計算機組織

實驗三

結果報告

第十二組

組員:

|  |  |
| --- | --- |
| 學號 | 姓名 |
| E24099025 | 李宇洋 |
| E24099033 | 蔡晏齊 |
| E24099059 | 陳旭祺 |
| F74044046 | 蕭佑永 |

日期:2020/10/13

一、實驗內容

(1)實作一

1.題目:完成新的 ALU

2.實現方式(Verilog):

module ALU (a, b, op, result, overflow);

input [7:0] a;

input [7:0] b;

input [2:0] op;

output [7:0] result;

output overflow;

reg [8:0] Result;

//==============================================================

/\* Ins. op code

ADD 000

SUB 001

AND 100

OR 101

XOR 110

NOR 111 \*/

//==============================================================

assign result = Result[7:0];

assign overflow = (((op[2:0] == 3'b000)||(op[2:0] == 3'b001)) && (Result[8] == 1'b1))?1'b1:1'b0;

always@(a or b or op)

begin

case(op)

3'b000: Result = a+b; // ADD

3'b001: Result = a-b; // SUB

3'b100: Result = a&b; // AND

3'b101: Result = a|b; // OR

3'b110: Result = a^b; // XOR

3'b111: Result = ~(a|b); // NOR

endcase

end

endmodule

一張含有 文字 的圖片

自動產生的描述

3.結果分析(modelsim)

一張含有 監視器, 電腦, 微波爐 的圖片

自動產生的描述

(2)實作二

1.題目：完成這顆經過簡化的RISC-V CPU

2.實現方式(Verilog)

module kronos\_alu( op1,

op2,

aluop,

result

);

input [31:0] op1;

input [31:0] op2;

input [3:0] aluop;

output reg [31:0] result;

wire cin;

wire rev;

wire [31:0] r\_adder;

wire [31:0] r\_and;

wire [31:0] r\_or;

wire [31:0] r\_xor;

wire [31:0] r\_shift;

wire [31:0] adder\_A;

wire [31:0] adder\_B;

wire cout;

wire r\_lt;

wire r\_ltu;

wire r\_comp;

wire [31:0] data;

wire [4:0] shamt;

wire shift\_in;

wire [31:0] p0;

wire [31:0] p1;

wire [31:0] p2;

wire [31:0] p3;

wire [31:0] p4;

// ============================================================

// Operation Decode

assign cin = aluop[3] || aluop[1]; // SUB & Compare

assign rev = ~aluop[2];

// ============================================================

// ADDER

// ============================================================

// if the operation is SUB, invert op2 (adder\_b) before add operation

assign adder\_A = op1;

assign adder\_B = ((aluop[3:0] == 4'b1000) || (aluop[3:0] == 4'b0010))?((~op2)):op2;//當aluop=1000(SUB)跟aluop=0010(SLT)的時候將op2做inverse，之所以會需要把SLT也加入條件是因為這個指令會先用到SUB來幫助判斷大小

// Add Operation

assign {cout, r\_adder} = {1'b0, adder\_A} + {1'b0, adder\_B} + cin;

// ============================================================

// LOGIC

// ============================================================

assign r\_and = op1 & op2;

assign r\_or = op1 | op2;

assign r\_xor = op1 ^ op2;

// ============================================================

// COMPARATOR (SLT & SLTU)

//-------------------------------------------------------------

// if (aluop[0]==1) ==> Unsigned

// op1[31] op2[31]

// 0 0 --> = 2'complement

// 0 1 --> op1 > op2 0

// 1 0 --> op1 < op2 1

// 1 1 --> = 2'complement

// ============================================================

// Signed Less Than (SLT)

assign r\_lt = (op1[31] > op2[31])?1'b1:

(op1[31] < op2[31])?1'b0:

r\_adder[31];

// Unsigned Less Than (SLTU) : check the carry out on op1-op2

assign r\_ltu = ~cout;

// Select output

assign r\_comp = (aluop[0] == 1) ? r\_ltu : r\_lt;

// ============================================================

// BARREL SHIFTER

// ============================================================

// Reverse data to the shifter for SHL operations

assign data = rev ? {<<{op1}} : op1;

assign shift\_in = cin & op1[31];

assign shamt = op2[4:0];

// The barrel shifter is formed by a 5-level fixed RIGHT-shifter

// that pipes in the value of the last stage

assign p0 = shamt[0] ? { shift\_in , data[31:1]} : data;

assign p1 = shamt[1] ? {{ 2{shift\_in}}, p0[31:2]} : p0;

assign p2 = shamt[2] ? {{ 4{shift\_in}}, p1[31:4]} : p1;

assign p3 = shamt[3] ? {{ 8{shift\_in}}, p2[31:8]} : p2;

assign p4 = shamt[4] ? {{16{shift\_in}}, p3[31:16]} : p3;

// Reverse last to get SHL result

assign r\_shift = rev ? {<<{p4}} : p4;

// ============================================================

// Result Mux

always@(\*) begin

case(aluop)

4'b0010, // SLT

4'b0011 : result = {31'b0, r\_comp}; // SLTU

4'b0100 : result = r\_xor; // XOR

4'b0110 : result = r\_or; // OR

4'b0111 : result = r\_and; // AND

4'b0001, // SLL

4'b0101, // SRL

4'b1101 : result = r\_shift; // SRA

default : result = r\_adder; // ADD, SUB

endcase

end

endmodule

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

3.結果分析(modelsim)

一張含有 監視器, 電腦, 坐, 螢幕 的圖片

自動產生的描述

(3)挑戰題

1.題目:

2.實現方式(Verilog):

module DECODE (op1, op2, aluop, ins);

input [31:0] ins;

output [31:0] op1;

output [31:0] op2;

output [3:0] aluop;

wire [6:0] opcode, funct7;

wire [4:0] rs1, rs2, rd;

wire [2:0] funct3;

reg [31:0] REG [0:32];

integer i;

// initial REG data

initial begin

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

REG[i] = i\*2;

end

end

assign opcode = ins[6:0];

assign rs1 = ins[19:15];

assign rs2 = (opcode[6:0] == 7'b0110011)?ins[24:20]:5'd0;

assign rd = ins[11:7];

assign funct3 = ins[14:12];

assign funct7 = (opcode[6:0] == 7'b0110011)?ins[31:25]:7'd0;

assign op1 = REG[rs1];

assign op2 = (opcode[6:0] == 7'b0110011)?REG[rs2]:ins[31:20];

//=============================================

// Ins funct7 funct3 ALUOP

// +-----------+----------+----------+--------+

// ADD 0000000 000 0000

// ADDI - 000 0000

// SUB 0100000 000 1000

// SLT 0000000 010 0010

// SLTI - 010 0010

// SLTU 0000000 011 0011

// SLTIU - 011 0011

// XOR 0000000 100 0100

// XORI - 100 0100

// OR 0000000 110 0110

// ORI - 110 0110

// AND 0000000 111 0111

// ANDI - 111 0111

// SLL 0000000 001 0001

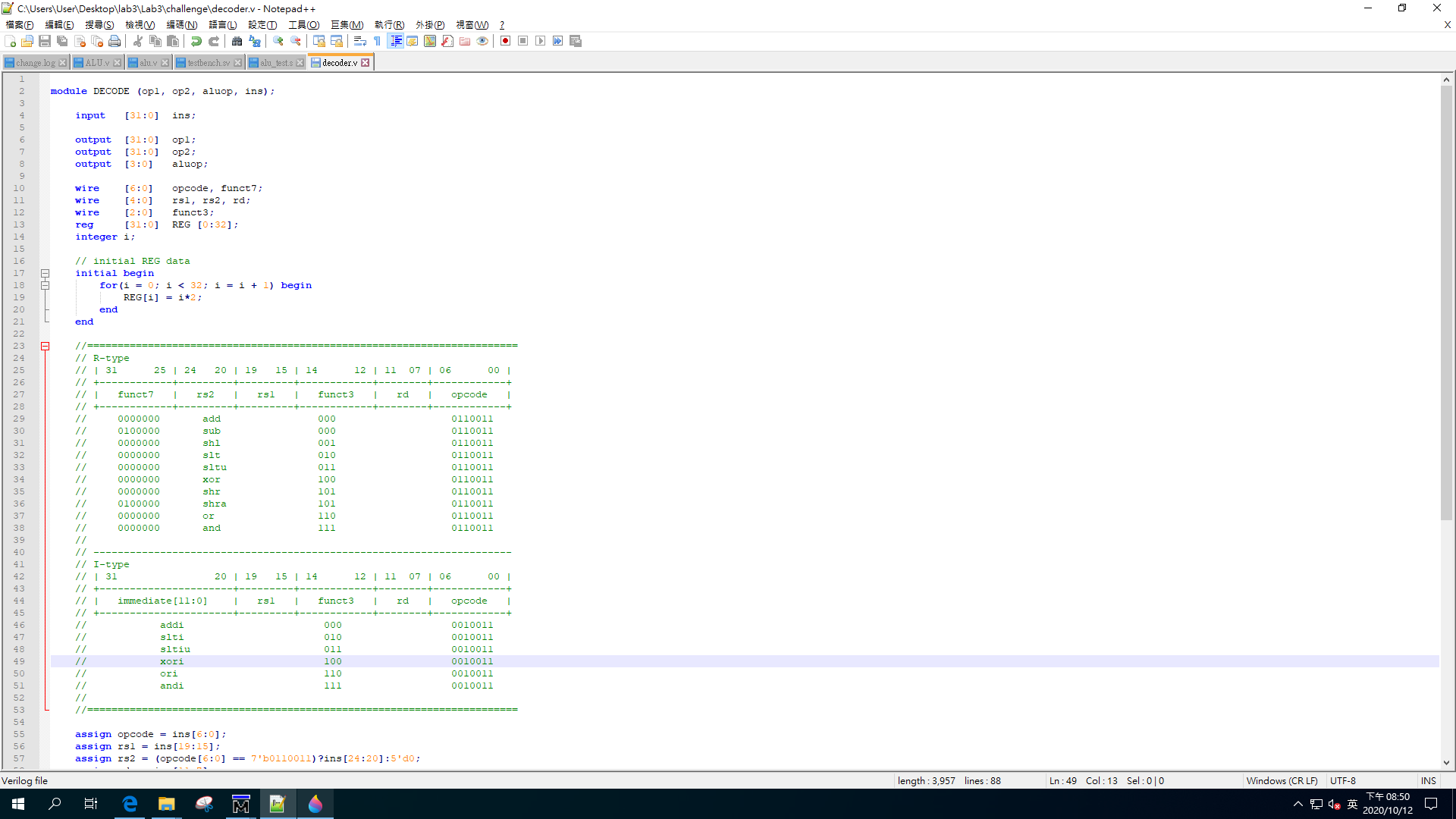
// SRL 0000000 101 0101

// SRA 0100000 101 1101

//=============================================

assign aluop = (opcode[6:0] == 7'b0110011)?{funct7[5], funct3}:{1'b0, funct3};

endmodule



一張含有 文字 的圖片

自動產生的描述

3.結果分析(modelsim)

一張含有 電子用品, 電腦, 監視器, 坐 的圖片

自動產生的描述

二、實驗心得(心得可隨意發揮，但是內容會做工程認證保留檔案)

李宇洋:

利用此次實驗了解了verilog的使用方法和CPU中ALU的運算流程，並從中熟悉ALU在各種運算過程的設計，且透過設計RISC-V中R-type和I-type指令的decoder 更加了解RISC-V ISA。

陳旭祺:

這次實驗是幫助我們熟練Verilog，以前雖然修過數位邏輯設計的課程，但沒寫過HDL，很多地方感覺與C語言相仿，但還是略微不同，畢竟使用功能不同，像always區塊描述(always@(感應列) begin 電路描述區域; end)，在這次實驗就讓我熟悉到他的指令與其代表的行為模式，還需多練習才能跟上大家的進度。

蔡晏齊:對於verilog不熟，於是再由assembly language轉換成verilog時，有點卡到，但後來發現verilog語言。只是指令上表達的不同，表達的程序邏輯是不變的，於是要更加強關於其指令的熟悉性。

蕭佑永:

本次實驗讓我了解了ALU跟Decoder大概是怎麼運作的，也複習了verilog！