

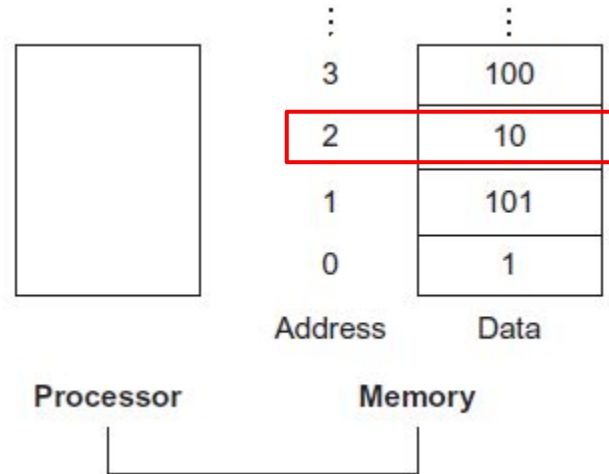
LEGv8 básico

Accesos a memoria

OdC 2025

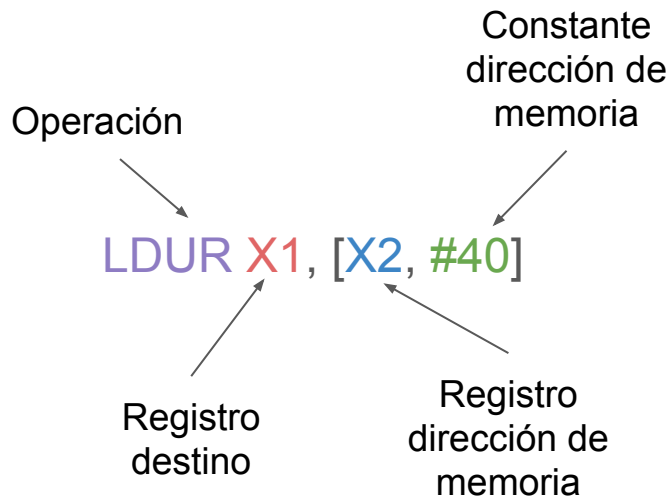
Accediendo a la memoria

La memoria es un gran array unidimensional, donde la dirección actúa como índice de ese array, comenzando en 0. Por ejemplo, en la figura la **dirección** del tercer elemento es 2 y el **contenido** o valor de la memoria es 10.



Instrucción Load (“cargar”)

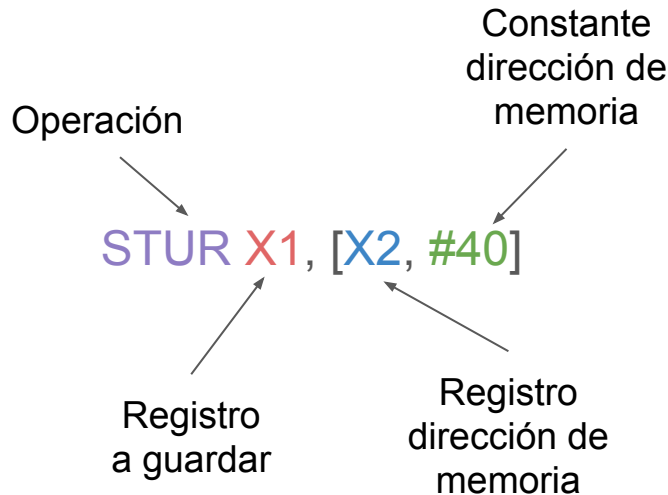
Copia al registro **X1** el contenido de la memoria direccionada por el contenido del registro **X2** sumado a la constante **#40**.



$$\mathbf{X1} = \text{Memory}[\mathbf{X2} + \mathbf{\#40}]$$

Instrucción Store (“guardar”)

Copia el contenido del registro **X1** en la posición de memoria direccionada por el contenido del registro **X2** sumado a la constante **#40**.



$$\text{Memory}[\text{X2} + \text{\#40}] = \text{X1}$$

Tamaño de palabra

DATA ALIGNMENT

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

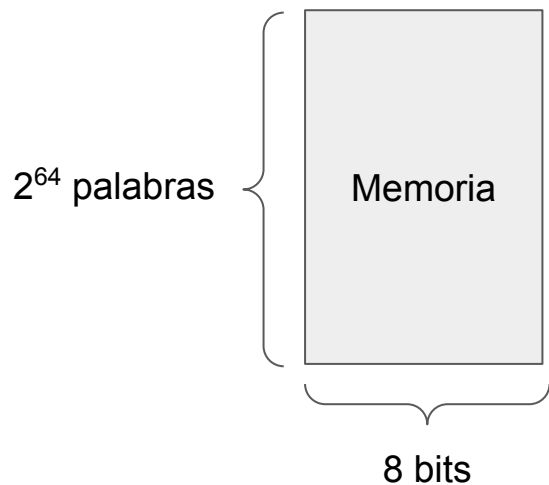
- Tamaño de un registro = 64 bits = Double Word
- Address para load y store = 64 bits = Double Word

Conjunto de instrucciones - Transferencia de datos

Data transfer	load register	LDUR X1, [X2,40]	$X1 = \text{Memory}[X2 + 40]$	Doubleword from memory to register
	store register	STUR X1, [X2,40]	$\text{Memory}[X2 + 40] = X1$	Doubleword from register to memory
	load signed word	LDURSW X1, [X2,40]	$X1 = \text{Memory}[X2 + 40]$	Word from memory to register
	store word	STURW X1, [X2,40]	$\text{Memory}[X2 + 40] = X1$	Word from register to memory
	load half	LDURH X1, [X2,40]	$X1 = \text{Memory}[X2 + 40]$	Halfword memory to register
	store half	STURH X1, [X2,40]	$\text{Memory}[X2 + 40] = X1$	Halfword register to memory
	load byte	LDURB X1, [X2,40]	$X1 = \text{Memory}[X2 + 40]$	Byte from memory to register
	store byte	STURB X1, [X2,40]	$\text{Memory}[X2 + 40] = X1$	Byte from register to memory
	load exclusive register	LDXR X1, [X2,0]	$X1 = \text{Memory}[X2]$	Load; 1st half of atomic swap
	store exclusive register	STXR X1, X3 [X2]	$\text{Memory}[X2] = X1; X3 = 0 \text{ or } 1$	Store; 2nd half of atomic swap
	move wide with zero	MOVZ X1, 20, LSL 0	$X1 = 20 \text{ or } 20 * 2^{16} \text{ or } 20 * 2^{32} \text{ or } 20 * 2^{48}$	Loads 16-bit constant, rest zeros
	move wide with keep	MOVK X1, 20, LSL 0	$X1 = 20 \text{ or } 20 * 2^{16} \text{ or } 20 * 2^{32} \text{ or } 20 * 2^{48}$	Loads 16-bit constant, rest unchanged

Dimensiones de la memoria

En LEGv8 la memoria se direcciona de a byte:



Ejemplo:

Dirección	Contenido
0	0x0A
1	0x0B
2	0x0C
3	0x0D
4	0x0E
5	0x0F
6	0x10
7	0x11
8	0x12
9	0x13
10	0x14
11	0x15
12	0x16
13	0x17
14	0x18
15	0x19
16	0x1A
...	...

Acceso a memoria de byte (LDURB/STURB)

Dirección	Contenido
0	0x0A
1	0x0B
2	0x0C
3	0x0D
4	0x0E
5	0x0F
6	0x10
7	0x11
8	0x12
9	0x13
10	0x14
11	0x15
12	0x16
13	0x17
14	0x18
15	0x19
16	0x1A
...	...

LDURB X1, [XZR, #0] // X1 = 0x00000000000000000A
ceros

LDURB X1, [XZR, #3] // X1 = 0x00000000000000000D
ceros

LDURB X1, [XZR, #10] // X1 = 0x000000000000000014
ceros

Acceso a memoria de doubleword (LDUR/STUR)

En LEGv8 la memoria se direcciona de a byte => si se accede de forma secuencial a datos doubleword las direcciones varían de a 8.

En este ejemplo el
microprocesador
LEGv8 está
configurado en modo
big-endian

Dirección	Contenido
0	0x0A
1	0x0B
2	0x0C
3	0x0D
4	0x0E
5	0x0F
6	0x10
7	0x11
8	0x12
9	0x13
10	0x14
11	0x15
12	0x16
13	0x17
14	0x18
15	0x19
16	0x1A
...	...

palabra 0

palabra 1

Dirección	0	1	2	3	4	5	6	7
0:	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	0x10	0x11
8:	0x12	0x13	0x14	0x15	0x16	0x17	0x18	0x19
16:	0x1A
...

Contenido (64 bits)

LDUR X1, [XZR, #0]

// X1 = 0x0A0B0C0D0E0F1011

LDUR X1, [XZR, #8]

// X1 = 0x1213141516171819

Acceso a memoria de doubleword (LDUR/STUR)

Si se configura el microprocesador en modo little-endian los bits **menos** significativos de una palabra se guardan en las posiciones más bajas de memoria.

Dirección	Contenido
0	0x0A
1	0x0B
2	0x0C
3	0x0D
4	0x0E
5	0x0F
6	0x10
7	0x11
8	0x12
9	0x13
10	0x14
11	0x15
12	0x16
13	0x17
14	0x18
15	0x19
16	0x1A
...	...

palabra 0

palabra 1

LDUR X1, [XZR, #0]

// X1 = 0x11100F0E0D0C0B0A

LDUR X1, [XZR, #8]

// X1 = 0x1918171615141312

En este ejemplo el
microprocesador
LEGv8 está
configurado en modo
little-endian

Acceso a memoria de doubleword (LDUR/STUR)

Dirección Contenido

...	...
16	0x1A
15	0x19
14	0x18
13	0x17
12	0x16
11	0x15
10	0x14
9	0x13
8	0x12
7	0x11
6	0x10
5	0x0F
4	0x0E
3	0x0D
2	0x0C
1	0x0B
0	0x0A

palabra 1

palabra 0

Microprocesador en modo **big-endian**:

LDUR X1, [XZR, #0] // X1 = 0A0B0C0D0E0F1011

LDUR X1, [XZR, #8] // X1 = 1213141516171819

Microprocesador en modo **little-endian**:

LDUR X1, [XZR, #0] // X1 = 0x11100F0E0D0C0B0A

LDUR X1, [XZR, #8] // X1 = 0x1918171615141312

Por defecto asumimos
que el microprocesador
está en modo **big-endian**

Acceso a memoria de word (LDURSW/STURW)

En LEGv8 la memoria se direcciona de a byte, por lo que si se accede de forma secuencial a datos word las direcciones varían de a 4.

Dirección	Contenido	
...	...	
16	0x1A	palabra 3
15	0x19	
14	0x18	
13	0x17	
12	0x86	palabra 2
11	0x15	
10	0x14	
9	0x13	
8	0x12	palabra 1
7	0x11	
6	0x10	
5	0x0F	
4	0x0E	palabra 0
3	0x0D	
2	0x0C	
1	0x0B	
0	0x0A	

Dirección	Contenido			
0	0x0A	0x0B	0x0C	0x0D
4	0x0E	0x0F	0x10	0x11
8	0x12	0x13	0x14	0x15
12	0x86	0x17	0x18	0x19
16	0x1A
...

32 bits

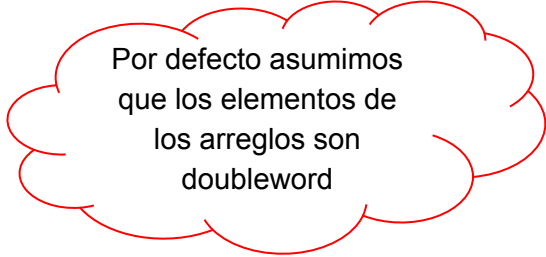
En este ejemplo el microprocesador LEGv8 está configurado en modo *big-endian*

LDURSW X1, [XZR, #0] // X1 = 0x00000000A0B0C0D

LDURSW X1, [XZR, #12] // X1 = 0xFFFFFFFF86171819

Ejercicio 5(a)

Dada la siguiente sentencia en “C”:

$$f = -g - A[4];$$


Por defecto asumimos
que los elementos de
los arreglos son
doubleword

5.1) Escribir la secuencia mínima de código assembler LEGv8 asumiendo que f, g, i y j se asignan en los registros X0, X1, X2 y X3 respectivamente, y que la dirección base de los arreglos A y B se almacenan en los registros X6 y X7 respectivamente.

5.2) ¿Cuántos registros se utilizan para llevar a cabo las operaciones anteriores?

Ejercicio 5(a)

$f = -g - A[4];$

$X0 \leftarrow f$

$X1 \leftarrow g$

$X6 \leftarrow \text{dirección base del arreglo } A = \&A[0]$

$\&A[4] = \&A[0] + 4 \cdot 8$

LDUR $X0, [X6, \#32]$ // $f = A[4]$

ADD $X0, X0, X1$ // $f = A[4] + g$

SUB $X0, XZR, X0$ // $f = -g - A[4]$

5.2) Se utilizaron 4 registros.

Ejercicio 5(b)

Dada la siguiente sentencia en “C”:

$$B[8] = A[i - j];$$

5.1) Escribir la secuencia mínima de código assembler LEGv8 asumiendo que f, g, i y j se asignan en los registros X0, X1, X2 y X3 respectivamente, y que la dirección base de los arreglos A y B se almacenan en los registros X6 y X7 respectivamente.

5.2) ¿Cuántos registros se utilizan para llevar a cabo las operaciones anteriores?

Ejercicio 5(b)

$B[8] = A[i - j];$

$X2 \leftarrow i$

$X3 \leftarrow j$

$X6 \leftarrow \text{dirección base del arreglo } A = \&A[0]$

$X7 \leftarrow \text{dirección base del arreglo } B = \&B[0]$

$\&A[i-j] = \&A[0] + (i-j)*8$

SUB X9, X2, X3 // X9=i-j

LSL X9, X9, #3 ★ // X9=(i-j)*8

ADD X10, X6, X9 // X10=&A+[(i-j)*8]

LDUR X11, [X10, #0] // X11=A[i-j]

STUR X11, [X7, #64] // B[8]=A[i-j]

$\&B[8] = \&B[0] + 8*8$

5.2) Se utilizaron 7 registros.

★ Usando LSL para multiplicar...

ADDI X0, XZR, #15 // $X0_{10} = 15$, o su equivalente en binario:

[illegible]

Luego:

```
LSL X1, X0, #1 // X1= 0b0...00011110 →  $X1_{10} = 30$  → equivale a  $X0 \cdot 2^1$ 
```

```
LSL X2, X0, #2 // X2= 0b0...00111100 →  $X2_{10} = 60$  → equivale a  $X0 \cdot 2^2$ 
```

```
LSL X3, X0, #3 // X3= 0b0...01111000 →  $X3_{10} = 120$  → equivale a  $X0 \cdot 2^3$ 
```

Ejercicio 6(b)

Dadas las siguientes sentencias en assembler LEGv8:

```
LSL X9, X3, #3
ADD X9, X6, X9
LSL X10, X4, #3
ADD X10, X7, X10
LDUR X12, [X9, #0]
ADDI X11, X9, #8
LDUR X9, [X11, #0]
ADD X9, X9, X12
STUR X9, [X10, #0]
```

6.1) Escribir la secuencia mínima de código “C” asumiendo que los registros X0, X1, X2, X3 y X4 contienen las variables f, g, h, i y j respectivamente, y los registros X6, X7 contienen las direcciones base de los arreglos A y B.

6.2) Para las instrucciones LEGv8 anteriores, re-escriba el código para minimizar (de ser posible) la cantidad de instrucciones manteniendo la funcionalidad.

Ejercicio 6(b)

$X3 \leftarrow i$

$X4 \leftarrow j$

$X6 \leftarrow$ dirección base del
arreglo $A = \& A[0]$

$X7 \leftarrow$ dirección base del
arreglo $B = \& B[0]$


LSL X9, X3, #3 //X9 = $i * 8$

ADD X9, X6, X9 //X9 = $\&A + (i * 8) = \&A[i]$

LSL X10, X4, #3 //X10 = $j * 8$

ADD X10, X7, X10 //X10 = $\&B + (j * 8) = \&B[j]$

LDUR X12, [X9, #0] //X12 = $A[i]$

 ADDI X11, X9, #8 //X11 = $\&A[i] + 8 = \&A[i+1]$

LDUR X9, [X11, #0] //X9 = $A[i+1]$

ADD X9, X9, X12 //X9 = $A[i+1] + A[i]$

STUR X9, [X10, #0] //B[j] = $A[i+1] + A[i]$

6.1) $B[j] = A[i+1] + A[i]$

Ejercicio 7

Dirección	Valor
0x0000000040080030	0x64
0x0000000040080038	0xC8
0x0000000040080040	0x12C

Dadas las siguientes sentencias en assembler LEGv8:

```
ADDI X9, X6, #8
ADD  X10, X6, XZR
STUR X10, [X9, #0]
LDUR X9, [X9, #0]
ADD  X0, X9, X10
```

7.1) Asumiendo que los registros X0, X6 contienen las variables f y A (dirección base del arreglo), escribir la secuencia mínima de código “C” que representa.

7.2) Asumiendo que los registros X0, X6 contienen los valores 0xA, 0x40080030, y que la memoria contiene los valores de la tabla, encuentre el valor del registro X0 al finalizar el código assembler.

Ejercicio 7

$X0 \leftarrow f$

$X6 \leftarrow$ dirección base del
arreglo $A = \&A[0]$

ADDI X9, X6, #8 //X9 = $\&A[0]+8 = \&A[1]$

ADD X10, X6, XZR //X10 = $\&A[0]+0 = \&A[0]$

STUR X10, [X9, #0] //A[1] = $\&A[0]$

 LDUR X9, [X9, #0] //X9 = $A[1] = \&A[0]$

ADD X0, X9, X10 //f = $\&A[0] + \&A[0]$

Dirección	Valor
0x0000000040080030	0x64
0x0000000040080038	0xC8
0x0000000040080040	0x12C

7.1) $f = \&A[0] + \&A[0]$

Bibliografía

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