# CPU

## Sccomp.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/12/23 19:51:45

// Design Name:

// Module Name: sccomp

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module sccomp(

input clk,

input rstn,

input [15:0]sw\_i,

output [7:0] disp\_seg\_o,

output [7:0] disp\_an\_o

);

reg [31:0] clkdiv;

wire Clk\_CPU;

always @(posedge clk or negedge rstn)begin

if(!rstn)

clkdiv <= 0;

else

clkdiv <= clkdiv + 1'b1;

end

assign Clk\_CPU = (sw\_i[15] ) ? clkdiv[27]:clkdiv[25];

reg [63:0 ]display\_data;

reg [5:0] led\_data\_addr;

reg [73:0] led\_disp\_data;

parameter LED\_DATA\_NUM = 19;

parameter ROM\_ADDR\_NUM = 64;

parameter DM\_DATA\_NUM = 16;

reg [63:0] LED\_DATA [18:0];

initial begin

LED\_DATA[0] = 64'hC6F6F6F0C6F6F6F0;

LED\_DATA[1] = 64'hF9F6F6CFF9F6F6CF;

LED\_DATA[2] = 64'hFFC6F0FFFFC6F0FF;

LED\_DATA[3] = 64'hFFC0FFFFFFC0FFFF;

LED\_DATA[4] = 64'hFFA3FFFFFFA3FFFF;

LED\_DATA[5] = 64'hFFFFA3FFFFFFA3FF;

LED\_DATA[6] = 64'hFFFF9CFFFFFF9CFF;

LED\_DATA[7] = 64'hFF9EBCFFFF9EBCFF;

LED\_DATA[8] = 64'hFF9CFFFFFF9CFFFF;

LED\_DATA[9] = 64'hFFC0FFFFFFC0FFFF;

LED\_DATA[10] = 64'hFFA3FFFFFFA3FFFF;

LED\_DATA[11] = 64'hFFA7B3FFFFA7B3FF;

LED\_DATA[12] = 64'hFFC6F0FFFFC6F0FF;

LED\_DATA[13] = 64'hF9F6F6CFF9F6F6CF;

LED\_DATA[14] = 64'h9EBEBEBC9EBEBEBC;

LED\_DATA[15] = 64'h2737373327373733;

LED\_DATA[16] = 64'h505454EC505454EC;

LED\_DATA[17] = 64'h744454F8744454F8;

LED\_DATA[18] = 64'h0062080000620800;

end

// define

//指令

wire [31:0] rom\_addr;

wire [31:0] instr;

//寄存器

reg [31:0] reg\_data;

reg [5:0] reg\_addr;

//alu

reg [2:0] alu\_addr;

reg [31:0] alu\_disp\_data;

//内存

reg [31:0] dmem\_data;

reg [5:0] dmem\_addr;

wire signed [31:0] A,B,aluout;

//Branch相关指令使用到的信号

wire Zero,LT,GE;

wire Unsigned;

// 指令跳转

wire [2:0] NPCOp;

wire [31:0] PCout;

wire [31:0] NPC;

wire RegWrite,MemWrite,ALUSrc;

wire [5:0] EXTOp;

wire [4:0] ALUOp;

wire [2:0] DMType;

wire [1:0] WDSel;

wire [31:0] immout;

reg [31:0] WD;

wire [31:0]RD1,RD2;

wire [31:0] dout;

//Decode

wire[6:0] Op = instr[6:0];//op

wire[6:0] Funct7 = instr[31:25];//funct7

wire[2:0] Funct3 = instr[14:12];//funct3

wire[4:0] rs1 = instr[19:15];//rs1;

wire[4:0] rs2 = instr[24:20];//rs2

wire[4:0] rd = instr[11:7];

wire[4:0] iimm\_shamt = instr[24:20];//slli

wire[11:0] iimm=instr[31:20];//addi

wire[11:0] simm={instr[31:25],instr[11:7]};//sw

wire[11:0] bimm={instr[31],instr[7],instr[30:25],instr[11:8]};//sb

wire[11:0] jimm={instr[31],instr[19:12],instr[20],instr[30:21]};

//ROM

dist\_mem\_gen\_0 U\_IM(.a(rom\_addr[31:2]),.spo(instr));

//Ctrl

Ctrl U\_Ctrl(.Op(Op),.Funct7(Funct7),.Funct3(Funct3),.Zero(Zero),.LT(LT),.GE(GE),.RegWrite(RegWrite),

.MemWrite(MemWrite),.EXTOp(EXTOp),.ALUOp(ALUOp),.ALUSrc(ALUSrc),.DMType(DMType),.WDSel(WDSel),.NPCOp(NPCOp),.Unsigned(Unsigned));

//EXT

EXT U\_EXT(.iimm\_shamt(iimm\_shamt),.iimm(iimm),.simm(simm),.bimm(bimm),.jimm(jimm),.EXTOp(EXTOp),.immout(immout));

//RF

`define WDSel\_FromALU 2'b00

`define WDSel\_FromMEM 2'b01

`define WDSel\_FromPC 2'b10

always @(\*)

begin

case(WDSel)

`WDSel\_FromALU: WD<=aluout;

`WDSel\_FromMEM: WD<=dout;

`WDSel\_FromPC: WD<=rom\_addr+4;

endcase

end

RF U\_RF(.clk(Clk\_CPU),.rst(rstn),.RFWr(RegWrite),.sw\_i(sw\_i),.A1(rs1),.A2(rs2),.A3(rd),.WD(WD),.RD1(RD1),.RD2(RD2));

// ALU

assign A = RD1;

assign B = (ALUSrc)? immout:RD2;

ALU U\_ALU(.A(A),.B(B),.ALUOp(ALUOp),.Unsigned(Unsigned),.C(aluout),.Zero(Zero),.LT(LT),.GE(GE));

// DM

DM U\_DM(.clk(Clk\_CPU),.DMWr(MemWrite),.addr(aluout),.din(RD2),.DMType(DMType),.dout(dout));

//PC&NPC

PC U\_PC(.clk(Clk\_CPU),.rstn(rstn),.sw\_i(sw\_i),.PC(rom\_addr),.NPC(NPC),.PCout(rom\_addr));

NPC U\_NPC(.PC(rom\_addr),.NPCOp(NPCOp),.immout(immout),.aluout(aluout),.NPC(NPC));

// Generate LED\_DATA

always @(posedge Clk\_CPU or negedge rstn) begin

if (!rstn) begin

led\_data\_addr <= 6'd0;

led\_disp\_data <= 64'b1;

reg\_addr = 6'b0;

alu\_addr = 3'b0;

dmem\_addr = 6'b0;

end else if (sw\_i[0] == 1'b1) begin

if (led\_data\_addr == LED\_DATA\_NUM) begin

led\_data\_addr <= 6'd0;

led\_disp\_data <= 64'b1;

end

else begin

led\_disp\_data <= LED\_DATA[led\_data\_addr];

led\_data\_addr <= led\_data\_addr + 1'b1;

end

// end else if(sw\_i[14]==1'b1) begin

// if(rom\_addr==ROM\_ADDR\_NUM)

// begin

// rom\_addr <= 1'b0;

// end

// else if(sw\_i[1] == 1'b0)begin

// led\_data\_addr <= led\_data\_addr;

// rom\_addr=rom\_addr+1'b1;

// end

// else begin

// led\_data\_addr <= led\_data\_addr;

// rom\_addr = rom\_addr;

// end

end else if(sw\_i[13]==1'b1) begin

reg\_data = U\_RF.rf[reg\_addr];

reg\_addr = reg\_addr+1'b1;

end else if(sw\_i[12]==1'b1)begin

case(alu\_addr)

3'b000:alu\_disp\_data = U\_ALU.A;

3'b001:alu\_disp\_data = U\_ALU.B;

3'b010:alu\_disp\_data = U\_ALU.C;

3'b011:alu\_disp\_data = U\_ALU.Zero;

default:alu\_disp\_data = 32'hFFFFFFFF;

endcase

alu\_addr = alu\_addr + 1'b1;

end else if(sw\_i[11]==1'b1)begin

dmem\_addr = dmem\_addr + 1'b1;

dmem\_data = U\_DM.dmem[dmem\_addr][7:0];

if(dmem\_addr == DM\_DATA\_NUM)begin

dmem\_addr = 6'b0;

dmem\_data = 32'hFFFFFFFF;

end

end

end

// Choose display source data

always @(sw\_i) begin

if (sw\_i[0] == 1'b0) begin

case (sw\_i[14:11])

4'b1000: display\_data = instr; // ROM

4'b0100: display\_data = reg\_data; // RF

4'b0010: display\_data = alu\_disp\_data; // ALU

4'b0001: display\_data = dmem\_data; // DMEM

endcase

end else begin

display\_data = led\_disp\_data;

end

end

seg7x16 u\_seg7x16(

.clk(clk),

.rstn(rstn),

.i\_data(display\_data),

.disp\_mode(sw\_i[0]),

**.o\_seg(disp\_seg\_o),**

**.o\_sel(disp\_an\_o)//注意名字**

);

endmodule

## Ctrl.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/12/23 22:26:09

// Design Name:

// Module Name: Ctrl

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module Ctrl(

input [6:0] Op,//opcode

input [6:0] Funct7,//funct7

input [6:0] Funct3,//funct3

input Zero,// signal for blt|bne

input LT,//signal for blt|bltu

input GE,//signal for bge|bgeu

output RegWrite,//control signal for regsiter write

output MemWrite,// control signal for memort write

output [5:0] EXTOp,// control signa to signed extension

output [5:0] ALUOp,//alu operation

output ALUSrc ,//ALU source for b

output [2:0] DMType,// dm r/w type

output [1:0] WDSel,// register write data selection

output [2:0] NPCOp,//next pc operation

output Unsigned// signal for bltu|bgeu

);

//R\_Type

wire rtype = ~Op[6]&Op[5]&Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0]; //op 0110011

wire i\_add=rtype&~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&~Funct3[0]; // add funct7 0000000 funct3 000

wire i\_sub=rtype&~Funct7[6]&Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&~Funct3[0]; // sub funct7 0100000 funct3 000

wire i\_sll=rtype&~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&Funct3[0]; // sll funct7 0000000 funct3 001

wire i\_srl=rtype&~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct7[2]&~Funct7[1]&~Funct7[0]&Funct3[2]&~Funct3[1]&Funct3[0]; // srl funct7 0000000 funct3 101

wire i\_sra=rtype&~Funct7[6]&Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct7[2]&~Funct7[1]&~Funct7[0]&Funct3[2]&~Funct3[1]&Funct3[0]; // sra funct7 0100000 funct3 101

//i\_l Type load

wire itype\_l = ~Op[6]&~Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0]; //op 0000011

wire i\_lb=itype\_l&~Funct3[2]& ~Funct3[1]& ~Funct3[0]; //lb funct3 000

wire i\_lh=itype\_l&~Funct3[2]& ~Funct3[1]& Funct3[0]; //lh funct3 001

wire i\_lw=itype\_l&~Funct3[2]& Funct3[1]& ~Funct3[0]; //lw funct3 010

//i\_i Type addi alli

wire itype\_r = ~Op[6]&~Op[5]&Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0]; //op 0010011

wire i\_addi = itype\_r& ~Funct3[2]& ~Funct3[1]& ~Funct3[0]; // addi funct3 000

wire i\_slli = itype\_r& ~Funct3[2]& ~Funct3[1]& Funct3[0]; //slli funct3 001

//s Type

wire stype = ~Op[6]&Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];//op 0100011

wire i\_sb = stype& ~Funct3[2]& ~Funct3[1]&~Funct3[0];//sb funct3 000

wire i\_sh = stype& ~Funct3[2]&~Funct3[1]&Funct3[0]; //sh funct3 001

wire i\_sw = stype& ~Funct3[2]& Funct3[1]&~Funct3[0]; // sw funct3 010

//UJ

wire i\_jal=Op[6]&Op[5]&~Op[4]&Op[3]&Op[2]&Op[1]&Op[0];// jal op 1101111

wire i\_jalr=Op[6]&Op[5]&~Op[4]&~Op[3]&Op[2]&Op[1]&Op[0];//jalr op 1100111

//SB

wire sbtype=Op[6]&Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];//op 1100011

wire i\_beq=sbtype&~Funct3[2]&~Funct3[1]&~Funct3[0];//beq funct3 000

wire i\_bne=sbtype&~Funct3[2]&~Funct3[1]&Funct3[0];//bne funct3 001

wire i\_blt=sbtype&Funct3[2]&~Funct3[1]&~Funct3[0];//blt funct3 100

wire i\_bge=sbtype&Funct3[2]&~Funct3[1]&Funct3[0];//bge funct3 101

wire i\_bltu=sbtype&Funct3[2]&Funct3[1]&~Funct3[0];//bltu funct3 110

wire i\_bgeu=sbtype&Funct3[2]&Funct3[1]&Funct3[0];//bgeu funct3 111

//

assign RegWrite = rtype|itype\_l|itype\_r|i\_jal|i\_jalr;// 所有 Rtype ，load指令，addi slli 指令，jal指令，jalr指令 需要写入寄存器

assign MemWrite = stype;// save指令 需要写入内存

assign ALUSrc = itype\_l|itype\_r|stype|i\_jal|i\_jalr;//ALU B is from ext module

assign Unsigned = i\_bltu | i\_bgeu;// ALU A B are unsigned

//ALUOp

assign ALUOp[4] = i\_srl | i\_sra;

assign ALUOp[3] = i\_sll | i\_slli;

assign ALUOp[2] = sbtype;

assign ALUOp[1] = i\_add|itype\_l|i\_addi|stype|i\_jalr;

assign ALUOp[0] = i\_add|itype\_l|i\_addi|stype|i\_jalr;

// EXT

assign EXTOp[5] = i\_slli;

assign EXTOp[4] = itype\_l|i\_addi|i\_jalr;

assign EXTOp[3] = stype;

assign EXTOp[2] = sbtype;

assign EXTOp[1] = 0;

assign EXTOp[0] = i\_jal;

// DMType

assign DMType[2] = 1'b0;

assign DMType[1] = i\_lb | i\_sb;

assign DMType[0] = i\_lh | i\_sh | i\_lb | i\_sb;

//WDSel

assign WDSel[1] = i\_jal|i\_jalr;

assign WDSel[0] = itype\_l;

//NPCOp

assign NPCOp[2] = i\_jalr;

assign NPCOp[1] = i\_jal;

assign NPCOp[0] = (i\_beq&Zero) | (i\_bne&~Zero) | (i\_blt&LT) |(i\_bge&GE) | (i\_bltu&LT) |(i\_bgeu&GE);

endmodule

## EXT.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/12/23 21:40:00

// Design Name:

// Module Name: EXT

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

`define EXT\_CTRL\_ITYPE\_SHAMT 6'b100000

`define EXT\_CTRL\_ITYPE 6'b010000

`define EXT\_CTRL\_STYPE 6'b001000

`define EXT\_CTRL\_BTYPE 6'b000100

`define EXT\_CTRL\_JTYPE 6'b000001

module EXT(

input [4:0]iimm\_shamt,

input [11:0] iimm,//instr[31:20] 12bits

input [11:0] simm,//instr[31:25,11:7],12bits

input [11:0] bimm,//instr[31, 7, 30:25, 11:8],12bits

input [19:0] jimm,

input [5:0] EXTOp,

output reg [31:0] immout

);

always@(\*)

begin

case(EXTOp)

`EXT\_CTRL\_ITYPE\_SHAMT:immout<={27'b0,iimm\_shamt[4:0]};

`EXT\_CTRL\_ITYPE:immout<={{20{iimm[11]}},iimm};

`EXT\_CTRL\_STYPE:immout<={{20{simm[11]}},simm};

`EXT\_CTRL\_BTYPE:immout<={{19{bimm[11]}},bimm,1'b0};

`EXT\_CTRL\_JTYPE:immout<={{11{jimm[19]}},jimm,1'b0};

default:immout<=32'h00000000;

endcase

end

endmodule

## RF.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/12/23 20:38:01

// Design Name:

// Module Name: RF

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module RF(

input clk,

input rst,//reset signal

input RFWr,//Register Write

input [15:0] sw\_i,

input [4:0] A1,A2,A3,// rs1,rs2,rd

input [31:0] WD,//write data

output [31:0]RD1,RD2//data output port

);

reg[31:0] rf[31:0];

integer i;

always @(posedge clk or negedge rst)begin

if(!rst)begin

for(i = 0 ; i < 32 ; i = i + 1)begin

rf[i] <= 0;

end

end

else if(RFWr && (!sw\_i[1]))begin

rf[A3] <= WD;

end

end

assign RD1 = (A1!=0)? rf[A1]:0;

assign RD2 = (A2!=0)? rf[A2]:0;

endmodule

## ALU.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/12/23 21:18:53

// Design Name:

// Module Name: ALU

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ALU(

input signed [31:0] A,B,//alu input num

input [4:0] ALUOp,//alu how to do

input Unsigned,

output reg signed [31:0] C,// alu result

output reg Zero ,

output reg LT,

output reg GE

);

wire [31:0] unsigned\_A = A;

wire [31:0] unsigned\_B = B;

always @(\*) begin

if(Unsigned)begin

case(ALUOp)

5'b00011: C = unsigned\_A + unsigned\_B;

5'b00100: C = unsigned\_A - unsigned\_B;

5'b01000: C = unsigned\_A << unsigned\_B;

5'b10000: C = unsigned\_A >> unsigned\_B;

endcase

end

else begin

case(ALUOp)

5'b00011: C = A + B;

5'b00100: C = A - B;

5'b01000: C = A << B;

5'b10000: C = A >> B;

endcase

end

Zero = (C==0)?1:0;

LT = (C<0)?1:0;

GE = (C>=0)?1:0;

end

endmodule

## DM.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/12/23 21:32:30

// Design Name:

// Module Name: DM

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module DM(

input clk,

input DMWr,//write signal

input [5:0] addr,// ram address

input [31:0] din,//data to ram

input [2:0] DMType,//data type

output reg [31:0] dout//data from ram

);

reg[7:0] dmem[6:0];

always @(posedge clk)begin

if(DMWr)begin

case(DMType)

3'b000:begin //word

dmem[addr] = din[7:0];

dmem[addr + 1] = din[15:8];

dmem[addr + 2] = din[23:16];

dmem[addr + 3] = din[31:24];end

3'b001:begin // half word

dmem[addr] = din[7:0];

dmem[addr + 1] = din[15:8];end

3'b011:dmem[addr] <= din[7:0]; //byte

endcase

end

end

always @(\*)begin

case(DMType)

3'b000:dout = {dmem[addr+3][7:0],dmem[addr+2][7:0],dmem[addr+1][7:0],dmem[addr][7:0]};

3'b001:dout = {{16{dmem[addr + 1][7]}},dmem[addr + 1][7:0],dmem[addr][7:0]};

3'b011:dout = {{24{dmem[addr][7]}},dmem[addr][7:0]};

endcase

end

endmodule

## PC.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/12/23 22:20:20

// Design Name:

// Module Name: PC

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module PC(

input clk,

input rstn,

input [15:0]sw\_i,

input [31:0]NPC,

input [31:0]PC,

output reg[31:0]PCout

);

always@(posedge clk or negedge rstn)

begin

if(!rstn)begin

case(sw\_i[5:2])

4'b0000:PCout <= 32'h0000\_0000; //beq

4'b0001:PCout <= 32'h0000\_0080; //bne

4'b0010:PCout <= 32'h0000\_0100; //blt

4'b0011:PCout <= 32'h0000\_0180; //bge

4'b0100:PCout <= 32'h0000\_0200; //bltu

4'b0101:PCout <= 32'h0000\_0280; //bgeu

4'b0110:PCout <= 32'h0000\_0300; //jal

4'b0111:PCout <= 32'h0000\_037c; //jalr

4'b1000:PCout <= 32'h0000\_03f0; //sll

4'b1001:PCout <= 32'h0000\_0410; //srl

4'b1010:PCout <= 32'h0000\_0430; //sra

endcase

end

else

begin

if(sw\_i[1]==1'b0)

PCout<=NPC;

else

PCout<=PC;

end

end

endmodule

## NPC.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/12/23 22:14:17

// Design Name:

// Module Name: NPC

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

`define NPC\_PLUS4 3'b000

`define NPC\_BRANCH 3'b001

`define NPC\_JAL 3'b010

`define NPC\_JALR 3'b100

module NPC(

input [31:0]PC,

input [2:0]NPCOp,

input [31:0]immout,

input [31:0]aluout,

output reg [31:0]NPC

);

always@(\*)

begin

case(NPCOp)

`NPC\_PLUS4:NPC<=PC+4;

`NPC\_BRANCH:NPC<=PC+immout;

`NPC\_JAL:NPC<=PC+immout;

`NPC\_JALR:NPC<=aluout;

endcase

end

endmodule

## Seg7x16.v

`timescale 1ns/1ps

module seg7x16(

input clk,

input rstn,

input disp\_mode,

input [63:0] i\_data,

output [7:0]o\_seg,

output [7:0]o\_sel

);

reg [14:0] cnt;

wire seg7\_clk;

always@(posedge clk,negedge rstn)

if (!rstn)

cnt<=0;

else

cnt<=cnt + 1'b1;

assign seg7\_clk = cnt[14];

reg [2:0] seg7\_addr;

always @ (posedge seg7\_clk,negedge rstn)

if(!rstn)

seg7\_addr<=0;

else

seg7\_addr<= seg7\_addr + 1'b1;

reg [7:0] o\_sel\_r;

always @(\*)

case(seg7\_addr)//?????

7:o\_sel\_r = 8'b01111111;

6:o\_sel\_r = 8'b10111111;

5:o\_sel\_r = 8'b11011111;

4:o\_sel\_r = 8'b11101111;

3:o\_sel\_r = 8'b11110111;

2:o\_sel\_r = 8'b11111011;

1:o\_sel\_r = 8'b11111101;

0:o\_sel\_r = 8'b11111110;

endcase

reg[63:0]i\_data\_store;

always@(posedge clk,negedge rstn)

if(!rstn)

i\_data\_store<=0;

else i\_data\_store <=i\_data;

reg [7:0] seg\_data\_r;

always@(\*)

if (disp\_mode==1'b1) begin

case(seg7\_addr)

0:seg\_data\_r = i\_data\_store[7:0];

1:seg\_data\_r = i\_data\_store[15:8];

2:seg\_data\_r = i\_data\_store[23:16];

3:seg\_data\_r = i\_data\_store[31:24];

4:seg\_data\_r = i\_data\_store[39:32];

5:seg\_data\_r = i\_data\_store[47:40];

6:seg\_data\_r = i\_data\_store[55:48];

7:seg\_data\_r = i\_data\_store[63:56];

endcase

end

else begin

case(seg7\_addr)

0:seg\_data\_r = i\_data\_store[3:0];

1:seg\_data\_r = i\_data\_store[7:4];

2:seg\_data\_r = i\_data\_store[11:8];

3:seg\_data\_r = i\_data\_store[15:12];

4:seg\_data\_r = i\_data\_store[19:16];

5:seg\_data\_r = i\_data\_store[23:20];

6:seg\_data\_r = i\_data\_store[27:24];

7:seg\_data\_r = i\_data\_store[31:28];

endcase

end

reg [7:0] o\_seg\_r;

always@(posedge clk,negedge rstn)

if(!rstn)

o\_seg\_r<= 8'hff;

else if (disp\_mode==1'b0) begin

case(seg\_data\_r)

4'h0:o\_seg\_r<=8'hC0;

4'h1:o\_seg\_r<=8'hF9;

4'h2:o\_seg\_r<=8'hA4;

4'h3:o\_seg\_r<=8'hB0;

4'h4:o\_seg\_r<=8'h99;

4'h5:o\_seg\_r<=8'h92;

4'h6:o\_seg\_r<=8'h82;

4'h7:o\_seg\_r<=8'hF8;

4'h8:o\_seg\_r<=8'h80;

4'h9:o\_seg\_r<=8'h90;

4'hA:o\_seg\_r<=8'h88;

4'hB:o\_seg\_r<=8'h83;

4'hC:o\_seg\_r<=8'hC6;

4'hD:o\_seg\_r<=8'hA1;

4'hE:o\_seg\_r<=8'h86;

4'hF:o\_seg\_r<=8'h8E;

default: o\_seg\_r<=8'hFF;

endcase end

else begin

o\_seg\_r<=seg\_data\_r;

end

assign o\_sel =o\_sel\_r;

assign o\_seg =o\_seg\_r;

endmodule