Computer Architecture Lab 07

Integration of ALU and Register File

Introduction

Register files form the local memory of the processor, acting as a bridge between the external memory and the processing unit. They provide fast, temporary storage for operands and intermediate results, allowing the processor to access data much more quickly than it could from main memory.

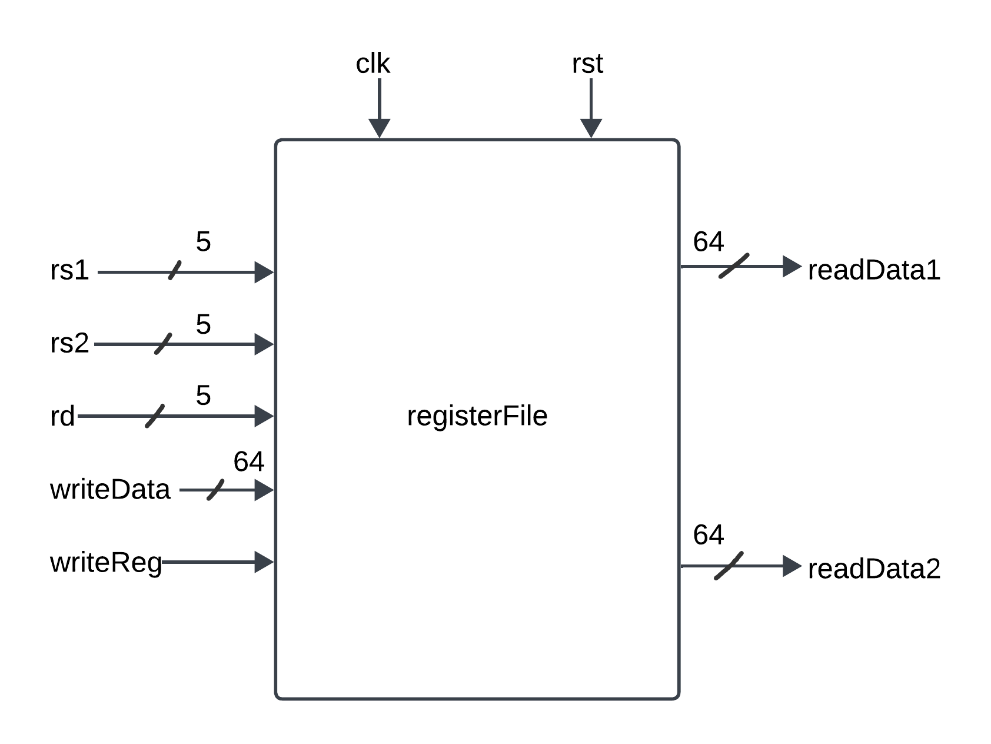
In typical architectures, the ALU performs computations using values stored in the register file. After each operation, results can be written back into registers for immediate reuse or stored in memory for long-term retention.

Understanding how register files and the ALU interact is crucial for appreciating how instructions execute efficiently in This lab will guide you through designing the register file and simulating its interaction with the arithmetic logic unit (ALU), enabling you to understand how they connect to form the computational backbone of a processor.

Objectives

Task 1: Forming the Register File

In this task, you must develop a module called *registerFile* with the top level diagram being as follows:



As an example, the following command creates an Array of type reg containing  
10 elements, each of which are 5 bits wide.

reg [4:0] Array [9:0]

where Array is the name of a variable. In your case, it should be Registers. As there are 32 registers, it will contain 32 elements and the data which a register takes up is 64-bit so register will be 64 bits wide defined as [63:0].

* Initialize the register file with random values.
* The operation of writing data into a Registers should always be done when there is a positive edge of clk and regWrite signal is asserted (or set i.e., HIGH).
* On reset, place a value 0 at both readData output ports.
* Reading a data from the register file should be made independent of clk signal. Reading should rather be sensitive to the change in inputs rs1, rs2, or reset or registers.

Provide code for the module and the testbench here, and attach waveform results:

|  |  |
| --- | --- |
| Code | Testbench |
|  |  |
| Waveform | |
|  | |

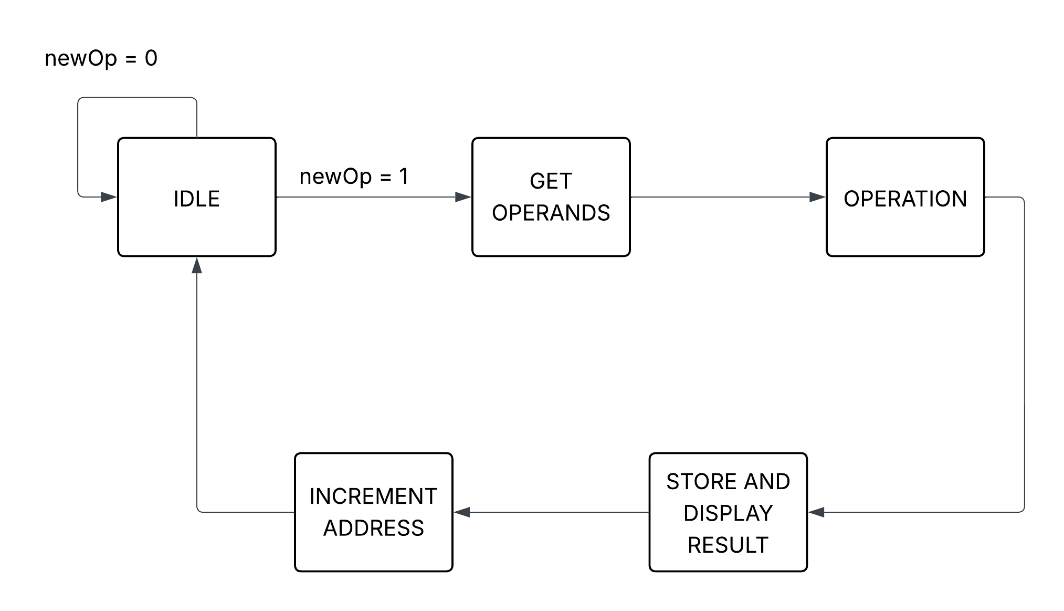
By the end of this task, you have a functioning register file and ALU from the previous lab.

Task 2: Designing the FSM

The design requirements are as follows:

* The design initializes with an idle state. After 10s, it begins the following process.
* The system cycles through the register file, reading two consecutive addresses at each cycle.
* The fetched data registers are added to each other if the address is even, and subtracted from each other if odd.
* The resulting data is stored back into the register file at the second consecutive address.
* The resulting data is displayed on the seven segment.
* Address is incremented to point to the next two register file locations, and resets to zero if it has reached the end of the register file.
* The design goes back into an idle state, and waits 10s before restarting the process.

Your FSM may thus look something like this:



Provide relevant module codes and top module schematic here:

|  |
| --- |
|  |