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：林宪亮　学号：2022150130　班级：国际班

实 验 时 间：2024 年 11 月 21 日

实验报告提交时间：2024 年 11 月 21 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。

一、调整指令序列

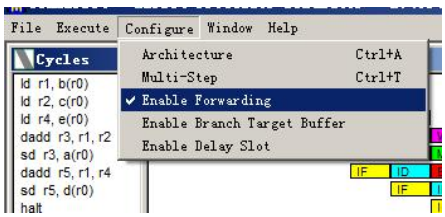
在这一部分，我们利用指令调整的方法对数据相关代码进行优化，规避数据相关。

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。

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。

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：

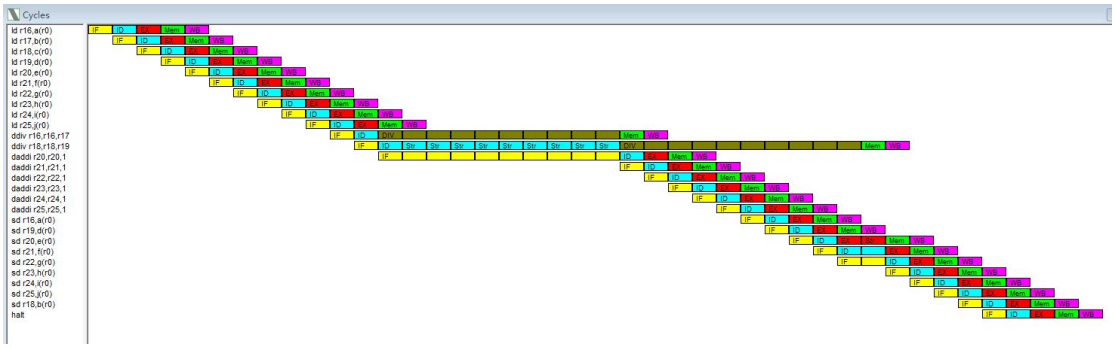
```
a = a / b
c = c / d
e = e + 1
f = f + 1
g = g + 1
h = h + 1
i = i + 1
j = j + 1
```

根据上述的 C 代码，我们给出数据相关优化的指令如下：

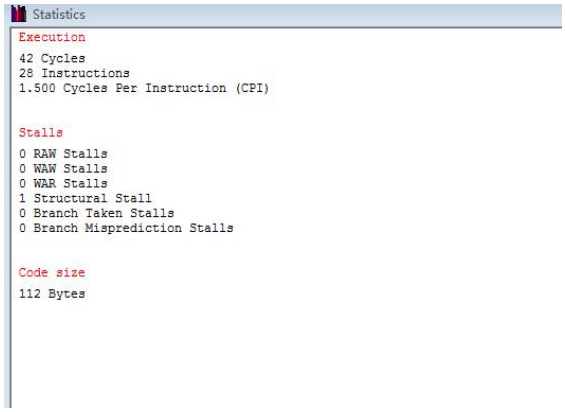
```
        .data
a:      .word     12
b:      .word     3
c:      .word     15
d:      .word     5
e:      .word     1
f:      .word     2
g:      .word     3
h:      .word     4
i:      .word     5

        .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。

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。

1、初始代码准备

在开始实验前，我确保 WinMIPS64 的配置中未选中 “Enable Forwarding” 选项。随后，我会运行一段包含明显数据相关的代码，通过流水线执行过程定位和分析数据相关性引发的延迟，并记录相关统计信息：

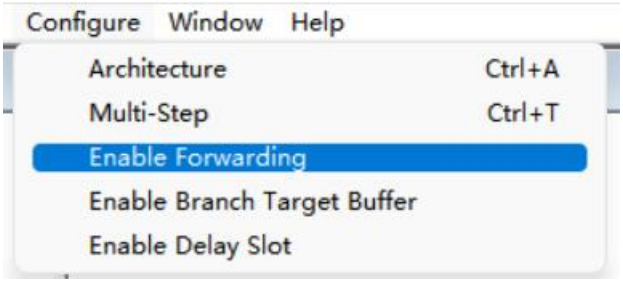


图 1 Enable Forwarding

接下来，我将实验提供的矩阵相加代码加载到 WinMIPS64 中运行，观察流水线执行的结果。在执行过程中，我特别关注指令之间的延迟现象，以分析数据相关性对流水线性能的影响。

运行完成后，我记录了 Statistic 窗口的统计信息，并截图保存，结果如下所示：

```
Execution
509 Cycles
260 Instructions
1.958 Cycles Per Instruction (CPI)

Stalls
220 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
25 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
100 Bytes
```

图 2 统计信息 1

如图可知，共发生 220 次数据相关 RAW Stalls：

通过这些统计信息，我可以清晰地看到因数据相关导致的流水线停顿次数，以及其他性能指标。这些数据为后续优化提供了重要参考。

2、优化 1—调整指令序列

接下来，我将逐个分析产生数据相关的代码，列出具体的相关指令，并对这些指令序列进行调整，以减少数据相关的发生。通过优化指令顺序，我可以减少流水线停顿，提高程序的执行效率。

(1)第一处数据冒险

代码段如下所示，r16 发生了数据相关：

```
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
```

所以可以调整 ld 指令的顺序，调整后的代码段如下所示：

```
start:
      ld r16,len(r0)
      daddi r17,r0,0
      daddi r23,r0,c
loop1: slt r8,r17,r16
      daddi r19,r0,0
      beq r8,r0,exit1
```

改变这条指令的顺序后可以发现减少了两次数据冒险：

```
Execution
507 Cycles
260 Instructions
1.950 Cycles Per Instruction (CPI)

Stalls
218 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
25 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
100 Bytes
```

图 3 统计信息 2

(2)第二处数据冒险

```
          daddi r19,r0,0
    loop2: slt r8,r19,r16
          beq r8,r0,exit2
```

我发现 r19 发生了数据冒险，所以我调整了 daddi 指令的顺序：

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

loop2: slt r8,r19,r16
      daddi r21,r0,a
      daddi r22,r0,b
      beq r8,r0,exit2
```

如下图所示，改变这条指令的顺序后可以发现减少到 205 次次数据冒险：

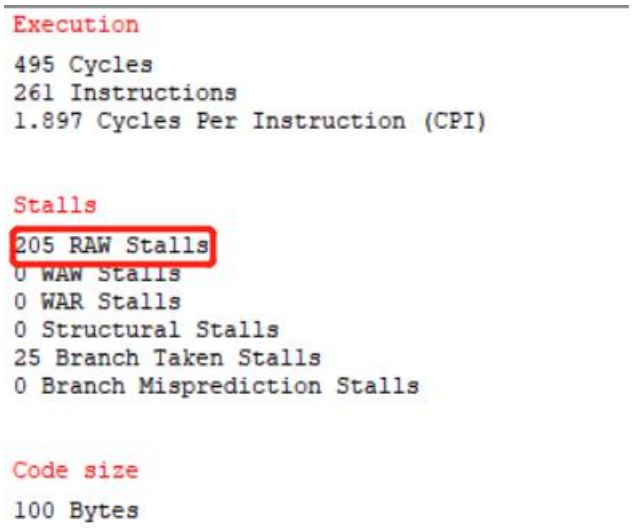


图 4 统计信息 3

(3)第三处数据冒险

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

这里我把上面获取数组首地址的 daddi 指令移下来：

```
loop2: slt r8,r19,r16
       daddi r21,r0,a
       daddi r22,r0,b
       beq r8,r0,exit2
```

可以发现，数据冒险减少为 166 次：

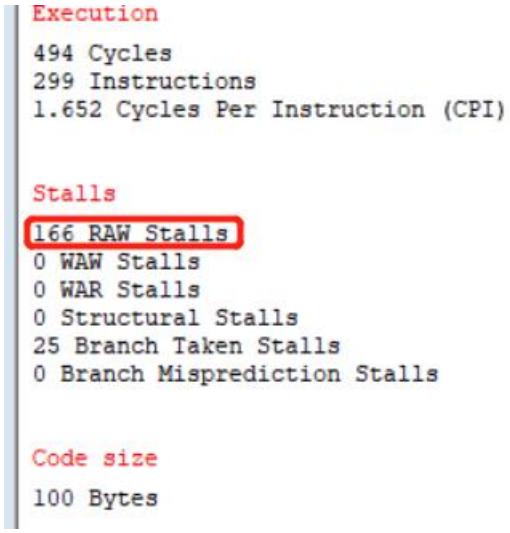


图 5 统计信息 4

(4)其它数据冒险

其实中间还有几次数据冒险，比如这里 r12 发生了数据冒险：

```
ld r9,0(r9)
ld r10,0(r10)
dadd r12,r9,r10
sd r12,0(r11)
```

将下面的计数器加 1 指令搬到 dadd 与 sd 指令中间：

```
ld r9,0(r9)
ld r10,0(r10)
dadd r12,r9,r10
daddi r19,r19,1
sd r12,0(r11)
```

数据冒险减少为 150 次：

```
Stalls
150 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
25 Branch Taken Stalls
0 Branch Misprediction Stalls
```

图 6 统计信息 5

在代码中，还存在多次数据冒险问题。我列举了其中四个数据冒险。经过调整指令序列后，原本的 220 次 RAW Stalls 已减少到 150 次。

```
Execution
478 Cycles
299 Instructions
1.599 Cycles Per Instruction (CPI)

Stalls
150 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
25 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
100 Bytes
```

图 7 统计信息 6

调整后的代码如下所示：

```
.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:
      ld r16,len(r0)
      daddi r17,r0,0
      daddi r23,r0,c
loop1: slt r8,r17,r16
      daddi r19,r0,0
      beq r8,r0,exit1

loop2: slt r8,r19,r16
      daddi r21,r0,a
      daddi r22,r0,b
      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
      daddi r19,r19,1
      sd r12,0(r11)

      j loop2
exit2: daddi r17,r17,1
      j loop1
exit1: halt
```

3、优化 2—Forwarding 功能开启

首先，开启 Forwarding 功能。

我将执行未经优化的代码，并开启 Forwarding 功能。通过这种设置，我就可以观察到在数据相关性存在时，流水线如何处理这些冒险，并统计数据冒险的次数。开启 Forwarding 后，处理器会尝试在流水线中动态转发数据，以减少因数据冒险引起的停顿。这将有助于观察在开启转发功能时，数据冒险的次数是如何变化的。执行过程中，我将记录数据冒险的次数，特别是 RAW Stalls，并与关闭转发时的结果进行对比。通过对比，我可以更清楚地了解数据转发对流水线性能的影响，并分析转发机制在减少数据冒险中的作用。

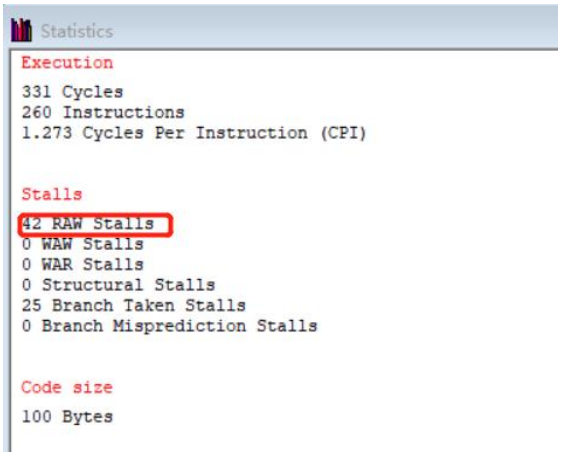


图 8 统计信息 7

如图，未调整指令序列，仅做 Forwarding 功能的开启，发生 42 次 RAW Stalls。

如下图，在优化 1 的基础上开启 Forwarding 功能：

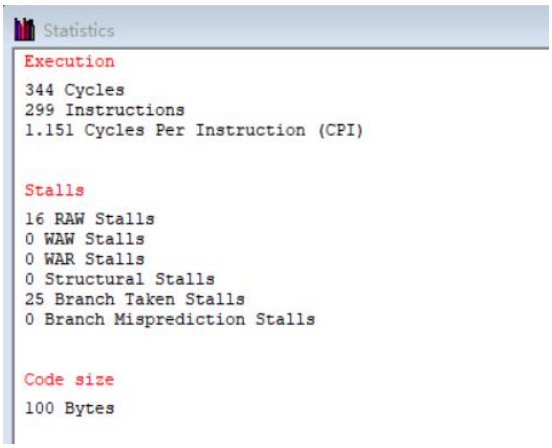


图 9 统计信息 8

仅发生 16 次 RAW Stalls。优化效果明显。

4、优化 3—结构相关优化

首先，将下述文档所给代码存入.s 文件中：

```
.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 r19,d(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)
        sd r18,b(r0)
        Halt
```

将该文件载入，在程序运行前将 configure->architecture->division latency 改为 10。
之后运行程序，观察 ***Cycle*** 窗口结果如下所示：

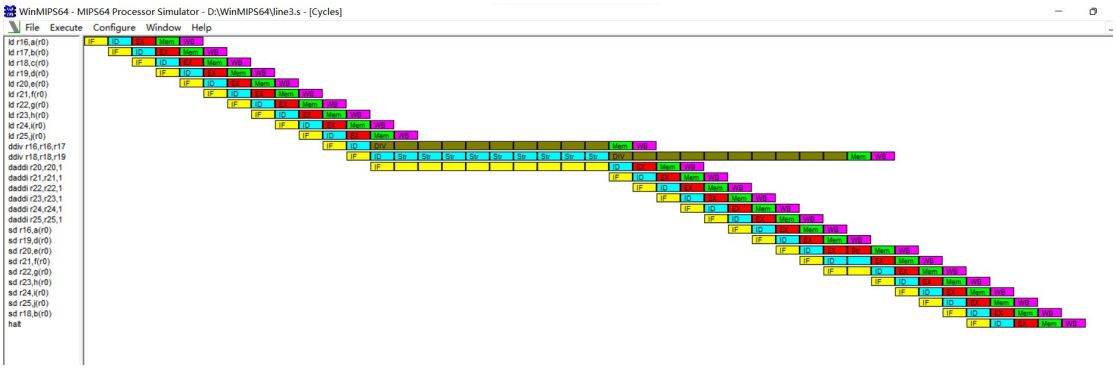


图 10 ***Cycle*** 窗口

在 ***Statistics*** 窗口的结果如下：

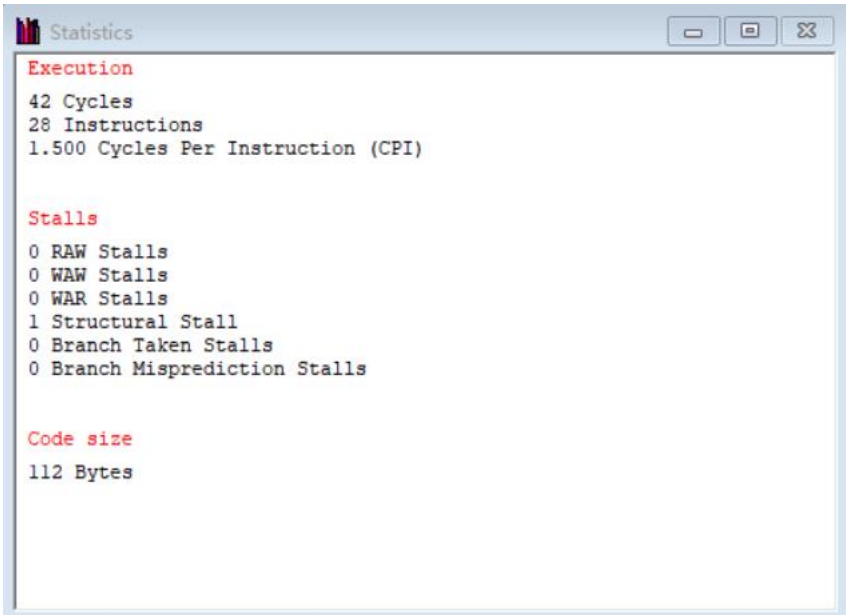


图 11 统计信息 9

通过观察，我发现两个连续的除法操作产生了明显的结构相关性。第二个除法指令需要等待第一个除法指令在执行阶段占用的资源，导致其阻塞了 9 个周期。

显然，这种连续的除法操作所引发的结构相关性极大地降低了流水线效率。为了消除这种结构相关性，我需要调整指令序列，将一些无关的指令插入到这两条连续的除法指令之间。这样做可以让第二个除法指令在等待资源的同时，执行这些插入的指令，从而减少阻塞的周期，提高流水线的整体效率。

通过这种方式，我能够有效地减少结构相关性带来的影响，优化流水线的性能：

```
 .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)
      ddiv r16,r16,r17
      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
      daddi r22,r22,1
      ddiv r18,r18,r19
      daddi r23,r23,1
      daddi r24,r24,1
      daddi r25,r25,1
      sd r16,a(r0)
      sd r19,d(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)
      sd r18,b(r0)
      halt
```

这样代码中两个除法指令分隔开来。

运行后可以观察到 ***Cycle*** 窗口结果如下所示：

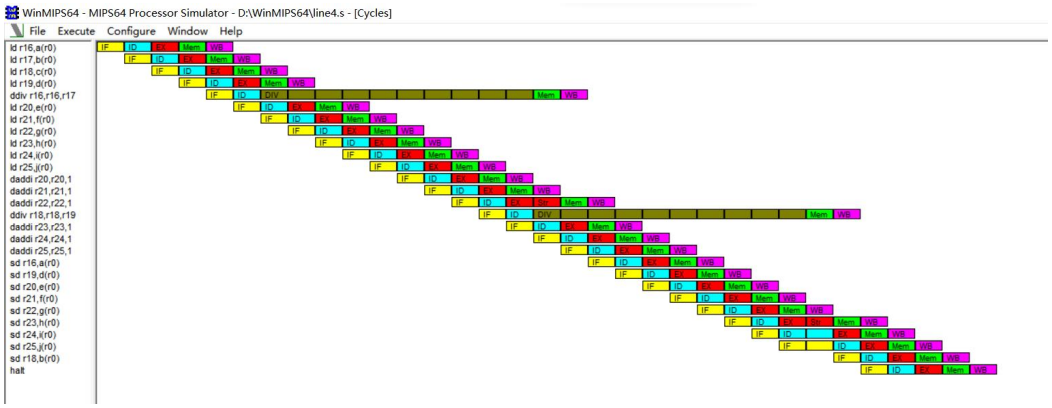


图 12 *Cycle* 窗口

此时由于两条除法指令相隔较远，不会发生阻塞。

在 ***Statistics*** 窗口的结果如下：

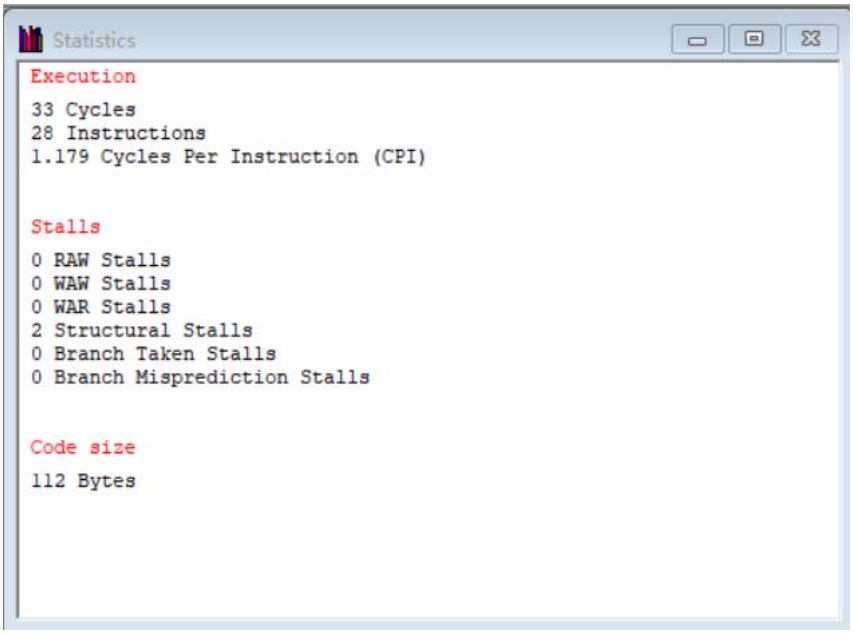


图 13 统计信息 10

通过对比，我发现优化后的代码在运行时减少了 9 个周期，从原来的 42 个周期降低到了 33 个周期，优化效果十分显著。

这表明，通过调整指令序列并消除结构相关性，流水线的执行效率得到了显著提升。优化后的代码不仅减少了阻塞时间，还提高了整体的执行速度。

五、实验结果

1、初始代码运行结果

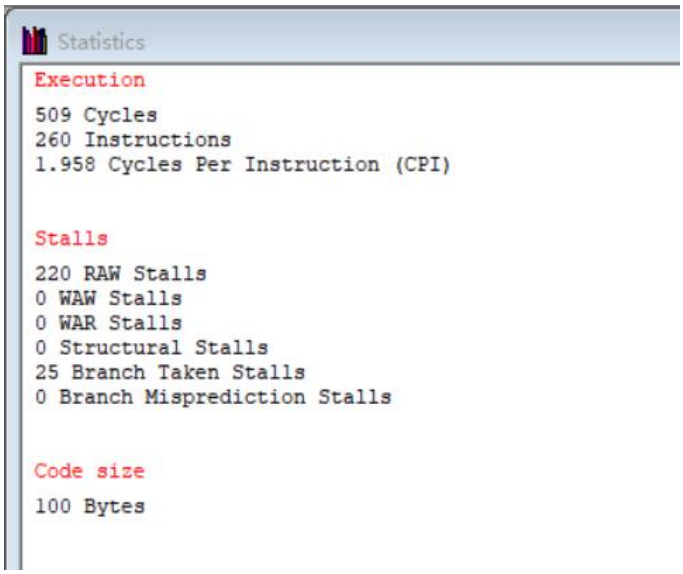


图 14 结果 1

共发生 220 次 RAW Stalls。

2、优化 1：调整指令序列的运行结果

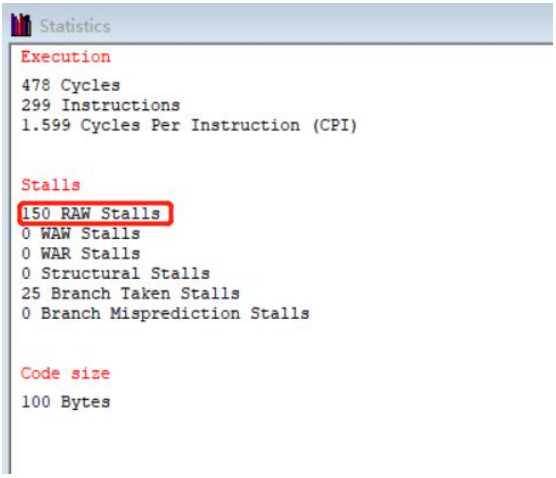


图 14 结果 2

由于指令序列的调整，较原先代码降低了 70 次数据冒险。

3、优化 2：Forwarding 功能开启

(1)在原有代码基础上 Forwarding 功能开启

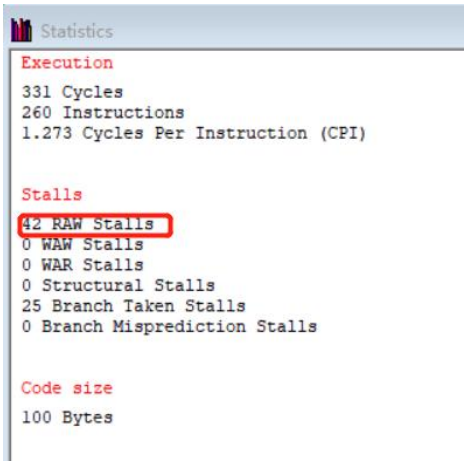


图 15 结果 3

(2)在优化 1 的代码上 Forwarding 功能开启

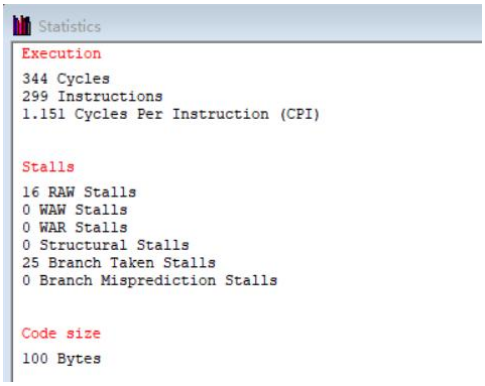


图 16 结果 4

在开启 Forwarding 功能后，数据冒险发生的次数得到了有效降低。在基于优化 1 的基础上启用该功能后，数据冒险的次数减少到仅发生了 16 次。
这进一步表明，数据转发机制在优化流水线性能方面发挥了重要作用，通过动态转发数据，能够显著减少因数据相关性引起的停顿和延迟，从而提升了程序的执行效率。

4、优化 3：结构相关优化

(1)结构优化前的结果

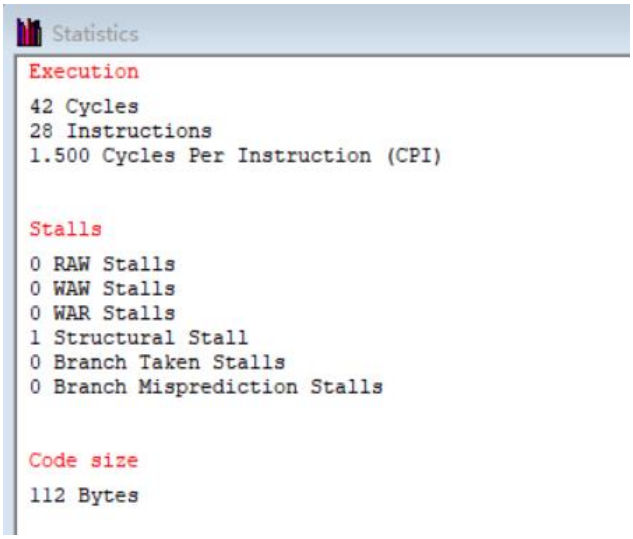


图 17 结果 5

未经结构优化前，两条除法指令会发生阻塞。

(2)结构优化后的结果

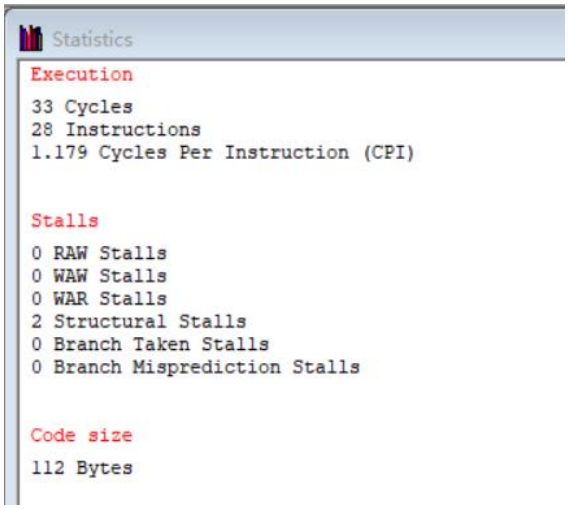


图 18 结果 6

通过将两条除法指令分隔开可以实现结构优化，此时较原先的代码减少 9 个指令周期，由原先的 42 个 Cycles 减少为 33Cycles，优化效果显著。

五、实验总结与体会

在本次实验中，我对指令序列进行了一系列优化，以提升流水线执行效率，并减少数据相关性和结构相关性所带来的性能损失。通过逐步应用三种优化方法，我深入理解了流水线中数据冒险、结构冒险以及如何通过调整指令顺序和启用硬件特性来有效提高程序的执行效率。以下是对三种优化方法的总结与体会：

1. 优化 1：调整指令序列的运行结果

首先，我通过分析代码中的数据相关性，调整了指令序列。在原始代码中，多个指令之间存在数据依赖，导致了 RAW Stalls 和流水线停顿。通过将无关的指令插入到数据依赖的指令之间，我成功减少了数据冒险的发生次数，并减少了流水线停顿时间。这一优化显著提升了程序的执行效率，从 220 次 RAW Stalls 降低到 150 次，证明了指令调度在减少数据相关性方面的有效性。

2. 优化 2：Forwarding 功能开启

在进行了优化 1 后，我开启了 Forwarding 功能，以进一步减少数据冒险的次数。数据转发允许流水线动态地将数据从一个阶段转移到另一个阶段，从而减少了等待数据的时间。在开启该功能后，数据冒险的次数显著减少，仅发生了 16 次 RAW Stalls。这一优化表明，数据转发功能在现代处理器中起到了至关重要的作用，能够有效减轻数据相关性对流水线性能的影响，进一步提高了程序的执行速度。

3. 优化 3：结构相关优化

在优化 2 的基础上，我又针对结构相关性进行了优化。代码中两个连续的除法操作由于需要共享相同的硬件资源，导致了结构相关性和资源阻塞。通过调整指令序列，将无关指令插入到这两条除法指令之间，我成功减少了流水线的阻塞时间。这一优化将原先的 42 个周期的执行时间降低到了 33 个周期，显著提高了流水线的效率，减少了由于结构相关性带来的性能损失。

总结与体会

通过这三种优化方法，我深刻理解了流水线的工作原理以及如何通过合理的指令调度和硬件功能来减少性能瓶颈。优化 1 强调了通过调整指令顺序来减少数据冒险，优化 2 展示了硬件层面数据转发的优势，而优化 3 则体现了在硬件资源共享时如何避免结构相关性引发的性能下降。

这些优化不仅提升了程序的执行效率，还让我更加明白了流水线设计中的挑战和优化策略。在未来的开发过程中，我将继续关注指令级优化，合理利用硬件资源，最大限度地提高程序的执行性能。

指导教师批阅意见： **成绩评定：** 指导教师签字： 年　月　日
备注：

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。