

# Ensamblado y desensamblado de LEGv8

Parte 2

OdC - 2020

# Conjunto de instrucciones - Saltos

Conditional branch	compare and branch on equal 0	CBZ X1, 25	if (X1 == 0) go to PC + 100	Equal 0 test; PC-relative branch
	compare and branch on not equal 0	CBNZ X1, 25	if (X1 != 0) go to PC + 100	Not equal 0 test; PC-relative branch
	branch conditionally	B.cond 25	if (condition true) go to PC + 100	Test condition codes; if true, branch
Unconditional branch	branch	B 2500	go to PC + 10000	Branch to target address; PC-relative
	branch to register	BR X30	go to X30	For switch, procedure return
	branch with link	BL 2500	X30 = PC + 4; PC + 10000	For procedure call PC-relative

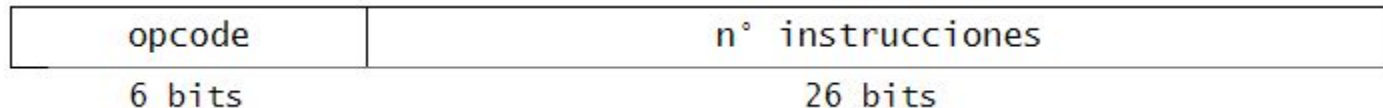
# Campos de LEGv8 - Instrucción tipo B

B loop // go to “loop”

- Instruction set: Branch B B 0A0-0BF PC = PC + BranchAddr (3,9)
- Instruction format: B 

opcode	BR_address
31 26 25	0
- Note: (3) BranchAddr = { 36{BR\_address [25]}, BR\_address, 2'b0 }

Since all LEGv8 instructions are 4 bytes long, LEGv8 stretches the distance of the branch by having PC-relative addressing refer to the number of words to the next instruction instead of the number of bytes.



# Campos de LEGv8 - Instrucciones CBZ y CBNZ (CB)

CBNZ X19, Exit // go to Exit if X19  $\neq$  0

Unlike the branch instruction, a conditional branch instruction can specify one operand in addition to the branch address, leaving only 19 bits for the branch address.

- |                    |                              |      |    |         |  |
|--------------------|------------------------------|------|----|---------|--|
| ● Instruction set: | Compare & Branch if Not Zero | CBNZ | CB | 5A8-5AF | if(R[Rt] $\neq$ 0)<br>PC = PC + CondBranchAddr |
|                    | Compare & Branch if Zero     | CBZ  | CB | 5A0-5A7 | if(R[Rt]==0)<br>PC = PC + CondBranchAddr       |

- Instruction format:
 

CB	Opcode	COND_BR_address	Rt
31	24 23	5 4	0

- Note: (4) CondBranchAddr = { 43 {COND\_BR\_address ~~18~~ }, COND\_BR\_address, 2'b0 }

opcode	n° instrucciones	Rt
8 bits	19 bits	5 bits

# Campos de LEGv8 - Instrucciones B.cond (CB)

The conditional branch instructions that rely on condition codes also use the CB-type format, but they use the final field to select among the many possible branch conditions.

Branch conditionally      B.cond      CB      2A0-2A7      if(FLAGS==cond)  
PC = PC + CondBranchAddr      (4,9)

Instruction	Rt [4:0]	Instruction	Rt [4:0]
B.EQ	00000	B.VC	00111
B.NE	00001	B.HI	01000
B.HS	00010	B.LS	01001
B.LO	00011	B.GE	01010
B.MI	00100	B.LT	01011
B.PL	00101	B.GT	01100
B.VS	00110	B.LE	01101

opcode	n° instrucciones	condición
8 bits	19 bits	5 bits

# Ejercicio 7

Ensamblar estos delay loops:

<pre>MOVZ X0, 0x1, LSL #48 L1: SUBI X0,X0,#1     CBNZ X0, L1</pre>	<pre>MOVZ X0, 0xFFFF, LSL #32 L1: SUBIS X0,X0,#1     B.NE L1</pre>	<pre>MOVZ X0, 0x2, LSL #16 L1:  SUBIS XZR,X0,#0     B.EQ EXIT     SUBI X0,X0,#1     B L1 EXIT:</pre>
--	--	--

# Ejercicio 7-a

```
MOVZ X0, 0x1, LSL #48
L1: SUBI X0,X0,#1
    CBNZ X0, L1
```

Usamos el main.list a modo demostrativo, pero el proceso de ensamblado es el que vimos en la "Parte 1"

De main.list:

Disassembly of section .text:

0000000000000000 <L1-0x4>:

0: d2e00020 mov x0, #0x1000000000000

0000000000000004 <L1>:

4: d1000400 sub x0, x0, #0x1

8: b5ffffe0 cbnz x0, 4 <L1>

opcode	n° instrucciones	Rt
10110101	111111111111111111 (-1 <sub>10</sub> )	00000
8 bits	19 bits	5 bits

0b1011010111111111111111111111110000 = 0xb5ffffe0

# Ejercicio 7-b

```
MOVZ X0, 0xFFFF, LSL #32
L1: SUBIS X0,X0,#1
    B.NE L1
```

De `main.list`:

Disassembly of section `.text`:

0000000000000000 <L1-0x4>:

0: d2dffffe0 mov x0, #0xffff00000000

0000000000000004 <L1>:

4: f1000400 subs x0, x0, #0x1

8: 54ffffe1 b.ne 4 <L1>

opcode	n° instrucciones	condición
01010100	111111111111111111 (-1 <sub>10</sub> )	00001
8 bits	19 bits	5 bits

0b010101001111111111111111111100001 = 0x54ffffe1



## Ejercicio 7-c

```
MOVZ X0, 0x2, LSL #16
L1:  SUBIS XZR,X0,#0
      B.EQ EXIT
      SUBI X0,X0,#1
      B L1
EXIT:
```

De `main.list`:

Disassembly of section `.text`:

0000000000000000 <L1-0x4>:

0: d2a00040 mov x0, #0x20000

0000000000000004 <L1>:

4: f100001f cmp x0, #0x0 

8: 54000060 b.eq 14 <EXIT>

c: d1000400 sub x0, x0, #0x1

10: 17ffffff b 4 <L1>

14: <EXIT>

# Ejercicio 7-c

8:      **54000060**      b.eq **14** <EXIT>

MOVZ X0, 0x2, LSL #16  
L1: SUBIS XZR,X0,#0  
B.EQ EXIT  
SUBI X0,X0,#1  
B L1  
EXIT:

opcode	n° instrucciones	condición
01010100	00000000000000000011 ( $3_{10}$ )	00000
8 bits	19 bits	5 bits

0b01010100000000000000000000001100000 = **0x54000060**

10:      **17ffffffd**      b      **4** <L1>

opcode	n° instrucciones
000101	1111111111111111111111111101 ( $-3_{10}$ )
6 bits	26 bits

0b00010111111111111111111111111101 = **0x17ffffffd**

# Ejercicio 8

Dadas las siguientes direcciones de memoria:

0x00014000

0x00114524

0x0F000200

8.1) Si el valor del PC es 0x00000000, ¿es posible llegar con una sola instrucción **conditional branch** a las direcciones de memoria arriba listadas?

8.2) Si el valor del PC es 0x00000600, ¿es posible llegar con una sola instrucción **branch** a las direcciones de memoria arriba listadas?

8.3) Si el valor del PC es 0x00000000 y quiero saltar al primer GiB de memoria 0x40000000 . Escribir exactamente 2 instrucciones contiguas que posibilitan el **salto lejano** (far jump).

## Ejercicio 8.2

- Campo de inmediato de branch = 26 bits.

$$(3) \quad \text{BranchAddr} = \{ 36\{\text{BR\_address}[25]\}, \text{BR\_address}, 2'b0 \}$$

- Máxima cantidad de **instrucciones** “hacia adelante” =  $2^{25} - 1 = 0x1FF\ FFFF$
- Máxima cantidad de **posiciones de memoria** “hacia adelante” =  $0x7FF\ FFFC$

$$\text{Branch } B \quad B \quad 0A0-0BF \quad PC = PC + \text{BranchAddr} \quad (3,9)$$

- Partiendo de  $PC = 0x00000600$  se alcanza la dirección:

$$PC = 0x0000\ 0600 + 0x7FF\ FFFC = \mathbf{0x0800\ 05FC}$$

- $0x0001\ 4000$  es posible llegar con una sola instrucción
- $0x0011\ 4524$  es posible llegar con una sola instrucción
- $0x0F00\ 0200$  NO es posible llegar con una sola instrucción

## Ejercicio 8.3

Si el valor del PC es `0x00000000` y quiero saltar al primer GiB de memoria `0x40000000` . Escribir exactamente 2 instrucciones contiguas que posibilitan el **salto lejano** (far jump).

```
MOVZ X0, #0x4000, LSL 16  
BR X0
```

- Si el PC es distinto de `0x0` la respuesta es la misma!!
- Notar que BR se ensambla como instrucción tipo R (ver erratas).

### Branching Far Away

Given a branch on register `X19` being equal to register zero,

```
CBZ    X19, L1
```

replace it by a pair of instructions that offers a much greater branching distance.  
These instructions replace the short-address conditional branch:

```
CBNZ   X19, L2  
B      L1
```

L2:

# Ejercicio 9

Suponiendo que el PC está en la primera palabra de memoria  $0x00000000$  y se desea saltar a la última instrucción de los primeros 4 GiB o sea a  $0xFFFF\ FFFC$ , ¿Cuántas instrucciones B son necesarias? (no se puede usar BR).

Máxima cantidad de **posiciones de memoria** “hacia adelante” de B =  $0x7FF\ FFFC$

- 1) Partiendo de PC =  $0x0$  se alcanza la dirección =  $0x7FF\ FFFC$
- 2) PC =  $0x7FF\ FFFC + 0x7FF\ FFFC = 0xFFF\ FFF8$
- 3) PC =  $0xFFF\ FFF8 + 0x7FF\ FFFC = 17FF\ FFF4$
- 4) ...

$$\begin{aligned}\text{Cantidad de instrucciones B} &= 0xFFFF\ FFFC / 0x7FF\ FFFC \\ &= 4294967292 / 134217724 = 32,00000092 = \mathbf{33 \text{ instrucciones}}\end{aligned}$$

# Ejercicio 10

¿Qué valor devuelve en X0 este programa?

```
.org 0x0000  
MOVZ X0, 0x0400, LSL #0  
MOVK X0, 0x9100, LSL #16  
STURW X0, [XZR,#12]  
STURW X0, [XZR,#12]
```

# Ejercicio 10 - Self-modifying code

```
0: MOVZ X0, 0x0400, LSL #0    // X0 = 0x0000 0000 0000 0400
4: MOVK X0, 0x9100, LSL #16   // X0 = 0x0000 0000 9100 0400
8: STURW X0, [XZR,#12]        // Sobre-escribe la siguiente instrucción con (*)
12: STURW X0, [XZR,#12]
```

(\*) -  $0x91000400 = 0b10010001000000000000000100000000$

$0b10010001000 = 0x488 = \text{ADDI}$ , Tipo I:  $Rd = Rn + \text{Inmediato}$

-  $0x91000400 = 0b\ 1001000100\ 000000000001\ 00000\ 00000$

Opcode=  $1001000100$  (10 bits) = ADDI,

Inmediato=  $000000000001$  (12 bits),  $Rn = 00000 = X0$ ,  $Rd = 00000 = X0$

```
12: ADDI X0, X0, #1    // X0 = 0x0000000091000401
```



# Introducción a la Experiencia CARDIAC

[https://cs.famaf.unc.edu.ar/~nicolasw/cardiac\\_2.mp4](https://cs.famaf.unc.edu.ar/~nicolasw/cardiac_2.mp4)

# Bibliografía

Patterson and Hennessy, “Computer Organization and Design: The Hardware/Software Interface ARM Edition”, Morgan kaufmann, 2016.