浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：刘思锐

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：3200102708

指导教师：陈文智

2022 年　12 月　20 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：Dynamically Scheduled Pipelines using Scoreboarding

学生姓名：刘思锐 专业：计算机科学与技术 学号：3200102708

同组学生姓名：陈镛屹 指导老师：陈文智

实验地点：家 实验日期：2022 年 12 月 20 日

一、 实验目的和要求

1. Understand the principle of piplines that support multicycle operations.
2. Understand the principle of Dynamic Scheduling with a Scoreboard.
3. Master the design methods of piplines that support multicycle operations.
4. Master the design methods of Dynamically Scheduled Pipelines using Scoreboarding.
5. Master verification methods of Dynamically Scheduled Pipelines using Scoreboarding.

二、 实验内容和原理

1. Redesign the pipelines with IF/IS/RO/FU/WB stages and supporting multicycle operations.
2. Design of a scoreboard and integrate it to CPU.
3. Verify the Pipelined CPU with program and observe the execution of program.
4. 本次实验相对 Lab5 最大的改动是控制单元从简单的有 FU 被占用即 stall 变为基于计分板的动态流水线。

三、 实验过程和数据记录

1. Normal_stall

在发生结构竞争或 WAW 时，需要 normal_stall 来解决。下图中前 5 行为逐个 FU

判断是否存在结构竞争，最后一行为通过寄存器计分板判断是否存在 WAW。

```
// fill sth. here
assign normal_stall = (use_ALU  &  FUS[`FU_ALU][`BUSY] )  |
                      (use_MEM  &  FUS[`FU_MEM][`BUSY] )  |
                      (use_MUL  &  FUS[`FU_MUL][`BUSY] )  |
                      (use_DIV  &  FUS[`FU_DIV][`BUSY] )  |
                      (use_JUMP &  FUS[`FU_JUMP][`BUSY])  |
                      (|RRS[dst]);
```

2. 判断 WAR

以 ALU 为例，当另外四个运算单元还没有读寄存器的值时，ALU 不能写入。因此判断的逻辑是当其他 FU 的源和 ALU 的目标寄存器不相同，或者其他 FU 的源已经完成读数时，ALU_WAR 置 1 表示可以写回。

```
// ensure WAR:
// If an FU hasn't read a register value (R0), don't write to it.
wire ALU_WAR = (
    (FUS[`FU_MEM][`SRC1_H:`SRC1_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ~FUS[`FU_MEM][`RDY1])  &   //fill sth. here
    (FUS[`FU_MEM][`SRC2_H:`SRC2_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ~FUS[`FU_MEM][`RDY2])  &   //fill sth. here
    (FUS[`FU_MUL][`SRC1_H:`SRC1_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ~FUS[`FU_MUL][`RDY1])  &   //fill sth. here
    (FUS[`FU_MUL][`SRC2_H:`SRC2_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ~FUS[`FU_MUL][`RDY2])  &   //fill sth. here
    (FUS[`FU_DIV][`SRC1_H:`SRC1_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ~FUS[`FU_DIV][`RDY1])  &   //fill sth. here
    (FUS[`FU_DIV][`SRC2_H:`SRC2_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ~FUS[`FU_DIV][`RDY2])  &   //fill sth. here
    (FUS[`FU_JUMP][`SRC1_H:`SRC1_L] != FUS[`FU_ALU][`DST_H:`DST_L] | ~FUS[`FU_JUMP][`RDY1]) &   //fill sth. here
    (FUS[`FU_JUMP][`SRC2_H:`SRC2_L] != FUS[`FU_ALU][`DST_H:`DST_L] | ~FUS[`FU_JUMP][`RDY2])      //fill sth. here
);
```

另外四个 FU 的 WAR 信号与此同理，在此省略。

3. IS 级流水线控制

当本条指令所需的 FU 有空闲时，将本条指令的相关信息放入计分板。但是这里不能确定的是 IMM 计分板的注释是不是写的不太对，以及为什么用的是 RO_en 而不是 IS_en。

```
if (RO_en) begin
    // not busy, no WAW, write info to FUS and RRS
    if (|dst) RRS[dst] <= use_FU;
    FUS[use_FU][`BUSY] <= 1'b1;
    FUS[use_FU][`OP_H:`OP_L] <= op;
    FUS[use_FU][`DST_H:`DST_L] <= dst;
    FUS[use_FU][`SRC1_H:`SRC1_L] <= src1;
    FUS[use_FU][`SRC2_H:`SRC2_L] <= src2;
    FUS[use_FU][`FU1_H:`FU1_L] <= fu1;
    FUS[use_FU][`FU2_H:`FU2_L] <= fu2;
    FUS[use_FU][`RDY1] <= rdy1;
    FUS[use_FU][`RDY2] <= rdy2;
    FUS[use_FU][`FU_DONE] <= 1'b0;
    IMM[use_FU] <= imm;
    PCR[use_FU] <= PC;
end
```

4. RO 级流水线控制

以 JUMP、ALU 为例，当某个 FU 所需的两个数据源都 ready 时，将 Ready 的值置 0。CtrlUnit 只用生成控制信号即可，取数在另外的地方。

```
    // RO
    if (FUS[`FU_JUMP][`RDY1] & FUS[`FU_JUMP][`RDY2]) begin
        // JUMP
        FUS[`FU_JUMP][`RDY1] <= 1'b0;
        FUS[`FU_JUMP][`RDY2] <= 1'b0;
    end
    else if (FUS[`FU_ALU][`RDY1] & FUS[`FU_ALU][`RDY2]) begin
        // ALU
        // fill sth. here.
        FUS[`FU_ALU][`RDY1] <= 1'b0;
        FUS[`FU_ALU][`RDY2] <= 1'b0;
    end
```

另外三个运算单元同理，在此省略。

5. EX 级流水线控制

根据 FU 传回的 done 信号更新计分板即可。

```
// EX
FUS[`FU_ALU][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_ALU][`FU_DONE];
FUS[`FU_MEM][`FU_DONE] <= MEM_done ? 1 : FUS[`FU_MEM][`FU_DONE];
FUS[`FU_MUL][`FU_DONE] <= MUL_done ? 1 : FUS[`FU_MUL][`FU_DONE];
FUS[`FU_DIV][`FU_DONE] <= DIV_done ? 1 : FUS[`FU_DIV][`FU_DONE];
FUS[`FU_JUMP][`FU_DONE] <= JUMP_done ? 1 : FUS[`FU_JUMP][`FU_DONE];
```

6. WB 级流水线控制

首先前面已经生成了 WAR 信号，只有不会发生 WAR 冲突时允许写入。写入时需要做两件事，首先将记分牌中指令相关内容清空，然后更新其相关的 Ready 信号表示数据已经可用。

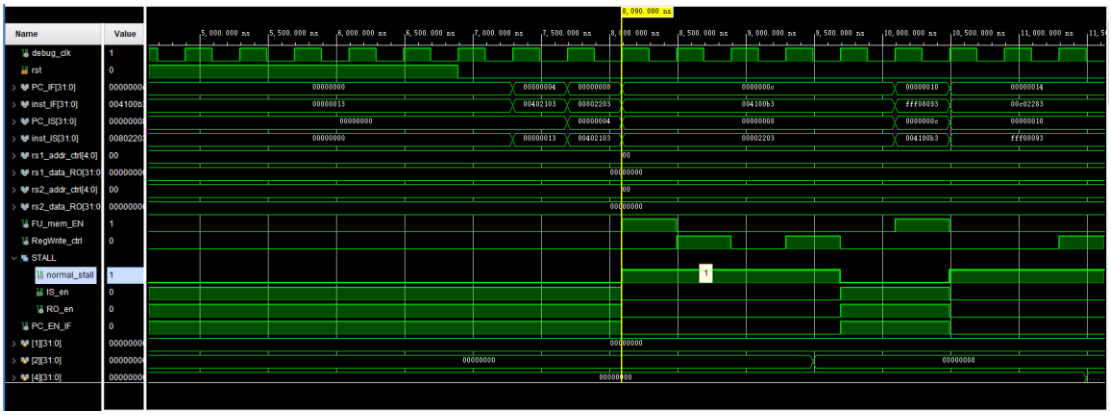
```
// WB
// JUMP
if (FUS[`FU_JUMP][`FU_DONE] & JUMP_WAR) begin
    FUS[`FU_JUMP] <= 32'b0;
    RRS[FUS[`FU_JUMP][`DST_H:`DST_L]] <= 3'b0;

    // ensure RAW
    if (FUS[`FU_ALU][`FU1_H:`FU1_L] == `FU_JUMP) FUS[`FU_ALU][`RDY1] = 1'b1;
    if (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_JUMP) FUS[`FU_MEM][`RDY1] = 1'b1;
    if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_JUMP) FUS[`FU_MUL][`RDY1] = 1'b1;
    if (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_JUMP) FUS[`FU_DIV][`RDY1] = 1'b1;

    if (FUS[`FU_ALU][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_ALU][`RDY2] = 1'b1;
    if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_MEM][`RDY2] = 1'b1;
    if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_MUL][`RDY2] = 1'b1;
    if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_DIV][`RDY2] = 1'b1;
end
// ALU
// fill sth. here
else if (FUS[`FU_ALU][`FU_DONE] & ALU_WAR) begin
    FUS[`FU_ALU] <= 32'b0;
    RRS[FUS[`FU_ALU][`DST_H:`DST_L]] <= 3'b0;

    // ensure RAW
    if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_JUMP][`RDY1] = 1'b1;        //fill sth. here
    if (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_MEM][`RDY1] = 1'b1;          //fill sth. here
    if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_MUL][`RDY1] = 1'b1;          //fill sth. here
    if (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_DIV][`RDY1] = 1'b1;          //fill sth. here

    if (FUS[`FU_JUMP][`FU2_H:`FU2_L] == `FU_ALU) FUS[`FU_JUMP][`RDY2] = 1'b1;        //fill sth. here
    if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_ALU) FUS[`FU_MEM][`RDY2] = 1'b1;          //fill sth. here
    if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_ALU) FUS[`FU_MUL][`RDY2] = 1'b1;          //fill sth. here
    if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_ALU) FUS[`FU_DIV][`RDY2] = 1'b1;          //fill sth. here
end
```

这里以 JUMP、ALU 两个 FU 为例，其余三者同理，在此省略。

四、 仿真结果分析

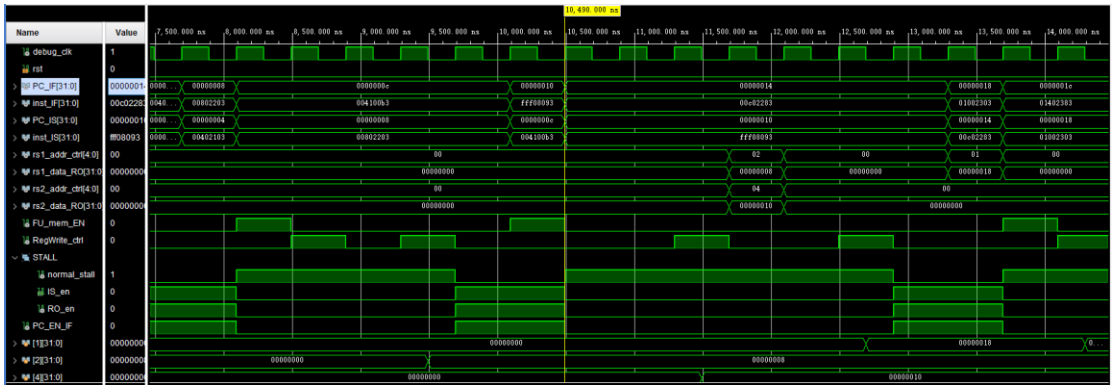
第一第二条指令发生了结构竞争，这里可以观察到正常产生了 stall 信号，第二条指令直到三个周期后才发射。

1	00402103	4		lw x2, 4(x0)	
2	00802203	8		lw x4, 8(x0)	Structural Hazard



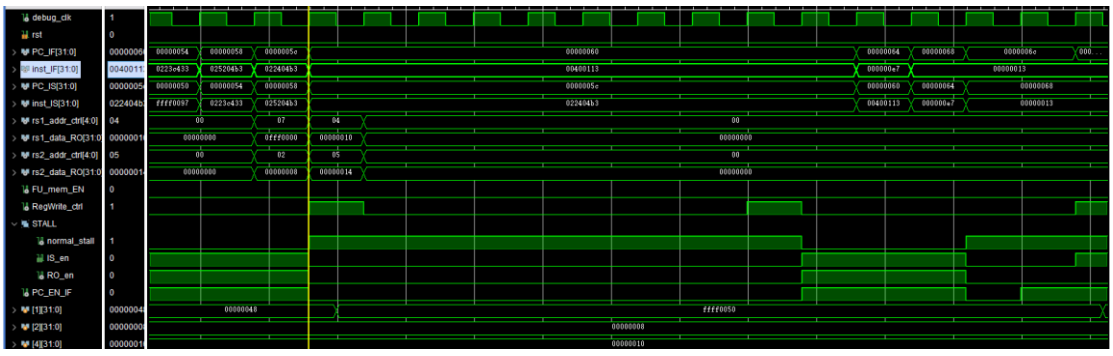
第三第四条指令发生了 WAW 冲突，同样成功产生了 stall 信号，直到前一条指令完成写入之后才发射。

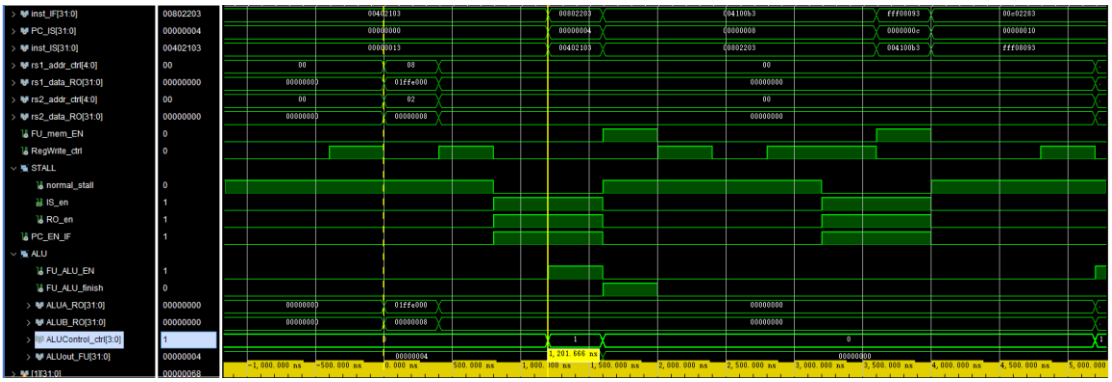
3	004100b3	C		add x1, x2, x4	
4	fff08093	10		addi x1, x1, -1	WAW



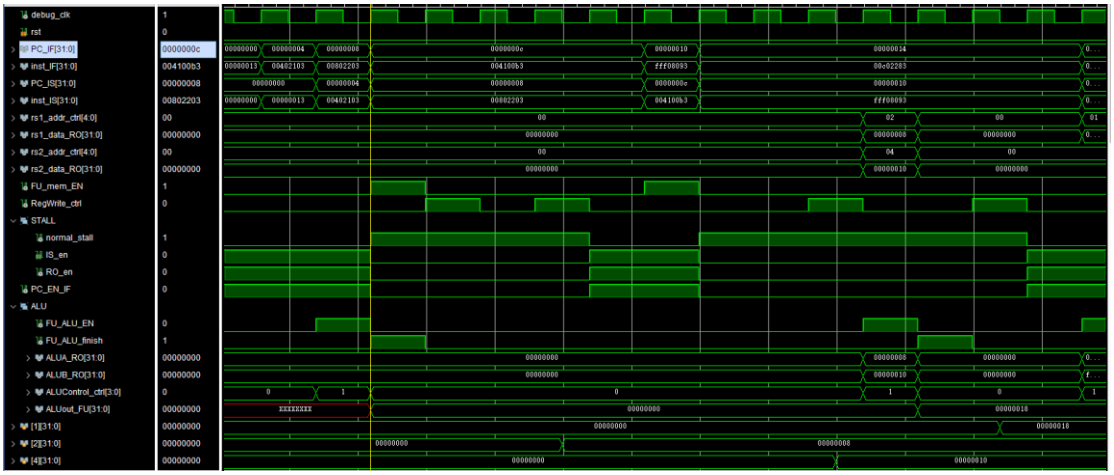
下面的情况中，0x58 位置的指令与 0x5C 位置的指令发生了结构竞争，产生了 stall，前者竞争完成后后者才能发射。而直到 0x5C 位置的指令读取了 x2 之后，0x60 位置的指令才能写回结果。

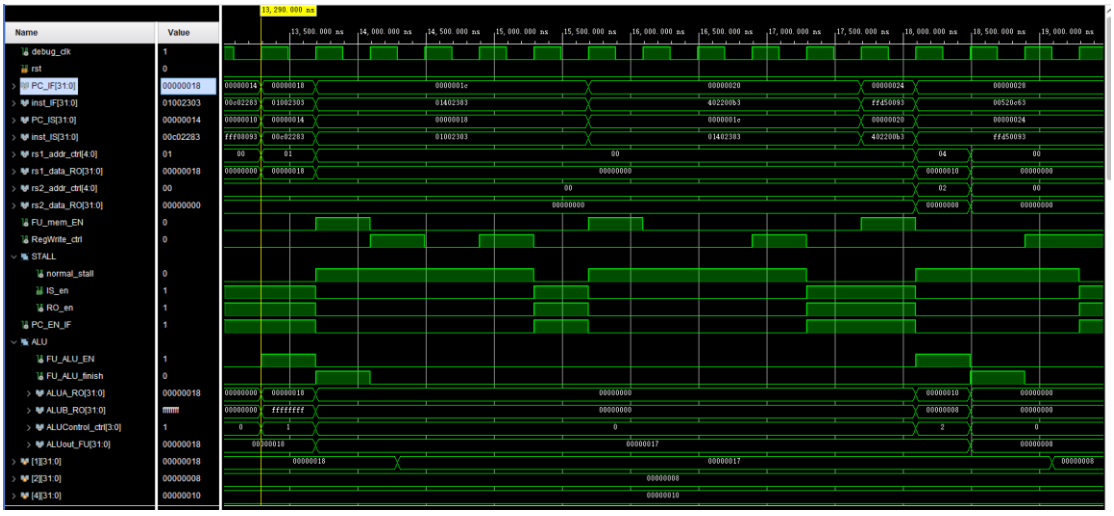
22	025204b3	58		mul x9, x4, x5	
23	022404b3	5C		mul x9, x8, x2	St. Ha./RAW/WAW
24	00400113	60		addi x2, x0, 4	WAR

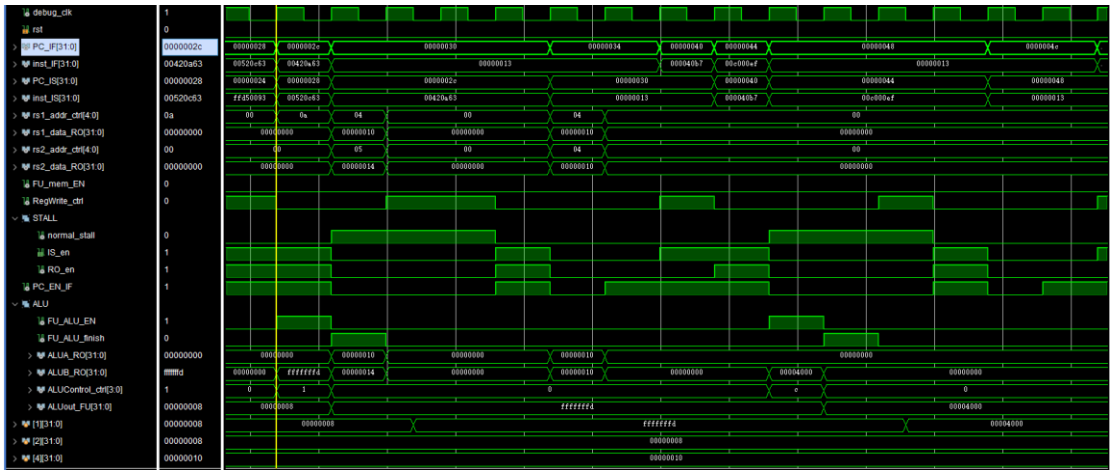


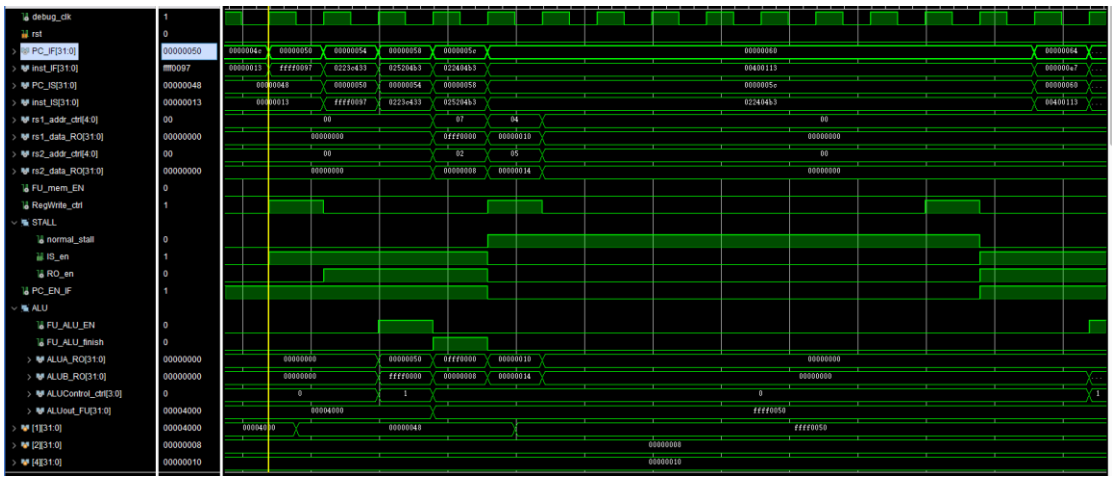


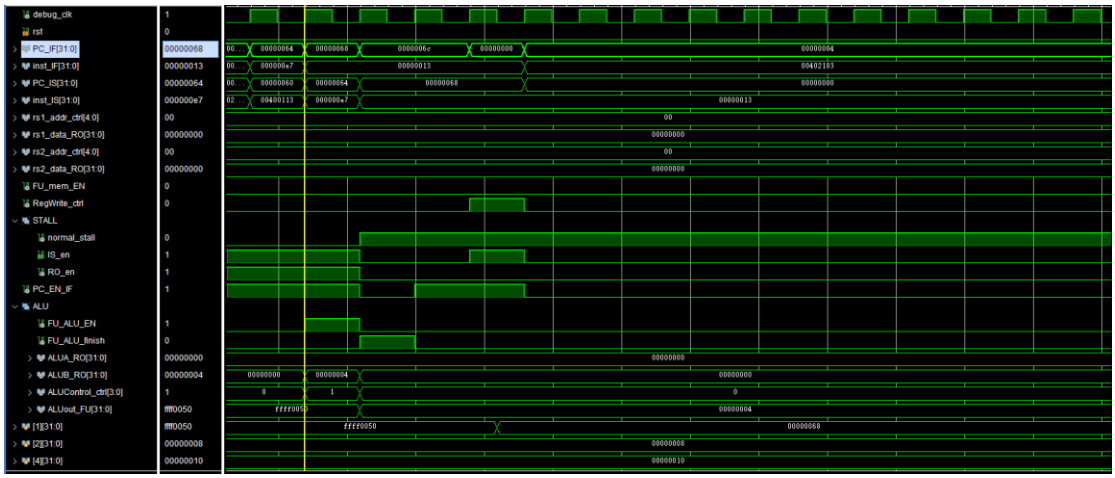
完整的仿真内容如下：

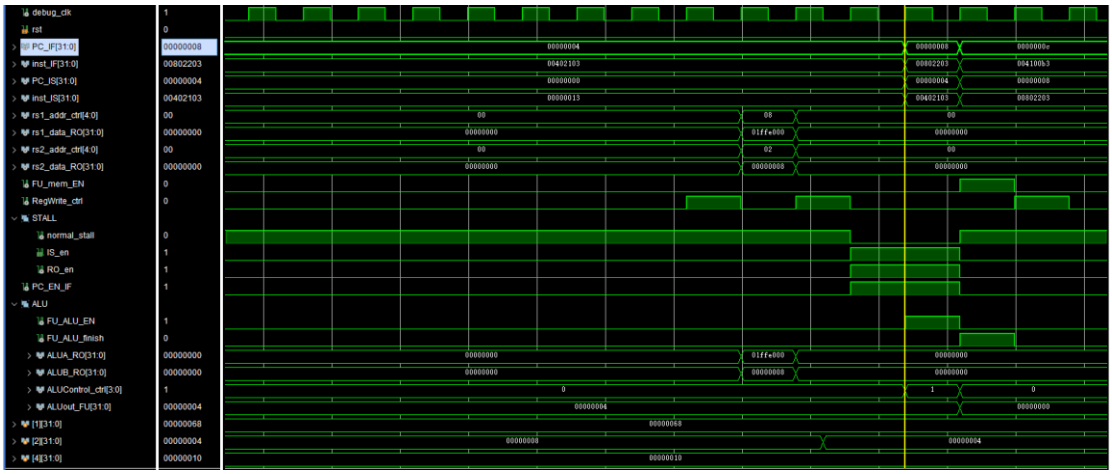





debug_clk
rst
PC_IF[31:0]
inst_IF[31:0]
PC_IS[31:0]
inst_IS[31:0]
rs1_addr_ctrl[4:0]
rs1_data_RO[31:0]
rs2_addr_ctrl[4:0]
rs2_data_RO[31:0]
FU_mem_EN
RegWrite_ctrl
STALL
normal_stall
IS_en
RO_en
PC_EN_IF
ALU
FU_ALU_EN
FU_ALU_finish
ALUA_RO[31:0]
ALUB_RO[31:0]
ALUControl_ctrl[3:0]
ALUout_FU[31:0]
[1][31:0]
[2][31:0]
[4][31:0]


debug_clk
rst
PC_IF[31:0]
inst_IF[31:0]
PC_IS[31:0]
inst_IS[31:0]
rs1_addr_ctrl[4:0]
rs1_data_RO[31:0]
rs2_addr_ctrl[4:0]
rs2_data_RO[31:0]
FU_mem_EN
RegWrite_ctrl
STALL
normal_stall
IS_en
RO_en
PC_EN_IF
ALU
FU_ALU_EN
FU_ALU_finish
ALUA_RO[31:0]
ALUB_RO[31:0]
ALUControl_ctrl[3:0]
ALUout_FU[31:0]
[1][31:0]
[2][31:0]
[4][31:0]


debug_clk
rst
PC_IF[31:0]
inst_IF[31:0]
PC_IS[31:0]
inst_IS[31:0]
rs1_addr_ctrl[4:0]
rs1_data_RO[31:0]
rs2_addr_ctrl[4:0]
rs2_data_RO[31:0]
FU_mem_EN
RegWrite_ctrl
STALL
normal_stall
IS_en
RO_en
PC_EN_IF
ALU
FU_ALU_EN
FU_ALU_finish
ALUA_RO[31:0]
ALUB_RO[31:0]
ALUControl_ctrl[3:0]
ALUout_FU[31:0]
[1][31:0]
[2][31:0]
[4][31:0]



五、 讨论与心得

本次实验各个功能模块的划分比较清晰，并且各个控制单元之间代码的重复程度很高，代码本身并不困难。困难的是理清代码框架的逻辑，状态机各自管理哪一部分的内容，各个计分板的含义及内部各个 bit 的作用，各种冲突的判断和避免分别应该在哪个部分实现等等。

在真正开始写代码之前经历了一段非常长的看着留空无法下手的时期，但是在整理思路的过程中，我对动态流水线的理解也有了相当的进步。