

# Informática: Arquitectura de Ordenadores

## Curso 2020-2021

### Práctica 3

Esta práctica tiene como objetivo presentar el simulador QtARMSim<sup>1</sup>, estudiar la representación de datos en el mismo y realizar con un ejercicio introductorio de ARM/Thumb. QtARMSim es un simulador gráfico de la arquitectura ARM/Thumb sencillo y fácil de usar, diseñado específicamente para cursos introductorios de Arquitectura de Ordenadores. Se distribuye bajo la licencia GPL v3+. En la página web <https://pypi.org/project/qtarmsim/> están disponibles las instrucciones de instalación de este simulador para las diferentes plataformas de escritorio. También se ha dispuesto en Moovi una máquina virtual (para VirtualBox) con una versión de Ubuntu con el simulador instalado.

## 1. Presentación de QtARMSim

1. Para ejecutar QtARMSim, abre un terminal y escribe `qtarmsim`. Aparecerá una nueva ventana (ver Fig. 1). Cambia el tamaño y la ubicación de la ventana como desees. La ventana principal tiene cuatro zonas diferentes: (i) Registros; (ii) Editor/Desensamblar; (iii) Memoria, y (iv) Mensajes/Volcado de memoria/Pantalla LCD.

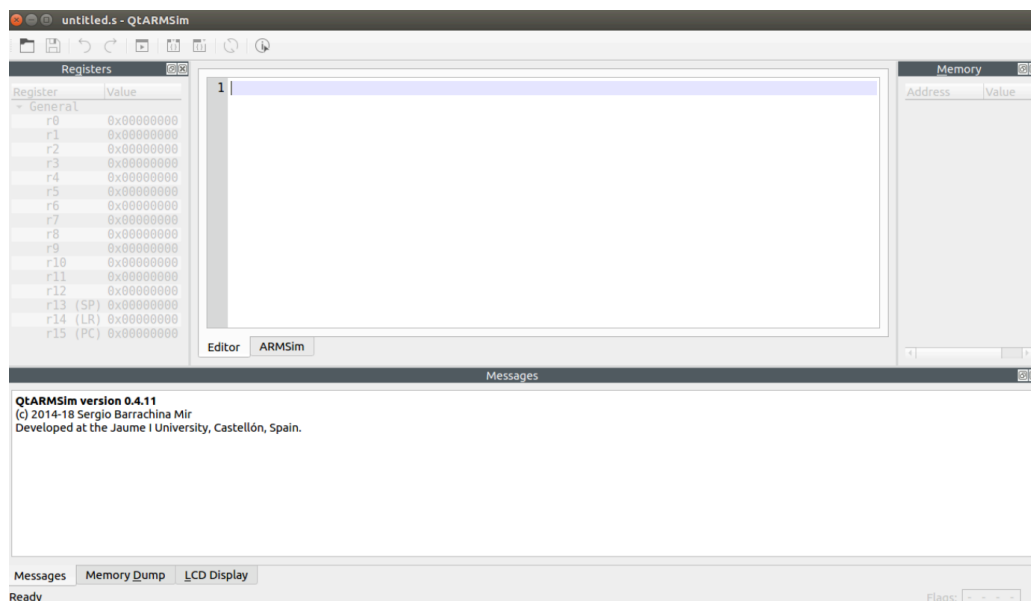


Figura 1: Ventana principal de QtARMSim.

2. Selecciona la pestaña Editor de la parte inferior del área de edición, en caso de que no esté ya seleccionada. Copia el programa ARM/Thumb que aparece a continuación. Una vez editado, guárdalo en un archivo usando esa opción de la barra de menús (📁) o el atajo de teclado **CTRL + S**.

```
.data
ins: .asciz "Hola, mundo!"
.balign 4
nc: .space 4
```

---

<sup>1</sup>Introducción a la arquitectura de computadores con QtARMSim y Arduino, por Sergio Barrachina Mir, Maribel Castillo Catalán, Germán Fabregat Llueca y Juan Carlos Fernández Fer. (<http://lorca.act.uji.es/project/qtarmsim>)

```

        .text
        ldr r0, =ins      @ dirección cadena de entrada
        mov r1, #0        @ contador a 0

loop:   ldrb r2, [r0]
        cmp r2, #0
        beq endc

        add r1, r1, #1    @ contador de incrementos
        add r0, r0, #1    @ puntero de incremento
        b loop

endc:   ldr r0, =nc
        str r1, [r0]      @ almacenamos el resultado
        wfi

```



3. Analiza el programa anterior y describe lo que hace.
4. Selecciona la pestaña ARMSim en la parte inferior de la zona de edición. El simulador intentará ensamblar automáticamente el programa. En caso de que haya errores, se notificará en el área de mensajes que está debajo de la zona de edición. Si es así, vuelve a la pestaña del Editor, corrige esos errores, e inténtalo nuevamente.
5. Una vez que el programa esté libre de errores, al seleccionar la pestaña ARMSim se ensamblará automáticamente el código fuente y QtARMSim pasará al modo de emulación. En este modo, cada línea se corresponde con una instrucción del nivel de máquina convencional almacenada en la memoria. La información que se muestra se obtiene mediante un proceso llamado desensamblado, y consiste en lo siguiente (de izquierda a derecha):
  - La dirección de memoria en la que se almacena la instrucción.
  - La instrucción expresada en hexadecimal.
  - La instrucción desensamblada, es decir, la instrucción en lenguaje ensamblador.
  - La línea de texto original del código fuente que produjo la instrucción durante el proceso de ensamblado.


Por ejemplo, la primera línea del programa anterior:

```
[0x00180000] 0x4805 ldr r0, [pc, #20]; 7 ldr r0, =ins @ dirección cadena de entrada
```

indica que:

- La instrucción está almacenada en la dirección de memoria 0x00180000.
  - La instrucción expresada en hexadecimal es 0x4805.
  - La instrucción desensamblada es **ldr r0, [pc, #20]**. Ten en cuenta que las instrucciones desensambladas se representan utilizando la sintaxis ARM completa (es decir, no solo el subconjunto Thumb).
  - La instrucción se generó a partir de la línea número 7 del código fuente original, cuyo contenido era: **ldr r0, =ins @dirección cadena de entrada**.
6. Ahora vamos a ejecutar el programa. Para esto, selecciona el icono correspondiente de la barra de menús (■), la opción del menú Run, o el atajo de teclado **CTRL + F11**.  
 Cuando el programa finalice, verás que los registros **r0**, **r1**, **r2** y **r15**, y la posición de memoria **0x20070010** tienen fondo azul y estén en negrita. Esto se debe a que el emulador resalta los registros y las posiciones de memoria que se modificaron como consecuencia de la ejecución del programa. Ten en cuenta que **r15** es el contador del programa o PC, por lo que se modifica internamente para apuntar siempre a la siguiente instrucción a ejecutar.

7. Explora el resto de las pestañas. Pon especial atención en la pestaña de volcado de memoria. ¿El programa hace lo previsto en el paso 3 anterior?
8. Restablece la simulación (pulsas el icono  o presiona la tecla F4) y vuelve a ejecutar el programa paso a paso. Para eso, selecciona la opción del menú de ejecución paso a paso () o presiona F5. Observa cómo cambian los valores de los registros y las posiciones de memoria durante la ejecución, y también los indicadores del procesador N, Z, C y V.
9. Un punto de ruptura (breakpoint) es una marca para que el simulador detenga la ejecución antes de ejecutar la instrucción donde se colocó dicho punto. Veamos cómo funcionan.

Restablece el simulador usando F4. Haz clic en el margen de la ventana de simulación junto a la instrucción de máquina `ldr r0, =nc`. Aparecerá un octógono rojo () que indica que se definió un punto de ruptura. Ejecuta el programa con CTRL + F11. ¿Qué ocurre?

Para eliminar un punto de ruptura ya definido, simplemente haz clic en su octógono rojo.

## 2. Representación de datos en QtARMSim

1. El siguiente código inicializa los registros r0 a r3 con 4 valores numéricos en decimal, hexadecimal, octal y binario, respectivamente. Copia el programa en QtARMSim, cambia al modo de simulación y responde las siguientes preguntas.

```
.text
mov r0, #66
mov r1, #0x42
mov r2, #0102
mov r3, #0b1000010
wfi
```

- a) ¿Cómo se muestran los números anteriores en la pestaña ARMSim? ¿Están en la misma base que en el código original? ¿En qué base están representados?
  - b) Ejecuta el programa paso a paso. ¿Qué números se almacenan en los registros r0 a r3?
2. Edita el código presentado a continuación, ensambla, y responde las preguntas siguientes.

```
.data
word1: .word 11
word2: .word 0x11
word3: .word 011
word4: .word 0b11

.text
stop: wfi
```

Al situar el cursor sobre el contenido de una posición de memoria en el área de volcado de memoria, aparecerá una ventana amarilla con información sobre el contenido de dicha posición de memoria.

- a) Identifica las posiciones de memoria donde se almacenaron los datos anteriores en el panel de volcado de memoria. Identifica los cuatro valores en hexadecimal.
  - b) ¿En qué direcciones se almacenan los cuatro valores? ¿Por qué estas direcciones de memoria son múltiplos de cuatro en lugar de ser direcciones consecutivas?
  - c) ¿Cuáles son los valores de las etiquetas word1, word2, word3 y word4.
3. Ensambla el siguiente código:

```
.data
wrds: .word 11, 0x11, 011, 0b11

.text
stop: wfi
```

- a) ¿Hay algún cambio en los valores almacenados en la memoria con respecto a los almacenados en el caso del programa anterior? ¿Están en el mismo lugar?

4. Ensambla el siguiente código:

```
.data
bys: .byte 0x18, 0x19, 0x1a, 0x1b

.text
stop: wfi
```

- a) ¿Qué valores se almacenaron en la memoria? ¿En qué posiciones?  
b) ¿Cuál es el valor de la etiqueta **bys**?

5. Ahora ensambla el siguiente código:

```
.data
strg: .ascii "abác"
byte: .byte 0xff

.text
stop: wfi
```

- a) ¿Qué rango de posiciones de memoria se reservó para la variable etiquetada como **strg**? Los caracteres en QtARMSim se representan como caracteres UTF-8 (es decir, se pueden representar con 1 o 2 bytes, dependiendo del carácter que sea). ¿Cuál es el código UTF-8 de la letra “a”? ¿Y de la letra “á”?  
b) ¿Cuántos bytes se reservan para la cadena **abác**?  
c) ¿Cuál es el valor de la etiqueta **byte**?

6. Ensambla el siguiente código:

```
.data
b1: .hword 0x11
gap: .space 4
b2: .byte 0x22
bign: .word 0x33445566

.text
stop: wfi
```

- a) ¿Cuántas posiciones de memoria se reservan para la variable **gap**?  
b) ¿Podrían leerse o escribirse los cuatro bytes utilizados por la variable **gap** como si fueran una palabra? ¿Por qué?  
c) ¿Cuál es el valor de la etiqueta **b1**? ¿Y de la etiqueta **b2**?  
d) ¿Cuál es el valor de la etiqueta **bign**? ¿Podrían leerse o escribirse los cuatro bytes que comienzan en la dirección **bign** como si fueran una palabra? ¿Por qué?  
e) Agrega una directiva **.balign** al código anterior para que la variable etiquetada como **bign** se alinee con un límite de palabra (es decir, con una dirección múltiplo de cuatro).

### 3. Realización de un programa

Dado un número representado por sus dígitos en ASCII (y terminado por un 0), escriba un programa en ARM/Thumb que calcule cuantos dígitos “5” contiene. El número estará almacenado a partir de la posición etiquetada como **numero**, y el programa dejará el resultado en la posición etiquetada como **numde5**.

```

        .data
numero: .asciz "12534543"
        .balign 4
numde5: .space 4

        .text
main:    ...

```

## Atajos de teclado de QtARMSim

F1	Ayuda
Shift + F1	¿Qué es ...?
F3	Restablecer la configuración predeterminada
F4	Restablecer la simulación
F5	<i>Step into</i> (Paso a paso)
F6	<i>Step over</i> (saltar subprograma)
CTRL + F11	Ejecutar
CTRL + N	Nuevo archivo
CTRL + O	Abrir archivo
CTRL + S	Guardar archivo
CTRL + Q	Salir de QtARMSim
ALT+P	Preferencias