**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(3)**

**实验项目名称： 处理器结构实验一**

**学 院： 计算机与软件学院**

**专 业： 计算机与软件学院所有专业**

**指 导 教 师： 刘刚**

**报告人： 何泽锋 学号： 2022150221 班级： 高性能特色班**

**实 验 时 间： 2024年10月31日-2024年11月7日**

**实验报告提交时间： 2024年11月7日**

**教务处制**

**一、 实验目标：**

了解MIPS的五级流水线，和在运行过程中的所产生的各种不同的流水线冒险

通过指令顺序调整，或旁路与预测技术来提高流水线效率

更加了解流水线细节和其指令的改善方法

更加熟悉MIPS指令的使用

**二、实验内容**

观察一段代码并运行，观察其中的流水线冒险，并记录统计信息。

对所给的代码进行指令序列的调整，以期避免数据相关，并记录统计信息。

启动forward功能，以获得性能提升，并且记录统计信息。

（选做：用perf记录x86中的数据相关于指令序列调整后的时间统计、

调整指令，以避免连续乘法间的阻塞。）

**三、实验环境**

硬件：桌面PC

软件：Windows，WinMIPS64仿真器

**四、****实验步骤及说明**

首先，我们给出一段C代码，该段代码实现的是两个矩阵相加。

设有4\*4矩阵A和4\*4矩阵B相加，得到4\*4矩阵C：

for(int i = 0; i < 4; i++)

For(int j = 0; j < 4; j++)

C[i][j] = A[i][j] + B[i][j];

根据上述的C代码，我们将其转换成MIPS语言，然后运行，并进行分析。

MIPS代码如下：

.data

a: .word 1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4

b: .word 4,4,4,4,3,3,3,3,2,2,2,2,1,1,1,1

c: .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

len: .word 4

control: .word32 0x10000

data: .word32 0x10008

.text

start:daddi r17,r0,0

daddi r21,r0,a

daddi r22,r0,b

daddi r23,r0,c

ld r16,len(r0)

loop1: slt r8,r17,r16

beq r8,r0,exit1

daddi r19,r0,0

loop2: slt r8,r19,r16

beq r8,r0,exit2

dsll r8,r17,2

dadd r8,r8,r19

dsll r8,r8,3

dadd r9,r8,r21

dadd r10,r8,r22

dadd r11,r8,r23

ld r9,0**(r9**)

ld r10,0(r10)

dadd r12,r9,r10

sd r12,0(r11)

daddi r19,r19,1

j loop2

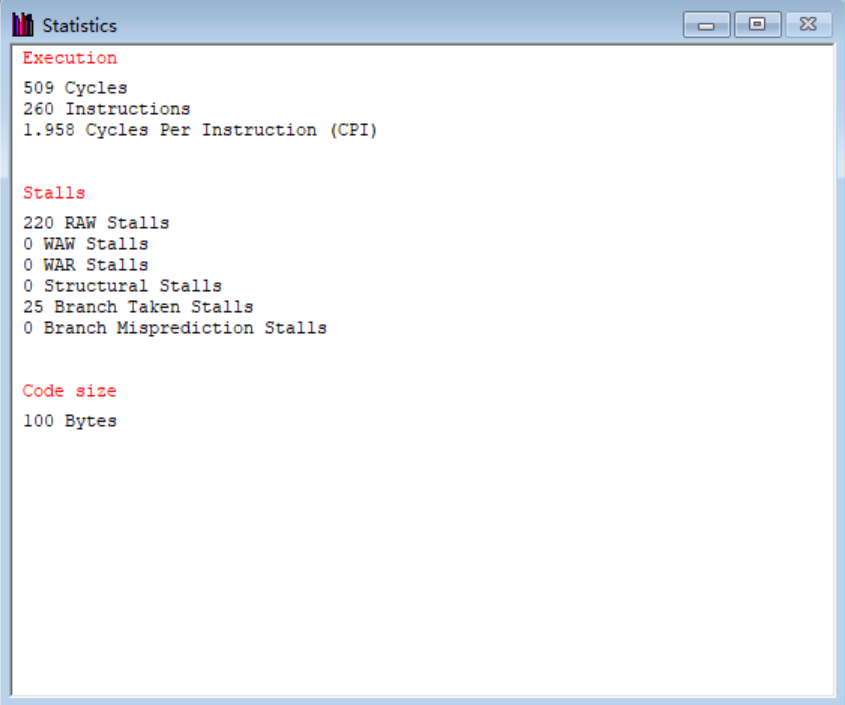
exit2:daddi r17,r17,1

j loop1

exit1: halt

实验前请保证winMIPS64配置中“Enable Forwarding”没有选中。将这段代码加载到WinMIPS64中，运行后观察结果（提供Statistic窗口截图）。 从Statistic窗口记录：本程序运行过程中总共产生了多少次RAW的数据相关。接下来，我们对产生数据相关的代码逐个分析，请列出产生数据相关的代码，并在下一步中进行分析和优化。

观察Statistic窗口截图，可以发现在关闭Enable Forwarding时发生了220次RAW，将此时记录初始状态，接下来尝试优化



1. 调整指令序列

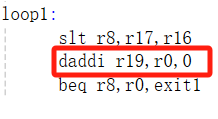
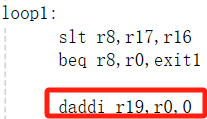
在这一部分，我们利用指令调整的方法对数据相关代码进行优化，规避数据相关。

通过**调整序列**来规避这个数据相关，在statics窗口中记录其效果。将此结果与初始的结果进行对比，报告**RAW相关的次数减少**的数量。

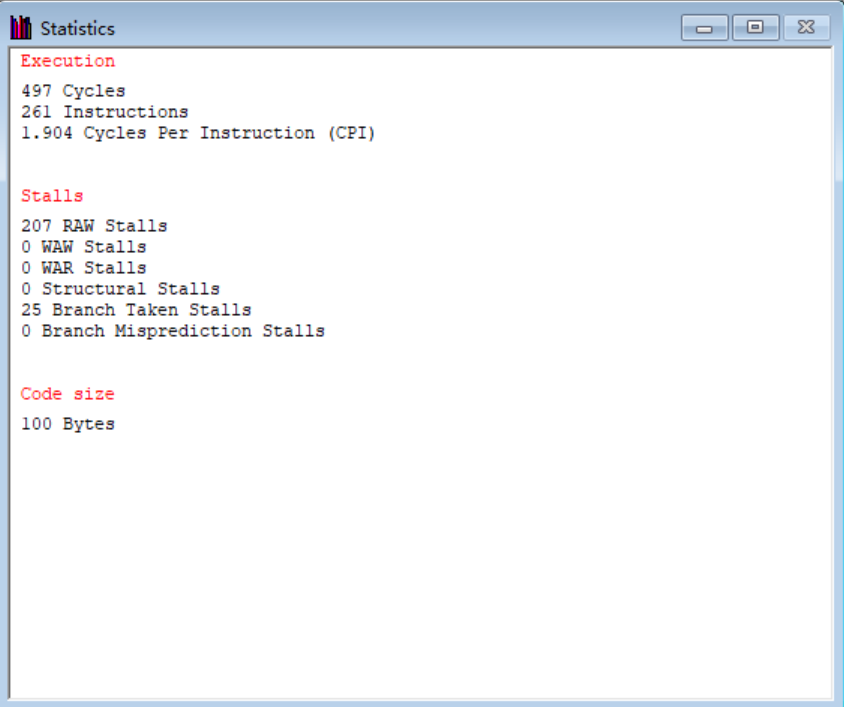
（1）观察Cycles窗口，可以发现beq指令在取数时出现了数据冒险，尝试将该指令延后，进而等待slt指令对r8的存储完成。此时虽然是跳转指令，但exit1条件成立时会直接跳转到程序结束，因此将daddi指令提前不会有过多影响



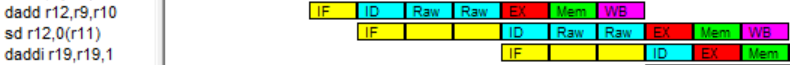
调整指令顺序：



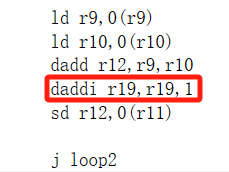
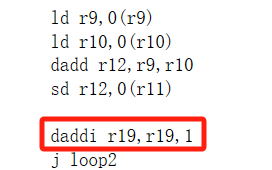
测试发现此时RAW减少到207次



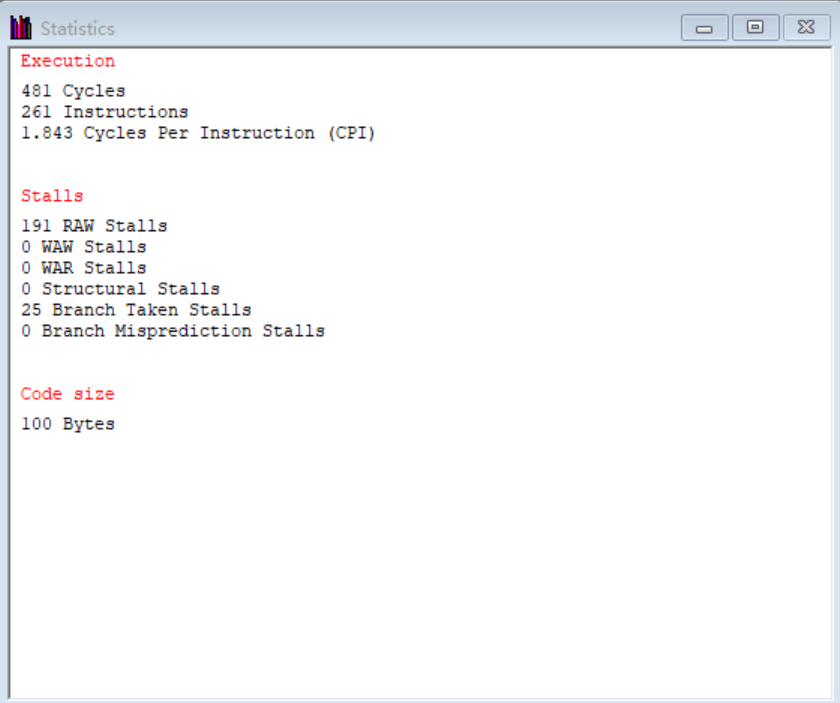
（2）继续调试发现dadd与sd指令之间发生了数据冒险，进而导致后方daddi指令也需要等待，因此可以将daddi调整到二者之间



调整指令顺序：

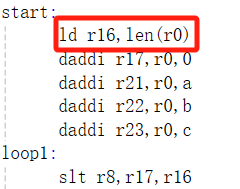
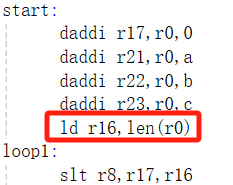


测试发现此时RAW为191次，在207次的基础上继续减少了16次

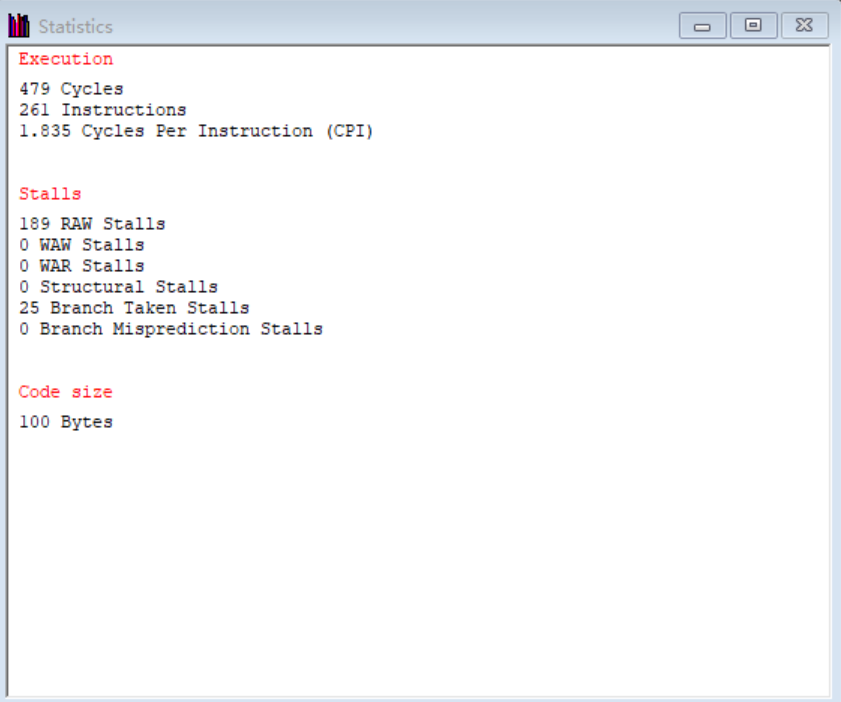


（3）在start部分，当ld读取数据到r16时，下一条指令则是对r16数据进行使用，因此发生数据冒险，可以将ld指令移到前方

调整指令顺序：

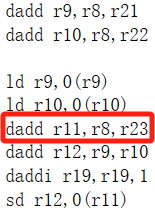
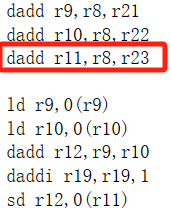


测试发现此时RAW为189次

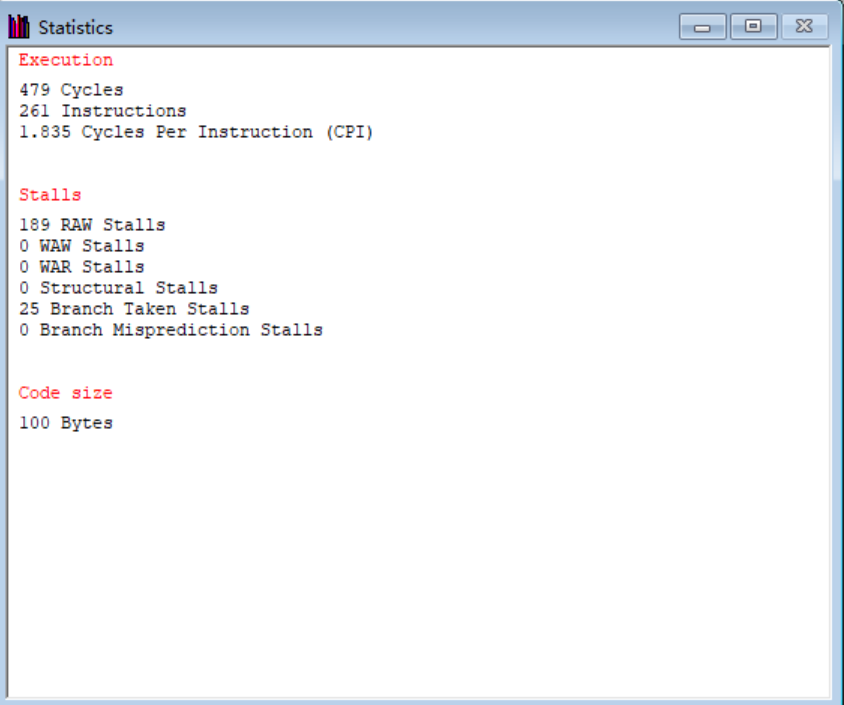


（4）取值到r10与调用r10进行加法时发生了数据冒险，观察代码可以发现dadd r11,r8,r23指令移动不会影响上下代码结果，并且还可以减少自身和r8使用的冲突，因此将该指令下移。

调整指令顺序：



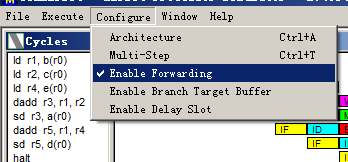
测试发现并没有减少RAW



1. Forwarding功能开启

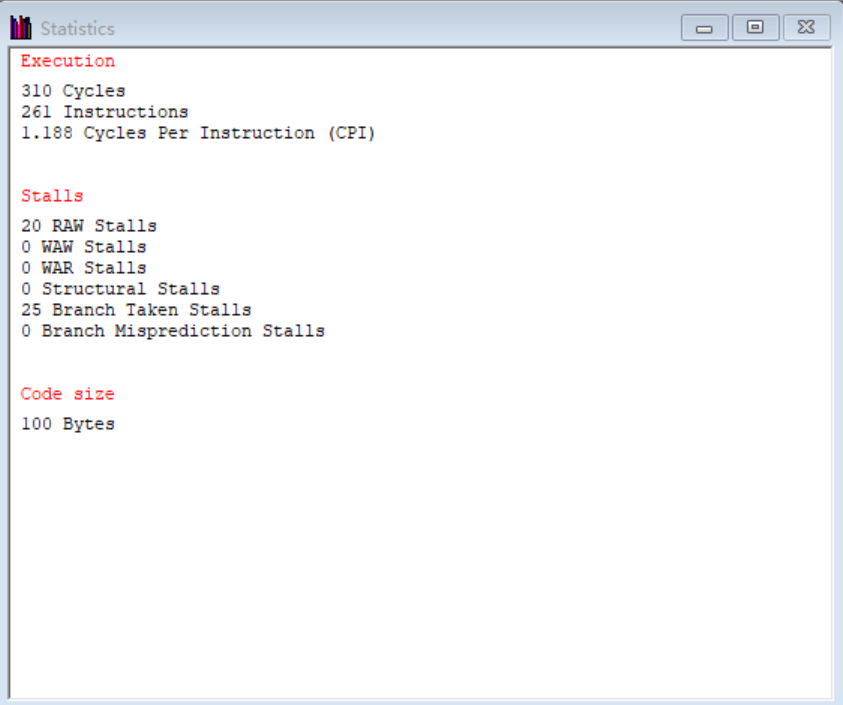
接下来，我们要展示Forwarding功能的优化效果。

首先，我们要知道如何开启Forwarding功能。法如下：点开***configure***下拉窗口，给***Enable Forwarding***选项左侧点上勾。



开启了Forwarding功能之后，我们再运行，查看结果，解释哪些数据相关的问题得到解决，并以截图说明问题解决前后的差异所在。

开启Enable Forwarding功能后，RAW从189降低至20次



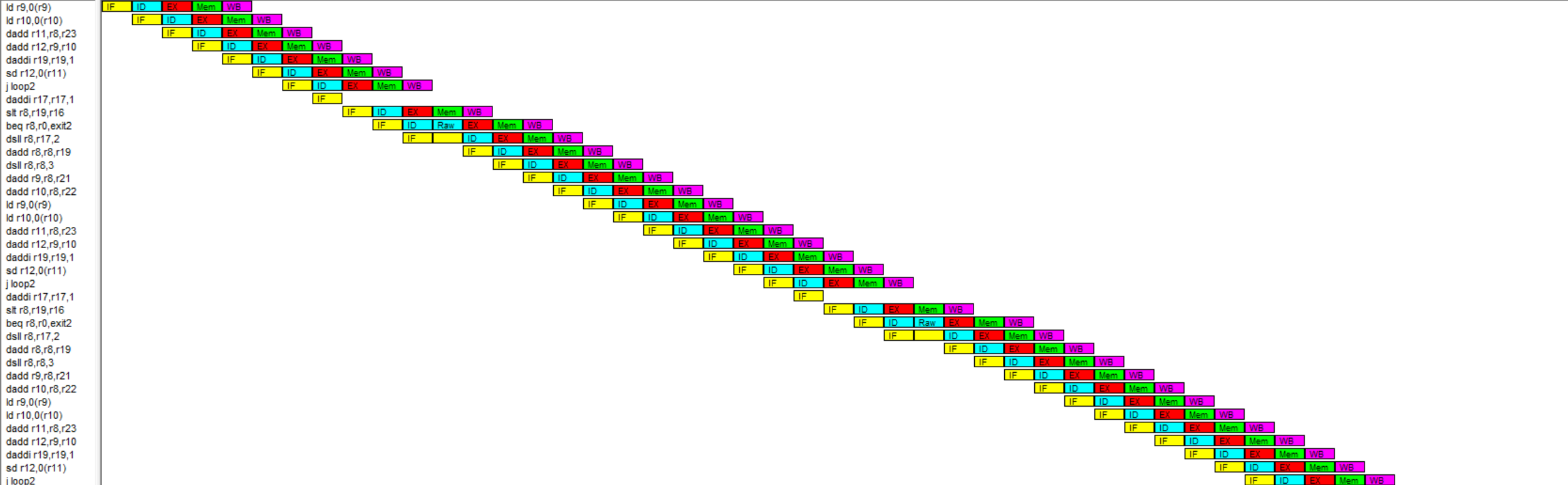
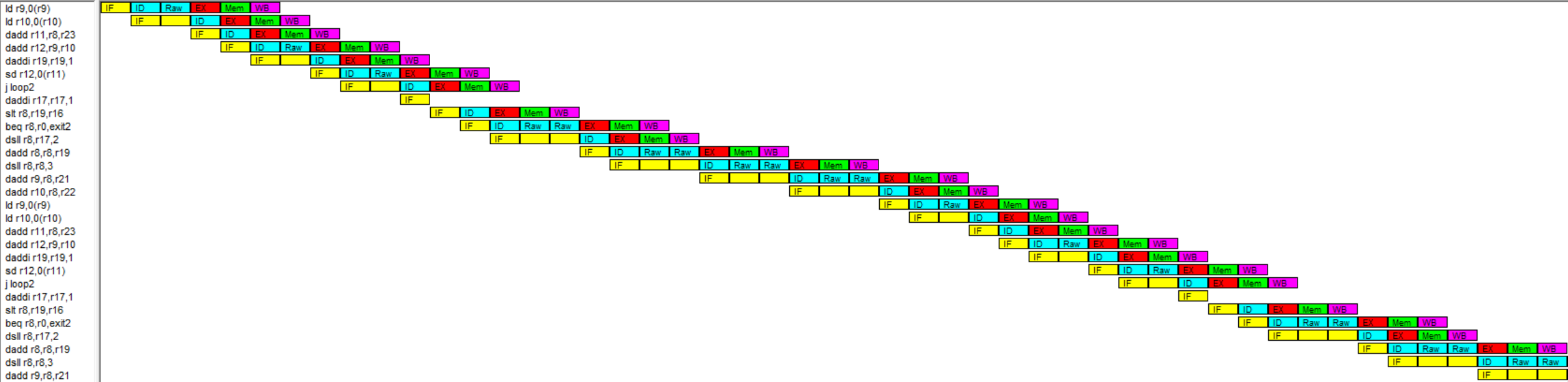
观察cycle变化，未开启前冲突主要发生在以下三条指令：

dsll r8,r17,2

dadd r8,r8,r19

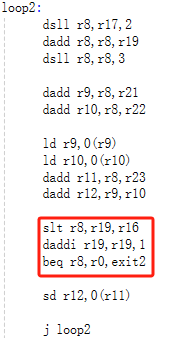
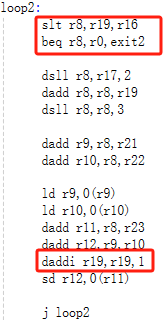
dsll r8,r8,3

当开启Enable Forwarding后，这三条指令的冲突被解决。

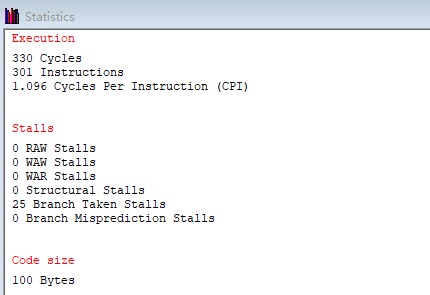


此处提供另一种调整方式（保证答案正确的情况下），需要注意该方式虽然能降低RAW次数，但会破坏循环的逻辑，导致代码额外运行一轮指令

如下图，分别为RAW为20与RAW为0的情况，可以看到RAW为0的代码会在运算完再增加循环控制变量值，这会导致内层循环每轮都会多运行一次，即多运算9\*i（i为外层循环次数）条指令，因此实际的时钟周期并不一定比原来要短，这取决于外层循环的次数



结合statistic可以看到，在RAW=0的情况下cycles为330，而RAW=20的cycles为310，额外增加了20个cycles，同时也破坏代码结构，因此不推荐该修改方式。



1. 结构相关优化

流水线中的结构相关，指的是流水线中多条指令在同一时钟周期内争用同一功能部件现象。即因硬件资源满足不了指令重叠执行的要求而发生的冲突。

在WinMIPS64中，我们可以在除法中观察到这种现象。要消除这种结构相关，我们可以采取调整指令位置的方法进行优化。在这个部分，我们首先给出几条C代码，然后将该代码翻译成MIPS代码（为了观察的方便，我们这里MIPS代码并不是逐一翻译，而是调整代码，使得其他部分数据相关已经优化，而两条除法指令连续出现），运行并查看结果。接着，调整代码序列，重新运行。观察优化效果。

下面是给出的C代码：

a = a / b

c = c / d

e = e + 1

f = f + 1

g = g + 1

h = h + 1

i = i + 1

j = j + 1

根据上述的C代码，我们给出数据相关优化的指令如下：

.data

a: .word 12

b: .word 3

c: .word 15

d: .word 5

e: .word 1

f: .word 2

g: .word 3

h: .word 4

i: .word 5

j: .word 6

.text

start:

ld r16, a(r0)

ld r17, b(r0)

ld r18, c(r0)

ld r19, d(r0)

ld r20, e(r0)

ld r21, f(r0)

ld r22, g(r0)

ld r23, h(r0)

ld r24, i(r0)

ld r25, j(r0)

ddiv r16, r16, r17

ddiv r18, r18, r19

daddi r20, r20, 1

daddi r21, r21, 1

daddi r22, r22, 1

daddi r23, r23, 1

daddi r24, r24, 1

daddi r25, r25, 1

sd r16, a(r0)

sd r18, c(r0)

sd r20, e(r0)

sd r21, f(r0)

sd r22, g(r0)

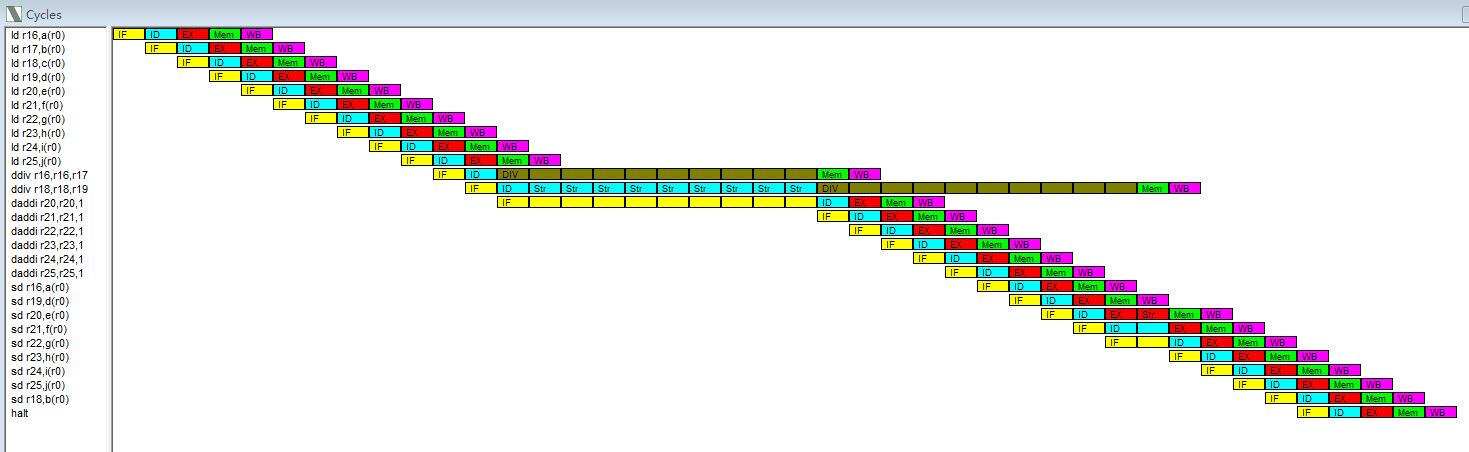
sd r23, h(r0)

sd r24, i(r0)

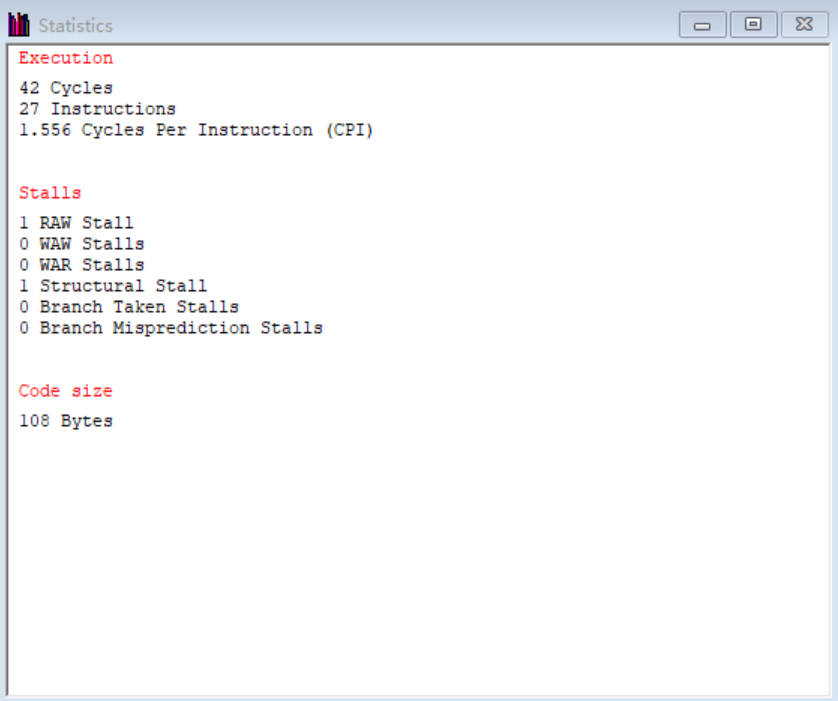
sd r25, j(r0)

halt

上面的指令运行，在***Cycle***窗口结果如下（程序运行前请将configure->architecture->division latency改为10）：



在***Statistics***窗口的结果如下：

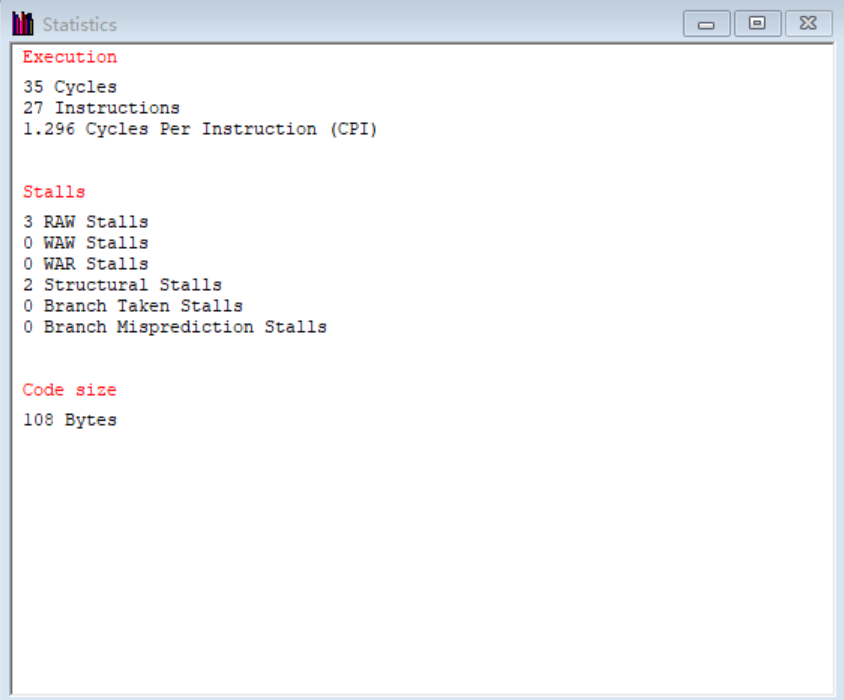


通过观察，我们可以发现，两个连续的除法产生了明显的结构相关，第二个除法为了等待上一个除法指令在执行阶段所占用的资源，阻塞了9个周期。

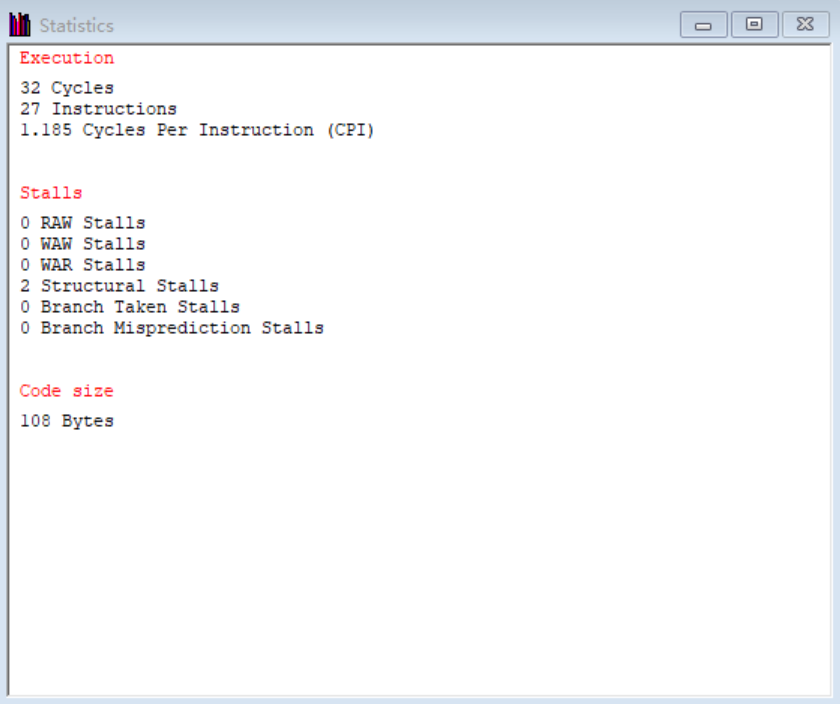
显然，这样的连续的除法所导致的结构相关极大的降低了流水线效率，为了消除结构相关，我们需要做的是调整指令序列，将其他无关的指令塞入两条连续的除法指令中。

给出指令序列的调整方案并给出流水线工作状态的截图，做出解释。

（1）将ddiv r16, r16, r17前移至r16、r17数据刚好读完，即ld r18, c(r0)后，此时可以在避免r16、r17数据冒险的情况下最早开始运行除法，然后需要将ddiv r18, r18, r19放在上一个除法之后的第10条指令，即相隔9条指令，此时运行的statistics如下，可以发现运行时间减少了，但此时数据冒险增加了



（2）分析数据冒险增加的原因可以发现，是因为第二个除法结束前执行了sd r18, c(r0)，此时结果未计算完成，r18无法使用，发生数据冒险，因此可以将sd指令后移，结果如下所示，此时没有数据冒险，并且运行所需时间较之前也更少了。



1. 提交报告

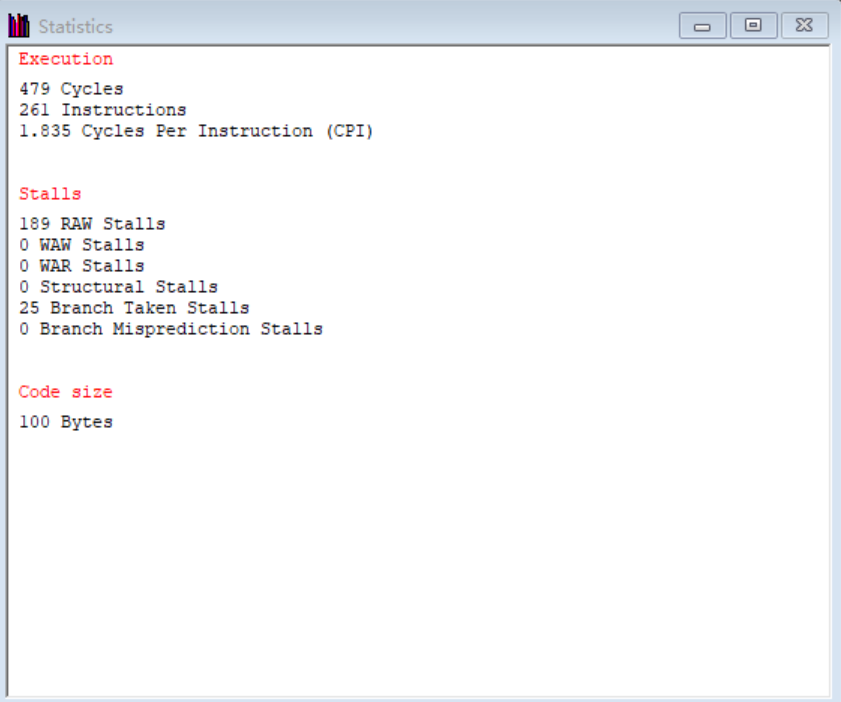
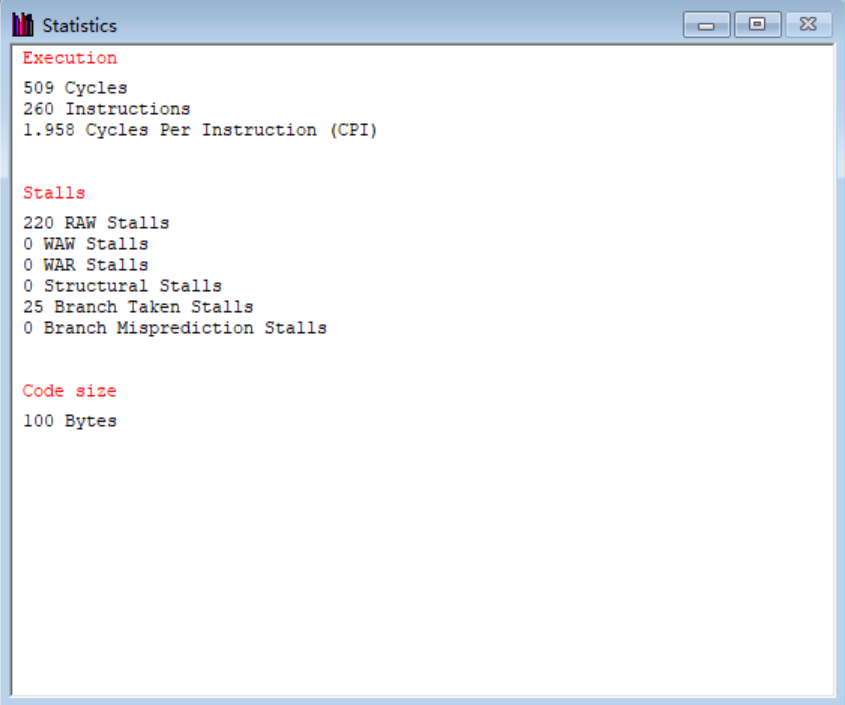
记录实验过程，保存实验截图，给出分析结果，形成实验报告。初始代码准备（10分），后面每个优化方法各30分。

**五、实验结果**

本次实验主要包含三个方面，分别是调整指令序列、开启Enable Forwarding以及结构优化

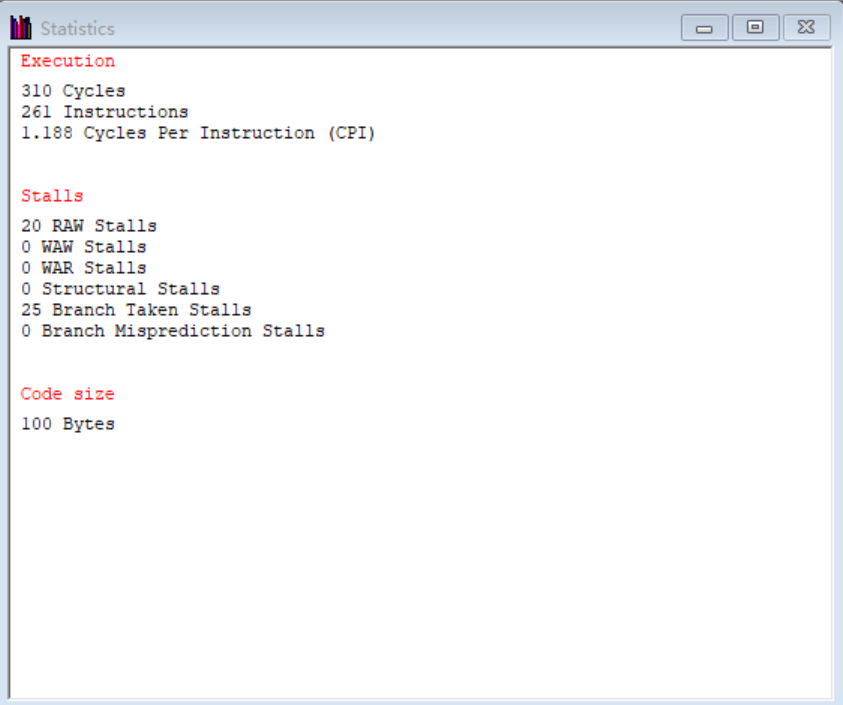
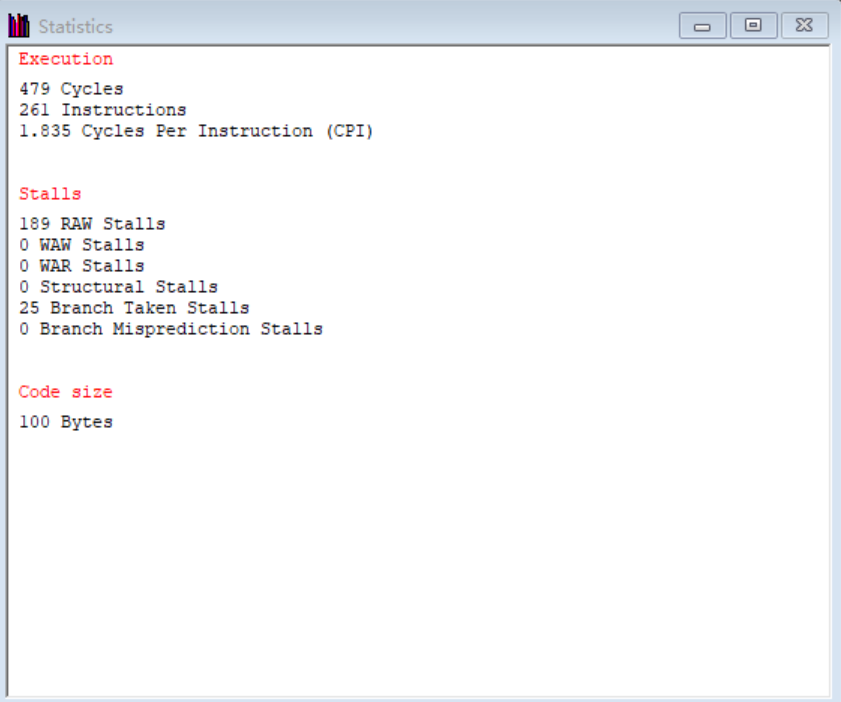
1.指令序列

通过调整指令的序列可以减少数据冒险，即RAW的次数，在本次实验中将初始的220次RAW优化至189次，减少31次数据冒险。



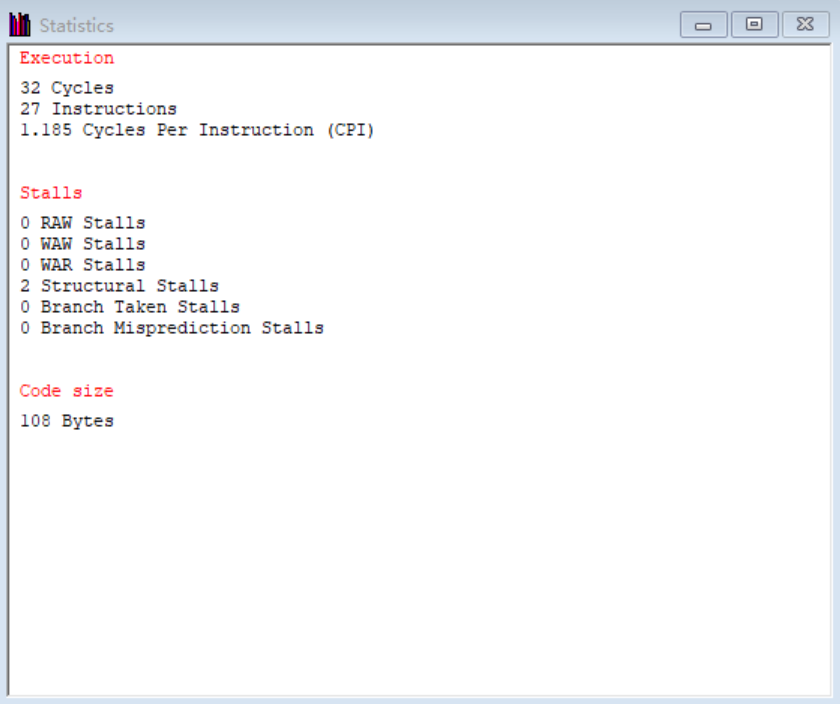
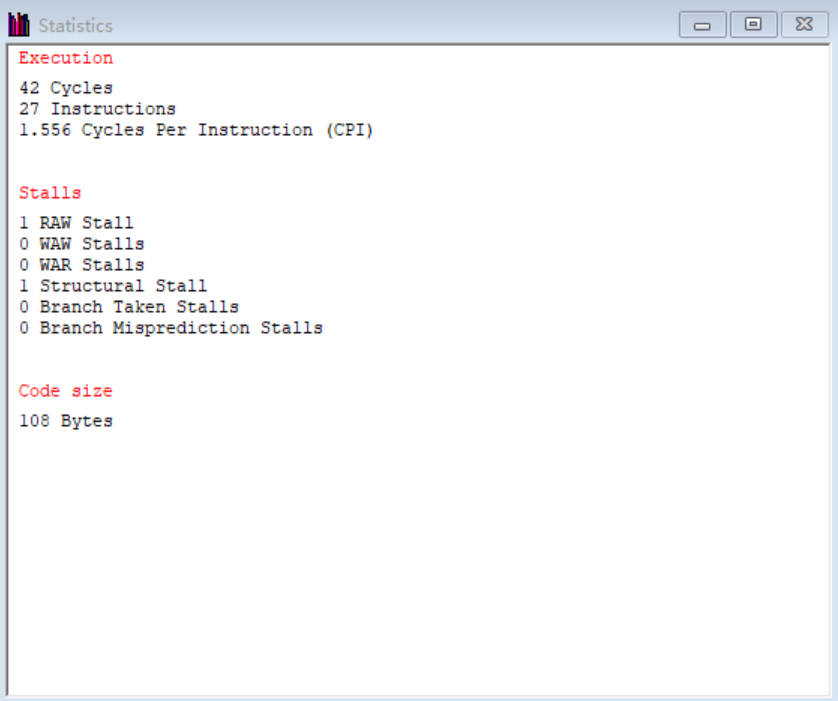
2.开启Enable Forwarding

未开启时RAW的次数为189次，即上方调整指令序列后的次数。开启之后RAW的次数为20次，大幅减少了数据冒险，主要解决了连续调用同一个寄存器的数据冒险



3.结构优化

初始代码连续进行了两次除法，此时时钟周期为42个，调整除法指令位置以及存储的指令位置后时钟周期为32个减少了约1/4。



**六、实验总结与体会**

通过本次实验，基本了解了流水线冒险的部分原因，也更加深入的了解到处理器内部的流水线情况，学习了如何优化汇编代码来提高运行效率。通过对每条指令取指-译码等操作过程可以了解到指令之间的冲突关系，在出现数据冒险时可以适当调整指令的序列，避免数据相关。在出现结构问题时也可以通过调整指令序列来减少多条指令在同一时钟周期内争用同一功能部件。实验还使用了Forwarding功能，该功能减少数据相关引起的流水线停顿，从而提高性能。

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字：  年 月 日 |
| 备注： |