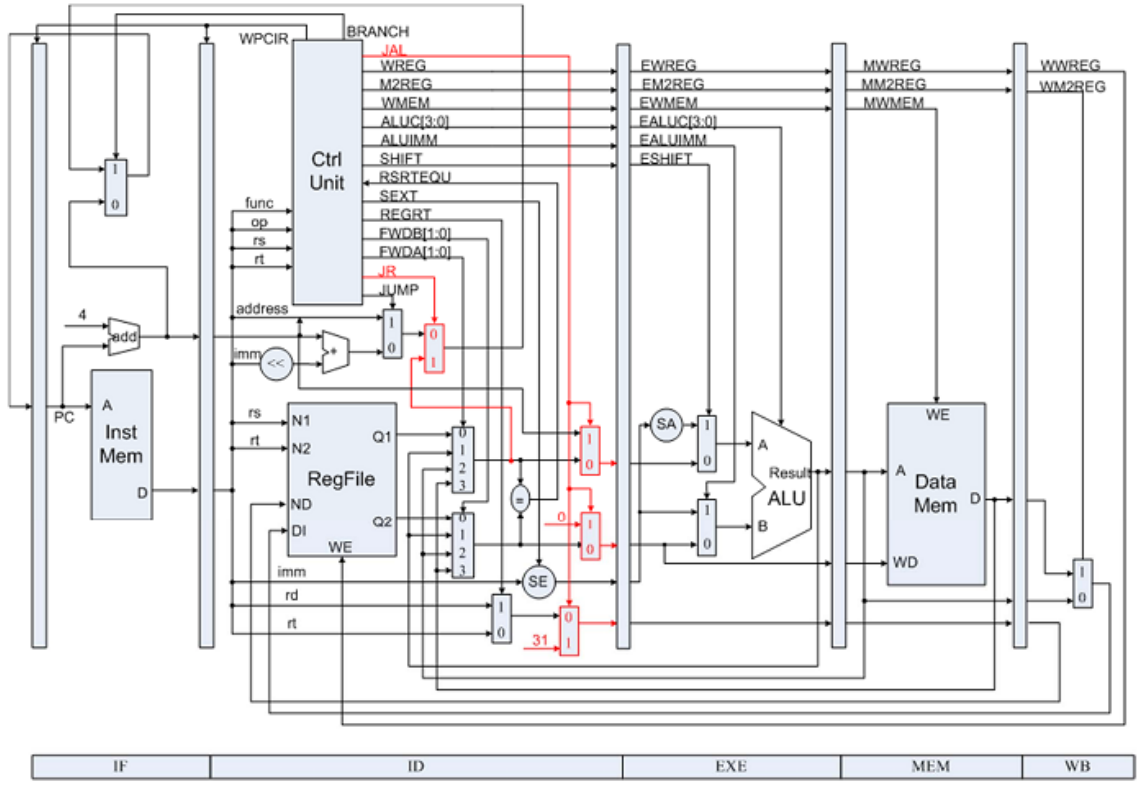
实验项目名称：Pipelined CPU support execution 31 MIPS Instructions

一、 实验目的和要求

1. Design Pipelined CPU executing 31 MIPS instructions.
   1) Design datapath.
   2) Design CPU Controller.
2. Verify the Pipelined CPU with program and observe the execution of program

二、 实验内容和原理

1. Datapath of Pipelined CPU executing 31 MIPS instructions



2. 31 MIPS instructions

MIPS Instructions								
Bit #	31..26	25..21	20..16	15..11	10..6	5..0	Operations	
R-type	op	rs	rt	rd	sa	func		
add	000000	rs	rt	rd	00000	100000	rd = rs + rt; with overflow	PC+=4
addu		rs	rt	rd	00000	100001	rd = rs + rt; without overflow	PC+=4
sub		rs	rt	rd	00000	100010	rd = rs - rt; with overflow	PC+=4
subu		rs	rt	rd	00000	100011	rd = rs - rt; without overflow	PC+=4
and		rs	rt	rd	00000	100100	rd = rs & rt;	PC+=4
or		rs	rt	rd	00000	100101	rd = rs | rt;	PC+=4
xor		rs	rt	rd	00000	100110	rd = rs ^ rt;	PC+=4
nor		rs	rt	rd	00000	100111	rd = ~(rs | rt);	PC+=4
slt		rs	rt	rd	00000	101010	if(rs < rt)rd = 1; else rd = 0; <(signed)	PC+=4
sltu		rs	rt	rd	00000	101011	if(rs < rt)rd = 1; else rd = 0; <(unsigned)	PC+=4
sll		00000	rt	rd	sa	000000	rd = rt << sa;	PC+=4
srl		00000	rt	rd	sa	000010	rd = rt >> sa (logical);	PC+=4
sra		00000	rt	rd	sa	000011	rd = rt >> sa (arithmetic);	PC+=4
sllv		rs	rt	rd	00000	000100	rd = rt << rs;	PC+=4
srlv		rs	rt	rd	00000	000110	rd = rt >> rs (logical);	PC+=4
srav		rs	rt	rd	00000	000111	rd = rt >> rs(arithmetic);	PC+=4
jr		rs	00000	00000	00000	001000		PC=rs

MIPS Instructions								
Bit #	31..26	25..21	20..16	15..11	10..6	5..0	Operations	
I-type	op	rs	rt	immediate				
addi	001000	rs	rt	imm			rt = rs + (sign_extend)imm; with overflow	PC+=4
addiu	001001	rs	rt	imm			rt = rs + (sign_extend)imm;without overflow	PC+=4
andi	001100	rs	rt	imm			rt = rs & (zero_extend)imm;	PC+=4
ori	001101	rs	rt	imm			rt = rs | (zero_extend)imm;	PC+=4
xori	001110	rs	rt	imm			rt = rs ^ (zero_extend)imm;	PC+=4
lui	001111	00000	rt	imm			rt = imm * 65536;	PC+=4
lw	100011	rs	rt	imm			rt = memory[rs + (sign_extend)imm];	PC+=4
sw	101011	rs	rt	imm			memory[rs + (sign_extend)imm] <-- rt;	PC+=4
beq	000100	rs	rt	imm			if (rs == rt) PC+=4 + (sign_extend)imm <<2;	PC+=4
bne	000101	rs	rt	imm			if (rs != rt) PC+=4 + (sign_extend)imm <<2;	PC+=4
slti	001010	rs	rt	imm			if (rs < (sign_extend)imm) rt =1 else rt = 0; less than signed	PC+=4
sltiu	001011	rs	rt	imm			if (rs < (zero_extend)imm) rt =1 else rt = 0; less than unsigned	PC+=4
J-type	op	address						
j	000010	address					PC = (PC+4)[31..28],address<<2	
jal	000011	address					PC = (PC+4)[31..28],address<<2 ; $31 = PC+4	

三、 实验过程和数据记录

1. The top module:

There’re only 4 switches to serve as the address of register, which sets the limit of register number to 16. So an additional button has to be used to expand the limit to 32. In the modified code, we use “BTN1” (ease button) and “high_reg_content” to

represent content of register from r16 to r31. Moreover, in order to show whether overflow occurs, we output the overflow signal to LED1. The overflow signal will be assigned when the calculation of EX stage finishes.

The modified code of the top module is as follows:

```
module top(input wire CCLK, BTN0_IN, BTN1_IN, BTN2_IN, BTN3_IN, input wire
[3:0]SW, output wire LED0,
output wire LED1,    //add LED1 to indicate whether overflow occurs
LCDE, LCDRS, LCDRW, output wire [3:0] LCDDAT);
    wire [31:0] high reg content;   //content of register from r16 to r31
    reg [31:0] high_reg_content_old;
    wire BTN3;                      //west: execute clock
    wire BTN2;                      //south: show high 16 bits
    wire BTN1;                      //east: show high 16 registers
    wire BTN0;                      //north: reset
    //anti-jitter
    pbdebounce M0(clk0, BTN0_IN, BTN0);
    pbdebounce M1(clk0, BTN1_IN, BTN1);
    pbdebounce M2(clk0, BTN2_IN, BTN2);
    pbdebounce M3(clk0, BTN3_IN, BTN3);
    initial begin
        //omit some codes here
        high_reg_content_old <= 32'b0;
    end
    always @(posedge CCLK  ) begin
        //real-time display the content of register
        if((BTN1 == 1'b1) || (BTN3 == 1'b1) || (BTN2 == 1'b1) || (SW_old !=
SW)  ||  (BTN0  !=  BTN_OLD)  ||  (reg_content_old  !=  reg_content)  ||
(high_reg_content_old != high_reg_content)) begin
           high_reg_content_old <= high_reg_content;
           //Press the button "BTN1" to show "high_reg_content"
           if (BTN1 == 1'b1) begin
               if (BTN0 == 1'b1)  begin
                   if (high_reg_content[15:12] < 10)
                       strdata[159:152]        <=        8'h30        +
high_reg_content[31:28];
                   else
                      strdata[159:152] <= 8'h61 + high_reg_content[31:28]
- 10;
                   if (high_reg_content[11:8] < 10)
                       strdata[151:144]        <=        8'h30        +
high_reg_content[27:24];
                   else
                       strdata[151:144]        <=        8'h61        +
high_reg_content[27:24]- 10;
                   if (high_reg_content[7:4] < 10)
                       strdata[143:136]        <=        8'h30        +
high_reg_content[23:20];
                   else
                      strdata[143:136] <= 8'h61 + high reg content[23:20]
- 10;
                   if (high_reg_content[3:0] < 10)
                       strdata[135:128]        <=        8'h30        +
high_reg_content[19:16];
                   else
                      strdata[135:128] <= 8'h61 + high_reg_content[19:16]
- 10;
               end
```

```
              else begin
                  if (high_reg_content[15:12] < 10)
                        strdata[159:152]       <=       8'h30       +
high_reg_content[15:12];
                  else
                     strdata[159:152] <= 8'h61 + high_reg_content[15:12]
- 10;
                  if (high_reg_content[11:8] < 10)
                      strdata[151:144]        <=        8'h30        +
high_reg_content[11:8];
                  else
                      strdata[151:144]        <=        8'h61        +
high_reg_content[11:8]- 10;
                  if (high_reg_content[7:4] < 10)
                      strdata[143:136] <= 8'h30 + high_reg_content[7:4];
                  else
                      strdata[143:136] <= 8'h61 + high_reg_content[7:4]
- 10;
                  if (high_reg_content[3:0] < 10)
                      strdata[135:128] <= 8'h30 + high_reg_content[3:0];
                  else
                      strdata[135:128] <= 8'h61 + high_reg_content[3:0]
- 10;
              end
          end
          //otherwise show "reg_content"
          else begin
              //omit some codes here
          end
          //omit some codes here
       end
       //omit some codes here
    end
    id_stage x_id_stage( BTN3, rst, if_inst, if_pc4, wb_destR, ex_aluR,
mem_aluR, mem_mdata, wb_dest, wb_wreg, id_wreg, id_m2reg, id_wmem,
id_aluc, id_shift, id_aluimm, id_branch, id_pc4, nid_pc, jmp_stall,
    id_a_in, id_b_in, id_imm, id_destR, id_wpcir,id_fwda, id_fwdb,
ID_ins_type,      ID_ins_number,      EX_ins_type,      EX_ins_number,
{1'b0,which_reg}, reg_content, high_reg_content);
   ex_stage x_ex_stage(BTN3, id_imm, id_a_in, id_b_in, id_wreg, id_m2reg,
id_wmem, id_aluc, id_aluimm, id_shift, id_branch, id_pc4, id_destR,
ex_wreg, ex_m2reg, ex_wmem, ex_aluR, ex_inB, ex_destR, ex_branch, ex_pc,
ex_zero, EX_ins_type, EX_ins_number, MEM_ins_type, MEM_ins_number,
LED1);
```

2. The id_stage module:

The main part of change is how to handle BNE, JAL and JR. BNE can be treated like BEQ. For JAL and JR, a stall after the instruction is necessary. In addition, attention should be paid to the forwarding signal caused by JR. Also, the signals “nid_pc” and “id_destR” needs to be modified.

The modified code of the id_stage module:

```
module id_stage (clk, rst, if_inst, if_pc4, wb_destR, ex_aluR, mem_aluR,
mem_mdata, wb_dest, wb_wreg,    cu_wreg, cu_m2reg, cu_wmem, cu_aluc,
cu_shift, cu_aluimm, cu_branch, id_pc4,nid_pc,   jmp_stall, id_a_in,
id_b_in,  id_imm,  id_destR,  cu_wpcir,id_fwda,id_fwdb,  IF_ins_type,
IF_ins_number,  ID_ins_type,  ID_ins_number,  which_reg,  reg_content,
```

```
high_reg_content);
    output [31:0] high_reg_content; //content of register from r16 to r31
    wire cu_jal;
    wire cu_jr;
    wire rsrtneq;
    //temp variables
    wire [31:0] id_a;
    wire [31:0] id_b;
    wire [31:0] nid_pc_temp;
    assign  id_a  =  (id_fwda==2'b11)?  mem_mdata:  ((id_fwda==2'b10)?
mem_aluR: ( (id_fwda==2'b01)? ex_aluR: id_inA));
    assign  id_b  =  (id_fwdb==2'b11)?  mem_mdata:  ((id_fwdb==2'b10)?
mem_aluR: ( (id_fwdb==2'b01)? ex_aluR: id_inB));
    assign id_a_in = (cu_jal)? reg_if_pc4: id_a;
    assign id_b_in = (cu_jal)? 0: id_b;

    assign rsrtneq = (id_a_in - id_b_in) ? 1: 0;

    assign nid_pc_temp =(cu_jump)? id_jump_pc: id_branch_pc;
    assign nid_pc =(cu_jr)? id_a: nid_pc_temp;

    assign id_destR_temp = (cu_regrt)? rt: rd;
    assign id_destR = (cu_jal)? 31: id_destR_temp;

    regfile  x_regfile(clk,  rst,  reg_inst[25:21],  reg_inst[20:16],
wb_destR, wb_dest, wb_wreg, rdata_A, rdata_B, which_reg, reg_content,
high_reg_content);
    ctrl_unit  x_ctrl_unit(clk,  rst,  if_inst[31:0],  reg_inst[31:0],
rsrtequ, rsrtneq, cu_branch, cu_jump, jmp_stall, cu_wreg, cu_m2reg,
cu_wmem, cu_aluc, cu_shift, cu_aluimm, cu_sext, cu_regrt, cu_wpcir,
id_fwda, id_fwdb, cu_jal, cu_jr);
```

The modified code of the ctrl_unit module:

```
module ctrl_unit(clk, rst, if_instr, instr,
    rsrtequ,
    rsrtneq,      //added signal
    cu_branch, cu_jump, jmp_stall, cu_wreg, cu_m2reg, cu_wmem, cu_aluc,
cu_shift, cu_aluimm, cu_sext, cu_regrt, cu_wpcir, cu_fwda, cu_fwdb,
    //added signals
    cu_jal,
    cu_jr);
    //rs-not-equal-to-rt signal, used as a condition of  branch
    input rsrtneq;
    output cu_jal;
    output cu_jr;
    wire branch_bne;
//If the instruction is JR and ex_rd is r13, then forwarding occurs
    assign AfromEx = ((ex_op==`OP_ALUOp) &&(ex_rd==rs) &&(ex_rd!=0))
                     || ((func==`FUNC_JR) &&(ex_rd==31));
//If the instruction is JR and the destination register of next stage is
r13(Note that JAL instruction is also this case), then forwarding occurs
   assign AfromMemALU =((mem_op==`OP_ALUOp) &&(ex_rd!=rs) &&(mem_rd==rs)
&&(mem_rd!=0))   ||   ((mem_op==`OP_JAL)   &&   (func==`FUNC_JR))   ||
((func==`FUNC_JR) && (mem_rd==31));
    //forwarding signals
    assign cu_fwda[1:0]=(AfromMemLW==1)?2'b11: ((AfromMemALU==1)?2'b10:
((AfromEx == 1)? 2'b01: 2'b00));
    assign cu_fwdb[1:0]=(BfromMemLW==1)?2'b11: ((BfromMemALU==1)?2'b10:
((BfromEx == 1)? 2'b01: 2'b00));
    //stall signal caused by JMP, JAL and JR
    assign jmp_stall = (opcode == `OP_JMP || opcode == `OP_JAL ||(opcode
```

```
== `OP_ALUOp && func ==`FUNC_JR))? 1: 0;

    assign branch_bne = (opcode == `OP_BNE)? 1: 0;
    assign cu_jump = (opcode == `OP_JMP || opcode == `OP_JAL || opcode
== `OP_ALUOp  && func == `FUNC_JR )? 1: 0;
    assign cu_branch = (branch_beq & rsrtequ)| (branch_bne & rsrtneq)|
cu_jump;
    assign cu_jal = (opcode == `OP_JAL)? 1:0;
    assign cu_jr = (opcode == `OP_ALUOp && func == `FUNC_JR)?1:0;

    assign cu_sext=  ((opcode == `OP_SW) || (opcode == `OP_LW) || (opcode
== `OP_BEQ) || (opcode == `OP_ADDI) || (opcode == `OP_BEQ) || (opcode ==
`OP_BNE) || (opcode == `OP_SLTI)) ? 1: 0; //when need to sign extend?
    assign cu_wreg = ((opcode == `OP_ALUOp) || (opcode == `OP_LW)
                    || (opcode == `OP_ADDI) || (opcode == `OP_ADDIU)
                    || (opcode == `OP_ANDI) || (opcode == `OP_ORI)
                    || (opcode == `OP_XORI) || (opcode == `OP_LUI)
                    || (opcode == `OP_SLTI) || (opcode == `OP_SLTIU)
                    || (opcode == `OP_JAL))? 1: 0; //when need to write reg?
    assign cu_aluimm = ((opcode == `OP_ADDIU)||(opcode == `OP_ADDI)
                    || (opcode == `OP_SW) || (opcode == `OP_LW)
                    || (opcode == `OP_ANDI) || (opcode == `OP_ORI)
                    || (opcode == `OP_XORI) || (opcode == `OP_LUI)
                    || (opcode == `OP_SLTI) || (opcode == `OP_SLTIU)
                    || (opcode == `OP_ADDI) || (opcode == `OP_ADDIU))? 1:0;
```

The modified code of the regfile module:

```
module regfile(clk, rst, raddr_A, raddr_B, waddr, wdata, we, rdata_A,
rdata_B, which_reg, reg_content, high_reg_content);
    output [31:0] high_reg_content; //content of register from r16 to r31
    reg [31:0] high_reg_content;
always @ (clk or which_reg) begin //when clk or which_reg is changed
    //refresh reg_content and high_reg_content
    case(which_reg)
        5'b00000: begin
            reg_content <= r0;
            high_reg_content <= r16;
        end
        5'b00001: begin
            reg_content <= r1;
            high_reg_content <= r17;
        end
        5'b00010: begin
            reg_content <= r2;
            high_reg_content <= r18;
        end
        5'b00011: begin
            reg_content <= r3;
            high_reg_content <= r19;
        end
        5'b00100: begin
            reg_content <= r4;
            high_reg_content <= r20;
        end
        5'b00101: begin
            reg_content <= r5;
            high_reg_content <= r21;
        end
        5'b00110: begin
            reg_content <= r6;
            high_reg_content <= r22;
```

```
        end
        5'b00111: begin
            reg_content <= r7;
            high_reg_content <= r23;
        end
        5'b01000: begin
            reg_content <= r8;
            high_reg_content <= r24;
        end
        5'b01001: begin
            reg_content <= r9;
            high_reg_content <= r25;
        end
        5'b01010: begin
            reg_content <= r10;
            high_reg_content <= r26;
        end
        5'b01011: begin
            reg_content <= r11;
            high_reg_content <= r27;
        end
        5'b01100: begin
            reg_content <= r12;
            high_reg_content <= r28;
        end
        5'b01101: begin
            reg_content <= r13;
            high_reg_content <= r29;
        end
        5'b01110: begin
            reg_content <= r14;
            high_reg_content <= r30;
        end
        5'b01111: begin
            reg_content <= r15;
            high_reg_content <= r31;
        end
    endcase
end
```

3. The ex_stage module:

The main change in the ex_stage module is in the Alu module. Singned and unsigned calculation is required to be handled and the overflow signal should be assigned correctly in this stage.

The modified code of the ex_stage module:

```
module ex_stage (clk,  id_imm, id_a_in, id_b_in, id_wreg, id_m2reg,
id_wmem,  id_aluc,  id_aluimm,id_shift,  id_branch,  id_pc4,id_destR,
ex_wreg, ex_m2reg, ex_wmem, ex_aluR, ex_inB, ex_destR, ex_branch, ex_pc,
ex_zero,  ID_ins_type,  ID_ins_number,  EXE_ins_type,  EXE_ins_number,
LED1);
    output LED1;  //LED1 indicates whether overflow occurs
    Alu x_Alu(a_in,b_in,ealuc,ex_aluR, LED1);
```

The modified code of the Alu module:

```
module Alu(A, B, ALUoper, Result, LED1);
    output LED1;  //LED1 indicates whether overflow occurs
    reg LED1;
```

```
    reg signed [31:0] Result_temp;
    always@(ALUoper or A or B) begin
        case(ALUoper[5:0])
           `ALU_ADD:  begin
               Result= A+B;
               LED1 = ( (~Result[31]) &A[31] &B[31] ) || ( Result[31]
&(~A[31]) &(~B[31]) );
            end
            `ALU_ADDU: begin
               Result= A+B;
               LED1 = 0;
            end
            `ALU_ADDI:    begin
               Result = A+B;
               LED1  =  ((~Result[31])  &A[31]  &B[31])  ||  (Result[31]
&(~A[31]) &(~B[31]));
            end
            `ALU_ADDIU:   begin
               Result = A+B;
               LED1 = 0;
            end
           `ALU_SUB: begin
               Result_temp = $signed(A)-$signed(B);
               Result = $unsigned(Result_temp);
               LED1 = ((~Result[31]) &A[31] &(~B[31])) || (Result[31]
&(~A[31]) &(B[31]));
            end
            `ALU_SUBU:    begin
               Result = A-B;
               LED1 = 0;
            end
           `ALU_AND:  begin
               Result = A&B;
               LED1 = 0;
            end
           `ALU_OR:   begin
               Result = A|B;
               LED1 = 0;
            end
            `ALU_ORI:     begin
               Result = A|B;
               LED1 = 0;
            end
            `ALU_XOR: begin
               Result = A^B;
               LED1 = 0;
            end
            `ALU_XORI:    begin
               Result = A^B;
               LED1 = 0;
            end
           `ALU_NOR:  begin
               Result = ~(A|B);
               LED1 = 0;
            end
           `ALU_SLT:  begin
               if(A[31]==1 && B[31]==0)
                   Result = 1;
               else if (A[31]==0 && B[31]==1)
                   Result = 0;
               else if (A[31]==0 && B[31]==0)
```

```
                Result = (A<B)?1:0;
            else
                Result = (B<A)?1:0;
            LED1 = 0;
        end
        `ALU_SLTU:    begin
            if(A[31]==1 && B[31]==0)
                Result = 0;
            else if (A[31]==0 && B[31]==1)
                Result = 1;
            else
                Result = (A<B)?1:0;
            LED1 = 0;
        end
        `ALU_SLTI:    begin
            if(A[31]==1 && B[31]==0)
                Result = 1;
            else if (A[31]==0 && B[31]==1)
                Result = 0;
            else if (A[31]==0 && B[31]==0)
                Result = (A<B)?1:0;
            else
                Result = (B<A)?1:0;
            LED1 = 0;
        end
        `ALU_SLTIU: begin
            if(A[31]==1 && B[31]==0)
                Result = 0;
            else if (A[31]==0 && B[31]==1)
                Result = 1;
            else
                Result = (A<B)?1:0;
            LED1 = 0;
        end
        `ALU_SLL:  begin
            Result = B << A;
            LED1 = 0;
        end
        `ALU_SLLV: begin
            Result = B << A;
            LED1 = 0;
        end
        `ALU_SRL:  begin
            Result = $unsigned(B)>>A;
            LED1 = 0;
        end
        `ALU_SRLV: begin
            Result = $unsigned(B)>>A;
            LED1 = 0;
        end
       `ALU_SRA:  begin
            Result = $signed(B)>>>A;
            LED1 = 0;
        end
        `ALU_SRAV: begin
            Result = $signed(B)>>>A;
            LED1 = 0;
        end
        `ALU_JR:      begin
            Result = A+B;
            LED1 = 0;
```

```
            end
            `ALU_LUI: begin
                Result = B*65536;
                LED1 = 0;
            end
            default: begin
                Result ={32{1'bx}};
                LED1 = 0;
            end
        endcase
    end
```

四、 实验结果分析

Program for verification is as follows:

MIPS code	Bin data	Address	Inst. Type
lw r1, $20(r0)	8c010014	0	6
lw r2,$21(r0)	8c020015	1	6
add r3,r1,r2	00221820	2	1
sub r4,r1,r3	00232022	3	2
sllv r6,r1,r4	00243004	4	f
srlv r7,r1,r4	00243806	5	f
srav r8,r1,r4	00244007	6	f
sll r9,r4,16	00044C00	7	c
srl r10,r4,16	00045402	8	d
sra r11,r4,16	00045C03	9	e
beq r8,r11,-6	110BFFFA	a	8
add r12,r2,r7	00476020	b	1
addu r13,r2,r7	00476821	c	f
sub r14,r7,r8	00E87022	d	2
subu r15,r7,r8	00E87823	e	f
beq r14,r15,2	11CF0002	f	8
addi r31,r31,1	23FF0001	10	f
add r31,r1,r2	0022F820	11	1
sw r7,$22(r0)	AC070016	12	7
and r5,r3,r4	00642824	13	3
xor r6,r3,r4	00643026	14	f

or r7,r3,r4	00643825	15	4
nor r8,r3,r4	00644027	16	5
slt r9,r11,r1	0161482A	17	b
sltu r10,r1,r11	002B502B	18	f
lw r8, $22(r0)	8C080016	19	6
addi r9,r8,2	21090002	1a	f
addiu r10,r8,2	250A0002	1b	f
andi r11,r8,3	310B0003	1c	f
ori r12,r8,3	350C0003	1d	f
xori r13,r8,3	390D0003	1e	f
slti r14, r11, ffff	296EFFFF	1f	f
sltiu r15, r11, ffff	2D6FFFFF	20	f
lui r14, 3	3C0E0003	21	f
bne r14, r15, 2	15CF0002	22	f
addi r31, r31, 1	23FF0001	23	f
jr r31	03E00008	24	f
j L1	08000029	25	9
addi r1,r1,1	20210001	26	f
L2: jr r31	03E00008	27	f
addi r2,r2,-1	2042FFFF	28	f
L1: jal L2	0C000027	29	f
beq r1,r31,-6	103FFFFA	2a	8
addi r31,r31,1	23FF0001	2b	f
beq r2,r31,-6	105FFFFA	2c	8

Data Mem[20]=1

Data Mem[21]=4

The result of execution is shown as follows:

Clock Count	Instruction Code	Instruction	“stage name” /number / type LED1 state	Comment
00	00000000	nop	f0fd00e00m00w00	The first instruction “lw r1, $20(r0)” enters the IF stage.
01	8c010014	lw r1, $20(r0)	f1fd06e00m00w00	The first instruction “lw r1, $20(r0)” enters the ID stage. And a new instruction “lw r2, $21(r0)” enters the IF stage.
02	8c020015	lw r2, $21(r0)	f2fd16e06m00w00	The first instruction “lw r1, $20(r0)” enters the EX stage. And a new instruction “ add r3, r1, r2” enters the IF stage.
03	00221820	stall	f3fd2fe16m06w00	The first instruction “lw r1, $20(r0)” enters the MEM stage. And the instruction “add r3, r1, r2” enters the ID stage and find that the RAW data dependency with the first instruction “lw r1, $20(r0)” on r1 is resolved by

				forwarding. However, there still exits RAW data dependency with the instruction “lw r2, $21(r0)” on r2. And the data hazard can’t be resolved by forwarding, so a stall is inserted.
04	00221820	add r3,r1,r2	f3fd21e1fm16w06	The first instruction “lw r1, $20(r0)” enters the WB stage. Now the instruction “add r3, r1, r2” can actually enter the ID stage.
05	00232022	sub r4,r1,r3	f4fd32e21m1fw16	“lw r1, $20 (r0)” completes the WB stage: r1 = 1 In addition, the instruction “lw r2, $21(r0)” completes the MEM stage, so the content of r2 is forwarding to “add r3, r1, r2”.
06	00243004	sllv r6,r1,r4	f5fd4fe32m21w1f	“lw r2, $21 (r0)” completes the WB stage: r2 = 4
07	00243806	srlv r7,r1,r4	f6fd5fe4fm32w21	The inserting stall

				completes the WB stage.
08	00244007	srav r8,r1,r4	f7fd6fe5fm4fw32	“add r3, r1, r2” completes the WB stage: r3 = 5
09	00044C00	sll r9,r4,16	f8fd7ce6fm5fw4f	“sub r4, r1, r3” completes the WB stage: r4 = 0xffff_fffc
0a	00045402	srl r10,r4,16	f9fd8de7cm6fw5f	“sllv r6, r1, r4” completes the WB stage: r6 = 0xffff_fff8
0b	00045C03	sra r11,r4,16	fafd9ee8dm7cw6f	“srlv r7, r1, r4” completes the WB stage: r7 = 0xffff_fffe
0c	110BfffA	beq r8,r11,-6	fbfda8e9em8dw7c	“srav r8, r1, r4” completes the WB stage: r8 = 0xffff_fffe
0d	00476020	add r12,r2,r7	fcfdb1ea8m9ew8d	“sll r9, r4, 16” completes the WB stage: r9 = 0xfffc_0000 In addition, the branch condition of “beq r8, r11, -6” is not taken.
0e	00476821	addu r13,r2,r7	fdfdcfeb1ma8w9e	“srl r10, r4, 16” completes the WB stage: r10 = 0x0000_ffff
0f	00E87022	sub r14,r7,r8	fefdd2ecfmb1wa8	“sra r11, r4, 16”

			LED1 lights.	completes the WB stage: r11 = 0xffff_ffff In addition, “add r12, r2, r7” completes the EX stage and the ALU detects that the destination r12 overflows (The result is 0x8000_0002).
10	00E87823	subu r15,r7,r8	fffdefed2mcfwb1	“beq r8, r11, -6” completes the WB stage.
11	11Cf0002	beq r14,r15,2	f0fdf8eefmd2wcf LED1 lights.	“add r12, r2, r7” completes the WB stage: r12 = 0x8000_0002 In addition, “sub r14, r7, r8” completes the EX stage and the ALU detects that the destination r14 overflows (The result is 0x8000_0000).
12	23ff0001	addi r31,r31,1	f2fd01ef8mefwd2	“addu r13, r2, r7” completes the WB stage: r13 = 0x8000_0002 In addition, the branch condition of “beq r14,

				r15, 2” is taken.
13	AC070016	sw r7,$22(r0)	f3fd27e01mf8wef	“sub r14, r7, r8” completes the WB stage: r14 = 0x8000_0000
14	00642824	and r5,r3,r4	f4fd33e27m01wf8	“subu r15, r7, r8” completes the WB stage: r15 = 0x8000_0000
15	00643026	xor r6,r3,r4	f5fd4fe33m27w01	“beq r14, r15, 2” completes the WB stage.
16	00643825	or r7,r3,r4	f6fd54e4fm33w27	The delay slot “addi r31, r31, 1” completes the WB stage: r31 = 1
17	00644027	nor r8,r3,r4	f7fd65e54m4fw33	“sw r7, $22 (r0)” completes the WB stage.
18	0161482A	slt r9,r11,r1	f8fd7be65m54w4f	“and r5, r3, r4” completes the WB stage: r5 = 4
19	002B502B	sltu r10,r1,r11	f9fd8fe7bm65w54	“xor r6, r3, r4” completes the WB stage: r6 = 0xffff_fff9
1a	8C080016	lw r8, $22(r0)	fafd96e8fm7bw65	“or r7, r3, r4” completes the WB stage: r7 = 0xffff_fffd
1b	21090002	stall	fbfdafe96m8fw7b	“nor r8, r3, r4” completes the WB

				stage: r8 = 2 The instruction “ addi r9, r8, 2” enters the ID stage and find that there still exits RAW data dependency with the instruction “lw r8, $22(r0)” on r8. And the data hazard can’t be resolved by forwarding, so a stall is inserted.
1c	21090002	addi r9,r8,2	fbfda1e9fm96w8f	“slt r9, r11, r1” completes the WB stage. signed number comparison: 0xffff_ffff=-1<1 So r9 = 1
1d	250A0002	addiu r10,r8,2	fcfdbfea1m9fw96	“sltu r10, r1, r11” completes the WB stage. unsigned number comparison: 1<0xffff_ffff So r10 = 1
1e	310B0003	andi r11,r8,3	fdfdc3ebfma1w9f LED1 lights.	“lw r8, $22(r0)” completes the WB stage: r8 = 7fff_fffe

				In addition, “addi r9, r8, 2” completes the EX stage and the ALU detects that the destination r9 overflows (The result is 0x8000_0000).
1f	350C0003	ori r12,r8,3	fefdd4ec3mbfwa1	The inserting stall completes the WB stage.
20	390D0003	xori r13,r8,3	fffdefed4mc3wbf	“addi r9, r8, 2” completes the WB stage: r9 = 0x8000_0000
21	296Effff	slti r14, r11, ffff	f0fdffeefmd4wc3	“addiu r10, r8, 2” completes the WB stage: r10 = 0x8000_0000
22	2D6fffff	sltiu r15, r11, ffff	f1fd0feffmefwd4	“andi r11, r8, 3” completes the WB stage: r11 =2
23	3C0E0003	lui r14, 3	f2fd1fe0fmffwef	“ori r12, r8, 3” completes the WB stage: r12 = 0x7fff_ffff
24	15Cf0002	bne r14, r15, 2	f3fd2ae1fm0fwff	“xori r13, r8, 3” completes the WB stage:

				r13 = 0xffff_fffd
25	23ff0001	addi r31, r31, 1	f5fd31e2am1fw0f	“slti r14, r11, ffff” completes the WB stage. signed number comparison: r11=2>-1=0xffff_ffff(sign_extend) So r14 = 0. In addition, the branch condition of “bne r14, r15, 2” is taken.
26	08000029	j L1	f6fd59e31m2aw1f	“sltiu r15, r11, ffff” completes the WB stage. unsigned number comparison: r11=2<0x0000_ffff(zero_extend) So r15 = 1.
27	20210001	stall	f9fd6fe59m31w2a	“lui r14, 3” completes the WB stage: r14 = 0x0003_0000 The instruction entering the IF stage is the jump target of L1 (PC =0x29).
28	0C000027	L1: jal L2	fafd9fe5fm59w31	“bne r14, r15, 2” completes the WB

				stage.
29	103ffffA	stall	f7fdafe9fm5fw59	The delay slot “addi r31, r31, 1” completes the WB stage: r31 = 2. The instruction entering the IF stage is the jump target of L2 (PC =0x27).
2a	03E00008	jr r31	f8fd7fe9fm9fw5f	“j L1” completes the WB stage. And “jr r31” enters the ID stage and find that the content of r31 can be forwarded from “L1: jal L2”.
2b	2042ffff	stall	fafd8fe7fm9fw9f	The instruction entering the IF stage is the jump target (PC =0x29). The instruction entering the IF stage is the jump target of r31 (PC =0x2a). Note that 0x2a is a forwarding result.
2c	103ffffa	beq r1,r31,-6	fbfda8e7fm7fw9f	“L1: jal L2” completes the WB stage: r31 = 0x0000_002a

				(PC of JAL+4)
2d	23ff0001	addi r31,r31,1	f5fdb1ea8m7fw7f	The branch condition of “beq r1, r31, -6” is not taken.
2e	105ffffa	beq r2,r31,-6	fdfdc8eb1ma8w7f	“jr r31” completes the WB stage.
2f	00000000	nop	fefddcec8mb1wa8	The branch condition of “beq r2, r31, -6” is not taken.
30	00000000	nop	fffdecebcmc8wb1	“beq r1, r31, -6” completes the WB stage.
31	00000000	nop	f0fdfceecmdcwc8	“addi r31, r31, 1” completes the WB stage: r31 = 0x0000_002b
32	00000000	nop	f1fd0cefcmecwdc	“beq r2, r31, -6” completes the WB stage.

The result indicates that the code implements the required functions correctly.

五、 讨论与心得

Attention should be paid to the forwarding caused by JR instruction, especially when there’s a JAL instruction before it. Note that JMP, JAL and JR will insert a stall to obtain the jump target, so only if the JAL is completing EX stage and JR is entering the ID stage should the forwarding occur.