1.文件夹名字为SampleCPU-main

里面有11个文件，和一个文件夹lib

这11个文件分别为

##########

CTRL.v

`include "lib/defines.vh"

module CTRL(

input wire rst,

input wire stallreq\_from\_ex,

input wire stallreq\_from\_id,

// output reg flush,

// output reg [31:0] new\_pc,

output reg [`StallBus-1:0] stall

);

//stall[0]为1表示没有暂停

//stall[1]为1 if段暂停

//stall[2]为1 id段暂停

//stall[3]为1 ex段暂停

//stall[4]为1 mem段暂停

//stall[5]为1 wb段暂停

always @ (\*) begin

if (rst) begin

stall <= `StallBus'b0;

end

else if(stallreq\_from\_ex == 1'b1) begin

stall <= 6'b001111;

end

else if(stallreq\_from\_id == 1'b1) begin

stall <= 6'b000111;

end else begin

stall <= 6'b000000;

end

end

endmodule

############

EX.v

`include "lib/defines.vh"

module EX(

input wire clk,

input wire rst,

// input wire flush,

input wire [`StallBus-1:0] stall,

input wire [`ID\_TO\_EX\_WD-1:0] id\_to\_ex\_bus,

output wire [`EX\_TO\_MEM\_WD-1:0] ex\_to\_mem\_bus,

output wire data\_sram\_en,

output wire [3:0] data\_sram\_wen,

output wire [31:0] data\_sram\_addr,

output wire [37:0] ex\_to\_id,

output wire [31:0] data\_sram\_wdata,

output wire stallreq\_from\_ex,

output wire ex\_is\_load,

output wire [65:0] hilo\_ex\_to\_id

);

reg [`ID\_TO\_EX\_WD-1:0] id\_to\_ex\_bus\_r;

always @ (posedge clk) begin

if (rst) begin

id\_to\_ex\_bus\_r <= `ID\_TO\_EX\_WD'b0;

end

// else if (flush) begin

// id\_to\_ex\_bus\_r <= `ID\_TO\_EX\_WD'b0;

// end

else if (stall[2]==`Stop && stall[3]==`NoStop) begin

id\_to\_ex\_bus\_r <= `ID\_TO\_EX\_WD'b0;

end

else if (stall[2]==`NoStop) begin

id\_to\_ex\_bus\_r <= id\_to\_ex\_bus;

end

end

wire [31:0] ex\_pc, inst;

wire [11:0] alu\_op;

wire [2:0] sel\_alu\_src1;

wire [3:0] sel\_alu\_src2;

wire data\_ram\_en;

wire [3:0] data\_ram\_wen,data\_ram\_readen;

wire rf\_we;

wire [4:0] rf\_waddr;

wire sel\_rf\_res;

wire [31:0] rf\_rdata1, rf\_rdata2;

reg is\_in\_delayslot;

assign {

data\_ram\_readen,//168:165

inst\_mthi, //164

inst\_mtlo, //163

inst\_multu, //162

inst\_mult, //161

inst\_divu, //160

inst\_div, //159

ex\_pc, // 148:117

inst, // 116:85

alu\_op, // 84:83

sel\_alu\_src1, // 82:80

sel\_alu\_src2, // 79:76

data\_ram\_en, // 75

data\_ram\_wen, // 74:71

rf\_we, // 70

rf\_waddr, // 69:65

sel\_rf\_res, // 64

rf\_rdata1, // 63:32

rf\_rdata2 // 31:0

} = id\_to\_ex\_bus\_r;

assign ex\_is\_load = (inst[31:26] == 6'b10\_0011) ? 1'b1 : 1'b0;

wire [31:0] imm\_sign\_extend, imm\_zero\_extend, sa\_zero\_extend;

assign imm\_sign\_extend = {{16{inst[15]}},inst[15:0]};

assign imm\_zero\_extend = {16'b0, inst[15:0]};

assign sa\_zero\_extend = {27'b0,inst[10:6]};

wire [31:0] alu\_src1, alu\_src2;

wire [31:0] alu\_result, ex\_result;

assign alu\_src1 = sel\_alu\_src1[1] ? ex\_pc :

sel\_alu\_src1[2] ? sa\_zero\_extend : rf\_rdata1;

assign alu\_src2 = sel\_alu\_src2[1] ? imm\_sign\_extend :

sel\_alu\_src2[2] ? 32'd8 :

sel\_alu\_src2[3] ? imm\_zero\_extend : rf\_rdata2;

alu u\_alu(

.alu\_control (alu\_op ),

.alu\_src1 (alu\_src1 ),

.alu\_src2 (alu\_src2 ),

.alu\_result (alu\_result )

);

assign ex\_result = alu\_result;

assign ex\_to\_mem\_bus = {

data\_ram\_readen,//79:76

ex\_pc, // 75:44

data\_ram\_en, // 43

data\_ram\_wen, // 42:39

sel\_rf\_res, // 38

rf\_we, // 37

rf\_waddr, // 36:32

ex\_result // 31:0

};

assign ex\_to\_id ={

rf\_we, // 37

rf\_waddr, // 36:32

ex\_result // 31:0

};

assign data\_sram\_en = data\_ram\_en;

assign data\_sram\_wen = (data\_ram\_readen==4'b0101 && ex\_result[1:0] == 2'b00 )? 4'b0001

:(data\_ram\_readen==4'b0101 && ex\_result[1:0] == 2'b01 )? 4'b0010

:(data\_ram\_readen==4'b0101 && ex\_result[1:0] == 2'b10 )? 4'b0100

:(data\_ram\_readen==4'b0101 && ex\_result[1:0] == 2'b11 )? 4'b1000

:(data\_ram\_readen==4'b0111 && ex\_result[1:0] == 2'b00 )? 4'b0011

:(data\_ram\_readen==4'b0111 && ex\_result[1:0]== 2'b10 )? 4'b1100

: data\_ram\_wen;//写使能信号

assign data\_sram\_addr = ex\_result; //内存的地址

assign data\_sram\_wdata = data\_sram\_wen==4'b1111 ? rf\_rdata2

:data\_sram\_wen==4'b0001 ? {24'b0,rf\_rdata2[7:0]}

:data\_sram\_wen==4'b0010 ? {16'b0,rf\_rdata2[7:0],8'b0}

:data\_sram\_wen==4'b0100 ? {8'b0,rf\_rdata2[7:0],16'b0}

:data\_sram\_wen==4'b1000 ? {rf\_rdata2[7:0],24'b0}

:data\_sram\_wen==4'b0011 ? {16'b0,rf\_rdata2[15:0]}

:data\_sram\_wen==4'b1100 ? {rf\_rdata2[15:0],16'b0}

:32'b0;

wire hi\_wen,lo\_wen,inst\_mthi,inst\_mtlo;

wire [31:0] hi\_data,lo\_data;

assign hi\_wen = inst\_divu | inst\_div | inst\_mult | inst\_multu | inst\_mthi;//hi寄存器 写

assign lo\_wen = inst\_divu | inst\_div | inst\_mult | inst\_multu | inst\_mtlo;//lo寄存器 写

assign hi\_data = (inst\_div|inst\_divu) ? div\_result[63:32] //高32位为余数

: (inst\_mult|inst\_multu) ? mul\_result[63:32]

: (inst\_mthi) ? rf\_rdata1

: (32'b0);

assign lo\_data = (inst\_div|inst\_divu) ? div\_result[31:0] //低32位为商

: (inst\_mult|inst\_multu) ? mul\_result[31:0]

: (inst\_mtlo) ? rf\_rdata1

: (32'b0);

assign hilo\_ex\_to\_id = {

hi\_wen, // 65

lo\_wen, // 64

hi\_data, // 63:32

lo\_data // 31:0

};

// MUL part

wire inst\_mult,inst\_multu;

wire [63:0] mul\_result;

//\*\*\*\*\*\*\*\*\*\*\*\*\*原有的 booth-Wallace 乘法器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// wire mul\_signed; // 有符号乘法标记

// assign mul\_signed = inst\_mult ? 1

// : inst\_multu ? 0

// : 0;

// wire [31:0] mul\_data1,mul\_data2;

// assign mul\_data1 = (inst\_mult | inst\_multu) ? rf\_rdata1 : 32'b0;

// assign mul\_data2 = (inst\_mult | inst\_multu) ? rf\_rdata2 : 32'b0;

// mul u\_mul(

// .clk (clk ),

// .resetn (~rst ),

// .mul\_signed (mul\_signed ),

// .ina (mul\_opdata1\_o ), // 乘法源操作数1

// .inb (mul\_opdata2\_o ), // 乘法源操作数2

// .result (mul\_result ) // 乘法结果 64bit

// );

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//自己家的32周期移位乘法器

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

reg stallreq\_for\_mul;

wire mul\_ready\_i;

reg signed\_mul\_o; //是否是有符号乘法

reg [31:0] mul\_opdata1\_o;

reg [31:0] mul\_opdata2\_o;

reg mul\_start\_o;

mymul my\_mul(

.rst (rst ),

.clk (clk ),

.signed\_mul\_i (signed\_mul\_o ),

.a\_o (mul\_opdata1\_o ),

.b\_o (mul\_opdata2\_o ),

.start\_i (mul\_start\_o ),

.result\_o (mul\_result ),

.ready\_o (mul\_ready\_i )

);

always @ (\*) begin

if (rst) begin

stallreq\_for\_mul = `NoStop;

mul\_opdata1\_o = `ZeroWord;

mul\_opdata2\_o = `ZeroWord;

mul\_start\_o = `MulStop;

signed\_mul\_o = 1'b0;

end

else begin

stallreq\_for\_mul = `NoStop;

mul\_opdata1\_o = `ZeroWord;

mul\_opdata2\_o = `ZeroWord;

mul\_start\_o = `MulStop;

signed\_mul\_o = 1'b0;

case ({inst\_mult,inst\_multu})

2'b10:begin

if (mul\_ready\_i == `MulResultNotReady) begin

mul\_opdata1\_o = rf\_rdata1;

mul\_opdata2\_o = rf\_rdata2;

mul\_start\_o = `MulStart;

signed\_mul\_o = 1'b1;

stallreq\_for\_mul = `Stop;

end

else if (mul\_ready\_i == `MulResultReady) begin

mul\_opdata1\_o = rf\_rdata1;

mul\_opdata2\_o = rf\_rdata2;

mul\_start\_o = `MulStop;

signed\_mul\_o = 1'b1;

stallreq\_for\_mul = `NoStop;

end

else begin

mul\_opdata1\_o = `ZeroWord;

mul\_opdata2\_o = `ZeroWord;

mul\_start\_o = `MulStop;

signed\_mul\_o = 1'b0;

stallreq\_for\_mul = `NoStop;

end

end

2'b01:begin

if (mul\_ready\_i == `MulResultNotReady) begin

mul\_opdata1\_o = rf\_rdata1;

mul\_opdata2\_o = rf\_rdata2;

mul\_start\_o = `MulStart;

signed\_mul\_o = 1'b0;

stallreq\_for\_mul = `Stop;

end

else if (mul\_ready\_i == `MulResultReady) begin

mul\_opdata1\_o = rf\_rdata1;

mul\_opdata2\_o = rf\_rdata2;

mul\_start\_o = `MulStop;

signed\_mul\_o = 1'b0;

stallreq\_for\_mul = `NoStop;

end

else begin

mul\_opdata1\_o = `ZeroWord;

mul\_opdata2\_o = `ZeroWord;

mul\_start\_o = `MulStop;

signed\_mul\_o = 1'b0;

stallreq\_for\_mul = `NoStop;

end

end

default:begin

end

endcase

end

end

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DIV part

wire [63:0] div\_result;

wire inst\_div, inst\_divu; //inst\_div为有符号除 inst\_divu无符号

wire div\_ready\_i;

reg stallreq\_for\_div;

assign stallreq\_from\_ex = stallreq\_for\_div | stallreq\_for\_mul;

reg [31:0] div\_opdata1\_o; //被除数

reg [31:0] div\_opdata2\_o; //除数

reg div\_start\_o;

reg signed\_div\_o; //是否是有符号除法

div u\_div(

.rst (rst ), //复位

.clk (clk ), //时钟

.signed\_div\_i (signed\_div\_o ), //是否为有符号除法运算，1位有符号

.opdata1\_i (div\_opdata1\_o ), //被除数

.opdata2\_i (div\_opdata2\_o ), //除数

.start\_i (div\_start\_o ), //是否开始除法运算

.annul\_i (1'b0 ), //是否取消除法运算，1位取消

.result\_o (div\_result ), // 除法结果 64bit

.ready\_o (div\_ready\_i ) // 除法是否结束

);

always @ (\*) begin

if (rst) begin

stallreq\_for\_div = `NoStop;

div\_opdata1\_o = `ZeroWord;

div\_opdata2\_o = `ZeroWord;

div\_start\_o = `DivStop;

signed\_div\_o = 1'b0;

end

else begin

stallreq\_for\_div = `NoStop;

div\_opdata1\_o = `ZeroWord;

div\_opdata2\_o = `ZeroWord;

div\_start\_o = `DivStop;

signed\_div\_o = 1'b0;

case ({inst\_div,inst\_divu})

2'b10:begin

if (div\_ready\_i == `DivResultNotReady) begin

div\_opdata1\_o = rf\_rdata1;

div\_opdata2\_o = rf\_rdata2;

div\_start\_o = `DivStart;

signed\_div\_o = 1'b1;

stallreq\_for\_div = `Stop;

end

else if (div\_ready\_i == `DivResultReady) begin

div\_opdata1\_o = rf\_rdata1;

div\_opdata2\_o = rf\_rdata2;

div\_start\_o = `DivStop;

signed\_div\_o = 1'b1;

stallreq\_for\_div = `NoStop;

end

else begin

div\_opdata1\_o = `ZeroWord;

div\_opdata2\_o = `ZeroWord;

div\_start\_o = `DivStop;

signed\_div\_o = 1'b0;

stallreq\_for\_div = `NoStop;

end

end

2'b01:begin

if (div\_ready\_i == `DivResultNotReady) begin

div\_opdata1\_o = rf\_rdata1;

div\_opdata2\_o = rf\_rdata2;

div\_start\_o = `DivStart;

signed\_div\_o = 1'b0;

stallreq\_for\_div = `Stop;

end

else if (div\_ready\_i == `DivResultReady) begin

div\_opdata1\_o = rf\_rdata1;

div\_opdata2\_o = rf\_rdata2;

div\_start\_o = `DivStop;

signed\_div\_o = 1'b0;

stallreq\_for\_div = `NoStop;

end

else begin

div\_opdata1\_o = `ZeroWord;

div\_opdata2\_o = `ZeroWord;

div\_start\_o = `DivStop;

signed\_div\_o = 1'b0;

stallreq\_for\_div = `NoStop;

end

end

default:begin

end

endcase

end

end

// mul\_result 和 div\_result 可以直接使用

endmodule

##############

ID.v

`include "lib/defines.vh"

module ID(

input wire clk,

input wire rst,

// input wire flush,

input wire [`StallBus-1:0] stall,

input wire ex\_is\_load,

output wire stallreq,

input wire [`IF\_TO\_ID\_WD-1:0] if\_to\_id\_bus,

input wire [31:0] inst\_sram\_rdata,

input wire [`WB\_TO\_RF\_WD-1:0] wb\_to\_rf\_bus,

input wire [37:0] ex\_to\_id,

input wire [37:0] mem\_to\_id,

input wire [37:0] wb\_to\_id,

input wire [65:0] hilo\_ex\_to\_id,

output wire [`ID\_TO\_EX\_WD-1:0] id\_to\_ex\_bus,

output wire [`BR\_WD-1:0] br\_bus,

output wire stallreq\_from\_id

);

reg [`IF\_TO\_ID\_WD-1:0] if\_to\_id\_bus\_r;

wire [31:0] inst;

wire [31:0] id\_pc;

wire ce;

wire wb\_rf\_we;

wire [4:0] wb\_rf\_waddr;

wire [31:0] wb\_rf\_wdata;

wire wb\_id\_we;

wire [4:0] wb\_id\_waddr;

wire [31:0] wb\_id\_wdata;

wire mem\_id\_we;

wire [4:0] mem\_id\_waddr;

wire [31:0] mem\_id\_wdata;

reg q;

wire ex\_id\_we;

wire [4:0] ex\_id\_waddr;

wire [31:0] ex\_id\_wdata;

always @ (posedge clk) begin

if (rst) begin

if\_to\_id\_bus\_r <= `IF\_TO\_ID\_WD'b0;

end

// else if (flush) begin

// ic\_to\_id\_bus <= `IC\_TO\_ID\_WD'b0;

// end

else if (stall[1]==`Stop && stall[2]==`NoStop) begin

if\_to\_id\_bus\_r <= `IF\_TO\_ID\_WD'b0;

end

else if (stall[1]==`NoStop) begin

if\_to\_id\_bus\_r <= if\_to\_id\_bus;

end

end

always @(posedge clk) begin

if (stall[1]==`Stop) begin

q <= 1'b1;

end

else begin

q <= 1'b0;

end

end

assign inst = (q) ?inst: inst\_sram\_rdata;

//assign inst = inst\_sram\_rdata;

assign {

ce,

id\_pc

} = if\_to\_id\_bus\_r;

assign {

wb\_rf\_we,

wb\_rf\_waddr,

wb\_rf\_wdata

} = wb\_to\_rf\_bus;

assign {

wb\_id\_we,

wb\_id\_waddr,

wb\_id\_wdata

} = wb\_to\_id;

assign {

mem\_id\_we,

mem\_id\_waddr,

mem\_id\_wdata

} = mem\_to\_id;

assign {

ex\_id\_we,

ex\_id\_waddr,

ex\_id\_wdata

} = ex\_to\_id;

wire [5:0] opcode;

wire [4:0] rs,rt,rd,sa;

wire [5:0] func;

wire [15:0] imm;

wire [25:0] instr\_index;

wire [19:0] code;

wire [4:0] base;

wire [15:0] offset;

wire [2:0] sel;

wire [63:0] op\_d, func\_d;

wire [31:0] rs\_d, rt\_d, rd\_d, sa\_d;

wire [2:0] sel\_alu\_src1;

wire [3:0] sel\_alu\_src2;

wire [11:0] alu\_op;

wire data\_ram\_en;

wire [3:0] data\_ram\_wen;

wire [3:0] data\_ram\_readen;

wire rf\_we;

wire [4:0] rf\_waddr;

wire sel\_rf\_res;

wire [2:0] sel\_rf\_dst;

wire [31:0] rdata1, rdata2;

wire [31:0] rdata11, rdata22;

wire hi\_r,hi\_wen,lo\_r,lo\_wen;

wire [31:0] hi\_data;

wire [31:0] lo\_data;

wire [31:0] hilo\_data;

assign {

hi\_wen, // 65

lo\_wen, // 64

hi\_data, // 63:32

lo\_data // 31:0

} = hilo\_ex\_to\_id;

assign hi\_r = inst\_mfhi;

assign lo\_r = inst\_mflo;

regfile u\_regfile(

.clk (clk ),

.raddr1 (rs ),

.rdata1 (rdata1 ),

.raddr2 (rt ),

.rdata2 (rdata2 ),

.we (wb\_rf\_we ),

.waddr (wb\_rf\_waddr ),

.wdata (wb\_rf\_wdata ),

.hi\_r ( hi\_r ),

.hi\_we ( hi\_wen ),

.hi\_data ( hi\_data ),

.lo\_r ( lo\_r ),

.lo\_we ( lo\_wen ),

.lo\_data ( lo\_data ),

.hilo\_data ( hilo\_data )

);

wire [31:0] mf\_data;

assign mf\_data = (inst\_mfhi & hi\_wen) ? hi\_data

:(inst\_mfhi) ? hilo\_data

:(inst\_mflo & lo\_wen) ? lo\_data

:(inst\_mflo) ? hilo\_data

:(32'b0);

assign rdata11 = (inst\_mfhi | inst\_mflo) ? mf\_data

:(ex\_id\_we &(ex\_id\_waddr==rs))?ex\_id\_wdata

: (mem\_id\_we &(mem\_id\_waddr==rs)) ? mem\_id\_wdata

: (wb\_id\_we &(wb\_id\_waddr==rs)) ? wb\_id\_wdata

: rdata1;

assign rdata22 = (inst\_mfhi | inst\_mflo) ? mf\_data

:(ex\_id\_we &(ex\_id\_waddr==rt))?ex\_id\_wdata

: (mem\_id\_we &(mem\_id\_waddr==rt)) ? mem\_id\_wdata

: (wb\_id\_we &(wb\_id\_waddr==rt)) ? wb\_id\_wdata

: rdata2;

assign opcode = inst[31:26];

assign rs = inst[25:21];

assign rt = inst[20:16];

assign rd = inst[15:11];

assign sa = inst[10:6];

assign func = inst[5:0];

assign imm = inst[15:0];

assign instr\_index = inst[25:0];

assign code = inst[25:6];

assign base = inst[25:21];

assign offset = inst[15:0];

assign sel = inst[2:0];

wire inst\_ori, inst\_lui, inst\_addiu, inst\_beq,

//inst\_ori 寄存器 rs 中的值与 0 扩展至 32 位的立即数 imm 按位逻辑或，结果写入寄存器 rt 中。

//inst\_lui 将 16 位立即数 imm 写入寄存器 rt 的高 16 位，寄存器 rt 的低 16 位置 0

//inst\_addiu 将寄存器 rs 的值与有符号扩展 ．．．．．至 32 位的立即数 imm 相加，结果写入 rt 寄存器中。

//inst\_beq 如果寄存器 rs 的值等于寄存器 rt 的值则转移，否则顺序执行。转移目标由立即数 offset 左移 2 位

//并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。

inst\_subu,//将寄存器 rs 的值与寄存器 rt 的值相减，结果写入 rd 寄存器中

inst\_jr,// 无条件跳转。跳转目标为寄存器 rs 中的值

inst\_jal,//无条件跳转。跳转目标由该分支指令对应的延迟槽指令的 PC 的最高 4 位与立即数 instr\_index 左移

//2 位后的值拼接得到。同时将该分支对应延迟槽指令之后的指令的 PC 值保存至第 31 号通用寄存

//器中。

inst\_lw,//将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 4 的整数倍

//则触发地址错例外，否则据此虚地址从存储器中读取连续 4 个字节的值，写入到 rt 寄存器中。

inst\_or, //寄存器 rs 中的值与寄存器 rt 中的值按位逻辑或，结果写入寄存器 rd 中

inst\_sll, //由立即数 sa 指定移位量，对寄存器 rt 的值进行逻辑左移，结果写入寄存器 rd 中。

inst\_addu,//将寄存器 rs 的值与寄存器 rt 的值相加，结果写入 rd 寄存器中

inst\_bne,//如果寄存器 rs 的值不等于寄存器 rt 的值则转移，否则顺序执行。转移目标由立即数 offset 左移 2

//位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到

inst\_xor,//寄存器 rs 中的值与寄存器 rt 中的值按位逻辑异或，结果写入寄存器 rd 中。

inst\_xori,//寄存器 rs 中的值与 0 扩展至 32 位的立即数 imm 按位逻辑异或，结果写入寄存器 rt 中。

inst\_nor,//寄存器 rs 中的值与寄存器 rt 中的值按位逻辑或非，结果写入寄存器 rd 中。

inst\_sw,//将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 4 的整数倍

//则触发地址错例外，否则据此虚地址将 rt 寄存器存入存储器中。

inst\_sltu,//将寄存器 rs 的值与寄存器 rt 中的值进行无符号数比较，如果寄存器 rs 中的值小，则寄存器 rd 置 1；

//否则寄存器 rd 置 0。

inst\_slt,//将寄存器 rs 的值与寄存器 rt 中的值进行有符号数比较，如果寄存器 rs 中的值小，则寄存器 rd 置 1；

//否则寄存器 rd 置 0。

inst\_slti,//将寄存器 rs 的值与有符号扩展至 32 位的立即数 imm 进行有符号数比较，如果寄存器 rs 中的值小，

//则寄存器 rt 置 1；否则寄存器 rt 置 0。

inst\_sltiu,//将寄存器 rs 的值与有符号扩展 ．．．．．至 32 位的立即数 imm 进行无符号数比较，如果寄存器 rs 中的值小，

//则寄存器 rt 置 1；否则寄存器 rt 置 0。

inst\_j,//无条件跳转。跳转目标由该分支指令对应的延迟槽指令的 PC 的最高 4 位与立即数 instr\_index 左移

//2 位后的值拼接得到。

inst\_add,//将寄存器 rs 的值与寄存器 rt 的值相加，结果写入寄存器 rd 中。如果产生溢出，则触发整型溢出例

//外（IntegerOverflow）。

inst\_addi,//将寄存器 rs 的值与有符号扩展至 32 位的立即数 imm 相加，结果写入 rt 寄存器中。如果产生溢出，

// 则触发整型溢出例外（IntegerOverflow）。

inst\_sub,//将寄存器 rs 的值与寄存器 rt 的值相减，结果写入 rd 寄存器中。如果产生溢出，则触发整型溢出例

//外（IntegerOverflow）。

inst\_and,//寄存器 rs 中的值与寄存器 rt 中的值按位逻辑与，结果写入寄存器 rd 中。

inst\_andi,//寄存器 rs 中的值与 0 扩展至 32 位的立即数 imm 按位逻辑与，结果写入寄存器 rt 中。

inst\_sllv,//由寄存器 rs 中的值指定移位量，对寄存器 rt 的值进行逻辑左移，结果写入寄存器 rd 中。

inst\_sra,//由立即数 sa 指定移位量，对寄存器 rt 的值进行算术右移，结果写入寄存器 rd 中。

inst\_srav,//由寄存器 rs 中的值指定移位量，对寄存器 rt 的值进行算术右移，结果写入寄存器 rd 中。

inst\_srl,//由立即数 sa 指定移位量，对寄存器 rt 的值进行逻辑右移，结果写入寄存器 rd 中。

inst\_srlv,//由寄存器 rs 中的值指定移位量，对寄存器 rt 的值进行逻辑右移，结果写入寄存器 rd 中。

inst\_bgez,//如果寄存器 rs 的值大于等于 0 则转移，否则顺序执行。转移目标由立即数 offset 左移 2 位并进行有

//符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。

inst\_bgtz,//如果寄存器 rs 的值大于 0 则转移，否则顺序执行。转移目标由立即数 offset 左移 2 位并进行有符号

//扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。

inst\_blez,//如果寄存器 rs 的值小于等于 0 则转移，否则顺序执行。转移目标由立即数 offset 左移 2 位并进行有

//符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。

inst\_bltz,//如果寄存器 rs 的值小于 0 则转移，否则顺序执行。转移目标由立即数 offset 左移 2 位并进行有符号

//扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。

inst\_bltzal,//如果寄存器 rs 的值小于 0 则转移，否则顺序执行。转移目标由立即数 offset 左移 2 位并进行有符号

//扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。无论转移与否，将该分支对应延迟槽

//指令之后的指令的 PC 值保存至第 31 号通用寄存器中。

inst\_bgezal,inst\_jalr,inst\_div,inst\_divu,

inst\_mflo,//将 LO 寄存器的值写入到寄存器 rd 中

inst\_mfhi,//将 HI 寄存器的值写入到寄存器 rd 中

inst\_mult,inst\_multu,inst\_mthi,inst\_mtlo,inst\_lb,

inst\_lbu,//将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，据此虚地址从存储器中读

//取 1 个字节的值并进行 0 扩展，写入到 rt 寄存器中

inst\_lh,//将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 2 的整数倍

//则触发地址错例外，否则据此虚地址从存储器中读取连续 2 个字节的值并进行符号扩展，写入到

//rt 寄存器中。

inst\_lhu,//将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 2 的整数倍

//则触发地址错例外，否则据此虚地址从存储器中读取连续 2 个字节的值并进行 0 扩展，写入到 rt

//寄存器中。

inst\_sb, //将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，据此虚地址将 rt 寄存器的

//最低字节存入存储器中。

inst\_lsa,

inst\_sh; //将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 2 的整数倍

//则触发地址错例外，否则据此虚地址将 rt 寄存器的低半字存入存储器中。

wire op\_add, op\_sub, op\_slt, op\_sltu;

wire op\_and, op\_nor, op\_or, op\_xor;

wire op\_sll, op\_srl, op\_sra, op\_lui;

decoder\_6\_64 u0\_decoder\_6\_64(

.in (opcode ),

.out (op\_d )

);

decoder\_6\_64 u1\_decoder\_6\_64(

.in (func ),

.out (func\_d )

);

decoder\_5\_32 u0\_decoder\_5\_32(

.in (rs ),

.out (rs\_d )

);

decoder\_5\_32 u1\_decoder\_5\_32(

.in (rt ),

.out (rt\_d )

);

assign inst\_ori = op\_d[6'b00\_1101];

assign inst\_lui = op\_d[6'b00\_1111];

assign inst\_addiu = op\_d[6'b00\_1001];

assign inst\_beq = op\_d[6'b00\_0100];

assign inst\_subu = op\_d[6'b00\_0000] && func\_d[6'b10\_0011];

assign inst\_jr = op\_d[6'b00\_0000] && func\_d[6'b00\_1000];

assign inst\_jal = op\_d[6'b00\_0011];

assign inst\_lw = op\_d[6'b10\_0011];

assign inst\_addu = op\_d[6'b00\_0000] && func\_d[6'b10\_0001];

assign inst\_or = op\_d[6'b00\_0000] && func\_d[6'b10\_0101];

assign inst\_sll = op\_d[6'b00\_0000] && func\_d[6'b00\_0000];

assign inst\_bne = op\_d[6'b00\_0101];

assign inst\_xor = op\_d[6'b00\_0000] && func\_d[6'b10\_0110];

assign inst\_xori = op\_d[6'b00\_1110];

assign inst\_nor = op\_d[6'b00\_0000] && func\_d[6'b10\_0111];

assign inst\_sw = op\_d[6'b10\_1011];

assign inst\_sltu = op\_d[6'b00\_0000] && func\_d[6'b10\_1011];

assign inst\_slt = op\_d[6'b00\_0000] && func\_d[6'b10\_1010];

assign inst\_slti = op\_d[6'b00\_1010];

assign inst\_sltiu = op\_d[6'b00\_1011];

assign inst\_j = op\_d[6'b00\_0010];

assign inst\_add = op\_d[6'b00\_0000] && func\_d[6'b10\_0000];

assign inst\_addi = op\_d[6'b00\_1000];

assign inst\_sub = op\_d[6'b00\_0000] && func\_d[6'b10\_0010];

assign inst\_and = op\_d[6'b00\_0000] && func\_d[6'b10\_0100];

assign inst\_andi = op\_d[6'b00\_1100];

assign inst\_sllv = op\_d[6'b00\_0000] && func\_d[6'b00\_0100];

assign inst\_sra = op\_d[6'b00\_0000] && func\_d[6'b00\_0011];

assign inst\_srav = op\_d[6'b00\_0000] && func\_d[6'b00\_0111];

assign inst\_srl = op\_d[6'b00\_0000] && func\_d[6'b00\_0010];

assign inst\_srlv = op\_d[6'b00\_0000] && func\_d[6'b00\_0110];

assign inst\_bgez = op\_d[6'b00\_0001] && rt\_d[5'b00001];

assign inst\_bgtz = op\_d[6'b00\_0111] && rt\_d[5'b00000];

assign inst\_blez = op\_d[6'b00\_0110] && rt\_d[5'b00000];

assign inst\_bltz = op\_d[6'b00\_0001] && rt\_d[5'b00000];

assign inst\_bltzal = op\_d[6'b00\_0001] && rt\_d[5'b10000];

assign inst\_bgezal = op\_d[6'b00\_0001] && rt\_d[5'b10001];

assign inst\_jalr = op\_d[6'b00\_0000] && func\_d[6'b00\_1001];

assign inst\_div = op\_d[6'b00\_0000] && func\_d[6'b01\_1010];

assign inst\_divu = op\_d[6'b00\_0000] && func\_d[6'b01\_1011];

assign inst\_mflo = op\_d[6'b00\_0000] && func\_d[6'b01\_0010];

assign inst\_mfhi = op\_d[6'b00\_0000] && func\_d[6'b01\_0000];

assign inst\_mult = op\_d[6'b00\_0000] && func\_d[6'b01\_1000];

assign inst\_multu = op\_d[6'b00\_0000] && func\_d[6'b01\_1001];

assign inst\_mthi = op\_d[6'b00\_0000] && func\_d[6'b01\_0001];

assign inst\_mtlo = op\_d[6'b00\_0000] && func\_d[6'b01\_0011];

assign inst\_lb = op\_d[6'b10\_0000];

assign inst\_lbu = op\_d[6'b10\_0100];

assign inst\_lh = op\_d[6'b10\_0001];

assign inst\_lhu = op\_d[6'b10\_0101];

assign inst\_sb = op\_d[6'b10\_1000];

assign inst\_sh = op\_d[6'b10\_1001];

assign inst\_lsa = op\_d[6'b01\_1100] && func\_d[6'b11\_0111];

// rs to reg1

assign sel\_alu\_src1[0] =inst\_sh | inst\_sb | inst\_lhu | inst\_lh | inst\_lbu | inst\_bgez | inst\_srlv | inst\_srav | inst\_sllv | inst\_andi | inst\_and | inst\_sub | inst\_addi | inst\_add | inst\_sltiu | inst\_slti | inst\_slt | inst\_sltu | inst\_sw | inst\_nor | inst\_xori | inst\_xor | inst\_ori | inst\_addiu | inst\_subu | inst\_jr | inst\_lw | inst\_addu |

inst\_or | inst\_mflo |inst\_mfhi | inst\_lb |inst\_lsa;

// pc to reg1

assign sel\_alu\_src1[1] = inst\_jal | inst\_bltzal | inst\_bgezal |inst\_jalr;

// sa\_zero\_extend to reg1

assign sel\_alu\_src1[2] =inst\_srl |inst\_sra | inst\_sll;

// rt to reg2

assign sel\_alu\_src2[0] =inst\_lsa|inst\_mfhi|inst\_mflo | inst\_srl | inst\_srlv | inst\_srav | inst\_sra | inst\_sllv | inst\_and | inst\_sub | inst\_add | inst\_slt | inst\_sltu | inst\_nor | inst\_xor | inst\_subu | inst\_addu | inst\_or | inst\_sll |inst\_div | inst\_divu;

// imm\_sign\_extend to reg2

assign sel\_alu\_src2[1] =inst\_sh | inst\_sb | inst\_lhu | inst\_lh | inst\_lbu | inst\_addi | inst\_sltiu | inst\_slti | inst\_sw | inst\_lui | inst\_addiu | inst\_lw |inst\_lb;

// 32'b8 to reg2

assign sel\_alu\_src2[2] = inst\_jal | inst\_bltzal | inst\_bgezal |inst\_jalr;

// imm\_zero\_extend to reg2

assign sel\_alu\_src2[3] = inst\_andi | inst\_xori | inst\_ori;

assign op\_add =inst\_lsa|inst\_sh | inst\_sb | inst\_lhu | inst\_lh | inst\_lbu | inst\_lb | inst\_addi | inst\_add | inst\_addiu | inst\_lw | inst\_addu | inst\_jal | inst\_sw | inst\_bltzal |inst\_bgezal|inst\_jalr;

assign op\_sub =inst\_sub | inst\_subu;

assign op\_slt = inst\_slt | inst\_slti; //有符号比较

assign op\_sltu = inst\_sltu|inst\_sltiu; //无符号比较

assign op\_and = inst\_andi | inst\_and | inst\_mflo |inst\_mfhi;

assign op\_nor = inst\_nor;

assign op\_or = inst\_ori | inst\_or;

assign op\_xor = inst\_xori |inst\_xor;

assign op\_sll = inst\_sllv | inst\_sll;//逻辑左移

assign op\_srl = inst\_srl | inst\_srlv;//逻辑右移

assign op\_sra = inst\_srav | inst\_sra;//算术右移

assign op\_lui = inst\_lui;

assign alu\_op = {op\_add, op\_sub, op\_slt, op\_sltu,

op\_and, op\_nor, op\_or, op\_xor,

op\_sll, op\_srl, op\_sra, op\_lui};

// mem load and store enable

assign data\_ram\_en =inst\_sh | inst\_sb | inst\_lhu | inst\_lh | inst\_lbu | inst\_lw | inst\_sw | inst\_lb;

// mem write enable

assign data\_ram\_wen = inst\_sw ? 4'b1111 : 4'b0000;

//mem read enable

assign data\_ram\_readen = inst\_lw ? 4'b1111

:inst\_lb ? 4'b0001

:inst\_lbu ? 4'b0010

:inst\_lh ? 4'b0011

:inst\_lhu ? 4'b0100

:inst\_sb ? 4'b0101

:inst\_sh ? 4'b0111

:4'b0000;

// regfile sotre enable

assign rf\_we =inst\_lsa|inst\_lhu | inst\_lh | inst\_lbu | inst\_lb| inst\_mfhi | inst\_mflo | inst\_jalr |inst\_bgezal | inst\_bltzal|inst\_srl | inst\_srlv | inst\_srav | inst\_sra | inst\_sllv | inst\_andi | inst\_and | inst\_sub | inst\_addi | inst\_add | inst\_sltiu | inst\_slti | inst\_slt | inst\_sltu | inst\_nor |inst\_xori | inst\_xor | inst\_sll | inst\_ori | inst\_lui | inst\_addiu | inst\_subu | inst\_jal | inst\_lw | inst\_addu | inst\_or;

// store in [rd]

assign sel\_rf\_dst[0] = inst\_lsa|inst\_mfhi | inst\_mflo | inst\_jalr |inst\_srl | inst\_srlv | inst\_srav | inst\_sra | inst\_sllv | inst\_and | inst\_sub | inst\_add | inst\_slt | inst\_sltu | inst\_nor | inst\_xor | inst\_subu | inst\_addu | inst\_or | inst\_sll;

// store in [rt]

assign sel\_rf\_dst[1] = inst\_lhu | inst\_lh | inst\_lbu | inst\_lb |inst\_andi | inst\_addi | inst\_sltiu | inst\_slti | inst\_xori | inst\_ori | inst\_lui | inst\_addiu | inst\_lw;

// store in [31]

assign sel\_rf\_dst[2] = inst\_jal | inst\_bltzal | inst\_bgezal;

// sel for regfile address

assign rf\_waddr = {5{sel\_rf\_dst[0]}} & rd

| {5{sel\_rf\_dst[1]}} & rt

| {5{sel\_rf\_dst[2]}} & 32'd31;

// 0 from alu\_res ; 1 from ld\_res

assign sel\_rf\_res = 1'b0;

//LSA指令 最后一次加指令测试的内容

wire [31:0] rdata111;

assign rdata111 = (inst\_lsa &inst[7:6]==2'b11) ? {rdata11[27:0] ,4'b0}

:(inst\_lsa & inst[7:6]==2'b10) ? {rdata11[28:0] ,3'b0}

:(inst\_lsa & inst[7:6]==2'b01) ? {rdata11[29:0] ,2'b0}

:(inst\_lsa & inst[7:6]==2'b00) ? {rdata11[30:0] ,1'b0}

:rdata11;

assign id\_to\_ex\_bus = {

data\_ram\_readen,//168:165

inst\_mthi, //164

inst\_mtlo, //163

inst\_multu, //162

inst\_mult, //161

inst\_divu, //160

inst\_div, //159

id\_pc, // 158:127

inst, // 126:95

alu\_op, // 94:83

sel\_alu\_src1, // 82:80

sel\_alu\_src2, // 79:76

data\_ram\_en, // 75

data\_ram\_wen, // 74:71

rf\_we, // 70

rf\_waddr, // 69:65

sel\_rf\_res, // 64

rdata111, // 63:32

rdata22 // 31:0

};

wire br\_e;

wire [31:0] br\_addr;

wire rs\_eq\_rt;

wire rs\_ge\_z;

wire rs\_gt\_z;

wire rs\_le\_z;

wire rs\_lt\_z;

wire [31:0] pc\_plus\_4;

assign pc\_plus\_4 = id\_pc + 32'h4;

assign rs\_ge\_z = (rdata11[31] == 1'b0); //大于等于0

assign rs\_gt\_z = (rdata11[31] == 1'b0 & rdata11 != 32'b0 ); //大于0

assign rs\_le\_z = (rdata11[31] == 1'b1 | rdata11 == 32'b0 ); //小于等于0

assign rs\_lt\_z = (rdata11[31] == 1'b1); //小于0

assign rs\_eq\_rt = (rdata11 == rdata22);

assign br\_e = inst\_jalr | (inst\_bgezal & rs\_ge\_z ) | ( inst\_bltzal & rs\_lt\_z) | (inst\_bgtz & rs\_gt\_z ) | (inst\_bltz & rs\_lt\_z) | (inst\_blez & rs\_le\_z) | (inst\_bgez & rs\_ge\_z ) | (inst\_beq & rs\_eq\_rt) | inst\_jr | inst\_jal | (inst\_bne & !rs\_eq\_rt) | inst\_j ;

assign br\_addr = inst\_beq ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b0})

:(inst\_jr |inst\_jalr) ? (rdata11)

: inst\_jal ? ({pc\_plus\_4[31:28],inst[25:0],2'b0})

: inst\_j ? ({pc\_plus\_4[31:28],inst[25:0],2'b0})

:(inst\_bgezal|inst\_bltzal |inst\_blez | inst\_bltz |inst\_bgez |inst\_bgtz ) ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b00})

:inst\_bne ? (pc\_plus\_4 + {{14{inst[15]}},{inst[15:0],2'b00}}) : 32'b0;

assign br\_bus = {

br\_e,

br\_addr

};

assign stallreq\_from\_id = (ex\_is\_load & ex\_id\_waddr == rs) | (ex\_is\_load & ex\_id\_waddr == rt) ;

endmodule

#############

IF.v

`include "lib/defines.vh"

module IF(

input wire clk,

input wire rst,

input wire [`StallBus-1:0] stall,

// input wire flush,

// input wire [31:0] new\_pc,

input wire [`BR\_WD-1:0] br\_bus,

output wire [`IF\_TO\_ID\_WD-1:0] if\_to\_id\_bus,

output wire inst\_sram\_en,

output wire [3:0] inst\_sram\_wen,

output wire [31:0] inst\_sram\_addr,

output wire [31:0] inst\_sram\_wdata

);

reg [31:0] pc\_reg;

reg ce\_reg;

wire [31:0] next\_pc;

wire br\_e;

wire [31:0] br\_addr;

assign {

br\_e,

br\_addr

} = br\_bus;

always @ (posedge clk) begin

if (rst) begin

pc\_reg <= 32'hbfbf\_fffc;

end

else if (stall[0]==`NoStop) begin

pc\_reg <= next\_pc;

end

end

always @ (posedge clk) begin

if (rst) begin

ce\_reg <= 1'b0;

end

else if (stall[0]==`NoStop) begin

ce\_reg <= 1'b1;

end

end

assign next\_pc = br\_e ? br\_addr : pc\_reg + 32'h4;

assign inst\_sram\_en = ce\_reg;

assign inst\_sram\_wen = 4'b0;

assign inst\_sram\_addr = pc\_reg;

assign inst\_sram\_wdata = 32'b0;

assign if\_to\_id\_bus = {

ce\_reg,

pc\_reg

};

endmodule

###############

MEM.v

`include "lib/defines.vh"

module MEM(

input wire clk,

input wire rst,

// input wire flush,

input wire [`StallBus-1:0] stall,

input wire [`EX\_TO\_MEM\_WD-1:0] ex\_to\_mem\_bus,

input wire [31:0] data\_sram\_rdata,

output wire [37:0] mem\_to\_id,

output wire [`MEM\_TO\_WB\_WD-1:0] mem\_to\_wb\_bus

);

reg [`EX\_TO\_MEM\_WD-1:0] ex\_to\_mem\_bus\_r;

always @ (posedge clk) begin

if (rst) begin

ex\_to\_mem\_bus\_r <= `EX\_TO\_MEM\_WD'b0;

end

// else if (flush) begin

// ex\_to\_mem\_bus\_r <= `EX\_TO\_MEM\_WD'b0;

// end

else if (stall[3]==`Stop && stall[4]==`NoStop) begin

ex\_to\_mem\_bus\_r <= `EX\_TO\_MEM\_WD'b0;

end

else if (stall[3]==`NoStop) begin

ex\_to\_mem\_bus\_r <= ex\_to\_mem\_bus;

end

end

wire [31:0] mem\_pc;

wire data\_ram\_en;

wire [3:0] data\_ram\_wen , data\_ram\_readen;

wire sel\_rf\_res;

wire rf\_we;

wire [4:0] rf\_waddr;

wire [31:0] rf\_wdata;

wire [31:0] ex\_result;

assign {

data\_ram\_readen,//79:76

mem\_pc, // 75:44

data\_ram\_en, // 43

data\_ram\_wen, // 42:39

sel\_rf\_res, // 38

rf\_we, // 37

rf\_waddr, // 36:32

ex\_result // 31:0

} = ex\_to\_mem\_bus\_r;

assign rf\_wdata = (data\_ram\_readen==4'b1111 && data\_ram\_en==1'b1) ? data\_sram\_rdata

: (data\_ram\_readen==4'b0001 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b00) ?({{24{data\_sram\_rdata[7]}},data\_sram\_rdata[7:0]})

: (data\_ram\_readen==4'b0001 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b01) ?({{24{data\_sram\_rdata[15]}},data\_sram\_rdata[15:8]})

: (data\_ram\_readen==4'b0001 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b10) ?({{24{data\_sram\_rdata[23]}},data\_sram\_rdata[23:16]})

: (data\_ram\_readen==4'b0001 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b11) ?({{24{data\_sram\_rdata[31]}},data\_sram\_rdata[31:24]})

: (data\_ram\_readen==4'b0010 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b00) ?({24'b0,data\_sram\_rdata[7:0]})

: (data\_ram\_readen==4'b0010 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b01) ?({24'b0,data\_sram\_rdata[15:8]})

: (data\_ram\_readen==4'b0010 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b10) ?({24'b0,data\_sram\_rdata[23:16]})

: (data\_ram\_readen==4'b0010 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b11) ?({24'b0,data\_sram\_rdata[31:24]})

: (data\_ram\_readen==4'b0011 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b00) ?({{16{data\_sram\_rdata[15]}},data\_sram\_rdata[15:0]})

: (data\_ram\_readen==4'b0011 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b10) ?({{16{data\_sram\_rdata[31]}},data\_sram\_rdata[31:16]})

: (data\_ram\_readen==4'b0100 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b00) ?({16'b0,data\_sram\_rdata[15:0]})

: (data\_ram\_readen==4'b0100 && data\_ram\_en==1'b1 && ex\_result[1:0]==2'b10) ?({16'b0,data\_sram\_rdata[31:16]})

: ex\_result;

assign mem\_to\_wb\_bus = {

mem\_pc, // 41:38

rf\_we, // 37

rf\_waddr, // 36:32

rf\_wdata // 31:0

};

assign mem\_to\_id =

{ rf\_we, // 37

rf\_waddr, // 36:32

rf\_wdata // 31:0

};

endmodule

###############

mycpu\_core.v

`include "lib/defines.vh"

module mycpu\_core(

input wire clk,

input wire rst,

input wire [5:0] int,

output wire inst\_sram\_en,

output wire [3:0] inst\_sram\_wen,

output wire [31:0] inst\_sram\_addr,

output wire [31:0] inst\_sram\_wdata,

input wire [31:0] inst\_sram\_rdata,

output wire data\_sram\_en,

output wire [3:0] data\_sram\_wen,

output wire [31:0] data\_sram\_addr,

output wire [31:0] data\_sram\_wdata,

input wire [31:0] data\_sram\_rdata,

output wire [31:0] debug\_wb\_pc,

output wire [3:0] debug\_wb\_rf\_wen,

output wire [4:0] debug\_wb\_rf\_wnum,

output wire [31:0] debug\_wb\_rf\_wdata

);

wire [`IF\_TO\_ID\_WD-1:0] if\_to\_id\_bus;

wire [`ID\_TO\_EX\_WD-1:0] id\_to\_ex\_bus;

wire [`EX\_TO\_MEM\_WD-1:0] ex\_to\_mem\_bus;

wire [37:0] ex\_to\_id;

wire [37:0] mem\_to\_id;

wire [37:0] wb\_to\_id;

wire [`MEM\_TO\_WB\_WD-1:0] mem\_to\_wb\_bus;

wire [`BR\_WD-1:0] br\_bus;

wire [`DATA\_SRAM\_WD-1:0] ex\_dt\_sram\_bus;

wire [`WB\_TO\_RF\_WD-1:0] wb\_to\_rf\_bus;

wire [`StallBus-1:0] stall;

wire stallreq\_from\_id;

wire stallreq\_from\_ex;

wire ex\_is\_load;

wire [65:0] hilo\_ex\_to\_id;

IF u\_IF(

.clk (clk ),

.rst (rst ),

.stall (stall ),

.br\_bus (br\_bus ),

.if\_to\_id\_bus (if\_to\_id\_bus ),

.inst\_sram\_en (inst\_sram\_en ),

.inst\_sram\_wen (inst\_sram\_wen ),

.inst\_sram\_addr (inst\_sram\_addr ),

.inst\_sram\_wdata (inst\_sram\_wdata )

);

ID u\_ID(

.clk (clk ),

.rst (rst ),

.stall (stall ),

.ex\_is\_load (ex\_is\_load ),

.stallreq (stallreq ),

.if\_to\_id\_bus (if\_to\_id\_bus ),

.inst\_sram\_rdata (inst\_sram\_rdata ),

.wb\_to\_rf\_bus (wb\_to\_rf\_bus ),

.ex\_to\_id (ex\_to\_id ),

.mem\_to\_id (mem\_to\_id ),

.wb\_to\_id (wb\_to\_id ),

.hilo\_ex\_to\_id (hilo\_ex\_to\_id ),

.id\_to\_ex\_bus (id\_to\_ex\_bus ),

.br\_bus (br\_bus ),

.stallreq\_from\_id(stallreq\_from\_id)

);

EX u\_EX(

.clk (clk ),

.rst (rst ),

.stall (stall ),

.id\_to\_ex\_bus (id\_to\_ex\_bus ),

.ex\_to\_mem\_bus (ex\_to\_mem\_bus ),

.data\_sram\_en (data\_sram\_en ),

.data\_sram\_wen (data\_sram\_wen ),

.data\_sram\_addr (data\_sram\_addr ),

.ex\_to\_id (ex\_to\_id ),

.data\_sram\_wdata (data\_sram\_wdata ),

.stallreq\_from\_ex(stallreq\_from\_ex),

.ex\_is\_load (ex\_is\_load ),

.hilo\_ex\_to\_id (hilo\_ex\_to\_id )

);

MEM u\_MEM(

.clk (clk ),

.rst (rst ),

.stall (stall ),

.ex\_to\_mem\_bus (ex\_to\_mem\_bus ),

.data\_sram\_rdata (data\_sram\_rdata ),

.mem\_to\_id (mem\_to\_id ),

.mem\_to\_wb\_bus (mem\_to\_wb\_bus )

);

WB u\_WB(

.clk (clk ),

.rst (rst ),

.stall (stall ),

.mem\_to\_wb\_bus (mem\_to\_wb\_bus ),

.wb\_to\_rf\_bus (wb\_to\_rf\_bus ),

.wb\_to\_id (wb\_to\_id ),

.debug\_wb\_pc (debug\_wb\_pc ),

.debug\_wb\_rf\_wen (debug\_wb\_rf\_wen ),

.debug\_wb\_rf\_wnum (debug\_wb\_rf\_wnum ),

.debug\_wb\_rf\_wdata (debug\_wb\_rf\_wdata )

);

CTRL u\_CTRL(

.rst (rst ),

.stallreq\_from\_ex (stallreq\_from\_ex ),

.stallreq\_from\_id (stallreq\_from\_id ),

.stall (stall )

);

endmodule

##########

mycpu\_top.v

`include "lib/defines.vh"

module mycpu\_top(

input wire clk,

input wire resetn,

input wire [5:0] ext\_int,

output wire inst\_sram\_en,

output wire [3:0] inst\_sram\_wen,

output wire [31:0] inst\_sram\_addr,

output wire [31:0] inst\_sram\_wdata,

input wire [31:0] inst\_sram\_rdata,

output wire data\_sram\_en,

output wire [3:0] data\_sram\_wen,

output wire [31:0] data\_sram\_addr,

output wire [31:0] data\_sram\_wdata,

input wire [31:0] data\_sram\_rdata,

output wire [31:0] debug\_wb\_pc,

output wire [3:0] debug\_wb\_rf\_wen,

output wire [4:0] debug\_wb\_rf\_wnum,

output wire [31:0] debug\_wb\_rf\_wdata

);

wire [31:0] inst\_sram\_addr\_v, data\_sram\_addr\_v;

mycpu\_core u\_mycpu\_core(

.clk (clk ),

.rst (~resetn ),

.int (ext\_int ),

.inst\_sram\_en (inst\_sram\_en ),

.inst\_sram\_wen (inst\_sram\_wen ),

.inst\_sram\_addr (inst\_sram\_addr\_v ),

.inst\_sram\_wdata (inst\_sram\_wdata ),

.inst\_sram\_rdata (inst\_sram\_rdata ),

.data\_sram\_en (data\_sram\_en ),

.data\_sram\_wen (data\_sram\_wen ),

.data\_sram\_addr (data\_sram\_addr\_v ),

.data\_sram\_wdata (data\_sram\_wdata ),

.data\_sram\_rdata (data\_sram\_rdata ),

.debug\_wb\_pc (debug\_wb\_pc ),

.debug\_wb\_rf\_wen (debug\_wb\_rf\_wen ),

.debug\_wb\_rf\_wnum (debug\_wb\_rf\_wnum ),

.debug\_wb\_rf\_wdata (debug\_wb\_rf\_wdata )

);

mmu u0\_mmu(

.addr\_i (inst\_sram\_addr\_v ),

.addr\_o (inst\_sram\_addr )

);

mmu u1\_mmu(

.addr\_i (data\_sram\_addr\_v ),

.addr\_o (data\_sram\_addr )

);

endmodule

############

mymul.v

`include "lib/defines.vh"

module mymul(

input wire rst, //复位

input wire clk, //时钟

input wire signed\_mul\_i, //是否为有符号乘法运算，1位有符号

input wire[31:0] a\_o, //被乘数

input wire[31:0] b\_o, //乘数

input wire start\_i, //是否开始乘法运算

output reg[63:0] result\_o, //乘法运算结果

output reg ready\_o //乘法运算是否结束

);

reg [31:0] temp\_opa,temp\_opb;

reg [63:0] pv;

reg [63:0] ap;

reg [5:0] i;//进行到第几位

reg [1:0] state;// 00:空闲 10:开始 11:结束

always @ (posedge clk) begin

if (rst) begin

state <= `MulFree;

result\_o <= {`ZeroWord,`ZeroWord};

ready\_o <= `MulResultNotReady;

end else begin

case(state)

`MulFree: begin //乘法器空闲

if (start\_i== `MulStart) begin

state <= `MulOn;

i <= 6'b00\_0000;

if(signed\_mul\_i == 1'b1 && a\_o[31] == 1'b1) begin //被乘数为负数

temp\_opa = ~a\_o + 1;

end else begin

temp\_opa = a\_o;

end

if(signed\_mul\_i == 1'b1 && b\_o[31] == 1'b1 ) begin //乘数除数为负数

temp\_opb = ~b\_o + 1;

end else begin

temp\_opb = b\_o;

end

ap <= {32'b0,temp\_opa};

ready\_o <= `MulResultNotReady;

result\_o <= {`ZeroWord, `ZeroWord};

pv <= 64'b0;

end

end

`MulOn: begin //乘法运算

if(i != 6'b100000) begin

if(temp\_opb[0]==1'b1) begin

pv <= pv + ap;

ap <= {ap[62:0],1'b0};

temp\_opb <= {1'b0,temp\_opb[31:1]};

end

else begin

ap <= {ap[62:0],1'b0};

temp\_opb <= {1'b0,temp\_opb[31:1]};

end

i <= i + 1;

end

else begin

if ((signed\_mul\_i == 1'b1) && ((a\_o[31] ^ b\_o[31]) == 1'b1))begin

pv <= ~pv + 1;

end

state <= `MulEnd;

i <= 6'b00\_0000;

end

end

`MulEnd: begin //乘法结束

result\_o <= pv;

ready\_o <= `MulResultReady;

if (start\_i == `MulStop) begin

state <= `MulFree;

ready\_o <= `MulResultNotReady;

result\_o <= {`ZeroWord, `ZeroWord};

end

end

endcase

end

end

endmodule

##########

test.s

太长了我会单独发给你

2.lib文件夹中有一个文件夹mul和6个.v文件和1个.vh文件

alu.v

`include "defines.vh"

module alu(

input wire [11:0] alu\_control,

input wire [31:0] alu\_src1,

input wire [31:0] alu\_src2,

output wire [31:0] alu\_result

);

wire op\_add;

wire op\_sub;

wire op\_slt;

wire op\_sltu;

wire op\_and;

wire op\_nor;

wire op\_or;

wire op\_xor;

wire op\_sll;

wire op\_srl;

wire op\_sra;

wire op\_lui;

assign {op\_add, op\_sub, op\_slt, op\_sltu,

op\_and, op\_nor, op\_or, op\_xor,

op\_sll, op\_srl, op\_sra, op\_lui} = alu\_control;

wire [31:0] add\_sub\_result;

wire [31:0] slt\_result;

wire [31:0] sltu\_result;

wire [31:0] and\_result;

wire [31:0] nor\_result;

wire [31:0] or\_result;

wire [31:0] xor\_result;

wire [31:0] sll\_result;

wire [31:0] srl\_result;

wire [31:0] sra\_result;

wire [31:0] lui\_result;

assign and\_result = alu\_src1 & alu\_src2;

assign or\_result = alu\_src1 | alu\_src2;

assign nor\_result = ~or\_result;

assign xor\_result = alu\_src1 ^ alu\_src2;

assign lui\_result = {alu\_src2[15:0], 16'b0};

wire [31:0] adder\_a;

wire [31:0] adder\_b;

wire adder\_cin;

wire [31:0] adder\_result;

wire adder\_cout;

assign adder\_a = alu\_src1;

assign adder\_b = (op\_sub | op\_slt | op\_sltu) ? ~alu\_src2 : alu\_src2;

assign adder\_cin = (op\_sub | op\_slt | op\_sltu) ? 1'b1 : 1'b0;

assign {adder\_cout, adder\_result} = adder\_a + adder\_b + adder\_cin;

assign add\_sub\_result = adder\_result;

assign slt\_result[31:1] = 31'b0;

assign slt\_result[0] = (alu\_src1[31] & ~alu\_src2[31])

| (~(alu\_src1[31]^alu\_src2[31]) & adder\_result[31]);

assign sltu\_result[31:1] = 31'b0;

assign sltu\_result[0] = ~adder\_cout;

assign sll\_result = alu\_src2 << alu\_src1[4:0];

assign srl\_result = alu\_src2 >> alu\_src1[4:0];

assign sra\_result = ($signed(alu\_src2)) >>> alu\_src1[4:0];

assign alu\_result = ({32{op\_add|op\_sub }} & add\_sub\_result)

| ({32{op\_slt }} & slt\_result)

| ({32{op\_sltu }} & sltu\_result)

| ({32{op\_and }} & and\_result)

| ({32{op\_nor }} & nor\_result)

| ({32{op\_or }} & or\_result)

| ({32{op\_xor }} & xor\_result)

| ({32{op\_sll }} & sll\_result)

| ({32{op\_srl }} & srl\_result)

| ({32{op\_sra }} & sra\_result)

| ({32{op\_lui }} & lui\_result);

endmodule

############

decorder\_5\_32.v文件

`include "defines.vh"

module decoder\_5\_32 (

input wire [4:0] in,

output reg [31:0] out

);

always @ (\*) begin

case(in)

5'd00:begin out=32'b00000000000000000000000000000001; end

5'd01:begin out=32'b00000000000000000000000000000010; end

5'd02:begin out=32'b00000000000000000000000000000100; end

5'd03:begin out=32'b00000000000000000000000000001000; end

5'd04:begin out=32'b00000000000000000000000000010000; end

5'd05:begin out=32'b00000000000000000000000000100000; end

5'd06:begin out=32'b00000000000000000000000001000000; end

5'd07:begin out=32'b00000000000000000000000010000000; end

5'd08:begin out=32'b00000000000000000000000100000000; end

5'd09:begin out=32'b00000000000000000000001000000000; end

5'd10:begin out=32'b00000000000000000000010000000000; end

5'd11:begin out=32'b00000000000000000000100000000000; end

5'd12:begin out=32'b00000000000000000001000000000000; end

5'd13:begin out=32'b00000000000000000010000000000000; end

5'd14:begin out=32'b00000000000000000100000000000000; end

5'd15:begin out=32'b00000000000000001000000000000000; end

5'd16:begin out=32'b00000000000000010000000000000000; end

5'd17:begin out=32'b00000000000000100000000000000000; end

5'd18:begin out=32'b00000000000001000000000000000000; end

5'd19:begin out=32'b00000000000010000000000000000000; end

5'd20:begin out=32'b00000000000100000000000000000000; end

5'd21:begin out=32'b00000000001000000000000000000000; end

5'd22:begin out=32'b00000000010000000000000000000000; end

5'd23:begin out=32'b00000000100000000000000000000000; end

5'd24:begin out=32'b00000001000000000000000000000000; end

5'd25:begin out=32'b00000010000000000000000000000000; end

5'd26:begin out=32'b00000100000000000000000000000000; end

5'd27:begin out=32'b00001000000000000000000000000000; end

5'd28:begin out=32'b00010000000000000000000000000000; end

5'd29:begin out=32'b00100000000000000000000000000000; end

5'd30:begin out=32'b01000000000000000000000000000000; end

5'd31:begin out=32'b10000000000000000000000000000000; end

default:begin

out=32'b0;

end

endcase

end

endmodule

########################

`include "defines.vh"

module decoder\_6\_64 (

input wire [5:0] in,

output reg [63:0] out

);

always @ (\*) begin

case(in)

6'd00:begin out=64'b0000000000000000000000000000000000000000000000000000000000000001; end

6'd01:begin out=64'b0000000000000000000000000000000000000000000000000000000000000010; end

6'd02:begin out=64'b0000000000000000000000000000000000000000000000000000000000000100; end

6'd03:begin out=64'b0000000000000000000000000000000000000000000000000000000000001000; end

6'd04:begin out=64'b0000000000000000000000000000000000000000000000000000000000010000; end

6'd05:begin out=64'b0000000000000000000000000000000000000000000000000000000000100000; end

6'd06:begin out=64'b0000000000000000000000000000000000000000000000000000000001000000; end

6'd07:begin out=64'b0000000000000000000000000000000000000000000000000000000010000000; end

6'd08:begin out=64'b0000000000000000000000000000000000000000000000000000000100000000; end

6'd09:begin out=64'b0000000000000000000000000000000000000000000000000000001000000000; end

6'd10:begin out=64'b0000000000000000000000000000000000000000000000000000010000000000; end

6'd11:begin out=64'b0000000000000000000000000000000000000000000000000000100000000000; end

6'd12:begin out=64'b0000000000000000000000000000000000000000000000000001000000000000; end

6'd13:begin out=64'b0000000000000000000000000000000000000000000000000010000000000000; end

6'd14:begin out=64'b0000000000000000000000000000000000000000000000000100000000000000; end

6'd15:begin out=64'b0000000000000000000000000000000000000000000000001000000000000000; end

6'd16:begin out=64'b0000000000000000000000000000000000000000000000010000000000000000; end

6'd17:begin out=64'b0000000000000000000000000000000000000000000000100000000000000000; end

6'd18:begin out=64'b0000000000000000000000000000000000000000000001000000000000000000; end

6'd19:begin out=64'b0000000000000000000000000000000000000000000010000000000000000000; end

6'd20:begin out=64'b0000000000000000000000000000000000000000000100000000000000000000; end

6'd21:begin out=64'b0000000000000000000000000000000000000000001000000000000000000000; end

6'd22:begin out=64'b0000000000000000000000000000000000000000010000000000000000000000; end

6'd23:begin out=64'b0000000000000000000000000000000000000000100000000000000000000000; end

6'd24:begin out=64'b0000000000000000000000000000000000000001000000000000000000000000; end

6'd25:begin out=64'b0000000000000000000000000000000000000010000000000000000000000000; end

6'd26:begin out=64'b0000000000000000000000000000000000000100000000000000000000000000; end

6'd27:begin out=64'b0000000000000000000000000000000000001000000000000000000000000000; end

6'd28:begin out=64'b0000000000000000000000000000000000010000000000000000000000000000; end

6'd29:begin out=64'b0000000000000000000000000000000000100000000000000000000000000000; end

6'd30:begin out=64'b0000000000000000000000000000000001000000000000000000000000000000; end

6'd31:begin out=64'b0000000000000000000000000000000010000000000000000000000000000000; end

6'd32:begin out=64'b0000000000000000000000000000000100000000000000000000000000000000; end

6'd33:begin out=64'b0000000000000000000000000000001000000000000000000000000000000000; end

6'd34:begin out=64'b0000000000000000000000000000010000000000000000000000000000000000; end

6'd35:begin out=64'b0000000000000000000000000000100000000000000000000000000000000000; end

6'd36:begin out=64'b0000000000000000000000000001000000000000000000000000000000000000; end

6'd37:begin out=64'b0000000000000000000000000010000000000000000000000000000000000000; end

6'd38:begin out=64'b0000000000000000000000000100000000000000000000000000000000000000; end

6'd39:begin out=64'b0000000000000000000000001000000000000000000000000000000000000000; end

6'd40:begin out=64'b0000000000000000000000010000000000000000000000000000000000000000; end

6'd41:begin out=64'b0000000000000000000000100000000000000000000000000000000000000000; end

6'd42:begin out=64'b0000000000000000000001000000000000000000000000000000000000000000; end

6'd43:begin out=64'b0000000000000000000010000000000000000000000000000000000000000000; end

6'd44:begin out=64'b0000000000000000000100000000000000000000000000000000000000000000; end

6'd45:begin out=64'b0000000000000000001000000000000000000000000000000000000000000000; end

6'd46:begin out=64'b0000000000000000010000000000000000000000000000000000000000000000; end

6'd47:begin out=64'b0000000000000000100000000000000000000000000000000000000000000000; end

6'd48:begin out=64'b0000000000000001000000000000000000000000000000000000000000000000; end

6'd49:begin out=64'b0000000000000010000000000000000000000000000000000000000000000000; end

6'd50:begin out=64'b0000000000000100000000000000000000000000000000000000000000000000; end

6'd51:begin out=64'b0000000000001000000000000000000000000000000000000000000000000000; end

6'd52:begin out=64'b0000000000010000000000000000000000000000000000000000000000000000; end

6'd53:begin out=64'b0000000000100000000000000000000000000000000000000000000000000000; end

6'd54:begin out=64'b0000000001000000000000000000000000000000000000000000000000000000; end

6'd55:begin out=64'b0000000010000000000000000000000000000000000000000000000000000000; end

6'd56:begin out=64'b0000000100000000000000000000000000000000000000000000000000000000; end

6'd57:begin out=64'b0000001000000000000000000000000000000000000000000000000000000000; end

6'd58:begin out=64'b0000010000000000000000000000000000000000000000000000000000000000; end

6'd59:begin out=64'b0000100000000000000000000000000000000000000000000000000000000000; end

6'd60:begin out=64'b0001000000000000000000000000000000000000000000000000000000000000; end

6'd61:begin out=64'b0010000000000000000000000000000000000000000000000000000000000000; end

6'd62:begin out=64'b0100000000000000000000000000000000000000000000000000000000000000; end

6'd63:begin out=64'b1000000000000000000000000000000000000000000000000000000000000000; end

default:begin

out=64'b0;

end

endcase

end

endmodule

############

defines.vn

`define IF\_TO\_ID\_WD 33

`define ID\_TO\_EX\_WD 169

`define EX\_TO\_MEM\_WD 80

`define MEM\_TO\_WB\_WD 70

`define BR\_WD 33

`define DATA\_SRAM\_WD 69

`define WB\_TO\_RF\_WD 38

`define StallBus 6

`define NoStop 1'b0

`define Stop 1'b1

// 2021-11-29 add

`define ZeroWord 32'b0

//除法div

`define DivFree 2'b00

`define DivByZero 2'b01

`define DivOn 2'b10

`define DivEnd 2'b11

`define DivResultReady 1'b1

`define DivResultNotReady 1'b0

`define DivStart 1'b1

`define DivStop 1'b0

// 2021-11-29 add

/\*\*

\* @Author: zht、szw

\* @Date: 2021-12-10

\*/

`define MulFree 2'b00

`define MulResultNotReady 1'b0

`define MulOn 2'b10

`define MulEnd 2'b11

`define MulResultReady 1'b1

`define MulStop 1'b0

`define MulStart 1'b1

################

div.v

`include "defines.vh"

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

// Company:

// Engineer:

//

// Create Date: 2019/06/25 13:51:28

// Design Name:

// Module Name: div

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module div(

input wire rst, //复位

input wire clk, //时钟

input wire signed\_div\_i, //是否为有符号除法运算，1位有符号

input wire[31:0] opdata1\_i, //被除数

input wire[31:0] opdata2\_i, //除数

input wire start\_i, //是否开始除法运算

input wire annul\_i, //是否取消除法运算，1位取消

output reg[63:0] result\_o, //除法运算结果

output reg ready\_o //除法运算是否结束

);

wire [32:0] div\_temp;

reg [5:0] cnt; //记录试商法进行了几轮

reg[64:0] dividend; //低32位保存除数、中间结果，第k次迭代结束的时候dividend[k:0]保存的就是当前得到的中间结果，

//dividend[31:k+1]保存的是被除数没有参与运算的部分，dividend[63:32]是每次迭代时的被减数

reg [1:0] state; //除法器处于的状态

reg[31:0] divisor;

reg[31:0] temp\_op1;

reg[31:0] temp\_op2;

assign div\_temp = {1'b0, dividend[63: 32]} - {1'b0, divisor};

always @ (posedge clk) begin

if (rst) begin

state <= `DivFree;

result\_o <= {`ZeroWord,`ZeroWord};

ready\_o <= `DivResultNotReady;

end else begin

case(state)

`DivFree: begin //除法器空闲

if (start\_i == `DivStart && annul\_i == 1'b0) begin

if(opdata2\_i == `ZeroWord) begin //如果除数为0

state <= `DivByZero;

end else begin

state <= `DivOn; //除数不为0

cnt <= 6'b000000;

if(signed\_div\_i == 1'b1 && opdata1\_i[31] == 1'b1) begin //被除数为负数

temp\_op1 = ~opdata1\_i + 1;

end else begin

temp\_op1 = opdata1\_i;

end

if (signed\_div\_i == 1'b1 && opdata2\_i[31] == 1'b1 ) begin //除数为负数

temp\_op2 = ~opdata2\_i + 1;

end else begin

temp\_op2 = opdata2\_i;

end

dividend <= {`ZeroWord, `ZeroWord};

dividend[32: 1] <= temp\_op1;

divisor <= temp\_op2;

end

end else begin

ready\_o <= `DivResultNotReady;

result\_o <= {`ZeroWord, `ZeroWord};

end

end

`DivByZero: begin //除数为0

dividend <= {`ZeroWord, `ZeroWord};

state <= `DivEnd;

end

`DivOn: begin //除数不为0

if(annul\_i == 1'b0) begin //进行除法运算

if(cnt != 6'b100000) begin

if (div\_temp[32] == 1'b1) begin

dividend <= {dividend[63:0],1'b0};

end else begin

dividend <= {div\_temp[31:0],dividend[31:0], 1'b1};

end

cnt <= cnt + 1; //除法运算次数

end else begin

if ((signed\_div\_i == 1'b1) && ((opdata1\_i[31] ^ opdata2\_i[31]) == 1'b1)) begin

dividend[31:0] <= (~dividend[31:0] + 1);

end

if ((signed\_div\_i == 1'b1) && ((opdata1\_i[31] ^ dividend[64]) == 1'b1)) begin

dividend[64:33] <= (~dividend[64:33] + 1);

end

state <= `DivEnd;

cnt <= 6'b000000;

end

end else begin

state <= `DivFree;

end

end

`DivEnd: begin //除法结束

result\_o <= {dividend[64:33], dividend[31:0]};

ready\_o <= `DivResultReady;

if (start\_i == `DivStop) begin

state <= `DivFree;

ready\_o <= `DivResultNotReady;

result\_o <= {`ZeroWord, `ZeroWord};

end

end

endcase

end

end

endmodule

#####################

mmu.v

`include "defines.vh"

module mmu (

input wire[31:0] addr\_i,

output wire [31:0] addr\_o

);

wire [2:0] addr\_head\_i, addr\_head\_o;

wire kseg0, kseg1, other\_seg;

assign addr\_head\_i = addr\_i[31:29];

assign kseg0 = addr\_head\_i == 3'b100;

assign kseg1 = addr\_head\_i == 3'b101;

assign other\_seg = ~kseg0 & ~kseg1;

assign addr\_head\_o = {3{kseg0}}&3'b000 | {3{kseg1}}&3'b000 | {3{other\_seg}}&addr\_head\_i;

assign addr\_o = {addr\_head\_o, addr\_i[28:0]};

endmodule

########################

regfile.v

`include "defines.vh"

module regfile(

input wire clk,

input wire [4:0] raddr1,

output wire [31:0] rdata1,

input wire [4:0] raddr2,

output wire [31:0] rdata2,

input wire we,

input wire [4:0] waddr,

input wire [31:0] wdata,

input wire hi\_r,

input wire hi\_we,

input wire [31:0] hi\_data,

input wire lo\_r,

input wire lo\_we,

input wire [31:0] lo\_data,

output wire [31:0] hilo\_data

);

//自己家的hilo寄存器

reg [31:0] hi\_o;

reg [31:0] lo\_o;

// write

always @ (posedge clk) begin

if (hi\_we) begin

hi\_o <= hi\_data;

end

end

always @ (posedge clk) begin

if (lo\_we) begin

lo\_o <= lo\_data;

end

end

//read

assign hilo\_data = (hi\_r) ? hi\_o

:(lo\_r) ? lo\_o

: (32'b0);

reg [31:0] reg\_array [31:0];

// write

always @ (posedge clk) begin

if (we && waddr!=5'b0) begin

reg\_array[waddr] <= wdata;

end

end

// read out 1

assign rdata1 = (raddr1 == 5'b0) ? 32'b0 : reg\_array[raddr1];

// read out2

assign rdata2 = (raddr2 == 5'b0) ? 32'b0 : reg\_array[raddr2];

endmodule

3.mul文件夹中有三个.v文件