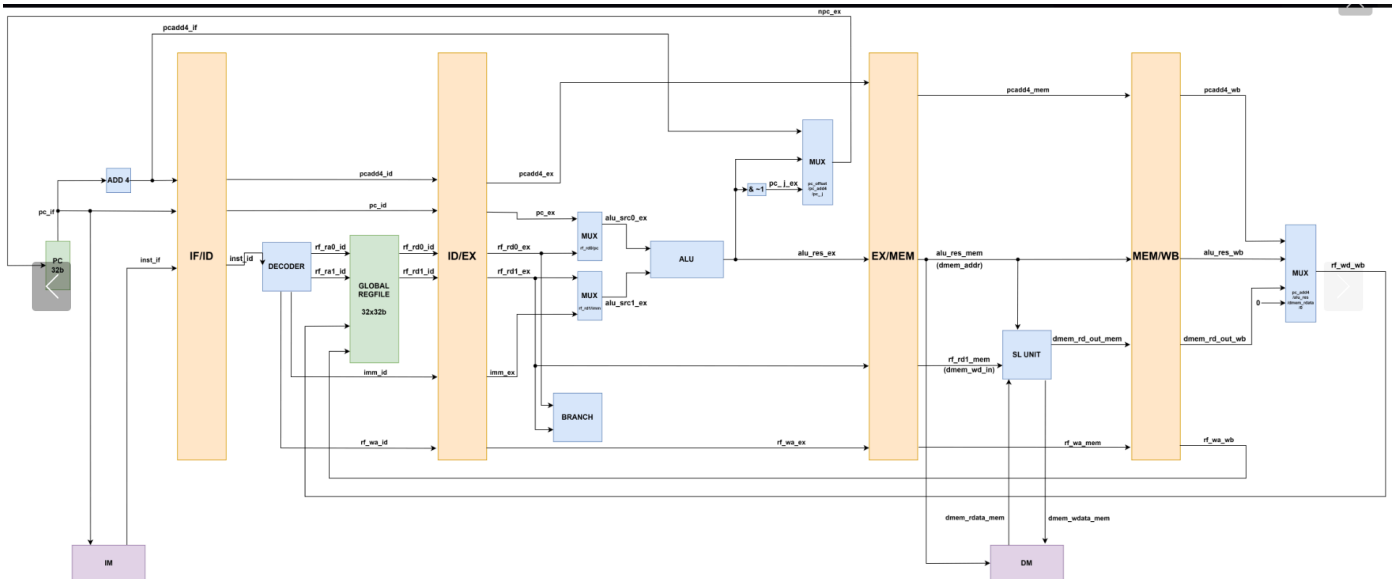
Lab 5 report

实验目的与内容

1. **简述本次实验做的内容以及本次实验的目的、逻辑设计**
   - 通过 Verilog 硬件描述语言实现一个可用的简单流水线CPU。实验五进行简单流水线CPU基本功能的搭建。
   - 此次实验在实验四的基础上进行修改，实现简单流水线CPU的设计，包括实现段间寄存器、以及 `REG_FILE` 模块**写优先**的修改等，最后需要在龙芯框架中运行仿真测试并上板验证结果。
2. 请参照PPT画出你设计的各模块的框图和数据通路，



3. 如果存在状态机，请绘制出状态机的状态转换图；

   > 无。

4. 请贴出较为核心的代码设计代码，并加以解释说明。
   见第二部分。

具体实验任务

段间寄存器

任务：实现流水线的段间存储需求。我采用了实验文档的推荐方法：**使用统一的段间寄存器模块**，让信号一经产生就传递到最后，从而简化了段间寄存器的设计。 对于在上一阶段尚未产生的信号，直接在输入处接 0，忽略其输出，即只保留有效端口的使用。

> 当然，这样也就会导致我们的段间寄存器十分庞大（去年的实验中，它共有 73 个输入输出端口）。
> 在本次实验中，笔者总共用了五十多个端口。

- 在代码中，对 `flush` 信号做出 `commit` 信号的响应，当 `flush` 信号为高电平时，意味着出现分支错误，我们需要清除前面产生的信号。因此 `flush` 信号笔者设置在 `NPC_MUX` 模块中产生。

```
module  Inter_stage_Regs(
    //common port
    input       [  0 : 0]       clk     ,
    input       [  0 : 0]       rst     ,
    input       [  0 : 0]       en      ,
    input       [  0 : 0]       flush   ,
    input       [  0 : 0]       commit  ,//5

    //generated from IF/ID
    input       [ 31 : 0]       pcadd4  ,
    input       [ 31 : 0]       pc      ,
    input       [ 31 : 0]       inst    ,//3

    //generated from ID/EX
    input       [  4 : 0]       alu_op  ,
    input       [ 31 : 0]       rf_rd0  ,
    input       [ 31 : 0]       rf_rd1  ,//yet in EX/MEM
    input       [ 31 : 0]       imm     ,
    input       [  4 : 0]       rf_wa   ,//
    input       [  0 : 0]       rf_we   ,
    input       [  0 : 0]       dmem_we ,
    input       [  4 : 0]       rf_ra0  ,
    input       [  4 : 0]       rf_ra1  ,

    input       [  3 : 0]       br_type ,

    input       [  3 : 0]       dmem_access,
    input       [  1 : 0]       rf_wd_sel   ,
    input       [  0 : 0]       alu_src0_sel,
    input       [  0 : 0]       alu_src1_sel,//14

    //generated from EX/MEM
    input       [ 31 : 0]       alu_res ,
    input       [  1 : 0]       npc_sel ,
    input       [ 31 : 0]       alu_src0,
    input       [ 31 : 0]       alu_src1,

    //generated from MEM/WB
    input       [ 31 : 0]       dmem_rd_out,
    input       [ 31 : 0]       rf_wd,//6
//28

    //output port
```

```
    output reg  [  0 : 0]       commit_out,
    //IF/ID needed

    output reg  [ 31 : 0]       pcadd4_out,
    output reg  [ 31 : 0]       pc_out,
    output reg  [ 31 : 0]       inst_out,//4

    //ID/EX needed
    output reg  [  4 : 0]       alu_op_out,
    output reg  [ 31 : 0]       rf_rd0_out,
    output reg  [ 31 : 0]       rf_rd1_out,
    output reg  [ 31 : 0]       imm_out,
    output reg  [  4 : 0]       rf_wa_out,
    output reg  [  0 : 0]       rf_we_out,
    output reg  [  0 : 0]       dmem_we_out,

    output reg  [  4 : 0]       rf_ra0_out,
    output reg  [  4 : 0]       rf_ra1_out,

    output reg  [  3 : 0]       br_type_out,

    output reg  [  3 : 0]       dmem_access_out,
    output reg  [  1 : 0]       rf_wd_sel_out,
    output reg  [  0 : 0]       alu_src0_sel_out,
    output reg  [  0 : 0]       alu_src1_sel_out,//14


    //EX/MEM needed
    output reg  [ 31 : 0]       alu_res_out,
    output reg  [  1 : 0]       npc_sel_out,
    output reg  [ 31 : 0]       alu_src0_out,
    output reg  [ 31 : 0]       alu_src1_out,
    output reg  [ 31 : 0]       dmem_addr_out,

    //MEM/WB needed
    output  reg [ 31 : 0]       dmem_rd_out_out,
    output  reg [ 31 : 0]       rf_wd_out//7

//25
);

wire        [  0 : 0 ]    t_commit;
wire        [ 31 : 0 ]    t_pcadd4;
```

```
wire        [ 31 : 0 ]    t_pc;
wire        [ 31 : 0 ]    t_inst;
wire        [ 31 : 0 ]    t_rf_rd0;
wire        [ 31 : 0 ]    t_rf_rd1;
wire        [ 31 : 0 ]    t_imm;
wire        [  4 : 0 ]    t_rf_wa;
wire        [  0 : 0 ]    t_rf_we;
wire        [  0 : 0 ]    t_dmem_we;
wire        [ 31 : 0 ]    t_alu_res;
wire        [ 31 : 0 ]    t_dmem_rd_out;

assign      t_commit      = commit;
assign      t_pcadd4      = pcadd4;
assign      t_pc          = pc;
assign      t_inst        = inst;
assign      t_rf_rd0      = rf_rd0;
assign      t_rf_rd1      = rf_rd1;
assign      t_imm         = imm;
assign      t_rf_wa       = rf_wa;
assign      t_rf_we       = rf_we;
assign      t_dmem_we     = dmem_we;
assign      t_alu_res     = alu_res;
assign      t_dmem_rd_out = dmem_rd_out;


always @(posedge clk or posedge rst) begin
    if (rst) begin
        pcadd4_out      <= 32'b0;
        pc_out          <= 32'b0;
        inst_out        <= 32'b0;
        rf_rd0_out      <= 32'b0;
        rf_rd1_out      <= 32'b0;
        imm_out         <= 32'b0;
        rf_wa_out       <= 5'b0;
        rf_we_out       <= 1'b0;
        dmem_we_out     <= 1'b0;
        alu_res_out     <= 32'b0;
        dmem_addr_out   <= 32'b0;
        dmem_rd_out_out <= 32'b0;

        rf_ra0_out      <= 5'b0;
        rf_ra1_out      <= 5'b0;
        br_type_out     <= 4'b0;
```

```
        dmem_access_out <= 4'b0;
        alu_src0_sel_out <= 1'b0;
        alu_src1_sel_out <= 1'b0;

        alu_src0_out    <= 32'b0;
        alu_src1_out    <= 32'b0;
        rf_wd_sel_out   <= 2'b0;
        alu_op_out      <= 5'b0;
        npc_sel_out     <= 2'b0;

        rf_wd_out       <= 32'b0;
        commit_out      <= 1'b0;
    end
    else if (en) begin
        if (flush) begin
            pcadd4_out    <= t_pcadd4;
            pc_out        <= t_pc;
            inst_out      <= t_inst;
            rf_rd0_out    <= t_rf_rd0;
            rf_rd1_out    <= t_rf_rd1;
            imm_out       <= t_imm;
            rf_wa_out     <= t_rf_wa;
            rf_we_out     <= t_rf_we;
            dmem_we_out   <= t_dmem_we;
            alu_res_out   <= t_alu_res;
            dmem_addr_out <= t_alu_res; // 注意这里有待确认，是否应该是 t_dmem_addr
            dmem_rd_out_out <= t_dmem_rd_out;
            rf_ra0_out      <= rf_ra0;
            rf_ra1_out      <= rf_ra1;
            br_type_out     <= br_type;
            dmem_access_out <= dmem_access;
            alu_src0_sel_out <= alu_src0_sel;
            alu_src1_sel_out <= alu_src1_sel;

            alu_src0_out <= alu_src0;
            alu_src1_out <= alu_src1;
            rf_wd_sel_out <= rf_wd_sel;
            alu_op_out <= alu_op;
            npc_sel_out <= npc_sel;

            rf_wd_out       <= rf_wd;
            commit_out <= 1'b0; // 如果 flush 为 1，则 commit 为 0
        end
```

```
        else begin
            pcadd4_out    <= t_pcadd4;
            pc_out        <= t_pc;
            inst_out      <= t_inst;
            rf_rd0_out    <= t_rf_rd0;
            rf_rd1_out    <= t_rf_rd1;
            imm_out       <= t_imm;
            rf_wa_out     <= t_rf_wa;
            rf_we_out     <= t_rf_we;
            dmem_we_out   <= t_dmem_we;
            alu_res_out   <= t_alu_res;
            dmem_addr_out <= t_alu_res; // 注意这里有待确认，是否应该是 t_dmem_addr
            dmem_rd_out_out <= t_dmem_rd_out;

            rf_ra0_out      <= rf_ra0;
            rf_ra1_out      <= rf_ra1;
            br_type_out     <= br_type;
            dmem_access_out <= dmem_access;
            alu_src0_sel_out <= alu_src0_sel;
            alu_src1_sel_out <= alu_src1_sel;
            rf_wd_out       <= rf_wd;
            alu_src1_out <= alu_src1;
            rf_wd_sel_out <= rf_wd_sel;
            alu_op_out <= alu_op;
            npc_sel_out <= npc_sel;

            commit_out    <= t_commit; // 如果 flush 为 0，则 commit 为 1
        end
    end
end

endmodule
```

CPU模块相应的修改

任务：因为存在段间寄存器，所以每一阶段产生和传递的信号不再直接接到下一个功能单元，而是作为输入暂存，再从区别于输入的输出端口输出到下一个阶段需要用到该信号的功能单元。

- 以下列出四个段间寄存器的例化情况，他们分别反映了信号生成情况和接线情况。

```
NPC_MUX npc_m(//这块的大部分信号来自于执行阶段生成。
    .pc_add4(cur_npc),
    .pc_offset(alu_res_ex),
    .npc_sel(npc_sel_ex),
    .flush(flush),
    .npc(npc_ex)
);
```

- 第一个段间寄存器暂时放出所有的接口例化情况。后面只展示关键例化。

```
Inter_stage_Regs IF_ID(
    //common port
    .clk(clk),
    .rst(rst),
    .en(global_en),
    .flush(flush),
    .commit(commit_if),//一直传到最后

    //generated from IF/ID
    .pcadd4(pcadd4_if),
    .pc(pc_if),
    .inst(inst_if),

    //generated from ID/EX
    .rf_rd0(32'b0),
    .rf_rd1(32'b0), //yet in EX/MEM
    .imm(32'b0),
    .rf_wa(5'b0), //
    .rf_we(1'b0),
    .dmem_we(1'b0),

    .rf_ra0(5'b0),
    .rf_ra1(5'b0),
    .br_type(4'b0),
    .dmem_access(4'b0),
    .alu_src0_sel(1'b0),
    .alu_src1_sel(1'b0),

    //generated from EX/MEM
    .alu_res(32'b0),
    .alu_op(5'b0),
    .rf_wd_sel(2'b0),
    .npc_sel(2'b0),
    .alu_src0(32'b0),
    .alu_src1(32'b0),

    //generated from MEM/WB
    .dmem_rd_out(32'b0),
    .rf_wd(32'b0),


//输出部分
    //output port
```

```
    .commit_out(commit_id),
    //IF/ID needed
    .pcadd4_out(pcadd4_id),
    .pc_out(pc_id),
    .inst_out(inst_id),

    //ID/EX needed
    .rf_rd0_out(),
    .rf_rd1_out(),
    .imm_out(),
    .rf_wa_out(),
    .rf_we_out(),
    .dmem_we_out(),

    .rf_ra0_out(),
    .rf_ra1_out(),
    .br_type_out(),
    .dmem_access_out(),
    .alu_src0_sel_out(),
    .alu_src1_sel_out(),

    //EX/MEM needed
    .alu_res_out(),
    .dmem_addr_out(),

    .alu_op_out(),
    .rf_wd_sel_out(),
    .npc_sel_out(),
    .alu_src0_out(),
    .alu_src1_out(),

    //MEM/WB needed
    .dmem_rd_out_out(),
    .rf_wd_out()

);
```

- 译码器的输出内容全部作为第二个段间寄存器的输入，而不再像单周期那样直接接入运算器和寄存器堆。

```
DECODE deco(
    .inst(inst_id),
    .alu_op(alu_op_id),
    .dmem_access(dmem_access_id),
    .imm(imm_id),
    .rf_ra0(rf_ra0_id),
    .rf_ra1(rf_ra1_id),
    .rf_wa(rf_wa_id),
    .rf_we(rf_we_id),
    .alu_src0_sel(alu_src0_sel_id),
    .alu_src1_sel(alu_src1_sel_id),
    .rf_wd_sel(rf_wd_sel_id),
    .br_type(br_type_id),
    .dmem_we(dmem_we_id)
);
```

- 最值得一提的是寄存器堆，写寄存器的写回的内容需要等待 `WB` 阶段的信号，因此不是把 `rf_wb_id` 接入。

```
REG_FILE reg_f(
    .clk(clk),
    .rf_ra0(rf_ra0_id),
    .rf_ra1(rf_ra1_id),
    .rf_wa(rf_wa_id),//有一些特殊处理，可能会有错不过没关系。
    .rf_we(rf_we_id),//
    .rf_wd(rf_wd_wb),//
    .rf_rd0(rf_rd0_id),
    .rf_rd1(rf_rd1_id),
    .dbg_reg_ra(debug_reg_ra),
    .dbg_reg_rd(debug_reg_rd)
);
```

- 此外其他寄存器连接都是常规连接从段间寄存器中更新的暂存的上一级的信号。以此达到时钟周期控制流水线的效果。

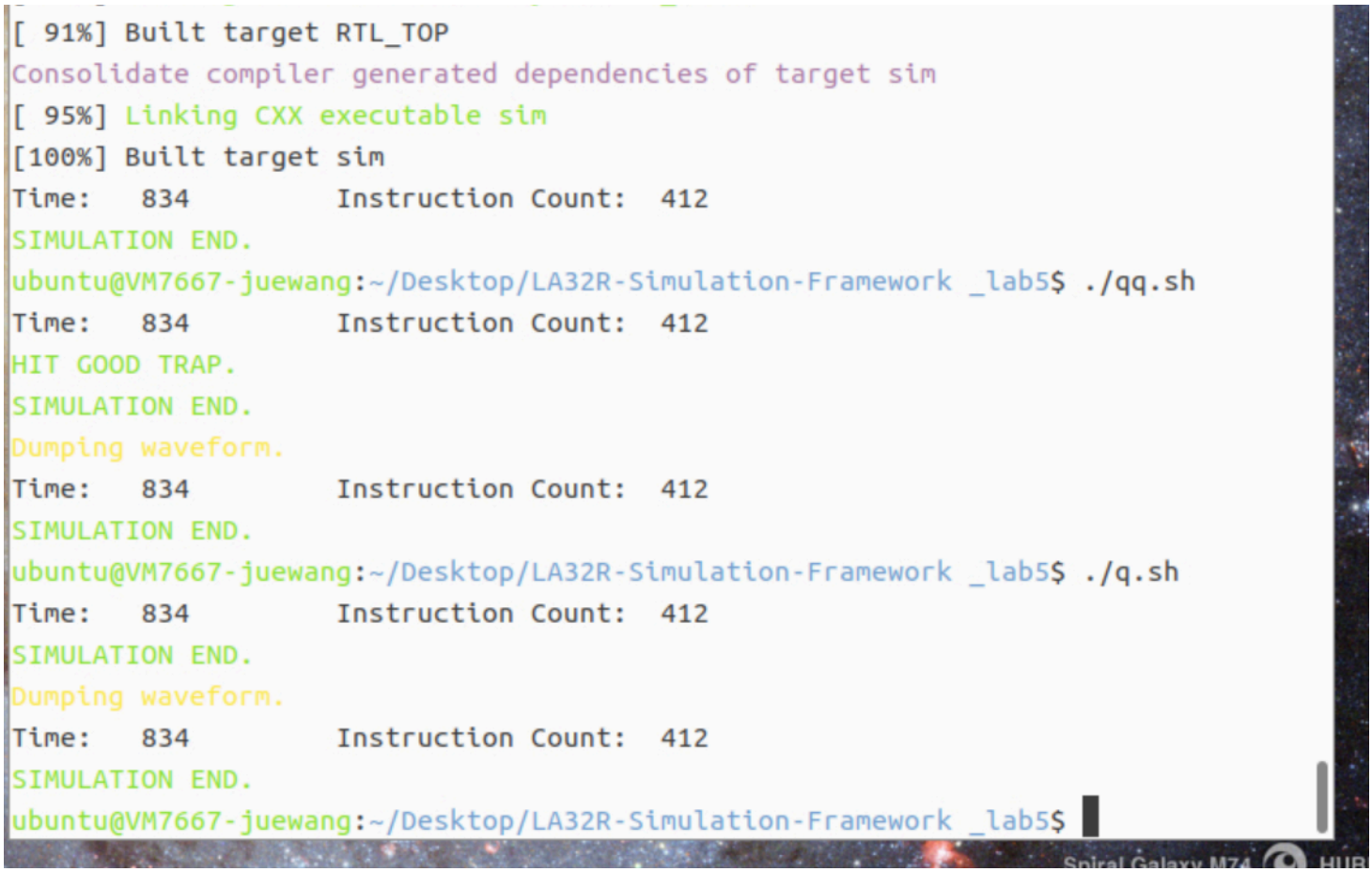
寄存器堆的写优先

任务：读和写同时发生时，先写再读。此处列出改动的部分代码。

- 当要读数据的寄存器和要写入数据的寄存器编号相等时，读数据直接被赋值为写入数据。
- 注意0号寄存器不做处理。

```
always @(*) begin
        rf_rd0 = reg_file[rf_ra0];
        rf_rd1 = reg_file[rf_ra1];
        if(rf_wa != 0 && rf_we)begin
            if(rf_ra0 == rf_wa)begin
                rf_rd0 = rf_wd;
            end
            if(rf_ra1 == rf_wa)begin
                rf_rd1 = rf_wd;
            end
        end
        else if(rf_wa == 0 && rf_we)begin
            if(rf_ra0 == rf_wa)begin
                rf_rd0 = 0;
            end
            if(rf_ra1 == rf_wa)begin
                rf_rd1 = 0;
            end
        end
    end
```

仿真结果



在龙芯仿真框架下进行仿真验证，结果正确符合预期。

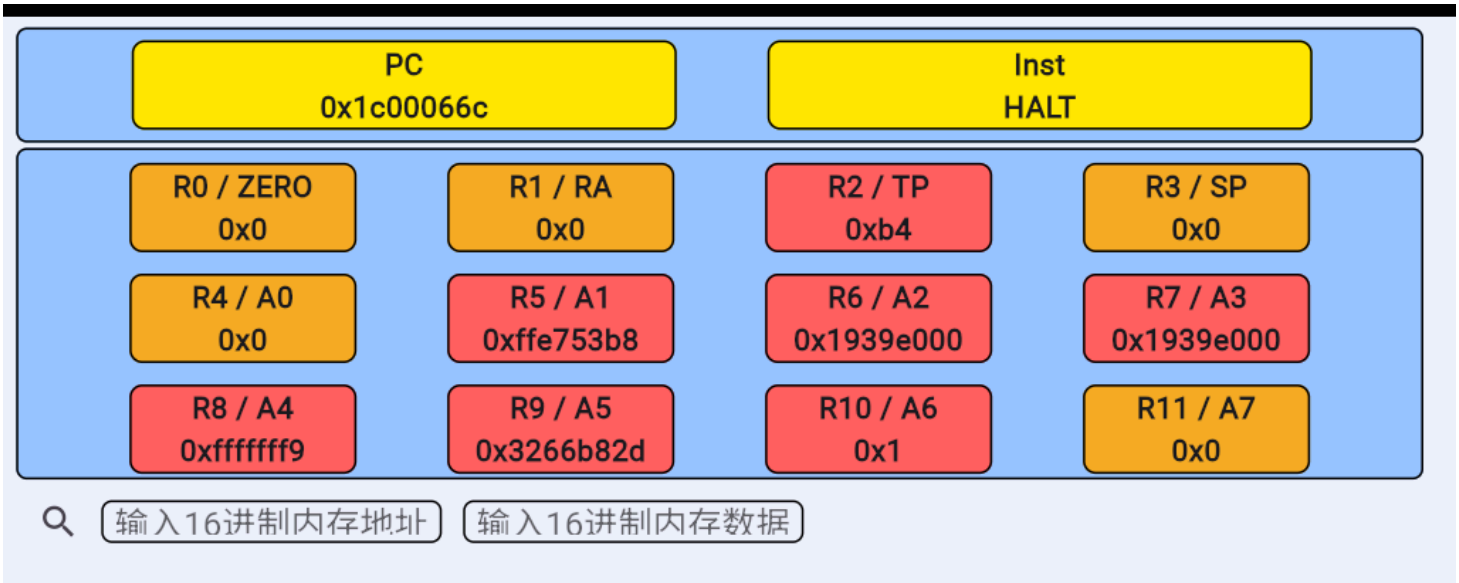
上板结果

```
FPGAOL UART xterm.js 1.1

USTC COD Project
User: R;
---------- CPU ----------


User: RR 0 16;
00000000 00000000 000000B4 00000000
00000000 FFE753B8 1939E000 1939E000
FFFFFFF9 3266B82D 00000001 00000000
B432D000 B432D000 000000B4 00000001


User:
```



上板结果前16个寄存器的值与 LARS 运行结果一致。

总结

1. 如图

- 2. **在实验报告中回答**，对于本次实验中的五级流水线 CPU，连续执行以下的指令序列后，**若寄存器堆没有使用写优先**，运行结束后 x4 与 x5 中的结果是什么？

```
addi x1, x0, 1
addi x2, x0, 2
addi x3, x0, 3
addi x1, x0, 10
addi x2, x0, 20
addi x3, x0, 30
addi x0, x0, 0
add  x4, x1, x2
add  x5, x2, x3
nop
nop
nop
```

- `x4` = 12, `x5` = 23;
- 原因：当 `add x4 , x1 , x2` 在译码的时候，由于 `x2` 的写回操作还没有进行（没有写优先的时候），则读出来的值仍为2，所以结果是 `10+2` ，同理 `x5 = 3 + 20` 。