SEE2030: Introduction to Computer Systems (Fall 2017)

Programming Assignment #3:

Pixelizing an Image

Due: November 26th (Sunday), 11:59PM

1. Introduction

In this assignment, you will implement a basic image processing program using the x64 assembly language. An image file in BMP format will be given as an input to your program. This assignment aims at introducing various primitive instructions provided by the x64 assembly language. In addition, you will learn the basic structure of the BMP image file.

2. Problem specification

Complete the file **`bmpmosaic.s`** which implements the function **`bmp_mosaic()`** in x64 assembly language. The prototype of **`bmp_mosaic()`** is as follows:

```
void bmp_mosaic (unsigned char *imgptr, int width, int height, int size);
```

The first argument, **`imgptr`**, points to the bitmap data which stores the actual image, pixel by pixel. The next two arguments, **`width`** and **`height`**, represent the width and the height of the given image, respectively. And the last argument, **`size`**, indicates the size of square for pixelization. Your task is to pixelize by manipulating the bitmap data in **`bmp_mosaic()`**.

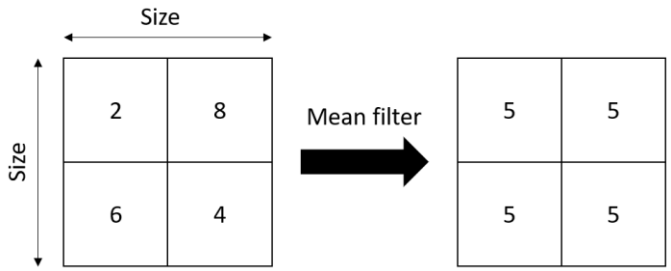
3. Backgrounds

3.1. Pixelization (mosaic)

Pixelization is any technique used in editing images or video, whereby an image is blurred by displaying part or all of it at a markedly lower resolution. It is primarily used for censorship. The effect is a standard graphics filter, available in all but the most basic bitmap graphics editors.
In this assignment, we will implement pixelization using mean filter.

3.1.1. Mean filter

Mean filter is also known as box filter an average filter. Mean filter just set average of given size of pixels. The process of mean filtering is as follows.



In case of pixels that do not complete square, we just calculate average on remaining pixels.

- Size of square: 2

1	2	3	4	1	2	3	4	1	2	3
4	1	2	3	4	1	2	3	4	1	2
3	4	1	2	3	4	1	2	3	4	1
2	3	4	1	2	3	4	1	2	3	4
1	2	3	4	1	2	3	4	1	2	3

→

2	2	3	3	2	2	3	3	2	2	2
2	2	3	3	2	2	3	3	2	2	2
3	3	2	2	3	3	2	2	3	3	2
3	3	2	2	3	3	2	2	3	3	2
1	1	3	3	1	1	3	3	1	1	3

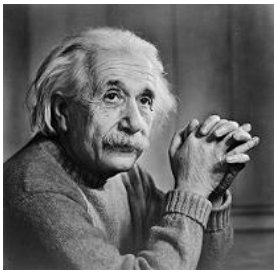
- Size of square: 4

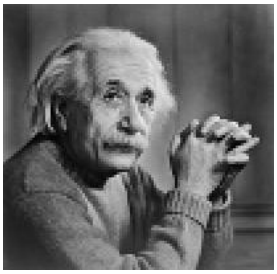
1	2	3	4	1	2	3	4	1	2	3
4	1	2	3	4	1	2	3	4	1	2
3	4	1	2	3	4	1	2	3	4	1
2	3	4	1	2	3	4	1	2	3	4
1	2	3	4	1	2	3	4	1	2	3

→

2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2

Here are examples of pixelization result using mean filter below.


Original image

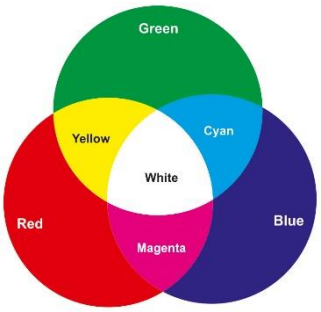

Size of square: 2


Size of square: 4


Size of square: 8

3.2. RGB color model

The RGB color model is one of the most common ways to encode color images in the digital world. The RGB color model is based on the theory that all visible colors can be created using the primary additive colors, red, green, and blue. When two or three of them are combined in different amounts, other colors are produced. The RGB model is important to graphic design as it is used in computer monitors.

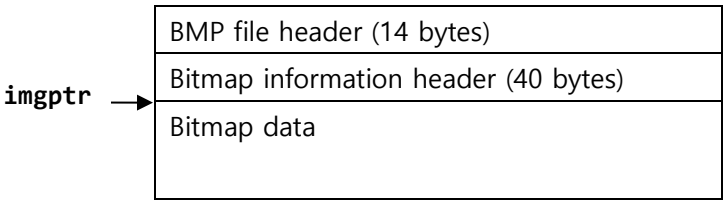


3.3. BMP file format

The BMP file format is an image file format used to store digital images, especially on Microsoft Windows operating systems. A BMP file contains a BMP file header, a bitmap information header, an optional color palette, and an array of bytes that defines the bitmap data. Since the BMP file format has been extended several times, it supports several different types of encoding modes. For example, image pixels can be stored with a color depth of 1 (black and white), 4, 8, 16, 24 (true color, 16.7 million colors) or 32 bits per pixel. Images of 8 bits and fewer can be either grayscale or indexed color mode. More details on the BMP file format can be found at http://en.wikipedia.org/wiki/BMP_file_format.

In this assignment, we will focus only on the 24-bit uncompressed RGB color mode with the

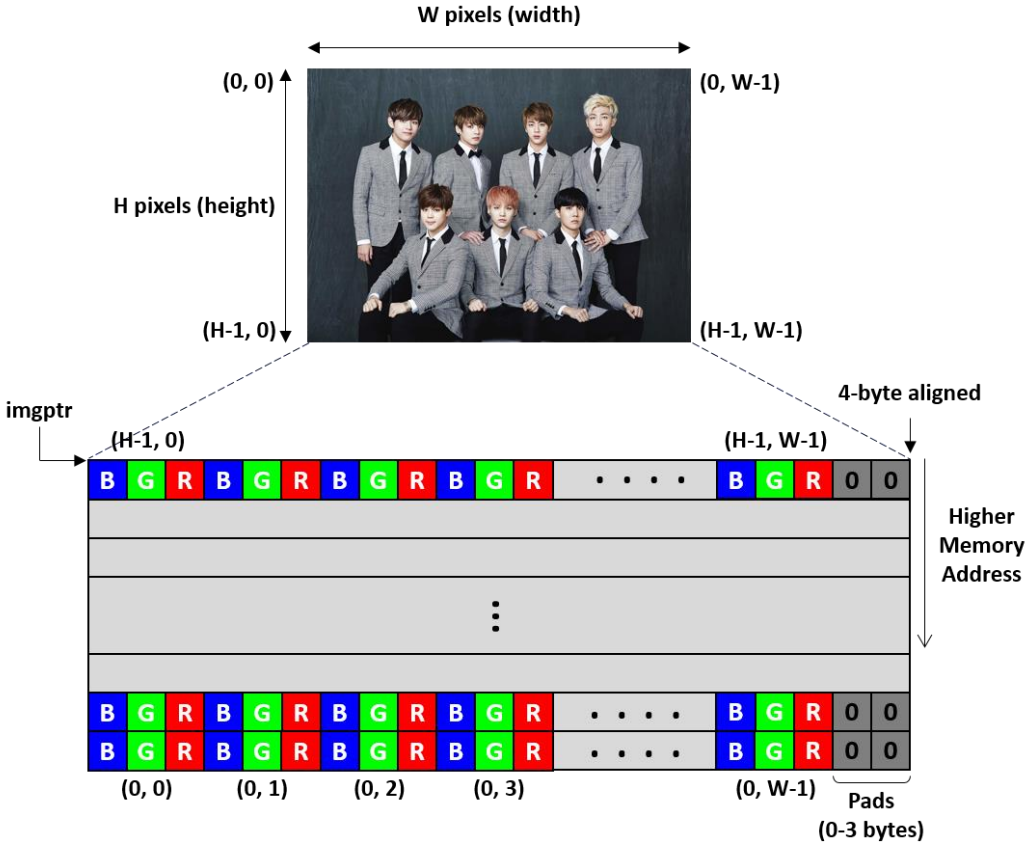
"Windows V3" bitmap information header. Under this mode, our target image file has the following structure.



We will provide you with the skeleton codes, in which all the BMP file header and the Bitmap information header are parsed. So you don't have to worry about these headers. Before manipulating the bitmap data, we check for these headers to make sure the target image file is in the right mode, and then extract the width and the height of the image. The first argument of **`bmp_mosaic()`**, **`imgptr`**, will point to the memory address that contains the actual bitmap data.

3.4. Bitmap data format

The bitmap data describes the image, pixel by pixel. Each pixel consists of an 8-bit blue (B) byte, a green (G) byte, and a red (R) byte in that order. Pixels are stored "upside-down" with respect to normal image raster scan order, starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image. Note that the number of bytes occupied by each row should be a multiple of 4. If that's not the case, the remaining bytes are padded with zeroes. All you have to do is to find all the pixels on grid lines and change the corresponding bytes of such pixels to (B = 0, G = 0, R = 255). The following figure summarizes the structure of the bitmap data.



4. Skeleton codes

The following skeleton codes are provided for this assignment.

Makefile :	This is a file used by the GNU **`make`** utility.
bmp.c:	This is a C program which contains **`main()`**, **`bmp_in()`**, and **`bmp_out()`** functions. The **`bmp_in()`** function reads the content of the BMP file into the memory and parses the header. The **`bmp_out()`** function creates a new image file which is modified by **`bmp_mosaic()`**.
bmpmosaic.s	This is a skeleton assembly code for **`bmp_mosaic()`**. You are supposed to fill the main body of this file.

You can build the executable file using the "**`make`**" command. The name of the final executable file is **`bmpmosaic`**. The skeleton codes can be downloaded from the course homepage.

5. Requirements

- In the main body of **`bmp_mosaic()`**, you should use `%rax`, `%rbx`, `%rcx`, `%rdx`, `%rsi`, and `%rdi` registers only. If you are running out of registers, use stack as temporary storage.
- Among the registers you can use, **`%rbx`** is one of callee-save registers. Therefore, you have to save and restore the original value of the **`%rbx`** register in **`bmp_mosaic()`**.
- Your program should work for BMP images of any size.

6. Sample output

This is one of sample BMP file (**twice.bmp**).

If you run your program as follows, it will create a new file named "**result.bmp**".

```
$ ./bmpmosaic twice.bmp result.bmp 4
```

The resulting **result.bmp** file should look like this. In this example, the size of pixelizing square is 4.

7. Verification

To verify whether your output is correct or not, we provide two sample input BMP files (einstein.bmp and twice.bmp) and the corresponding reference output BMP files (einstein_2.bmp, einstein_4.bmp, einstein_8.bmp, and twice_4.bmp, suffix number indicates the pixelization square size). BMP files generated by your program should be identical to the reference BMP files. To check whether the contents of two BMP files are the same or not, use the **cmp** command as follows:

```
$ ./bmpmosaic twice.bmp output.bmp 4
$ cmp output.bmp twice_4.bmp
$ <no output if two files are identical>
```

8. Hand in instructions

- Submit only the **bmpmosaic.s** file to the submission site (http://sys.skku.edu).

9. Logistics

- You will work on this assignment alone.
- Only the assignments submitted before the deadline will receive the full credit. 25% of the credit will be deducted for every single day delay.
  - You can use up to 5 slip days
- Any attempt to copy others' work will result in heavy penalty (for both the copier and the originator). Don't take a risk.

Good luck!

TA: Kisik Jeong (kisik.jeong@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University