Organización del Computador 2018

Trabajo de Laboratorio: VC FrameBuffer en RPi3

Objetivos

- Escribir programas en lenguaje ensamblador ARMv8.
- Comprender la interfaz de entrada/salida en memoria del microprocesador, utilizando una interfaz visual.
- Comprobar el principio de funcionamiento de una estructura FrameBuffer de video.

Condiciones

- Realizar el trabajo en grupos de exactamente *4 personas*.
- Entregar el trabajo por mail resuelto hasta el **jueves 14 de Junio** inclusive. (Los trabajos entregados después de esa fecha se consideran desaprobados.)
- Defender en una exposición oral el trabajo presentado. Deben presentarse todos los integrantes del grupo y responder preguntas acerca del desarrollo del mismo. La defensa puede realizarse en cualquier momento, una vez finalizado el desarrollo. La fecha límite para la defensa está prevista para el dia **viernes 15 de junio** en las horas del práctico.
- La aprobación del ejercicio 1 de este trabajo es requisito obligatorio para obtener la REGULARIDAD de la materia. La aprobación del ejercicio 2 es requisito obligatorio para obtener la PROMOCIÓN de la materia.

Formato de entrega

Deben entregar un tarball con el nombre: *TPRPi3_Apellido1_Apellido2_Apellido3_Apellido4.tar.gz*, respetando mayúsculas y minúsculas. El tarball debe contener dos archivos con la siguiente denominación: *ejercicio1.s, ejercicio2.s*. Estos archivos deben seguir el estilo de código del programa del ejemplo y contener comentarios que ayuden a comprender la manera en que solucionaron el problema. En el inicio de cada archivo describir en dos líneas lo que el código genera en la pantalla.

Los ejercicios deben enviarse por email a: gonzalovodanovic@gmail.com

Calificación

El Trabajo Práctico 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 validado por inspección ocular. Finalmente se deberá defender el trabajo oralmente.

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

Introducción

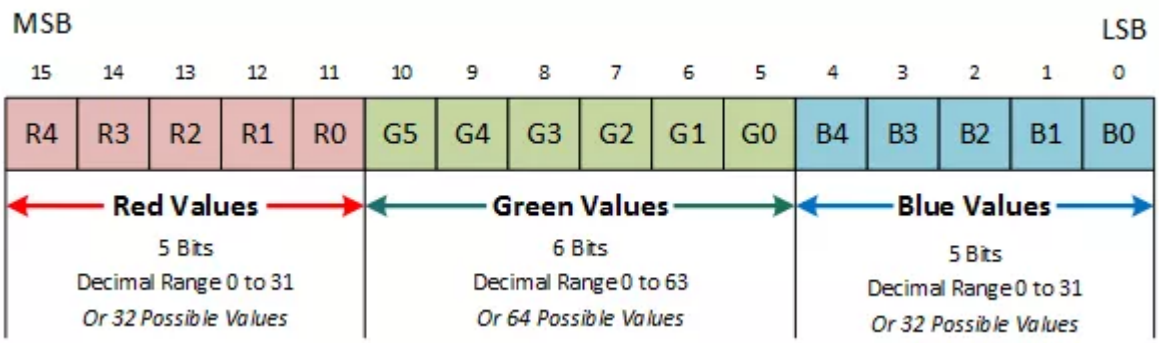
En términos generales, se denomina *framebuffer* al método de acceso de dispositivos gráficos de un sistema computacional, en el cual se representa cada uno de los píxeles de la pantalla como ubicaciones de una porción específica del mapa de memoria de acceso aleatorio (sistema de memoria principal).

En particular, la plataforma Raspberry Pi 3 soporta este método de manejo gráfico en su Video Core (VC). Para esto, hay que realizar una inicialización del VC por medio de un servicio implementado para comunicar el CPU con el VC llamado *mailbox*. Un mailbox es uno o varios registros ubicados en direcciones específicas del mapa de memoria (en zona de periféricos) cuyo contenido es "enviado" a los registros correspondientes de control/status de algún periférico del sistema. Este método simplifica las tareas de inicialización de hardware (escrito en código de bajo nivel), previos al proceso de carga de un Sistema Operativo en el sistema.

Luego del proceso de inicialización del VC via mailbox, los registros de control y status del VC pueden ser consultados en una estructura de memoria cuya ubicación puede ser definida (en rigor "virtualizada") por el usuario. Entre los parámetros que se pueden consultar se encuentra la dirección de memoria (puntero) donde se ubica el comienzo del FrameBuffer.

Para la realización del presente trabajo, se adjunta un código ejemplo donde se realizan todas las tareas de inicialización de VC explicadas anteriormente. Este código inicializa el FrameBuffer con la siguiente configuración::

- Tamaño en X = 512 píxeles
- Tamaño en Y = 512 píxeles
- Formato de color: RGB de 16 bits:



El color del píxel será el resultado de la combinación aditiva de los tres colores (rojo, verde y azul). De esta forma el color negro se representa con el valor 0x0000, el blanco con 0xFFFF, el rojo con 0xF800, el verde con 0x07E0, y el azul con 0x001F.

En base a esta configuración, el FrameBuffer queda organizado en palabras de 16 bits (2 bytes) cuyos valores establecen el color que tomará cada pixel de la pantalla. La palabra contenida en la primer posición del FrameBuffer determina el color del primer píxel, ubicado en el extremo superior izquierdo, incrementando el número de píxel hacia la derecha en eje X hasta llegar al pixel 511. De esta forma, el pixel 512 representa el primer pixel de la segunda línea. La estructura resultante se muestra en el siguiente diagrama

	Columna						
linea	0	1	2	…	509	510	511
0	0	1	2	..	509	510	511
1	512	513	514	…	1021	1022	1023
2	1024	1025	…				
…				…			
509					…		
510						…	
511							…

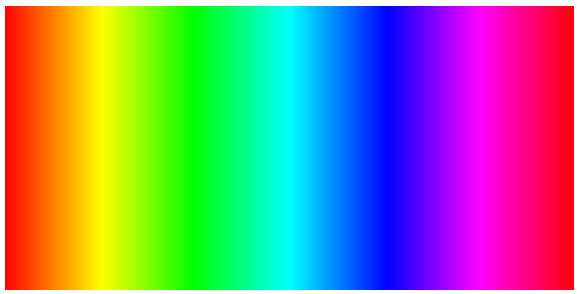
Debido a que la palabra que contiene el estado de cada pixel es de 16 bits, la dirección de memoria que contiene el estado del pixel N se calcula como:

Dirección = Dirección de inicio + (2 * N)

Dirección = Dirección de inicio + 2 * [x + (y * 512)]

Ejercicio 1

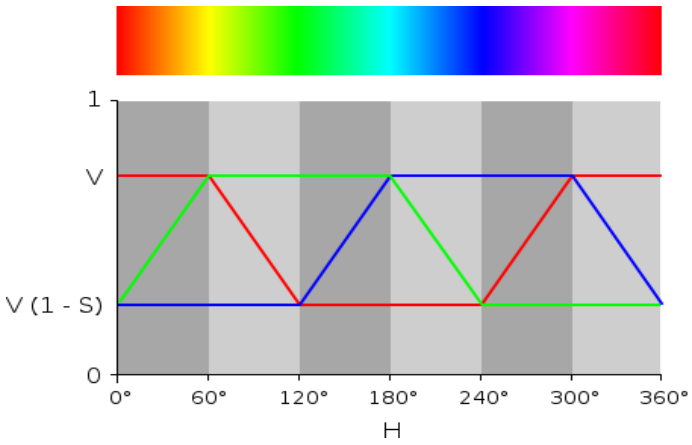
Escribir un programa en assembler ARMv8 sobre el código de ejemplo dado, que genere en la pantalla una paleta de colores continua, tal como se muestra en la siguiente figura:



La paleta de colores puede mostrar una distribución horizontal, vertical (como la del ejemplo), circular, diagonal, etc., sin embargo, las condiciones que debe cumplir el práctico son las siguientes:

- Que el patrón dibujado utilice toda la extensión de la pantalla.
- Que el patrón sea continuo (la transición de colores no debe mostrar límites visibles).
- Que al menos se utilice una vez la gama de colores completa (es decir que se llegue al mismo color del que se partió al menos una vez).

TIP: a continuación se muestra un gráfico con la función de la evolución de cada grupo de bits (R, G, B) para generar una paleta completa.



Ejercicio 2

Escribir un programa en assembler ARMv8 sobre el código de ejemplo dado que genere figuras definidas de distintos colores (a elección del grupo) con movimiento, cuya secuencia tenga una duración no menor que 10 segundos (pudiendo no concluir jamás). Las condiciones que debe cumplir el práctico son las siguientes:

- No puede tratarse de un patrón aleatorio.
- Se debe utilizar al menos 3 colores diferentes.
- El patron debe involucrar al menos dos figuras de distinta forma.

Se valorará especialmente la relación del efecto logrado vs. el tamaño del código generado.

NOTA IMPORTANTE: Tener en cuenta las diferencias existentes entre el set de instrucciones ARMv8 que se debe utilizar, con el set LEGv8 estudiado. Apoyarse en el Manual de Referencia: “ARMv8_Reference_Manual” que se acompaña como adjunto.