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：软件工程（腾班）

指 导 教 师：王 毅

报告人：黄亮铭　学号：2022155028　班级：腾班

实 验 时 间：2024 年 11 月 8 日

实验报告提交时间：2024 年 11 月 29 日

教务处制

一、实验目标：

1. 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
2. 通过指令顺序调整，或旁路与预测技术来提高流水线效率
3. 更加了解流水线细节和其指令的改善方法
4. 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(inti = 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。

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. 将上述代码保存到 matrix.s 文件（与 asm.exe 在同一个文件夹）中，使用 asm.exe 检查语法，发现报错。定位到错误的行后，发现是标记的冒号和指令之间没有空

格，我选择在标记的冒号和指令之间添加换行。再次运行代码代码没有错误。

```
E:\winmips64>asm.exe matrix.s
Pass 1 - Error on line 10
Pass 1 - Error on line 37
Pass 1 completed with 2 errors

E:\winmips64>asm.exe matrix.s
Pass 1 - Error on line 38
Pass 1 completed with 1 errors
```

```
Pass 2 completed with 0 errors
Code Symbol Table
                start = 00000000
                loop1 = 00000014
                loop2 = 00000020
                exit2 = 00000058
                exit1 = 00000060
Data Symbol Table
                    a = 00000000
                    b = 00000080
                    c = 00000100
                  len = 00000180
              control = 00000188
                 data = 00000190
```

图 1 检查结果（左：修改前 右：修改后）

2. 将 matrix.s 加载到 winmips64 中，同时关闭 forwarding 功能。

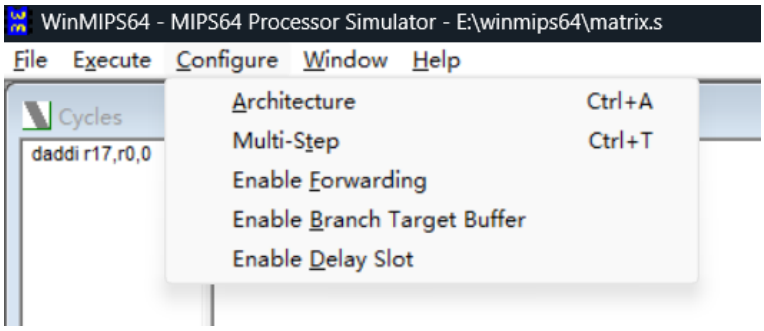


图 2 加载文件并关闭 forwarding 功能

3. 执行程序，结果如下图所示。发现有 220RAW，C 矩阵的元素均为 5，结果与预期相符。

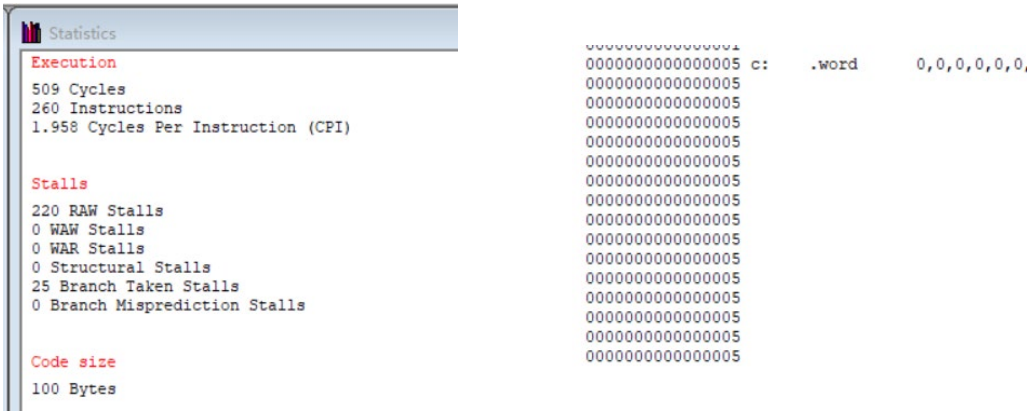


图 3 运行结果

4. 修改程序办法：1）单步执行程序，程序在哪个位置有指令阻塞则说明附近发生数据冒险，将其修改即可。2）观察代码结构，将可能发生数据冒险的指令推后或把其他代码推前。
   I. 通过单步执行程序发现 r16 需要在写回之后才能被下一条指令使用，发生了数据冒险。

```
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
```

```
ld r16,len(r0)
daddi r17,r0,0
daddi r21,r0,a
daddi r22,r0,b
daddi r23,r0,c
loop1:
slt r8,r17,r16
```

图 4 修改位置（左：修改前 右：修改后）

II. 通过单步执行程序发现在 loop1 的判断循环是否结束时 r8 也发生了数据冒险，原因和 I 相同。

```
daddi r23,r0,c
loop1:
slt r8,r17,r16
beq r8,r0,exit1
daddi r19,r0,0
```

```
loop1:
slt r8,r17,r16
daddi r19,r0,0
beq r8,r0,exit1
```

图 5 修改位置（左：修改前 右：修改后）

III. 通过单步执行程序发现在 loop2 的判断循环是否结束时 r8 也发生了数据冒险，原因和 II 相同。但是后一条指令也使用到 r8，因此无法简单地将后一条指令推前。但是经过分析我们可以将 loop2 的累加器即 r19 的累加指令推前。

```
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
```

```
loop2:
slt r8,r19,r16
daddi r19,r19,1
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)

j loop2
exit2:
daddi r17,r17,1
```

图 5 修改位置（左：修改前 右：修改后）

IV. 在 loop2 的计算基址偏移时 r8 也发生了数据冒险，但是后面的指令均依赖于前一条指令，因此无法优化。

V. 在 loop2 中获取加数和结果的地址的时候也发生了数据冒险，我们可以将不冲突的关于 r11 的指令推后就可以解决此问题。但是这样带来新的问题：关于 r12 的存储发生了冲突。综上所述，修改后只有一条指令发生数据冒险，修改

前有两条指令发生数据冒险，因此这个修改是值得的。

```
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)
```

```
dadd r9,r8,r21
dadd r10,r8,r22
ld r9,0(r9)
ld r10,0(r10)
dadd r11,r8,r23
dadd r12,r9,r10
sd r12,0(r11)
```

图 6 修改位置（左：修改前 右：修改后）

5. 将优化后的代码保存，使用 asm.exe 检查语法错误。

```
Pass 2 completed with 0 errors
Code Symbol Table
                start = 00000000
                loop1 = 00000014
                loop2 = 00000020
                exit2 = 00000058
                exit1 = 00000060
Data Symbol Table
                    a = 00000000
                    b = 00000080
                    c = 00000100
                  len = 00000180
              control = 00000188
                 data = 00000190
```

图 7 检查结果

6. 将 matrix.s 加载到 winmips64 中，同时关闭 forwarding 功功能。然后执行程序，结果如下图所示。发现只有 185RAW，C 矩阵的元素均为 5，结果与预期相符。

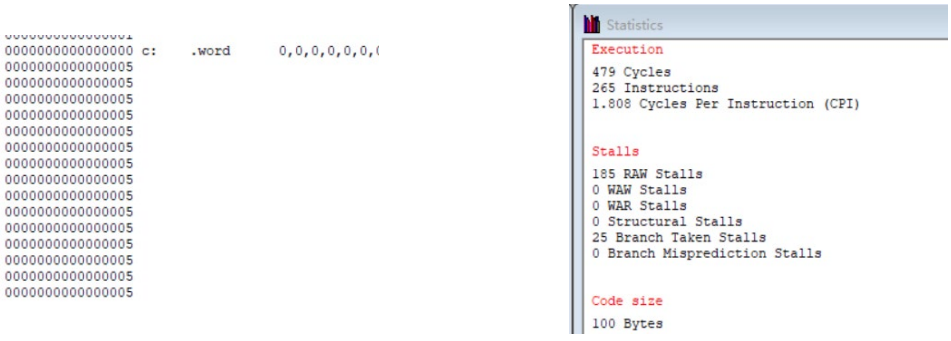


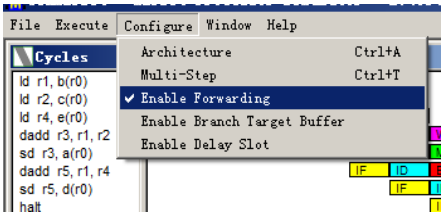
图 8 运行结果

7. 优化前有 220RAW，优化后有 185RAW。两者相差 35RAW，说明优化有一定的效果。

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。

1. 结果对比。发现 RAW 下降到 0 次。

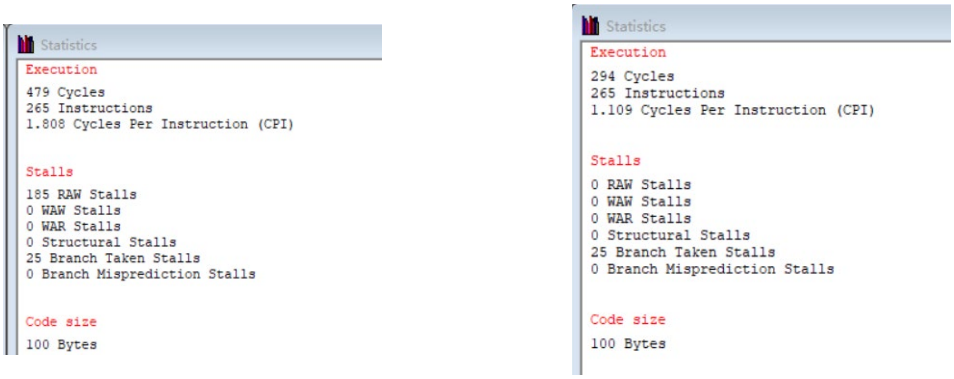


图 9 结果对比（左：开启前 右：开启后）

2. 分析原因。

   修改 III、IV 和 V 中所述的指令存在的数据冒险在开启 forwarding 功能后得到解决，因此 RAW 下降到 0 次。

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
```

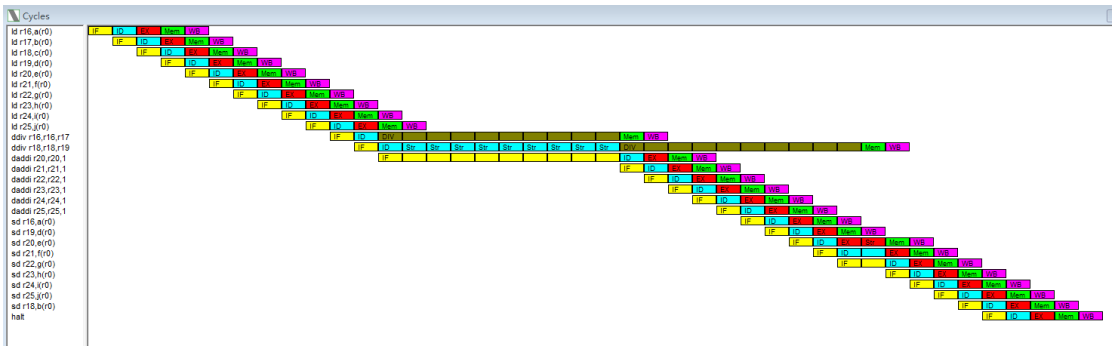
根据上述的 C 代码，我们给出数据相关优化的指令如下：

```
        .data
```

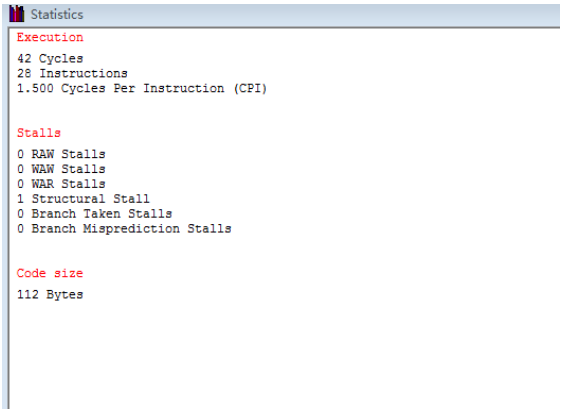
```
    a:    .word    12
    b:    .word    3
    c:    .word    15
    d:    .word    5
    e:    .word    1
    f:    .word    2
    g:    .word    3
    h:    .word    4
    i:    .word    5

          .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)
    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
    st r16,a(r0)
    st r17,b(r0)
    st r18,c(r0)
    st r19,d(r0)
    st r20,e(r0)
    st r21,f(r0)
    st r22,g(r0)
    st r23,h(r0)
    st r24,i(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：

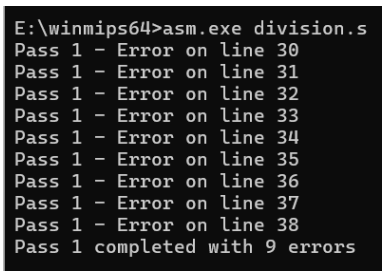


通过观察，我们可以发现，两个连续的除法产生了明显的结构相关，第二个除法为了等待上一个除法指令在执行阶段所占用的资源，阻塞了 9 个周期。

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. 将上述代码保存到 division.s 文件中，运行 asm.exe，发现报错。根据报错提示的行数，依次排查，发现错误：**st 不是 mips 的指令，将其修改为 sd 即可**。将修改后的代码保存，运行 asm.exe，语法正确。



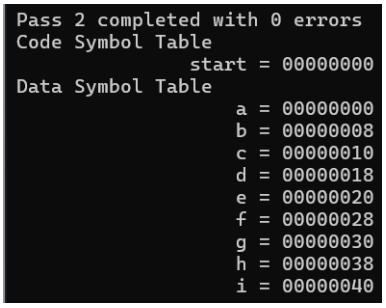


图 10 代码修改前后语法检查对比（左：修改前 右：修改后）

2. 观察上面给出的 cycles 窗口，观察到一条除法指令会占用除法组件 **10 个周期**。因此，我们需要在两条除法指令之间插入 9 条无关的指令。通过前面的分析，我们现在的任务是将**第一条除法指令尽可能地提前**，将**第二条除法指令尽可能地推后**。但是，**1）第一条指令需要在 r16、r17 的取数指令完成后才能执行。2）将 r16、r17 中的数值写入内存的指令需要推后，因为第二条除法指令也会使用 10 个周期进行计算**。

   修改前和修改后的代码如下图所示。

```
12  .text
13  start:
14  ld r16,a(r0)
15  ld r17,b(r0)
16  ld r18,c(r0)
17  ld r19,d(r0)
18  ld r20,e(r0)
19  ld r21,f(r0)
20  ld r22,g(r0)
21  ld r23,h(r0)
22  ld r24,i(r0)
23  ddiv r16,r16,r17
24  ddiv r18,r18,r19
25  daddi r20,r20,1
26  daddi r21,r21,1
27  daddi r22,r22,1
28  daddi r23,r23,1
29  daddi r24,r24,1
30  sd r16,a(r0)
31  sd r17,b(r0)
32  sd r18,c(r0)
33  sd r19,d(r0)
34  sd r20,e(r0)
35  sd r21,f(r0)
36  sd r22,g(r0)
37  sd r23,h(r0)
38  sd r24,i(r0)
39  halt
40
```

```
12  .text
13  start:
14  ld r16,a(r0)
15  ld r17,b(r0)
16  ld r18,c(r0)
17  ddiv r16,r16,r17
18  ld r19,d(r0)
19  ld r20,e(r0)
20  ld r21,f(r0)
21  ld r22,g(r0)
22  ld r23,h(r0)
23  ld r24,i(r0)
24  daddi r20,r20,1
25  daddi r21,r21,1
26  ddiv r18,r18,r19
27  daddi r22,r22,1
28  daddi r23,r23,1
29  daddi r24,r24,1
30  sd r18,c(r0)
31  sd r19,d(r0)
32  sd r20,e(r0)
33  sd r21,f(r0)
34  sd r22,g(r0)
35  sd r23,h(r0)
36  sd r24,i(r0)
37  sd r16,a(r0)
38  sd r17,b(r0)
39  halt
40
```

图 10 代码修改前后对比（左：修改前 右：修改后）

3. 运行程序，流水线结果截图如下图所示。发现程序所需时间由原本的 42 周期下降为 37 周期，效率有比较明显的提升。

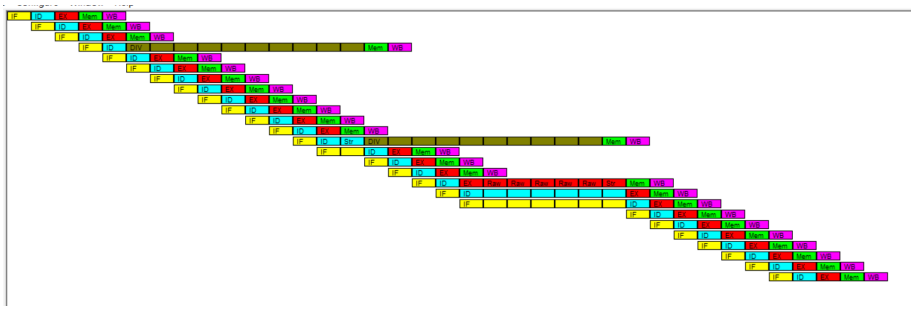


图 11 流水线

```
Execution

37 Cycles
26 Instructions
1.423 Cycles Per Instruction (CPI)


Stalls

5 RAW Stalls
0 WAW Stalls
0 WAR Stalls
1 Structural Stall
0 Branch Taken Stalls
0 Branch Misprediction Stalls


Code size

104 Bytes
```

图 12 Statics 窗口

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

调整指令序列

调整指令序列之前为 220RAW，调整之后只有 185RAW，两者相差 35RAW，说明调整指令序列的优化有效。

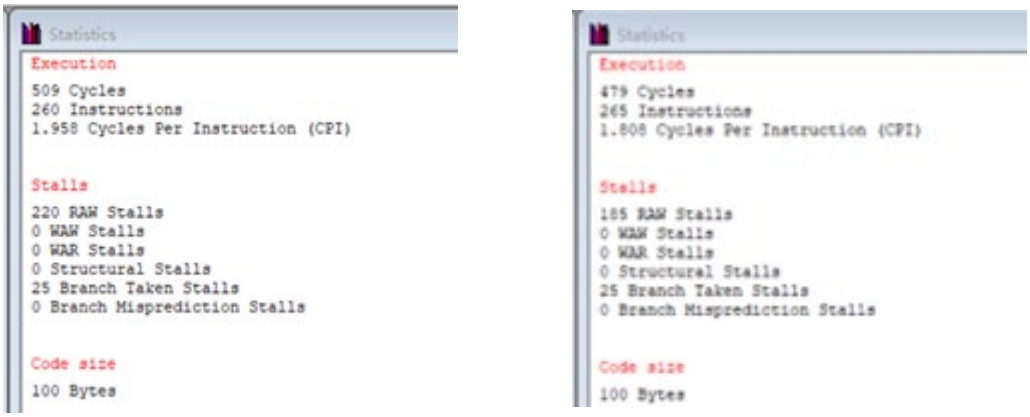


图 13 调整结果

Forwarding 功能开启

开启 forwarding 功能前为 185RAW，开启之后为 0RAW，说明 forwarding 功能的优化效果十分显著。

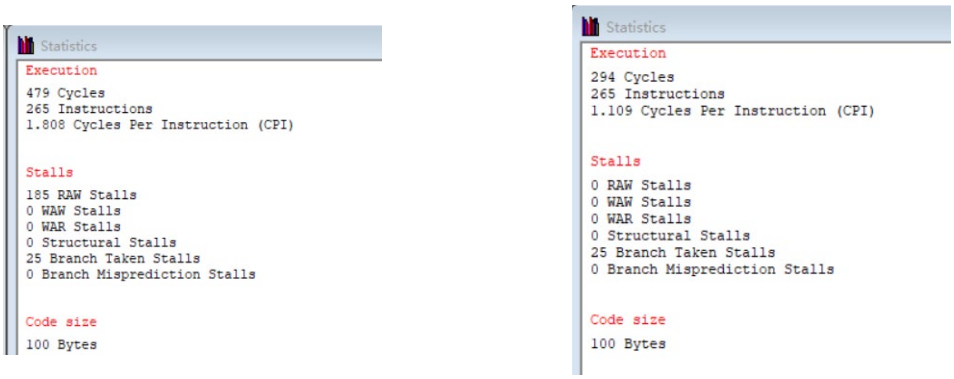


图 14 结果对比（左：开启前 右：开启后）

结构相关优化

优化前程序执行需要 42 个时钟周期，而优化程序之后只需要 37 个时钟周期。说明结构优化有效。

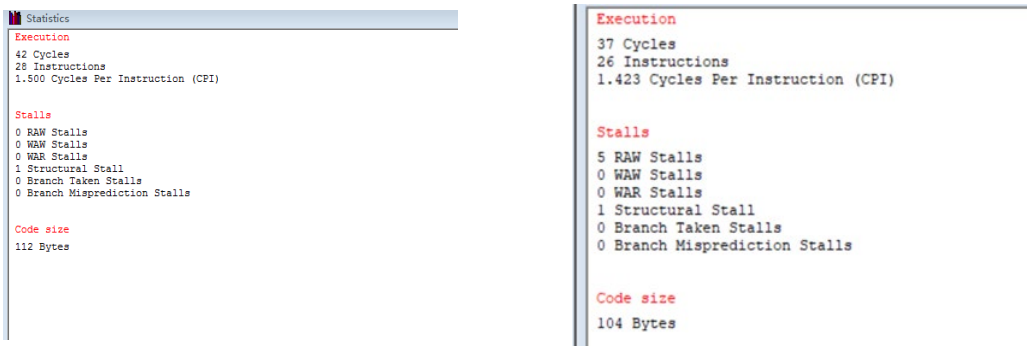


图 14 结果对比（左：优化前 右：优化后）

五、实验总结与体会

- 通过本次实验，我了解了 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。
- 通过本次实验，我了解了如何通过指令顺序调整、旁路与预测技术来提高流水线效率。
- 通过本次实验，我了解流水线细节和其指令的改善方法。
- 通过本次实验，我进一步熟悉了 winmips64 的使用方法，对 mips 有了更加深刻的理解。

指导教师批阅意见： **成绩评定：** 指导教师签字：王毅 2024 年 12 月 12 日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。