서울대학교
SEOUL NATIONAL UNIVERSITY

Systems Software Laboratory
Dept. of Computer Science and Engineering

4190.308: Computer Architecture (Spring 2018)

Project #3: Drawing grid lines in an image

Due: May 13th (Sunday), 11:59PM

1. Introduction

In this project, you will implement a basic image processing program using the x86-64 assembly language. An image file in the BMP format will be given as an input to your program. This assignment aims at introducing various primitive instructions provided by the x86-64 assembly language. In addition, you will learn the basic structure of the BMP image file.

2. Problem specification

2.1 Overview

Complete the file `bmpgrid.s` which implements the function `bmp_grid()` in the x86-64 assembly language. The prototype of `bmp_grid()` is as follows:

```
void bmp_grid (unsigned char *imgptr, long long width, long long height,
               long long gap);
```

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. The last argument, `gap`, indicates the number of pixels between consecutive horizontal or vertical lines. Your task is to draw grid lines in red color every "`gap`" pixels in x and y directions by manipulating the bitmap data in `bmp_grid()`.

3. Backgrounds

3.1 RGB color model

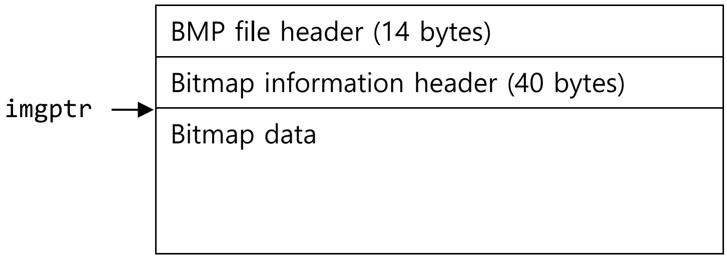
RED
YELLOW
MAGENTA
GREEN
CYAN
BLUE

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.

3.2 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.

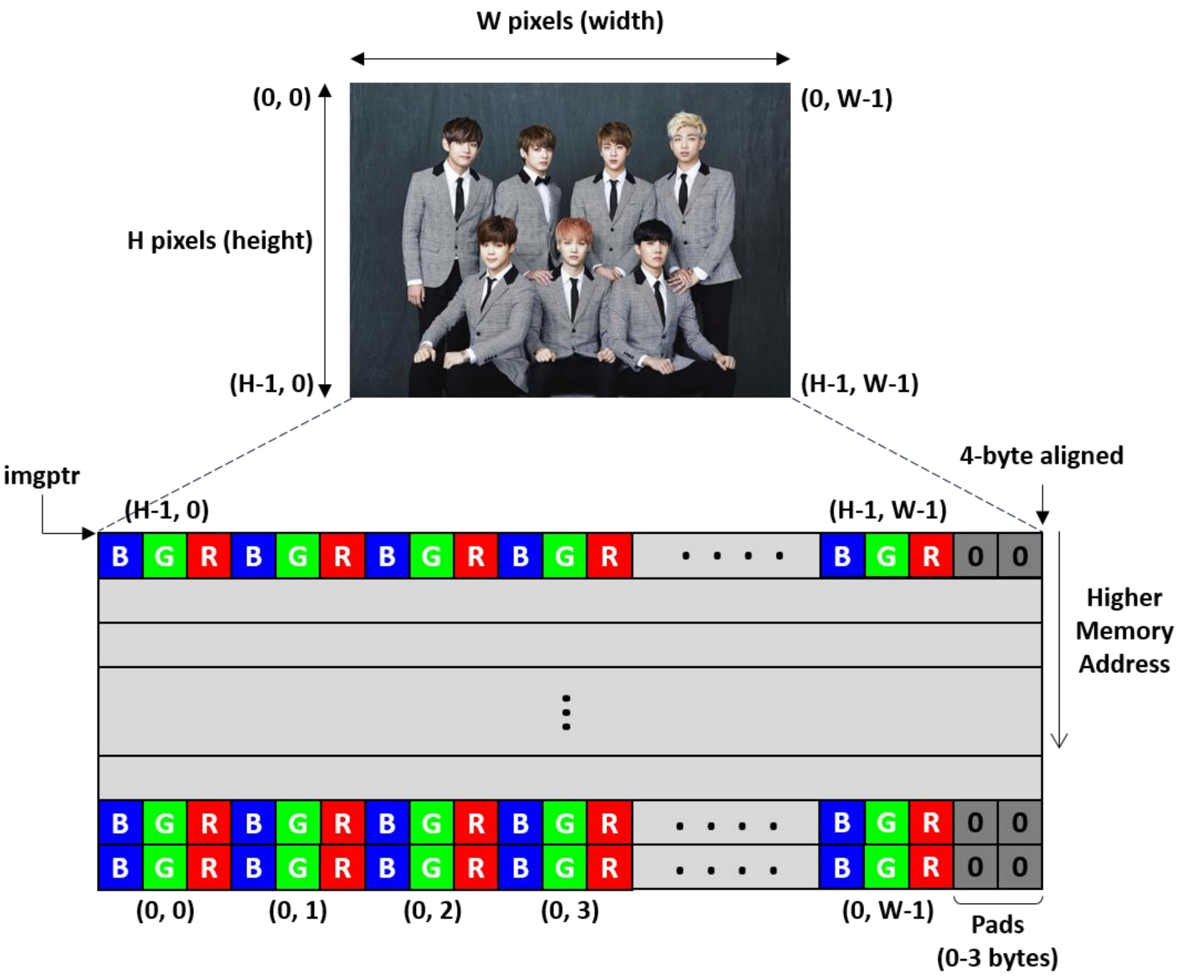
In this project, we will focus only on the 24-bit uncompressed RGB color mode with the "Windows V3" bitmap information header. Under this mode, our target image file has the following structure.


BMP file header (14 bytes)
Bitmap information header (40 bytes)
imgptr
Bitmap data

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_grid()`, `imgptr`, will point to the memory address that contains the actual bitmap data.

3.3 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.

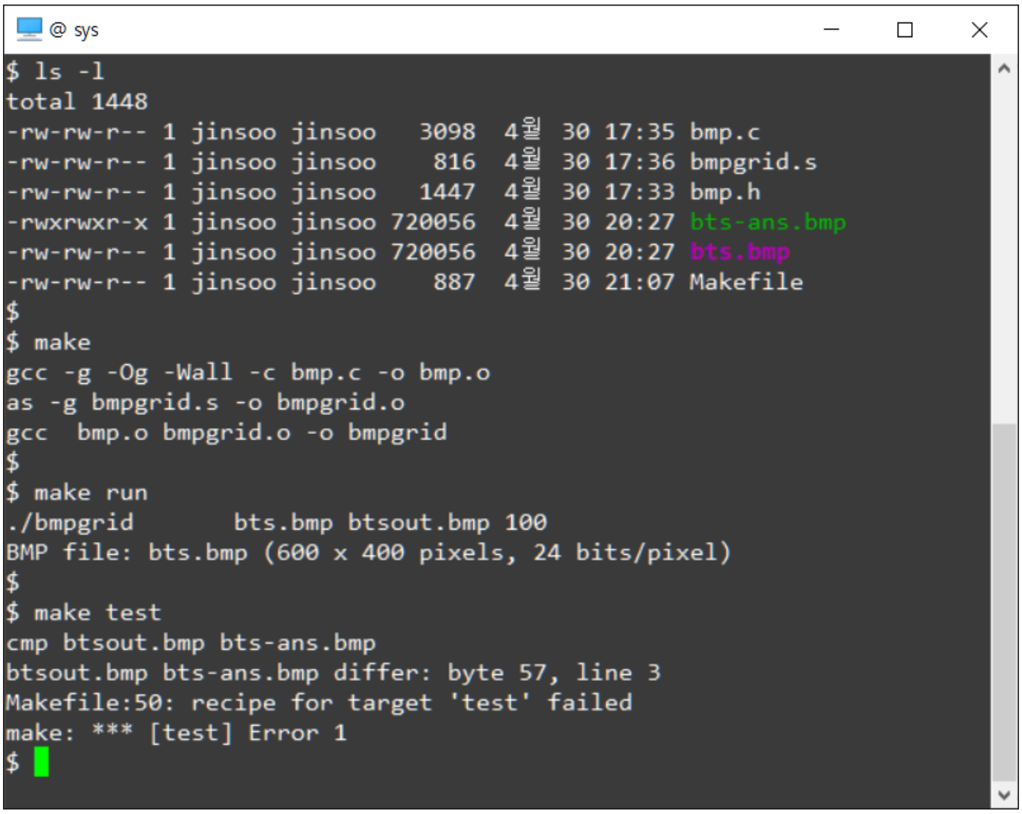


4. Skeleton codes and sample data

The following skeleton codes and sample data are provided for this project.

`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 its header. The `bmp_out()` function creates a new image file which is modified by `bmp_grid()`.
`bmpgrid.s`	This is a skeleton assembly code for `bmp_grid()`. You are supposed to fill the main body of this file.
`bts.bmp`	This is an example BMP file.
`bts-ans.bmp`	This is the output BMP file when the `gap` is set to 100.

You can build the executable file using the "`make`" command. The name of the final executable file is `bmpgrid`. You can also perform "`make run`" and "`make test`" to see if your output is correct or not, as shown in the following screenshot. The skeleton codes and sample data can be downloaded from the `sys.snu.ac.kr` server.



```
$ ls -l
total 1448
-rw-rw-r-- 1 jinsoo jinsoo   3098 4월 30 17:35 bmp.c
-rw-rw-r-- 1 jinsoo jinsoo    816 4월 30 17:36 bmpgrid.s
-rw-rw-r-- 1 jinsoo jinsoo   1447 4월 30 17:33 bmp.h
-rwxrwxr-x 1 jinsoo jinsoo 720056 4월 30 20:27 bts-ans.bmp
-rw-rw-r-- 1 jinsoo jinsoo 720056 4월 30 20:27 bts.bmp
-rw-rw-r-- 1 jinsoo jinsoo    887 4월 30 21:07 Makefile
$
$ make
gcc -g -Og -Wall -c bmp.c -o bmp.o
as -g bmpgrid.s -o bmpgrid.o
gcc  bmp.o bmpgrid.o -o bmpgrid
$
$ make run
./bmpgrid       bts.bmp btsout.bmp 100
BMP file: bts.bmp (600 x 400 pixels, 24 bits/pixel)
$
$ make test
cmp btsout.bmp bts-ans.bmp
btsout.bmp bts-ans.bmp differ: byte 57, line 3
Makefile:50: recipe for target 'test' failed
make: *** [test] Error 1
$ █
```

5. Requirements

- In the main body of `bmp_grid()`, 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_grid()`.
- Your program should work for BMP images of any size.
- Your program should work for any positive value of "`gap`".
- You should leave the bytes in the padding area untouched.

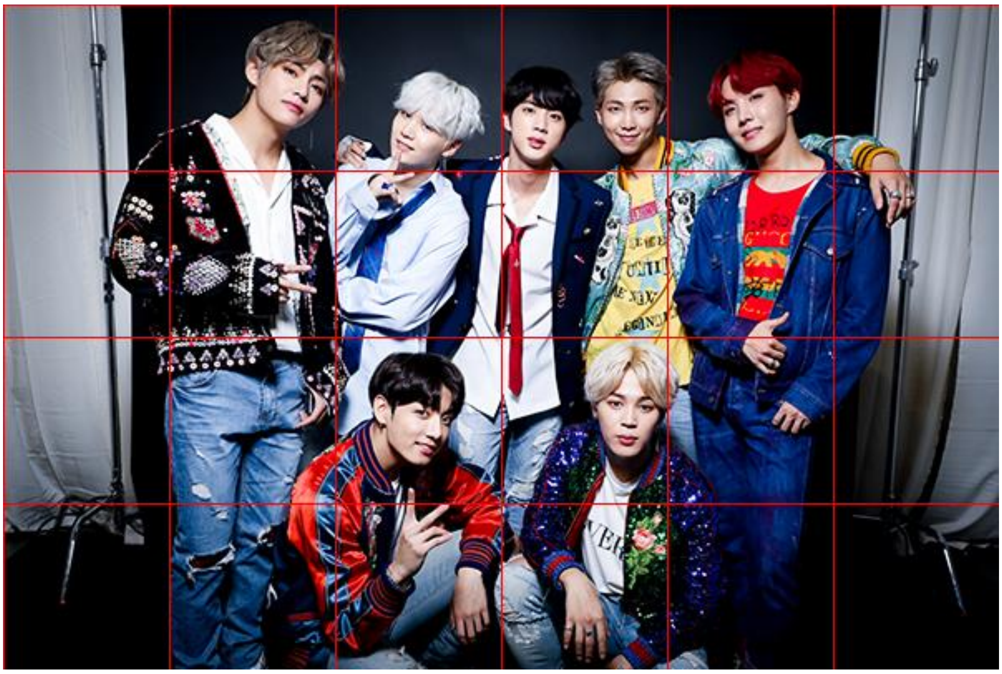
6. Sample output

This is one of sample BMP files with 600 x 400 pixels (`bts.bmp`).

If you run your program as follows, it will create a new file named “`btsout.bmp`”.

```
$ ./bmpgrid bts.bmp btsout.bmp 100
```

The resulting `btsout.bmp` file should look like this. The gap between grid lines is 100 pixels. Your output file `btsout.bmp` should be identical to the `bts-ans.bmp` file.



7. Hand in instructions

- Submit only the `bmpgrid.s` file to the submission site (http://sys.skku.edu).
- If your file contains any register names other than the allowed ones, your file will be rejected by the server.

8. 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*** during this semester. Please let us know the number of slip days you want to use in the QnA board in the submission site within 1 week after the deadline.
- Any attempt to copy others' work will result in heavy penalty (for both the copier and the originator). Don't take a risk.

Have fun!

Jin-Soo Kim
Systems Software & Architecture Laboratory
Dept. of Computer Science and Engineering
Seoul National University

Appendix. **GDB cheat sheet** (More info at https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf)

```
$ gdb ./bmpgrid
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
...
Reading symbols from ./bmpgrid...done.
(gdb) break bmp_grid
Breakpoint 1 at 0x400b34: file bmpgrid.s, line 31.
(gdb) run bts.bmp btsout.bmp 100
Starting program: /home/jinsoo/pa3/bmpgrid bts.bmp btsout.bmp 100
BMP file: bts.bmp (600 x 400 pixels, 24 bits/pixel)

Breakpoint 1, bmp_grid () at bmpgrid.s:31
31              movb    $0x55, (%rdi)
(gdb) list
26              #   gap    is in %rcx
27              #-------------------------------------------------------------
28
29              # --> FILL HERE <--
30
31              movb    $0x55, (%rdi)
32              movb    $0x88, 1(%rdi)
33              movb    $0xff, 2(%rdi)
34              ret
35
(gdb) print $rdi
$1 = 140737353244742
(gdb) print/x $rdi
$2 = 0x7ffff7f26046
(gdb) print $rsi
$3 = 600
(gdb) print $rdx
$4 = 400
(gdb) print $rcx
$5 = 100
(gdb) x/8b $rdi
0x7ffff7f26046: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) step
32              movb    $0x88, 1(%rdi)
(gdb) x/8b $rdi
0x7ffff7f26046: 0x55    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) step
33              movb    $0xff, 2(%rdi)
(gdb) x/8b $rdi
0x7ffff7f26046: 0x55    0x88    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) s
bmp_grid () at bmpgrid.s:34
34              ret
(gdb) x/8b $rdi
0x7ffff7f26046: 0x55    0x88    0xff    0x00    0x00    0x00    0x00    0x00
(gdb) x/8b $rdi+600
0x7ffff7f2629e: 0x65    0x49    0x2a    0x60    0x45    0x26    0x5d    0x42
(gdb)
```