國 立 台 灣 科 技 大 學

計 算 機 組 織

指 導 教 授：陳雅淑 教 授

計算機組織作業一報告

班 級　：　四電機三甲
學 生　：　呂佳祐
學 號　：　B11107055

A. 32-bits Read Only Register File

1. 主程式

```v
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];
    assign Src_1 = R[Rs_addr];
    assign Src_2 = R[Rt_addr];
endmodule
```

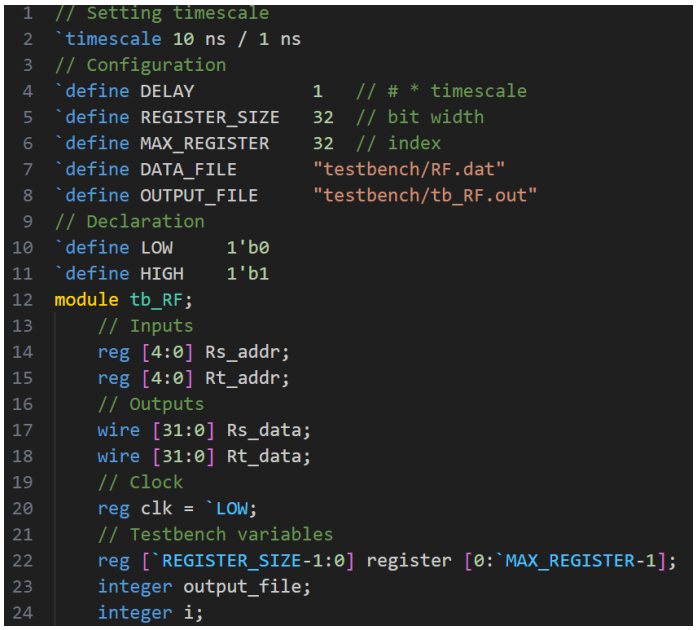
RF.v 程式碼

RF 唯讀暫存器寬度大小為 32bit、深度為 32bit。將輸入的 Rs_addr、Rt_addr 作為索引，透過陣列表示法，讀取對應暫存器內的值。因為 Register File 內有 32(2^5)個暫存器，所以 Address 大小宣告為 5 個 bit。而各暫存器內大小為 32bit，所以 Src_1、Src_2 大小宣告為 32 個 bit。

2. Testbench and Simulation

```
723D_1A01
4E19_B210
D476_4746
4472_EE1F
5002_4638
8371_E367
6DEC_8FB2
AACC_9602
7A4F_D3F4
49A3_C944
4E4B_5007
EF8E_3656
6171_70A5
E8CB_FA87
162F_31FF
B487_3F9A
A4F8_0A70
E649_F2DB
8D4F_C5FF
EC46_E907
45B3_C03E
F900_9998
EA24_FDD1
AB65_B051
E95F_ECD8
B980_5B12
3732_D400
E0E2_2572
4079_917D
BE28_A15F
FF64_D472
E856_F102
```

RF.dat 儲存 Register File 內各暫存器內資料



```
// Setting timescale
`timescale 10 ns / 1 ns
// Configuration
`define DELAY           1   // # * timescale
`define REGISTER_SIZE   32  // bit width
`define MAX_REGISTER    32  // index
`define DATA_FILE       "testbench/RF.dat"
`define OUTPUT_FILE     "testbench/tb_RF.out"
// Declaration
`define LOW     1'b0
`define HIGH    1'b1
module tb_RF;
    // Inputs
    reg [4:0] Rs_addr;
    reg [4:0] Rt_addr;
    // Outputs
    wire [31:0] Rs_data;
    wire [31:0] Rt_data;
    // Clock
    reg clk = `LOW;
    // Testbench variables
    reg [`REGISTER_SIZE-1:0] register [0:`MAX_REGISTER-1];
    integer output_file;
    integer i;
```

tb_RF.v 初始化設定

設定時間單位 10ns、時間精度 1ns。

```
    // Instantiate the Unit Under Test (UUT)
    RF UUT(
        // Inputs
        .Rs_addr(Rs_addr),
        .Rt_addr(Rt_addr),
        // Outputs
        .Src_1(Rs_data),
        .Src_2(Rt_data)
    );
    initial
    begin : Preprocess
        // Initialize inputs
        Rs_addr = 32'b0;
        Rt_addr = 32'b0;

        // Initialize testbench files
        $readmemh(`DATA_FILE, register);
        output_file = $fopen(`OUTPUT_FILE);

        // Initialize internal register
        for (i = 0; i < `MAX_REGISTER; i = i + 1)
        begin
            UUT.R[i] = register[i];
        end

        #`DELAY;    // Wait for global reset to finish
    end
```

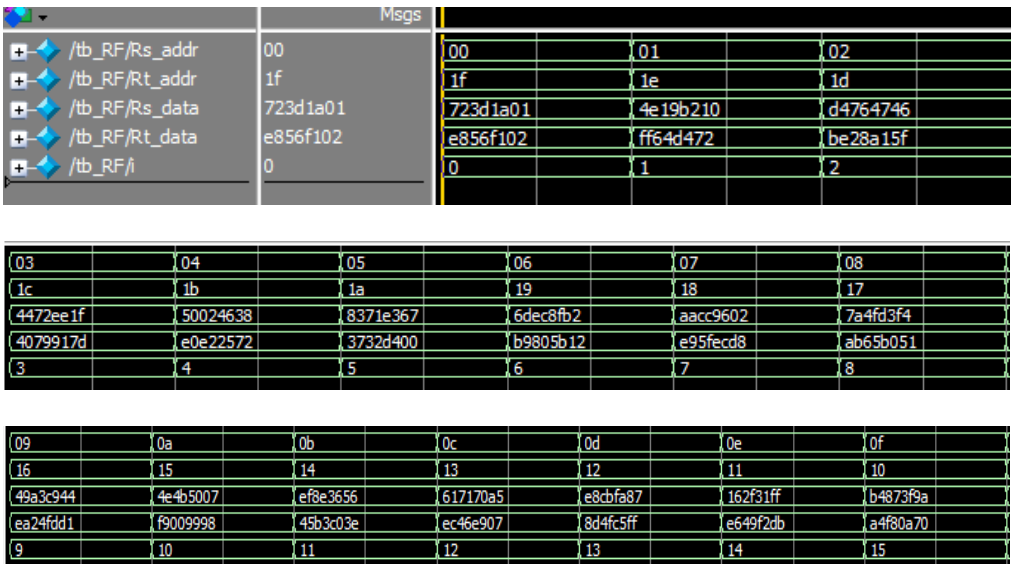
tb_RF.v 連接 RF.v module 與預處理

與主程式 module 連接。將 RF.dat 內資料讀取並存入 register 變數內，再將資料轉移至 Register File 裡，之後從 Register File 讀到的就是 RF.dat 內的資料。

```
52      always
53      begin : ClockGenerator
54          #`DELAY;
55          clk <= ~clk;
56      end
57      always
58      begin : StimuliProcess
59          // Start testing
60          for (i = 0; i < `MAX_REGISTER; i = i + 1)
61          begin
62              Rs_addr = i[`REGISTER_SIZE-1:0];
63              Rt_addr = `REGISTER_SIZE'd`MAX_REGISTER-1 - i[`REGISTER_SIZE-1:0];
64              @(clk); // Wait clock
65              $display("Rs_addr:%h, Rt_addr:%h", Rs_addr, Rt_addr);
66              $display("Src_1:%h, Src_2:%h", Rs_data, Rt_data);
67              $fdisplay(output_file, "%t,%h,%h", $time, Rs_data, Rt_data);
68          end
69
70          // Close output file for safety
71          $fclose(output_file);
72
73          // Stop the simulation
74          $stop();
75      end
76  endmodule
```

tb_RF.v 讀取暫存器資料

設定 clock 時序，週期為兩個`DELAY 的時間(20ns)。Source Register 是從頭開始讀取，而 Target Register 從最後一個暫存器開始讀。變數 i 資料型態為 integer，大小 32 個 bit；而 Address 大小為 5 個 bit，所以要用陣列表示法限定 i 的前 5 個 bit 賦值到 Rs_addr/Rt_addr。最後將讀取結果輸出到終端並記錄到 tb_RF.out。



tb_RF.v 模擬

模擬後總共會有 32 次讀取的資料，因為有兩個暫存器，一個從頭開始讀，一個從尾，所以這邊截圖到第 16 次讀取的資料就可以把所以暫存器的值都讀完。對比一下各位址的值，可以發現是與 RF.dat 吻合的。

B. 32-bits Arithmetic Logic Unit

1. 主程式

```
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      // Inputs
9      input  [31:0] Src_1,
10     input  [31:0] Src_2,
11     input  [4:0]  Shamt,
12     input  [5:0]  Funct,
13     // Outputs
14     output reg [31:0] ALU_result,
15     output reg        Zero,
16     output reg        Carry
17 );
```

ALU.v 初始化設定

定義各 Function Code 對應的功能常數(ADDU、SUBU、…)，宣告此 module 要用到的 input 及 output。

```
18     always @(*) begin
19         {Carry, Zero, ALU_result} = 0;
20         case (Funct)
21             `ADDU: {Carry, ALU_result} = Src_1 + Src_2;
22             `SUBU: {Carry, ALU_result} = Src_1 - Src_2;
23             `OR:   ALU_result = {1'b0, Src_1 | Src_2};
24             `SRL:  ALU_result = Src_1 >> Shamt;
25             `SLL:  ALU_result = Src_1 << Shamt;
26             default: {Carry, ALU_result, Zero} = 0;
27         endcase
28         Zero = (ALU_result == 0) ? 1 : 0;
29     end
30
31 endmodule
```

ALU.v 主要功能

定義此 ALU 為由其中一個輸入變化而改變輸出的組合邏輯電路。

利用 Switch-Case 判斷 Function Code 對應的功能做運算，最後再用三元運算子判斷 ALU_result 是否全為 0；若是，則 Zero 旗標為 1。

2. Testbench and Simulation

```
00011000000110000000000000000011_00010000000100000010000110100001_00001_100100
00000000000000000000000000000000_00000000000000000000000000000000_00100_100100
00000000000011100000000000100000_11111111111111111111111111111111_00101_100100
00010000001000000000000000000011_00000000000000000000000000000001_00001_100011
00000010000000010000000000000100_00000010000000010000000000000100_00100_100011
00000000011111000000000000100000_11111111111111111111111111111111_00101_100011
00000010000000000000000000000011_00000001100000000000000001100001_00001_100101
00000110000010000000011000010000_00011000000000000000111000000100_00100_100101
00010000000000000110000000100000_11100000000011100000000000001000_00101_100101
11111111111111111111111111111111_00000000000000011100000000000000_00001_000010
11111111111111111111111111111111_00000000000000000000011000000000_11111_000010
11111111111111111111111111111111_00000000000000000000000000000000_00101_000010
11111111111111111111111111111111_00000000000000000000000000110000_10000_000000
11111111111111111111111111111111_00000000110000000000000000000000_11111_000000
11111111111111111111111111111111_00000000000000000110000000000000_00101_000000
```

tb_ALU.in 所有測試向量

測試向量將 5 個功能都測試到，每個功能都有三組資料。

```
// Setting timescale
`timescale 10 ns / 1 ns
// Configuration
`define DELAY           1   // # * timescale
`define INPUT_FILE      "testbench/tb_ALU.in"
`define OUTPUT_FILE     "testbench/tb_ALU.out"
// Declaration
`define LOW     1'b0
`define HIGH    1'b1
module tb_ALU;
    // Inputs
    reg [31:0] Src_1;
    reg [31:0] Src_2;
    reg [4:0] Shamt;
    reg [5:0] Funct;
    // Outputs
    wire [31:0] ALU_result;
    wire Zero;
    wire Carry;
    // Clock
    reg clk = `LOW;
    // Testbench variables
    reg [74:0] read_data;
    integer input_file;
    integer output_file;
    integer i;
```

```
    // Instantiate the Unit Under Test (UUT)
    ALU UUT(
        // Inputs
        .Src_1(Src_1),
        .Src_2(Src_2),
        .Shamt(Shamt),
        .Funct(Funct),
        // Outputs
        .ALU_result(ALU_result),
        .Zero(Zero),
        .Carry(Carry)
    );
```

tb_ALU.v 初始化設定

設定時間單位 10ns、時間精度 1ns。宣告各暫存器、變數、連接線，並與 ALU Module 連接。

```
40      begin : Preprocess
41          // Initialize inputs
42          Src_1   = 32'b0;
43          Src_2   = 32'b0;
44          Shamt   = 5'b0;
45          Funct   = 6'b0;
46          // Initialize testbench files
47          input_file  = $fopen(`INPUT_FILE, "r");
48          output_file = $fopen(`OUTPUT_FILE);
49          #`DELAY;    // Wait for global reset to finish
50      end
51      always
52      begin : ClockGenerator
53          #`DELAY;
54          clk <= ~clk;
55      end
```

tb_ALU.v 預處理

初始化各變數、讀取 tb_ALU.in 內測試向量、設定 clock。

```
56      always
57      begin : StimuliProcess
58          // Start testing
59          while (!$feof(input_file))
60          begin
61              $fscanf(input_file, "%b\n", read_data);
62              @(posedge clk); // Wait clock
63              {Src_1, Src_2, Shamt, Funct} = read_data;
64              @(negedge clk); // Wait clock
65              $display("Src_1:%b, Src_2:%b, Shamt:%b, Funct:%b", Src_1, Src_2, Shamt, Funct);
66              $display("ALU_result:%d, Z:%b, C:%b", ALU_result, Zero, Carry);
67              $fdisplay(output_file, "%t,%b,%b,%b", $time, ALU_result, Zero, Carry);
68          end
69          // Close output file for safety
70          $fclose(output_file);
71          // Stop the simulation
72          $stop();
73      end
74  endmodule
```

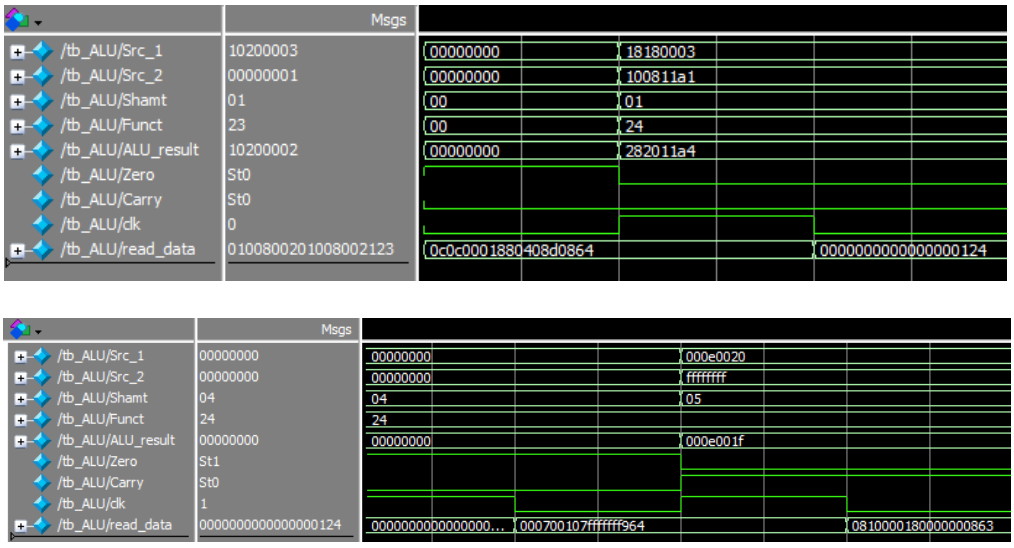
tb_ALU.v 測試

將各行測試向量賦值進 read_data 變數，並利用連結運算子將 read_data 拆分進 Src_1、Src_2、Shamt、Funct。最後輸出結果至終端與 tb_ALU.out。

```
56      always
57      begin : StimuliProcess
58          // Start testing
59          while (!$feof(input_file))
60          begin
61              $fscanf(input_file, "%b\n", read_data);
62              @(posedge clk); // Wait clock
63              {Src_1, Src_2, Shamt, Funct} = read_data;
64              @(negedge clk); // Wait clock
65              $display("Src_1:%b, Src_2:%b, Shamt:%b, Funct:%b", Src_1, Src_2, Shamt, Funct);
66              $display("ALU_result:%d, Z:%b, C:%b", ALU_result, Zero, Carry);
67              $fdisplay(output_file, "%t,%b,%b,%b", $time, ALU_result, Zero, Carry);
68          end
69          // Close output file for safety
70          $fclose(output_file);
71          // Stop the simulation
72          $stop();
73      end
74  endmodule
```

tb_ALU.v 測試

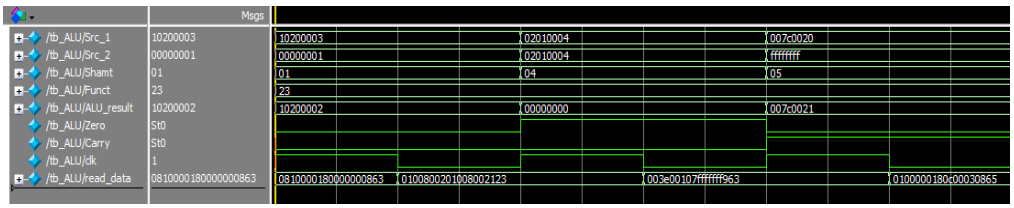
I. 加法測試



tb_ALU.v 模擬加法部分

三組資料測試後可發現加法後的結果正確、Zero 與 Carry 旗標正常運作、Shamt 不影響加法功能。由此可判斷加法功能正常執行。

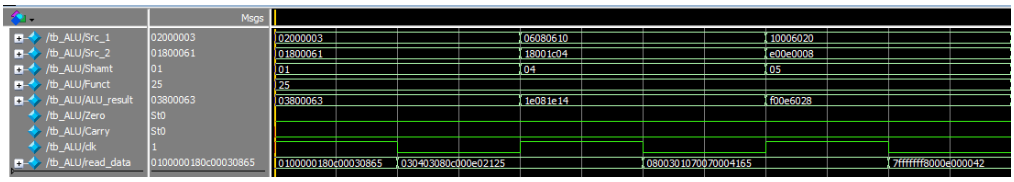
II. 減法測試



tb_ALU.v 模擬減法部分

三組資料測試發現減法結果正確、Zero 與 Carry(借位)旗標正常運作、Shamt 不影響減法功能。由此可判斷減法功能正常執行。

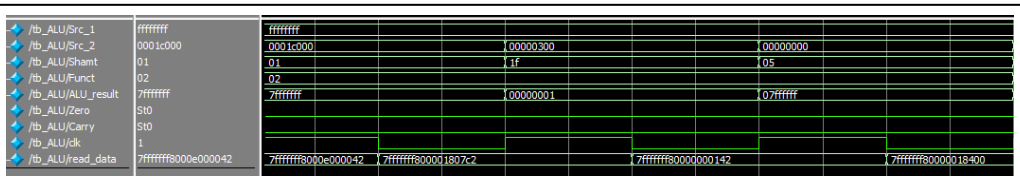
III. OR 測試



tb_ALU.v 模擬 OR 部分

三組資料測試發現或閘結果正確、Zero 與 Carry 正常運作、Shamt 不影響或閘功能。由此可判斷或閘功能正常執行。

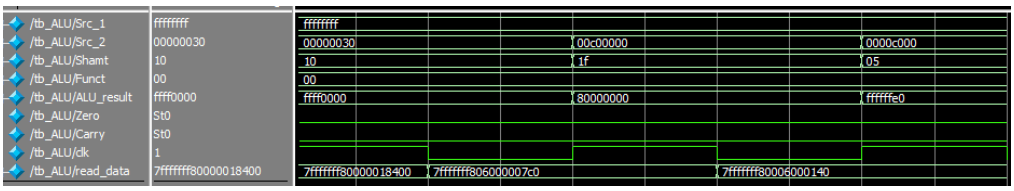
IV. 右移測試



tb_ALU.v 模擬右移部分

三組資料測試發現右移正常、Zero 與 Carry 正常運作、右移位元數確實依照 Shamt 決定。由此可判斷右移功能正常執行。

V. 左移測試



tb_ALU.v 模擬左移部分

三組資料測試發現左移正常、Zero 與 Carry 正常運作、左移位元數確實依照 Shamt 決定。由此可判斷左移功能正常執行。

C. 32-bits Complete ALU

1. 主程式

```
module CompALU(
    // Inputs
    input  [31:0] Instruction,
    // Outputs
    output [31:0] CompALU_out,
    output        CompALU_zero,
    output        CompALU_carry
);
wire [31:0] Inner_Src_1;
wire [31:0] Inner_Src_2;
RF Register_File(
    // Inputs
    .Rs_addr(Instruction[25:21]),
    .Rt_addr(Instruction[20:16]),
    // Outputs
    .Src_1(Inner_Src_1),
    .Src_2(Inner_Src_2)
);
ALU Arithmetic_Logical_Unit(
    // Inputs
    .Src_1(Inner_Src_1),
    .Src_2(Inner_Src_2),
    .Shamt(Instruction[10:6]),
    .Funct(Instruction[5:0]),
    // Outputs
    .ALU_result(CompALU_out),
    .Zero(CompALU_zero),
    .Carry(CompALU_carry)
);
endmodule
```

CompALU.v 程式碼

這程式把 Register File 與 ALU 結合起來，輸入 32-bits 的 Instruction，利用陣列表示法將 Instruction 分離出 Source Register、Target Register、Shamt、Function Code，並各自傳入 RF 與 ALU module。就像是在 CompALU 這黑盒子內將 RF 與 ALU 連接起來。

2. Testbench and Simulation

```
000000_10001_00010_00000_00000_100100
000000_01011_00100_00000_00000_100100
000000_00101_00110_00000_00000_100100
000000_00111_00010_00000_00000_100011
000000_10011_00100_00000_00000_100011
000000_00001_01000_00000_00000_100011
000000_01001_00100_00000_00000_100101
000000_10101_00110_00000_00000_100101
000000_00111_01000_00000_00000_100101
000000_11001_00000_00000_10101_000010
000000_00000_00000_00000_11111_000010
000000_10111_00000_00000_00100_000010
000000_11101_00000_00000_00000_000000
000000_11111_00000_00000_11011_000000
000000_00100_00000_00000_01100_000000
```

tb_CompALU.in 測試向量

每個功能都有三組測試資料，總共十五組測試向量。

```
// Setting timescale
`timescale 10 ns / 1 ns
// Declarations
`define DELAY           1   // # * timescale
`define REGISTER_SIZE   32  // bit width
`define MAX_REGISTER    32  // index
`define DATA_FILE       "testbench/RF.dat"
`define INPUT_FILE      "testbench/tb_CompALU.in"
`define OUTPUT_FILE     "testbench/tb_CompALU.out"
// Declaration
`define LOW 1'b0
`define HIGH    1'b1
module tb_CompALU;
    // Inputs
    reg [31:0] Instruction;
    // Outputs
    wire [31:0] CompALU_data;
    wire CompALU_zero;
    wire CompALU_carry;
    // Clock
    reg clk = `LOW;
    // Testbench variables
    reg [`REGISTER_SIZE-1:0] register [0:`MAX_REGISTER-1];
    reg [31:0] read_data;
    integer input_file;
    integer output_file;
    integer i;
    // Instantiate the Unit Under Test (UUT)
    CompALU UUT(
        // Inputs
        .Instruction(Instruction),
        // Outputs
        .CompALU_data(CompALU_out),
        .CompALU_zero(CompALU_zero),
        .CompALU_carry(CompALU_carry)
    );
```

tb_CompALU.v 初始化設定

設定時間單位 10ns、時間精度 1ns。宣告各變數並連接至 CompALU module。

```
37      initial
38      begin : Preprocess
39          // Initialize inputs
40          // Format: OpCode_Src1addr_Src2addr_RESERVED_shamt_funct
41          Instruction = 32'b000000_00000_00000_00000_00000_000000;
42          // Initialize testbench files
43          $readmemh(`DATA_FILE, register);
44          input_file  = $fopen(`INPUT_FILE, "r");
45          output_file = $fopen(`OUTPUT_FILE);
46          // Initialize internal register
47          for (i = 0; i < `MAX_REGISTER; i = i + 1)
48          begin
49              UUT.Register_File.R[i] = register[i];
50          end
51          #`DELAY;    // Wait for global reset to finish
52      end
53      always
54      begin : ClockGenerator
55          #`DELAY;
56          clk <= ~clk;
57      end
```

tb_CompALU.v 預處理

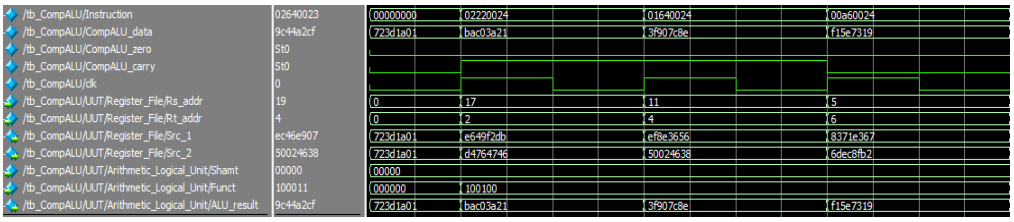
將 RF.dat 記憶體資料一一匯入 Redister File 個暫存器內，並打開 tb_CompALU.in，將檔案位址傳進 input_file 中，方便之後使用。ClockGenerator 產生週期為 20ns 的 clock。

```
58      always
59      begin : StimuliProcess
60          // Start testing
61          while (!$feof(input_file))
62          begin
63              $fscanf(input_file, "%b\n", read_data);
64              @(posedge clk); // Wait clock
65              Instruction = read_data;
66              @(negedge clk); // Wait clock
67              $display("Instruction:%b", read_data);
68              $display("CompALU_data:%d, Z:%b, C:%b", CompALU_data, CompALU_zero, CompALU_carry);
69              $fdisplay(output_file, "%t,%b,%b,%b", $time, CompALU_data, CompALU_zero, CompALU_carry);
70          end
71          // Close output file for safety
72          $fclose(output_file);
73          // Stop the simulation
74          $stop();
75      end
76  endmodule
```

tb_CompALU.v 開始傳入資料

將 tb_CompALU.in 內的測試向量，每隔一個 clock 週期，在上緣觸發時傳入 CompALU 中。最後將結果輸出至終端與 tb_CompALU.out 檔案裡。

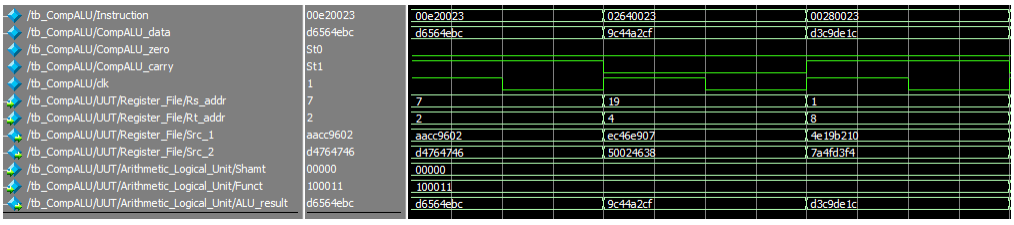
I. 加法測試



tb_CompALU.v 模擬加法測試

根據 address 從 RF 取的值與 RF.dat 對比後相符、加法結果正確、Zero 與 Carry 旗標正確。以上確認加法功能正常。

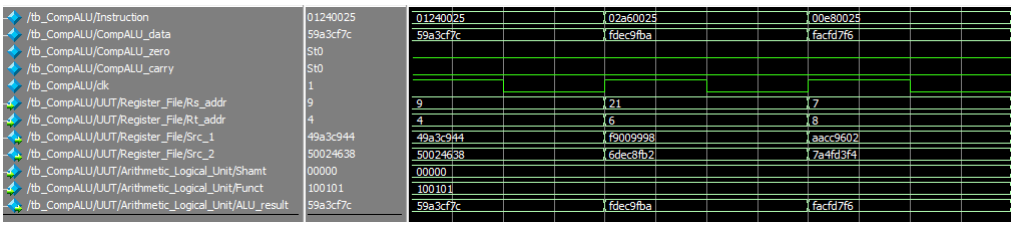
II. 減法測試



tb_CompALU.v 模擬減法測試

根據 address 從 RF 取的值與 RF.dat 對比後相符、減法結果正確、Zero 與 Carry 旗標正確。以上確認減法功能正常。

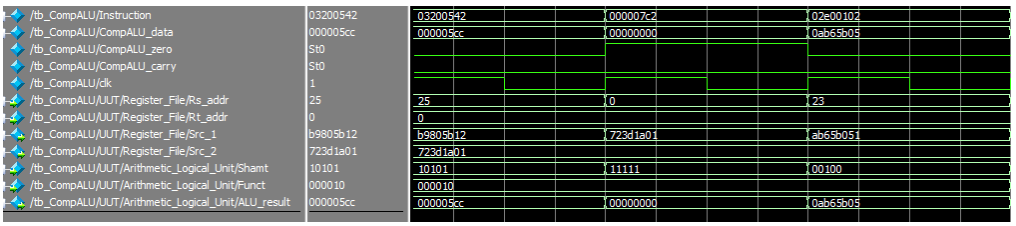
III. OR 測試



tb_CompALU.v 模擬 OR 測試

或閘運算結果正確、Zero 與 Carry 旗標正常反應。以上確認減法功能正常。

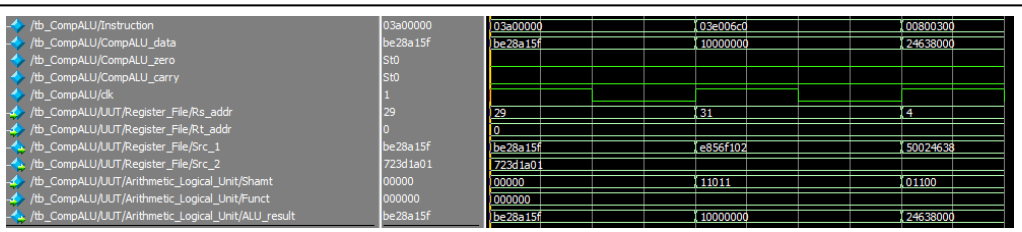
IV. 右移測試



tb_CompALU.v 模擬右移測試

右移數量有確實依照 shamt 決定，結果正確、旗標正常運作，以上確定右移功能正常。

V. 左移測試



tb_CompALU.v 模擬左移測試

左移數量與 shamt 相符，輸出結果正確、旗標正常運作，以上確定左移功能正常。

D. Conclusion and Insight

一個 CPU 裡包含許多功能，我們可以把這些功能先分開實作，而每個獨立的實作就像是個黑盒子。確定個別實作沒問題後，我們只要把各個黑盒子相對應的接腳連接上就行，最後再組成一個更大的黑盒子。如此這樣模組化、階層化的去看待電路，會使得開發及維護更輕鬆。

在設計測試向量時，除了要有正常的數據外，還需添加邊緣條件，確保電路的每個輸出端都正常運作、電路可以正常處理近乎全部的輸入。像是在此專案，為了要測試 Zero 旗標，可能就要故意將兩個指定暫存器的值設相同，或兩個 Address 都指向同個暫存器。或是測試在不需要用到 Shamt 時的場景，例如

加法時，給 Shamt 一個值，看會不會對加法功能有影響，若有，表示電路不如預期。