Trabajo de Laboratorio: VC FrameBuffer y GPIO en RPi3

Objetivos

- Escribir programas en lenguaje ensamblador ARMv8.
- Comprender la interfaz de entrada/salida en memoria del microprocesador, utilizando la plataforma Raspberry Pi y una interfaz visual.
- Interactuar con los GPIO del procesador a través de pulsadores y leds.
- Comprobar el principio de funcionamiento de una estructura framebuffer de video.

Condiciones

- Realizar el trabajo en grupos de *3 personas*.
- Entregar en el aula virtual dos carpetas (“ejercicio1” y “ejercicio2”) con los códigos necesarios para la compilación de cada ejercicio (debe hacerse un *make clean* antes de comprimir) y un breve informe del trabajo realizado, donde muestren imágenes de los resultados obtenidos. El archivo principal (app.s) debe seguir el estilo de código del programa de ejemplo y contener comentarios que ayuden a comprender la manera en que solucionaron el problema. La entrega debe realizarse antes de las 23:59hs del: **lunes 11 de noviembre (cátedra B)** ó **miércoles 13 de noviembre (cátedra A).**
- Presentar el trabajo el día **martes 12 de noviembre (cátedra B)** ó **jueves 14 de noviembre (cátedra A)** en las horas del teórico.
- La aprobación de este trabajo es requisito obligatorio para obtener la regularidad de la materia.
- Para obtener la promoción de la materia, el trabajo debe cumplir con la totalidad de las consignas y condiciones sugeridas.

Introducción al uso del framebuffer

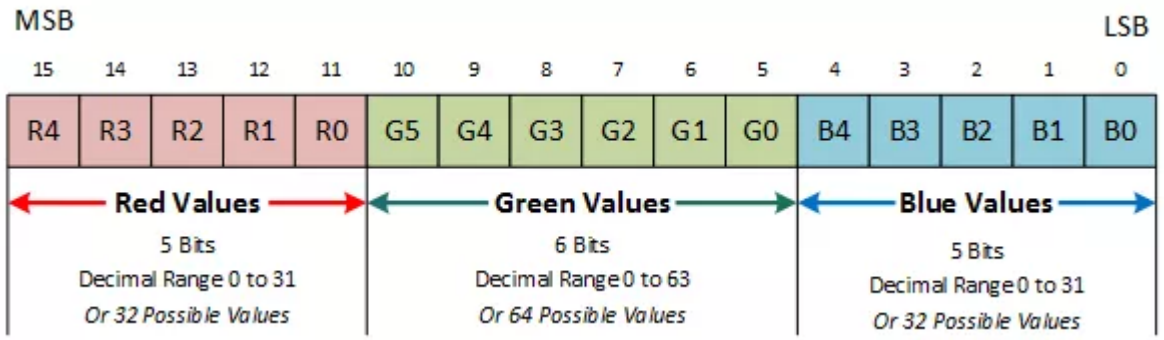
En términos generales, se denomina *framebuffer* al método de acceso de dispositivos gráficos de un sistema computacional, en el cual se representa cada uno de los píxeles de la pantalla como ubicaciones de una porción específica del mapa de memoria de acceso aleatorio (sistema de memoria principal).

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 = 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 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 pixel 511. De esta forma, el pixel 512 representa el primer píxel 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)

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 + 2 * [x + (y * 512)]

Introducción al uso de los GPIO

- El pdf “BCM2835-ARM-Peripherals” contiene la documentación de los periféricos del procesador ARM de la Raspberry Pi 3 Model B+ (BCM2837). En ese archivo, buscar las direcciones de los registros necesarios para la configuración y control de los GPIO. Importante: hay un error en la dirección base de los registros del GPIO (GPFSEL0), se debe **reemplazar 0x7E200000 por 0x3F200000**.

- Para habilitar un puerto y configurarlo como entrada o salida: Configurar los **GPFSELn** correspondientes a los GPIO que se desee utilizar, siendo: “000” = entrada y “001” = salida. Notar que cada registro GPFSEL permite configurar 10 puertos (GPFSEL0 del 0 al 9, GPFSEL1 del 10 al 19…) y que se utilizan 3 bits para la configuración de cada GPIO.

- Luego de configurar un puerto como **entrada**, utilizar el registro de lectura **GPLEVn**, que devuelve en el bit correspondiente al 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 para leer el estado de un bit.

- Luego de configurar un puerto como **salida**, utilizar **GPCLRn** colocando “1” en el bit correspondiente al GPIO que se desea poner a nivel lógico “0”. De esta manera se **enciende** el led conectado a ese GPIO (leds activos por bajo).

- De manera análoga, para **apagar** un led utilizar **GPSETn**. Colocar un “1” en el bit correspondiente al GPIO que se desea poner a nivel lógico “1”, de esta manera se **apaga** el led conectado a ese GPIO.

Metodología de trabajo

1) Cada grupo recibirá una RaspberryPi 3B, una SD y una placa con pulsadores y leds. Para trabajar necesitarán, además, un cargador de celular con cable micro USB, un monitor y un cable HDMI.

2) La placa debe conectarse en los pines 1 a 12 del terminal J8 de la Raspberry. De esta forma, los pulsadores se conectan a los GPIO 14, 15, 17 y 18, y los leds a los GPIO 2 y 3. Luego conectar el monitor al puerto HDMI.

3) Descargar de Moodle la carpeta “boot_SD”, extraer todos los archivos que contiene y copiarlos en la SD.

4) Descargar de Moodle la carpeta “TemplateUCC”. El archivo **main.s** contiene la inicialización del framebuffer, los GPIO y la llamada a app.s (NO EDITAR).

5) Para escribir el código assembly: Abrir el archivo **app.s** e introducir allí el código de su ejercicio. Para que el código resulte más organizado, hay un archivo **gpio.s** donde pueden colocar las funciones de manejo de los GPIO, respetando la estructura del código de ejemplo dado. Tener en cuenta:

    a) Crear un bucle infinito para que el programa se ejecute mientras la Raspberry Pi permanezca encendida.

b) Dejar una línea vacía al final del código. El toolchain espera esta línea vacía para asegurarse de que el archivo realmente terminó. Si no se coloca, aparece una advertencia cuando se corre el ensamblador.

6) Para crear la imagen de su sistema operativo: abrir el terminal de la computadora, dirigirse al directorio de la carpeta “TemplateUCC”, escribir “make” y presionar enter. Si no ocurre ningún error, dentro de la carpeta deben generarse los archivos: main.elf, main.hex, main.list, main.o, memory_map.txt y **kernel8.img**, que es la imagen de su sistema operativo.

7) Para correr el programa en la Raspberry: Copiar el archivo kernel8.img en la SD, reemplazando la imagen actual. Luego montar la SD en la Raspberry y, finalmente, alimentar la placa desde el USB.

Ejercicio 1

Escribir un programa en assembler ARMv8 sobre el código de ejemplo dado, que genere en la pantalla una imagen estática que **se asemeje lo más posible** a la siguiente:

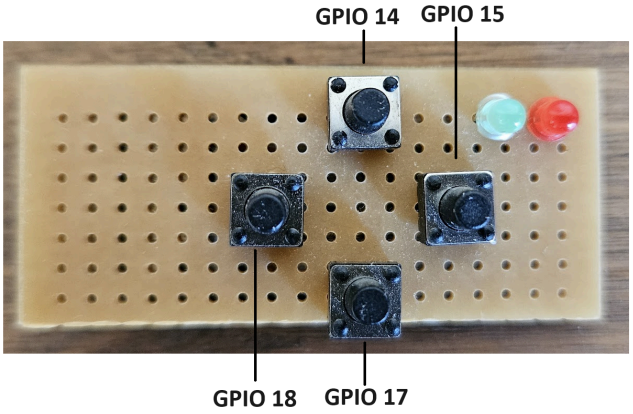
Condiciones mínimas:

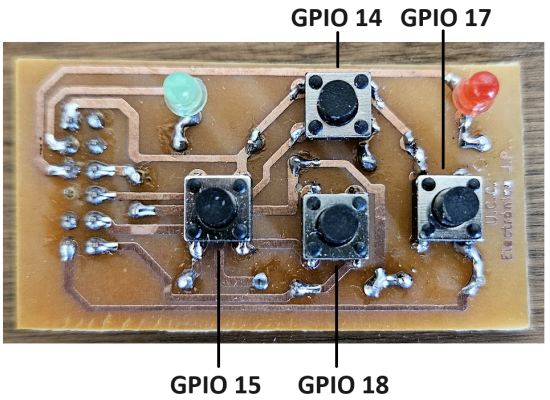
- La imagen debe utilizar toda la extensión de la pantalla (512x512 píxeles).
- Las figuras deben conformarse a partir del código generado. Es deseable que se modelen procedimientos para cada operación (dibujar, rellenar, borrar, etc.) o figuras (círculo, rectángulo, triángulo, etc.) y luego utilizarlos. No está permitido guardar una figura en memoria y cargarla en pantalla por código.
- Aproximar lo mejor posible los colores del personaje y el fondo.

Ejercicio 2

Escribir un programa en assembler ARMv8 sobre el código de ejemplo dado, que genere un laberinto que ocupe toda la pantalla y un jugador que se desplace hacia arriba, abajo, izquierda y derecha cuando se presione el pulsador correspondiente (ver mapeo de GPIOs en la figura).

Utilizar los leds para indicar eventos en el juego. Por ejemplo: al iniciar el juego encender el led rojo (GPIO3) y mantenerlo encendido hasta terminar el recorrido del laberinto. Cuando se alcance la meta, encender el led verde (GPIO2) durante unos segundos.





Las condiciones que debe cumplir el ejercicio son las siguientes:

- El recorrido completo del laberinto (desde la largada hasta la meta) debe requerir múltiples cambios de dirección. Se valorará positivamente para acceder a la promoción, que el fondo del laberinto (por donde se desplaza el jugador) no sea liso.
- Utilizar como mínimo 3 colores diferentes.
- El diseño debe involucrar al menos dos figuras de distinta forma (ej: rectángulos para la confección del laberinto y un círculo como jugador).
- El diseño de la o las pantallas de juego, del jugador y todas las características que deseen agregar (premios, agujeros de gusano, etc.) serán valorados en la calificación obtenida.
- También se valorará la relación del diseño 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” adjunto.