```
############################################################
#           cs315 Week 2
#
#   Review MIPS components in-details.
#
############################################################

Directives:
    * commands for the assembler
    * start with a period
    * examples:
        .data   # starts data section (data section contains data for static memory)
        .text   # starts text section (text section contains instructions for program memory)

Labels:
    * text names used to identify a variable or a position in a program
    * labels are bound to addresses.
    * examples:
        --> load address macro (la) loads the static memory address of the given variable (e.g. la $a0, hello_p)
        --> jump and branch commands change the program counter (PC) to the address of the given label (e.g. blez invalid_entry)

    * label naming:
        --> must start with a letter
        --> may contain letters, number, and underscores
        --> a colon ( : ) denotes the end of the name (the colon is not part of the name)
        --> label names may not be duplicated (e.g. there can be only one string named hello_p)
        --> QtSpim is case sensitive (e.g. hello_p and Hello_p are different labels)

Variables:
    * integers:     {name:}  .word value {: number_of_elements}
    * examples:
        count:      .word 10    #   declares one word named 'count', initializes it to 10
        valueArray: .word 0:15  #   declares an array named 'valueArray' of 15 words, initializes all 15 words to 0
    * strings: {name:}  .asciiz "text"
        examples:
            hello_p:  .asciiz "Hello, world!"   # declares a string named hello_p containing the text "Hello, world!"
        notes:
            .asciiz declares "null terminated" strings, i.e. strings with a 0 appended immediately after the last character.
                --> the 0 (the null terminator) is used to denote the end of the string.
                    this null terminator will increase the string length by one character
                    e.g. prompt_p:  .asciiz "xyz" will contain four characters: x, y, z, and the null terminator
            .ascii declares "non-null terminated" strings, i.e. strings without a null terminator.
                --> without a null terminator, the last character of the string must be known ahead of time
                    i.e. we can't determine the end by examining the string memory
                    we will most likely never use .ascii and will be using .asciiz instead

    * bytes (characters):   {name:}  .byte value
        examples:
            newline:    .byte 10

Program labels:
    * used to jump or branch around in the program
    * used for clarity (e.g. 'print_array' should precede code which will print an array)
    * are bound to the instruction immediately following the label.
        --> example:
            average_loop:
                blez $t2, end_loop
                lw $t1, 0($t0)
                ...
                b average_loop
```

```
            end_loop:

        --> average_loop is bound to the address of the 'blez' command
    * may not be duplicated (e.g. only one 'average_loop' per program)

Immediate values:
    * immediate values are 'hard coded' in instruction (16 bit constant values, NOT 32 bit)
    * may be thought of constants
    * example:
        li $t0, 10          # loads the immediate value '4' into $v0
        addiu $a0, $t2, 7   # adds the immediate value '7' to the value in $t2 and stores the result in $a0

Arithmetic:
    * see appendices in MIPS book for full list on instructions
    * have instructions for add, subtract, multiply, and divide
    * addition examples:
        add $t2, $t0, $t1   # $t2 <-- $t0 + $t1
        addiu $t2, $t0, 12  # $t2 <-- $t0 + 12

Subtraction examples:
    * sub $t2, $t0, $t1     # $t2 <-- $t0 - $t1
    * No 'subtract immediate' (adding a negative immediate will work)

Multiplication:
    * uses 'HI' and 'LO' registers to hold result
        multiplying two 32 bit numbers yields a 64 bit result
        HI and LO are not connected to processor (i.e. cannot be used directly)
        'mflo' and 'mfhi' commands move values from LO and HI respectively.
    * example:
        mult $t2, $t4
        mflo $t1
        # ^^^ multiplies $t2 and $t4, then moves the lower 32 bits of the result into $t1

    * there is a multiplication instruction (mult) and a multiplication macro (mul).

Division:
    * integer division (i.e. yields an integer quotient and remainder, not a real number)
    * uses 'HI' and 'LO' registers for result
    * LO - quotient
    * HI - remainder
    * 'mflo' and 'mfhi' used to move values (see multiplication)

There is a divide instruction and a divide macro!
    * both are 'div'
    * whether the command is an instruction or a macro depends on the number of registers given
    * two registers - true assembly instruction
    * three registers - macro instruction
    * examples:
        div $t2, $t3        # true assembly instruction, divides $t2 by $t3, puts quotient and remainder in LO and HI, respectively
        div $t1, $t2, $t3   # macro, divides $t2 by $t3, puts quotient and remainder in LO and HI, also moves LO into $t1
    * DO NOT DIVIDE BY 0
    * it is the programmer's responsibility to avoid division by 0
    * QtSpim will throw an error if a divide by 0 is attempted
    * example:
        beqz $t3, avoid_div_by_zero
        div $t0, $t2, $t3
        # ^^^ branches away from the div command if $t3 is 0

Jumping and branching:
    * jumping writes a value to PC, branching adds a value to PC
```

```
* jumping is used to move between subprograms
* branching is used to move around within a subprogram
* branching
    --> unconditional branch (b) - always branches
    --> conditional branch (beqz, blt, etc.) - branches if the given condition is true, does nothing if the given condition is false
    --> examples:
            bgt $t4, $t2, some_label    # branches if $t4 > $t2, does not branch if $t4 <= $t2
            beqz $t1, some_label        # branches if $t1 == 0, does not branch if $t1 != 0
            # ^^^ see appendices (instructions and macros) for list of possible branch conditions
```