浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：展翼飞

学　　院：计算机科学与技术学院

　　系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3190102196

指导教师：姜晓红

2024 年　9　月　23　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

实验项目名称：Lab1：Pipelined CPU supporting RISC-V RV32I Instructions

学生姓名：展翼飞 专业：计算机科学与技术 学号：3190102196

同组学生姓名：无 指导老师：姜晓红

实验地点：曹光彪西楼 301 实验日期：2024 年 9 月 23 日

一、 实验目的和要求

二、 实验内容和原理

Tips: 请解释 predict not taken 的实现思路和 3 个 forward 的实现思路，并简要解释顶层 RV32core 的连线

2.1 Predict not taken 实现思路

Predict not taken 的核心思想是对于每一条指令 PC，都默认预测它的下一条指令为 PC+4 而不发生跳转。若预测正确，则整个流水线继续顺序执行；若预测错误，即此条指令（B 类型或 Jal，Jalr）实际需要跳转，则需要对后续错误指令进行 flush，并将实际的指令地址重新取指到 ID 阶段执行。

在实际的 datapath 的设计过程中，我们将与跳转指令相关的运算从 EXE 阶段提前到 ID 阶段，减少了 predict 错误需要 flush 的时钟周期，提高了效率。即 B 类型指令进行的判断由 ID 阶段的比较器完成，判断结果输入到加法器，由 PC(或寄存器 rs1 取值，取决于跳转指令类型是否是 Jalr)和立即数计算出下条指令实际地址。如果跳转发生，则由 control unit 发出 flush 信号到 IF-ID 寄存器清除错误指令，同时改变 IF 阶段指令取值二路选择器读取正确指令。

If 阶段指令地址二路选择器：

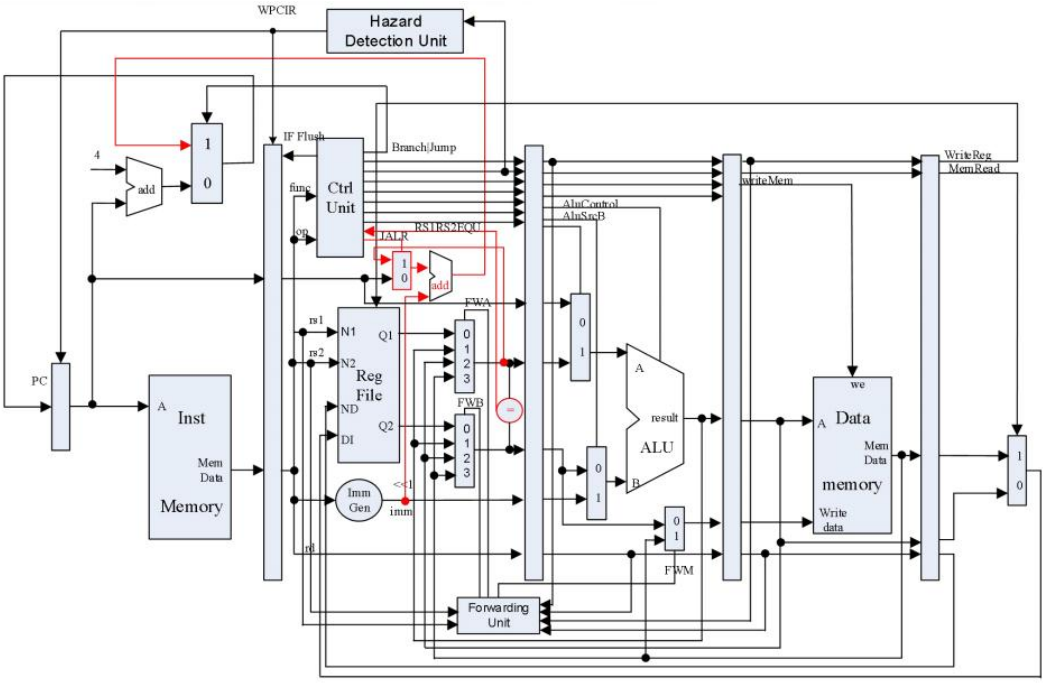
```
//modified
//IF_BRANCH_CTRL
MUX2T1_32 mux_IF(.I0(jump_PC_ID),.I1(PC_4_IF),.s(Branch_ctrl),.o(next_PC_IF));
```

ID 阶段跳转地址计算：

```
MUX2T1_32 mux_branch_ID(.I0(PC_ID),.I1(rs1_data_ID),.s(JALR),.o(addA_ID));

//跳转地址计算
add_32 add_branch_ID(.a(addA_ID),.b(Imm_out_ID),.c(jump_PC_ID));

cmp_32 cmp_ID(.a(rs1_data_ID),.b(rs2_data_ID),.ctrl(cmp_ctrl),.c(cmp_res_ID));
```



2.2 Foward 实现思路

在此实验中 DataPath 加入 Foward 主要是为了解决数据冒险，其发生于指令出现 RAW(read after write)情况，寄存器读取发生在 ID 阶段而写入在三个时钟周期后的 WB 阶段，可能出现后面指令需要读取的寄存器数据前面指令在读取时尚未完成写入，形成数据冒险。

根据指令的类型与出现顺序，考虑不同指令在写寄存器数据时该数据产生的时钟周期，大部分写寄存器数据在 EXE 阶段由 ALU 产生，L 类型指令写寄存器数据在 MEM 阶段产生，我们可以将 RAW 分为以下五种情况：

1. 非 L 类写寄存器指令后紧接读相应寄存器

解决方法：将 EXE 阶段的 ALU 运算结果 ALUout_EXE forward 到 ID 阶段，作为读寄存器输出提供给紧接的指令。

2. 非 L 类写寄存器指令后隔一条出现读相应寄存器指令

解决方法：将此时处在 MEM 阶段该写寄存器指令的 ALU 运算结果 ALUout_MEM forward

到 ID 阶段，作为读寄存器输出提供给紧接的指令。

3. L 类写寄存器（读存储器）指令后紧接 S 类型写存储器（读相应寄存器）指令

解决方法：由于读写存储器均发生在 MEM 阶段，在 L 类写寄存器指令读出存储器数据后 S 类处于 EXE 阶段，可以将数据从 MEM 读取结果 Datain_MEM forward 到 EXE 阶段。

4. L 类写寄存器指令后隔一条出现读相应寄存器指令

解决方法：此时无论是 S 类还是非 S 类指令，均可以通过 MEM 读取结果 Datain_MEM forward 到 ID 阶段解决。

5. L 类写寄存器指令后紧接非 S 类型读相应寄存器指令

解决方法：此时只能将身后指令 stall 一个时钟周期，之后转变为第四类情况通过 MEM 读取结果 Datain_MEM forward 到 ID 阶段解决。

综上所述，forward 的数据输入在 EXE 的 ALUout_EXE，MEM 阶段的 ALUout_MEM，Datain_MEM，而数据接受在 ID 阶段的 mux_forward_A 与 mux_forward_B 两个多路选择器、EXE 阶段的 mux_forward_EXE。

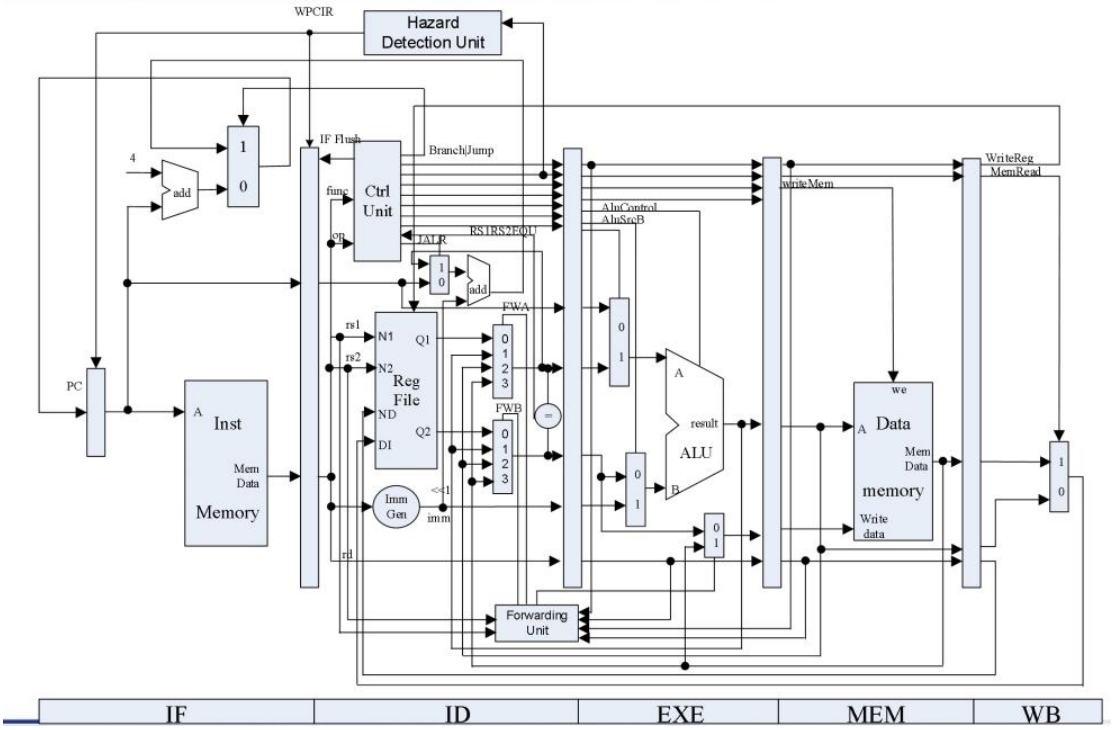
由此我们得到三个 forward 的多路选择器实现：

```
MUX4T1_32 mux_forward_A(.I0(rs1_data_reg),.I1(ALUout_EXE),.I2(ALUout_MEM),.I3(Datain_MEM),
    .s(forward_ctrl_A),.o(rs1_data_ID));

MUX4T1_32 mux_forward_B(.I0(rs2_data_reg),.I1(ALUout_EXE),.I2(ALUout_MEM),.I3(Datain_MEM),
    .s(forward_ctrl_B),.o(rs2_data_ID));

MUX2T1_32 mux_forward_EXE(.I0(rs2_data_EXE),.I1(Datain_MEM),.s(forward_ctrl_ls),.o(Dataout_EXE));
```

2.3 RV32core 连线解析

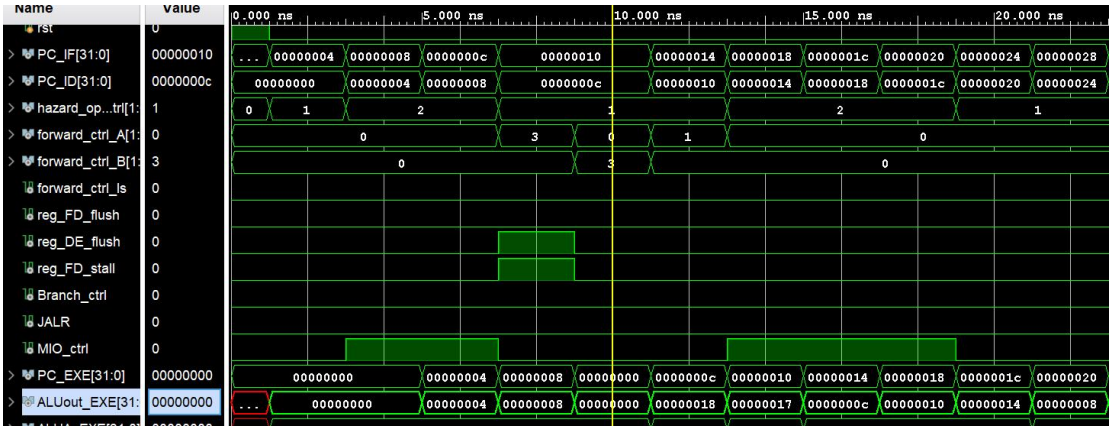


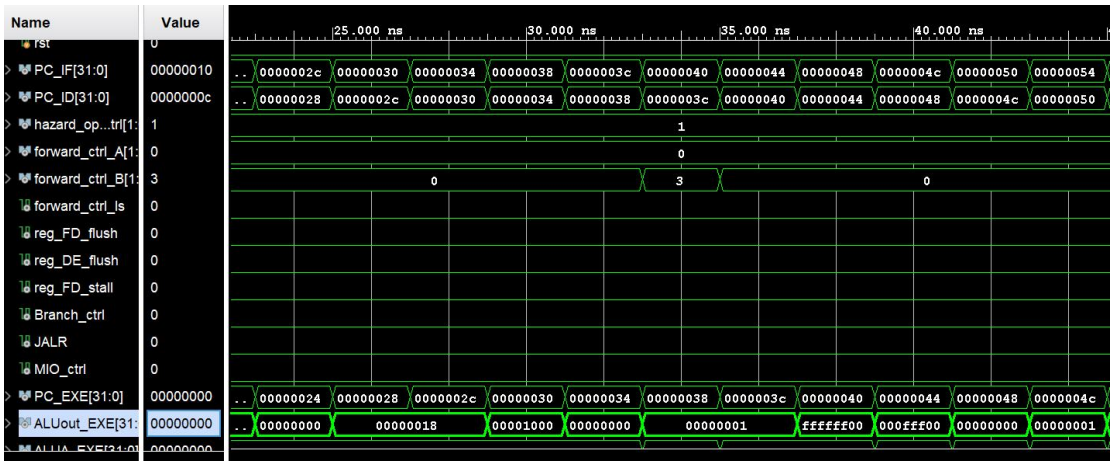
需修改部分为控制冒险的指令生成与数据冒险的 forward control，已于前两部分阐明。

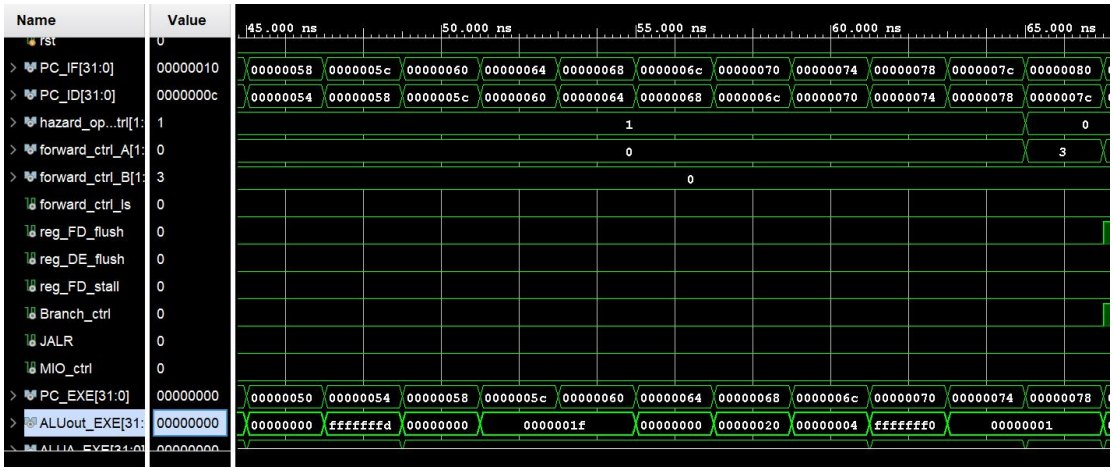
三、 实验过程和数据记录及结果分析

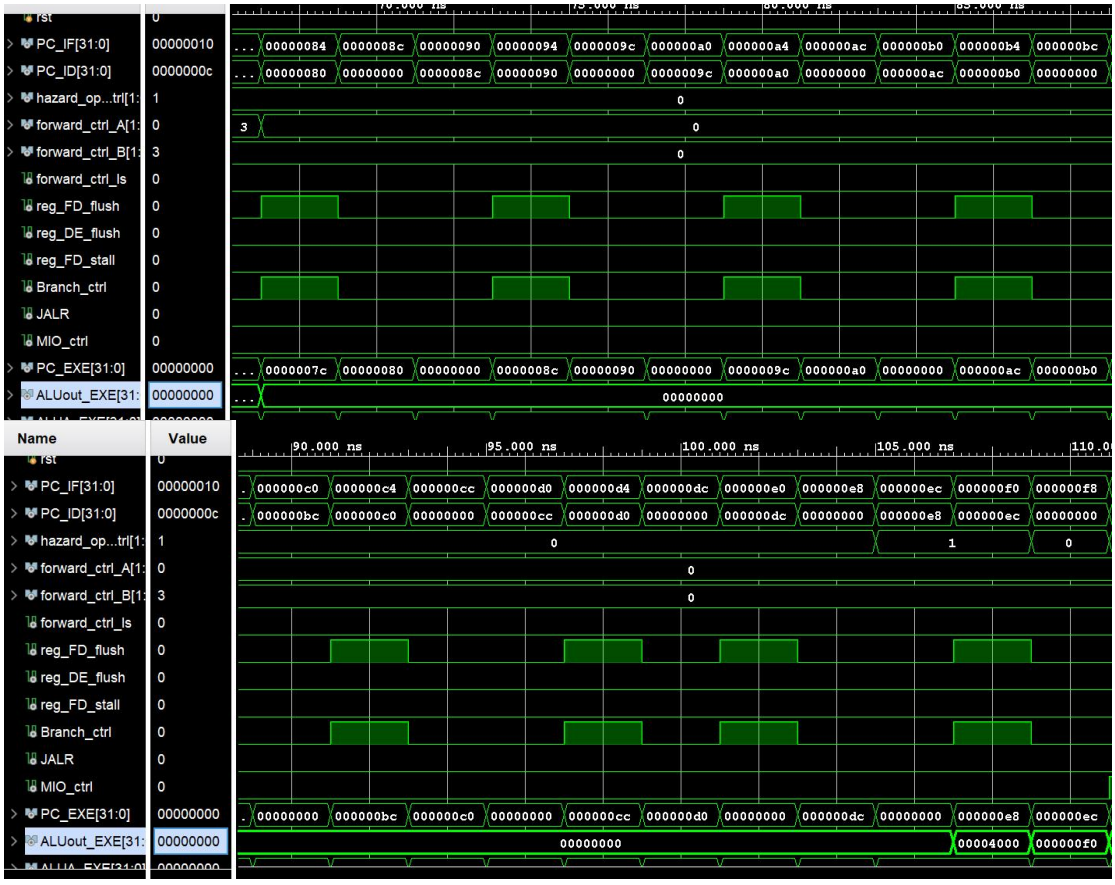
Tips: 请给出本次实验仿真的完整截图与各种 forward 和 predict not taken 发生时的截图，并简要解释

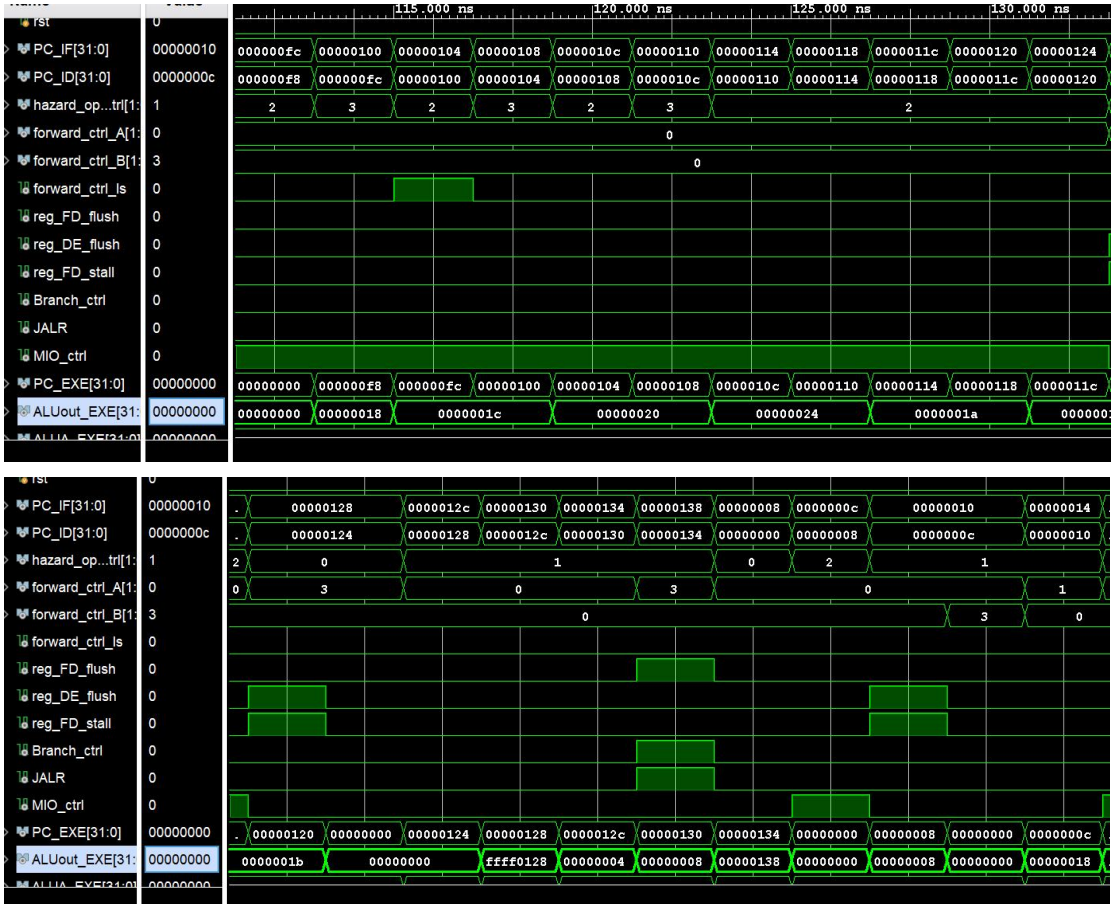
3.1 仿真完整截图：




Name
Value
25.000 ns
30.000 ns
35.000 ns
40.000 ns
rst
PC_IF[31:0]
PC_ID[31:0]
hazard_op...trl[1:
forward_ctrl_A[1:
forward_ctrl_B[1:
forward_ctrl_ls
reg_FD_flush
reg_DE_flush
reg_FD_stall
Branch_ctrl
JALR
MIO_ctrl
PC_EXE[31:0]
ALUout_EXE[31:

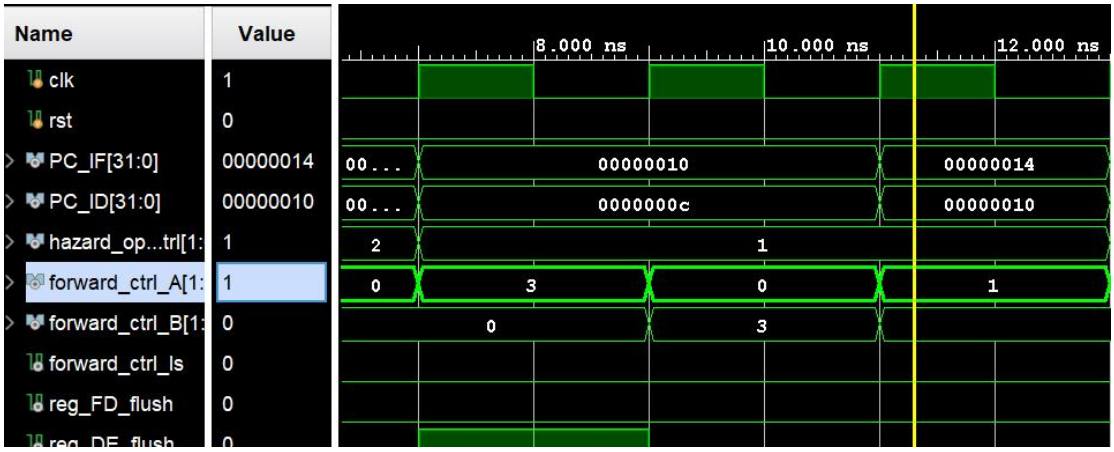

Name
Value
45.000 ns
50.000 ns
55.000 ns
60.000 ns
65.000 ns
rst
PC_IF[31:0]
PC_ID[31:0]
hazard_op...trl[1:
forward_ctrl_A[1:
forward_ctrl_B[1:
forward_ctrl_ls
reg_FD_flush
reg_DE_flush
reg_FD_stall
Branch_ctrl
JALR
MIO_ctrl
PC_EXE[31:0]
ALUout_EXE[31:


Name
Value
90.000 ns
95.000 ns
100.000 ns
105.000 ns
rst
PC_IF[31:0]
PC_ID[31:0]
hazard_op...trl[1:
forward_ctrl_A[1:
forward_ctrl_B[1:
forward_ctrl_ls
reg_FD_flush
reg_DE_flush
reg_FD_stall
Branch_ctrl
JALR
MIO_ctrl
PC_EXE[31:0]
ALUout_EXE[31:



3.2 Forward 与 Predict not taken

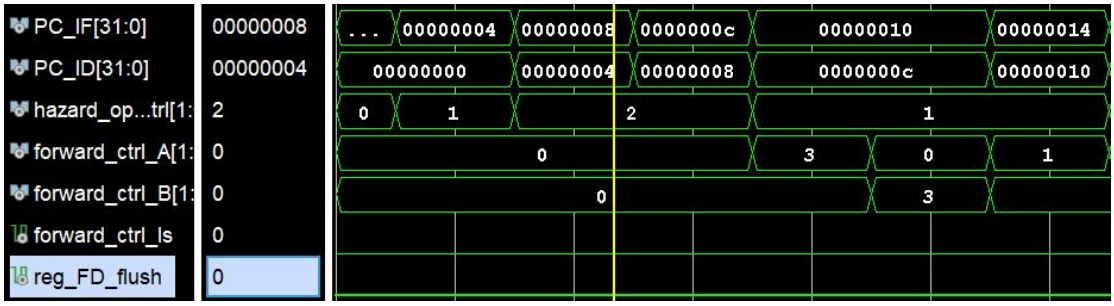
（1）EXE 阶段 alu_out forward 到 ID



3	004100b3	C		add x1, x2, x4
4	fff08093	10		addi x1, x1, -1

此时 C 指令 alu 运算结果需要被下一条指令读取，结果从 EXE 阶段 forward 到 ID

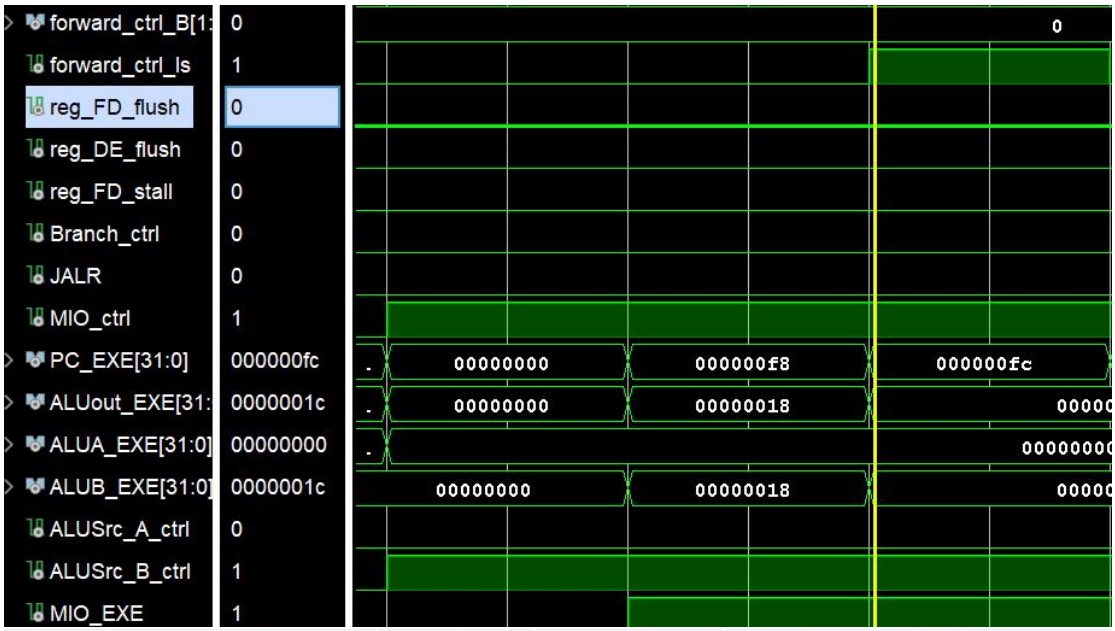
（2）MEM 阶段 L 类型指令 Datain forward 到 ID



1	00402103	4		lw x2, 4(x0)
2	00802203	8		lw x4, 8(x0)
3	004100b3	C		add x1, x2, x4

此时 4 指令在 MEM 阶段且为 load 指令，需要将读内存结果 x2 结果 forward 到 ID 阶段 C 指令

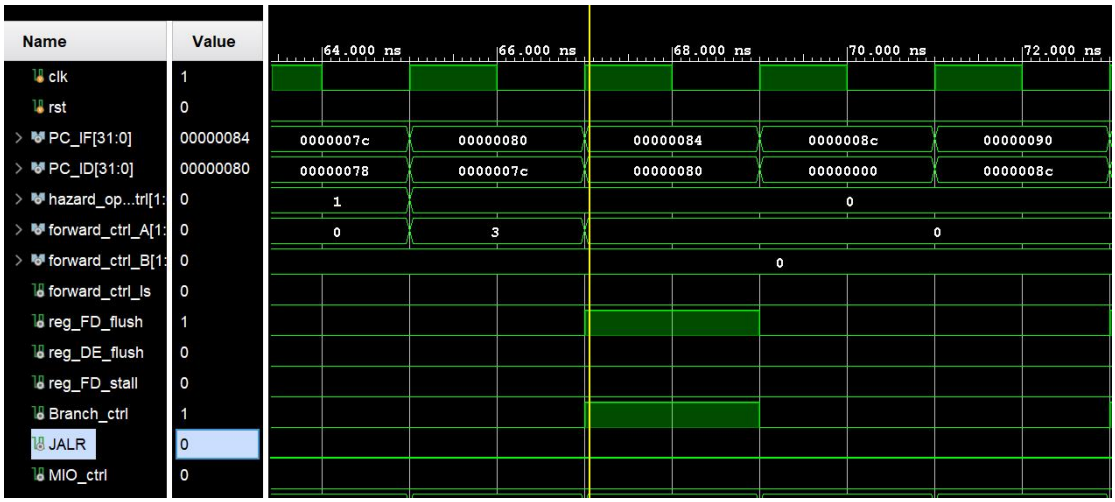
（3）MEM 阶段 L 类型指令 Datain forward 到 EXE



62	01802403	F8		lw x8, 24(x0)
63	00802e23	FC		sw x8, 28(x0)

此处 F8 指令处在 MEM 阶段而它的读内存结果需要被紧接的 FC 指令写内存，需从 MEM forward 到 EXE 阶段

（4）Predict Not Taken



32	00420663	80		beq x4,x4,label0
33	00000013	84		addi x0,x0,0
34	00000013	88		addi x0,x0,0
35	00421863	8C	label0:	bne x4,x4,label1

此处 ID 指令 PC 为 80 时，predict not taken 策略预测错误，跳转发生，产生 IF-ID 阶段寄存器的 flush 信号冲刷并重新读取正确 PC，可见 PC_ID 信号在被冲刷后正确读取了 Label0（8C）地址的指令。

四、讨论与心得

1. 在编写较大的模块时，通常可以先完成部分小模块，在小模块完善时针对其编写测试文件进行功能测试，保证每一个小模块都正常运行，降低顶层模块的 debug 难度。

2. 对于理解复杂的模块，可以通过追踪每一个信号的流程，理解每一个信号的含义，进而理解数据流和整体模块功能。

3. 本次在下载 bitstream 中遇到笔记本无法检测到板子信号的问题，原因可能在于软件版本过久，没有完整阅读实验指南进行软件更新，需要更加仔细。