Design source:

(AKD\_T1.v)

/\*

都是有符号数，暂时无前递机制

jl:x[rs1]<0则跳转/je:x[rs1]==0则跳转/jg:x[rs1]>0则跳转

j类指令在ID阶段同时完成检查数据、保存pc(所以不需要检测j指令的写回相关)、选择并更新pc

ISA

kind

k(j/l/a/s)(4) funct( 7) immediate(16) rd(5)

k(j/l/a/s)(4) funct(13)[27:15] rs1(5) rs2(5) rd(5)

1. j 1000 1110000000000 rs1 rs2 rd (check rs1, pc\_jump = x[rs2], x[rd] = pc)

2. jl 1000 1000000000000 rs1 rs2 rd (check rs1, pc\_jump = x[rs2], x[rd] = pc)

3. je 1000 0100000000000 rs1 rs2 rd (check rs1, pc\_jump = x[rs2], x[rd] = pc)

4. jg 1000 0010000000000 rs1 rs2 rd (check rs1, pc\_jump = x[rs2], x[rd] = pc)

5. jle 1000 1100000000000 rs1 rs2 rd (check rs1, pc\_jump = x[rs2], x[rd] = pc)

6. jlg 1000 1010000000000 rs1 rs2 rd (check rs1, pc\_jump = x[rs2], x[rd] = pc)

7. jeg 1000 0110000000000 rs1 rs2 rd (check rs1, pc\_jump = x[rs2], x[rd] = pc)

8. ldi 0100 0000000 immediate rd

9. ldhi 0100 0100000 immediate rd

10.ldm 0100 1000000000000 rs1 00000 rd

11.and 0010 1101000000000 rs1 rs2 rd

12.or 0010 1100000000000 rs1 rs2 rd

13.nand 0010 0000000000000 rs1 rs2 rd

14.nor 0010 0001000000000 rs1 rs2 rd

15.xor 0010 0010000000000 rs1 rs2 rd

16.xnor 0010 0110000000000 rs1 rs2 rd

17.add 0010 0011000000000 rs1 rs2 rd

18.sub 0010 0111000000000 rs1 rs2 rd

19.st 0001 0000000000000 rs1 rs2 00000 (M[rs1] = x[rs2])

20.nop 0000 0000000000000 00000 00000 00000

21.hlt 0000 1000000000000 00000 00000 00000

\*/

module AKD\_T1 #(

parameter widthi = 32,

parameter widthd = 32,

parameter depthim = 65536,

parameter depthmem = 65536

)(

input clk, rst,

output cpu\_halt,

output[widthd - 1:0] result\_mem,

output[15:0] detector\_pc,

output[31:0] detector\_pipe\_ID,

output[31:0] detector\_pipe\_EX,

output[31:0] detector\_pipe\_MEM,

output[31:0] detector\_pipe\_WB,

output detector\_hazard\_data,

output detector\_hazard\_structure,

output detector\_jump

);

wire jump;

wire[15:0] pc, pc\_jump;

wire hazard\_data, hazard\_structure;

PC #(widthi - 1) PC(

.clk (clk),

.rst (rst),

.wea\_pc (~(cpu\_halt | hazard\_data | hazard\_structure)),//总是预测不跳转

.pc\_next(jump ? pc\_jump : pc + 1),

.pc (pc)

);

wire[widthi - 1:0] pipe\_ID;

IM\_entire IM\_entire (

.clka (clk), // input wire clka

.ena (~(cpu\_halt | hazard\_data | hazard\_structure)), // input wire ena

.addra(pc), // input wire [15 : 0] addra

.douta(pipe\_ID) // output wire [31 : 0] douta

);

wire[5 + widthd + widthi - 1:0] pipe\_WB;

/\*

[68:64]waddr\_WB\_reg

[63:32]result\_alu

[31: 0]inst

\*/

wire wb\_reg = |pipe\_WB[68:64];

wire[widthd - 1:0] result1\_reg, result2\_reg;

wire[widthd - 1:0] wbdata;

wire[4:0] raddr1\_ID\_reg = pipe\_ID[14:10] & {5{pipe\_ID[31] | pipe\_ID[30] & pipe\_ID[27] | pipe\_ID[29] | pipe\_ID[28]}};

wire[4:0] raddr2\_ID\_reg = pipe\_ID[ 9: 5] & {5{pipe\_ID[31] & jump | pipe\_ID[29] | pipe\_ID[28]}};

wire[4:0] waddr\_ID\_reg = pipe\_ID[ 4: 0] & {5{pipe\_ID[31] & jump | pipe\_ID[30] & ~pipe\_ID[27]}};

wire[4:0] waddr\_WB\_reg = pipe\_ID[ 4: 0] & {5{pipe\_ID[30] & pipe\_ID[27] | pipe\_ID[29]}};

REG #(widthd, 32) REG(

.clk (clk),

.rst (rst),

.ena\_low(~(pipe\_ID[30] & ~pipe\_ID[27] & pipe\_ID[26])),

.raddr1 (raddr1\_ID\_reg),

.raddr2 (raddr2\_ID\_reg),

.waddr (wb\_reg ? pipe\_WB[68:64] : waddr\_ID\_reg),

.regdata({

wb\_reg ? wbdata[31:16] :

(pipe\_ID[30] & ~pipe\_ID[27]) ? {

pipe\_ID[26] ? pipe\_ID[20:5] : {16{pipe\_ID[20]}}

} : 0,//选择高16位数据

wb\_reg ? wbdata[15:0] : (

pc & {16{jump}} |

pipe\_ID[20:5] & {16{pipe\_ID[30] & ~pipe\_ID[27] & ~pipe\_ID[26]}}

)//选择低16位数据

}),

.result1\_reg(result1\_reg),

.result2\_reg(result2\_reg)

);

wire satisfy\_l = result1\_reg[widthd - 1];

wire satisfy\_e = ~(|result1\_reg);

wire satisfy\_g = ~(satisfy\_l | satisfy\_e);

assign jump = pipe\_ID[31] & (satisfy\_l & pipe\_ID[27] | satisfy\_e & pipe\_ID[26] | satisfy\_g & pipe\_ID[25]) & ~hazard\_data;

assign pc\_jump = result2\_reg[15:0];

wire[5 + widthd \* 2 + widthi - 1:0] pipe\_EX;

/\*

[100:96]waddr\_WB\_reg

[ 95:64]result1\_reg

[ 63:32]result2\_reg

[ 31: 0]inst

\*/

reg bubble;

always@(posedge clk, posedge rst) begin

if (rst == 1) bubble <= 0;

else bubble = jump;

end

Pipeline #(5 + widthd \* 2 + widthi) ID\_EX(

.clk(clk),

.rst(rst),

.wea(~cpu\_halt),

.pipe\_prev((hazard\_data | hazard\_structure | bubble) ? 0 : {waddr\_WB\_reg, result1\_reg, result2\_reg, pipe\_ID}),

.pipe(pipe\_EX)

);

wire[widthd - 1:0] result\_alu;

ALU #(widthd) ALU(

.data1\_alu(pipe\_EX[95:64]),

.data2\_alu(pipe\_EX[63:32]),

.ALUop(pipe\_EX[27:15]),

.result\_alu(result\_alu)

);

wire[5 + widthd \* 3 + widthi - 1:0] pipe\_MEM;

/\*

[132:128]waddr\_WB\_reg

[127: 96]result1\_reg

[ 95: 64]result2\_reg

[ 63: 32]result\_alu

[ 31: 0]inst

\*/

Pipeline #(5 + widthd \* 3 + widthi) EX\_MEM(

.clk(clk),

.rst(rst),

.wea(~cpu\_halt),

.pipe\_prev({pipe\_EX[100:32], result\_alu, pipe\_EX[31:0]}),

.pipe(pipe\_MEM)

);

MEM MEM (

.clka (clk), // input wire clka

.wea (~cpu\_halt & pipe\_MEM[28]), // input wire [ 0 : 0] wea

.addra(pipe\_MEM[95:64]), // input wire [15 : 0] addra

.dina (pipe\_MEM[127:96]), // input wire [31 : 0] dina

.douta(result\_mem) // output wire [31 : 0] douta

);

Pipeline #(5 + widthd + widthi) MEM\_WB(

.clk(clk),

.rst(rst),

.wea(~cpu\_halt),

.pipe\_prev({pipe\_MEM[132:128], pipe\_MEM[63:0]}),

.pipe(pipe\_WB)

);

assign cpu\_halt = (~|pipe\_WB[31:28]) & pipe\_WB[27];

assign wbdata = (result\_mem & {widthd{pipe\_WB[30] & pipe\_WB[27]}}) | (pipe\_WB[63:32] & {widthd{pipe\_WB[29]}});

assign hazard\_data =

raddr1\_ID\_reg != 0 & (//不检测零号寄存器相关

(raddr1\_ID\_reg == pipe\_EX[100:96])|

(raddr1\_ID\_reg == pipe\_MEM[132:128])|

(raddr1\_ID\_reg == pipe\_WB[68:64])

) |

raddr2\_ID\_reg != 0 & (//不检测零号寄存器相关

(raddr2\_ID\_reg == pipe\_EX[100:96])|

(raddr2\_ID\_reg == pipe\_MEM[132:128])|

(raddr2\_ID\_reg == pipe\_WB[68:64])

);//不需要考虑EX/MEM/WB之间的数据冒险，因为已经在ID阶段通过流水线暂停解决了，而且是顺序执行

assign hazard\_structure = (waddr\_ID\_reg != 0) & (pipe\_WB[68:64] != 0);

assign detector\_pc = pc;

assign detector\_pipe\_ID = pipe\_ID;

assign detector\_pipe\_EX = pipe\_EX;

assign detector\_pipe\_MEM = pipe\_MEM;

assign detector\_pipe\_WB = pipe\_WB;

assign detector\_hazard\_data = hazard\_data;

assign detector\_hazard\_structure = hazard\_structure;

assign detector\_jump = jump;

Endmodule

(PC.v)

module PC #(

parameter width = 16

)(

input clk, rst, wea\_pc,

input [width - 1:0] pc\_next,

output reg [width - 1:0] pc

);

always@(posedge clk, posedge rst) begin

if (rst == 1) pc <= 0;

else if(wea\_pc == 1) pc <= pc\_next;

end

endmodule

(IM\_entire)

调用Block Memory Generator

Memory Type:Single Port ROM

Port A Width:32

Port A Depth:65536

Write First

Use Enable Pin

无Output Register

Load Init File:../../../../../test\_program/Fibonacci.coe

尾部填充零

(Fibonacci.coe)

memory\_initialization\_radix = 2;

memory\_initialization\_vector =

01000000000000000000000010000100,

01000000000000000000000000100101,

00100111000000000001000010100100,

01000000000000000000000000000001,

01000000000000000000000000100010,

01000000000000000000000011000011,

00100111000000000001000010100100,

00100011000000000000010001000010,

00100111000000000000100000100001,

10000010000000000001000001100000,

00010000000000000000100000000000,

00001000000000000000000000000000;

(Fibonacci.txt)(仅作为解释性说明，不参与仿真)

0 1 2 3 4 5 6

0 1 1 2 3 5 8

x[1]:F(n-1)

x[2]:F(n)

x[3]:跳转地址

x[4]:初始内容为4则表示计算F(4)，后续代表剩余迭代次数

x[5]:每迭代一次，剩余迭代次数要减一，所以找个地方(x[6])放置立即数1

0: 01000000\_00000000\_00000000\_10000100 // ldi 4, x[4]

1: 01000000\_00000000\_00000000\_00100101 // ldi 1, x[5]

2: 00100111\_00000000\_00010000\_10100100 // sub x[4], x[5], x[4]

3: 01000000\_00000000\_00000000\_00000001 // ldi 0, x[1]

4: 01000000\_00000000\_00000000\_00100010 // ldi 1, x[2]

5: 01000000\_00000000\_00000000\_11000011 // ldi 6, x[3]

6: 00100111\_00000000\_00010000\_10100100 // sub x[4], x[5], x[4]

7: 00100011\_00000000\_00000100\_01000010 // add x[1], x[2], x[2]

8: 00100111\_00000000\_00001000\_00100001 // sub x[2], x[1], x[1]

9: 10000010\_00000000\_00010000\_01100000 // jg x[4], x[3], x[0]

10: 00010000\_00000000\_00001000\_00000000 // st x[2], M[0]

11: 00001000\_00000000\_00000000\_00000000 // hlt

//预期:F(4) = 3

(ns) IF(hide) ID EX MEM WB //x[1]/x[2]/x[4]

0: ldi->4 nop nop nop nop

1: ldi->5 ldi->4 nop nop nop

2: sub 454 ldi->5 ldi->4 nop nop

3: ldi->1 sub 454 ldi->5 ldi->4 nop

4: ldi->2 ldi->1 sub 454 ldi->5 ldi->4

5: ldi->3 ldi->2 ldi->1 sub 454 ldi->5 //004

6: sub 454 ldi->3 ldi->2 ldi->1 sub 454

7: sub 454 ldi->3 nop ldi->2 ldi->1 //003

8: add 122 sub 454 ldi->3 nop ldi->2 //003

9: sub 211 add 122 sub 454 ldi->3 nop //013

10: jg sub 211 add 122 sub 454 ldi->3

11: jg sub 211 nop add 122 sub 454

12: jg sub 211 nop nop add 122 //012

13: jg sub 211 nop nop nop //012

14: st jg sub 211 nop nop

15: sub 454 st jg sub 211 nop

16: add 122 sub 454 nop jg sub 211

17: sub 211 add 122 sub 454 nop jg //112

18: jg sub 211 add 122 sub 454 nop

19: jg sub 211 nop add 122 sub 454

20: jg sub 211 nop nop add 122 //111

21: jg sub 211 nop nop nop //121

22: st jg sub 211 nop nop

23: sub 454 st jg sub 211 nop

24: add 122 sub 454 nop jg sub 211

25: sub 211 add 122 sub 454 nop jg //121

26: jg sub 211 add 122 sub 454 nop

27: jg sub 211 nop add 122 sub 454

28: jg sub 211 nop nop add 122 //120

29: jg sub 211 nop nop nop //130

30: st jg sub 211 nop nop

31: hlt st jg sub 211 nop

32: nop hlt st jg sub 211

33: nop nop hlt st jg //230

34: nop nop nop hlt st

35: nop nop nop nop hlt

(REG.v)

module REG #(

parameter width = 32,

parameter depth = 32

)(

input clk, rst, ena\_low,

input[4:0] raddr1, raddr2, waddr,

input[width - 1:0] regdata,

output[width - 1:0] result1\_reg, result2\_reg

);

reg[width - 1:0] registers[depth - 1:1];

integer i;

always@(posedge clk, posedge rst) begin

if (rst == 1)begin

for (i = 1; i < depth; i = i + 1) registers[i] <= 0;

end

else begin

registers[waddr][31:16] <= regdata[31:16];

if (ena\_low) registers[waddr][15:0] <= regdata[15:0];

end

end

wire[width - 1:0] results\_reg[depth - 1:0];

assign results\_reg[0] = 0;

genvar j;

generate

for (j = 1; j < depth; j = j + 1) begin

assign results\_reg[j] = registers[j];

end

endgenerate

assign result1\_reg = results\_reg[raddr1];

assign result2\_reg = results\_reg[raddr2];

Endmodule

(Pipeline.v)

module Pipeline #(

parameter width = 32

)(

input clk, rst, wea,

input[width - 1:0] pipe\_prev,

output reg[width - 1:0] pipe

);

always@(posedge clk, posedge rst) begin

if (rst == 1) pipe <= 0;

else if (wea == 1) pipe <= pipe\_prev;

end

Endmodule

(ALU.v)

/\*

and 1101

or 1100

nand 0000

nor 0001

xor 0010/1110

xnor 0110/1010

add 0011

sub 0111

\*/

module ALU #(

parameter width = 32

)(

input [width - 1:0] data1\_alu, data2\_alu,

input [12:0] ALUop,

output[width - 1:0] result\_alu

);

wire[width - 1:0] cout;

ALU\_single ALU\_single\_0(data1\_alu[0], data2\_alu[0], ALUop[11], ALUop[12:9], result\_alu[0], cout[0]);

generate

genvar i;

for (i = 1; i < width; i = i + 1) begin

ALU\_single ALU\_single(data1\_alu[i], data2\_alu[i], cout[i - 1], ALUop[12:9], result\_alu[i], cout[i]);

end

endgenerate

endmodule

module ALU\_single(

input a, b, cin,

input [3:0] ALUop,

output result, cout

);

wire[3:0] preresult;

wire[1:0] MUXsel;

assign {aconv, bconv, MUXsel} = ALUop;

assign a\_after = a ^ aconv;

assign b\_after = b ^ bconv;

assign preresult[0] = ~(a\_after & b\_after);

assign preresult[1] = ~(a\_after | b\_after);

assign preresult[2] = a\_after ^ b\_after;

assign preresult[3] = preresult[2] ^ cin;

assign cout = ~(preresult[0] & ~(cin & a\_after) & ~(cin & b\_after));

assign result = preresult[MUXsel];

Endmodule

(MEM)

调用Block Memory Generator

Memory Type:Single Port RAM

Width:32

Depth:65536

Write First

Use Enable Pin

无Output Register

无初始化文件

尾部填充零

Simulation source:

(test\_AKD\_T1.v)

`timescale 1ns / 1ps

module test\_AKD\_T1(

output[31:0] result\_mem,

output[15:0] detector\_pc,

output[31:0] detector\_pipe\_ID,

output[31:0] detector\_pipe\_EX,

output[31:0] detector\_pipe\_MEM,

output[31:0] detector\_pipe\_WB,

output detector\_hazard\_data,

output detector\_hazard\_structure,

output detector\_jump

);

reg clk, rst;

wire cpu\_halt;

always #0.5 clk = ~clk;

AKD\_T1 #(

.widthi(32),

.widthd(32),

.depthim(65536),

.depthmem(65536)

) AKD\_T1(

.clk(clk),

.rst(rst),

.cpu\_halt(cpu\_halt),

.result\_mem(result\_mem),

.detector\_pc(detector\_pc),

.detector\_pipe\_ID(detector\_pipe\_ID),

.detector\_pipe\_EX(detector\_pipe\_EX),

.detector\_pipe\_MEM(detector\_pipe\_MEM),

.detector\_pipe\_WB(detector\_pipe\_WB),

.detector\_hazard\_data(detector\_hazard\_data),

.detector\_hazard\_structure(detector\_hazard\_structure),

.detector\_jump(detector\_jump)

);

always@(posedge cpu\_halt) begin

#1 $finish();

end

initial begin

clk = 0;

#0.1 rst = 1;

#0.1 rst = 0;

end

endmodule