Introduction

A grayscale image is internally represented as a 2 dimensional array. Size is defined by the number of pixels in a row (width), and number of pixels in a column (height). Each cell represents the pixel value. Pixel values in grayscale are represented as white, black, and shades of gray using different numerical data type representation depending on the library used.

Some Libraries represent using 8 bit unsigned integer. 0 for black, 255 for white, 64 light gray, 191 dark gray, and all other gray values in between.

Image =

64, 89, 114, 84

140, 166, 191, 84

216, 242, 38, 84

Some Image Processing Libraries represent grayscale images as having single float based pixel values between 0 to 1. 0 for black, 1 for white, 0.25 light gray, 0.75 darker gray, and all other gray values in between.

Image =

0.25, 0.35, 0.45, 0.33

0.55, 0.65, 0.75, 0.33

0.85, 0.95, 0.15, 0.33

To enable using different image processing libraries simultaneously, conversion from one format to the other has to be done. **Perform Mapping from uint8 based integer grayscale to *single precision* float representation**. Each pixel is calculated using ratio and proportion equation

$$\frac{f}{i} = \frac{1}{255}$$

Where f is the float based pixel value and i is the integer based pixel value

Task

Implement a program that converts the grayscale image representation from int based to float based using C and x86 assembly language. Implement the function i**mgCvtGrayInttoFloat()**.

*Required to use functional scalar SIMD registers

*Required to use functional scalar SIMD floating-point instructions

Input: height, width, integer pixel values

Example.

3 4

64, 89, 114, 84

140, 166, 191, 84

216, 242, 38, 84

Output: Single Float pixel values

Example.

0.25 0.35 0.45 0.33

0.55 0.65 0.75 0.33

0.85 0.95 0.15 0.33

Note:

1.) C is responsible for: collecting the inputs, allocating memory spaces for the images, and printing the outputs.

2.) Function implemented in assembly is responsible for calculating and converting the data type from the input int pixels into the output single float pixels.

3) Time the asm function only for input image size *width*height*= {10*10, 100*100, and 1000*1000}. If 1000*1000 is impossible, you may reduce it to the point your machine can support. You may use a random number generator to generate pixel values for the input.

4.) You must run at least 30 times to get the average execution time.

5.) For the data, you may initialize each pixel with the same or different random value.

6.) You will need to check the correctness of your output.

7.) Output in GitHub (make sure that I can access your Github):

a.) Github readme containing the following (C and x86-64):

i.) execution time and short analysis of the performance

ii.) Take a screenshot of the program output with the correctness check (C).

iii.) short videos (5-10mins) showing your source code, compilation, and execution of the C and x86-64 program

b.) Submit all files needed to run your project. (source code: C, x86-64, and all other required files) for others to load and execute your program.

Rubric:

C main program with initialization and correct call/passing parameters to C and x86-64	25
Correct output (x86-64)	45
Performance result	20
Video	10
not following instructions	-10/instruction
Note: No usage of functional scalar SIMD registers and scalar SIMD instructions, function not in assembly	grade = 0