浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：官泽隆

学　　院：计算机科学与技术学院

系：

专　　业：计算机科学与技术

学　　号：3180103008

指导教师：翁恺

2020 年 12 月 7 日

浙江大学实验报告

课程名称：计算机体系结构 实验类型：综合

实验项目 **PCPU resolving control hazard**

学生姓名：官泽隆 专业：计算机科学与技术 学号：3180103008

同组学生姓名：/ 指导老师：翁恺

实验地点：曹光彪二期-301 实验日期：2020 年 12 月 7 日

一、 实验目的和要求

Objective:

1. Learn methods on how to resolve control hazards.

Task:

1. implement Predict-not-taken scheme on branch handling
2. reduce the stall caused by branch to 1 clock cycle
3. In the VGABased-Debugger project, replace module *mips* with your own PCPU
4. Validate the design of your own PCPU on SWORD Board

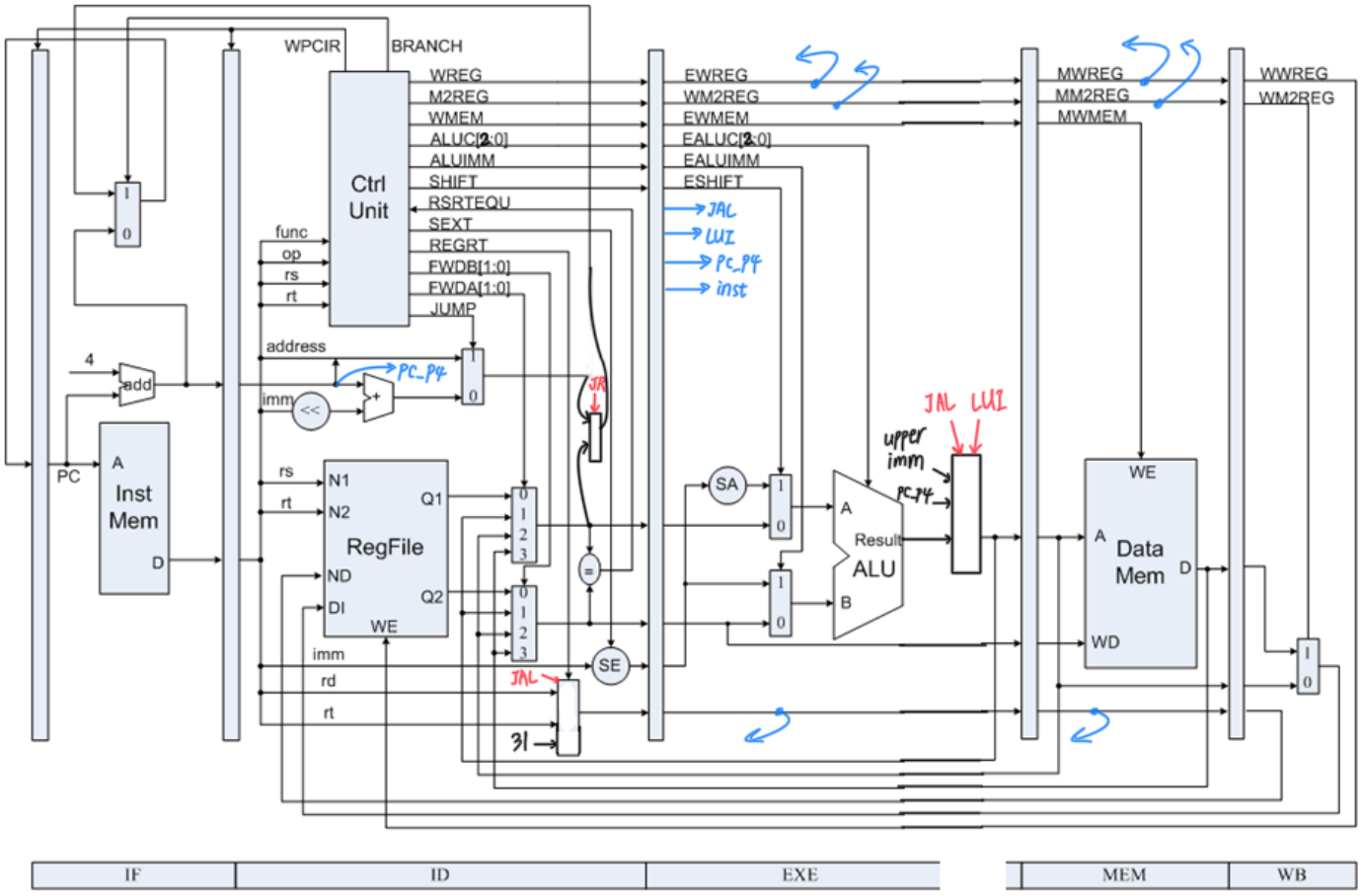
二、实验内容和原理

原理：

前情提要：

注：手绘线，黑色线表示数据，红色线表示控制，蓝色线表示向后流水的或是向前转发的信号

略去 ID.stall 的线



~~跳转为 delayed branch（长度为 1 个指令的延迟槽）模式，在 ID 解析完毕。~~

相比于 forwarding 实验的工程，数据通路无需改变，只有一处控制信号的改动。

我们在 ID 级检测到 branch taken，IF 级读入错误的下一条指令时，输出 reset 到 IF/ID 间的暂存器。由于暂存器使用的是标准的 *registers w/ synchronous RST and CE*， reset 是最高优先级，因此下一个时钟周期上升沿，ID 级的指令复位成 *NOP*，IF 级的错误指令被毁无法传递到下一级。

这样就从 delayed branch 改成了 predict not taken.

这里可以体会到两种方式的区别：delayed branch 从指令集层面，把经典五级流水线引起的 branch taken 时 IF 级指令错误，认定成 feature；而 predict not taken 在硬件层面纠正，使得 branch 的语义直观。

三、实验过程和数据记录

1. 实现层面

对应 流水线级间（PCPU.v），1 LoC 的接线变化：（加粗部分）

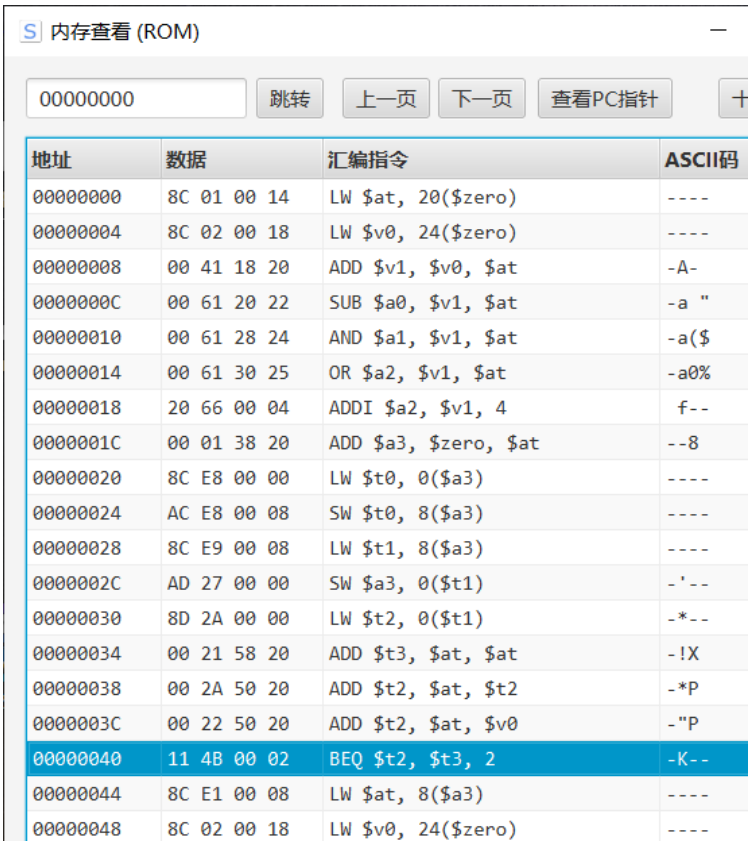
```
IfIdRegisters.rst( rst | (id_BRANCH & ~id_stall) )
```

要考虑到，ID 级解析时可能因 data hazard 导致 stall，此时的 control hazard 结果不可信。

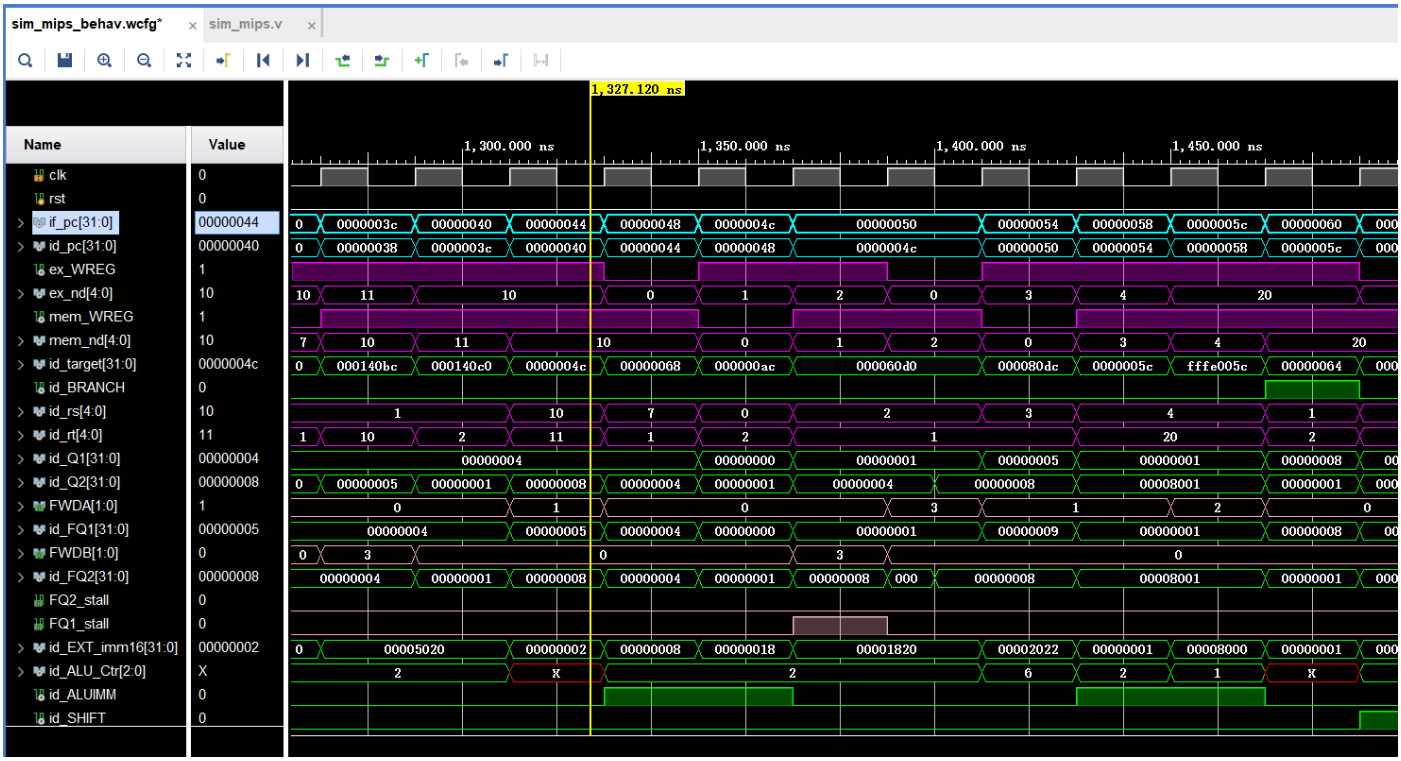
2. 行为仿真——助教给的测试程序

对应 sim_mips.v

```
1   begin:
2   Lw $1, 20($zero)        //R1=4
3   Lw $2, 24($zero)        //R2=1
4   Add $3,$2,$1        //R3=5  //2LW-ALU:forwarding:1 stall
5   Sub $4,$3,$1        //R4=1  //2ALU-ALU
6   And $5,$3,$1        //R5=4  //无冲突
7   Or $6,$3,$1     //R6=5  //无冲突
8   addi $6,$3,4        //$6=9  //无冲突
9   Add $7, $zero, $1       //R7=4  //无冲突
10  Lw $8,0($7)         //R8=8  //2ALU-LW
11  Sw $8,8($7)         //  //2LW-SW: forwarding可以解决
12  Lw $9, 8($7)        //R9=8
13  Sw $7, 0($9)        //  //2LW-SW: forwarding:1 stall
14  Lw $10,0($9)        //R10=4
15  Add $11, $1, $1     //R11=8
16  Add $10, $1, $10    //R10=8 //1LW-ALU:forwarding可以解决
17  Add $10,$1,$2       //R10=5
18  Beq $10, $11 ,lable1    // not taken//2+1ALU-BEQ//  branch x
19  Lw $1, 8($7)        //R1=8
20  Lw $2, 24($zero)        //R2=1
```



地址	数据	汇编指令	ASCII码
00000000	8C 01 00 14	LW $at, 20($zero)	----
00000004	8C 02 00 18	LW $v0, 24($zero)	----
00000008	00 41 18 20	ADD $v1, $v0, $at	-A-
0000000C	00 61 20 22	SUB $a0, $v1, $at	-a "
00000010	00 61 28 24	AND $a1, $v1, $at	-a($
00000014	00 61 30 25	OR $a2, $v1, $at	-a0%
00000018	20 66 00 04	ADDI $a2, $v1, 4	f--
0000001C	00 01 38 20	ADD $a3, $zero, $at	--8
00000020	8C E8 00 00	LW $t0, 0($a3)	----
00000024	AC E8 00 08	SW $t0, 8($a3)	----
00000028	8C E9 00 08	LW $t1, 8($a3)	----
0000002C	AD 27 00 00	SW $a3, 0($t1)	-'--
00000030	8D 2A 00 00	LW $t2, 0($t1)	-*--
00000034	00 21 58 20	ADD $t3, $at, $at	-!X
00000038	00 2A 50 20	ADD $t2, $at, $t2	-*P
0000003C	00 22 50 20	ADD $t2, $at, $v0	-"P
00000040	11 4B 00 02	BEQ $t2, $t3, 2	-K--
00000044	8C E1 00 08	LW $at, 8($a3)	----
00000048	8C 02 00 18	LW $v0, 24($zero)	----

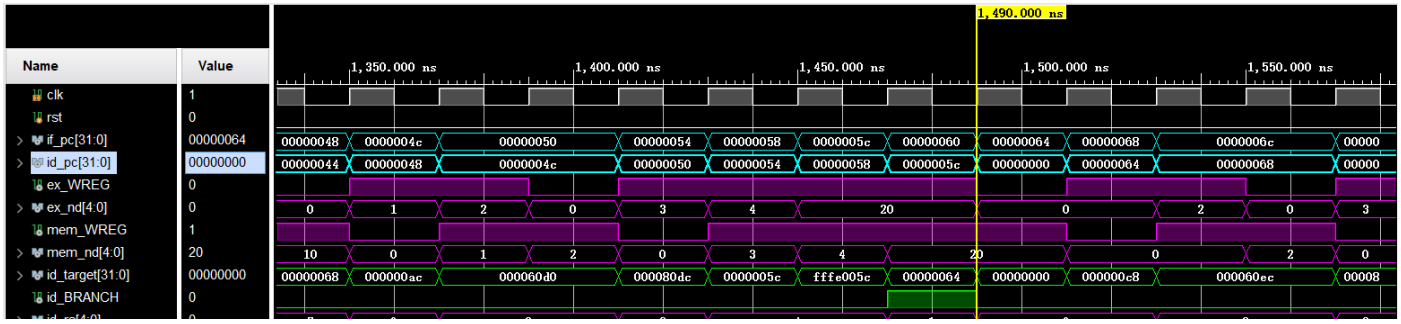


注意第 17 条指令（0x40），是一条 beq，助教注释 **not taken**；这里我的 PCPU 继续执行下一条指令，并毫无停顿，宛如顺序执行，验证 **predict not taken** 的更改

----------second case----------

```
21    lable1:
22    Add $3,$2,$1          //R3=9  //2LW-ALU
23    Sub $4,$3,$1          //R4=1   //2ALU-R-R
24    Addi $20, $4, 1       //R20=2 //2ALU-addi
25    Ori $20, $4, 0x8000     //R20=0x8001    //无冲突 无符号扩展
26    Bne $1, $2, lable2          //taken x
27    Lw $1,20($zero)     //不执行
28    lable2:
```

```
00000050    00 61 20 22    SUB $a0, $v1, $at          -a "
00000054    20 94 00 01    ADDI $s4, $a0, 1            ---
00000058    34 94 80 00    ORI $s4, $a0, -32768       4---
0000005C    14 22 00 01    BNE $at, $v0, 1            -"--
00000060    8C 01 00 14    LW $at, 20($zero)          ----
```



在 **branch taken** 时（0x5c），id_pc 下一拍被复位，再下一拍装入正确的跳转目标地址（0x64）。验证 **predict not taken** 对程序执行正确性的保证