实验 3　指令调度和延迟分支

3.1　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) 熟练掌握用指令调度技术来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。

3.2　实验平台

实验平台采用指令级和流水线操作级模拟器 MIPSsim。
设计：张晨曦教授，版权所有。

3.3　实验内容和步骤

首先要掌握 MIPSsim 模拟器的使用方法(见 1.4 节)。

(1) 启动 MIPSsim。

(2) 根据 2.5 节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，掌握各流水寄存器的含义(双击各段，就可以看到各流水寄存器的内容)。

(3) 选择“配置”→“流水方式”选项，使模拟器工作于流水方式下。

(4) 用指令调度技术解决流水线中的结构冲突与数据冲突。

① 启动 MIPSsim。

② 选择 MIPSsim 的“文件”→“载入程序”选项来加载 schedule.asm(在模拟器所在文件夹下的“样例程序”文件夹中)。

③ 关闭定向功能。这是通过在“配置”菜单中关闭“定向”(使该项前面没有√号)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。

⑤ 采用指令调度技术对程序进行指令调度，消除冲突。将调度后的程序保存到 after-schedule. asm 中。

⑥ 载入 after-schedule. asm。

⑦ 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。

⑧ 根据记录结果，比较调度前和调度后的性能。论述指令调度对于提高 CPU 性能的作用。

(5) 用延迟分支减少分支指令对性能的影响。

① 启动 MIPSsim。

② 载入 branch. asm。

③ 关闭延迟分支功能。这是通过选择“配置”→“延迟槽”选项来实现的。

④ 执行该程序。观察并记录发生分支延迟的时刻。

⑤ 记录执行该程序所用的总时钟周期数。

⑥ 假设延迟槽为一个，对 branch. asm 进行指令调度，然后保存到 delayed-branch. asm 中。

⑦ 载入 delayed-branch. asm。

⑧ 打开延迟分支功能。

⑨ 执行该程序，观察其时钟周期图。

⑩ 记录执行该程序所用的总时钟周期数。

⑪ 对比上述两种情况下的时钟周期图。

⑫ 根据记录结果，比较没有采用延迟分支和采用了延迟分支的性能，论述延迟分支对于提高 CPU 性能的作用。

3.4　MIPSsim 使用手册

见实验 1 的 1.4 节。

3.5　相关知识：指令调度和延迟分支

3.5.1　指令调度

为了减少停顿，对于无法用定向技术解决的数据冲突，可以通过在编译时让编译器重新组织指令顺序来消除冲突，这种技术称为“指令调度”或“流水线调度”。实际上，对于各种冲突，都有可能用指令调度来解决。

下面通过一个例子来进一步说明。考虑以下表达式：

A＝B＋C；

D＝E－F；

表3.1左边是这两个表达式编译后所形成的代码。在这个代码序列中,DADD Ra,Rb,Rc与LD Rc,C之间存在数据冲突,DSUB Rd,Re,Rf与LD Rf,F之间也是如此。为了保证流水线能正确执行调度前的指令序列,必须在指令的执行过程中插入两个停顿周期(分别在DADD和DSUB执行前)。而在调度后的指令序列中,加大了DADD和DSUB指令与LD指令的距离。通过采用定向,可以消除数据冲突,因而不必在执行过程中插入任何停顿周期。

表3.1　调度前后的指令序列

调度前的代码		调度后的代码	
LD	Rb,B	LD	Rb,B
LD	Rc,C	LD	Rc,C
DADD	Ra,Rb,Rc	LD	Re,E
SD	Ra,A	DADD	Ra,Rb,Rc
LD	Re,E	LD	Rf,F
LD	Rf,F	SD	Ra,A
DSUB	Rd,Re,Rf	DSUB	Rd,Re,Rf
SD	Rd,D	SD	Rd,D

3.5.2　延迟分支

在流水线中,控制冲突可能会比数据冲突造成更多的性能损失,所以同样需要得到很好的处理。执行分支指令的结果有两种:一种是分支"成功",PC值改变为分支转移的目标地址;另一种则是"不成功"或者"失败",这时PC的值保持正常递增,指向顺序的下一条指令。对分支指令"成功"的情况来说,是在条件判定和转移地址计算都完成后,才改变PC值。

处理分支指令最简单的方法是"冻结"或者"排空"流水线,即一旦在流水线的译码段ID检测到分支指令,就暂停其后的所有指令的执行,直到分支指令到达MEM段、确定出是否成功并计算出新的PC值为止。然后,按照新的PC值取指令。如图3.1所示,这种方法的优点在于其简单性,它给流水线带来了3个时钟周期的延迟(称为分支延迟)。

分支指令	IF	ID	EX	MEM	WB					
分支目标指令		**IF**	**stall**	**stall**	IF	ID	EX	MEM	WB	
分支目标指令+1						IF	ID	EX	MEM	WB
分支目标指令+2							IF	ID	EX	MEM

图3.1　简单处理分支指令:分支成功的情况

为了减少分支延迟,可以采用延迟分支技术。这种技术的主要思想是从逻辑上"延长"分支指令的执行时间,把延迟分支看成由原来的分支指令和若干延迟槽构成。不管分支是否成功,都要按顺序执行延迟槽中的指令。在采用延迟分支的实际计算机中,绝大多

数的延迟槽都是一个。MIPSsim 模拟器的也是按这样处理的，即

分支指令
延迟槽
后继指令

在这种情况下，流水线的执行情况如图 3.2 所示。可以看出，只要分支延迟槽中的指令是有用的，流水线中就没有出现停顿，这时延迟分支的方法能很好地减少分支延迟。

分支失败	分支指令 i	IF	ID	EX	MEM	WB					
	延迟槽指令 $i+1$		IF	ID	EX	MEM	WB				
	指令 $i+2$			IF	ID	EX	MEM	WB			
	指令 $i+3$				IF	ID	EX	MEM	WB		

分支成功	分支指令 i	IF	ID	EX	MEM	WB					
	延迟槽指令 $i+1$		IF	ID	EX	MEM	WB				
	分支目标指令 j			IF	ID	EX	MEM	WB			
	分支目标指令 $j+1$				IF	ID	EX	MEM	WB		

图 3.2　延迟分支的执行情况

放入延迟槽中的指令是由编译器来选择的。实际上，延迟分支能否带来好处完全取决于编译器能否把有用的指令调度到延迟槽中，这也是一种指令调度技术。常用的调度方法有 3 种：从前调度、从目标处调度和从失败处调度。如图 3.3 所示，上面的代码是调度前的，下面的代码是调度后的。

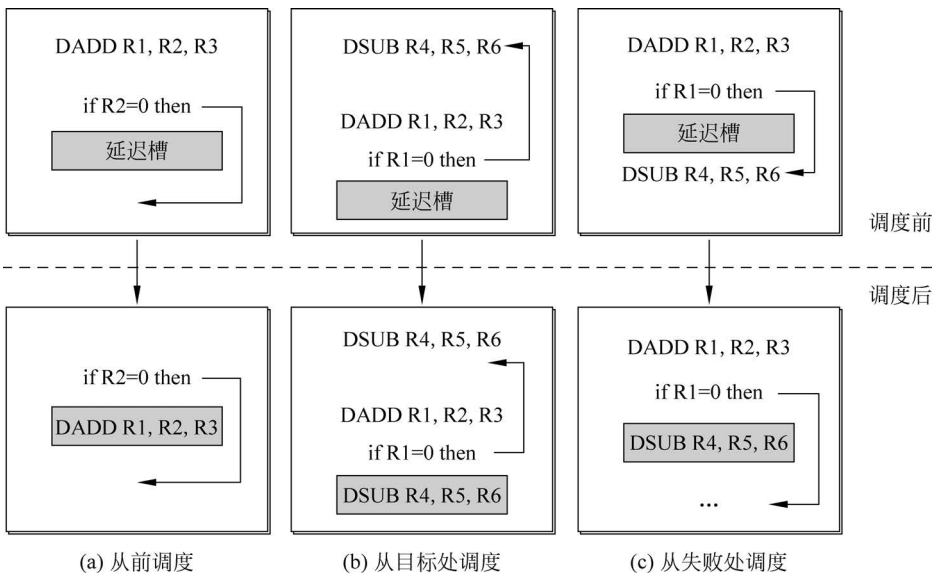

图 3.3　调度分支指令的 3 种常用方法

图 3.3(a)表示的是从前调度，它是把位于分支指令之前的一条独立的指令移到延迟槽。当无法采用从前调度时，就采用另外两种方法。图 3.3(b)表示的是从目标处调度，

它是把目标处的指令复制到延迟槽。同时,还要修改分支指令的目标地址,如图3.3(b)中的箭头所示。之所以是复制到延迟槽,而不是把该指令移过去,是因为从别的路径可能也要执行该指令,从目标处调度实际上是猜测了分支是成功的。所以当分支成功概率比较高时(例如循环转移),采用这种方法比较好;否则,采用从失败处调度比较好(见图3.3(c))。需要注意的是,当猜测错误时,要保证图3.3(b)和图3.3(c)中调度到延迟槽中的指令的执行不会影响程序的正确性(当然,这时延迟槽中的指令是无用的)。在图3.3(b)和图3.3(c)的指令序列中,由于分支指令是使用R1来判断的,所以不能把产生R1的值的DADD指令调度到延迟槽。