实验一 流水线及流水线中的冲突

实验目的

- 1. 加深对计算机流水线基本概念的理解;
- 2. 理解MIPS结构如何用5段流水线来实现,理解各段的功能和基本操作;
- 3. 加深对数据冲突、结构冲突的理解,理解这两类冲突对CPU性能的影响;
- 4. 进一步理解解决数据冲突的方法,掌握如何应用定向技术来减少数据冲突引起的停顿。
- 5. 加深对指令调度和延迟分支技术的理解;
- 6. 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;
- 7. 进一步理解指令调度技术和延迟分支技术对CPU性能的改进。

实验平台

指令级和流水线操作级模拟器MIPSsim,

实验内容和步骤

首先要掌握MIPSsim模拟器的使用方法。

一、流水线及流水线中的冲突观察

- 1. 启动MIPSsim。
- 2. 根据预备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。(用鼠标双击各段,就可以看到各流水寄存器的内容)
- 3. 熟悉MIPSsim模拟器的操作和使用方法。

可以先载入一个样例程序(在本模拟器所在的文件夹下的"样例程序"文件夹中),然后分别以单步执行一个周期、执行多个周期、连续执行、设置断点等的方式运行程序,观察程序的执行情况,观察CPU中寄存器和存储器的内容的变化,特别是流水寄存器内容的变化。

- 4. 勾选配置菜单中的"流水方式", 使模拟器工作于流水方式下。
- 5. 观察程序在流水线中的执行情况, 步骤如下:
 - (1) 用MIPSsim的"文件"菜单中的"载入程序"来加载pipeline.s(在模拟器所在文件夹下的"样例程序"文件夹中);
 - (2) 关闭定向功能。这是通过在"配置"菜单中去选"定向"(即使得该项前面没有"√"号)来实现的;
 - (3) 用单步执行一周期的方式("执行"菜单中,或用F7)执行该程序,观察每一周期中,各段流水寄存器内容的变化、指令的执行情况(代码窗口)以及时钟周期图;

(1)	小块/C对数10人叶幼田期叶	各段分别正在处理的指令是:
(4)	3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	育权分别正任处理的相令定。

IF: <u>ADDI \$r6, \$r0, 8</u>

ID: ADD \$r2, \$r1, \$r0

EX: none

MEM: BGEZAL \$r1, func

WB: none

画出这时的时钟周期图。

Instructions/Cycle	0	1	2	3	4	5	6	7	8	9
ADDIV \$r8,\$r0,76	IF	ID	EX	MEM	WB					
LW \$r1,0(\$r8)		IF	ID	STALL	ID	EX	MEM	WB]	
BGEZAL \$r1, func			IF	STA	LL	ID	STALL	ID	EX	MEM
ADD \$r2,\$r1,\$r0									IF	ID
ADDI \$r6,\$r0,8										IF
I	l									

6. 这时各流水寄存器中的内容为:

(十进制表示)

IF/ID.IR: <u>537264136</u>

IF/ID.NPC: <u>36</u>

ID/EX.A: ____5

ID/EX.B: _____0

ID/EX.Imm: ____4

ID/EX.IR: 2101280

EX/MEM.ALUo: 0

EX/MEM.IR: ______70320132

MEM/WB.LMD: _____0

MEM/WB.ALUo: _____0

7. 观察和分析结构冲突对CPU性能的影响,步骤如下:

- (1) 加载structure hz.s(在模拟器所在文件夹下的"样例程序"文件夹中);
- (2) 执行该程序, 找出存在结构冲突的指令对以及导致结构冲突的部件;

fadd导致ADD.D \$f2, \$f0, \$f1与 ADD.D \$f3, \$f0, \$f1有结构冲突 ID导致ADD.D \$f3, \$f0, \$f1与 ADD.D \$f4, \$f0, \$f1有结构冲突 fadd导致ADD.D \$f3, \$f0, \$f1与 ADD.D \$f4, \$f0, \$f1有结构冲突 ID导致ADD.D \$f4, \$f0, \$f1与 ADD.D \$f5, \$f0, \$f1有结构冲突 fadd导致ADD.D \$f4, \$f0, \$f1与 ADD.D \$f5, \$f0, \$f1有结构冲突 ID导致ADD.D \$f5, \$f0, \$f1

.

fadd导致ADD.D \$f7, \$f0, \$f1与 ADD.D \$f8, \$f0, \$f1有结构冲突 ID导致ADD.D \$f8, \$f0, \$f1与 ADD.D \$f9, \$f0, \$f1有结构冲突 fadd导致ADD.D \$f8, \$f0, \$f1与 ADD.D \$f9, \$f0, \$f1有结构冲突 ID导致ADD.D \$f9, \$f0, \$f1与 TEQ \$r0, \$r0有结构冲突

(3) 记录由结构冲突引起的停顿时钟周期数, 计算停顿时钟周期数占总执行周期数的 百分比;

结构停顿: 35, 占比67.30769%

- (4) 把浮点加法器的个数改为6个;
- (5) 再次重复上述(1)~(3)的工作; 浮点加法器改为6个之后,有0个结构冲突,占比0%
- 8. 观察数据冲突并用定向技术来减少停顿, 步骤如下:
 - (1) 把浮点加法器的个数改为1个;
 - (2) 加载data hz.s (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"菜单中去选"定向"(即使得该项前面没有"√"号)来实现的;
- (4) 用单步执行一个周期的方式(F7) 执行该程序,同时查看时钟周期图,列出在什么时刻发生了RAW(先写后读)冲突;

在instruction/cycle

3/5/6/8/9/12/13/16/17/19/20/24/25/27/28/31/32/35/36/38/39/43/44/46/47/50/51/54/55/57/58 发生了RAW冲突

(5) 记录数据冲突引起的停顿时钟周期数以及程序执行的总时钟周期数,计算停顿时钟周期数占总执行周期数的百分比;

数据冲突引起的停顿时钟周期数: 31程序执行的总时钟周期数: 65占比47.69231%

- (6) 复位CPU;
- (7) 打开定向功能。这是通过在"配置"菜单中勾选"定向"(即使得该项前面有一个"√"号)来实现的;

(8) 用单步执行一周期的方式(F7) 执行该程序,同时查看时钟周期图,列出在什么时刻发生了RAW(先写后读)冲突,并与(3)的结果进行比较;

在instruction/cycle 4/9/12/17/21/24/29/33/36 发生了RAW冲突,占比20.93023%,和(3)的结果相比较,说明运用'定向'技术能够在一定程度上避免RAW冲突

(9) 记录数据冲突引起的停顿时钟周期数以及程序执行的总时钟周期数。计算采用定向技术后性能提高的倍数。

数据冲突引起的停顿时钟周期数: 9

程序执行的总时钟周期数: 43

性能提高 (1/43)/(1/65)=1.5116倍, (即用定向技术时性能是不用定向技术时性能的1.5116倍)

二、指令调度和延迟分支

- 1. 启动MIPSsim。
- 2. 根据预备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。(用鼠标双击各段,就可以看到各流水寄存器的内容)
- 3. 勾选配置菜单中的"流水方式",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. 通过"配置"菜单中的"常规配置"项把加法、乘法、除法部件的个数设置为两个, 把它们的延迟时间都设置为3个时钟周期;
 - 3. 用MIPSsim的"文件"菜单中的"载入程序"来加载schedule.asm(在模拟器所在文件夹下的"样例程序"文件夹中);
 - 4. 关闭定向功能。这是通过在"配置"菜单中去选"定向"(即使得该项前面没有"√"号) 来实现的。
 - 5. 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;

LD \$r2,0(\$r1) and ADDIU \$r1,\$r0,56有RAW冲突

LD \$r2, 0(\$r1) and ADD \$r4, \$r0, R2有RAW冲突

ADD \$r4, \$r0, R2 and SW \$r4, 0(\$r1)有RAW冲突

LW \$r6, 4(\$r1) and ADD \$r8, \$r6, \$r1有RAW冲突

MUL \$r12, \$r10, \$r1 and ADD \$r16, \$r12, \$r1有RAW冲突

ADD \$r16, \$r12, \$r1 and ADD \$r18, \$r16, \$r1有RAW冲突

ADD \$r18, \$r16, \$r1 and SW \$r18, 16(\$r1)有RAW冲突

LW \$r20, 8(\$r1) and MUL \$r22, \$r20, \$r14有RAW冲突

各种停顿发生的次数:

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%

程序执行的总时钟周期数为33

6. 采用指令调度技术对程序进行指令调度,消除冲突。将调度后的程序放到after-schedule.asm中;

(手工进行指令调度?)

after-schedule.asm代码如下:

.text

main:

ADDIU \$r1,\$r0,A

MUL \$r24,\$r26,\$r14

MUL \$r12,\$r10,\$r1

LW \$r2,0(\$r1)

ADD \$r16,\$r12,\$r1

LW \$r6,4(\$r1)

ADD \$r18,\$r16,\$r1

ADD \$r8,\$r6,\$r1

ADD \$r4,\$r0,\$r2

LW \$r20,8(\$r1)

SW \$r4,0(\$r1)

SW \$r18,16(\$r1)

MUL \$r22,\$r20,\$r14

TEQ \$r0,\$r0

.data

A:

.word 4,6,8

结果如下:

RO = 0R1 = 56R2 = 4 R3 = 0R4 = 4R5 = 0R6 = 6R7 = 0R8 = 62R9 = 0R10 = 0R11 = 0R12 = 0R13 = 0R14 = 0R15 = 0R16 = 56R17 = 0R19 = 0R20 = 8R21 = 0R22 = 0R23 = 0R24 = 0R25 = 0R26 = 0R27 = 0R28 = 0

可验证其正确性。

- 7. 载入after-schedule.asm;
- 8. 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; 总共21个执行周期。
- 9. 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。

调度前要33个时钟周期才能执行完毕,调度后只要21个周期就可以执行完毕,后者性能是前者的(1/21)/(1/33)= 1.5714倍。

指令调度让指令顺序重新组织顺序从而消除部分的数据冲突,指令调度的优劣直接 影响着cpu性能的发挥好坏,好的指令调度可以让停顿周期大幅度减少。

- 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。这是通过在"配置"菜单中去选"延迟分支"来实现的;
 - 4. 执行该程序,观察并记录发生分支延迟的时刻,保存下其时钟周期图(可用拷屏的方法);

在 instruction/cycle 15 发生分支延迟

20									WB	MEM	EX	日	IF	
19									MEM	EX	fl	IF		•
18									EX	fl fl	IF		J	
17								WB	A	IF		ı		
16								MEM	STALL					
15								EX	A					
14							WB	ID	н					
13						#B	MEM	STALL	STALL					
12						MEM	EX	ID	IF					
11						EX	f	IF						
10					WB.	fi	П		ı					
თ					MEM	STALL	STALL							
ω					EX	А	IF							
t-				WB	П	1		•						
Ð			W.B	MEM	STALL	STALL								
ľ		WB	MEM	EX	IID	IF								
₹*	WB	MEM	EX	Π	IF									
ო	MEM	EX	A	IF		•								
8	EX	П	IF											
	А	IF		-										
0	IF													_
Instructions/Cycle	ADDI \$r2, \$r0, 1024	ADD \$r3, \$r0, \$r0	ADDI \$r4, \$r0, 8	LW \$r1,0(\$r2)	ADDI \$r1, \$r1, 1	SW \$r1,0(\$r2)	ADDI \$r3, \$r3, 4	SUB \$r5, \$r4, \$r3	BGTZ \$r5, loop	ADD \$r7, \$r0, \$r6	L# \$r1,0(\$r2)	ADDI \$r1, \$r1, 1	S# \$r1,0(\$r2)	_

88		i	i	i	i	i	i	i	ı	#B	EX	A	IF
35									#B	MEM	STALL	STALL	
34									MEM	EX	A	IF S	
33									EX	日日	IF		
32								W.B	f	IF			
31								MEM	STALL				
8								EX	П				
53							WB	日					
88						#B	MEM	STALL	STALL				
27						MEM	EX	П	IF				
56						EX	OI OI	IF					
52					WB	f							
24					MEM	STALL	STALL						
53					EX	日	IF						
55				WB	fl								
21			#B	MEM	STALL	STAIL							
20		WB	MEM	EX	ID	IF							
19		MEM	EX	A	IF		J						
18		EX	A	IF		I							
11	W.B	A	IF		J								
16	MEM	STAIL		I									
Instructions/Cycle	SUB \$r5, \$r4, \$r3	BGTZ \$r5, loop	ADD \$r7, \$r0, \$r6	L# \$r1,0(\$r2)	ADDI \$r1, \$r1, 1	SW \$r1,0(\$r2)	ADDI \$r3, \$r3, 4	SUB \$r5, \$r4, \$r3	BGTZ \$r5,100p	ADD \$r7, \$r0, \$r6	TEQ \$r0, \$r0	SIL \$r0, \$r0, 0	SIT \$r0, \$r0, 0

- 5. 记录执行该程序所花的总时钟周期数; 38个时钟周期。
- 6. 假设延迟槽为一个,对branch.asm进行指令调度,然后存到delayed-branch.asm中;

调度后delayed-branch.asm代码如下:

.text

main:

ADDI \$r2,\$r0,1024

ADD \$r3,\$r0,\$r0

ADDI \$r4,\$r0,8

LW \$r1,0(\$r2)

loop:

ADDI \$r3,\$r3,4

ADDI \$r1,\$r1,1

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.asm;
- 8. 打开延迟分支功能;
- 9. 执行该程序,观察其时钟周期图,保存下其时钟周期图; delayed-branch时钟周期图如下:

17 18 19 20	HEH WB	EX HEN WB ID ID EX HEN IF STALL ID EX IF IF ID IF SA 24 IF ID	E.A. WAW
16			W.B. W.B. W.B. W.B. W.B. W.B. W.B. W.B.
15	WB WB	STAIL	8.8 W.
14		20 14 14 18 18 18 18 18 18 18 18 18 18 18 18 18	MEM EX EX IID
13	WB MEM EX EX EX IP	19	MW EX XI III
12	MEAN EX EX EX ID II	8	WBW NEW ID ID STAIL
11	## ## ## ## ## ## ## ## ## ## ## ## ##	11	MEM TEX TO THE TEXT TO THE TEX
01	WB MEM ID STALL	91	
ø	MEM EX ID ID	15	WB WEW ID ID STAIL
60	EX E	4.	WB WEW WEW TO THE TRANSPORT OF THE TRANS
۲	WB WEM ID ID ID STAIL	13	WB MEM MEM ID ID ID
9	WB MEM EX	21	MEM EX ID ID
Ŋ	WB WBW WEW ID II	Ξ	EX KB
₹	WE KEY LID TO THE LID THE LID TO	10	HEW LD STALL
т	MEX EX LID	o,	HEM TID ID
2	N C C C C C C C C C C C C C C C C C C C	00	IP II
	A P	۲	WB WBW ID ID STAIL
• —	H	ω	WB NEW LEX ID IT
Instructions/Cycle	ADD \$42, \$40, 1024 ADD \$43, \$40, \$70 ADD \$44, \$70, \$70 ADD \$44, \$70, \$70 ADD \$73, \$43, 4 ADD \$73, \$44, \$73 SUB \$75, \$44, \$73 SUB \$75, \$70, 10 BGTZ \$75, 10 op LW \$71, 0 (\$72) ADD \$73, \$73, 4 ADD \$73, \$73, 4 ADD \$74, \$74, \$73 ADD \$75, \$74, \$74 ADD \$75, \$74, \$74 ADD \$75, \$75, 10	SUB \$r5, \$r4, \$r3 SW \$r1, 0 (\$r2) BGTZ \$r5, loop LW \$r1, 0 (\$r2) ADD \$r7, \$r0, \$r6 Instructions/Cycle	ADDI \$x-4, \$x-0, 8 LW \$x-1, 0 (\$x-2) ADDI \$x-3, \$x-3, 4 ADDI \$x-1, \$x-1, 1 SUB \$x-5, \$x-4, \$x-3 SW \$x-1, 0 (\$x-2) BGTZ \$x-5, 1-0-p LW \$x-1, 0 (\$x-2) ADDI \$x-3, \$x-3, 4 ADDI \$x-1, \$x-1, 1 SUB \$x-5, \$x-4, \$x-3 SW \$x-1, 0 (\$x-2) BGTZ \$x-5, 1-0-p LW \$x-1, 0 (\$x-2) BGTZ \$x-5, 1-0-p LW \$x-1, 0 (\$x-2) BGTZ \$x-5, 1-0-p LW \$x-1, 0 (\$x-2) ADDI \$x-7, \$x-0, \$x-6 ADDI \$x-7, \$x-7, \$x-7 ADDI \$x-7, \$x-7, \$x-7 ADDI \$x-7, \$x-7, \$x-7 ADDI \$x

10. 对比上述两种情况下的时钟周期图;

在没打开分支延迟槽时,BGTZ \$r5,loop指令停留在ID部件时,ADD \$r7,\$r0,\$r6指令不会被fetch;打开分支预测槽后,BGTZ \$r5,loop在ID部件时,CPU提前取ADD \$r7,\$r0,\$r6指令,充分利用CPU的各个部件。在循环次数多的时候效果比较好。

11. 根据记录结果,比较没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

流水线中,分支指令执行时因为确定下一条指令的目标地址一般要到第2步以后,在目标确定前流水线的取指级是不能工作的,即整个流水线就阻塞了一个时间片,为了利用这个时间片,在体系结构的层面上规定跳转指令后面的一个时间片为分支延迟槽。位于分支延迟槽中的指令总是被执行,与分支发生与否没有关系。这样就有效利用了一个时间片,消除了流水线的一个"气泡",从而充分发挥了CPU的性能。