Lab 5 实验报告

实验目的

1. 理解流水线CPU指令执行过程。
2. 理解流水线冒险处理的概念。
3. 理解不同流水线硬件结构对冒险处理方法的区别。

实验平台

WebRISCV: RISC_V架构 RV32/64IM 五级流水线CPU模型在线仿真平台（RV32IM模式）。

网页链接： https://webriscv.dii.unisi.it/index.php

任务一：测试代码1的仿真

测试代码段

```
addi x6,x6,2
loop: beq x6,x0,fi
addi x6,x6,-1
addi x5,x5,3
j loop
fi: add x4,x4,x5
```

四种模式的仿真结果

EXECUTION TABLE

FULL LOOPS

Instruction	CPU Cycles 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
addi t1, t1, 2	F	D	X	M	W														
beq t1, x0, 32		F	-	D	X	M	W												
addi t1, t1, -1				F	D	X	M	W											
addi t0, t0, 3					F	D	X	M	W										
jal x0, -24						F	D	X	M	W									
add tp, tp, t0							F												
beq t1, x0, 32								F	D	X	M	W							
addi t1, t1, -1									F	D	X	M	W						
addi t0, t0, 3										F	D	X	M	W					
jal x0, -24											F	D	X	M	W				
add tp, tp, t0												F							
beq t1, x0, 32													F	D	X	M	W		
addi t1, t1, -1														F					
add tp, tp, t0															F	D	X	M	W

图1： Mode 1 - With forward with flush

EXECUTION TABLE

FULL LOOPS

Instruction	CPU Cycles																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
addi t1, t1, 2	F	D	X	M	W															
beq t1, x0, 32		F	-	-	D	X	M	W												
addi t1, t1, -1					F	D	X	M	W											
addi t0, t0, 3						F	D	X	M	W										
jal x0, -24							F	D	X	M	W									
add tp, tp, t0								F												
beq t1, x0, 32									F	D	X	M	W							
addi t1, t1, -1										F	D	X	M	W						
addi t0, t0, 3											F	D	X	M	W					
jal x0, -24												F	D	X	M	W				
add tp, tp, t0													F							
beq t1, x0, 32														F	D	X	M	W		
addi t1, t1, -1															F					
add tp, tp, t0																F	D	X	M	W

图2： Mode 2 - No forward with flush

EXECUTION TABLE

FULL LOOPS

Instruction	CPU Cycles																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
addi t1, t1, 2	F	D	X	M	W														
beq t1, x0, 32		F	-	D	X	M	W												
addi t1, t1, -1				F	D	X	M	W											
addi t0, t0, 3					F	D	X	M	W										
jal x0, -24						F	D	X	M	W									
add tp, tp, t0							F	D	X	M	W								
beq t1, x0, 32								F	D	X	M	W							
addi t1, t1, -1									F	D	X	M	W						
addi t0, t0, 3										F	D	X	M	W					
jal x0, -24											F	D	X	M	W				
add tp, tp, t0												F	D	X	M	W			
beq t1, x0, 32													F	D	X	M	W		
addi t1, t1, -1														F	D	X	M	W	
add tp, tp, t0															F	D	X	M	W

图3： Mode 3 - With forward no flush

EXECUTION TABLE

FULL LOOPS

Instruction	CPU Cycles																					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
addi t1, t1, 2	F	D	X	M	W																	
beq t1, x0, 32		F	-	-	D	X	M	W														
addi t1, t1, -1					F	D	X	M	W													
addi t0, t0, 3						F	D	X	M	W												
jal x0, -24							F	D	X	M	W											
add tp, tp, t0								F	-	D	X	M	W									
beq t1, x0, 32										F	D	X	M	W								
addi t1, t1, -1											F	D	X	M	W							
addi t0, t0, 3												F	D	X	M	W						
jal x0, -24													F	D	X	M	W					
add tp, tp, t0														F	-	D	X	M	W			
beq t1, x0, 32																F	D	X	M	W		
addi t1, t1, -1																	F	D	X	M	W	
add tp, tp, t0																		F	D	X	M	W

图4： Mode 4 - No forward no flush

填表分析仿真结果

表 1

代码段 1	模式 1： with forward with flush	模式 2： no forward with flush	模式 3： with forward no flush	模式 4： no forward no flush
执行周期数	**19**	**20**	**19**	**22**
被执行 forward 的指令，执行 forward 次数和原因	addi x6,x6,2 beq x6,x0,fi 执行 **1** 次 不需要等待 addi 指令将 x6 写回寄存器就直接从 ALU 将结果前递给 beq 使用。	/	addi x6,x6,2 beq x6,x0,fi 执行 **1** 次 不需要等待 addi 指令将 x6 写回寄存器就直接从 ALU 将结果前递给 beq 使用。	/
被执行 flush 操作的指令和原因	j loop fi: add x4,x4,x5 执行 **3** 次 默认分支不跳转，当发现分支需要跳转时清除已有指令。	j loop fi: add x4,x4,x5 执行 **3** 次 默认分支不跳转，当发现分支需要跳转时清除已有指令。	/	/

表 2

代码段 1	分析：单周期 CPU 架构下，执行需（ **70** ）个时钟周期， 执行后，x6=（ **0** ），x5=（ **6** ），x4=（ **6** ）			
实际执行仿真情况	模式 1： with forward with flush	模式 2： no forward with flush	模式 3： with forward no flush	模式 4： no forward no flush
Reg X6=	**0**	**0**	**-1**	**-1**
Reg X5=	**6**	**6**	**6**	**6**
Reg X4=	**6**	**6**	**15**	**15**
执行结果正确与否	**正确**	**正确**	**不正确**	**不正确**
若不正确 如何修改	\	\	由于没有 flush，导致默认的分支不跳转预判每次都被完整执行。即每轮循环都会错误执行一遍 fi: add x4,x4,x5 指令（导致 x4 错误）；且 addi x6,x6,-1 指令也被多执行了一次（导致 x6 错误）。 一种解决方案是增加冗余的指令。如在 beq 语句后加 addi x6, x6,0；在 j 语句后加 addi x4, x4, 0。（时钟周期数会增加）	

修改后代码段：

```
addi x6,x6,2
loop: beq x6,x0,fi
addi x6, x6,0
addi x6,x6,-1
addi x5,x5,3
j loop
addi x4, x4, 0
fi: add x4,x4,x5
```

修改后结果正确：

4	tp	6
5	t0	6
6	t1	0

图5：结果正确的寄存器x4, x5, x6的值

选做：另找代码验证冒险处理

测试代码段：

```
addi x1, x0, 1
addi x4, x0, 1024
sw x1, 0(x4)
lw x2, 0(x4)
and x4, x1, x2
```

Instruction	1	2	3	4	5	6	7	8	9	10
addi ra, x0, 1	F	D	X	M	W					
addi tp, x0, 1024		F	D	X	M	W				
sw ra, 0(tp)			F	D	X	M	W			
lw sp, 0(tp)				F	D	X	M	W		
and tp, ra, sp					F	-	D	X	M	W

图6：执行情况

分析：

在取数指令后紧接着执行and操作，理论上需要停顿三个时钟周期，待取数写回寄存器x2，但由于forwarding开启，取数结果从memory前递到and指令的ALU，故只需停顿一个周期，大大提升了执行效率。

任务二：测试代码2的仿真

测试代码段

```
lui x10, 0
ori x4, x10, 1024
addi x25, x0, 1
addi x26, x0, 2
addi x27, x0, 3
addi x28, x0, 4
sw x25, 0(x4)
sw x26, 4(x4)
sw x27, 8(x4)
sw x28, 12(x4)
addi x5, x0, 4
call:
jal sum
sw x12, 0(x4)
lw x19, 0(x4)
sub x18, x19, x12
addi x5, x0, 3
loop2:
addi x5, x5, -1
ori x18, x5, -1
xori x18, x18, 1365
addi x19, x0, -1
andi x20, x19, -1
or x16, x20, x19
xor x18, x20, x19
and x17, x20, x16
beq x5, x0, shift
j loop2
shift:
addi x5, x0, -1
slli x18, x5, 15
slli x18, x18, 16
srai x18, x18, 16
srli x18, x18, 15
fi:
j fi
sum:
add x18, x0, x0
loop:
lw x19, 0(x4)
addi x4, x4, 4
add x18, x18, x19
addi x5, x5, -1
bne x5, x0, loop
slli x12, x18, 0
jr ra
```

填表分析仿真结果

表 3

PC 值	指令	冒险的种类	执行的操作
04	ori x4, x10, 0	数据冒险	从 00 指令前递 x10 的值
30	sw x12, 0(x4)	控制冒险	flush 清除指令
8c	add x18, x18, x19	数据冒险	从 84 指令前递 x19 的值
94	bne x5, x0, loop	数据冒险	从 90 指令前递 x5 的值，并 stall 一个周期
98	slli x12, x18, 0	控制冒险	flush 清除指令
38	sub x18, x19, x12	数据冒险	从 34 指令前递 x19 的值，并 stall 一个周期
40	loop2:addi x5, x5, -1	数据冒险	从 3c 指令前递 x5 的值
44	ori x18, x5, -1	数据冒险	从 40 指令前递 x5 的值
48	xori x18, x18, 1365	数据冒险	从 44 指令前递 x18 的值
50	andi x20, x19, -1	数据冒险	从 4c 指令前递 x19 的值
54	or x16, x20, x19	数据冒险	从 50 指令前递 x20 的值
5c	and x17, x20, x16	数据冒险	从 54 指令前递 x16 的值
68	shift:addi x5, x0, -1	控制冒险	flush 清除指令
6c	slli x18, x5, 15	数据冒险	从 68 指令前递 x5 的值
70	slli x18, x18, 16	数据冒险	从 6c 指令前递 x18 的值
74	srai x18, x18, 16	数据冒险	从 70 指令前递 x18 的值
78	srli x18, x18, 15	数据冒险	从 74 指令前递 x18 的值
80	sum:add x18, x0, x0	控制冒险	flush 清除指令

选做：不同模式执行周期对比分析

模式1：With forward with flush	**模式2：No forward with flush**	**模式3：With forward no flush**	**模式4：No forward no flush**	**单周期**
91 cycles	139 cycles	96 cycles*	142 cycles*	425 cycles

*注意到 no flush 模式下错误的分支预判不会被清除，在指令 `j loop2` 正确执行前，其下一条指令 `addi x5, x0, -1` 也被错误执行了，导致上述循环无法正确跳出，故将 68 和 6c 处的 `x5` 寄存器修改为 `x6`，并把 `beq x5, x0, shift` 指令提前以避免冲突导致的错误。

修改后的代码段：

```
...
loop2:
beq x5, x0, shift    //条件判断提前到此处
addi x5, x5, -1
ori x18, x5, -1
xori x18, x18, 1365
addi x19, x0, -1
andi x20, x19, -1
or x16, x20, x19
xor x18, x20, x19
and x17, x20, x16
j loop2
shift:
```

```
addi x6, x0, -1     //x5改为x6
slli x18, x6, 15
slli x18, x18, 16
srai x18, x18, 16
srli x18, x18, 15
...
```

分析结果：

与任务一的执行结果类似，且符合预期。总体而言，前递和flush都会提高这段代码的执行效率，而是否支持前递是主要影响因素。由之前的分析可知，代码段2的数据冒险较多，因此通过forwarding会带来很大的性能提升(35%左右)，支持flush也会带来5%左右的提升。与单周期相比，无论采取哪种模式，流水线执行都可以大幅提高效率（**3~4倍**）。

拓展思考

流水线分级

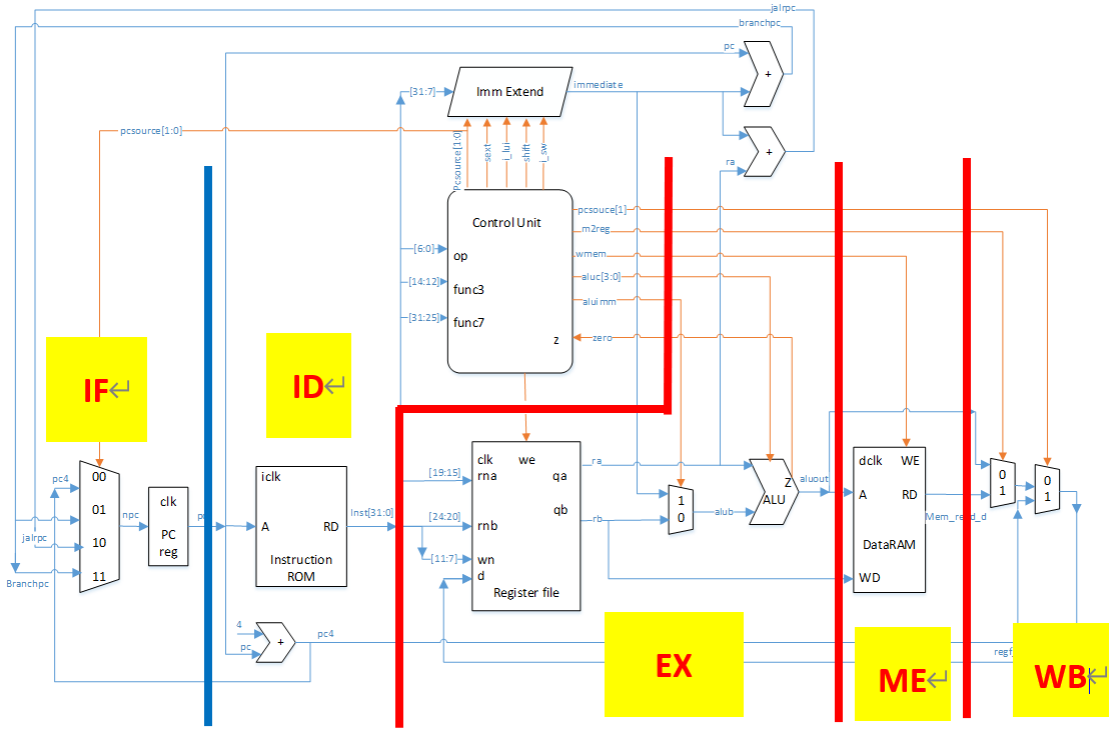

图7：流水线分级划分图

分析与问题回答

其中左侧四选一MUX应该算作哪一级？右上方两个加法器可以在ID级里面实现吗？最右侧的二选一MUX是否一定要放在最后一级里面实现？如果放在EXE级里是否可以？forward操作是否可以在ID级实现？stall操作应该控制什么模块？flush操作应该控制什么模块？图中蓝色粗线代表了第一级PC寄存器位置。可类似画出后面的几级寄存器应该加在哪里。

1. 左侧四选一 MUX 应算作 IF级。
2. 右上方两个加法器中：计算branchpc的可以放到ID级，而计算jalpc的不能，因为后者的输入需要在EX级由ALU计算结果提供。
3. 最右侧的二选一MUX不能在EX实现，因为其输入需要等待ME级的存储器输出。
4. forward操作不能在ID级实现，因为ID级进行冒险检测后才能决定是否前递。

5. stall操作将ID指令清除，即RegWrite和MemWrite信号置零即可。
6. flush操作要将IF、ID、EX阶段的指令全部清除，将所有控制信号置零。