4. In this question you will write a small MIPS program that halves/quarters the RGB color values of the entire image as described in the page before.

   (a) (10 points) Let us first develop a small MIPS assembly routine that will halve/quarter each 8-bit color component (Red, Green, Blue) individually and combine them together. The problem is that you can not just divide the 32-bit number that keeps the combined RGB value, you have to separate the values first, divide them individually and combine them again. (*Hint: consider using 'and', 'or' and 'shift' operators*)

   Assume that the 32-bit input value is located in the register \$a0. Please provide the dimmed value in the register \$v0.

```
        addi $t0, $t0, 255    # initialize a mask

        # GET B
        and  $t1, $a0, $t0    # mask only the B into $t1
        srl  $t1, $t1, 1      # divide B by 2 (unsigned number)

        # GET G
        srl  $a0, $a0, 8      # shift the number by 8 bits
        and  $t2, $a0, $t0    # mask only the G into $t2
        srl  $t2, $t2, 2      # divide G by 4 (unsigned number)

        # GET R
        srl  $a0, $a0, 8      # shift the number by 8 bits
        and  $v0, $a0, $t0    # mask only the R into $v0
        srl  $v0, $v0, 1      # divide R by 2 (unsigned number)

        # ASSEMBLE
        sll  $v0, $v0, 8      # shift the result by 8 bits left
        or   $v0, $v0, $t2    # or the divided value of G in $t2
        sll  $v0, $v0, 8      # shift the result by 8 bits left
        or   $v0, $v0, $t1    # or the divided value of B in $t1
```

   We use the mask in \$t0 to pick out exactly the 8 LSB in each step. We first start with B, mask it to \$t1, and divide it. Next we shift the entire number by 8 to the right. This way the next color value is ready to be processed (G). We mask and shift it by 2 to divede by 4 in \$t2. The last value (R) can then be already copied to \$v0. Now we move back and 'or' the calculated values in the reverse order first G in \$t2 and then B in \$t1.

   There are many alternative solutions that would work (i.e. using `lb` and `sb` which was not covered in class).

(b) (2 points) We want to convert the routine from the previous exercise into a subroutine named **dim_pixel** that can be called from a main program. Make the necessary modifications (additions and/or changes, if necessary) to your code from the previous part so that it can be a proper subroutine.

```

dim_pixel: # function label

       ### Your code from the previous part
       ### no need to replicate it if no changes
       ### are needed to the program

       jr $ra   # jump back to the main ($ra)
```

Since you will not be jumping out of this subroutine again, there is technically no need to save anything on stack. You will not lose points if you do so.

(c) (6 points) Now we need to make one program that will loop over the entire picture, load the color values, call the function **dim_pixel** and write the result back again. Write this program using MIPS assembly.

```
       # initializations
       lui  $s0, 0x2000         # load start address
       ori  $s0, $s0, 0x0000    # 0x2000 0000
       lui  $s1, 0x2040         # load end address
       ori  $s1, $s1, 0x0000    # 0x2040 0000

loop:  lw   $a0, 0($s0)     # load one value
       jal  dim_pixel       # call subroutine
       sw   $v0, 0($s0)     # store back value
       addi $s0, $s0, 4     # next pixel address
       beq  $s0, $s1, done  # end of loop?
       j    loop

done:  # finished execution
```

The solution has to be consistent with your answer to 4a. i.e. if you assumed `$a0` contains the address of the pixel, and not the value, the address needs to be loaded at this point.