Lab 3A – Register File

We want to simulate the Register File shown below. This has a collection of 32 registers (R0 to R31). Hence the address is 5 bits. Registers are 64 bits wide and so data in and data out are 64 bits.

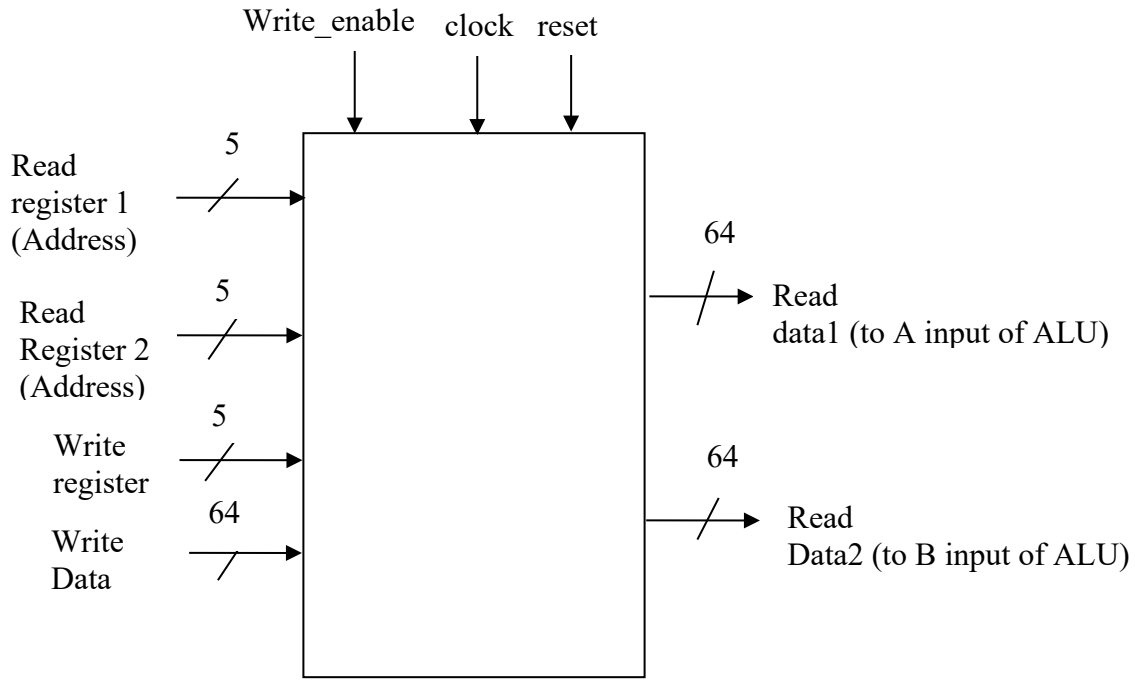

Figure 1. Register File

The Verilog code for the module is given below:

```
module regfile (clk, reset, wr_en, rd_addr_1, rd_addr_2,
                wr_addr, wr_data, rd_data_1, rd_data_2);

input clk, reset, wr_en;
input [4:0] rd_addr_1, rd_addr_2, wr_addr;
input [63:0] d_in;
output [63:0] reg d_out_1, d_out_2;
reg [63:0] RF[0:31];

// Add your code here

endmodule
```

Develop a Verilog Test Fixture. Our goal is to write into registers and read out register values. In the beginning in the test bench, we want to have two values 555… and aaa…. in the Register File in registers 5 and 10. In real life, registers come up with random values or zeros at boot up and our recourse to have values in registers is to LOAD them from nonvolatile Memory. That means we should have instructions in the Instruction Memory which when run, will load values from nonvolatile memory.
But in this lab, we do not have Memory module built yet. We also do not have Instruction Memory. Let us do a simple thing. Start by resetting register file: on reset, values of all registers become 0. Then WRITE to registers 5 and 10 in the beginning. That way, we will test that we can write to registers. Choose Register 5 and write to it 64'h5555555555555555 and then choose register 10 and write to it 64'haaaaaaaaaaaaaaaa.

Repeat this for other two pairs of registers with values you will leader use to test RF and ALU together. Remember writing can take place only on the positive edge of the clock and so clock must be zero initially and then become 1 for *each* write operation. Also the register address and register data to be written must be **set up prior** to the arrival of the clock edge. All changes must occur before the clock edge. Values must be stable before and after the clock edge (why?). So you may make changes to address and data at 60% of your clock cycle (does it need to be exactly this?). It is a good idea to *plan* and **draw a timing diagram** to indicate when some signal will go up and then come down with relation to others. The timing diagram will help a lot in writing the test module. Include your timing diagram in the report and show it to the instructor at the time of demo.

In the test bench, after writing, we can READ them and verify that they come out right. That way, we will test that we can read out register values. (Initially both data outputs will be all zeros and unknowns since we did not specify Read address (Read address may be 0 or x's and all registers may be 0 or x's initially).

Lab 3B – RF-ALU Combined

Now we want to **combine** RF with ALUwithControl as shown below. We will read a pair of register values and supply them to ALU and with corresponding ALU operation, we will check whether the ALU results are correct. In this part B, the results of ALU are just hanging out there, and are to be inspected by the testbench only. In Part C, we will *write* them *back* to the Register File.

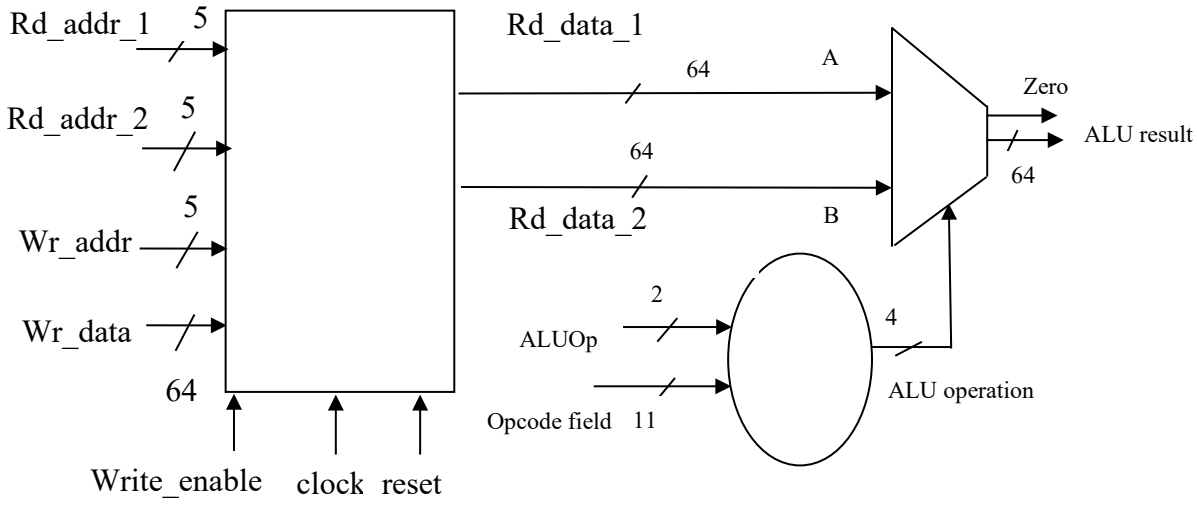

Figure 2. Register File feeding ALU

Plan and draw a timing diagram. Develop a Verilog Test Fixture to verify the operations.
Make sure to include the three intermediate signals i.e. the wires used to connect the modules in the waveform.

Lab 3C – Supporting R Type (ALU Instructions)

We want to extend Lab 3B to support **R Type instructions** or ALU instructions**.** R Type instructions are in the form ADD X1, X2, X3. This means:

Register X1 ←Register X2 OP Register X3

In Lab 3B, we obtained the results of some ALU operations, but they were not *written back to the Register File* as shown in the Figure 3. Create a module (name it RF_ALU_R_type) and instantiate the elements you implemented in 3B.You have two options how to implement this:

1) Enclose entire design of 3B into a module (name it RF_ALU). Insatiate RF_ALU module inside RF_ALU_R_type module, and create a wire that you will use to connect the ALU_Result output to Wr_data input
2) Copy all modules from 3B and create a wire that you will use to connect the ALU_Result output to Wr_data input

Create a separate testbench to test module RF_ALU_R_type (name it RF_ALU_R_type_tb)

Please note: you have two options for loading the data in the register file:

1) Initiate the data inside the array (in _tb.v or in the module itself .v)
2) Add a multiplexor at the Wr_data input that selects from ALU_result (on 0) and RF_write_data_in (on 1), and add the select signal that you will asset in the _tb.

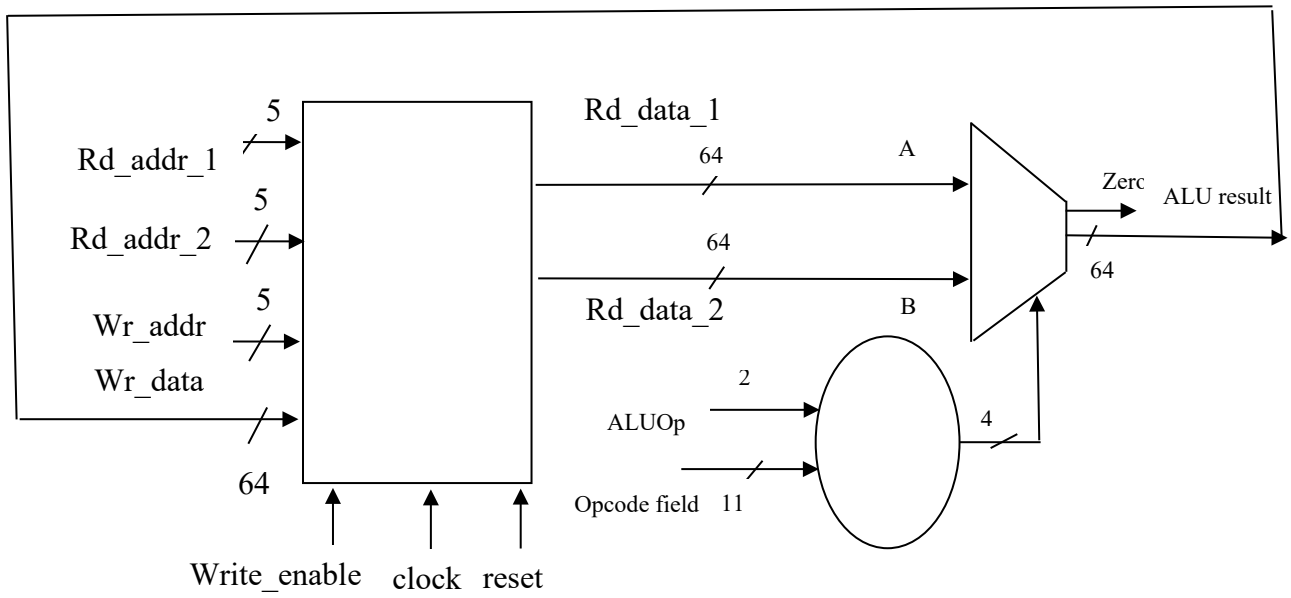

Figure 3. ALU reading and writing from Register File

The format for an R-type instruction in machine language is given in the Figure 4 with the corresponding bit-widths below each field. If ADD X1, X2, X3 is the instruction in assembly language will have Src reg 2 = X3, Src reg 1 = X2, Dest reg = X1 and Opcode = operation code for ADD, as per the Lab 2.

Opcode	Src reg 2	Shamt	Src reg 1	Dest reg
11	5	6	5	5

Figure 4. Operation format for R-type of instruction

Please, in your testbench, provide inputs that correspond to R-type of instruction – test sufficient number of instructions. If needed, design additional hardware to be able to load the data from the testbench or from the output of the ALU. Show the diagram of the design in your report. Make sure that you can check the intermediate results.

mc,s18 & jt, f18/f1/f20