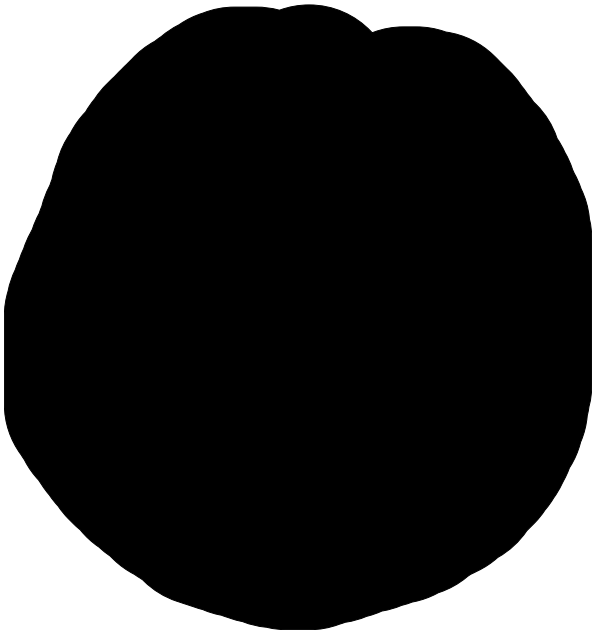
[redacted]

计算机组成原理课程实验报告



学　　号 [redacted]

姓　　名 [redacted]

专　　业 [redacted]

授课老师 [redacted]

一、实验内容

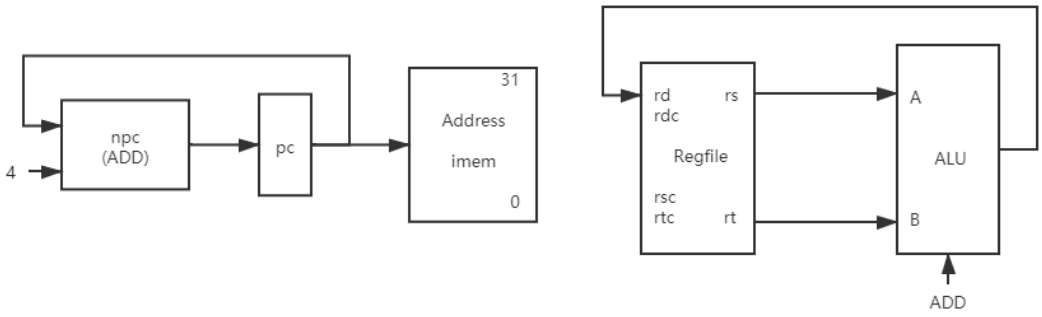
使用 Verilog HDL 实现 54 条 MIPS 指令的 CPU 的设计。

二、实验原理

1. ADD:

格式：ADD rd, rs, rt

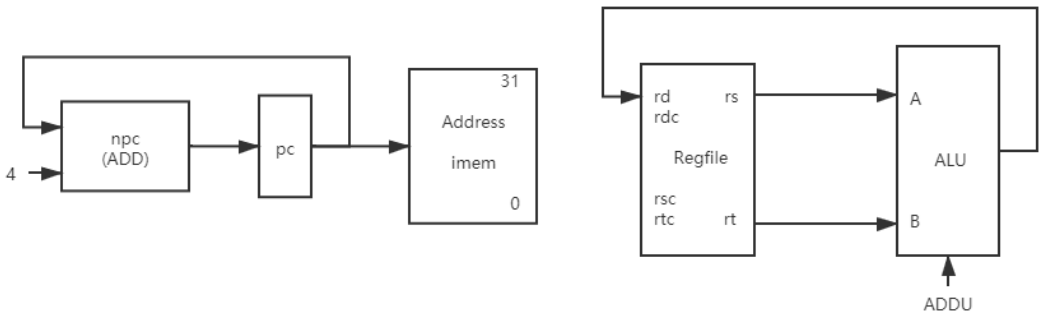
操作：取指令，rd←rs+rt, PC←NPC(PC+4)



2. ADDU:

格式：ADDU rd, rs, rt

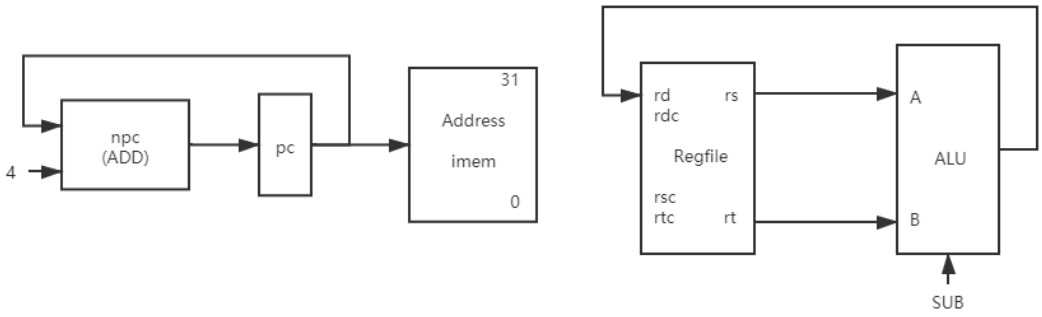
操作：取指令，rd←rs+rt, PC←NPC(PC+4)



3. SUB:

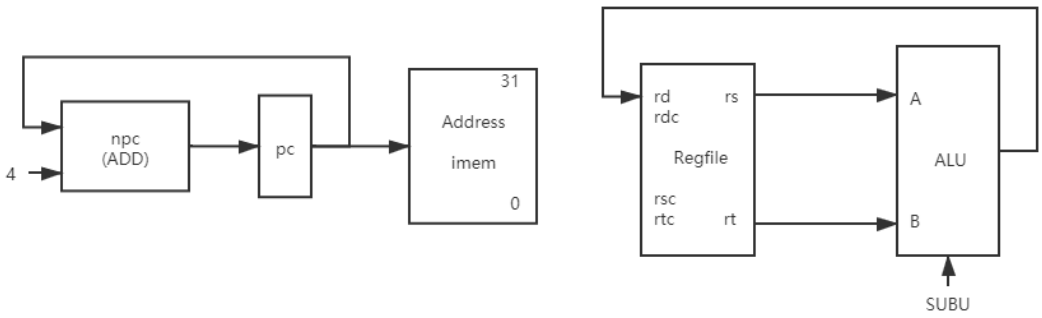
格式：SUB rd, rs, rt

操作：取指令，rd←rs-rt, PC←NPC(PC+4)

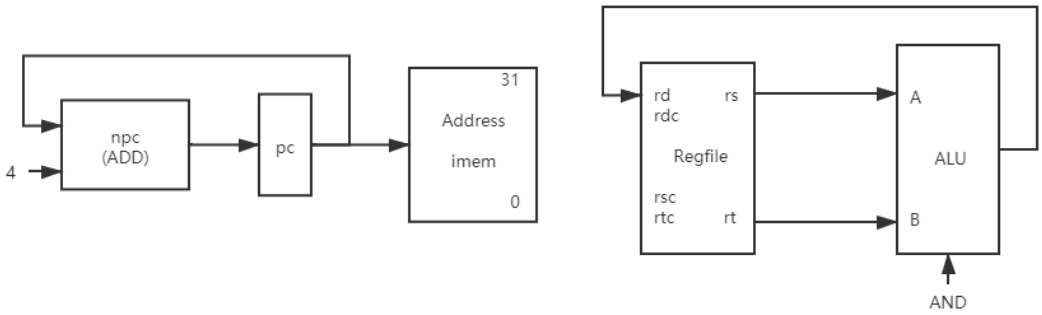


4. SUBU:

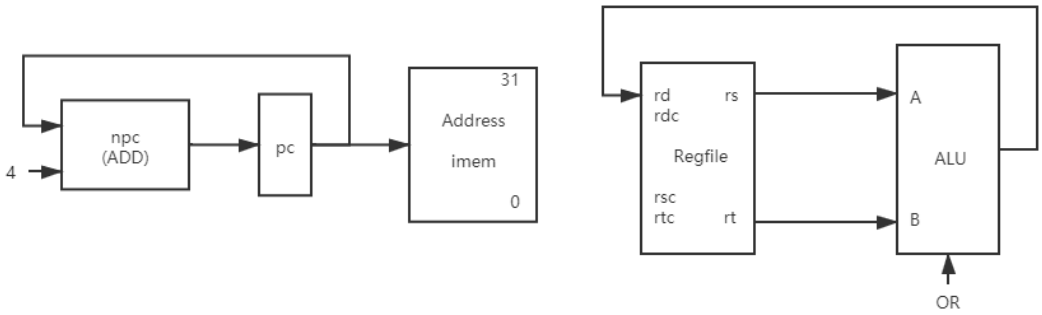
格式：SUBU rd, rs, rt
操作：取指令，rd←rs-rt, PC←NPC(PC+4)



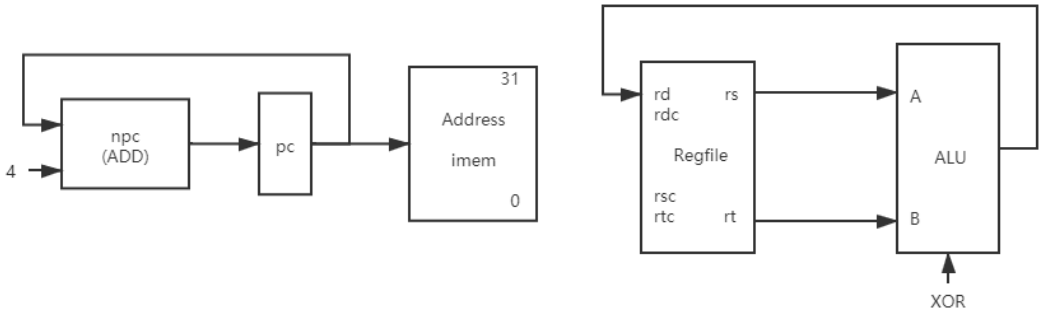
5. AND:
格式：AND rd, rs, rt
操作：取指令，rd←rs & rt, PC←NPC(PC+4)



6. OR:
格式：OR rd, rs, rt
操作：取指令，rd←rs | rt, PC←NPC(PC+4)



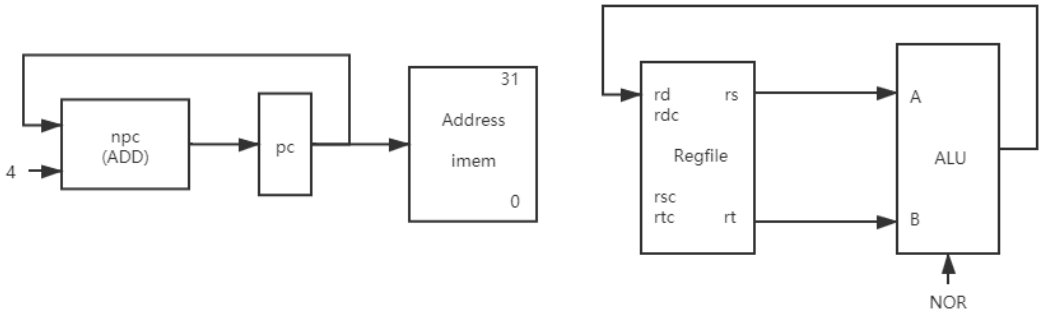
7. XOR:
格式：XOR rd, rs, rt
操作：取指令，rd←rs ^ rt, PC←NPC(PC+4)



8. NOR:

格式：NOR rd, rs, rt

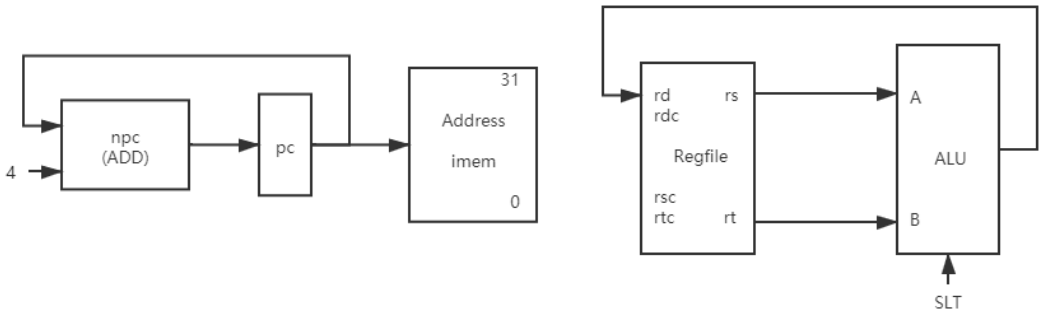
操作：取指令，rd←~(rs | rt), PC←NPC(PC+4)



9. SLT:

格式：NOR rd, rs, rt

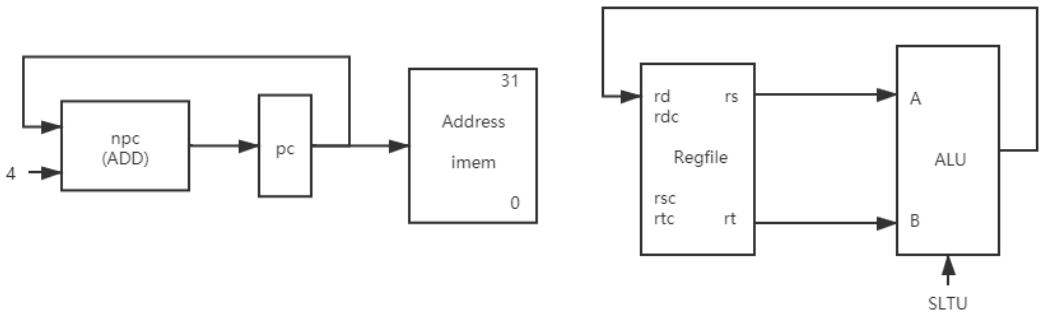
操作：取指令，rd←~(rs | rt), PC←NPC(PC+4)



10. SLTU:

格式：SLTU rd, rs, rt

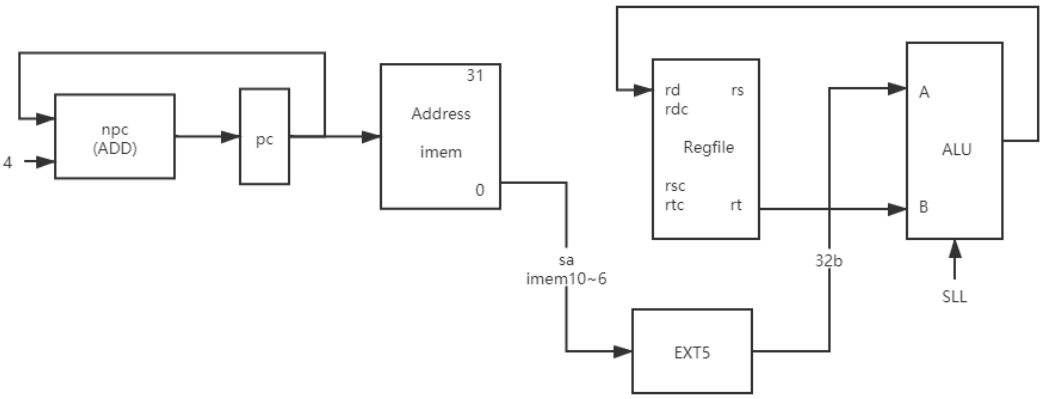
操作：取指令，rd← rs < rt , PC←NPC(PC+4)



11. SLL:

格式：SLL rd, rt, sa

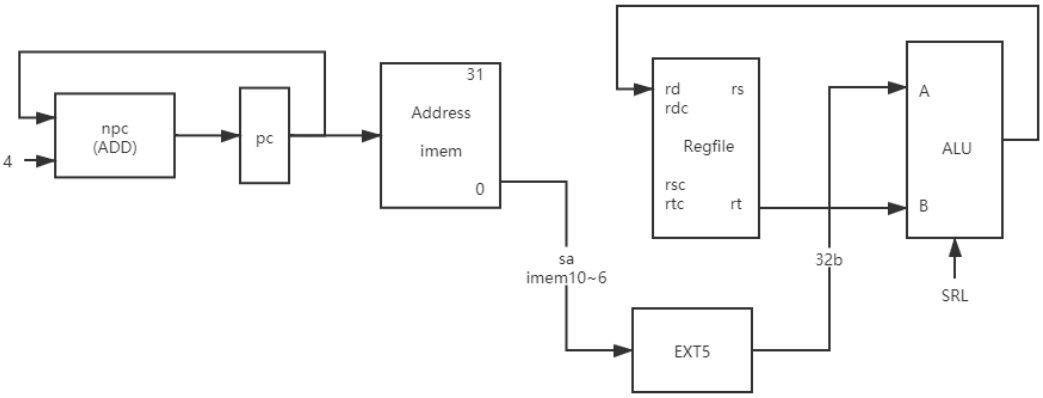
操作：取指令, rd←rt << sa , PC←NPC(PC+4)



12. SRL:

格式：SRL rd, rt, sa

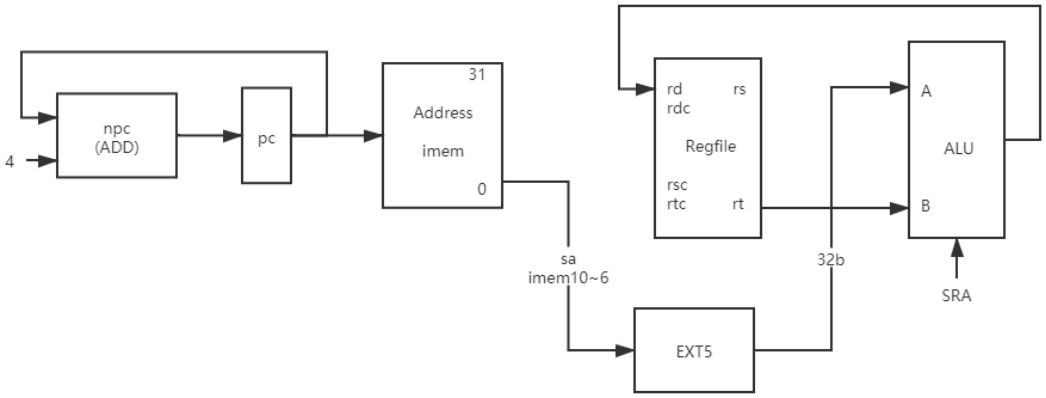
操作：取指令, rd←rt >> sa (logical) , PC←NPC(PC+4)



13.SRA:

格式：SRA rd, rt, sa

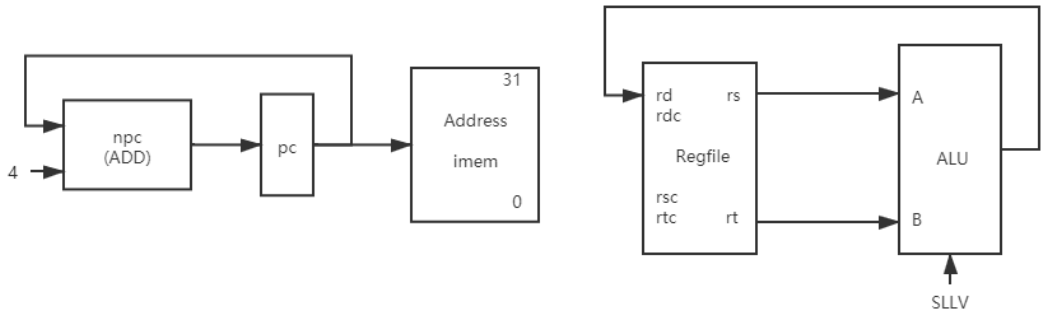
操作：取指令, rd←rt >> sa (arithmetic) , PC←NPC(PC+4)



14 SLLV

格式：SLLV rd, rt, rs,

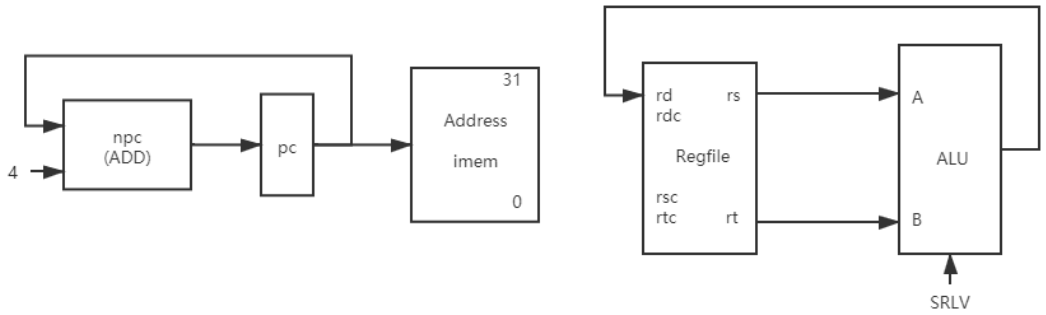
操作：取指令, rd←rt << rs , PC←NPC(PC+4)



15 SRLV

格式：SRLV rd, rt,rs

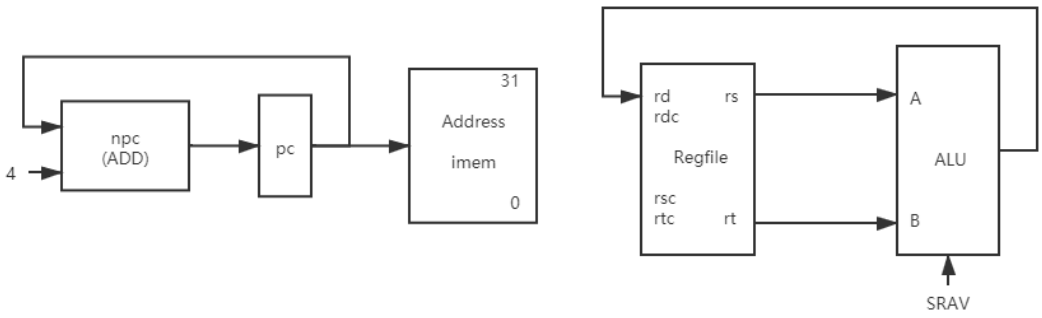
操作：取指令, rd←rt >> rs (logical) , PC←NPC(PC+4)



16 SRAV

格式：SRAV rd, rt,rs

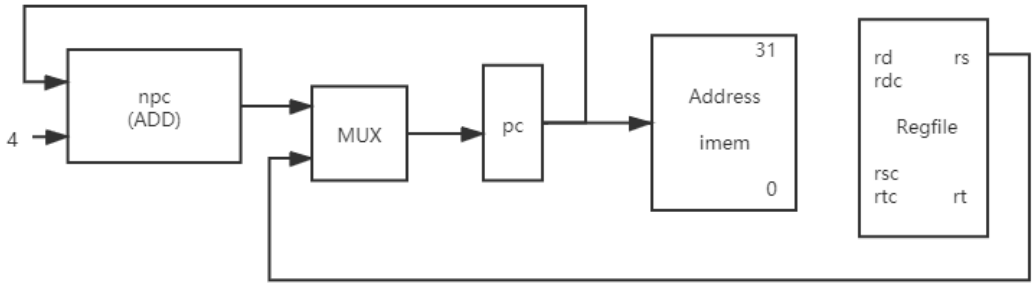
操作：取指令, rd←rt >> rs (arithmetic) , PC←NPC(PC+4)



17.JR

格式：SRAV rd, rt,rs

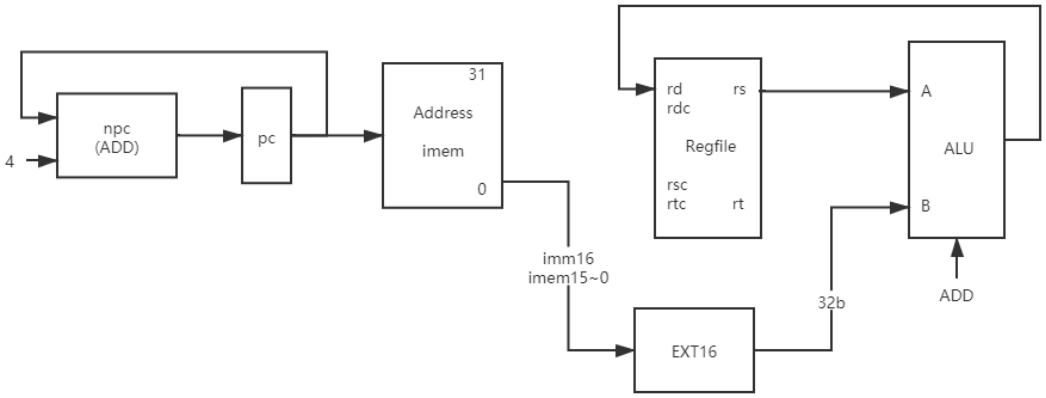
操作：取指令, rd←rt >> rs (arithmetic) , PC←NPC(PC+4)



18.ADDI

格式：ADDI rt, rs, imm16

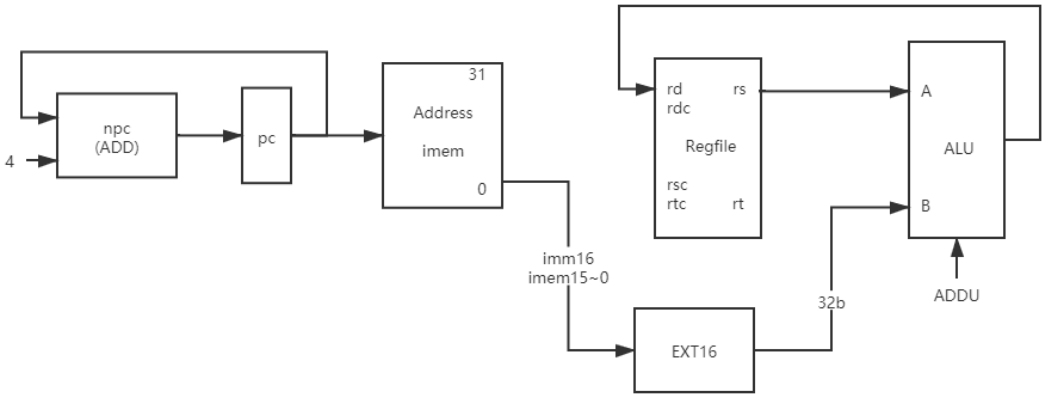
操作：取指令、rt←rs+imm16(sign_extend) 、PC←NPC(PC+4)



19.ADDIU

格式：ADDIU rt, rs, imm16

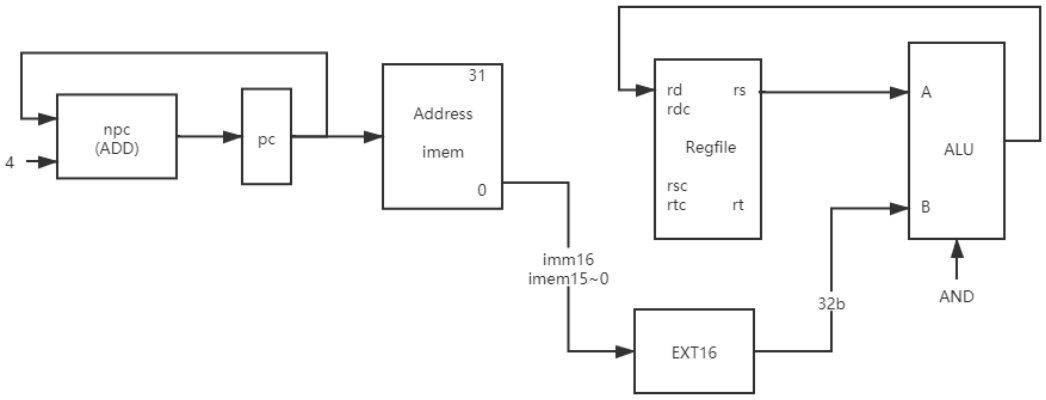
操作：取指令、rt←rs+imm16(sign_extend) 、PC←NPC(PC+4)



20.ANDI

格式：ANDI rt, rs, imm16

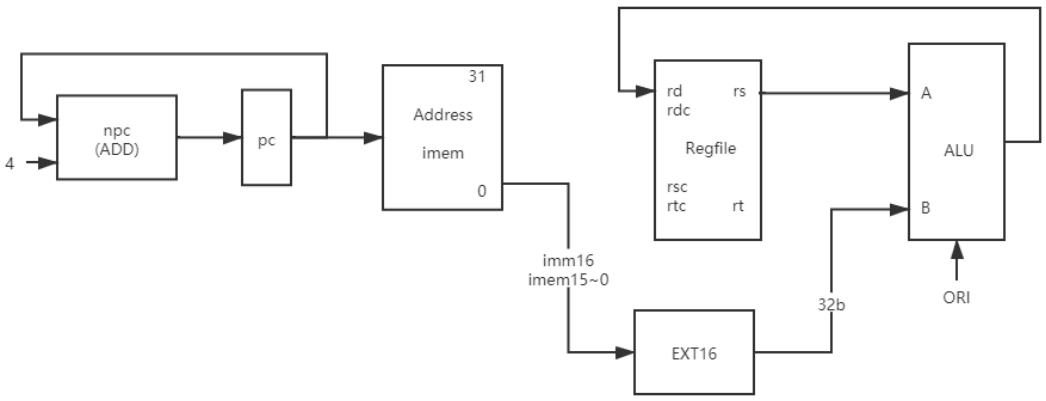
操作：取指令、rt←rs&imm16(zero_extend) 、PC←NPC(PC+4)



21.ORI

格式：ORI rt, rs, imm16

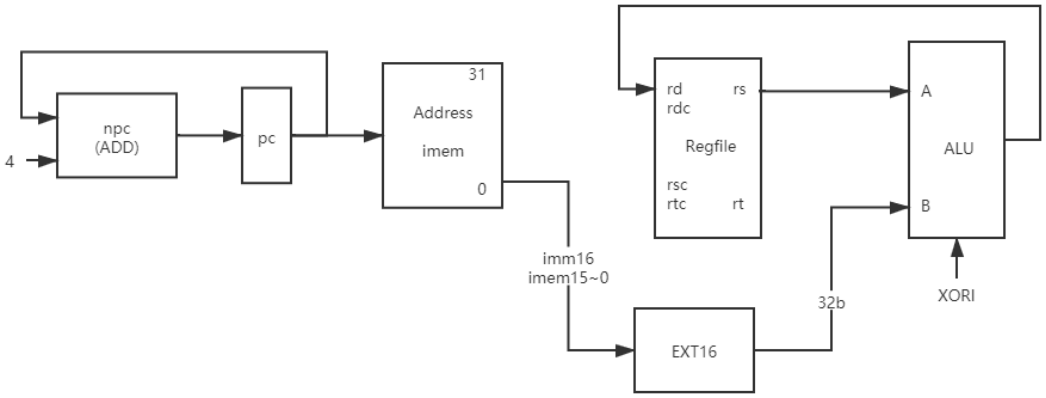
操作：取指令、rt←rs|imm16(zero_extend) 、PC←NPC(PC+4)



22.XORI

格式：XORI rt, rs, imm16

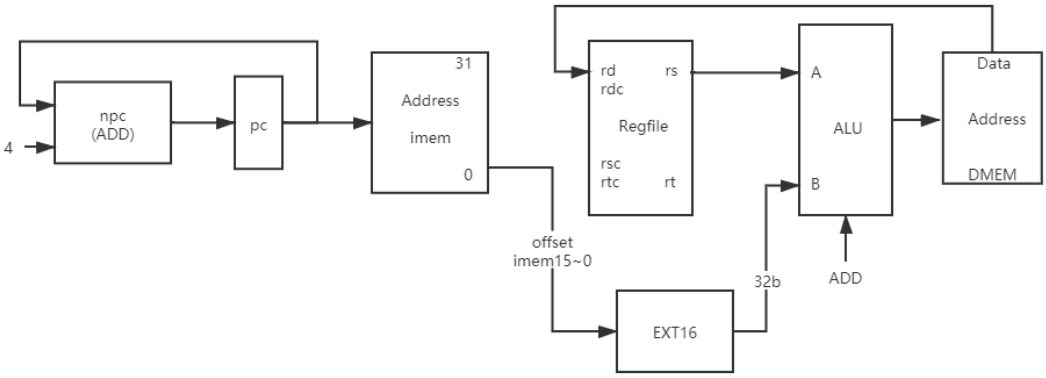
操作：取指令、rt←rs^imm16(zero_extend) 、PC←NPC(PC+4)



23.LW

格式：LW rt, offset(base)

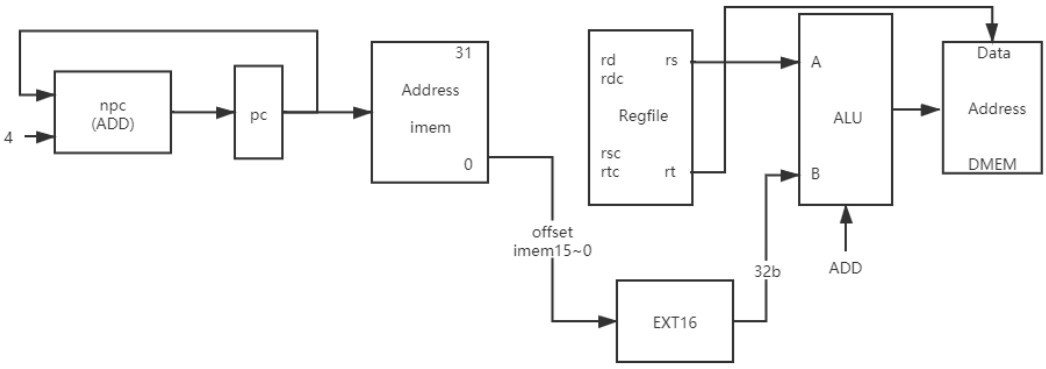
操作：取指令、rt←memory[rs + offset]、PC←NPC(PC+4)



24.SW

格式：SW rt, offset(base)

操作：取指令、memory[base + offset]←rt、PC←NPC(PC+4)

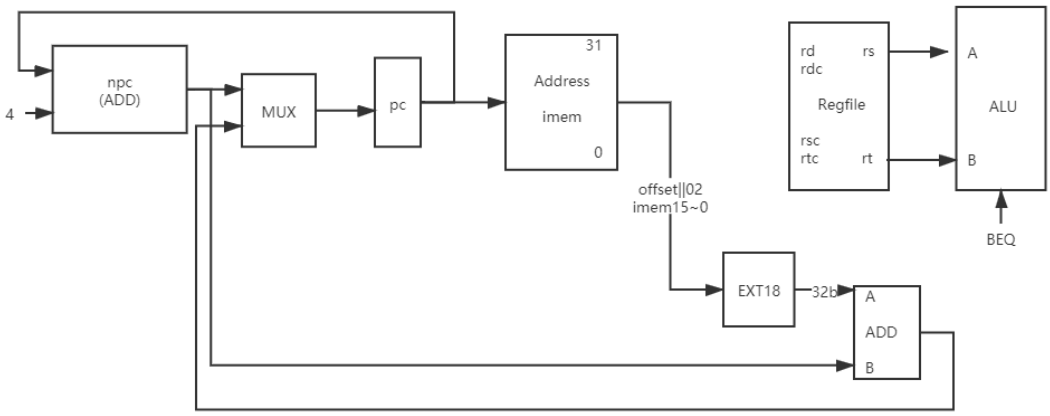


25.BEQ

格式：BEQ rs, rt, offset

操作: if (rs=rt)PC←NPC + Sign_ext(offset||02)

else PC← NPC(PC+4)

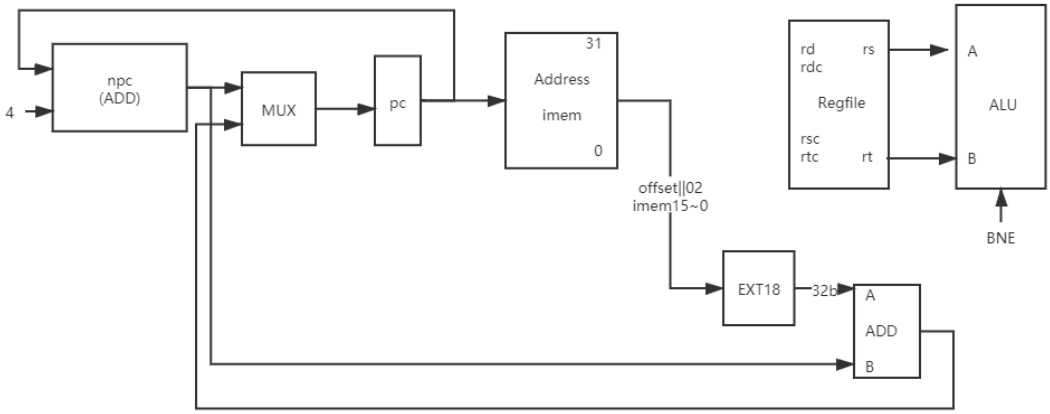


26.BNE

格式：BNE rs, rt, offset

操作：if (rs≠rt)PC←NPC + Sign_ext(offset||02)

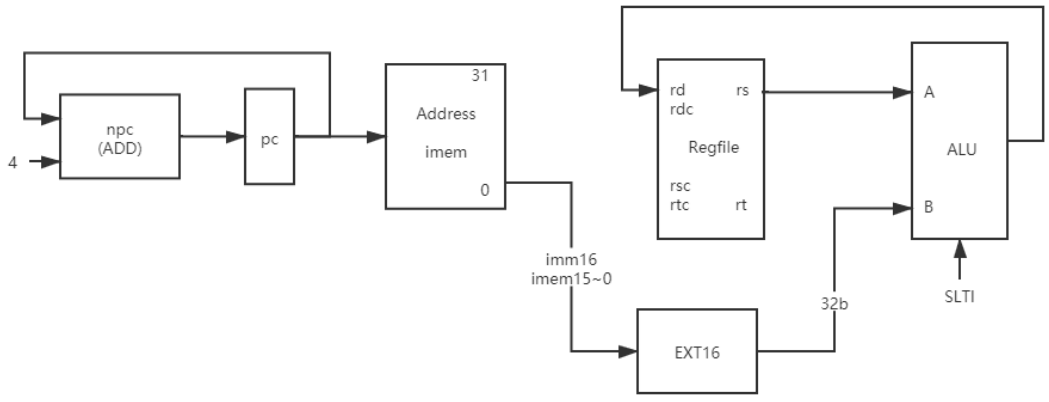
else PC← NPC(PC+4)



27.SLTI

格式：SLTI rt, rs, imm16

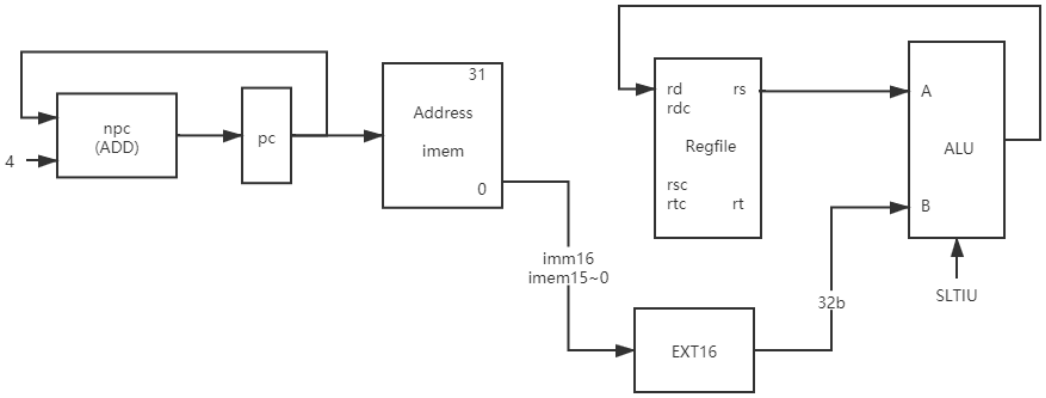
操作：取指令、rt←rs < ext.imm16(sign_extend) 、PC←NPC(PC+4)



28.SLTIU

格式：SLTI rt, rs, imm16

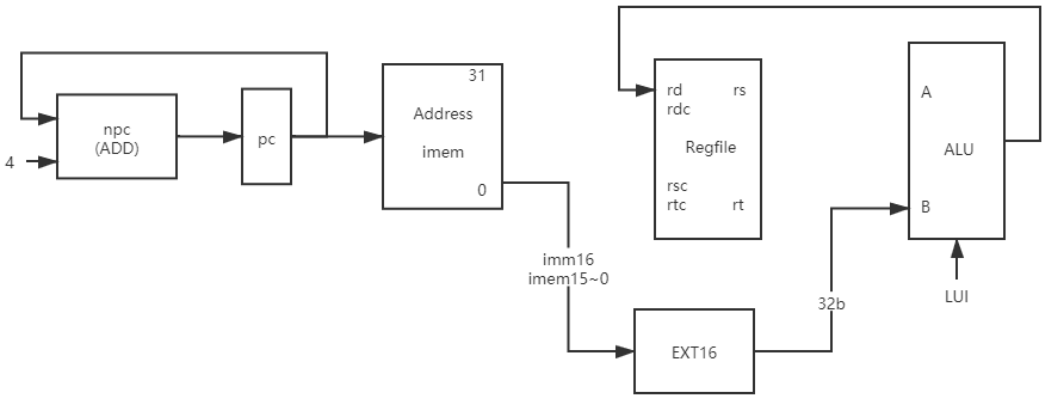
操作：取指令、rt←rs < ext.imm16(sign_extend) 、PC←NPC(PC+4)



29.LUI

格式：LUI rt, imm16

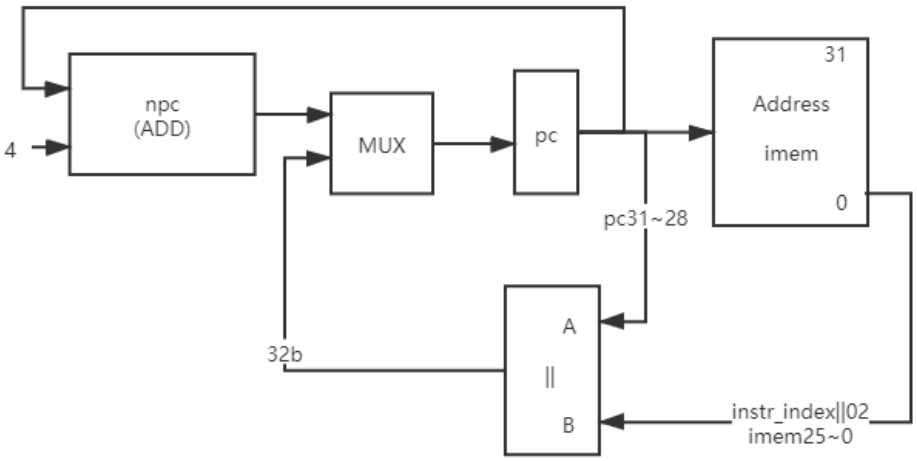
操作：取指令、rt←imm16||0^{16} 、PC←NPC(PC+4)



30.J

格式：J target

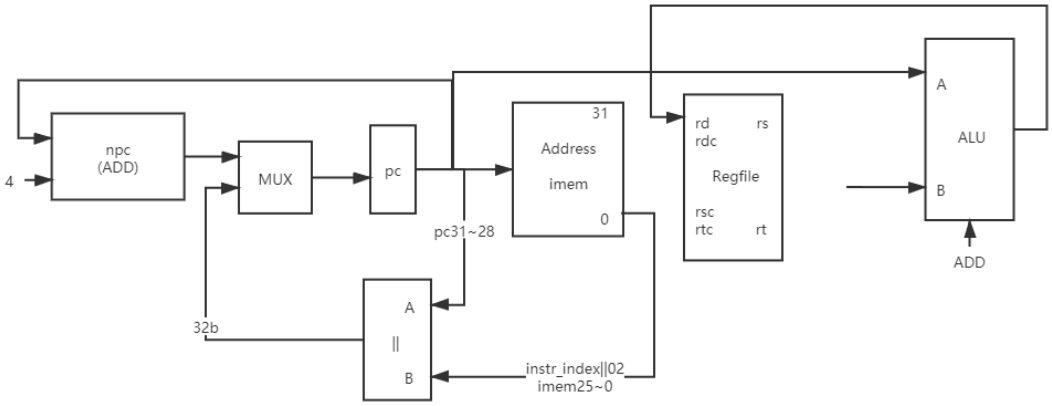
操作：取指令、PC ← $PC_{31\text{-}28}$||instr_index||0^2 , PC←NPC(PC+4)



31.JAL

格式：JAL target

操作：取指令、 R[31] ← PC + 8,PC ← $PC_{31\text{-}28}$||instr_index||0^2 , PC←PC+4

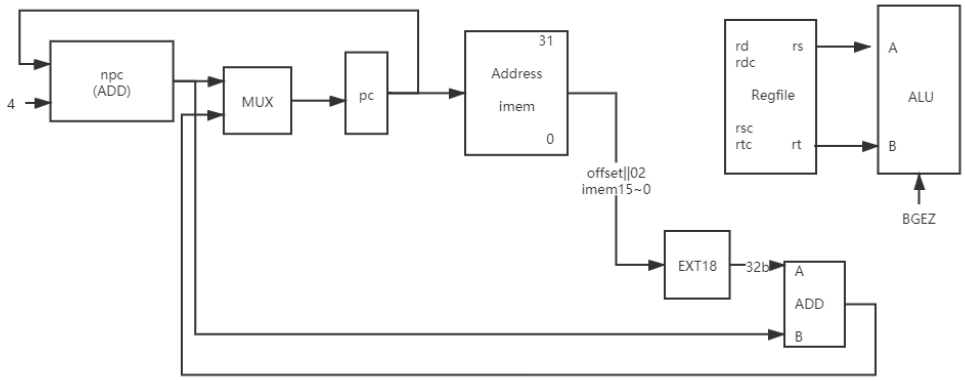


32.BGEZ

格式：BGEZ rs,offset

操作：if (rs≥0)PC←NPC + Sign_ext(offset||02)

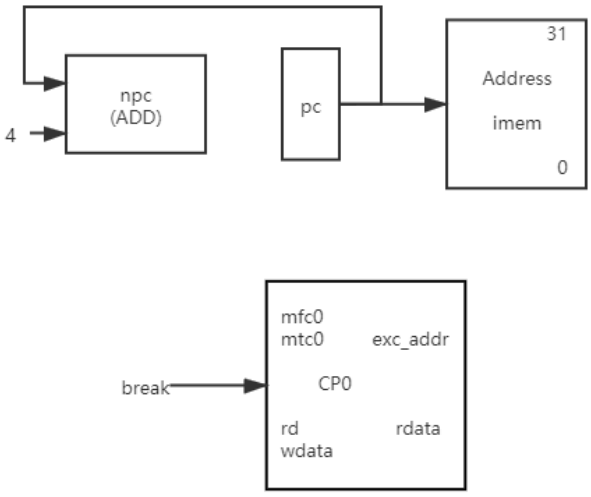
else PC← NPC(PC+4)



33.BREAK

格式：BREAK MIPS32

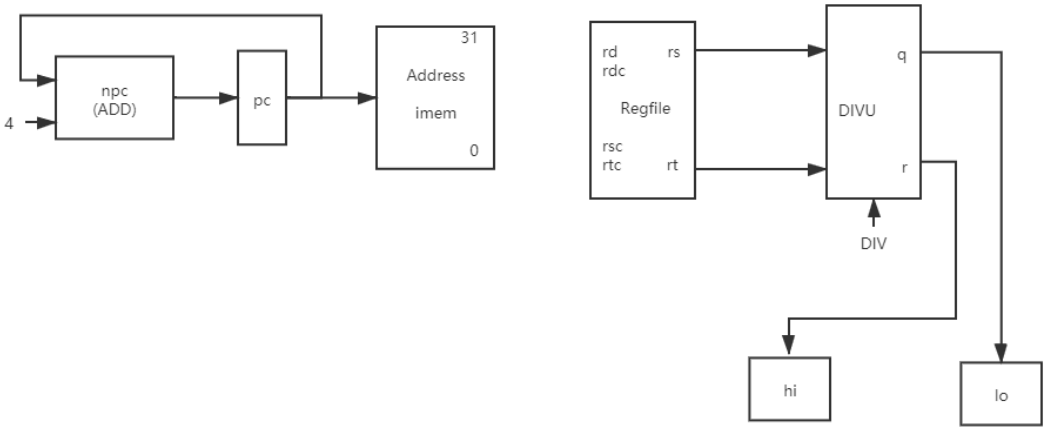
操作：当一个断电异常发生时，将立刻并且不可控制地转入异常处理。



34.DIV

格式：DIV rs, rt

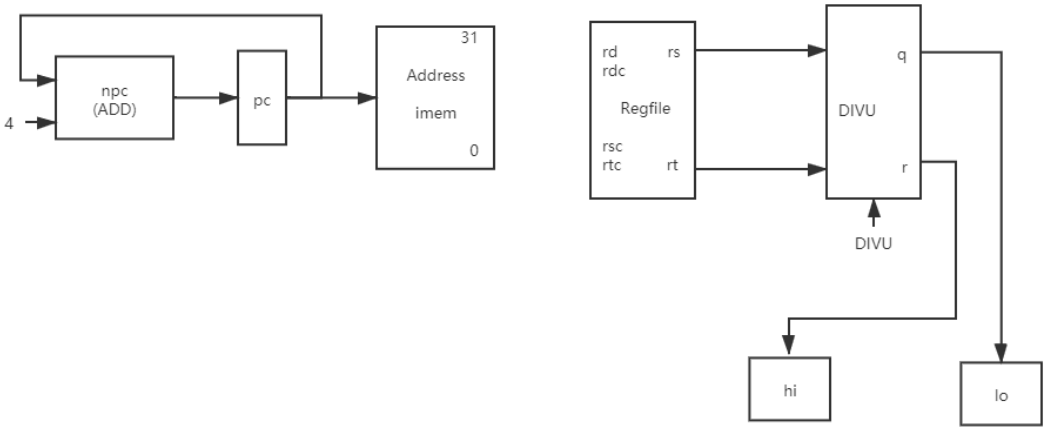
操作：(hi,lo)←rs/rt



35.DIVU

格式：DIVU rs, rt

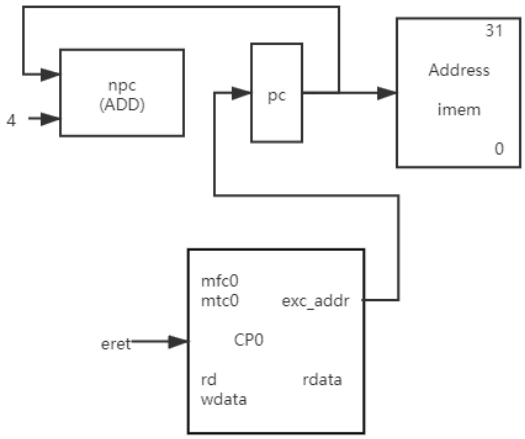
操作：(hi,lo)←rs/rt



36.ERET

格式：ERET

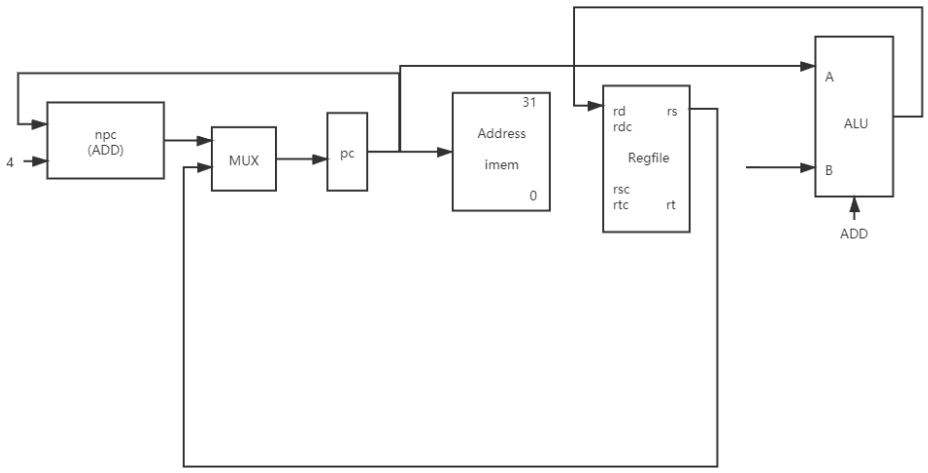
操作：在所有中断处理过程结束后返回到中断指令。



37.JALR

格式：JALR

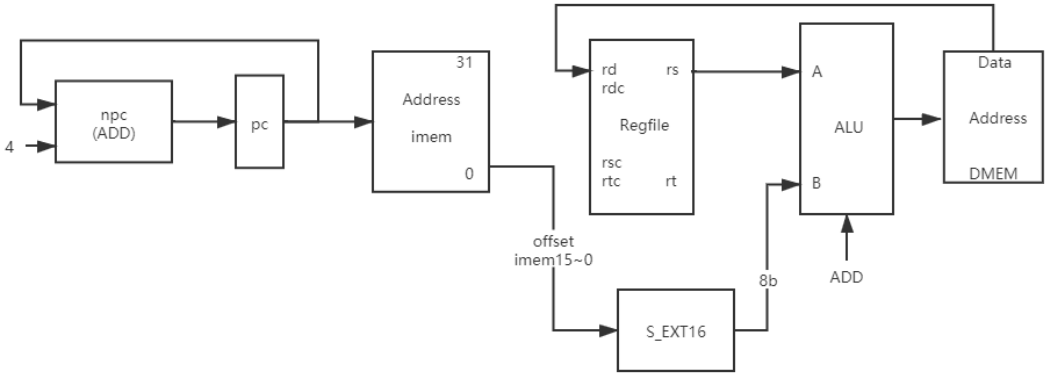
操作：rd←return_addr, pc←rs



38.LB

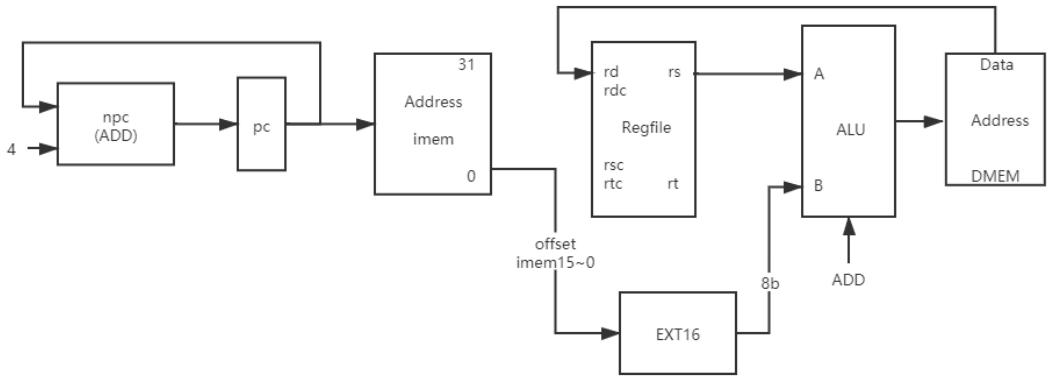
格式：LB rt, offset(base)

操作：rs←memory[base+offset]

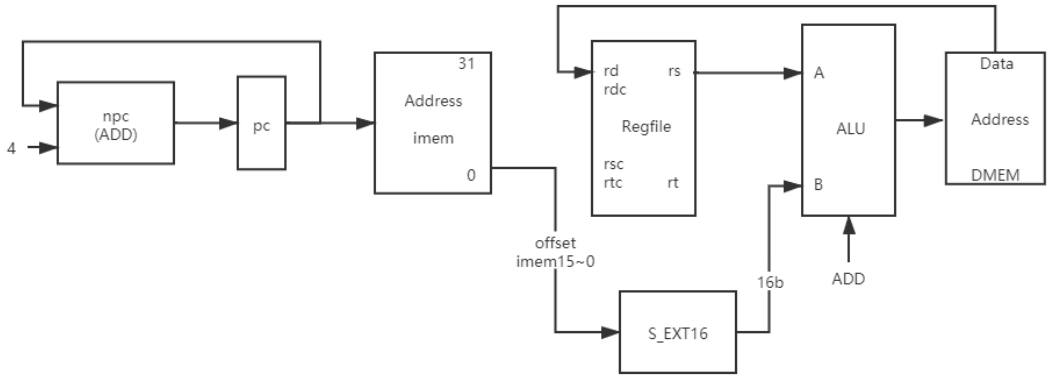


39.LBU

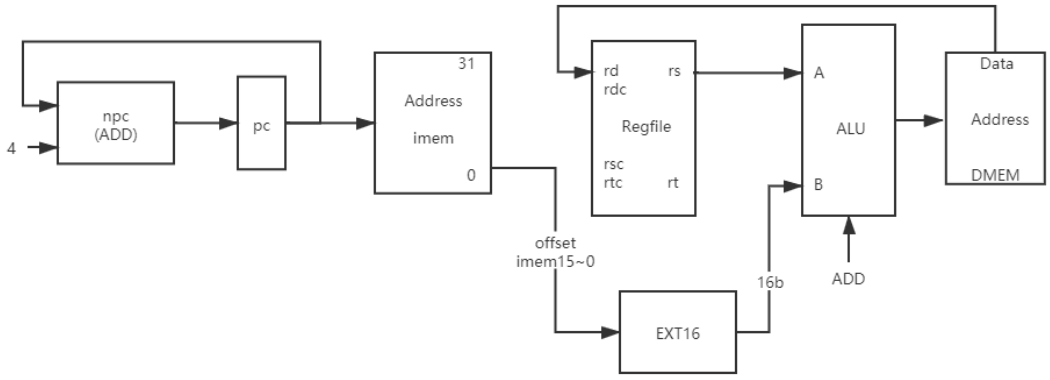
格式：LBU rt, offset(base)
操作：rs←memory[base+offset]



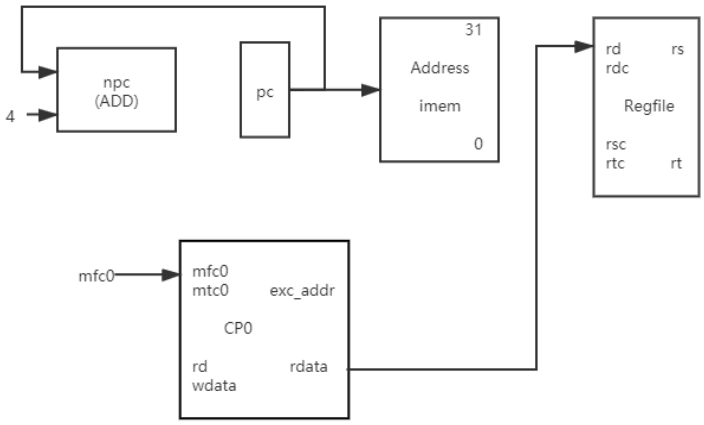
40.LH
格式：LH rt, offset(base)
操作：rs←memory[base+offset]



41.LHU
格式：LHU rt, offset(base)
操作：rs←memory[base+offset]



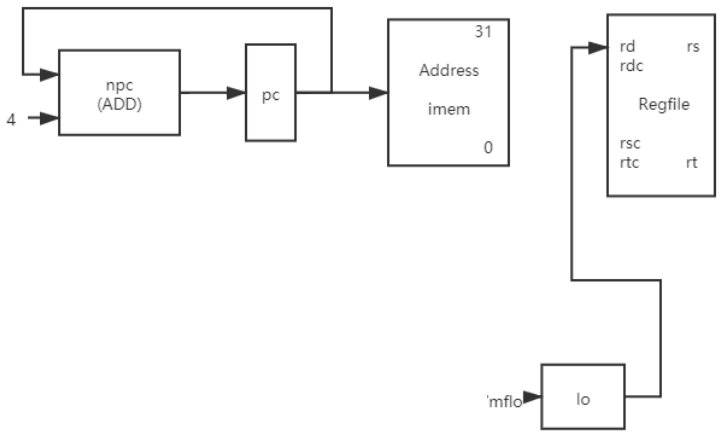
42.MFC0
格式：MFC0 rt, rd
操作：rt←CPR[0, rd, sel]



43.MFLO

格式：MFLO rd

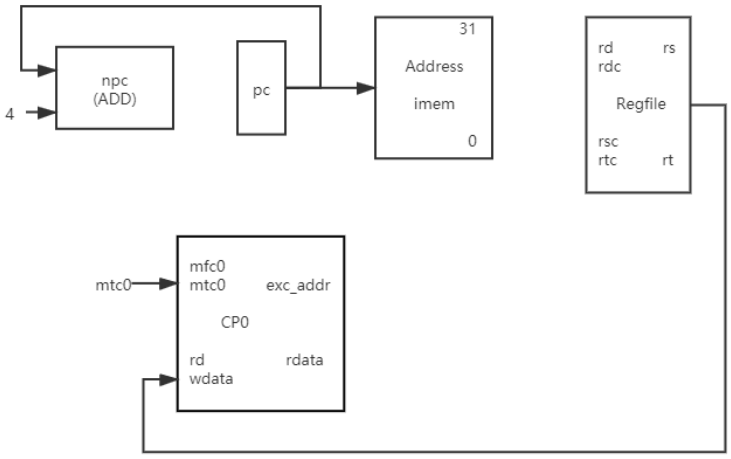
描述：rd←lo



44.MTC0

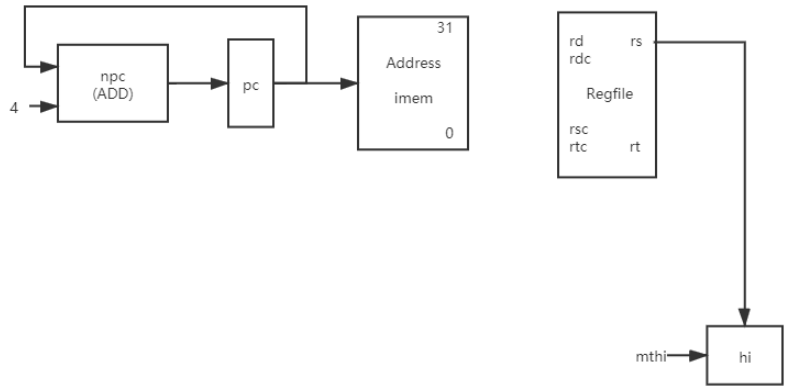
格式：MTC0 rt, rd

描述：CPR[r0, rd, sel]←rt

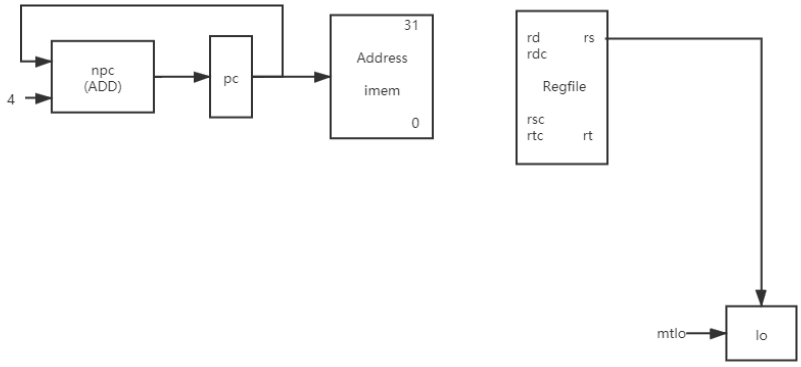


45.MTHI

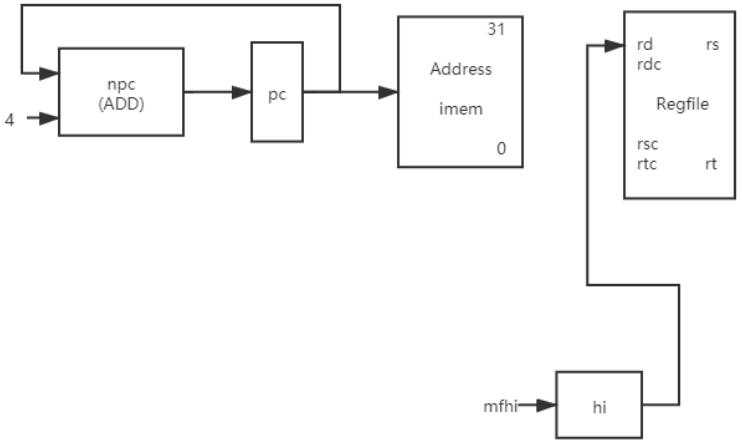
格式：MTHI rs
描述：hi←rs



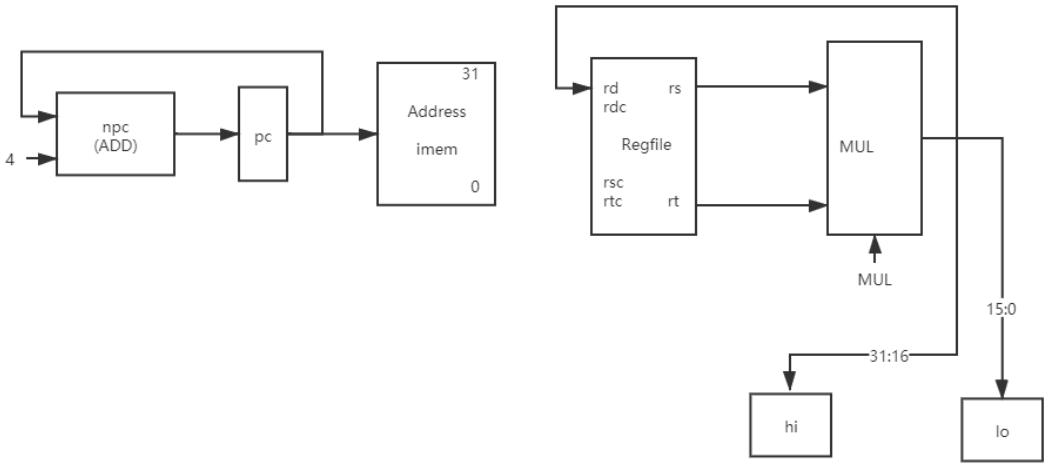
46.MTLO
格式：MTLO rs
操作：lo←rs



47.MFHI
格式：MFHI rd
操作：rd←hi



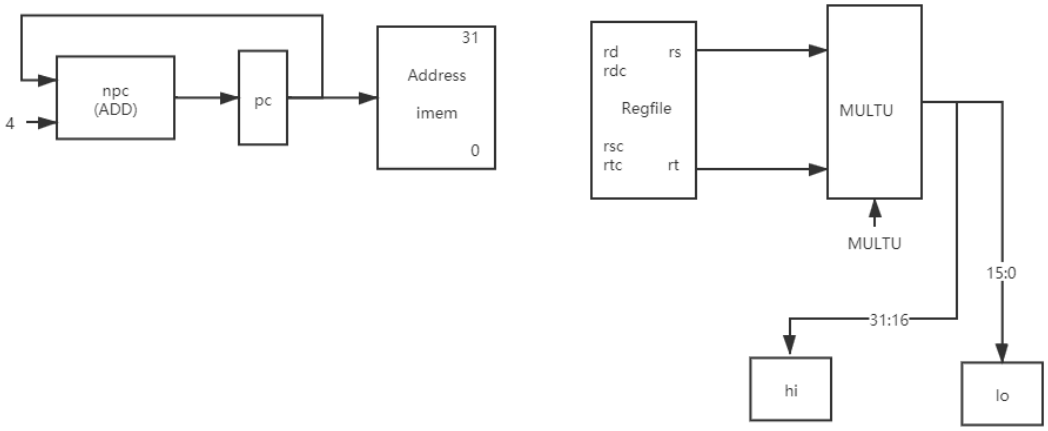
48.MUL
格式：MUL rs, rt
操作：(hi, lo)←rs*rt, rs←lo



49.MULTU

格式：MULTU rs, rt

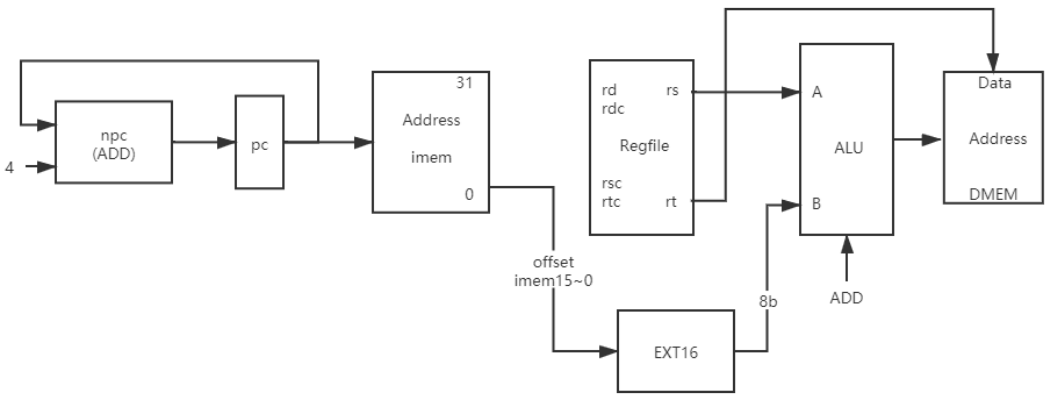
操作：(hi, lo)←rs*rt



50.SB

格式：SB←rt, offset(base)

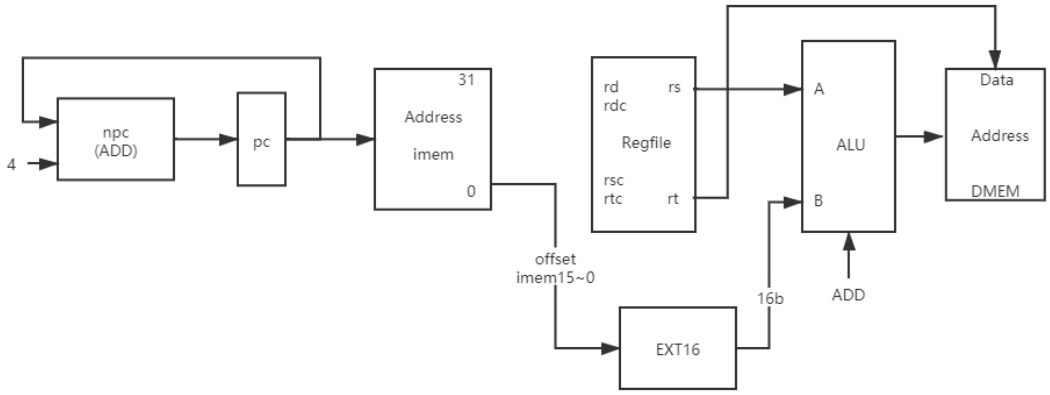
操作：memory[base+offset]←rt



51.SH

格式：SH←rt, offset(base)

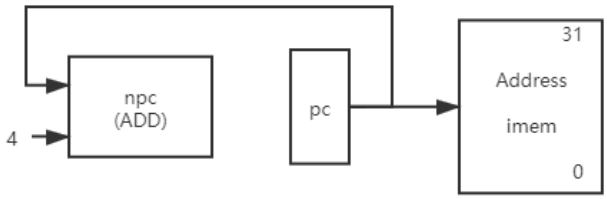
操作：memory[base+offset]←rt

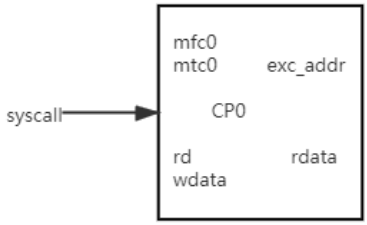


52.SYSCALL

格式：SYSCALL

操作：当一个系统调用发生时，将立刻并且不可控制地转入异常处理

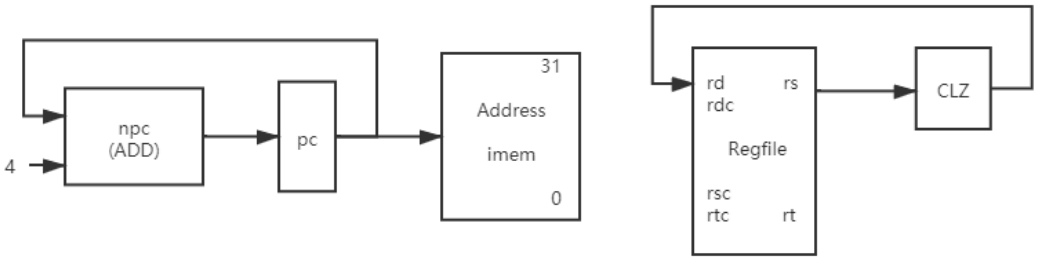




53.CLZ

格式：CLZ rd, rs

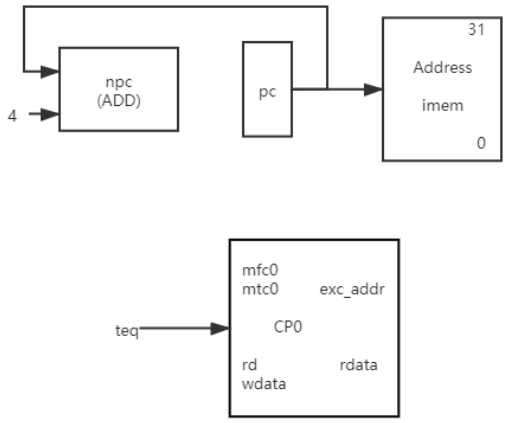
操作：计算 rs 寄存器中 32 位数前导零的个数，存入 rd 寄存器中



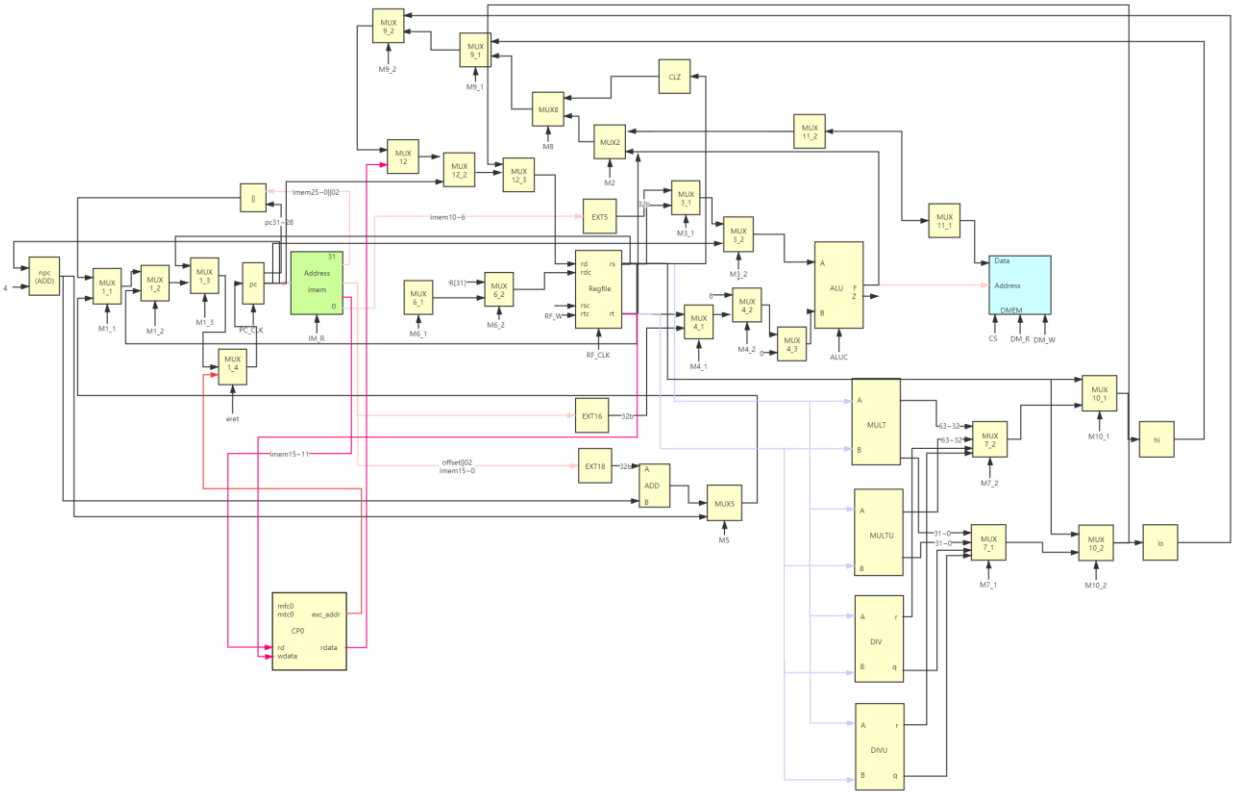
54.TEQ

格式：TEQ rs, rt

操作：比较寄存器 rs 和 rt 的值，若相等则引发一个自陷异常



总图：



三、模块建模

IMEM.v:

```
module IMEM(
    input [10:0] addr,
    output [31:0] instr
    );

    dist_mem_gen_0 imem(
            .a(addr),
```

```
                .spo(instr)
        );
endmodule
```

DMEM.v:

```
`timescale 1ns / 1ps
module DMEM(
        input clk,
        input ena,
        input rst,
        input [1:0] ssignal,
        input [2:0] lsignal,
        input write,
        input read,
        input [10:0] addr,
        input [31:0] wdata,
        output [31:0] rdata
);

        reg [7:0] data[0:1023];

        always @ (negedge clk)
        begin
                if(rst)
                begin
                data[0]<=0;
                data[1]<=0;
                data[2]<=0;
                data[3]<=0;
                data[4]<=0;
                data[5]<=0;
                data[6]<=0;
                data[7]<=0;
                data[8]<=0;
                data[9]<=0;
                data[10]<=0;
                data[11]<=0;
                data[12]<=0;
                data[13]<=0;
                end
                if (write && ena)
                begin
                if(ssignal==2'b00)
                begin
```

```
                data[addr] <= wdata[7:0];
                data[addr+1] <= wdata[15:8];
                data[addr+2] <= wdata[23:16];
                data[addr+3] <= wdata[31:24];
            end
            else if(ssignal==2'b01)
                data[addr] <=wdata[7:0];
            else if(ssignal==2'b10)
            begin
                data[addr] <= wdata[7:0];
                data[addr+1] <= wdata[15:8];
            end
            else
            ;
            end
        end

//      assign rdata = (read && ena) ?
(ssignal==2'b00?{data[addr+3],data[addr+2],data[addr+1],data[addr]}:
//
ssignal==2'b01?{8'b0,data[addr+2],data[addr+1],data[addr]} :
//                                                  ssignal==2'b10?{16'b0,data[addr+1],data[addr]}:
32'bz);
        assign rdata =((read && ena) ?
(lsignal==3'b00)?{data[addr+3],data[addr+2],data[addr+1],data[addr]}:
                                                ((lsignal==3'b01||lsignal==3'b11)?{24'b0,data[addr]}:

((lsignal==3'b10||lsignal==3'b100)?{16'b0,data[addr+1],data[addr]}:32'bz)):32'bz);
endmodule

cpu.v:
`timescale 1ns / 1ps

module cpu(
        input               clk,
        input               reset,
        input    [31:0] inst,
        input    [31:0] rdata,
        output [31:0] pc,
        output [31:0] DM_addr,
        output [31:0] DM_wdata,
        output              DM_CS,
        output              DM_R,
        output              DM_W,
```

```
        output [1:0] ssignal,
        output [2:0] lsignal
);

        wire PC_CLK;
        wire PC_ENA;
        wire M1_1,M1_2,M1_3, M2, M3_1,M3_2, M4_1,M4_2,M4_3,M5,M6_1,
M6_2,M8,M9_1,M9_2,M10_1,M10_2,M12,M12_2,M12_3;
        wire [1:0] M11_1;
        wire [2:0] M11_2;
        wire [1:0] M7;
        wire [3:0] ALUC;
        wire RF_W;
        wire RF_CLK;
        wire C_EXT16;
        wire hi_ena;
        wire lo_ena;
        wire MFC0,MTC0;



        wire [54:0] ins_code;

        wire [31:0] ALU;
        wire [31:0] PC;
        wire [31:0] RF;
        wire [31:0] Rs;
        wire [31:0] Rt;
        wire [31:0] IM;
        wire [31:0] DM;
        wire [31:0] Mux1_1;
        wire [31:0] Mux1_2;
        wire [31:0] Mux1_3;
        wire [31:0] Mux2;
        wire [31:0] Mux3_1;
        wire [31:0] Mux3_2;
        wire [31:0] Mux4_1;
        wire [31:0] Mux4_2;
        wire [31:0] Mux4_3;
        wire [31:0] Mux5;
        wire [4:0]    Mux6_1;
        wire [4:0]    Mux6_2;
        wire [31:0] Mux7_1;
        wire [31:0] Mux7_2;
```

```
        wire [31:0] Mux7;
        wire [31:0] Mux8;
        wire [31:0] Mux9_1;
        wire [31:0] Mux9_2;
        wire [31:0] Mux10_1;
        wire [31:0] Mux10_2;
        wire [31:0] Mux11_1;
        wire [31:0] Mux11_2;
        wire [31:0] Mux12;
        wire [31:0] Mux12_2;
        wire [31:0] Mux12_3;
//      wire        Mux11;

        wire [31:0] EXT1;
        wire [31:0] EXT5;
        wire [31:0] EXT16;
        wire [31:0] EXT18;
        wire [31:0] ADD;
        wire [31:0] ADD8;
        wire [31:0] NPC;
        wire [31:0] ii;
        wire [31:0] CP0_out;

        wire zero;
        wire carry;
        wire negative;
        wire overflow;
        //hi lo registers
        wire hi_w;
        wire lo_w;
        wire start,div_busy,divu_busy;
        wire [63:0] multu_result;
        wire [63:0] mult_result;
        wire [31:0] div_q;
        wire [31:0] div_r;
        wire [31:0] divu_q;
        wire [31:0] divu_r;
        wire [31:0] hi_data;
        wire [31:0] lo_data;
        wire div_ena,divu_ena;

        //clz
        wire [5:0] zeros;
        //cp0
```

```
    wire exception;
    wire eret;
    wire [4:0] cause;
    wire [31:0] status;
    wire [31:0] exc_addr;

    wire [31:0] temp_out;


    assign PC_ENA = 1;

    assign pc = PC;
    assign DM_addr = ALU;
    assign DM_wdata = Mux11_1;
    assign ssignal=M11_1;
    assign lsignal=M11_2;

    instr_decode cpu_inst (.instr_in(inst), .instr_out(ins_code));
    controller cpu_control (.clk(clk), .z(zero), .i(ins_code),.neg(negative), .PC_CLK(PC_CLK),
        .IM_R(IM_R), .M1_1(M1_1), .M1_2(M1_2),.M1_3(M1_3), .M2(M2), .M3_1(M3_1), .M3_2(M3_2), .M4_1(M4_1), .M4_2(M4_2), .M4_3(M4_3),.M5(M5), .M6_1(M6_1),.M6_2(M6_2),.M7(M7),.M8(M8),
        .M9_1(M9_1),.M9_2(M9_2),.M10_1(M10_1),.M10_2(M10_2), .M11_1(M11_1),.M11_2(M11_2),.M12(M12),.M12_2(M12_2),.M12_3(M12_3),
        .ALUC(ALUC), .RF_W(RF_W), .RF_CLK(RF_CLK), .DM_w(DM_W), .DM_r(DM_R), .DM_cs(DM_CS), .C_EXT16(C_EXT16),.hi_ena(hi_ena),.lo_ena(lo_ena),
        .div_ena(div_ena),.divu_ena(divu_ena),.MFC0(MFC0),.MTC0(MTC0),.exception(exception),.eret(eret),.cause(cause)
    );
    pcreg cpu_pc (.clk(PC_CLK), .rst(reset), .ena(PC_ENA),.data_in(Mux1_3), .data_out(PC));
    pcreg temp(.clk(PC_CLK),.rst(reset),.ena(~M12_2),.data_in(Rs),.data_out(temp_out));
    npc cpu_npc    (.in(PC), .out(NPC));
    alu                                                                    cpu_alu (.a(Mux3_2), .b(Mux4_3),.aluc(ALUC), .r(ALU),.zero(zero), .carry(carry), .negative(negative), .overflow(overflow));
    II cpu_ii(.a(PC[31:28]),.b(inst[25:0]),.out_data(ii));
    mux cpu_mux1_1(.a(ii),.b(Mux5),.choice(M1_1),.out_data(Mux1_1));
    mux cpu_mux1_2(.a(Mux1_1),.b(Rs),.choice(M1_2),.out_data(Mux1_2));
    mux cpu_mux1_3(.a(Mux1_2),.b(exc_addr),.choice(M1_3),.out_data(Mux1_3));
    mux cpu_mux2(.a(Mux11_2),.b(ALU),.choice(M2),.out_data(Mux2));
    mux cpu_mux3_1(.a(Rs),.b(EXT5),.choice(M3_1),.out_data(Mux3_1));
    mux cpu_mux3_2(.a(Mux3_1),.b(PC),.choice(M3_2),.out_data(Mux3_2));
    mux cpu_mux4_1(.a(Rt),.b(EXT16),.choice(M4_1),.out_data(Mux4_1));
    mux cpu_mux4_2(.a(Mux4_1),.b(32'd4),.choice(M4_2),.out_data(Mux4_2));
```

```
        mux cpu_mux4_3(.a(Mux4_2),.b(32'd0),.choice(M4_3),.out_data(Mux4_3));
        mux cpu_mux5(.a(NPC),.b(ADD),.choice(M5),.out_data(Mux5));
        mux_5bits cpu_mux6_1(.a(inst[15:11]),.b(inst[20:16]),.choice(M6_1),.out_data(Mux6_1));
        mux_5bits cpu_mux6_2(.a(Mux6_1),.b(5'd31),.choice(M6_2),.out_data(Mux6_2));
        mux4_1
cpu_mux7_1(.a(mult_result[31:0]),.b(multu_result[31:0]),.c(div_q),.d(divu_q),.choice(M7),.out_d
ata(Mux7_1));//connected to lo
        mux4_1
cpu_mux7_2(.a(mult_result[63:32]),.b(multu_result[63:32]),.c(div_r),.d(divu_r),.choice(M7),.out_
data(Mux7_2));//connected to hi
        mux cpu_mux8(.a(Mux2),.b({26'b0,zeros}),.choice(M8),.out_data(Mux8));
        mux cpu_mux9_1(.a(Mux8),.b(hi_data),.choice(M9_1),.out_data(Mux9_1));
        mux cpu_mux9_2(.a(Mux9_1),.b(lo_data),.choice(M9_2),.out_data(Mux9_2));
        mux cpu_mux10_1(.a(Mux7_2),.b(Rs),.choice(M10_1),.out_data(Mux10_1));
        mux cpu_mux10_2(.a(Mux7_1),.b(Rs),.choice(M10_2),.out_data(Mux10_2));
        mux4_1
cpu_mux11_1(.a(Rt),.b({24'b0,Rt[7:0]}),.c({16'b0,Rt[15:0]}),.d(32'bz),.choice(M11_1),.out_data(
Mux11_1));
        mux5_1
cpu_mux11_2(.a(rdata),.b({24'b0,rdata[7:0]}),.c({16'b0,rdata[15:0]}),.d({{24{rdata[7]}},rdata[7:
0]}),.e({{16{rdata[15]}},rdata[15:0]}),.choice(M11_2),.out_data(Mux11_2));
        mux cpu_mux12(.a(Mux9_2),.b(CP0_out),.choice(M12),.out_data(Mux12));
        mux cpu_mux12_2(.a(Mux12),.b(temp_out),.choice(M12_2),.out_data(Mux12_2));
        mux cpu_mux12_3(.a(Mux12_2),.b(Mux10_2),.choice(M12_3),.out_data(Mux12_3));
        ext5 cpu_ext5(.a(inst[10:6]),.b(EXT5));
        ext16 cpu_ext16 (.a(inst[15:0]), .sext(C_EXT16), .b(EXT16));
        ext18 cpu_ext18(.a(inst[15:0]),.b(EXT18));
        add cpu_add(.add_1(EXT18),.add_2(NPC),.out_data(ADD));
        regfile
cpu_ref(.clk(RF_CLK),.rst(reset), .we(RF_W),.Rsc(inst[25:21]), .Rtc(inst[20:16]), .Rdc(Mux6_2),
.Rs(Rs), .Rt(Rt), .Rd(Mux12_3));
        MULTU cpu_multu(.clk(clk),.reset(reset),.a(Rs),.b(Rt),.z(multu_result));
        MULT cpu_mult(.clk(clk),.reset(reset),.a(Rs),.b(Rt),.z(mult_result));
        DIVU cpu_divu(.rst(rst),.dividend(Rs),.divisor(Rt),.q(divu_q),.r(divu_r));
        DIV cpu_div(.rst(rst),.dividend(Rs),.divisor(Rt),.q(div_q),.r(div_r));
        hi_reg cpu_hi(.clk(clk),.rst(reset),.we(hi_ena),.in_data(Mux10_1),.out_data(hi_data));
        lo_reg cpu_lo(.clk(clk),.rst(reset),.we(lo_ena),.in_data(Mux10_2),.out_data(lo_data));
        clz cpu_clz(.in_data(Rs),.zeros(zeros));
        CP0
cpu_cp0(.clk(clk),.rst(reset),.mfc0(MFC0),.mtc0(MTC0),.pc(PC),.Rd(inst[15:11]),.wdata(Rt),.exc
eption(exception),.eret(eret),.cause(cause),.rdata(CP0_out),.status(status),.exc_addr(exc_addr));

endmodule
```

instr_decode.v:

```
`timescale 1ns / 1ps

module instr_decode(
        input [31:0] instr_in,
        output reg [54:0] instr_out
        );
        parameter _add    =55'b0000000000000000000000000000000000000000000000000000000001;
        parameter _addu =55'b0000000000000000000000000000000000000000000000000000000010;
        parameter _sub    =55'b0000000000000000000000000000000000000000000000000000000100;
        parameter _subu =55'b0000000000000000000000000000000000000000000000000000001000;
        parameter _and    =55'b0000000000000000000000000000000000000000000000000000010000;
        parameter _or     =55'b0000000000000000000000000000000000000000000000000000100000;
        parameter _xor    =55'b0000000000000000000000000000000000000000000000000001000000;
        parameter _nor    =55'b0000000000000000000000000000000000000000000000000010000000;
        parameter _slt    =55'b0000000000000000000000000000000000000000000000000100000000;
        parameter _sltu =55'b0000000000000000000000000000000000000000000000001000000000;
        parameter _sll    =55'b0000000000000000000000000000000000000000000000010000000000;
        parameter _srl    =55'b0000000000000000000000000000000000000000000000100000000000;
        parameter _sra    =55'b0000000000000000000000000000000000000000000001000000000000;
        parameter _sllv =55'b0000000000000000000000000000000000000000000010000000000000;
        parameter _srlv =55'b0000000000000000000000000000000000000000000100000000000000;
        parameter _srav =55'b0000000000000000000000000000000000000000001000000000000000;
        parameter _jr     =55'b0000000000000000000000000000000000000000010000000000000000;
        parameter _addi =55'b0000000000000000000000000000000000000000100000000000000000;
        parameter _addiu=55'b0000000000000000000000000000000000000001000000000000000000;
        parameter _andi =55'b0000000000000000000000000000000000000010000000000000000000;
        parameter _ori    =55'b0000000000000000000000000000000000000100000000000000000000;
        parameter _xori =55'b0000000000000000000000000000000000001000000000000000000000;
        parameter _lw     =55'b0000000000000000000000000000000000010000000000000000000000;
        parameter _sw     =55'b0000000000000000000000000000000000100000000000000000000000;
        parameter _beq    =55'b0000000000000000000000000000000001000000000000000000000000;
        parameter _bne    =55'b0000000000000000000000000000000010000000000000000000000000;
        parameter _slti =55'b0000000000000000000000000000000100000000000000000000000000;
        parameter _sltiu=55'b0000000000000000000000000000001000000000000000000000000000;
        parameter _lui    =55'b0000000000000000000000000000010000000000000000000000000000;
        parameter _j      =55'b0000000000000000000000000000100000000000000000000000000000;
        parameter _jal    =55'b0000000000000000000000000001000000000000000000000000000000;
        parameter _clz    =55'b0000000000000000000000000010000000000000000000000000000000;
        parameter _divu =55'b0000000000000000000000000100000000000000000000000000000000;
        parameter _eret =55'b0000000000000000000000001000000000000000000000000000000000;
        parameter _jalr =55'b0000000000000000000000010000000000000000000000000000000000;
        parameter _lb     =55'b0000000000000000000001000000000000000000000000000000000000;
        parameter _lbu    =55'b0000000000000000000010000000000000000000000000000000000000;
```

```
parameter _lhu    =55'b0000000000000000100000000000000000000000000000000000000;
parameter _sb     =55'b0000000000000000100000000000000000000000000000000000000;
parameter _sh     =55'b0000000000000000100000000000000000000000000000000000000;
parameter _lh     =55'b0000000000000000100000000000000000000000000000000000000;
parameter _mfc0 =55'b0000000000000100000000000000000000000000000000000000000;
parameter _mfhi =55'b0000000000001000000000000000000000000000000000000000000;
parameter _mflo =55'b0000000000010000000000000000000000000000000000000000000;
parameter _mtc0 =55'b0000000000100000000000000000000000000000000000000000000;
parameter _mthi =55'b0000000001000000000000000000000000000000000000000000000;
parameter _mtlo =55'b0000000010000000000000000000000000000000000000000000000;
parameter _mult =55'b0000000100000000000000000000000000000000000000000000000;
parameter _multu=55'b0000001000000000000000000000000000000000000000000000000;
parameter _sysc =55'b0000010000000000000000000000000000000000000000000000000;
parameter _teq    =55'b0000100000000000000000000000000000000000000000000000000;
parameter _bgez =55'b0001000000000000000000000000000000000000000000000000000;
parameter _break=55'b0010000000000000000000000000000000000000000000000000000;
parameter _div    =55'b0100000000000000000000000000000000000000000000000000000;
parameter _mul    =55'b1000000000000000000000000000000000000000000000000000000;



always@(*)
begin
      casex({instr_in[31:21],instr_in[5:0]})
            17'b000000_?????_100000:instr_out<=_add;
            17'b000000_?????_100001:instr_out<=_addu;
            17'b000000_?????_100010:instr_out<=_sub;
            17'b000000_?????_100011:instr_out<=_subu;
            17'b000000_?????_100100:instr_out<=_and;
            17'b000000_?????_100101:instr_out<=_or;
            17'b000000_?????_100110:instr_out<=_xor;
            17'b000000_?????_100111:instr_out<=_nor;
            17'b000000_?????_101010:instr_out<=_slt;
            17'b000000_?????_101011:instr_out<=_sltu;
            17'b000000_?????_000000:instr_out<=_sll;
            17'b000000_?????_000010:instr_out<=_srl;
            17'b000000_?????_000011:instr_out<=_sra;
            17'b000000_?????_000100:instr_out<=_sllv;
            17'b000000_?????_000110:instr_out<=_srlv;
            17'b000000_?????_000111:instr_out<=_srav;
            17'b000000_?????_001000:instr_out<=_jr;
            17'b001000_?????_??????:instr_out<=_addi;
            17'b001001_?????_??????:instr_out<=_addiu;
            17'b001100_?????_??????:instr_out<=_andi;
```

```
                17'b001101_?????_??????:instr_out<=_ori;
                17'b001110_?????_??????:instr_out<=_xori;
                17'b100011_?????_??????:instr_out<=_lw;
                17'b101011_?????_??????:instr_out<=_sw;
                17'b000100_?????_??????:instr_out<=_beq;
                17'b000101_?????_??????:instr_out<=_bne;
                17'b001010_?????_??????:instr_out<=_slti;
                17'b001011_?????_??????:instr_out<=_sltiu;
                17'b001111_?????_??????:instr_out<=_lui;
                17'b000010_?????_??????:instr_out<=_j;
                17'b000011_?????_??????:instr_out<=_jal;
                17'b011100_?????_100000:instr_out<=_clz;
                17'b000000_?????_011011:instr_out<=_divu;
                17'b010000_?????_011000:instr_out<=_eret;
                17'b000000_?????_001001:instr_out<=_jalr;
                17'b100000_?????_??????:instr_out<=_lb;
                17'b100100_?????_??????:instr_out<=_lbu;
                17'b100101_?????_??????:instr_out<=_lhu;
                17'b101000_?????_??????:instr_out<=_sb;
                17'b101001_?????_??????:instr_out<=_sh;
                17'b100001_?????_??????:instr_out<=_lh;
                17'b010000_00000_000000:instr_out<=_mfc0;
                17'b000000_?????_010000:instr_out<=_mfhi;
                17'b000000_?????_010010:instr_out<=_mflo;
                17'b010000_00100_000000:instr_out<=_mtc0;
                17'b000000_?????_010001:instr_out<=_mthi;
                17'b000000_?????_010011:instr_out<=_mtlo;
                17'b000000_?????_011000:instr_out<=_mul;
                17'b000000_?????_011001:instr_out<=_multu;
                17'b000000_?????_001100:instr_out<=_sysc;
                17'b000000_?????_110100:instr_out<=_teq;
                17'b000001_?????_??????:instr_out<=_bgez;
                17'b000000_?????_001101:instr_out<=_break;
                17'b000000_?????_011010:instr_out<=_div;
                17'b011100_?????_000010:instr_out<=_mul;
                default:instr_out<=55'bz;
            endcase
        end
endmodule

controller.v:
`timescale 1ns / 1ps

module controller(
```

```
input clk,
input z,
input [54:0] i,
input neg,
output PC_CLK,
output IM_R,
output M1_1,
output M1_2,
output M1_3,
output M2,
output M3_1,
output M3_2,
output M4_1,
output M4_2,
output M4_3,
output M5,
output M6_1,
output M6_2,
output [1:0] M7,
output M8,
output M9_1,
output M9_2,
output M10_1,
output M10_2,
output [1:0] M11_1,
output [2:0] M11_2,
output M12,
output M12_2,
output M12_3,
output [3:0] ALUC,
output RF_W,
output RF_CLK,
output DM_w,
output DM_r,
output DM_cs,
output C_EXT16,
output hi_ena,
output lo_ena,
output start,
output div_ena,
output divu_ena,
output MFC0,
output MTC0,
output exception,
```

```
        output eret,
        output [4:0] cause
);
        assign _add    =i[0];
        assign _addu =i[1];
        assign _sub    =i[2];
        assign _subu =i[3];
        assign _and    =i[4];
        assign _or     =i[5];
        assign _xor    =i[6];
        assign _nor    =i[7];
        assign _slt    =i[8];
        assign _sltu =i[9];
        assign _sll    =i[10];
        assign _srl    =i[11];
        assign _sra    =i[12];
        assign _sllv =i[13];
        assign _srlv =i[14];
        assign _srav =i[15];
        assign _jr     =i[16];
        assign _addi =i[17];
        assign _addiu=i[18];
        assign _andi =i[19];
        assign _ori    =i[20];
        assign _xori =i[21];
        assign _lw     =i[22];
        assign _sw     =i[23];
        assign _beq    =i[24];
        assign _bne    =i[25];
        assign _slti =i[26];
        assign _sltiu=i[27];
        assign _lui    =i[28];
        assign _j      =i[29];
        assign _jal    =i[30];
        assign _clz    =i[31];
        assign _divu =i[32];
        assign _eret =i[33];
        assign _jalr =i[34];
        assign _lb     =i[35];
        assign _lbu    =i[36];
        assign _lhu    =i[37];
        assign _sb     =i[38];
        assign _sh     =i[39];
        assign _lh     =i[40];
```

```
assign _mfc0 =i[41];
assign _mfhi =i[42];
assign _mflo =i[43];
assign _mtc0 =i[44];
assign _mthi =i[45];
assign _mtlo =i[46];
assign _mult =i[47];
assign _multu=i[48];
assign _sysc =i[49];
assign _teq    =i[50];
assign _bgez =i[51];
assign _break=i[52];
assign _div    =i[53];
assign _mul    =i[54];


assign PC_CLK    = clk;
assign IM_R       = 1;
assign M1_1       = ~(_j |_jr |_jal|_jalr);
assign M1_2       =    _jr|_jalr;
assign M1_3       =    _eret;
assign M2         = ~(_lw|_lb|_lbu|_lh|_lhu);
assign M3_1       = _sll|_srl|_sra;//1
assign M3_2       = _jal|_jalr;
assign                M4_1                                                                  =
_lui|_addi|_addiu|_andi|_ori|_xori|_lw|_sw|_slti|_sltiu|_lb|_lbu|_lh|_lhu|_sb|_sh;//0
assign M4_2       = _jal|_jalr;
assign M4_3       = _bgez;
assign M5         =(_beq&z)|(_bne&~z)|(_bgez&~neg);
assign                M6_1                                                                  =
_lui|_addi|_addiu|_andi|_ori|_xori|_lw|_slti|_sltiu|_lb|_lbu|_lh|_lhu|_mfc0;//0
assign M6_2       =_jal|_jalr;
assign M7         =(_mult|_mul)?2'b00:(_multu?2'b01:(_div?2'b10:2'b11));
assign M8         =_clz;
assign M9_1       =_mfhi;
assign M9_2       =_mflo|_mul;
assign M10_1       =_mthi;
assign M10_2       =_mtlo;
assign M11_1       =_sw?2'b00:(_sb?2'b01:(_sh?2'b10:2'b11));
assign M11_2       =_lw?3'b00:(_lbu?3'b01:(_lhu?3'b10:(_lb?3'b11:(_lh?3'b100:3'b101))));
assign M12         =_mfc0;
assign M12_2       =0;
assign M12_3       =_mul;
assign ALUC[3] =_slt|_sltu|_sll|_srl|_sra|_sllv|_srlv|_srav|_lui|_slti|_sltiu|_bgez;//1
```

```
    assign ALUC[2] =_and|_or|_xor|_nor|_sll|_srl|_sra|_sllv|_srlv|_srav|_andi|_ori|_xori;//1
    assign ALUC[1] =_add|_sub|_xor|_nor|_slt|_sltu|_sll|_sllv|_addi|_xori|_slti|_sltiu|_bgez;//1
    assign ALUC[0] =_sub|_subu|_or|_nor|_slt|_srl|_srlv|_ori|_beq|_bne|_slti|_bgez;//0
    assign RF_W=~(_jr|_sw|_beq|_bne|_j|_sb|_sh);//1
    assign RF_CLK    =clk;
    assign DM_cs=_lw|_sw|_lb|_lbu|_lh|_lhu|_sb|_sh;
    assign DM_r=_lw|_lb|_lbu|_lh|_lhu;
    assign DM_w=_sw|_sb|_sh;
    assign C_EXT16 =~(_andi|_ori|_xori|_lui);//1
    assign hi_ena=_mult|_multu|_div|_divu|_mthi|_mul;
    assign lo_ena=_mult|_multu|_div|_divu|_mtlo|_mul;
    assign div_ena=_div;
    assign divu_ena=_divu;
    assign MFC0=_mfc0;
    assign MTC0=_mtc0;
    assign exception=_break|_teq|_sysc;
    assign eret=_eret;
    assign cause=_sysc?5'b01000:(_break?5'b01001:(_teq?5'b01101:5'bz));
endmodule
```

pcreg.v:

```
`timescale 1ns / 1ps
module pcreg(
    input clk,
    input rst,
    input ena,
    input [31:0] data_in,
    output [31:0] data_out
    );

    reg [31:0] data=32'b0;

    always @(posedge clk or posedge rst)
    begin
        if(rst)
        data<=32'h00400000;
        else
        begin
            if(ena)
            data<=data_in;
        end
    end

    assign data_out=data;
```

```
endmodule
```

npc.v:

```
`timescale 1ns / 1ps
module npc(
	input [31:0] in,
	output [31:0] out
	);
	assign out=in+4;
endmodule
```

alu.v:

```
`timescale 1ns / 1ps
module alu(
	input [31:0] a,
	input [31:0] b,
	input [3:0] aluc,
	output reg [31:0] r,
	output reg zero,
	output reg carry,
	output reg negative,
	output reg overflow
	);
	reg [32:0] tempu;
	reg [32:0] tempru;
	reg signed [32:0] tempr;
	reg signed [32:0] temp;

	always@(*)
	begin
	if(aluc==4'b0000)
	begin
	tempu<=a+b;
	r<=tempu[31:0];
	if(r==0)
		zero<=1;
	else
		zero<=0;
	carry<=tempu[32];
	negative<=r[31];
	overflow<=0;
	end
```

```
else if(aluc==4'b0010)
begin
temp<=$signed(a)+$signed(b);
r<=temp[31:0];
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
if($signed(r)<0)
	negative<=1;
else
	negative<=0;
if($signed(a)<0&&$signed(b)<0)
begin
	if((-$signed(a))+(-$signed(b))>32'h7fffffff)
	overflow=1;
	else
	overflow=0;
end
else if($signed(a)>0&&$signed(b)>0)
begin
	if($signed(a)+$signed(b)>32'h7fffffff)
		overflow=1;
		else
		overflow=0;
end
else
overflow=0;
end

else if(aluc==4'b0001)
begin
tempu<=a-b;
r<=tempu[31:0];
if(r==0)
	zero<=1;
else
	zero<=0;
if(a<b)
	carry<=1;
else
	carry<=0;
negative=r[31];
```

```
overflow<=0;
end

else if(aluc==4'b0011)
begin
tempu<=a-b;
temp<=$signed(a)-$signed(b);
r<=temp[31:0];
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
if($signed(r)<0)
	negative<=1;
else
	negative<=0;
if($signed(a)<0&&$signed(b)>0)
	begin
		if((-$signed(a))+$signed(b)>32'h7fffffff)
		overflow=1;
		else
		overflow=0;
	end
	else if($signed(a)>0&&$signed(b)<0)
	begin
		if($signed(a)+(-$signed(b))>32'h7fffffff)
			overflow=1;
			else
			overflow=0;
	end
	else
	overflow=0;
end

else if(aluc==4'b0100)
begin
r<=a&b;
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
negative<=r[31];
```

```
overflow<=0;
end

else if(aluc==4'b0101)
begin
r<=a|b;
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
negative<=r[31];
overflow<=0;
end

else if(aluc==4'b0110)
begin
r<=a^b;
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
negative<=r[31];
overflow<=0;
end

else if(aluc==4'b0111)
begin
r<=~(a|b);
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
negative<=r[31];
overflow<=0;
end

else if(aluc==4'b1000||aluc==4'b1001)
begin
r<={b[15:0],16'b0};
if(r==0)
	zero<=1;
```

```
else
        zero<=0;
carry<=0;
negative<=r[31];
overflow<=0;
end

else if(aluc==4'b1011)
begin
r<=($signed(a)<$signed(b))? 1:0;
if($signed(a)==$signed(b))
        zero<=1;
else
        zero<=0;
carry<=0;
if($signed(a)<$signed(b))
        negative<=1;
else
        negative<=0;
overflow<=0;
end

else if(aluc==4'b1010)
begin
r<=(a<b)? 1:0;
if(a==b)
        zero<=1;
else
        zero<=0;
carry<=(a<b)? 1:0;
negative<=r[31];
overflow<=0;
end

else if(aluc==4'b1100)
begin
if(a>0&&a<=32)
carry<=b[a-1];
else if(a==0)
carry<=0;
else
carry<=b[31];
r<=$signed(b)>>>a;
if(r==0)
```

```
            zero<=1;
        else
            zero<=0;
        negative<=r[31];
        overflow<=0;
        end

        else if(aluc==4'b1110||aluc==4'b1111)
        begin
        tempu<=b<<a;
        r<=tempu[31:0];
        carry=tempu[32];
        if(r==0)
            zero<=1;
        else
            zero<=0;
        negative<=r[31];
        overflow<=0;
        end

        else if(aluc==4'b1101)
        begin
        if(a>0&&a<=32)
        carry<=b[a-1];
        else if(a==0)
        carry<=0;
        else
        carry<=0;
        r<=b>>a;
        if(r==0)
            zero<=1;
        else
            zero<=0;
        negative<=r[31];
        overflow<=0;
        end

        end
endmodule
```

II.v:

```
`timescale 1ns / 1ps

module II(
```

```
        input [3:0] a,
        input [25:0] b,
        output [31:0] out_data
);

        assign out_data = {a, b<<2};

endmodule
```

mux.v:

```
`timescale 1ns / 1ps
module mux(
        input [31:0] a,
        input [31:0] b,
        input choice,
        output [31:0] out_data
        );
        assign out_data=(choice)?b:a;
endmodule
```

mux_5bits.v:

```
`timescale 1ns / 1ps
module mux_5bits(
        input [4:0] a,
        input [4:0] b,
        input choice,
        output [4:0] out_data
        );
        assign out_data=(choice)?b:a;
endmodule
```

ext5.v:

```
`timescale 1ns / 1ps
module ext5 #(parameter WIDTH=5)(
        input [WIDTH-1:0] a,
        output [31:0] b
        );
        assign b={{(32-WIDTH){0}},a};
endmodule
```

ext16.v:

```
`timescale 1ns / 1ps
module ext16 #(parameter WIDTH=16)(
        input [WIDTH-1:0] a,
```

```
        input sext,
        output [31:0] b
        );
        assign b=sext?{{(32-WIDTH){a[WIDTH-1]}},a}:{27'b0,a};
endmodule
```

ext18.v:

```
`timescale 1ns / 1ps
module ext18 (
        input [15:0] a,
        output [31:0] b
        );
        assign b={{14{a[15]}},a,2'b0};
endmodule
```

add.v:

```
`timescale 1ns / 1ps
module add(
        input [31:0] add_1,
        input [31:0] add_2,
        output [31:0] out_data
        );
        assign out_data=add_1+add_2;
endmodule
```

regfile.v:

```
`timescale 1ns / 1ps

module regfile(
        input clk,
        input rst,
        input we,
        input [4:0] Rsc,
        input [4:0] Rtc,
        output [31:0] Rs,
        output [31:0] Rt,
        input [4:0] Rdc,
        input [31:0] Rd
);

        reg [31:0] array_reg[31:0];

        always @ (posedge clk or posedge rst)
        begin
```

```
        if (rst)
        begin
            array_reg[0]    <= 32'b0;
            array_reg[1]    <= 32'b0;
            array_reg[2]    <= 32'b0;
            array_reg[3]    <= 32'b0;
            array_reg[4]    <= 32'b0;
            array_reg[5]    <= 32'b0;
            array_reg[6]    <= 32'b0;
            array_reg[7]    <= 32'b0;
            array_reg[8]    <= 32'b0;
            array_reg[9]    <= 32'b0;
            array_reg[10] <= 32'b0;
            array_reg[11] <= 32'b0;
            array_reg[12] <= 32'b0;
            array_reg[13] <= 32'b0;
            array_reg[14] <= 32'b0;
            array_reg[15] <= 32'b0;
            array_reg[16] <= 32'b0;
            array_reg[17] <= 32'b0;
            array_reg[18] <= 32'b0;
            array_reg[19] <= 32'b0;
            array_reg[20] <= 32'b0;
            array_reg[21] <= 32'b0;
            array_reg[22] <= 32'b0;
            array_reg[23] <= 32'b0;
            array_reg[24] <= 32'b0;
            array_reg[25] <= 32'b0;
            array_reg[26] <= 32'b0;
            array_reg[27] <= 32'b0;
            array_reg[28] <= 32'b0;
            array_reg[29] <= 32'b0;
            array_reg[30] <= 32'b0;
            array_reg[31] <= 32'b0;
        end
        else
        begin
            if (we&& Rdc != 5'b0) begin
                array_reg[Rdc] <= Rd;
            end
        end
end

assign Rs = array_reg[Rsc];
```

```
        assign Rt = array_reg[Rtc];

endmodule
```

MULTU.v:

```
`timescale 1ns / 1ps
module MULTU(
input clk,                  // 乘法器时钟信号
input reset,
input [31:0] a, //  输入  a(被乘数)
input [31:0] b, //  输入  b(乘数  )
output [63:0] z //  乘积输出  z
);

wire [63:0] result;

wire [63:0] store0;
wire [63:0] store1;
wire [63:0] store2;
wire [63:0] store3;
wire [63:0] store4;
wire [63:0] store5;
wire [63:0] store6;
wire [63:0] store7;
wire [63:0] store8;
wire [63:0] store9;
wire [63:0] store10;
wire [63:0] store11;
wire [63:0] store12;
wire [63:0] store13;
wire [63:0] store14;
wire [63:0] store15;
wire [63:0] store16;
wire [63:0] store17;
wire [63:0] store18;
wire [63:0] store19;
wire [63:0] store20;
wire [63:0] store21;
wire [63:0] store22;
wire [63:0] store23;
wire [63:0] store24;
wire [63:0] store25;
wire [63:0] store26;
wire [63:0] store27;
```

```
wire [63:0] store28;
wire [63:0] store29;
wire [63:0] store30;
wire [63:0] store31;

wire [63:0] store0_1;
wire [63:0] store2_3;
wire [63:0] store4_5;
wire [63:0] store6_7;
wire [63:0] store8_9;
wire [63:0] store10_11;
wire [63:0] store12_13;
wire [63:0] store14_15;
wire [63:0] store16_17;
wire [63:0] store18_19;
wire [63:0] store20_21;
wire [63:0] store22_23;
wire [63:0] store24_25;
wire [63:0] store26_27;
wire [63:0] store28_29;
wire [63:0] store30_31;

wire [63:0] store0_3;
wire [63:0] store4_7;
wire [63:0] store8_11;
wire [63:0] store12_15;
wire [63:0] store16_19;
wire [63:0] store20_23;
wire [63:0] store24_27;
wire [63:0] store28_31;

wire [63:0] store0_7;
wire [63:0] store8_15;
wire [63:0] store16_23;
wire [63:0] store24_31;

wire [63:0] store0_15;
wire [63:0] store16_31;


assign store0=b[0]?{32'b0,a}:64'b0;
assign store1=b[1]?{31'b0,a,1'b0}:64'b0;
assign store2=b[2]?{30'b0,a,2'b0}:64'b0;
assign store3=b[3]?{29'b0,a,3'b0}:64'b0;
```

```
assign store4=b[4]?{28'b0,a,4'b0}:64'b0;
assign store5=b[5]?{27'b0,a,5'b0}:64'b0;
assign store6=b[6]?{26'b0,a,6'b0}:64'b0;
assign store7=b[7]?{25'b0,a,7'b0}:64'b0;
assign store8=b[8]?{24'b0,a,8'b0}:64'b0;
assign store9=b[9]?{23'b0,a,9'b0}:64'b0;
assign store10=b[10]?{22'b0,a,10'b0}:64'b0;
assign store11=b[11]?{21'b0,a,11'b0}:64'b0;
assign store12=b[12]?{20'b0,a,12'b0}:64'b0;
assign store13=b[13]?{19'b0,a,13'b0}:64'b0;
assign store14=b[14]?{18'b0,a,14'b0}:64'b0;
assign store15=b[15]?{17'b0,a,15'b0}:64'b0;
assign store16=b[16]?{16'b0,a,16'b0}:64'b0;
assign store17=b[17]?{15'b0,a,17'b0}:64'b0;
assign store18=b[18]?{14'b0,a,18'b0}:64'b0;
assign store19=b[19]?{13'b0,a,19'b0}:64'b0;
assign store20=b[20]?{12'b0,a,20'b0}:64'b0;
assign store21=b[21]?{11'b0,a,21'b0}:64'b0;
assign store22=b[22]?{10'b0,a,22'b0}:64'b0;
assign store23=b[23]?{9'b0,a,23'b0}:64'b0;
assign store24=b[24]?{8'b0,a,24'b0}:64'b0;
assign store25=b[25]?{7'b0,a,25'b0}:64'b0;
assign store26=b[26]?{6'b0,a,26'b0}:64'b0;
assign store27=b[27]?{5'b0,a,27'b0}:64'b0;
assign store28=b[28]?{4'b0,a,28'b0}:64'b0;
assign store29=b[29]?{3'b0,a,29'b0}:64'b0;
assign store30=b[30]?{2'b0,a,30'b0}:64'b0;
assign store31=b[31]?{1'b0,a,31'b0}:64'b0;

assign store0_1=store0+store1;
assign store2_3=store2+store3;
assign store4_5=store4+store5;
assign store6_7=store6+store7;
assign store8_9=store8+store9;
assign store10_11=store10+store11;
assign store12_13=store12+store13;
assign store14_15=store14+store15;
assign store16_17=store16+store17;
assign store18_19=store18+store19;
assign store20_21=store20+store21;
assign store22_23=store22+store23;
assign store24_25=store24+store25;
assign store26_27=store26+store27;
assign store28_29=store28+store29;
```

```
assign store30_31=store30+store31;
assign store0_3=store0_1+store2_3;
assign store4_7=store4_5+store6_7;
assign store8_11=store8_9+store10_11;
assign store12_15=store12_13+store14_15;
assign store16_19=store16_17+store18_19;
assign store20_23=store20_21+store22_23;
assign store24_27=store24_25+store26_27;
assign store28_31=store28_29+store30_31;

assign store0_7=store0_3+store4_7;
assign store8_15=store8_11+store12_15;
assign store16_23=store16_19+store20_23;
assign store24_31=store24_27+store28_31;

assign store0_15=store0_7+store8_15;
assign store16_31=store16_23+store24_31;

assign result=store0_15+store16_31;

assign z=result;

endmodule
```

MULT.v:

```
`timescale 1ns / 1ps
module MULT(
input clk,        // 乘法器时钟信号
input reset,
input [31:0] a, // 输入 a(被乘数)
input [31:0] b, // 输入 b(乘数)
output [63:0] z // 乘积输出 z
) ;
        wire [31:0]a_temp=a[31]?(~a+1):a;
        wire [31:0]b_temp=b[31]?(~b+1):b;
        wire [63:0]temp;

        MULTU multu_inst(.clk(clk),.reset(reset),.a(a_temp),.b(b_temp),.z(temp));

        assign z=a[31]==b[31]?temp:(~temp+1);

endmodule
```

DIVU.v:

```
`timescale 1ns / 1ps
module DIVU(
	input rst,
	input [31:0] dividend,
	input [31:0] divisor,
	output [31:0] q,
	output [31:0] r
);
	assign q=dividend/divisor;
	assign r=dividend%divisor;

endmodule
```

DIV.v:

```
`timescale 1ns / 1ps
module DIVU(
	input rst,
	input [31:0] dividend,
	input [31:0] divisor,
	output [31:0] q,
	output [31:0] r
);
	assign q=dividend/divisor;
	assign r=dividend%divisor;

endmodule
```

hi_reg.v:

```
`timescale 1ns / 1ps
module hi_reg(
	input clk,
	input rst,
	input we,
	input [31:0] in_data,
	output [31:0] out_data
	);

	reg [31:0]hi=32'b0;
	always @ (posedge clk or posedge rst)
	begin
		if (rst)
		begin
		hi<=32'b0;
		end
```

```
            else
            if(we)
            begin
            hi<=in_data;
            end
        end
        assign out_data=hi;
endmodule
```

lo_reg.v:

```
`timescale 1ns / 1ps
module lo_reg(
    input clk,
    input rst,
    input we,
    input [31:0] in_data,
    output [31:0] out_data
    );

    reg [31:0]lo;
    always @ (posedge clk or posedge rst)
    begin
        if (rst)
        begin
        lo<=32'b0;
        end
        else
        if(we)
        begin
        lo<=in_data;
        end
    end
    assign out_data=lo;
endmodule
```

clz.v:

```
`timescale 1ns / 1ps

module clz(
    input [31:0] in_data,
    output reg [5:0] zeros
    );
    always@(*)
    begin
```

```
    casex(in_data)
        32'b1???_????_????_????_????_????_????_????:zeros<=6'd0;
        32'b01??_????_????_????_????_????_????_????:zeros<=6'd1;
        32'b001?_????_????_????_????_????_????_????:zeros<=6'd2;
        32'b0001_????_????_????_????_????_????_????:zeros<=6'd3;
        32'b0000_1???_????_????_????_????_????_????:zeros<=6'd4;
        32'b0000_01??_????_????_????_????_????_????:zeros<=6'd5;
        32'b0000_001?_????_????_????_????_????_????:zeros<=6'd6;
        32'b0000_0001_????_????_????_????_????_????:zeros<=6'd7;
        32'b0000_0000_1???_????_????_????_????_????:zeros<=6'd8;
        32'b0000_0000_01??_????_????_????_????_????:zeros<=6'd9;
        32'b0000_0000_001?_????_????_????_????_????:zeros<=6'd10;
        32'b0000_0000_0001_????_????_????_????_????:zeros<=6'd11;
        32'b0000_0000_0000_1???_????_????_????_????:zeros<=6'd12;
        32'b0000_0000_0000_01??_????_????_????_????:zeros<=6'd13;
        32'b0000_0000_0000_001?_????_????_????_????:zeros<=6'd14;
        32'b0000_0000_0000_0001_????_????_????_????:zeros<=6'd15;
        32'b0000_0000_0000_0000_1???_????_????_????:zeros<=6'd16;
        32'b0000_0000_0000_0000_01??_????_????_????:zeros<=6'd17;
        32'b0000_0000_0000_0000_001?_????_????_????:zeros<=6'd18;
        32'b0000_0000_0000_0000_0001_????_????_????:zeros<=6'd19;
        32'b0000_0000_0000_0000_0000_1???_????_????:zeros<=6'd20;
        32'b0000_0000_0000_0000_0000_01??_????_????:zeros<=6'd21;
        32'b0000_0000_0000_0000_0000_001?_????_????:zeros<=6'd22;
        32'b0000_0000_0000_0000_0000_0001_????_????:zeros<=6'd23;
        32'b0000_0000_0000_0000_0000_0000_1???_????:zeros<=6'd24;
        32'b0000_0000_0000_0000_0000_0000_01??_????:zeros<=6'd25;
        32'b0000_0000_0000_0000_0000_0000_001?_????:zeros<=6'd26;
        32'b0000_0000_0000_0000_0000_0000_0001_????:zeros<=6'd27;
        32'b0000_0000_0000_0000_0000_0000_0000_1???:zeros<=6'd28;
        32'b0000_0000_0000_0000_0000_0000_0000_01??:zeros<=6'd29;
        32'b0000_0000_0000_0000_0000_0000_0000_001?:zeros<=6'd30;
        32'b0000_0000_0000_0000_0000_0000_0000_0001:zeros<=6'd31;
        32'b0000_0000_0000_0000_0000_0000_0000_0000:zeros<=6'd32;

        endcase
    end
endmodule
```

CP0.v:

```
`timescale 1ns / 1ps

module CP0(
    input clk,
```

```
input rst,
input mfc0,//读
input mtc0,//写
input [31:0] pc,
input [4:0] Rd,
input [31:0] wdata,
input exception,
input eret,
input [4:0] cause,
output [31:0] rdata,
output [31:0] status,
output [31:0] exc_addr
);

reg [31:0] array_reg [31:0];
reg [36:0] temp;
always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        array_reg[0]  <= 32'b0;
        array_reg[1]  <= 32'b0;
        array_reg[2]  <= 32'b0;
        array_reg[3]  <= 32'b0;
        array_reg[4]  <= 32'b0;
        array_reg[5]  <= 32'b0;
        array_reg[6]  <= 32'b0;
        array_reg[7]  <= 32'b0;
        array_reg[8]  <= 32'b0;
        array_reg[9]  <= 32'b0;
        array_reg[10] <= 32'b0;
        array_reg[11] <= 32'b0;
        array_reg[12] <= 32'b0;
        array_reg[13] <= 32'b0;
        array_reg[14] <= 32'b0;
        array_reg[15] <= 32'b0;
        array_reg[16] <= 32'b0;
        array_reg[17] <= 32'b0;
        array_reg[18] <= 32'b0;
        array_reg[19] <= 32'b0;
        array_reg[20] <= 32'b0;
        array_reg[21] <= 32'b0;
        array_reg[22] <= 32'b0;
        array_reg[23] <= 32'b0;
```

```
                array_reg[24] <= 32'b0;
                array_reg[25] <= 32'b0;
                array_reg[26] <= 32'b0;
                array_reg[27] <= 32'b0;
                array_reg[28] <= 32'b0;
                array_reg[29] <= 32'b0;
                array_reg[30] <= 32'b0;
                array_reg[31] <= 32'b0;
            end
            else
            begin
                if(mtc0)
                    array_reg[Rd]<=wdata;
                if(exception)
                begin
                    array_reg[12]<={array_reg[12][26:0],5'b0};
                    array_reg[13]<={25'b0,cause,2'b0};
                    array_reg[14]<=pc-4;
                end
                if(eret)
                    array_reg[12]<={5'b0,array_reg[12][31:5]};
            end
        end


        assign status=array_reg[12];
        assign exc_addr=array_reg[14];
        assign rdata=mfc0?array_reg[Rd]:32'bz;
endmodule
```

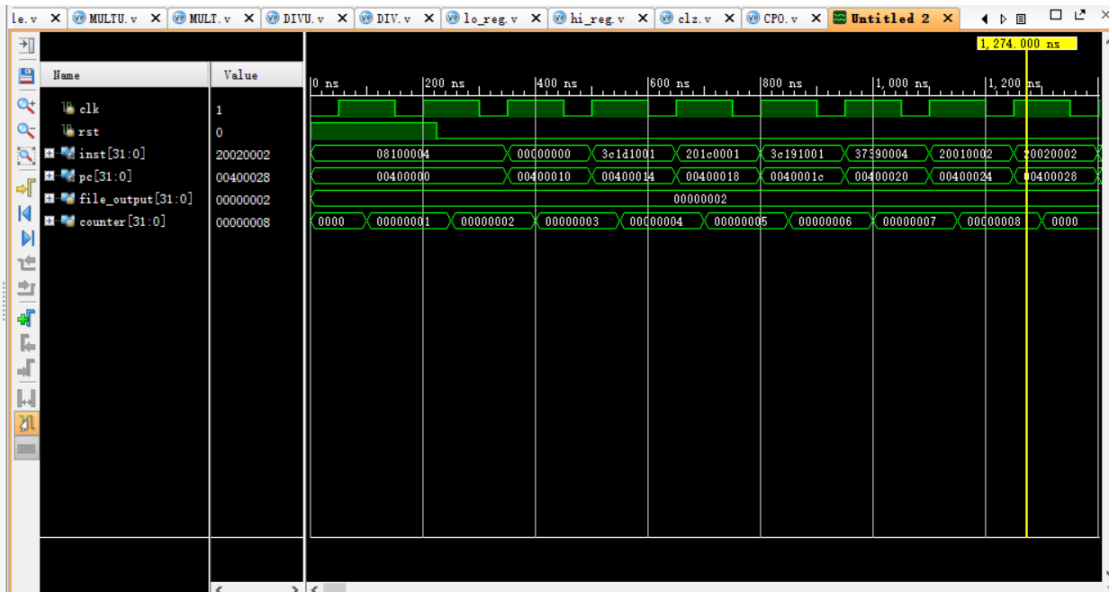
四、测试模块建模

```
 module test();
        reg clk;
        reg rst;
        wire [31:0] inst;
        wire [31:0] pc;

        integer file_output;
        integer counter = 0;


        sccomp_dataflow uut(.clk_in(clk),.reset(rst),.inst(inst),.pc(pc)//.addr(addr)
        );
```

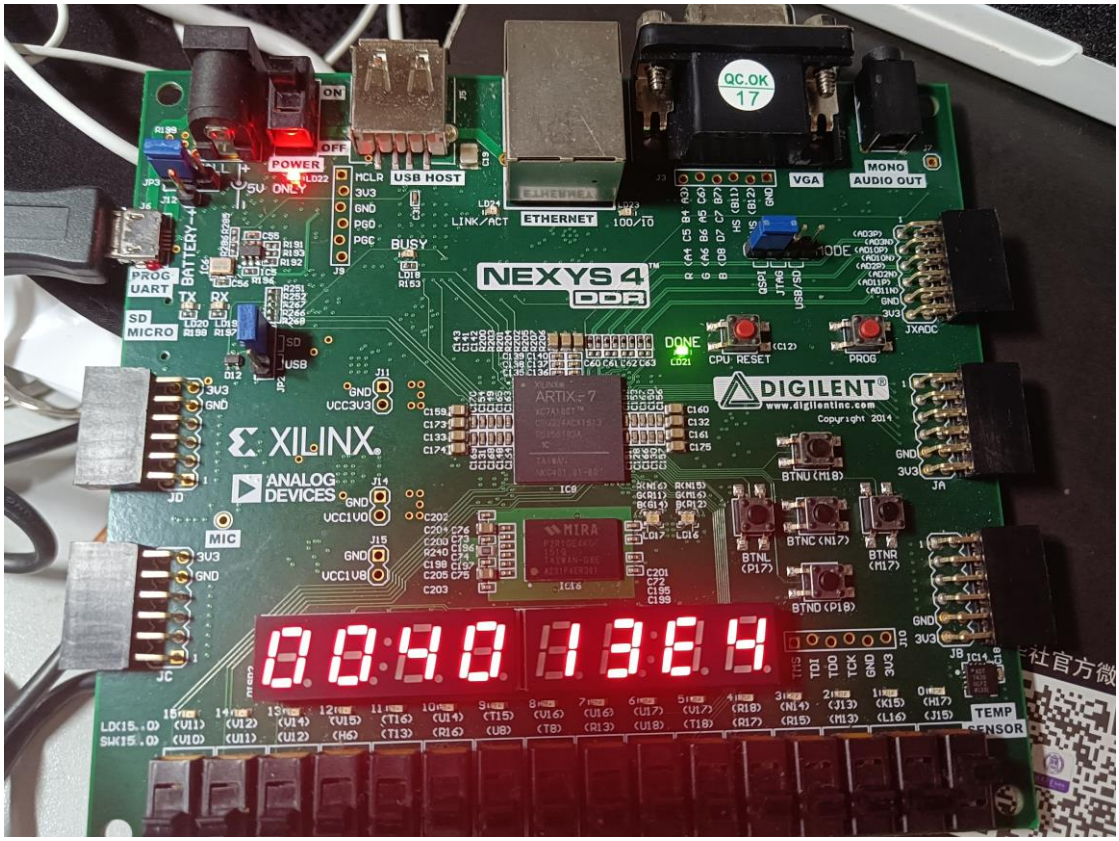
```
        initial
        begin
            file_output = $fopen("D:/result.txt");
            clk = 0;
            rst = 1;
            #225;
            rst = 0;

end

always
  begin
#50;
  clk = ~clk;
if (clk == 1'b1)
begin
if (counter == 5000)
begin
  $fclose(file_output);
end
#2
        counter = counter + 1;
        if (clk==1'b1&&rst==0)
      begin
        $fdisplay(file_output,"regfiles0 = %h",uut.sccpu.cpu_ref.array_reg[0]);
        $fdisplay(file_output,"regfiles1 = %h",uut.sccpu.cpu_ref.array_reg[1]);
        $fdisplay(file_output,"regfiles2 = %h",uut.sccpu.cpu_ref.array_reg[2]);
        $fdisplay(file_output,"regfiles3 = %h",uut.sccpu.cpu_ref.array_reg[3]);
        $fdisplay(file_output,"regfiles4 = %h",uut.sccpu.cpu_ref.array_reg[4]);
        $fdisplay(file_output,"regfiles5 = %h",uut.sccpu.cpu_ref.array_reg[5]);
        $fdisplay(file_output,"regfiles6 = %h",uut.sccpu.cpu_ref.array_reg[6]);
        $fdisplay(file_output,"regfiles7 = %h",uut.sccpu.cpu_ref.array_reg[7]);
        $fdisplay(file_output,"regfiles8 = %h",uut.sccpu.cpu_ref.array_reg[8]);
        $fdisplay(file_output,"regfiles9 = %h",uut.sccpu.cpu_ref.array_reg[9]);
        $fdisplay(file_output,"regfiles10 = %h",uut.sccpu.cpu_ref.array_reg[10]);
        $fdisplay(file_output,"regfiles11 = %h",uut.sccpu.cpu_ref.array_reg[11]);
        $fdisplay(file_output,"regfiles12 = %h",uut.sccpu.cpu_ref.array_reg[12]);
        $fdisplay(file_output,"regfiles13 = %h",uut.sccpu.cpu_ref.array_reg[13]);
        $fdisplay(file_output,"regfiles14 = %h",uut.sccpu.cpu_ref.array_reg[14]);
        $fdisplay(file_output,"regfiles15 = %h",uut.sccpu.cpu_ref.array_reg[15]);
        $fdisplay(file_output,"regfiles16 = %h",uut.sccpu.cpu_ref.array_reg[16]);
        $fdisplay(file_output,"regfiles17 = %h",uut.sccpu.cpu_ref.array_reg[17]);
```

```
            $fdisplay(file_output,"regfiles18 = %h",uut.sccpu.cpu_ref.array_reg[18]);
            $fdisplay(file_output,"regfiles19 = %h",uut.sccpu.cpu_ref.array_reg[19]);
            $fdisplay(file_output,"regfiles20 = %h",uut.sccpu.cpu_ref.array_reg[20]);
            $fdisplay(file_output,"regfiles21 = %h",uut.sccpu.cpu_ref.array_reg[21]);
            $fdisplay(file_output,"regfiles22 = %h",uut.sccpu.cpu_ref.array_reg[22]);
            $fdisplay(file_output,"regfiles23 = %h",uut.sccpu.cpu_ref.array_reg[23]);
            $fdisplay(file_output,"regfiles24 = %h",uut.sccpu.cpu_ref.array_reg[24]);
            $fdisplay(file_output,"regfiles25 = %h",uut.sccpu.cpu_ref.array_reg[25]);
            $fdisplay(file_output,"regfiles26 = %h",uut.sccpu.cpu_ref.array_reg[26]);
            $fdisplay(file_output,"regfiles27 = %h",uut.sccpu.cpu_ref.array_reg[27]);
            $fdisplay(file_output,"regfiles28 = %h",uut.sccpu.cpu_ref.array_reg[28]);
            $fdisplay(file_output,"regfiles29 = %h",uut.sccpu.cpu_ref.array_reg[29]);
            $fdisplay(file_output,"regfiles30 = %h",uut.sccpu.cpu_ref.array_reg[30]);
            $fdisplay(file_output,"regfiles31 = %h",uut.sccpu.cpu_ref.array_reg[31]);
            $fdisplay(file_output,"instr = %h",uut.inst);
            $fdisplay(file_output,"pc = %h",pc);
          end
      end
      end
endmodule
```

五、实验结果



下板结果：


NEXYS 4
DDR
DIGILENT
www.digilentinc.com
Copyright 2014
XILINX
ANALOG DEVICES
ARTIX-7
POWER
ON
OFF
USB HOST
ETHERNET
VGA
MONO AUDIO OUT
PROG UART
SD MICRO
BATTERY
MIC
TEMP SENSOR
CPU RESET
PROG
DONE
BTNU (M18)
BTNL (P17)
BTNC (N17)
BTNR (M17)
BTND (P18)
JA
JB
JC
JD
JXADC
QC.OK
17