

# 04 Lenguaje ensamblador, svc y transferencia



Arquitectura de Computadoras y Ensambladores 1  
M.Sc. Luis Fernando Espino Barrios  
2024

# Directivas

- Declaración de símbolos globales: `.global` o `.globl`
- Declaración de código en el segmento text: `.text`
- Declaración de datos en el segmento data: `.data`
- Declaración de cadenas: `.ascii` o `.asciz`

# Etiquetas

- Las etiquetas son nombres de una ubicación específica del código.
- En la llamada solo se escribe el nombre y en la ubicación se agregan (:) dos puntos al final.
- Entre algunas ubicaciones importantes están:
  - Inicio del código: `_start`
  - Puntos de referencia
  - Inicio o fin de ciclos
  - Destino de saltos condicionales e incondicionales.

ASM 01\_test.s

1     .global \_start

2

3     \_start:

4         mov x0, 5         // return value

5         mov x8, 93        // exit syscall\_num

6         svc 0             // generic syscall



luisespino@raspberrypi: ~/nfs



```
luisespino@raspberrypi:~/nfs $ ./01_test ; echo $?
```

```
5
```

```
luisespino@raspberrypi:~/nfs $
```



# Carga y almacenamiento de registro simple

Instrucción	Descripción
ldr Rd, <addr>	Carga un registro de la memoria (puede usar una dirección, un offset, una etiqueta, un inmediato o un símbolo)
str Rd, <addr>	Almacena un registro en la memoria (puede usar una dirección, un offset, una etiqueta, un inmediato o un símbolo)

# Operaciones de movimiento de datos

Instrucción	Descripción
mov Rd, Rn	Mueve (copia) un registro Rn a otro Rd (puede usar un registro Rn, un inmediato o un patrón)
movz Rd, imm{, lsl <shift>}	Establece un inmediato con corrimiento y llenado de ceros hacia un registro Rd.
movn Rd, imm{, lsl <shift>}	Igual que movz solo que negado o complemento a 1.
movk Rd, imm{, lsl <shift>}	Establece un inmediato con corrimiento hacia un registro Rd, dejando el resto igual.

ASM 02\_print.s

```
1  .global _start
2
3  _start:
4      mov x0, 1          // set stdout
5      ldr x1, =msg        // load msg
6      mov x2, 14          // size msg
7      mov x8, 64          // write syscall_num
8      svc 0              // generic syscall
9
10     mov x0, 0            // return value
11     mov x8, 93           // exit syscall_num
12     svc 0               // generic syscall
13
14     .data
15     msg: .asciz "Hello, World!\n"
```





luisespino@raspberrypi: ~/nfs



```
luisespino@raspberrypi:~/nfs $ ./02_print
```

```
Hello, World!
```

```
luisespino@raspberrypi:~/nfs $
```

# Otras directivas

- Declaración de una sección específica: `.section`
- Declaración de constantes simbólicas: `.equ`
- Declaración de datos no inicializados: `.bss`
- Declaración de espacio para almacenar: `.space`

```

1  .global _start
2
3  _start:
4      mov x0, 1      // set stdout
5      ldr x1, =msg    // load msg
6      mov x2, 18     // size msg
7      mov x8, 64     // write syscall_num
8      svc 0          // generic syscall
9
10     mov x0, 0       // set stdin
11     ldr x1, =buf    // load buffer
12     mov x2, 16     // size buffer
13     mov x8, 63     // read syscall_num
14     svc 0          // generic syscall
15
16     mov x0, 1       // set stdout
17     ldr x1, =buf    // load buffer
18     mov x8, 64     // write syscall_num
19     svc 0          // generic syscall
20
21     mov x0, 0       // return value
22     mov x8, 93     // exit syscall_num
23     svc 0          // generic syscall
24
25     .data
26     msg:
27         .ascii "Enter a text: "
28
29     .bss
30     buf:
31         .space 16

```



luisespino@raspberrypi: ~/nfs



```
luisespino@raspberrypi:~/nfs $ ./03_read
```

```
Enter a text: Hola
```

```
Hola
```

```
luisespino@raspberrypi:~/nfs $
```



# Ejercicios

- Realizar el proceso de copiado de código, ensamblado, enlazado y ejecución de los tres ejemplos vistos.
- Puede realizarse en Qemu, en UserLAnd o físicamente en la Raspberry.

# Bibliografía

- Elahi, A. (2022). Fundamentals of Computer Architecture and ARM Assembly Language. Springer.
- Harris, S. & Harris, D. (2016). Digital Design and Computer Architecture: ARM Edition. Elsevier Inc.
- Mazidi, M. & Otros. (2013). ARM Assembly Language: Programming & Architecture.
- Smith, S. (2019). Raspberry Pi Assembly Language Programming: ARM Processor Coding. Apress.