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：Topic 5. Pipelined CPU supporting multi-cycle operations

指导教师：何水兵　　完成时间：2023.12.5

姓名	詹含蓓	**学号**	3210106333
同组学生姓名	居圣桐	**学号**	3210105812
分工信息	共同完成		

一、 实验目的和要求

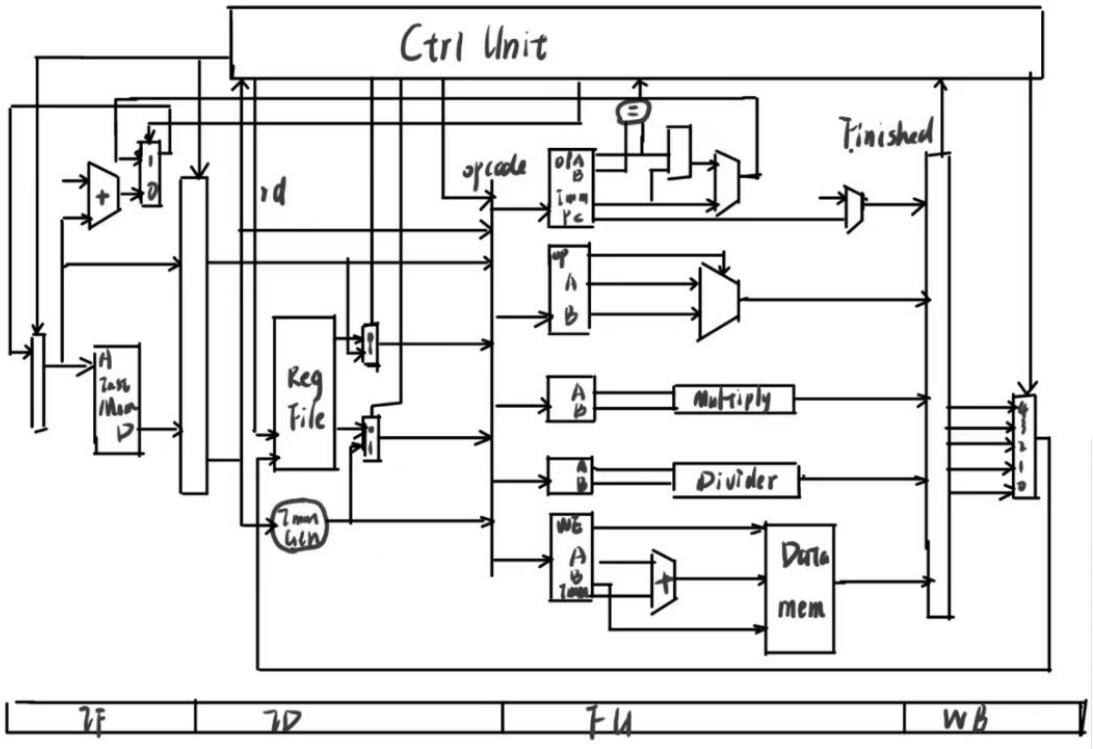
- Understand the principle of pipelines that support multicycle operations.
- Master the design methods of pipelines that support multicycle operations.
- Master verification methods of pipelined CPU supporting multicycle operations.

要求

- Redesign the pipelines with IF/ID/FU/WB stages and FU stage supporting multicycle operations.
- Redesign of CPU Controller.
- Verify the Pipelined CPU with program and observe the execution of program.

二、 实验内容和原理

1. 更新数据通路。相比于普通的五级流水线 CPU，IF、ID 阶段的功能不变，EXE 与 MEM 合二为一变成了 FU，在此阶段进行 ALU 运算、乘除法器、访问 mem 和跳转五项操作，WB 阶段通过多路选择器选择结果。



2. 不同的指令执行的周期数量可能不同，因此实现的 FU 周期数也不同。

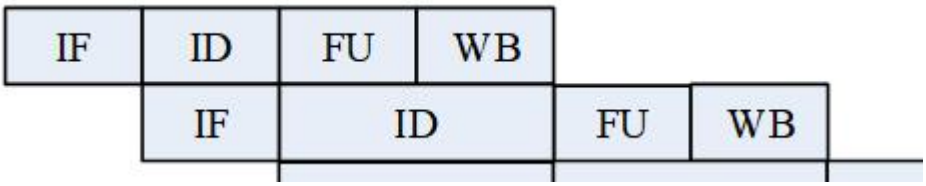
3. 由于本实验没有实现对 forward 进行处理，因此采用的方法是让下一条指令的 ID 等待上一条指令的 WB 执行结束之后在进行 FU，因此：

下一条指令的 ID 的周期数=上一条指令 FU 周期数+上一条指令 WB 周期数

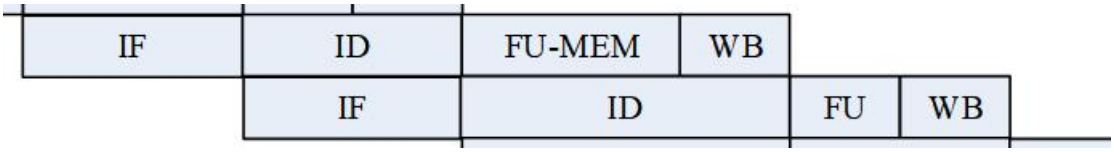
4. 下面对具体指令的 ID 周期数进行分析：

(1) 第一条指令不用等待上一条指令的 WB，因此 FU 是一个周期。第二条指令开始 ID 至少为两个周期

(2) ALU 指令等的下一条指令的 ID 执行周期就是两个

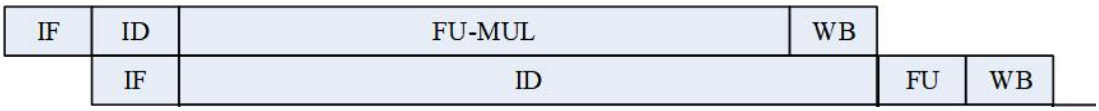


(3) MEM 操作指令（LD 和 ST 指令）的下一条指令的 ID 执行周期就是三个，即 MEM FU 的周期为 2 个，总的为 2+1=3



(4) MUL 操作的的下一条指令的 ID 执行周期就是 8 个，即 MULFU 的周期

为 7 个，总的为 7+1=8



三、 实验过程和数据记录及结果分析

（一）实验过程

1. ALU 单元：

① FU 执行一个周期，state 只有 1bit

② 如果 state=0 则说明 ALU FU 可用，执行计算操作。同时把 state 置于 1 表示该单元被占用

③ 如果 state=1 则说明 ALU FU 被占用，因为只需要执行一个周期所以可以直接置于 0，表示下一个周期可以用。

```
always@(posedge clk) begin
    if(EN & ~state) begin // state == 0
        A <= ALUA;
        B <= ALUB;
        Control <= ALUControl;          //to fill sth.in
        state <= 1;
    end
    else state <= 0;
end
```

2. 乘法单元：

1) 乘法需要执行七个周期的运算，因此 state 需要七位。最开始将最高位置于 1，再将 state 不断右移，判断当第一位为 1 的时候 finish，即乘法运算完成。

2) 如果 state=0 则说明 MUL FU 可用，执行计算操作。同时把 state 的最高位置于 1 表示该单元被占用

3) 如果 state 不为 0 则说明 MUL FU 被占用，将 state 执行右移操作进行周期计数。

```
reg[31:0] A_reg, B_reg;

always@(posedge clk) begin
    if(EN & ~|state) begin // state == 0
        A_reg <= A;
        B_reg <= B;
        state[6] <= 1'b1;
    end
    else
        state <= {1'b0, state[6:1]};
end
    //to fill sth.in
wire [31:0] mulres;
multiplier mul(.CLK(clk),.A(A_reg),.B(B_reg),.P(mulres));
```

3. 除法单元：

② 除法器按照代码要求 state 为一位，因此只需要进行一次判断。但是同时需要加入结果有效性的判断，等待除法器模块输出的结果为有效的时候才能够进行状态的转变和结果的输出。

```
always@(posedge clk) begin
    if(EN & ~state & ~res_valid) begin // state == 0
        A_reg <= A;
        B_reg <= B;
        A_valid <= 1'b1;
        B_valid <= 1'b1;
        state <= 1'b1;
    end
    else if(res_valid) begin
        A_valid <= 1'b0;
        B_valid <= 1'b0;
        state <= 1'b0;
    end
end
```

4. MEM 单元：

③ MEM FU 执行两个周期，state 有 2 bit

④ 如果 state=0 则说明 MEM FU 可用，执行内存访问操作。同时把 state 的最高位置于 1 表示该单元被占用

⑤ 如果 state 不为 0 则说明 MEM FU 被占用，则将 state 进行右移进行周期

计数

```
    always@(posedge clk) begin
        if(EN & ~|state) begin // state == 0
            rs1_data_reg <= rs1_data;
            rs2_data_reg <= rs2_data;
            imm_reg <= imm;
            mem_w_reg <= mem_w;
            bhw_reg <= bhw;
            addr <= rs1_data + imm;
            state[1] <= 1'b1;
        end
        else
            state <= {1'b0, state[1]};
    end
        //to fill sth.in
```

（3）Jump 单元：

① Jump 单元的 FU 执行一个周期，state 有 1bit

② 如果 state=0 则说明 Jump FU 可用，将跳转单元需要的信息赋值给 reg 信号。同时把 state 置于 1 表示该单元被占用。

③ 如果 state=1 则说明 ALU FU 被占用，因为只需要执行一个周期所以可以直接置于 0，表示下一个周期可以用。

④ 执行 cmp 模块来对寄存器的值进行比较，判断能否进行跳转，并且根据结果选择是 PC+4 还是跳转地址，输出下一条指令的 PC 值。

```
reg JALR_reg;
reg[2:0] cmp_ctrl_reg;
reg[31:0] rs1_data_reg, rs2_data_reg, imm_reg, PC_reg;

always@(posedge clk) begin
    if(EN & ~state) begin // state == 0
        rs1_data_reg <= rs1_data;
        rs2_data_reg <= rs2_data;
        imm_reg <= imm;
        PC_reg <= PC;
        cmp_ctrl_reg <= cmp_ctrl;
        JALR_reg <= JALR;
        state <= 1;
    end
    else state <= 0;
end

cmp_32 cmp(.a(rs1_data_reg), .b(rs2_data_reg), .ctrl(cmp_ctrl_reg), .c(cmp_res));
assign PC_jump = JALR_reg ? (rs1_data_reg + imm_reg) : (PC_reg + imm_reg);
assign PC_wb = PC_reg + 32'd4;
        //to fill sth.in
```

3. 更新处理器控制模块，在 FU 阶段可执行多周期的操作，在执行多周期操作时 FU 之外的部分会被 stall。

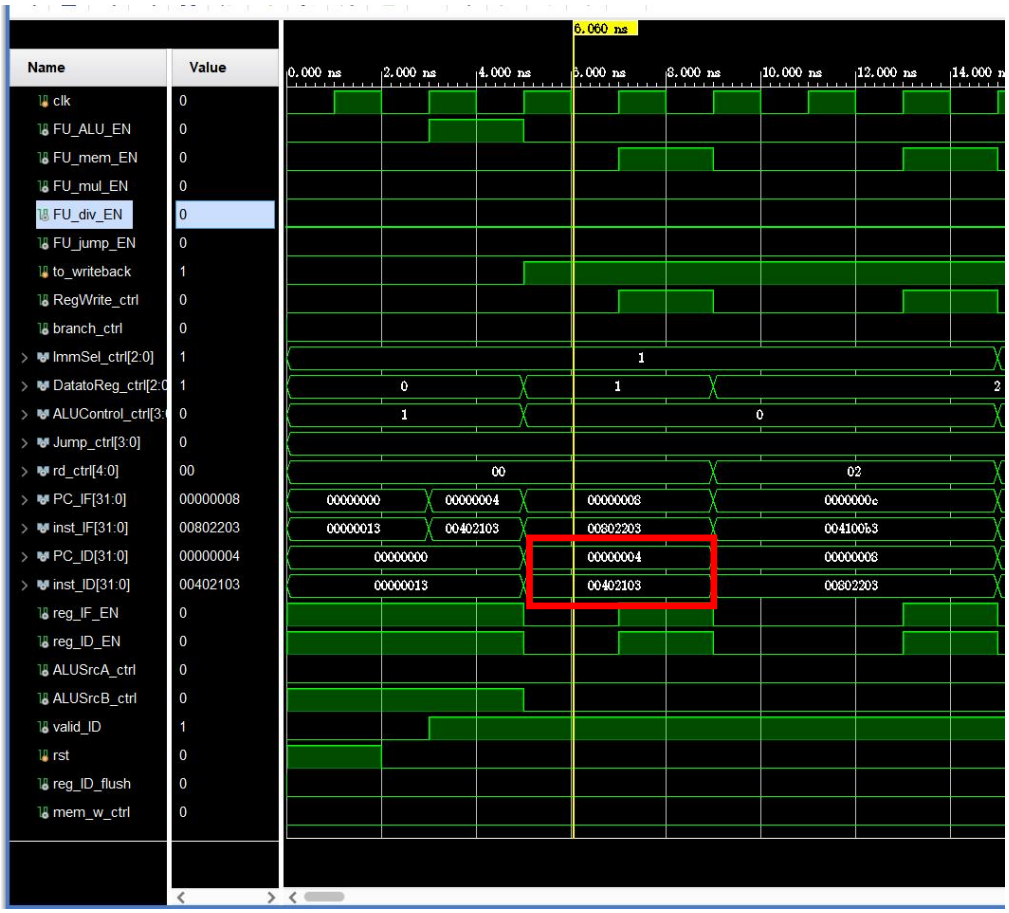
```
ImmGen imm_gen(.ImmSel(ImmSel_ctrl), .inst_field(inst_ID), .Imm_out(Imm_out_ID));          //to fill sth.in

Regs register(.clk(debug_clk),.rst(rst),
    .R_addr_A(inst_ID[19:15]),.rdata_A(rs1_data_ID),
    .R_addr_B(inst_ID[24:20]),.rdata_B(rs2_data_ID),
    .L_S(RegWrite_ctrl),.Wt_addr(rd_ctrl),.Wt_data(wt_data_WB),
    .Debug_addr(debug_addr[4:0]),.Debug_regs(debug_regs));

MUX2T1_32 mux_imm_ALU_ID_A(.I0(rs1_data_ID), .I1(PC_ID), .s(ALUSrcA_ctrl), .o(ALUA_ID));          //to fill sth.in

MUX2T1_32 mux_imm_ALU_ID_B(.I0(rs2_data_ID), .I1(Imm_out_ID), .s(ALUSrcB_ctrl), .o(ALUB_ID));          //to fill sth.in
```

```
MUX8T1_32 mux_DtR(.s(DatatoReg_ctrl),.I0(32'd0),.I1(ALUout_WB),.I2(mem_data_WB),.I3(mulres_WB),
    .I4(divres_WB),.I5(PC_wb_WB),.I6(32'd0),.I7(32'd0),.o(wt_data_WB));          //to fill sth.in
```

（二）实验验证

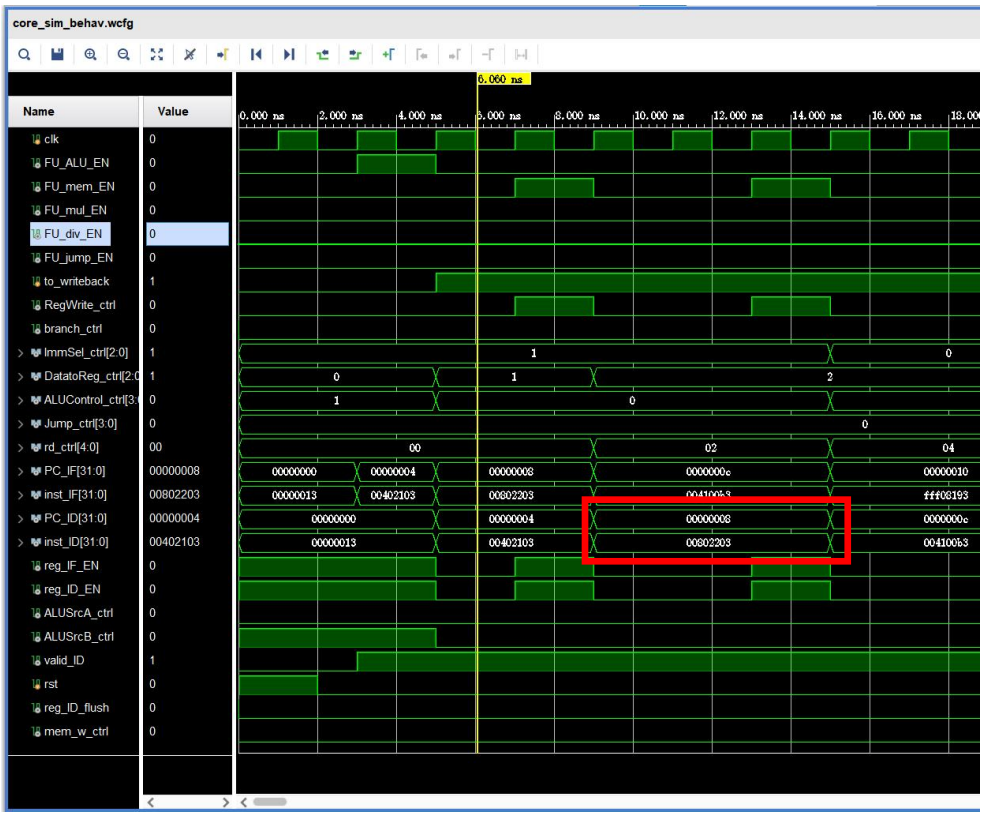
一、仿真验证

1. addi 指令 ID 执行两个周期



2. LW 指令的后一条指令的的 ID 执行 3 个周期，即 LW 的 FU 需要 2 个周期，

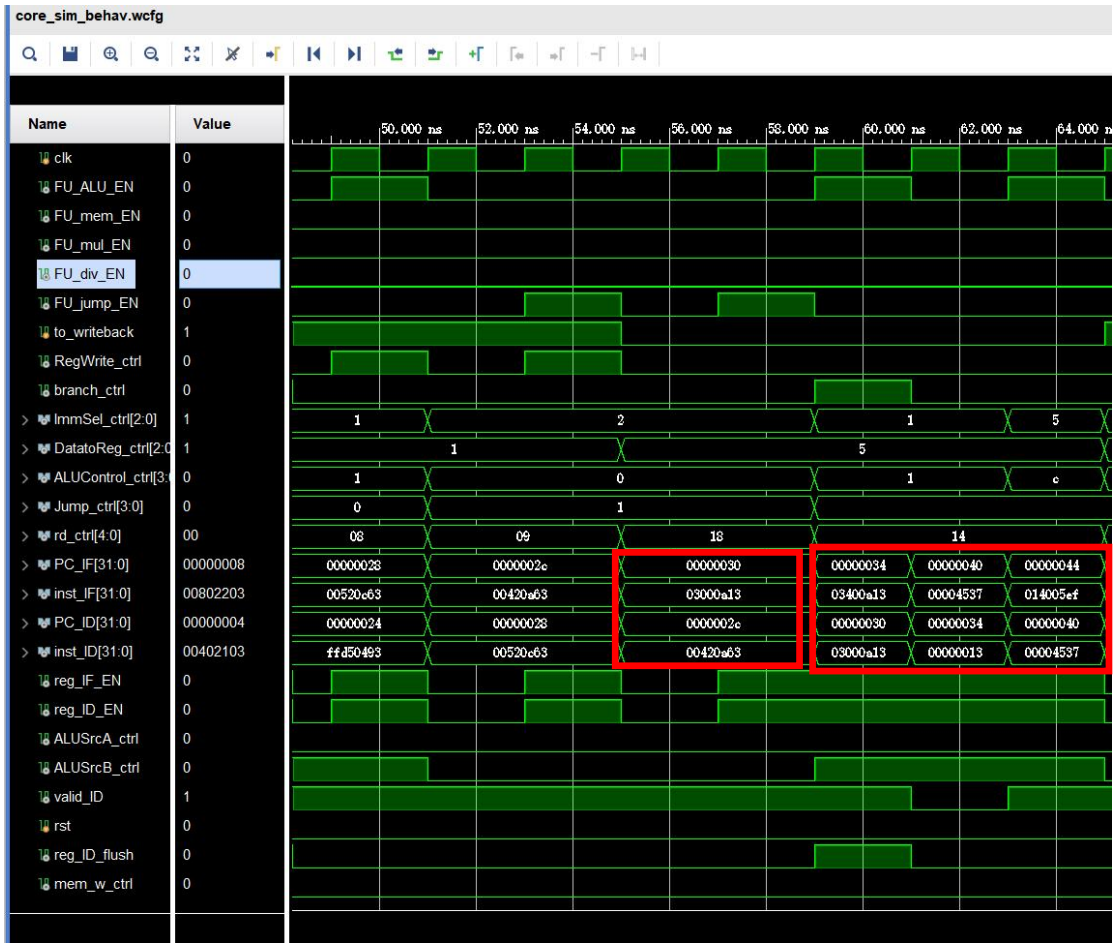
加上 WB 周期为三个



3. Jump 指令第一个指令条件不成立，不跳转，第二条指令跳转条件成立，跳转

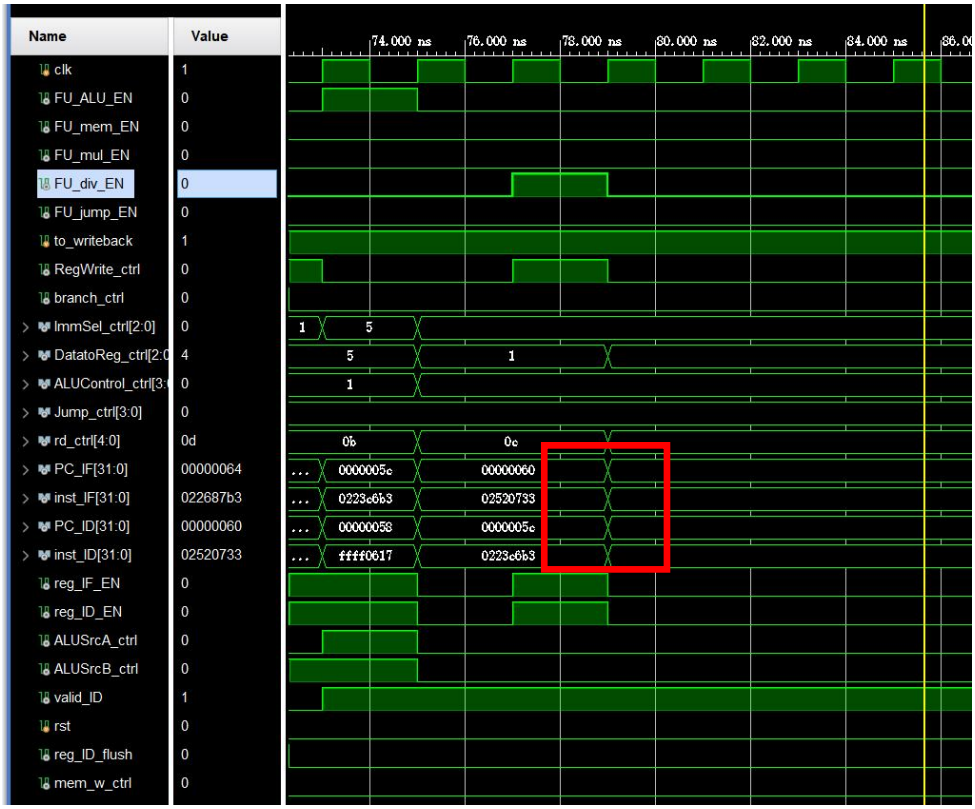
(1) 第一条跳转指令由于等待之前的 WB，ID 需要将进行两个周期，其本身 FU 是一个周期。

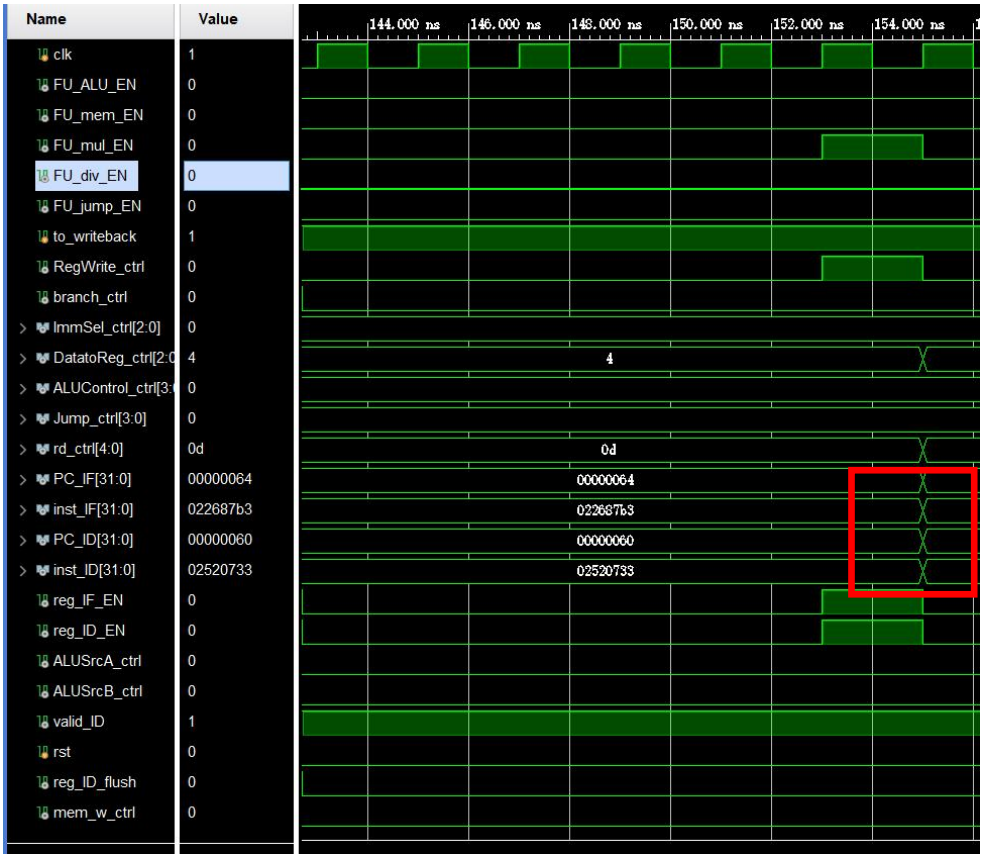
(2) 第二条跳转指令在进行 IF 先进入了后面的指令后，在 ID 阶段将先进入的指令 flush，PC 修改为跳转后的指令。



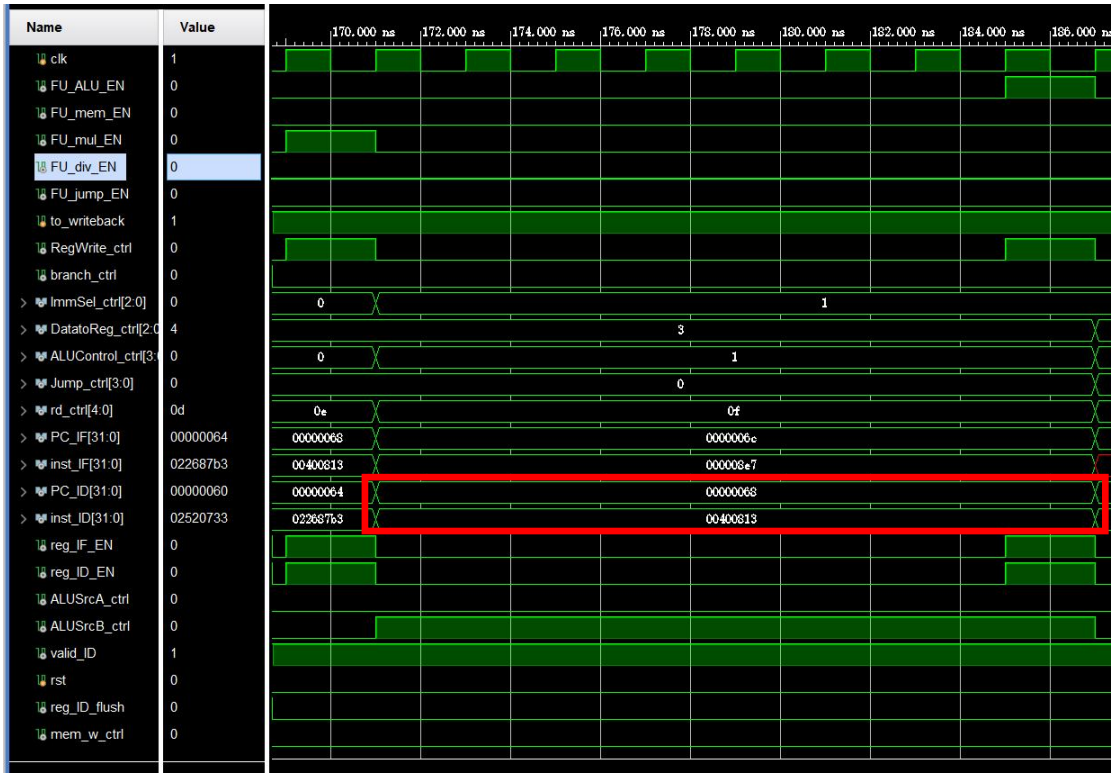
4. 除法指令的 FU 需要等待有效的结果值，因此其后面的指令的 ID 最终需要 38 个周期。

(1) 起始周期为 79ns，结束为 155ns，因此一共有（155-79）/2=38 个周期，符合要求

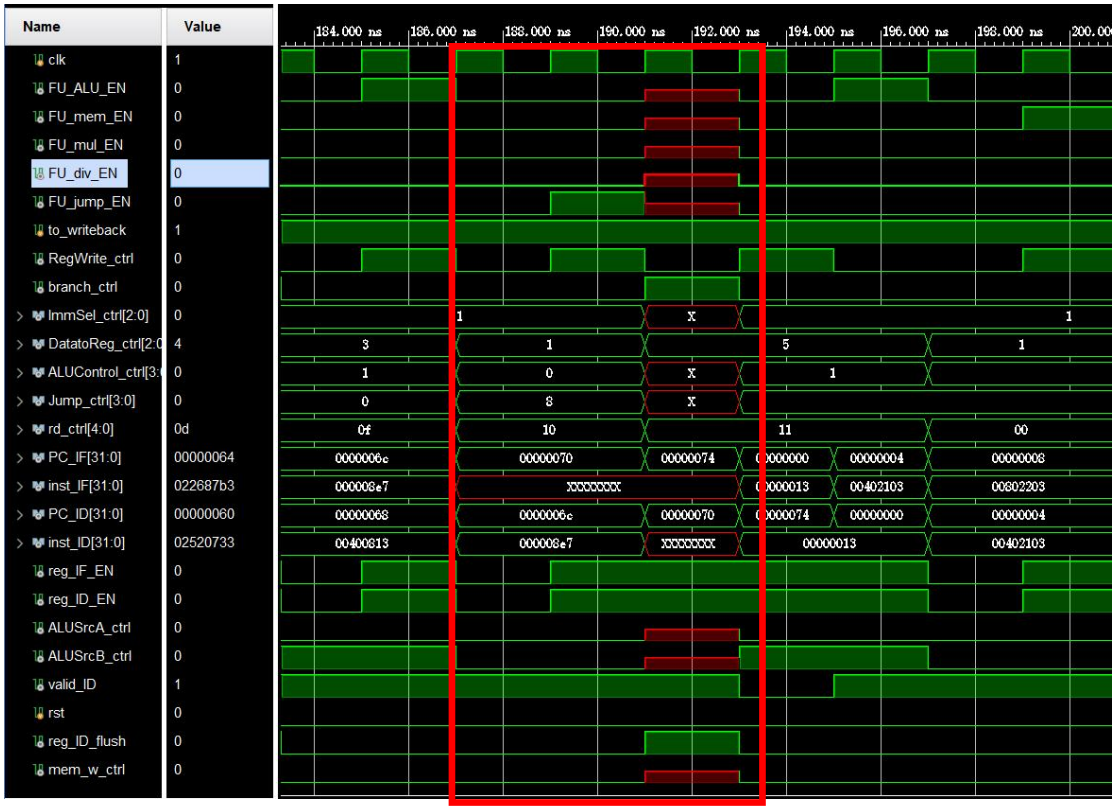




5. 乘法指令的 FU 阶段要 7 个周期，因此下一条指令的 ID 需要的周期数为 7+1=8 个周期



6. 最后的跳转指令由于后面没有指令输入所以在 flush 之前会出现高阻态



二、实验板验证

在仿真当中以及对代码和相应的周期数进行了很详尽的分析，因此实验板的照片就不做详细分析，主要为证明实验上板成功。


Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
POWER VOL- VOL+ MENU CH- CH+ MODE


Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
POWER VOL- VOL+ MENU CH- CH+ MODE


Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
POWER VOL- VOL+ MENU CH- CH+ MODE

四、　讨论与心得

本次实验实现了一个比较简单粗糙的多周期指令执行流水线，没有对冲突等进行处理。由于最开始的时候没有完全理解实验在做什么，一直以为是实现 scoreboard，ppt 里面的讲述也比较少，因此刚开始没有理清楚。后面理解了实验的内容之后就发现自己想的太复杂了。