浙江大学

本科实验报告

课程名称：　　计算机体系结构

姓　　名：　　葛现隆

学　　院：　　计算机科学与技术学院

系：　　计算机科学与技术系

专　　业：　　计算机科学与技术专业

学　　号：　　3120102146

指导教师：　　姜晓红

2015 年　7　月　2　日

浙江大学实验报告

课程名称： 计算机体系结构 实验类型： 综合

实验项目名称： Lab4：pipelined CPU support 31 MIPS instructions

学生姓名： 葛现隆 专业： 计科 学号： 3120102146

同组学生姓名： None 指导老师： 姜晓红

实验地点： 曹西-301 实验日期： 2015 年 6 月 22 日

一.实验目的和要求

1 Understand the principle of CPU Interrupt andits processing procedure;

2 Understandthefunctionof CP0 coprocessor;

3 Master the design methods of pipelined CPU supporting simple interrupt;

4 master methods of program verification of Pipelined CPU supporting interrupt;

二. 实验内容和原理

The CPU circuit graph, just the same like the design in lab5. In lab6, I made follow adjustment in circuit.

1. Add a new part cp0 to record and manage interruption;

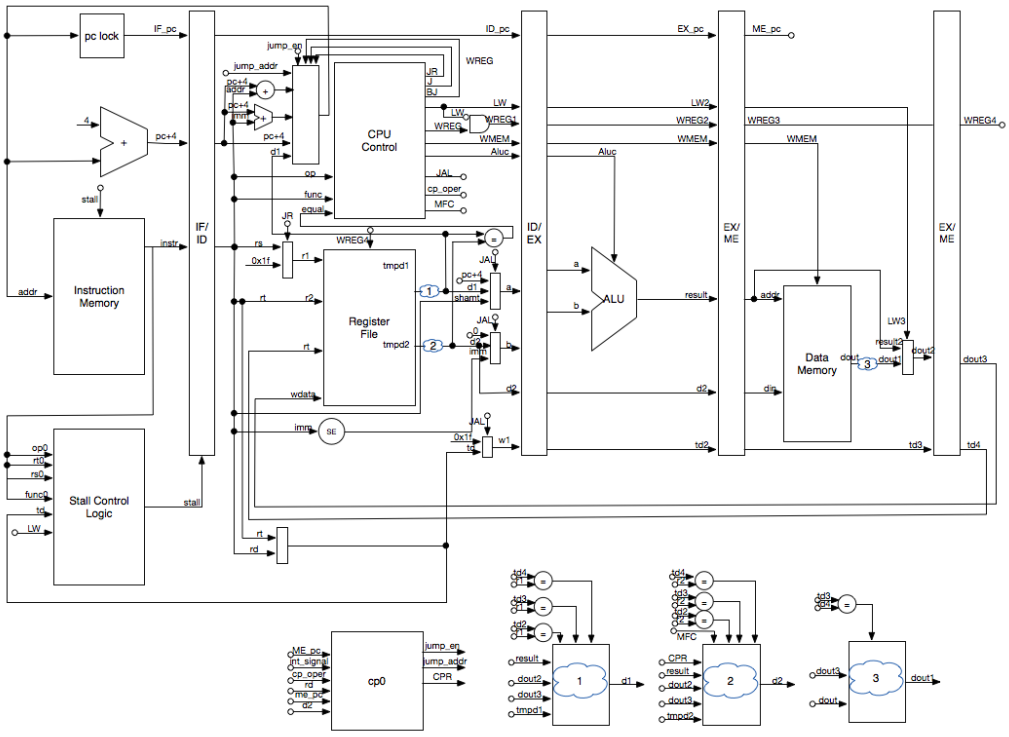
2. And a pc lock to record the clock of instruction in IF state, so that it can be passed to ID, EXE and MEM;

3. The CPU Control has two new signal, like cp_oper and MFC;

4. d2 part has a new input CPR<32> and control signal MFC;

5. the pc address multiple select has a new input jump_addr<32> and a signal jump_en;

6. the IF/ID, ID/EX, EX/ME have the ability to pass pc;



The cp0 Module:

```
module cp0(
     clk,rst,interrupt_signal,cp_oper,cp_cd,return_addr,GPR,jump_en,CPR,jump_addr

    );
     input clk,rst;
     input [1:0] interrupt_signal;
     input [1:0] cp_oper;//10 ret 01 mtc
     input [4:0] cp_cd;//read or write address rd
     input [31:0] return_addr,GPR;//write data
     output jump_en;//epc_ctrl;
     output [31:0] CPR;//read data
     output [31:0] jump_addr;//epc;
```

```
    reg [31:0] mem [31:0];

    wire inter;
    assign inter=|interrupt_signal;
    always @(negedge clk or posedge rst or posedge inter)
    begin
        if(rst)
            begin
                mem[0] <=32'b0;mem[1] <=32'b0;mem[2] <=32'b0;mem[3] <=32'b0;
                mem[4] <=32'b0;mem[5] <=32'b0;mem[6] <=32'b0;mem[7] <=32'b0;
                mem[8] <=32'b0;mem[9] <=32'b0;mem[10]<=32'b0;mem[11]<=32'b0;
                mem[12]<=32'b0;mem[13]<=32'b0;mem[14]<=32'b0;mem[15]<=32'b0;
                mem[16]<=32'b0;mem[17]<=32'b0;mem[18]<=32'b0;mem[19]<=32'b0;
                mem[20]<=32'b0;mem[21]<=32'b0;mem[22]<=32'b0;mem[23]<=32'b0;
                mem[24]<=32'b0;mem[25]<=32'b0;mem[26]<=32'b0;mem[27]<=32'b0;
                mem[28]<=32'b0;mem[29]<=32'b0;mem[30]<=32'b0;mem[31]<=32'b0;

            end
        else if(inter)
        begin
            mem[13] <= {30'b0,interrupt_signal};
            mem[14] <= return_addr;
        end
        else if(cp_oper[0])
            mem[cp_cd]<=GPR;
    end

    assign CPR = mem[cp_cd];
    assign jump_en = inter | cp_oper[1];
    assign jump_addr = (inter)?mem[1]:mem[14];//ehb=mem[1] epc = mem[14]; //R14 EPC

endmodule
```

The lock32 Module:

```
module lock32(clk,rst,in,out
    );
     input clk,rst;
     input [31:0] in;
     output reg [31:0] out;

     always @(posedge clk or posedge rst)
     begin
        if(rst)
            out<=32'b0;
        else
            out<=in;
     end

endmodule
```

The cpu_ctl Module (The whole order has changed):

```
Module cpu_ctl(
op,func,equal_result,rs,JR,J,JAL,LW,WREG,WMEM,RDorRT,SE,SA,IorR,BJ,Aluc,cp_oper,MFC
    );
    input wire [5:0] op, func;
    input wire [4:0] rs;
    input wire equal_result;
    output wire JR,J,JAL,LW,WREG,WMEM,RDorRT,SE,SA,IorR,BJ,MFC;
```

```
    output wire [4:0] Aluc;
    output wire [1:0] cp_oper;

    wire r_type, i_jr, i_sll, i_srl, i_sra;   //i_mfhi,i_mflo,i_mthi,i_mtlo;
    wire i_type, i_addi, i_addiu, i_andi, i_ori, i_xori, i_lui, i_lw, i_sw, i_slti,
i_sltiu;//i_lh,i_sh,i_mul,i_div,
    wire b_type, i_beq, i_bne;
    wire i_j, i_jal;
    wire pc_type, i_mfc, i_mtc, i_eret;

    /* R_type */
    and(r_type,~op[5],~op[4],~op[3],~op[2],~op[1],~op[0]);
    and(i_jr, r_type, ~func[5], ~func[4], func[3], ~func[2], ~func[1], ~func[0]);
    //func:001000
    and(i_sll, r_type, ~func[5], ~func[4], ~func[3], ~func[2], ~func[1], ~func[0]);
    //func:000000
    and(i_srl, r_type, ~func[5], ~func[4], ~func[3], ~func[2], func[1], ~func[0]);
    //func:000010
    and(i_sra, r_type, ~func[5], ~func[4], ~func[3], ~func[2], func[1], func[0]);
    //func:000011
//  and(i_sllv, r_type, ~func[5], ~func[4], ~func[3], func[2], ~func[1], ~func[0]);
    //func:000100
//  and(i_srlv, r_type, ~func[5], ~func[4], ~func[3], func[2], func[1], ~func[0]);
    //func:000110
//  and(i_srav, r_type, ~func[5], ~func[4], ~func[3], func[2], func[1], func[0]);
    //func:000111


    /* I_type */
    or(i_type, i_addi, i_addiu, i_andi, i_ori, i_xori, i_lui, i_lw, i_sw, i_slti,
i_sltiu, b_type );
    and(i_addi,  ~op[5],~op[4], op[3],~op[2],~op[1],~op[0]);    //001000
    and(i_addiu, ~op[5],~op[4], op[3],~op[2],~op[1], op[0]);    //001001
    and(i_andi,  ~op[5],~op[4], op[3], op[2],~op[1],~op[0]);    //001100
    and(i_ori,   ~op[5],~op[4], op[3], op[2],~op[1], op[0]);    //001101
    and(i_xori,  ~op[5],~op[4], op[3], op[2], op[1],~op[0]);    //001110
    and(i_lui,   ~op[5],~op[4], op[3], op[2], op[1], op[0]);    //001111
    and(i_lw,     op[5],~op[4],~op[3],~op[2], op[1], op[0]);    //100011
    and(i_sw,     op[5],~op[4], op[3],~op[2], op[1], op[0]);    //101011
    and(i_slti,  ~op[5],~op[4], op[3],~op[2], op[1],~op[0]);    //001010
    and(i_sltiu, ~op[5],~op[4], op[3],~op[2], op[1], op[0]);    //001011

    /* I_type(B) */
    or(b_type, i_beq, i_bne);
    and(i_beq, ~op[5],~op[4],~op[3], op[2],~op[1],~op[0]);  //000100
    and(i_bne, ~op[5],~op[4],~op[3], op[2],~op[1], op[0]);  //000101

    /* J_type */
    and(i_j, ~op[5],~op[4],~op[3],~op[2], op[1],~op[0]);//000010
    and(i_jal, ~op[5],~op[4],~op[3],~op[2], op[1], op[0]);//000011

    /* pc_type */
    and(pc_type, ~op[5],op[4],~op[3],~op[2], ~op[1],~op[0]);//010000
    and(i_mfc, pc_type, ~rs[4], ~rs[3], ~rs[2], ~rs[1], ~rs[0]);//rs 00000
    and(i_mtc, pc_type, ~rs[4], ~rs[3], rs[2], ~rs[1], ~rs[0]);//rs 00100
    and(i_eret, pc_type, rs[4], ~rs[3], ~rs[2], ~rs[1], ~rs[0]);//rs 10000 func 011000

    /* JR,J,JAL,LW,WREG,WMEM,RDorRT,SE,SA,IorR,AluCtl，BJ */
    assign JR = i_jr;
    assign J = i_j;
    assign JAL = i_jal;
    assign LW = i_lw;
    assign WREG = i_jal | (IorR & ~i_sw) | (r_type & ~i_jr) | i_mfc;//i_mtc
```

```
     assign WMEM = i_sw;
     assign RDorRT = r_type & ~i_jr;
     assign SE = i_addi | i_addiu | i_lw |i_sw | i_slti;//i_andi i_ori zero_extend
     assign SA = i_sll | i_srl | i_sra;
//   assign IR =( r_type | i_type ) & ~i_jr & ~b_type & ~i_lw & ~i_sw;
     assign IorR = i_type & ~b_type;
     alt_ctl AC(.op(op),.func(func),.aluc(Aluc));
     assign BJ = ( i_beq & equal_result ) | ( i_bne & ~equal_result );
     assign cp_oper[1] = i_eret;
     assign cp_oper[0] = i_mtc;
     assign MFC = i_mfc;

endmodule
```

The top module(The red is the changed part):

```
module top(CCLK, SW, BTNN, BTNE, BTNS, BTNW,LED,
     LCDE, LCDRS, LCDRW, LCDDAT
);
     input wire CCLK;
     input wire [3:0] SW;
     input wire BTNN,BTNE,BTNW,BTNS;
     output wire LCDE, LCDRS, LCDRW;
     output wire [3:0] LCDDAT;
     output wire [7:0] LED;

     wire btnclk;
     wire rst;
     reg sw_reg,sw_reg_ins;
     wire btsw,btreg;
     assign LED[3:0]=SW[3:0];
     assign LED[4]=btnclk;
     assign LED[5]=rst;
     assign LED[6]=btsw;
     assign LED[7]=btreg;

     reg [7:0] clk_cnt;

     wire btnbtn = btsw | btreg | btnclk ;

     always @(posedge btnclk or posedge rst)
     begin
          if(rst)
               clk_cnt <= 0;
          else if(btnclk)
               clk_cnt <= clk_cnt+1;
     end

     always@ (posedge btreg or posedge rst)
     begin
          if(rst)
               sw_reg<=0;
          else
               sw_reg <= ~sw_reg;
     end

     always@ (posedge btsw or posedge rst)
     begin
          if(rst)
               sw_reg_ins<=0;
          else
               sw_reg_ins <= ~sw_reg_ins;
```

```
    end

    wire [31:0] instr;
    wire [5:0] op, func;
    wire [4:0] rs,rt,rd,shamt,td;
    wire [15:0] imm;
    wire [25:0] addr;
    wire [31:0] saout;
    wire JR, J, JAL, LW, WREG, WMEM, RDorRT, SE, SA, IorR, BJ;
    wire [4:0] Aluc;
    wire [31:0] a, b, aluresult;     //alu in/out
    wire [4:0] r1, r2, r3, w1;  //Reg
    wire [31:0] d1, d2, d3, wdata;   //Reg
    wire [31:0] seout; //signed extended
    wire [31:0] pcplus4, pcin, pcout, immaddr, jaddr;
    wire [31:0] memdata;
    wire equal_result;

    //IF
    wire [31:0] tmp_pcin,tmp_pcout;
    //   ID
    wire [31:0] id_pcplus4, id_instr;//,tmp_d1,tmp_d2;
    wire [4:0] id_td;
//  wire WREG_id;
    //   EX
    wire [31:0] ex_a, ex_b, ex_d2,ex_instr,ex_pc;
    wire [4:0] ex_td;
    wire [4:0] ex_Aluc;
    wire ex_WREG, ex_WMEM, ex_LW;
    //   ME
    wire [31:0] me_aluresult, me_d2,me_instr,me_memdata,me_pc;
    wire [4:0] me_td;
    wire me_WREG, me_WMEM, me_LW;

    //   WB
    wire [31:0] wb_memdata;
    wire [4:0] wb_td;
    wire wb_WREG;//wb_LW;

    //
    wire stall;

    //pc0
    wire jump_en;
    wire [1:0] cp_oper;
    wire [31:0] CPR, jump_addr, d2_tmp4, pcintmp,pcouttmp,if_pc,id_pc;
    wire MFC,swstall;

    assign swstall=SW[3] | SW[2];

    wire [31:0] dis_data;
    wire [7:0] dis_addr;

    wire [7:0] IF,ID,EX,MEM;
    exin exin1(btnclk, rst, instr, IF);
    exin exin2(btnclk, rst, id_instr, ID);
    exin exin3(btnclk, rst, ex_instr, EX);
    exin exin4(btnclk, rst, me_instr, MEM);


    assign                      dis_data=(sw_reg_ins)?instr:{IF,ID,EX,MEM};//assign
dis_data=(sw_reg_ins)?instr:d3;
    assign dis_addr=(sw_reg_ins)?{8'b00100001}:{3'b000,r3};
```

```
    assign r2=rt;
    assign r3[4:0]={sw_reg,SW[3:0]};

    assign                                                  btnclk=BTNN;//anti_jitter
at0(.clk(CCLK),.rst(rst),.sig_i(BTNN),.sig_o(btnclk));//
    assign                  rst                 =                   BTNE;//anti_jitter
at1(.clk(CCLK),.rst(rst),.sig_i(BTNE),.sig_o(rst));//
    assign                  btsw                =                   BTNW;//anti_jitter
at2(.clk(CCLK),.rst(rst),.sig_i(BTNW),.sig_o(btsw));//
    assign                 btreg                =                   BTNS;//anti_jitter
at3(.clk(CCLK),.rst(rst),.sig_i(BTNS),.sig_o(btreg));//

    ////////////////////////////////////////////////////////////////////////////////
//
    //   PC
    ////////////////////////////////////////////////////////////////////////////////
//
    assign pcin = (jump_en)?(jump_addr+4):pcintmp;
    assign pcintmp=(J|JAL|JR|BJ)?(tmp_pcin+4):(tmp_pcin);
    assign pcout=(jump_en)?(jump_addr):pcouttmp;
    assign pcouttmp=(J|JAL|JR|BJ)?(tmp_pcin):(tmp_pcout);


    lock32 lock32(.clk(btnclk), .rst(rst), .in(pcout-4), .out(if_pc));

    pc pc(
    .clk(btnclk), .rst(rst), .stall(stall),
        .i_pc(pcin), .o_pc(tmp_pcout)//?
    );

    pc_plus4 pc_plus4(
        .i_pc(pcout), .o_pc(pcplus4)
   );

    decode4_32 pc_in(
        .in1(jaddr), .in2({d1[29:0],2'd0}), .in3(immaddr), .in4(id_pcplus4), .c1(J |
JAL), .c2(JR), .c3(BJ), .out(tmp_pcin)
    );

    instrmem instrmem(
        .clka(btnclk),.addra(pcout[11:2]),.douta(instr[31:0])
    );

    ////////////////////////////////////////////////////////////////////////////////
//
    //   IF-ID
    ////////////////////////////////////////////////////////////////////////////////
//
    IF_ID IF_ID(
        .clk(btnclk), .rst(rst), .stall(stall|swstall), .BJ(J|JAL|JR|BJ|jump_en),
        .if_pcplus4(pcplus4), .if_instr(instr), .if_pc(if_pc),
        .id_pcplus4(id_pcplus4), .id_instr(id_instr), .id_pc(id_pc)
    );

    assign op=id_instr[31:26];
    assign rs=id_instr[25:21];
    assign rt=id_instr[20:16];
    assign rd=id_instr[15:11];
    assign shamt=id_instr[10:6];
    assign func=id_instr[5:0];
    assign imm=id_instr[15:0];
    assign addr=id_instr[25:0];
```

```
    assign saout={27'b0,shamt};

    cpu_ctl cpu_ctl(
        .op(op), .func(func), .equal_result(equal_result), .rs(rs),
        .JR(JR), .J(J), .JAL(JAL), .LW(LW),
        .WREG(WREG), .WMEM(WMEM),. RDorRT(RDorRT),
        .SE(SE), .SA(SA), .IorR(IorR), .BJ(BJ),
        .Aluc(Aluc), .cp_oper(cp_oper), .MFC(MFC)
    );

//  assign WREG_id=JAL & WREG;
    assign td=(JAL)?6'd31:((RDorRT)?rd:rt);
//  assign d1 = (wb_td==r1)?wb_memdata:tmp_d1;

    //forwarding
    wire [31:0] d1_tmp1, d1_tmp2, d2_tmp1, d2_tmp2, d1_tmp3, d2_tmp3;

    assign d1 = (ex_td==r1 & |r1 & (ex_WREG | ex_LW))? aluresult : d1_tmp1;
    assign d1_tmp1 = (me_td==r1 & |r1 & (me_WREG|me_LW))? me_memdata : d1_tmp2;
    assign d1_tmp2 = (wb_td==r1 & |r1 & wb_WREG)? wb_memdata : d1_tmp3;

    assign d2 = (MFC)? CPR : d2_tmp4;
    assign d2_tmp4 = (ex_td==r2 & |r2 & (ex_WREG | ex_LW))? aluresult : d2_tmp1;
    assign d2_tmp1 = (me_td==r2 & |r2 & (me_WREG | me_LW))? me_memdata : d2_tmp2;
    assign d2_tmp2 = (wb_td==r2 & |r2 & wb_WREG)? wb_memdata : d2_tmp3;

//  assign d2 = (wb_td==r2)?wb_memdata:tmp_d2;

    reg32 reg32(
        .clk(btnclk), .rst(rst), .wea(wb_WREG),//.wea(WREG_id | me_WREG | me_LW),//
        .r1(r1), .r2(r2), .r3(r3), .w1(w1),
        .wdata(wdata), .out1(d1_tmp3), .out2(d2_tmp3), .out3(d3)
    );

    imm_addr imm_addr(
        .imm(imm), .pc(id_pcplus4), .out(immaddr)
    );

    j_addr j_addr(
        .addr(addr), .pc(id_pcplus4), .out(jaddr)
    );

    isequal isequal(
        .in1(d1), .in2(d2), .result(equal_result)
    );

    se se(
        .in(imm), .SE(SE), .out(seout)
    );

    decode3_32 alu_a(
        .in1(id_pcplus4-8), .in2(saout), .in3(d1), .c1(JAL), .c2(SA), .out(a)
    );

    decode3_32 alu_b(
        .in1(32'b0), .in2(seout), .in3(d2), .c1(JAL), .c2(IorR), .out(b)
    );

    assign r1=rs;
//  decode2_5 reg_read(
//      .in1(5'b11111), .in2(rs), .c(JR), .out(r1)
//  );
```

```
    decode2_5 reg_write(

    .in1(wb_td), .in2(5'b11111), .c(wb_WREG), .out(w1)//.in1(me_td), .in2(5'b11111),
 .c(wb_WREG), .out(w1)//
    );

    decode2_32 reg_wdata(

    .in1(wb_memdata), .in2(id_pcplus4), .c(wb_WREG), .out(wdata)//.in1(me_memdata),
.in2(id_pcplus4), .c(wb_WREG), .out(wdata)//
    );


    ////////////////////////////////////////////////////////////////////////////////
//
    //   ID-EX
    ////////////////////////////////////////////////////////////////////////////////
//
    wire [31:0] ex_d2_tmp;
    assign ex_d2 = ((ex_td==me_td) & (me_WREG|me_LW)) ? me_memdata : ex_d2_tmp;

    ID_EX ID_EX(
        .clk(btnclk), .rst(rst), .stall(swstall),
        .id_a(a), .id_b(b), .id_td(td), .id_d2(d2), .id_Aluc(Aluc), .id_WREG(JAL |
WREG & ~LW & |td), .id_WMEM(WMEM), .id_LW(LW & |td),.id_instr(id_instr),.id_pc(id_pc),

    .ex_a(ex_a), .ex_b(ex_b), .ex_td(ex_td), .ex_d2(ex_d2_tmp), .ex_Aluc(ex_Aluc), .
ex_WREG(ex_WREG), .ex_WMEM(ex_WMEM), .ex_LW(ex_LW),.ex_instr(ex_instr),.ex_pc(ex_pc)
    );

    alu alu(
        .a(ex_a), .b(ex_b), .aluc(ex_Aluc), .result(aluresult)
    );

    ////////////////////////////////////////////////////////////////////////////////
//
    //   EX-MEM
    ////////////////////////////////////////////////////////////////////////////////
//
    wire [9:0] ex_addra;
    assign ex_addra = (me_WMEM)?me_aluresult[11:2]:aluresult[11:2];
    EX_ME EX_ME(
        .clk(btnclk), .rst(rst), .stall(swstall),
        .ex_aluresult(alure-
sult), .ex_td(ex_td), .ex_d2(ex_d2), .ex_WREG(ex_WREG ), .ex_WMEM(ex_WMEM), .ex_LW(ex
_LW ),.ex_instr(ex_instr),.ex_pc(ex_pc),
        .me_aluresult(me_alure-
sult), .me_td(me_td), .me_d2(me_d2), .me_WREG(me_WREG), .me_WMEM(me_WMEM), .me_LW(me_
LW),.me_instr(me_instr),.me_pc(me_pc)
    );

    Data_Mem Data_Mem(
        .clka(btnclk),.wea(me_WMEM),.addra(ex_ad-
dra),.dina(me_d2),.douta(memdata)//.clka(btnclk),.wea(me_WMEM),.addra(me_alure-
sult[11:2]),.dina(me_d2),.douta(memdata)
    );

    assign me_memdata=(me_WREG)?me_aluresult:memdata;
    ////////////////////////////////////////////////////////////////////////////////
//
    //   MEM-WB
    ////////////////////////////////////////////////////////////////////////////////
//
```

```
    ME_WB ME_WB(
        .clk(btnclk), .rst(rst), .stall(swstall),
        .me_memdata(me_memdata), .me_td(me_td), .me_WREG(me_WREG | me_LW),
        .wb_memdata(wb_memdata), .wb_td(wb_td), .wb_WREG(wb_WREG)
    );

    display2 ds(
        .clk(CCLK),.rst(rst),.instr(in-
str),.reg_data(d3),.stage({IF,ID,EX,MEM}),.clk_cnt(clk_cnt),.reg_addr({3'b0,r3}),
        .lcd_rs(LCDRS), .lcd_rw(LCDRW), .lcd_e(LCDE), .lcd_dat(LCDDAT)
    );

    ////////////////////////////////////////////////////////////////////////////
//
    //  WB
    ////////////////////////////////////////////////////////////////////////////
//

    pc0 pc0(
        .clk(CCLK),                  .rst(rst),                  .interrupt_sig-
nal(SW[3:2]),            .cp_oper(cp_oper),            .cp_cd(rd),            .re-
turn_addr(me_pc), .GPR(d2), .jump_en(jump_en), .CPR(CPR), .jump_addr(jump_addr)
    );

    ////////////////////////////////////////////////////////////////////////////
//
    //  Stall Control Logic
    ////////////////////////////////////////////////////////////////////////////
//
    StallControlLogic scl(
        .rs(instr[25:21]),.rt(instr[20:16]),.op(instr[31:26]), .func(instr[5:0]),
        .id_td(td), .id_LW(LW & |td),
        .stall(stall)
    );
endmodule
```

三. 实验过程和数据记录及结果分析

Test Code:

```
0:  17 3c010000 lui R1,0x0
4:  13 24210020 addiu R1,R1,32
8:  21 40810800 mtc R1, R1
c:  01 00001020 add R2,R0,R0
10: 01 00001820 add R3,R0,R0
14: 12 20420001 addi R2,R2,1
18: 1E 08000005 j 14

1c: 0B 00000000 nop

20: 20 40027000 mfc R14,R2
24: 20 40036800 mfc R13,R3
28: 22 42000018 eret
2c: 0B 00000000 nop
```

The initial of the program:

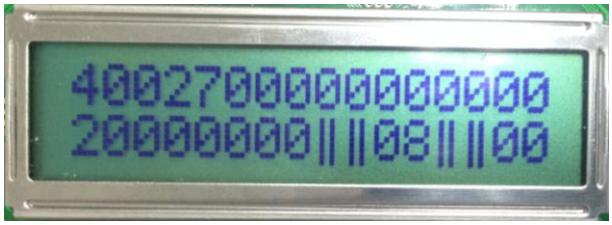
The first four instructions:

```
lui R1,0x0
addiu R1,R1,32
mtc R1, R1
add R2,R0,R0
```

and after addiu is in state WB, reg 1 = 20;

The MTC instruction has been executed, so EHB =20 , we can start using interruption now, and the SW<4> is on(LCD3 is on), and press btn, we start interrupt;

The IF/ID, ID/EX, EX/ME,ME/WB are cleaned, and the `mfc R14,R2` loaded;

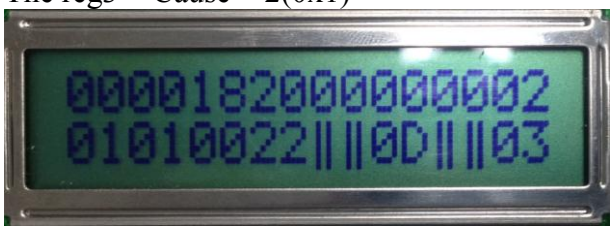


The four interruption part instruction:

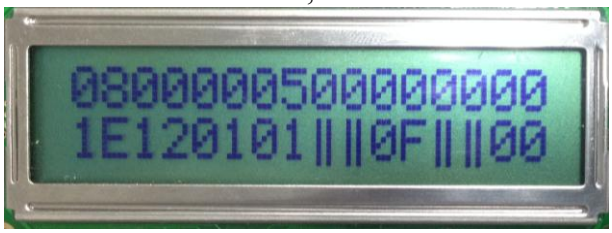
```
mfc R14,R2
mfc R13,R3
eret
nop
```

After eret, we jump back to `add R2,R0,R0 ;`
and the nop is cleaned, and the reg2 = EPC = C;

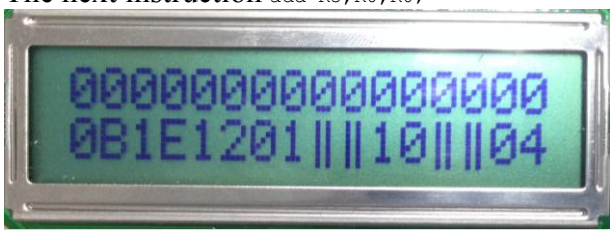
The reg3 = Cause = 2(0x1)



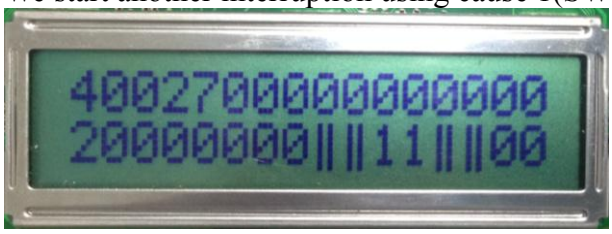
Back to the state before;



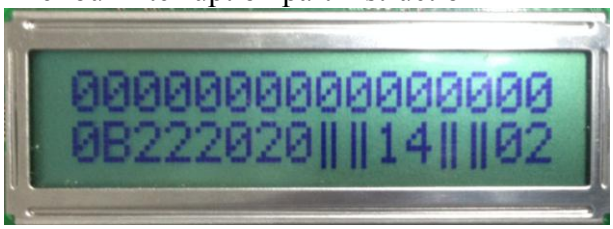
The next instruction `add R3,R0,R0;`



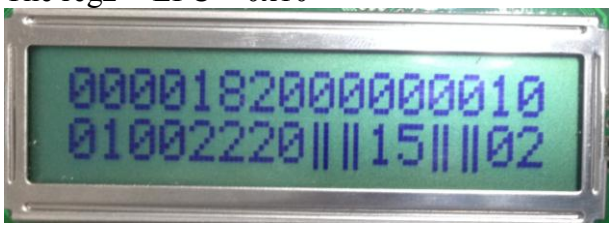
We start another interruption using cause 1(SW<3>)



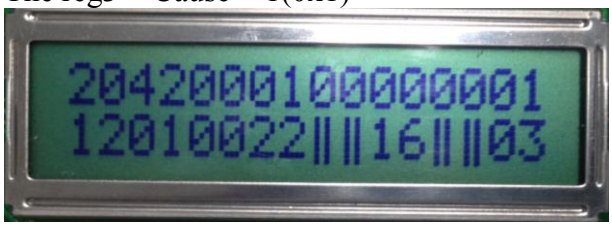
The four interruption part instruction



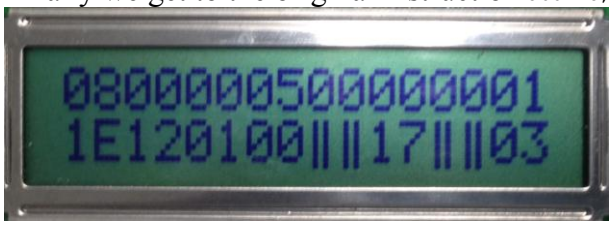
The reg2 = EPC = 0x10



The reg3 = Cause = 1(0x1)



Finally we get to the original instruction `add R3,R0,R0;`



All the result is all right!

四. 讨论与心得

The most difficult part of the lab is the pc pushing problem, where to push the pc into (IF) state and get back (ME) from, is very important.