Organización del Computador 2019

# Trabajo de Laboratorio: 1-bit audio con RPi3

## Objetivo

Utilizando los conceptos aprendidos de programación en assembler, aplicados a la programación de procesadores ARMv8, se propone utilizar una plataforma real (Raspberry Pi 3/3+) y hacer uso de bloques periféricos (GPIO) para generar un respuesta audible controlada con un bit que en 1 sube el cono del parlante y con 0 baja el cono del parlante.

## Condiciones

- Realizar el trabajo en grupos de *3 a 4 personas*.
- Entregar el trabajo por mail resuelto hasta el **lunes 10 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 día **miércoles 12 de Junio**.
- 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 pocas líneas el efecto sonoro que genera el código.

Los ejercicios deben enviarse por email a: `delfinavelez@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. 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**

La variedad de tonos que nuestro oído es capaz de percibir es muy elevada, estando acotada tan sólo por los límites de sensibilidad de nuestro sistema auditivo, normalmente desde los 20Hz hasta los 20kHz. La gama usual de frecuencias de los sonidos musicales es considerablemente más pequeña que la gama audible, siendo el tono más alto de un piano el de frecuencia 13.186kHz, valor que podemos considerar como límite superior de los tonos fundamentales.

La nota de referencia del sistema musical occidental es actualmente el “LA” a 440Hz (porque la Organización Internacional de Estandarización así lo fijó en 1955). A partir de esta nota de referencia se calculan las frecuencias que deben tener el resto de notas para que todos los instrumentos suenen correctamente afinados.

Las frecuencias de las notas musicales en la octava correspondiente al LA a 440Hz son:

| Nota en español | DO | RE | MI | FA | SOL | LA | SI |
|---|---|---|---|---|---|---|---|
| Nota en inglés | C | D | E | F | G | A | B |
| Frecuencia [Hz] | 261.63 | 293.66 | 329.63 | 349.23 | 392.00 | 440.00 | 493.88 |

**Ejercicio 1 (para regularizar)**

Escribir un programa en assembler ARMv8, sobre el código de ejemplo dado, que genere una secuencia infinita de las notas DO - RE - MI - FA - SOL - LA - SI en la octava dada en la introducción. Cada nota debe reproducirse durante aproximadamente 5 segundos.

**TIP:** Para medir la frecuencia de la nota que se está generando utilizar el Afinador Cromático Gismart (en el modo “afinador cromático”).

**Ejercicio 2 (para promocionar)**

Escribir un programa en assembler ARMv8, sobre el código de ejemplo dado, que genere un efecto sonoro “llamativo” durante al menos 30 segundos.

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

Hay mucho hecho y escrito respecto a cómo generar **1-bit audio**. Varias microcomputadoras de principios de los 80’s traían exactamente el mismo soporte que este laboratorio, un 1 en un puerto levantaba el altavoz, un 0 lo bajaba. Los ejemplos clásicos son Apple ][ (1977) y ZX Spectrum (1982). Más adelante se incorporaron PSG (programmable sound generators), circuitos integrados específicos que generaban el audio, ejemplos típicos son el SID de la Commodore 64 y el AY-3-8910 de la MSX. Finalmente con la introducción de Paula de Commodore Amiga, se inició la posibilidad de tocar samples de 8 o más bits de profundidad en varios canales. Actualmente esta es la forma de generar sonido en todas las computadoras, un procesador especial a través de acceso directo a memoria (DMA) reproduce samples en la salida de audio.

En este artículo, Kenneth McAlpine explora parte de la historia y las técnicas utilizadas por las computadoras con audio de 1-bit de profundidad.

- Kenneth B. McAlpine, *The Sound of 1-bit: Technical Constraint and Musical Creativity on the 48k Sinclair ZX Spectrum*, The Italian Journal of Game Studies, 6/2017.

Las restricciones respecto a la capacidad de hacer sonido jamás fueron un impedimento. La Altair 8800, una de las primeras microcomputadoras accesible al bolsillo de la clase media, pudo entonar Daisy Bell como lo hizo la IBM 7094 en 1961. Mucho antes, en 1951, un tal Alan Turing ayudó un tal Christopher Strachey a registrar la primera melodía interpretada por una computadora.

- Steve Dompier, *Music of a sort*, Altair 8800, DrDobbs, 1976.

La música 1-bit sigue viva, como en el proyecto 1-bit Symphony de Tristan Perish, o en las demos extremas que combinan muchas Apple ][ o muchas ZX Spectrum para hacer sinfonías.

En la demoscene local se destacan los Pungas de Villa Martelli (PVM), un grupo que crea arte a partir de tecnologías obsoletas. Aunque no producen piezas de 1-bit audio, los PVM recrean a través de PSG (programmable sound generators) de Commodore 64 y Nintendo Gameboy todo un repertorio vernáculo, como el Cancionero Argentino Vol 1 y 2.

Tal vez la mejor referencia sea el libro recientemente editado

- Kenneth B. McAlpine, "Bits and Pieces. A History of Chiptunes", OUP, 2018

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

**Antes de comenzar**

1) Cada grupo deberá contar con una computadora, un lector de SD o microSD y un cargador de celular tipo micro USB.

2) Verificar que la SD esté en formato FAT32 y con los archivos de booteo. En caso de no ser así, formatearla, descargar los archivos del enlace y copiarlos en el directorio raíz de la tarjeta.

3) Instalar el **GNU Aarch64 toolchain**. Quienes usan Ubuntu, lo pueden bajar del gestor de paquetes con el comando:

```
$ sudo apt install gcc-aarch64-linux-gnu
```

4) Descargar el template del enlace.

**Para escribir el código assembler**

- Abrir el archivo `main.s`. Respetar las líneas de encabezado (directivas para el ensamblador).
- En el PDF "*BCM2835-ARM-Peripherals*" se detalla el funcionamiento del bloque GPIO. Identificar 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, esto quiere decir que se deben ignorar las direcciones que figuran y deben ser reemplazadas por:
    - `Peripheral Base Address = 0x3F000000`
    - `GPIO Offset Address = 0x200000`

De esta forma la dirección del primer registro del bloque de los GPIO (correspondiente al registro `GPFSEL0`) está dada por:

- `Peripheral Base Address + GPIO Offset Address = 0x3F200000`.

- El programa debe habilitar el/los puerto a utilizar y configurarlos como salida. Para ello se debe identificar los campos **`FSELn`** (donde **"n"** corresponde al número del GPIO que se desea configurar) de 3 bits c/u, sabiendo qué : 000 = entrada y 001 = salida. Notar que cada registro **`GPFSELx`** permite configurar 10 puertos (`GPFSEL0` del 0 al 9, `GPFSEL1` del 10 al 19…).
- Encender una salida: utilizar el campo **`SETn`**, dentro del registro **`GPSETx`**, colocando 1 en los bits correspondientes a los GPIO que se desea encender.
- Apagar una salida: utilizar el campo **`CLRn`**, dentro del registro **`GPCLRx`**, colocando 1 en los bits correspondientes a los GPIO que se desea apagar.
- Crear un **bucle infinito** para que el programa se ejecute mientras la Raspberry Pi permanezca encendida.
- 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.

**Para crear la imagen**

Abrir el terminal de la computadora y dirigirse al directorio de la carpeta “template”, escribir el comando `make` y presionar enter. Si no ocurre ningún error, la salida del comando debería ser la siguiente:

```
aarch64-linux-gnu-as --warn --fatal-warnings main.s -o main.o
aarch64-linux-gnu-ld main.o -T memmap -o main.elf -M > memory_map.txt
aarch64-linux-gnu-objdump -D main.elf > main.list
aarch64-linux-gnu-objcopy main.elf -O ihex main.hex
aarch64-linux-gnu-objcopy main.elf -O binary kernel8.img
```

Como puede verse, en dicho directorio se generan varios archivos, siendo los principales: `main.list`, `main.o` y `kernel8.img`.

**Para correr el programa en la Raspberry Pi**

- Copiar el archivo `kernel8.img` en la SD, reemplazando la imagen actual.
- Poner la SD en la Raspberry.
- Conectar los módulos periféricos en los puertos de GPIO correspondientes.
- Alimentar la placa desde el USB con el cargador de celular.