ECE 571
Introduction to SystemVerilog
Spring 2021
Homework 5

Assume a memory, M, is modeled using a dynamic array of bytes as declared below along with a 32-bit register R and a memory address.

```
bit [31:0] R;
byte M[];
int address;
```

Using a combination of streaming operators and *indexed part selects*, write SystemVerilog code to load a 4-byte integer value from memory **M** at **address** into a 32-bit register **R**.

Submit your code demonstrating your solution works. This should be a single initial block in a module called **top()** in a file called **simple.sv**. Be sure to indicate with comments which problem the portions of your code corresponds to. Choose your sample data so that it demonstrates that your code is correct.

1) Assume the architecture is big-endian – that is the address of the operand is the address of the most significant byte with the next most significant byte at **address+1** and the least significant byte at **address+3**.

2) Assume the architecture is little-endian – that is the address of the operands is the address of the least significant byte with the next least significant byte at **address+1** and the most significant byte at **address+3**.

SIMD (single instruction, multiple data) instructions are a common microprocessor architecture feature that support parallel operations using "sub-word" parallelism. The standard architecture is extended to provide very large registers which can contain multiple operands allowing the ALU to perform the same operation in parallel.
Intel released several generations of SIMD instructions and correspondingly longer registers for the x86 architecture since their introduction (as the MMX extension for multimedia applications) to the Pentium in 1997. The most recent generation, AVX-512, in 2017, supports 32 512-bit ZMM registers.

3) Write SystemVerilog code to declare a new type (using **typedef** and a **union**) for an AVX-512 register that can be accessed as **byte**, **word**, **doubleword**, **quadword**, **single precision floating point**, or **double precision floating point** values. Use the SystemVerilog types (including two-state types where possible) that most closely match these types. Put this code in a package called SIMD (call the file **simd.sv**).

To make effective use of parallel ALU operations on multiple operands, the architecture must also support instructions to move data between these registers and memory.
Create a module that imports this package and declares an *unpacked* array of 32 ZMM registers (called ZMM) using the typedef, and a *dynamic* array of bytes called M used to model a memory.

Assume a testbench uses an unpacked array of 32 short unsigned integer operands as shown below.

```
    shortint unsigned operands[32], results[32];
```

4) Write a **`void function CopyToZMM`** that will *copy* all 32 shortints to ZMM register **`i`** from **`operands`**. Pass **`operands`** and the ZMM array by reference. Include it in your SIMD **`package`**.

5) Write a **`void function CopyFromZMM`** to *copy* the contents of ZMM register **`i`** into **`results`**. Pass **`results`** and the ZMM array by reference. Include it in your SIMD **`package`**.

6) Assuming the architecture is little-endian, write a **`void function LoadZMM`** that will *load* ZMM register **`i`** with 32 unsigned short integers from memory beginning at **`address`**. The **`shortint`** located at **`M[address]`** should be stored in the *most* significant two bytes of the ZMM register **`i`** , with the **`shortint`** located at **`M[address + 62]`** stored in the least significant two bytes of the ZMM register. The memory array should be passed by reference. Include it in your SIMD package.

7) Write the corresponding **`void function StoreZMM`** to *store* ZMM register **`i`** as 32 unsigned short integers into memory beginning at **`address`**. The memory array should be passed by reference. Include it in your SIMD package.

Create a file called **`stream.sv`** to import the SIMD package and define a module **`top()`** that demonstrates that your solutions are correct. The ZMM registers and the memory should be defined in **`top()`**.

8) What does the string method **`s.substr(i,j)`** return if the starting index of the substring is < 0 or the ending index is > **`s.len-1`**?

9) Create a **package** called strings in a file **`strings.sv`**. Write a string function **`splice`** that takes three arguments: a string s, a starting index int i, and an ending index int j, and if i and j are both valid indexes it returns the string that remains if the substring **`s.substr(i,j)`** were chopped out of s. It returns the empty string otherwise.

Create a module **`top()`** in a file called **`teststrings.sv`**. Import the strings package you created above. Execute enough testcases to demonstrate that your function works correctly.

Summary of files and modules (all included in single HW5.zip file):

`simple.sv`
module top() demonstrating solutions for (1, 2)
`simd.sv`
contains SIMD package with LoadZMM, StoreZMM, etc. functions (3-7)
`stream.sv`
imports SIMD package, module top() demonstrates functions (3-7)
`strings.sv`
strings package with splice function (8,9)
`teststrings.sv`
imports strings package, module top() demonstrates splice function (8,9)

Numbers in () correspond to homework problem numbers.