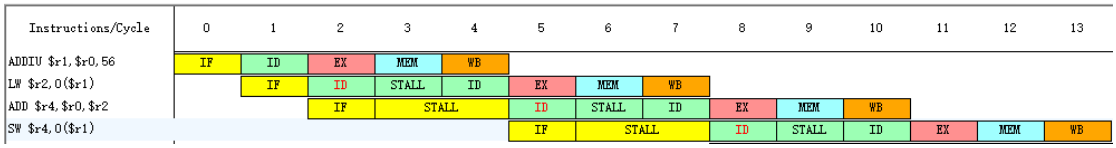
4.3. 关闭定向功能

4.4 执行所载入的程序，通过查看统计数据和时钟周期图，找出并记录程序执行过程中各种冲突发生的次数，发生冲突的指令组合以及程序执行的总时钟周期数

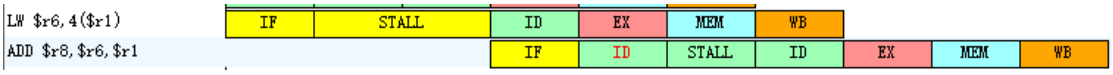


通过统计可以看出，执行周期总数为 33，其中停顿周期总数为 17，RAW 停顿为 16 次，自陷停顿为 1 次。

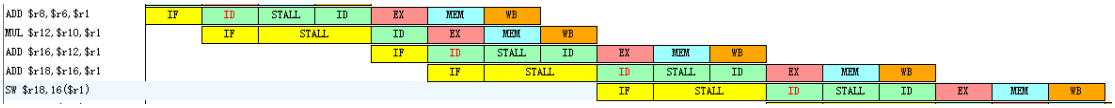
通过观察流水线时钟周期图，可以查找了冲突的指令组合如下：



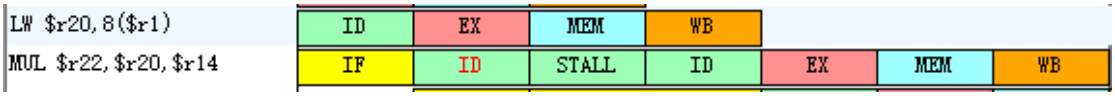
ADDIU $r1,$r0,56	
LW $r2,0($r1)	与上一指令存在 RAW 冲突
ADD $r4,$r0,$r2	与上一指令存在 RAW 冲突
SW $r4,0($r1)	与上一指令存在 RAW 冲突



LW $r6,4($r1)	
ADD $r8,$r6,$r1	与上一指令存在 RAW 冲突



MUL $r12,$r10,$r1	
ADD $r16,$r12,$r1	与上一指令存在 RAW 冲突
ADD $r18,$r16,$r1	与上一指令存在 RAW 冲突
SW $r18,16($r1)	与上一指令存在 RAW 冲突



LW $r20,8($r1)	
MUL $r22,$r20,$r14	与上一指令存在 RAW 冲突

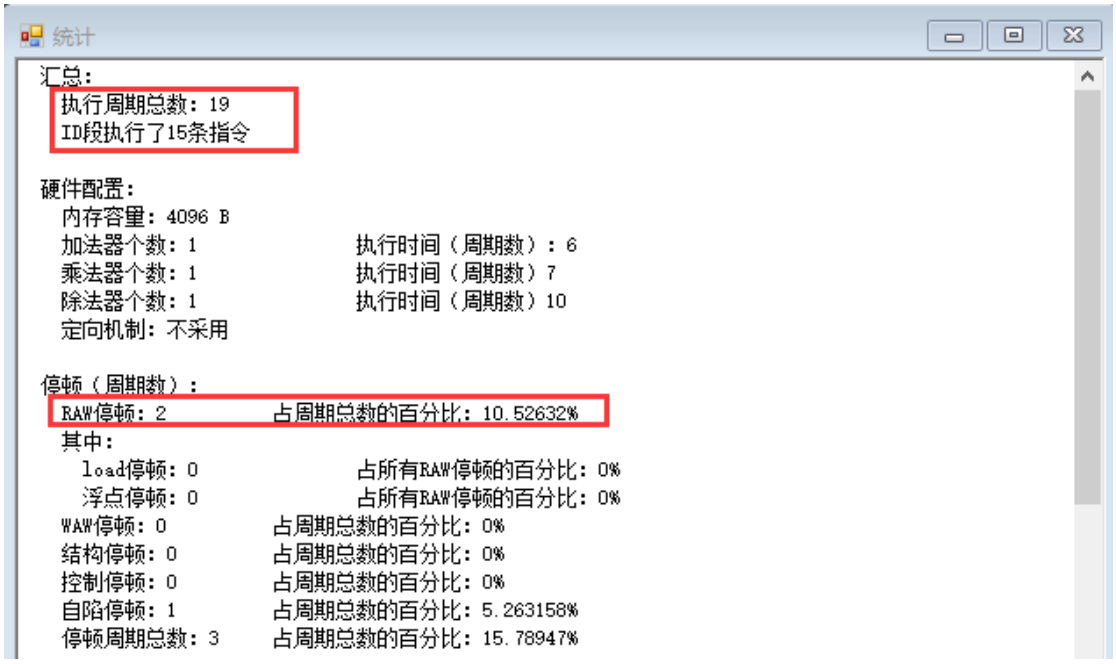
4.5 自己采用调度技术对程序进行指令调度，消除冲突（自己修改源程序）。将调度（修改）后的程序重新命名为 afer-schedule.s。（注意：调度方法灵活多样，在保证程序正确性的前提下自己随意调度，尽量减少冲突即可，不要求要达到最优。）

根据冲突，可以将代码静态优化为如下的结构，即 after-schedules 为：

```
    .text
main:
    ADDIU  $r1,$r0,A
    MUL    $r24,$r26,$r14
    LW     $r2,0($r1)
    LW     $r20,8($r1)
    MUL    $r12,$r10,$r1
```

```
ADD    $r4,$r0,$r2

LW     $r6,4($r1)

ADD    $r16,$r12,$r1

MUL    $r22,$r20,$r14

SW     $r4,0($r1)

ADD    $r18,$r16,$r1

ADD    $r8,$r6,$r1

SW     $r18,16($r1)

TEQ $r0,$r0



.data

A:

.word 4,6,8
```

4.6 载入 afer-schedule.s，执行该程序，观察程序在流水线中的执行情况，记录程序执行的总时钟周期数



可以看到总时钟周期数减少为 19，RAW 停顿两次，总停顿 3 次占周期总数

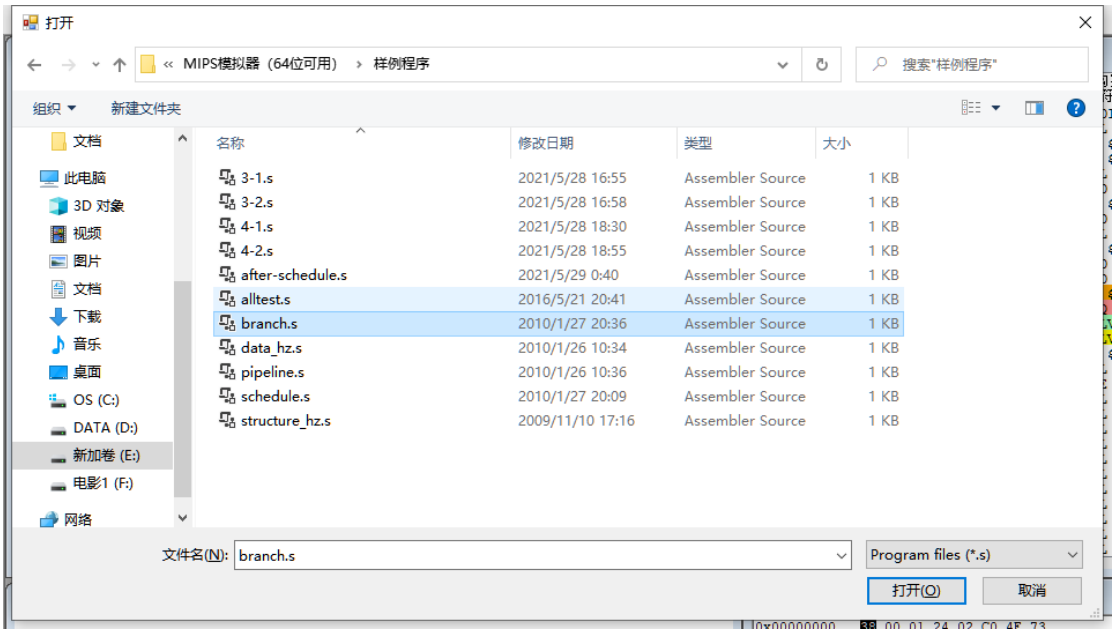
的百分比为 15.78947%

4.7 比较调度前和调度后的性能，论述指令调度对提高 CPU 性能的作用

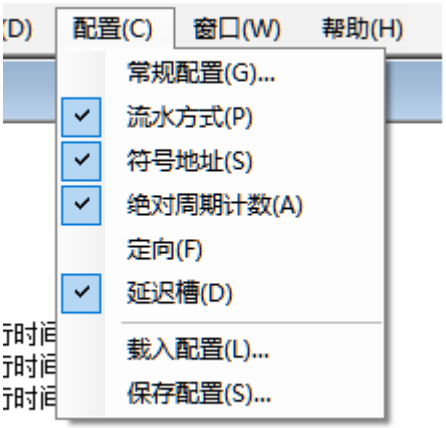
调度前和调度后的性能对比为，指定调度后时钟总周期数从 33 降到了 19，指定调度使指令顺序重新组合，减少了部分的数据冲突，使得 RAW 冲突的数目减少到了 2，通过指令调度技术显著地提高了 CPU 的使用率，大大减少了指令冲突的次数。

5. 用延迟分支技术减少分支指令对性能的影响

5.1 在 MIPSsim 中载入 branch.s 样例程序（在本模拟器目录的“样例程序”文件夹中

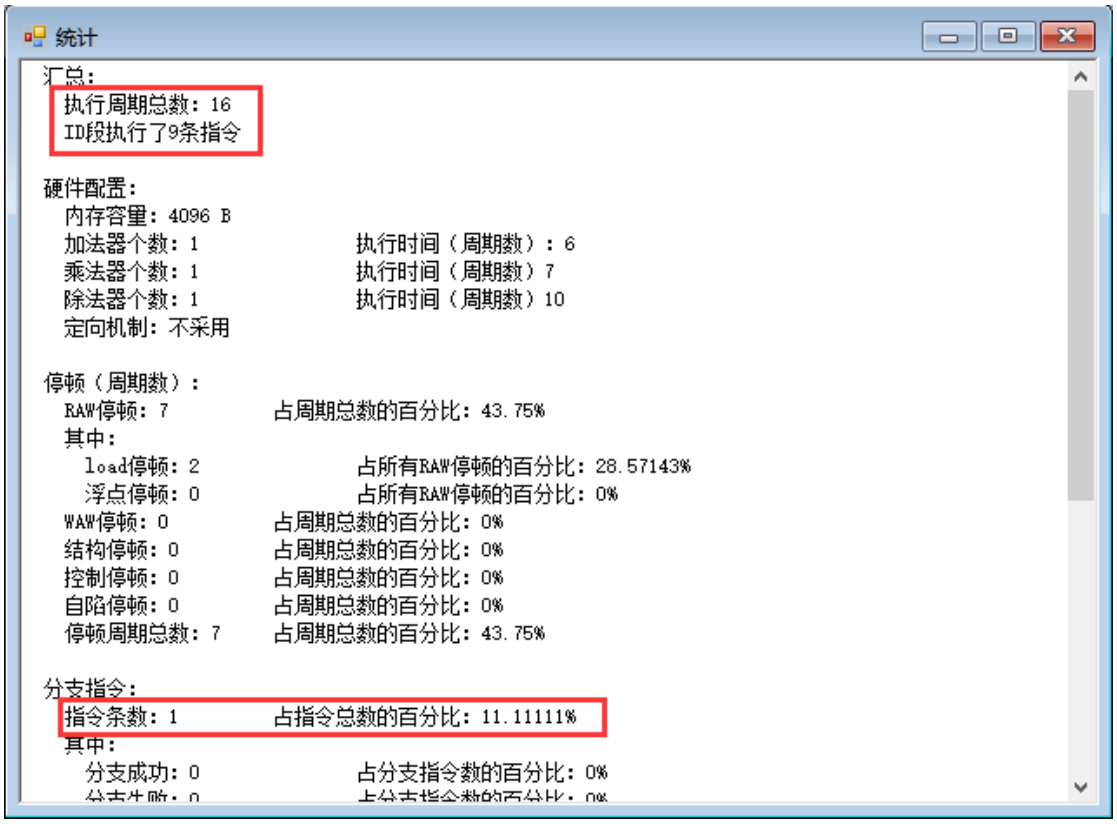


5.2 关闭延迟分支功能。这是通过在“配置”->“延迟槽”选项来实现的

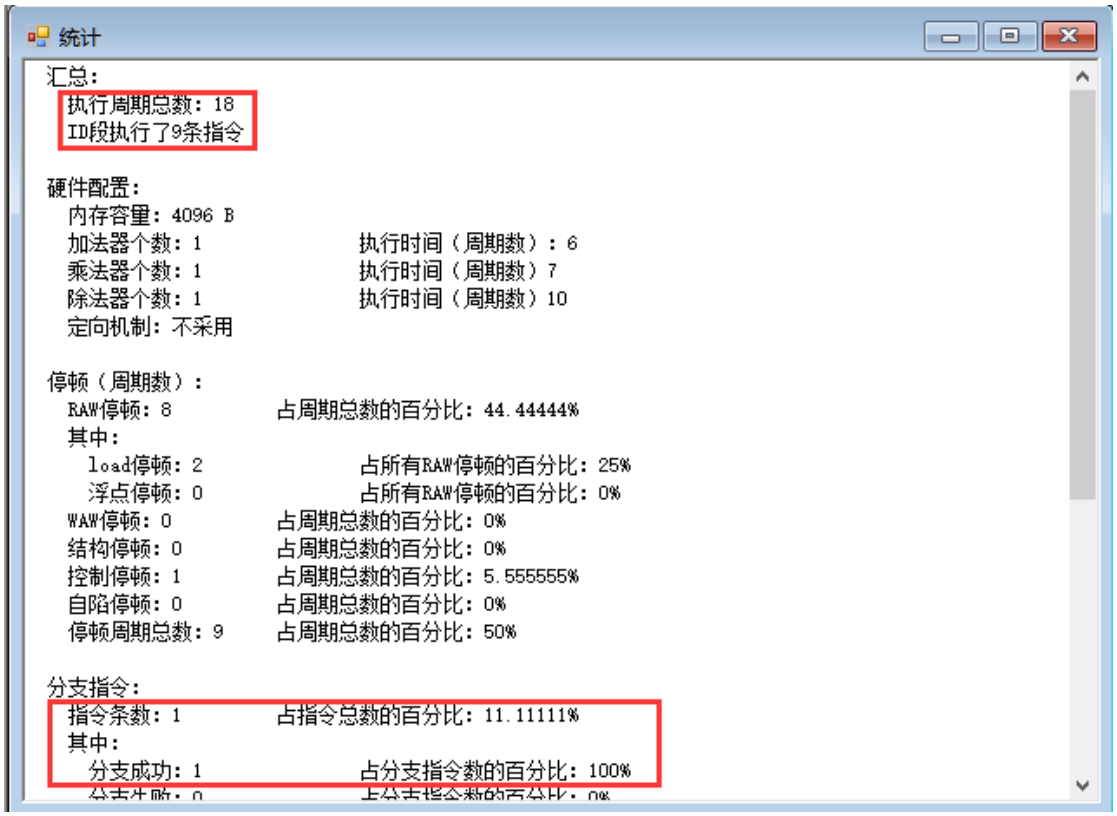


5.3 执行该程序，观察并记录发生分支延迟的时刻，记录该程序执行的总时钟周期数

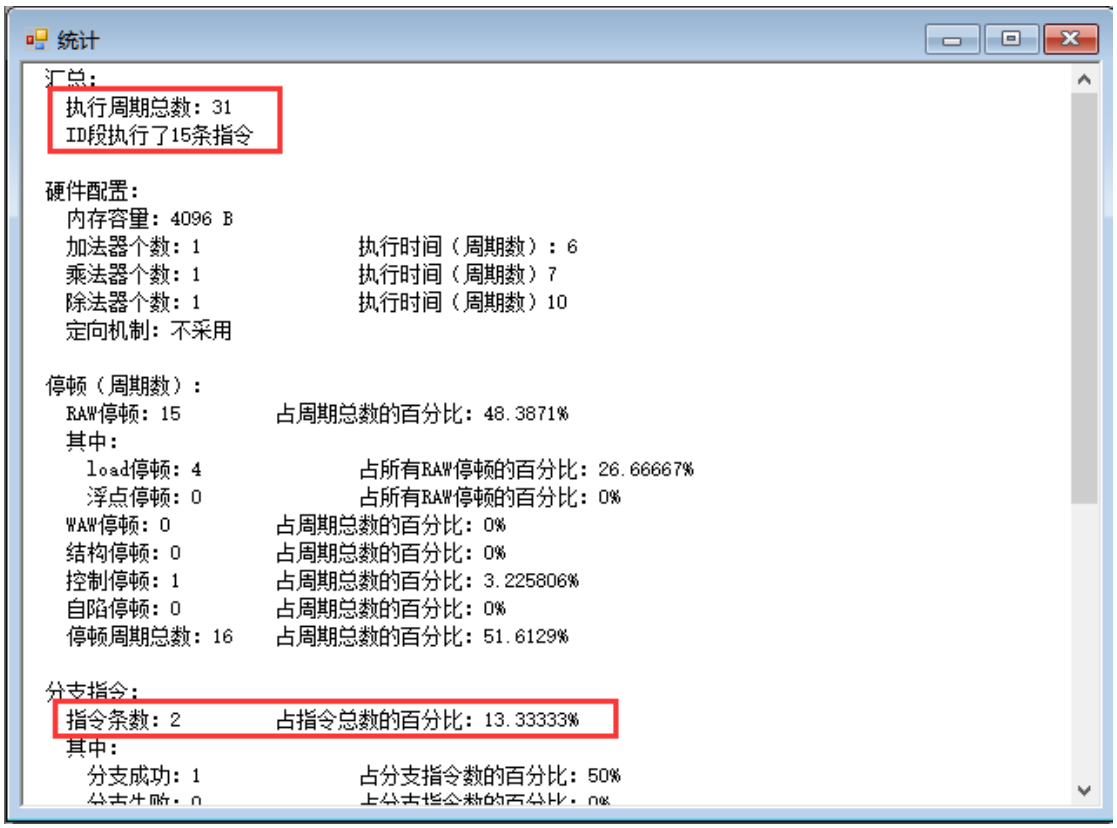
- 第 16 周期开始第一次分支



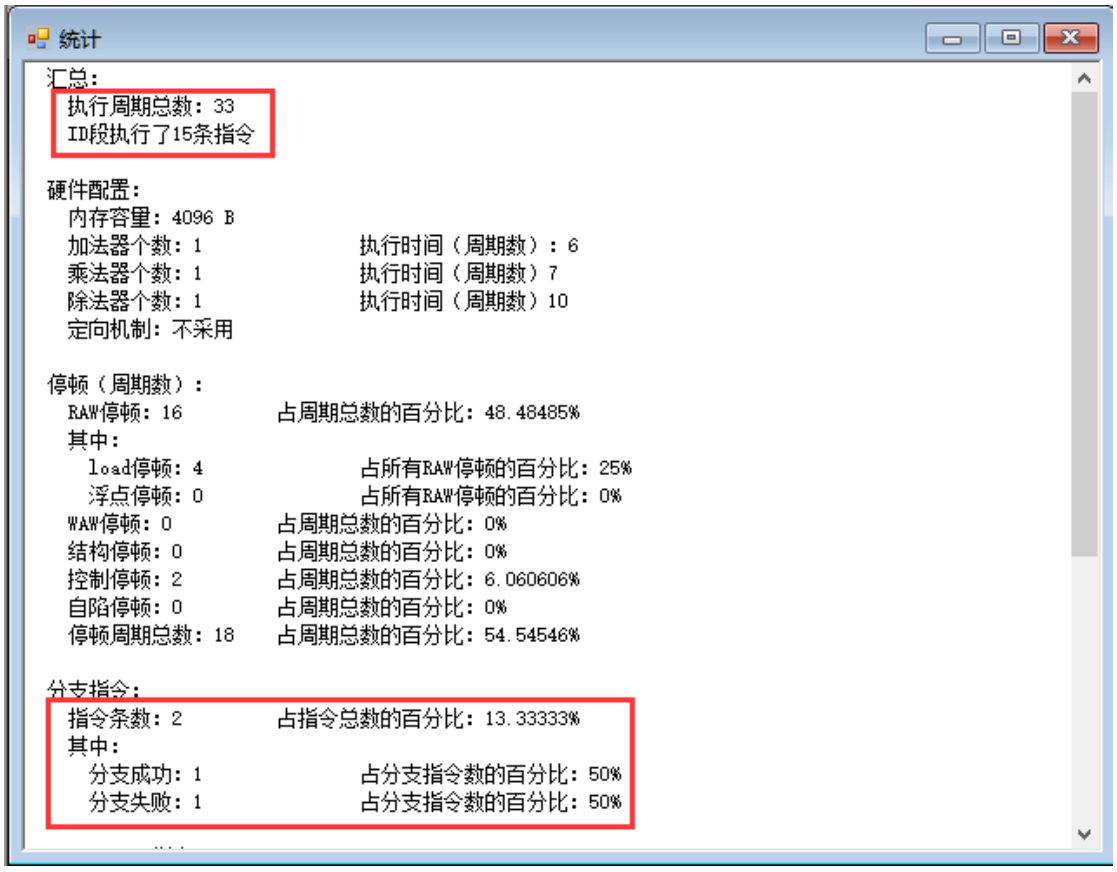
- 第 18 周期得到分支结果，分支成功



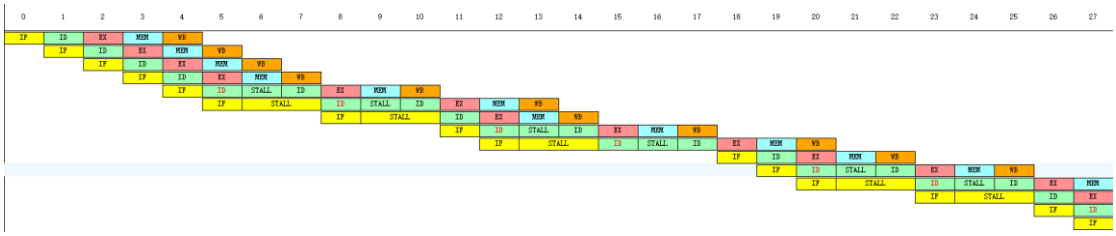
- 第 31 周期开始第二次分支



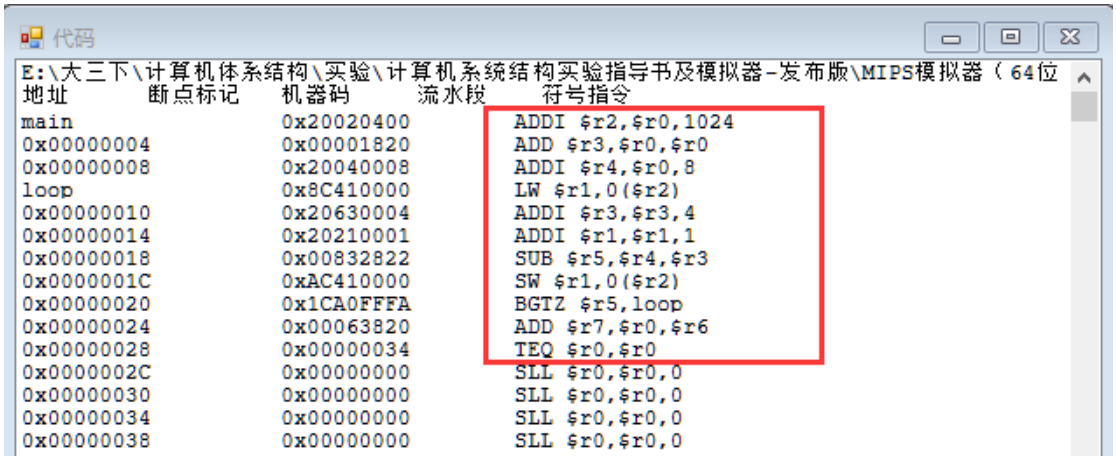
- 第 33 周期得到分支结果



- 第 38 周期执行结束，RAW 停顿 16 次，停顿周期总数 19，占比 50%

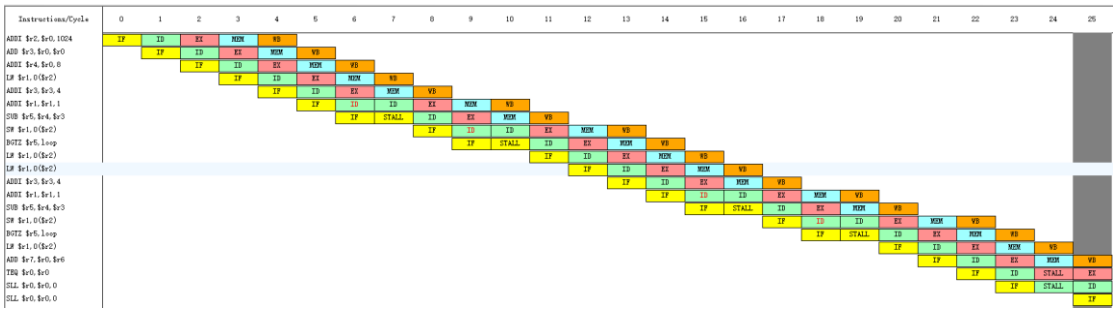


5.4 假设延迟槽为一个，自己对 branch.s 程序进行指令调度（自己修改源程序），将调度后的程序重新命名为 delayed-branch.s。

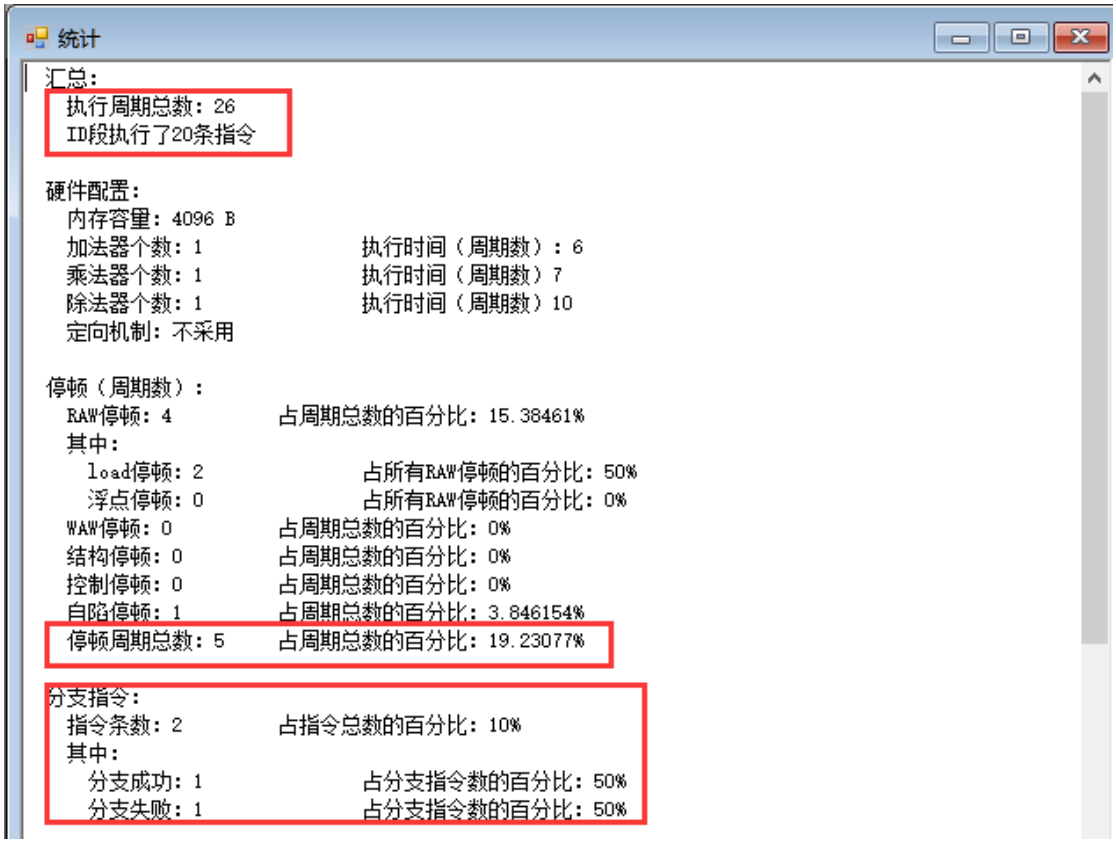


打开延迟槽，执行该程序，观察其时间周期图

5.5 载入 delayed-branch.s，打开延迟分支功能，执行该程序，观察其时钟周期图，记录程序执行的总时钟周期数



可以看到总时钟周期数为 26，停顿周期总数为 5，RAW 停顿 4 次，占周期总数的百分比 19.23077%



5.6 实验结论

没采用分支延迟的时候周期总数为 38，采用分支后的周期总数为 26，可知，在使用延迟槽后，指令在运行到跳转指令时，不会出现延迟等待，提高 CPU 的性能，并且在使用延迟后，指令在运行到跳转指令时，不会出现延迟等待，能够提高 CPU 的性能。

四、实验问题与心得

本次实验使用指令级和流水线操作级模拟器 MIPSsim 分析了指令调度和延迟分支的过程，了解了指令调度和延迟分支解决数据冲突的原理，加深了我对

于计算机流水线基本概念的理解，理解了 MIPS 结构是如何使用 5 段流水线来实现的，理解了各段的功能和基本操作，加深了我对数据冲突和结构冲突的理解，感受到了指令调度与延迟分支对 CPU 效率的提升，以及采用定向技术解决数据冲突带来的好处和性能的提升，进一步掌握了解决数据冲突的方法，掌握了如何应用定向技术来减少数据冲突引起的停顿，增强汇编语言编程能力，了解了对代码进行优化的方法。通过设置延迟槽，使得 CPU 提前执行一些指令，减少流水线的停顿，提高了 CPU 的效率。