

HW1: 32-bits complete ALU

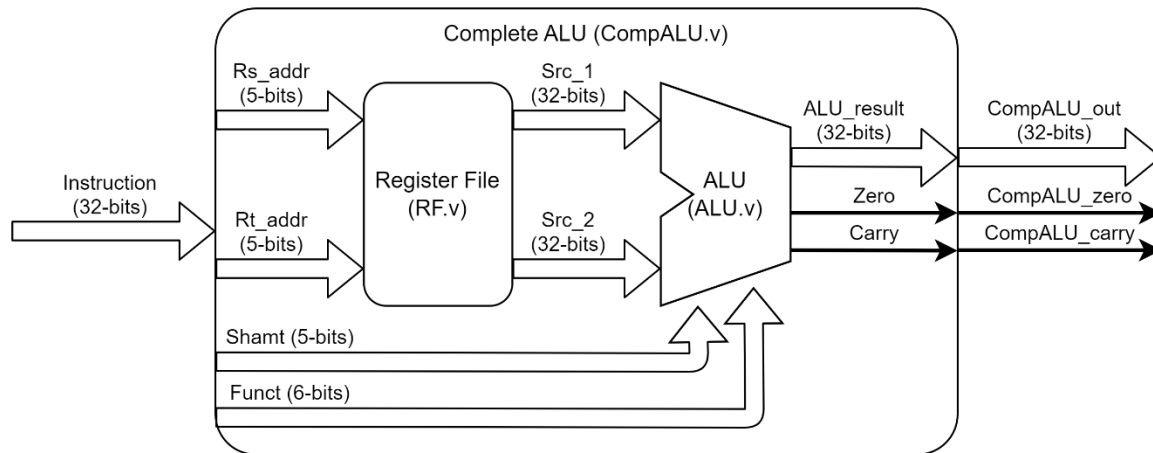


Figure 1: Path of Complete ALU

Implement a 32-bits Complete ALU module, which consists of a Register File module and an ALU module. The Register File module is composed of 32 read-only registers with a width of 32- bits. The ALU module is a 32-bits wide arithmetic logic unit.

a. 32-bits Read Only Register File

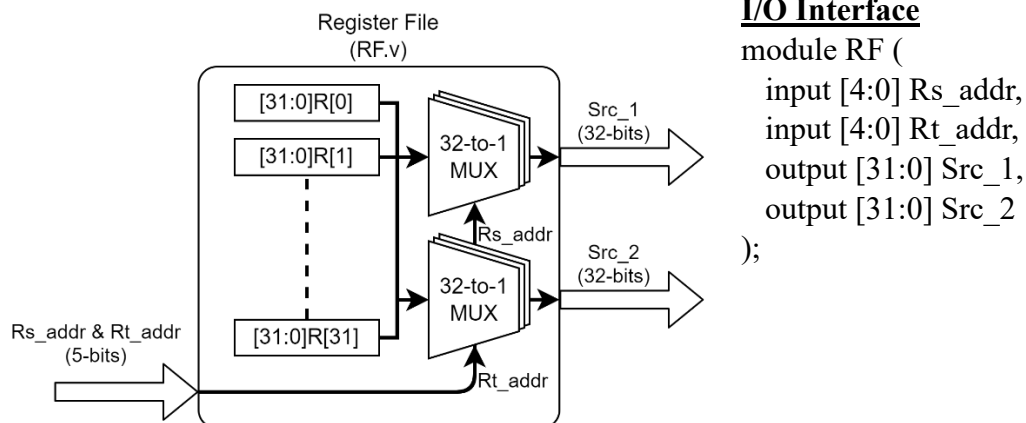
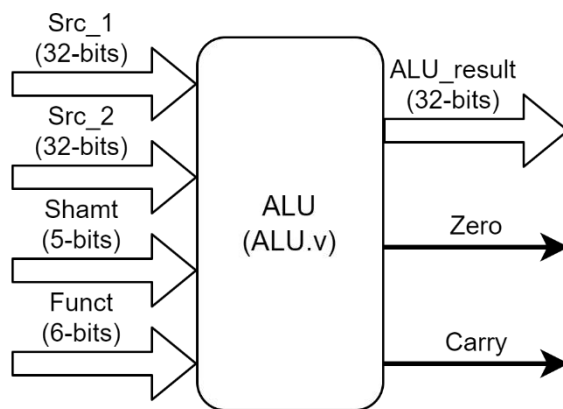


Figure 2: Path of Register File

Implement a register file with 32 registers having a width of 32- bits which are addressed by 5-bits. This module only performs the read operation with two outputs marked as "Rs_data" and "Rt_data". The initial value of each register is stored in the "RF.dat" file in the "testbench" folder in hexadecimal format, and the register file is initialized by "tb_RF" or "tb_CompALU" according to the file during simulation.

b. 32-bits Arithmetic Logic Unit



I/O Interface

```

module ALU (
    input [31:0] Src_1,
    input [31:0] Src_2,
    input [4:0] Shamt,
    input [5:0] Funct,
    output [31:0] ALU_result,
    output Zero,
    output Carry
);

```

Figure 3: I/O Interface of ALU

Implement an arithmetic logic unit, which performs related operations based on the register file and then outputs a result. The input of this module contains two 32-bits wide data bus from the register file, a 5-bits control bus, and a 6-bits control bus from the given instructions. The output is a 32-bits wide data bus to output the operation result, and two 1-bit flags, namely "Zero" and "Carry". The Zero flag is used to indicate whether the operation result is zero, and the Carry flag is used for the carry/borrow flag of the result.

This module needs to support the following operation, and the operation result is directly output to "ALU_result".

Instruction	Example	Meaning	Funct code
Add unsigned	Addu Result, Src_1, Src_2	Result = Src_1 + Src_2	100100
Subtract unsigned	Subu Result, Src_1, Src_2	Result = Src_1 - Src_2	100011
Or	Or Result, Src_1, Src_2	Result = Src_1 Src_2	100101
Shift right logical	Srl Result, Src_1, Shamt	Result = Src_1 >> Shamt	000010
Shift left logical	Sll Result, Src_1, Shamt	Result = Src_1 << Shamt	000000

Note: The value required for the Shift action is stored in "Shamt".

c. 32-bits Complete ALU

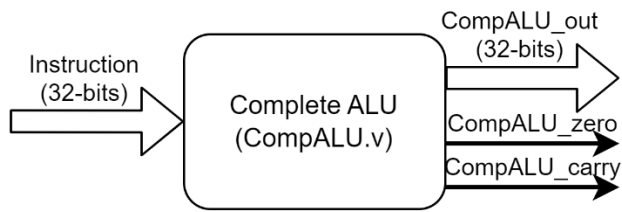


Figure 4: I/O Interface of Complete ALU

I/O Interface

```
module CompALU (
    input [31:0] Instruction,
    output [31:0] CompALU_out,
    output CompALU_zero,
    output CompALU_carry
);
```

Implement 32-bits complete ALU by combining the register file and the arithmetic logic unit. The input of the module is a 32-bits instruction, and the output of this module is a 32-bits result, Zero flag and Carry flag. The instruction format is as follows:

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

OP Code: Indicate the command to be executed, this homework is set as “000000”.

Source Register: Indicate the first register to be executed.

Target Register: Indicate the second register to be executed.

Destination Register: The register used to store the execution result. It can be set to zero in this homework.

Shamt: Indicate the number of "Shift" actions.

Funct: Indicate the command to be executed.

Examples:

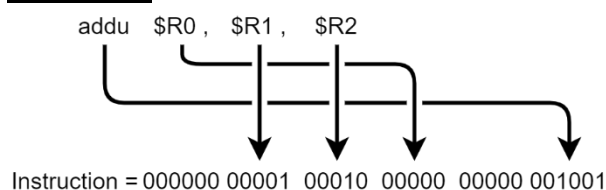


Figure 5: Example of Add Unsigned Instruction

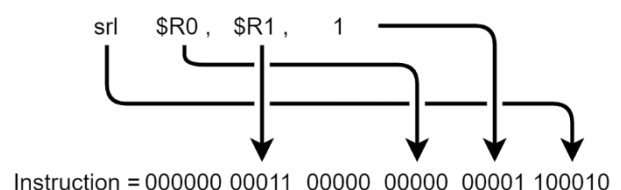


Figure 6: Example of Shift Right Logical Instruction

e. Submission

Report structure:

- a. Cover.
- b. Screenshots and descriptions of each module.
- c. Screenshots and descriptions of test commands for each module (.in file).
- d. Screenshots and explanations of the test results (hexadecimal waveforms) for each module.
- e. Conclusion and insight on this homework.

※ Convert the report to PDF and name it the student ID - "BYYYDDXXX.pdf".

Files required for HW1:

- BYYYDDXXX.zip
 - All module.v files (RF.v, ALU.v, CompALU.v)
 - Report (BYYYDDXXX.pdf)
- **Note: Please make sure all your program files and PDF files are directly in the zipped file, not all wrapped in one folder.**

Grading:

- a. (25 points) RF: Complete output of RF.dat is required.
- b. (25 points) ALU: 5 points for each instruction
- c. (25 points) CompALU: 5 points for each instruction.
- d. (25 points) Report.
- e. Follow naming rules and file formats.
- f. No plagiarism.

We will test your modules with another generated testbench.

Deadline: 2025-03-13 12:00 on moodle

Appendix: Testbench description

Ex: tb_CompALU.v

```
28 // Setting timescale
29 `timescale 10 ns / 1 ns
30
31 // Declarations
32 `define DELAY 1 // # * timescale
33 `define REGISTER_SIZE 32 // bit width
34 `define MAX_REGISTER 32 // index
35 `define DATA_FILE "testbench/RF.dat"
36 `define INPUT_FILE "testbench/tb_CompALU.in"
37 `define OUTPUT_FILE "testbench/tb_CompALU.out"
38
39 // Declaration
40 `define LOW 1'b0
41 `define HIGH 1'b1
42
43 module tb_CompALU;
44
45 // Inputs
46 reg [31:0] Instruction;
47
48 // Outputs
49 wire [31:0] CompALU_out;
50 wire CompALU_zero;
51 wire CompALU_carry;
52
53 // Clock
54 reg clk = `LOW;
55
56 // Testbench variables
57 reg [`REGISTER_SIZE-1:0] register [0:`MAX_REGISTER-1];
58 reg [31:0] read_data;
59 integer input_file;
60 integer output_file;
61 integer i;
62
63 // Instantiate the Unit Under Test (UUT)
64 CompALU UUT(
65 // Inputs
66 .Instruction(Instruction),
67 // Outputs
68 .CompALU_out(CompALU_out),
69 .CompALU_zero(CompALU_zero),
70 .CompALU_carry(CompALU_carry)
71 );
72
```

```

72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
initial
begin : Preprocess
    // Initialize inputs
    // Format: OpCode_Src1addr_Src2addr_RESERVED_shamt_func
    Instruction = 32'b000000_00000_00000_00000_00000_000000;

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

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

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

end

always
begin : ClockGenerator
    #`DELAY;
    clk <= ~clk;
end

always
begin : StimuliProcess
    // Start testing
    while (!$feof(input_file))
    begin
        $fscanf(input_file, "%b\n", read_data);
        @(posedge clk); // Wait clock
        Instruction = read_data;
        @(negedge clk); // Wait clock
        $display("Instruction:%b", read_data);
        $display("CompALU_data:%d, Z:%b, C:%b", CompALU_out, CompALU_zero, CompALU_carry);
        $display(output_file, "%t,%b,%b,%b", $time, CompALU_out, CompALU_zero, CompALU_carry);
    end

    // Close output file for safety
    $fclose(output_file);

    // Stop the simulation
    $stop();
end
endmodule

```

Line	Description
29	The magnitude of the simulation timeline.
93~97	Generate oscillating clocks.
105	After the positive edge of the clk signal occurs, the program can continue to be executed.
107	After the negative edge of the clk signal occurs, the program can continue to be executed.
117	Interrupts the simulation to end the simulation.