Organización del Computador 2024

Trabajo de Laboratorio n°2:
VC FrameBuffer en QEMU emulando una RPi3

Objetivos

- Escribir programas en lenguaje ensamblador ARMv8.
- Comprender la interfaz de entrada/salida de un microprocesador ARM, utilizando una interfaz visual.
- Comprobar el principio de funcionamiento de una estructura FrameBuffer de video en una plataforma Raspberry Pi 3 emulada.
- Emular el comportamiento de los pines de entrada-salida de propósitos generales (GPIO) del microprocesador ARM usando el teclado de la computadora.

Condiciones

- Realizar el trabajo en grupos de exactamente *3 personas* (**deben inscribirse en el siguiente formulario: link antes del miércoles 22 de mayo**).
- Tienen tiempo de entregar el laboratorio hasta **las 8:00hs** del **viernes 07 de junio** (No se considerarán nuevos trabajos pasado este límite de tiempo. Los trabajos entregados por cualquier otro medio al establecido se consideran desaprobados).
- La aprobación del ejercicio 1 de este trabajo es requisito obligatorio para obtener la REGULARIDAD de la materia. La aprobación de los ejercicios 1 y 2 es requisito obligatorio para obtener la PROMOCIÓN de la materia.

Formato de entrega

Se debe trabajar en el repositorio git asignado por la cátedra. En el mismo habrá 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 del commit). Se debe agregar un archivo de texto plano que contenga una descripción en dos líneas del funcionamiento de cada ejercicio.

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.

Se va a considerar el último commit antes de la fecha y hora de entrega.

Calificación

El trabajo de laboratorio no lleva nota. Para aprobar, los códigos deben realizar la tarea pedida, que será corroborada cargando, ensamblando y ejecutando el programa; y luego validado por inspección ocular. También se va a verificar que todos los integrantes hayan trabajado en el repositorio.

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 al uso del framebuffer

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).

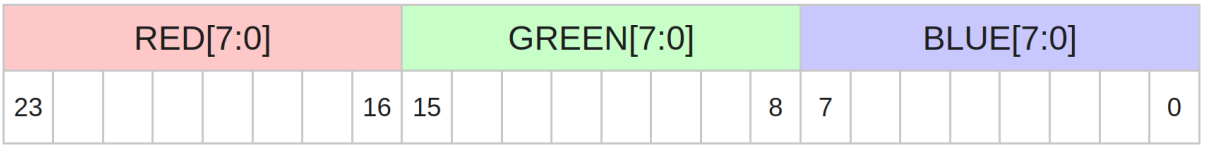
En particular, la plataforma Raspberry Pi 3 soporta este método de manejo gráfico en su Video Core (VC). Para esto, hay que realizar una inicialización del VC por medio de un servicio implementado para comunicar el CPU con el VC llamado *mailbox*. Un mailbox es uno o varios registros ubicados en direcciones 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.



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 0x00FF00, 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 píxel 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 píxel 639. De esta forma, el pixel 640 representa el primer píxel 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 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.

Introducción al uso de los GPIO (OPCIONAL)

- El pdf “BCM2835-ARM-Peripherals” contiene la documentación de los periféricos del procesador ARM de la Raspberry Pi (BCM2837). En ese archivo, buscar las especificaciones de los registros necesarios para la configuración y control de los GPIO. Importante: hay un diferencia en la dirección base de los registros del GPIO (GPFSEL0) reportadas en el manual respecto a las emuladas, debiéndose **reemplazar 0x7E200000 por 0x3F200000**.

- Para configurar un puerto como entrada (input) se debe utilizar el conjunto de registros **GPFSELn** (con n entre 0 a 5). Cada registro contiene un grupo de 3 bits que representa la configuración de cada GPIOx, llamados **FSELx**. (FSEL0 configura GPIO0; FSEL1 corresponde a GPIO1, y así sucesivamente). El valor de FSELx para que un GLIPOx sea **entrada** es el **“000”**.

- Luego de configurar un puerto como **entrada**, utilizar los registros de lectura **GPLEVn**, que contienen un bit por cada GPIO que se desea leer: “0” si la entrada está en

nivel bajo (pulsador liberado) y “1” si está en nivel alto (pulsador presionado). Recordar el uso de máscaras y operadores lógicos para leer el estado de un solo bit.

Ejercicio 1

Escribir un programa en assembler ARMv8 sobre el código de ejemplo dado, que genere una imagen diseñada por el grupo. Cada imagen debe ser **estática (sin movimiento)** y debe cumplir con los siguientes requisitos:

- Utilizar toda la extensión de la pantalla.
- No puede tratarse de un patrón aleatorio (excepto el fondo).
- Se deben utilizar al menos 3 colores diferentes.
- La imagen debe involucrar al menos dos figuras de distinta forma.
- La imagen debe generarse en el código, preferentemente mediante el uso de procedimientos para dibujar figuras básicas (círculos, rectángulos, líneas, triángulos, etc.) parametrizadas. No está permitido implementar un código que levante una imagen mapeada en memoria.
- La imagen debe poder explicarse en dos líneas de texto.

Opcional: Utilizar las teclas disponibles para cambiar alguna característica de la imagen (agregar o mover alguna figura, cambiar un color, cambiar alguna forma, etc.). Por ejemplo: al iniciar se pinta un paisaje de día en pantalla y al presionar la tecla “w” (GPIO 1) se cambia el color del cielo para mostrar el mismo paisaje de noche.

Ejercicio 2

Escribir un programa en assembler ARMv8 que genere una **animación** por pantalla, 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 ejercicio son:

- Utilizar toda la extensión de la pantalla.
- No puede tratarse de un patrón aleatorio (excepto el fondo).
- Se debe utilizar al menos 3 colores diferentes y dos figuras de distinta forma.
- Las imágenes deben generarse en el código, preferentemente mediante el uso de procedimientos para dibujar figuras básicas (círculos, rectángulos, líneas, triángulos, etc.) parametrizadas. No está permitido implementar un código que levante una secuencia de imágenes mapeadas en memoria.
- La animación debe contar una “historia corta”, que pueda explicarse en pocas líneas de texto.

Opcional: utilizar las teclas: “w”, “a”, “s”, “d” y la barra espaciadora (GPIOs 1, 2, 3, 4 y 5, respectivamente) para interactuar con las imágenes (cambiar forma, dirección del movimiento de una figura, etc.).

En todos los casos 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.