

Laboratorio N° 2: usando MIPS para retocar imágenes

Objetivos:

- Escribir programas correctos en ensamblador MIPS.
- Comprender la interfaz de entrada/salida de MARS utilizando dispositivos mapeados en memoria (framebuffer o bitmap) y llamadas al sistema para el acceso a archivos.
- Comprobar la corrección del código por simple inspección visual del resultado.

Introducción

El programa MARS es un ensamblador de código MIPS32 que a su vez incorpora un simulador de MIPS y una IDE (integrated development environment) especialmente diseñado para la enseñanza de assembler MIPS¹. Una vez bajado en Linux lo ejecutan (deben tener la JVM instalada, y si no lean la página del MARS²) con:

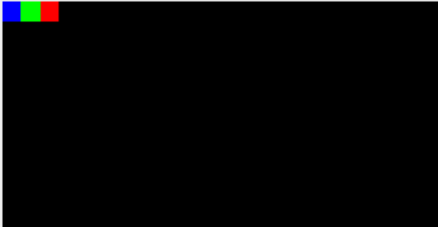
```
$ java -jar Mars4_4.jar
```

Dentro de las herramientas disponibles cuenta con un framebuffer³ configurable.

Este framebuffer, o bitmap como se denomina dentro de MARS, es una zona de memoria que empieza en el segmento de datos (.data) que usualmente está en la dirección de memoria 0x1001000⁴. Esta región de memoria se mapea a un rectángulo en la pantalla del ancho y alto configurado dentro de la herramienta. A los fines de este Laboratorio fijaremos el ancho y el alto en 256 pixeles⁵. Cada pixel de pantalla toma su color a partir de una palabra de 32 bits (4 bytes), donde el color final estará dada por la intensidad de 0 a 255 de los colores primarios mapeados dentro de la palabra de la siguiente manera:

[31:25]	[24:16]	[15:8]	[7:0]
NA	Red	Green	Blue

Como ejemplo, se muestra un pequeño programa y el borde superior izquierdo de la pantalla que produce (fijarse que la imagen está muy aumentada: cada cuadradito es un pixel!).

<pre>.data FB: .space 262144 # 256*256*4 .text main: la \$t0, FB li \$s0, 0x000000FF sw \$s0, 0(\$t0) sll \$s0, \$s0, 8 sw \$s0, 4(\$t0) sll \$s0, \$s0, 8 sw \$s0, 8(\$t0)</pre>	
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------

¹ Vollmar, K. "MARS: an education-oriented MIPS assembly language simulator." 2006.

<<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.85.2262>>

² MARS MIPS Simulator <<http://courses.missouristate.edu/kenvollmar/mars/>>

³ "Framebuffer - Wikipedia, the free encyclopedia." 2004. 17 May. 2013 <<http://en.wikipedia.org/wiki/Framebuffer>>

⁴ Configurar el modelo de memoria a "Default". Esta opción está en Settings->Memory Configuration de MARS 4.3.

⁵ "Pixel - Wikipedia, the free encyclopedia." 2011. 17 May. 2013 <<https://en.wikipedia.org/wiki/Pixel>>

La imagen

Lenna⁶ o Lena es una imagen de 512*512 pixeles utilizada desde 1973 para mostrar efectos y manipulaciones en imágenes digitales, particularmente en esquemas de compresión con pérdida de datos. Esta imagen puede obtenerse en diferentes formatos de almacenamiento de imágenes digitales. Nosotros utilizaremos la versión provista por Wikipedia, un archivo Portable Network Graphics (png).

Para poder operarlo dentro de nuestro framebuffer hay que convertirlo a un formato crudo (*raw format*) de 8 bits de profundidad de color por canal en formato RGBA (*red, green, blue, alpha*) y bajar la resolución a 256*256.

```
$ wget http://upload.wikimedia.org/wikipedia/en/2/24/Lenna.png
$ convert Lenna.png -resize 256x256 -depth 8 Lenna.rgb
$ ls -la Lenna.rgb
-rw-rw-r-- 1 user group 262144 Apr 22 17:45 Lenna.rgb
```

Para cargar esta imagen dentro del simulador MARS, utilizamos las *syscalls* open, read y close que se proveen. El programa que realiza la carga se muestra a continuación:

```
# Constants
.eqv WIDTH 256          # FrameBuffer Width
.eqv FB_LENGTH 262144  # 256*256*4

# Data Segment
.data
FB: .space FB_LENGTH    # Reserve FB_LENGTH Space in Data Segment in FB label
file: .asciiz "Lenna.rgb" # File name
.word 0

# Text Segment
.text
.globl main
main:
# Open File
li $v0, 13          # $v0 specifies the syscall type, where 13=open file
la $a0, file         # $a2 = address of the name of file to read
add $a1, $0, $0      # $a1=flags, 0 is O_RDONLY
add $a2, $0, $0      # $a2=mode, 0 is ignore
syscall             # Open File, $v0 stores file descriptor (fd)
move $s0, $v0        # store fd in $s0

# Read FB_LENGTH bytes from file, storing in framebuffer
li $v0, 14          # $v0 specifies the syscall type, where 14=read from file
move $a0, $s0        # $a0=file_descriptor
la $a1, FB           # $a1=address of input buffer (frame buffer)
li $a2, FB_LENGTH    # $a2=maximum numbers of characters to read
syscall              # Read From File, $v0 contains number of characters read or 0 if EOF

# Workaround Bitmap Display Bug
li $s5, 0           # i=0
move $t0, $a1        # $t0 is FB base address
loop: bge $s5, $a2, done # while i<FB_LENGTH
lw $s6, ($t0)         # load ith pixel in $s6
sw $s6, ($t0)         # store ith pixel
addiu $t0, $t0, 4      # step address fw
addiu $s5, $s5, 4      # i++
j loop
```

⁶ "Lenna - Wikipedia, the free encyclopedia." 2004. 17 May. 2013 <<http://en.wikipedia.org/wiki/Lenna>>

```
done:
    # Close File
    li $v0, 16          # $v0 specifies the syscall type, where 16=close file
    move $a0, $s0       # $a0=file_descriptor
    syscall             # Close File

    # Exit Gracefully
    li $v0, 10          # $v0 specifies the syscall type, where 10=exit
    syscall             # Exit
```

El código que está en **negrita** es para solucionar un **bug** del *bitmap display*, ya que el display solamente se refresca cuando hay una instrucción de **escritura en memoria** y no cuando internamente el simulador carga las direcciones de memoria a través de la *syscall* `read`⁷. El lazo simplemente recorre todos los píxeles, leyendo la memoria y sobre-escribiendo el mismo valor.

La Tarea

El grupo deberá manipular la imagen para lograr primero solucionar un defecto de colores, y luego transformar el espacio de colores a blanco y negro. Opcionalmente se puede realizar una transformación de simetría.

Se entrega un **código base** `load_lena.s` y la imagen en formato RGBA `Lenna.rgba` que el código carga en el framebuffer o bitmap de MARS.

Ejercicio 1

El resultado del código base muestra una Lenna azulina. Hay algún tipo de intercambio entre los canales de color del archivo y los de la pantalla.

Escriba un programa que recorra todos los puntos de la imagen intercambiando las componentes que sean necesarias dentro de cada pixel para que la imagen mostrada sea la correcta.

Para obtener la solución evitando realizar mezclas al azar hasta dar con el resultado correcto, piense en los formatos de almacenamiento, en posibles problemas de *endianess*, y en el formato de representación de colores dentro del bitmap.

Ejercicio 2

Una imagen monocromática en blanco y negro se obtiene cuando todos sus píxeles cumplen que la componente roja, azul y verde tienen la misma intensidad.

$$pixel[i].R = pixel[i].G = pixel[i].B$$

Realice una transformación de la imagen pixel a pixel para obtener una versión blanco y negro.

Ejercicio 3

Realice una **reflexión** de la imagen en alguno de estos tres ejes: horizontal, vertical o diagonal.

⁷ La persona que solucione este bug, genere un parche y lo envíe al desarrollador de MARS para incorporarlo a su siguiente versión, tiene aprobado este Laboratorio.

Como entregar

Deberá entregar dos códigos en un tarball⁸ con el nombre (respetando mayúsculas y minúsculas):

Lab2_apellidoNombre1_apellidoNombre2_apellidoNombre3.tar.gz

El tarball deberá contener tres archivos con la siguiente denominación: ejercicio1.s, ejercicio2.s, ejercicio3.s. Estos archivos deberán seguir el estilo de código⁹ del programa de ejemplo load_lena.s y deberán tener **comentarios** que ayuden a comprender la manera que solucionaron el problema. Además se deberá incluir un cuarto archivo en informe.pdf de dos o tres páginas en el que se explique cómo se resolvieron cada uno de los tres ejercicios.

Mail de entrega: entregas.odc@gmail.com

Fecha de entrega: **miércoles 11 de junio**

Calificación

El Laboratorio 2 se aprueba o desaprueba (A o N). Para aprobar los códigos deben realizar la tarea pedida que será corroborada, cargando, ensamblando y ejecutando el programa; y luego validando por simple inspección ocular.

Aunque no se califica: estilo de código, simpleza, elegancia, comentarios adecuados y velocidad; si los docentes que corrigen detectan un código que está muy por afuera de los parámetros aceptables, se podrá desaprobar el trabajo aunque éste sea funcionalmente correcto.

Ayudas

Ejercicio 1

En el código de ejemplo se muestra cómo recorrer todos los pixeles. Habrá que solamente operar sobre ellos. Se pueden utilizar máscaras con la operación and y constantes del tipo 0x00FF0000 para **aislar las intensidades de los colores primarios**. Pensar en rgba, abgr, argb.

Ejercicio 2

No hay una única forma de hacerlo. Piense en alternativas sencillas que tal vez no tomen las tres componentes de color.

Ejercicio 3

Intente resolver la simetría que, aunque no sea la más sencilla como resultado, el programa que la produzca si lo sea.

⁸ "tar (computing) - Wikipedia, the free encyclopedia." 2012. 17 May. 2013
<[http://en.wikipedia.org/wiki/Tar_\(computing\)](http://en.wikipedia.org/wiki/Tar_(computing))>

⁹ "Programming style - Wikipedia, the free encyclopedia." 2003. 17 May. 2013
<http://en.wikipedia.org/wiki/Programming_style>