HW1: 32-bits complete ALU

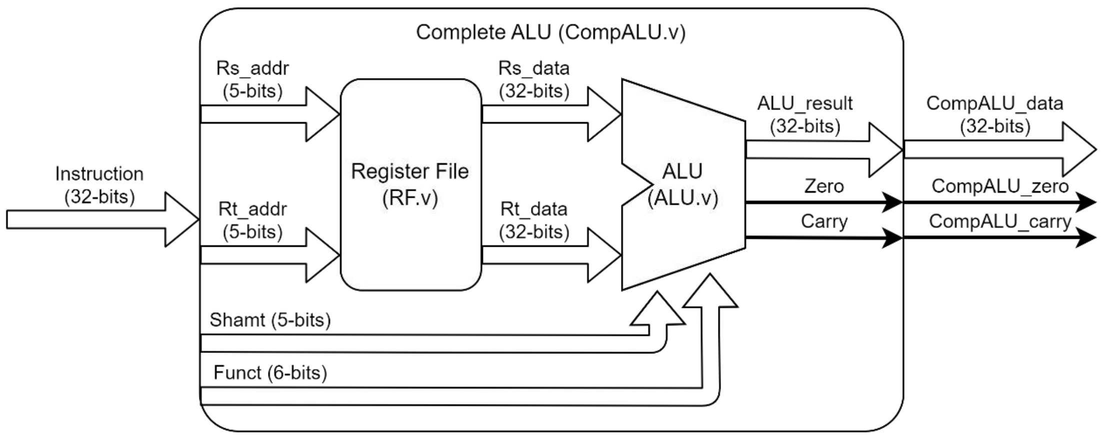


Figure 1: Path of Complete ALU

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

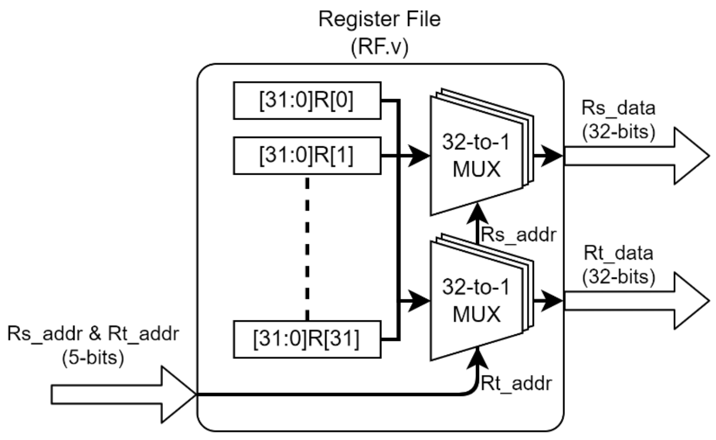


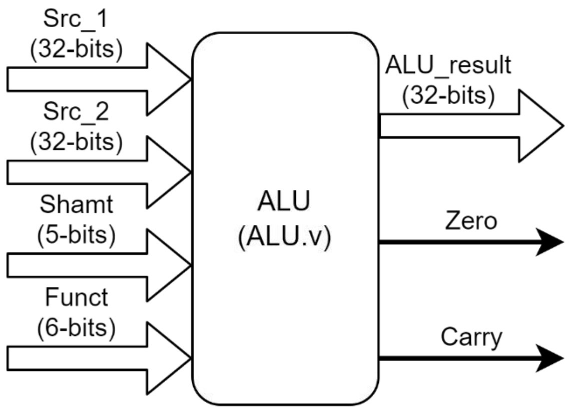
Figure 2: Path of Register File

I/O Interface

```
module RF (
  input [4:0] Rs_addr,
  input [4:0] Rt_addr,
  output [31:0] Rs_data,
  output [31:0] Rt_data
);
```

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] 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	001001
Subtract unsigned	Subu Result, Src_1, Src_2	Result = Src_1 − Src_2	001010
And	And Result, Src_1, Src_2	Result = Src_1 & Src_2	010001
Shift right logical	Srl Result, Src_1, Shamt	Result = Src_1 ≫ Shamt	100010

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

c. 32-bits Complete ALU

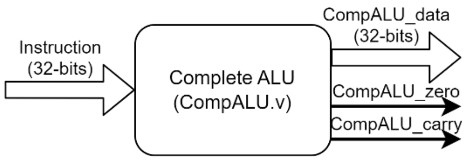


Figure 4: I/O Interface of Complete ALU

I/O Interface

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

Implement 32-bits complete ALU by combing 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:

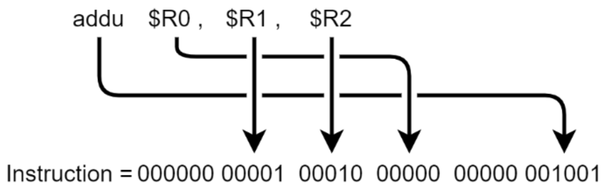


Figure 5: Example of Add Unsigned Instruction

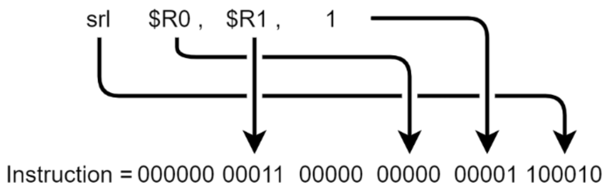
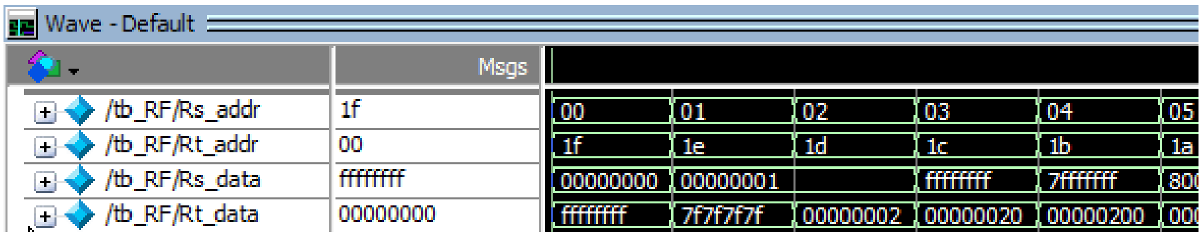


Figure 6: Example of Shift Right Logical Instruction

d. Functional Simulation

1. tb_RF

The testbench "tb_RF" initializes the register according to "testbench/RF.dat", and then sequentially sends the address to Rs_addr and Rt_addr. Its output waveform (in hexadecimal) is similar to the figure below. Please show the screenshot and explain the results in your report.

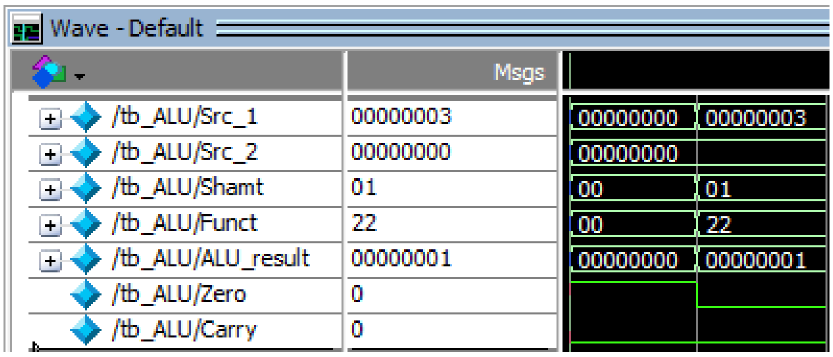


	Msgs						
/tb_RF/Rs_addr	1f	00	01	02	03	04	05
/tb_RF/Rt_addr	00	1f	1e	1d	1c	1b	1a
/tb_RF/Rs_data	ffffffff	00000000	00000001		ffffffff	7fffffff	800
/tb_RF/Rt_data	00000000	ffffffff	7f7f7f7f	00000002	00000020	00000200	000

Figure 7: Waveform of tb_RF

2. tb_ALU

The testbench "tb_ALU" initializes all inputs to zero, and then executes commands in "testbench/tb_ALU.in" one by one. Its output waveform (in hexadecimal) is similar to the figure below. Please show a screenshot and explain the results in your report.
“tb_ALU.in” only provides an example for the “srl” operation. You should add other commands and explain the results in the report.

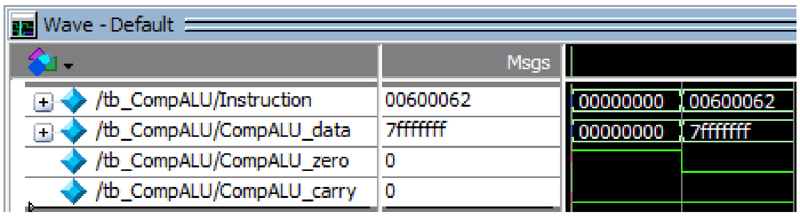


	Msgs		
/tb_ALU/Src_1	00000003	00000000	00000003
/tb_ALU/Src_2	00000000	00000000	
/tb_ALU/Shamt	01	00	01
/tb_ALU/Funct	22	00	22
/tb_ALU/ALU_result	00000001	00000000	00000001
/tb_ALU/Zero	0		
/tb_ALU/Carry	0		

Figure 8: Waveform of tb_ALU

3. tb_CompALU

The testbench "tb_CompALU" initializes all the input to zero and initialized the register file. Then, the testbench executes the commands in "testbench/tb_CompALU.in" one by one. Its output waveform (in hexadecimal) is similar to the figure below. Please show a screenshot and explain the results in your report. “tb_CompALU.in” only provides one example. You can add other test commands and describe them in the report.



	Msgs		
/tb_CompALU/Instruction	00600062	00000000	00600062
/tb_CompALU/CompALU_data	7fffffff	00000000	7fffffff
/tb_CompALU/CompALU_zero	0		
/tb_CompALU/CompALU_carry	0		

Figure 9: Waveform of tb_CompALU

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)
All test.in files	(tb_ALU.in, tb_CompALU.in)
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:

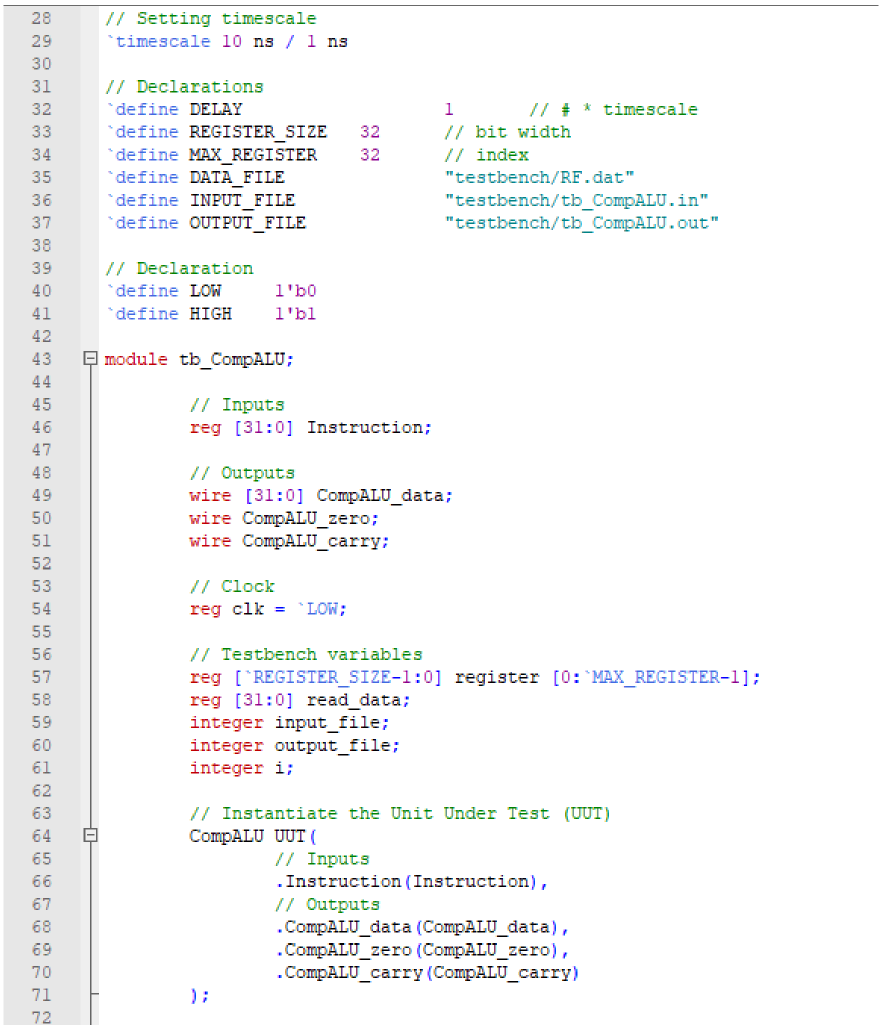
a. (20 points) RF: Complete output of RF.dat is required.
b. (20 points) ALU: 5 points for each instruction
c. (20 points) CompALU: 5 points for each instruction.
d. (40 points) Report.
e. Follow naming rules and file formats.
f. No plagiarism.

We will test your modules with another generated testbench.

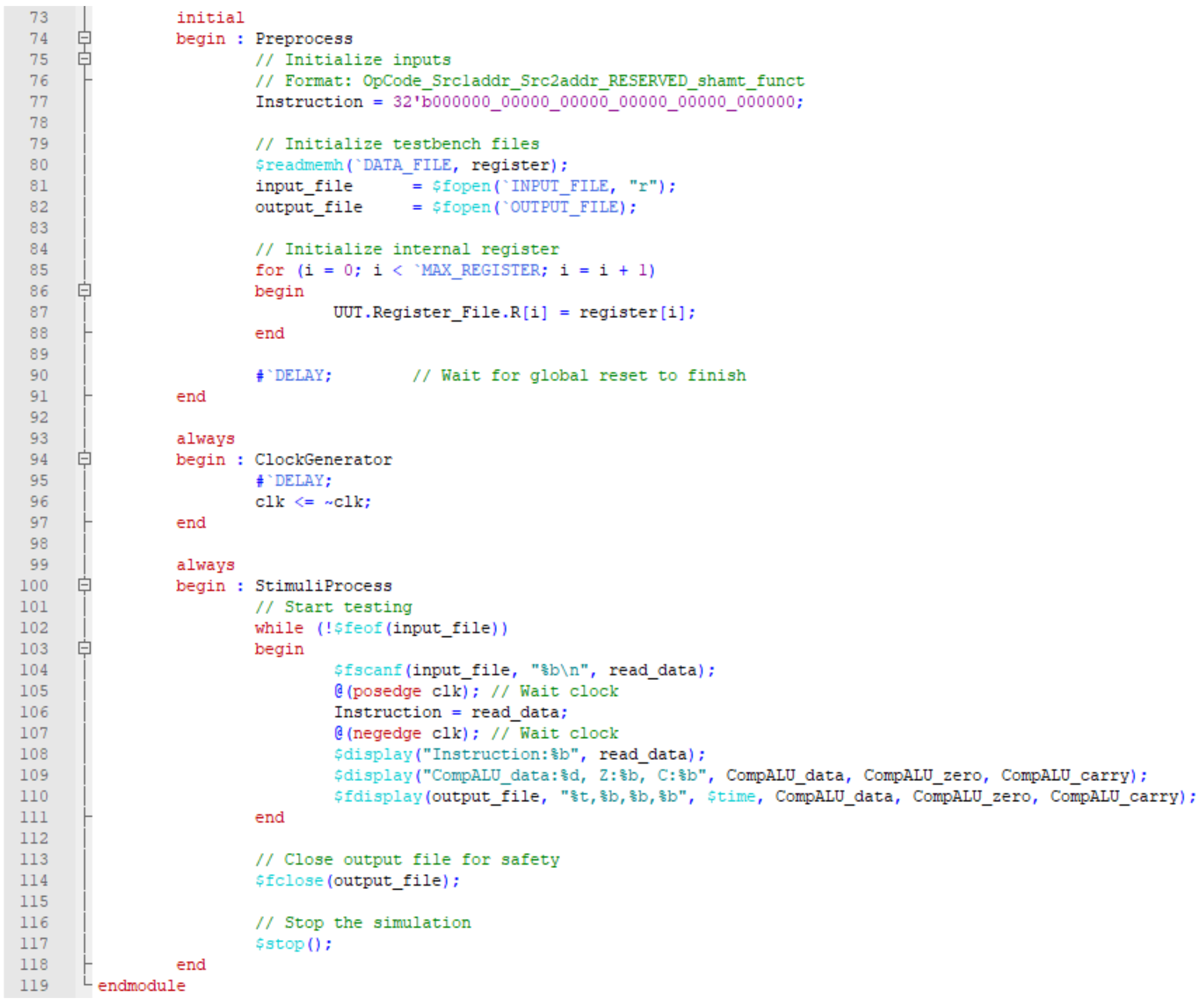
Deadline: 2024-03-20 13:00 on moodle

Appendix: Testbench description

Ex: tb_CompALU.v



```verilog
// 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_data),
                .CompALU_zero(CompALU_zero),
                .CompALU_carry(CompALU_carry)
        );

```



```
73          initial
74          begin : Preprocess
75                  // Initialize inputs
76                  // Format: OpCode_Src1addr_Src2addr_RESERVED_shamt_funct
77                  Instruction = 32'b000000_00000_00000_00000_00000_000000;
78
79                  // Initialize testbench files
80                  $readmemh(`DATA_FILE, register);
81                  input_file      = $fopen(`INPUT_FILE, "r");
82                  output_file     = $fopen(`OUTPUT_FILE);
83
84                  // Initialize internal register
85                  for (i = 0; i < `MAX_REGISTER; i = i + 1)
86                  begin
87                          UUT.Register_File.R[i] = register[i];
88                  end
89
90                  #`DELAY;        // Wait for global reset to finish
91          end
92
93          always
94          begin : ClockGenerator
95                  #`DELAY;
96                  clk <= ~clk;
97          end
98
99          always
100         begin : StimuliProcess
101                 // Start testing
102                 while (!$feof(input_file))
103                 begin
104                         $fscanf(input_file, "%b\n", read_data);
105                         @(posedge clk); // Wait clock
106                         Instruction = read_data;
107                         @(negedge clk); // Wait clock
108                         $display("Instruction:%b", read_data);
109                         $display("CompALU_data:%d, Z:%b, C:%b", CompALU_data, CompALU_zero, CompALU_carry);
110                         $fdisplay(output_file, "%t,%b,%b,%b", $time, CompALU_data, CompALU_zero, CompALU_carry);
111                 end
112
113                 // Close output file for safety
114                 $fclose(output_file);
115
116                 // Stop the simulation
117                 $stop();
118         end
119 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.