

实验 5：指令调度与延迟分支

于海鑫

2017211240

版本：8

更新：2020 年 4 月 16 日

1 实验目的

- (1). 加深对指令调度技术的理解
- (2). 加深对延迟分支技术的理解
- (3). 熟练账务用指令调度技术解决流水线中的数据冲突的方法
- (4). 进一步理解指令调度技术对 CPU 性能的改进
- (5). 进一步理解延迟分支技术对 CPU 性能的改进

2 实验平台

实验平台采用指令级和流水线操作级模拟器 MIPSsim。

3 实验内容和步骤

- (1). 启动 MIPSsim(用鼠标双击 MIPSsim.exe)。
- (2). 根据实验 2 的相关知识中关于流水线各段操作的描述，进一步理解流水线窗口中各段的功能，掌握各流水线寄存器的含义（双击各段，就可以看到各流水线寄存器中的内容）。
- (3). 选择“配置”→“流水方式”选项，使模拟器工作在流水方式下
- (4). 用指令调度技术解决流水线中的结构冲突与数据冲突：
 - 启动 MIPSsim。

- 用 MIPSsim 的“文件”→“载入程序”选项来加载 `schedule.s`（在模拟器所在文件夹下的“样例程序”文件夹中）。
- 关闭定向功能，这是通过“配置”→“定向”选项来实现的。
- 执行所载入的程序，通过查看统计数据 and 时钟周期图，找出并记录程序执行过程中各种冲突发生的次数，发生冲突的指令组合以及程序执行的总时钟周期数。

• RAW 停顿：16

• 自陷停顿：1

• 发生冲突的指令组合：

* LW \$r2,0(\$r1) 和 ADD \$r4,\$r0,\$r2

* ADD \$r4,\$r0,\$r2 和 SW \$r4,0(\$r1)

* SW \$r4,0(\$r1) 和 LW \$r6,4(\$r1)

* ADD \$r8,\$r6,\$r1 和 MUL \$r12,\$r10,\$r1

* ADD \$r16,\$r12,\$r1 和 ADD \$r18,\$r16,\$r1

* ADD \$r18,\$r16,\$r1 和 SW \$r18,16(\$r1)

* SW \$r18,16(\$r1) 和 LW \$r20,8(\$r1)

* MUL \$r22,\$r20,\$r14 和 MUL \$r24,\$r26,\$r14

总执行周期：33

运行报告：

```

1  汇 总：
2      执行周期总数：33
3      ID段执行了15条指令
4
5  硬件配置：
6      内存容量：4096 B
7      加法器个数：1          执行时间（周期数）：6
8      乘法器个数：1          执行时间（周期数）7
9      除法器个数：1          执行时间（周期数）10
10     定向机制：不采用
11
12     停顿（周期数）：
13         RAW停顿：16          占周期总数的百分比：48.48485%
14         其中：
15             load停顿：6          占有所有RAW停顿的百分比：37.5%
16             浮点停顿：0          占有所有RAW停顿的百分比：0%
17         WAW停顿：0          占周期总数的百分比：0%
18         结构停顿：0          占周期总数的百分比：0%
19         控制停顿：0          占周期总数的百分比：0%

```

20 自陷停顿: 1 占周期总数的百分比: 3.030303%
 21 停顿周期总数: 17 占周期总数的百分比: 51.51515%
 22
 23 分支指令:
 24 指令条数: 0 占指令总数的百分比: 0%
 25 其中:
 26 分支成功: 0 占分支指令数的百分比: 0%
 27 分支失败: 0 占分支指令数的百分比: 0%
 28
 29 load/store指令:
 30 指令条数: 5 占指令总数的百分比: 33.33333%
 31 其中:
 32 load: 3 占load/store指令数的百分比: 60%
 33 store: 2 占load/store指令数的百分比: 40%
 34
 35 浮点指令:
 36 指令条数: 0 占指令总数的百分比: 0%
 37 其中:
 38 加法: 0 占浮点指令数的百分比: 0%
 39 乘法: 0 占浮点指令数的百分比: 0%
 40 除法: 0 占浮点指令数的百分比: 0%
 41
 42 自陷指令:
 43 指令条数: 1 占指令总数的百分比: 6.666667%

时钟周期图:

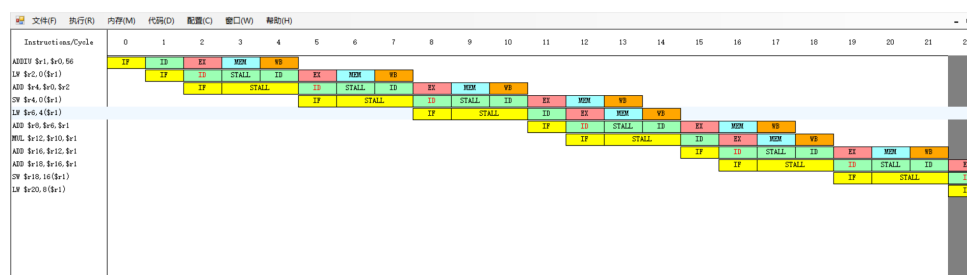


图 1: 时钟周期图

- 自己采用调度技术对程序进行指令调度，消除冲突（自己修改源程序）。将调度（修改）后的程序重新命名为 `afer-schedule.s`。（注意：调度方法灵活多样，在保证程序正确性的前提下自己随意调度，尽量减少冲突即可，不要求要达到最优。）

优化后的程序:

```

1  .text
2  main:
3  ADDIU  $r1,$r0,A
4  MUL    $r24,$r26,$r14
5  LW     $r2,0($r1)
6  MUL    $r12,$r10,$r1
7  LW     $r6,4($r1)
8  ADD    $r4,$r0,$r2
9  ADD    $r16,$r12,$r1
10 LW     $r20,8($r1)
11 SW     $r4,0($r1)
12 ADD    $r18,$r16,$r1
13 ADD    $r8,$r6,$r1
14 MUL    $r22,$r20,$r14
15 SW     $r18,16($r1)
16 TEQ    $r0,$r0
17
18 .data
19 A:
20 .word 4,6,8

```

- 载入 `afer-schedule.s`，执行该程序，观察程序在流水线中的执行情况，记录程序执行的总时钟周期数。

总执行周期：18

运行情况：

```

1  汇 总：
2      执行周期总数：18
3      ID段执行了15条指令
4
5  硬件配置：
6      内存容量：4096 B
7      加法器个数：1          执行时间（周期数）：6
8      乘法器个数：1          执行时间（周期数）7
9      除法器个数：1          执行时间（周期数）10
10     定向机制：不采用
11
12     停顿（周期数）：
13     RAW停顿：1          占周期总数的百分比：5.555555%

```

```

14      其中：
15          load停顿：0          占有所有RAW停顿的百分比：0%
16          浮点停顿：0          占有所有RAW停顿的百分比：0%
17      WAW停顿：0          占周期总数的百分比：0%
18      结构停顿：0          占周期总数的百分比：0%
19      控制停顿：0          占周期总数的百分比：0%
20      自陷停顿：1          占周期总数的百分比：5.555555%
21      停顿周期总数：2      占周期总数的百分比：11.111111%
22
23      分支指令：
24          指令条数：0          占指令总数的百分比：0%
25          其中：
26              分支成功：0          占分支指令数的百分比：0%
27              分支失败：0          占分支指令数的百分比：0%
28
29      load/store指令：
30          指令条数：5          占指令总数的百分比：33.333333%
31          其中：
32              load：3          占load/store指令数的百分比：60%
33              store：2          占load/store指令数的百分比：40%
34
35      浮点指令：
36          指令条数：0          占指令总数的百分比：0%
37          其中：
38              加法：0          占浮点指令数的百分比：0%
39              乘法：0          占浮点指令数的百分比：0%
40              除法：0          占浮点指令数的百分比：0%
41
42      自陷指令：
43          指令条数：1          占指令总数的百分比：6.666667%

```

时钟周期图：

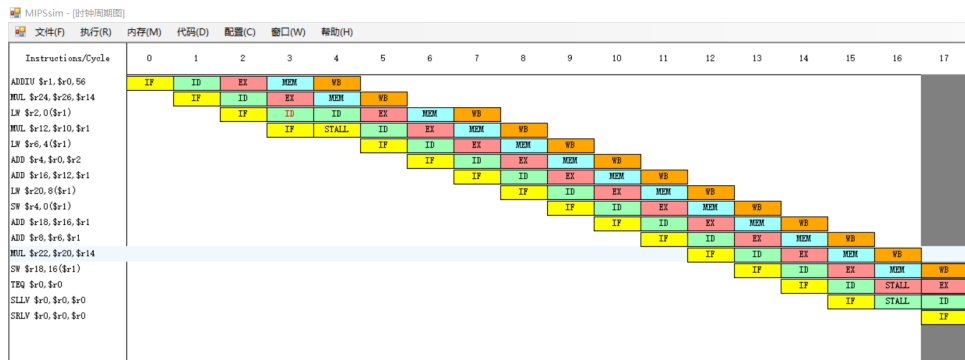


图 2: 时钟周期图

- 比较调度前和调度后的性能，论述指令调度对提高 CPU 性能的作用。

调度前的执行周期为 33，调度后的执行周期数为 18。

指令调度可以消除部分的数据冲突，通过使用指令调度提高了 CPU 的使用率，大大减少了指令冲突的次数，提高了 CPU 性能。

(5). 用延迟分支技术减少分支指令对性能的影响：

- 在 MIPSsim 中载入 branch.s 样例程序（在本模拟器目录的“样例程序”文件夹中）。
- 关闭延迟分支功能。这是通过在“配置”→“延迟槽”选项来实现的。
- 执行该程序，观察并记录发生分支延迟的时刻，记录该程序执行的总时钟周期数。

总执行周期：38

第 6, 9, 13, 21, 24, 28 周期发生了分支延迟 运行情况：

1	汇总：
2	执行周期总数：38
3	ID段执行了18条指令
4	
5	硬件配置：
6	内存容量：4096 B
7	加法器个数：1 执行时间（周期数）：6
8	乘法器个数：1 执行时间（周期数）7
9	除法器个数：1 执行时间（周期数）10
10	定向机制：不采用
11	
12	停顿（周期数）：
13	RAW停顿：16 占周期总数的百分比：42.10526%
14	其中：
15	load停顿：4 占有所有RAW停顿的百分比：25%
16	浮点停顿：0 占有所有RAW停顿的百分比：0%

```

17 WAW停顿: 0          占周期总数的百分比: 0%
18 结构停顿: 0          占周期总数的百分比: 0%
19 控制停顿: 2          占周期总数的百分比: 5.263158%
20 自陷停顿: 1          占周期总数的百分比: 2.631579%
21 停顿周期总数: 19      占周期总数的百分比: 50%
22
23 分支指令:
24   指令条数: 2          占指令总数的百分比: 11.11111%
25   其中:
26     分支成功: 1          占分支指令数的百分比: 50%
27     分支失败: 1          占分支指令数的百分比: 50%
28
29 load/store指令:
30   指令条数: 4          占指令总数的百分比: 22.22222%
31   其中:
32     load: 2             占load/store指令数的百分比: 50%
33     store: 2            占load/store指令数的百分比: 50%
34
35 浮点指令:
36   指令条数: 0          占指令总数的百分比: 0%
37   其中:
38     加法: 0             占浮点指令数的百分比: 0%
39     乘法: 0             占浮点指令数的百分比: 0%
40     除法: 0             占浮点指令数的百分比: 0%
41
42 自陷指令:
43   指令条数: 1          占指令总数的百分比: 5.55555%

```

时钟周期图:

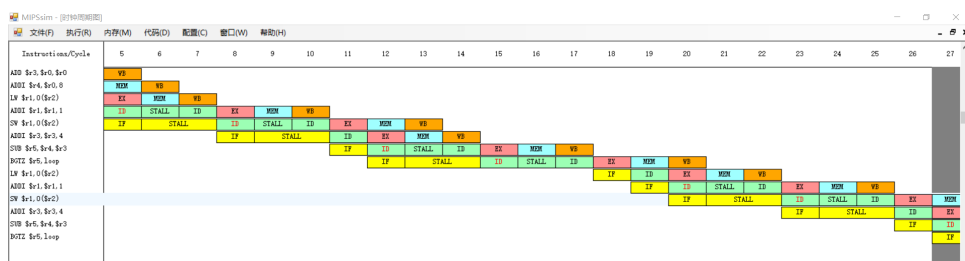


图 3: 时钟周期图

- 假设延迟槽为一个，自己对 `branch.s` 程序进行指令调度（自己修改源程序），将调度后的程序重新命名为 `delayed-branch.s`。

优化后的程序:

```

1  .text
2  main:
3  ADDI  $r2,$r0,1024
4  ADD   $r3,$r0,$r0
5  ADDI  $r4,$r0,8
6  LW    $r1,0($r2)
7  loop:
8  ADDI  $r1,$r1,1
9  ADDI  $r3,$r3,4
10 SUB   $r5,$r4,$r3
11 SW    $r1,0($r2)
12 BGTZ  $r5,loop
13 LW    $r1,0($r2)
14
15 ADD   $r7,$r0,$r6
16 TEQ   $r0,$r0

```

- 载入 `delayed-branch.s`，打开延迟分支功能，执行该程序，观察其时钟周期图，记录程序执行的总时钟周期数。总执行周期：30

运行情况：

```

1  汇总：
2      执行周期总数：30
3      ID段执行了18条指令
4
5  硬件配置：
6      内存容量：4096 B
7      加法器个数：1          执行时间（周期数）：6
8      乘法器个数：1          执行时间（周期数）7
9      除法器个数：1          执行时间（周期数）10
10     定向机制：不采用
11
12  停顿（周期数）：
13     RAW停顿：8              占周期总数的百分比：26.66667%
14     其中：
15         load停顿：2          占有所有RAW停顿的百分比：25%
16         浮点停顿：0          占有所有RAW停顿的百分比：0%
17     WAW停顿：0              占周期总数的百分比：0%
18     结构停顿：0            占周期总数的百分比：0%

```


19 控制停顿：2 占周期总数的百分比：6.666667%

20 自陷停顿：1 占周期总数的百分比：3.333333%

21 停顿周期总数：11 占周期总数的百分比：36.666667%

22

23 分支指令：

24 指令条数：2 占指令总数的百分比：11.111111%

25 其中：

26 分支成功：1 占分支指令数的百分比：50%

27 分支失败：1 占分支指令数的百分比：50%

28

29 load/store指令：

30 指令条数：4 占指令总数的百分比：22.222222%

31 其中：

32 load：2 占load/store指令数的百分比：50%

33 store：2 占load/store指令数的百分比：50%

34

35 浮点指令：

36 指令条数：0 占指令总数的百分比：0%

37 其中：

38 加法：0 占浮点指令数的百分比：0%

39 乘法：0 占浮点指令数的百分比：0%

40 除法：0 占浮点指令数的百分比：0%

41

42 自陷指令：

43 指令条数：1 占指令总数的百分比：5.555555%

时钟周期图：

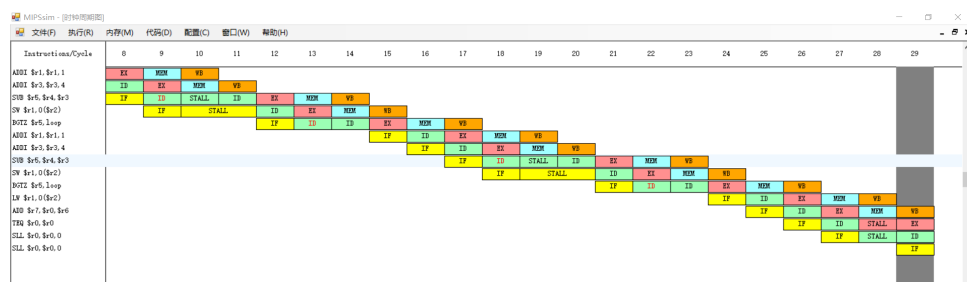


图 4: 时钟周期图

4 实验中的问题与心得

实验中没有遇到任何问题。

本次实验的心得是，分支延迟作为 MIPS 当年提出时的特性之一，在大家都是顺序流水线时对性能做出了很好的优化。但是随着世代的更迭以及分支预测技术的逐渐成熟，到了上个世纪末分支延迟技术就成了 MIPS 前端性能的拖油瓶，现代的架构，例如 RISC-V 也都取消了这一设计。无论如何，我们尊重这一在当时有效的提升了性能的技术。