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：王涛　　学　号：2020080902001　　指导教师：王华

实验地点： 清水河校区主楼 A2 区 413-1　　**实验时间：2023.4.27**

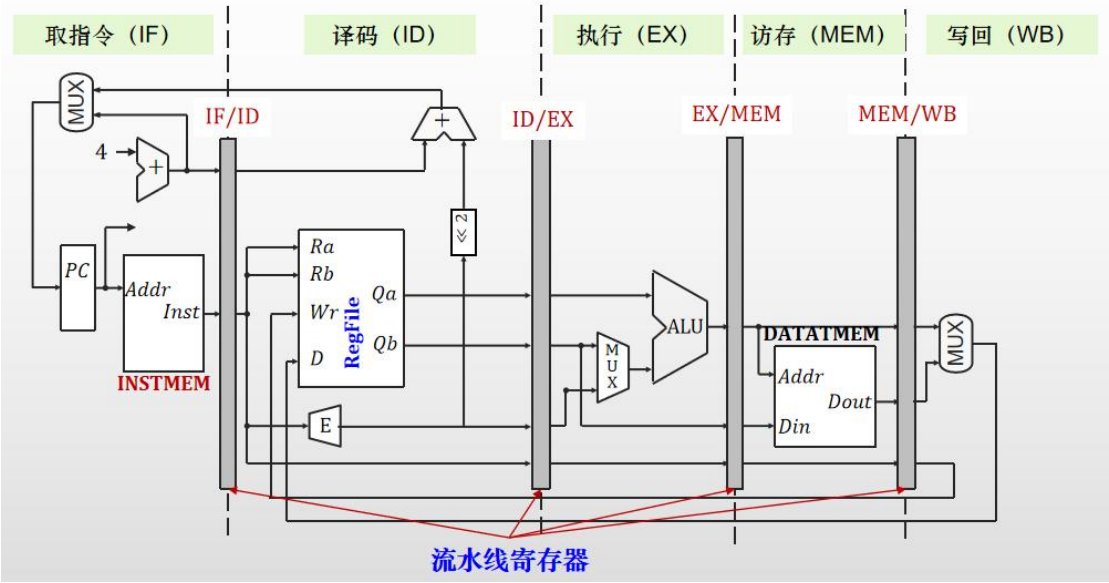
一、实验室名称：国家级计算机实验教学示范中心

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：

```
module IF_IDreg(clk,clrn,if_pc4,if_inst,id_pc4,id_inst
    );
    input [31:0] if_pc4,if_inst;
    input clk,clrn;
    output [31:0] id_pc4,id_inst;

    reg [31:0] id_pc4,id_inst;

    always @ (posedge clk or negedge clrn)
        if(clrn ==0 )
            begin
                id_pc4<=0;
                id_inst<=0;
            end
        else
            begin
                id_pc4<=if_pc4;
                id_inst <=if_inst;
            end
endmodule
```

2) ID_EXE 级流水线寄存器（id_exe_register）

完整代码如下：

```
module ID_EXEreg(clk,clrn,
    id_m2reg,id_wmem,id_aluc,id_aluimm,id_ra,id_rb,id_imm,id_shift,id_wreg,id_rn,
    exe_m2reg,exe_wmem,exe_aluc,exe_aluimm,exe_ra,exe_rb,exe_imm,exe_shift,exe_wreg,ex
e_rn
    );
    input id_m2reg,id_wmem,id_aluimm,id_shift,id_wreg;
    input [2:0] id_aluc;
    input [4:0] id_rn;
    input [31:0] id_ra,id_rb,id_imm;
    input clk,clrn;
    output exe_m2reg,exe_wmem,exe_aluimm,exe_shift,exe_wreg;
    output [2:0] exe_aluc;
    output [4:0] exe_rn;
    output [31:0] exe_ra,exe_rb,exe_imm;

    reg exe_m2reg,exe_wmem,exe_aluimm,exe_shift,exe_wreg;
```

```
    reg [2:0] exe_aluc;
    reg [4:0] exe_rn;
    reg [31:0] exe_ra,exe_rb,exe_imm;

     always @ (posedge clk or negedge clrn)
        if(clrn ==0 )
            begin
                exe_m2reg<=0;
                exe_wmem<=0;
                exe_aluimm<=0;
                exe_shift<=0;
                exe_wreg<=0;
                exe_aluc<=0;
                exe_ra<=0;
                exe_rb<=0;
                exe_rn<=0;
                exe_imm<=0;
            end
        else
            begin
                exe_m2reg<=id_m2reg;
                exe_wmem<=id_wmem;
                exe_aluimm<=id_aluimm;
                exe_shift<=id_shift;
                exe_wreg<=id_wreg;
                exe_aluc<=id_aluc;
                exe_ra<=id_ra;
                exe_rb<=id_rb;
                exe_rn<=id_rn;
                exe_imm<=id_imm;
            end
endmodule
```

3) EXE_MEM 级流水线寄存器（exe_mem_register）

完整代码如下：

```
module EXE_MEMreg(clk,clrn,
    exe_Alu_Result,exe_rb,exe_wmem,exe_m2reg,exe_wreg,exe_rn,
    mem_Alu_Result,mem_rb,mem_wmem,mem_m2reg,mem_wreg,mem_rn
     );
    input exe_wmem,exe_m2reg,exe_wreg;
    input clk, clrn;
    input [4:0] exe_rn;
    input [31:0] exe_Alu_Result,exe_rb;
```

```
    output mem_wmem,mem_m2reg,mem_wreg;
    output [4:0] mem_rn;
    output [31:0] mem_Alu_Result,mem_rb;

    reg mem_wmem,mem_m2reg,mem_wreg;
    reg [4:0] mem_rn;
    reg [31:0] mem_Alu_Result,mem_rb;

    always @ (posedge clk or negedge clrn)
        if(clrn ==0 )
            begin
                mem_wmem<=0;
                mem_m2reg<=0;
                mem_wreg<=0;
                mem_rn<=0;
                mem_Alu_Result<=0;
                mem_rb<=0;
            end
        else
            begin
                mem_wmem<=exe_wmem;
                mem_m2reg<=exe_m2reg;
                mem_wreg<=exe_wreg;
                mem_rn<=exe_rn;
                mem_Alu_Result<=exe_Alu_Result;
                mem_rb<=exe_rb;
            end

endmodule
```

4) MEM_WB 级流水线寄存器（mem_wb_register）；
完整代码如下：

```
module MEM_WBreg(clk,clrn,
   mem_Alu_Result,mem_mo,mem_m2reg,mem_wreg,mem_rn,
    wb_Alu_Result,wb_mo,wb_m2reg,wb_wreg,wb_rn
     );
    input mem_m2reg,mem_wreg;
    input clk, clrn;
    input [4:0] mem_rn;
    input [31:0] mem_Alu_Result,mem_mo;
    output wb_m2reg,wb_wreg;
    output [4:0] wb_rn;
    output [31:0] wb_Alu_Result,wb_mo;
```

```
    reg wb_m2reg,wb_wreg;
    reg [4:0] wb_rn;
    reg [31:0] wb_Alu_Result,wb_mo;


    always @ (posedge clk or negedge clrn)
        if(clrn ==0 )
            begin

                wb_m2reg<=0;
                wb_wreg<=0;
                wb_rn<=0;
                wb_Alu_Result<=0;
                wb_mo<=0;
            end
        else
            begin

                wb_m2reg<=mem_m2reg;
                wb_wreg<=mem_wreg;
                wb_rn<=mem_rn;
                wb_Alu_Result<=mem_Alu_Result;
                wb_mo<=mem_mo;
            end


endmodule
```

核心思想：建立的流水线寄存器的核心思想即为连接两个阶段需要传递的信号，首先将需要传递的信号（区分两个阶段）定义，因为流水线寄存器需要将信号储存起来，因此还应该将信号声明为 reg 型，然后等到时钟上升沿到来，判断如果复位信号为 0，则将所有信号置 0 并传递给下一阶段，否则将上一阶段的信号值赋值给下一阶段对应的信号值。最后在顶层模块中将各个阶段模块与流水线寄存器连接起来，最后形成标准的五级流水线 CPU。

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：

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'h38000866;       /ore r6,0x0002(r3)    m5=0x000000ce
assign rom[6'h04]=32'h34000489;       //load r9,0x0001(r4)    r9=0x000000ce
assign rom[6'h05]=32'h14002d29;            //addi r9,r9,0x000b    r9=0x000000d9
assign rom[6'h06]=32'h3c000c21;       //beq r1,r1,6'h0a    相等转移到 PC+0aH 处
assign rom[6'h07]=32'h48000001;          / 0x0000001    无条件转移到 01H 处
assign rom[6'h08]=32'h00100421;            //add r1,r1,r1
```

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）顶层模块完整代码（含关键注释）

```
module SCCPU(Clock, Resetn, PC, if_Inst, exe_Alu_Result,mem_mo
    );
    input Clock, Resetn;//输入的时钟及复位信号
    output [31:0] PC, if_Inst, exe_Alu_Result,mem_mo; //输出当前 PC 值，指令，alu 计算结果，DM 取出的数据

    wire [31:0] mem_Alu_Result, wb_Alu_Result;//ALU 结果
    wire [1:0] pcsource;   //PC 选择器

    wire [31:0] bpc, jpc, if_pc4, id_pc4, id_Inst,if_Inst; //pc 及指令

    wire [31:0] wdi, id_ra, exe_ra, id_rb, exe_rb, mem_rb, id_imm, exe_imm;
    wire id_m2reg, exe_m2reg,mem_m2reg,wb_m2reg;//写回数据选择器

    wire id_wmem, exe_wmem, id_aluimm, exe_aluimm, id_shift, exe_shift, z;

    wire [2:0] id_aluc, exe_aluc; //alu 操作码

    wire id_wreg, exe_wreg, mem_wreg, wb_wreg; //写寄存器堆信号传递
    wire [4:0] id_rn, exe_rn, mem_rn, wb_rn; //rn 传递

    wire [31:0] mem_mo, wb_mo;//DM 取出的数据

    //取指阶段
    IF_STAGE stage1 (Clock, Resetn, pcsource, bpc, jpc, if_pc4, if_Inst, PC);

    //锁存并传递 pc4，Inst
    IF_IDreg   IF_ID (Clock,Resetn,if_pc4,if_Inst,id_pc4,id_Inst);

    //译码阶段
    ID_STAGE stage2 (id_pc4, id_Inst, wdi, Clock, Resetn, bpc, jpc, pcsource,
                     id_m2reg, id_wmem, id_aluc, id_aluimm, id_ra, id_rb, id_imm,
```

```
                    id_shift, z,id_wreg,id_rn);

    //锁存并传递   m2reg, wmem, aluc, aluimm, ra, rb, imm, shift，wreg，rn
    ID_EXEreg ID_EXE (Clock,Resetn,id_m2reg,id_wmem,id_aluc,id_aluimm,
            id_ra,id_rb,id_imm,id_shift,id_wreg,id_rn,exe_m2reg,exe_wmem,
            exe_aluc,exe_aluimm,exe_ra,exe_rb,exe_imm,exe_shift,exe_wreg,exe_rn);

    //执行阶段
     EXE_STAGE stage3 (exe_aluc, exe_aluimm, exe_ra, exe_rb, exe_imm, exe_shift,
exe_Alu_Result,
            z,exe_m2reg,exe_wmem,exe_wreg,exe_rn);

    //锁存并传递 Alu_Result, rb，wmem，m2reg，wreg，rn
                                                                EXE_MEMreg
EXE_MEM(Clock,Resetn,exe_Alu_Result,exe_rb,exe_wmem,exe_m2reg,exe_wreg,exe_rn,
            mem_Alu_Result,mem_rb,mem_wmem,mem_m2reg,mem_wreg,mem_rn);

    //访存阶段
      MEM_STAGE stage4 (mem_wmem, mem_Alu_Result[4:0], mem_rb, Clock,
mem_mo,
            mem_wreg,mem_m2reg,mem_rn);

    //锁存并传递 Alu_Result，m2reg，wreg，rn，mo
                                                                MEM_WBreg
MEM_WB(Clock,Resetn,mem_Alu_Result,mem_mo,mem_m2reg,mem_wreg,mem_rn,
            wb_Alu_Result,wb_mo,wb_m2reg,wb_wreg,wb_rn);

    //写回寄存器堆
    WB_STAGE stage5 (wb_Alu_Result, wb_mo, wb_m2reg, wdi,wb_wreg,wb_rn);


endmodule
```

2）IF 阶段模块完整代码（含关键注释）

```
module IF_STAGE(clk,clrn,pcsource,bpc,jpc,pc4,inst, PC
    );
    input clk, clrn;
    input [31:0] bpc,jpc;
    input [1:0] pcsource;

    output [31:0] pc4,inst;
    output [31:0] PC;

    wire [31:0] pc;
```

```
    wire [31:0] npc;      //下一条指令地址

    dff32 program_counter(npc,clk,clrn,pc);     //利用 32 位的 D 触发器实现 PC
    add32 pc_plus4(pc,32'h4,pc4);//32 位加法器，用来计算 PC+4
    mux32_4_1 next_pc(pc4,bpc,jpc,32'b0,pcsource,npc);//根据 pcsource 信号选择下一条指令的地址
    Inst_ROM inst_mem(pc[7:2],inst); //指令存储器

    assign PC=pc;

endmodule
```

3）ID 阶段模块完整代码（含关键注释）

```
module ID_STAGE(pc4,inst,
                wdi,clk,clrn,bpc,jpc,pcsource,
                m2reg,wmem,aluc,aluimm,a,b,imm,
                shift,rsrtequ,wreg,rn
    );
    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
    output [31:0] bpc,jpc,a,b,imm;          /c-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 [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;
    output [4:0] rn;

    wire wreg;
    wire [4:0] rn;          //写回寄存器号
    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,rn,wreg,clk,clrn,qa,qb);//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,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};      /指令的目标地址的计算

endmodule
```

4）EXE 阶段模块完整代码（含关键注释）

```
module
EXE_STAGE(ealuc,ealuimm,ea,eb,eimm,eshift,ealu,z,em2reg,ewmem,ewreg,ern
    );
    input [31:0] ea,eb,eimm;      //ea-由寄存器读出的操作数 a；eb-由寄存器读出的操作数 a；eimm-经过扩展的立即数；
    input [2:0] ealuc;          //ALU 控制码
    input ealuimm,eshift,em2reg,ewmem,ewreg;        //ALU 输入操作数的多路选择器
    input [4:0] ern;
    output [31:0] ealu;         //alu 操作输出
    output z;


    wire [31:0] alua,alub,sa;
    wire em2reg,ewmem,ewreg;
    wire [4:0] ern;


    assign sa={27'b0,eimm[9:5]};//移位位数的生成


    mux32_2_1 alu_ina (ea,sa,eshift,alua);//选择 ALU a 端的数据来源
    mux32_2_1 alu_inb (eb,eimm,ealuimm,alub);//选择 ALU b 端的数据来源
```

```
        alu al_unit (alua,alub,ealuc,ealu,z);//ALU

endmodule
```

5）MEM 阶段模块完整代码（含关键注释）

```
module MEM_STAGE(we,addr,datain,clk,dataout,mwreg,mm2reg,mrn
        );
        input [31:0] datain;
        input [4:0] addr;
        input clk,we;
        output [31:0] dataout;
        output mwreg,mm2reg;
        output [4:0] mrn;

        wire mwreg,mm2reg;
        wire [4:0] mrn;

        reg [31:0] ram [0:31];
        assign dataout=ram[addr];          //读出常有效
        always @(posedge clk)begin
        if(we)ram[addr]=datain;
        end

        integer i;
        initial begin            //存储器初始化
        for(i=0;i<32;i=i+1)              //存储器清零
            ram[i]=0;
        ram[5'h01]=32'h00000001;         //存储器对应地址初始化赋值
        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;

             end

endmodule
```

6）WB 阶段模块完整代码（含关键注释）

```
module WB_STAGE(r_alu,m_o,m2reg,wdi,wwreg,wrn
        );
        input [31:0] r_alu;
```

```
    input [31:0] m_o;
    input m2reg;
    output [31:0] wdi;
    output wwreg;
    output [4:0] wrn;

    wire wwreg;
    wire [4:0] wrn;

    mux32_2_1 wb_stage (r_alu, m_o, m2reg, wdi); //写回到 regFile 的值二选一
    //

endmodule
```

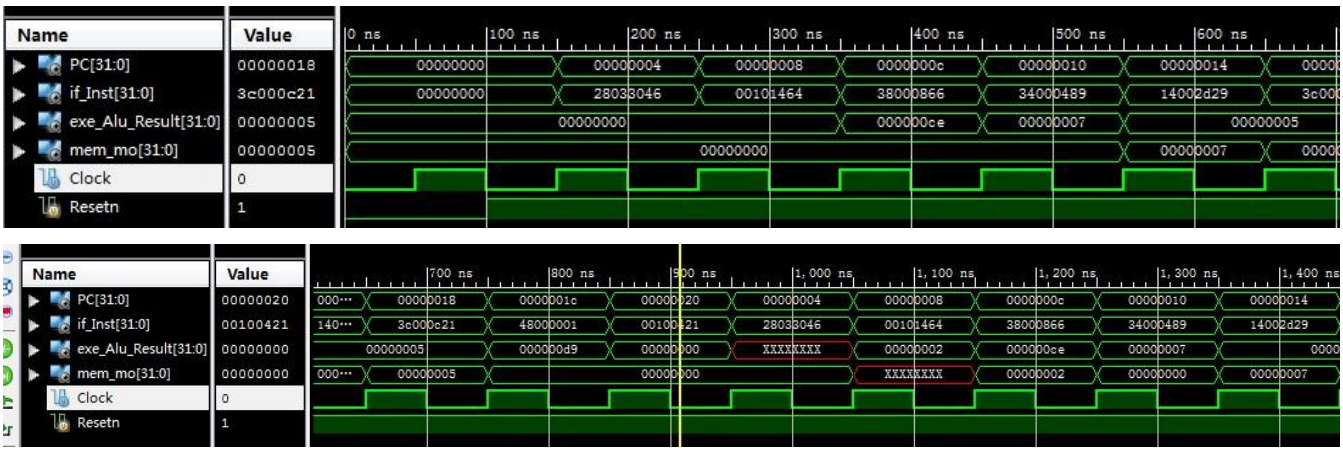
4. 仿真

对顶层模块仿真测试代码如下：

```
module CPU_tb;

    // Inputs
    reg Clock;
    reg Resetn;

    // Outputs
    wire [31:0] PC;
    wire [31:0] if_Inst;
    wire [31:0] exe_Alu_Result;
    wire [31:0] mem_mo;

    // Instantiate the Unit Under Test (UUT)
    SCCPU uut (
        .Clock(Clock),
        .Resetn(Resetn),
        .PC(PC),
        .if_Inst(if_Inst),
        .exe_Alu_Result(exe_Alu_Result),
        .mem_mo(mem_mo)
    );

    initial begin
        // Initialize Inputs
        Clock = 0;
        Resetn = 0;

        // Wait 100 ns for global reset to finish
```

```
        #100;
        Resetn=1;
        // Add stimulus here

    end
    always #50 Clock=~Clock;
endmodule
```

仿真结果如下：



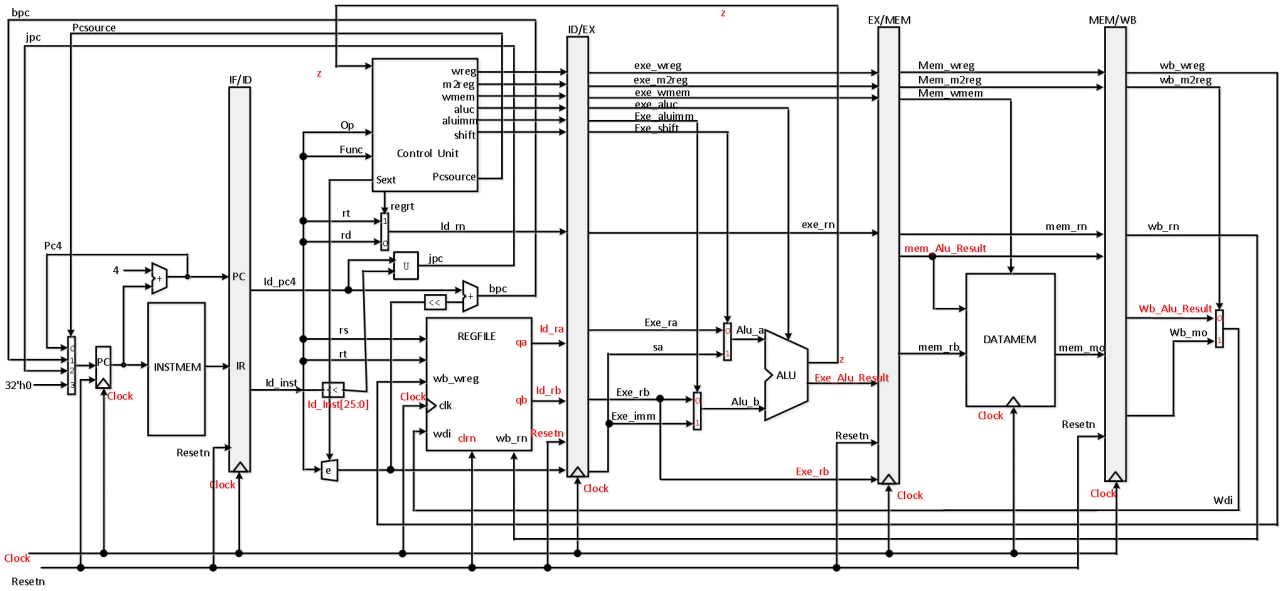
对结果进行说明：

根据流水线的特点以及本实验中使用的指令集，将指令的时空图画出如下图所示，下图展示的是理想情况下指令的执行流水线过程，可以清晰的看到指令中存在数据冒险（ori 和 store）以及控制冒险（beq 和 jump），通过仿真结果可以看到，store 指令在 ID 级取出 r6 的值是 ori 指令还未更新的值，同时在 beq 指令还在等待 EXE 级 0 标志位的出现时，jump 指令已经拿到了跳转的地址，因此在 950ns 时，PC 转向 00000004，可见是执行了 jump 指令，这里存在控制冒险。不过从整体上来看虽然 CPU 存在冒险，但是整体的流水线功能正确，每个周期的结果也和预期相同。

时钟周期数	2	3	4	5	6	7	8	9	10	11	12	13	14
ori r6,r2,0x00cc	ori	r2,0x00cc	or		r6								
add r5,r3,r4		add	r3,r4	+		r5							
store r6,0x0002(r3)			store	r6,r3	+	mem							
load r9,0x0001(r4)				load	r4	+	mem	r9					
addi r9,r9,0x000b					addi	r9	+		r9				
beq r1,r1,6'h0a						beq	r1	-					
jump 0x0000001							jump	addr					
add r1,r1,r1								addr	r1	+		r1	
									ori	r2	or		r6

结论：**结果符合预期**

5. **基础五级流水线 CPU 架构如下：**



修正问题：

1. 修改 ALU a 端和 b 端，WB 级的 2 选 1 选择器，上面为 0，下面为 1；
2. 修正 jpc 的生成方式，jpc 为 pc4 的高 4 位与 inst 低 26 位左移两位进行拼接
3. 修改信号名称与代码一致（信号以顶层模块名称为准）

图中关键信号说明：

Clock：时钟信号
Resetn:复位信号
PC：当前程序计数器值
Jpc: jump 指令跳转地址
Bpc：branch 指令跳转地址
Wreg:是否写寄存器堆
M2reg：是否使用储存器中的数据写寄存器堆
Wmem：是否写储存器
Aluimm：alu 的 b 端是否选择立即数
Shift：alu 的 a 端是否使用立即数
Pcsource：npc 选择信号
Regrt:选择 rt 还是 rd 作为目的寄存器
Aluc：ALU 运算的操作码

十、实验结论：（联系理论知识进行说明）

在本次实验中，对单周期 CPU 进行了改进，在过程中容易犯错的地方就是流水线寄存器与其他模块的结合，在更改信号名称时不仅需要注意上层模块中的信号名统一。还要注意子模块的信号名的一一对应，同时还要将模块之间连接的一些信号声明为 wire 变量，才能正确的仿真。

十一、总结及心得体会：

通过本次实验，加深了对流水线 CPU 的理解，同时也明白了冒险存在的危害，了解了单周期 CPU 与流水线 CPU 在仿真效果上的区别，同时在设计指令集的过程中也加深了对各种指令的理解以及它们在流水线执行中的详细过程。

十二、对本实验过程及方法、手段的改进建议：

可以通过设计更多的指令和更多的输出来加深对指令的理解，同时也能培养调试代码的能力。

报告评分：

指导教师签字：