Project Assignment 1: Image Processing using Intel Assembly

Important notes:

- This assignment is taken directly from the following link. The sample source codes provided at the link may be helpful. However, please keep in mind that the assignment given at the link requires the use of MMX, but for this assignment there is NO such requirement.
  Link: *http://cs.fit.edu/~mmahoney/cse3101/*
- Group members will be graded individually based on their performance in the lab session. All members of the group are expected to know every detail about the project. Apart from the implementation details of the project, there will also be questions about the image processing issues used in the assignment in general. It is not recommended that you write the required functions in C and convert them to Assembly by disassembling them. Although it is not forbidden, you will be expected to be able to explain in detail every compiler-getenerated *instruction.*

Description of Assignment

An image is represented as an array of unsigned bytes, each representing one pixel, scanning left to right starting at the lower left corner. A value of 0 represents black and a value of 255 represents white. An image is grayscale, or may be one of the three components of a color image (blue, green, or red).

A group of functions written in Intel assembly language will be called from a program in C as shown below. Functions are not required to perform error checking on the parameters, but must follow C calling conventions and must fully conform to the given prototypes.

For all functions, if a pixel result is outside the range 0 to 255 then the result should be replaced with the closest legal value. The result should be written back to the image unless otherwise specified. You may assume that width is a positive multiple of 16 and height is a positive even number. You may assume that image starting addresses will be aligned on 16 byte boundaries.

Functions to be Implemented

Function 1:

```
void brightness(unsigned char *image, int width, int height, int val);
```

This function increases or decreases the brightness of an image. The image consists of width*height pixels. val is a number in the range -255 to 255 which is to be added to each pixel.

Function 2:

```
void contrast(unsigned char *image, int width, int height, int val);
```

This function changes the contrast of an image by multiplying each byte by val/64, i.e.
pixel = (pixel*val)/64; val will be in the range 0 to 255.

Function 3:

```
void average(unsigned char *image, unsigned char *image2,
             int width, int height);
```

This function replaces the image with the average of image and image2, rounding up, e.g. image[i] = (image[i] + image2[i] + 1) / 2. Both images have the same dimensions.

Function 4:

```
void grow(unsigned char *image, unsigned char *newimage,
          int width, int height);
```

This fuction doubles the size of image and puts the result in newimage. The new image should have dimensions of width*2 by height*2. You may assume newimage points to an array of width*height*4 bytes. Expand the image by copying each pixel 4 times (twice horizontaly and twice vertically).

Function 5:

```
void shrink(unsigned char *image, unsigned char *newimage,
            int width, int height);
```

This function is the opposite of grow(). The function shrinks the image by sampling every other pixel horizontally and vertically. The result should be placed in newimage, which has dimensions width/2 by height/2 and points to an array of width*height/4 bytes. The sampled pixels should have even numbered indices both horizontally and vertically, i.e. should sample the lower left corner of each 2 by 2 block.

Function 6:

```
void blur(unsigned char *image, int width, int height);
```

blur() is an antialiasing filter normally applied before shrink(). The image is divided into 2 by 2 blocks. The 4 pixels in each block are replaced with the average of those 4 pixels (rounding up or down as you choose).

Function 7:

```
void invert(unsigned char *image, int width, int height);
```

Produces a "negative" image by subtracting each pixel from 255 (i.e swap black with white). Two invert() operations will restore the original image.

Function 8:

```
void edge(unsigned char *image, int width, int height);
```

If a pixel is brighter than the one above it, replace it with a white pixel (255). If it is darker or equal, replace it with black (0). Pixels on the top row are set to black.

Function 9:

```
int atleast(unsigned char *image, int width, int height, int val);
```

This function returns the number of pixels with values above or equal to val (0 to 255). The image is not modified.

Function 10:

```
void clear(unsigned char *image, int width, int height, int val);
```

This function sets all pixels to val (0 to 255).

Submission Details

You are required to implement the given 10 functions in Intel assembly. The function implementations must fully conform to the provided prototypes since they are expected to be linked to the main program implemented in C which will be provided through the Ninova system. The main program will handle all input/output operations, so you do not need to read or write to the bitmap image files. Your submissions will be tested automatically via a script, therefore, you should use the provided C program without modifying it. Also you should not modify the bitmap file headers. We will provide the image files which will be used during the lab session to test your programs. These files may not be among the ones given on Ninova.

You are required to submit the assembly language source code file(s) through the Ninova system as a zip file.Each member of the group must make a submission, even though the submitted files may be the same for all group members.

Group members will be graded individually based on their performance in the lab session and the submitted group project. The students who are not present duing the lab session will not receive a grade for the project, even though they may have made a submission through the Ninova system.

Any form of cheating or plagiarism will not be tolerated. This includes actions such as, but not limited to, submitting the work of others as one's own (even if in part and even with modifications) and copy/pasting from other resources (even when attributed). Serious offenses will be reported to the administration for disciplinary measures.