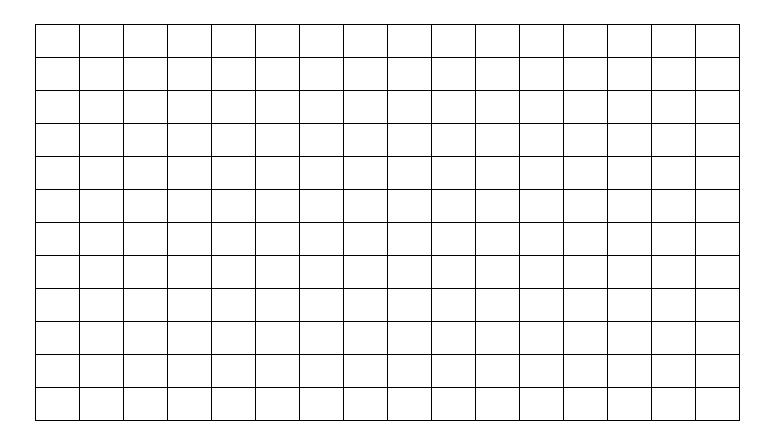
Floating point multiplication exercise:



Write a subprogram <code>allocate_double_array</code> that will prompt a user for array size and then allocates a dynamic of doubles array given array size. This subprogram does not get any arguments IN but returns two argument OUT, array base address and array size.

Write a subprogram <code>get_average</code> that will get as argument IN base address of an array of doubles and array size. This subprogram returns an average in double precision floating point format.

Floating point multiplication exercise:

```
--> 1 - 1000 0000 (1) 110 0000
--> 0 - 1000 0001 (1) 010 1000
         5.250
         -18.375 (expected)
    1 XOR \emptyset = 1 < -- sign of result is negative
    tentative exponent = 1 + 2 = 3
         1110 0000
        1010 1000
        1111 1
0000 0000 0000 0000
         0000 0000 0000 0000
         0000 0000 0000 0000
         0000 0111 0000 0000
         0000 0000 0000 0000
         0001 1100 0000 0000
         0000 0000 0000 0000
         0111 0000 0000 0000
         1223 2211 0000 0000
    %2 1001 0011 0000 0000
    16th bit is 1, hence, add 1 to tentative exponent to get final exponent
         \Rightarrow final exponent = 3 + 1 = 4
    1 - 10000011 (1) 001 0011
    1.0010011 * 2^4 = 10010.011 = -18.375 (expected)
Write a subprogram allocate_double_array that will prompt a user for array size and then
allocates a dynamic of doubles array given array size. This subprogram does not get any
arguments IN but returns two argument OUT, array base address and array size.
         .data
allocate_double_array_prompt_p: .asciiz "Enter array size ( > 0 ): "
         .text
allocate_double_array:
allocate_double_array_loop:
    li $v0, 4
    la $a0, allocate_double_array_prompt_p syscall
                                                 # prompt for array size
    li $v0, 5
                                                 # read array size from console
    syscalĺ
    blez $v0, allocate_double_array_loop
                                                 # if array size <= 0 then re-prompt</pre>
    move $t1, $v0
                                                 # copy array size into register $t1
    li $v0, 9
sll $a0, $t1, 3
                                                 # allocate array using system call 9
# multiply array size by 2^3 = 8
    syscall
    sw $v0, 0($sp)
                                                 # return base address
    sw $t1, 4($sp)
                                                 # return array size
    jr $ra
                                                 # jump back to the main
Write a subprogram get_average that will get as argument IN base address of an array
of doubles and array size. This subprogram returns an average in double precision floating
point format.
         .data
         .text
get_average:
 save arguments so we do not lose them
lw $t0, 0($sp)  # load array
lw $t1, 4($sp)  # load array
                               # load array base address
                               # load array length
    li.d $f4, 0.0
li.d $f6, 0.0
                               # initialize sum to zero
                               # initialize average to zero
    move $t2, $t1
                               # copy length into $t2 so we do not lose it
```

```
get_average_loop:
   blez $t2, get_average_loop_end # while($t2 > 0)
                   l.d $f8, 0($t0)
add.d $f4, $f4, $f8
                                                                                                                                           # load array value
# add the number with the sum and put the result back to the sum
                   addi $t0, $t0, 8
addi $t2, $t2, -1
                                                                                                                                           \# increment array pointer (address) to next two words (each double-precision floating-precision floating-
                    b get_average_loop
                                                                                                                                             # branch unconditionally back to beginning of the loop
get_sum_avg_loop_end:
                  mtc1 $t1, $f8
cvt.d.w $f8, $f8
div.d $f6, $f4, $f8
                                                                                                                                            # move to co-processor 1 from $t1
                                                                                                                                          # convert count to double from integer
# $f6|$f7 <-- $f4|$f5 / $f8|$f9
get_sum_avg_end:
                   s.d $f6, 8($sp)
                                                                                                                                            # return average
                    jr $ra
                                                                                                                                             # jump back to the main
```