Art of Engineering: CE/CS Dept. Project
Processor Design Part 2: Register File

Introduction
In this assignment, we will work on another of the building blocks for our processor, this time using clocked state elements (specifically, registers) as well as combinational logic. This is the **register file**, and it is the part of the computer that temporarily holds the data it is currently working with -- like a workbench or table. Later in the project, we will connect the register file and the ALU to form the computer's **datapath**.

What should the register file be able to do?
The register file should include 16 registers, each 8 bits wide.

We should be able to write new data to any of these registers; which one should be determined by a 4-bit control signal called "rd" (destination register). Recall from the asynchronous module why we need 4 bits. This type of control signal is called an **address**. New data should be written on the rising edge of the clock. Only the register specified by rd should be written on a given clock cycle; all other registers should hold their current values.

We should also be able to read two pieces of data from the file at any given time; again, which registers we are accessing should be determined by 4-bit addresses called "rs1" and "rs2" (source registers 1 and 2). Thus, in total we will have 3 4-bit addresses.

The register file should also have an option to, for a given clock cycle, not overwrite any of the old data - in other words, we should be able to enable/disable the write option.

Finally, the register file should have an asynchronous reset input: when this input becomes 1, all registers are immediately cleared (regardless of the clock value - hence asynchronous). Each register has one of these inputs, so all you need to do is connect them together and add a pin.

What components to use
You will find memory elements, including flip-flops and registers, in Logisim's Memory library. For this assignment, the only one you need is called "Register". You are also expected to use logic gates from the Gates library, and multiplexers/demultiplexers from the Plexers library. In addition, the Wiring library contains some useful components, such as the Splitter, which allows you to separate the wires from a bus, or combine wires into a bus.

Hexadecimal Data

You will notice that the Register component shows its current state in hexadecimal. In everyday life, we use decimal (base 10), while computers have two physical logic levels, as we discussed in the first asynchronous module, so they use binary (base 2). However, writing a whole bunch of 0s and 1s gets tedious, so computer engineers also work with hexadecimal (base 16). The idea of hexadecimal is to combine 4 binary digits into 1 hexadecimal digit:

Binary (base 2)	Hexadecimal (base 16)	Decimal (base 10)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Since we are using 8-bit data, each register will show 2 hexadecimal digits, e.g. “B4” (also written as “b4”; the upper/lower case does not matter). A piece of 8-bit data is called a **byte**. Each hexadecimal digit, representing 4 bits, is called a **nibble**.

How to test your design

The Wiring library contains the clock, and the Input/Output library has a button you can use to test the reset functionality. **The clock and reset button should not be inside your register file**. Instead, you should build another circuit, called the **testbench**. Then add 1 instance of your register file to this testbench, add a clock and a reset button, and connect their outputs to the corresponding inputs of your register file. You'll also need to add pins to your testbench for the register file's inputs and outputs, connect them, and use the Poke tool to test your design.

Since we now have a clocked system, we need some way to "tick" the clock. One way to easily toggle the clock from 0 to 1 or 1 to 0 is to manually click it using the Poke tool; this will probably be the most useful technique for this assignment, since it will allow you to see the register file operate in slow-motion. Try writing some data to a few different registers, and see if you can read the data back out. Also make sure to try the case where we *do not* write any data.

What to turn in

Please submit to courseworks a **1-page** report with

1. A screenshot showing your top-level register file schematic
2. A screenshot of your testbench, with one of your verification tests, demonstrating correct functionality.
3. A short (3-4 sentences) explanation summarizing how you assembled the register file, and how you assessed whether your design's functionality is correct.

Please adhere to the page limit. Remember that this will be assessed on a 1 / 0 basis. Even if your design is not working or not complete, submit something to let us know how it is going!

A Note on the Textbook

Flip-flops, registers, and register files are discussed in Appendix A, section A.8. You may ignore the Verilog examples, since we will not be using Verilog in this project.