M11212913 HW3 report

ALU.v

```
module ALU (
    input [31:0] data1_i,
    input [31:0] data2_i,
    input [2:0] ALUCtrl_i,
    output reg [31:0] data_o,
    output reg zero_o);
```

定義作業規定的輸入輸出 port 型別。

```
    always @(*) begin
        if (ALUCtrl_i == 3'b000) begin // Addition
            data_o = data1_i + data2_i;
        end else if (ALUCtrl_i == 3'b001) begin
            data_o = data1_i - data2_i; // Subtraction
        end else if (ALUCtrl_i == 3'b010) begin
            data_o = data1_i & data2_i; // Bitwise AND
        end else if (ALUCtrl_i == 3'b011) begin
            data_o = data1_i | data2_i; // Bitwise OR
        end else if (ALUCtrl_i == 3'b100) begin
            data_o = data1_i ^ data2_i; // Bitwise XOR
        end else if (ALUCtrl_i == 3'b101) begin
            data_o = data1_i << data2_i[4:0]; // Left Shift
        end else if (ALUCtrl_i == 3'b110) begin
            data_o = $signed(data1_i) >>> data2_i[4:0]; // Arithmetic Right Shift
        end else if (ALUCtrl_i == 3'b111) begin
            data_o = data1_i >> data2_i[4:0]; // Logical Right Shift
        end else begin
            data_o = 32'b0;
        end

        if (data_o == 32'b0) begin
            zero_o = 1;
        end else begin
            zero_o = 0;
        end
    end

endmodule
```

使用 always block 組合邏輯來根據輸入的 ALUCtrl_i 對 data1_i 與 data2_i 立即做對應的運算，結果存到 data_o。
再根據 data_o 的結果 zero_o 會放入 1(運算結果為零) 或 0(運算結果不為零)。

ALU_tb.v (ALU testbench)

OS: Windows

Commands:

```
iverilog -o alu ALU.v ALU_tb.v
vvp alu
gtkwave ALU.vcd
```

附檔 ALU_tb.v 為 ALU.v 的 testbench。
1.先是定義模擬的時間單位和精度為 1ns / 1ps
2.作業規定測試信號的宣告，以及對應到 module ALU 的接口
3.接下來的 initial block 內有生成 ALU.vcd 檔，以及每種運算至少都有一筆測試，結果正確或不正確都會打印出來，其中加法與減法有額外幾筆測試，測試超過可表示範圍時看是否會像 C 的無號一樣 wrap around
4.檢查運算的結果為 0 時， zero_o 是否為 1

附檔圖片 1.png 為運算結果顯示在 CMD 與用 GTKWave 生成測試的波形。

Registers.v

```
module Registers (
    input clk_i,
    input [4:0] RS1addr_i,
    input [4:0] RS2addr_i,
    input [4:0] RDaddr_i,
    input [31:0] RDdata_i,
    input RegWrite_i,
    output reg [31:0] RS1data_o,
    output reg [31:0] RS2data_o);
```

定義作業規定的輸入輸出 port 型別。

```
    reg [31:0] registers_array [0:31];

    initial begin
    registers_array[0] = 32'd0; // register 0 = 0
    end
```

定義一個 32 bits x 32 個的陣列存 data。
初始化 register 0 = 0。

```
    always @(posedge clk_i) begin
        if (RegWrite_i) begin
            if (RDaddr_i != 5'd0) begin // 不是 register 0 才能被寫入
                registers_array[RDaddr_i] <= RDdata_i;
            end
        end
    end

    always @(*) begin
        if (RS1addr_i == 5'd0) begin
            RS1data_o = 32'd0;
        end else begin
            RS1data_o = registers_array[RS1addr_i];
        end

        if (RS2addr_i == 5'd0) begin
            RS2data_o = 32'd0;
        end else begin
            RS2data_o = registers_array[RS2addr_i];
        end
    end

endmodule
```

第一個 always block 用序向邏輯，只有 clk_i 由 0 到 1 加上 RegWrite_i 是 1 時，RDdata_i 的 data 才能寫入 registers_array[RDaddr_i] (第 i 個 register)，第 0 個 register 都是 0，不允許被寫入。

第二個 always block 組合邏輯，隨時根據 RS1，RS2 來源地址立即輸出對應到的第 i 個 register 的值，第 0 個都是輸出 0。

Registers_tb.v (Registers testbench)

OS: Windows

Commands:

```
iverilog -o reg Registers.v Registers_tb.v
vvp reg
gtkwave Registers.vcd
```

附檔 Registers_tb.v 為 Registers.v 的 testbench。
1.先是定義模擬的時間單位和精度為 1ns / 1ps
2.作業規定測試信號的宣告，以及對應到 module Registers 的接口
3.一個 clk_i 的完整週期設 20 ns，接下來的 initial block 內有生成 Registers.vcd 檔，並初始化所有輸入信號為 0
4.隨意對幾個 Registers 做測試，看有沒有符合 clk 0 -> 1 且 RegWrite_i 為 1 才能寫入

5.測試 Register 0 無法被寫入
6.隨時能讀取指定的 register

附檔圖片 2.png 為測試結果顯示在 CMD 與用 GTKWave 生成測試的波形。