RAT Assignment #3
Memory Fundamentals

Learning Objectives:

- ✓ To implement the register file in the RAT MCU architecture.
- ✓ To learn how a register file operates and how it will work when integrated with the RAT CPU.
- ✓ To become familiar with the operation of the scratch memory, which the RAT CPU uses as Random Access Memory (RAM).

General Notes:

A computer is generally comprised of three many modules: 1) memory, 2) Input/Output, and 3) a CPU. A high-level view of "memory" in a computer architecture is further broken down into various types of memory based on their purpose. Though a typical computer has many types of memory elements (such as registers in the datapath, when we speak of "memory" in a computer, we are generally referring to the relatively large "chunks" of related memory in a given module.

Memory, e.g. RAM, may be thought of as simply a table for which an address is provided and then data words are read or written from/to the specified address. Control lines, which may include /CE, /OE, and /WE, are used to activate the chip and control the flow of highs and lows appropriately in and out of the chip. The number of storage locations for words of data is equal to 2^N, where N is the number of address lines. For example, if a memory chip has 10 address lines, then it will have 2^{10} (1024) storage locations for words.

The RAT architecture falls into the category of being a RISC architecture. Being labeled a RISC architecture has several implications, in the context of memory, the RICS moniker implies that the architecture has a relatively large register file. The register file is a set of registers that the RAT CPU can access in a relatively fast manner using the RAT instructions.

1) Register File: The Register File is a small but essential memory subsystem in your RAT CPU and is an integral part of the CPU's datapath. The register file is 32 locations deep by 8-bits wide (32x8). We refer to the register file as a "dual port RAM", which means you can simultaneously read from two separate memory locations in the register file, but you can only write to one of those locations. The location you write to is one of the locations you read from. Note that the register file is read asynchronously (no clock) and written to synchronously (on the rising edge of a clock). All logic and arithmetic operations can only be performed on data saved in registers. If data from another location, ie inputs or Scratch RAM, the data must first be transferred to a register. Figure 1 shows the black box diagram for the register file.

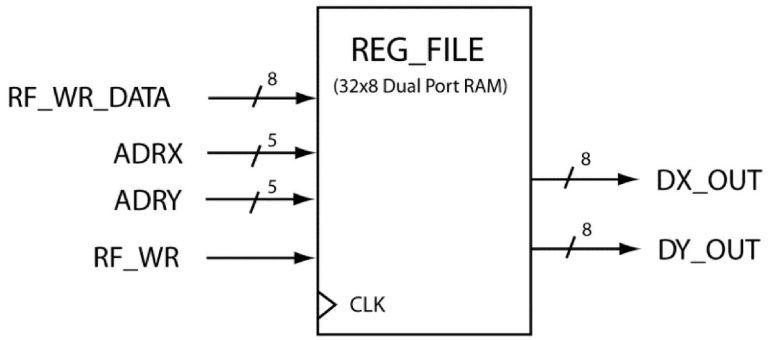

Figure 1: Register File Black Box Diagram

2) Scratch RAM: The scratch RAM in your RAT CPU serves as random access memory (RAM). The Scratch RAM serves two main purposes. First, the scratch RAM provides temporary storage that is accessible from using the RAT instruction set. In this manner, RAT instructions transfer data between the Register File and the scratch RAM. Second, the RAT architecture also uses the scratch RAM as the storage device for the stack. The RAT scratch RAM is 256 locations deep and 10 bits wide (256x10). Transferring data from the Scratch RAM to the register file uses only 8 bits while saving addresses for the stack uses all 10 bits. As with the register file, the scratch RAM is read asynchronously (no clock) and written to synchronously (rising edge of clock).

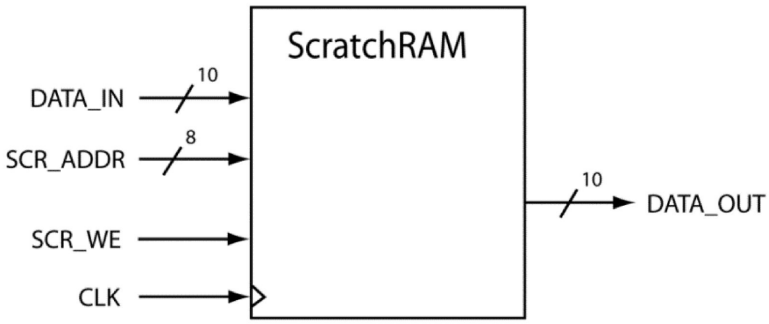

Figure 2: Scratch RAM Black Box Diagram.

VHDL Arrays:

Memory devices are easily defined in VHDL with arrays. Arrays are an indexed group of signals of uniformed size and type. Arrays size and type are defined first, and then a signal can be defined with the array type.

```
TYPE type_def is ARRAY (0 to locations-1) of STD_LOGIC_VECTOR (width-1 downto 0);
SIGNAL signal_name : type_def := (others => (others => '0')); -- initialize to 0s
```

The *signal_name* memory can be accessed by location index

```
signal_name(location_index)
```

The location_index must be an integer, so using another signal of std_logic_vector as an index must first be typecast or converted to an integer. This must be done in 2 stages, first defining the std_logic_vector as a numeric value of either signed (2's complement) or unsigned. Then the numeric value can be converted to an integer. This requires including the library **IEEE.NUMERIC_STD.ALL**

The code below assumes that **location_index** is a signal of type std_logic_vector and contains the address or location of the data to be accessed in the memory device **signal_name**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

Data_at_location <= signal_name(to_integer(unsigned(location_index)));
```

Assignment:

1) Using an array, design the REG_FILE memory module according to the black box diagram in Figure 1.
2) Using an array, design the Scratch RAM memory module according to the black box diagram shown in Figure 2.