北京郵電大學

实验报告

题目：指令调度与延迟分支

班　　级：2019211306

学　　号：2019211397

姓　　名：毛子恒

学　　院：计算机学院

2022 年 5 月 1 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。

二、 实验内容

（1）启动 MIPSsim。
（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。
（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。
（5）用延迟分支技术减少分支指令对性能的影响。

三、 实验平台和环境

指令级和流水线操作级模拟器 MIPSsim。

四、 实验步骤及实验分析

（4）用指令调度技术解决流水线中的结构冲突与数据冲突:
1）启动 MIPSsim。
2）用 MIPSsim 的“文件”->“载入程序”选项来加载 schedule.s。
3）关闭定向功能，这是通过“配置“->”定向“选项来实现的。
4）执行所载入的程序，通过查看统计数据和时钟周期图，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。

汇总：
执行周期总数：33
ID段执行了15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：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%

RAW停顿：16，自陷停顿：1，执行周期总数：33。
发生 RAW 冲突的指令组合：
ADDIU $r1,$r0,A 和 LW $r2,0($r1)
LW $r2,0($r1)和 ADD $r4,$r0,$r2
ADD $r4,$r0,$r2 和 SW $r4,0($r1)
LW $r6,4($r1)和 ADD $r8,$r6,$r1
MUL $r12,$r10,$r1 和 ADD $r16,$r12,$r1
ADD $r16,$r12,$r1 和 ADD $r18,$r16,$r1
ADD $r18,$r16,$r1 和 SW $r18,16($r1)
LW $r20 8($r1)和 MUL $r22,$r20,$r14

5）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。

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

.data
A:
.word 4,6,8
```

6）载入 after-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。

汇总：

执行周期总数：18

ID段执行了15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：1　　占周期总数的百分比：5.555555%

其中：

load停顿：0　　占所有RAW停顿的百分比：0%

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：5.555555%
停顿周期总数：2　　占周期总数的百分比：11.11111%

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%

执行周期总数：18。
7）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。
调度后的性能提升了 33/18=1.83 倍。
指令调度可以消除部分的数据冲突，减少了停顿的周期数，提高了 CPU 使用率。

(5) 用延迟分支技术减少分支指令对性能的影响：
1）在 MIPSsim 中载入 branch.s 样例程序。
2）关闭延迟分支功能。这是通过在“配置”->“延迟槽”选项来实现的。
3）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。
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%

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：

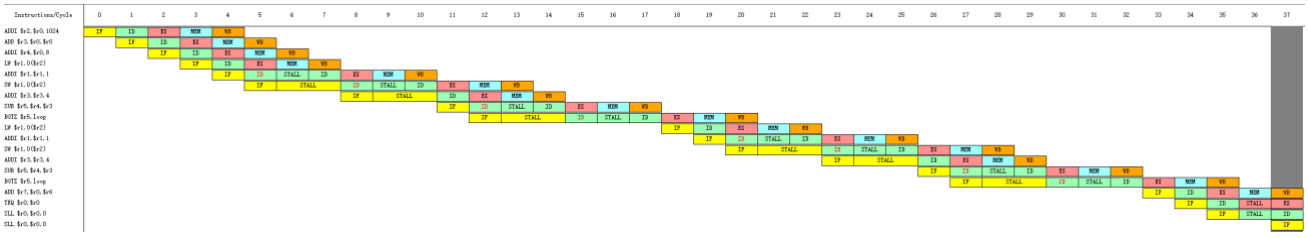
加法：0　　占浮点指令数的百分比：0%

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%



发生分支延迟的时刻：15，执行周期总数：38。

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

```
.text
main:
ADDI  $r2,$r0,1024
ADD   $r3,$r0,$r0
ADDI  $r4,$r0,8
loop:
LW    $r1,0($r2)
ADDI  $r3,$r3,4
ADDI  $r1,$r1,1
SUB   $r5,$r4,$r3
SW    $r1,0($r2)
BGTZ  $r5,loop
ADD   $r7,$r0,$r6
TEQ   $r0,$r0
```

5）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。

汇总：
执行周期总数：25
ID段执行了19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：4　　占周期总数的百分比：16%
其中：
load停顿：2　　占所有RAW停顿的百分比：50%
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：4%
停顿周期总数：5　　占周期总数的百分比：20%

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：4　　占指令总数的百分比：21.05263%

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：

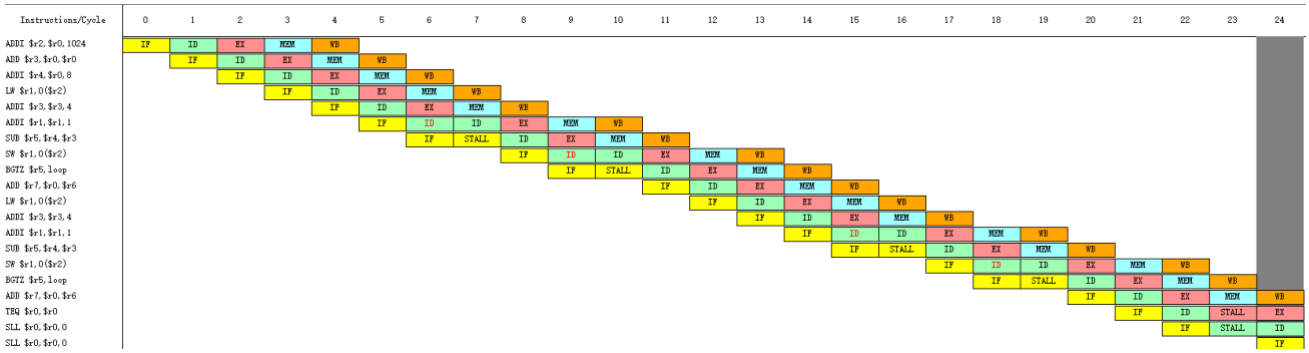
加法：0　　占浮点指令数的百分比：0%

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：5.263158%



执行周期总数：25。

6）对比不采用延迟分支和采用延迟分支两种情况下的时钟周期图，比较两种情况下的性能之间的不同，论述延迟分支对提高 CPU 性能的作用。

延迟分支技术是由编译器通过重排指令序列，在分支指令后紧跟一条或几条延迟槽指令，不管分支是否成功，都顺序执行延迟槽中的指令，从而逻辑上“延长”分支指令的执行时间，减少甚至消除了控制停顿，从而提高 CPU 性能。