**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。

|  |  |
| --- | --- |
|  |  |

图1检查结果（左：修改前 右：修改后）

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

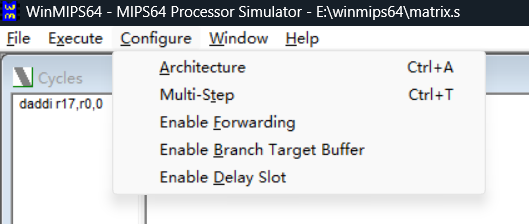


图2加载文件并关闭forwarding功能

1. 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

1. 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。
2. 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。

|  |  |
| --- | --- |
|  |  |

图4修改位置（左：修改前 右：修改后）

1. 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。

|  |  |
| --- | --- |
|  |  |

图5修改位置（左：修改前 右：修改后）

1. 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。

|  |  |
| --- | --- |
|  |  |

图5修改位置（左：修改前 右：修改后）

1. 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。
2. 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。

|  |  |
| --- | --- |
|  |  |

图6修改位置（左：修改前 右：修改后）

1. 将优化后的代码保存，使用asm.exe检查语法错误。

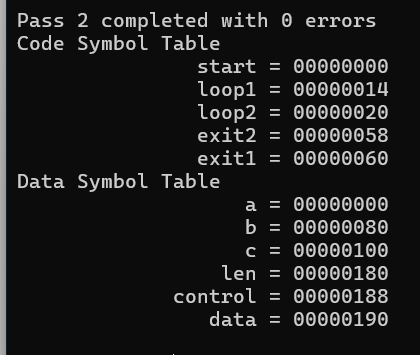


图7检查结果

1. 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。

|  |  |
| --- | --- |
|  |  |

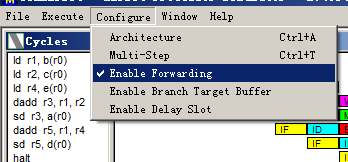
图8运行结果

1. 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）

1. 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

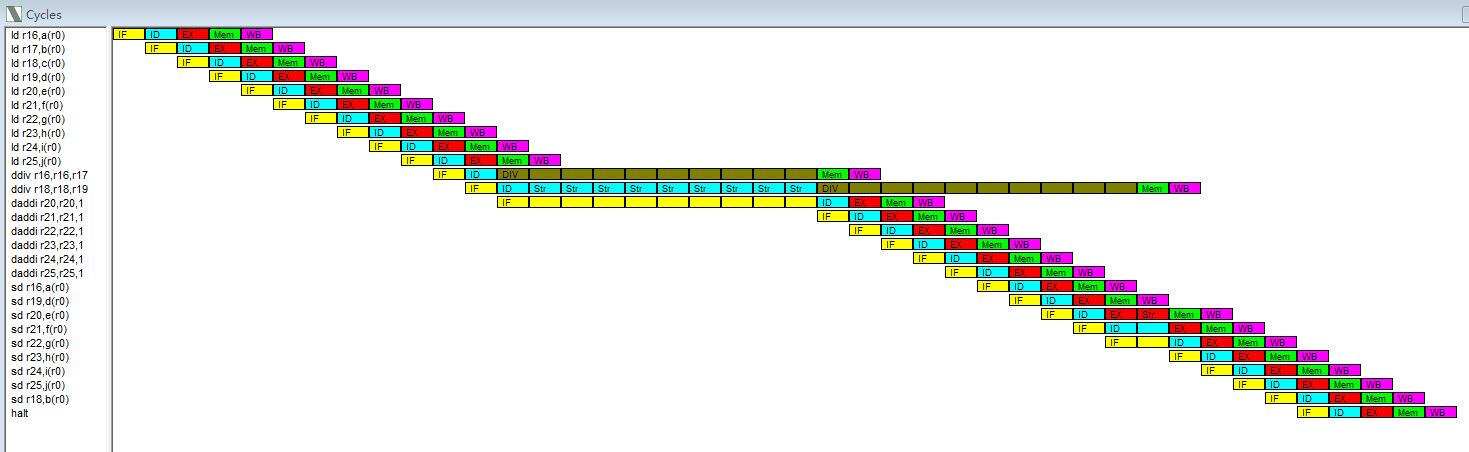
i = i + 1

j = j + 1

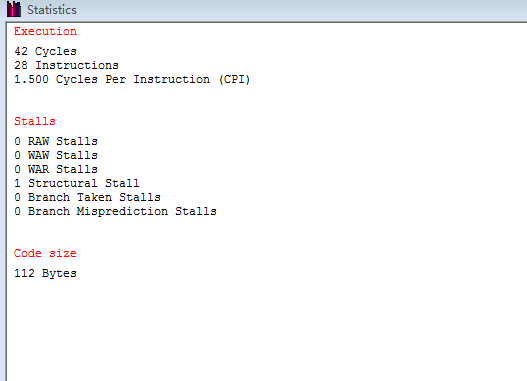
根据上述的C代码，我们给出数据相关优化的指令如下：

|  |
| --- |
| .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）

1. 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。

|  |  |
| --- | --- |
|  |  |

图10代码修改前后对比（左：修改前 右：修改后）

1. 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。

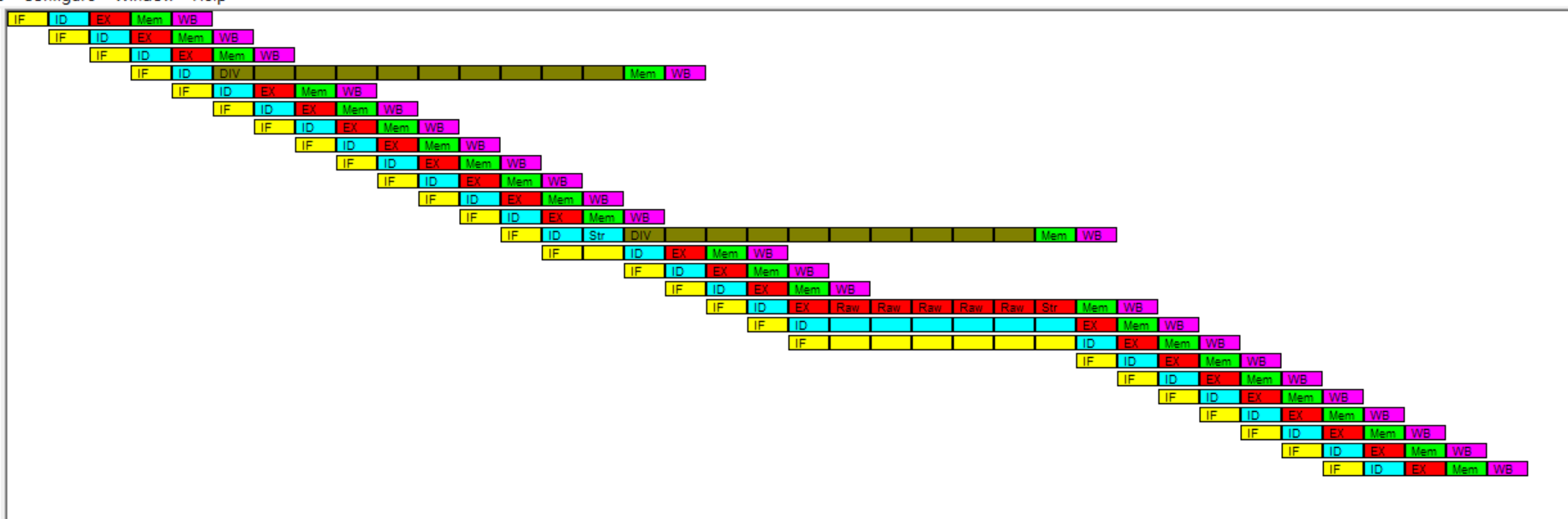


图11流水线

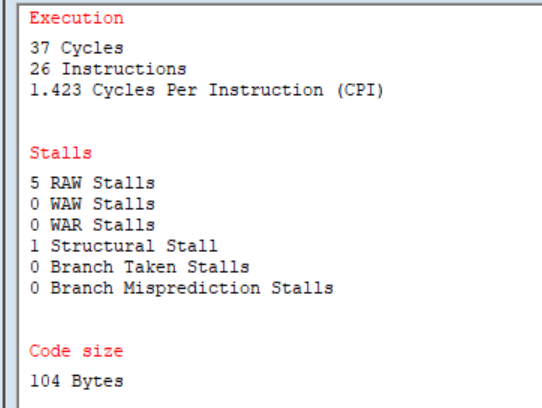


图12 Statics窗口

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： |