

实验三：流水线中的相关

19281171 王雨潇

实验报告要求：实验报告用 Word 排版，文件内一定要有"学号姓名_学号"。实验报告应提交到课程平台，按时提交。实验报告的内容应尽量全面，避免简单化。简述实验内容，实验过程，实验结果，实验分析，心得体会等。答每道题前应将题目拷贝到实验报告上。

一、实验目的

本次实验的主要目的是加深对结构相关、数据相关和控制相关的理解。以及如何通过定向技术，指令重排、延迟槽等技术减少各相关带来的暂停。

二、实验过程

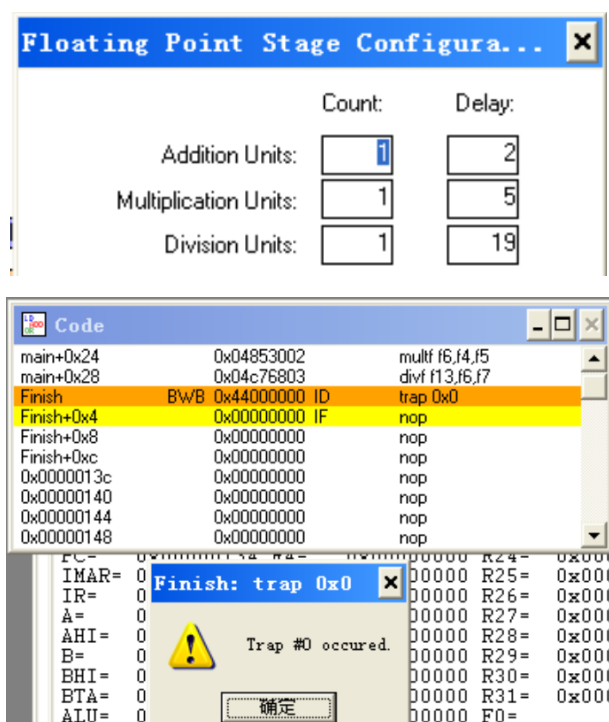
1. 实验内容一

利用 WinDLX 模拟器运行以下两段程序。

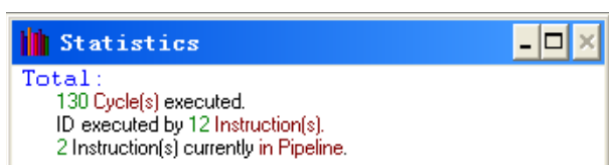
程序段 1	程序段 2
<pre>.data .global ONE ONE: .word 1 .text .global main main: If f1,ONE ;turn divf into a move cvti2f f7,f1 ;by storing in f7 1 in nop ;floating-point format divf f1,f8,f7 ;move Y=(f8) into f1 divf f2,f9,f7 ;move Z=(f9) into f2 addf f3,f1,f2 divf f10,f3,f7 ;move f3 into X=(f10) divf f4,f11,f7 ;move B=(f11) into f4 divf f5,f12,f7 ;move C=(f12) into f5 multf f6,f4,f5 divf f13,f6,f7 ;move f6 into A=(f13) Finish: trap 0</pre>	<pre>.data .global ONE ONE: .word 1 .text .global main main: If f1,ONE ;turn divf into a move cvti2f f7,f1 ;by storing in f7 1 in nop ;floating-point format divf f1,f8,f7 ;move Y=(f8) into f1 divf f2,f9,f7 ;move Z=(f9) into f2 divf f4,f11,f7 ;move B=(f11) into f4 divf f5,f12,f7 ;move C=(f12) into f5 addf f3,f1,f2 multf f6,f4,f5 divf f10,f3,f7 ;move f3 into X=(f10) divf f13,f6,f7 ;move f6 into A=(f13) Finish: trap 0</pre>

1) 程序段 1 的执行周期数是多少？分析程序中出现的暂停，都是由什么原因导致的？出现了哪些相关，导致这些相关的原因是什么？各种相关暂停的比例是多少？建议结合 clock cycle diagram 进行分析，计算周期时请指明设置的各运算单元的周期数。

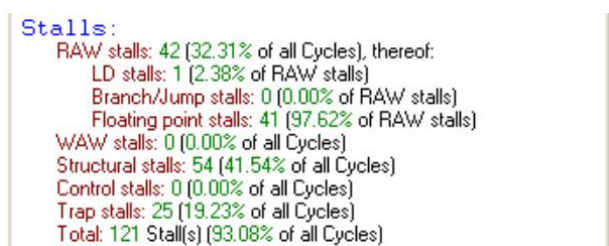
答：采用默认各运算单元周期数（加法 2，乘法 5，除法 19）、启用转发功能，运行一遍程序（设置断点为最后一条 trap 0 的 WB 段）



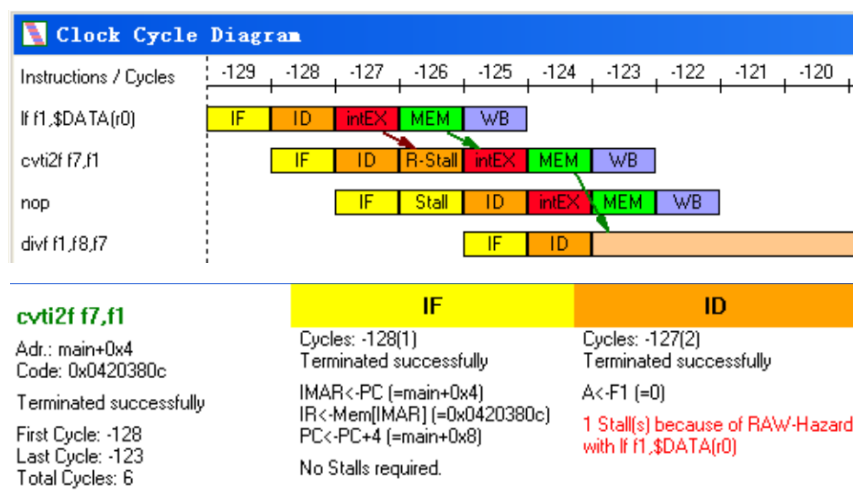
从 Statistics 窗口的统计数据可知，程序段 1 的执行时钟周期数是 130；



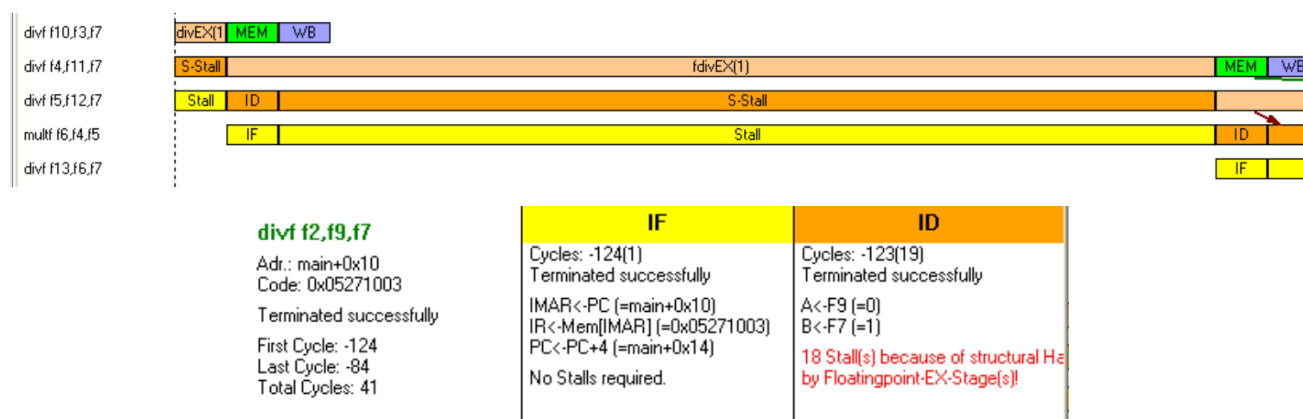
暂停的目的即避免各类相关造成错误冲突，Statistics 窗口显示，该程序中的暂停分几种原因：RAW 暂停 42 个周期（在所有暂停中占比 32.31%）、结构暂停 54 个周期（占比 41.54%）、陷阱暂停 25 个周期（占比 19.23%）



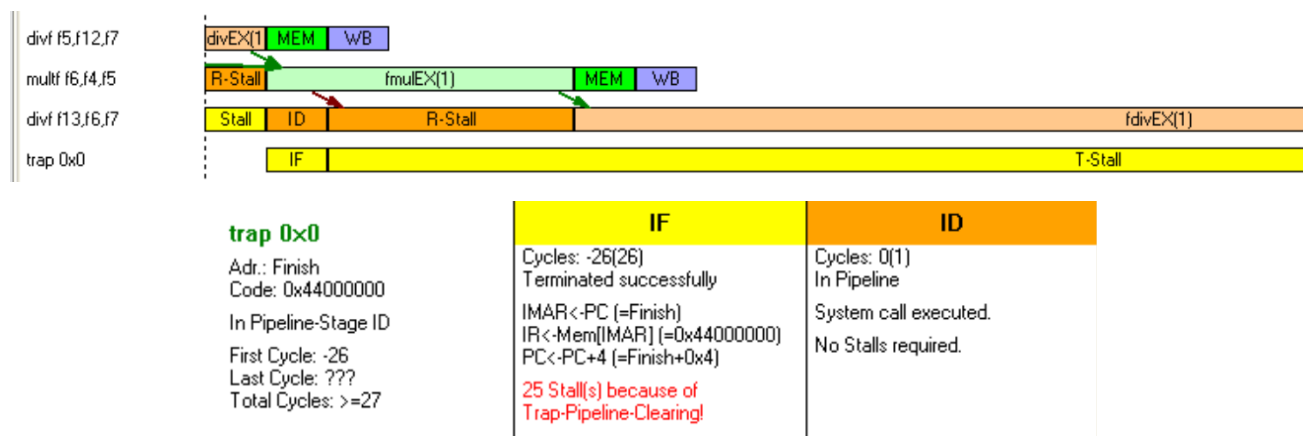
时空图显示此处 If f1,ONE 和 cvti2f f7,f1 发生了 **RAW 相关** (cvti2f 不能在 if 写入之前先读 f1) , 所以 cvti2f 指令发生了 RAW 暂停, 停留在 ID 阶段, nop 指令也由于 cvti2f 的暂停而顺延;



时空图显示此处三条连续的 divf 指令之间都发生了**结构相关** (divf 浮点数除法操作的 EX 阶段需要的硬件资源无法满足指令重叠) , 所以发生了结构暂停, 停留在 ID 阶段

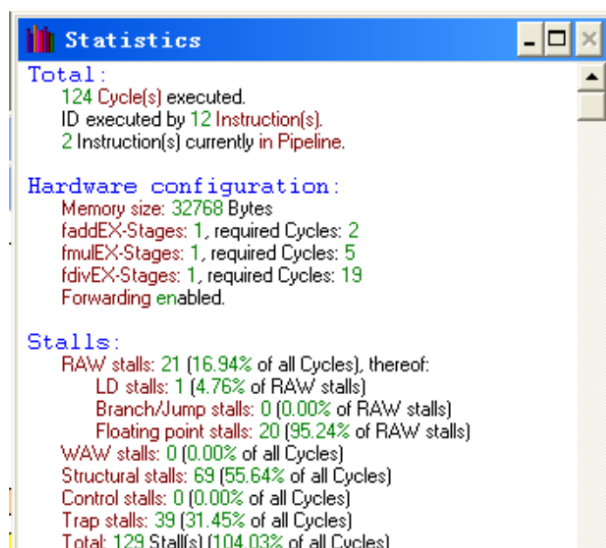


时空图显示此处 trap 0x0 作为陷阱指令产生了**陷阱相关**, 查阅帮助手册得知, 直到前面的最后一条指令 divf f13,f6,f7 结束 (即流水线全部清空) 前, trap 指令都停留在 IF 阶段



2) 对比程序段 1，程序段 2 的周期数是多少？与程序段 1 相比，程序段 2 有何不同？
 (可用如下思路：例如：程序 1 采用 XX 方法，减少了程序 2 中的 XX 相关导致的暂停周期数，或者增加了 XX 相关，XX 暂停周期数。)

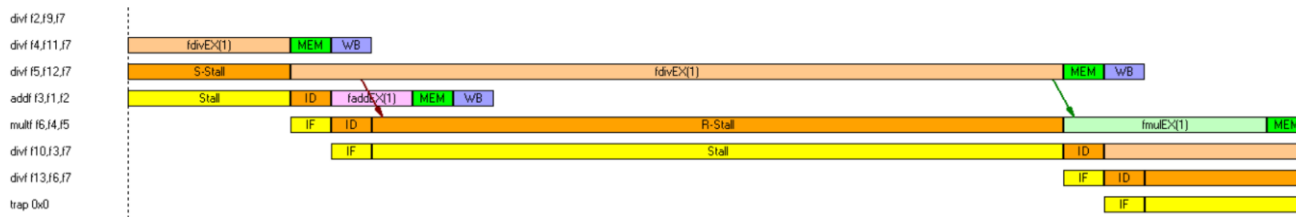
答：（模拟器设置同程序段 1）从 Statistics 窗口可知，程序段 2 的时钟周期数是 124；



对比程序 1、2 的不同，发现程序 2 在程序 1 的基础上，在代码层面改变了指令的执行顺序，把需要用到相邻的前面指令的结果的指令尽量后延（如下表所示）；

程序段 1	程序段 2
main: If f1,ONE ;turn divf into a move cvti2f f7,f1 ;by storing in f7 1 in nop ;floating-point format divf f1,f8,f7 ;move Y=(f8) into f1 divf f2,f9,f7 ;move Z=(f9) into f2 addf f3,f1,f2 divf f10,f3,f7 ;move f3 into X=(f10) divf f4,f11,f7 ;move B=(f11) into f4 divf f5,f12,f7 ;move C=(f12) into f5 multf f6,f4,f5 divf f13,f6,f7 ;move f6 into A=(f13)	main: If f1,ONE ;turn divf into a move cvti2f f7,f1 ;by storing in f7 1 in nop ;floating-point format divf f1,f8,f7 ;move Y=(f8) into f1 divf f2,f9,f7 ;move Z=(f9) into f2 divf f4,f11,f7 ;move B=(f11) into f4 divf f5,f12,f7 ;move C=(f12) into f5 addf f3,f1,f2 multf f6,f4,f5 divf f10,f3,f7 ;move f3 into X=(f10) divf f13,f6,f7 ;move f6 into A=(f13)

程序 2 修改源码的结果是，减少了程序 1 中由于数据相关造成的停顿（由 42 降低到 21 个周期），但也增加了结构相关造成的暂停（由 54 增加到 69 个周期），从时空图可以看出乘法除法的周期太长，错开排列指令虽然减少了数据相关，但也无法避免硬件资源拥堵。



3) 上述两段程序中是否存在结构相关? Why would a designer sometimes allow structural hazards?

答: 上面两段程序都存在结构相关; 设计者有些时候允许结构相关, 推测依据是量化设计准则中的“大概率事件优先原则”, 比起不发生结构相关, 结构相关的发生只是小概率事件, 完全避免结构相关可能会造成更大的流水线、硬件设计开销。

2. 实验内容二

将以下程序段修改为可利用 WinDLX 模拟器运行的程序。假设 R3 的初始值为 R2+40

程序段 3			
Loop:	LW	R1,0(R2);	load R1 from address 0+R2
	ADDI	R1,R1,#1;	R1=R1+1
	SW	0(R2),R1;	store R1 at address 0+R2
	ADDI	R2,R2,#4;	R2=R2+4
	SUB	R4,R3,R2;	R4=R3-R2
	BNEZ	R4,Loop;	Branch to loop if R4!=0

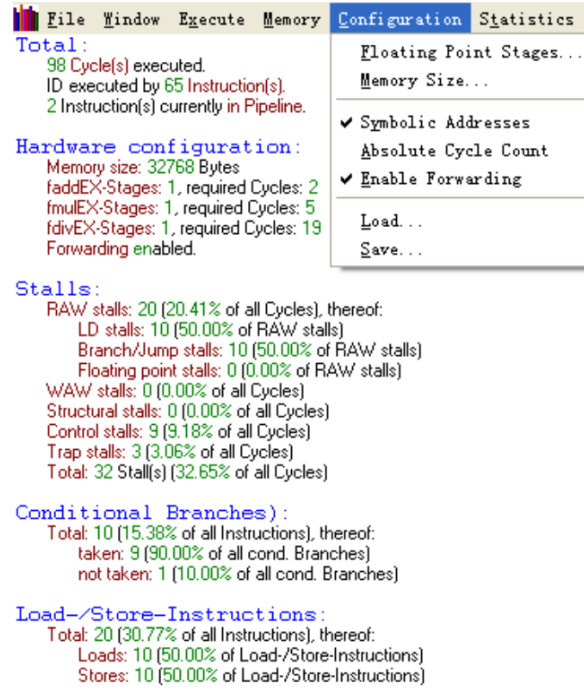
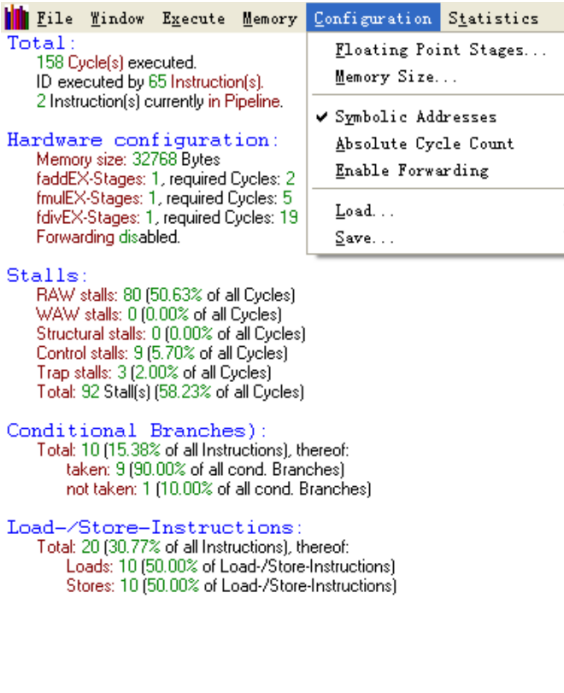
答: 该程序段主要是缺少数据段 (.data) 和变量初始化, 参考程序段 1, 代码如下所示

```
.data
.global Table
Table:
    .space 200

.text
.global main
main:
    ADDI    R1,R0,0;        R1 = 0
    ADDI    R2,R0,0;        R2 = 0
    ADDI    R3,R0,40;       R3 = 40
    ADDI    R4,R0,0;        R4 = 0
Loop:
    LW      R1,0(R2);        load R1 from address 0+R2
    ADDI    R1,R1,#1;        R1=R1+1
    SW      0(R2),R1;        store R1 at address 0+R2
    ADDI    R2,R2,#4;        R2=R2+4
    SUB     R4,R3,R2;        R4=R3-R2
    BNEZ    R4,Loop;        Branch to loop if R4!=0
Finish:
    trap 0
```

1) 观察分析使用定向技术与不使用定向技术的情况下，该程序段的执行周期数。画出在使用定向技术的情况下，一个循环的流水线时空图，并根据一个循环的时空图推算出整个程序的周期数。对比是否与模拟器中的 clock cycle diagram 以及程序中统计出的周期数一致。

答：该程序段的执行周期数在使用定向时为 98，不使用定向时为 158，定向技术大幅减少了数据相关造成的暂停，可以看出定向技术对流水线性能的加速比为 1.6122；

使用定向	不使用定向
	

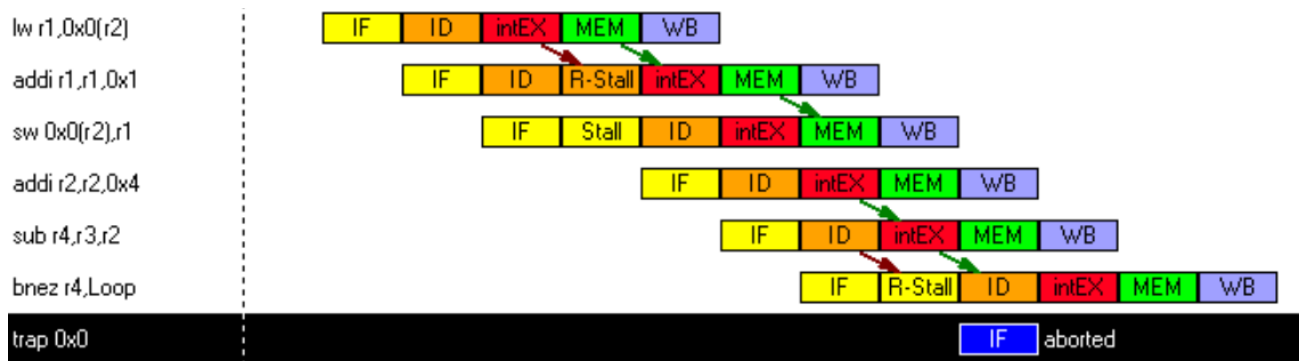
使用定向技术的情况下，可以画出一个循环的流水线时空图，如下所示：

	时钟周期											
指令	1	2	3	4	5	6	7	8	9	10	11	12
LW R1,0(R2)	IF	ID	EX	MEM	WB							
ADDI R1,R1,#1		IF	ID	STALL	EX	MEM	WB					
SW 0(R2),R1			IF	STALL	ID	EX	MEM	WB				
ADDI R2,R2,#4				等R1	IF	ID	EX	MEM	WB			
SUB R4,R3,R2						IF	ID	EX	MEM	WB		
BNEZ R4,Loop							IF	STALL	ID	EX	MEM	WB
								等R4				
										下一次循环		

因为 R3 的初始值为 R2+40，所以整个程序 Loop 段执行 10 次，重叠长度为 3，循环的部分应该是 $12+9\times 9=93$ 个周期，模拟器中使用定向运行共有 98 个执行周期，差距部分应该是初始赋值语句；

2) 一次循环中的暂停周期数是多少，引起这些暂停的原因是什么？一次循环中有哪些相关，引起这些相关的原因是什么？

答：以下回答分析的是使用定向技术时的一次循环



一次循环中共有 3 次暂停，2 次相关：

lw r1,0x0(r2)和 addi r1,r1,0x1 指令之间的暂停是为了 r1 避免 RAW 数据相关；

addi r1,r1,0x1 和 sw 0x0(r2),r1 之间的暂停是跟着上一条指令延迟的暂停；

sub r4,r3,r2 和 bnez r4,Loop 之间的暂停是为了 r4 避免 RAW 数据相关；

3) 为了减少控制相关带来的暂停，我们学习了多种方法，该模拟器中使用的是哪种方法？该方法的思路是什么？在什么情况下可得到改进？

答：在流水线中可以看出，即使前 9 次循环的 bnez 转移不满足转移条件，trap 指令也执行，到 IF 段后才终止，所以 winDLX 模拟器默认预测分支转移成功，按成功执行后续指令；预测分支转移成功的思路是始终假设分支转移成功，如果转移不成功，再终止执行；

```
Conditional Branches):
Total: 10 (15.38% of all Instructions), thereof:
taken: 9 (90.00% of all cond. Branches)
not taken: 1 (10.00% of all cond. Branches)

Load-/Store-Instructions:
Total: 20 (30.77% of all Instructions), thereof:
Loads: 10 (50.00% of Load-/Store-Instructions)
Stores: 10 (50.00% of Load-/Store-Instructions)
```

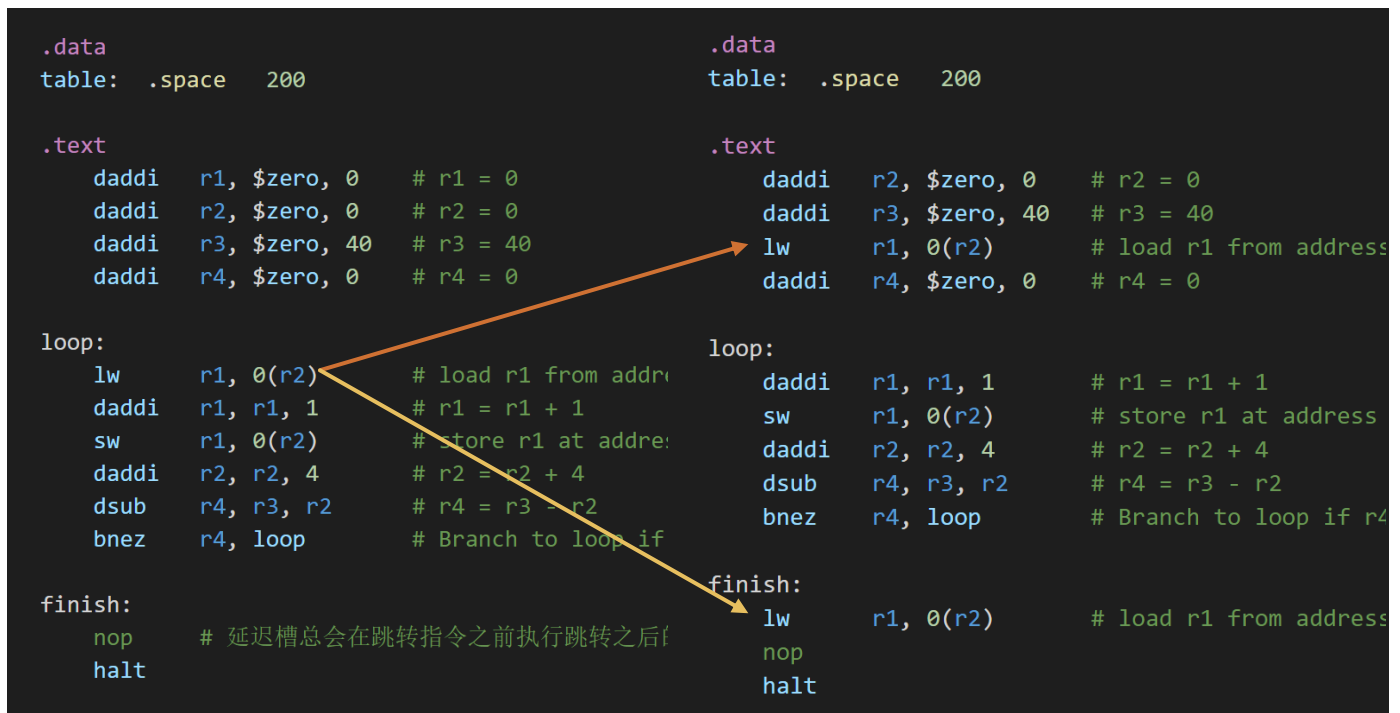
在本程序中，预测分支转移成功的正确率只有 10%，可以认为预测分支转移成功更适合用于改进转移成功比转移失败发生频率更高的程序。

3. 实验内容三

将上述程序段 3 修改为可利用 WinMIPS64 模拟器运行的程序。在**使用定向技术**的情况下，对指令进行重排，采用 delayed branch 来减少控制相关带来的损失。进行分析：延迟槽中调度的是哪条指令？为什么选择该指令？调度后的程序与调度前的程序周期数相差多少？改进是如何体现的？

答：采用默认各运算单元周期数（FP4，乘法 7，除法 24），开启定向和延迟槽技术，查阅教材和资料^[1]发现，delayed branch 技术会在跳转指令之前无条件执行跳转指令后的指令；观察现象进一步确认，WinMIPS64 的延迟槽调度是**无条件执行** bnez r4,loop 后的 1 条指令，无论 bnez 执行时寄存器 r4 是否为 0；

修改为 MIPS 汇编的程序 3（左图）和重排指令后的程序 3（右图）如下所示



修改前的程序延迟槽中调度的只是 nop 指令（什么都不做），原因是开启延迟槽功能后，如果不加 nop，程序就会在第一次循环时跳过 bnez 执行延迟槽中的 halt 直接退出；

对程序结构进行分析，单次循环内的依赖关系是取 r1 用到 r2，加 r1 用到 r1，存 r1 用到 r2，加 r2 用到 r2，减法用 r3-r2 得 r4，条件转移用 r4

[1] <https://devblogs.microsoft.com/oldnewthing/20180411-00/?p=98485>


```
10  loop:
11      lw      r1, 0(r2)      # load r1 from address 0 + r2
12      daddi   r1, r1, 1      # r1 = r1 + 1
13      sw      r1, 0(r2)      # store r1 at address 0 + r2
14      daddi   r2, r2, 4      # r2 = r2 + 4
15      dsub    r4, r3, r2      # r4 = r3 - r2
16      bnez    r4, loop       # Branch to loop if r4 != 0
```

根据“从目标处调度”的处理思想，既然后续操作都从 r1 开始，就把 lw r1,0(r2)这条取数指令放进延迟槽中调度，运行结果对比如图所示，使用延迟槽调度后的程序与调度前相比节省了 10 个时钟周期，减少了 1 次 RAW 相关造成的暂停。

不使用 delayed branch	使用 delayed branch
<div>Execution</div> <div>99 Cycles</div> <div>75 Instructions</div> <div>1.320 Cycles Per Instruction (CPI)</div> <div>Stalls</div> <div>20 RAW Stalls</div> <div>0 WAW Stalls</div> <div>0 WAR Stalls</div> <div>0 Structural Stalls</div> <div>0 Branch Taken Stalls</div> <div>0 Branch Misprediction Stalls</div>	<div>Execution</div> <div>89 Cycles</div> <div>66 Instructions</div> <div>1.348 Cycles Per Instruction (CPI)</div> <div>Stalls</div> <div>19 RAW Stalls</div> <div>0 WAW Stalls</div> <div>0 WAR Stalls</div> <div>0 Structural Stalls</div> <div>0 Branch Taken Stalls</div> <div>0 Branch Misprediction Stalls</div>

指令重排、使用延迟槽后的加速比= $\frac{99}{89}=1.112$ ，11.2%的加速并不是很显著，推测是重排后的指令依然数据相关依赖比较紧密；

三、 心得体会

通过本次实验的三个内容，我通过 WinDLX 和 WinMIPS64 模拟器加深了对结构相关、数据相关和控制相关的理解。实验内容一主要通过模拟编译器重排指令序列的过程演示如何减少相关，实验内容二介绍了通过定向技术减少暂停，实验内容三则是综合运用重排指令序列、延迟槽减少相关的实际训练。

手动排列指令、定向、延迟槽这些办法都能减少一定比例的暂停，减少暂停就意味着流水线的性能提高，这些方法对不同类型的暂停各有专攻，需要配合使用以优化性能。