In this lab assignment, you will implement two image processing kernels in MIPS assembly. The input for both functions is a square sized grayscale image. Refer to the handout on memory layout to learn how a 2D array is stored in memory. In this case, images are stored in a 2D array where each element is a byte that represent each pixel's intensity (0x00 used for bla to 0xFF. Output image in both parts will have the same size as the input image. We have included a utility function that load and store image data from images stored on disk with pgm file format (a simple raw format). Your task is to write the body of the functions that loads an array of pixels along with its dimensions and process them and stores the result back in memory.

Fig. 1. Image processing pipeline

Part 1) Image Thresholding:

Image thresholding is a simple way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. An example of such conversion is shown below:

Lena.png

Lena_binary.png

Your function should replace the value of each pixel in the grayscale image with the maximum possible value (0xFF, i.e. brightest value), if the intensity (value) of that pixel is greater than or equal to some constant T (function input), or otherwise with the minimum value (0x00, i.e. darkest value). Remember that these values are unsigned.

Part 2) Affine Transformation:

For the second part, you are implementing a function to apply an affine transformation to the input image. Such transformation is described using a 2x3 transform matrix (M) that map pixels at input image to different coordinates in output image. Affine transformations have this property that they map parallel lines in input image to parallel lines in output image. Examples of affine transformations include: translation, rotation, reflection and scale.

The following algorithm demonstrates applying affine transformation to an input image. In this algorithm x and y represents current pixel coordinates -- x being the column index and y being the row index. Top-left pixel has coordinates of (0, 0).

$$for\ each\ input\ image\ pixel\ at\ x,y\ coordinate:$$

$$x_0 = M_{oo} \times x + M_{o1} \times y + M_{02}$$

$$y_0 = M_{1o} \times x + M_{11} \times y + M_{12}$$

$$output(x,y) = input(x0, y0)\ if\ cols > x_o \geq 0\ and\ rows > y_0 \geq 0\ otherwise\ 0$$

Although image pixel are bytes, matrix elements are integers. Here's some examples of affine transformations:

Sample input

1	0	0
0	1	0
0	0	1

Identity matrix

2	0	0
0	1	0
0	0	1

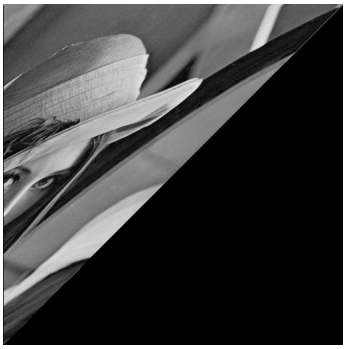
Scale matrix (x = 2)

0	1	0
1	0	0
0	0	1

Rotation matrix (+ translate)

1	1	0
0	1	0
0	0	1

Shear matrix



Testing

Lab3.s comes with a tester which tests your function with proper parameters such as input arrays and images. If your program passes tests on arrays correctly, you are guaranteed to receive full points for this lab.

To test your code against images, you need to modify the source code to point to input images location on your computer. “Lenna.pgm” is the input image and “lenna_thresh.pgm”, “lenna_scaled.pgm”, “lenna_shear.pgm” and “lenna_scale.pgm” are output images that will be generated by parts 1 and 2 of this lab. You need to verify them manually.


```
fin: .asciiz "PATH/TO/lenna.pgm"
fout_thresh: .asciiz "PATH/TO/lenna_thresh.pgm"
fout_rotate: .asciiz "PATH/TO/lenna_rotation.pgm"
fout_shear: .asciiz "PATH/TO/lenna_shear.pgm"
fout_scale: .asciiz "PATH/TO/lenna_scale.pgm"
```


- Submit ONLY **labs3.s** file to the EEE dropbox folder.
- Do NOT submit your project output files.