



# Universidad Nacional Autónoma de México



## Facultad de Ingeniería

### Integrantes:

Espinoza Matamoros Percival Ulises - 320025561

Flores Colin Victor Jaziel - 320266083

Lara Hernandez Angel Husiel - 320060829

## Laboratorio de Microcomputadoras

Grupo: 06 - Semestre: 2026-2

### Practica 2:

Programación en Ensamblador. Direccionamiento  
Indirecto

### Profesor:

Ing. Moises Melendez Reyes

### Fecha de Entrega:

8 de Marzo del 2026



## 1. Objetivo:

Programar las variantes del modo de direccionamiento indirecto existentes para los procesadores ARM.

## Actividad 1

Escribir, comentar, compilar y comprobar el funcionamiento del siguiente programa.

### Propuesta de solución

#### Desarrollo

Listing 1: Código de la Actividad 1

```
1  /* ACTIVIDAD 1: Direccionamiento con desplazamiento a la izquierda (LSL)
2   Objetivo: Guardar el valor del contador en un arreglo de 16
3   posiciones.
4 */
5 .data
6   i: .skip 64           @ Reserva 64 bytes de memoria (16
7   palabras de 4 bytes)
8
9 .text
10 .global main          @ Define 'main' como global para Code
11   ::Blocks / Linker
12
13 main:
14   ldr r1, =i            @ Carga en R1 la dirección base de la
15   variable 'i'
16   mov r2, #0             @ R2 será nuestro contador,
17   inicializado en 0
18
19 loop:
20   cmp r2, #16           @ Compara el contador R2 con el límite
21   de 16
```



```
16    beq fin          @ Si R2 es igual a 16 (Branch if EQUAL)
     ) , salta a la etiqueta 'fin'

17
18    add r3, r1, r2, LSL #2      @ R3 = R1 + (R2 desplazado a la
     izquierda 2 bits). Equivale a R3 = R1 + (R2 * 4). Calcula la
     dirección en memoria.

19    str r2, [r3]           @ Guarda el valor actual del contador
     (R2) en la dirección de memoria apuntada por R3
20    add r2, r2, #1         @ Incrementa el contador (R2 = R2 + 1)
21    b loop               @ Salto incondicional (Branch) de
     regreso a 'loop'

22
23 fin:
24    MOV R7, #1           @ Carga la llamada al sistema sys_exit
     (1)
25    SVC 0                @ Ejecuta la llamada para salir
     limpiamente al SO
```

## Análisis de resultados



## Actividad 2

Modificar el programa de la actividad 1, para usar el direccionamiento indexado de su preferencia con el doble de datos.

### Propuesta de solución

#### Desarrollo

Listing 2: Código de la Actividad 2

```
1  /* ACTIVIDAD 2: Direccionamiento Post-indexado con 32 datos
2   Objetivo: Guardar 32 números usando auto-incremento de dirección.
3 */
4 .data
5   i: .skip 128           @ Reserva 128 bytes (32 elementos * 4
6   bytes cada uno)
7
8 .text
9 .global main
10
11 main:
12   ldr r1, =i           @ Carga en R1 la dirección base del
13   arreglo 'i'
14   mov r2, #0            @ R2 es el contador, inicia en 0
15
16 loop2:
17   cmp r2, #32          @ Compara el contador con 32 (el doble
18   que la act. 1)
19   beq salir             @ Si llegamos a 32, salta a 'salir'
20
21   str r2, [r1], #4      @ DIRECCIONAMIENTO POST-INDEXADO:
22   Guarda R2 en la memoria de R1, y LUEGO suma 4 a R1 automá-
23  ticamente.
24   add r2, r2, #1        @ Incrementa el contador R2 en 1
25   b loop2               @ Repite el bucle
26
27 salir:
```



---

23	<b>MOV R7 , #1</b>	<i>@ Prepara sys_exit</i>
24	<b>SVC 0</b>	<i>@ Termina ejecución</i>

## Análisis de resultados



## Actividad 3

Realizar un programa almacene en memoria un arreglo de datos de 32 bits con 16 elementos; una vez transferidos, realizar la copia en sentido inverso en otro arreglo.

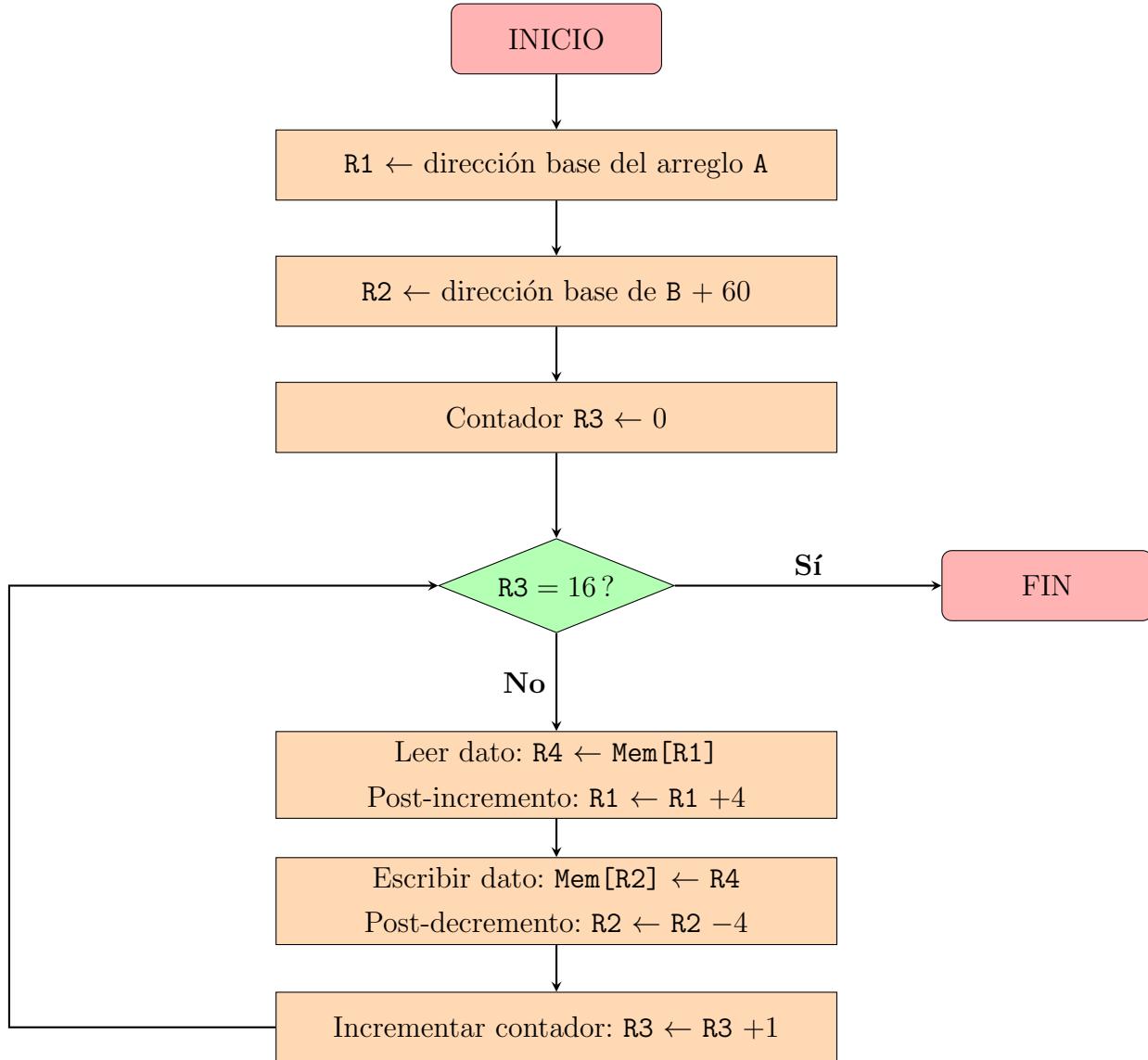
$$A = [\text{dato}_1, \text{dato}_2, \text{dato}_3, \text{dato}_4, \dots, \text{dato}_{15}, \text{dato}_{16}] \quad @\text{Original}$$

$$B = [\text{dato}_{16}, \text{dato}_{15}, \text{dato}_{14}, \text{dato}_{13}, \dots, \text{dato}_2, \text{dato}_1] \quad @\text{Copia}$$

### Propuesta de solución

Para resolver el problema se emplea una estrategia basada en dos apuntadores que recorren los arreglos en sentidos opuestos, aprovechando el modo de direccionamiento **post-indexado** de la arquitectura ARM. El apuntador de lectura R1 se inicializa en el primer elemento del arreglo A y avanza de forma **ascendente** (+4 bytes por iteración), mientras que el apuntador de escritura R2 se posiciona en la *última celda reservada* del arreglo destino B desplazándose 60 bytes desde su base, y retrocede de forma **descendente** (-4 bytes por iteración). Un contador R3, inicializado en cero, controla el número de iteraciones del ciclo; cuando su valor alcanza 16, la instrucción CMP activa la bandera Z del registro de estado y la instrucción BEQ transfiere el control fuera del bucle, finalizando la ejecución mediante una llamada al sistema (SVC 0). De esta forma, cada elemento leído secuencialmente de A se escribe en la posición inversa correspondiente de B, logrando la copia invertida sin consumir registros adicionales para el cálculo de direcciones.

A continuación se presenta el diagrama de flujo correspondiente al algoritmo descrito:



Se cargan las direcciones base de los arreglos mediante LDR: R1 queda apuntando al primer elemento de A; R2 recibe la dirección base de B y se desplaza +60 bytes para posicionarse en la última celda (índice 15), y el contador R3 se pone a cero. A continuación inicia el **ciclo principal**: al comienzo de cada iteración se evalúa la condición de salida  $R3 = 16$ ; si es **falsa**, la instrucción LDR con post-incremento lee el siguiente dato de A en R4 y adelanta R1 en +4 bytes, luego la instrucción STR con post-decremento escribe R4 en la posición actual de B y retrocede R2 en -4 bytes, efectuando el espejismo del arreglo elemento a elemento. Tras la escritura, el contador R3 se incrementa en uno y el flujo regresa a la condición. Cuando R3 alcanza el valor 16 (los 16 elementos han sido copiados en orden inverso).



## Desarrollo

Listing 3: Código de la Actividad 3

```
1  /* ACTIVIDAD 3: Copia de arreglo invertida
2      Objetivo: A = [1..16], B = [16..1]
3 */
4 .data
5     A: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
6             @ Arreglo original de 16 datos
7     B: .skip 64                      @ Arreglo vacío 'B' para la copia (64
8             bytes)
9
10
11 .text
12 .global main
13
14 main:
15     ldr r1, =A                     @ R1 apunta al INICIO del arreglo
16             original 'A'
17     ldr r2, =B                     @ R2 apunta al INICIO del arreglo
18             destino 'B'
19     add r2, r2, #60                @ Movemos R2 para que apunte al ÚLTIMO
20             espacio de 'B' (15 posiciones * 4 bytes = +60)
21     mov r3, #0                     @ R3 es el contador, inicia en 0
22
23 loop_copia:
24     cmp r3, #16                  @ ¿Ya copiamos 16 elementos?
25     beq fin_copia               @ Si sí, termina el ciclo
26
27     ldr r4, [r1], #4              @ Lee el dato apuntado por R1, lo
28             guarda en R4 y avanza R1 hacia ADELANTE (+4 bytes)
29     str r4, [r2], #-4            @ Escribe R4 en la dirección R2, y
30             mueve R2 hacia ATRÁS (-4 bytes)
31
32     add r3, r3, #1                @ Aumenta el contador de copiados
33     b loop_copia                @ Repite el ciclo
```



```
27 fin_copia:  
28     MOV R7, #1                      @ sys_exit  
29     SVC 0                          @ Termina programa
```

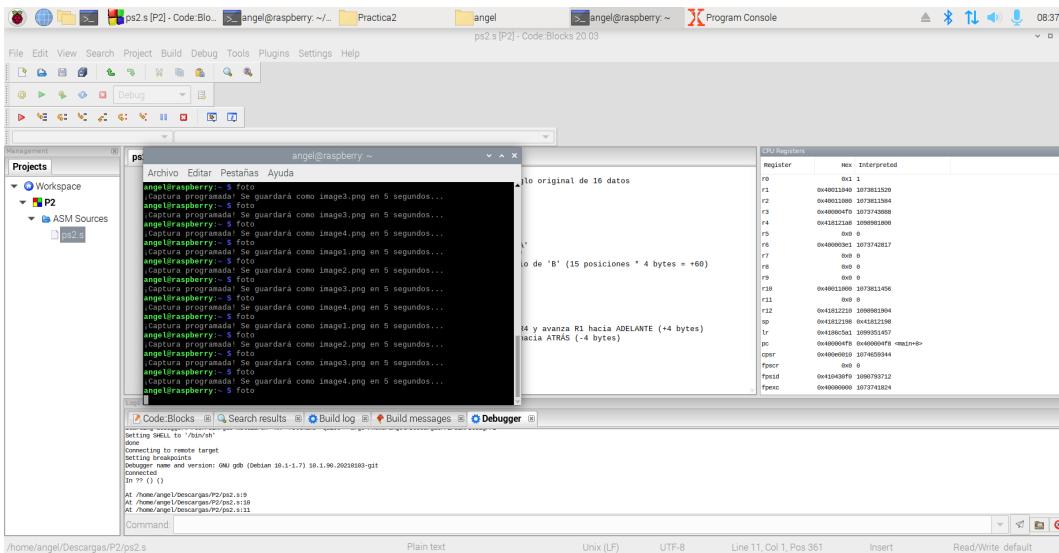


Figura 1: Entorno Code::Blocks preparado para iniciar la depuración del código fuente.

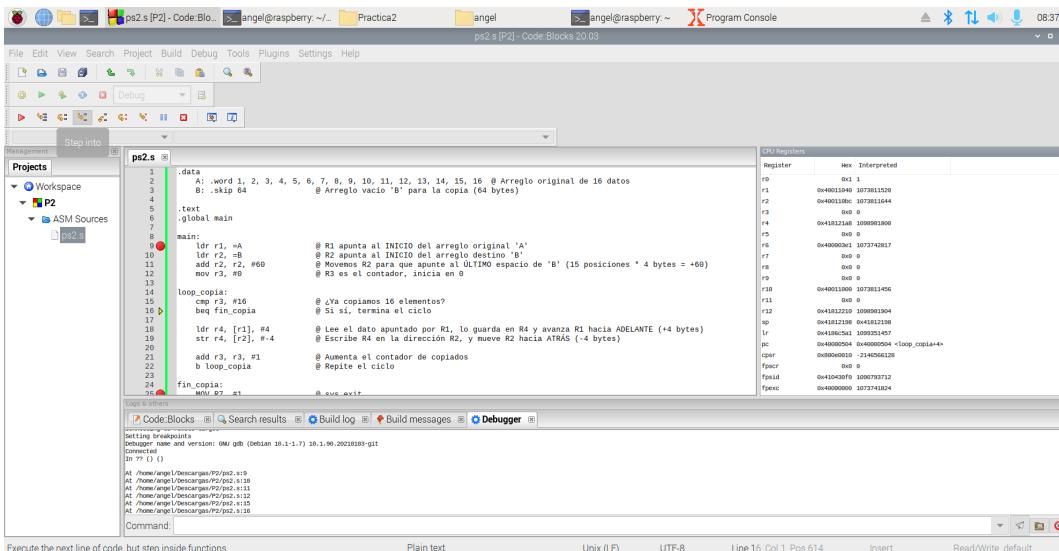


Figura 2: Inicialización de apuntadores. Se observa que R1 apunta al inicio de A (0x40011040) y R2 apunta al final de B (0x400110bc). El contador R3 inicia en 0.



The screenshot shows the Code::Blocks IDE interface with the following details:

- Title Bar:** ps2.s [P2] - Code.Blo... > angel@raspberry:~/... Practica2 angel > angel@raspberry:~ - X Program Console
- Menu Bar:** File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, Help
- Toolbars:** Standard, Debug, Project
- Projects:** Workspace (P2)
- Code Editor:** ps2.s (ASM Sources)

```
ps2.s
1 .data
2 A: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 @ Arreglo original de 16 datos
3 B: .skip 64 @ Arreglo vacío 'B' para la copia (64 bytes)
4
5 .text
6 .global main
7
8 main:
9     ldr r1, =A @ R1 apunta al INICIO del arreglo original 'A'
10    ldr r2, =B @ R2 apunta al INICIO del arreglo destino 'B'
11    add r3, r2, #60 @ Movemos R2 para que apunte al ÚLTIMO espacio de 'B' (15 posiciones * 4 bytes = +60)
12    mov r4, #0 @ R4 es el contador, inicial en 0
13
14    loop_copia:
15        cmp r3, #16 @ ¿Ya copiamos 16 elementos?
16        beq fin_copia @ Si sí, termina el ciclo
17
18        ldr r4, [r1], #4 @ Lee el dato apuntado por R1, lo guarda en R4 y avanza R1 hacia ADELANTE (+4 bytes)
19        str r4, [r2], #-4 @ Escribe R4 en la dirección R2, y mueve R2 hacia ATRÁS (-4 bytes)
20
21        add r3, r3, #1 @ Aumenta el contador de copiados
22        b loop_copia @ Repite el ciclo
23
24    fin_copia:
25        Mov r2, =1 @ .curs_mvt
```
- Registers:** Shows register values for ps2\_1.asm, corresponding to the assembly code above.
- Code Block:** Shows search results, build log, messages, and debugger tabs.
- Bottom Status Bar:** Line 18, Col 1, Pos 672, Insert, Read/Write default.

Figura 3: Evaluación de la condición de salida (`CMP r3, #16`) en las primeras fases del ciclo. La ejecución entra al bloque de copiado.

The screenshot shows the Code::Blocks IDE interface with the following details:

- File Menu:** File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, Help.
- Toolbar:** Includes icons for Open, Save, Build, Run, Stop, and others.
- Project Explorer:** Shows "Workspace" and "P2" selected under "AS Sources".
- Code Editor:** Displays assembly code for "ps2.s".

```
ps2.s
1 .data
2 A: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16    @ Arreglo original de 16 datos
3 B: .skip 64          @ Arreglo vacío 'B' para la copia (64 bytes)
4
5 .text
6 global main
7
8 main:
9     ldr r1, =A          @ R1 apunta al INICIO del arreglo original 'A'
10    ldr r2, =B          @ R2 apunta al INICIO del arreglo destino 'B'
11    add r2, r2, #60      @ Movemos R2 para que apunte al ULTIMO espacio de 'B' (15 posiciones * 4 bytes = +60)
12    mov r3, #0           @ R3 es el contador, inicia en 0
13
14    loop_copia:
15        cmp r3, #16      @ ¿Ya copiamos 16 elementos?
16        beq fin_copia    @ Sí sí, termina el ciclo
17
18        ldr r4, [r1], #4    @ Lee el dato apuntado por R1, lo guarda en R4 y avanza R1 hacia ADELANTE (-4 bytes)
19        str r4, [r2], #4    @ Escribe R4 en la dirección R2, y mueve R2 hacia ATRÁS (-4 bytes)
20
21    ldi add r3, r3, #1    @ Aumenta el contador de copiados
22    b loop_copia         @ Repite el ciclo
23
24    fin_copia:
25        MOV.w r7, #1
26        B .here_exit
```
- CPU Registers Window:** Shows register values for R9 through R12, R14, R15, R16, and R17, along with their hex and interpreted values.
- Log and Errors:** Shows build log and messages.
- Bottom Status Bar:** Execute the next line of code, Command:, Plain text, Unix (LF), UTF-8, Line 21, Col 1, Pos 887, Insert, Read/Write default.

Figura 4: Cuarta iteración. El contador R3 tiene el valor de 3, y en R4 se puede observar cargado el valor 0x4, demostrando que los apuntadores R1 y R2 se han actualizado correctamente.



The screenshot shows the ps2s assembly editor interface. The top menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, Help, and a toolbar with various icons. The main window displays assembly code for a copy operation between arrays A and B.

```
.data
A: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16    @ Arreglo original de 16 datos
B: .skip 64          @ Arreglo vacío "B" para la copia (64 bytes)

.text
.global main

main:
    ldr r1, =A      @ R1 apunta al INICIO del arreglo original 'A'
    ldr r2, =B      @ R2 apunta al INICIO del arreglo destino 'B'
    mov r3, #0      @ Movemos R3 para que apunte al ULTIMO espacio de 'B' (15 posiciones * 4 bytes = +60)
    mov r3, #0      @ R3 es el contador, inicia en 0

loop_copia:
    cmp r3, #16    @ ¿Ya copiamos 16 elementos?
    beq fin_copia  @ Si sí, termina el ciclo

    ldr r4, [r1], #4  @ Lee el dato apuntado por R1, lo guarda en R4 y avanza R1 hacia ADELANTE (+4 bytes)
    str r4, [r2], #4  @ Escribe R4 en la dirección R2, y mueve R2 hacia ATRAS (-4 bytes)
    add r3, r3, #1   @ Aumenta el contador de copiados
    b loop_copia    @ Repite el ciclo

fin_copia:
    MOVW xzr, x1    @ Salir al FP
```

The right side of the interface shows the CPU Registers panel, listing registers R9 through R0 and their corresponding hex and interpreted values. Below the assembly code, there is a list of assembly instructions and their addresses.

Figura 5: Octava iteración del ciclo. El contador R3 llega a 7 y en R4 se carga el valor 0x7, confirmando la constancia y estabilidad del bucle.

The screenshot shows the Code::Blocks IDE interface with the following details:

- Title Bar:** ps2.s [P2] - Code Blo... angel@raspberry:~/... Practica2
- File Menu:** File Edit View Search Project Build Debug Tools Plugins Settings Help
- Toolbar:** Includes icons for Run, Stop, Break, and others.
- Project Explorer:** Shows a workspace named "P2" containing an "ASM Sources" folder with "ps2.s".
- Code Editor:** Displays the assembly code for "ps2.s". The comments explain the purpose of each instruction, such as copying array A to array B and moving R2 to the end of array B.
- Registers Window:** Shows the CPU Registers for the ARM processor. It includes columns for Register, Hex, and Interpreted values. Registers R0 through R12, R14, R15, and R16 are listed, along with PC, SP, and FP.
- Log & Errors:** Shows build log and messages.
- Status Bar:** Displays file paths, line numbers, and current position (Line 26, Col 1, Pos 1052).

Figura 6: Fin de la ejecución. El contador R3 alcanza el valor de 0x10 (16 en decimal). El programa sale del ciclo y ejecuta la llamada al sistema (**SVC 0**).

## Análisis de resultados

En una primera instancia, antes de ingresar al ciclo principal, es necesario cargar las direcciones base de los arreglos definidos en memoria. Por medio de la instrucción LDR, se



---

carga en el registro R1 la dirección de inicio del arreglo original A, la cual corresponde al valor 0x40011040. De la misma forma, se carga en R2 la dirección base del arreglo vacío B (0x40011080). Sin embargo, dado que la copia debe realizarse en sentido inverso, se emplea la instrucción ADD R2, R2, #60 para desplazar el apuntador de escritura hacia el final del espacio reservado para B. Como cada uno de los 16 datos es de 32 bits (4 bytes), el desplazamiento total es de 60 bytes, lo que posiciona correctamente a R2 en la dirección 0x400110BC. Adicionalmente, se inicializa el registro R3 con el valor de 0 para fungir como la variable de control (contador) del ciclo. Todos estos valores iniciales coinciden exactamente con lo que se muestra en los registros de la CPU en las primeras etapas de la depuración.

Una vez dentro de la etiqueta `loop_copia`, el programa ejecuta la lógica central del copiado haciendo uso del direccionamiento indirecto con post-indexado. La instrucción LDR R4, [R1], #4 accede a la dirección de memoria que contiene R1, extrae el dato y lo guarda en el registro temporal R4; inmediatamente después de la lectura, el procesador incrementa automáticamente el valor de R1 en 4 bytes para apuntar al siguiente dato del arreglo A. Posteriormente, la instrucción STR R4, [R2], #-4 toma el dato recién cargado en R4 y lo guarda en la dirección de memoria apuntada por R2; una vez almacenado, el apuntador R2 se decrementa automáticamente en 4 bytes. Este emparejamiento de instrucciones permite leer el arreglo original de inicio a fin mientras se escribe simultáneamente en el arreglo destino de fin a inicio, sin necesidad de emplear instrucciones aritméticas extra para recalcular las direcciones.

Al analizar la evolución dinámica a través de las iteraciones capturadas, se verifica que los datos se transfieren correctamente. Por ejemplo, en las primeras iteraciones se observa que R4 adquiere los valores de 0x1 y posteriormente 0x4, reflejando la extracción secuencial de los datos. De forma concurrente, el apuntador de lectura R1 incrementa progresivamente (pasando por 0x40011044, 0x40011050, hasta 0x4001105C), mientras que el apuntador de escritura R2 decrementa su valor (pasando por 0x400110B8, 0x400110AC, hasta 0x400110A0). En cada vuelta, la instrucción ADD R3, R3, #1 incrementa el contador, lo que permite llevar el control exacto de los elementos transferidos.

Finalmente, el ciclo se rompe gracias a la instrucción de comparación CMP R3, #16. Cuando el contador R3 alcanza el valor de 0x10 (16 en decimal), la comparación resulta en cero, lo que actualiza el registro de estado (CPSR) levantando la bandera de cero (Z). Al detectarse esta bandera, se cumple la condición de la instrucción BEQ `fin_copia`, realizando el salto fuera



del bucle. Al finalizar el programa, los registros muestran que R1 terminó en la dirección 0x40011080 (habiendo recorrido exactamente los 64 bytes del arreglo A) y R2 terminó en 0x4001107C (habiendo retrocedido 64 bytes desde su punto de inicio).

## Actividad 4

Realizar un programa que forme un arreglo de 20 elementos, con el siguiente criterio:

$$A = [i, 2i, 4i, 8i, 16i, \dots, ni]$$

Donde  $i$  es un número considerado como valor inicial.

- a) Enviar a memoria cada uno de ellos.
- b) Sumar y almacenar en memoria el resultado.

## Propuesta de solución

### Desarrollo

Listing 4: Código de la Actividad 4

```
1  /* ACTIVIDAD 4: Arreglo exponencial y su suma
2      Objetivo: Generar serie multiplicando por 2 (Shift), y sumar
3          elementos.
4 */
5  .data
6      A:      .skip 80           @ Reserva memoria para 20 elementos
7          (20 * 4 bytes = 80)
8      SUMA: .word 0            @ Variable para guardar la sumatoria
9          final
10
11 .text
12 .global main
13
14 main:
15     ldr r0, =A              @ R0 apunta a la dirección de memoria
16     de A
```



```
13    mov r1, #3           @ R1 será la variable 'i' inicial (Ejemplo: usamos 3)
14    mov r2, #0           @ R2 es el contador de elementos creados
15    mov r3, #0           @ R3 será el Acumulador (Sumatoria), inicia en 0
16
17 loop_potencias:
18    cmp r2, #20          @ Compara si ya generamos los 20 elementos
19    beq fin_potencias    @ Si llegamos a 20, salimos del bucle
20
21    str r1, [r0], #4      @ Guarda el valor actual en memoria y avanza el puntero R0
22    add r3, r3, r1        @ Suma el valor actual de 'i' al Acumulador Total (R3)
23    lsl r1, r1, #1        @ Desplazamiento Izquierdo: Multiplica 'i' por 2 para la siguiente iteración
24    add r2, r2, #1        @ Incrementa contador
25    b loop_potencias    @ Repite
26
27 fin_potencias:
28    ldr r0, =SUMA         @ Carga la dirección de la variable SUMA
29    str r3, [r0]          @ Guarda el resultado total (R3) en esa memoria
30    MOV R7, #1            @ sys_exit
31    SVC 0                @ Termina
```

## Análisis de resultados

## Actividad 5

Realizar un programa que multiplique dos matrices de 2x2; los datos podrán ser de 8 bits.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} I & J \\ K & L \end{bmatrix}$$

## Propuesta de solución

### Desarrollo

Listing 5: Código de la Actividad 5

```

1  /* ACTIVIDAD 5: Multiplicación de matrices 2x2
2      Objetivo: [A B] x [E F] = [I J]
3                  [C D]      [G H]      [K L]
4 */
5 .data
6     M1: .byte 2, 1, 3, 4          @ Matriz 1 (A,B,C,D) -> Valores de
7                 ejemplo de 8 bits
8     M2: .byte 1, 5, 2, 1          @ Matriz 2 (E,F,G,H) -> Valores de
9                 ejemplo de 8 bits
10    MR: .byte 0, 0, 0, 0         @ Matriz Resultado (I,J,K,L)
11
12
13 .text
14 .global main
15
16 main:
17     ldr r0, =M1                @ Dirección Matriz 1
18     ldr r1, =M2                @ Dirección Matriz 2
19     ldr r2, =MR                @ Dirección Matriz Resultado
20
21     @ Cargamos los elementos de M1 (Usamos LDRB por ser Bytes)
22     ldrb r3, [r0, #0]          @ R3 = A (Posición 0)
23     ldrb r4, [r0, #1]          @ R4 = B (Posición 1)
24     ldrb r5, [r0, #2]          @ R5 = C (Posición 2)
25     ldrb r6, [r0, #3]          @ R6 = D (Posición 3)
26
27     @ Cargamos los elementos de M2
28     ldrb r7, [r1, #0]          @ R7 = E
29     ldrb r8, [r1, #1]          @ R8 = F

```

```

27    ldrb r9, [r1, #2]           @ R9 = G
28    ldrb r10,[r1, #3]          @ R10 = H

29
30    @ Calculando I = A*E + B*G
31    mul r11, r3, r7            @ R11 = A * E
32    mla r11, r4, r9, r11      @ Multiply-Accumulate: R11 = (B * G) +
     R11
33    strb r11, [r2, #0]         @ Guardamos 'I' en la matriz resultado

34
35    @ Calculando J = A*F + B*H
36    mul r11, r3, r8            @ R11 = A * F
37    mla r11, r4, r10, r11     @ R11 = (B * H) + R11
38    strb r11, [r2, #1]         @ Guardamos 'J'

39
40    @ Calculando K = C*E + D*G
41    mul r11, r5, r7            @ R11 = C * E
42    mla r11, r6, r9, r11      @ R11 = (D * G) + R11
43    strb r11, [r2, #2]         @ Guardamos 'K'

44
45    @ Calculando L = C*F + D*H
46    mul r11, r5, r8            @ R11 = C * F
47    mla r11, r6, r10, r11     @ R11 = (D * H) + R11
48    strb r11, [r2, #3]         @ Guardamos 'L'

49
50    MOV R7, #1                @ sys_exit
51    SVC 0                     @ Termina

```

## Análisis de resultados

## Actividad 6

Realizar un programa que encuentre el número con valor mayor en un arreglo de 20 elementos que serán almacenados en memoria; para lo cual:

- Indicar cuál fue el valor mayor.

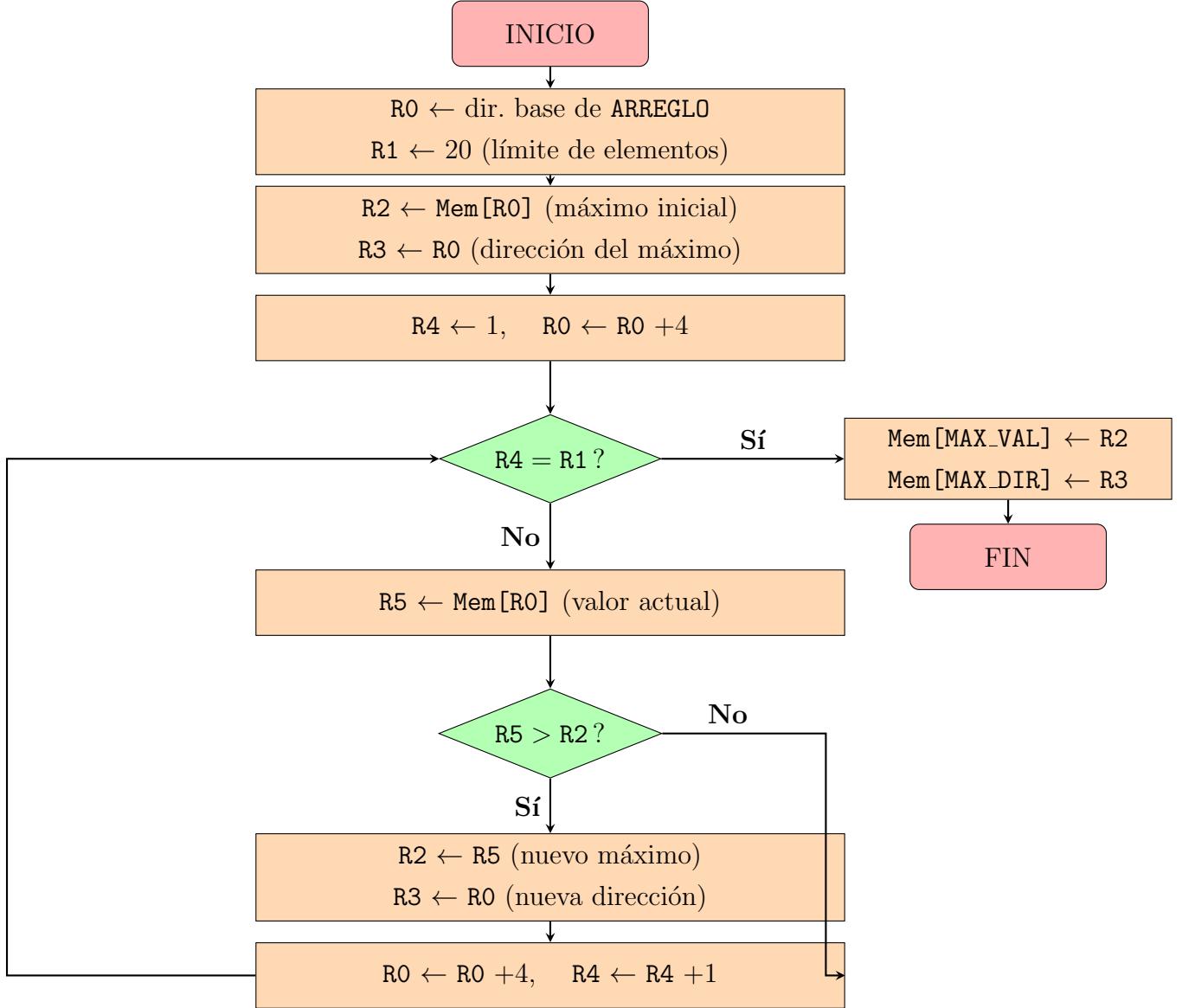


- 
- b) Ubicar la dirección donde se encontró este número.
  - c) Usar las direcciones que requiera para cumplir lo solicitado.

## Propuesta de solución

Se emplea un recorrido lineal sobre el arreglo de 20 elementos, comparando cada valor contra un máximo registrado. Se inicializa el registro R2 con el primer elemento del arreglo, asumiendo que este es el mayor, y R3 con su dirección de memoria correspondiente. A partir del segundo elemento, un ciclo iterativo controlado por el contador R4 recorre las posiciones restantes: en cada paso, la instrucción LDR carga el valor actual en R5 y la instrucción CMP lo compara contra el máximo almacenado en R2. Si el nuevo valor supera al registrado, las instrucciones MOV actualizan tanto el valor máximo como su dirección. Cuando el contador R4 iguala al límite de 20, el programa sale del ciclo y almacena el resultado final en las variables de memoria MAX\_VAL y MAX\_DIR mediante instrucciones STR, cumpliendo así con los tres incisos solicitados.

A continuación se presenta el diagrama de flujo correspondiente al algoritmo descrito:



The diagram starts by loading the array base address into  $R0$  and setting the limit to 20 elements in  $R1$ . It is assumed that the first data is the maximum, so  $R2$  stores its value and  $R3$  stores its memory address. The counter  $R4$  is initialized to 1 and the pointer  $R0$  advances to the second element (+4 bytes). In each iteration of the main loop, it first checks if the counter has reached the limit: when  $R4 = R1$  (20), the flow goes to the storage block where the results are written into the variables  $\text{MAX\_VAL}$  and  $\text{MAX\_DIR}$ , thus ending the program. If the condition is not met, it loads the current memory value into  $R5$  and compares it with the maximum registered in  $R2$ . If  $R5$  is greater, the registers  $R2$  and  $R3$  are updated with the new value and the program loops back to the start of the main loop.



su dirección; en caso contrario, se omite la actualización saltando directamente al avance del puntero. Finalmente, se incrementa R0 en 4 bytes y R4 en una unidad, regresando el flujo al inicio del ciclo.

## Desarrollo

Listing 6: Código de la Actividad 6

```
1  /* ACTIVIDAD 6: Búsqueda del número mayor en arreglo
2      Objetivo: Encontrar el máximo y guardar su valor y su dirección
3          de memoria.
4 */
5  .data
6      @ Arreglo de 20 números al azar para la prueba
7      ARREGLO: .word 5, 12, 3, 45, 2, 105, 1, 8, 33, 10, 11, 14, 0,
8          77, 21, 6, 9, 88, 4, 15
9      MAX_VAL: .word 0           @ Variable para guardar el número más
10     grande
11     MAX_DIR: .word 0         @ Variable para guardar la dirección
12     de memoria de ese número
13
14 .text
15 .global main
16
17 main:
18     ldr r0, =ARREGLO          @ R0 = Puntero principal que recorrerá
19         el arreglo
20     mov r1, #20              @ R1 = Límite de elementos (20)
21     ldr r2, [r0]              @ R2 = Guarda el MÁXIMO (Inicia
22         asumiendo que el índice 0 es el mayor)
23     mov r3, r0                @ R3 = Guarda la DIRECCIÓN del máximo
24         (Inicia con la del índice 0)
25     mov r4, #1                @ R4 = Contador de ciclo (inicia en 1
26         porque ya evaluamos el 0)
27     add r0, r0, #4            @ Avanzamos el puntero de memoria al índice 1
28
```



```
21 buscar_mayor:  
22     cmp r4, r1                      @ Compara el contador con 20  
23     beq fin_busqueda                @ Si terminamos, salta al final  
24  
25     ldr r5, [r0]                    @ R5 = Lee el valor actual de la  
26     memoria  
27     cmp r5, r2                      @ Compara (Valor_Actual vs Má  
28     ximo_Registrado)  
29     ble siguiente                  @ Branch if Less or Equal: Si es menor  
30     o igual, ignóralo y salta a 'siguiente'  
31  
32     @ Si llegó a esta linea, encontramos un nuevo mayor  
33     mov r2, r5                      @ R2 adopta el nuevo valor mayor  
34     mov r3, r0                      @ R3 adopta la dirección de memoria de  
35     este nuevo mayor  
36  
37 siguiente:  
38     add r0, r0, #4                 @ Avanzamos la lectura en la memoria  
39     (4 bytes)  
40     add r4, r4, #1                 @ Incrementamos el contador de ciclo  
41     b buscar_mayor                @ Repetimos  
42  
43 fin_busqueda:  
44     ldr r6, =MAX_VAL              @ Carga dirección para guardar el  
45     valor  
46     str r2, [r6]                   @ Almacena en memoria el valor mayor  
47     ldr r6, =MAX_DIR              @ Carga dirección para guardar la  
48     ubicación  
49     str r3, [r6]                   @ Almacena en memoria la dirección del  
50     mayor  
51  
52     MOV R7, #1                   @ sys_exit  
53     SVC 0                        @ Terminar
```



The screenshot shows the Code-Blocks IDE interface. The assembly code for file ps2.s is displayed in the main window, showing a program to find the maximum value in an array. The CPU Registers window on the right shows the initial state of registers after variable initialization. The registers are as follows:

Register	Hex	Interpretation
r0	0x40011040	1073811230
r1	0x40011044	1073811234
r2	0x40011048	1073811238
r3	0x4001104C	1073811242
r4	0x40011050	1073811244
r5	0x40000009	10737412825
r7	0x0	0
r8	0x0	0
r9	0x0	0
r10	0x40011000	1073811406
r11	0x0	0
r12	0x41400000	1073741284
sp	0x41400000	1073741288
lr	0x41400001	1073741289
pc	0x40000000	1073811200
cpar	0x40000000	1073811244
tpsr	0x0	0
tpsid	0x41400000	1073811212
tpsec	0x40000000	1073741284

Figura 7: Estado de los registros inmediatamente después de la inicialización de variables. Se asume que el índice 0 es el máximo.

The screenshot shows the assembly code for ps2.s running in the debugger. The CPU Registers window shows the state of registers during the second iteration of the loop. The registers are as follows:

Register	Hex	Interpretation
r0	0x40011040	1073811230
r1	0x40011044	1073811234
r2	0x40011048	1073811238
r3	0x4001104C	1073811242
r4	0x1	0x1
r5	0x0	0
r6	0x444	0x444
r7	0x1	0x1
r8	0x0	0
r9	0x2	0x2
r10	0x41400000	1073741284
r11	0x1	0x1
r12	0x41400000	1073741284
sp	0x41400000	1073741288
lr	0x41400001	1073741289
pc	0x40000000	1073811200
cpar	0x40000000	1073811244
tpsr	0x0	0
tpsid	0x41400000	1073811212
tpsec	0x40000000	1073741284

Figura 8: Interrupción dentro del bloque de actualización (`mov r3, r0`) durante la segunda iteración, al encontrar un número mayor que el inicial.



The screenshot shows the Code::Blocks IDE interface. The assembly code for file ps2.s is displayed in the main window. The CPU Registers panel shows the state of registers r0 through r13. The value of register r4 is 0x6. The assembly code includes comments explaining the logic of finding the maximum value in memory.

```
ps2.s
13 ldr r2, [r0]          @ R2 = Guarda el MAXIMO (Inicia asumiendo que el indice 0 es el mayor)
14 mov r3, r0            @ R3 = Guarda la DIRECCION del maximo (Inicia con la del indice 0)
15 mov r4, #1             @ R4 = Contador de ciclo (Inicia en 1 porque ya evaluamos el 0)
16 add r0, r0, #4          @ Avanzamos el puntero de memoria al indice 1
17
18 buscar_mayor:
19     cmp r4, r1           @ Compara el contador con 20
20     beq fin_busqueda    @ Si terminamos, salta al final
21
22     ldr r5, [r0]          @ R5 = Lee el valor actual de la memoria
23     cmp r5, r2            @ Compara (Valor_Actual vs Maximo_Registrado)
24     ble siguiente         @ Branch if Less or Equal: Si es menor o igual, ignóralo y salta a 'siguiente'
25
26     @ Si llega a esta linea, encontramos un nuevo mayor
27     mov r3, r5            @ R2 adopta la dirección de memoria de este nuevo mayor
28     mov r3, r0            @ R3 adopta el nuevo valor mayor
29
30 siguiente:
31     add r0, r0, #4          @ Avanza la lectura en la memoria (4 bytes)
32     add r4, r4, #1           @ Incrementamos el contador de ciclo
33     b buscar_mayor        @ Repetimos
34
35 fin_busqueda:
36     ldr r6, =MAX_VAL      @ Carga dirección para guardar el valor
37     str r2, [r6]            @ Almacena en memoria el valor mayor
```

CPU Registers:

Register	Hex	Interpreted
r0	0x40011000	1073811544
r1	0x14	20
r2	0x69	105
r3	0x40011004	1073811540
r4	0x6	6
r5	0x69	105
r6	0x40000000	1073742825
r7	0x0	0
r8	0x0	0
r9	0x0	0
r10	0x40011000	1073811456
r11	0x0	0
r12	0x415e0000	1073742824
sp	0x415e2100	1073742828
lr	0x415e0001	1073742827
pc	0x40000010	1073742825 <buscar_mayor>
cpar	0x20000010	1073742842
fpcr	0x0	0
fpcid	0x415e0000	1073742812
fpxc	0x40000000	1073742824

Figura 9: El ciclo cursando la iteración 6 ( $R4 = 0x6$ ). El programa ya ha registrado el verdadero número máximo ( $0x69$ ).

The screenshot shows the Code::Blocks IDE interface. The assembly code for file ps2.s is displayed in the main window. The CPU Registers panel shows the state of registers r0 through r13. The value of register r4 is 0x8. The assembly code includes comments explaining the logic of finding the maximum value in memory.

```
ps2.s
13 ldr r2, [r0]          @ R2 = Guarda el MAXIMO (Inicia asumiendo que el indice 0 es el mayor)
14 mov r3, r0            @ R3 = Guarda la DIRECCION del maximo (Inicia con la del indice 0)
15 mov r4, #1             @ R4 = Contador de ciclo (Inicia en 1 porque ya evaluamos el 0)
16 add r0, r0, #4          @ Avanzamos el puntero de memoria al indice 1
17
18 buscar_mayor:
19     cmp r4, r1           @ Compara el contador con 20
20     beq fin_busqueda    @ Si terminamos, salta al final
21
22     ldr r5, [r0]          @ R5 = Lee el valor actual de la memoria
23     cmp r5, r2            @ Compara (Valor_Actual vs Maximo_Registrado)
24     ble siguiente         @ Branch if Less or Equal: Si es menor o igual, ignóralo y salta a 'siguiente'
25
26     @ Si llega a esta linea, encontramos un nuevo mayor
27     mov r3, r5            @ R2 adopta la dirección de memoria de este nuevo mayor
28     mov r3, r0            @ R3 adopta el nuevo valor mayor
29
30 siguiente:
31     add r0, r0, #4          @ Avanza la lectura en la memoria (4 bytes)
32     add r4, r4, #1           @ Incrementamos el contador de ciclo
33     b buscar_mayor        @ Repetimos
34
35 fin_busqueda:
36     ldr r6, =MAX_VAL      @ Carga dirección para guardar el valor
37     str r2, [r6]            @ Almacena en memoria el valor mayor
```

CPU Registers:

Register	Hex	Interpreted
r0	0x40011000	1073811532
r1	0x14	20
r2	0x69	105
r3	0x40011004	1073811540
r4	0x8	8
r5	0x69	105
r6	0x40000000	1073742825
r7	0x0	0
r8	0x0	0
r9	0x0	0
r10	0x40011000	1073811456
r11	0x0	0
r12	0x415e2200	1073742804
sp	0x415e0000	1073742844
lr	0x415e0001	1073742845 <buscar_mayor>
pc	0x40000010	1073742825
cpar	0x20000010	1073742842
fpcr	0x0	0
fpcid	0x415e0000	1073742812
fpxc	0x40000000	1073742824

Figura 10: Iteración 8 del ciclo ( $R4 = 0x8$ ). La condición de salto evita que los números menores sobreescriban el valor máximo ya encontrado.

The screenshot shows the Code::Blocks IDE interface. On the left, the Projects panel displays a workspace named 'P2' containing an ASIM Sources folder with 'ps2.s'. The main window shows the assembly code for 'ps2.s' with comments in Spanish. The code includes instructions like BEQ, LDR, CMP, BLE, ADD, and MOV, along with comments explaining the logic of finding the maximum value in an array. The CPU Registers window on the right shows the state of the processor registers (R0-R13, PC, SP, LR, PCER, CPSR) at the end of the program. The PC register shows the value 0x40011040, which corresponds to the SVC 0 instruction.

```

ps2.s
20      beq fin_busqueda    @ Si terminamos, salta al final
21
22      ldr r5, [r0]          @ R5 = Lee el valor actual de la memoria
23      cmp r5, r2            @ compara (Valor_Actual vs Máximo_Registrado)
24      ble siguiente        @ Branch if Less or Equal: Si es menor o igual, ignóralo y salta a 'siguiente'
25
26      @ Si llego a esta linea, encontramos un nuevo mayor
27      mov r3, r0             @ R3 adopta el nuevo valor mayor
28      mov r3, r0             @ R3 adopta la dirección de memoria de este nuevo mayor
29
30      siguiente:           @ Repetimos
31      add r0, r0, #4         @ Avanzamos la lectura en la memoria (4 bytes)
32      add r4, r4, #1         @ Incrementamos el contador de ciclo
33      b buscar_mayor       @ Repetimos
34
35      fin_busqueda:         @ Carga dirección para guardar el valor
36      ldr r6, =MAX_VAL      @ Almacena en memoria el valor mayor
37      str r2, [r6]           @ Carga dirección para guardar la ubicación
38      ldr r6, =MAX_DIR      @ Almacena en memoria la dirección del mayor
39
40      r0                   @ sys_exit
41      MOV R7, #1             @ SVC 0
42      |                   @ Terminar
43

```

Register	Hex	Interpreted
R0	0x40011000	0x737311600
R1	0x14	20
R2	0x09	005
R3	0x40011054	0x737311540
R4	0x00	0
R5	Ref 15	
R6	0x40011004	0x737311604
R7	0x1	1
R8	0x0	0
R9	0x0	0
R10	0x40011000	0x737311400
R11	0x0	0
R12	0x41414140	0x737311004
sp	0x40000010	0x414141200
lr	0x41414141	0x737311407
pc	0x40000040	0x40000040 <fin_busqueda>+0
cpsr	0x60000010	0x111130256
fpacr	0x0	0
fpcsr	0x41414140	0x737311372
fpcr	0x40000000	0x737311204

Figura 11: Fin de la ejecución (SVC 0). El valor máximo y su dirección se han almacenado en memoria correctamente.

## Análisis de resultados

Antes de entrar al ciclo de evaluación, el programa establece las condiciones iniciales cargando la dirección base del arreglo en el registro R0, la cual corresponde a 0x40011040. Se carga el límite de iteraciones en R1 con el valor de 0x14 (20 en decimal). Asumiendo que el primer dato es el mayor por defecto, la instrucción LDR R2, [R0] extrae el valor almacenado en esa primera dirección, cargando un 0x5 (5) en el registro R2, mientras que R3 guarda la dirección 0x40011040. El contador R4 se inicializa en 0x1 y el puntero R0 se incrementa en 4 bytes para apuntar al siguiente elemento (0x40011044). Todos estos valores se confirman en los registros de la CPU mostrados en la primera captura.

Al entrar al bucle **buscar\_mayor**, se evalúan secuencialmente los datos mediante la instrucción LDR R5, [R0]. En la segunda iteración, el puntero se encuentra en 0x40011044 y carga el valor 0xC (12) en R5. La instrucción CMP R5, R2 compara este 0xC contra el 0x5 registrado. Como 12 es mayor que 5, no se activa la condición de salto de BLE **siguiente**, permitiendo que el flujo entre al bloque de actualización. En este punto, R2 adopta el nuevo valor máximo (0xC) y R3 adopta su respectiva dirección (0x40011044), tal como se evidencia en la segunda captura donde el PC está detenido justo en la reasignación de direcciones.

A medida que el ciclo avanza, el programa detecta el verdadero valor máximo del arreglo. En



---

la captura de la iteración 6 ( $R4 = 0x6$ ), se observa que el registro  $R2$  contiene el valor  $0x69$  (105 en decimal). Este número corresponde al sexto elemento del arreglo original. Consecuentemente, el registro  $R3$  preserva la dirección exacta de este dato, indicando  $0x40011054$  (calculado como la dirección base  $0x40011040 + 5$  desplazamientos de 4 bytes). A partir de este punto, en las iteraciones subsecuentes (como se observa en la iteración 8 de la cuarta captura), los valores leídos en  $R5$  resultan ser menores a  $0x69$ . Esto provoca que la instrucción **BLE siguiente** se cumpla de forma continua, saltando la actualización y manteniendo intactos los registros  $R2$  y  $R3$ .

El ciclo iterativo finaliza cuando el contador  $R4$  alcanza el valor de  $0x14$ , activando el salto condicional **BEQ fin\_busqueda**. En esta última sección, el programa cumple con los incisos solicitados trasladando los resultados retenidos en el procesador hacia la memoria principal. Se carga la dirección de la variable **MAX\_VAL** en  $R6$  ( $0x40011090$ ) y mediante **STR R2, [R6]**,  $[R6]$  se escribe permanentemente el valor  $0x69$ . A continuación, se carga la dirección de **MAX\_DIR** en  $R6$  ( $0x40011094$ ) y se guarda el contenido de  $R3$  ( $0x40011054$ ). La última captura valida que el registro  $R2$  retuvo satisfactoriamente el número 105 y el registro  $R3$  su ubicación exacta, demostrando que el manejo de punteros y los saltos condicionales operaron con total precisión sobre la memoria estática.



## Actividad 7

Realizar un programa que ordene de manera ascendente un arreglo de 32 elementos de 32 bits; deberá:

- Mantener el arreglo original.
- Generar otro arreglo con el ordenamiento del original.

**Arreglo original.**

$A[0]$	$A[1]$	$A[2]$	$\cdots$	$A[31]$
--------	--------	--------	----------	---------

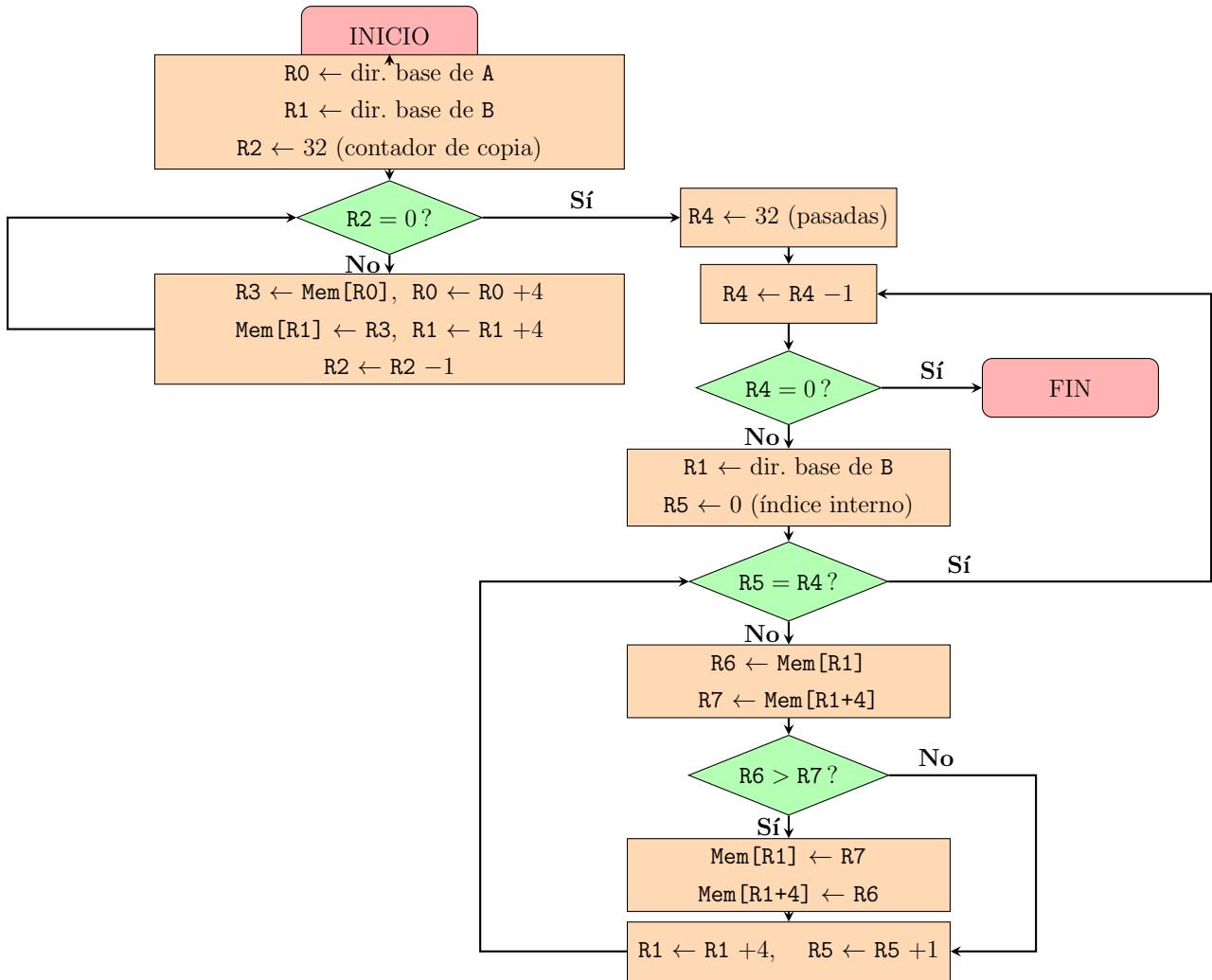
**Arreglo ordenado.**

Menor $A[x]$	Mayor $A[y]$
--------------	--------------

### Propuesta de solución

La solución se estructura en dos fases secuenciales. En la **primera fase** se realiza una copia íntegra del arreglo original A hacia un arreglo auxiliar B, recorriendo los 32 elementos mediante un ciclo con post-indexado que lee de A y escribe en B, avanzando ambos apuntadores en 4 bytes por iteración y decrementando un contador hasta llegar a cero. Una vez completada la copia, la **segunda fase** aplica el algoritmo de ordenamiento burbuja sobre el arreglo B, preservando intacto el arreglo original. El algoritmo utiliza dos bucles anidados: el bucle externo decremente el límite de comparaciones en cada pasada (de 32 hasta 0), mientras que el bucle interno recorre los elementos adyacentes de B comparándolos entre sí; cuando el elemento izquierdo es mayor que el derecho, se efectúa un intercambio cruzado mediante instrucciones STR que escriben los valores en posiciones invertidas. Este proceso se repite hasta que el bucle externo agota sus pasadas, dejando el arreglo B completamente ordenado de menor a mayor.

A continuación se presenta el diagrama de flujo correspondiente al algoritmo descrito:



El diagrama se divide en dos fases claramente diferenciadas. En la **Fase 1** se inicializan los apuntadores  $R0$  (origen en A) y  $R1$  (destino en B) junto con el contador  $R2$  con valor 32. El ciclo de copia evalúa si  $R2$  ha llegado a cero; mientras no lo sea, se lee un dato de A mediante post-indexado (incrementando  $R0$ ), se escribe en B (incrementando  $R1$ ), y se decrementa el contador, repitiendo el proceso. Cuando  $R2$  alcanza cero, los 32 elementos han sido transferidos y el flujo pasa a la Fase 2.

En la **Fase 2** se inicializa  $R4$  con 32 para controlar las pasadas del algoritmo burbuja. Al inicio de cada pasada se decrementa  $R4$ ; si llega a cero, el ordenamiento está completo y el programa termina. En caso contrario, se reinicializa el apuntador  $R1$  a la base de B y el índice interno  $R5$  a cero. El bucle interno compara  $R5$  con  $R4$ : si son iguales, la pasada terminó y



se regresa al bucle externo para decrementar R4 nuevamente. De lo contrario, se cargan los dos elementos adyacentes B[i] y B[i+1] en R6 y R7 respectivamente. Si R6 es mayor que R7, se ejecuta el intercambio cruzado escribiendo los valores en posiciones invertidas; si no, se omite el intercambio. Finalmente, se avanza el apuntador R1 en 4 bytes y se incrementa R5, repitiendo el bucle interno hasta completar la pasada.

## Desarrollo

Listing 7: Código de la Actividad 7

```
1  /* ACTIVIDAD 7: Ordenamiento Burbuja de 32 elementos (32 bits)
2   Objetivo: Conservar arreglo original, ordenar la copia.
3 */
4 .data
5   @ Arreglo original desordenado (32 elementos)
6   A: .word
7     32,31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14,13,
8     12,11,10,9,8,7,6,5,4,3,2,1
9   @ Arreglo copia donde se hará el ordenamiento
10  B: .skip 128           @ Reserva 128 bytes (32 words x 4)
11
12 .text
13 .global main
14
15 main:
16   @ --- FASE 1: COPIAR A en B ---
17   ldr r0, =A             @ R0 apunta a Original
18   ldr r1, =B             @ R1 apunta a Copia
19   mov r2, #32            @ R2 contador para copiar
20
21 copiar:
22   cmp r2, #0             @ ¿Quedan elementos por copiar?
23   beq iniciar_orden      @ Si es 0, terminamos de copiar y
24   vamos a ordenar
25   ldr r3, [r0], #4        @ Lee de A y avanza
26   str r3, [r1], #4        @ Escribe en B y avanza
27   sub r2, r2, #1          @ Resta 1 al contador
28   b copiar
```

```

26
27     @ --- FASE 2: ORDENAMIENTO BURBUJA (Sobre B) ---
28
29     iniciar_orden:
30         mov r4, #32                      @ R4 = N (Cantidad total de elementos)
31
32     bucle_externo:
33         subs r4, r4, #1                  @ Resta 1 a N (N = N - 1) y actualiza
34             flags (S final)
35         beq fin_ordenamiento          @ Si N llega a 0, todo está ordenado
36
37         ldr r1, =B                     @ Resetea el puntero R1 al inicio de B
38             para cada pasada
39         mov r5, #0                      @ R5 = 'i' (Índice del bucle interno)
40
41     bucle_interno:
42         cmp r5, r4                  @ Compara el índice interno 'i' con 'N
43             ,
44         beq bucle_externo           @ Si i == N, terminó esta pasada,
45             regresa al bucle externo
46
47         ldr r6, [r1]                 @ R6 = B[i] (Valor actual)
48         ldr r7, [r1, #4]              @ R7 = B[i+1] (Valor adyacente derecho
49             )
50
51         cmp r6, r7                  @ Comparamos si el actual es mayor que
52             el derecho
53         ble no_cambiar            @ Branch if Less or Equal: Si B[i] <=
54             B[i+1] están bien, no cambies
55
56         @ Si llegamos aquí, B[i] es mayor, tenemos que hacer INTERCAMBIO
57             (Swap)
58         str r7, [r1]                 @ Escribimos el valor menor (R7) en la
59             posición izquierda B[i]
60         str r6, [r1, #4]              @ Escribimos el valor mayor (R6) en la
61             posición derecha B[i+1]
62
63     no_cambiar:

```

```

52      add r1, r1, #4          @ Avanzamos el puntero de memoria para
53      evaluar los siguientes
54      add r5, r5, #1          @ i++
55      b bucle_interno        @ Repetimos el bucle interno
56
57 fin_ordenamiento:
58     MOV R7, #1              @ sys_exit
59     SVC 0                  @ Fin

```

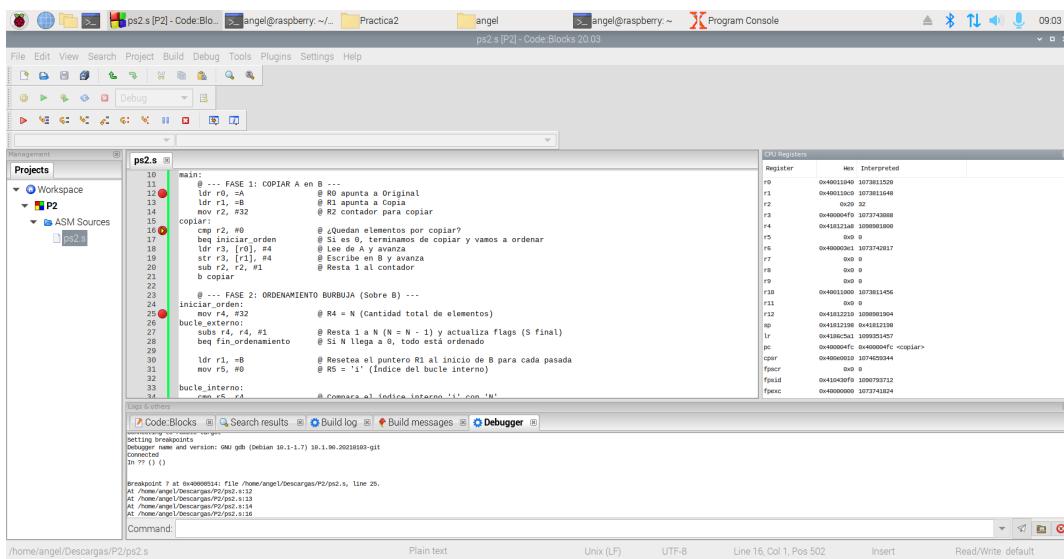


Figura 12: Fase 1 (Copiado). Inicialización de los apuntadores base para el arreglo original (R0) y el arreglo copia (R1).



The screenshot shows the Code::Blocks IDE interface with the assembly file `ps2.s` open. The assembly code is as follows:

```
main:    ; -- FASE 1: COPIAR A en B --
        ldr r0, =A          ; R0 apunta a Original
        ldr r1, =B          ; R1 apunta a Copia
        mov r2, #32         ; R2 contador para copiar
        copiar:             ; 
            cmp r2, #0        ; @ Quedan elementos por copiar?
            bne fin_ordenado ; @ Si es 0, terminamos de copiar y vamos a ordenar
            ldr r3, [r0], #4   ; Lee de A y avanza
            str r3, [r1], #4   ; Escribe en B y avanza
            sub r2, #1         ; Resta 1 al contador
            b copiar           ; 
        ; -- FASE 2: ORDENAMIENTO BURBUJA (Sobre B) --
        iniciar_orden:      ; 
            mov r4, #32         ; R4 = N (Cantidad total de elementos)
            bucle_externo:     ; 
                subs r4, r2, #1 ; Resta 1 a N (N - 1) y actualiza flags (S Final)
                beq fin_ordenado ; @ Si N llega a 0, todo està ordenado
                ldr r1, =B          ; 
                mov r5, #0          ; Resetea el puntero R1 al inicio de B para cada pasada
                bucle_interno:     ; 
                    cmn r5, r4       ; @ Compara el indice interno 'i' con 'N'
```

The CPU Registers window shows the following values at the start of the first iteration:

Register	Hex	Interpreted
r0	0x40001044	1073811234
r1	0x40011044	1073811652
r2	0x00 32	32
r3	0x00 32	32
r4	0x41010000	1098981800
r5	0x40000000	0
r6	0x40000001	1073742817
r7	0x00 0	0
r8	0x00 0	0
r9	0x00 0	0
r10	0x40011000	1073811456
r11	0x41010000	1098981800
r12	0x41010000	1098981800
sp	0x41010100	1098981208
lr	0x410c0001	1098981207
pc	0x40000000	0x40000000 <copiar>+10
cpar	0x20000010	107378432
fpscr	0x00 0	0
fpcsr	0x41010000	1098983712
fpcsc	0x40000000	1073741824

Figura 13: Primera iteración del ciclo de copiado. El registro R3 extrae el primer valor a transferir.

The screenshot shows the Code::Blocks IDE interface with the assembly file `ps2.s` open. The assembly code is identical to Figure 13, but the CPU Registers window shows the state after five iterations of the copy loop.

The CPU Registers window shows the following values at the start of the fifth iteration:

Register	Hex	Interpreted
r0	0x40011044	1073811234
r1	0x40011044	1073811652
r2	0x00 28	28
r3	0x00 28	28
r4	0x41012000	1098981800
r5	0x00 0	0
r6	0x40000001	1073742817
r7	0x00 0	0
r8	0x00 0	0
r9	0x00 0	0
r10	0x40011000	1073811456
r11	0x41010000	1098981800
r12	0x41010000	1098981800
sp	0x41010100	1098981208
lr	0x410c0001	1098981207
pc	0x40000000	0x40000000 <copiar>+10
cpar	0x20000010	107378432
fpscr	0x00 0	0
fpcsr	0x41010000	1098983712
fpcsc	0x40000000	1073741824

Figura 14: Avance del ciclo de copiado en la iteración correspondiente al quinto elemento transferido ( $R2 = 0x1C$ ).

```

ps2.s
10 main: ... FASE 1: COPIAR A en B ...
11    ldr r0, =A          @ R0 apunta a Original
12    ldr r1, =B          @ R1 apunta a Copia
13    mov r2, #32         @ R2 contador para copiar
14    mov r3, r2
15    copiar:
16        cmp r2, #0        @ Quedan elementos por copiar?
17        bne fin_orden    @ Si es cero terminamos de copiar y vamos a ordenar
18        ldr r3, [r0], #4    @ Lee de A y avanza
19        str r3, [r1], #4    @ Escribe en B y avanza
20        add r3, r1, r1      @ Resta 1 al contador
21        b copiar
22
23    ... FASE 2: ORDENAMIENTO BURBUJA (Sobre B) ...
24    inicial_orden:
25        mov r4, #32         @ R4 = N (Cantidad total de elementos)
26        bule_externo:
27            subs r4, r4, #1   @ Resta 1 a N (N - 1) y actualiza flags (S final)
28            beq fin_ordenamiento
29            ldr r1, =B
30            mov r5, #0
31            sub r2, r2, #1
32            bule_interno:
33                cmp r5, r4
34                bne bucle_externo
35            ldr r2, =r1
36            ldr r3, =r1
37            ldr r4, =r1
38
39    .compara_a1 indice_interno '1' con 'N'
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
908
909
910
911
912
913
914
915
916
917
917
918
919
920
921
922
923
924
925
926
927
927
928
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1066
1067
1067
1068
1069
1069
1070
1071
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1112
1113
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1
```



The screenshot shows the Code::Blocks IDE interface with the following details:

- File Menu:** File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, Help.
- Project Manager:** Projects, Workspace, P2, ASM Sources, ps2.s.
- Code Editor:** The main editor displays assembly code for file `ps2.s`. The code includes comments explaining the logic of the program, such as resetting pointers, comparing indices, and handling loops. It also includes assembly instructions like `ldr`, `cmp`, and `str`.
- Registers Window:** Shows the CPU registers (R0-R15, PC, SP) and their hex/interpreted values. For example, R0 is 0x40011000 and R1 is 0x40011004.
- Code Block Status Bar:** Shows tabs for Code, Results, Search results, Build log, Build messages, and Debugger.
- Command Line:** At the bottom, the terminal window shows the command `arm-none-eabi-objdump -D ps2.s` being run.

Figura 17: Carga de elementos adyacentes en R6 y R7 para su respectiva comparación.

Figura 18: Ejecución del intercambio (*swap*) al detectar que el elemento izquierdo es mayor que el derecho.



The screenshot shows the Code::Blocks IDE interface with the following details:

- Title Bar:** ps2.s [P2] - Code.Blo... > angel@raspberry:~/... Practica2 angel > angel@raspberry:~ X Program Console
- Menu Bar:** File Edit View Search Project Build Debug Tools Plugins Settings Help
- Toolbar:** Includes icons for New, Open, Save, Print, Run, Stop, and Break.
- Project Explorer:** Shows a workspace named "P2" containing "ASM Sources" and "ps2.s".
- Code Editor:** Displays assembly code for a bubble sort algorithm. The code includes comments explaining the logic of comparing adjacent elements and swapping them if they are in the wrong order. It also handles boundary conditions and loops.
- Registers Window:** Shows the CPU registers (r9 to r15) and their corresponding memory addresses and values.
- Stack Window:** Shows the stack contents, including the current PC value.
- Output Window:** Displays assembly language mnemonics and their corresponding opcodes.

Figura 19: Actualización del apuntador interno R1 y del contador interno R5 antes de la siguiente comparación.

The screenshot shows the Code::Blocks IDE interface with multiple windows open. The main window displays assembly code for a swap operation between memory locations B[1] and B[2]. The assembly code includes comments in Spanish explaining the logic: comparing indices, handling boundary conditions, and performing the swap. Below the assembly code, there is a register dump table showing the state of various CPU registers (R0-R12, SP, PC, CCR, FPCR, FPSID, FPCEx) at the current instruction. The bottom of the screen shows the terminal window with the assembly code and the command 'at \$0x40000000' entered.

Register	Hex	Interpreted
R0	0x40011000	0x73011400
R1	0x40011004	0x73011000
R2	0x0	
R3	0x1	
R4	0x0	
R5	0x1	
R6	0x2	
R7	0x1	
R8	0x0	
R9	0x0	
R10	0x40011000	0x73011400
R11	0x0	
R12	0x1B	0x1B
SP	0x41812210	1098901904
PC	0x40000000	0x40000000 <fn_ordenamiento>
CCR	0x00000010	1611530256
FPCR	0x0	
FPSID	0x0	
FPCEx	0x40000000	1097373712

Figura 20: Finalización del programa. Los contadores de ambos bucles agotaron sus iteraciones.

## Análisis de resultados

La ejecución inicia configurando la transferencia de datos. Las instrucciones LDR R0, =A y LDR R1, =B asignan las direcciones base de ambos arreglos. Como se observa en la primera



captura, R0 almacena la dirección 0x40011040 (inicio de A) y R1 almacena 0x400110C0 (inicio de B). Se establece el límite de 32 elementos en R2 (0x20). Al entrar a la etiqueta **copiar**, la instrucción LDR R3, [R0], #4 carga el primer elemento del arreglo (0x20 o 32 en decimal) en R3 y avanza el apuntador original a 0x40011044. Seguidamente, STR R3, [R1], #4 guarda este valor en la dirección apuntada por R1 y desplaza dicho apuntador a 0x400110C4. Este avance coordinado de ambos apuntadores se mantiene constante, comprobándose en las capturas 2 y 3, hasta que el contador R2 llega a cero, momento en el que R1 alcanza la dirección 0x40011140 (exactamente 128 bytes de desplazamiento, calculados como  $32 \times 4$ ).

Una vez completada la copia intacta, inicia la fase de ordenamiento bajo la etiqueta **iniciar\_orden**. Se carga la constante de 32 elementos en R4 para fungir como el límite decreciente del bucle externo. La instrucción SUBS R4, R4, #1 resta una unidad en cada pasada general y actualiza la bandera de estado para verificar si se alcanzó el límite. En la captura 5, se observa que R4 adquiere el valor de 0x1F (31). Es vital notar la instrucción LDR R1, =B, la cual re-inicializa el apuntador R1 a la dirección base 0x400110C0 al inicio de cada pasada, mientras que R5 se limpia con 0x0 para funcionar como el índice del bucle interno.

Dentro del **bucle\_interno**, las instrucciones LDR R6, [R1] y LDR R7, [R1, #4] acceden simultáneamente a dos valores adyacentes en la memoria de la copia. En la primera evaluación (captura 6), R6 carga el valor 0x20 (32) correspondiente a B[0], y R7 carga 0x1F (31) correspondiente a B[1]. La instrucción CMP R6, R7 compara ambos registros. Puesto que 32 es mayor que 31, no se cumple la condición BLE **no\_cambiar**, por lo que el programa prosigue a intercambiar los datos en memoria.

El intercambio se efectúa de manera cruzada: la instrucción STR R7, [R1] toma el valor menor (31) y lo sobreescribe en la dirección inferior, mientras que STR R6, [R1, #4] toma el valor mayor (32) y lo aloja en la dirección superior inmediata. Esto se observa en progreso en la captura 7. Después del intercambio, bajo la etiqueta **no\_cambiar**, se incrementa el apuntador base con ADD R1, R1, #4 (desplazando a 0x400110C4 como se ve en la captura 8) y se aumenta el índice interno R5 en una unidad. Este patrón de comparación y desplazamiento en la memoria se repite iterativamente hasta que el bucle interno alcanza al bucle externo, empujando los valores más altos hacia las direcciones de memoria más elevadas, lo que a nivel del procesador cumple satisfactoriamente el algoritmo de burbuja y satisface todas las directrices establecidas.



---

## 2. Conclusiones:

- Espinoza Matamoros Percival Ulises:
- Flores Colin Victor Jaziel:
- Lara Hernandez Angel Husiel:



---

## Referencias

- Anaya, R. (s.f.). *Manual de recursos y aplicaciones Plataforma Raspberry Pi*. <https://odin.filb.unam.mx/micros/docs/Tutoriales%20Raspberry.pdf>
- Elahi, A. (2022, 17 de marzo). *Computer systems: Digital Design, Fundamentals of Computer Architecture and ARM Assembly Language* (2.<sup>a</sup> ed.). Springer. <https://doi.org/10.1007/978-3-030-93449-1>
- Harris, D., & Harris, S. (2015, 22 de abril). *Digital Design and Computer Architecture* (Arm Edition). Morgan Kaufmann Pub.
- Smith, S. (2019, octubre). *Raspberry Pi Assembly Language Programming*. Apress. <https://doi.org/10.1007/978-1-4842-5287-1>