Arquitectura de Ordenadores

Codificación de la Información

Abelardo Pardo
abel@it.uc3m.es

Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

Lógica Binaria

- Internamente el ordenador sólo es capaz de distinguir entre dos símbolos: 1 y 0.
- Toda **información** debe ser codificada utilizando únicamente estos dos símbolos.
- Toda **operación** debe ser codificada también en binario.
- ¿Cómo codificar **toda la información utilizada** utilizando lógica binaria?

Propiedades de una Codificación

- A cada símbolo le corresponde **un único** código binario.
- A cada código binario le corresponde **un único** símbolo.
- ¿**Cuántos** símbolos se quieren codificar?
- Con n bits se pueden codificar **hasta** 2^n símbolos diferentes.
- Para codificar N símbolos se rpecisan como mínimo $\lceil \log_2 N \rceil$.
- **Ejemplo:** ¿Cuantos bits se necesitan para codificar 33 símbolos?

Codificación de Números Naturales

- En lugar de utilizar **base 10** se utiliza **base 2**.
- Con un número finito de bits tan sólo se puede representar un **subconjunto** de los números naturales.
- **Ejemplo:** Un número de cuatro cifras en base 10 $d_3d_2d_1d_0$ se obtiene mediante la fórmula:

$$\text{valor} = \Sigma_{i=0}^{3}(d_i * base^i) = (d_0 * 10^0) + (d_1 * 10^1) + (d_2 * 10^2) + (d_3 * 10^3)$$

- Análogamente, un número de cuatro cifras en base 2, se obtiene **mediante la misma fórmula**.
- **Ejemplo:** El valor del número 1101 se obtiene

$$\text{valor} = \Sigma_{i=0}^{3}(d_i * 2^i) = (d_0 * 1) + (d_1 * 2) + (d_2 * 4) + (d_3 * 8)$$

- Otra forma de cálculo: Los dígitos de un número binario tienen como peso relativo las potencias de 2: 1, 2, 4, 8, 16, ...
- El valor de un número **no varía** si se añaden ceros a su izquierda.
- Al conjunto de **ocho bits** se le denomina **byte**.

Ejemplo de Codificación Binario - Decimal

Número en Binario		Número en Decimal
00100111	→	39
10110010	→	178
11011001	→	217

El peso de cada bit es 2^p siendo p la posición que ocupa el bit en el número binario empezando por el bit menos significativo y contando desde el cero.

Bit número	0	1	2	3	4	5	6	7
Peso	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
Equivalente	1	2	4	8	16	32	64	128

Bit menos significativo: El de menor peso, es decir $2^0 = 1$. El primero por la derecha.

Bit más significativo: El de mayor peso. En un byte, el de peso $2^7 = 128$. El último por la derecha.

Conversión de Binario a Decimal

Peso	Bit	Valor	Bit	Valor	Bit	Valor
128	0		1	128	1	128
64	0		0		1	64
32	1	32	1	32	0	
16	0		1	16	1	16
8	0		0		1	8
4	1	4	0	4	0	
2	1	2	1	2	0	
1	1	1	0	1	1	1
	Total	39		178		217

Codificación en Base 16 o Hexadecimal

- Los números en lógica binaria son **demasiado largos**.
- Para expresar un byte se precisa especificar los **ocho bits**.
- Tanto la notación binaria como su traducción a decimal es muy **incómoda de manipular**.
- La base **16** o **hexadecimal** permite una traducción instantánea de códigos binarios a un formato más compacto.
- Los 16 dígitos se representan mediante los números 0 al 9 y las letras de la **'A'** a la **'F'**.
- Cada **dígito** en hexadecimal representa **cuatro bits** (y viceversa).
- Para denotar que un número está codificado en hexadecimal se precede con el prefijo **0x**

Dígito Hex.	0	1	2	3	4	5	6	7
Binario	0000	0001	0010	0011	0100	0101	0110	0111

Dígito Hex.	8	9	A	B	C	D	E	F
Binario	1000	1001	1010	1011	11100	1101	1110	1111

Codificación Binario/Hexadecimal

Número en Binario		Número en Hexacimal
00100111	$\rightarrow$	0x27
10110010	$\rightarrow$	0x$B2$
11011001	$\rightarrow$	0x$D9$

Se agrupan los bits de cuatro en cuatro comenzando por el menos significativo. El número binario se completa con los ceros a la izquierda necesarios.

Número en Binario	Grupos de 4 bits		Número en Hexadecimal
00100111	$0010 = 2$	$0111 = 7$	0x27
10110010	$1011 = 11$	$0010 = 2$	0x$B2$
11011001	$1101 = 13$	$1001 = 9$	0x$D9$

Conversión Hexadecimal/Decimal: Idéntica a la conversión binaria a decimal, pero utilizando 16 como base en lugar de 2 y teniendo en cuenta los valores de las letras A, B, C, D, E y F.

$$0x27 \rightarrow \quad 2 * 16^1 + 7 * 16^0 \quad = 39$$
$$0xB2 \rightarrow \quad 11 * 16^1 + 2 * 16^0 \quad = 178$$
$$0xD9 \rightarrow \quad 13 * 16^1 + 9 * 16^0 \quad = 217$$

Codificación de Números Enteros

- Al intentar codificar también **números negativos** la codificación aplicada para los número naturales **no sirve**.
- **Primera Aproximación:** Se codifica el valor absoluto como un número natural, y se añade un bit adicional para codificar su signo (positivo, negativo).
- En este caso, se mezcla la codificación numérica, con la **codificación de símbolos**: positivo, negativo.
- Esta codificación **no es inequívoca**: El cero puede tener dos códigos igualmente válidos.
- **Segunda Aproximación:** Representación en complemento a dos. Soluciona el problema anterior. La codificación sí es inequívoca.
- Con n bits se pueden codificar los números enteros desde $-2^{(n-1)}$ hasta $2^{(n-1)} - 1$.

$$-128, -127, \ldots, -2, -1, 0, 1, 2, \ldots, 126, 127$$
$$10000000, 10000001, \ldots, 11111110, 11111111, 00000000, 00000001, 00000010, \ldots, 01111110, 01111111$$

- Además, los números positivos comienzan por **cero** y los negativos comienzan por **uno**.

Codificación de Números Enteros

- Si el número entero N es positivo, **comienza por cero**.
  - Si N es positivo entonces $N \in [0, 2^{n-1} - 1]$.
  - Como máximo $N = 2^{n-1} - 1$, y $2^{n-1} - 1$ requiere $n - 1$ dígitos.
  - El dígito n es **cero**.
- Si el número entero N es negativo, **comienza por uno**.
  - Si N es negativo entonces $N \in [-2^{n-1}, -1]$.
  - $N = -2^n + \text{ABS}(N)$.
  - Lo **mínimo** que puede valer $\text{ABS}(N)$ es tal que $-2^n + \text{ABS}(N) = -2^{n-1}$.
  - $\text{ABS}(N) = 2^n - 2^{n-1} = 2^{n-1}$, entonces N **comienza por uno**.

Números	Representación
Naturales $N \in [0, 2^n - 1]$	$N = \sum_0^{n-1} 2^i b^i$
Enteros $N \in [0, 2^{n-1} - 1]$	N empieza por cero, $N = \sum_0^{n-1} 2^i b^i$
$N \in [-2^{n-1}, -1]$	N empieza por uno, $N = -2^n + \text{ABS}(N)$ $N = -2^n + \sum_0^{n-1} 2^i b^i$

$$-128, -127, \ldots, -2, -1, 0, 1, 2, \ldots, 126, 127$$

$$10000000, 10000001, \ldots, 11111110, 11111111, 00000000, 00000001, 00000010, \ldots, 01111110, 01111111$$

Aritmética con Números Enteros

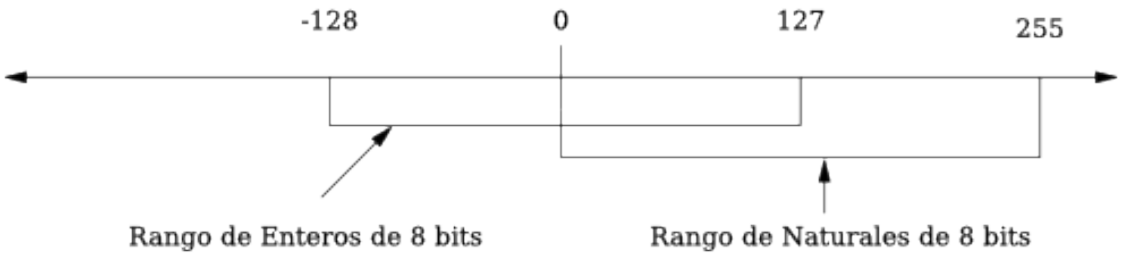
- Los números enteros se representan **con un conjunto finito de bits**.
- Las operaciones entre enteros reciben operandos y producen un **resultado entero**.
- ¿Qué pasa si sólo podemos manipular números decimales de 4 dígitos y sumamos **9999 + 1000**?
- Cuando un número no se puede representar con los dígitos permitidos se produce una situación de **desbordamiento** (overflow).
- **Ejemplo:** Representamos números decimales con 1 byte. ¿Cuál es el resultado de la suma **0xFC + 0xA5**?

Aritmética con Números Enteros Representados en Binario

- En base 10 (números decimales), las operaciones de multiplicación y división entera por 10 son **triviales**.
- En general, en base b, las operaciones de multiplicación y división entera por b son **triviales**.
- **Ejemplo:** Al multiplicar por 2 el número **01101011** se obtiene **11010110** (o en hexadecimal 0x6B*2 = 0xD6).
- **Ejemplo:** Al dividir por 2 el número **01101011** se obtiene **00110101** (o en hexadecimal 0x6B/2 = 0x35).
- ¿Cómo se realiza una multiplicación o división por una **potencia de 2** de un número binario?
- ¿Qué sucede en la operación **0x90*4** si los números se representan por 1 byte?

Representación de Números Reales

- La representación de números enteros y naturales en binario se reduce a un **intervalo definido por el número de bits utilizado por la representación**.



- En un intervalo de números reales existen **infinitos números**.
- ¿Cómo se puede representar un conjunto infinito con un conjunto finito de bits?.
- Sólo se representan **ciertos números**.
- Todo número real se puede representar mediante la expresión: mantisa $*$ base^e, donde la mantisa es un número cuyo primer decimal es **diferente de cero**.
- **Ejemplo:** $15{,}4 = 0{,}154 * 10^2$: Ambos datos se pueden codificar como **enteros**
- La representación en **coma flotante** consiste en codificar las dos partes de un número: **mantisa** y **exponente** como dos enteros: mantisa $* 2^e$.

Representación de Números Reales

- Los dígitos a la derecha de la coma tienen como peso **las potencias negativas de 2**.
- **Ejemplo:** $110,101 = 1*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 6,625$
- Para representar un número en **coma flotante** se desplaza hasta que no haya dígitos en su parte entera y se ajusta el exponente.
- **Ejemplo:** $110,101 = 0,110101 * 2^3$
- Supongamos una representación con 8 bits para la mantisa y 4 para el exponente. La codificación sería: $\text{mantisa} = 11010100, \text{exp} = 0011$.
- ¿Cómo puedo **incrementar** la precisión de la mantisa manteniendo el mismo número de bits?

Representación de Conjuntos de Símbolos

- Un conjunto arbitrario de **n símbolos** se puede representar mediante un conjunto de bits.
- Con un bit podemos codificar un **máximo** de dos elementos.
- ¿**Cuantos** bits se precisan?
- Se debe especificar el **tamaño de la representación** (número de bits por símbolo) y la correspondencia símbolo-número binario.

Representación de Conjuntos de Símbolos

- **Ejemplo:** Se quiere representar la colección de colores {rojo, verde, azul}.
- Se necesitan **como mínimo** dos bits.
- 2 bits permiten **cuatro combinaciones**, por tanto se pueden codificar cuatro elementos como **máximo**.

$$\begin{aligned} 00 &\rightarrow \text{Elemento } 1 \\ 01 &\rightarrow \text{Elemento } 2 \\ 10 &\rightarrow \text{Elemento } 3 \\ 11 &\rightarrow \text{Elemento } 4 \end{aligned}$$

- ¿Cuantos elementos **como máximo** podemos codificar con n bits?
- ¿Por qué **"como máximo"**?

$$\begin{aligned} 01 &\rightarrow \text{Elemento } 1 \\ 10 &\rightarrow \text{Elemento } 2 \end{aligned}$$

- **Posibles codificaciones:**

Valor	Cod. 1	Cod. 2	Cod. 3
Rojo	00	11	001
Verde	01	10	010
Azul	10	01	100

Representación de Cadenas de Caracteres (Strings)

- Aparte de datos numéricos, se precisan manipular **cadenas de caracteres**.
- La codificación de los caracteres tiene que ser **común para todas las aplicaciones**.
- Los códigos más comunes utilizados para esta representación son **ASCII (8 bits)** y **Unicode (16 bits)**.
- El código ASCII, al utilizar **8 bits** puede representar un máximo de **256 símbolos**.
- **Ejemplo:** Los caracteres '1', '2', '3', '4' y '5' se representan como **0x31, 0x32, 0x33, 0x34, 0x35**

Almacenamiento de un Programa

- Un programa es un **texto** que se escribe mediante un **editor** en un **fichero**.
- Los símbolos incluidos en el fichero se codifican mediante **código ASCII**, y estos números son los que se almacenan en el fichero.

```
        .data
20 20 20 20 20 20 20 20 2E 64 61 74 61 0A
msg:    .asciz "Hello world\n"
6D 73 67 3A 20 20 20 20 2E 61 73 63 69 7A 20 22 48 65 6C 6C...
        .text
20 20 20 20 20 20 20 20 2E 74 65 78 74 0A
        .globl start
20 20 20 20 20 20 20 20 2E 67 6C 6F 62 6C 20 73 74 61 72 74...
start:  push $msg
73 74 61 72 74 3A 20 20 70 75 73 68 20 24 6D 73 67 0A
        call printf
20 20 20 20 20 20 20 20 63 61 6C 6C 20 70 72 69 6E 74 66 0A
        add $4, %esp
20 20 20 20 20 20 20 20 61 64 64 20 24 34 2C 20 25 65 73 70....
        ret
20 20 20 20 20 20 20 20 72 65 74 0A
```


Codificación de Operaciones

- En lugar de codificar números o símbolos se quieren codificar **operaciones**.
- **Ejemplo:** Operaciones de **suma**, **resta**, **multiplicación** y **división** de **dos números de 8 bits**.
- ¿Cuántas posibles **combinaciones** de operaciones son posibles?
- Existen **4** operaciones, **256** números posibles como primer operando, y **256** como segundo, total: $2^8 * 2^8 * 2^2 = 2^{18}$.
- Se deben codificar 2^{18} elementos posibles. Se procede a codificarlos por partes.

1. Conjunto de operaciones **ADD, SUB, MUL, DIV**: se precisan **2 bits como mínimo**.
2. Primer operando: **se precisan 8 bits**.
3. Segundo operando: **se precisan 8 bits**.

Formato de Instrucciones

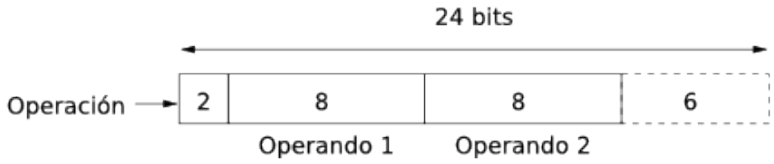
- Se decide representar la instrucción como una **secuencia** de 2 bits que denotan la **operación**, seguida de los 8 bits que codifican el **primer operando** y 8 más que codifican el **segundo**.

Operación	Codificación
ADD	00
SUB	01
MUL	10
DIV	11

  - Con esta codificación la operación para **sumar los números 0x27 y 0xB2** se codifica:

ADD 0x27 0xB2 = 00 00100111 10110010

  - ¿Cuántos **bytes** son precisos para codificar la instrucción anterior?
- Se debe decidir cómo completar la representación con dígitos extra. Por ejemplo **ceros al final del último byte**.

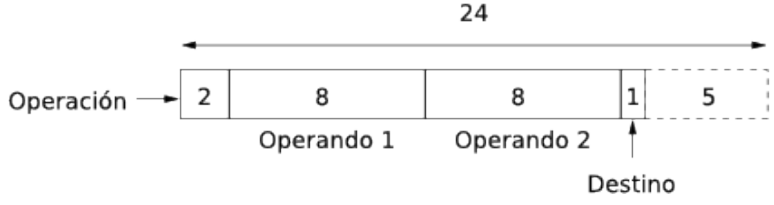


- Cada instrucción tiene una **única representación** hexadecimal en bytes:

Instrucción	Codificación
ADD 0x27 0xB2	0x09EC80
SUB 0x11 0x2F	0x444BC0
MUL 0xFA 0x2B	0xBE8AC0
DIV 0x10 0x02	0xC40080

Almacenamiento de Resultado

- Las instrucciones se complican: **el resultado se guarda o almacena en dos lugares posibles**.
- Supongamos que estos lugares son: **LugarA** y **LugarB**.
- El resultado se guarda con tamaño **1 byte**.
- **Ejemplo:**
  1. **Inicialmente:** LugarA = 0x00, LugarB = 0x00
  2. Se ejecuta la instrucción: **DIV 0x10 0x02 LugarA**
  3. **Resultado: LugarA = 0x08**, LugarB = 0
  4. Se ejecuta la instrucción: **ADD 0x10 0x02 LugarB**
  5. **Resultado:** LugarA = 0x08, **LugarB = 0x12**
  6. Se ejecuta la instrucción: **MUL 0x10 0x02 LugarA**
  7. **Resultado: LugarA = 0x20**, LugarB = 0x12
- Las instrucciones deben **codificar donde se almacena el resultado**. Se utiliza **un bit**.
- Este bit se escribe **a continuación de los dos operandos**



Codificación de Instrucciones con Resultado

- Se decide codificar **LugarA como 0**, **LugarB como 1**.
- Ejemplos de **codificación**:

Instrucción	Cod. Bin.	Cod. Hex.
DIV 0x10 0x02 LugarA	1100 0100 0000 0000 1000 0000	0xC40080
ADD 0x10 0x02 LugarB	0000 0100 0000 0000 1010 0000	0x0400A0
MUL 0x10 0x02 LugarA	1000 0100 0000 0000 1000 0000	0x840080

- **Observación:** Como el destino de la operación es un lugar de **dos posibles** con un único bit es suficiente para codificarlo.
- El **formato de instrucción** con el que se ha realizado la codificación es:

  Operación Operando1 Operando2 Destino BitsExtra

- Dados **3 bytes** se puede obtener **de forma inequívoca** la instrucción que representa.
- Dada una instrucción se puede obtener **3 bytes** que la representan.
- ¿Qué instrucción representa el código **0x04023F**?
- ¿Qué se hacen con los bits de la representación que **sobran**?

Resultados Previos como Operandos

- Se quieren **extender** las instrucciones tal que un operando pueda ser **uno de los dos lugares en los que se ha almacenado un resultado**.
- **Ejemplo:** Sería deseable tener la instrucción **ADD LugarA 0x10 LugarB**.
- ¿Qué modificaciones son necesarias en la **codificación** de las instrucciones?
- Los operandos pueden ser de **dos tipos**:

  1. Un número codificado con **8 bits**.
  2. Uno de los lugares **LugarA** o **LugarB**

- La codificación **debe cambiar** para poder expresar las nuevas instrucciones.

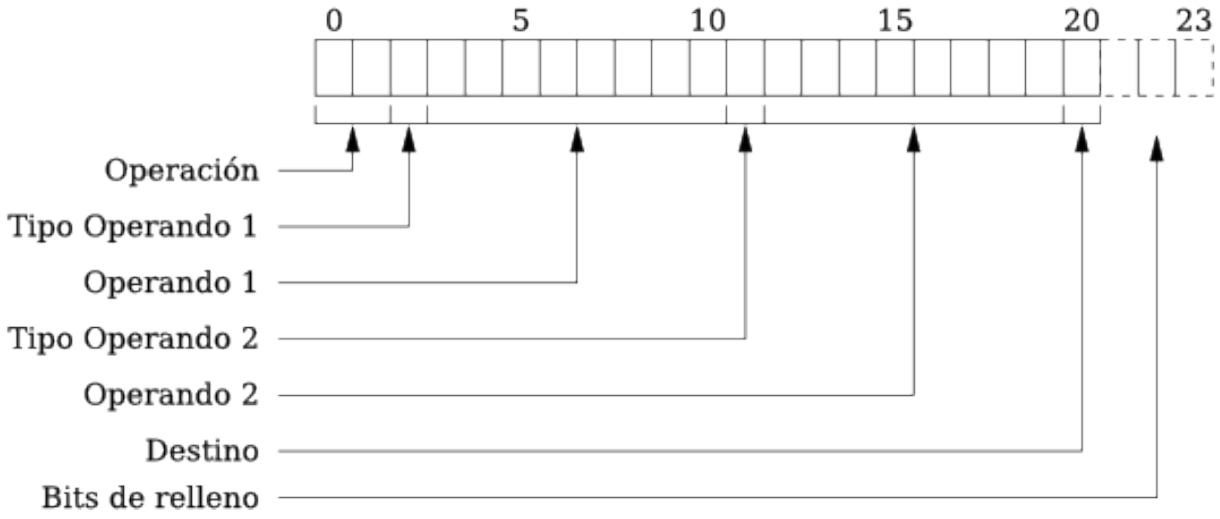
Códigos que Incluyen Condiciones

- La única condición de la codificación es **que sea inequívoca**.
- Se añade **un bit** que dice si los siguientes 8 bits representan **un número** o uno de los dos lugares para obtener el operando.
- El operando se codifica con **9 bits**.
  1. Si el primer bit es **0**: los ocho bits siguientes representan el operando como un número.
  2. Si el primer bit es **1**: de los ocho bits siguientes se toma el último que es el que indica de qué lugar obtener el operando.
- **Ejemplo de codificación de operando:**

Cod. Bin.	Operando
000010010	Número 0x12 (18 en decimal)
100010011	LugarB
001111010	Número 0x7A (122 en decimal)
100010010	LugarA
1XXXXXXX0	LugarA

Formato Extendido de Instrucción

- El **formato extendido** de instrucción consta ahora de 6 partes:
  1. 2 bits para codificar el **tipo de operación**
  2. 1 bit para codificar el **tipo del primer operando**
  3. 8 bits para codificar el **dato del primer operando**
  4. 1 bit para codificar el **tipo del segundo operando**
  5. 8 bits para codificar el **dato del segundo operando**
  6. 1 bit para codificar el **destino**



- **Ejemplos:**

Instrucción	Codificación
ADD 0x01 0x00 LugarA	0x002000
ADD 0x02 0x00 LugarB	0x004008
MUL LugarB 0x03 LugarA	0xA02030
MUL LugarB 0x04 LugarA	0xA02040

- No todos los operandos posible **deben aparecer necesariamente en una instrucción**.
- La codificación **sí debe contemplar todos los casos posibles**.
- Todas las instrucciones **tienen la misma longitud**.
- ¿Se pueden codificar estas instrucciones con **menos bits**?
- El formato de instrucción puede tener **longitud variable**.

- Se pueden interpretar los bits que codifican el tipo de operando y a continuación **obtener los bits necesarios**.
- **En el ejemplo**: 8 si es un número y 1 si es uno de los dos lugares (LugarA o LugarB).
- La instrucción **ADD LugarA LugarB LugarA** se puede codificar como **0010110**.
- La **interpretación de una instrucción** mira el código del operando y a continuación interpreta los bits necesarios de la manera pertinente.
- Esta tarea se puede hacer porque los bits de relleno **están incluidos al final de la instrucción**.
- **Ejemplo: ADD LugarA LugarB LugarA** se codifica como **0010110**, por lo que se puede codificar en un byte como **00101100 = 0x2C**

- Extraído del **manual de instrucciones** del procesador Intel Pentium (Instruction Set Reference)

ADD–Add

Opcode	Instruction	Description
04 ib	ADD AL, imm8	Add imm8 to AL
...	...	...

Description

Adds the first operand (destination operand) and the second operand (source operand) and stores the result in the destination operand (...).

Operation

DEST ← DEST + SRC;
...

- Supongamos que inicialmente: LugarA = 0, LugarB = 0
- A continuación se ejecutan **por orden** las siguientes instrucciones:

```
ADD 0x01 0x00 LugarA
ADD 0x00 0x00 LugarB
MUL 0x02 LugarA LugarA
ADD 0x02 LugarB LugarB
MUL 0x03 LugarA LugarA
ADD 0x03 LugarB LugarB
MUL 0x04 LugarA LugarA
ADD 0x04 LugarB LugarB
```

- Esta secuencia de instrucciones se codifica como **un conjunto de bytes**.
- El procesador es **un sistema digital** capaz de **interpretar y ejecutar** un conjunto de instrucciones codificadas en binario.
- ¿Cuál es el resultado final contenido en **LugarA**?
- ¿Cuál es el resultado final contenido en **LugarB**?

