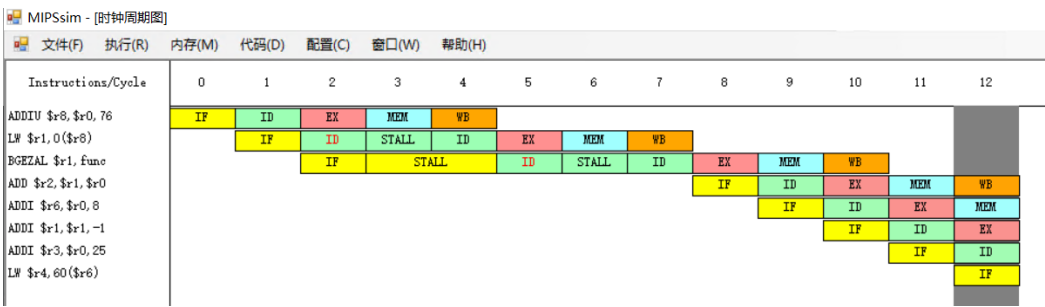


计算机系统结构实验报告

班级	计科 1 班	实验日期	2024.3.5	实验成绩	
姓名	李梓涵	学号	34520212201574		
实验名称	实验 2 流水线与流水线中的冲突				
实验目的、要求	<p>实验目的：</p> <p>1、 加深对计算机流水线基本概念的理解；</p> <p>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；</p> <p>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。</p> <p>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。</p>				
实验内容及步骤及结果	<p>一、 观察程序在流水线中的执行情况（pipeline.s）</p> <p>1. pipeline.s 程序执行到第 13 个时钟周期时，各段分别正在处理的指令是</p> <p>IF: LW \$r4, 60(\$r6)</p> <p>ID: ADDI \$r3, \$r0, 25</p> <p>EX: ADDI \$r1, \$r1, -1</p> <p>MEM: ADDI \$r6, \$r0, 8</p> <p>WB: ADD \$r2, \$r1, \$r0</p> <div></div> <p>2. 这时各流水寄存器中的内容为</p>				

流水寄存器

☒ 十进制 ☐ 十六进制

IF/ID. IR = 2361655356
 IF/ID. NPC = 48
 ID/EX. A = 0
 ID/EX. B = 0
 ID/EX. Imm = 25
 ID/EX. IR = 537067545
 EX/MEM. ALUo = 4
 EX/MEM. B = 0
 EX/MEM. IR = 539099135
 MEM/WB. LMD = 0
 MEM/WB. ALUo = 8
 MEM/WB. IR = 537264136

二、观察和分析结构冲突对 CPU 性能的影响 (structure_hz.s)

1. 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存在结构冲突的指令：

C:\Users\Administrator\Desktop\comparch-simulator\MIPSSim\

地址	断点标记	机器码	流水段	符号指令
main		0x46210080		ADD.D \$f2,\$f0,\$f1
0x00000004		0x462100C0		ADD.D \$f3,\$f0,\$f1
0x00000008		0x46210100		ADD.D \$f4,\$f0,\$f1
0x0000000C		0x46210140		ADD.D \$f5,\$f0,\$f1
0x00000010		0x46210180		ADD.D \$f6,\$f0,\$f1
0x00000014		0x462101C0		ADD.D \$f7,\$f0,\$f1
0x00000018		0x46210200		ADD.D \$f8,\$f0,\$f1
0x0000001C		0x46210240	WB	ADD.D \$f9,\$f0,\$f1
0x00000020		0x00000034	EX	TEQ \$r0,\$r0
0x00000024		0x00000000	ID	SLL \$r0,\$r0,0
0x00000028		0x00000000	IF	SLL \$r0,\$r0,0
0x0000002C		0x00000000		SLL \$r0,\$r0,0

导致结构冲突的部件：fadd

2. 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

统计

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

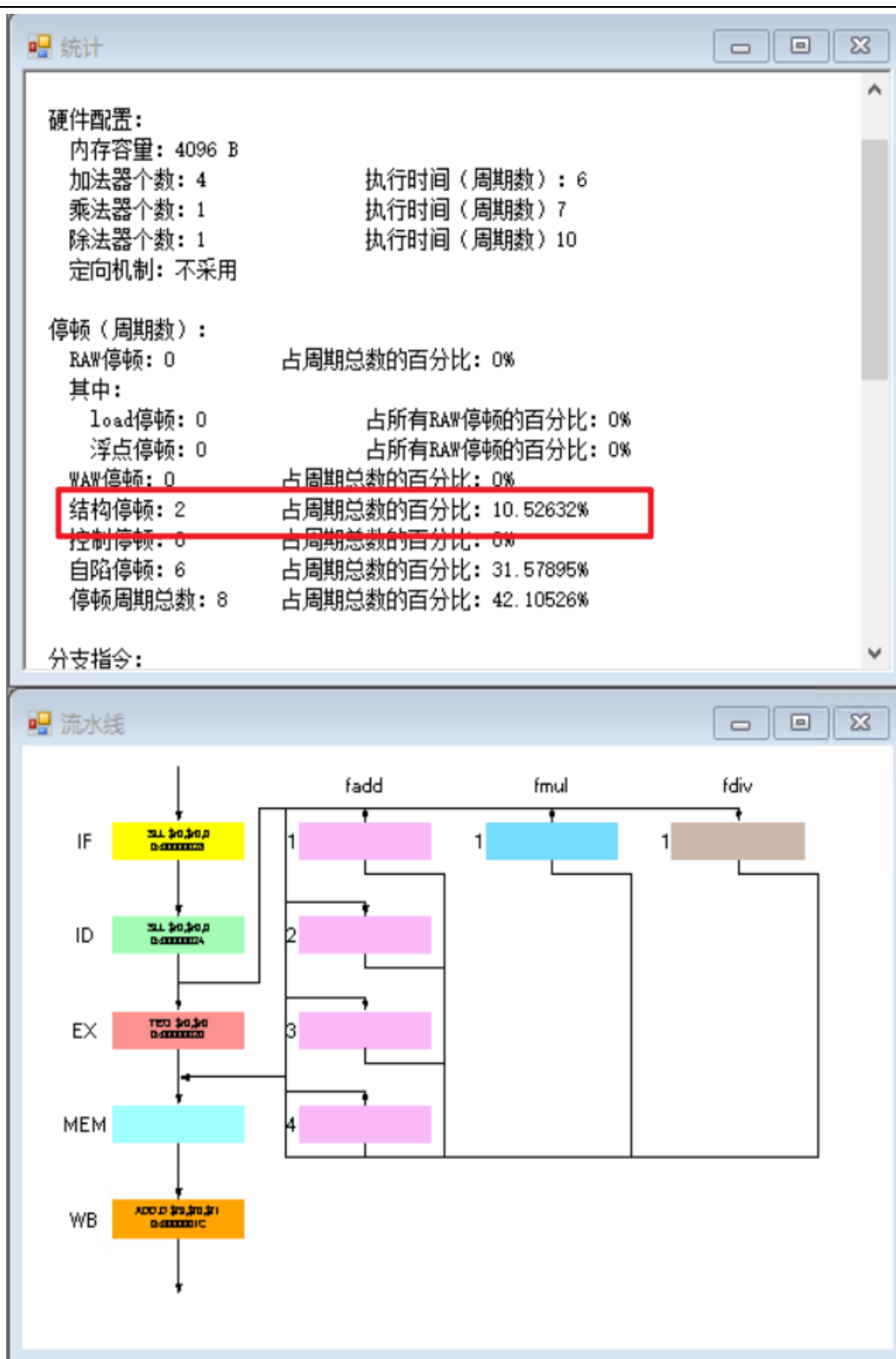
执行时间 (周期数): 7

执行时间 (周期数): 10

停顿 (周期数):
RAW停顿: 0 占周期总数的百分比: 0%
其中:
load停顿: 0 占有RAW停顿的百分比: 0%
浮点停顿: 0 占有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: 35 占周期总数的百分比: 67.30769%
控制停顿: 0 占周期总数的百分比: 0%
自陷停顿: 6 占周期总数的百分比: 11.53846%
停顿周期总数: 41 占周期总数的百分比: 78.84615%

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%

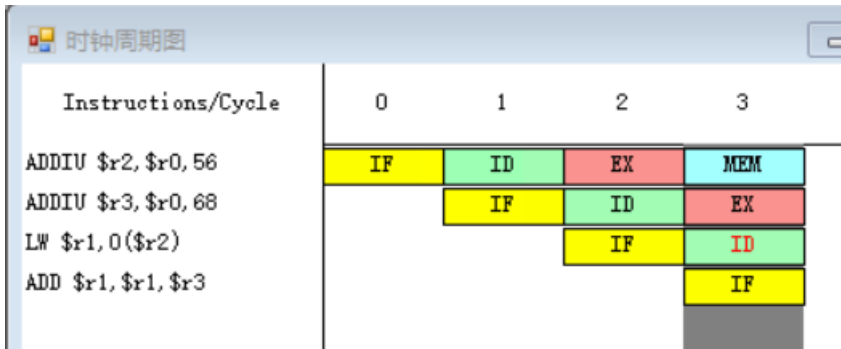
3. 把浮点加法器的个数改为 4 个



4. 分析结构冲突对 CPU 性能的影响, 讨论解决结构冲突的方法
 结构冲突使流水处理机出现流水线气泡, 降低了流水线工作效率。可以对功能部件进行全流水处理或者设置足够多的重复资源, 以消除结构冲突。如可以分别设置单独的指令存储器和数据存储器, 或者采用两个分离的 Cache

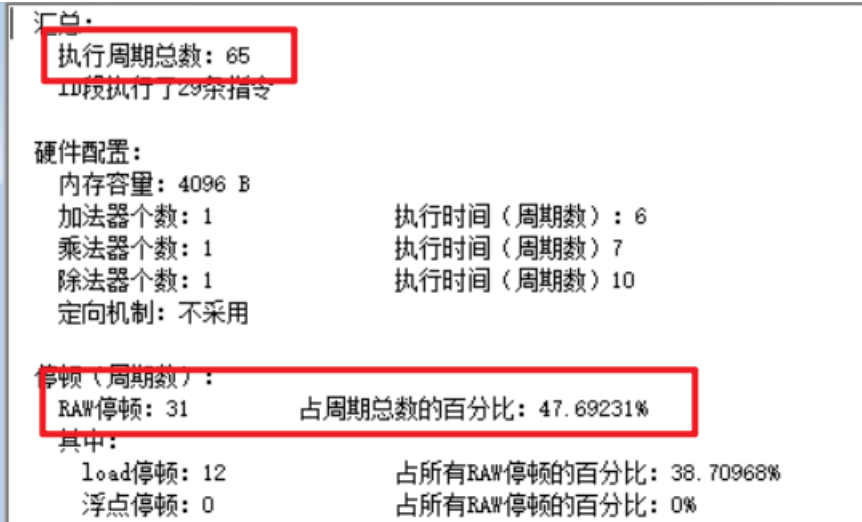
三、观察数据冲突并用定向技术来减少停顿 (data_hzs)

1. 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

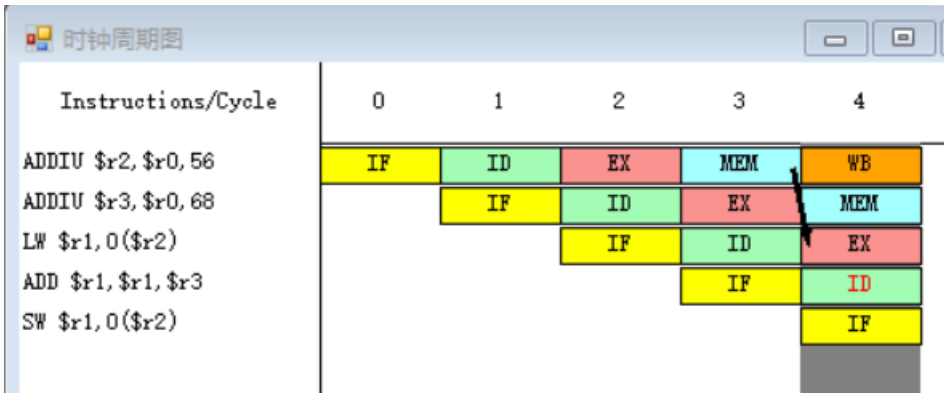


如图，在第 4 个时钟周期时发生了 RAW（先写后读）冲突，LW 指令先读取了 r2 寄存器，之后 ADDIU 指令进行了写入 r2 寄存器的操作。

2. 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。



3. 打开定向功能。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 1 的结果进行比较



如图，在第 5 个时钟周期发生了 RAW（先写后读）冲突。此时是 ADD 指令与第二个 ADDIU 指令发生了冲突。

与 1 中结果相比，发生冲突的时间向后推迟了一个周期，定向技术消除

了第 4 个时钟周期的 RAW 冲突。

4. 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。

汇总:

执行周期总数: 43

10段执行了29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: 采用

停顿(周期数):

RAW停顿: 9

占周期总数的百分比: 20.93023%

其中:

Load停顿: 6

占有所有RAW停顿的百分比: 66.66666%

浮点停顿: 0

占有所有RAW停顿的百分比: 0%

性能提高倍数: $65/43=1.512$

四、 补充实验 1

1. 立方和的两种计算公式:

$$a^3+b^3$$

$$(a+b)(a^2-ab+b^2)$$

a 和 b 都是双精度浮点数, 初始值存放在 F1、F2 寄存器中, 计算结果存放在 F3 寄存器中。从 F4 开始使用寄存器, 数量不限。

编写程序并验证, 在流水线 CPU 中, 立方和公式的两种计算方法应该如何安排指令使得结构停顿占周期总数的百分比最小? 两种方法的性能之比? (结构冲突)

2. 在编写代码时, 我尽量在相邻的代码中选用不同的处理器, 以减小数据冲突的可能性。

3. a^3+b^3

.text

main:

MUL.S \$f4, \$f1, \$f1

MUL.S \$f5, \$f2, \$f2

MUL.S \$f6, \$f4, \$f1

MUL.S \$f7, \$f5, \$f2

ADD.S \$f3, \$f6, \$f7

TEQ \$r0, \$r0

TEQ \$r0, \$r0

TEQ \$r0, \$r0

TEQ \$r0, \$r0

汇总:

执行周期总数: 40
ID段执行了7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: 10 占周期总数的百分比: 25%

其中:

load停顿: 0	占所有RAW停顿的百分比: 0%
浮点停顿: 10	占所有RAW停顿的百分比: 100%
WAW停顿: 0	占周期总数的百分比: 0%
结构停顿: 16	占周期总数的百分比: 40%
控制停顿: 0	占周期总数的百分比: 0%
自陷停顿: 6	占周期总数的百分比: 15%
停顿周期总数: 32	占周期总数的百分比: 80%

结构停顿数量: 16, 占周期总数百分比: 40%

4. $(a+b)(a^2-ab+b^2)$

.text

main:

ADD.S \$f9, \$f1, \$f2

MUL.S \$f6, \$f1, \$f2

MUL.S \$f4, \$f1, \$f1

MUL.S \$f5, \$f2, \$f2

ADD.S \$f7, \$f4, \$f5

SUB.S \$f8, \$f7, \$f6

MUL.S \$f3, \$f9, \$f8

TEQ \$r0, \$r0

TEQ \$r0, \$r0

TEQ \$r0, \$r0

TEQ \$r0, \$r0

TEQ \$r0, \$r0

NOP

| 汇总:

执行周期总数: 51
ID段执行了9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: 22	占周期总数的百分比: 43.13726%
其中:	
load停顿: 0	占所有RAW停顿的百分比: 0%
浮点停顿: 22	占所有RAW停顿的百分比: 100%
WAW停顿: 0	占周期总数的百分比: 0%
结构停顿: 12	占周期总数的百分比: 23.52941%
控制停顿: 0	占周期总数的百分比: 0%
自陷停顿: 7	占周期总数的百分比: 13.72549%
停顿周期总数: 41	占周期总数的百分比: 80.39216%

结构停顿数量: 12, 占周期总数百分比: 23.53%

5. 性能之比: $60/51 = 1.17$

五、 补充实验 2

下面是一段 MIPS 指令序列:

ADD \$r1,\$r1,\$r0

SUB \$r2,\$r0,\$r3

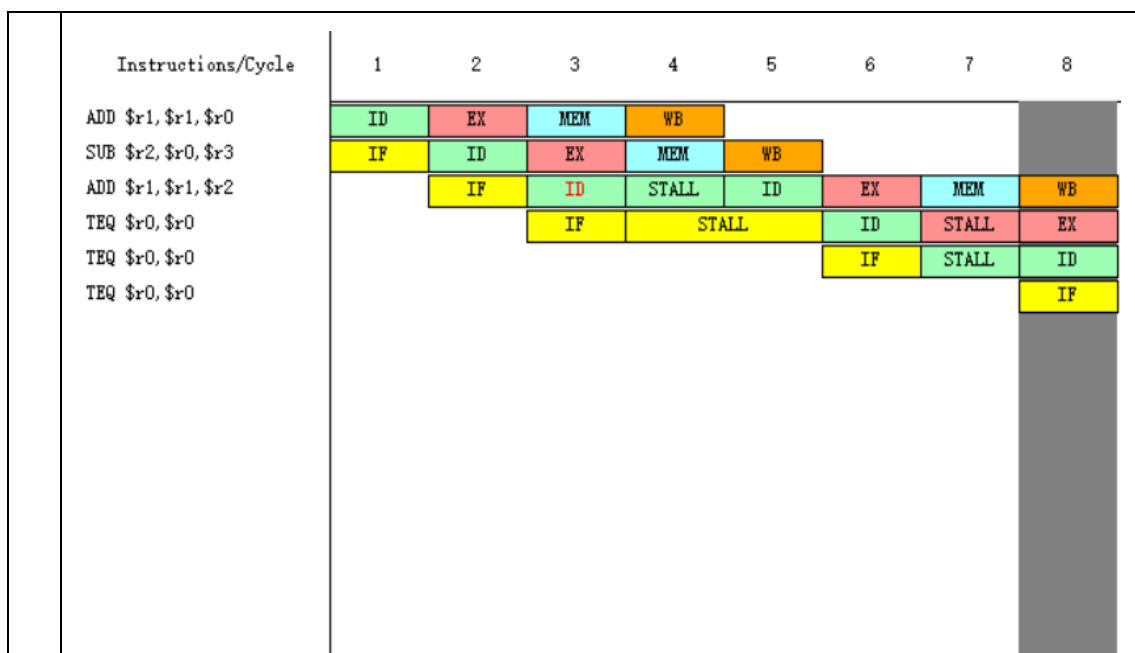
ADD \$r1,\$r1,\$r2

假定在五段流水线处理器中执行上述指令序列, 请思考:

(1) 以上指令序列, 哪些指令发生了数据相关?

两个 ADD 发生了 RAW (写后读冲突)

SUB 与第二个 ADD 发生 RAW (写后读冲突)

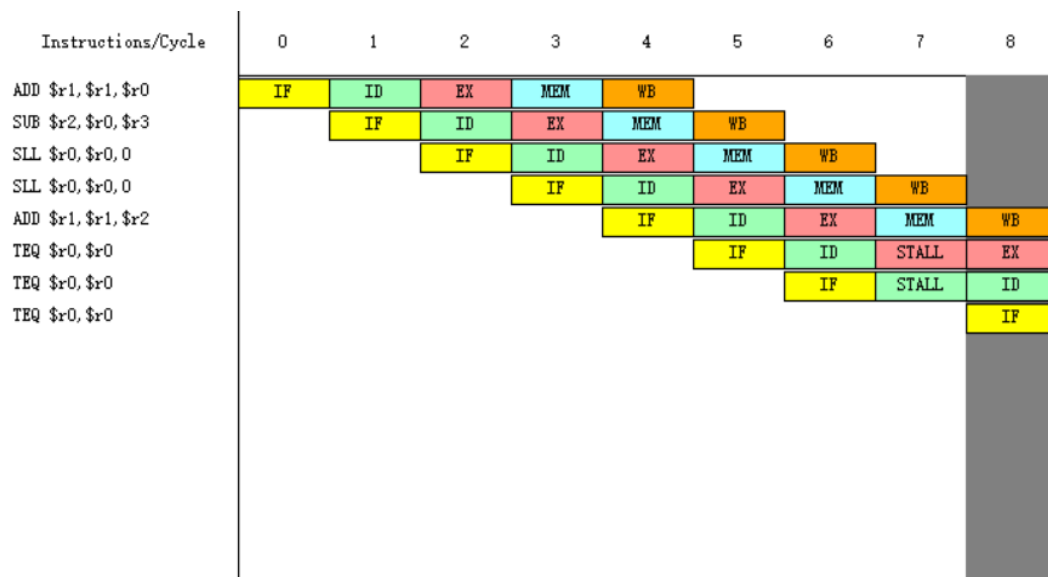


- (2) 不采用“定向”技术，需要在何处、加上几条 nop 指令才能使这段指令序列的执行避免数据冲突？
在 SUB 指令后面，加 2 个 NOP 指令

```

ADD $r1,$r1,$r0
SUB $r2,$r0,$r3
SLL $r0,$r0,0
SLL $r0,$r0,0
ADD $r1,$r1,$r2
TEQ $r0,$r0
TEQ $r0,$r0
TEQ $r0,$r0

```



- (3) 如果采用“定向”技术，是否可以完全解决数据冲突？如果不能的话，需要在何处、加上几条 nop 指令才能使这段指令序列的执行避免数据冲突？

采用定向技术能完全解决数据冲突

- (4) 将自己的分析结果在仿真器上验证，充分理解“定向”技术如何

	<p>解决流水线的数据冲突问题。 仿真验证如上。</p>
总结	<ol style="list-style-type: none">1. 在实验指导书的部分，我通过几个示例程序，观察到了正常的流水线执行情况、结构冲突导致的性能下降和定向技术对冲突情况的缓解。对流水线的各个阶段 IF、ID、EX、MEM、WB 有了更直观的认识和理解。2. 在补充实验部分，我通过调整程序语句顺序来控制对寄存器的读写顺序，进而起到减少数据冲突的效果，我意识到可以通过让对同一寄存器的操作尽量远的方式减少冲突，因此我也是按照这种方式调整程序的。3. 在最后一个补充实验中，我认识到要“预知”后续的数据冲突，即使采用一个 <code>nop</code> 指令解决冲突后，仍然可能和后续指令产生冲突，因此要仔细分析需要几个 <code>nop</code> 指令。同时，我也认识到了定向技术的好处。