```
############################################################
#           cs315 Week 4 - part 1
#
#   ->  Subprograms, passing arguments IN and OUT using registers (WITHOUT using stack)
#       use of static variable, use of $a and $v registers for argument IN & argument OUT respectively
#       use of [subprogram_template.s] that is on D2L
#
############################################################

What is a subprogram? It's a way we organize our code (the flow, the layout, etc.)

    * the terms 'subprogram', 'subroutine', or 'function' may be used interchangeably here
    * a subprogram template is provided on D2L[subprogram_template.s], similar to the main program template
    * subprogram template must be used
    * processor has no concept of 'subprogram', we must build this ourselves
    * processor wants to execute instructions one at a time in linear order
    * all 'non-linear' structure (loops, if statements, functions, etc.) is created by us
    * following convention is necessary for full credit
    * subprograms is very similar to function or methods in higher level languages
    * subprograms uses the same 'call/return' concept as high level languages.
    * subprograms can pass arguments IN, can return values (also called 'arguments OUT')
    * subprograms DO NOT have their own set of registers, they use the same 32 registers as everything else in the program


* Problem:  Two subprograms may need to use the same register
* Solutions:
    Avoid using the same registers
    It is Not a best solution but is good enough for now (in future lectures, we will use stack to to overcome this issue)

We call subprogram using 'jal' instruction (jump and link)
    --> Jump:   jumps to the given label
    --> Link:   loads $ra with the address of the instruction immediately after the "jal" instruction

$ra:  return address register
    * used to store the address to return to
    * returning from a subprogram
    * return with 'jr' command: Jump register (not "jump return")
    * jumps to an address specified in a register
    * almost always used with $ra for our purpose (i.e. "jr $ra")

Ex: calling subprogram "read_values" from main:

        .text
    main:

        jal read_values     # jump to subprogram "read_values" and put the address of next instruction in register "$ra"
        add $t0, ....       # Thus, after subprogram finishes it's task, it will jumps back to this line of code
        ...
        ..
        .


    read_values:
        ...
        ..
```

```
        .

        jr $ra              # jump to the address that is in register "$ra". In this case, address of next instruction after "jal read_values"

What about arguments and return values from subprogram?
    * we will call arguments passed before subprogram call "arguments IN"
    * and values returned from subprogram as "arguments OUT"

Essentially, "arguments IN" is the same concept as passing arguments into function or methods, and "arguments OUT" is similar to return values


Passing "arguments IN":
    * use $a registers
    * moves values into $a registers BEFORE using "jal" instruction
    * load values from $a registers AT START OF SUBPROGRAM into $t registers
    * DO NOT PASS ARGUMENTS USING $t REGISTERS
    * passing values through $t registers violates convention
    * processor will allow it, HOWEVER by doing so you will lose points


Problem:    Only four $a registers exist, but more than four words need to be passed in
Solution:   Pass values on the stack (it will be covered later)


Returning values (or passing arguments OUT):
    * use $v registers
    * move values into $v register BEFORE "jr"
    * load values from $v registers AFTER the "jal" which called the subprogram
    * DO NOT PASS VALUES USING $t REGISTERS


Problem:    Only two $v registers exist, but more than two words need to be returned
Solution:   Pass values on the stack


Use of $a and $v registers will be noted in the subprogram description (see D2L template[subprogram_template.s] register usage section)

Ex: If description specifies $a0 as 'width', then 'width' MUST be passed in $a0 for full credit

Problem:    Only one "$ra" register exists
            Thus, calling a subprogram from another subprogram will overwrite "$ra"
            If return point to first subprogram is overwritten, there is no way to return to it


Ex:  If subprogram A calls subprogram B and subprogram B calls subprogram C, the "jal" in B will overwrite $ra and lose the return address to A
Solutions:  back up "$ra" on the stack
            For now, only 'leaf subprograms' will be used (i.e. subprograms which don't call other subprograms)


Subprogram convention
    * arguments IN are passed through $a registers
    * all arguments in must be in the appropriate $a registers when the "jal" command is executed
    * arguments OUT are passed through $v registers
    * all arguments out must be in the appropriate $v registers when the "jr" command is executed
    * subprograms have their own .data section
    * DO NOT use strings or variables declared in other subprograms (doing so will lose points)
    * DO not branch into a subprogram using anything other than "jal"
```

```
    * DO not branch out of a subprogram using anything other than "jr"


Why follow convention?
    * Programmers will work together as a team and must follow common methods for seamless work


Complete example (main code and 2 subprograms):
    * it demonstrates the convention of using subprogram (i.e. argument IN, OUT, use of register $a & $v)

*** pay attention to naming scheme of labels and variables

*** inside a subprogram (not inside main):
        [subprogram name] followed by "_" [variable or label name]

        Ex:
            read_integer_prompt_p
              ^             ^
        subprogram name   variable name


            print_integer_sum_p
              ^             ^
        subprogram name   variable name

#############################################################
#       Program Description
#
#   Description:
#       Write main code that call subprogram "read_integer". Then pass the save returned values
#       in static variables "integer_var_p". Then main will reload the value from static variable
#       and pass it as arguments to "print_integer" subprogram.
#
#############################################################
#       Register Usage
#   $t0     Holds sum of two integers
#   $t1
#   $t2
#   $t3
#   $t4
#   $t5
#   $t6
#   $t7
#   $t8
#   $t9     temporally
#############################################################
        .data
integer_var_p:      .word   0   # static integer variable initialized to 0
#############################################################
        .text
main:

    jal read_integer        # calling subprogram "read_integer"
                            # arguments IN: NONE
                            # arguments OUT: sum of two integers that it just read

    move $t0, $v0           # save returned value by moving it into register $t0
```

```
    la $t9, integer_var_p   # put the memory address of static variable "integer_var_p" into register $t9
    sw $t0, 0($t9)          # store value that is in register $t0 into memory at the address of [$t9 + 0]

    la $t9, integer_var_p   # put the memory address of static variable "integer_var_p" into register $t9
    lw $t0, 0($t9)          # load value that is in memory at the address of [$t9 + 0] into register $t0

    move $a0, $t0           # passing arguments IN before calling subprogram "print_integer"

    jal print_integer       # calling subprogram "print_integer"
                            # arguments IN: sum of two integers that is just read
                            # arguments OUT: NONE

    li $v0, 10              # halt
    syscall
############################################################
#       read_integer
#
#   Description:
#       read two integers, adds them together and return the sum back to main
#       using register $v0
############################################################
#       Arguments In and Out of subprogram
#
#   $a0
#   $a1
#   $a2
#   $a3
#   $v0     Holds return sum back to main
#   $v1
#   $sp
#   $sp+4
#   $sp+8
#   $sp+12
############################################################
#       Register Usage
#   $t0     Holds first integer
#   $t1     Holds second integer
#   $t2     Holds first integer + second integer
############################################################
        .data
read_integer_prompt_p:      .asciiz "Enter integer: "           # static ASCII string
############################################################
        .text
read_integer:
    li $v0, 4               # print enter integer:
    la $a0, read_integer_prompt_p
    syscall

    li $v0, 5               # read integer using system call 5
    syscall

    move $t0, $v0           # move first integer from register $v0 into $t0


    li $v0, 4               # print enter integer:
    la $a0, read_integer_prompt_p
```

```
    syscall

    li $v0, 5               # read integer using system call 5
    syscall

    move $t1, $v0           # move second integer from register $v0 into $t1


    add $t2, $t0, $t1       # add first integer with second integer and put result in $t2

    move $v0, $t2           # return $t2 (or sum) back to the main by moving it into register $v0

    jr $ra                  # return to calling location which is main
#############################################################
#       print_integer
#
#   Description:
#       receives as argument IN an integer, and then it will print the integer
#############################################################
#       Arguments In and Out of subprogram
#
#   $a0     Holds integer (sum of two numbers)
#   $a1
#   $a2
#   $a3
#   $v0
#   $v1
#   $sp
#   $sp+4
#   $sp+8
#   $sp+12
#############################################################
#       Register Usage
#   $t0     Holds first integer + second integer
#############################################################
        .data
print_integer_sum_p:        .asciiz "Sum is: "          # static ASCII string
#############################################################
        .text
print_integer:
    move $t0, $a0           # move sum into register $t0 so we do not lose it

    li $v0, 4               # print sum is:
    la $a0, print_integer_sum_p
    syscall

    li $v0, 1               # print integer using system call 1
    move $a0, $t0           # put integer into register $a0 from $t0
    syscall

    jr $ra                  # return to calling location which is main
#############################################################
```