Question 1 (30 points): You are provided with the listing of an assembly program in MIPS that contains the code for two versions of a routine that computes the number of characters in a null-terminated string: `StringLenByte` and `StringLenWord`.

Assume that arithmetic and logic instructions (`add`, `addi`, `li`, `move`, `and`, `sll`) each take one cycle to execute; branches and jumps each take three cycles to execute, and loads (`lw`, `lb`) each take ten cycles to execute.

You are asked to answer the following questions about these routines.

a. (**5 points**) How many instructions are executed by each of the subroutines if `$a0` contains the address of `StringA` when the routine is called?

`StringLenByte:`

$$\begin{aligned}\#\text{instructions} &= \text{Instr. Before Loop} + \text{Instr. in Loop} + \text{Instr. After Loop}\\ &= 3 + 5 \times 4 + 1 = 3 + 20 + 1 = 24 \text{ instructions}\end{aligned}$$

`StringLenWord:`

$$\begin{aligned}\#\text{instructions} &= \text{Instr. Outside Loops} + \text{Instr. Outer Loop} + \text{Instr. Inner Loop}\\ &= 2 + (2 \times 2 + 2) + (5 \times 5 + 2 \times 1) = 2 + 4 + 2 + 25 + 2 = 35 \text{ instructions}\end{aligned}$$

b. (**5 points**) What are the values, expressed in hexadecimal, in registers `$t0`, `$t1`, and \$t2 at the end of the execution of `StringLenWord` if this routine is called with the address of StringA in `$a0`? (in the ASCII code: 'a' = 0x61, 'c' = 0x63, 'l' = 0x6C, 's' = 0x73)

```
$t0: 0x 0000 0073
$t1: 0x 0000 FF00
$t2: 0x 0000 0000
```

c. (**10 points**) Assume that `$a0` contains the address of a null-terminated string that is formed by 255 characters when each subroutine is called. What is the average number of clock per instruction (CPI) for each subroutine? Show your calculations.

`StringLenByte:`

The loop will be executed 255 times. The loop executes two arithmetic instructions, one branch and one load. Each instruction outside of the loop will execute a single time. Therefore:

$$\begin{aligned}\text{CPI} &= \frac{1 + 10 + 3 + 255 \times (2 + 10 + 3) + 3}{3 + 255 \times 4 + 1}\\ &= \frac{17 + 255 \times 15}{4 + 255 \times 4} = \frac{3842}{1024} = 3.75 \ \frac{\text{cycles}}{\text{instruction}}\end{aligned}$$

`StringLenWord:`

The string will be contained in $\frac{256}{4} = 64$ words. Therefore the `NextWord` loop will execute 64 times. For each iteration of the `NextWord` loop the inner `NextByte` loop will execute four times. Therefore:

$$
\begin{aligned}
\text{CPI} &= \frac{1 + 64 \times (1 + 10 + 4 \times (3 \times 1 + 2 \times 3) + 1 + 3) + 3}{1 + 64 \times (2 + 4 \times 5 + 2) + 1} \\
&= \frac{1 + 64 \times (11 + 4 \times 9 + 4) + 3}{1 + 64 \times 24 + 1} = \frac{1 + 64 \times 51 + 3}{1538} = \frac{3268}{1538} = 2.12 \frac{\text{cycles}}{\text{instruction}}
\end{aligned}
$$

d. (**5 points**) For the 255-character null-terminated string, which subroutine is faster? By how much? Show your calculations.

Because both routines execute in the same machine (and thus have the same clock cycle), we can simply count the number of clock cycles (which we did in part (c)):

$$
\frac{\texttt{StringLenByte}_{\text{Time}}}{\text{StringLenWord}_{\text{Time}}} = \frac{3842}{3268} = 1.18
$$

`StringLenWord` is 1.18 times faster than `StringLenByte`.

e. (**5 points**) The subroutine `StringLenWord` is written for a little-endian machine. Would this routine work in a big-endian machine? If yes, explain why the endianness of the machine does not affect how the routine works. If no, explain which lines of the routine need to be changed and how they should be changed to create a big-endian version of the routine.

The routine, as written, would not work in a big-endian machine. Two changes are necessary for it to work in a big-endian machine:

(a) The initialization of the mask in line 79 must be changed to:

```
lui        $t1,     0xFF 00        # masc <-- 0x0000 00FF
```

(b) the shifting of the mask in line 85 should be changed to:

```
srl        $t1, $t1, 8                  # mask <-- mask << 8
```