

Código ensamblador: ARMv4

José Eduardo Campos Salazar
Escuela Ingeniería en Computadores
Tecnológico de Costa Rica
Cartago, Costa Rica
josc4188@gmail.com

Resumen—This report presents the development and implementation of three problems using ARM Assembly language, executed and tested in a digital design environment. The first problem modifies an array in memory based on a conditional operation with a constant. The second problem calculates the factorial of a given number using a loop-based algorithm. The third problem demonstrates input handling by modifying a counter based on specific key inputs. Each case was validated with simulation results, illustrating the correct execution of memory operations, arithmetic instructions, and control flow. The report highlights the importance of low-level programming in understanding hardware-software interaction.

Index Terms—ARM Assembly, factorial calculation, conditional operations, memory access, input handling, digital design, low-level programming.

I. INTRODUCCIÓN

1. Investigue los tipos de instrucción que tiene ARMv4. Nombre dos ejemplos de cada tipo y un posible uso.

Los tipos de instrucciones que existen son:

- a) Investigue los tipos de instrucción que tiene ARMv4. Nombre dos ejemplos de cada tipo y un posible uso.

Los tipos de instrucciones que existen en ARMv4 son: [1]

- 1) **Data processing instructions:** permiten realizar operaciones aritméticas y lógicas sobre valores en los registros. Es decir, modifican valores de datos.

Ejemplos: ADD, SUB, CMP

Uso:

```
ADD r0, r1, r2 ; r0 = r1 + r2
```

- 2) **Data movement instructions:** mueven datos entre los registros y la memoria en ARM. Las tres formas básicas son:

a' *Single register load and store instructions:* permiten transferir un solo dato (byte o palabra de 32 bits) entre registros y memoria.

b' *Multiple register load and store instructions:* permiten transferencias más eficientes de bloques de datos completos.

c' *Single register swap instruction:* permiten intercambiar el contenido de un registro con el de una ubicación de memoria.

Ejemplos: LDR, STR, SWP

Uso:

```
LDR r0, [r1] ; r0 := mem32[r1]
```

- 3) **Control flow instructions:** no procesan ni mueven datos directamente, sino que determinan la siguiente instrucción a ejecutar, permitiendo condicionales, bucles y subrutinas.

Ejemplos: BEQ, BNE, BL

Uso:

```
CMP r0, #5 ; if (r0 != 5) {  
BE BYPASS  
ADD r1, r1, r0 ; r1 = r1 + r0  
SUB r1, r1, r2 ; r1 = r1 - r2  
BYPASS...
```

- 4) **Special instructions:** incluyen instrucciones que controlan aspectos específicos del procesador, como el acceso a registros de estado, cambio de modo del procesador o manipulación de interrupciones. Estas instrucciones son clave para el manejo de excepciones, interrupciones y control del sistema.

Ejemplos: MRS, MSR

Uso:

```
MRS r0, CPSR ; Copia el estado  
del procesador a r0  
MSR CPSR, r1 ; Modifica el estado con  
el contenido de r1
```

2. Explique el set de registros que tiene ARM. ¿Cual es el contenido de cada uno?

El set de registros de ARM contiene 16 registros de acceso directo, de r0 a r15. el CPSR contiene el condition flags y el mode bits actual, registro del r0 a r13 son de propósito general del r14 a r15 tiene los siguientes usos:

- a) Link Register: El registro r14 es usado como el "subroutine Link Register"(LR) este recibe una copia de r15 cuando una "branch with link"(BL) es ejecutada

- b) **Program counter**: El registro r15 mantiene el PC los bits[1:0] de r15 con indefinidos y son ignorados, bits[31:2] contienen el PC

Por convención el r13 es usado como el Stack Pointer (SP) [2]

3. Explique el funcionamiento del branch.

Cuando un programa necesita desviarse de su secuencia de instrucciones default, se usa una instrucción de control de flujo; se usa para modificar el program counter (PC) explícitamente. La más simple se denomina branches, o jumps. [1]

4. ¿Cómo se implementa un condicional (if /else) en ARMv4?

Este se implementa con un condition code register, que es un registro de propósito especial en el procesador. Siempre que se ejecuta una instrucción que habilita al condition code register, este registra si el resultado es 0, negativo, overflow, produce un acarreo, etc. Entonces, la instrucción de branch es controlada por el estado del condition code register. [1]

5. Explique el procedimiento para transformar el código en ensamblador a binario. Investigue herramientas que realizan el proceso.

Para traducir instrucciones a su representación binaria en ARM, se utiliza el **A32 instruction set encoding**, el cual se compone de los siguientes campos: [3]

- a) **Cond** (4 bits): determina la condición de ejecución de la instrucción.

Ejemplos:

- 0000 → EQ (igual)
- 0001 → NE (no igual)

- b) **Bits [27:26]**: normalmente son 00 para instrucciones de procesamiento de datos.

- c) **I** (bit 25): indica si Operand2 es un valor inmediato o un registro.

Ejemplos:

- I = 0 → Operand2 = r2
- I = 1 → Operand2 = #5

- d) **Opcode** (bits [24:21]): define qué operación se va a realizar.

Ejemplos:

- 0100 → ADD
- 0010 → SUB

- e) **S** (bit 20): indica si se deben actualizar las banderas (Flags).

Si S = 1, las banderas del programa son actualizadas.

- f) **Rn** (bits [19:16]): registro que contiene el primer operando.

Ejemplo: ADD r0, r1, r2 → Rn = r1

- g) **Rd** (bits [15:12]): registro donde se almacena el resultado.

Ejemplo: ADD r0, r1, r2 → Rd = r0

- h) **Operand2** (bits [11:0]): segundo operando de la instrucción.

Ejemplo: ADD r0, r1, r2 → Operand2 = r2

Por lo tanto, una instrucción como ADD r0, r1, r2 se traduce a binario como:

```
1110 00 0 0100 0 0001 0000 0000 0000
                                0010
```

Algunas herramientas utilizadas para este proceso incluyen:

- **GNU Assembler (GAS)**
- **ARM Compiler**

6. Explique la diferencia de little endian y big endian. ¿Cuál es su implicación cuando pasan las instrucciones a lenguaje máquina (binario)? [4]

- a) **Big-endian (BE)**: Guarda el byte mas significativo de primero, quiere decir que el primer byte es el mas grande

- b) **Little-endian (LE)**: Guarda el byte menos significativo de primero

Suponiendo que las instrucciones se van a guardar en memoria, dependen de cual se use puede detener problemas leyendo las instrucciones o por ejemplo si dos maquinas con diferente endian tratan de comunicarse van a tener problemas con la interpretación.

II. DESARROLLO

II-A. Problema 1

Este programa recorre un arreglo de 10 enteros y modifica cada elemento dependiendo de una constante. Si el valor del elemento es mayor o igual a la constante, se multiplica por ella; si es menor, se le suma dicha constante. Este comportamiento se basa en el siguiente algoritmo:

Pasos del algoritmo:

1. Inicializar el índice en 0.
2. Verificar si el índice es igual a 10.
3. Si lo es, finalizar el programa.
4. Si no lo es, leer el valor del arreglo en la posición del índice.
5. Comparar el valor con una constante predefinida.
6. Si el valor es mayor o igual a la constante, multiplicarlo por la constante y almacenarlo de nuevo en el arreglo.
7. Si es menor, sumarle la constante y almacenar el nuevo valor en el arreglo.
8. Incrementar el índice y repetir el proceso desde el paso 2.

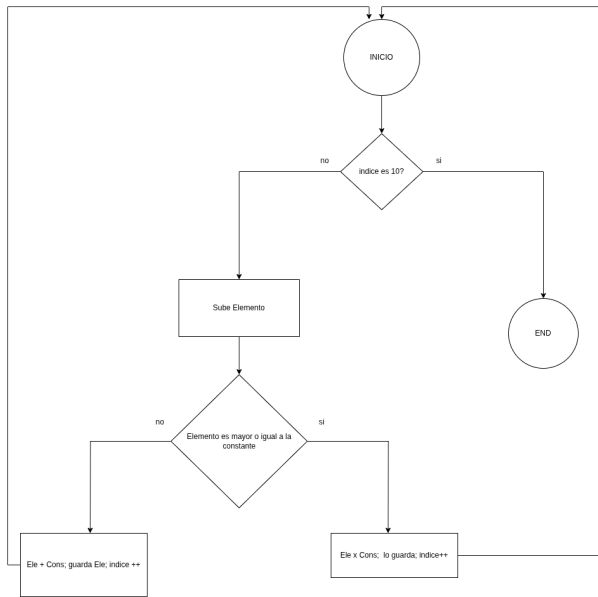


Figura 1. Algoritmo de recorrido y modificación de arreglo

Bloques funcionales utilizados:

- **Contador (índice):** Registra la posición actual dentro del arreglo.
- **Memoria de datos:** Almacena el arreglo de 10 enteros.
- **Comparador:** Compara el valor del elemento con la constante.
- **Unidad aritmética:** Realiza operaciones de suma o multiplicación según el caso.
- **Control de flujo:** Maneja las bifurcaciones del algoritmo y el bucle principal.

II-B. Problema 2

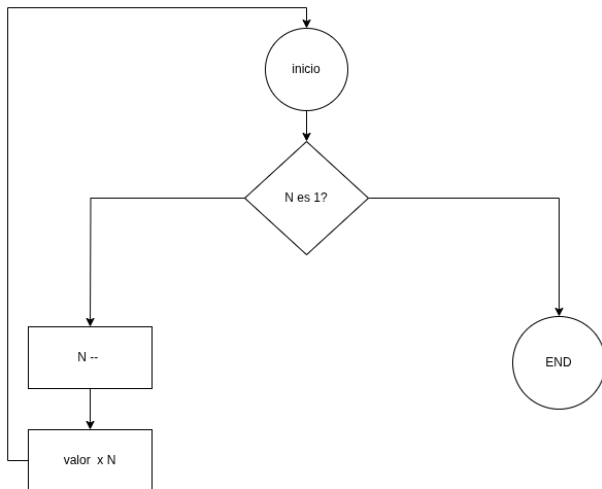


Figura 2. Operación de resta en la FPGA

Se realizó un algoritmo para procesar el factorial de un número X . El algoritmo toma un valor inicial X y lo multiplica sucesivamente por $(X - 1)$ hasta llegar a 1. El resultado se almacena en el registro $r2$.

Pasos del algoritmo:

1. Inicializar el registro $r0$ con el valor de entrada X .
2. Copiar el valor de $r0$ al registro $r2$, que almacenará el resultado del factorial.
3. Comparar si el valor de $r0$ es menor o igual a 1.
4. Si lo es, terminar el algoritmo.
5. Si no, decrementar el valor de $r0$ en 1.
6. Multiplicar el valor actual de $r0$ por $r2$, y almacenar el resultado nuevamente en $r2$.
7. Repetir el proceso desde el paso 3.

Bloques funcionales utilizados:

- **Registro de entrada:** Para cargar el valor inicial X (registro $r0$).
- **Comparador:** Compara si el valor actual es igual o menor a 1 (`cmp r0, #1`).
- **Unidad aritmética:**
 - Para la resta (`sub r0, r0, #1`).
 - Para la multiplicación (`mul r2, r2, r0`).
- **Registro acumulador:** Almacena el resultado parcial en $r2$.
- **Control de flujo:** Ciclo mediante instrucción `b LOOP` y condición de parada con `ble END`.

II-C. Problema 3

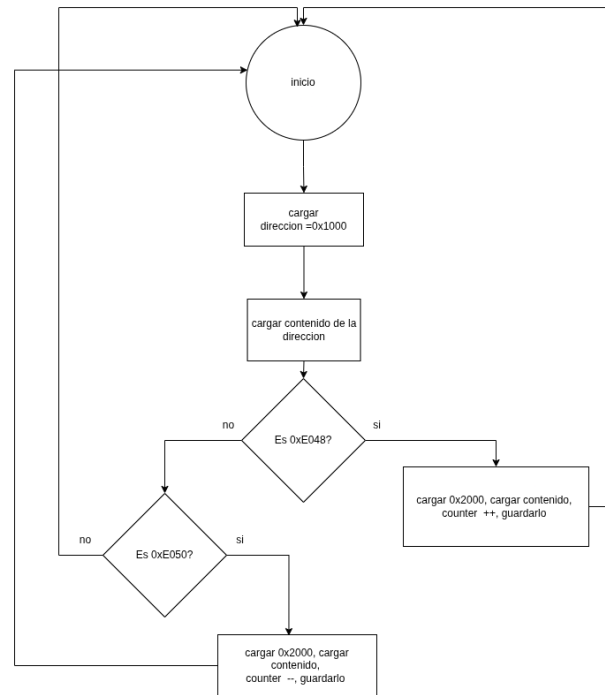


Figura 3. Control de contador mediante botones en la FPGA

Este programa permite incrementar o decrementar un contador dependiendo de si se presiona el botón arriba o abajo en la FPGA o teclado. Se usa lectura de memoria mapeada para detectar la tecla presionada y modificar el valor del contador en consecuencia.

Pasos del algoritmo:

1. Cargar la dirección de memoria donde está almacenado el contador (0×2000).
2. Reiniciar el valor del contador a 0.
3. Cargar los códigos asociados a los botones arriba y abajo.
4. Entrar en un ciclo infinito donde:
 - a) Se lee la dirección de la tecla presionada (0×1000).
 - b) Se compara el valor leído con el código de la tecla arriba.
 - c) Si coincide, se incrementa el contador.
 - d) Si no, se compara con el código de la tecla abajo.
 - e) Si coincide, se decrementa el contador.
 - f) Si no coincide con ninguno, se vuelve a leer.

Bloques funcionales utilizados:

- **Bloque de entrada:**
 - Dirección de la tecla presionada (0×1000).
 - Códigos de teclas arriba ($0 \times E048$) y abajo ($0 \times E050$).
- **Comparador:** Compara el valor leído con los códigos de las teclas.
- **Contador:**
 - Dirección de contador en 0×2000 .
 - Incrementa o decrementa según la tecla presionada.
- **Unidad aritmética:** Realiza las operaciones de suma y resta sobre el contador.
- **Control de flujo:** Usa saltos condicionales para actuar según la tecla presionada y volver al ciclo.

III. RESULTADOS

III-A. Problema 1

En este problema, se procesó un arreglo de 10 enteros donde cada elemento fue modificado según una constante (5). Si el valor era mayor o igual a la constante, se multiplicó por ella; si era menor, se le sumó.

r0	00001000	
r1	0000e048	
r2	0000e048	
r3	0000e050	
r4	00000005	
r5	00002000	
r6	00000001	
r7	00000000	
r8	00000000	
r9	00000000	
r10	00000000	
r11	00000000	
r12	00000000	
sp	00000000	
lr	00000000	
pc	0000003c	
cpsr	600001d3	NZCVI SVC
spsr	00000000	NZCVI ?

Figura 4. Suma de la constante al primer elemento del arreglo (posición 0)

r0	5	
r1	72	
r2	25	
r3	0	
r4	5	
r5	0	
r6	0	
r7	0	
r8	0	
r9	0	
r10	0	
r11	0	
r12	0	
sp	0	
lr	0	
pc	12	
cpsr	1610613203	NZCVI SVC
spsr	0	NZCVI ?

Figura 5. Multiplicación del elemento en la posición 4 por la constante

00000030	3758228626	3883999488	3800039425	3942645746!.....
00000040	3942645758	72	6	7H.....
00000050	8	9	25	30H.....
00000060	35	40	45	50(.....2.....
00000070	2863311530	2863311530	2863311530	2863311530
00000080	2863311530	2863311530	2863311530	2863311530
00000090	2863311530	2863311530	2863311530	2863311530

Figura 6. Resultado final del arreglo después del procesamiento

III-B. Problema 2

En el segundo problema, se implementó un algoritmo para calcular el factorial de un número ingresado. El resultado fue almacenado en el registro r2.

r0	1	
r1	72	
r2	24	
r3	0	
r4	5	
r5	0	
r6	0	
r7	0	
r8	0	
r9	0	
r10	0	
r11	0	
r12	0	
sp	0	
lr	0	
pc	28	
cpsr	1610613203	NZCVI SVC
spsr	0	NZCVI ?

Figura 7. Resultado del factorial de 4 ($4! = 24$) almacenado en r2

r0	1	
r1	72	
r2	120	
r3	0	
r4	5	
r5	0	
r6	0	
r7	0	
r8	0	
r9	0	
r10	0	
r11	0	
r12	0	
sp	0	
lr	0	
pc	28	
cpsr	1610613203	NZCVI SVC
spsr	0	NZCVI ?

Figura 8. Resultado del factorial de 5 ($5! = 120$) almacenado en r2

r0	00001000	
r1	0000e048	
r2	0000e048	
r3	0000e050	
r4	00000005	
r5	00002000	
r6	00000002	
r7	00000000	
r8	00000000	
r9	00000000	
r10	00000000	
r11	00000000	
r12	00000000	
sp	00000000	
lr	00000000	
pc	0000003c	
cpsr	600001d3	NZCVI SVC
spsr	00000000	NZCVI ?

Figura 11. Segunda iteración: nuevo incremento del contador

r0	1	
r1	72	
r2	40320	
r3	0	
r4	5	
r5	0	
r6	0	
r7	0	
r8	0	
r9	0	
r10	0	
r11	0	
r12	0	
sp	0	
lr	0	
pc	28	
cpsr	1610613203	NZCVI SVC
spsr	0	NZCVI ?

Figura 9. Resultado del factorial de 8 ($8! = 40320$) almacenado en r2

r0	00001000	
r1	0000e048	
r2	0000e048	
r3	0000e050	
r4	00000005	
r5	00002000	
r6	00000003	
r7	00000000	
r8	00000000	
r9	00000000	
r10	00000000	
r11	00000000	
r12	00000000	
sp	00000000	
lr	00000000	
pc	0000003c	
cpsr	600001d3	NZCVI SVC
spsr	00000000	NZCVI ?

Figura 12. Tercera iteración: el contador continúa incrementando

III-C. Problema 3

En este caso, se desarrolló un programa que incrementa o decrementa un contador dependiendo de la tecla presionada. Se utilizaron los códigos de teclas para determinar la acción a realizar.

r0	00001000	
r1	0000e048	
r2	0000e048	
r3	0000e050	
r4	00000005	
r5	00002000	
r6	00000001	
r7	00000000	
r8	00000000	
r9	00000000	
r10	00000000	
r11	00000000	
r12	00000000	
sp	00000000	
lr	00000000	
pc	0000003c	
cpsr	600001d3	NZCVI SVC
spsr	00000000	NZCVI ?

Figura 10. Primera iteración: incremento del contador

r0	00001000	
r1	aaaaaaaa	
r2	0000e048	
r3	0000e050	
r4	00000005	
r5	00002000	
r6	00000003	
r7	00000000	
r8	00000000	
r9	00000000	
r10	00000000	
r11	00000000	
r12	00000000	
sp	00000000	
lr	00000000	
pc	00000018	
cpsr	a00001d3	NZCVI SVC
spsr	00000000	NZCVI ?

Figura 13. Cuarta iteración: se presiona una tecla no válida, el contador no cambia

r0	00001000	
r1	0000e050	
r2	0000e048	
r3	0000e050	
r4	00000005	
r5	00002000	
r6	00000002	
r7	00000000	
r8	00000000	
r9	00000000	
r10	00000000	
r11	00000000	
r12	00000000	
sp	00000000	
lr	00000000	
pc	0000004c	
cpsr	600001d3	NZCVI SVC
spsr	00000000	NZCVI ?

Figura 14. Quinta iteración: decremento del contador

IV. CONCLUSIONES

- El desarrollo de los tres problemas permitió aplicar de forma práctica los conocimientos sobre programación en lenguaje ensamblador, así como la manipulación directa de registros, memoria y estructuras de control.
- En el **Problema 1**, se logró modificar correctamente un arreglo en memoria utilizando operaciones condicionales, reforzando el uso de comparaciones y acceso indexado a memoria, así como la importancia de optimizar ciclos en ARM.
- El **Problema 2** demostró la eficiencia del lenguaje ensamblador para realizar cálculos matemáticos como el factorial, trabajando con bucles controlados por registros. Este ejercicio reforzó el uso de registros acumuladores y control de flujo con saltos.
- En el **Problema 3**, se integró el uso de entradas externas (teclado) para modificar un contador, lo cual evidenció la interacción entre software y hardware a través de direcciones de memoria mapeadas. Esto resultó útil para entender cómo leer entradas del sistema y tomar decisiones en tiempo real.
- En general, la práctica consolidó conceptos fundamentales del lenguaje ensamblador, como la lógica condicional, las operaciones aritméticas a bajo nivel y el acceso a memoria. Además, se reforzó el pensamiento lógico y secuencial requerido para el desarrollo en sistemas embebidos.

REFERENCIAS

- [1] S. Furber, *ARM System-on-Chip Architecture*, 2nd ed. Addison-Wesley, 2000.
- [2] ARM Ltd., "Arm7tdmi technical reference manual - programmer's model: Registers," 2000, accessed: 2025-05-10. [Online]. Available: <https://developer.arm.com/documentation/ddi0210/c/Programmer-s-Model/Registers?lang=en>
- [3] Arm Ltd., *A32 Instructions by Encoding*, Arm Developer, mar 2025, accessed: 2025-05-10. [Online]. Available: <https://developer.arm.com/documentation/ddi0597/2025-03/A32-Instructions-by-Encoding?lang=en>
- [4] GeeksforGeeks. (2023) Little and big endian mystery. Accessed: 2025-05-10. [Online]. Available: <https://www.geeksforgeeks.org/little-and-big-endian-mystery/>