



NOMBRE: SOLUCIÓN

EJERCICIOS PARA RESOLVER

1) (21pts) Relacione el valor de la columna izquierda con el equivalente de la derecha, de acuerdo con sus conocimientos sobre la arquitectura ARM de 32-bits estudiada en clase:

- | | |
|--|--|
| a) Contador de Programa. | d. SUB PC, LR, #0. |
| b) Registros, Caché, RAM, Disco. | c. Referido a la descripción estructural del procesador. |
| c) Microarquitectura. | e. Pasar argumentos a las funciones. |
| d) MOV PC, LR. | f. Mismo bus para acceder a instrucciones y datos. |
| e) R0-R3. | g. 0x0000. |
| f) Característica de Arq. Von Neumann. | a. Dirección de la siguiente instrucción a ejecutar. |
| g) ANDEQ R0, R0, R0, LSL #0. | b. Jerarquía de memoria. |

2) (21pts) Afirmaciones para completar relacionadas con la arquitectura ARM de 32-bits estudiada en clase:

- La arquitectura **HARVARD** permite el acceso simultáneo a la memoria de instrucciones y datos.
- La instrucción LDR R0, =0xABCD, es en realidad un LDR R0, [REGISTRO, #CTE], donde REGISTRO corresponde a **PC**.
- La instrucción MOV R0, #0xFFFFF00 se puede reemplazar por la instrucción **MVN R0, #0xFF**.
- Las instrucciones de carga y almacenamiento en memoria son típicas de la filosofía de diseño de instrucciones **RISC**.
- Cuando un procesador con dos niveles de memoria caché, L1 y L2, solicita leer la memoria, accederá al nivel **L2** siempre que el dato no se encuentre disponible en el nivel **L1**.
- Si la instrucción BL F1 está en la dirección 0x0C y la primera instrucción de la función F1 está en la dirección 0x100, el valor inmediato que se codifica en el campo imm24 de la instrucción BL F1 es **0x3B**.
- El valor inmediato de la instrucción MOV R0, #0xC0000004 se puede reemplazar por el valor 0x13 siempre que se le indique a la CPU rotar éste último 2 posiciones hacia la derecha.

3) (12pts) Para los dos esquemas de memoria mostrados en la siguiente figura, Little-Endian y Big-Endian, indique como quedaría almacenado el valor de 64-bits 0xFA102C3E_1B2E3C12 en la dirección de memoria 0x20 en cada uno de ellos:

Little-Endian				Dirección	Big-Endian			
12	3C	2E	1B		FA	10	2C	3E
Byte 20 Byte 21 Byte 22 Byte 23				0000_0020	Byte 20 Byte 21 Byte 22 Byte 23			
3E	2C	10	FA	0000_0024	1B	2E	3C	12
Byte 24 Byte 25 Byte 26 Byte 27					Byte 24 Byte 25 Byte 26 Byte 27			

4) (16 pts) Una función en ensamblador para ARM denominada **£2** recibe seis parámetros, los cuatro primeros en los registros R0-R3 y los últimos dos en el *stack*: el quinto en la dirección `[SP]` y el sexto en la dirección `[SP, #4]`. El retorno se realiza en el registro R0. Esta función (**£2**) es llamada dentro de otra función (**£1**), la cual deberá pasarle los seis argumentos. En el siguiente fragmento de código ya se han cargado los primeros cuatro argumentos en los registros R0 y R3, pero hace falta cargar los argumentos cinco y seis en el *stack*, los cuales se encuentran en los registros R7 y R8, respectivamente. Indique las instrucciones que se



encargan de llevar los argumentos cinco y seis al *stack*, junto con la apropiada manipulación del registro SP.
Restricción: sólo pueden emplearse las instrucciones ADD, LDR y STR.

F1:

...
BL F2
...

Líneas antes de la instrucción BL F2

STR R8, [SP, #-4]!

STR R7, [SP, #-4]!

Líneas luego de la instrucción BL F2

ADD SP, SP, #8

- 5) (30pts) Dado el prototipo de una función en lenguaje C: `int32_t findValue(int32_t array[], uint32_t size, int32_t value)`, escriba el cuerpo de dicha función empleando lenguaje ensamblador para el procesador ARM estudiado en clase, que retorne el índice de la primera posición del vector **array** de tamaño **size** que sea igual a **value**. Si ninguna posición dentro del vector **array** es igual a **value**, entonces se deberá retornar **-1**. Para los parámetros de entrada, emplee los registros **R0**, **R1** y **R2**, para recibir **array**, **size** y **value**, respectivamente. Para el retorno de la función, utilice el registro **R0**. Emplee el *stack* para guardar en memoria el valor de los registros preservados que vaya a emplear en la función, si es requerido.

```
findValue:
    MOV R3, #0                // Inicializa index a 0
Loop:
    CMP R3, R1                // Si index >= size, no encontrado
    BHS EndLoop
    LDR R12, [R0, R3, LSL #2]  // Cargar array[index] en R12
    CMP R12, R2                // Compara array[index] con value
    BEQ EndFindValue          // Si iguales, retornar index
    ADD R3, R3, #1            // Si no, intentar sgte. index
    B Loop
EndLoop:
    MOV R3, #-1               // Value no encontrado
EndFindValue:
    MOV R0, R3                // Retornar index o -1 en R0
    MOV PC, LR
```