

《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

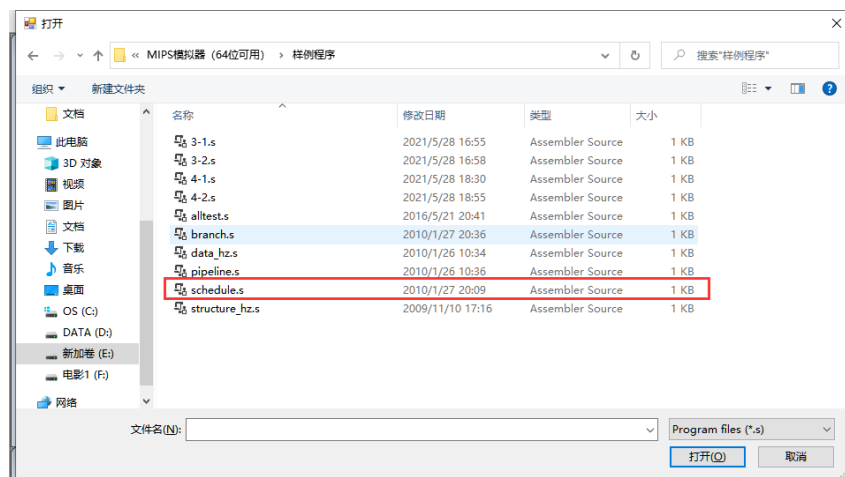
指令级和流水线操作级模拟器 MIPSsim

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



```

代码
E:\大三下\计算机体系结构\实验\计算机系统结构实验指导书及模拟器-发布版\MIPS模拟器 (64位)
地址 断点标记 机器码 流水段 符号指令
main 0x24010038 ADDIU $r1,$r0,56
0x00000004 0x8C220000 LW $r2,0($r1)
0x00000008 0x00022020 ADD $r4,$r0,$r2
0x0000000C 0xAC240000 SW $r4,0($r1)
0x00000010 0x8C260004 LW $r6,4($r1)
0x00000014 0x00C14020 ADD $r8,$r6,$r1
0x00000018 0x71416002 MUL $r12,$r10,$r1
0x0000001C 0x01818020 ADD $r16,$r12,$r1
0x00000020 0x02019020 ADD $r18,$r16,$r1
0x00000024 0xAC320010 SW $r18,16($r1)
0x00000028 0x8C340008 LW $r20,8($r1)
0x0000002C 0x728EB002 MUL $r22,$r20,$r14
0x00000030 0x734EC002 MUL $r24,$r26,$r14
0x00000034 0x00000034 TEQ $r0,$r0
A 0x00000004 SLLV $r0,$r0,$r0
0x0000003C 0x00000006 SRLV $r0,$r0,$r0
0x00000040 0x00000008 JR $r0
0x00000044 0x00000000 SLL $r0,$r0,0
0x00000048 0x00000000 SLL $r0,$r0,0
0x0000004C 0x00000000 SLL $r0,$r0,0
0x00000050 0x00000000 SLL $r0,$r0,0
0x00000054 0x00000000 SLL $r0,$r0,0

```

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 and 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

统计

汇总:

执行周期总数: 33
ID段执行了15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: 不采用

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: 16 占周期总数的百分比: 48.48485%

其中:

load停顿: 6 占有所有RAW停顿的百分比: 37.5%
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: 0 占周期总数的百分比: 0%
自陷停顿: 1 占周期总数的百分比: 3.030303%
停顿周期总数: 17 占周期总数的百分比: 51.51515%

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：

Instructions/Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13
ADDIU \$r1,\$r0,56	IF	ID	EX	MEM	WB									
LW \$r2,0(\$r1)		IF	ID	STALL	ID	EX	MEM	WB						
ADD \$r4,\$r0,\$r2			IF	STALL		ID	STALL	ID	EX	MEM	WB			
SW \$r4,0(\$r1)						IF	STALL		ID	STALL	ID	EX	MEM	WB

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)	IF STALL ID EX MEM WB
ADD \$r8,\$r6,\$r1	IF ID STALL ID EX MEM WB
LW \$r6,4(\$r1)	
ADD \$r8,\$r6,\$r1	与上一指令存在 RAW 冲突

ADD \$r8,\$r6,\$r1	IF ID STALL ID EX MEM WB
MUL \$r12,\$r10,\$r1	IF ID STALL ID EX MEM WB
ADD \$r16,\$r12,\$r1	IF ID STALL ID EX MEM WB
ADD \$r18,\$r16,\$r1	IF ID STALL ID EX MEM WB
SW \$r18,16(\$r1)	IF ID STALL ID EX MEM WB
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)	ID EX MEM WB
MUL \$r22,\$r20,\$r14	IF ID STALL ID EX MEM WB
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 after-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：

<pre> .text main: ADDIU \$r1,\$r0,A MUL \$r24,\$r26,\$r14 LW \$r2,0(\$r1) LW \$r20,8(\$r1) MUL \$r12,\$r10,\$r1 </pre>

```

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

统计

汇总:

- 执行周期总数: 19
- ID段执行了15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: 不采用

执行时间 (周期数):

- 加法器: 6
- 乘法器: 7
- 除法器: 10

停顿 (周期数):

- RAW停顿: 2 占周期总数的百分比: 10.52632%
- 其中:
 - load停顿: 0 占有RAW停顿的百分比: 0%
 - 浮点停顿: 0 占有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: 0 占周期总数的百分比: 0%
 - 自陷停顿: 1 占周期总数的百分比: 5.263158%
- 停顿周期总数: 3 占周期总数的百分比: 15.78947%

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

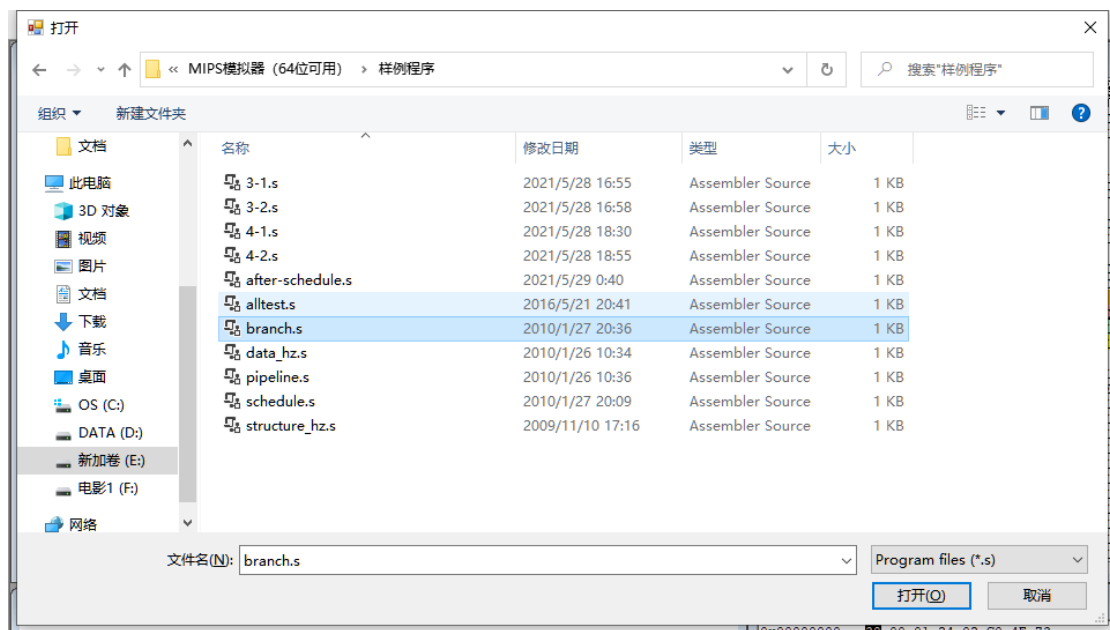
的百分比为 15.78947%

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

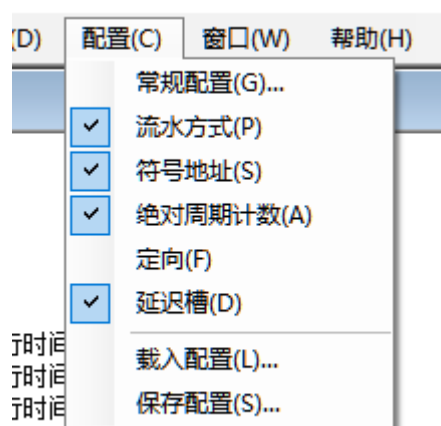
调度前和调度后的性能对比为，指定调度后时钟总周期数从 33 降到了 19，指定调度使指令顺序重新组合，减少了部分的数据冲突，使得 RAW 冲突的数目减少到了 2，通过指令调度技术显著地提高了 CPU 的使用率，大大减少了指令冲突的次数。

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）

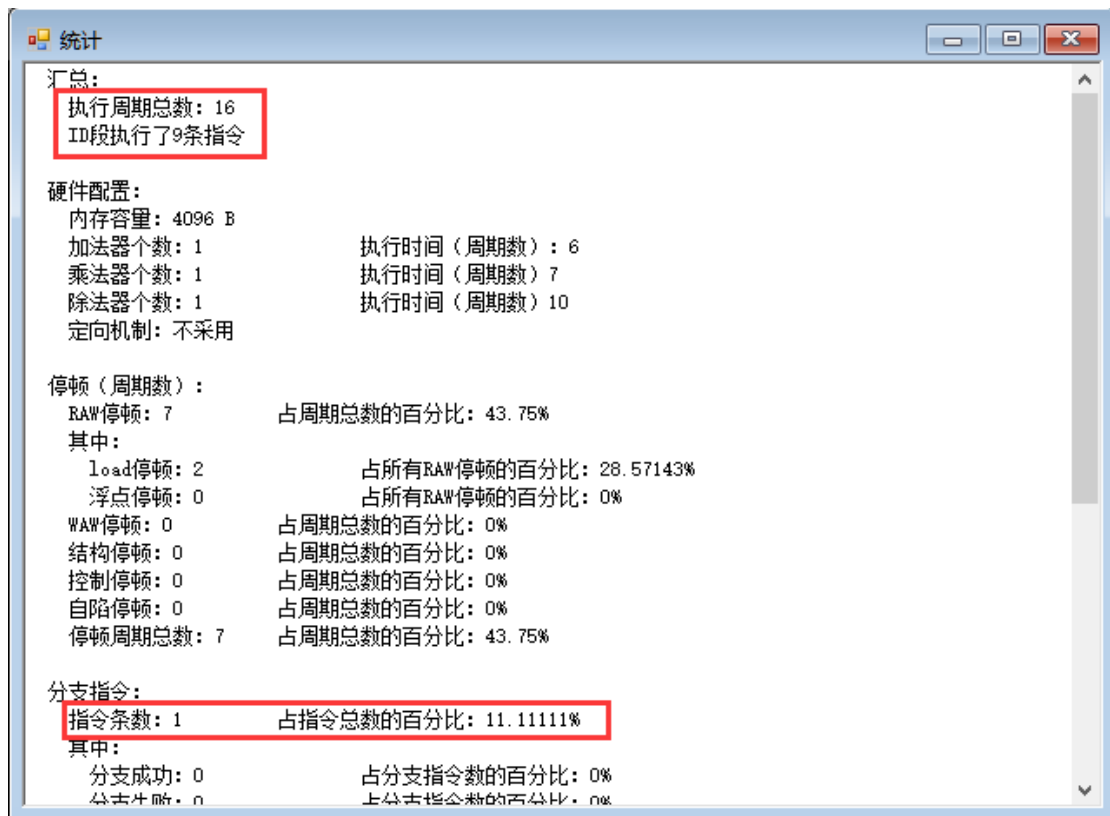


5.2 关闭延迟分支功能。这是通过在“配置”->“延迟槽”选项来实现的

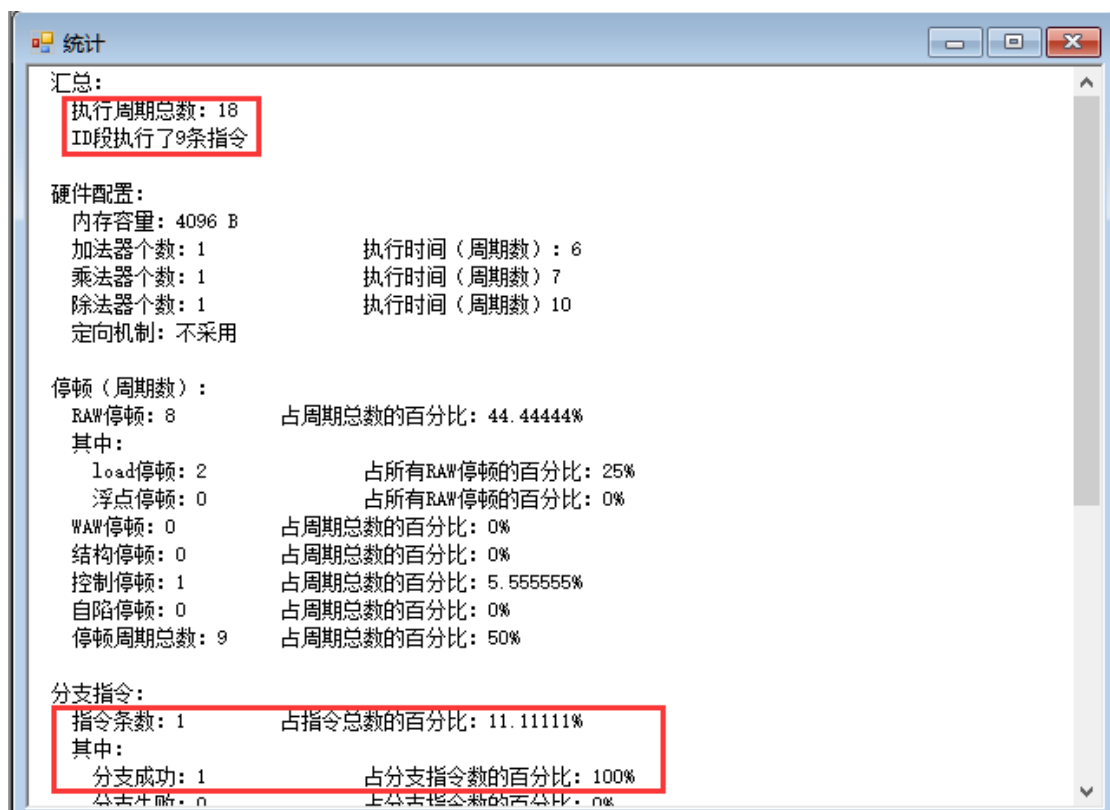


5.3 执行该程序，观察并记录发生分支延迟的时刻，记录该程序执行的总时钟周期数

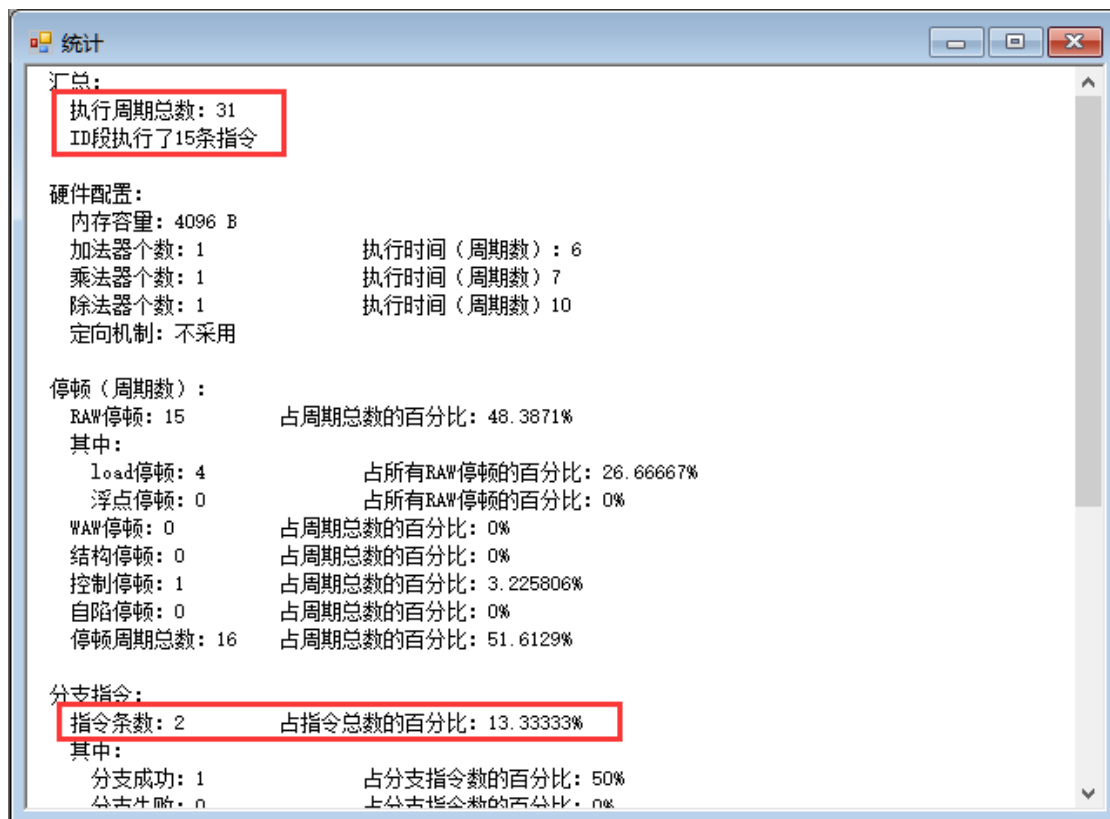
- 第 16 周期开始第一次分支



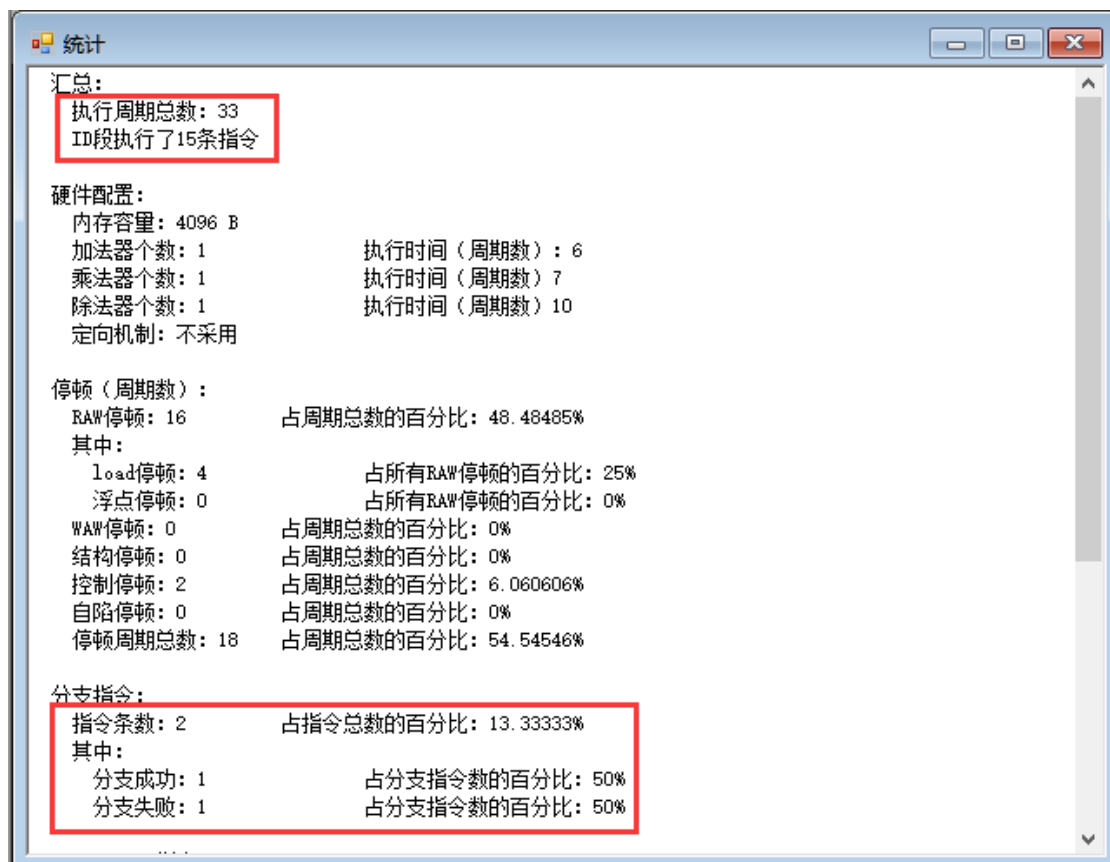
- 第 18 周期得到分支结果，分支成功



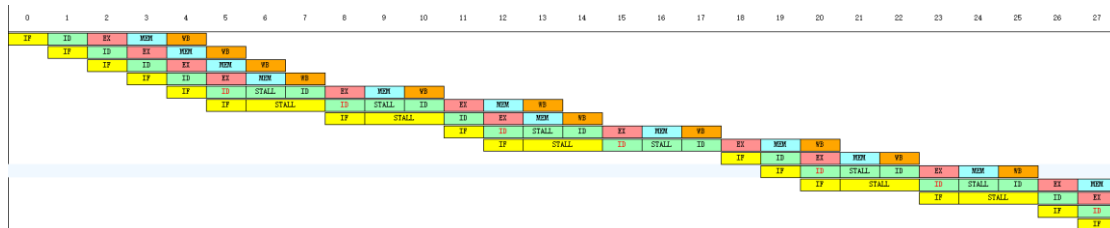
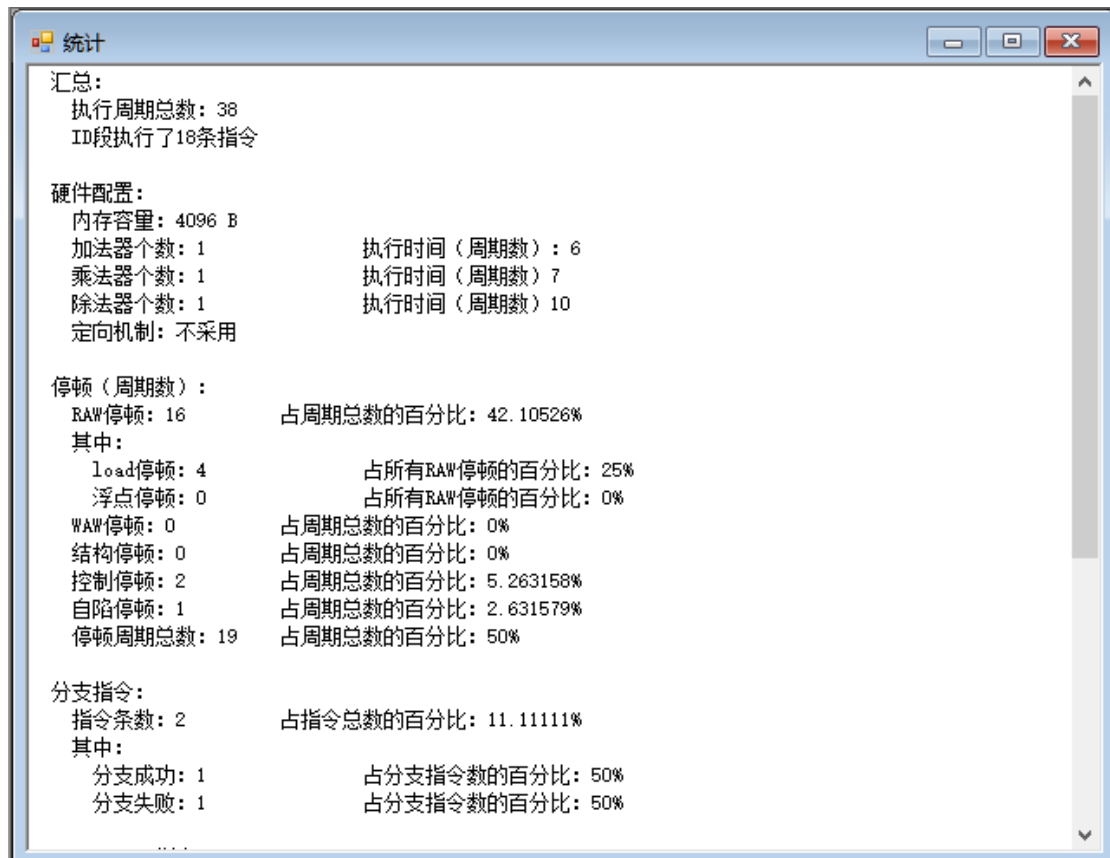
● 第 31 周期开始第二次分支



● 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%

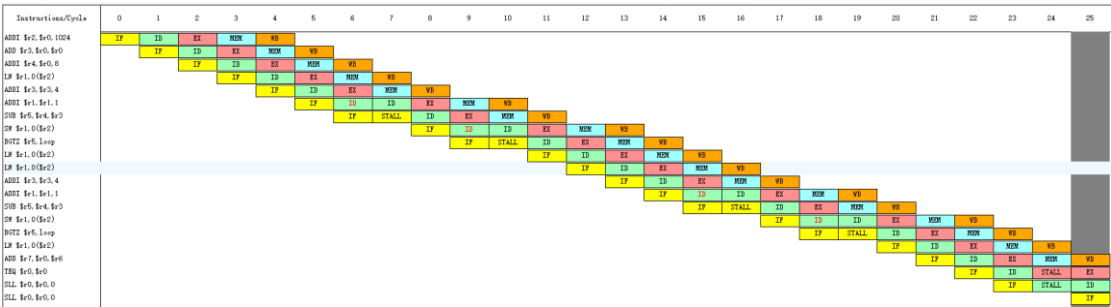


5.4 假设延迟槽为一个，自己对 branch.s 程序进行指令调度（自己修改源程序），将调度后的程序重新命名为 delayed-branch.s。

地址	断点标记	机器码	流水段	符号指令
main		0x20020400		ADDI \$r2,\$r0,1024
0x00000004		0x00001820		ADD \$r3,\$r0,\$r0
0x00000008		0x20040008		ADDI \$r4,\$r0,8
loop		0x8C410000		LW \$r1,0(\$r2)
0x00000010		0x20630004		ADDI \$r3,\$r3,4
0x00000014		0x20210001		ADDI \$r1,\$r1,1
0x00000018		0x00832822		SUB \$r5,\$r4,\$r3
0x0000001C		0xAC410000		SW \$r1,0(\$r2)
0x00000020		0x1CA0FFFA		BGTZ \$r5,loop
0x00000024		0x00063820		ADD \$r7,\$r0,\$r6
0x00000028		0x00000034		TEQ \$r0,\$r0
0x0000002C		0x00000000		SLL \$r0,\$r0,0
0x00000030		0x00000000		SLL \$r0,\$r0,0
0x00000034		0x00000000		SLL \$r0,\$r0,0
0x00000038		0x00000000		SLL \$r0,\$r0,0

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



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。