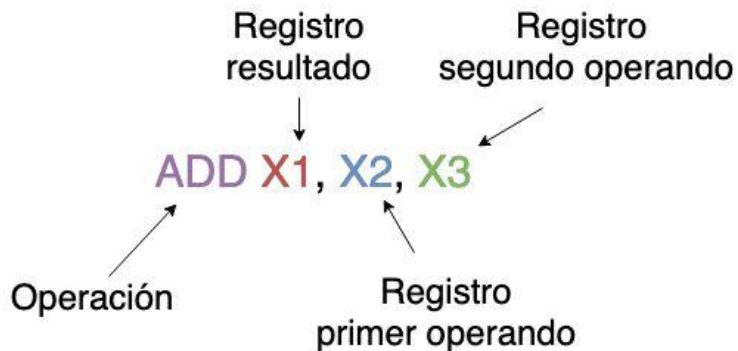


LEGv8 básico

OdC - 2020

Lenguaje ensamblador



Si se realiza la siguiente asignación de registros:

`X1 → a`
`X2 → b`
`X3 → c`

Esta instrucción podría escribirse en lenguaje C como:

`a = b + c`

Conjunto de instrucciones - Aritmética

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	ADD X1, X2, X3	$X1 = X2 + X3$	Three register operands
	subtract	SUB X1, X2, X3	$X1 = X2 - X3$	Three register operands
	add immediate	ADDI X1, X2, 20	$X1 = X2 + 20$	Used to add constants
	subtract immediate	SUBI X1, X2, 20	$X1 = X2 - 20$	Used to subtract constants
	add and set flags	ADDs X1, X2, X3	$X1 = X2 + X3$	Add, set condition codes
	subtract and set flags	SUBS X1, X2, X3	$X1 = X2 - X3$	Subtract, set condition codes
	add immediate and set flags	ADDIS X1, X2, 20	$X1 = X2 + 20$	Add constant, set condition codes
	subtract immediate and set flags	SUBIS X1, X2, 20	$X1 = X2 - 20$	Subtract constant, set condition codes

Figura 2.1 - Computer Organization and design, Arm Edition - Patterson & Hennessy

Conjunto de instrucciones - Lógica

Category	Instruction	Example	Meaning	Comments
Logical	and	AND X1, X2, X3	$X1 = X2 \& X3$	Three reg. operands; bit-by-bit AND
	inclusive or	ORR X1, X2, X3	$X1 = X2 X3$	Three reg. operands; bit-by-bit OR
	exclusive or	EOR X1, X2, X3	$X1 = X2 \wedge X3$	Three reg. operands; bit-by-bit XOR
	and immediate	ANDI X1, X2, 20	$X1 = X2 \& 20$	Bit-by-bit AND reg. with constant
	inclusive or immediate	ORRI X1, X2, 20	$X1 = X2 20$	Bit-by-bit OR reg. with constant
	exclusive or immediate	EORI X1, X2, 20	$X1 = X2 \wedge 20$	Bit-by-bit XOR reg. with constant
	logical shift left	LSL X1, X2, 10	$X1 = X2 \ll 10$	Shift left by constant
	logical shift right	LSR X1, X2, 10	$X1 = X2 \gg 10$	Shift right by constant

Figura 2.1 - Computer Organization and design, Arm Edition - Patterson & Hennessy

Instalación QEMU

0-TENER ACTUALIZADOS LOS REPOSITORIOS

```
$ sudo apt update
```

1- SETTING UP AARCH64 TOOLCHAIN

```
$ sudo apt install gcc-aarch64-linux-gnu
```

2- SETTING UP QEMU ARM (incluye aarch64)

```
$ sudo apt install qemu-system-arm
```

3- FETCH AND BUILD AARCH64 GDB

```
$ sudo apt install gdb-multiarch
```

4- CONFIGURAR GDB PARA QUE HAGA LAS COSAS MÁS AMIGABLES

```
$ wget -P ~ git.io/.gdbinit
```

Ensamblado y simulación (1/4)

El código a simular se debe escribir en el archivo main.s

```
.data  
b: .dword 15  
c: .dword 2
```



Segmento de datos

```
.text  
LDR X2, b  
LDR X3, c  
ADD X1, X2, X3
```



Segmento de código

Ensamblado y simulación (2/4)

Ensamblado

```
$ make
```

```
aarch64-linux-gnu-as -g --warn --fatal-warnings main.s -o main.o
```

```
aarch64-linux-gnu-ld main.o -T memmap -o main.elf -M > memory_map.txt
```

```
aarch64-linux-gnu-objdump -D main.elf > main.list
```

```
aarch64-linux-gnu-objcopy main.elf -O ihex main.hex
```

```
aarch64-linux-gnu-objcopy main.elf -O binary kernel.img
```

Inicio del simulador

```
$ qemu-system-aarch64 -s -S -machine virt -cpu cortex-a53 -machine type=virt  
-nographic -smp 1 -m 64 -kernel kernel.img
```

Nota: Al ejecutar este comando, la terminal queda ejecutándolo. Para continuar con la ejecución del dashboard, se debe abrir una nueva terminal.

Ensamblado y simulación (3/4)

Inicio del debugger GDB

```
$ gdb-multiarch -ex "set architecture aarch64" -ex "target remote  
localhost:1234"
```

Configurar la arquitectura a utilizar

```
>>> set architecture aarch64
```

Importar al GDB los símbolos de debug en la dirección de memoria donde se encuentra el programa

```
>>> add-symbol-file main.o 0x0000000040080000
```


Ensamblado y simulación (4/4)

```
Output/messages
Assembly
0x0000000040080004 ? ldr    x3, 0x40080014
0x0000000040080008 ? add    x1, x2, x3
0x000000004008000c ? .inst  0x0000000f ; undefined
0x0000000040080010 ? .inst  0x00000000 ; undefined
0x0000000040080014 ? .inst  0x00000002 ; undefined
0x0000000040080018 ? .inst  0x00000000 ; undefined
0x000000004008001c ? .inst  0x00000000 ; undefined
Expressions
History
Memory
Registers
x0 0x0000000042000000   x1 0x0000000000000000   x2 0x000000000000000f   x3 0x0000000000000000   x4 0x0000000040080000   x5 0x0000000000000000
x8 0x0000000000000000   x9 0x0000000000000000   x10 0x0000000000000000  x11 0x0000000000000000  x12 0x0000000000000000  x13 0x0000000000000000
x16 0x0000000000000000  x17 0x0000000000000000  x18 0x0000000000000000  x19 0x0000000000000000  x20 0x0000000000000000  x21 0x0000000000000000
x24 0x0000000000000000  x25 0x0000000000000000  x26 0x0000000000000000  x27 0x0000000000000000  x28 0x0000000000000000  x29 0x0000000000000000
pc 0x0000000040080004  cpsr 0x400003c5        fpsr 0x00000000        fpcr 0x00000000
Source
2 b: .dword 15
3 c: .dword 2
4
5 .text
6 LDR X2, b
7 LDR X3, c
8 ADD X1, X2, X3
Stack
[0] from 0x0000000040080004
(no arguments)
Threads
[1] id 1 from 0x0000000040080004

7 LDR X3, c
>>> 
```

Ejercicio 1

Dadas las siguientes sentencias en “C”:

a) $f = g + h + i + j$;

b) $f = g + (h + 5)$;

c) $f = (g + h) + (g + h)$;

1.1) Escribir la secuencia **mínima** de código assembler LEGv8 asumiendo que f, g, h, i y j se asignan en los registros X0, X1, X2, X3 y X4 respectivamente.

1.2) Dar el valor de cada variable en cada instrucción assembler si f, g, h, i y j se inicializan con valores de 1, 2, 3, 4, 5, en base 10, respectivamente.

Resolución ejercicio 1.1-a

Para poder resolver esta sentencia de C con instrucciones de LEGv8:

$f = g + h + i + j;$

Es necesario descomponerla en sentencias de dos operandos:

$f = g + h;$

$f = f + i;$

$f = f + j;$

Dado que f, g, h, i y j se asignan en los registros X0, X1, X2, X3 y X4 respectivamente, el código es:

ADD X0, X1, X2

ADD X0, X0, X3

ADD X0, X0, X4

Resolución ejercicio 1.2-a con QEMU (1/2)

Si f, g, h, i y j se inicializan con valores de 1, 2, 3, 4, 5

X0 \leftarrow f // f \leftarrow 1

X1 \leftarrow g // g \leftarrow 2

X2 \leftarrow h // h \leftarrow 3

X3 \leftarrow i // i \leftarrow 4

X4 \leftarrow j // j \leftarrow 5

ADD X0, X1, X2 // X0 = 2 + 3 = 5

ADD X0, X0, X3 // X0 = 5 + 4 = 9

ADD X0, X0, X4 // X0 = 9 + 5 = 14

Resolución ejercicio 1.2-a con QEMU (1/2)

```
.data
```

```
f: .dword 1
```

```
g: .dword 2
```

```
h: .dword 3
```

```
i: .dword 4
```

```
j: .dword 5
```

↓

```
.text
```

```
LDR X0, f
```

```
LDR X1, g
```

```
LDR X2, h
```

```
LDR X3, i
```

```
LDR X4, j
```

```
ADD X0, X1, X2
```

```
ADD X0, X0, X3
```

```
ADD X0, X0, X4
```

```
infloop: B infloop
```

Resolución ejercicio 1.1-b

Para poder resolver esta sentencia de C con instrucciones de LEGv8:

$f = g + (h + 5);$

Es necesario descomponerla en sentencias de dos operandos:

$f = h + 5;$

$f = f + g;$

Dado que f, g, h, i y j se asignan en los registros X0, X1, X2, X3 y X4 respectivamente, el código es:

ADDI X0, X2, #5

ADD X0, X0, X1

Resolución ejercicio 1.2-b con QEMU

```
.data
```

```
f: .dword 1
```

```
g: .dword 2
```

```
h: .dword 3
```

```
i: .dword 4
```

```
j: .dword 5
```

↓

```
.text
```

```
LDR X0, f
```

```
LDR X1, g
```

```
LDR X2, h
```

```
LDR X3, i
```

```
LDR X4, j
```

```
ADDI X0, X2, #5
```

```
ADD X0, X0, X1
```

```
inloop: B inloop
```

Ejercicio 2

Luego, dadas las siguientes sentencias en assembler LEGv8:

a) `ADD X0, X1, X2`

b) `ADDI X0, X0, #1`

`ADD X0, X1, X2`

2.1) Escribir la secuencia mínima de código “C” asumiendo que los registros X0, X1 y X2 contienen las variables f, g y h respectivamente.

2.2) Dar el valor de cada variable en cada instrucción assembler si f, g y h se inicializan con valores de 1, 2, 3, en base 10, respectivamente.

Ejercicio 2.1

a) `ADD X0, X1, X2` `// f = g + h`

`f = g + h`

b) `ADDI X0, X0, #1` `// f = f + 1`

`ADD X0, X1, X2` `// f = g + h`

`f = g + h`

Ejercicio 3

Ejercicio 3:

Dadas las siguientes sentencias en “C”:

a) $f = -g - f;$

b) $f = g + (-f - 5);$

3.1) Escribir la secuencia mínima de código assembler LEGv8 asumiendo que f y g se asignan en los registros X0 y X1 respectivamente.

3.2) Dar el valor de cada variable en cada instrucción assembler si f y g se inicializan con valores de 4 y 5, en base 10, respectivamente.

Ejercicio 3.1

a) $f = -g - f;$
 $f = -(g + h);$

ADD X0, X1, X2
SUB X0, XZR, X0

b) $f = g + (-f - 5);$
 $f = g - (f + 5);$

ADDI X0, X0, #5
SUB X0, X1, X0