电子科技大学计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称 计算机系统结构综合实验

电子科技大学教务处制表

实　验　报　告

学生姓名：卢晓雅　　学　号：2020080904026　　指导教师：王华

实验地点：　清水河校区主楼 A2 区　　**实验时间：4.28 下午**

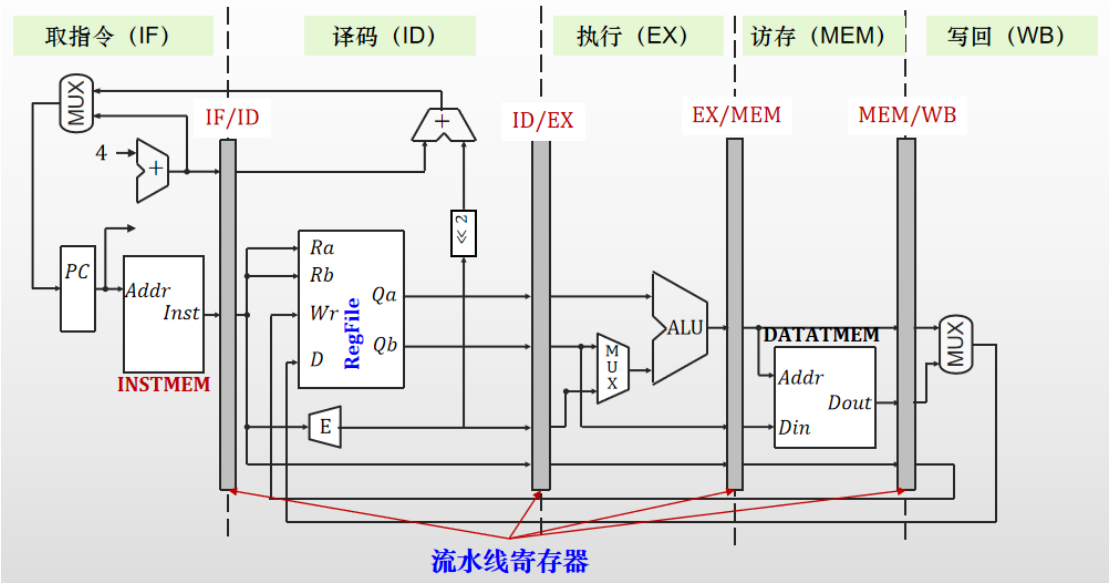
一、实验室名称：国家级计算机实验教学示范中心

二、实验项目名称：五级流水线 CPU 设计

三、实验学时：4

四、实验原理：

一）基本的五级流水线 CPU 模型架构图如下所示：



图中五级分别代表一条指令执行的五个不同的阶段，分别是：

1. IF STAGE：取指令阶段
2. ID STAGE：指令译码阶段
3. EXE STAGE：执行指令阶段
4. MEM STAGE：访存阶段
5. WB STAGE：写回阶段

二）在单周期 CPU 架构基础上实现五级流水线 CPU 架构的改造，重点在于各级之间流水线寄存器的构造。流水线寄存器是为了实现流水线作业而新增的硬件，能隔离和保护不同指令的相关控制信息。

具体来说，每级流水线寄存器之间传递的信号分别如下：

1. IF_ID 级流水线寄存器的信号分析

IF_STAGE stage1 (Clock, Resetn, pcsource, bpc, jpc, pc4, Inst, PC);

- 输出信号有：pc4, Inst, PC
- 内部 wire 有：pc，npc（该组信号仅影响 PC，且当次完成更新，故而不需要传递）

ID_STAGE stage2 (pc4, Inst, wdi, ~Clock, Resetn, bpc, jpc, pcsource,
m2reg, wmem, aluc, aluimm, ra, rb, imm, shift, z);

- 输入信号有：pc4, Inst, wdi, ~Clock, Resetn,z

【结论】IF_ID 级流水线寄存器需要传递的信号有：

pc4, Inst

2. ID_EXE 级流水线寄存器的信号分析

ID_STAGE stage2 (pc4, Inst, wdi, ~Clock, Resetn, bpc, jpc, pcsource, m2reg, wmem, aluc, aluimm, ra, rb, imm, shift, z);

- 输出信号有：bpc, jpc, pcsource, m2reg, wmem, aluc, aluimm, ra, rb, imm, shift（bpc, jpc, pcsource 当次完成控制任务，不需要传递）
- 内部 wire 有：

func, op, wreg, rs,rt,rd,qa,qb,br_offset,ext16,regrt,sext,e,rn（wreg、rn 信号当次不能完成控制任务，因为写入数据还没准备好，所以需要传

递。qa,qb 就是 ra, rb，不需要重复传递）

EXE_STAGE stage3 (aluc, aluimm, ra, rb, imm, shift, Alu_Result, z);

输入信号有：aluc, aluimm, ra, rb, imm, shift

- 内部 wire 有：alua,alub,sa（都在当次处理，不需要传递）

【结论】ID_EXE 级流水线寄存器需要传递的信号有：

m2reg, wmem, aluc, aluimm, ra, rb, imm, shift，wreg，rn

3. EXE_MEM 级流水线寄存器的信号分析

EXE_STAGE stage3 (aluc, aluimm, ra, rb, imm, shift, Alu_Result, z);

- 输出信号有：Alu_Result, z（z 当次输出到 ID_STAGE，不需要传递）
- 内部 wire 有：alua,alub,sa（都在当次完成控制任务，不需要传递）

MEM_STAGE stage4 (wmem, Alu_Result[6:2], rb, ~Clock, mo);

- 输入信号有：wmem, Alu_Result[6:2], rb（不考虑时钟信号）
- 无内部 wire 信号

【结论】EXE_MEM 级流水线寄存器需要传递的信号有：

Alu_Result, rb，wmem，m2reg，wreg，rn

4. MEM_WB 级流水线寄存器的信号分析

MEM_STAGE stage4 (wmem, Alu_Result[6:2], rb, ~Clock, mo);

- 输出信号有：mo
- 无内部 wire 信号

WB_STAGE stage5 (Alu_Result, mo, m2reg, wdi);

- 输入信号有：Alu_Result, mo, m2reg

- 无内部 wire 信号

【结论】MEM_WB 级流水线寄存器需要传递的信号有：

Alu_Result，m2reg，wreg，rn，mo

三）本次课程的软件环境：

同实验一。

四）本次实验设计的 CPU 支持的指令集（32 位）：

同实验一。

五、实验目的：

1. 掌握流水线 CPU 和单周期 CPU 的区别；

2. 进一步熟悉 Verilog HDL 硬件设计语言；

3. 进一步掌握开发平台 Xilinx ISE Design Suite 14.7 集成开发系统的操作方法。

六、实验内容：

1. 在单周期 CPU 代码的基础上添加流水线 CPU 相关代码，完成下列流水线寄存器的构造：

1) IF_ID 级流水线寄存器（instruction_register）

2) ID_EXE 级流水线寄存器（id_exe_register）

3) EXE_MEM 级流水线寄存器（exe_mem_register）

4) MEM_WB 级流水线寄存器（mem_wb_register）；

2. 按以下方式对寄存器与数据存储器进行初始化：

寄存器：

register[5'h01]<=32'h00000001;

register[5'h02]<=32'h00000002;

register[5'h03]<=32'h00000003;

register[5'h04]<=32'h00000004;

register[5'h05]<=32'h00000005;

register[5'h06]<=32'h00000006;

register[5'h07]<=32'h00000007;

register[5'h08]<=32'h00000008;

数据存储器：

ram[5'h01]=32'h00000001;

ram[5'h02]=32'h00000002;

ram[5'h03]=32'h00000003;

ram[5'h04]=32'h00000004;

ram[5'h05]=32'h00000005;

ram[5'h06]=32'h00000006;

ram[5'h07]=32'h00000007;

ram[5'h08]=32'h00000008;

3. 自行设计相关指令序列，对所实现的流水线 CPU 进行仿真，验证并分析该指令序列的运行结果，指令需事先写入指令存储器。

4. 根据代码补充绘制完整的五级流水线 CPU 电路结构图，完整标出 CPU 结构中各信号名称及传递方向，并说明各信号在 CPU 工作流程中的作用。

七、实验器材：

同实验一

八、实验步骤：

同实验一

九、实验数据及结果分析：

1.流水线寄存器的构造

1) IF_ID 级流水线寄存器（instruction_register）

完整代码如下：

```
21  module IF_ID( if_pc4, if_inst, clk, clrn, id_pc4, id_inst
22      );
23
24      input[31:0] if_pc4, if_inst;
25      input clk, clrn;
26      output[31:0] id_pc4, id_inst;
27
28      // if_pc4, if_inst是引线传入的信号
29      // 传入寄存器id_pc4, id_inst中保存
30      reg[31:0] id_pc4, id_inst;
31
32      // 时钟上升沿，clrn=0，则寄存器信号清零
33      // 否则，将if阶段的信号存入寄存器中，作为id阶段的输入
34      always @ (posedge clk or negedge clrn)
35        if(clrn==0)
36           begin
37              id_pc4<=0;
38              id_inst<=0;
39           end
40        else
41           begin
42              id_pc4<=if_pc4;
43              id_inst<=if_inst;
44           end
45
46  endmodule
```

2) ID_EXE 级流水线寄存器（id_exe_register）

完整代码如下：

```
21  module ID_EXE(clk, clrn,
22                id_m2reg,id_wmem,id_aluc,id_aluimm,id_shift,id_wreg,
23                id_qa,id_qb,id_imm,id_rn,
24                exe_m2reg,exe_wmem,exe_aluc,exe_aluimm,exe_shift,exe_wreg,
25                exe_qa,exe_qb,exe_imm,exe_rn
26      );
27
28      input clk, clrn;
29      input id_m2reg,id_wmem,id_aluimm,id_shift,id_wreg;
30      input[2:0] id_aluc;
31      input[31:0] id_qa,id_qb,id_imm;
32      input[4:0] id_rn;
33
34      output exe_m2reg,exe_wmem,exe_aluimm,exe_shift,exe_wreg;
35      output[2:0] exe_aluc;
36      output[31:0] exe_qa,exe_qb,exe_imm;
37      output[4:0] exe_rn;
38
39      reg exe_m2reg,exe_wmem,exe_aluimm,exe_shift,exe_wreg;
40      reg[2:0] exe_aluc;
41      reg[31:0] exe_qa,exe_qb,exe_imm;
42      reg[4:0] exe_rn;
```

```
44       always @ (posedge clk or negedge clrn)
45         if(clrn==0)
46            begin
47               exe_m2reg<=0;
48               exe_wmem<=0;
49               exe_aluc<=0;
50               exe_aluimm<=0;
51               exe_shift<=0;
52               exe_wreg<=0;
53               exe_qa<=0;
54               exe_qb<=0;
55               exe_imm<=0;
56               exe_rn<=0;
57            end
58         else
59            begin
60               exe_m2reg<=id_m2reg;
61               exe_wmem<=id_wmem;
62               exe_aluc<=id_aluc;
63               exe_aluimm<=id_aluimm;
64               exe_shift<=id_shift;
65               exe_wreg<=id_wreg;
66               exe_qa<=id_qa;
67               exe_qb<=id_qb;
68               exe_imm<=id_imm;
69               exe_rn<=id_rn;
70            end
71  endmodule
```

3) EXE_MEM 级流水线寄存器（exe_mem_register）

完整代码如下：

```
21  module EXE_MEM(clk, clrn,
22              exe_Alu_Result, exe_rb, exe_wmem, exe_m2reg, exe_wreg, exe_rn,
23              mem_Alu_Result, mem_rb, mem_wmem, mem_m2reg, mem_wreg, mem_rn
24      );
25
26      input clk,clrn;
27      input[31:0] exe_Alu_Result, exe_rb;
28      input[4:0] exe_rn;
29      input exe_wmem, exe_m2reg, exe_wreg;
30
31      output[31:0] mem_Alu_Result, mem_rb;
32      output[4:0] mem_rn;
33      output mem_wmem, mem_m2reg, mem_wreg;
34
35      reg[31:0] mem_Alu_Result, mem_rb;
36      reg[4:0] mem_rn;
37      reg mem_wmem, mem_m2reg, mem_wreg;
38
```

```
39        always @ (posedge clk or negedge clrn)
40        if(clrn==0)
41            begin
42               mem_Alu_Result<=0;
43               mem_rb<=0;
44               mem_wmem<=0;
45               mem_m2reg<=0;
46               mem_wreg<=0;
47               mem_rn<=0;
48            end
49         else
50            begin
51               mem_Alu_Result<=exe_Alu_Result;
52               mem_rb<=exe_rb;
53               mem_wmem<=exe_wmem;
54               mem_m2reg<=exe_m2reg;
55               mem_wreg<=exe_wreg;
56               mem_rn<=exe_rn;
57            end
58  endmodule
```

4) MEM_WB 级流水线寄存器（mem_wb_register）；

完整代码如下：

```
21  module MEM_WB(clk, clrn,
22                mem_Alu_Result,mem_m2reg,mem_wreg,mem_rn,mem_mo,
23                wb_Alu_Result,wb_m2reg,wb_wreg,wb_rn,wb_mo
24      );
25
26      input clk, clrn;
27      input[31:0] mem_Alu_Result,mem_mo;
28      input mem_m2reg,mem_wreg;
29      input[4:0] mem_rn;
30
31      output[31:0] wb_Alu_Result,wb_mo;
32      output wb_m2reg,wb_wreg;
33      output[4:0] wb_rn;
34
35      reg[31:0] wb_Alu_Result,wb_mo;
36      reg wb_m2reg,wb_wreg;
37      reg[4:0] wb_rn;
38
39      always @ (posedge clk or negedge clrn)
40        if(clrn==0)
41           begin
42              wb_Alu_Result<=0;
43              wb_mo<=0;
44              wb_m2reg<=0;
45              wb_wreg<=0;
46              wb_rn<=0;
47           end
48        else
49           begin
50              wb_Alu_Result<=mem_Alu_Result;
51              wb_mo<=mem_mo;
52              wb_m2reg<=mem_m2reg;
53              wb_wreg<=mem_wreg;
54              wb_rn<=mem_rn;
55           end
56
57  endmodule
```

2. 初始化部分

1）寄存器堆初始化：

同上所述

2）数据存储器初始化：

同上所述

3）指令存储器初始化：

对应的初始化 inst_mem 部分 Verilog 代码如下：

1）无冒险、不含转移指令的指令序列

```
assign rom[6'h00]=32'h00000000;    //0 地址为空，从 1 地址开始执行；
assign rom[6'h01]=32'h28033046;    //ori r6,r2,0x00cc    r6=0x000000ce
assign rom[6'h02]=32'h00101464;    //add r5,r3,r4    r5=0x00000007
assign rom[6'h03]=32'h04100841;     //and r2,r2,r1    r2=0x00000000
assign rom[6'h04]=32'h04200823;     //or r2,r1,r3    r2=0x00000003
assign rom[6'h05]=32'h38000866;    //store          r6,          0x0002(r3)
m5=0x000000ce
assign rom[6'h06]=32'h14000901;     //addi r1,r8,0x02 r1=0x0000000a
assign rom[6'h07]=32'h34000489;    //load           r9,          0x0001(r4)
r9=0x000000ce
assign rom[6'h08]=32'h044020e5;      //xor r8,r7,r5    r8=0x00000000
```

2）含冒险的指令序列

```
assign rom[6'h00]=32'h00000000;    //0 地址为空，从 1 地址开始执行；
assign rom[6'h01]=32'h28033046;    //ori r6,r2,0x00cc    r6=0x000000ce
assign rom[6'h02]=32'h00101464;    //add r5,r3,r4    r5=0x00000007
assign rom[6'h03]=32'h04100841;     //and r2,r2,r1
assign rom[6'h04]=32'h38000866;      //store r6,0x0002(r3)
assign rom[6'h05]=32'h14000901;      //addi r1,r8,0x02
```

3）含转移指令的指令序列

assign rom[6'h00]=32'h00000000;　//0 地址为空，从 1 地址开始执行；

assign rom[6'h01]=32'h28033046;　//ori r6,r2,0x00cc　r6=0x000000ce

assign rom[6'h02]=32'h00101464;　//add r5,r3,r4　r5=0x00000007

assign rom[6'h03]=32'h04100841;　//and r2,r2,r1

assign rom[6'h04]=32'h3c000c21;　//beq　r1,r1,6'h03　　offset=0x0003 相等转移到 5+3=8 处

assign rom[6'h05]=32'h04200823;　　//or r2,r1,r3

assign rom[6'h06]=32'h14000901;　　//addi r1,r8,0x02

assign rom[6'h07]=32'h34000489;　//load　　r9,0x0001(r4) r9=0x00000005

assign rom[6'h08]=32'h044020e5;　　//xor r8,r7,r5

3. 修改后的相关模块（贴出修改过的模块的完整代码，如顶层模块等）：

1）ID_STAGE 模块完整代码（含关键注释）

在 ID_STAGE 模块中将 ID 阶段译码输出的 wreg 信号和目的寄存器 rn 编号 WB 阶段回写 RegFile 使用的 wb_wreg, wb_rn 区分开

```
// 单周期的wreg, rn在ID级内部消化了，所以定义成wire类型
// 流水线要改成output类型: id_wreg, id_rn, input类型 wb_wreg, wb_rn
module ID_STAGE(pc4,inst,id_rn,
                wdi,clk,clrn,bpc,jpc,pcsource,
                m2reg,wmem,aluc,aluimm,a,b,imm,r9,
                shift,wreg,
                rsrtequ,
                wb_wreg,wb_rn //WB级回传
    );

```

```
    input [31:0] pc4,inst,wdi;      //pc4-PC值用于计算jpc; inst-读取的指令; wdi-向寄存器写入的数据
    input clk,clrn;     //clk-时钟信号; clrn-复位信号;
    input rsrtequ;      //branch控制信号
    input wb_wreg;      // WB阶段回传的wreg信号
    input [4:0] wb_rn;

    output [31:0] bpc,jpc,a,b,imm,r9;     //bpc-branch_pc; jpc-jump_pc; a-寄存器操作数a;
                                          //b-寄存器操作数b; imm-立即数操作数
    output [4:0] id_rn;    //写回寄存器号
    output [2:0] aluc;     //ALU控制信号
    output [1:0] pcsource;    //下一条指令地址选择
    output m2reg,wmem,aluimm,shift,wreg;

    wire [5:0] op,func;
    wire [4:0] rs,rt,rd;
    wire [31:0] qa,qb,br_offset;
    wire [15:0] ext16;
    wire regrt,sext,e;

    assign func=inst[25:20];
    assign op=inst[31:26];
    assign rs=inst[9:5];
    assign rt=inst[4:0];
    assign rd=inst[14:10];
    Control_Unit cu(rsrtequ,func,                          //控制部件
                op,wreg,m2reg,wmem,aluc,regrt,aluimm,
                sext,pcsource,shift);
```

```
    Regfile rf (rs,rt,wdi,wb_rn,wb_wreg,clk,clrn,qa,qb,r9);//寄存器堆，有32个32位的寄存器，0号寄存器恒为0
    mux5_2_1 des_reg_num (rd,rt,regrt,id_rn) ; //选择目的寄存器是来自于rd,还是rt

    assign a=qa;
    assign b=qb;

    assign e=sext&inst[25];//符号拓展或0拓展
    assign ext16={16{e}};//符号拓展
    assign imm={ext16,inst[25:10]};    //将立即数进行符号拓展

    assign br_offset={imm[29:0],2'b00};      //计算偏移地址
    add32 br_addr (pc4,br_offset,bpc);    //beq,bne指令的目标地址的计算
    assign jpc={pc4[31:28],inst[25:0],2'b00};      //jump指令的目标地址的计算

endmodule
```

2）顶层模块完整代码（含关键注释）

```
// 增加相邻阶段之间的流水段寄存器
module SCCPU(Clock, Resetn, PC,
    if_inst, id_inst,
    exe_Alu_Result, mem_Alu_Result, wb_Alu_Result,
    mem_mo, wb_mo,
    m5, r9
     );
     input Clock, Resetn;
     output [31:0] PC, if_inst, id_inst, exe_Alu_Result, mem_Alu_Result, wb_Alu_Result;
     output [31:0] mem_mo, wb_mo; // 输出pc, 各阶段的inst,Alu_Result, mo
     output [31:0] m5, r9;        // 输出rom5和reg9的内容，检验指令是否正确执行

     wire [1:0] pcsource;
     wire [31:0] bpc, jpc, if_pc4, id_pc4;

     wire [31:0] wdi;
     wire [31:0] id_ra, exe_ra, id_rb, exe_rb, mem_rb;
     wire [31:0] id_imm, exe_imm;

     wire [4:0] id_rn, exe_rn, mem_rn, wb_rn;

     wire id_wreg, exe_wreg, mem_wreg, wb_wreg;
     wire id_m2reg, exe_m2reg, mem_m2reg, wb_m2reg;
     wire id_wmem, exe_wmem, mem_wmem, id_aluimm, exe_aluimm;
     wire [2:0] id_aluc, exe_aluc;
     wire id_shift, exe_shift, z;
```

```
47          _          _
48      IF_STAGE stage1 (Clock, Resetn, pcsource, bpc, jpc, if_pc4, if_inst, PC);
49
50      IF_ID IR (if_pc4, if_inst, Clock, Resetn, id_pc4, id_inst);
51
52      ID_STAGE stage2 (id_pc4, id_inst, id_rn, wdi, Clock, Resetn, bpc, jpc, pcsource,
53                id_m2reg, id_wmem, id_aluc, id_aluimm, id_ra, id_rb, id_imm, r9, id_shift, id_wreg,
54                z, wb_wreg, wb_rn);
55
56      ID_EXE id_exe_reg (Clock, Resetn,
57                id_m2reg,id_wmem,id_aluc,id_aluimm,id_shift,id_wreg,
58                id_ra,id_rb,id_imm,id_rn,
59                exe_m2reg,exe_wmem,exe_aluc,exe_aluimm,exe_shift,exe_wreg,
60                exe_ra,exe_rb,exe_imm,exe_rn
61                );
62
63      //z信号直接连到pc模块，不使用nop的情况下不会正确转移
64      EXE_STAGE stage3 (exe_aluc, exe_aluimm, exe_ra, exe_rb, exe_imm, exe_shift, exe_Alu_Result, z);
65
66      EXE_MEM exe_mem_reg (Clock, Resetn,
67              exe_Alu_Result, exe_rb, exe_wmem, exe_m2reg, exe_wreg, exe_rn,
68              mem_Alu_Result, mem_rb, mem_wmem, mem_m2reg, mem_wreg, mem_rn
69      );
```

```
70
71      // Alu_Result[6:2]报错: Cannot index into non-array mem_Alu_Result
72      // 将Alu_Result[6:2]改为直接传入完整的Alu_Result信号
73      MEM_STAGE stage4 (mem_wmem, mem_Alu_Result, mem_rb, Clock, mem_mo, m5);
74
75      MEM_WB mem_wb_reg (Clock, Resetn,
76                mem_Alu_Result,mem_m2reg,mem_wreg,mem_rn,mem_mo,
77                wb_Alu_Result,wb_m2reg,wb_wreg,wb_rn,wb_mo);
78
79      WB_STAGE stage5 (wb_Alu_Result, wb_mo, wb_m2reg, wdi); //wdi写入寄存器堆的数据直连到ID_STAGE即可
80
81  endmodule
```

4. 仿真

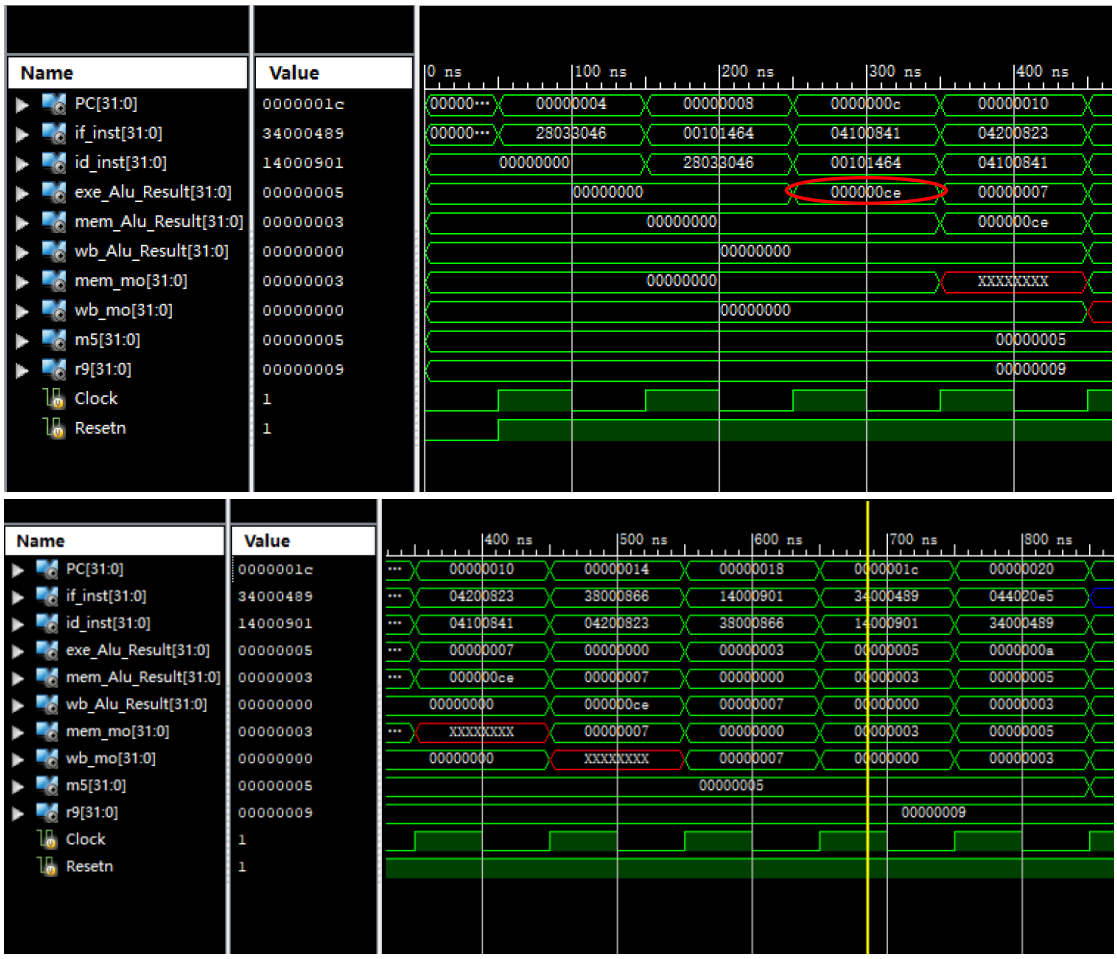
对顶层模块仿真测试代码如下：

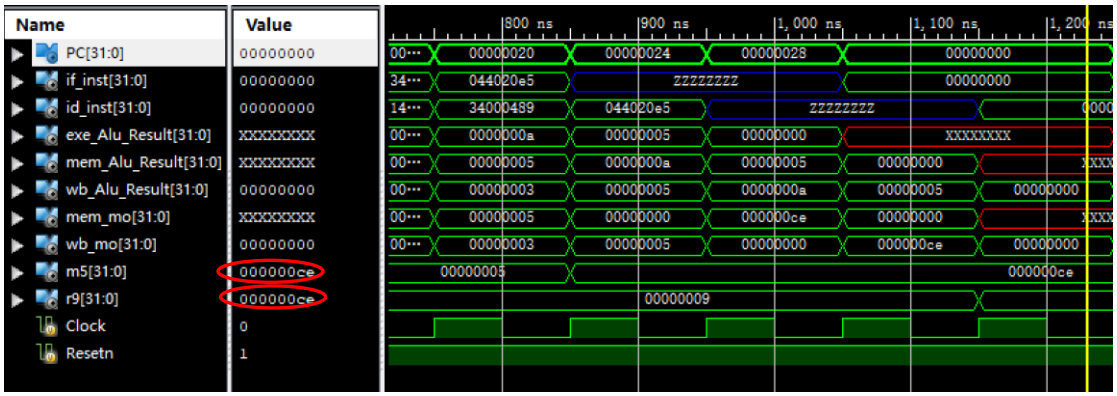
```
25  module SCCPU_test;
26
27      // Inputs
28      reg Clock;
29      reg Resetn;
30
31      // Outputs
32      wire [31:0] PC;
33      wire [31:0] if_inst;
34      wire [31:0] id_inst;
35      wire [31:0] exe_Alu_Result;
36      wire [31:0] mem_Alu_Result;
37      wire [31:0] wb_Alu_Result;
38      wire [31:0] mem_mo;
39      wire [31:0] wb_mo;
40      wire [31:0] m5, r9;
41
42      // Instantiate the Unit Under Test (UUT)
43      SCCPU uut (
44          .Clock(Clock),
45          .Resetn(Resetn),
46          .PC(PC),
47          .if_inst(if_inst),
```

```
48          .id_inst(id_inst),
49          .exe_Alu_Result(exe_Alu_Result),
50          .mem_Alu_Result(mem_Alu_Result),
51          .wb_Alu_Result(wb_Alu_Result),
52          .mem_mo(mem_mo),
53          .wb_mo(wb_mo),
54          .m5(m5),
55          .r9(r9)
56      );
57
58      initial begin
59          // Initialize Inputs
60          Clock = 0;
61          Resetn = 0;
62
63          // Wait 100 ns for global reset to finish
64          #50;
65          Resetn = 1;
66          // Add stimulus here
67
68      end
69      always #50 Clock=~Clock;
70
71  endmodule
72
```

仿真结果如下：

1）无冒险、不含转移的指令序列





【分析】

如图所示，Resetn=0 时，所有寄存器信号清零；

Resetn 置为 1 后，每次时钟的上升沿，pc 增加 4，if_inst 表示下一条指令，寄存器 id_inst 写入上一阶段 if_inst 的指令；

第三个周期结束，第四个周期刚开始时，第一条指令 ori r6,r2,0x00cc 的计算结果 0x000000ce 写入 EXE/MEM 部分的流水段寄存器 mem_Alu_Result，下一周期再写入寄存器 wb_Alu_Result 中。

第八个周期，第五条指令 store r6, 0x0002(r3)执行到 MEM 阶段，将 r6 的内容 0x000000ce（r6 的初始值为 6，被修改为第一条指令的计算结果）存入地址为 5 的内存空间，从输出结果可看出 m5 被修改为 0x000000ce；

第十一个周期，第七条指令 load r9, 0x0001(r4)执行到 WB 阶段，将 m5 的内容 r9，从输出结果可看出 r9 的初始值为 9，被修改为 0x000000ce。

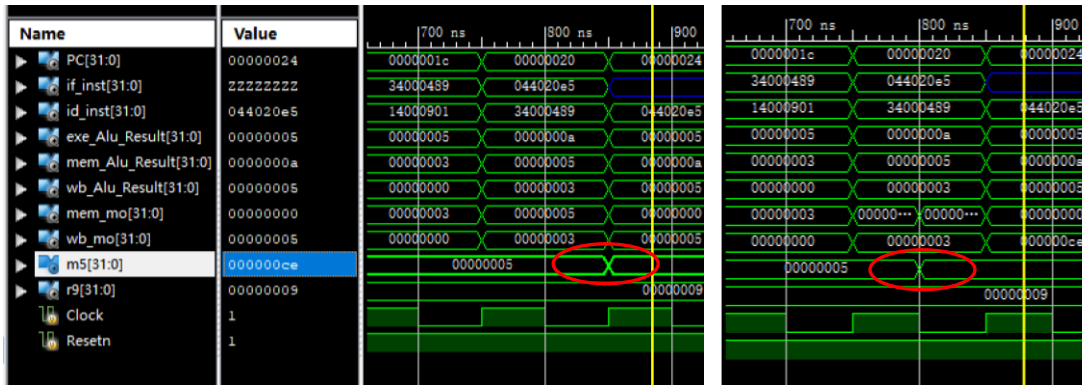
其他输出结果也均符合预期，说明指令正确执行。

补充：探究 regfile 和 mem 的写时钟信号对结果是否有影响？

1. 将传入 MEM_STAGE 的时钟信号改为与顶层模块的时钟相反

的信号，代码如下：

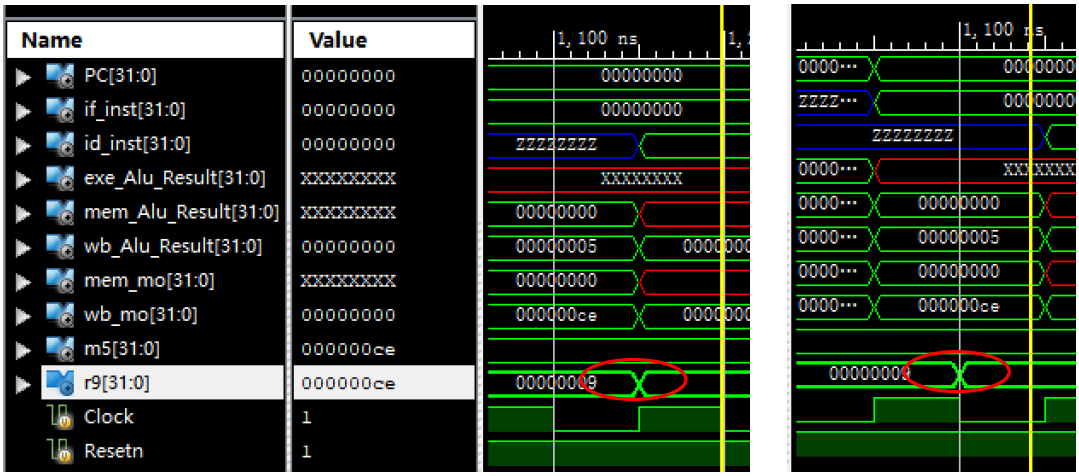
MEM_STAGE stage4 (mem_wmem, mem_Alu_Result, mem_rb, ~Clock, mem_mo, m5);



如上图所示，修改前 CPU 在时钟的上升沿完成写入 mem 的操作（左图），修改后在时钟下降沿完成，提前了半个周期（右图）。

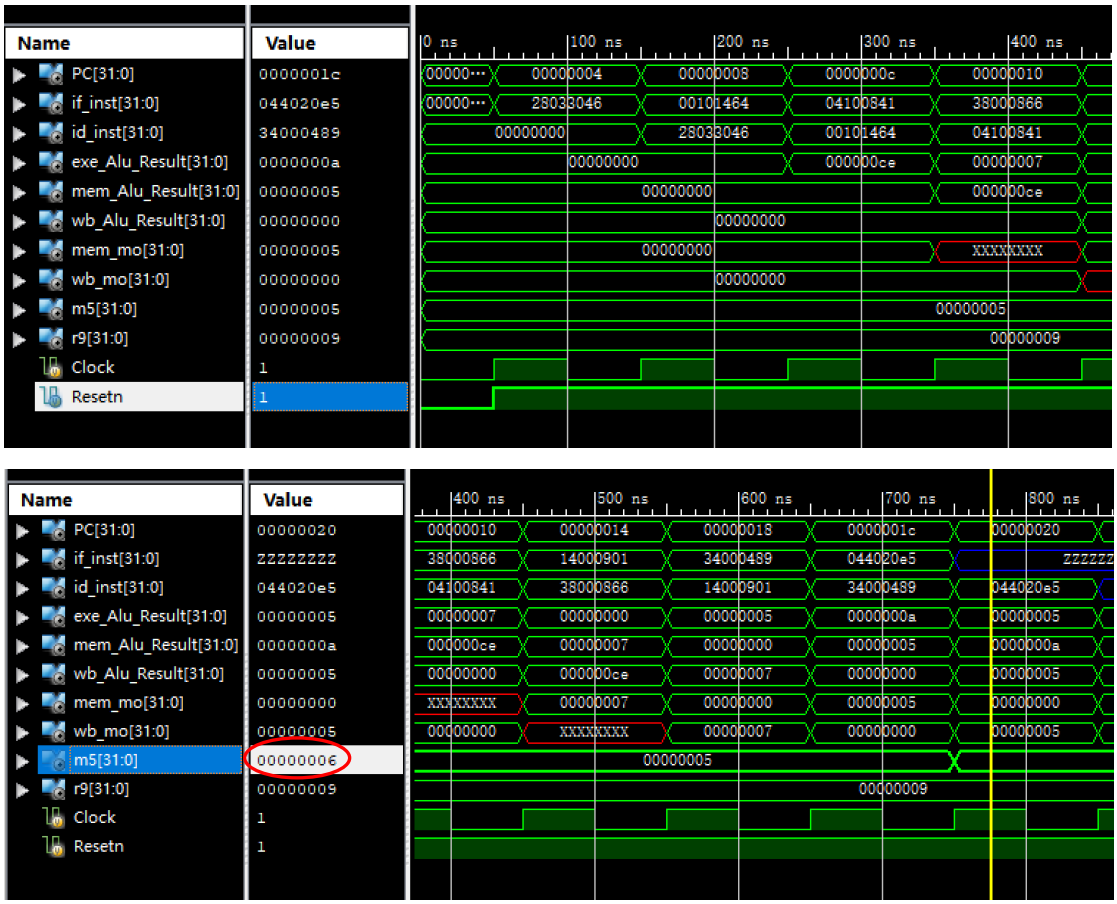
2. 将传入 MEM_STAGE 的时钟信号改为相反的信号，代码如下：

ID_STAGE stage2 (id_pc4, id_inst, id_rn, wdi, ~Clock, Resetn, bpc, jpc, pcsource, id_m2reg, id_wmem, id_aluc, id_aluimm, id_ra, id_rb, id_imm, r9, id_shift, id_wreg, z, wb_wreg, wb_rn);



如上图所示，修改前 CPU 在时钟的上升沿完成回写寄存器的操作（左图），修改后在时钟下降沿完成，提前了半个周期（右图）。

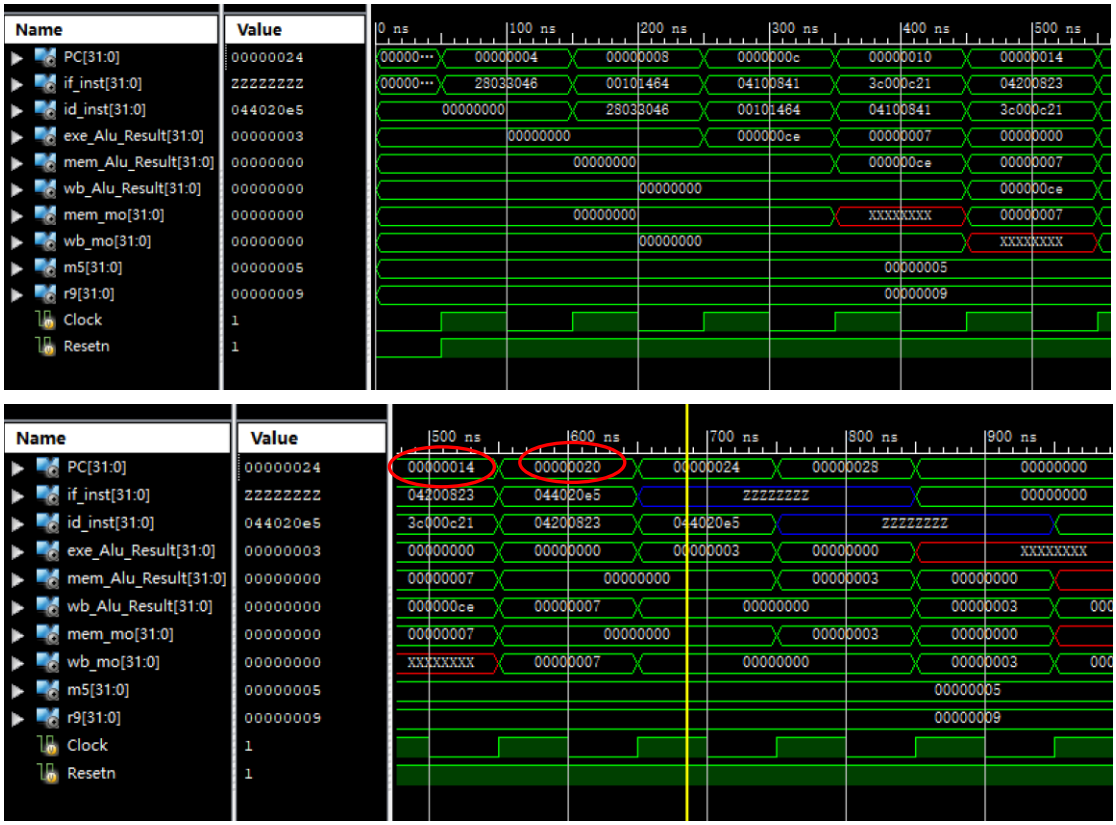
2）含冒险的指令序列



【分析】

第一条指令 ori r6,r2,0x00cc 位于 WB 阶段时，第四条指令 store r6,0x0002(r3)位于 ID 阶段，根据样例代码的设计，写寄存器操作由时序逻辑控制，在时钟上升沿时写入，读寄存器操作由组合逻辑控制，也是在时钟上升沿，ra 和 rb 更新，读出的 qa 和 qb 随之更新，所以不是先写后读，WB 阶段和 ID 阶段的指令也可能存在冒险。如图所示，store 指令读取的 r6 的内容是更新之前的初始值 6，而不是第一条指令更新之后的值 0x00ce 。

3）含转移指令的序列



【分析】

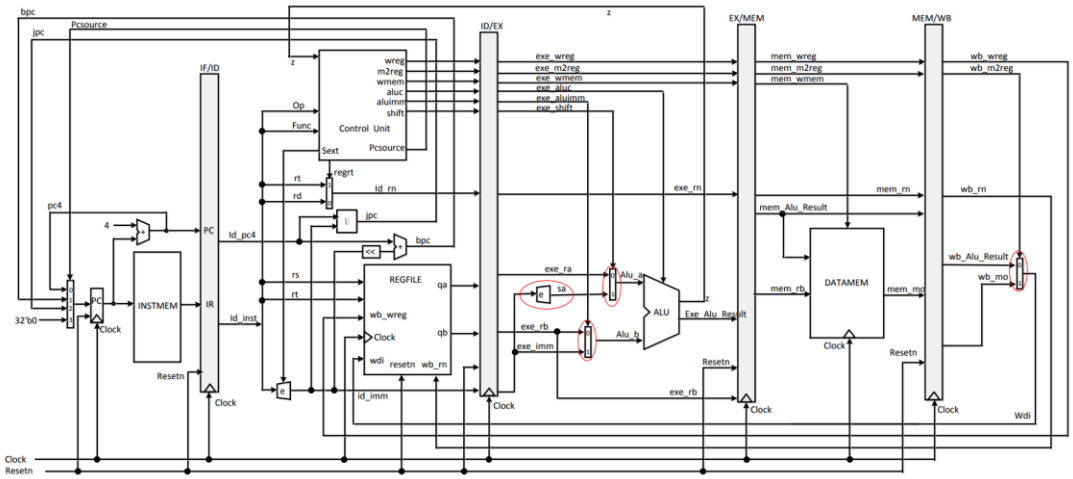
如图所示，第四条指令 beq r1,r7,6'h03 执行到 ID 阶段时，r1 和 r7 的内容不相等，但仍然发生了跳转，这是因为第三条指令执行到 EXE 阶段，且计算结果为 0，所以 z 标志被写为 0，z 信号和 beq 指令操作码传入控制器后，控制器判断应该跳转，可见目前的代码在不插入 nop 指令的情况下转移指令不能正确执行。

【结论】

结论符合预期，不含冒险和转移指令的指令序列能正确执行，无法处理冒险和转移指令。

5. 基础五级流水线 CPU 架构如下：（请根据代码对给出的五级流水

线 CPU 架构图.vsd 进行修改）



修改说明：移位指令的 shift 是从 32 位的 exe_imm 中取出 5 位，然后扩展成 32 位输入多路选择器的，所以图中缺少了一个扩展器。多个多路选择器的 0 和 1 位需要按照代码修改。

图中关键信号说明：

流水线级	控制信号	说明
IF 级	pcsource	控制四路选择器，选择下一条指令的地址
ID 级	regrt	选择目的寄存器
EXE 级	aluimm	选择立即数
	aluc	ALU 操作码
	shift	选择移位位数
MEM 级	wmem	写存储器使能信号

WB 级	m2reg	选择写回的数据
	wreg	写寄存器堆

控制信号在 ID 级产生，在被消耗之前要随着流水线逐级传递，在流水段寄存器中保存。

在单周期 CPU 中，寄存器堆的写使能信号 wreg 和目的寄存器编号 rn 是产生于 ID_STAGE 部件，并作用于该部件的，所以不用输出，定义成内部的 wire 变量即可，但流水线 CPU 中，wreg 和 rn 从 ID 阶段产生后要逐级传递到 WB 阶段才会起作用。

十、实验结论：（联系理论知识进行说明）

流水线 CPU 中，wreg、rn 信号要用流水段寄存器存储，逐级向前传递，不可以直接连线在 ID_STAGE 内部传递给寄存器堆，因为流水段 CPU 的每一级只能在某一时刻完成特定的任务。

该级不使用的控制信号不用传递到该级的部件内部，直接作为下一级的流水段寄存器的输入信号即可。

十一、总结及心得体会：

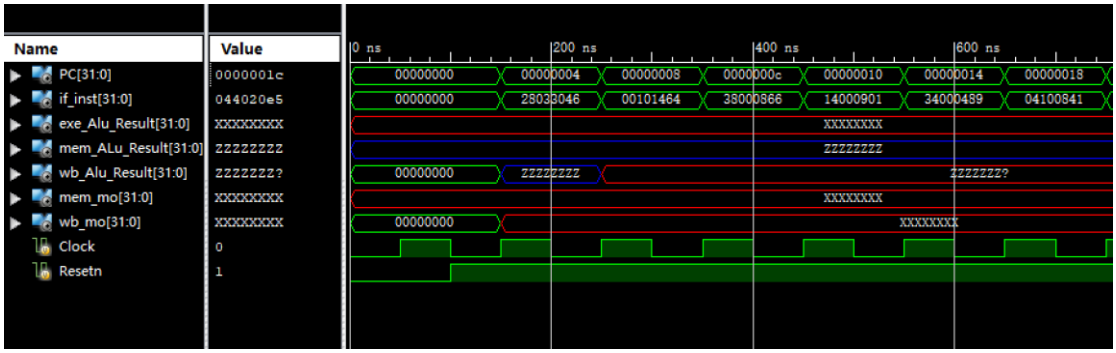
通过改写代码，我对五级流水线 CPU 的结构有了更深的理解和更清晰的认知。通过仿真实验，我很直观地感受到了指令在流水线 CPU 中执行的过程。通过实验探究，我发现了时钟信号对 mem 和 regFile 写入时机的影响，从该思路出发，我们可以将寄存器的读操作也变成由时序逻辑控制，然后通过调整时钟信号，使前半周期回写寄存器

（WB 级），后半周期读取寄存器（ID 级），从而解决同时使用寄存器导致的结构冒险。

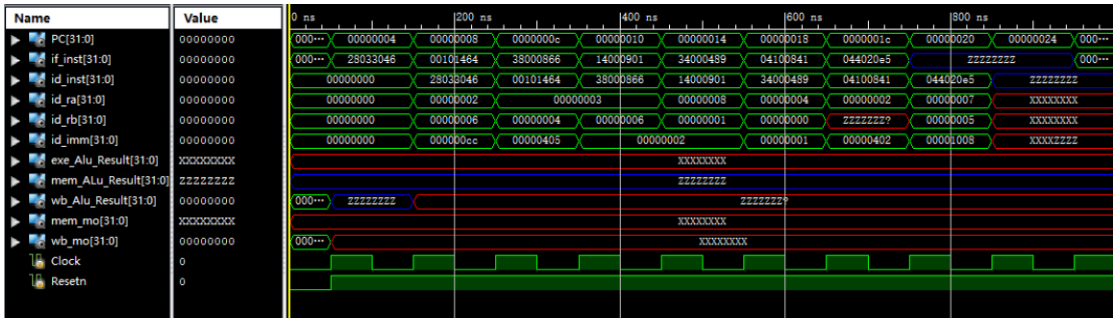
十二、对本实验过程及方法、手段的改进建议：

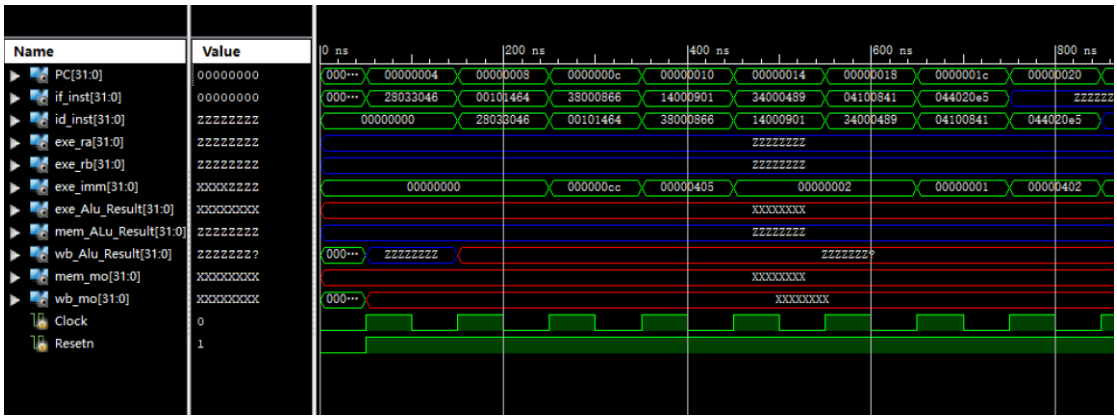
在改写的过程中可能会出现一些小错误导致仿真结果不符合预期，因此我们需要掌握一些 debug 方法：

1） 输出一些中间结果来推测 bug 的位置。例如下图，我们可以看出 EXE 之后的阶段的输出都不符合预期。

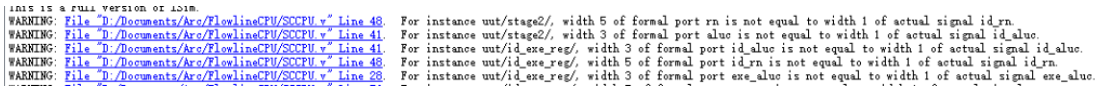


我们将 ID 和 EXE 阶段的 ra，rb 输出，如下图，发现 ID 阶段的信号正确，EXE 阶段的信号错误，于是可推测是 ID_EXE 流水段寄存器附近的信号传递发生了错误，检查后发现是输出信号 ra 误写成 qa 导致的错误。





2）可以查看 WARNING，可能有助于发现 BUG



WARNING 提示信号的宽度不一致，检查后发现某些信号忘记定义了。

由此可见，写代码的过程中一定要细致，注意大小写，命名一致等，不要漏掉信号的定义，要想清楚信号类型和宽度。一旦细节出错可能要花费大量时间定位和排除 BUG。

报告评分：

指导教师签字：