浙江大学

本科实验报告

课程名称：	计算机体系结构
姓　　名：	姜雨童
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
邮　　箱：	3220103450@zju.edu.cn
QQ　　号：	1369218489
电　　话：	18867766468
指导教师：	王小航
报告日期：	2024 年 12 月 14 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：Lab06 Dynamically Scheduled Pipelines using Scoreboarding

学生姓名：姜雨童 学号：33220103450 同组学生姓名：/

实验地点：玉泉曹西 301 实验日期：2024 年 12 月 14 日

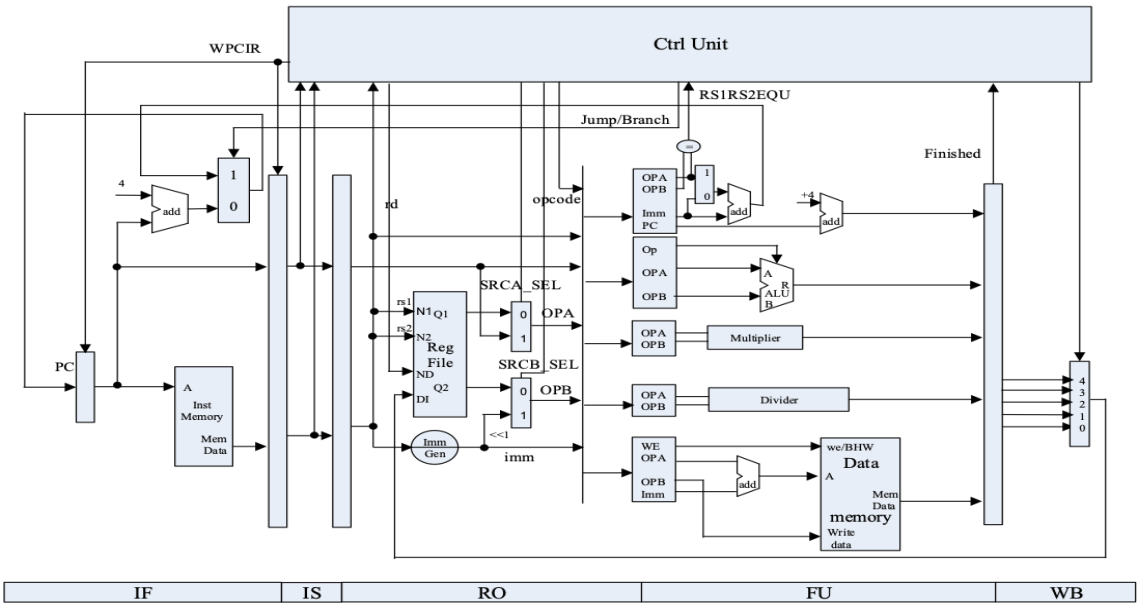
一、目标与原理

1-1 实验目标

- Redesign the pipelines with IF/IS/RO/FU/WB stages and supporting multicycle operations.
- Design of a scoreboard and integrate it to CPU.

1-2 实验内容

现代处理器的指令是顺序发射、乱序执行的，本实验要利用 scoreboard 实现指令的乱序执行。使用的 pipeline CPU 架构和实验五类似，但是分为五个阶段（如图）：
IF 取指令，IS 发射指令，RO 读取操作数，FU 计算，WB 写回。



最大的不同在于 Ctrl Unit 的控制逻辑不同，需要实现 scoreboard 里面的三张表。而 Instruction status 可以通过指令目前所处的流水线阶段得知，因此只需要实现 Functional Unit status 和 Register result status 的支持。

对于 Functional unit status，本实验在原本数据的基础上加上了 FU_DONE，用于记录指令是否已经完成。因此表示方式为：**reg**[31:0] FUS[1:5]（e.g. FUS[`FU_MEM][`SRC1_H:`SRC1_L]）。

	Functional unit status								
Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2	R3				No	
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

```
// function unit
`define FU_BLANK    3'd0
`define FU_ALU      3'd1
`define FU_MEM      3'd2
`define FU_MUL      3'd3
`define FU_DIV      3'd4
`define FU_JUMP     3'd5
```

```
// bits in FUS
`define BUSY     0
`define OP_L     1
`define OP_H     5
`define DST_L    6
`define DST_H    10
`define SRC1_L   11
`define SRC1_H   15
`define SRC2_L   16
`define SRC2_H   20
```

```
`define FU1_L    21
`define FU1_H    23
`define FU2_L    24
`define FU2_H    26
`define RDY1     27
`define RDY2     28
`define FU_DONE  29
```

对于 Register result status，以 **reg**[2:0] RRS[0:31]的形式记录 32 个寄存器会被哪些 FU 使用。

	Register result status								
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1	Integer			Add	Divide			

二、操作方法与实验步骤

2-1 Normal_stall

Normal_stall 用于检测 structure hazard 和 WAW 。利用 FUS 中的 BUSY 位来判断是否存在 structure hazard，利用 RRS[dst]（表示目标寄存器 rd 的 RRS）来判断是否有 WAW：

```
// normal stall: structural hazard or WAW
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]);
```

2-2 WAR

如果其他运算单元还没有读寄存器的值，则当前寄存器不能执行写操作，因此判断其他 FU 的 src 和当前 FU 的 dst 是否一致；若一致但其他 FU 已经读完，则也能进行当前 FU 的写回。xxx_WAR 信号置一表示 xxx 对应 FU 可以写回。

ALU：

```
    // If an FU hasn't read a register value (RO), 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
    );
```

其他四个 FU 也按照此逻辑进行代码补充，此处不做赘述。

2-3 IS 阶段

当前指令所需 FU 闲置时，发射指令，将信息加载到 scoreboard 里：

```
            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;
                //fill sth. here.
```

```
            IMM[use_FU] <= imm;
            PCR[use_FU] <= PC;
        end
```

2-4 RO 阶段

代码给出了 JUMP 的例子：当 JUMP 中两个数据源都 ready 了，才能读取操作数，且需要把 Ready 值都置零。由于读取的操作在别处执行，此处只需要生成控制信号。类似地，可以补全其他四个 FU 对应代码：

```
        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
            FUS[`FU_ALU][`RDY1] <= 1'b0;
            FUS[`FU_ALU][`RDY2] <= 1'b0;
        end
        else if (FUS[`FU_MEM][`RDY1] & FUS[`FU_MEM][`RDY2]) begin
            // MEM
            FUS[`FU_MEM][`RDY1] <= 1'b0;
            FUS[`FU_MEM][`RDY2] <= 1'b0;
        end
        else if (FUS[`FU_MUL][`RDY1] & FUS[`FU_MUL][`RDY2]) begin
            // MUL
            FUS[`FU_MUL][`RDY1] <= 1'b0;
            FUS[`FU_MUL][`RDY2] <= 1'b0;
        end
        else if (FUS[`FU_DIV][`RDY1] & FUS[`FU_DIV][`RDY2]) begin
            // DIV
            FUS[`FU_DIV][`RDY1] <= 1'b0;
            FUS[`FU_DIV][`RDY2] <= 1'b0;
        end
```

2-5 EX 阶段

Ctrl Unit 只负责控制信号，此处只要更新 scoreboard 即可：

```
        FUS[`FU_ALU][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_ALU][`FU_DONE];
        FUS[`FU_MEM][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_MEM][`FU_DONE];
        FUS[`FU_MUL][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_MUL][`FU_DONE];
        FUS[`FU_DIV][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_DIV][`FU_DONE];
        FUS[`FU_JUMP][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_JUMP][`FU_DONE];
```

2-6 WB 阶段

代码中给出了 JUMP 部分的框架：当 JUMP 模块结束且数据可以写回（不发生 WAR 冲突，前面将 JUMP_WAR 置一时），清空 scoreboard（包括 FUS 和 RRS 两张表），并更新其他 FU 中存在依赖的数据的 Ready 值（即寄存器可用）：

```
            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
```

同理可以写出另外四个 FU 对应的代码，此处不做赘述。

三、实验结果与分析

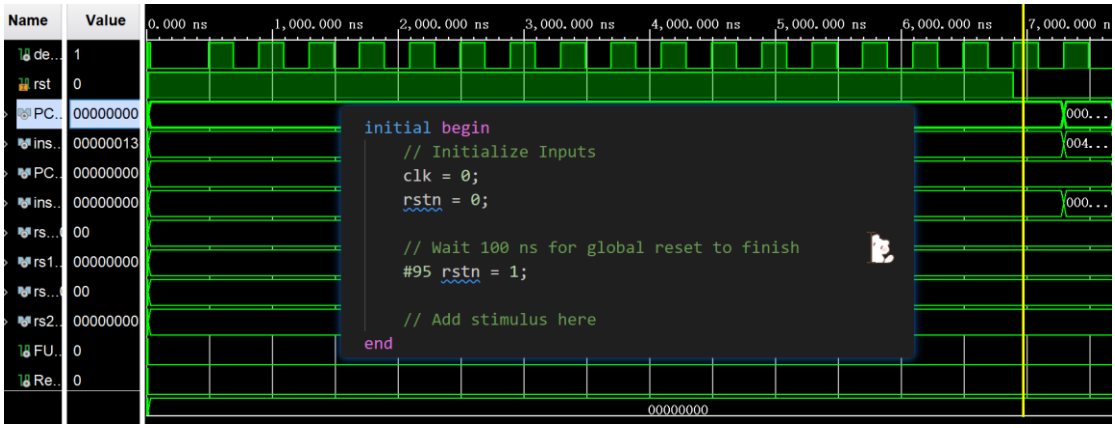
3-1 仿真

Rom.hex：

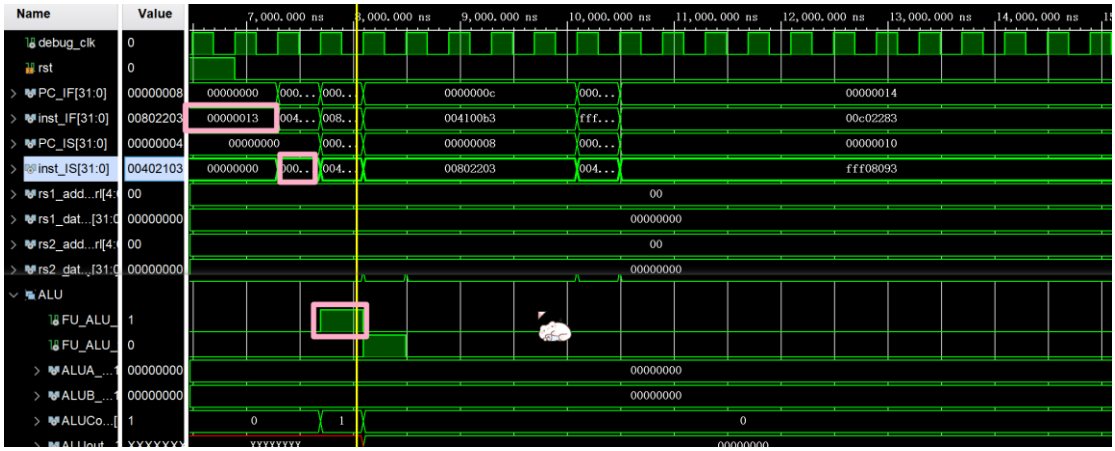
Instruction	Addr.	Label	ASM	Comment
00000013	0	__start:	addi x0, x0, 0	
00402103	4		lw x2, 4(x0)	
00802203	8		lw x4, 8(x0)	Structural Hazard
004100b3	C		add x1, x2, x4	
fff08093	10		addi x1, x1, -1	WAW
00c02283	14		lw x5, 12(x0)	
01002303	18		lw x6, 16(x0)	
01402383	1C		lw x7, 20(x0)	
402200b3	20		sub x1,x4,x2	
ffd50093	24		addi x1,x10,-3	
00520c63	28		beq x4,x5,label0	
00420a63	2C		beq x4,x4,label0	
00000013	30		addi x0,x0,0	
00000013	34		addi x0,x0,0	
00000013	38		addi x0,x0,0	

Instruction	Addr.	Label	ASM	Comment
00000013	3C		addi x0,x0,0	
000040b7	40	label0:	lui x1,4	
00c000ef	44		jal x1,12	
00000013	48		addi x0,x0,0	
00000013	4C		addi x0,x0,0	
ffff0097	50		auipc x1, 0xffff0	
0223c433	54		div x8, x7, x2	
025204b3	58		mul x9, x4, x5	St. Ha./RAW/WAW
022404b3	5C		mul x9, x8, x2	WAR
00400113	60		addi x2, x0, 4	
000000e7	64		jalr x1,0(x0)	
00000013	68		addi x0,x0,0	
00000013	6C		addi x0,x0,0	

前面一部分时间用于等待 global reset：

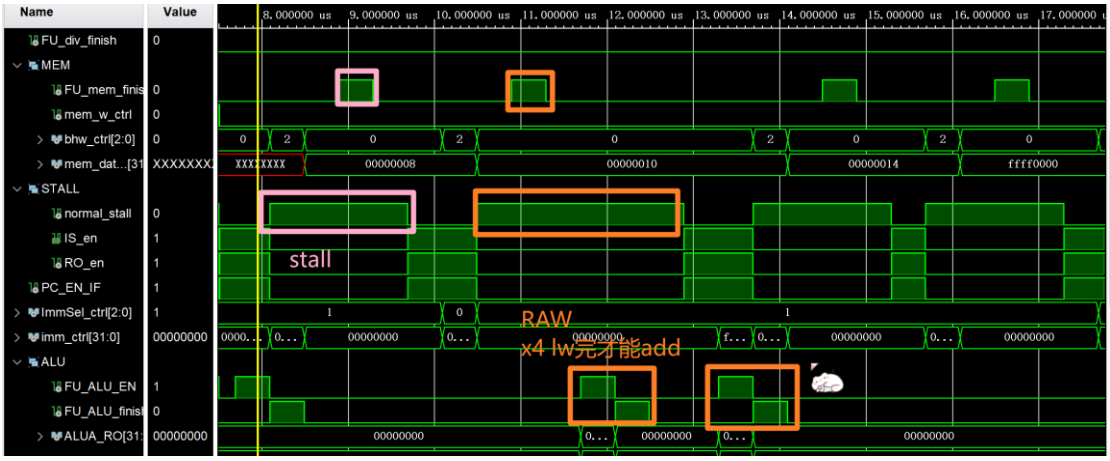


然后从第一条 add 置零开始运行 pipeline CPU（用到 FU 为 ALU）：



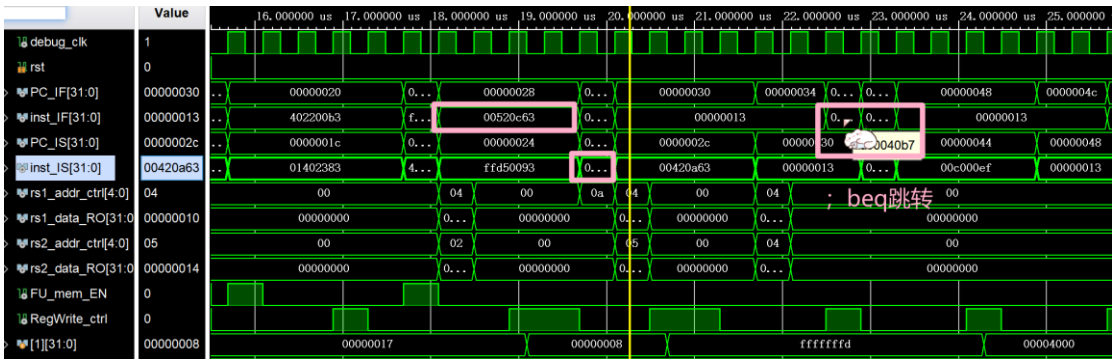
00402103	4		lw x2, 4(x0)	
00802203	8		lw x4, 8(x0)	Structural Hazard
004100b3	C		add x1, x2, x4	
fff08093	10		addi x1, x1, -1	WAW

第一条 lw 指令和第二条 lw 指令发生 **structure hazard**，产生 stall，直到三个周期后才发射第二条 lw 指令（粉色框）；addi 指令与前一条 add 发生 **WAW**（橙色框）：



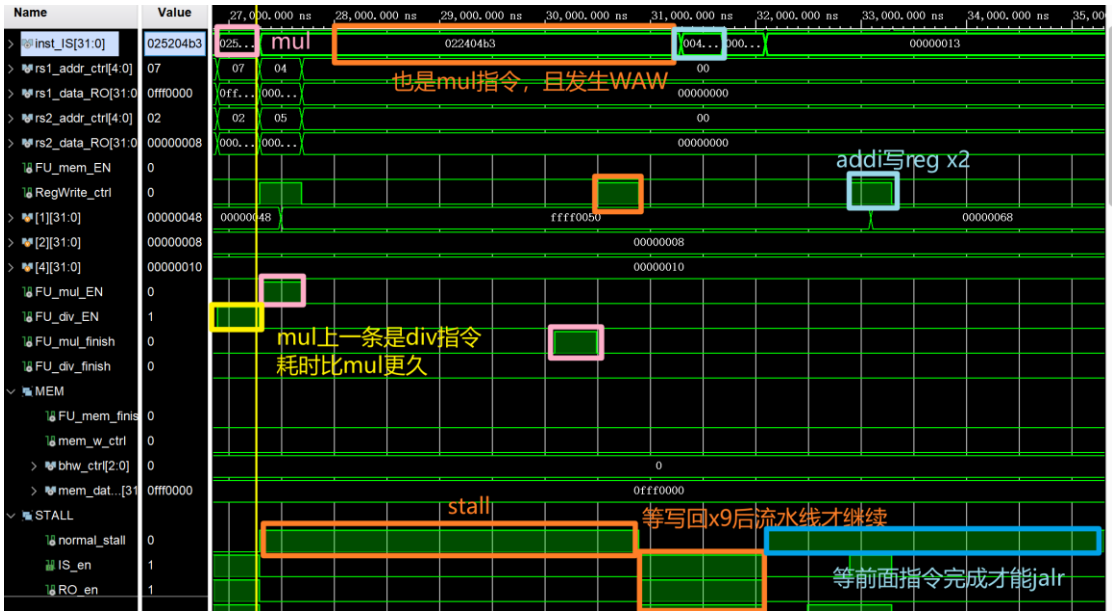
接下来几条指令类似，不做赘述。

在 0x28 处 beq，发生跳转：

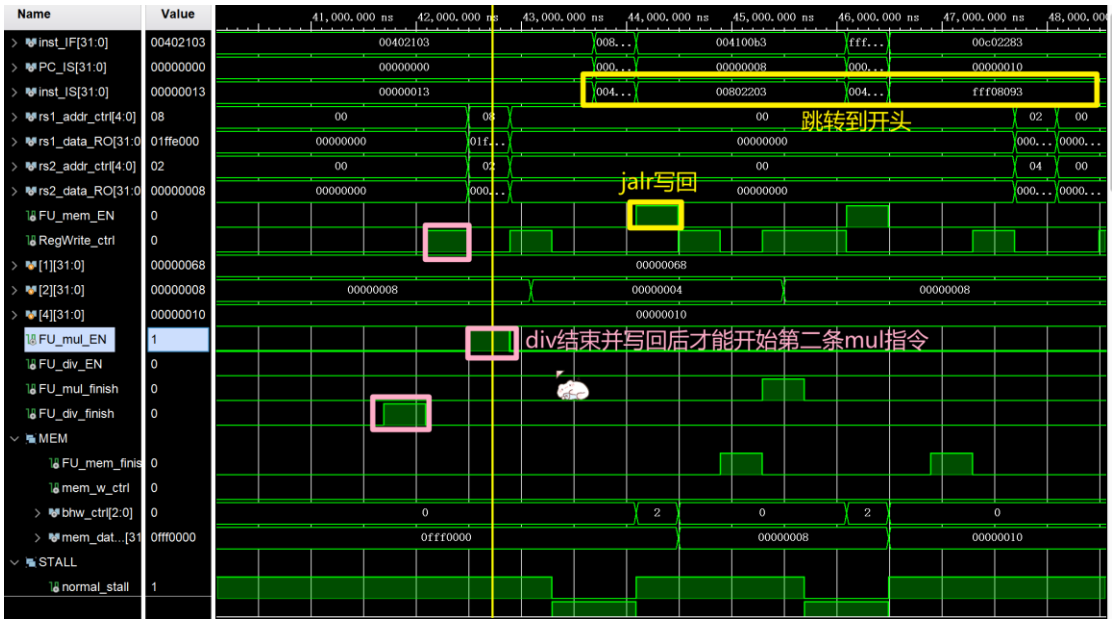


0223c433	54		div x8, x7, x2	
025204b3	58		mul x9, x4, x5	St. Ha./RAW/WAW
022404b3	5C		mul x9, x8, x2	WAR
00400113	60		addi x2, x0, 4	
000000e7	64		jalr x1,0(x0)	

随后是 div 和 mul 指令部分，发生 **WAW**，**RAW** 等：

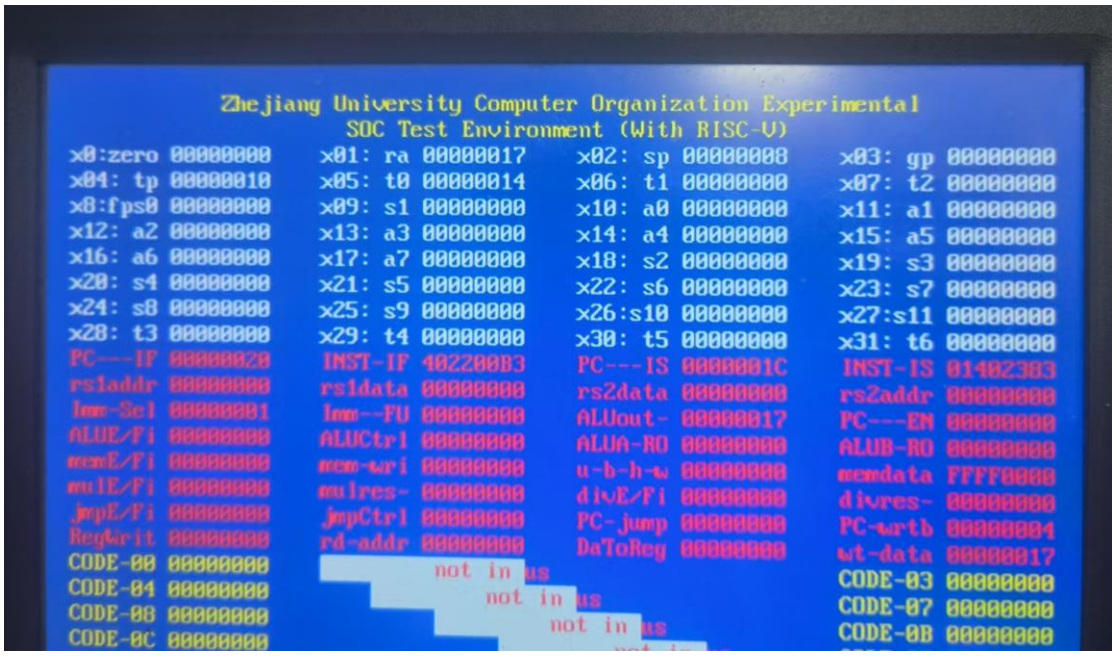


一直到 div 指令结束后，第二条 mul 指令才可以开始取操作数并执行，等待 mul 执行结束后才可以开始 addi 指令（因为有 **WAR**），最后跳转到指令开头，进行循环。此时不会立刻执行开头的 addi 指令，因为和上述 addi 指令存在 **structure hazard**，需要等到它写回后，流水线才继续：



3-2 上板

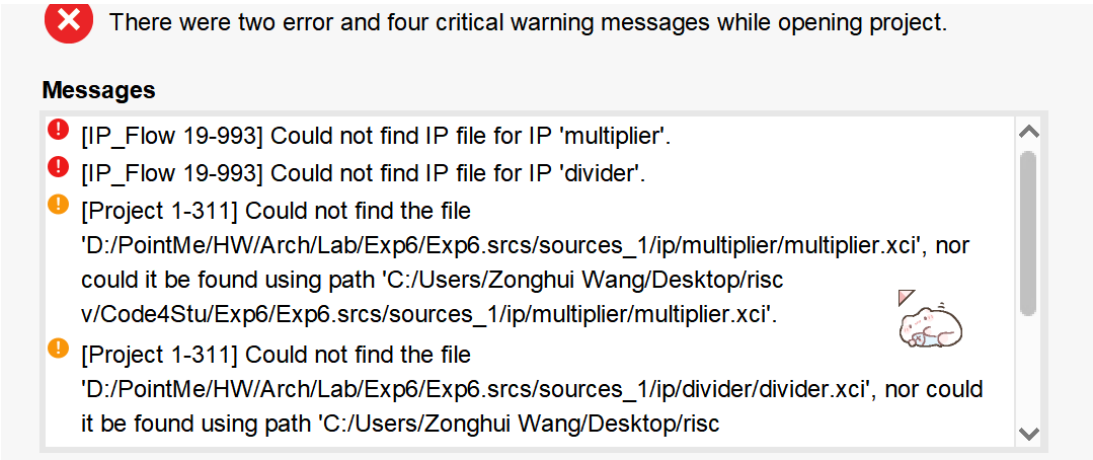
由于本实验涉及到较长时间的 stall，上板结果较难用静态图片呈现，且线下已经验收过上板结果，这里只给出一张上板图作为辅证。



四、讨论、心得

本实验需要补充的代码较多为重复性结构，写起来没有那么困难，就是要仔细检查防止微小处出问题。相比起来，理清仿真结果是一项更有挑战性的任务。实验中也遇到过一些小问题，现列举如下：

打开工程的时候显示 IP 核有问题（下图），由于本实验使用的核与上一个实验一致，只需要根据报错中提示的路径将 lab5 的 ip 核复制到对应文件夹。



另外就是宏定义前面要记得加上`，否则后续一个个补很麻烦（也可以用[`RDY 查找替换[RDY]）；以及复制代码再更改时要注意检查是不是要改的地方全部改完了：

```
RRS[FUS[`FU_ALU][`DST_H:`DST_L]] <= 3'b0;

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

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

```
// EX
FUS[`FU_ALU][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_ALU][`FU_DONE];   //fill sth. here
FUS[`FU_MEM][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_MEM][`FU_DONE];                    //fill sth. here
FUS[`FU_MUL][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_MUL][`FU_DONE];
FUS[`FU_DIV][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_DIV][`FU_DONE];
FUS[`FU_JUMP][`FU_DONE] <= ALU_done ? 1 : FUS[`FU_JUMP][`FU_DONE];
// WB
```