Name – Megha Y RayanaGoudra

SRN – PES2UG22EC077

Single cycle RISC-V :

1. Instruction fetch

module mux\_1 #(parameter N=32)(a,b,s,y);

input logic [N-1:0] a,b;

output logic [N-1:0]y;

input logic s;

assign y=s?a:b;

endmodule

module PC #(parameter N=32) (input logic clk,input logic rst,input logic [N-1:0] pc\_next,output logic [N-1:0] pc);

always\_ff@(posedge clk)

begin

if (rst)

pc<= 32'b0 ;

else

pc<= pc\_next ;

end

endmodule

module instr\_mem #(parameter W=32, L=1024) (addr,instr,reset);

initial

begin

$readmemh("program\_dump.hex",mem);

end

input logic reset;

input logic [W-1:0] addr;

output logic [W-1:0] instr;

logic [W-1:0] mem [0:L-1];

assign instr=reset?0:mem[addr];

endmodule

module fulladder #(parameter N=32)(a,b,sum);

input logic [N-1:0] a,b;

output logic [N-1:0] sum;

assign sum=a+b;

endmodule

module instr\_fetch #(parameter N=32) (input logic clk,input logic reset, input logic sel, input logic [N-1:0]pc\_imm, output logic [N-1:0]pc\_new,output logic [N-1:0]inst,pc);

wire [N-1:0]line1;

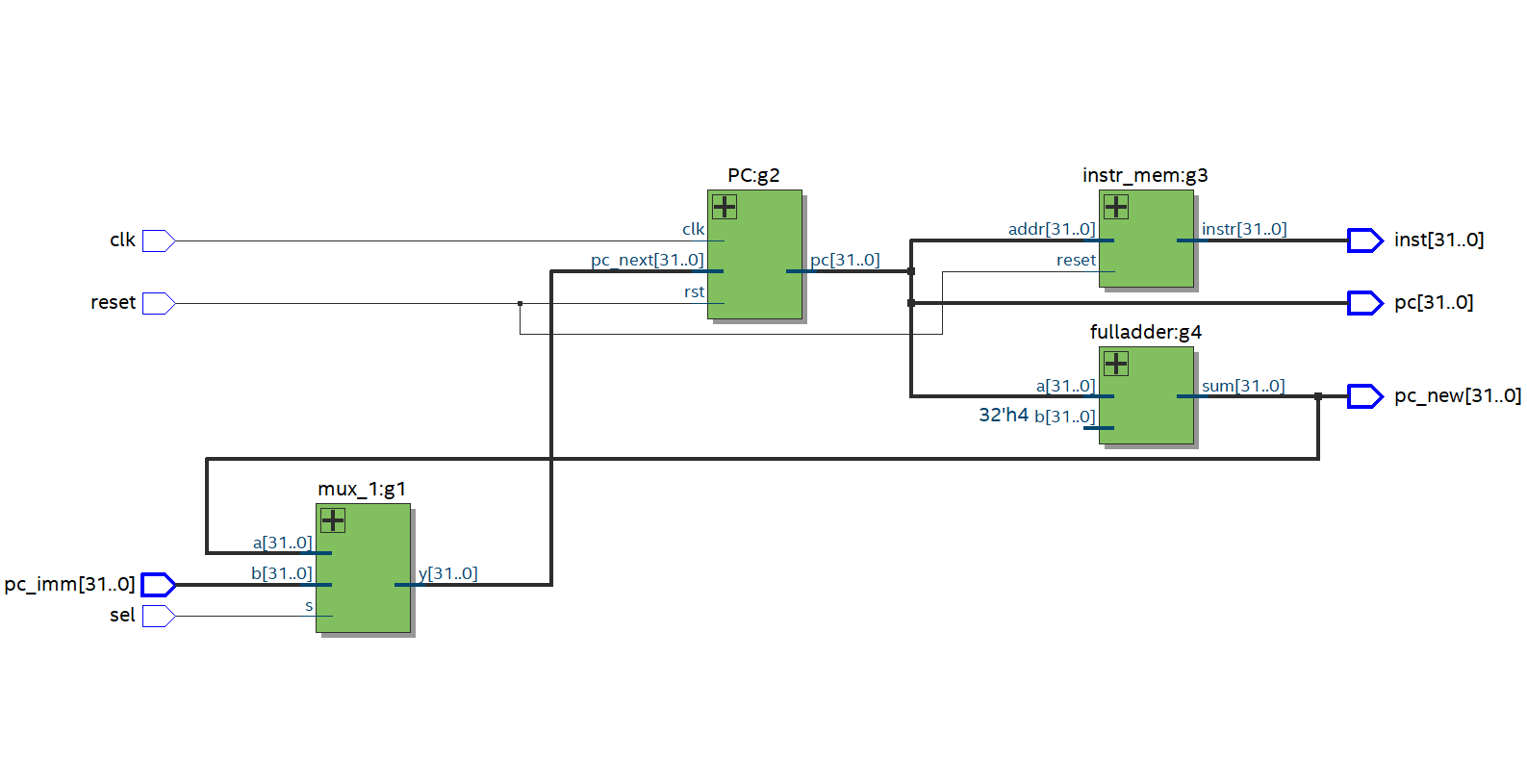
mux\_1 #(32) g1(pc\_new,pc\_imm,sel,line1);

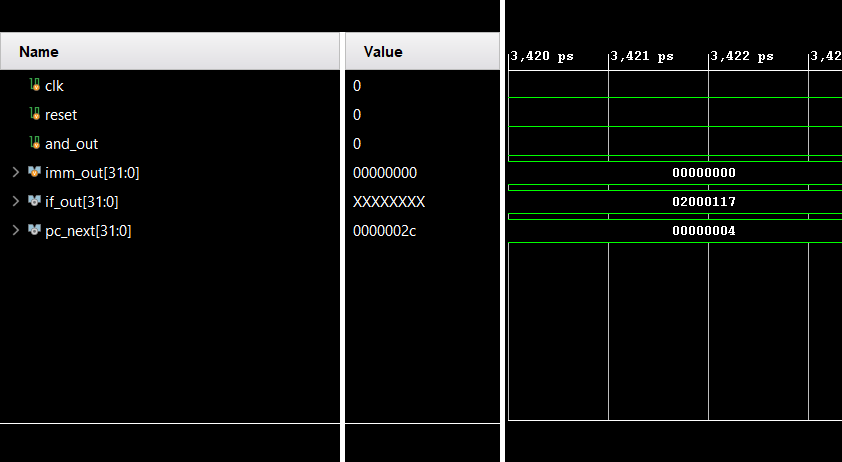
PC #(32) g2(clk,reset,line1,pc);

instr\_mem #(32) g3(pc,inst,reset);

fulladder #(32) g4(pc,32'd4,pc\_new);

endmodule





1. Instruction Decode

module decoder (input logic [31:0]inst, output logic [4:0]rd,output logic [4:0]rs1, output logic [4:0]rs2,output logic [2:0]func3, output logic [6:0]func7,output logic [11:0]imm,output logic [6:0]opcode);

always\_comb

begin

rd = 5'b0;

rs1 = 5'b0;

rs2 = 5'b0;

func3 = 3'b0;

func7 = 7'b0;

imm = 12'b0;

opcode=inst[6:0];

case(inst[6:0])

7'b0110011:

begin

rd=inst[11:7];

rs1=inst[19:15];

rs2=inst[24:20];

func3=inst[14:12];

func7=inst[31:25];

end

7'b0010011:

begin

rd=inst[11:7];

rs1=inst[19:15];

func3=inst[14:7];

imm=inst[11:0];

end

7'b1100011:

begin

rs1=inst[19:15];

rs2=inst[24:20];

func3=inst[14:12];

imm={inst[31],inst[30:25],inst[11:8],inst[7]};

end

7'b0100011:

begin

rs1=inst[19:15];

rs2=inst[24:20];

func3=inst[14:7];

imm={inst[31:25],inst[11:7]};

end

7'b0000011:

begin

rd=inst[11:7];

rs1=inst[19:15];

func3=inst[14:7];

imm=inst[11:0];

end

endcase

end

endmodule

module regfile(input logic [4:0]rs1\_addr,rs2\_addr,rd\_addr,input logic reg\_write,input logic [31:0]wr\_data, input logic reset,output logic [31:0]rs1\_data,rs2\_data);

logic [31:0]rfile[0:31];

always\_comb

begin

if(~reset)

begin

rs1\_data=rfile[rs1\_addr];

rs2\_data=rfile[rs2\_addr];

end

else

begin

rs1\_data=32'b0;

rs2\_data=32'b0;

end

end

always\_ff @(posedge reg\_write or posedge reset) begin

if (reset) begin

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

rfile[i] = 32'b0; // Clear all registers

end

end

else if (reg\_write) begin

rfile[rd\_addr] = wr\_data; // Write data to the specified register

end

end

endmodule

module sign\_exe (input logic [11:0]imm,output logic [31:0]imm\_exe);

always\_comb

begin

if(imm[11]==0)

begin

imm\_exe[31:12]=20'b0;

imm\_exe[11:0]=imm[11:0];

end

else

begin

imm\_exe[31:12]=20'b1;

imm\_exe[11:0]=imm[11:0];

end

end

endmodule

module instr\_decode (

input logic [31:0] inst,

output logic [31:0] rs1\_data,

output logic [31:0] rs2\_data,

output logic [31:0] imm32,

output logic [6:0] func7,

output logic [2:0] func3,

input logic [31:0] wr\_data,

input logic reset,

output logic [6:0]opcode

);

wire [4:0] rs1\_addr;

wire [4:0] rs2\_addr;

wire [4:0] rd\_addr;

wire [11:0] imm12;

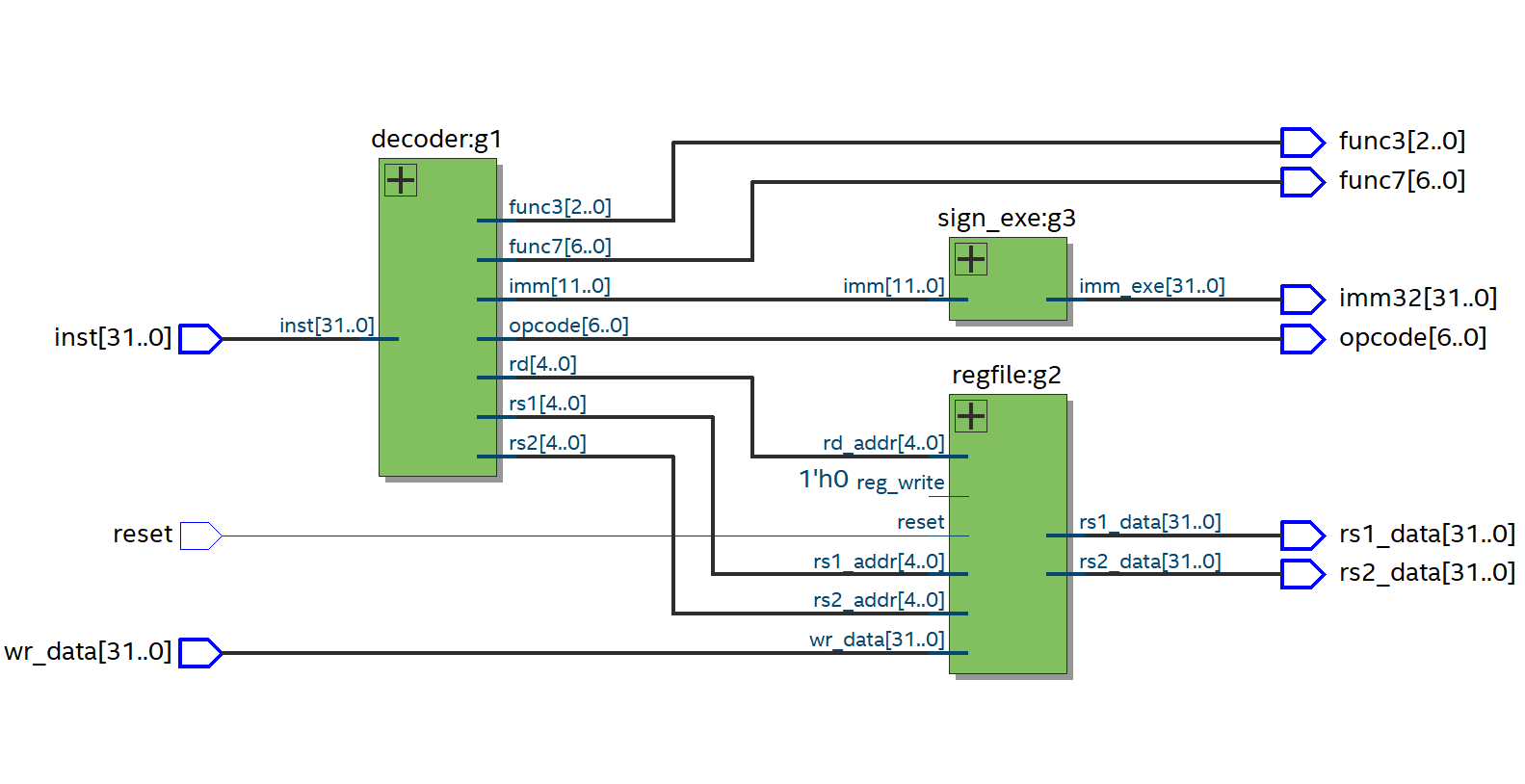
wire regwrite;

decoder g1(inst,rd\_addr,rs1\_addr,rs2\_addr,func3,func7,imm12,opcode);

regfile g2(rs1\_addr,rs2\_addr,rd\_addr,regwrite,wr\_data,reset,rs1\_data,rs2\_data);

sign\_exe g3(imm12,imm32);

endmodule



1. Instruction Execution

module alu #(parameter N=32)(input logic [N-1:0]rs1\_data,rs2\_data,input logic alu\_op, output logic [N-1:0]alu\_result,output logic zero);

always\_comb

begin

case(alu\_op)

4'b0000: alu\_result=rs1\_data&rs2\_data;

4'b0001: alu\_result=rs1\_data|rs2\_data;

4'b0010: alu\_result=rs1\_data+rs2\_data;

4'b0110: alu\_result=rs1\_data-rs2\_data;

endcase

if(alu\_result==0)

zero=1;

else

zero=0;

end

endmodule

module and\_gate(input logic a, input logic b, output logic q);

assign q=a&b;

endmodule

module instr\_exec #(parameter N=32) (input logic [N-1:0]rs1\_data,rs2\_data,imm,pc,input logic alusrc,branch,input logic alu\_op,output logic [N-1:0]alu\_result,pc\_imm,output logic and\_result);

wire [N-1:0]mux\_result,imm32\_new;

wire zero;

mux\_1 g1(rs2\_data,imm,alusrc,mux\_result);

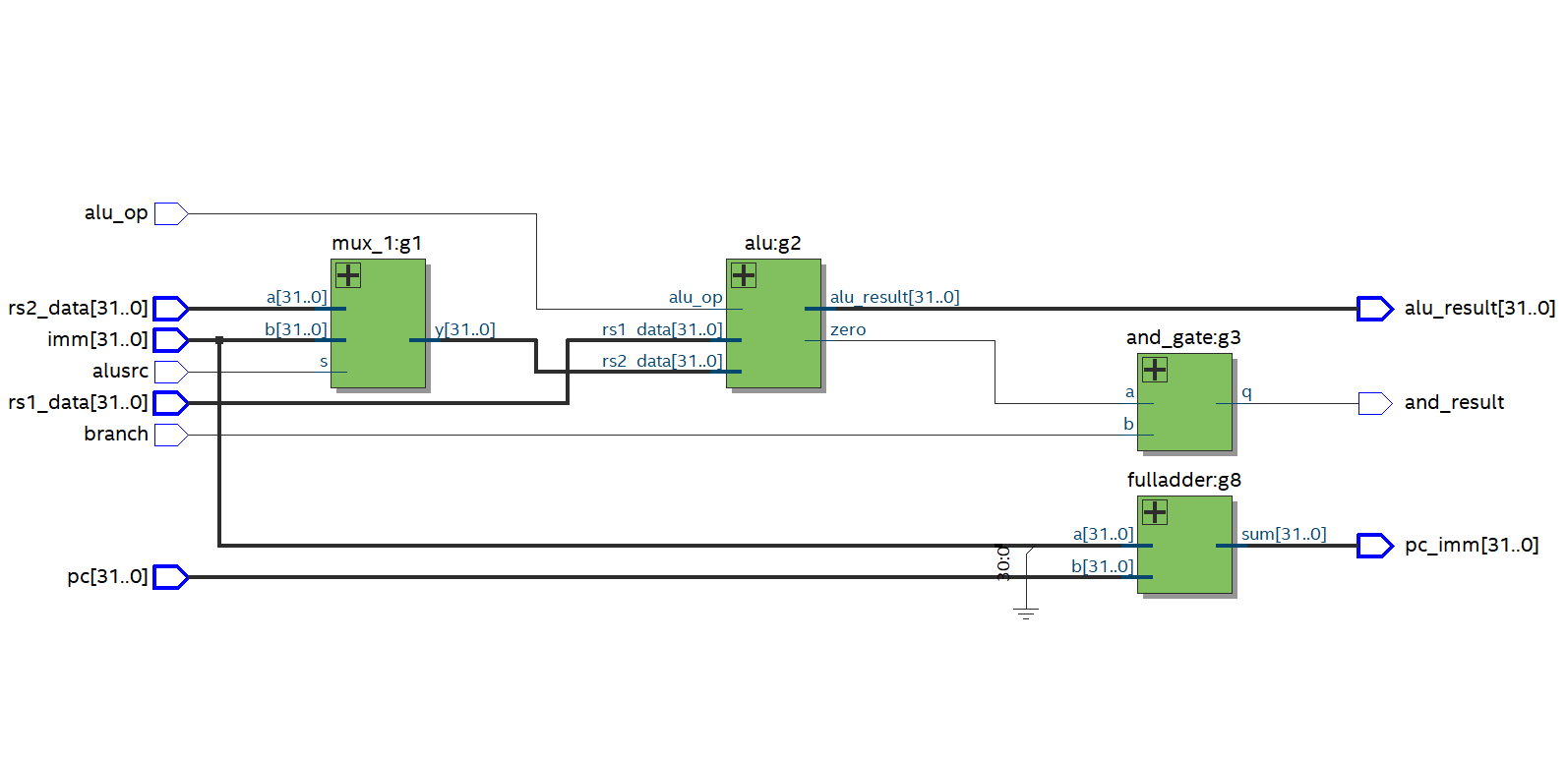
alu g2(rs1\_data,mux\_result,alu\_op,alu\_result,zero);

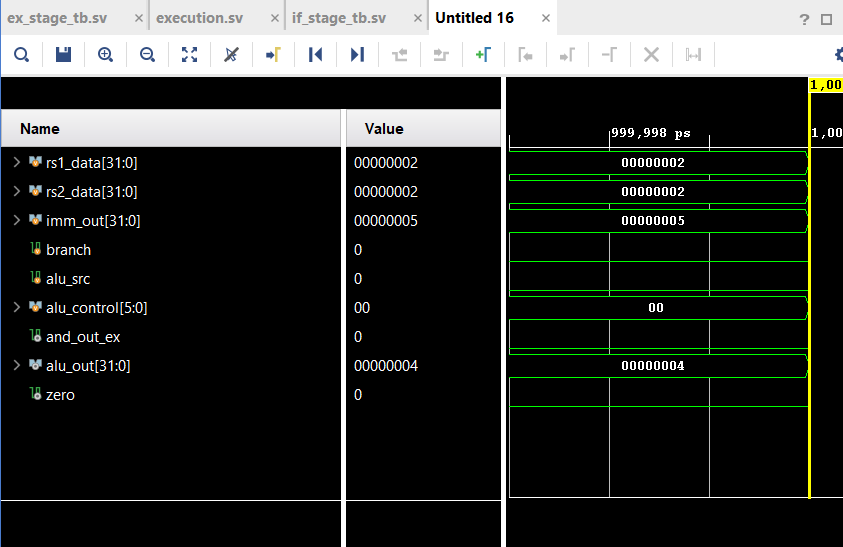
and\_gate g3(zero,branch,and\_result);

assign imm32\_new = imm<<1;

fulladder g8(imm32\_new,pc,pc\_imm);

endmodule





1. Memory Access

module memstage #(parameter N=32, M=1024) (

input logic [N-1:0] alu\_out, data\_in,

input logic mem\_read, mem\_write,

output logic [N-1:0] data\_out

);

logic [N-1:0] mem [0:M-1];

always\_comb

begin

if( mem\_read==1 && mem\_write==0)

data\_out= mem[alu\_out];

else if (mem\_read==0 && mem\_write==1) begin

mem[alu\_out]= data\_in;

data\_out= 32'b0;

end

else

data\_out= 32'b0;

end

endmodule

module data\_mem #(parameter N=32) (

input logic [N-1:0] alu\_out, data\_in,

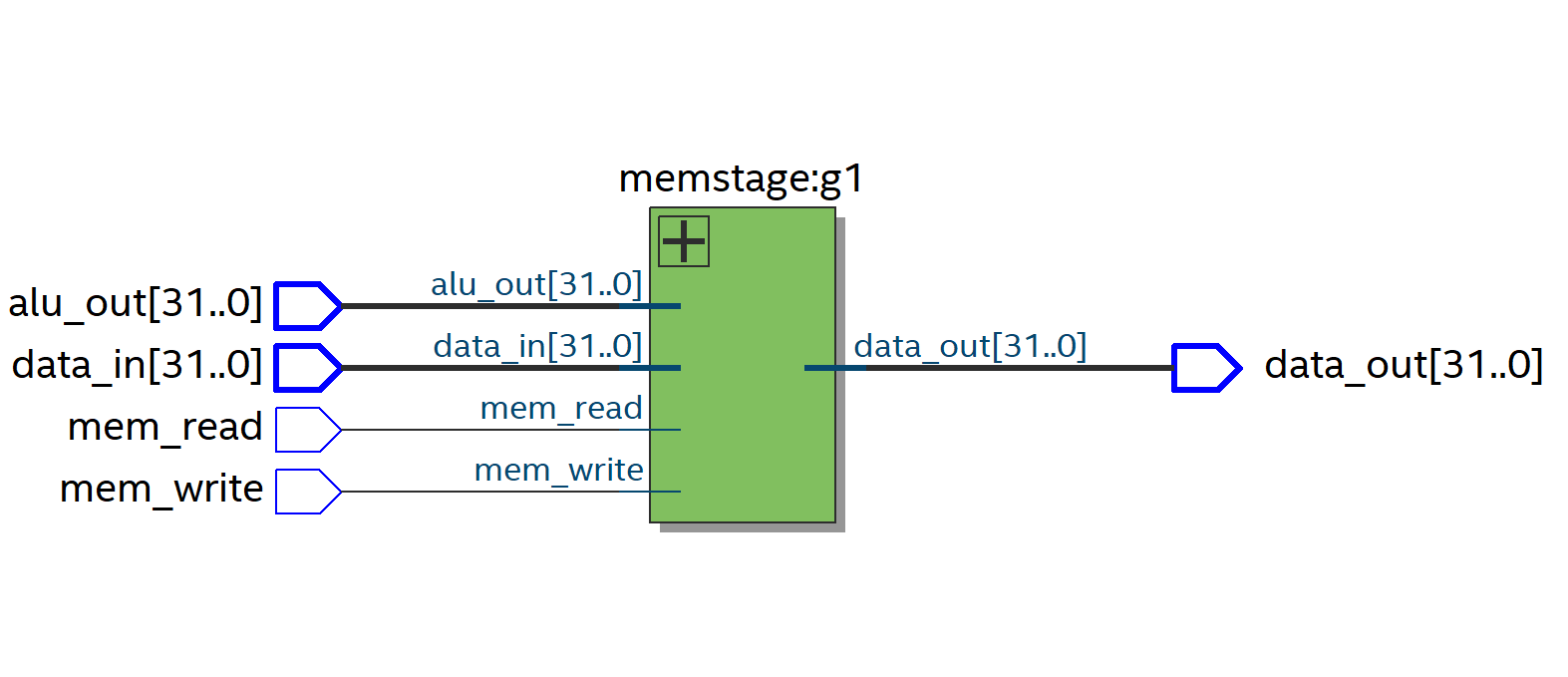
input logic mem\_read, mem\_write,

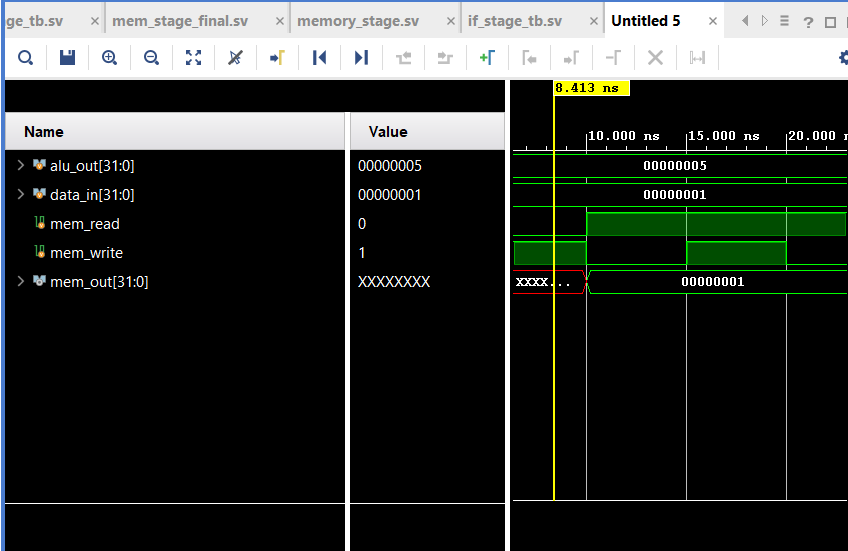
output logic [N-1:0] data\_out

);

memstage g1(alu\_out, data\_in,mem\_read, mem\_write,data\_out);

endmodule





1. Write Back

module write\_back1 #(parameter N=32)(

input logic [N-1:0] alu\_result,

input logic [N-1:0] data\_out,

input logic mem\_reg,

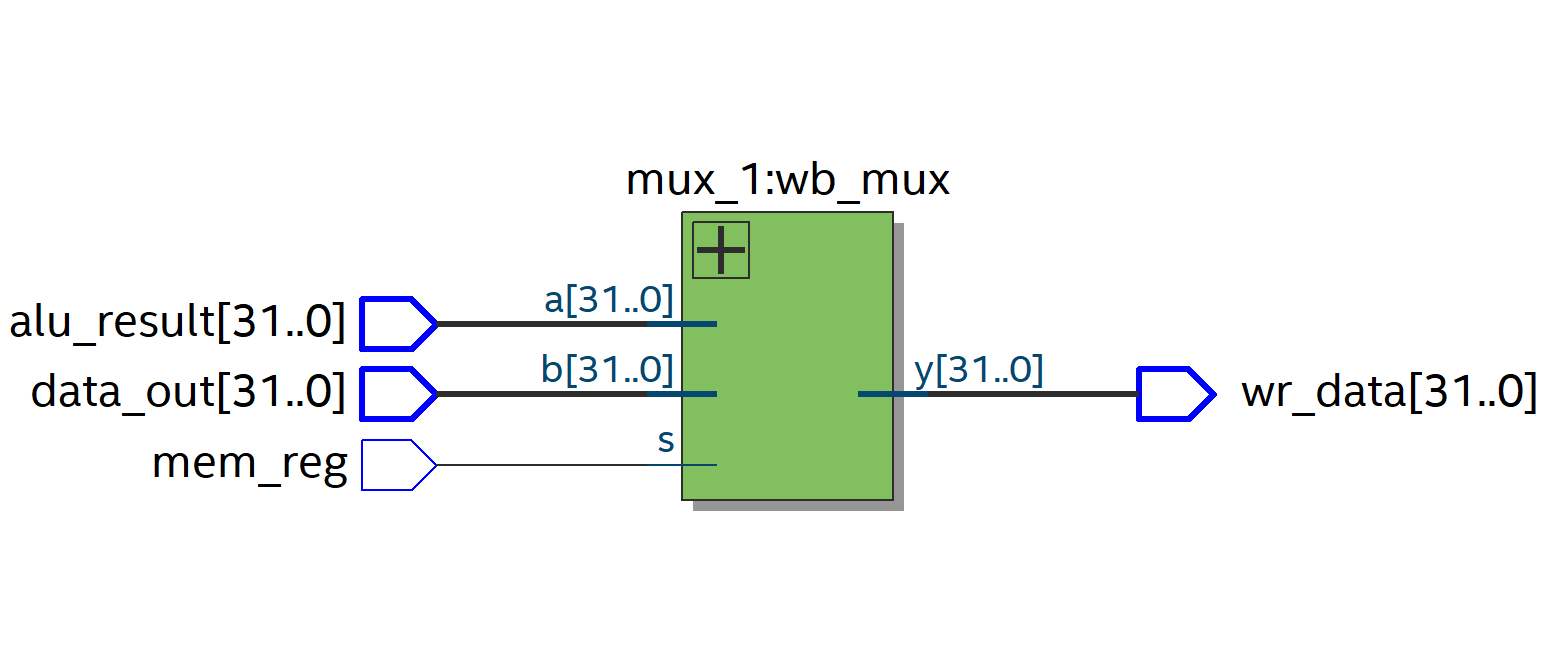
output logic [N-1:0] wr\_data

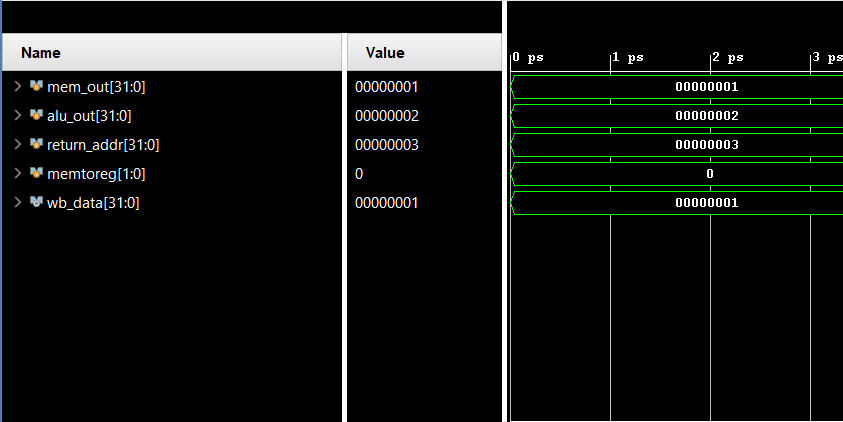
);

(.alu\_result(alu\_result),.data\_in(data\_in),.mem\_read(mem\_read),.mem\_write(mem\_write),.data\_out(data\_out));

mux\_1 wb\_mux (alu\_result,data\_out,mem\_reg,wr\_data);

endmodule





1. Main Control Unit

module Main\_control(input logic [6:0] opcode,

output logic regWrite, output logic ALUSrc,

output logic memRead,

output logic memWrite,

output logic branch,

output logic [1:0] ALUOp, output logic mem\_reg

);

always\_comb begin

regWrite = 0;

mem\_reg=0;

ALUSrc = 0;

memRead = 0;

memWrite = 0;

branch = 0;

ALUOp = 2'b00;

case (opcode)

7'b0110011: begin // R-type instructions

regWrite = 1;

ALUOp = 2'b10;

end

7'b0010011: begin // I-type instructions

regWrite = 1;

ALUSrc = 1;

ALUOp = 2'b00;

end

7'b0000011: begin // Load instructions

regWrite = 1;

memRead = 1;

ALUSrc = 1;

mem\_reg=1;

end

7'b0100011: begin // Store instructions

memWrite = 1;

ALUSrc = 1;

end

7'b1100011: begin // Branch instructions

branch = 1;

ALUOp = 2'b01;

end

default: begin

regWrite=0;

ALUSrc=0;

memRead=0;

memWrite=0;

branch=0;

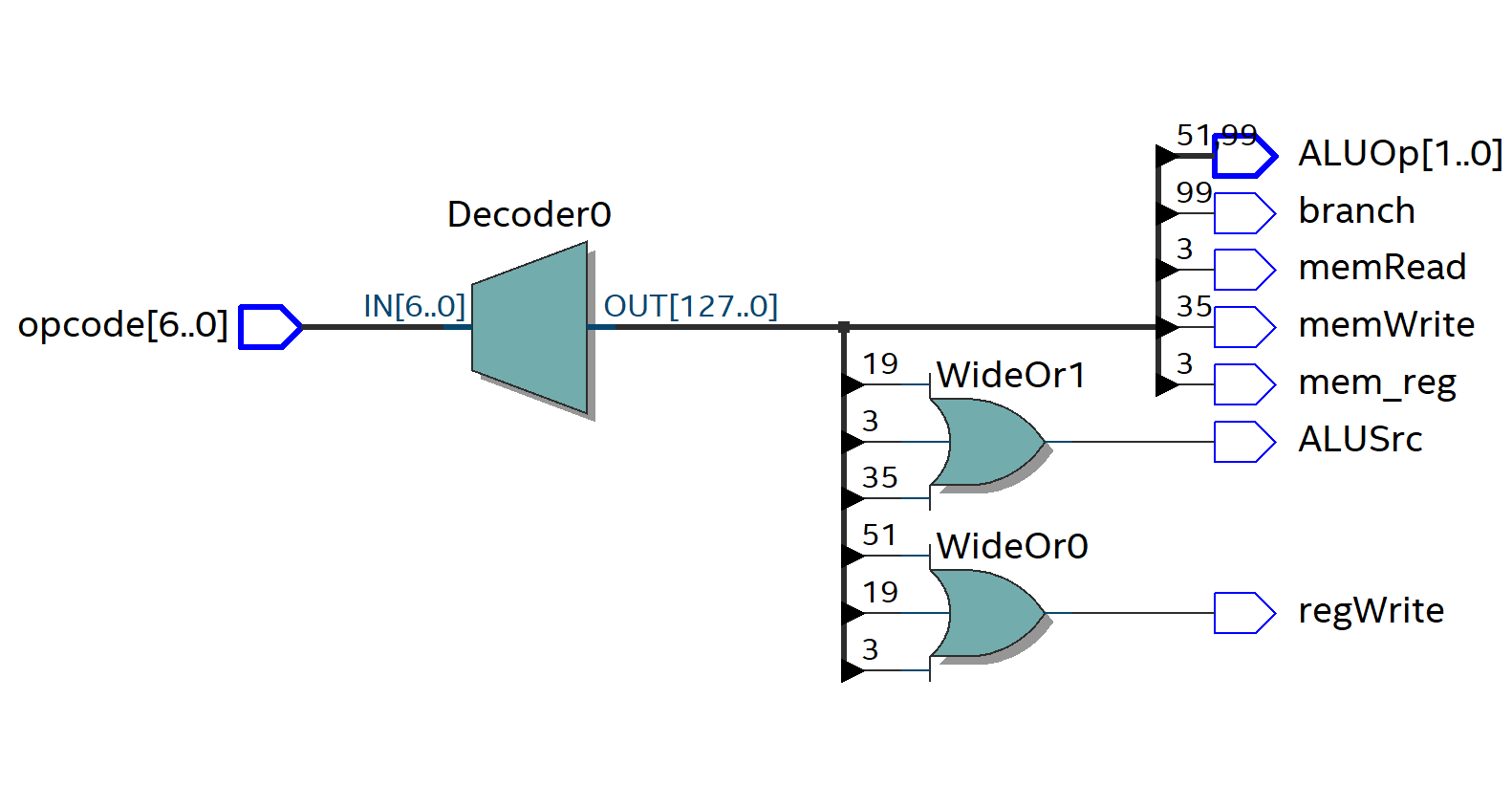
ALUOp=0;

end

endcase

end

endmodule



1. ALU Control Unit

module Alu\_control( input logic [1:0] ALUOp,

input logic [2:0] funct3,

input logic funct7\_5,

output logic [3:0] ALUControl

);

always\_comb begin

ALUControl = 4'b1111;

if (ALUOp == 2'b00) begin

ALUControl = 4'b0010; // ADD

end

else if (ALUOp == 2'b01) begin

ALUControl = 4'b0110; // SUB

end

else if (ALUOp == 2'b10) begin

case (funct3)

3'b000: ALUControl = funct7\_5 ? 4'b0110 : 4'b0010; // SUB if funct7\_5=1, ADD if funct7\_5=0

3'b111: ALUControl = 4'b0000; // AND

3'b110: ALUControl = 4'b0001; // OR

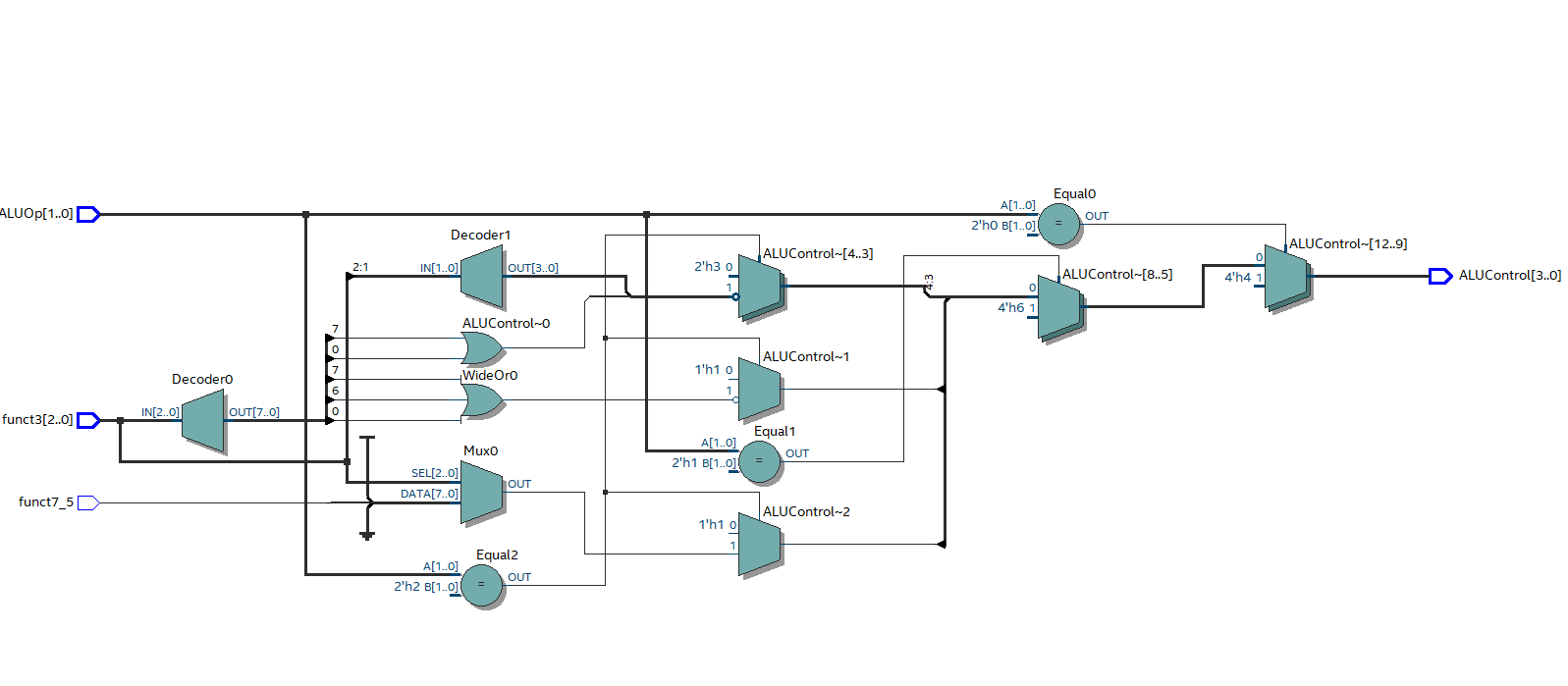
default: ALUControl = 4'b1111; // Undefined

endcase

end

end

endmodule



1. Top Module

module single\_cycle #(parameter N=32) (input logic clk,reset);

logic [N-1:0]inst,pc\_new,data\_in,data\_out, alu\_out,wd;

logic [2:0] func3;

logic [6:0] func7,opcode;

logic [31:0]imm32\_new,adder\_out;

logic [31:0]imm32;

logic [31:0]wr\_data;

logic [3:0]alu\_op;

logic branch,alu\_src,mem\_read, mem\_write,mem\_reg,sel;

logic [31:0]rs1, rs2;

logic [N-1:0] pc\_imm;

logic [N-1:0] pc;

logic reg\_write;

logic [1:0] alu\_control;

instr\_fetch g1 (clk,reset,and\_result,pc\_imm,pc\_new,inst,pc);

instr\_decode g2(inst,rs1,rs2,imm32,func7,func3,wr\_data,reset,opcode);

instr\_exec g3(rs1,rs2,imm32,pc,alu\_src,branch,alu\_control,alu\_out,pc\_imm,and\_result);

data\_mem g4(alu\_out, rs2,mem\_read, mem\_write,data\_out);

write\_back1 g5(alu\_out,data\_out,mem\_reg,wr\_data);

Main\_control g9(opcode,reg\_write,alu\_src,mem\_read,mem\_write,branch,alu\_op,mem\_reg);

Alu\_control g10(alu\_op,func3,func7,alu\_control);

endmodule

