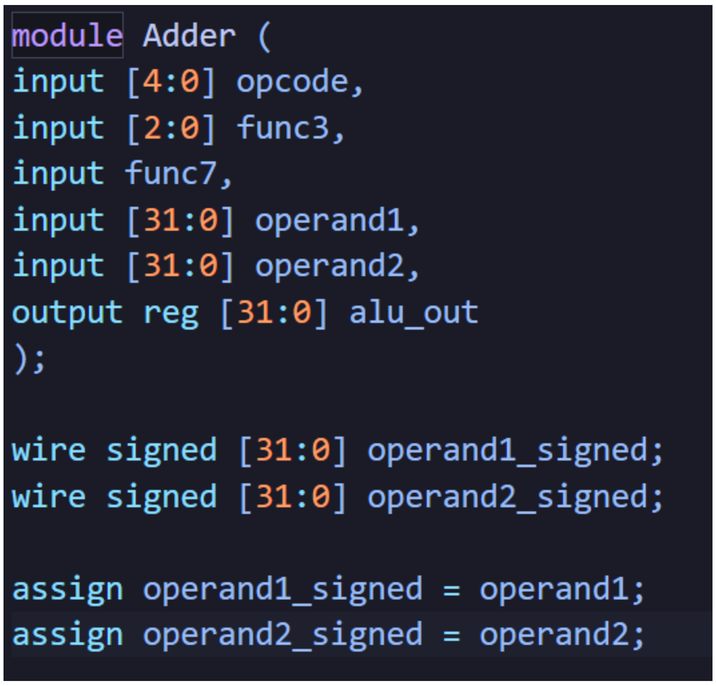
Introduce each module:

module Adder:

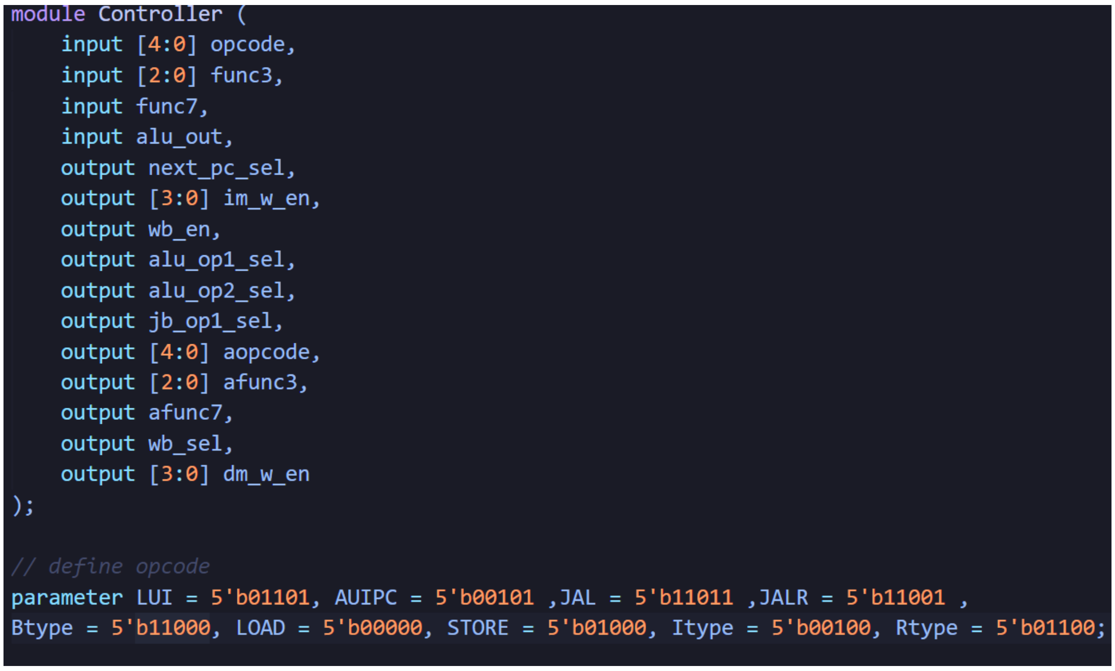


```
module Adder (
input [4:0] opcode,
input [2:0] func3,
input func7,
input [31:0] operand1,
input [31:0] operand2,
output reg [31:0] alu_out
);

wire signed [31:0] operand1_signed;
wire signed [31:0] operand2_signed;

assign operand1_signed = operand1;
assign operand2_signed = operand2;
```

ALU :

使用 opcode、func3 和 func7 來決定 ALU 對 operand1 及 operand2 做什麼運算，並新增兩個 wire 把 unsigned 的 operand1 及 operand2 轉乘 signed 做運算，輸出 ALU 運算的結果

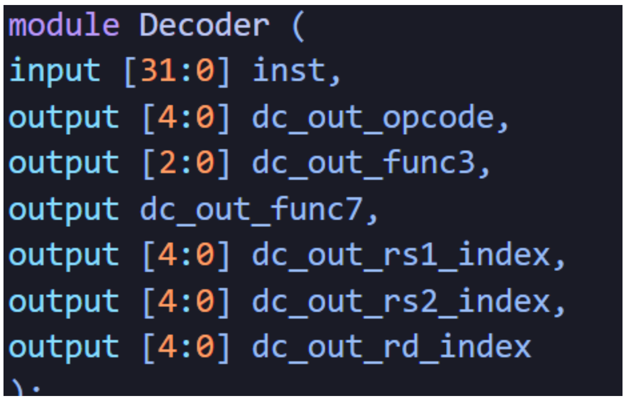
module Controller:



```verilog
module Controller (
    input [4:0] opcode,
    input [2:0] func3,
    input func7,
    input alu_out,
    output next_pc_sel,
    output [3:0] im_w_en,
    output wb_en,
    output alu_op1_sel,
    output alu_op2_sel,
    output jb_op1_sel,
    output [4:0] aopcode,
    output [2:0] afunc3,
    output afunc7,
    output wb_sel,
    output [3:0] dm_w_en
);

// define opcode
parameter LUI = 5'b01101, AUIPC = 5'b00101 ,JAL = 5'b11011 ,JALR = 5'b11001 ,
Btype = 5'b11000, LOAD = 5'b00000, STORE = 5'b01000, Itype = 5'b00100, Rtype = 5'b01100;
```

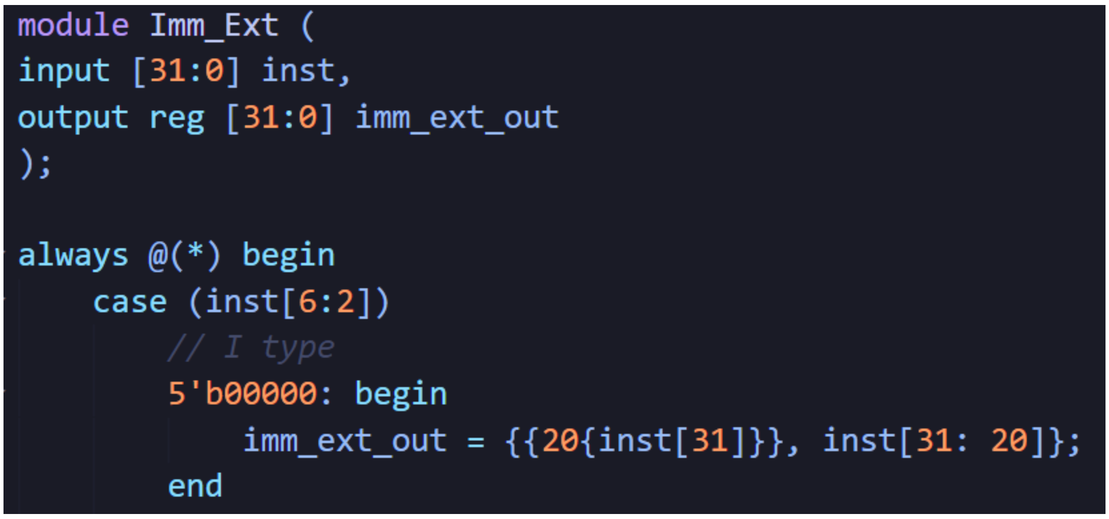
透過 opcode、func3、func7、alu_out 對各個 MUX、IM、Reg File、ALU 及 DM 生成控制訊號

module Decoder:



```verilog
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
);
```

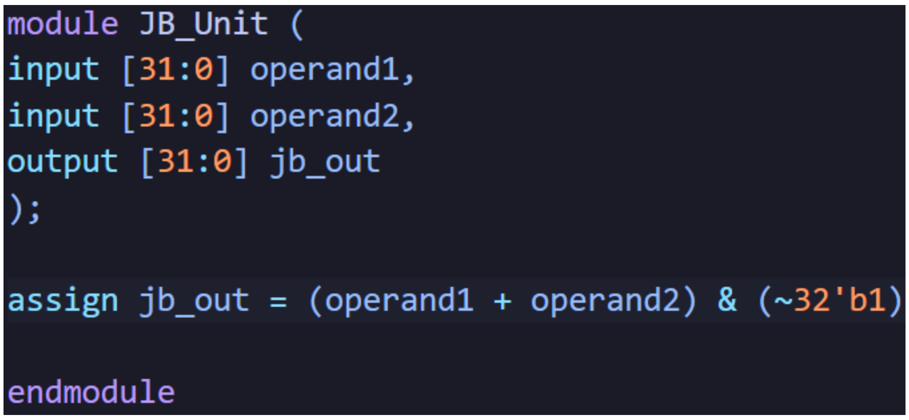
從 inst 中獲得 dc_out_opcode、dc_out_func3、dc_out_func7、dc_out_rs1_index、dc_out_rs2_index 及 dc_out__index module Imm_Ext:



```
module Imm_Ext (
input [31:0] inst,
output reg [31:0] imm_ext_out
);

always @(*) begin
    case (inst[6:2])
        // I type
        5'b00000: begin
            imm_ext_out = {{20{inst[31]}}, inst[31: 20]};
        end
```

根據 inst[6:2]=opcode 做 immediate exten

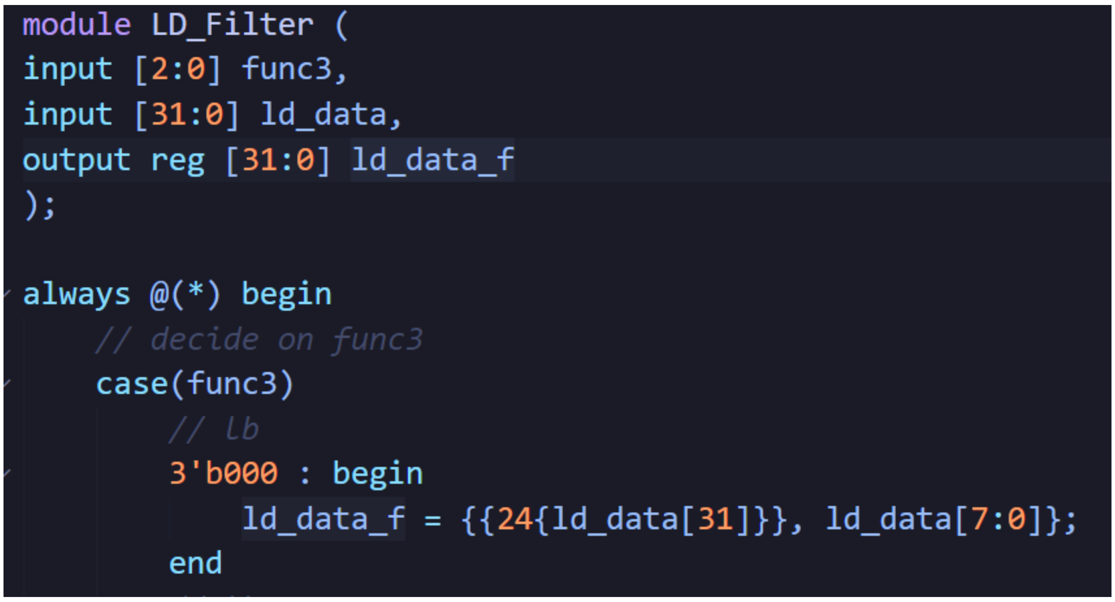
module JB_Unit:



```
module JB_Unit (
input [31:0] operand1,
input [31:0] operand2,
output [31:0] jb_out
);

assign jb_out = (operand1 + operand2) & (~32'b1)

endmodule
```

計算 jump address

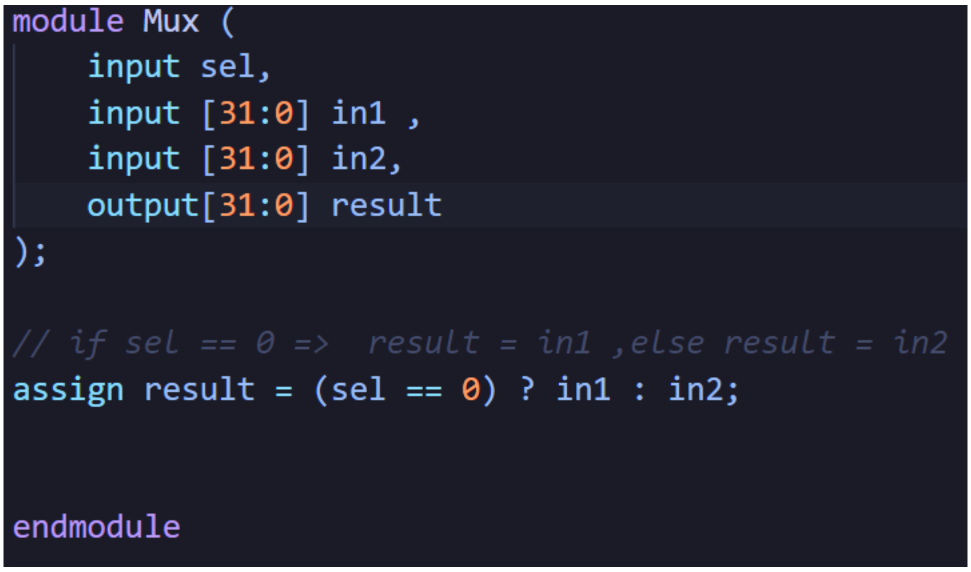
module LD_Filter:



```
module LD_Filter (
input [2:0] func3,
input [31:0] ld_data,
output reg [31:0] ld_data_f
);

always @(*) begin
    // decide on func3
    case(func3)
        // lb
        3'b000 : begin
            ld_data_f = {{24{ld_data[31]}}, ld_data[7:0]};
        end
```

根據 func3 決定是從 memory 做 lb、lh、lw、lbu 還是 lhu

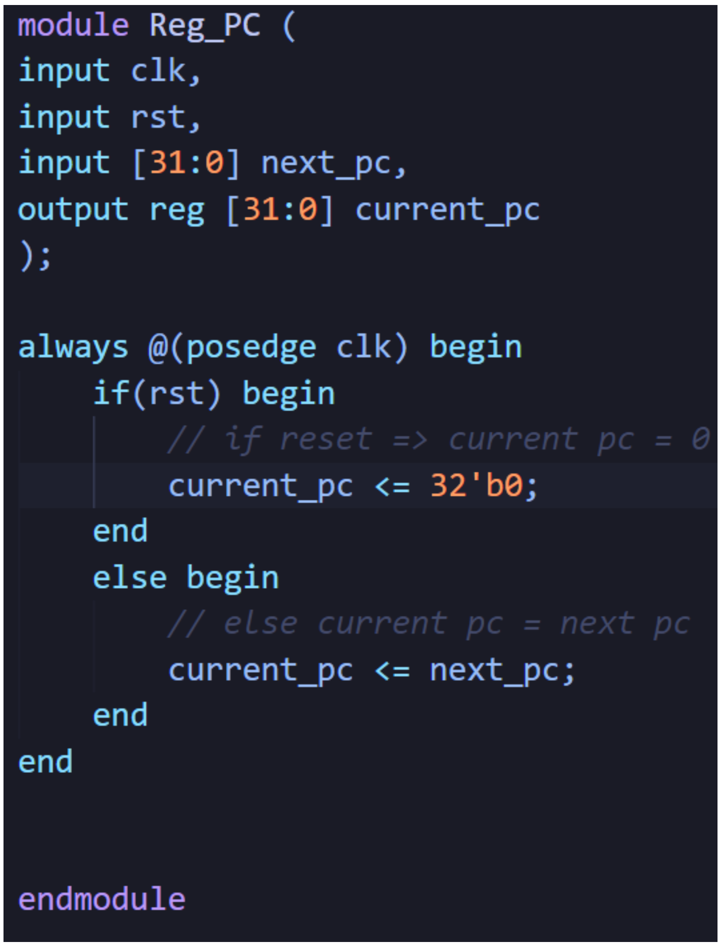
module Mux:



```
module Mux (
    input sel,
    input [31:0] in1 ,
    input [31:0] in2,
    output[31:0] result
);

// if sel == 0 =>  result = in1 ,else result = in2
assign result = (sel == 0) ? in1 : in2;


endmodule
```

根據 controller 生成的控制訊號，控制通過的參數

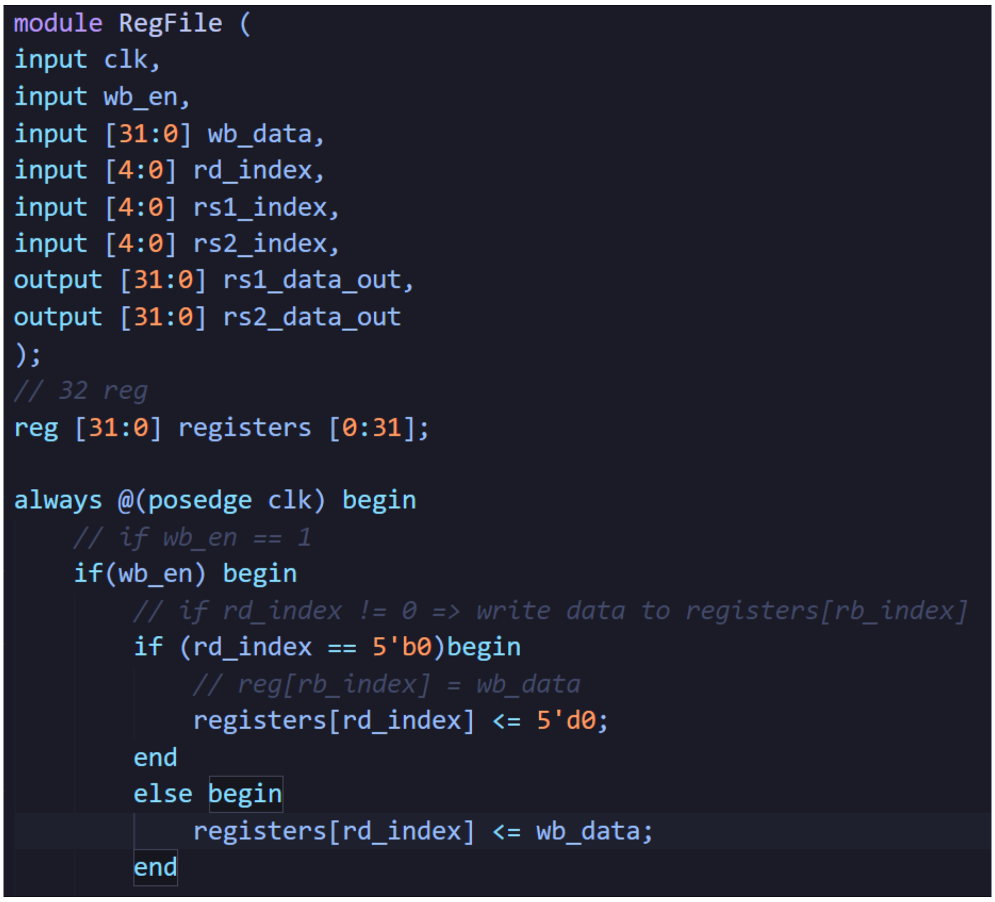
module Reg_PC:


```
module Reg_PC (
input clk,
input rst,
input [31:0] next_pc,
output reg [31:0] current_pc
);

always @(posedge clk) begin
    if(rst) begin
        // if reset => current pc = 0
        current_pc <= 32'b0;
    end
    else begin
        // else current pc = next pc
        current_pc <= next_pc;
    end
end


endmodule
```

If reset => current pc = 0

else current pc = next pc

module RegFile:



```verilog
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
);
// 32 reg
reg [31:0] registers [0:31];

always @(posedge clk) begin
    // if wb_en == 1
    if(wb_en) begin
        // if rd_index != 0 => write data to registers[rb_index]
        if (rd_index == 5'b0)begin
            // reg[rb_index] = wb_data
            registers[rd_index] <= 5'd0;
        end
        else begin
            registers[rd_index] <= wb_data;
        end
```

共有 32 個 reg

if wb_en == 1 =>

if rd_index == 0 => reg[rb_index] = 5'd0

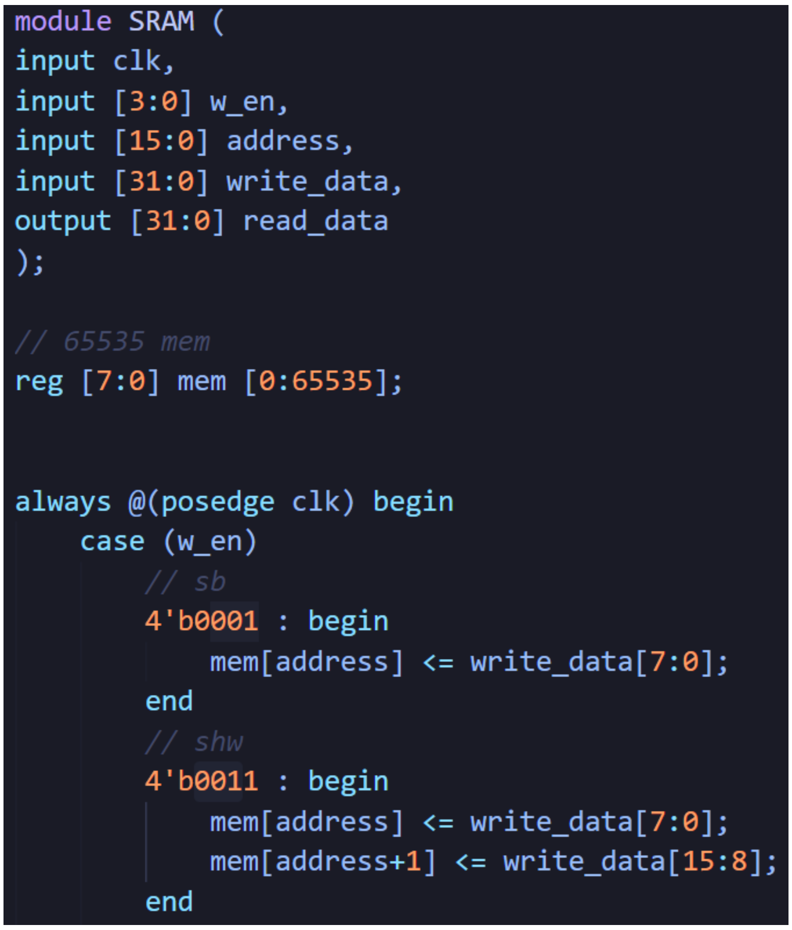
else => reg[rb_index] = wb_data

output:

data of rs1 = registers[res1_index]

data of rs2 = registers[res2_index]

module SRAM

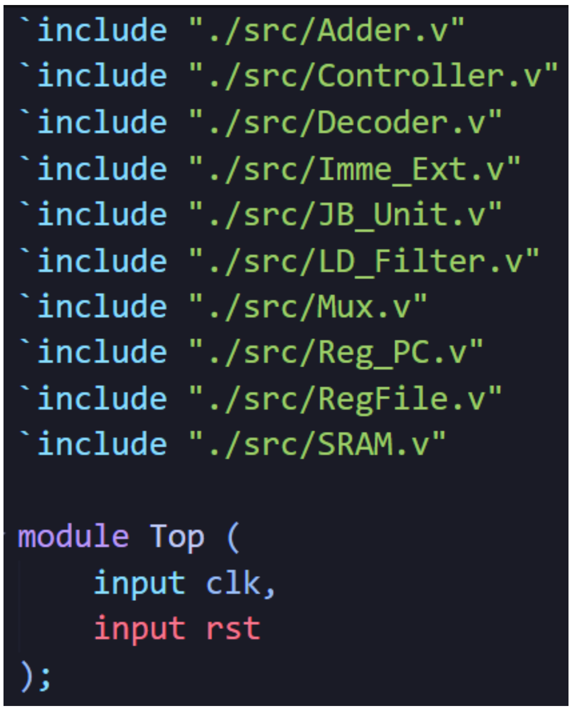


```
module SRAM (
input clk,
input [3:0] w_en,
input [15:0] address,
input [31:0] write_data,
output [31:0] read_data
);

// 65535 mem
reg [7:0] mem [0:65535];


always @(posedge clk) begin
    case (w_en)
        // sb
        4'b0001 : begin
            mem[address] <= write_data[7:0];
        end
        // shw
        4'b0011 : begin
            mem[address] <= write_data[7:0];
            mem[address+1] <= write_data[15:8];
        end
```

根據 w_en 決定是 sb、shw 還是 sw。

read_data=

{mem[address+3],mem[address+2],mem[address+1],mem[address]} module Top



```verilog
`include "./src/Adder.v"
`include "./src/Controller.v"
`include "./src/Decoder.v"
`include "./src/Imme_Ext.v"
`include "./src/JB_Unit.v"
`include "./src/LD_Filter.v"
`include "./src/Mux.v"
`include "./src/Reg_PC.v"
`include "./src/RegFile.v"
`include "./src/SRAM.v"

module Top (
    input clk,
    input rst
);
```

最後透過 Top module 將所有元件串起來