For questions 4,5 and 6 we will discuss a real-time graphics application where each picture element (pixel) is stored as an array of 32-bit values in the memory. In this example the 32-bits will hold the color information for each individual pixel in the following way:

Bits 31:24 all zeroes

Bits 23:16 an unsigned 8-bit value for the Red value

Bits 15:8 an unsigned 8-bit value for the Green value

Bits 7:0 an unsigned 8-bit value for the Blue value

The array is a continuous memory region arranged in 1024 rows containing 1024 pixels each. It occupies 4 Mbytes of memory starting from `0x2000 0000` and ending at `0x203F FFFC`

In this exercise we will want to apply a *special dimming function* to the picture. The *red* and *blue* color values will be **halved**, while the *green* component which was found to be too dominant will be **quartered**, i.e. divided by four. (*Note: divisions by powers of two can be performed by simple right shifts*).

An example input value (`0x00FF C436`) and how the values get modified in hexadecimal, binary and decimal notations is shown below. (*Note that the first 8 bits are always 0, and need not be processed.*)

Bit		31 EMPTY	RED	GREEN	BLUE 0
Original	(HEX)	0 0	F F	C 4	3 6
Input	(BIN)	0000 0000	1111 1111	1100 0100	0011 0110
	(DEC)	0	255	196	54
Dimmed	(HEX)	0 0	7 F	6 2	1 B
Output	(BIN)	0000 0000	0111 1111	0011 0001	0001 1011
	(DEC)	0	127	49	27
Operation		none	HALVE	QUARTER	HALVE

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.

```
1        addi $t0, $t0, 255    # initialize a mask
2
3        # GET B
4        and  $t1, $a0, $t0    # mask only the B into $t1
5        srl  $t1, $t1, 1      # divide B by 2 (unsigned number)
6
7        # GET G
8        srl  $a0, $a0, 8      # shift the number by 8 bits
9        and  $t2, $a0, $t0    # mask only the G into $t2
10       srl  $t2, $t2, 2      # divide G by 4 (unsigned number)
11
12       # GET R
13       srl  $a0, $a0, 8      # shift the number by 8 bits
14       and  $v0, $a0, $t0    # mask only the R into $v0
15       srl  $v0, $v0, 1      # divide R by 2 (unsigned number)
16
17       # ASSEMBLE
18       sll  $v0, $v0, 8      # shift the result by 8 bits left
19       or   $v0, $v0, $t2    # or the divided value of G in $t2
20       sll  $v0, $v0, 8      # shift the result by 8 bits left
21       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.

```
1
2 dim_pixel: # function label
3
4        ### Your code from the previous part
5        ### no need to replicate it if no changes
6        ### are needed to the program
7
8        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.

```
1        # initializations
2        lui  $s0, 0x2000          # load start address
3        ori  $s0, $s0, 0x0000     # 0x2000 0000
4        lui  $s1, 0x2040          # load end address
5        ori  $s1, $s1, 0x0000     # 0x2040 0000
6
7  loop: lw   $a0, 0($s0)      # load one value
8        jal  dim_pixel        # call subroutine
9        sw   $v0, 0($s0)      # store back value
10       addi $s0, $s0, 4      # next pixel address
11       beq  $s0, $s1, done   # end of loop?
12       j    loop
13
14 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.