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：罗秋明

报告人：林浩晟　学号：2022280310　班级：01

实 验 时　间：2024 年 11 月 1 日

实验报告提交时间：2024 年 11 月 7 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

1. 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。
2. 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。
3. 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。

一、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。

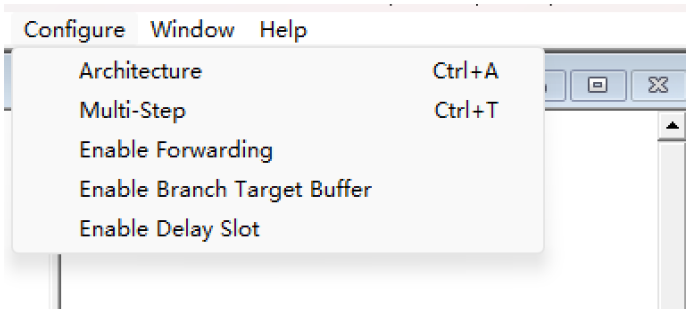
通过**调整序列**来规避这个数据相关，在 statics 窗口中记录其效果。将此结果与初始的结果进行对比，报告 **RAW 相关的次数减少**的数量。

1．先将上面的代码存入 exp3.s 文件中，再用 asm.exe 对其进行检查，如下；

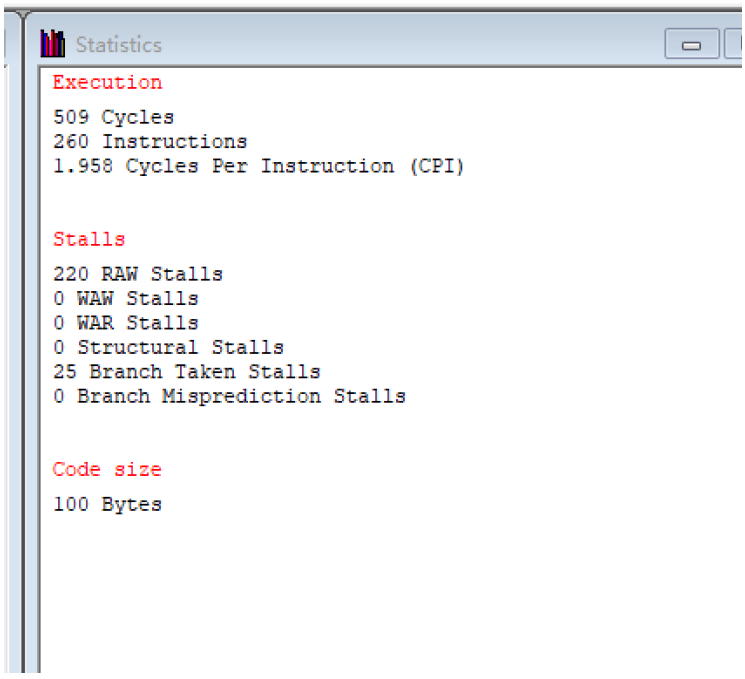
```
PS D:\WinMips\winmips64> ./asm.exe exp3.s
Pass 1 completed with 0 errors
00000000          .data
00000000 0000000000000001 a:    .word      1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4
         0000000000000001
         0000000000000001
         0000000000000001
         0000000000000002
         0000000000000002
```

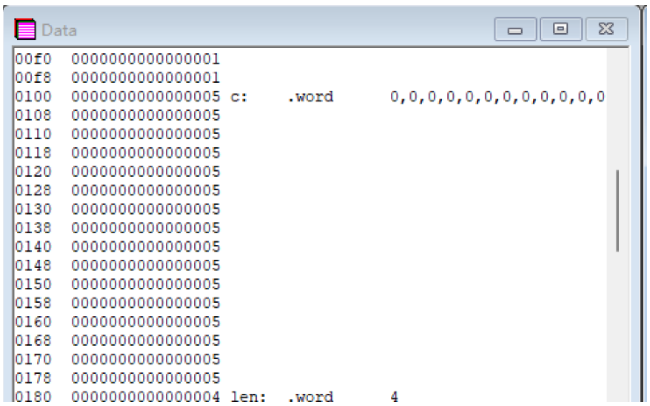
```
Pass 2 completed with 0 errors
Code Symbol Table
               start = 00000000
               loop1 = 00000014
               loop2 = 00000020
               exit2 = 00000058
               exit1 = 00000060
Data Symbol Table
                   a = 00000000
                   b = 00000080
     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

2. 将文件载入到 WinMIPS64 中，并关闭 Configure – Enable Forwarding；



3. 再运行程序，可以发现’Stalls’中有 220 RAW，同时 c 矩阵的元素都为 5，结果正确；





4. 现优化代码以此来减少 RAW 相关的次数减少，此处结合修改前图片（左侧）和修改后图片（右侧）加以说明；

①首先将 ld 指令提前，以此避免数据冲突；

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

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

② beq 指令读取 r8 时发生数据冒险，需要其滞后；

```
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
```

③将指令“daddi r19, r19, 1”前移到 beq 前面，此时开启前推后就可以防止堵塞；将指令“dadd r11, r8, r23”移至“dadd r12, r9, r10”前一条，滞后不冲突的指令，防止阻塞，前推时解除 r12 的数据冒险；代码前后对比图如下：

```
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
```

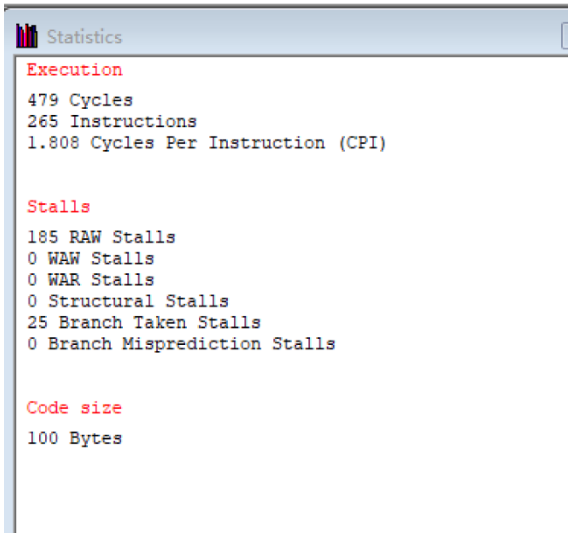
```
loop2: slt r8, r19, r16
       daddi r19, r19, 1
       beq r8, r0, exit2

       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
```

5. 重新保存代码并用 asm.exe 检查，可以发现代码正确；

```
PS D:\WinMips\winmips64> ./asm.exe exp3.s
Pass 1 completed with 0 errors
00000000          .data
00000000 0000000000000001 a:    .word      1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4
         0000000000000001
```

```
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
```

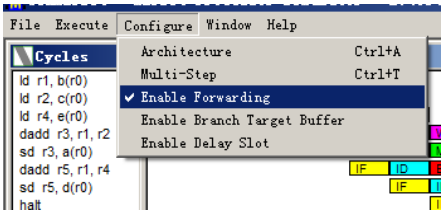
6. 载入 WinMips 中运行程序，得到以下结果；发现在关闭前推的条件下运行，发生了 185 次 RAW，较优化前减少了 35 次；可以证明优化成功。



二、Forwarding 功能开启

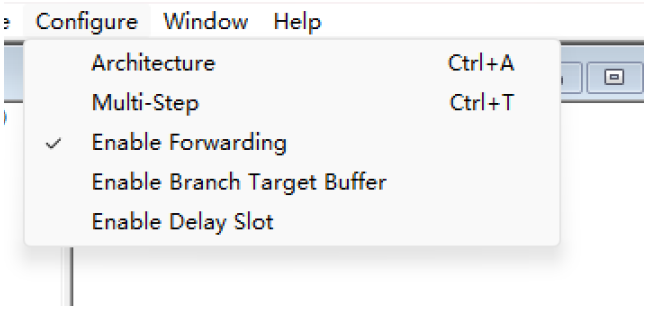
接下来，我们要展示 Forwarding 功能的优化效果。

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。

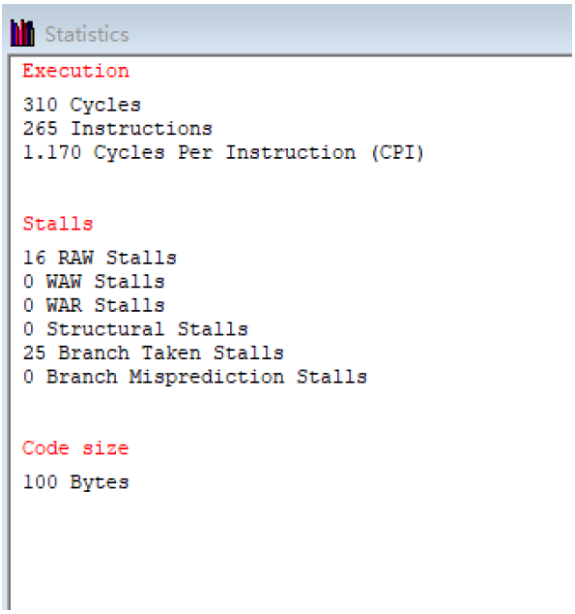


开启了 Forwarding 功能之后，我们再运行，查看结果，解释哪些数据相关的问题得到解决，并以截图说明问题解决前后的差异所在。

1. 首先开启 Enable Forwarding；



2. 再次载入 WinMips 中运行程序，得到以下结果；发现只进行了 16 次的 RAW；



3.开启前推后，可以解决优化代码分析第③点的问题；开启前推后，RAW 显著减少，从 185 次减少到了 16 次，差异说明了前推技术在减少流水线冒险、提高流水线效率方面的显著效果。

三、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：

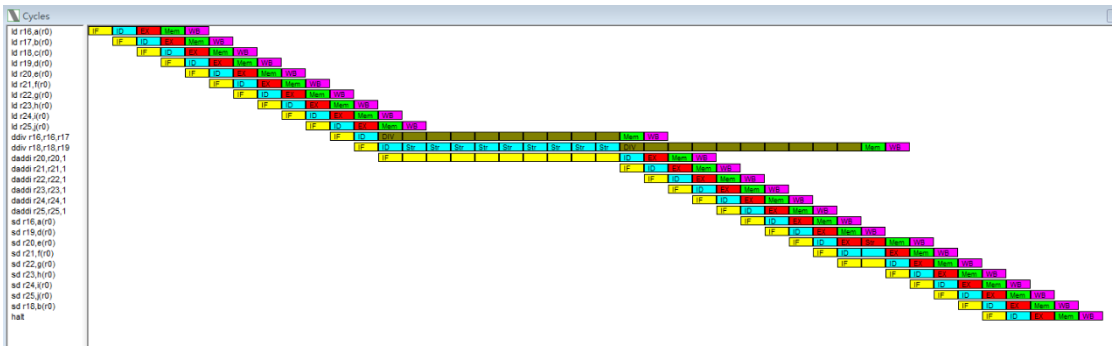
```
a = a / b
c = c / d
e = e + 1
f = f + 1
g = g + 1
h = h + 1
i = i + 1
j = j + 1
```

根据上述的 C 代码，我们给出数据相关优化的指令如下：

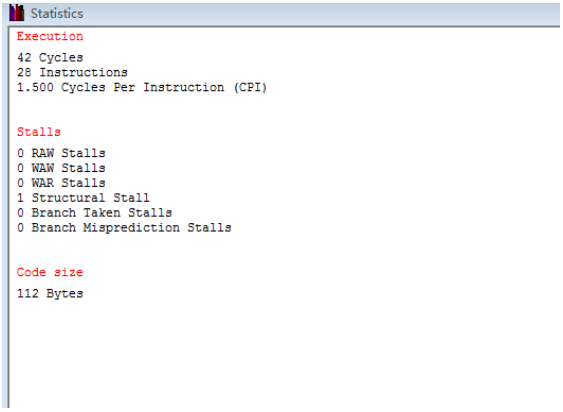
```
        .data
a:      .word       12
b:      .word       3
c:      .word       15
d:      .word       5
e:      .word       1
```

```
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
      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. 题中给出的代码不完整，少了参数 j，并且没有进行存储操作；以下为完整代码：

```
.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
```

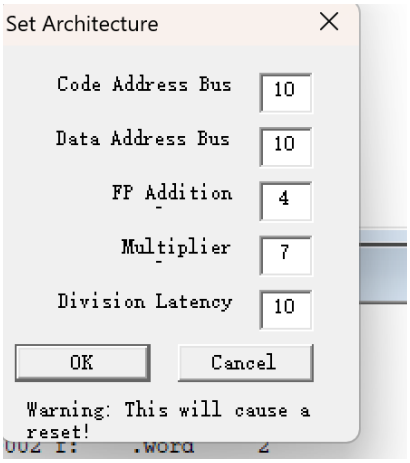
```
    daddi r22, r22, 1
    daddi r23, r23, 1
    daddi r24, r24, 1
    daddi r25, r25, 1
    sd r16, a(r0)
    sd r18, c(r0)
    sd r20, e(r0)
    sd r21, f(r0)
    sd r22, g(r0)
    sd r23, h(r0)
    sd r24, i(r0)
    sd r25, j(r0)
    halt
```

2. 用 asm.exe 进行检查，发现程序程序没有错误；

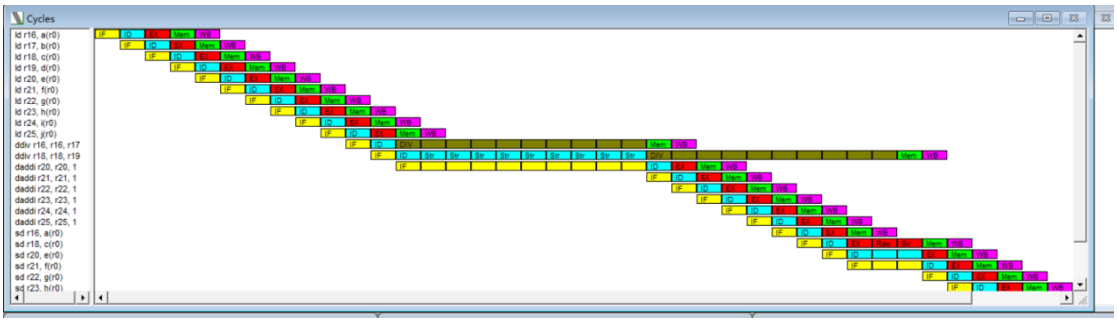
```
PS D:\WinMips\winmips64> ./asm.exe div.s
Pass 1 completed with 0 errors
00000000           .data
00000000 000000000000000c a: .word 12
00000008 0000000000000003 b: .word 3
00000010 000000000000000f c: .word 15
00000018 0000000000000005 d: .word 5
00000020 0000000000000001 e: .word 1
00000028 0000000000000002 f: .word 2
00000030 0000000000000003 g: .word 3
00000038 0000000000000004 h: .word 4
00000040 0000000000000005 i: .word 5
00000048 0000000000000006 j: .word 6
```

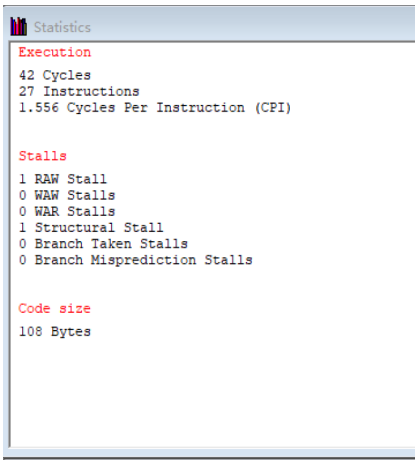
```
Pass 2 completed with 0 errors
Code Symbol Table
                start = 00000000
Data Symbol Table
                    a = 00000000
                    b = 00000008
                    c = 00000010
                    d = 00000018
                    e = 00000020
                    f = 00000028
                    g = 00000030
                    h = 00000038
                    i = 00000040
                    j = 00000048
```

3. 载入 WinMips，将 division latency 改为 10；



4. 运行程序得到以下结果，可以证明过程正确；





5. 为了避免流水线中的阻塞现象，我们需要在两个除法操作之间插入 9 条不相关的指令。这样做虽然会导致第一个除法指令后的第 9 条指令出现结构冲突，但由于后续的除法操作不会立即需要使用存储组件，这种结构冲突的影响可以忽略不计。然而，需要注意的是，第二个除法指令之后的第 9 条指令也会出现结构冲突，这可能会导致流水线的阻塞。

对于第一个除法指令，因为它需要使用寄存器 r16 和 r17，如果在取指令后立即执行除法操作，就会发生取指令和使用之间的冒险，这可能会导致流水线阻塞。因此，我们需要在取指令之后插入一些不会引起冲突的指令来避免这种阻塞。另外，由于除法操作需要将结果写回到内存，我们也需要相应地延迟存储指令的执行。

具体的优化措施如下：

将第一个除法指令提前到取指令完成后的第二条指令位置。

将第二个除法指令移动到第一个除法指令之后的第九条指令位置。

对应地，将存储指令也进行延迟。

通过这样的调整，我们得到了优化后的代码，既减少了流水线冒险，又提高了执行效率。

优化的代码如下：

```
.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)
    ddiv r16, r16, r17

    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)

    daddi r20, r20, 1
    daddi r21, r21, 1
    ddiv r18, r18, r19
    daddi r22, r22, 1
    daddi r23, r23, 1
    daddi r24, r24, 1
    daddi r25, r25, 1

    sd r20, e(r0)
    sd r21, f(r0)
    sd r22, g(r0)
    sd r23, h(r0)
    sd r24, i(r0)
    sd r25, j(r0)
    sd r16, a(r0)
```

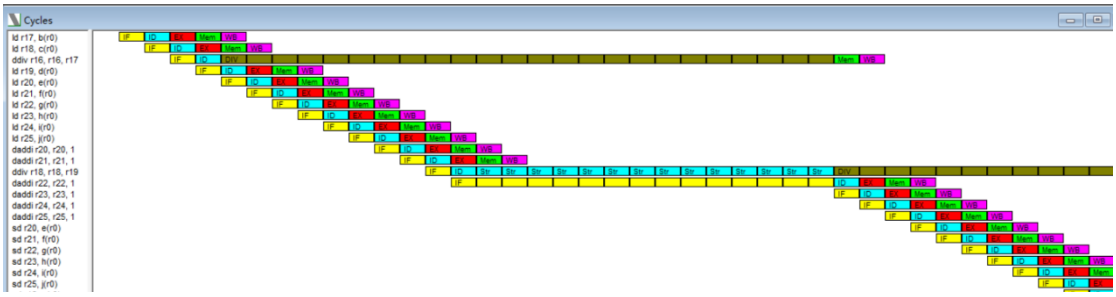
```
    sd r18, c(r0)


    halt
```

6. 使用 asm.exe 对优化后的代码进行检查，发现没有错误；

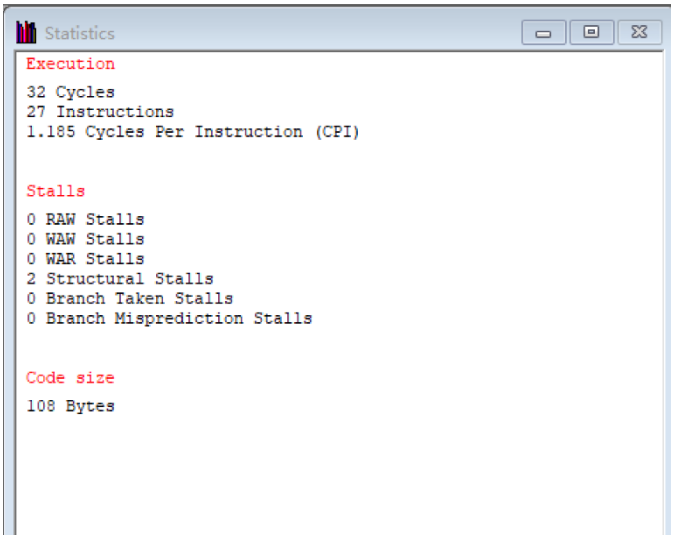
```
PS D:\WinMips\winmips64> ./asm.exe div1.s
Pass 1 completed with 0 errors
00000000           .data
00000000 000000000000000c a: .word 12
00000008 0000000000000003 b: .word 3
00000010 000000000000000f c: .word 15
00000018 0000000000000005 d: .word 5
00000020 0000000000000001 e: .word 1
00000028 0000000000000002 f: .word 2
00000030 0000000000000003 g: .word 3
00000038 0000000000000004 h: .word 4
00000040 0000000000000005 i: .word 5
00000048 0000000000000006 j: .word 6
```

```
Pass 2 completed with 0 errors
Code Symbol Table
                start = 00000000
Data Symbol Table
                    a = 00000000
                    b = 00000008
                    c = 00000010
                    d = 00000018
                    e = 00000020
                    f = 00000028
                    g = 00000030
                    h = 00000038
                    i = 00000040
                    j = 00000048
```

7. 载入 WinMips，开启前推的情况下运行代码，得到如下结果，观察到两条除法指令的流水之间插入了其他指令，由此大大提升效率；



8. 时钟周期数如下，可以发现从 42 优化到了 32；



四、提交报告

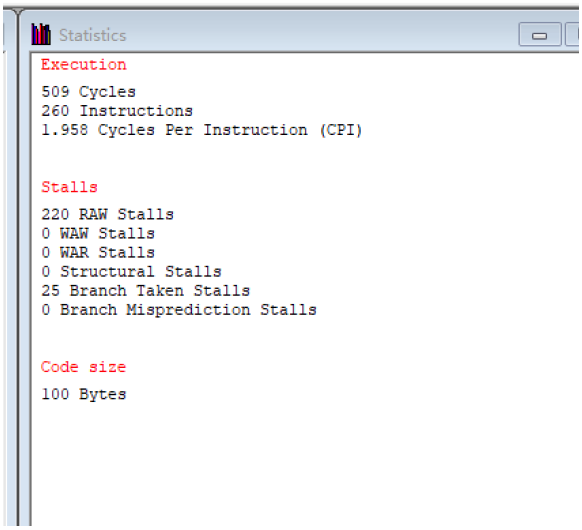
记录实验过程，保存实验截图，给出分析结果，形成实验报告。初始代码准备（10分），后面每个优化方法各 30 分。

五、实验结果

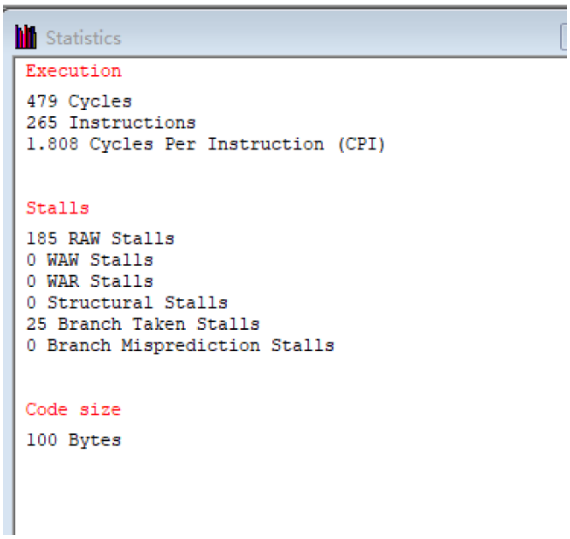
一、调整指令序列

经过优化代码，RAW 次数从 220 次减少到了 185 次；

优化前



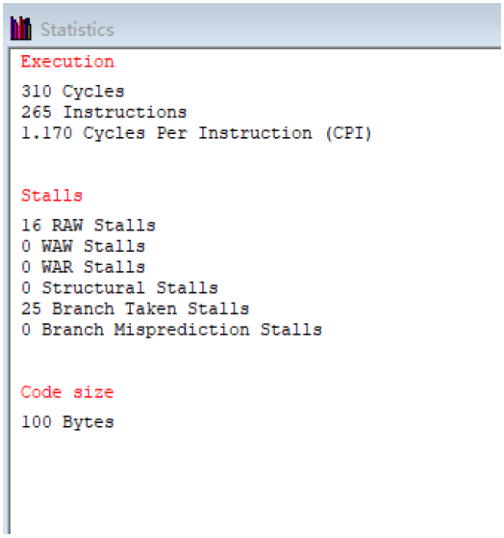
优化后



二、Forwarding 功能开启

开启前推后，优化后的代码的 RAW 次数从 185 次变成了 16 次，说明了前推技术在减少流水线冒险、提高流水线效率方面的重要性；

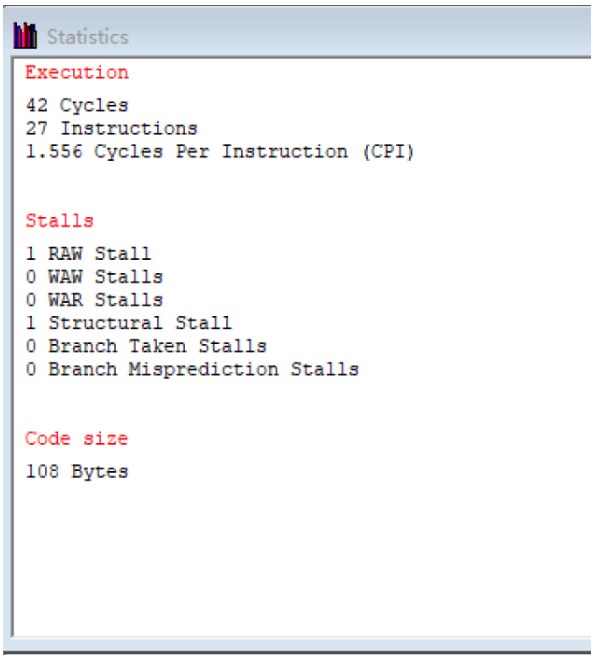
开启前推后



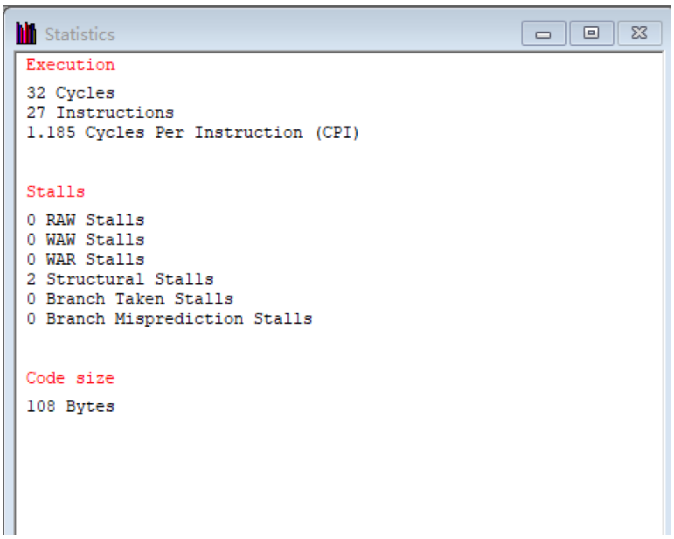
三、结构相关优化

经过代码的优化，时间周期数从 42 优化到了 32。

优化前



优化后



六、实验总结与体会

实验成功完成了三个部分。

这次的处理器结构实验让我深刻理解了 MIPS 流水线架构以及如何处理流水线中的冒险问题。通过分析和修改 MIPS 汇编代码，我获得了对现代处理器工作机制的深刻见解，并学习了提升流水线效率的方法。

首先我对提供的代码进行了深入分析，发现其中存在数据依赖，这导致了流水线冒险。通过执行代码并观察流水线冒险的发生，我能够明确地识别出数据依赖和控制依赖等不同流水线阶段的冒险，这帮助我理解了流水线冒险如何影响性能。

然后我调整了代码中的指令序列，以消除数据依赖，有效地减少了流水线冒险。在这个过程中，我掌握了如何分析和优化指令序列，以最小化性能损失。

最后我引入了 forwarding 机制，这是一种硬件优化技术，可以在存在数据依赖时提升性能。通过实施 forwarding，我减少了因数据依赖导致的流水线停滞，从而提升了性能。这个实验让我对现代处理器内部的优化技术有了更深入的认识，这些技术对于提高效率至关重要。

在整个实验过程中，我积累了丰富的 MIPS 指令集和流水线结构知识。我学会了如何分析和优化代码，以减少流水线冒险，提升性能。同时，我也更加熟练地掌握了 MIPS 指令，这对于深入理解计算机体系结构非常关键。

总体来看，我不仅深化了对流水线冒险和性能优化的理解，还增强了我在计算机体系结构领域的知识。

指导教师批阅意见： **成绩评定：** 指导教师签字： 年　月　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。