# 实验5 指令调度与延迟分支

## 1 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）熟练掌握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。

## 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。

## 3 实验说明

（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（双击各段，就可以看到各流水线寄存 器中的内容）。

（3） 选择“配置”→“流水方式”选项，使模拟器工作在流水方式下。

（4） 用指令调度技术解决流水线中的结构冲突与数据冲突：

1） 启动MIPSsim。

2） 用MIPSsim的“文件”->“载入程序”选项来加载schedule.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“配置“->”定向“选项来实现的。

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。

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。）

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。

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。

（5） 用延迟分支技术减少分支指令对性能的影响：

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。

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。

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。

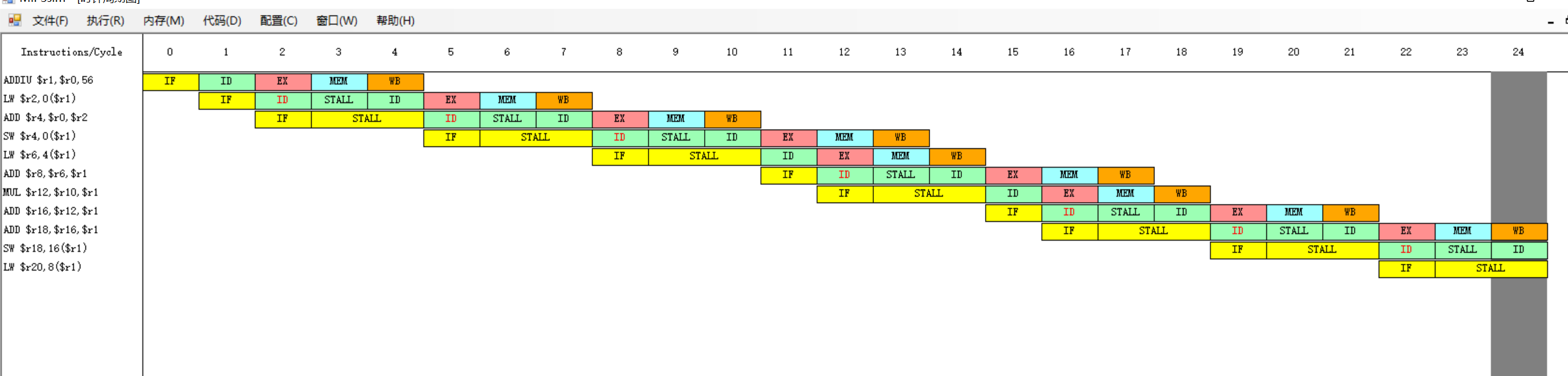
4） 假设延迟槽为一个，自己对 branch.s 程序进行指令调度（自己修改源程序），将 调度后的程序重新命名为delayed-branch.s。

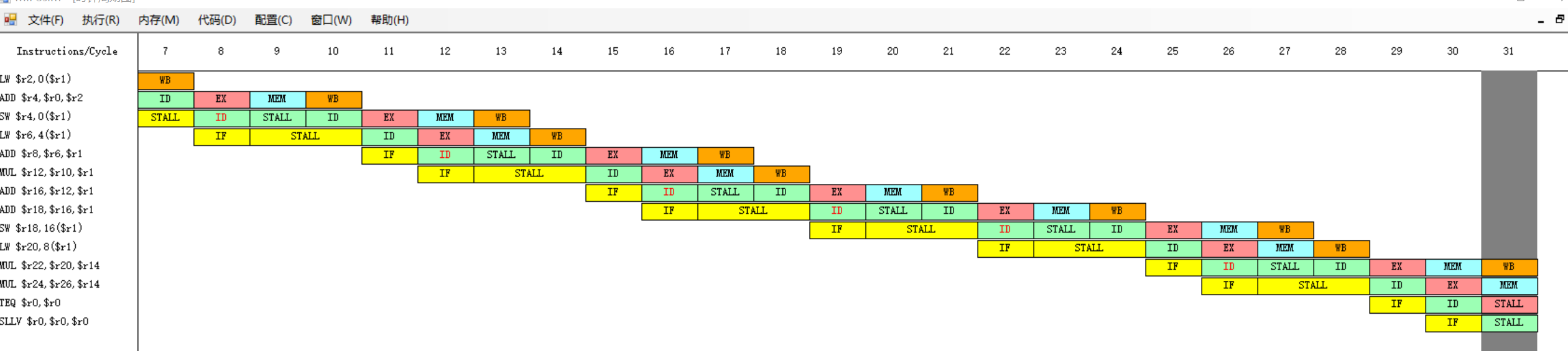
5） 载入delayed-branch.s，打开延迟分支功能，执行该程序，观察其时钟周期图，记 录程序执行的总时钟周期数。

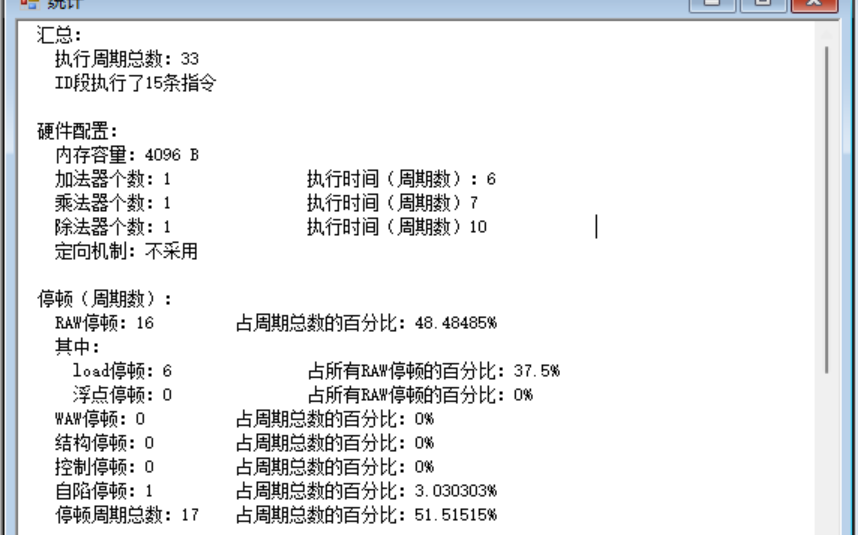
## 4 实验步骤及内容

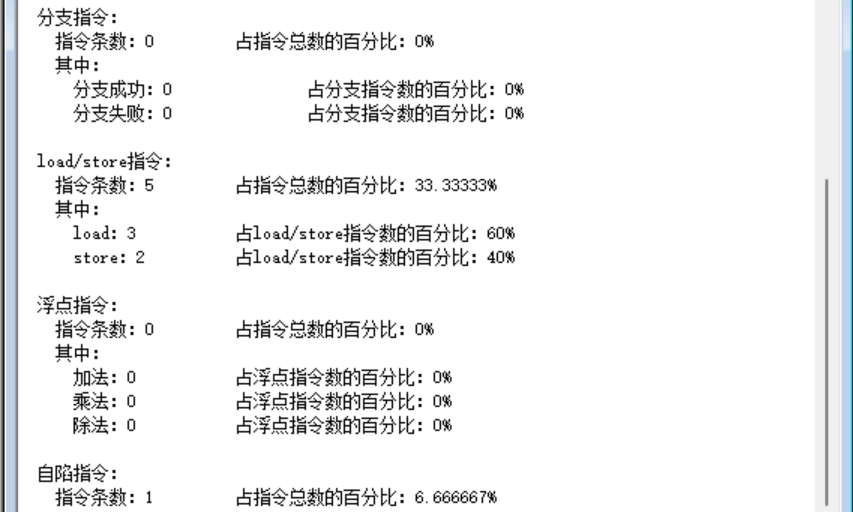
### 4.1 schedule实验

首先配置为流水方式，关闭定向功能。载入schedule.s，F7单步执行，得到结果如下：









执行的总周期为33， 停顿的周期数为17，占总周期的51.51515%。其中RAW冲突有16个周期，自馅停顿有1个周期。

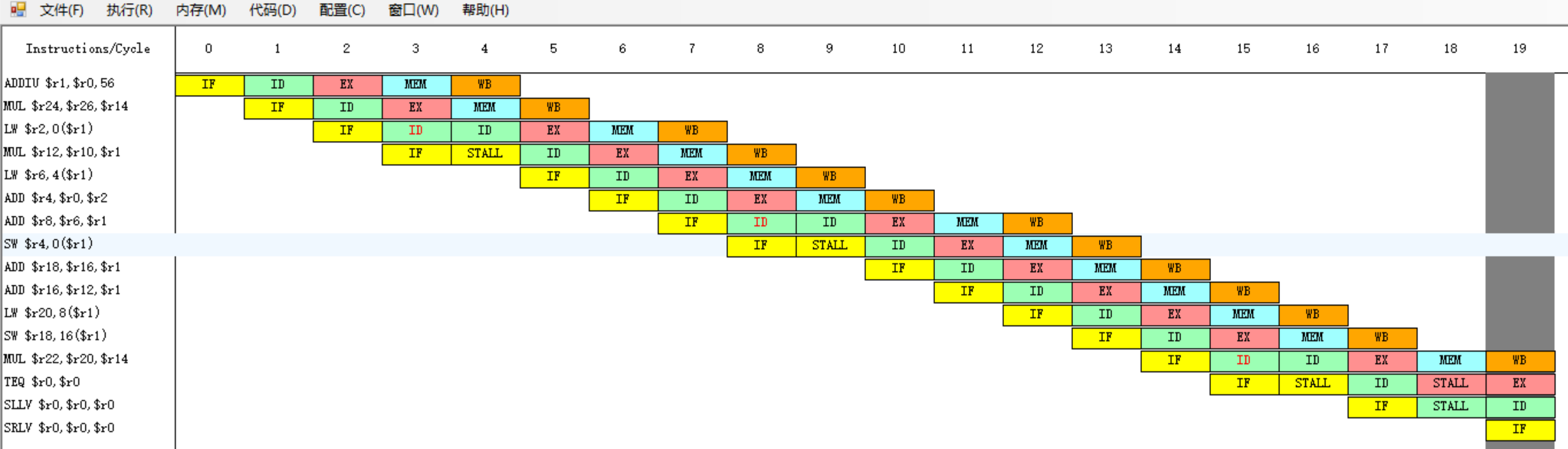
发生冲突的地方为：

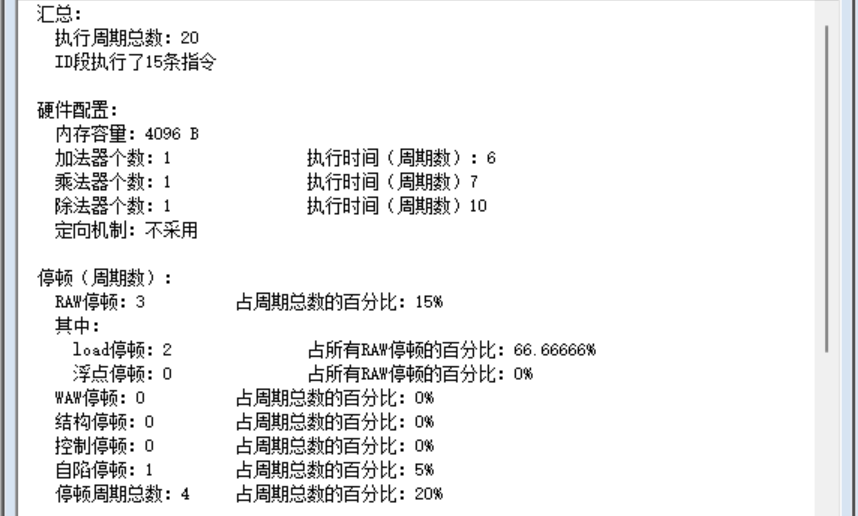
|  |  |
| --- | --- |
| 冲突指令 | 引起冲突的指令 |
| ADD $r4,$r0,$r2 | LW $r2,0($r1) |
| SW $r4,0($r1) | ADD $r4,$r0,$r2 |
| LW $r6,4($r1) | SW $r4,0($r1) |
| MUL $r12,$r10,$r1 | ADD $r8,$r6,$r1 |
| ADD $r18,$r16,$r1 | ADD $r16,$r12,$r1 |
| SW $r18,16($r1) | ADD $r18,$r16,$r1 |
| LW $r20,8($r1) | SW $r18,16($r1) |
| MUL $r24,$r26,$r14 | MUL $r22,$r20,$r14 |

针对这些冲突，我更改了一些指令执行的顺序，如下：

1. .text
2. main:
3. ADDIU  $r1,$r0,A
4. MUL    $r24,$r26,$r14
5. LW     $r2,0($r1)
6. MUL    $r12,$r10,$r1
7. LW     $r6,4($r1)   # 先把r1的东西存到r6里，因为r1和r6都不会动
8. ADD    $r4,$r0,$r2
9. ADD    $r8,$r6,$r1  # 先让r6 r1相加放到r8里，不影响r4放到r1地址
10. SW     $r4,0($r1)   # r1 \* r10放r12里跟把r4放到r1里不冲突
11. ADD    $r18,$r16,$r1
12. ADD    $r16,$r12,$r1 # 这两个交换，不影响r16使用
13. LW     $r20,8($r1)
14. SW     $r18,16($r1)
15. MUL    $r22,$r20,$r14
17. TEQ $r0,$r0
19. .data
20. A:
21. .word 4,6,8

但是还是会有冲突，执行结果如下：

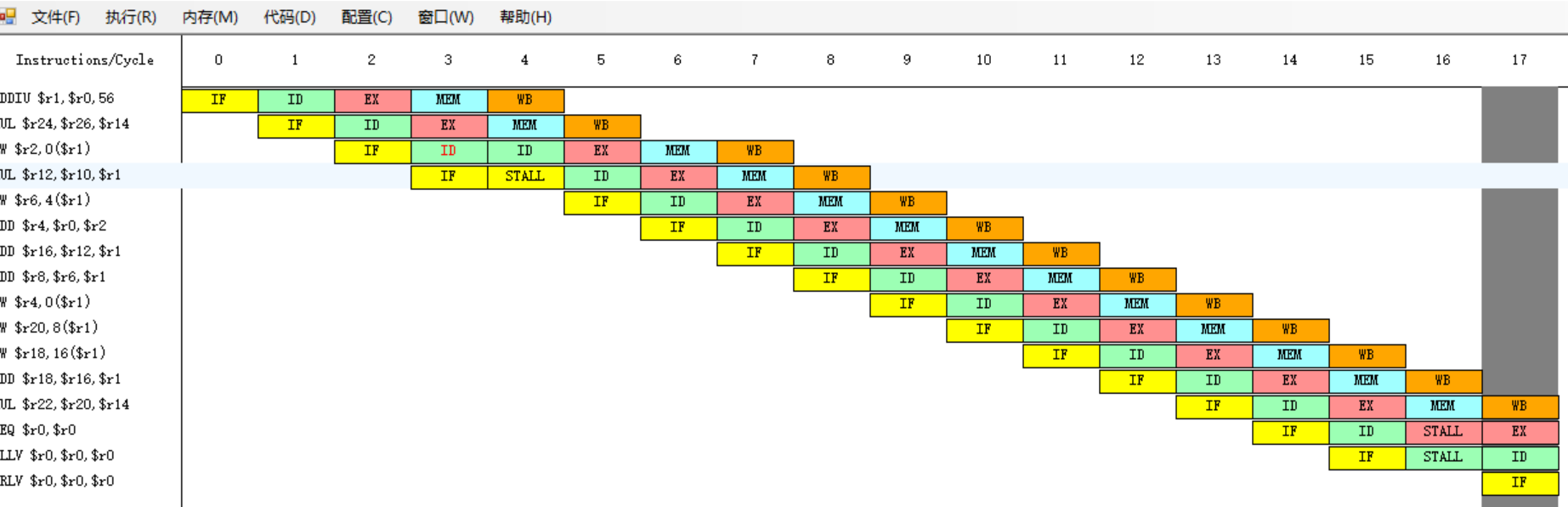


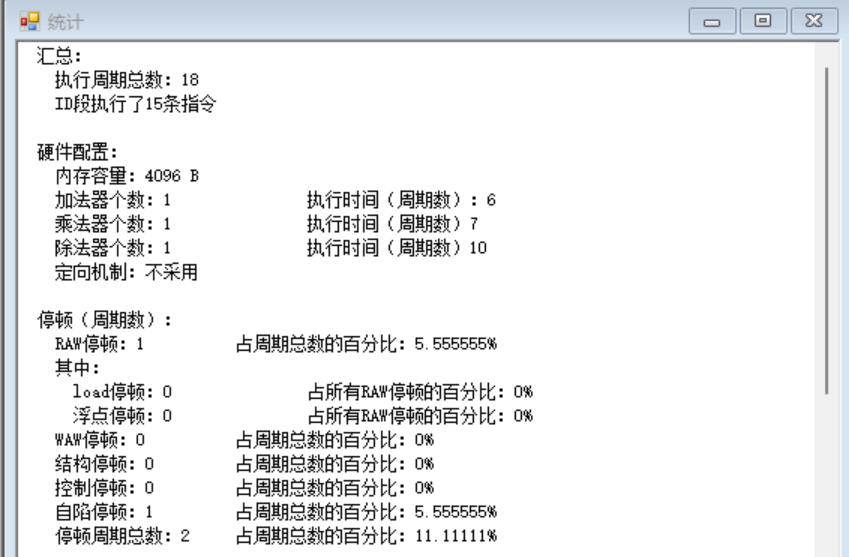


可以看到，还是会有stall的地方，以及还有三个RAW停顿。下面进行进一步优化。优化代码及注释如下：

1. .text
2. main:
3. ADDIU  $r1,$r0,A
4. MUL    $r24,$r26,$r14
5. LW     $r2,0($r1)
6. MUL    $r12,$r10,$r1    # r12 往前提
7. LW     $r6,4($r1)    # 先把r1的东西存到r6里，因为r1和r6都不会动
8. ADD    $r4,$r0,$r2
9. ADD    $r16,$r12,$r1
10. ADD    $r8,$r6,$r1
11. SW     $r4,0($r1)    # 得放到对r4的操作后三个位置才不冲突
12. LW     $r20,8($r1)
13. SW     $r18,16($r1)
14. ADD    $r18,$r16,$r1 # 原本SW和MUL之间有冲突，插入一条无关指令后不冲突
15. MUL    $r22,$r20,$r14
16. TEQ $r0,$r0
18. .data
19. A:   .word 4,6,8

优化结果如下：



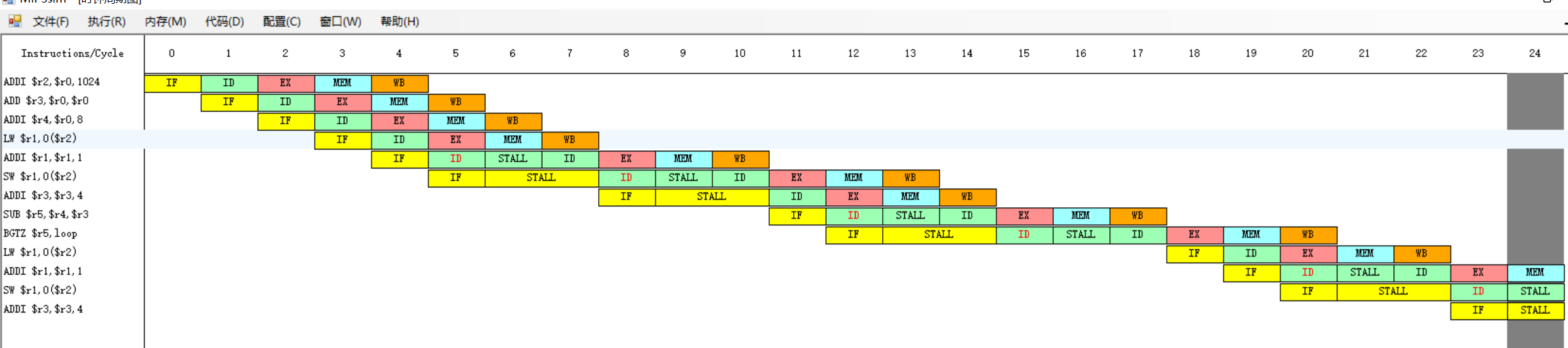


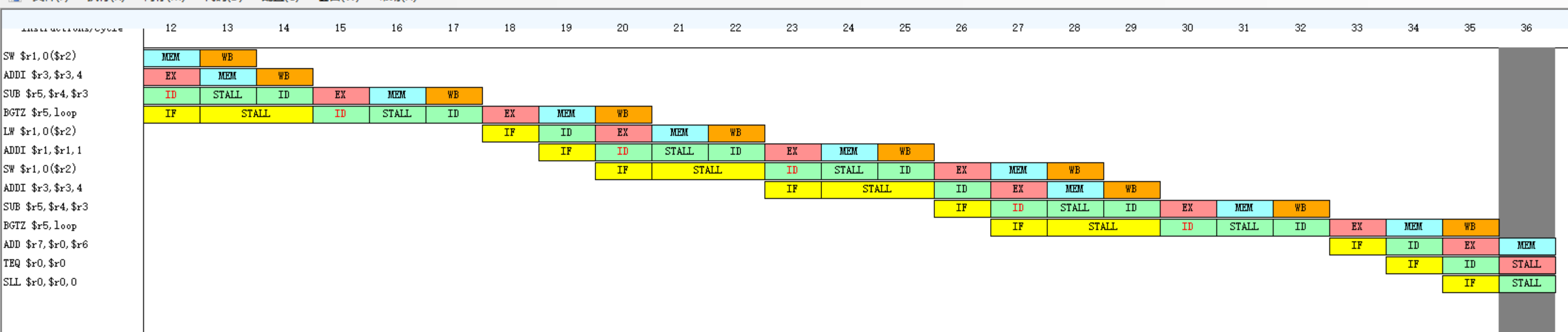
一共只有18个周期，其中RAW停顿一个周期，占总周期的百分比为5.55555%。

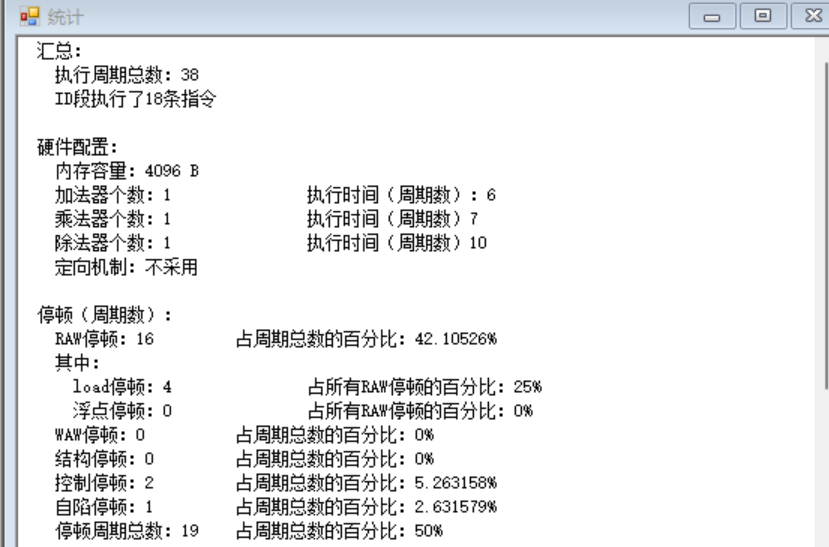
可以看到，较最开始已经有了很大提升。

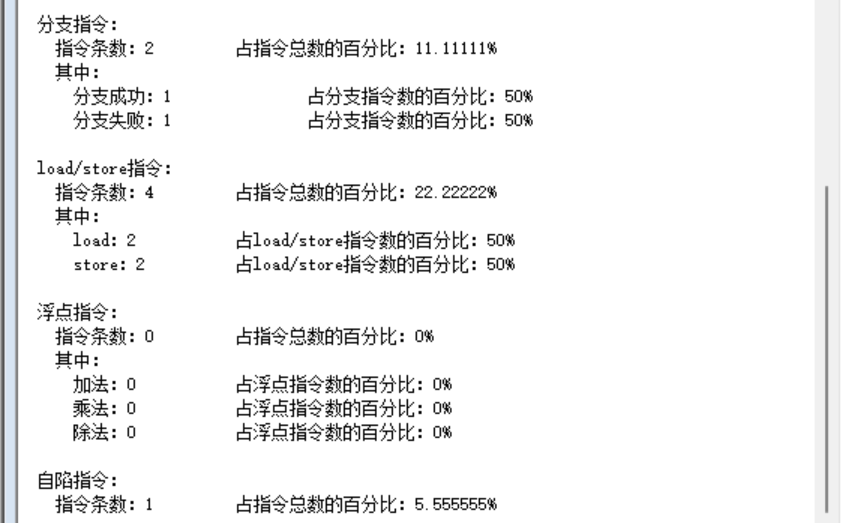
### 4.2 branch实验

载入branch.s，关闭延迟槽功能，F7单步运行，结果如下所示：



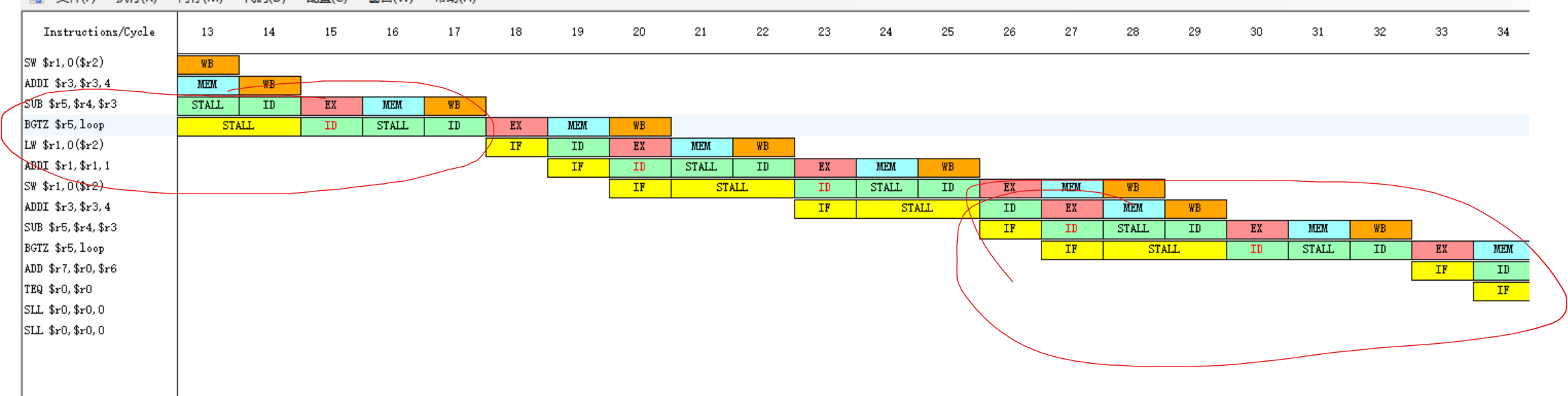






从结果来看，总共有38个周期，其中分支指令条数为2，成功失败各占50%。

跳转的周期如下所示：

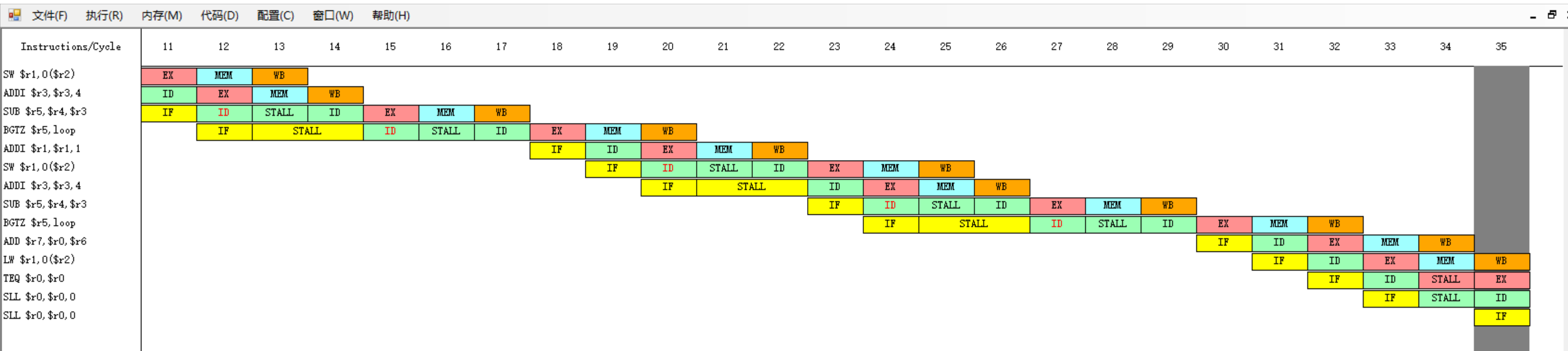


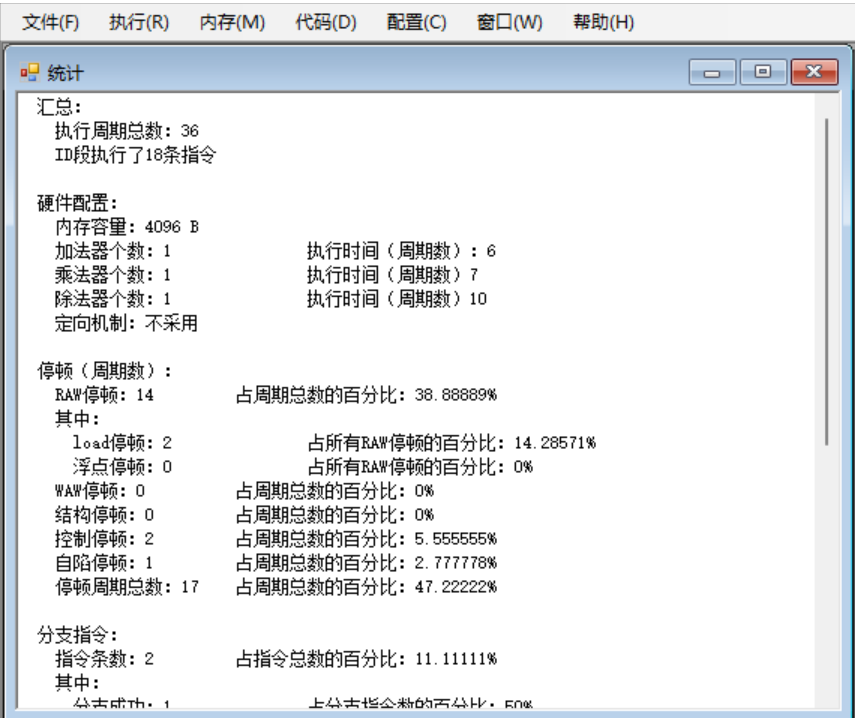
在branch.s程序中，只有一句BGTZ $r5, loop 进行条件跳转，所以要在这条语句后加入延迟分支，让流水线少一些停顿才行。

流水线中，分支指令执行时因为确定下一条指令的目标地址（紧随其后 or 跳转目标处？）一般要到第 2 级以后，在目标确定前流水线的取指级是不能工作的，即整个流水线就“浪费”（阻塞）了一个时间片，为了利用这个时间片，在体系结构的层面上规定跳转指令后 面的一个时间片为分支延迟槽（branch delay slot）。位于分支延迟槽中的指令总是被执行，与分支发生与否没有关系。这样就有效利用了一个时间片，消除了流水线的一个“气泡”。

所以我在分支后面加了一条总是被执行的loop的首条指令，并把这条指令移到loop之前先完成一次。

1. .text
2. main:
3. ADDI  $r2,$r0,1024
4. ADD   $r3,$r0,$r0
5. ADDI  $r4,$r0,8
6. # 这里可以先do loop的第一条
7. LW    $r1,0($r2)
8. loop:
9. ADDI  $r1,$r1,1
10. SW    $r1,0($r2) # 这里可以下移，减少r1的数据冲突
11. ADDI  $r3,$r3,4
12. SUB   $r5,$r4,$r3
13. BGTZ  $r5,loop          # 跳转到loop,loop先做LW $r1,0($r2)
14. ADD   $r7,$r0,$r6
15. # 此处可以放上loop最开始要做的事情
16. LW    $r1,0($r2)
17. TEQ   $r0,$r0





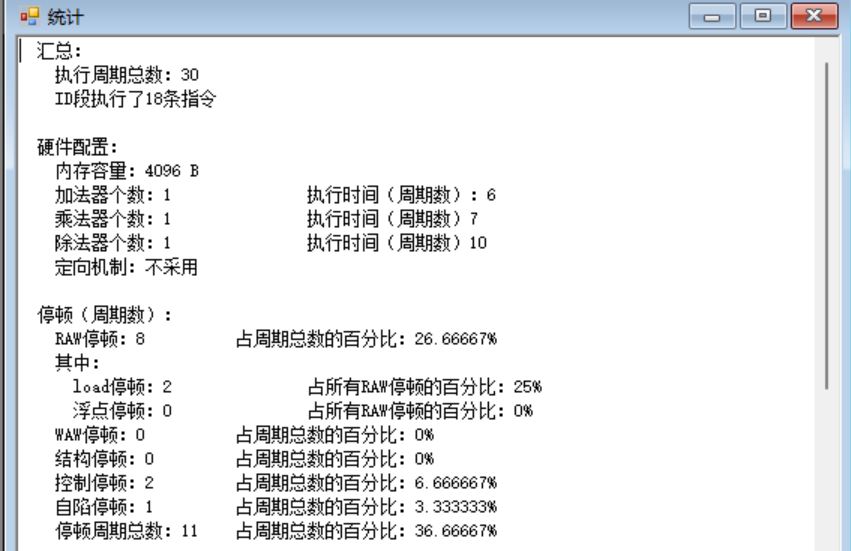
执行的周期总数下降了两个周期单位。

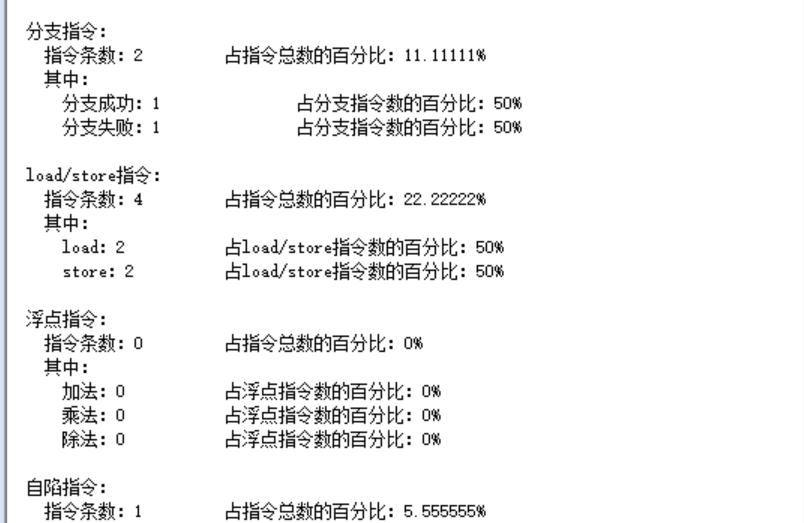
再次优化，可以将RAW冲突给优化掉。主要是对r1的冲突，

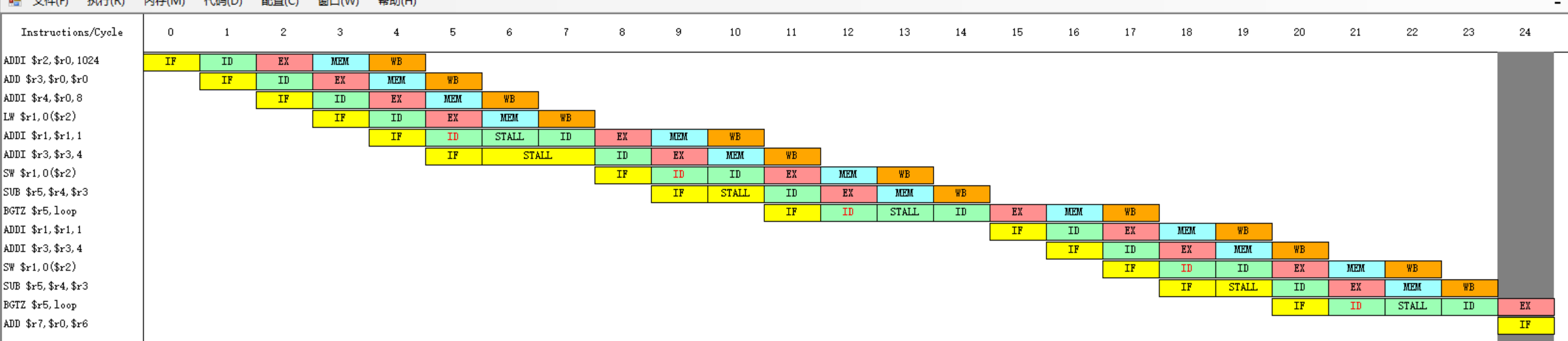
ADDI $r1, $r1, 1 指令和SW $r1, 0($r2)有冲突，可以将SW指令下移一条，和ADDI $r3, $r3, 4互换。优化后的代码如下：

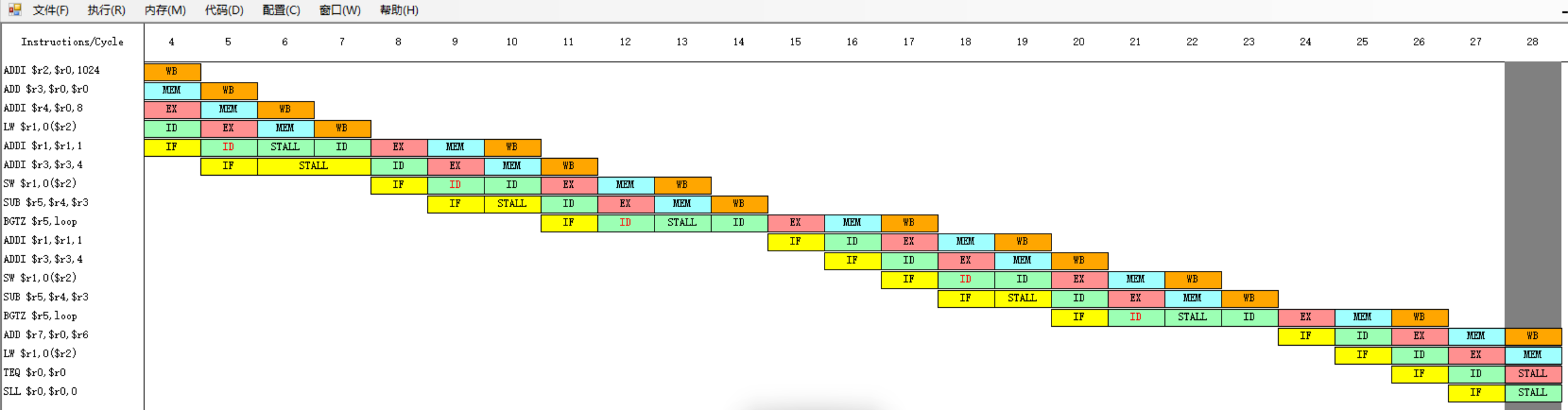
1. .text
2. main:
3. ADDI  $r2,$r0,1024
4. ADD   $r3,$r0,$r0
5. ADDI  $r4,$r0,8
6. # 这里可以先do loop的第一条
7. LW    $r1,0($r2)
8. loop:
9. ADDI  $r1,$r1,1
10. ADDI  $r3,$r3,4
11. SW    $r1,0($r2)        # 这里下移，减少r1的数据冲突
12. SUB   $r5,$r4,$r3
13. BGTZ  $r5,loop          # 跳转到loop,loop先做LW $r1,0($r2)
14. ADD   $r7,$r0,$r6
15. # 此处可以放上loop最开始要做的事情
16. LW    $r1,0($r2)
17. TEQ   $r0,$r0

执行结果如下所示：



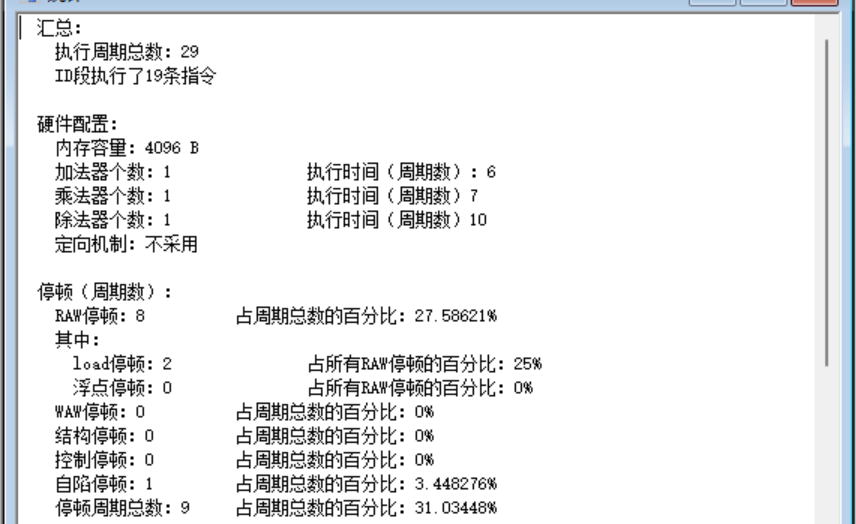


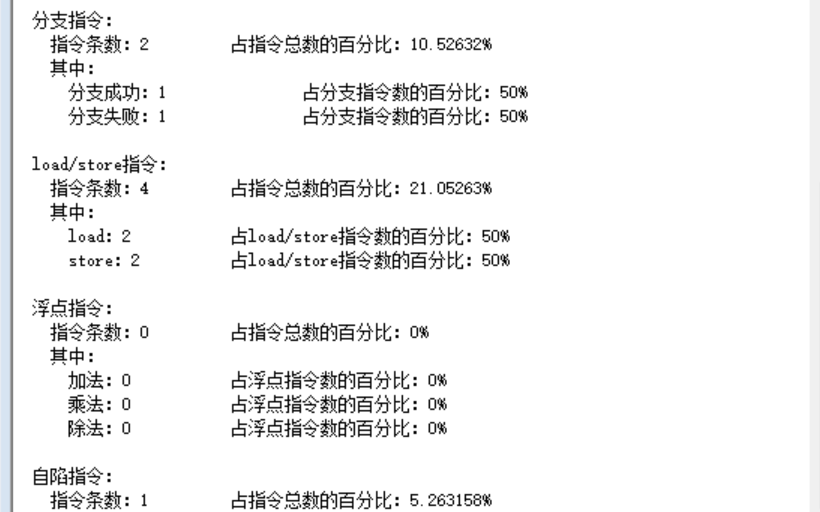


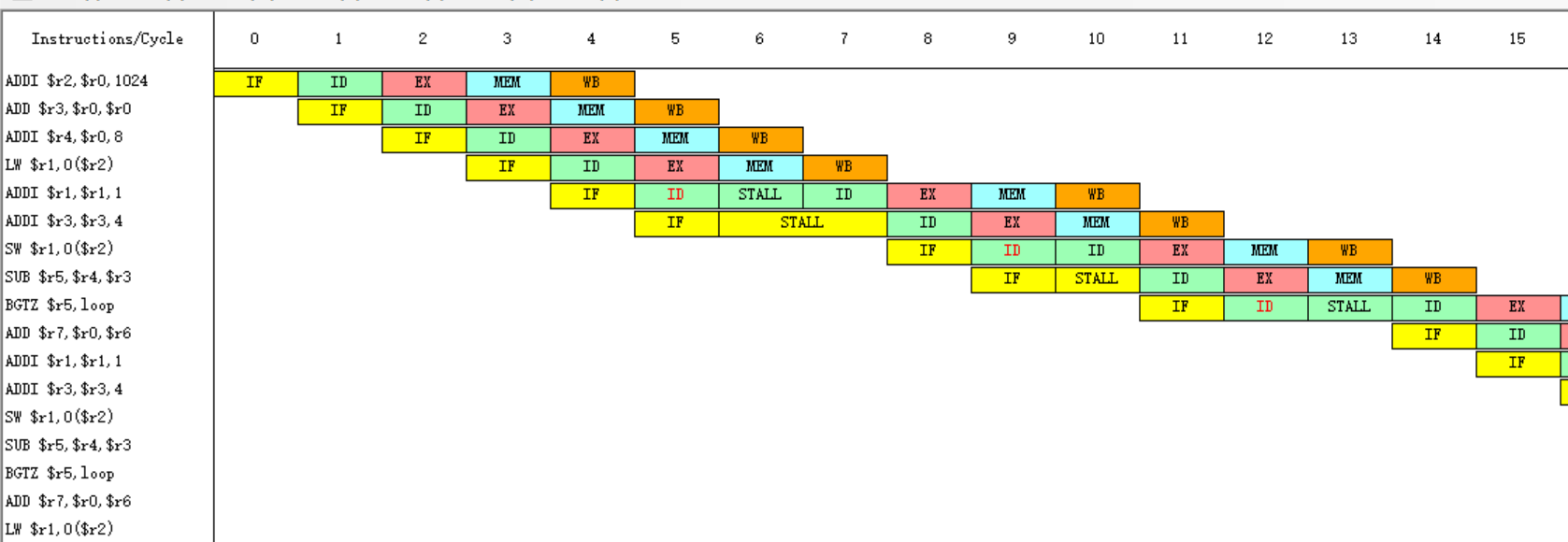


如上，为执行结果。整个的执行周期已经下降至30个周期。

下面加入延迟槽，执行结果如下：









加入延迟槽后，整个运行周期只有29个，并且停顿周期只有9个，下降至了31.03448%，有了很大提升。