# 計算機組織實驗

## 實驗七結果報告

### 第十六組組員

| 學號 | 姓名 |
| --- | --- |
| E24104189 | 謝宜烜 |
| E24106327 | 李貫銓 |
| H14086030 | 郭庭維 |

實驗日期:2023/1/8

1、 實驗內容

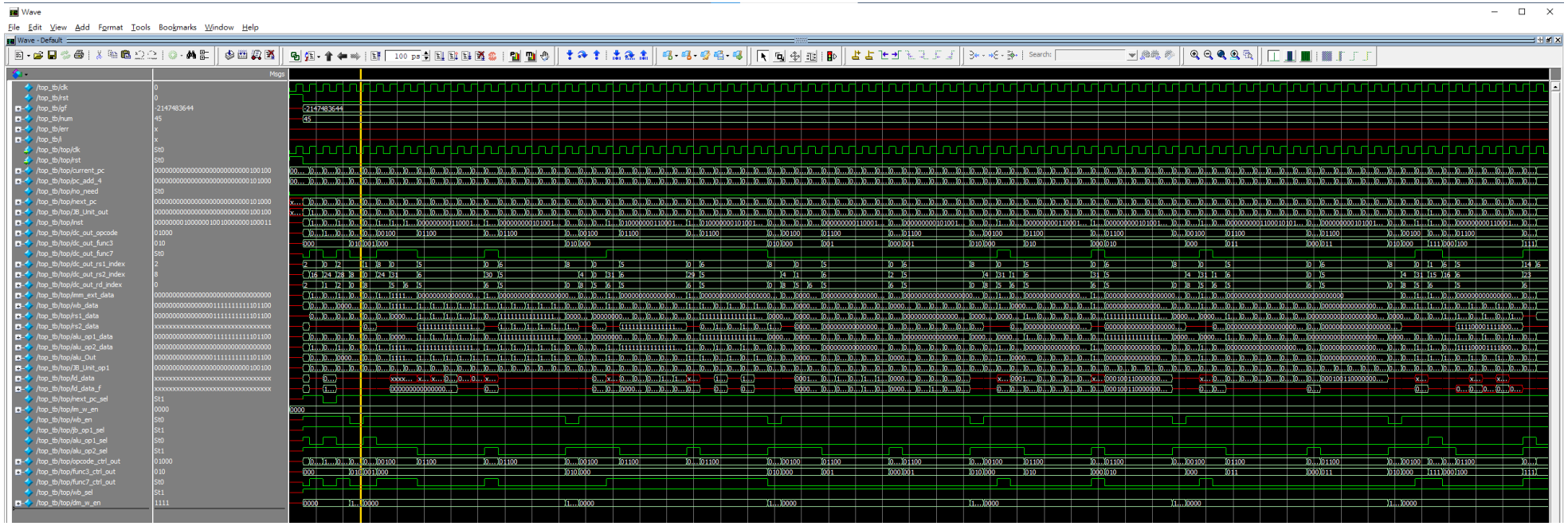
題目: Lab 7 Single-Cycle CPU Lab

2、 實驗說明:

(1) 完成結果圖

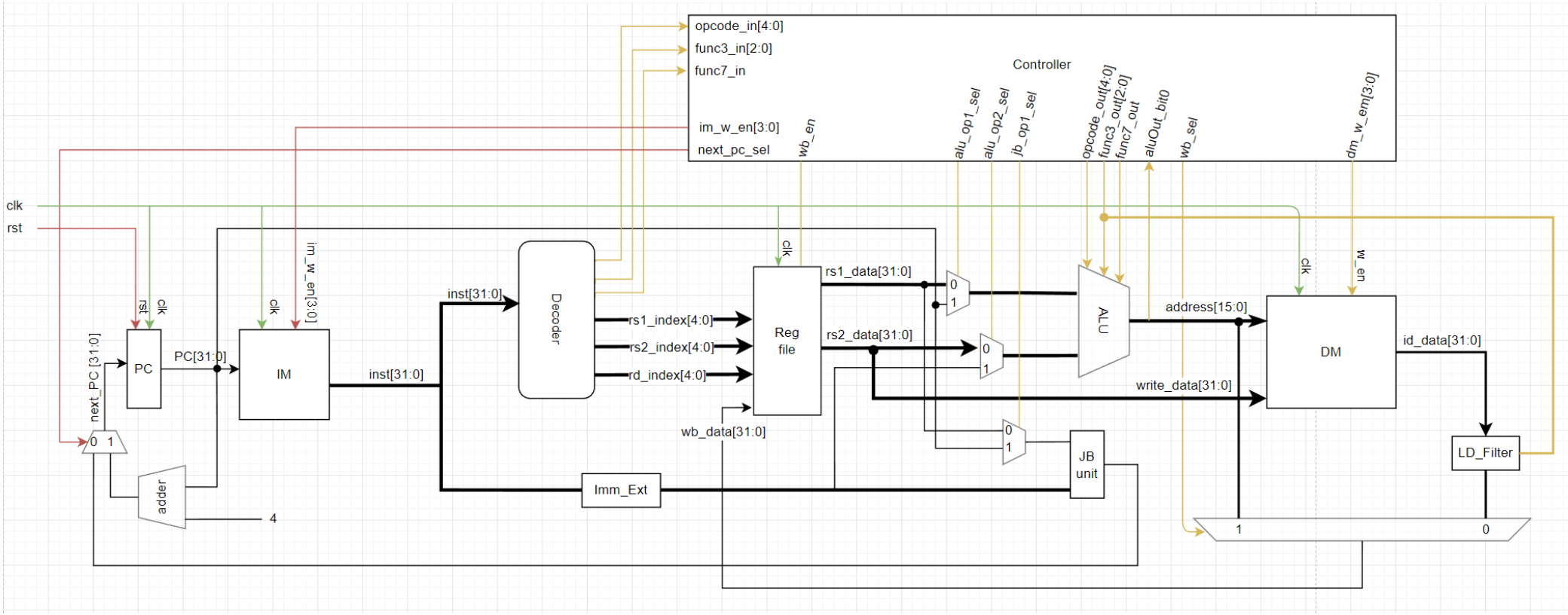
```
# DM['h9050] = 2468b7a8, pass
# DM['h9054] = 5dbf9f00, pass
# DM['h9058] = 00012b38, pass
# DM['h905c] = fa2817b7, pass
# DM['h9060] = ff000000, pass
# DM['h9064] = 12345678, pass
# DM['h9068] = 0000f000, pass
# DM['h906c] = 00000f00, pass
# DM['h9070] = 000000f0, pass
# DM['h9074] = 0000000f, pass
# DM['h9078] = 56780000, pass
# DM['h907c] = 78000000, pass
# DM['h9080] = 00005678, pass
# DM['h9084] = 00000078, pass
# DM['h9088] = 12345678, pass
# DM['h908c] = ce780000, pass
# DM['h9090] = fffff000, pass
# DM['h9094] = fffff000, pass
# DM['h9098] = fffff000, pass
# DM['h909c] = fffff000, pass
# DM['h90a0] = fffff000, pass
# DM['h90a4] = fffff000, pass
# DM['h90a8] = 13579d7c, pass
# DM['h90ac] = 13578000, pass
# DM['h90b0] = fffff004, pass
#
#
#
#
#                                        `;-.          ___,
#          ****************************    `.`\_...._/`.-"`
#          **                        **      \        /      ,
#          **                        **      /()    () \    .' `-._
#          **   Congratulations !!   **     |)  .    ()\ /   _.'
#          **                        **     \  -'-     ,; '. <
#          **                        **      ;.__     ,;|   > \
#          **   Simulation PASS!!    **     / ,    / ,  |.-'.-'
#          **                        **    (_/    (_/ ,;|.<`
#          **                        **      \    ,     ;-`
#          ****************************       >   \    /
#                                              (_,-'`> .'
#                                                   (_,'
#
#
# ** Note: $finish    : C:/modeltech64_10.1c/homework/lab7 2/top_tb.sv(96)
#    Time: 6225 ns  Iteration: 2  Instance: /top_tb
# 1
# Break in Module top_tb at C:/modeltech64_10.1c/homework/lab7 2/top_tb.sv line 96

VSIM 5>
```

(2) 波型圖


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
/top_tb/clk
/top_tb/rst
/top_tb/gf
/top_tb/num
/top_tb/err
/top_tb/i
/top_tb/top/clk
/top_tb/top/rst
/top_tb/top/current_pc
/top_tb/top/pc_add_4
/top_tb/top/no_need
/top_tb/top/next_pc
/top_tb/top/JB_Unit_out
/top_tb/top/inst
/top_tb/top/dc_out_opcode
/top_tb/top/dc_out_func3
/top_tb/top/dc_out_func7
/top_tb/top/dc_out_rs1_index
/top_tb/top/dc_out_rs2_index
/top_tb/top/dc_out_rd_index
/top_tb/top/imm_ext_data
/top_tb/top/wb_data
/top_tb/top/rs1_data
/top_tb/top/rs2_data
/top_tb/top/alu_op1_data
/top_tb/top/alu_op2_data
/top_tb/top/alu_Out
/top_tb/top/JB_Unit_op1
/top_tb/top/fd_data
/top_tb/top/fd_data_f
/top_tb/top/next_pc_sel
/top_tb/top/im_w_en
/top_tb/top/wb_en
/top_tb/top/jb_op1_sel
/top_tb/top/alu_op1_sel
/top_tb/top/alu_op2_sel
/top_tb/top/opcode_ctrl_out
/top_tb/top/func3_ctrl_out
/top_tb/top/func7_ctrl_out
/top_tb/top/wb_sel
/top_tb/top/dm_w_en

(3) CPU 架構圖

備註：CPU 架構採用助教給的格式。



3、 程式碼說明：（本次只修改了 ALU、Controller、RegFile、Adder、Imm_Ext，其餘使用助教範本）

(1) Top

連接成上圖的 CPU 架構，確保 Controller 的信號有正確接到每個 module 與 Mux 上面。並確保每個 module 之間的連接符合架構。

(2) Adder

將 lab4 的加法器改寫成 32bits 加法器，每個 cycle 將 PC+4。

(3) Mux

```
module Mux(
    input [31:0] in_0,
    input [31:0] in_1,
    input sel,
    output [31:0] o
);
    assign o = sel ? in_1 : in_0;
endmodule
```

廣泛用於程式中，透過”sel” 決定應該輸出哪個數值

(4) Reg_PC

```
module Reg_PC(
    input clk,
    input rst,
    input [31:0] next_pc,
    output reg [31:0] current_pc
);
    always @(posedge clk, posedge rst)begin
        if(rst)begin
            current_pc <= 32'd0;
        end
        else begin
            current_pc <= next_pc;
        end
    end
endmodule
```

決定下一個 PC 數值，一開始 rst posedge 會初始化到 0

(5) SRAM

```
module SRAM(
    input clk,
    input [3:0] w_en,
    input [15:0] address,
    input [31:0] write_data,
    output [31:0] read_data
);
    reg [7:0] mem[0:65535];

    // Sequential mem
    always @(posedge clk) begin
        if(w_en[0]) begin
            mem[address] <= write_data[7:0];
        end
        if(w_en[1]) begin
            mem[address + 1] <= write_data[15:8];
        end
        if(w_en[2]) begin
            mem[address + 2] <= write_data[23:16];
        end
        if(w_en[3]) begin
            mem[address + 3] <= write_data[31:24];
        end
    end

    // Output Combinational
    assign read_data = {mem[address + 3], mem[address + 2], mem[address + 1], mem[address]};

endmodule
```

負責與記憶體資料讀寫有關的功能、例如讀取對應位置(address)的指令(inst)，或將 write_data 的資料寫到對應的位置。

(6) Decoder

```
module Decoder(
    input [31:0] inst,
    output [4:0] dc_out_opcode,
    output [2:0] dc_out_func3,
    output       dc_out_func7,
    output [4:0] dc_out_rs1_index,
    output [4:0] dc_out_rs2_index,
    output [4:0] dc_out_rd_index
);
    assign dc_out_opcode = inst[6:2];
    assign dc_out_func3 = inst[14:12];
    assign dc_out_func7 = inst[30];
    assign dc_out_rs1_index = inst[19:15];
    assign dc_out_rs2_index = inst[24:20];
    assign dc_out_rd_index = inst[11:7];
endmodule
```

將 5.SRAM 讀取的到的指令（inst），拆分成 opcode、func3、func7、rs1_index、rs2_index、rd_index。並根據架構圖，分別傳給 controller 或 RegFile Module。

(7) RegFile

```
module RegFile (
            input clk,
            input wb_en,
            input [31:0] wb_data,
            input [4:0] rd_index,
            input [4:0] rs1_index,
            input [4:0] rs2_index,
            output [31:0] rs1_data_out,
            output [31:0] rs2_data_out
        );
reg [31:0] registers [0:31];
// register write
always @(posedge clk)
begin
    //hardwire reg x0 to gnd
    registers[0] <= 32'b0;
    // others registers
    if(wb_en && (rd_index !== 5'b0))
    begin
        registers[rd_index] <= wb_data;
    end
end
// output logic
// Finish by yourself
assign rs1_data_out = registers[rs1_index];
assign rs2_data_out = registers[rs2_index];

endmodule
```

將 rs1_index、rs2_index 的 reg 數值回傳。當 wb_en 且 rd 不是 x0 時，將 wb_data 寫到 wd_index 的 reg 中

(8) ALU：

```
// func3 define branch
`define BEQ     3'b000
`define BNE     3'b001
`define BLT     3'b100
`define BGE     3'b101
`define BLTU    3'b110
`define BGEU    3'b111

module ALU(
            input [4:0] opcode,
            input [2:0] func3,
            input       func7,
            input [31:0] operand1,
            input [31:0] operand2,
            output reg [31:0] alu_out
        );
always @(*)
begin
    case(opcode)
        `R_type, `I_arth://R-01100,I-00100
        begin
            case (func3)
                `ADD_SUB: //000
                begin
                    if(opcode == `R_type)
                    begin
                    // Finish by yourself
                        alu_out <= (func7)?(operand1-operand2):(operand1+operand2);
                        //func7=1->SUB,=0->ADD
                    end
                    else if(opcode == `I_arth)
                    begin
                        // Finish by yourself
                        alu_out <= operand1+operand2; //op1+imme[11,0]          /*/
```

```
            end
            else
            begin
                // Finish by yourself
                alu_out <= 32'bx;
            end
        end
        `SLL:       // Finish by yourself
            alu_out<=operand1<<operand2[4:0];
        `SLT:       // Finish by yourself  **set less than signed**  /*/
            alu_out<={{31{1'b0}},($signed(operand1)<$signed(operand2))};//?1'b1:1'b0;
        `SLTU:      // Finish by yourself  **set less than unsigned**   /*/
            alu_out<={{31{1'b0}},(operand1<operand2)};//?1'b1:1'b0;
        `XOR:       // Finish by yourself
            alu_out<=operand1^operand2;
        `SRL_SRA:   // Finish by yourself
            alu_out<= (~func7)? ($signed(operand1)>>operand2[4:0]) : ($signed(operand1) >>> operand2[4:0]);
        `OR:        // Finish by yourself
            alu_out<=operand1|operand2;
        `AND:       // Finish by yourself
            alu_out<=operand1&operand2;
        default:    // Finish by yourself
            alu_out<=32'bx;
    endcase
end
`LUI:           // Finish by yourself
    alu_out<=operand2;
`AUIPC:         // Finish by yourself
    alu_out<=operand1+operand2;
`I_load, `S_type:   // Finish by yourself
    alu_out<=operand1+operand2; //=>address
`J_type, `JALR:     // Finish by yourself
    alu_out<=operand1+4; //=>pc+4
`B_type:
```

根據 controller 給予的 opcode／func3 的數值不同判斷為何種類型的指令（R_type、U_type....），執行對應的運算或操作。例如：add、sll、slt、beq 等。

(9) Controller

```
`define is_LUI      (opcode_in == 5'b01101)
`define is_AUIPC    (opcode_in == 5'b00101)
`define is_U_type   (`is_LUI || `is_AUIPC)
`define is_J_type   (opcode_in == 5'b11011)
`define is_B_type   (opcode_in == 5'b11000)
`define is_I_load   (opcode_in == 5'b00000)
`define is_I_arth   (opcode_in == 5'b00100)
`define is_JALR     (opcode_in == 5'b11001)
`define is_I_type   (`is_I_load || `is_I_arth || `is_JALR)
`define is_S_type   (opcode_in == 5'b01000)
`define is_R_type   (opcode_in == 5'b01100)

module Controller(
            input       [4:0]   opcode_in,
            input       [2:0]   func3_in,
            input               func7_in,
            input               aluOut_bit0,
            output reg          next_pc_sel,
            output      [3:0]   im_w_en,
            output reg          wb_en,
            output reg          jb_op1_sel,
            output reg          alu_op1_sel,
            output reg          alu_op2_sel,
            output      [4:0]   opcode_out,
            output      [2:0]   func3_out,
            output              func7_out,
            output reg          wb_sel,
            output reg  [3:0]   dm_w_en
        );

assign im_w_en = 4'b0;
assign opcode_out = opcode_in;
assign func3_out = func3_in;
assign func7_out = func7_in;
```

```
// next_pc_sel
always @(opcode_in, aluOut_bit0)
begin
    if(`is_J_type || `is_JALR || (`is_B_type && aluOut_bit0))
    begin
        // Finish by yourself
        next_pc_sel <= 1'b0;
    end
    else
    begin
        // Finish by yourself
        next_pc_sel <= 1'b1;
    end
end

// wb_en
always @(opcode_in)
begin
    if(`is_U_type || `is_J_type || `is_I_type || `is_R_type)
    begin
        // Finish by yourself
        wb_en <= 1'b1;
    end
    else
    begin
        // Finish by yourself
        wb_en <= 1'b0;
    end
end

// jb_op1_sel
always @(opcode_in)
begin
    if(`is_JALR)
    begin
```

類似 ALU，根據讀到的 Opcode 不同，判斷應為何種指令，並決定每個 Mux 與 enable 腳位（e.g. wb_en, im_w_en），應該為 0 或 1。

(10)　Imm_Ext

```
module Imm_Ext(
            input [31:0] inst,
            output reg [31:0] imm_ext_out
        );
always @(*)
begin
    casex(inst[6:2])
        `R_type: imm_ext_out = 32'b0;   // Finish by yourself
        `I_type: imm_ext_out = {{20{inst[31]}}, inst[31: 20]}; // Finish by yourself
        `S_type: imm_ext_out = {{20{inst[31]}},inst[31:25],inst[11:7]};   // Finish by
        `B_type: imm_ext_out = {{20{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0};
        `U_type: imm_ext_out = {inst[31:12],12'b0}; // Finish by yourself
        `J_type: imm_ext_out = {{12{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0};
         default: imm_ext_out = 32'b0;  // Finish by yourself
    endcase
end
endmodule
```

根據不同的指令型態，將 inst 中的 immediate 數值挑出來並排好，相關格式規定參考 Lab7 投影片 P37

(11)　JB_Unit

```
module JB_Unit(
    input [31:0] operand1,
    input [31:0] operand2,
    output [31:0] jb_out
);
    assign jb_out = (operand1 + operand2) & {{31{1'b1}}, 1'b0};

endmodule
```

計算遇到 beq 等指令需要跳行時，計算新的 PC 位置。

(12)　LD_Filter

```
module LD_Filter(
    input [2:0] func3,
    input [31:0] ld_data,
    output reg [31:0] ld_data_f
);
    always @(*)begin
        case(func3)
            `lb:        ld_data_f <= {{24{ld_data[7]}}, ld_data[7:0]};
            `lh:        ld_data_f <= {{16{ld_data[15]}}, ld_data[15:0]};
            `lw:        ld_data_f <= ld_data;
            `lbu:       ld_data_f <= {24'b0, ld_data[7:0]};
            `lhu:       ld_data_f <= {16'b0, ld_data[15:0]};
            default:    ld_data_f <= 32'bx;
        endcase
    end
endmodule
```

處理 SRAM module 讀取到的資料，根據讀入的是 byte、half word、word、是否為 unsigned 等，來做 sign extended。

## 4、 實驗心得

謝宜烜：這次的 final lab 難度大幅的提升，除了要對語言的熟悉以外，也要對 CPU 有一定的了解，才可以用語言把他的結構寫出來。在這次的 lab 中我真的學了很多，之前不懂的也逐漸可以融會貫通。

郭庭維：最後一次實驗課程了，很高興這次有機會嘗試自己寫了一個 RISC-V 架構的 CPU。可以把老師上課的東西實作一次還是很有意義的。一開始看到還有點卻步，但透過助教給的 template 的協助就沒有原本想的那麼困難了，透過這次的 lab 學到了很多的東西，也謝謝助教一個學期以來的指導！

李貫銓：這次 lab7 內容相較之前的幾次課程，算是比較龐大的，但所幸我們需要自己撰寫的部分不多，所以有避免掉一些常發生的錯，像是不同 module 間接腳接錯之類的。而這次的內容跟上課時所教的息息相關，算是再次讓我們熟悉這學期所學到的知識，很開心能順利完成這學期的課！