

# 北京邮电大学

## 实验报告



题目： 指令调度与延迟分支

班 级： 2020211310

学 号： 2020211616

姓 名： 付容天

学 院： 计算机学院（国家示范性软件学院）

2023 年 5 月 24 日

一、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) 熟练掌握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。

二、实验平台

实验平台采用指令级和流水线操作级模拟器 MIPSsim。

三、实验内容

按照实验指导书内容，我进行了如下操作：

- (1) 启动 MIPSsim，并配置为流水方式，同时关闭定向技术。
- (2) 载入 schedule.s 样例程序，开始执行，得到下面的结果：

汇总：		结构停顿：0		占周期总数的百分比：0%	
执行周期总数：33		控制停顿：0		占周期总数的百分比：0%	
ID段执行了15条指令		自陷停顿：1		占周期总数的百分比：3.030303%	
		停顿周期总数：17		占周期总数的百分比：51.51515%	
硬件配置：					
内存容量：4096 B					
加法器个数：1		执行时间（周期数）：6		分支指令：	
乘法器个数：1		执行时间（周期数）7		指令条数：0	
除法器个数：1		执行时间（周期数）10		占指令总数的百分比：0%	
定向机制：不采用				其中：	
				分支成功：0	
				占分支指令数的百分比：0%	
				分支失败：0	
				占分支指令数的百分比：0%	
停顿（周期数）：					
RAW停顿：16		占周期总数的百分比：48.48485%			
其中：					
load停顿：6		占所有RAW停顿的百分比：37.5%			
浮点停顿：0		占所有RAW停顿的百分比：0%			
WAW停顿：0		占周期总数的百分比：0%			
结构停顿：0		占周期总数的百分比：0%			
控制停顿：0		占周期总数的百分比：0%			
自陷停顿：1		占周期总数的百分比：3.030303%			
停顿周期总数：17		占周期总数的百分比：51.51515%			
分支指令：					
指令条数：0		占指令总数的百分比：0%			
其中：					
分支成功：0		占分支指令数的百分比：0%			
分支失败：0		占分支指令数的百分比：0%			
load/store指令：					
指令条数：5		占指令总数的百分比：33.33333%			
其中：					
load：3		占load/store指令数的百分比：60%			
store：2		占load/store指令数的百分比：40%			
浮点指令：					
指令条数：0		占指令总数的百分比：0%			
其中：					
加法：0		占浮点指令数的百分比：0%			
乘法：0		占浮点指令数的百分比：0%			
除法：0		占浮点指令数的百分比：0%			
自陷指令：					
指令条数：1		占指令总数的百分比：6.666667%			

图 1.1: 样例程序执行结果（1）

图 1.2: 样例程序执行结果（2）

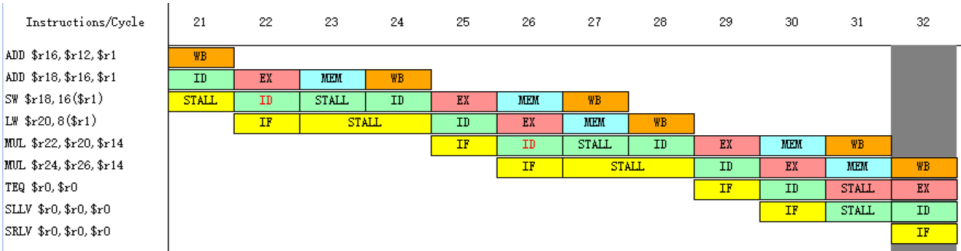


图 2: 样例程序执行时钟周期图

可以发现，程序执行的总时钟周期数位 33，其中冲突发生在以下指令组合中：

指令 1	指令 2
ADDIU \$r1, \$r0, 56	LW \$r2, 0(\$r1)
LW \$r2, 0(\$r1)	ADD \$r4, \$r0, \$r2
ADD \$r4, \$r0, \$r2	SW \$r4, 0(\$r1)
LW \$r6, 4(\$r1)	ADD \$r8, \$r6, \$r1
ADD \$r16, \$r12, \$r1	ADD \$r18, \$r16, \$r1
ADD \$r18, \$r16, \$r1	SW \$r18, 16(\$r1)
MUL \$r22, \$r20, \$r14	MUL \$r24, \$r26, \$r14

接下来，我进行了下面的内容：

- (3) 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 after-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，不要求达到最优）。我优化后的程序如下所示：

```
.text
main:
ADDIU  $r1, $r0, A
MUL    $r24,$r26,$r14
LW     $r2, 0($r1)
MUL    $r12,$r10,$r1
LW     $r6, 4($r1)
ADD    $r4, $r0, $r2
ADD    $r16,$r12,$r1
LW     $r20,8($r1)
SW     $r4, 0($r1)
ADD    $r18,$r16,$r1
ADD    $r8, $r6, $r1
MUL    $r22,$r20,$r14
SW     $r18,16($r1)
TEQ    $r0, $r0
.data
A:
.word  4, 6, 8
```

- (4) 载入 after-schedule.s 并执行该程序，观察程序在流水线中的执行情况，记录程序执行情况如下所示：

汇总: 执行周期总数: 18 ID段执行了15条指令			WAW停顿: 0 结构停顿: 0 控制停顿: 0 自陷停顿: 1 停顿周期总数: 2			占周期总数的百分比: 0% 占周期总数的百分比: 0% 占周期总数的百分比: 0% 占周期总数的百分比: 5.55555% 占周期总数的百分比: 11.11111%		
硬件配置: 内存容量: 4096 B 加法器个数: 1 乘法器个数: 1 除法器个数: 1 定向机制: 不采用			执行时间(周期数): 6 执行时间(周期数): 7 执行时间(周期数): 10			分支指令: 指令条数: 0 其中: 分支成功: 0 分支失败: 0		
停顿(周期数): RAW停顿: 1 其中: Load停顿: 0 浮点停顿: 0 WAW停顿: 0 结构停顿: 0 控制停顿: 0 自陷停顿: 1 停顿周期总数: 2			占周期总数的百分比: 5.55555% 占所有RAW停顿的百分比: 0% 占所有RAW停顿的百分比: 0% 占周期总数的百分比: 0% 占周期总数的百分比: 0% 占周期总数的百分比: 0% 占周期总数的百分比: 5.55555% 占周期总数的百分比: 11.11111%			load/store指令: 指令条数: 5 其中: load: 3 store: 2		
分支指令: 指令条数: 0 其中: 分支成功: 0 分支失败: 0			占指令总数的百分比: 0% 占分支指令数的百分比: 0% 占分支指令数的百分比: 0%			占指令总数的百分比: 0% 占指令总数的百分比: 0% 占指令总数的百分比: 0%		
			浮点指令: 指令条数: 0 其中: 加法: 0 乘法: 0 除法: 0			占指令总数的百分比: 0% 占浮点指令数的百分比: 0% 占浮点指令数的百分比: 0% 占浮点指令数的百分比: 0%		
			自陷指令: 指令条数: 1			占指令总数的百分比: 6.66667%		

图 3.1: 优化后执行结果 (1)

图 3.2: 优化后执行结果 (2)

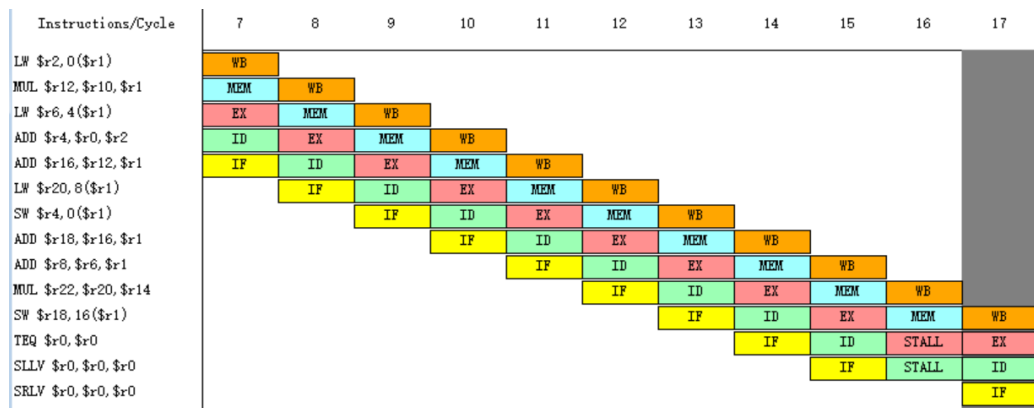


图 4: 优化后执行时钟周期图

可以发现, 程序总执行周期下降到了 18, 并且停顿周期占比也从 51.52%下降到了 11.11%, 可见优化效果显著, 说明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。

接下来我探究了使用延迟分支技术减少分支指令对性能的影响, 包括:

- (5) 在 MIPSsim 中载入样例程序 branch.s, 关闭延迟分支功能执行该程序, 观察并记录发生分支延迟的时刻, 并记录该程序执行的总时钟周期数, 结果如下图所示:

汇总:	
执行周期总数: 38	
ID段执行了18条指令	
硬件配置:	
内存容量: 4096 B	
加法器个数: 1	执行时间 (周期数): 6
乘法器个数: 1	执行时间 (周期数): 7
除法器个数: 1	执行时间 (周期数): 10
定向机制: 不采用	
停顿 (周期数):	
RAW停顿: 16	占周期总数的百分比: 42.10526%
其中:	
load停顿: 4	占所有RAW停顿的百分比: 25%
浮点停顿: 0	占所有RAW停顿的百分比: 0%
WAW停顿: 0	占周期总数的百分比: 0%
结构停顿: 0	占周期总数的百分比: 0%
控制停顿: 2	占周期总数的百分比: 5.263158%
自陷停顿: 1	占周期总数的百分比: 2.631579%
停顿周期总数: 19	占周期总数的百分比: 50%
分支指令:	
指令条数: 2	占指令总数的百分比: 11.11111%
其中:	
分支成功: 1	占分支指令数的百分比: 50%
分支失败: 1	占分支指令数的百分比: 50%

图 5.1: 分支样例程序执行结果 (1)

WAW停顿: 0	占周期总数的百分比: 0%
结构停顿: 0	占周期总数的百分比: 0%
控制停顿: 2	占周期总数的百分比: 5.263158%
自陷停顿: 1	占周期总数的百分比: 2.631579%
停顿周期总数: 19	占周期总数的百分比: 50%
分支指令:	
指令条数: 2	占指令总数的百分比: 11.11111%
其中:	
分支成功: 1	占分支指令数的百分比: 50%
分支失败: 1	占分支指令数的百分比: 50%
load/store指令:	
指令条数: 4	占指令总数的百分比: 22.22222%
其中:	
load: 2	占load/store指令数的百分比: 50%
store: 2	占load/store指令数的百分比: 50%
浮点指令:	
指令条数: 0	占指令总数的百分比: 0%
其中:	
加法: 0	占浮点指令数的百分比: 0%
乘法: 0	占浮点指令数的百分比: 0%
除法: 0	占浮点指令数的百分比: 0%
自陷指令:	
指令条数: 1	占指令总数的百分比: 5.555555%

图 5.2: 分支样例程序执行结果 (2)

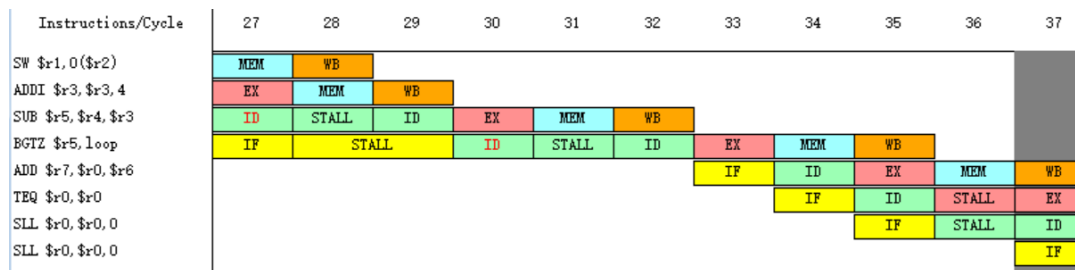


图 6: 分支样例程序执行时钟周期图

可以看到, 分支样例程序的总执行周期为 38, 并且在 14 和 29 周期处出现了分支延迟的情况。

接下来, 我进行了如下所述的实验内容:

- (6) 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。我优化后的程序具体内容如下所示:

```
.text
main:
ADDI    $r2, $r0, 1024
ADD     $r3, $r0, $r0
ADDI    $r4, $r0, 8
LW      $r1, 0($r2)
loop:
ADDI    $r1, $r1, 1
ADDI    $r3, $r3, 4
SUB     $r5, $r4, $r3
SW      $r1, 0($r2)
BGTZ    $r5, loop
```

LW	\$r1, 0(\$r2)
ADD	\$r7, \$r0, \$r6
TEQ	\$r0, \$r0

(7) 载入 delayed-branch.s 并打开延迟分支功能，执行该程序，记录执行结果如下所示：

汇总：		WAW停顿：0		占周期总数的百分比：0%	
执行周期总数：31		结构停顿：0		占周期总数的百分比：0%	
ID段执行了19条指令		控制停顿：0		占周期总数的百分比：0%	
		自陷停顿：1		占周期总数的百分比：3.225806%	
		停顿周期总数：11		占周期总数的百分比：35.48387%	
硬件配置：					
内存容量：4096 B					
加法器个数：1		执行时间（周期数）：6		分支指令：	
乘法器个数：1		执行时间（周期数）：7		指令条数：2	
除法器个数：1		执行时间（周期数）：10		占指令总数的百分比：10.52632%	
定向机制：不采用				其中：	
		分支成功：1		占分支指令数的百分比：50%	
		分支失败：1		占分支指令数的百分比：50%	
停顿（周期数）：					
RAW停顿：10		占周期总数的百分比：32.25806%			
其中：					
load停顿：4		占有所有RAW停顿的百分比：40%			
浮点停顿：0		占有所有RAW停顿的百分比：0%			
WAW停顿：0		占周期总数的百分比：0%			
结构停顿：0		占周期总数的百分比：0%			
控制停顿：0		占周期总数的百分比：0%			
自陷停顿：1		占周期总数的百分比：3.225806%			
停顿周期总数：11		占周期总数的百分比：35.48387%			
分支指令：					
指令条数：2		占指令总数的百分比：10.52632%			
其中：					
分支成功：1		占分支指令数的百分比：50%			
分支失败：1		占分支指令数的百分比：50%			
load/store指令：					
指令条数：5		占指令总数的百分比：26.31579%			
其中：					
load：3		占load/store指令数的百分比：60%			
store：2		占load/store指令数的百分比：40%			
浮点指令：					
指令条数：0		占指令总数的百分比：0%			
其中：					
加法：0		占浮点指令数的百分比：0%			
乘法：0		占浮点指令数的百分比：0%			
除法：0		占浮点指令数的百分比：0%			
自陷指令：					
指令条数：1		占指令总数的百分比：5.263158%			

图 7.1：优化后分支程序执行结果（1）

图 7.2：优化后分支程序执行结果（2）

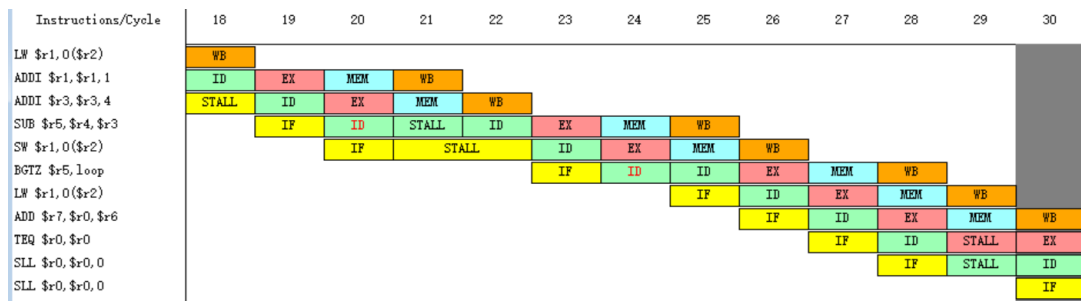


图 8：优化后分支程序执行时钟周期图

可以对比看到，总的执行时钟周期降至 31，并且，停顿周期占比从 50%下降到了 35%，性能提升明显。可以发现，采用延迟分支技术能够有效提高 CPU 性能。

## 四、实验总结

在本次实验中，我回顾了指令调度与延迟分支基本知识，并对样例程序进行了手动优化，显著提高了程序的执行效率，并进行了性能对比，圆满完成了实验五既定的实验任务。我还加深了对执行中 MIPS 程序行为的理解。我对实验五中程序的执行过程分析得很细致，弄清楚了每一处细节，收获颇丰！