Computer Organization

Homework 1

(32-bits Complete ALU)

Student :

B11107157 電機三乙 林明宏

Teacher :

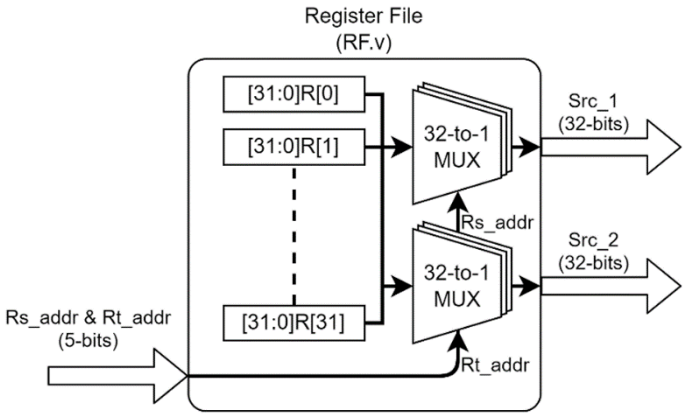
陳雅淑

RTL Simulator :

ModelSim-Intel FPGA Standard Edition, Version 20.1.1, windows

1. Screenshot and descriptions of each module :

(a) 32-bits Read Only Register File :



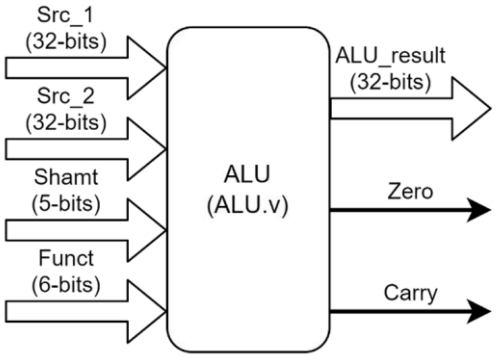

```
module RF(
    //Inputs
    input [4:0] Rs_addr,
    input [4:0] Rt_addr,
    //Outputs
    output [31:0] Src_1,
    output [31:0] Src_2

);

    reg [31:0]R[0:31]; //32bits Register File
    assign Src_1 = R[Rs_addr];
    assign Src_2 = R[Rt_addr];
endmodule
```

透過 Register File 圖
可以知道該架構是透過 Rs_addr 和 Rt_addr 兩個 Address 來選擇要輸出的 Register 資料
因此我們先宣告出 32 個 32bits 的 Register，再利用 assign 來輸出想要的位址即可完成。

(b) 32-bits Arithmetic Logic Unit :

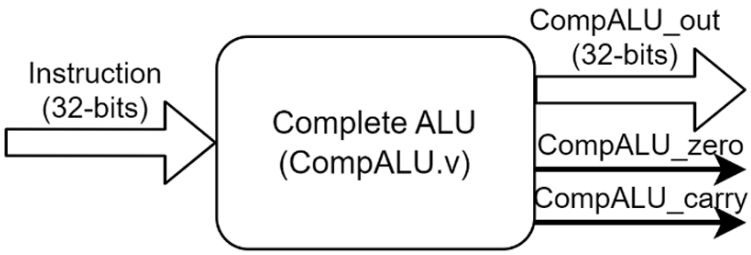



```
1  `define ADDU  6'b100100
2  `define SUBU  6'b100011
3  `define OR  6'b100101
4  `define SRL  6'b000010
5  `define SLL  6'b000000
6
7  module ALU(
8      input [31:0] Src_1,
9      input [31:0] Src_2,
10     input [4:0] Shamt,
11     input [5:0] Funct,
12     output reg [31:0] ALU_result,
13     output Zero,
14     output reg Carry
15 );
16     always @(*) begin
17         case(Funct)
18             `ADDU: {Carry,ALU_result} = Src_1 + Src_2;
19             `SUBU: begin
20                 ALU_result = Src_1 - Src_2;
21                 Carry = (Src_1>=Src_2)? 1'b0:1'b1;
22             end
23             `OR: {Carry,ALU_result} = (Src_1|Src_2);
24             `SRL: {Carry,ALU_result} = Src_1 >> Shamt;
25             `SLL: {Carry,ALU_result} = Src_1 << Shamt;
26             default:;
27         endcase
28     end
29     assign Zero = (ALU_result==32'b0);
30 endmodule
```

利用 case 語法去判斷 Funct 的值
然後分流各個不同的運算式
接著利用 assign 讓 Zero 在結果為 0 時為 High，反之則 Low。

Funct 的運算式如其定義名稱所示
例如 ADDU 代表無號數的加法...等。
Carry 則當有進位或借位的發生設為 1。

(c) 32-bits Complete ALU :




```
module CompALU(
    //  Inputs
    Instruction,
    //  Outputs
    CompALU_out,
    CompALU_zero,
    CompALU_carry
);
    input [31:0] Instruction;  //31-26 25-21 20-16 15-11 10-6 5-0
        output [31:0] CompALU_out;
        output CompALU_zero;
        output CompALU_carry;

    wire [31:0] Src_1,Src_2;
        RF Register_File(Instruction[25:21],Instruction[20:16],Src_1,Src_2);
        ALU Arithmetic_Logical_Unit(Src_1,Src_2,Instruction[10:6],Instruction[5:0],CompALU_out,CompALU_zero,CompALU_carry);
endmodule
```

利用一個 Complete ALU Module 把上述的(a)、(b) Module 連接起來。
而 32bits 的 Instruction 則是按照 R-Format 去設定，如下圖所示
依照 Instruction 各個部份的功能分別接入 RF 和 ALU 裡面。

OP Code	Source Register	Target Register	Destination Register	Shamt	Funct Code
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

(R-Format Instruction)

2. Screenshots and descriptions of test commands for each module :

(a) tb_ALU.in:

```
1  00000000000000000000000000000000_00000000000000000000000000000000_00001_100100
2  00000000000000000000000000000011_00000000000000000000000000000001_00100_100100
3  00000000000000000000000000000110_00000000000000000000000000001001_00010_100011
4  00000000000000001000000000000000_00000000000000001000000000000000_00100_100011
5  11111111111111111111111111111111_00000000000000000000000001111111_00011_100100
6  01010101010101010101010101010101_10101010101010101010101010101010_11111_100101
7  00000000000000000000000000000001_00000000000000000000010111100000_00010_000000
8  10000000000000000000000000000000_00000000000000000000001011110000_00001_000000
9  00000000000000000000000000100000_00000000000000000000010111100000_00100_000010
```

以下為每個 Test Pattern 的每個 input 表格 ：

Test Pattern	Src_1	Src_2	Shamt	Funct
1	00000000000000000000000000000000	00000000000000000000000000000000	00001	100100 (ADDU)
2	00000000000000000000000000000011	00000000000000000000000000000001	00100	100100 (ADDU)
3	00000000000000000000000000000110	00000000000000000000000000001001	00010	100011 (SUBU)
4	00000000000000001000000000000000	00000000000000001000000000000000	00100	100011 (SUBU)
5	11111111111111111111111111111111	00000000000000000000000001111111	00011	100100 (ADDU)
6	01010101010101010101010101010101	10101010101010101010101010101010	11111	100101 (OR)
7	00000000000000000000000000000001	00000000000000000000010111100000	00010	000000 (SLL)
8	10000000000000000000000000000000	00000000000000000000001011110000	00001	000000 (SLL)
9	00000000000000000000000000100000	00000000000000000000010111100000	00100	000010 (SRL)

測試目的 ：

Test Pattern 1： 測試加法和 Zero Flag

Test Pattern 2： 測試加法

Test Pattern 3： 測試減法與借位 Carry Flag

Test Pattern 4： 測試減法和 Zero Flag

Test Pattern 5： 測試加法和進位 Carry Flag

Test Pattern 6： 測試 OR

Test Pattern 7： 測試向左位移

Test Pattern 8： 測試向右位移和 Carry Flag 和 Zero Flag

Test Pattern 9： 測試向右位移

(b) tb_CompALU.in:

```
1   000000_00011_00000_00000_00001_100100
2   000000_00100_00011_00000_00000_100011
3   000000_01000_11000_00000_11111_100101
4   000000_00010_00110_00000_00010_000000
5   000000 00111 00010 00000 00100 000010
```

以下為每個 Test Pattern Instruction 各部分的表格 ：

Test Pattern	Opcode (31:26)	Source Register (25:21)	Target Register (20:16)	Destination Register (15:11)	Shamt (10:6)	Funct (5:0)
1	0000000	00011	00000	00000	00001	100100
2	0000000	00100	00011	00000	00000	100011
3	0000000	01000	11000	00000	11111	100101
4	0000000	00010	00110	00000	00010	000000
5	0000000	00111	00010	00000	00100	000010

測試目的 ：

Test Pattern 1： 測試加法

Test Pattern 2： 測試減法

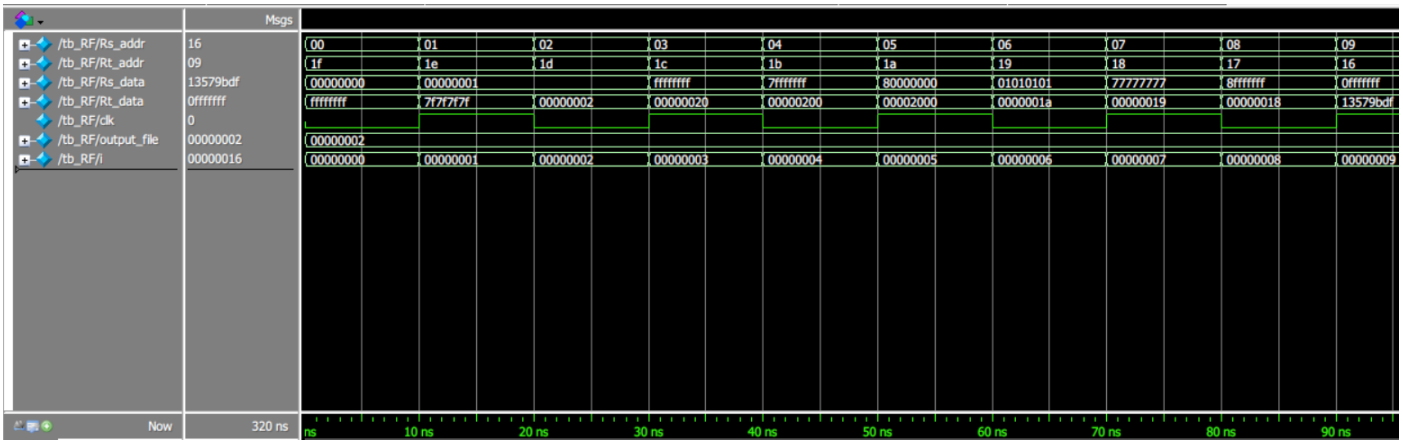
Test Pattern 3： 測試 OR

Test Pattern 4： 測試向左位移

Test Pattern 5： 測試向右位移

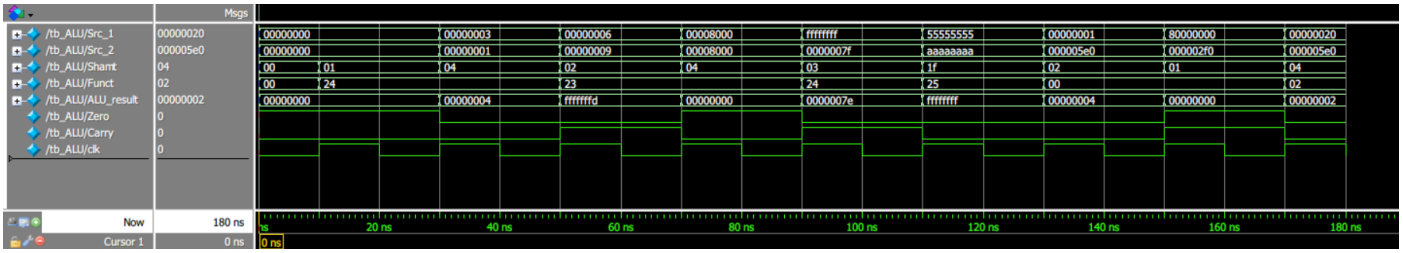
3. Screenshots and explanations of the test result for each module :

(a) tb_RF



該 Test 送入了很多組不同的 Address，由波型圖可觀察到其 Rs_data 和 Rt_data 可以參考 RF.dat 中的檔案來確定是否正確。

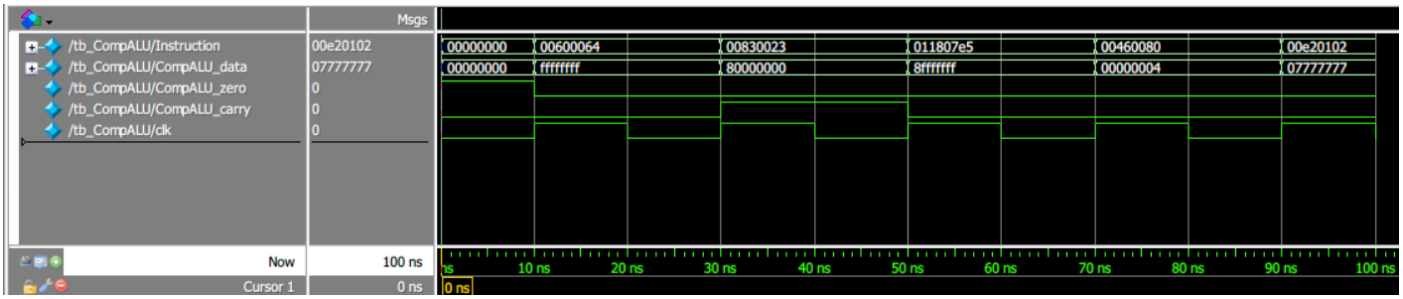
(b) tb_ALU



該 Test 如第二部分(a)所示，送入了 9 組不同的 Test Pattern，分別測試不同的運算式。
如上圖波型所示，皆為正確結果。

Test Pattern	運算式	正確 Carry	正確 Zero	正確 ALU_Result
1	00000000 + 00000000	0	1	00000000
2	00000003 + 00000001	0	0	00000004
3	00000006 - 00000009	1(借位)	0	fffffffd
4	00000080 - 00000800	0	1	00000000
5	ffffffff + 0000007f	1(進位)	0	0000007e
6	55555555 | aaaaaaaa	0	0	ffffffff
7	00000001 << 2	0	0	00000004
8	80000000 << 1	1(進位)	0	00000000
9	00000020 >> 4	0	0	00000002

(c) tb_CompALU



該 Test 把(a)和(b)統整成了 CompALU
送入 RF Address 得到 Data 後，在透過 ALU 來計算不同的運算式得到的結果
該 Test 也如第二部分(b)所示，送入了 5 組不同的 Test Pattern，分別測試不同的運算式。

4. Conclusion and insight on this homework :

對於本來就是數位 IC 組的我來說，在處理這份作業的過程中幾乎沒有遇到什麼問題，畢竟是我常常撰寫的 Verilog 和使用的 Modelsim，反而是在書面上的報告遇到了一點瓶頸，思考著要怎麼呈現我的 Test Pattern 於報告才是最好的，後來就決定自己用 Python 製作了一個表格，方便閱讀。

雖然本次作業對我難度不高，不過在第一次模擬的助教課上，我發現身邊有些同學對於 Modelsim 怎麼操作，TestBench 怎麼是什麼，甚至連 Verilog 的撰寫都快沒甚麼印象，看著助教都快應付不來大家的問題，有餘力的我當然也馬上幫助同學處理了許多問題，後來發現我還滿喜歡這個過程的，不過很可惜許多相關的課程都需要碩士才能當助教，沒有辦法一圓我的助教夢。

總之，在這次的作業中，我更加熟悉了關於 ALU 的運算、Register File 的架構，也和很多位同學互相交流許多相關知識，很期待未來的三個 Project，希望可以成功設計出一顆屬於自己的 CPU 架構 ~