**LAB 7 REPORT**

|  |  |
| --- | --- |
| Class: CE213.O12.MTCL.1  Student: Tran Nguyen Anh Khoa  ID: 21521004 | comment |

# THEORY

**1.1 Overview**

|  |
| --- |
|  |

# EXCIXE

**2.2 Verilog code**

**2.2.1 MIPS overview**

|  |
| --- |
| module MIPS(  //PORT    input clk, rst,  output [31:0] out  );  // WIRE  wire [4:0]WR;  wire [1:0] ALUOp;  wire [3:0] ALUsel;  wire [31:0] signEx, ALU\_result, ALU\_b, readdata, PC\_current, PC\_next, Inst, WD, Add1\_out, Add2\_out,  mux1, Jump\_address, ShiftLeft2, RD1, RD2;  wire RegWrite, RegDst, MemRead, MemWrite, MemToReg, ALUSrc, Jump, Branch, and1, isZero;  //LOGIC    ProgramCounter PC(.PC\_current(PC\_current[31:0]),  .PC\_next(PC\_next[31:0]),  .clk(clk),  .rst(rst)  );  InstructionMemory IM( .rst(rst),  .ReadAddress(PC\_current),  .ReadData(Inst)  );  Shift\_left2 sl2(.PCp4(Add1\_out[31:28]),  .instruc(Inst[25:0]),  .JumpA(Jump\_address)  );  Adder Add\_PC(.A(PC\_current),  .B(32'd4),  .Sum(Add1\_out)  );  Adder Add2( .A(Add1\_out),  .B(signEx << 2),  .Sum(Add2\_out)  );  Control CT( .opcode(Inst[31:26]),  .RegDst(RegDst),  .Jump(Jump),  .Branch(Branch),  .MemRead(MemRead),  .MemtoReg(MemToReg),  .MemWrite(MemWrite),  .ALUSrc(ALUSrc),  .RegWrite(RegWrite),  .ALUOp(ALUOp[1:0])  );  and a1(and1, isZero, Branch);  assign mux1 = and1 ? Add2\_out : Add1\_out;  assign PC\_next = Jump ? Jump\_address : mux1;    assign WR = RegDst ? Inst[15:11] : Inst[20:16];  RegisterFile RF( .RR1(Inst[25:21]),  .RR2(Inst[20:16]),  .WR(WR[4:0]),  .WD(out),  .RD1(RD1[31:0]),  .RD2(RD2[31:0]),  .RegWrite(RegWrite),  .clk(clk),  .rst(rst)  );  SignExtend SE( .i\_extend(Inst[15:0]),  .o\_extended(signEx[31:0])  );  assign ALU\_b = ALUSrc ? signEx : RD2;    ALUcontrol ALUct( .ALUOp(ALUOp),  .funct(Inst[5:0]),  .ALU\_sel(ALUsel)  );  ALU alu( .i\_A(RD1[31:0]),  .i\_B(ALU\_b[31:0]),  .i\_Operation(ALUsel[3:0]),  .o\_Result(ALU\_result[31:0]),  .o\_isZero(isZero)  );    DataMemory DM( .Address(ALU\_result[31:0]),  .WriteData(RD2[31:0]),  .MemRead(MemRead),  .MemWrite(MemWrite),  .ReadData(readdata[31:0]),  .clk(clk)  );  assign out = MemToReg ? readdata : ALU\_result;    endmodule |

**2.2.2 Program Counter**

|  |
| --- |
| module ProgramCounter(  input clk, rst,  input [31:0] PC\_next,  output reg [31:0] PC\_current  );    always @(posedge clk) begin  if(rst==1'b0) begin  PC\_current <= 32'd0;  end  else begin  PC\_current <= PC\_next;  end  end  initial begin  PC\_current=32'd0;  end    endmodule |

**2.2.3 Instruction Memory**

|  |
| --- |
| module InstructionMemory(rst, ReadAddress, ReadData);  input rst;  input [31:0] ReadAddress;  output [31:0] ReadData;  // creation of memory  reg [31:0] mem [1023:0];  assign ReadData = (~rst) ? {32{1'b0}} : mem[ReadAddress];  /\*  assgin $t2=18; $s1=9;    lw $t1, 0($s1);  add $t3, $t1, $t2;  sub $t4, $t2, $t1  sw $s2, 0($t3);  beq $t1, $t4, L1;  j Exit;  L1:or $t5, $t1, $t2;  and $t6, $t1, $t2;  slt $t7, $t1, $t2;  Exit:  \*/  initial begin  mem[32'd04] = 32'b10001110001010010000000000000000;  mem[32'd08] = 32'b00000001001010100101100000100000;  mem[32'd12] = 32'b00000001010010010110000000100010;  mem[32'd16] = 32'b10101110010010110000000000000000;  mem[32'd20] = 32'b00010001001011000000000000000001;  mem[32'd24] = 32'b00001000000000000000000000000011;  mem[32'd28] = 32'b00000001001010100110100000100101;  mem[32'd32] = 32'b00000001001010100111000000100100;  mem[32'd36] = 32'b00000001001010100111100000101010;  end  endmodule |

**2.2.4 Shift left 2**

|  |
| --- |
| module Shift\_left2(  input [3:0] PCp4,  input [25:0] instruc,  output reg [31:0] JumpA  );  always @(\*) begin  JumpA[25:0] <= instruc << 2;  JumpA[27:26] <= {2{instruc[25]}};  JumpA[31:28] <= PCp4;  end  endmodule |

**2.2.5 Adder**

|  |
| --- |
| module Adder(  input [31:0] A, B,  output [31:0] Sum  );  assign Sum = A+B;  endmodule |

**2.2.6 Control**

|  |
| --- |
| module Control(  //input  input [5:0] opcode,  //output  output reg RegDst, Jump, Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite,  output reg [1:0] ALUOp  );  always @(\*) begin  case(opcode)  6'b000000: begin //R-Type (and/sub/and/or/slt)  RegDst = 1'b1;  Jump = 1'b0;  Branch = 1'b0;  MemRead = 1'b0;  MemtoReg = 1'b0;  ALUOp = 2'b10;  MemWrite = 1'b0;  ALUSrc = 1'b0;  RegWrite = 1'b1;  end  // I-Type  6'b100011: begin //lw  RegDst = 1'b0;  Jump = 1'b0;  Branch = 1'b0;  MemRead = 1'b1;  MemtoReg = 1'b1;  ALUOp = 2'b00;  MemWrite = 1'b0;  ALUSrc = 1'b1;  RegWrite = 1'b1;  end  6'b101011: begin // sw  RegDst = 1'bx;  Jump = 1'b0;  Branch = 1'b0;  MemRead = 1'b0;  MemtoReg = 1'bx;  ALUOp = 2'b00;  MemWrite = 1'b1;  ALUSrc = 1'b1;  RegWrite = 1'b0;  end  6'b000100: begin // beq  RegDst = 1'bx;  Jump = 1'b0;  Branch = 1'b1;  MemRead = 1'b0;  MemtoReg = 1'bx;  ALUOp = 2'b01;  MemWrite = 1'b0;  ALUSrc = 1'b0;  RegWrite = 1'b0;  end  6'b000010: begin // jump  RegDst = 1'b0;  Jump = 1'b1;  Branch = 1'b0;  MemRead = 1'b0;  MemtoReg = 1'b0;  ALUOp = 2'bxx;  MemWrite = 1'b0;  ALUSrc = 1'b0;  RegWrite = 1'b0;  end  default: begin  RegDst = 1'b0;  Jump = 1'b0;  Branch = 1'b0;  MemRead = 1'b0;  MemtoReg = 1'b0;  ALUOp = 2'b10;  MemWrite = 1'b0;  ALUSrc = 1'b0;  RegWrite = 1'b0;  end  endcase  end  endmodule |

**2.2.7 Register File**

|  |
| --- |
| module RegisterFile(  // port declac  input [4:0] RR1, RR2, WR,  input RegWrite, clk, rst,  input [31:0] WD,  output [31:0] RD1, RD2  );  reg [31:0] RegMem [31:0];  // logical  assign RD1 = (!rst) ? 32'd0 : RegMem[RR1];  assign RD2 = (!rst) ? 32'd0 : RegMem[RR2];    always @(posedge clk) begin  if (RegWrite)  begin  RegMem[WR] <= WD;  end  end  //$t2=18;  initial begin  RegMem[32'd10]=32'd18;  RegMem[32'd1]= 32'd1;  RegMem[32'd2]= 32'd2;  RegMem[32'd3]= 32'd3;  RegMem[32'd4]= 32'd4;  RegMem[32'd5]= 32'd5;  RegMem[32'd6]= 32'd6;  RegMem[32'd7]= 32'd7;  RegMem[32'd8]= 32'd8;  RegMem[32'd9]= 32'd9;  RegMem[32'd11]= 32'd11;  RegMem[32'd12]= 32'd12;  RegMem[32'd13]= 32'd13;  RegMem[32'd14]= 32'd14;  RegMem[32'd15]= 32'd15;  RegMem[32'd16]= 32'd16;  RegMem[32'd17]= 32'd17;  RegMem[32'd18]= 32'd18;  RegMem[32'd19]= 32'd19;  RegMem[32'd20]= 32'd20;  RegMem[32'd21]= 32'd21;  RegMem[32'd22]= 32'd22;  RegMem[32'd23]= 32'd23;  RegMem[32'd24]= 32'd24;  RegMem[32'd25]= 32'd25;  end  endmodule |

**2.2.8 Sign Extend**

|  |
| --- |
| module SignExtend(i\_extend, o\_extended);  //PORT  input [15:0] i\_extend;  output [31:0] o\_extended;  reg [31:0] o\_extended;  wire [15:0] i\_extend;  //LOGIC  always @(\*) begin  o\_extended[31:16] = {16{i\_extend[15]}};  o\_extended[15:0] = i\_extend[15:0];  end  endmodule |

**2.2.9 ALU**

|  |
| --- |
| module ALU(  input wire [31:0] i\_A, i\_B,  input wire [3:0] i\_Operation,  output reg o\_isZero,  output reg [31:0] o\_Result  );  always@(\*) begin  case(i\_Operation)  4'b0000: o\_Result <= i\_A & i\_B;  4'b0001: o\_Result <= i\_A | i\_B;  4'b0010: o\_Result <= i\_A + i\_B;  4'b0110: o\_Result <= i\_A - i\_B;  4'b0111: o\_Result <= (i\_A > i\_B);  default: o\_Result <= 32'dx;  endcase    o\_isZero <= !(|(i\_A - i\_B));  end  endmodule |

**2.2.10 ALU control**

|  |
| --- |
| module ALUcontrol(  input [1:0] ALUOp,  input [5:0] funct,  output reg [3:0] ALU\_sel  );  always @(\*) begin  case(ALUOp)  2'b10: begin //R-Type  case(funct)  6'b100000: ALU\_sel = 4'b0010; //add  6'b100010: ALU\_sel = 4'b0110; //sub  6'b100100: ALU\_sel = 4'b0000; //and  6'b100101: ALU\_sel = 4'b0001; //or  6'b101010: ALU\_sel = 4'b0111; //slt  default ALU\_sel = 4'b0000;  endcase  end  // I-Type  2'b00: ALU\_sel = 4'b0010; //lw/sw  2'b01: ALU\_sel = 4'b0110; //beq  default: ALU\_sel = 4'b0000;  endcase  end  endmodule |

**2.2.11 Data Memory**

|  |
| --- |
| module DataMemory(clk, Address, WriteData, MemWrite, ReadData, MemRead);  input [31:0] Address, WriteData;  input MemWrite, MemRead, clk;  output reg [31:0] ReadData;  reg [31:0] sram [0:31];    always @(posedge clk) begin  if(MemWrite && !MemRead)  sram[Address] <= WriteData;  end  always @(\*) begin  ReadData <= (MemRead) ? sram[Address] : 32'dz;  end    //$s1=9;  initial begin  sram[31'd17]=32'd9;  end  endmodule |

**2.3 testbench**

|  |
| --- |
| `timescale 1ns/100ps  module IM\_tb;  reg rst=1'b0;  reg [31:0] RA;  wire [31:0] RD;  InstructionMemory IM(  .rst(rst),  .ReadAddress(RA),  .ReadData(RD)  );  initial begin    #5 rst=1;  #20 RA = 32'h00000000;  #20 RA = 32'h00000004;  #20 RA = 32'h00000008;  #20 RA = 32'h0000000C;  #20 RA = 32'h00000010;  #20 RA = 32'h00000014;  #20 RA = 32'h00000018;  #20 RA = 32'h0000001C;  #20 RA = 32'h00000020;  #20 RA = 32'h00000024;    #20 $finish;  end    endmodule |

**2.4 Valuation**

**I will run code**

$s1 = 9;

$t2 = 18;

|  |  |  |
| --- | --- | --- |
| address | assembly | Machine code |
| 04d | lw $t1, 0($s1); | 10001110001010010000000000000000 |
| 08d | add $t3, $t1, $t2; | 00000001001010100101100000100000 |
| 12d | Loop: sub $t4, $t2, $t1 | 00000001010010010110000000100010 |
| 16d | sw $s2, 0($t3); | 10101110010010110000000000000000 |
| 20d | beq $t1, $t4, L1; | 00010001001011000000000000000001 |
| 24d | j loop; | 00001000000000000000000000000011 |
| 28d | L1:or $t5, $t1, $t2; | 00000001001010100110100000100101 |
| 32d | and $t6, $t1, $t2; | 00000001001010100111000000100100 |
| 36d | slt $t7, $t1, $t2; | 00000001001010100111100000101010 |

|  |
| --- |
| lw $t1, 0($s1); |
|  |

|  |
| --- |
| add $t3, $t1, $t2  sub $t4, $t2, $t1 |
|  |

|  |
| --- |
| sw $s2, 0($t3);  beq $t1, $t4, L1;  comment: t1 – t4 = 0, next address is 28 |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | or $t5, $t1, $t2; | | and $t6, $t1, $t2; | | slt $t7, $t1, $t2; | |
|  |

**Jum instruction**

T change $s1 = 13;

Beq instruction will be wrong, address from 20 to 12

|  |
| --- |
|  |