```
###########################################################
#           cs315 Week 5 - part 3
#
#   ->  Common issues with Program 1
#       Read the following carefully. Ideally, these issues should not be repeated in Program 2
#
###########################################################

    1.  There is no multiply with immediate constant values. It is syntactically invalid.
        We need to put 10 into a register and then do the multiplication using 3 registers, NOT with 2 registers and an immediate constant value).

        For example, the following is syntactically invalid:
            mul $t0, $t0, 10        # invalid syntax. However, it works because assembler convert it into the following which is syntactically correct:


            li $at, 10              # put 10 into a register $at (reserved for assembler)
            mul $t0, $t0, $at       # do the multiplication using $at instead of 10

        But is the best approach? best approach is putting constant value 10 into a temporally register. For example:
            li $t9, 10              # load a temporally register with constant value 10
            mul $t0, $t0, $t9       # do multiplication using temporally register


    2.  There is no "immediate" branching. We need to put a constant value into a temporally register first and then do branching using register values

        For example, the following is syntactically invalid:
            beq $v0, 5, exit        # invalid syntax. However, it works because assembler convert it into the following which is syntactically correct:


            li $at, 5               # put 10 into a register $at (reserved for assembler)
            beq $v0, $at, exit

        But is the best approach? best approach is putting constant value 10 into a temporally register. For example:
            li $t9, 5               # load a temporally register with constant value 10
            beq $v0, $t9, exit      # do branching using temporally register instead of immediate constant value 5

    3. There is no direct loading a register with a static variable.
        The following codes are syntactically invalid. However, it works because assembler convert them into the following which is syntactically correc

        Invalid code (syntactically invalid):
                .data
            zero_p: .word 0
            five_p: .word 0

                .text
            ...
            lw $t0, zero_p
            lw $t1, zero_p
            lw $t2, five_p


        What actually happens thanks to the assembler's pre-processor (the following is syntactically correct):
                .text
            ...
            la $at, zero_p
            lw $t0, 0($at)

            la $at, zero_p
            lw $t1, 0($at)

            la $at, five_p
            lw $t2, 0($at)
```

```
4.  Instead of loading a temporally register with a constant value '1' in order to increment a register. We can use "addi" instruction.

    For example, instead of the following which is bad coding:
        li $t9, 1

        add $t0, $t0, $t9       # increment $t0 by 1:   $t0 <-- $t0 + $t9 (which holds value 1)
        sub $t0, $t0, $t9       # decrement $t0 by 1:   $t0 <-- $t0 - $t9 (which holds value 1)

    We can do something like this which is much easier to read and better coding style:
        addi $t0, $t0, -1       # increment $t0 by 1:   $t0 <-- $t0 + (+1)
        addi $t0, $t0, -1       # decrement $t0 by 1:   $t0 <-- $t0 + (-1)


5.  Zero is a non-negative number. When we say non-negative number, then it means numbers that are >= 0.
6.  By sufficiently enough comments, we meant writing comments on every block or system call describing in details what it is doing.
7.  Do not use "j" instruction (jump unconditionally). Use "b" instruction (branch unconditionally). It is against convention.

    For example, instead of the following which is bad coding style:
        loop:
        ...
        ...
            j loop      # jump unconditionally back to the beginning of the loop

    The correct coding style is the following:
        loop:
        ...
        ...
            b loop      # branch unconditionally back to the beginning of the loop

8.  Make sure program even runs on QtSpim before submitting it. You will loose point if your program doesn't even run in QtSpim.
9.  Extension of assembly language is ".s" (e.g. Hesamian_Program1.s). Do not submit a ".docx" or ".txt" as assembly files.
10. Do not branch to the "main". Create a label right after "main" and branch to that instead.

    For example, the following is bad coding design and can be problematic (when dealing with scopes in compiler):
        main:
            ...
            ...
            b main

    The following solves the issue by creating a dummy label after "main":
        main:
        loop:
            ...
            ...
            b loop

11. The following is syntactically invalid. However, it works because assembler corrects it for you.

    For example, instead of this which syntactically invalid:
        la $a0, 0($t0)

    Write something like this:
        move $a0, $t0

12. Labels should all start with lower case character. Instead of "Average", you should write "average".
13. Initialize sum and count to zero before the loop. Do NOT assume register values are zero by default. Following the convention is necessary.
14. Avoid unnecessary newlines. It makes code less readable.

    For example: why there are 7 empty lines before halt?
        addi $t8, $t8, 1
        blt $t8, $t5, ave
                                # why?
                                # why?
                                # why?
                                # why?
                                # why?
                                # why?
        li $v0, 10 # halt
        syscall
```

15. Please use a ASCII text editor (sublime text, vim, notepad, notepad++). There are only 128 ASCII characters. Avoid using Microsoft office or Wor
16. The following is syntactically incorrect:

    For example:
    ```
        addi $t3, -1        # However, it works because pre-processor of assembler fixes it for you, but it is syntactically invalid.
    ```

    Instead, you should have wrote which is syntactically correct:
    ```
        addi $t3, $t3, -1
    ```

17. Comments should be on this same line with code. Because pre-processor of assembler remove the lines that start with "#", so when we debug the co
    To solve the issue, write comments after each line of code.

    For example, instead of this which is bad coding design:
    ```
        # display prompt
        li $v0, 4
        la $a0, prompt
        syscall
    ```

    Write should write something like this:
    ```
        li $v0, 4           # display prompt
        la $a0, prompt
        syscall
    ```

18. Avoid unnecessary branches.
    For example,
    ```
        ...
        ...
        b exit              # branch unconditionally to the next line of code. Unnecessary branch.
    exit:
        ....
    ```