Organización del Computador 2022

Trabajo de Laboratorio: VC FrameBuffer en QEMU emulando una 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 en una plataforma Raspberry Pi 3 emulada.

Condiciones

- Realizar el trabajo en grupos de exactamente *3 personas*.
- Entregar el trabajo por el aula virtual resuelto hasta el **viernes 17 de Junio** inclusive. (El aula virtual solo aceptará trabajos hasta esta fecha, si no fue entregado a tiempo se lo considera desaprobado.)
- La aprobación del ejercicio 1 de este trabajo es requisito obligatorio para obtener la REGULARIDAD de la materia. La aprobación de ambos ejercicios es requisito obligatorio para obtener la PROMOCIÓN de la materia.

Formato de entrega

Deben entregar un tarball con el nombre:
TPQEMU_Apellido1_Apellido2.tar.gz, respetando mayúsculas y minúsculas. El tarball debe contener dos carpetas con la siguiente denominación: *ejercicio1, ejercicio2*. Estas carpetas deben contener, todos y únicamente, los archivos necesarios para la compilación (debe hacerse un make clean antes de comprimir). Con el agregado de un archivo de texto plano que describa en dos líneas lo que el código genera en la pantalla. En el comienzo de dicho archivo colocar los nombres completos y legajos de todos los integrantes del grupo.
El archivo principal (app.s) debe seguir el estilo de código del programa del ejemplo y contener comentarios que ayuden a comprender la manera en que solucionaron el problema.

Los ejercicios deben subirse al aula virtual al recurso llamado “Entrega Lab”.

Calificación

El Trabajo Práctico se Aprueba o Desaprueba (A o D). Para aprobar, los códigos deben realizar la tarea pedida, que será corroborada cargando, ensamblando y ejecutando el programa; y luego validado por inspección ocular.

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 vía 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 = 640 píxeles
- Tamaño en Y = 480 píxeles
- Formato de color: ARGB de 32 bits

El formato de colores se denomina ARGB, donde A es el alpha (transparencia), R es Red (rojo), G es Green (verde) y B es Blue (azul), el orden de la siglas determina el orden en que se ubican estos bits dentro de los 32 bits. Para cada uno de los componentes, el estándar ARGB le otorga 8 bits de resolución, es decir, existen 256 rojos, 256 verdes y 256 azules, más todas las combinaciones posibles entre estos tres colores.

RED[7:0]								GREEN[7:0]								BLUE[7:0]							
23							16	15							8	7							0

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 0x00, el blanco con 0xFFFFFF, el rojo con 0xFF0000, el verde con 0x0x00FF00, y el azul con 0x0000FF y, por ejemplo, 0xC71585 es "medium violet red" [ref].

En base a esta configuración, el FrameBuffer queda organizado en palabras de 32 bits (4 bytes) cuyos valores establecen el color que tomará cada pixel de la pantalla. La palabra contenida en la primera 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 639. De esta forma, el pixel 640 representa el primer pixel de la segunda línea. La estructura resultante se muestra en el siguiente diagrama

	Columna						
Fila	0	1	2	...	637	638	639
0	0	1	2	...	637	638	639
1	640	641	642	...	1277	1278	1279
2	1280	1281	….				
...				…			
477					…		
478						...	
479							...

Debido a que la palabra que contiene el estado de cada pixel es de 32 bits, la dirección de memoria que contiene el estado del pixel N se calcula como:

Dirección = Dirección de inicio + (4 * N)

Si se quisiera calcular la dirección de un píxel en función de las coordenadas x e y, la fórmula quedaría como:

Dirección = Dirección de inicio + 4 * [x + (y * 640)]

En la presente modalidad de cursado virtual, resulta imposible utilizar las placas raspberry pi 3 de la forma en que se hizo en años anteriores. Sin embargo, el programa QEMU, ya utilizado en la materia, tiene la capacidad de emular perfectamente este hardware. De tal forma en que el código generado para utilizar el framebuffer de la raspberry pi 3 puede correrse exactamente igual en este emulador. Es por esto que se utilizará el emulador QEMU para la realización de este laboratorio.

Ejercicio 1

Escribir un programa en assembler ARMv8 sobre el código de ejemplo dado, que genere una imagen diseñada por el grupo. El código debe generar la imagen, no siendo posible realizar un código que lea una imagen y únicamente la muestre.

La imagen debe ser **estática** y debe cumplir con los siguientes requisitos:

- Que utilice toda la extensión de la pantalla.
- No puede tratarse de un patrón aleatorio (excepto el fondo).
- Se debe utilizar al menos 3 colores diferentes.
- La imagen debe involucrar al menos dos figuras de distinta forma.
- La imagen debe poder explicarse en dos líneas de texto.

Ejercicio 2

Escribir un programa en assembler ARMv8 que genere figuras definidas de distintos colores (a elección del grupo) **con movimiento**, siendo posible reutilizar el código del ejercicio 1 pero considerando que la secuencia de movimiento debe tener una duración no menor a 10 segundos (pudiendo no concluir jamás). Las condiciones que debe cumplir el práctico son las siguientes:

- Que utilice toda la extensión de la pantalla.
- No puede tratarse de un patrón aleatorio (excepto el fondo).
- Se debe utilizar al menos 3 colores diferentes.
- La imagen debe involucrar al menos dos figuras de distinta forma.
- Las figuras deben moverse sobre un fondo no-continuo.
- La imagen debe poder explicarse en dos líneas de texto.

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.