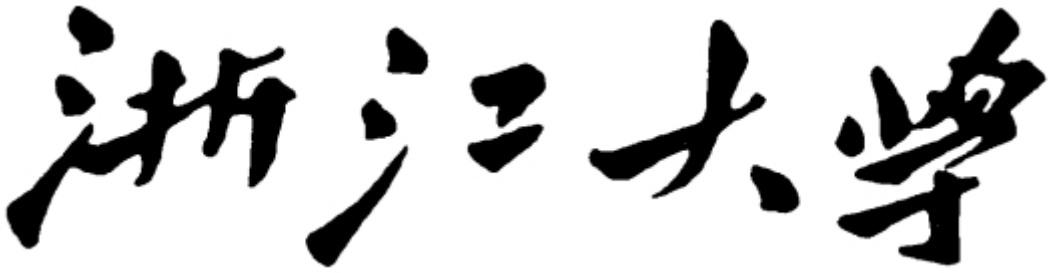


本科实验报告

课程名称:	**计算机组成**
姓 名:	臧可
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
学 号:	3180102095
指导教师:	姜晓红

2020年　8月　17日

浙江大学实验报告

课程名称：　计算机组成　　实验类型：　综合

实验项目名称：　Bonus Lab: Interruption implement extention to your CPU

学生姓名：　臧可　　专业：计算机科学与技术　　学号：3180102095

同组学生姓名：　None　　指导老师：　姜晓红

实验地点：东四509　　实验日期：　2020年　8月 17日

一、 实验目的和要求

Object:

- Design the simple interruption
- Design interrupt test program

Requirement:

- Add SCPU interrupt function
  - Compatible with SCPU data path
  - Add interruption path
  - Add interrupt control
- Fixed interrupt vector (ARM)
  - Illegal instruction interrupt
  - External interrupt
- This experiment is completed on the basis of compatibility with Exp07

二、 实验内容和原理

2.1 Interrupt instructions

eret

PC <= EPC (The Cause and Status registers of CP0 have changed)

Op=6bit	1	19bit	FUN
0x0	1	000 0000 0000 0000 0000	011000

syscall

EPC = PC+4;

PC <= Exception handling address;

Cause and Status registers have changed

Op=6bit	20bit	FUN
0x0	0000 0000 0000 0000 0000	0011000

2.2 MIPS interrupt response

initialization

- Set SR: turn off interrupt IE=0, KSU ERL EXL=00 0 0
- System initialization
- Set SR: open interrupt IE=1, set IM7-0

Interrupt response

- Hardware save breakpoint: EPC←PC+4
- Hardware modification PC: PC←vector, hardware turn off interrupt
- Interrupt service: protect register, open interrupt, service, restore register
- Open interrupt
- Return: eret (Hardware modify the Cause and Status registers)

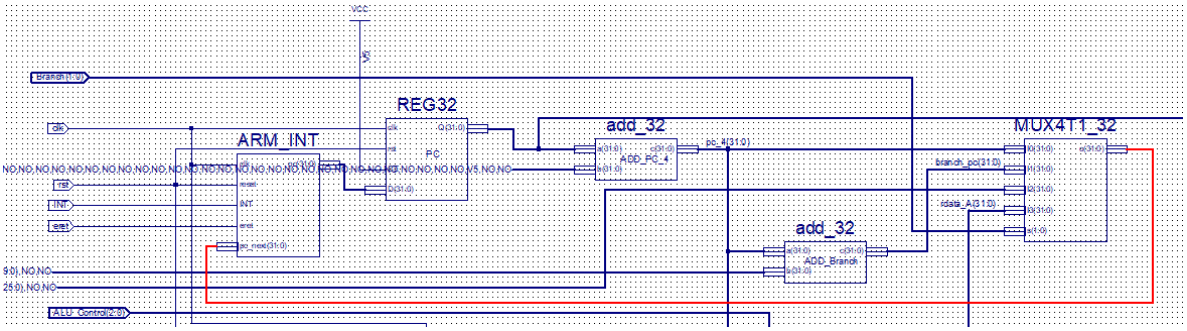
2.3 DataPath extended interrupt path

Interrupt path module: detection and latch

```
`timescale 1ns / 1ps

module ARM_INT
    (
     input wire clk,
     input wire reset,            // reset
     input wire INT,        // external interrupt
     input wire eret,             // return to exception address instruction
     input wire [31:0] pc_next,
     output reg [31:0] pc
    );

    reg int_req;//interrupt request

    reg [31:0] epc;
    reg [31:0] epc_next;

    reg handled;
    reg handled_next;

    //中断触发检测与服务锁存
    always @(posedge reset or posedge INT or posedge handled) begin
        if (reset) int_req <= 1'b0;
        else if (handled) int_req <= 1'b0;
        else int_req <= 1'b1;               //set interrupt request
    end

    //断点保护，中断开，关与返回
    always @(posedge clk or posedge reset) begin
        if (reset)
            begin
                epc <= 32'h00000000;
                handled <= 1'b0;
            end
        else
            begin
                epc <= epc_next;
                handled <= handled_next;
            end
    end

    //PC输出通路
    always @* begin
            epc_next = epc;
            handled_next = 1'b0;
            pc = pc_next;           //next instruction
```

```
            if (eret) pc = epc;     //interrupt return
            else if (int_req)
                begin
                    epc_next = pc_next;
                    handled_next = 1'b1;
                    pc = 32'h4;     //interrupt vector
                end
    end

endmodule

```

Add DataPath in ARM interrupt mode



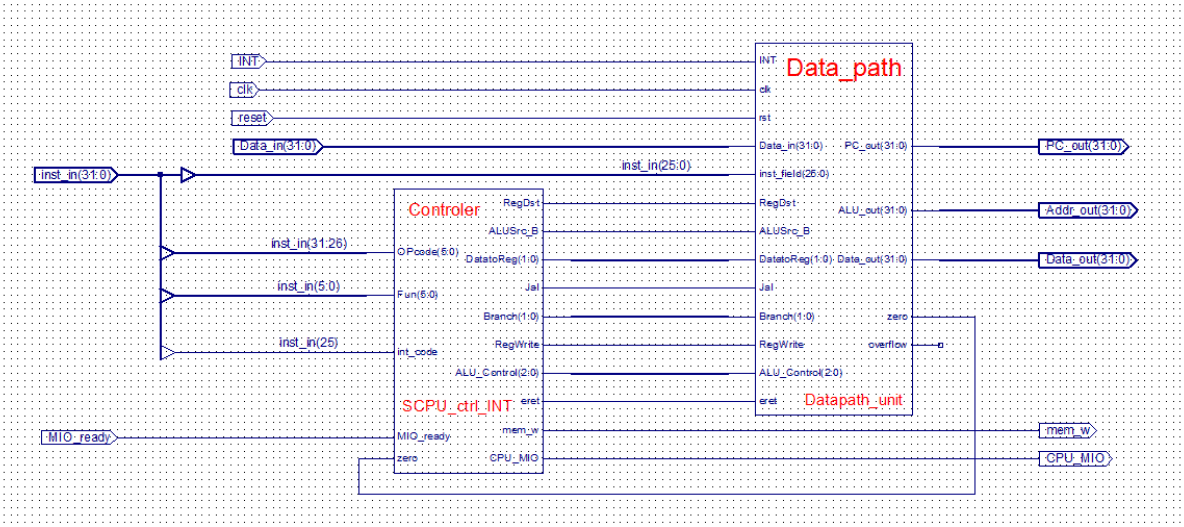
2.4 Controller extension interrupt decoding

```
`timescale 1ns / 1ps

module SCPU_ctrl_INT( input [5:0]OPcode,
                      input [5:0]Fun,
                      input int_code,
                      input MIO_ready,
                      input zero,
                      output reg RegDst,
                      output reg ALUSrc_B,
                      output reg [1:0] DatatoReg,
                      output reg Jal,
                      output reg [1:0]Branch,
                      output reg RegWrite,
                      output reg mem_w,
                      output reg [2:0]ALU_Control,
                      output reg CPU_MIO,
                   output reg eret
    );

    `define CPU_ctrl_signals{RegDst,ALUSrc_B,DatatoReg,Jal,Branch,RegWrite,mem_w,CPU_MIO,ALU_Control}
        always@*begin
            eret = 0;
            case(OPcode)
                6'b000000:begin
                    case(Fun)
                        6'b100000:begin `CPU_ctrl_signals = 13'b10000_00100_010;end //add
                        6'b100010:begin `CPU_ctrl_signals = 13'b10000_00100_110;end //sub
```

```
                        6'b100100:begin `CPU_ctrl_signals = 13'b10000_00100_000;end //and
                        6'b100101:begin `CPU_ctrl_signals = 13'b10000_00100_001;end //or
                        6'b000010:begin `CPU_ctrl_signals = 13'b10000_00100_101;end //srl
                        6'b101010:begin `CPU_ctrl_signals = 13'b10000_00100_111;end //slt
                        6'b100111:begin `CPU_ctrl_signals = 13'b10000_00100_100;end //nor
                        6'b100110:begin `CPU_ctrl_signals = 13'b10000_00100_011;end //xor
                        6'b001000:begin `CPU_ctrl_signals = 13'b10001_11000_000;end //jr
                        6'b001001:begin `CPU_ctrl_signals = 13'b10111_11100_010;end //jalr
                        default:  begin `CPU_ctrl_signals = 13'b00000_00000_000;end
                    endcase
                end
                6'b100011:begin `CPU_ctrl_signals = 13'b01010_00100_010;end //load
                6'b101011:begin `CPU_ctrl_signals = 13'b01000_00010_010;end //store
                6'b000100:begin
                    if(zero==1'b1) `CPU_ctrl_signals = 13'b00000_01000_110;
                    else           `CPU_ctrl_signals = 13'b00000_00000_110;
                end //beq
                6'b000101:begin
                    if(zero==1'b0) `CPU_ctrl_signals = 13'b00000_01000_110;
                    else           `CPU_ctrl_signals = 13'b00000_00000_110;
                end //bne
                6'b000010:begin `CPU_ctrl_signals = 13'b00000_10000_000;end //j
                6'b000011:begin `CPU_ctrl_signals = 13'b00111_10100_010;end //jal
                6'b001111:begin `CPU_ctrl_signals = 13'b00100_00100_010;end //lui
                6'b001010:begin `CPU_ctrl_signals = 13'b01000_00100_111;end //slti
                6'b001110:begin `CPU_ctrl_signals = 13'b01000_00100_011;end //xori
                6'b001101:begin `CPU_ctrl_signals = 13'b01000_00100_001;end //ori
                6'b001100:begin `CPU_ctrl_signals = 13'b01000_00100_000;end //andi
                6'b001000:begin `CPU_ctrl_signals = 13'b01000_00100_010;end //addi
                6'b010000:begin
                            `CPU_ctrl_signals = 13'b10000_11100_010;
                            eret = 1;
                            end //eret
                default:  begin `CPU_ctrl_signals = 10'b00000_00000_000;end
            endcase
        end

endmodule

```

2.5 Add CPU module after interrupt

SCPU

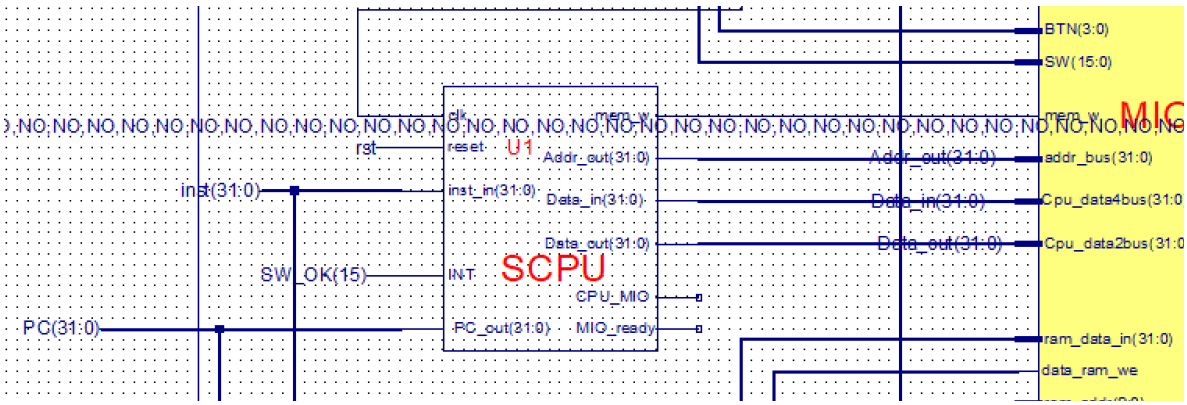


SCPU simulation code

```
`timescale 1ns / 1ps

module SCPU_SCPU_sch_tb();

// Inputs
   reg clk;
   reg reset;
   reg [31:0] Data_in;
   reg [31:0] inst_in;
   reg INT;
   reg MIO_ready;

// Output
   wire [31:0] PC_out;
   wire [31:0] Addr_out;
   wire [31:0] Data_out;
   wire mem_w;
   wire CPU_MIO;

// Bidirs

// Instantiate the UUT
   SCPU UUT (
        .clk(clk),
        .reset(reset),
        .Data_in(Data_in),
        .inst_in(inst_in),
        .INT(INT),
        .PC_out(PC_out),
        .Addr_out(Addr_out),
        .Data_out(Data_out),
        .MIO_ready(MIO_ready),
        .mem_w(mem_w),
        .CPU_MIO(CPU_MIO)
```

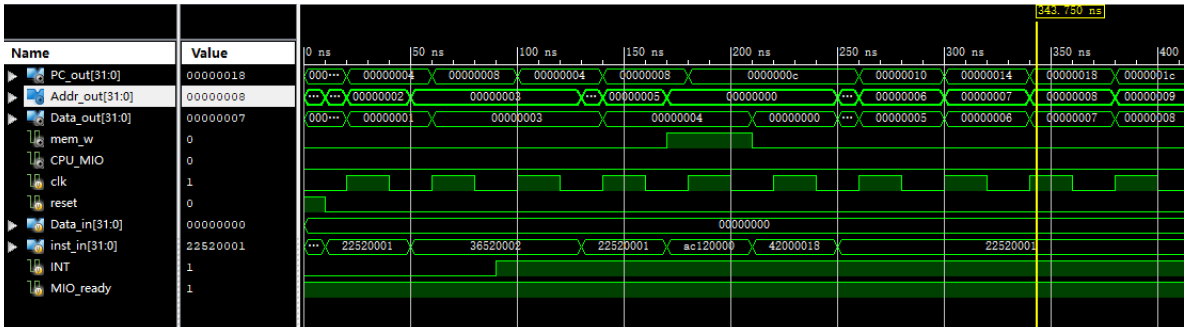
```
    );
// Initialize Inputs
  // `ifdef auto_init
       initial begin
        clk = 0;
        reset = 1;
        Data_in = 0;
        inst_in = 0;
        INT = 0;
        MIO_ready = 1;
   //`endif
        #10;reset=0;
        // Add stimulus here
            inst_in = 32'h22520001; // addi s2, s2, 1
            #40;
            inst_in = 32'h36520002; //ori $s2, $s2, 2
            #40;
            INT = 1;
            #40;
            inst_in = 32'h22520001; // addi s2, s2, 1
            #40;
            inst_in = 32'hac120000; //sw $s2, 0($zero)
            #40;
            inst_in = 32'h42000018; //eret
            #40;
            inst_in = 32'h22520001; // addi s2, s2, 1
        end

    always begin
        clk = 0;#20;
        clk = 1;#20;
    end
endmodule
```

2.6 Top



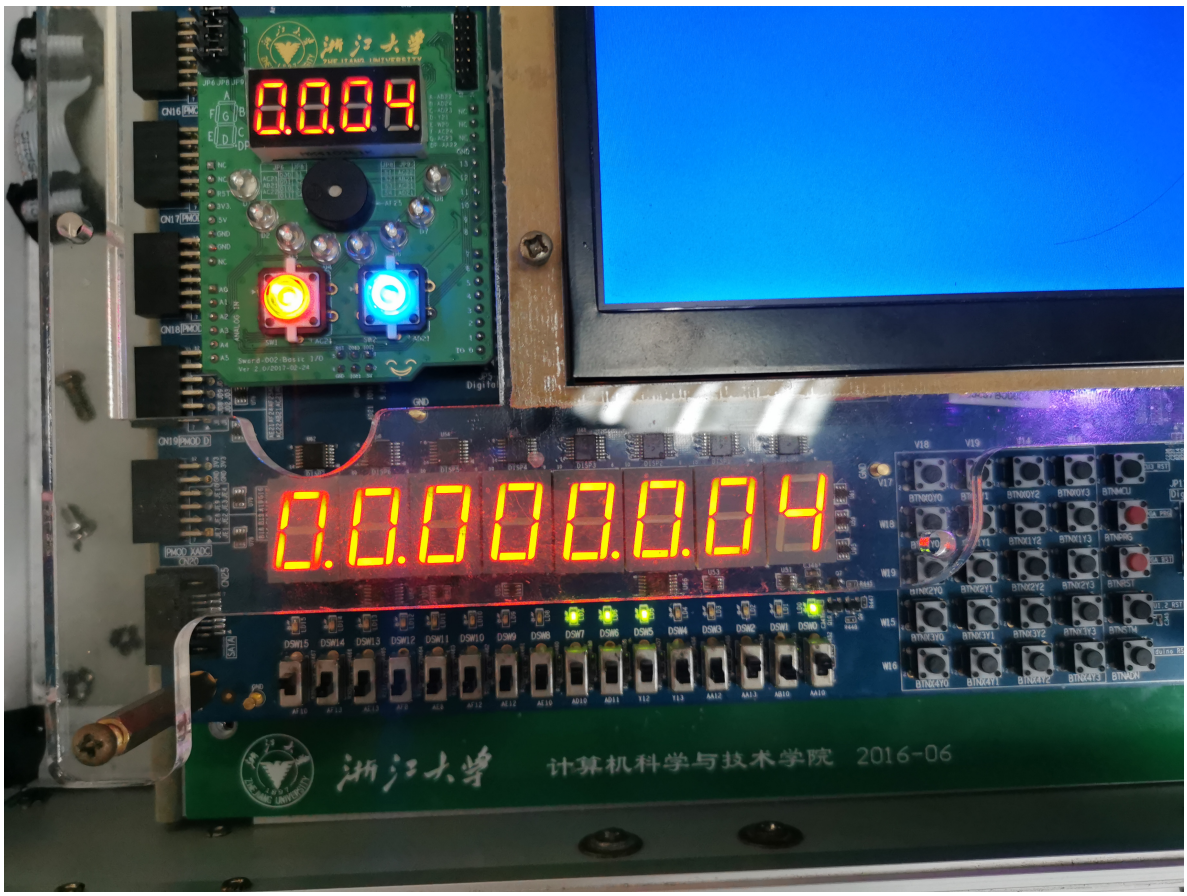
三、 实验过程和数据记录及结果分析

3.1 SCPU simulation results



From the simulation results, we can see that when INT=4, the PC jumps to 4, which is in line with expectations.

3.2 sword board verification



上拨sw5,6,7和sw2，在8位7段数码管显示pc值。每次上拨sw15的时候实现INT，pc跳转到04，然后进入coe写好的用户代码中，当pc值跳到114的时候遇到eret指令，回到上拨sw15输入INT信号时存在epc里面的pc值+4。实验结果符合预期。

实验视频：https://pan.baidu.com/s/1k1FEGVAGMl25C7MuTW3SSw 提取码：ffwu

四、 讨论与心得

Through this experiment, I have a deeper understanding of the principle of interruption.

In the process of debugging, it is necessary to combine the coe file to better understand the interruption. My mastery of mips instructions is also more comprehensive. After confirming that there is no problem with the code logic, the coe was checked, and a jump instruction before the eret instruction was modified to ensure that the eret code can be read after the interrupt instruction is executed.