**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： 数学与统计学院**

**专 业： 信息与计算科学（数学与计算机实验班）**

**指 导 教 师： 罗秋明**

**报告人： 王曦 学号： 2021192010 班级： 数计**

**实 验 时 间： 2023年10月12日**

**实验报告提交时间： 2023年10月12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 |

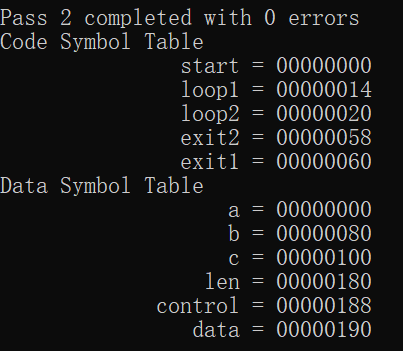
实验前请保证winMIPS64配置中“Enable Forwarding”没有选中。将这段代码加载到WinMIPS64中，运行后观察结果（提供Statistic窗口截图）。 从Statistic窗口记录：本程序运行过程中总共产生了多少次RAW的数据相关。接下来，我们对产生数据相关的代码逐个分析，请列出产生数据相关的代码，并在下一步中进行分析和优化。

1. 调整指令序列

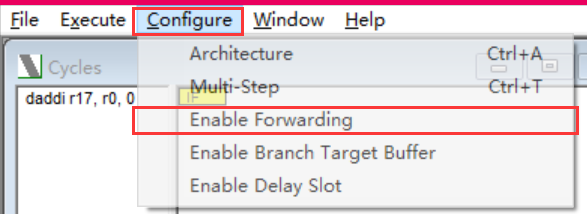
在这一部分，我们利用指令调整的方法对数据相关代码进行优化，规避数据相关。

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。

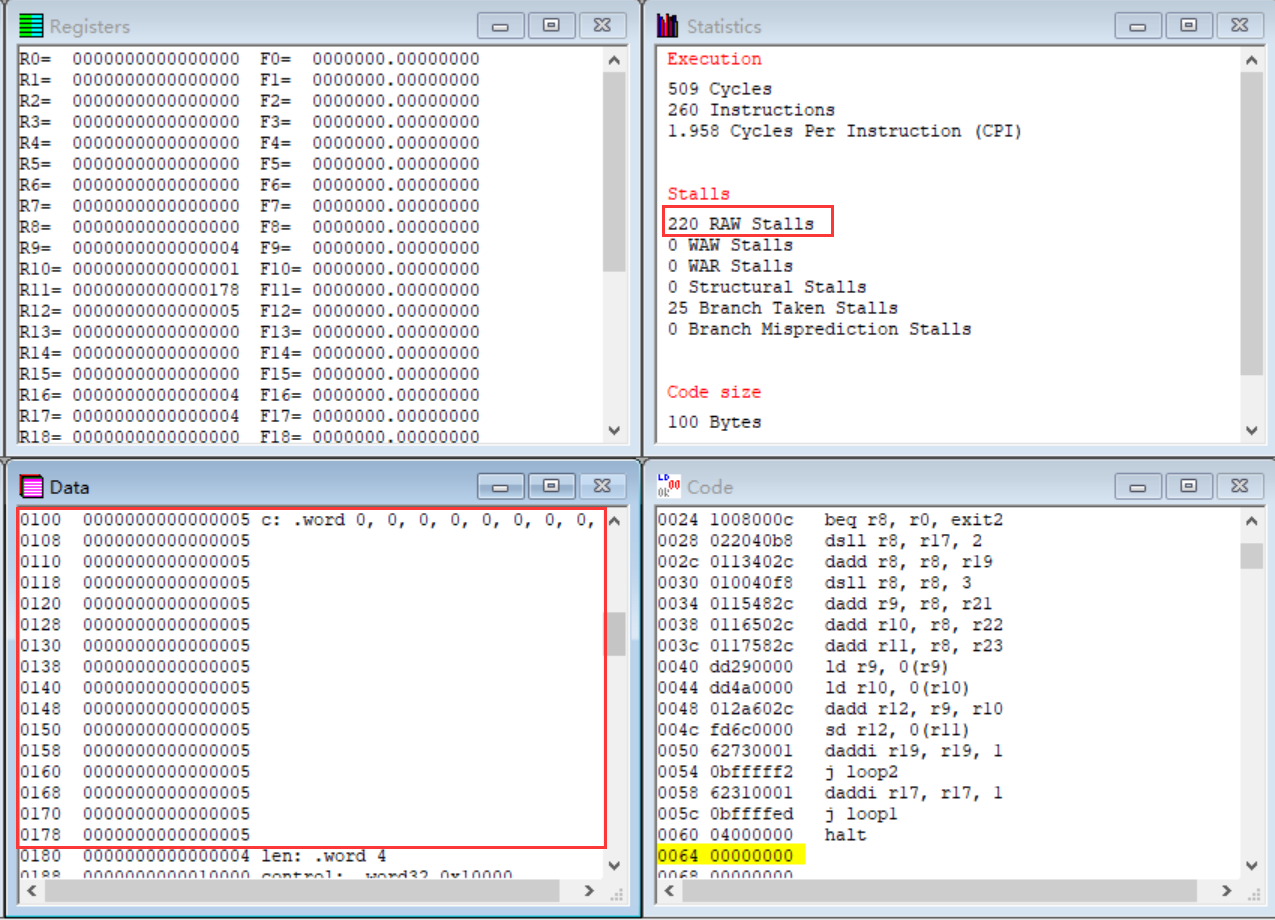
将上述代码存到 matrix.s 中，用 asm.exe 检查其语法，如下图所示。



将 matrix.s 载入 WinMIPS64 中，并关闭 Configure – Enable Forwarding ，如下图所示。



按 F4 执行程序，发现有 220 RAW，C 矩阵的元素都为 5 ，结果正确。



按 F7 单步执行程序或 F8 多步执行程序，发现有如下数据相关的指令造成堵塞。

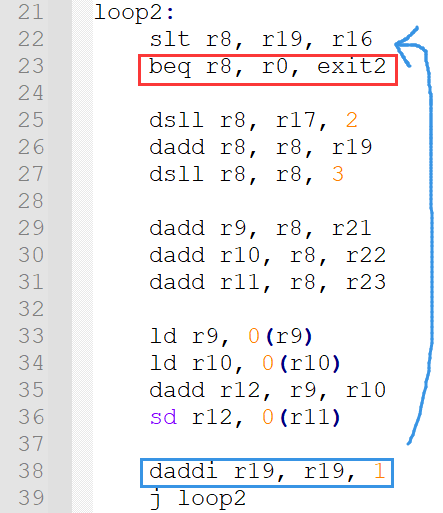
1. r16 需读取后再进行运算，发生数据冒险。将 ld 指令提前，以避免数据冲突。

|  |  |
| --- | --- |
| **修改前** | **修改后** |
| 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) # 发生数据冒险, 提前该指令以避免堵塞  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，将其滞后即可。因条件成立时条件到的 Label exit1 是 halt ，故提前 daddi 指令不会产生逻辑错误。

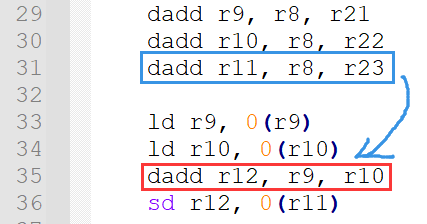
|  |  |
| --- | --- |
| **修改前** | **修改后** |
| loop1:  slt r8, r17, r16  beq r8, r0, exit1    daddi r19, r0, 0 | loop1:  slt r8, r17, r16  daddi r19, r0, 0  beq r8, r0, exit1 # 滞后发生数据冒险的指令, 防止堵塞 |

3. 如下图所示，红框内的指令发生数据相关，开启前推也无法解决。可将蓝框内不冲突的指令提前，此时开启前推即可防止堵塞。



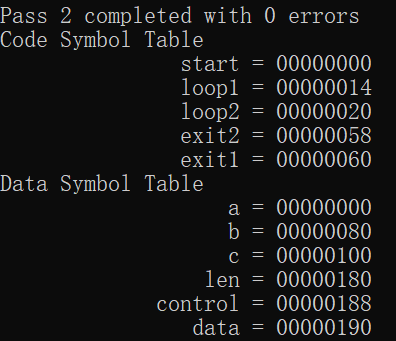
|  |  |
| --- | --- |
| **修改前** | **修改后** |
| loop2:  slt r8, r19, r16  beq r8, r0, exit2    dsll r8, r17, 2  dadd r8, r8, r19  dsll r8, r8, 3    dadd r9, r8, r21  dadd r10, r8, r22  dadd r11, r8, r23    ld r9, 0(r9)  ld r10, 0(r10)  dadd r12, r9, r10  sd r12, 0(r11)  daddi r19, r19, 1  j loop2 | loop2:  slt r8, r19, r16  daddi r19, r19, 1 # 提前该指令, 开启前推后可防止堵塞  beq r8, r0, exit2 # 发生数据相关    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 |

4. 如下图所示，红框内的指令发生数据冒险（取数-使用型），开启前推也无法解决。可将前面不冲突的关于 r11 的指令滞后，将其插入到关于 r10 的 ld 和 dadd 指令之间，此时 r12 的存储发生冲突，但开启前推可解决。



|  |  |
| --- | --- |
| **修改前** | **修改后** |
| 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 # 滞后不冲突的指令, 防止阻塞  dadd r12, r9, r10 # 发生数据冒险    sd r12, 0(r11) |

5. 优化后的代码存到 matrix (optimized).s 中，用 asm.exe 检查语法错误，如下图所示。



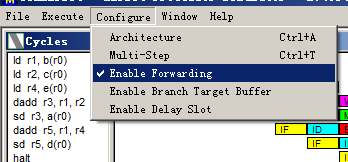
将其载入 WinMIPS64 中，在关闭前推的条件下运行，发生了 185 次 RAW，与优化前相比减少了 35 次，如下图所示。

|  |  |
| --- | --- |
| **优化前** | **优化后** |
|  |  |

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。

开启前推后运行程序，观察到 RAW 下降至 0 次，如下图所示。解释见 **一、调整指令序列** 中的优化过程。

|  |  |
| --- | --- |
| **关闭前推** | **开启前推** |
|  |  |

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。

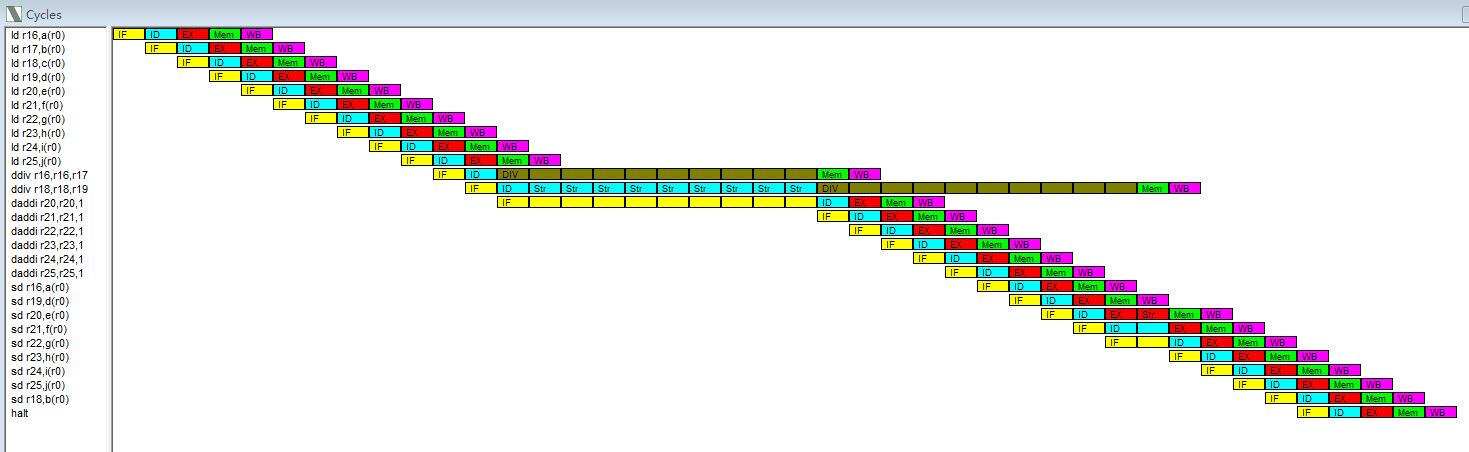
下面是给出的C代码：

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

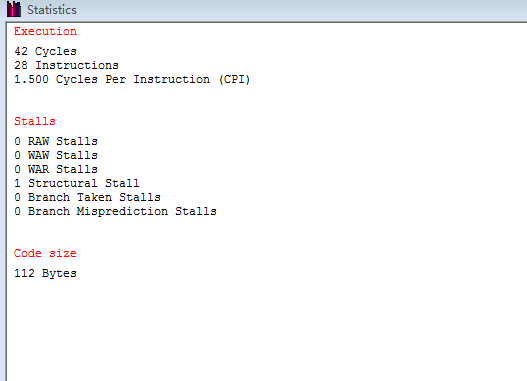
根据上述的C代码，我们给出数据相关优化的指令如下：

|  |
| --- |
| .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    .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。

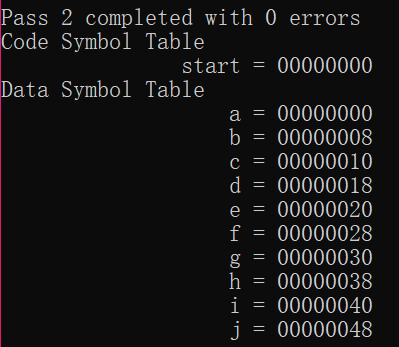
显然，这样的连续的除法所导致的结构相关极大的降低了流水线效率，为了消除结构相关，我们需要做的是调整指令序列，将其他无关的指令塞入两条连续的除法指令中。

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。

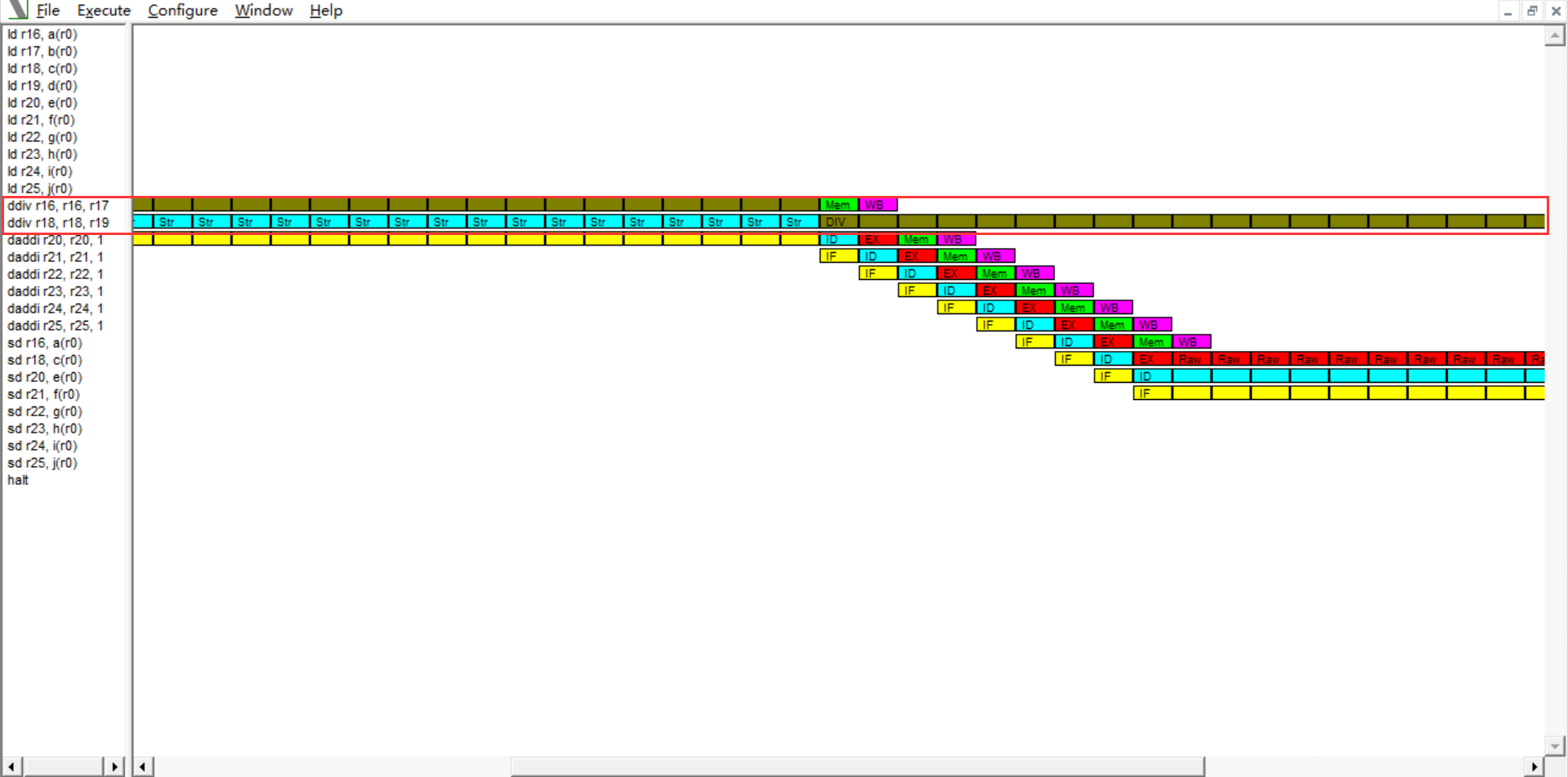
给出的 MIPS 代码不完整，补充如下。

|  |
| --- |
| .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 |

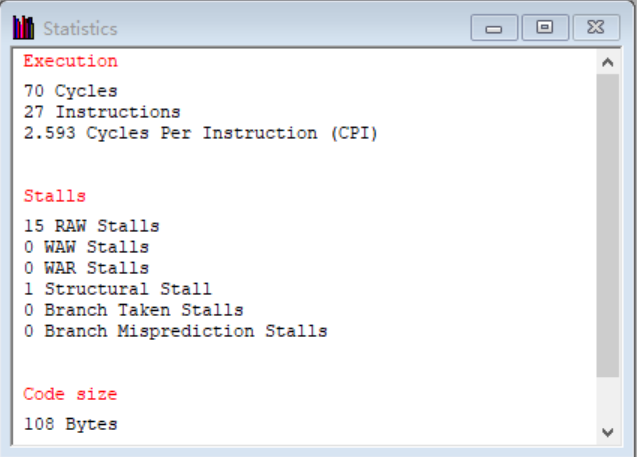
将上述代码存到 div.s 中，用 asm.exe 检查语法错误，如下图所示。



将其载入 WinMIPS64 并运行，如下图所示。



观察到两条连续的除法指令会占用同一组件，一次需 10 个周期。程序所用的时钟周期数如下图所示。



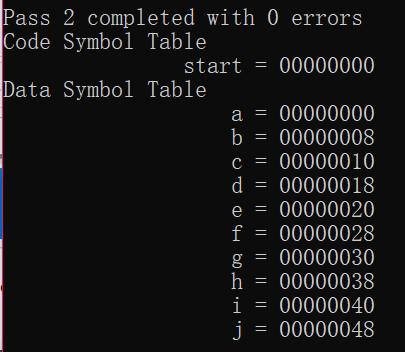
为防止流水阻塞，应在两条除法指令间插入 9 条无关的指令。此时第一条除法后的第 9 条指令的存储组件会发生结构冲突，但后面的除法指令不会很快用到存储组件，故该结构冲突影响可忽略。但注意第二条除法指令后的第 9 条指令会发生存储组件的结构冲突，此时会造成阻塞。

对第一条除法指令，因它需用到 r16 和 r17 ，而取数后立即执行除法指令会发生取数-使用型冒险，故需在取数后插入不冲突的指令以防止阻塞。注意除法需将结果写回内存，故也需滞后对应的存储指令。

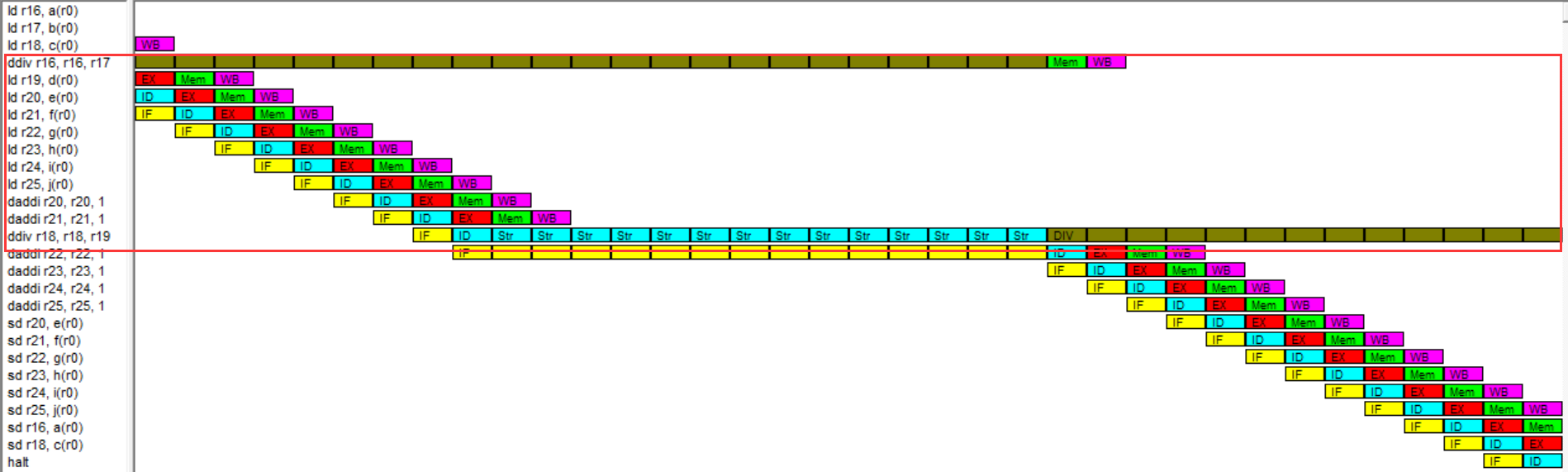
将第一条指令提前到取数完成后的第 2 条指令滞后，再将第二条指令移到第一条指令之后的第 9 条指令处，将对应的存储指令都滞后，优化后的代码如下。

|  |
| --- |
| .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) # 第一条除法指令取数完成  ld r18, c(r0) # 插入不冲突的指令, 防止阻塞  ddiv r16, r16, r17 # 将第一条除法指令提前至取数完成后的第 2 条指令    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)    # 提前不冲突的指令  daddi r20, r20, 1  daddi r21, r21, 1    ddiv r18, r18, r19 # 将第二条除法指令置于第一条除法指令之后的第 9 条指令    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)    # 滞后对应的存储指令  sd r16, a(r0)  sd r18, c(r0)    halt |

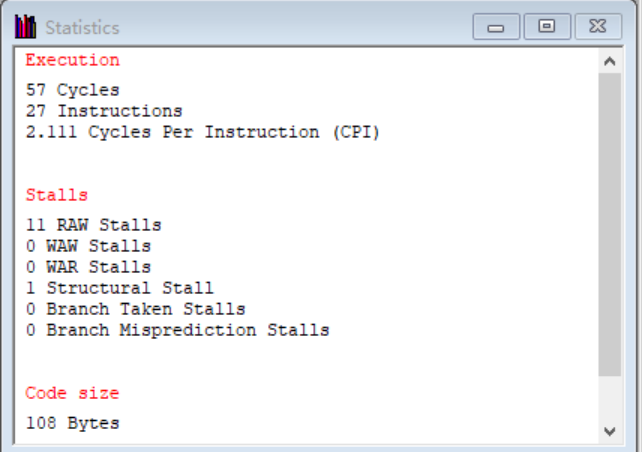
将上述代码存到 div (optimized).s 中，用 asm.exe 检查语法错误，如下图所示。



将其载入 WinMIPS64 ，开启前推后运行，结果如下图所示。



观察到两条除法指令的流水之间插入了其他指令，大大提升效率，此时程序所需的时钟周期数如下图所示。



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**

**1. 调整指令序列**

优化前发生 220 次 RAW，优化后发生了 185 次 RAW，与优化前相比减少了 35 次，如下图所示。

|  |  |
| --- | --- |
| **优化前** | **优化后** |
|  |  |

**2. 开启前推**

开启前推后运行程序，观察到 RAW 下降至 0 次，如下图所示。

|  |  |
| --- | --- |
| **关闭前推** | **开启前推** |
|  |  |

**3. 结构优化**

优化前程序需 70 个时钟周期，优化后程序需 57 个时钟周期，如下图所示。

|  |  |
| --- | --- |
| **优化前** | **优化后** |
|  |  |

**六、实验总结与体会**

本次处理器结构实验是一次深入理解MIPS流水线架构以及流水线冒险的学习经历。通过观察和修改一段MIPS汇编代码，我对现代处理器的内部工作原理有了更深入的了解，并学到了如何提高流水线效率。

首先，我对给定的代码进行了分析，该代码包含了数据相关，导致了流水线冒险。通过运行代码并观察流水线冒险的产生，我能够清楚地看到如何在不同阶段的流水线中出现冒险，包括数据相关、控制相关等。这为我提供了一个很好的起点，让我能够理解为什么流水线冒险会影响性能。

接下来，我对代码进行了一些指令序列的调整，以避免数据相关，成功地减少了流水线冒险的发生。这个过程中，我学会了如何分析和优化指令序列，以最大程度地减少性能下降。

最后，我启用了forwarding功能，这是一种硬件技术，可以在数据相关的情况下提供性能提升。通过启用forwarding，我能够减少数据相关引起的流水线停顿，从而提高了性能。这个实验让我更深入地了解了现代处理器内部的优化技术，以提高运行效率。

在整个实验过程中，我积累了大量关于MIPS指令集和流水线结构的知识。我学会了如何分析和优化代码，以减少流水线冒险，提高性能。此外，我也更加熟悉了MIPS指令的使用，这对于理解计算机体系结构非常重要。

总的来说，这次处理器结构实验是一次非常有益的学习经历。通过实际操作和观察，我不仅更深入地了解了流水线冒险和性能优化的原理，还增强了我的计算机体系结构知识。这将对我的未来学术和职业发展产生积极影响。

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