

**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

1. 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;

1. 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;

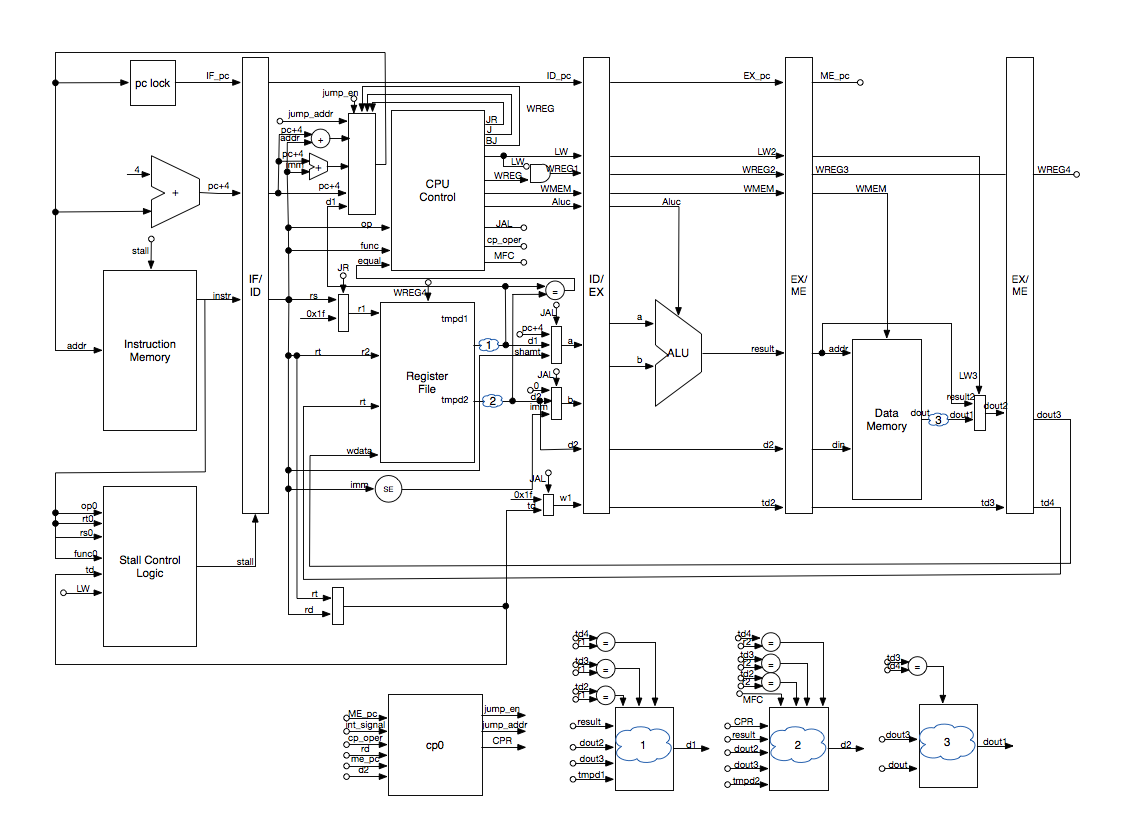
2. And a pc lock to record the clock of instruction in IF state, so that it can be passed to ID, EXE and MEM;

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):

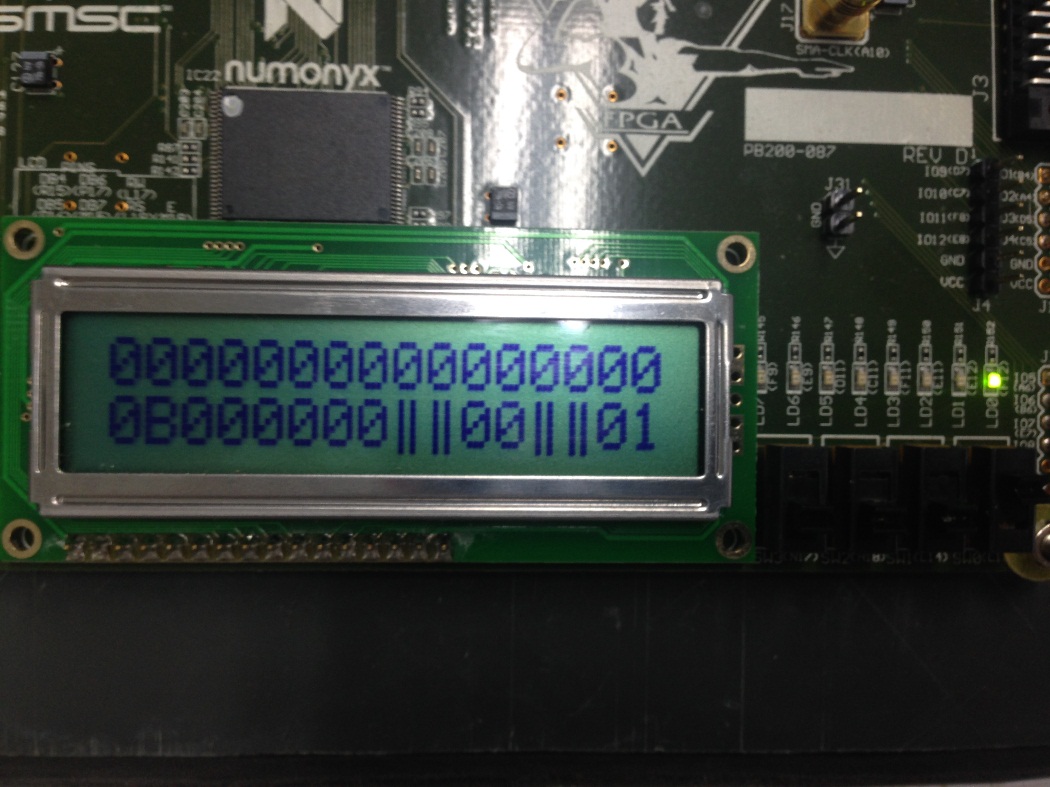
|  |
| --- |
| module top(CCLK, SW, BTNN, BTNE, BTNS, BTNW,LED,  LCDE, LCDRS, LCDRW, LCDDAT  );  input wire CCLK;  input wire [3:0] SW;  input wire BTNN,BTNE,BTNW,BTNS;  output wire LCDE, LCDRS, LCDRW;  output wire [3:0] LCDDAT;  output wire [7:0] LED;    wire btnclk;  wire rst;  reg sw\_reg,sw\_reg\_ins;  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(aluresult), .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\_aluresult), .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\_addra),.dina(me\_d2),.douta(memdata)//.clka(btnclk),.wea(me\_WMEM),.addra(me\_aluresult[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(instr),.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\_signal(SW[3:2]), .cp\_oper(cp\_oper), .cp\_cd(rd), .return\_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 |

1. 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

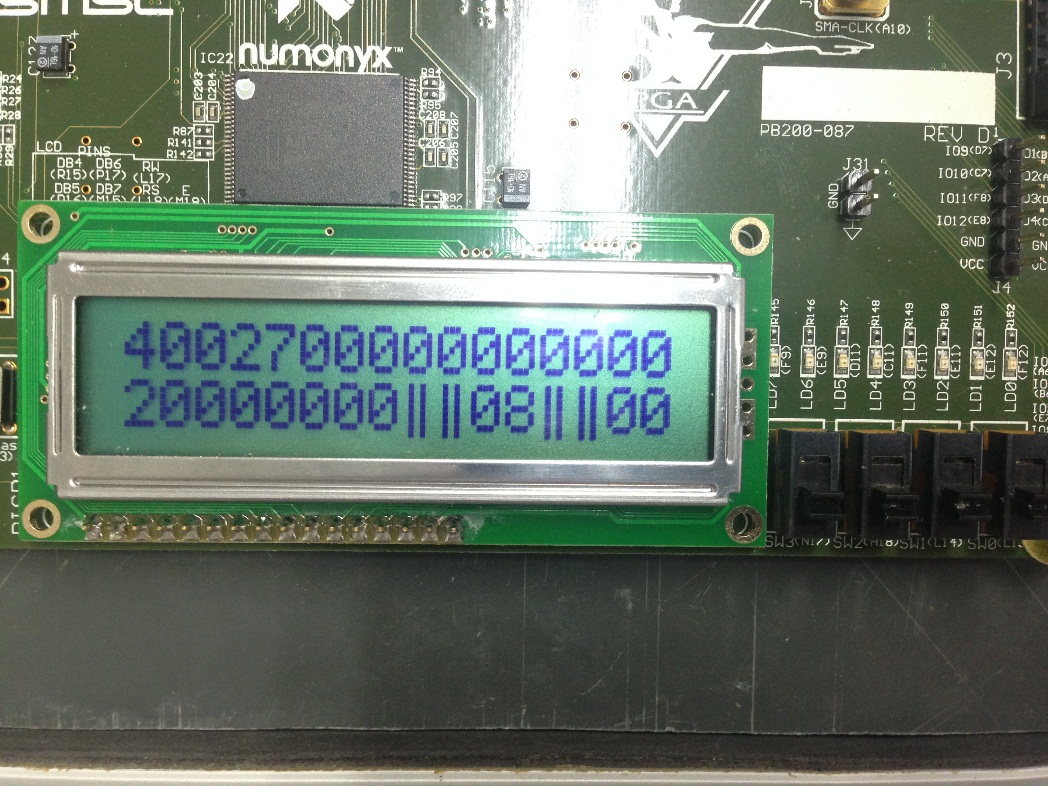
and after addiu is in state WB, reg 1 = 20;



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

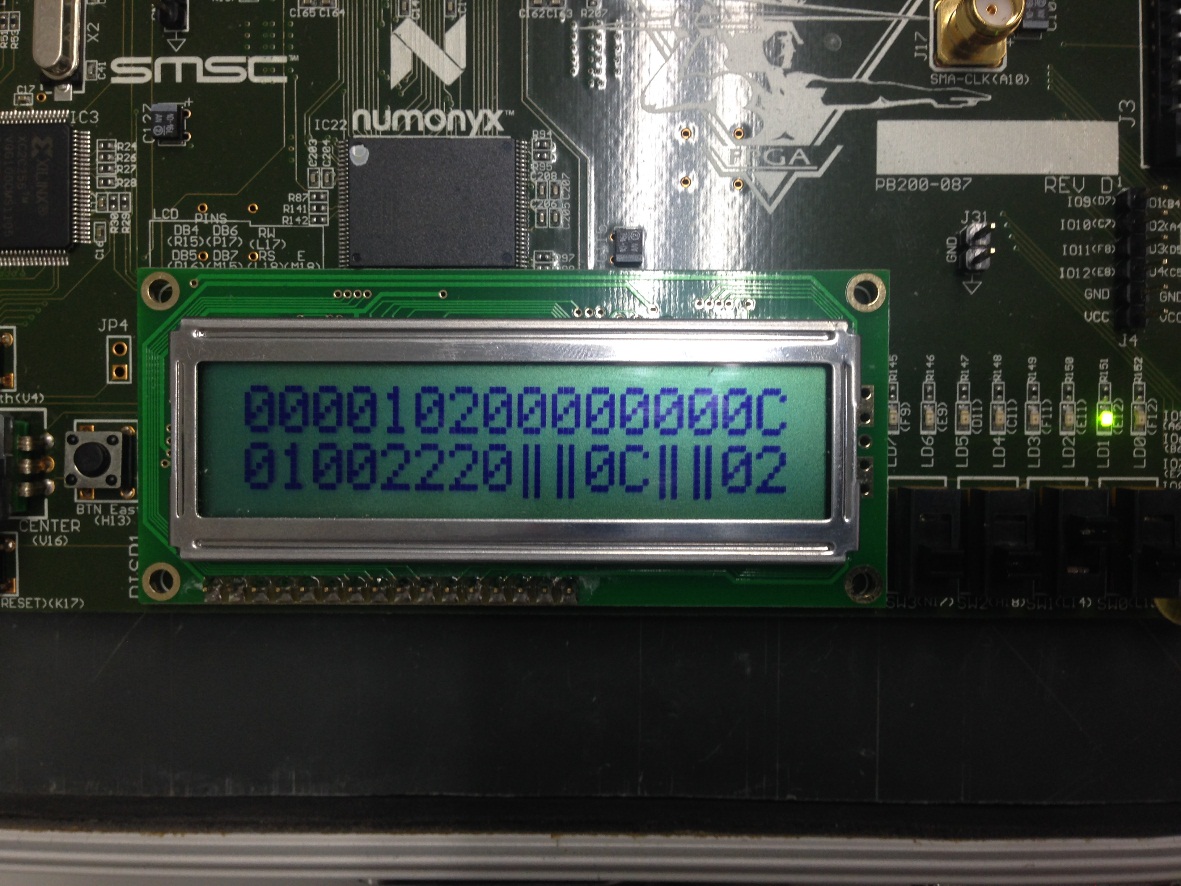
eret

nop



After eret, we jump back to add R2,R0,R0 ;

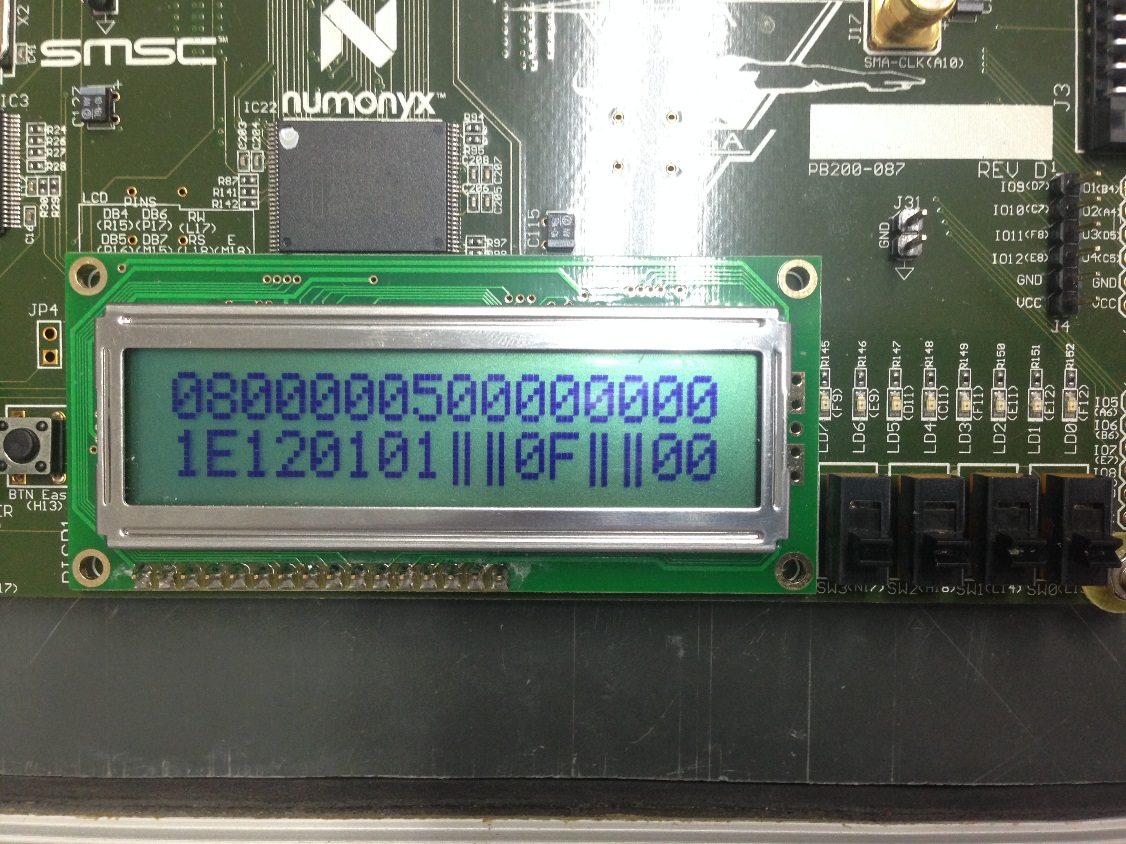
and the nop is cleaned, and the reg2 = EPC = C;



The reg3 = Cause = 2(0x1)



Back to the state before;



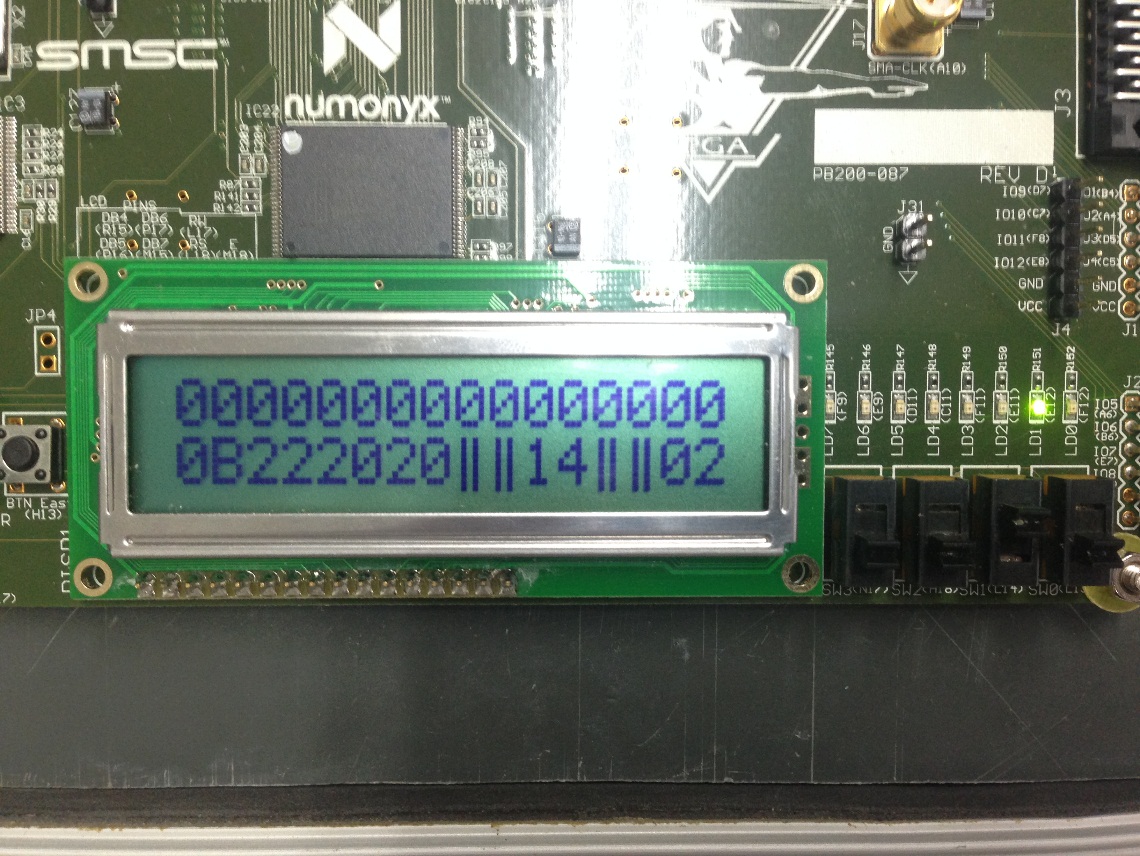
The next instruction add R3,R0,R0;



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!

1. 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.