

# Tema 2

## **Representación de la Información**

# Contenido

---


1. **Introducción**
2. **Clasificación**
3. **Datos numéricos**
  1. Naturales
  2. Corrección de errores
  3. Enteros
  4. Flotantes
4. **Datos Alfanuméricos**
5. **Instrucciones y direcciones**
  1. Modos de direccionamiento
  2. Formato de instrucciones
  3. Juego de instrucciones



## 2. 1 Representación de la Información

---

- ▶ **¿Qué es información?**
  - ▶ Instrucciones: Lo que hay que hacer
  - ▶ Datos (sobre los que actúan las instrucciones)
- ▶ **Representación: Traducción a un lenguaje específico**
  - ▶ ...que la máquina pueda “entender” y “manipular”.

(Se utilizará este símbolo  para indicar la operación de representar)
- ▶ **¿Cómo se representan instrucciones y datos?**
  - ▶ Utiliza un “lenguaje” con sólo dos “letras”: 0 , 1
  - ▶ (cada letra está asociada a un potencial eléctrico)
  - ▶ Las palabras son cadenas binarias: 00001010101



# ¿Qué tipos de instrucciones?

---

- ▶ Muy simples. Ejemplo:
  - ▶ Almacenar ciertos *datos* en registros
  - ▶ Almacenar ciertos *datos* en alguna posición de memoria
  - ▶ Realizar una operación elemental con los *datos* y guardar el resultado en algún sitio (memoria o registro)
    - ▶ sumas, restas, and, or
    - ▶ opuesto, inverso, not, etc...
- ▶ Se representan en binario: NOT Reg3 ⚡ 01001 00011



# ¿Qué tipo de datos?

---

- ▶ Letras del alfabeto y otros símbolos

- ▶ “alfanuméricos”: a, B, ñ, ), ?, +, 3, \$, ~, ♥, ↵
- ▶ Cadenas de alfanuméricos: “pepe”, “xDD”

- ▶ Números

- ▶ Naturales: 1, 25
- ▶ Enteros: -12, +48
- ▶ Racionales y reales

Algunos no son representables:  $\pi$

00110000

00001101

01010000

01000101

01010000

01000101

11.0010010101000101010100100101010101010100100.....


# Dos tipos de problemas

---

- ▶ Problemas directos: ¿Cómo representamos X?

X  00100100


- ▶ Problemas inversos ¿Qué representa 00100100?

00100100  X



# Inconvenientes:

---

- ▶ Símbolos iguales para instrucciones y datos
- ▶ 0010100101  ?
- ▶ El “contexto” es siempre necesario:
  - ▶ donde debe haber instrucciones, habrá instrucciones
  - ▶ donde se espera un dato de cierto tipo, allí debe estar
  - ▶ en caso contrario:  
“El programa ha efectuado una operación no valida”



# Ejemplo de problema inverso

---

▶ ¿Qué representa 01000101?

▶ No hay respuesta valida

▶ ¿Qué representa 01000101...

... si identifica un símbolo alfabético utilizando el formato de representación “ASCII”

01000101 :“E”





## 2.2 Clasificación de la información

---

### ▶ Datos

#### ▶ Numéricos

##### ▶ Naturales (sin signo)

- ☐ No posicionales
- ☐ Posicionales

##### ▶ Enteros (con signo)

- ☐ Signo-Magnitud
- ☐ Negativos en complemento a 1, a 2
- ☐ En exceso

##### ▶ Punto flotante (subconjunto de los reales) IEEE-754

#### ▶ Alfanuméricos

- ☐ ASCII
- ☐ Unicode

### ▶ Instrucciones



## 2.3 Representación de datos numéricos

# Números Naturales

---

- ▶ **Arbitrarios:**

- ▶ **Ejemplo:**

1 ⚡ 010001

2 ⚡ 001010

3 ⚡ 100101

- ▶ **No arbitrarios**

Responden a una ciertas reglas

1 ⚡ 000001

2 ⚡ 000010

3 ⚡ 000100



# Naturales

---

## ► Posicionales

- El valor de un dígito (0 o 1) depende de la posición:

	a b
► 0	00
► 1	01
► 2	10
► 3	11

( el dígito **a** tiene peso **2**, el dígito **b**, peso **1** )

- Similar a la numeración *indoarábiga*: binario natural



# Cuenta en binario

---



- ▶ 0
- ▶ 1
- ▶ 10
- ▶ 11
- ▶ 100
- ▶ 101
- ▶ 110
- ▶ 111
- ▶ 1000



# Cuenta en binario

---



- ▶ 0000
- ▶ 0001
- ▶ 0010
- ▶ 0011
- ▶ 0100
- ▶ 0101
- ▶ 0110
- ▶ 0111
- ▶ 1000

Ojo!!!  
La longitud de la secuencia es muy importante!



# Binario Natural

---

- ▶ El peso de cada dígito depende de su valor  $d$  (0, 1) y su posición  $i$

dígito	0	1	0	1	0	1
posición	5	4	3	2	1	0

- ▶ El peso de cada dígito es  $d_i \cdot 2^i$
- ▶ El número es  $0 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 21$



# Binario natural. Solución de problemas

---

- ▶ Imprescindible especificar el número de dígitos en el problema directo
- ▶ Completar con ceros a la izquierda
- ▶ Para el problema inverso:  $\sum (d_i \cdot 2^i)$

▶ Problema directo:

$$23:2 = 11, \text{ resto } 1$$

$$11:2 = 5, \text{ resto } 1$$

$$5:2 = 2, \text{ resto } 1$$

$$2:2 = 1, \text{ resto } 0$$

$$1:2 = 0, \text{ resto } 1$$

se repite

$$0:2 = 0, \text{ resto } 0 \quad \text{solución:}$$

00010111





# Binario Natural. Trucos imprescindibles

---

- ▶ Los ceros a la izquierda no cuentan: 001100 equivale a 1100
- ▶ Cualquier número en la forma 100000 es una potencia de 2
  - ▶ en base 10 sería una potencia de 10 (evidente, ¿no?)
- ▶ Cualquier número en la forma 11111 es una potencia de 2, menos 1
  - ▶ en base 10, existe una sentencia equivalente
- ▶ Del 0 al 15 deben memorizarse (0000 a 1111)
- ▶  $1K = 1024 = 2^{10} = 1000000000$  (1 seguido de 10 ceros)
- ▶  $1Mega = 1048576 = 2^{20} = 100000000000000000000$  (20 ceros)
- ▶  $1Giga = 1024 = 2^{30} = 100000000000000000000....000000$  (30 ceros)
- ▶ Todos los impares terminan en 1, y en cero los pares
- ▶ Si  $A = 00010101$  termina en 01, entonces  $A$  módulo 4 es 1 (01)
- ▶ Si  $A = 00010101$  termina en 101, entonces  $A$  módulo 8 es 5 (101)



# Problema

---

- ▶ En un torneo de tenis, por el sistema de eliminatorias, participan 1024 jugadores. ¿Cuántos partidos hay que programar, incluida la final?

Solución:

- ▶ 512 partidos en primera ronda + 256 partidos en segunda ronda + 128 en tercera ronda + 64 partidos en la cuarta + 32 partidos + 16 + 8 de partidos de octavos + 4 cuartos de final + 2 semifinales + final



## Otra solución

---

- ▶ Si en cada partido se elimina un jugador, y tras la final, hay 1023 eliminados y un solo ganador:

1023 partidos jugados



# Binarios naturales. Regla

---

$$2^n - 1 = 2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 4 + 2 + 1$$

Ejemplo:  $1 + 2 + 4 + 8 = 15 = (16 - 1)$

$$1111 = 10000 - 1$$



# Binario natural. Otras bases posicionales

---

- ▶ En otras bases, la cuenta es similar
  - ▶ base cuatro: 0, 1, 2, 3, 10, 11, 12, 13, 20
  - ▶ base ocho 0, 1, 2, 3, 4, 5, 6, 7, 10, 11
- ▶ Agrupaciones de bits:
  - ▶ 00 00 11 10 en binario natural es 0 0 3 2 en base cuatroSe agrupan los bits de dos en dos y se convierten los grupos
- ▶ 00 001 110 es 0 1 6 en base ocho



# Hexadecimal, o base 16

---

- ▶ Se necesitan 16 símbolos
- ▶ 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- ▶ 0110 1000 en binario natural es 68 en base 16
- ▶  $1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^3 = (0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0) \cdot 2^4 + (1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0) \cdot 2^0$   
 $= 6 \cdot 16^1 + 8 \cdot 16^0$
- ▶ Se puede deducir una regla general de agrupación de bits
- ▶ E incluso extrapolar a base 100:

$$\text{00429} = 69042299$$



# Otras representaciones de N

---

- ▶ Exceso a  $X$  (caso particular  $X=3$ ):  $N+X$  en  $bn$
- ▶ Complemento a  $X$  (caso,  $X=2^n-1$ ):  $X-N$  en  $bn$
- ▶ Binario natural con bit de paridad: 1 bit adicional, xor
- ▶ Códigos Hamming (SEC y DEC): más bits.
- ▶ BCD: cada dígito decimal es convertido a  $bn$  de 4 bits
- ▶ Gray, reflejo: 000 001 011 010 110 111 101 100
  
- ▶ Problema: ¿Cuáles son posicionales y cuáles no?



# Binario natural con bit de paridad

---

- ▶ Paridad de una cadena binaria:  
Paridad del número de “unos” contenidos en la cadena.
- ▶ Ejemplo:  $A = 100011101$  tiene 5 “unos”; 5 es impar, luego A tiene paridad impar
- ▶ Bit de paridad: 0 para cadenas pares, 1 para las impares (el bit de paridad de A es 1)
- ▶ Ejemplo: Representación de 23 ( $10111$  en *bn*) con bit de paridad a la derecha. Solución:  $10111\ 0$





# Representación de datos numéricos enteros (números con signo)

# Representaciones elementales (1)

---

## ▶ Signo – Magnitud

- ▶ 1 bit representa el signo (siempre se usa el de la izquierda)
  - ▶ 0 es positivo
  - ▶ 1 es negativo
- ▶  $n-1$  bits representan la magnitud (en binario natural)
- ▶ Doble representación para el cero. Ejemplo de 8 bits:
  - ▶ 00000000
  - ▶ 10000000

Notación:

$n \rightarrow$  es el número de bits, incluyendo siempre el signo

$N \rightarrow$  es el número que se quiere representar



# Representaciones (n=4)

base 10	signo magnitud				
+7	0 111				
+6	0 110				
+5	0 101				
+4	0 100				
+3	0 011				
+2	0 010				
+1	0 001				
+0	0 000				
-0	1 000				
-1	1 001				
-2	1 010				
-3	1 011				
-4	1 100				
-5	1 101				
-6	1 110				
-7	1 111				
-8					

# Representaciones elementales (2)

---

- ▶ Representaciones en exceso
  - ▶ Una representación en exceso a  $X$  de  $n$  bits permite representar números desde  $-X$  hasta  $2^n - X$
  - ▶ Ejemplo: exceso a 127 con 8 bits
    - 127 ⚡ 00000000 (-127 + 127 en  $bn$ )
    - 0 ⚡ 01111111 (0 + 127 en  $bn$ )
    - 128 ⚡ 11111111 (128 + 127 en  $bn$ )
  - ▶ El bit más significativo indica el signo, pero 1 es +
  - ▶ El cero tiene representación única



# Representaciones

base 10	signo magnitud	exceso a 8	exceso a 7		
+7	0 111	1111	1110 ↑		
+6	0 110	1110	1 101		
+5	0 101	1 101	1100		
+4	0 100	1100	1 011		
+3	0 011	1 011	1 010		
+2	0 010	1 010	1 001		
+1	0 001	1 001	1 000		
+0	0 000	1 000	0 111		
-0	1 000				
-1	1 001	0 111	0 110		
-2	1 010	0 110	0 101		
-3	1 011	0 101	0 100		
-4	1 100	0 100	0 011		
-5	1 101	0 011	0 010		
-6	1 110	0 010	0 001		
-7	1 111	0 001	0 000		
-8		0 000			

# Representación *de negativos en complemento a 2*

---

- ▶ Es la representación estándar para números enteros
- ▶ El bit de signo es el más significativo:
  - ▶ **0**1010101 → Este número de  $n=8$  bits es positivo
  - ▶ **1**001 → Este número de  $n=4$  bits es negativo
- ▶ Si el número es positivo, se representa en binario natural
  - ▶ De 0 a  $2^n - 1$
- ▶ Si el número es negativo...
  - ▶ La representación no es muy poco intuitiva
  - ▶ Los negativos se representan en complemento a  $2^n$  del valor absoluto:
    - ▶  $\text{BinarioNatural}(2^n - \text{abs}(N))$
- ▶ La denominación auténtica es *Negativos en complemento a  $2^n$*



# Representaciones

base 10	signo magnitud	exceso a 8	exceso a 7		negativos C-2
+7	0 111	1111	1110 ↑		0 111
+6	0 110	1110	1 101		0 110
+5	0 101	1 101	1100		0 101
+4	0 100	1100	1 011		0 100
+3	0 011	1 011	1 010		0 011
+2	0 010	1 010	1 001		0 010
+1	0 001	1 001	1 000		0 001
+0	0 000	1 000	0 111		0 000
-0	1 000				
-1	1 001	0 111	0 110		1 111
-2	1 010	0 110	0 101	Complemento a 16 de 2 →	1 110
-3	1 011	0 101	0 100		1101
-4	1 100	0 100	0 011		1 100
-5	1 101	0 011	0 010		1 011
-6	1 110	0 010	0 001	Complemento a 16 de 6 →	1 010
-7	1 111	0 001	0 000		1 001
-8		0 000		Complemento a 16 de 8 →	1 000

# Negativos en complemento a 2

---

- ▶ ¿Porqué un formato tan poco intuitivo?
- ▶ **Motivo nº 1: aritmética mucho más sencilla.**

Veamos ejemplos equivalentes en base 10, con  $n=2$ , y negativos en complemento a 100 (y utilizo colores en lugar de signos)

Tengo 23 euros, y como es positivo lo represento

23 euros

Gasto 12 euros en mascarillas.

Lo represento en complemento a 100:

88 euros

¿Cuánto suman las dos cantidades?

23

+88

111 (n es 2)





# Negativos en complemento a 2

---

- ▶ ¿Porqué un formato tan poco intuitivo?
- ▶ **Motivo nº 1: aritmética mucho más sencilla.**

Veamos ejemplos equivalentes en base 10, con  $n=2$ , y negativos en complemento a 100 (y utilizo colores en lugar de signos)

Debo 44 euros, y como es negativo lo represento 56 euros

.

Me tocan 16 euros en la primitiva 16 euros

¿Cuánto suman las dos cantidades?

56  
+16  
72 (Mi deuda es 28)



# Negativos en complemento a 2

---

- ▶ ¿Porqué un formato tan poco intuitivo?
- ▶ **Motivo nº 1: aritmética mucho más sencilla.**

Veamos ejemplos equivalentes en base 10, con  $n=2$ , y negativos en complemento a 100 (y utilizo colores en lugar de signos)

Debo 44 euros, y como es negativo lo represento 56 euros

.

Ahora me llega otra factura de 23 euros 77 euros

¿Cuánto suman las dos cantidades?

56

+77

133 (Mi deuda es 67)



# Negativos en complemento a 2


---

- ▶ ¿Porqué un formato tan poco intuitivo?
- ▶ **Motivo nº 2: ¡Es muy fácil calcular el complemento!.**

Veamos ejemplos equivalentes en base 10, con  $n=2$ , y negativos en complemento a 100

**Pregunta:** ¿Cuál es el complemento a 10000 del número **1327**?

**Respuesta rápida:** Uno más el complemento a **9999** del número **1327**

Vaya, pues me he quedado igual 

No te creas: El complemento a 9999 es el complemento a 9 de cada dígito separado:

$$9 - 1 = 8$$

$$9 - 3 = 6$$

$$9 - 2 = 7$$

$$9 - 7 = 2$$

---

Respuesta final: 8673

# Representaciones

base 10	signo magnitud	exceso a 8	exceso a 7		negativos C-2
+7	0 111	1111	1110 ↑		0 111
+6	0 110	1110	1 101		0 110
+5	0 101	1 101	1100		0 101
+4	0 100	1100	1 011		0 100
+3	0 011	1 011	1 010		0 011
+2	0 010	1 010	1 001		0 010
+1	0 001	1 001	1 000		0 001
+0	0 000	1 000	0 111		0 000
-0	1 000				
-1	1 001	0 111	0 110		1 111
-2	1 010	0 110	0 101	Complemento a 16 de 2 →	1 110
-3	1 011	0 101	0 100		1101
-4	1 100	0 100	0 011		1 100
-5	1 101	0 011	0 010		1 011
-6	1 110	0 010	0 001	Complemento a 16 de 6 →	1 010
-7	1 111	0 001	0 000		1 001
-8		0 000		Complemento a 16 de 8 →	1 000

# Antes de seguir...

---

- ▶ ¿Sabemos sumar uno, a un número binario?
- ▶ ¿Qué es el acarreo/carry?
- ▶ ¿Y el desbordamiento/overflow?



# Representaciones

base 10	signo magnitud	exceso a 8	exceso a 7	negativos C -1	
+7	0 111	1111	1110 ↑	0 111	
+6	0 110	1110	1 101	0 110	
+5	0 101	1 101	1100	0 101	
+4	0 100	1100	1 011	0 100	
+3	0 011	1 011	1 010	0 011	
+2	0 010	1 010	1 001	0 010	
+1	0 001	1 001	1 000	0 001	
+0	0 000	1 000	0 111	0 000	
-0	1 000			1 111	
-1	1 001	0 111	0 110	1 110	
-2	1 010	0 110	0 101	1101	
-3	1 011	0 101	0 100	1 100	
-4	1 100	0 100	0 011	1 011	
-5	1 101	0 011	0 010	1 010	
-6	1 110	0 010	0 001	1 001	
-7	1 111	0 001	0 000	1 000	
-8		0 000			

# Representación “negativos en complemento a 2”

- Negativos: complemento a 1, **más uno**
- Ejemplo: -23 con 6 bits

(recordemos  $bn(23)=10111$ )

Signo: 1

C-1(23)=01000  $\rightarrow$  C-2(23)= 01000+1= 01001

Resultado: 1 01001



# ¡ Aviso Importante !

---

¡Sólo los negativos invierten los bits!





# Representación “negativos en complemento a 2”

---

- ▶ Problema: ¿Qué representa 010111, si está en formato N-2?
- ▶ Respuesta: +23 (¡no invertimos bits, porque es positivo!)
- ▶ Problema: ¿Qué representa 100111, si está en formato N-2?





- ¿Qué representa 100111, si está en formato N-2?
  - Identificar signo: *es negativo*
  - Si es negativo: Por definición 00111 (7) es el complemento a 32 de 25; la solución es -25
  - Alternativa más fácil: si es negativo, al resto de la cadena:
    - a) Restar 1                                  00110
    - b) Invertir bits                                11001
    - c) Convertir de *bn* a base-10              25.              La solución es -25
  - Otra más fácil aún: si es negativo , al resto de la cadena :
    - a) Invertir bits                                11000
    - b) Sumar 1                                     11001
    - c) Convertir de *bn* a base-10              25.              La solución es -25

# Extensión de números y extensión del signo

---

- ▶ Dada una **representación** con  $n$  bits de un cierto número ¿cómo se representaría con  $n+m$  bits?
- ▶ s-m: Insertar  $m$  ceros entre el signo y la magnitud
- ▶ exceso: Insertar  $m$  ceros a la izquierda
- ▶ c-1 y c-2: Repetir el signo  $m$  veces a la izquierda

“Extensión del signo”



# Números en punto fijo

---

- ▶ Naturales y enteros se conocen como “Números en punto fijo, o coma fija”
- ▶ La coma de fracción está **fija** en el extremo derecho de los registros de almacenamiento:

1	1	0	1	1	1
---	---	---	---	---	---

,

- ▶ Se puede, no obstante, **fijar** en otra posición:



# Números en punto fijo

---

1	1	0	1	1	1
---	---	---	---	---	---

,

- ▶ Posiciones de los dígitos:  $i = -2..3$
- ▶ Pesos a la derecha de la coma:  $d_i \cdot 2^i$  (igual)
- ▶ En el ejemplo, los pesos son 0.5 y 0.25
- ▶ Para ejercicios sobre complemento a 2, se suma 1 al dígito más a la derecha.
- ▶ El problema directo es diferente para las partes entera y fracción:



# Convertir 2.5625 a *bn* de 4,4 bits

---

## ► Parte entera:

2:2=1 resto 0

1:2=0 resto 1

0:2=0 resto 0

0:2=0 resto 0

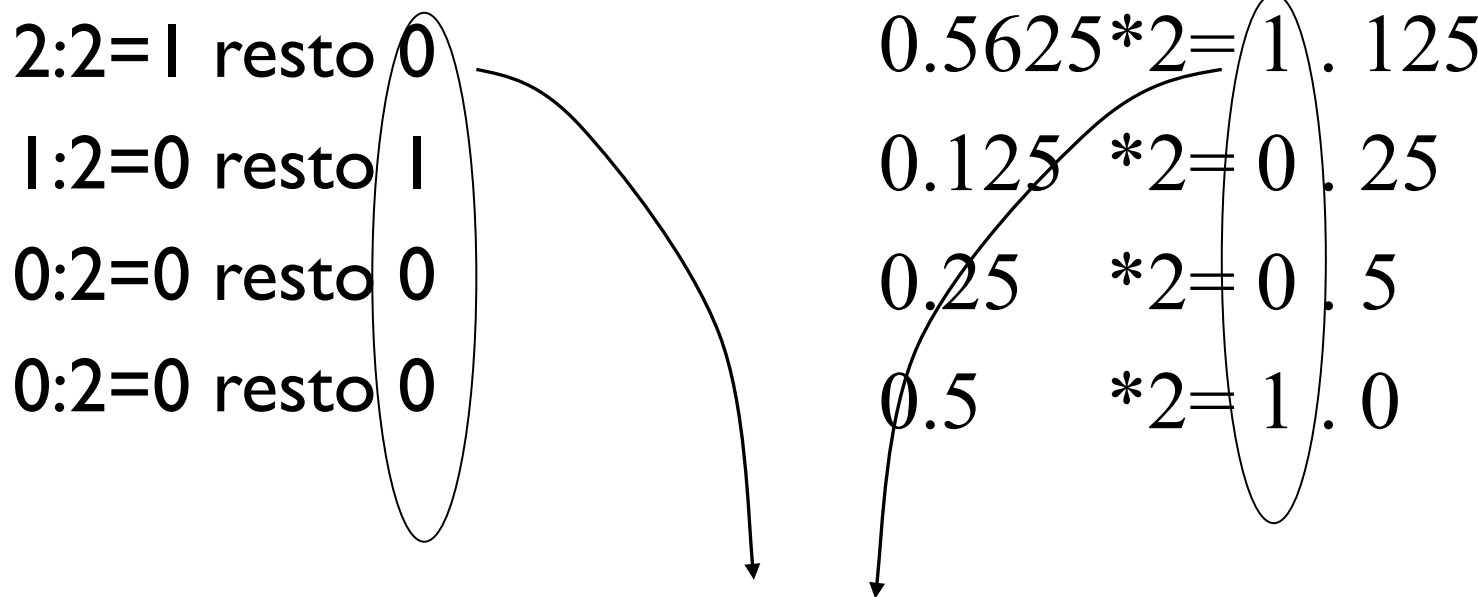
## • Parte fracción:

0.5625\*2=1.125

0.125\*2=0.25

0.25\*2=0.5

0.5\*2=1.0



0	0	1	0,	1	0	0	1
---	---	---	----	---	---	---	---

# Extensión de números por la derecha

---

- ▶ A la derecha de la coma, la extensión con  $m$  bits es:
  - ▶  $s-m$  y  $n-2$ : añadir  $m$  “ceros”
  - ▶  $n-1$ :
    - ▶ añadir  $m$  “ceros” si es positivo,
    - ▶  $m$  “unos” si es negativo



# Ejemplos (de 8 a 16 bits)

---

Sin fracción:

Consideremos el 23 y el -23

Formato	8 bits	16 bits	
Binario natural	00010111.	0000000000010111.	0's a izqda
S-M	00010111.	0000000000010111.	0's a izqda. magnitud
N-1	00010111.	0000000000010111.	Ext. de signo
N-2	00010111.	0000000000010111.	Ext. de signo

Formato	8 bits	16 bits	
Binario natural	No valido	No valido	
S-M	10010111.	1000000000010111.	0's a izqda. magnitud
N-1	11101000.	1111111111101000.	Ext. de signo
N-2	11101001.	1111111111101001.	Ext. de signo





# Ejemplos (de 8 a 16 bits)

Con fracción:  $6.2 \rightarrow 10.6$

Consideremos el 23.25 y el -23.25

Formato	8 bits	16 bits	
Binario natural	010111.01	0000010111.010000	0's a izqda., 0's a dcha
S-M	010111.01	0000010111.010000	0's a izqda. Magnitud, 0's dcha
N-1	010111.01	0000010111.010000	Ext. de signo, 0's a dcha
N-2	010111.01	0000010111.010000	Ext. de signo, 0's a dcha

Formato	8 bits	16 bits	
Binario natural	No valido	No valido	
S-M	110111.01	1000010111.010000	0's a izqda. Magnitud, 0's dcha
N-1	101000.10	1111101000.101111	Ext. de signo, 1's a dcha!!!
N-2	101000.11	1111101000.110000	Ext. de signo, 0's a dcha

Representación de datos numéricos reales  
(números en punto flotante)

# Coma flotante

---

- ▶ La representación en punto fijo no es apropiada para ciertos números:
  - ▶  $6.023 \cdot 10^{23}$  ,  $1.6 \cdot 10^{-19}$ , etc...
- ▶ Es preferible usar la forma: mantisa, base, exponente
- ▶ El exponente hace “flotar la coma” de la mantisa



# Coma flotante

---

- ▶ Si la mantisa está representada en una base B (p.e. B=2), entonces la base en punto flotante debe ser una potencia de B:

$$1.011 \cdot 2^{12} = 10.11 \cdot 2^{11}$$



$$1.011 \cdot 4^6 = 101.1 \cdot 4^5$$

- ▶ Incrementos y **D**ecrementos de exponente implican desplazamientos a **I**zquierda y **D**erecha de la coma *un número **entero** de posiciones*



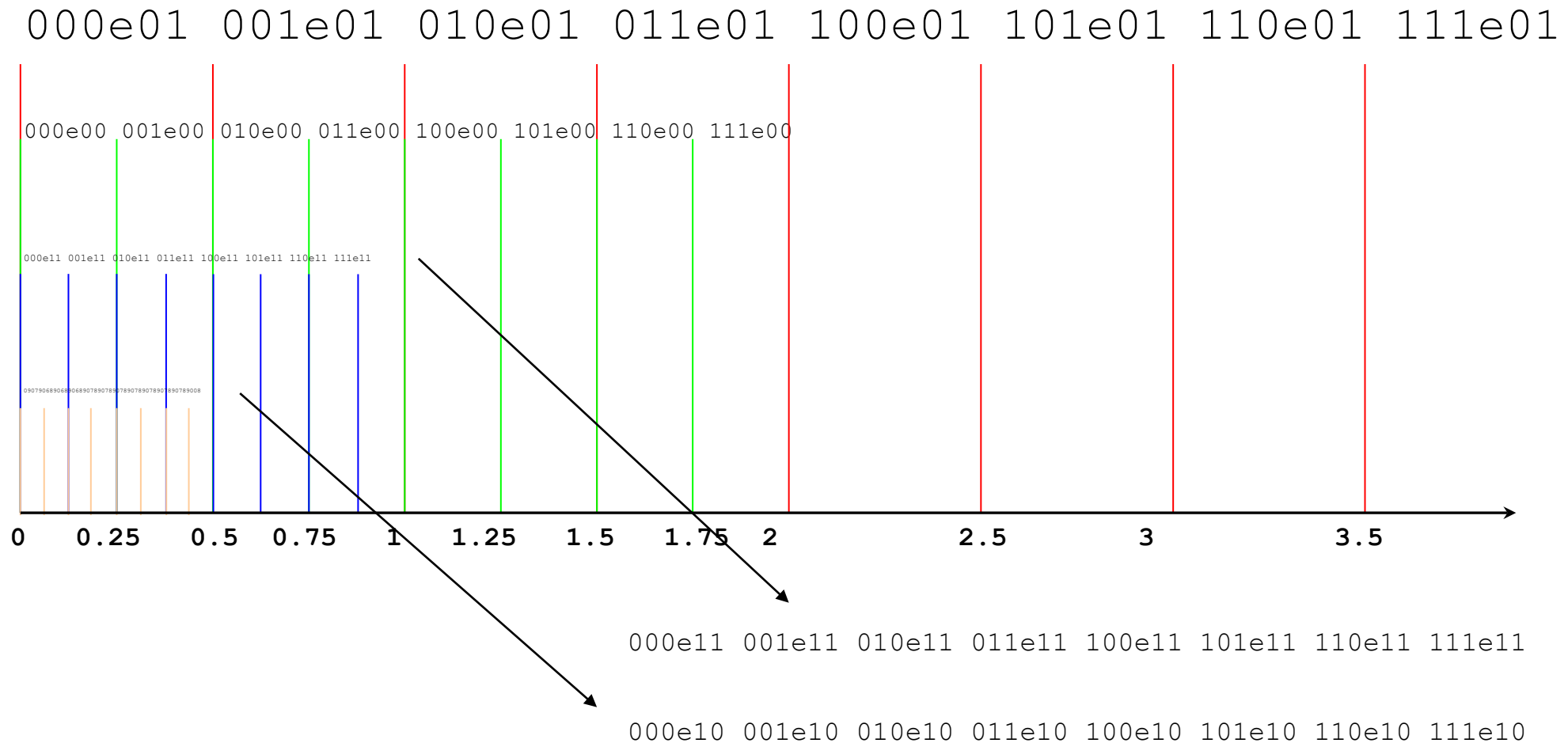
# Coma flotante

---

- ▶ Fijada la base, un número en coma flotante es una pareja de números en punto fijo: Mantisa-Exponente
- ▶ Ejemplo:
  - ▶ Base 2
  - ▶ 3 bits de mantisa  $bn$ , en la forma  $m,mm$
  - ▶ 2 bits exponentes (representados en C-2)
- ▶ 1,00 e 11   $1.0 \cdot 2^{-1} = 0.5$
- ▶ 1.10 e 01   $1.5 \cdot 2^1 = 3$

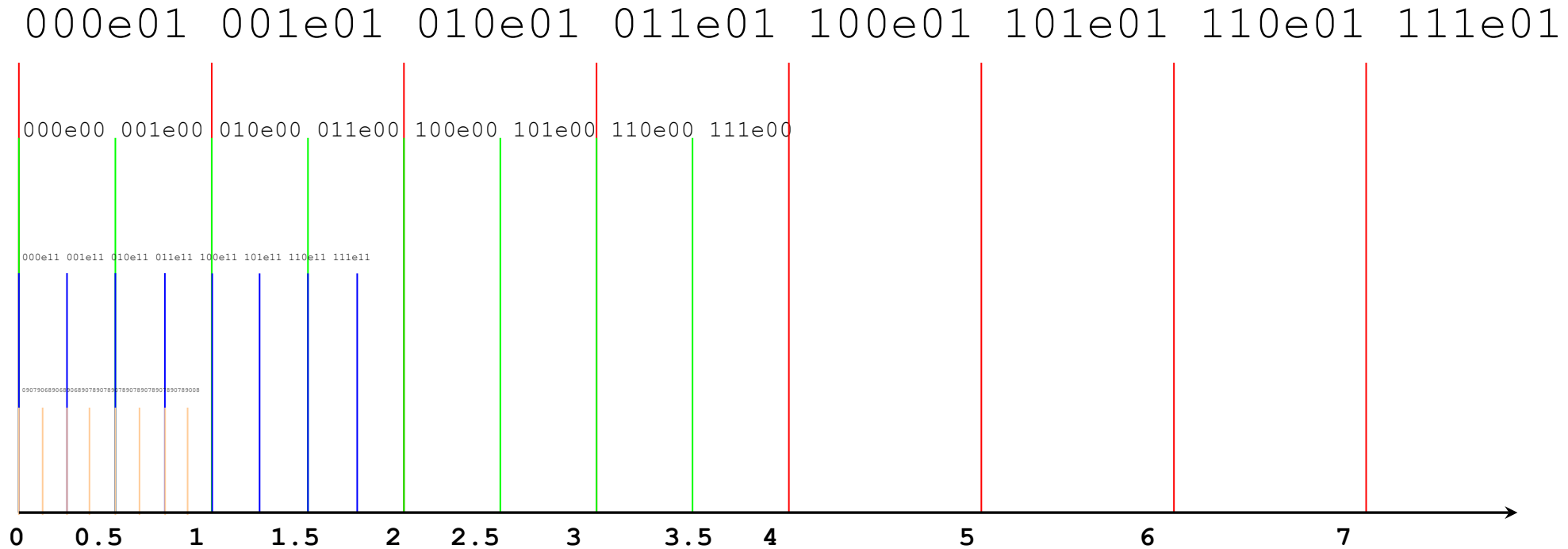


# Coma Flotante, N° representables



# Coma Flotante. Bases superiores. Ej. 4

---



Mayor rango de representación. Menor precisión

(misma cantidad de números representables, pero más dispersos)

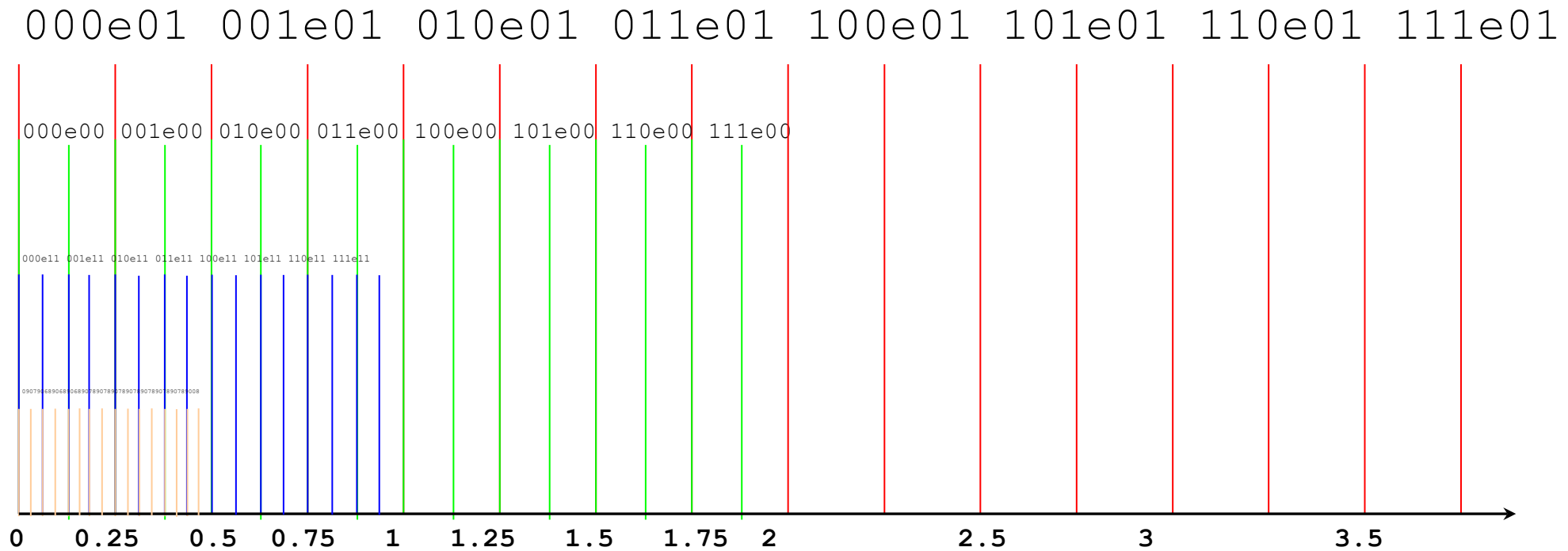






# Coma Flotante. Aumento de mantisa

---



Mismo rango de representación. Mayor precisión



# Normalización de mantisas

---

- ▶ 6,023e23   60,23e22   602,3e21 varias formas de representar un mismo número
- ▶ Ejemplo: 5 bits  $\rightarrow$  32 combinaciones; pero sólo 20 números representables (redundancia)
- ▶ Solución: Fijar la coma respecto al dígito más significativo (DMS): **Normalización.**
- ▶ Ej.: Sólo la forma 6.023e23 es válida
- ▶ El cero es un caso especial (no tiene DMS)



# Normalización de mantisas. Bit implícito

---

- ▶ Sólo las combinaciones: IMM-EE son válidas
  - ▶ “Gastamos” un biestable para guardar siempre un 1
  - ▶ Solución: Al representar, no guardar ese 1
  - ▶ Bit implícito: 1.011e11 se guarda como 011e11  
(aumentamos la precisión sin coste, al evitar la redundancia)
- ¡Al operar con el número, hay que recuperar el bit implícito!



# IEEE-754 de 32 bits

---

- ▶ Exponente E en exceso a 127 con 8 *bits*
- ▶ Mantisa en formato signo S (1 *bit*) - magnitud M
- ▶ La magnitud M tiene 24 bits, pero está normalizada con BMS a la izquierda de la coma, y el BMS se hace implícito (23 *bits*)

$$(-1)^S \cdot 1.M \cdot 2^{(E-127)}$$



# IEEE-754 de 32 bits

---

- ▶ 8 bits para exponentes (rangos de  $10^{\pm 38}$ )
- ▶ 23 bits para mantisa (precisión equivalente a 7 dígitos en base 10)

```
float a=3.14159263523452345;
```

```
main()
```

```
{
```

```
printf("%22.20f\n", a);
```

```
}
```

```
3.14159274101257324219
```



# Conversión de N a IEEE-754

---

- ▶ Paso 1. Identificar el signo S
- ▶ Paso 2. Expresar el número en formato mantisa  $\cdot 2^{\text{exponente}}$ 
  - ▶ Ya está así! (vete al paso 3)
  - ▶ Si es entero y relativamente pequeño, exprésalo como mantisa  $\cdot 2^0$
  - ▶ Igual si es entero y fracción, o fracción grande
  - ▶ Lo siguiente no es frecuente, y mucho menos en exámenes de esta asignatura:
    - ▶ Si es un entero relativamente grande, divídelo sucesivamente por 2
    - ▶ Si es fracción y relativamente pequeña, multiplica sucesivamente por 2
    - ▶ Si es entero y muy grande, o fracción muy pequeña, expresado como  $m \cdot 10^{\text{exp}}$

La forma más rápida suele consistir en determinar un valor aproximado para x en esta fórmula

$2^x \simeq 10^{\text{exponente}}$  y a partir de ahí, calcular el coeficiente que permita la conversión



# Conversión de N a IEEE-754 (2)

---

- ▶ Paso 3. (Innecesario) **Intenta facilitar el paso 4**
  - ▶ Si la parte fracción de la mantisa es 0.5, 0.25, 0.125, 0.0625 o alguna de sus sumas combinadas, lo más probable es que se pueda convertir a un entero. (Prueba a multiplicar la mantisa por 2 sucesivamente, a ver si se simplifica)
    - Ejemplo  $2.875 = 23 \cdot 2^{-3}$
- ▶ Paso 4. **Convierte la mantisa a binario**, usando el procedimiento ya conocido
- ▶ Paso 5. **Normaliza, flotando la coma y cambiando el exponente**
- ▶ Paso 6. **Suma 127 al exponente y representa en bn con 8 bits!**



# Conversión desde IEEE754

---

- ▶ Simplemente, Convierte el exponente a decimal y aplica la expresión

$$(-1)^S \cdot 1.M \cdot 2^{(E-127)}$$

- ▶ Y si es necesario, convierte la mantisa a base 10





# Casos especiales

---

- ▶ **E=255 (11111111)**

- ▶ **M=0**

- ▶ **S=0**      $\Rightarrow +\infty$

- ▶ **S=1**      $\Rightarrow -\infty$

- ▶ **M $\neq$ 0**      $\Rightarrow$  NaN (Not A Number)

- ▶ **E=0 (00000000)**

- ▶ **M=0**      $\Rightarrow 0$

- ▶ **M $\neq$ 0**      $\Rightarrow (-1)^s \cdot 0.M \cdot 2^{-126}$  (no está normalizado)



# IEEE-754 de 64 bits

---

- ▶ 11 bits para exponentes (rangos de  $10^{\pm 308}$  !)
- ▶ 52 bits para mantisa (precisión equivalente a 15 dígitos en base 10)

```
double a=3.14159263523452345;  
main()  
{  
    printf("%22.20f\n",a);  
}  
3.14159263523452336742
```



# Representación de datos alfanuméricos

# ASCII

---

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SPC	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	:	;	<	=	>	?
4	@	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>	<b>G</b>	<b>H</b>	<b>I</b>	<b>J</b>	<b>K</b>	<b>L</b>	<b>M</b>	<b>N</b>	<b>O</b>
5	<b>P</b>	<b>Q</b>	<b>R</b>	<b>S</b>	<b>T</b>	<b>U</b>	<b>V</b>	<b>W</b>	<b>X</b>	<b>Y</b>	<b>Z</b>	[	\	]	^	_
6	`	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>g</b>	<b>h</b>	<b>i</b>	<b>j</b>	<b>k</b>	<b>l</b>	<b>m</b>	<b>n</b>	<b>o</b>
7	<b>p</b>	<b>q</b>	<b>r</b>	<b>s</b>	<b>t</b>	<b>u</b>	<b>v</b>	<b>w</b>	<b>x</b>	<b>y</b>	<b>z</b>	{		}	~	DEL



# UNICODE

- ▶ Lista ordenada de todos los caracteres conocidos
  - ▶ Universal Character Set
  - ▶ Code Point (¿por qué no llamarlo identificador de carácter?)
- ▶ ¿8bit, 16bit, 32bit?
  - ▶ Opc. A: Compromiso entre eficiencia y universalidad
  - ▶ Opc. B: UTF-8 ✓
    - ▶ Utilizar 1 (ascii), 2 o 4 bytes (n ascii)
    - ▶ 1 byte, a menos que sea 11XXXXXX (multibyte char)
    - ▶ 10XXXXXX (continuation chars)

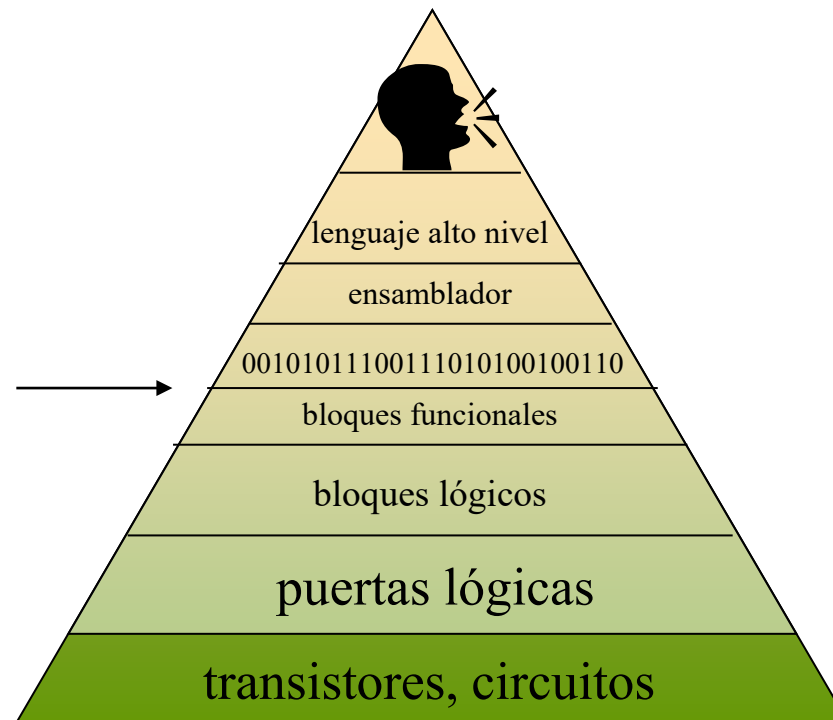
Hola este es un ejemplo:  
وسايل

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	48	6F	6C	61	20	65	73	74	65	20	65	73	20	75	6E	20	Hola este es un
00000010	65	6A	65	6D	70	6C	6F	3A	0D	0A	9	88	D8	B3	D8	A7	ejemplo: 00000010 00000020
00000020	DB	8C	D9	84	20												00000020

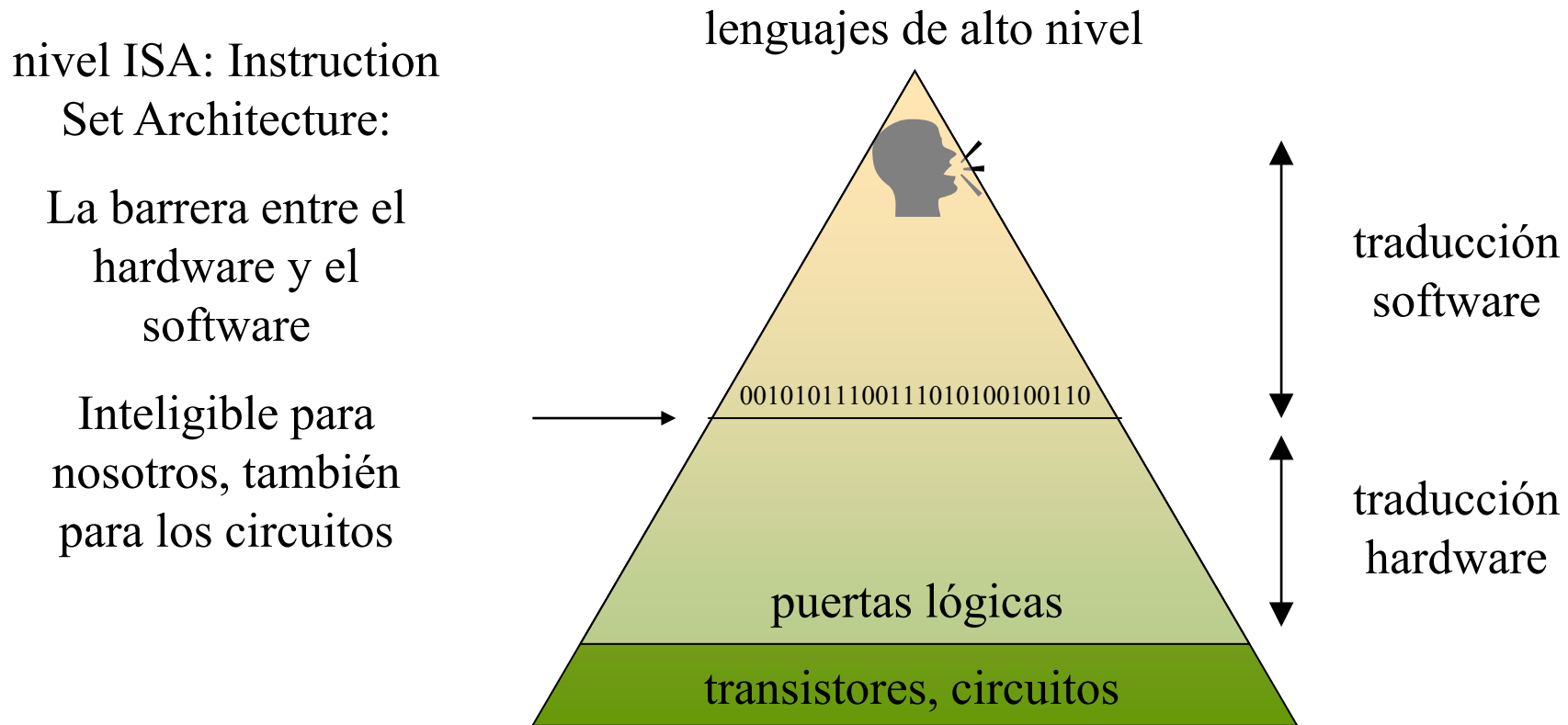
- ▶ UNICODE también define UTF-16 y UTF-32
  - ▶ necesitan especificar endianismo
  - ▶ Byte Ordering Mask: FEFF (big), FFFE (little)

# Instrucciones y Direcciones

# Arquitectura del Juego de Instrucciones



# Arquitectura del Juego de Instrucciones





# Las operaciones del hardware

---

- ▶ Operaciones que el computador realiza en el nivel que estudiamos:
  - ▶ Ej.: Sumar, Mover , Saltar
- ▶ Instrucción: Secuencia binaria donde se indica:
  - ▶ La operación a realizar (Código de Operación)
  - ▶ Con qué datos la va a realizar
- ▶ Efectos laterales (PC, SR, índices)
- ▶ Lenguaje ensamblador: representación intermedia formada por:
  - ▶ Mnemónico. Ej.: ADD, MOVE, JUMP
  - ▶ Operandos - El nombre de un registro, un dato, una dirección  
ADD REG1, REG2, REG3
- ▶ Traducción simple
  - ▶ Ej.: ADD → 0010.

# Operandos y direcciones

- ▶ Las operaciones requieren **operandos**

- ▶ add destino, dato1 (“fuente 1”), dato2 (“fuente 2”)
- ▶ jump destino
- ▶ mov destino, dato fuente    x=y , a=**17**

- ▶ **Constantes** o **variables**

**operandos** → **direcciones**

- ▶ mov dirección\_destino, dirección\_fuente
- ▶ Posibles direcciones: **Registros y Memoria**
- ▶ Modos de direccionamiento
- ▶ Operandos implícitos y explícitos: Ej.: INCR X

# Número de operandos explícitos

---

- ▶ Depende de la operación a realizar
  - ▶ Operaciones unarias, binarias, etc.
  - ▶ Las binarias (dos operandos, un resultado) se usan de referencia

## a) Máquinas de 3 direcciones

ADD Destino, Fuente, Fuente

## b) Máquinas de 2 direcciones

ADD Fuente, Destino    (Destino  $\leftarrow$  Fuente + Destino)



# Número de operandos explícitos

---

## c) Máquinas de 1 dirección

- ▶ Ej.: ADD dato    ( $A \leftarrow A + \text{dato}$ )  
                     $\longleftrightarrow$
- ▶ Registro acumulador A

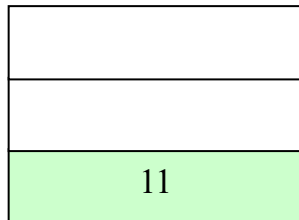
## d) Máquinas de 0 direcciones

- ▶ Ej.: ADD  
           $\longleftrightarrow$



# Cero direcciones: La Pila

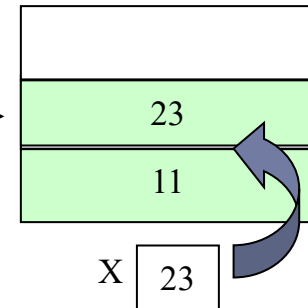
1



2

INTRODUCIR X (PUSH X)

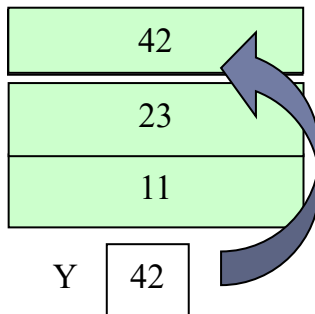
puntero →



3

INTRODUCIR Y (PUSH Y)

puntero →

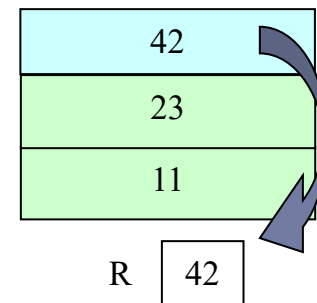


4

EXTRAER R (POP R)

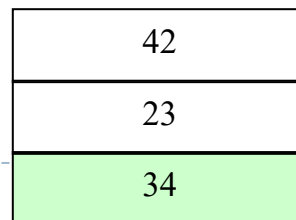
puntero →

puntero →



5

ADD



# Clasificación de las arquitecturas

---

## ▶ Arquitecturas de pila

- ▶ 2 instrucciones usan un operando, el resto, como ADD, ninguno.  
PUSH fuente, POP destino, ADD, MUL, SUB, NEG
- ▶ Ej.: TII 000, HP 3000, B5500, B6500, 80x87
- ▶ Notación polaca inversa (o postfija) es útil

## ▶ Arquitectura de acumulador

- ▶ Casi todas las instrucciones utilizan 1 operando
- ▶ Presente en las arquitecturas primitivas: IAS, EDSAC, IBM701, 6800, 8008

## ▶ Arquitecturas de registro de propósito general

- ▶ De 2 y 3 direcciones.



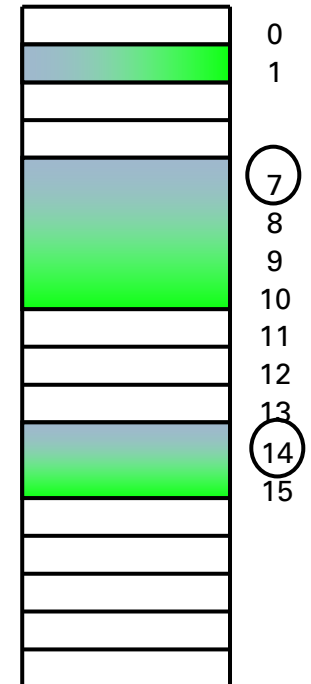


# Modos de direccionamiento



# Direcciones de memoria

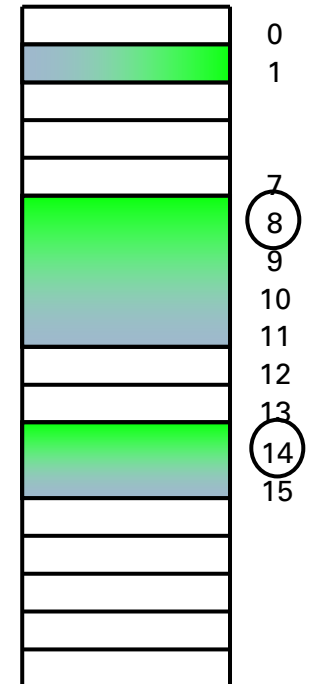
- ▶ La memoria es una estructura ordenada de los datos
- ▶ Un dato puede ocupar más de una posición
  - ✓ La dirección del dato es la de la posición más baja
- ▶ Los datos deben estar alineados en memoria
  - ✓ Datos de tamaño n en direcciones múltiplo de n
- ▶ Situación del dato en múltiples posiciones
  - ✓ Big “endian” ▶ dirección (dato) = dirección (palabra más significativa)
  - ✓ Little “endian” ▶ dirección (dato) = dirección (palabra menos significativa)





# Direcciones de memoria

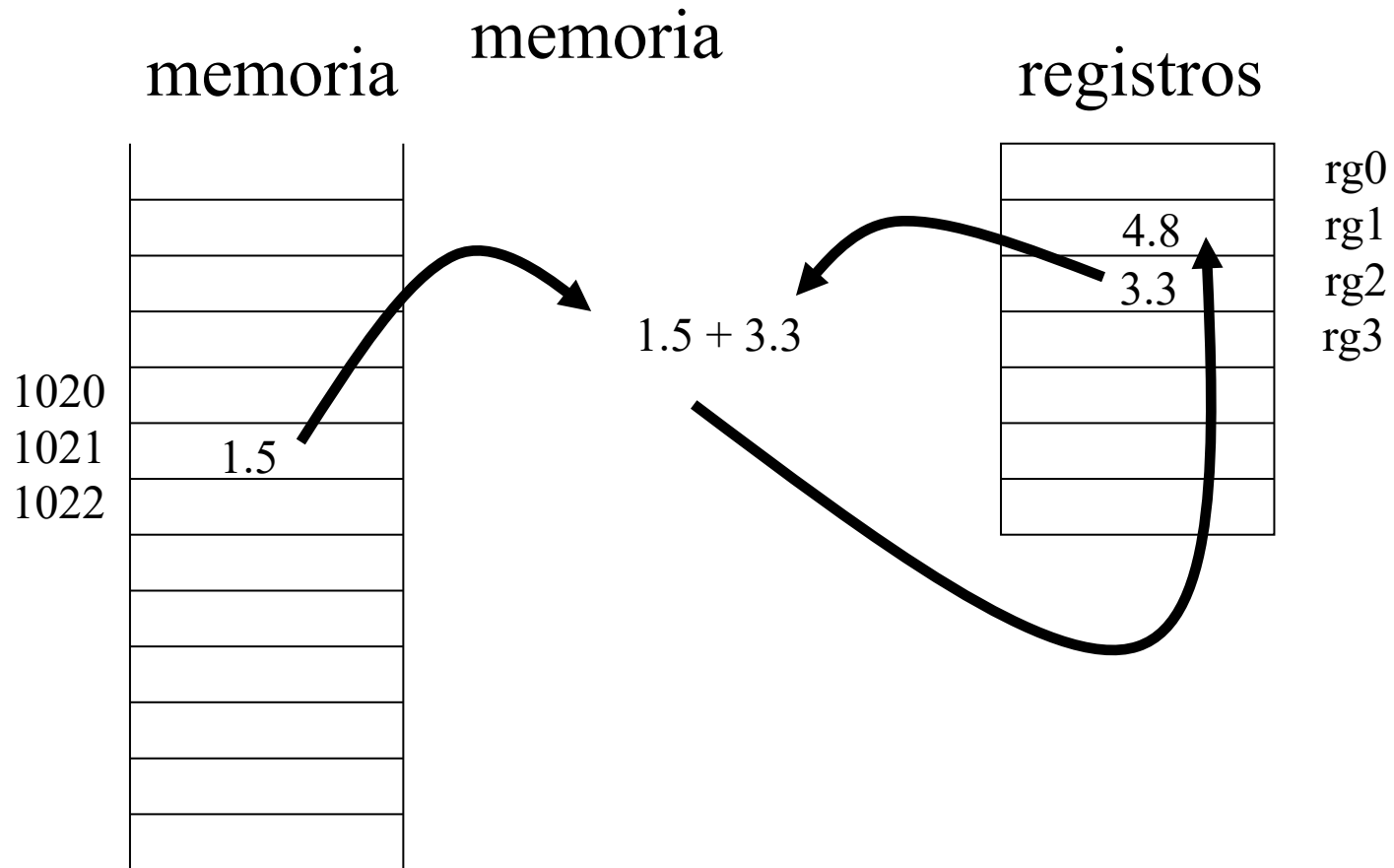
- ▶ La memoria es una estructura ordenada de los datos
- ▶ Un dato puede ocupar más de una posición
  - ✓ La dirección del dato es la de la posición más baja
- ▶ Los datos deben estar alineados en memoria
  - ✓ Datos de tamaño n en direcciones múltiplo de n
- ▶ Situación del dato en múltiples posiciones
  - ✓ Big “endian” ▶ dirección (dato) = dirección (palabra más significativa)
  - ✓ Little “endian” ▶ dirección (dato) = dirección (palabra menos significativa)



## Modos directos, Ejemplo

---

ADD R1,R2,1021



## MD indirectos

---

- ▶ La instrucción indica el lugar donde se encuentra la dirección del dato (“**indirecciones**”)
  - ✓ un registro que guarda la dirección en memoria de un dato (**MD indirecto por registro, indirecto**)
  - ✓ una posición de memoria que guarda la dirección en memoria de un dato (**MD indirecto por memoria**)
- ▶ En los MD indirectos, el dato casi siempre se encuentra en la memoria.
- ▶ Puede haber varios grados de indirección



# Modos de direccionamiento directos

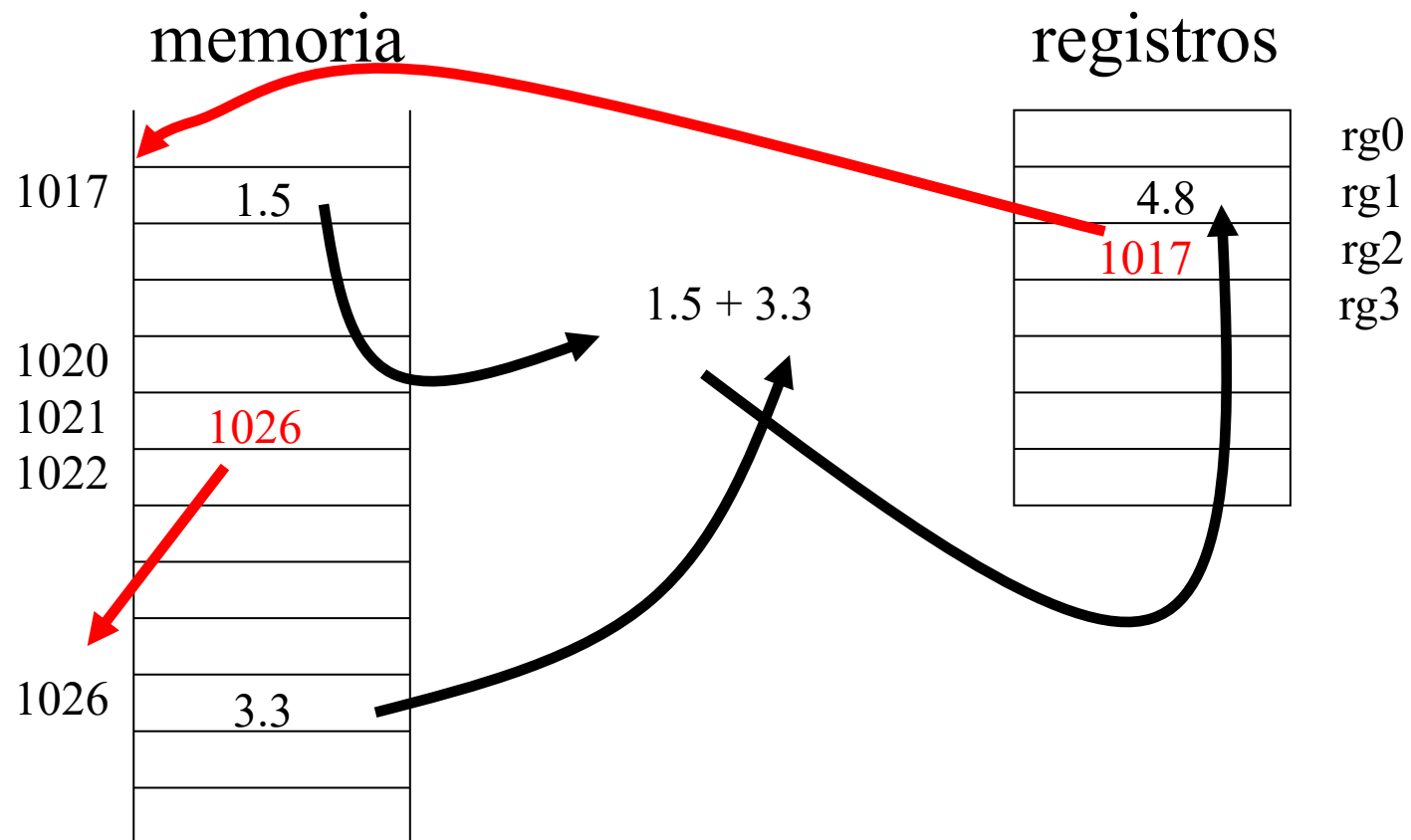
---

- ▶ Almacén del computador: registros y memoria.
- ▶ Una dirección puede indicar “directamente”
  - ✓ el nombre de un registro (MD directo por registro)
  - ✓ la posición en la memoria (MD directo por memoria)
- ▶ Utilizado para el acceso a variables escalares
- ▶ MD directo por registro se usa en el 50% de instrucciones



## Modos indirectos, ejemplo

ADD R1, (R2),(1021)



# Otros modos de direccionamiento

---

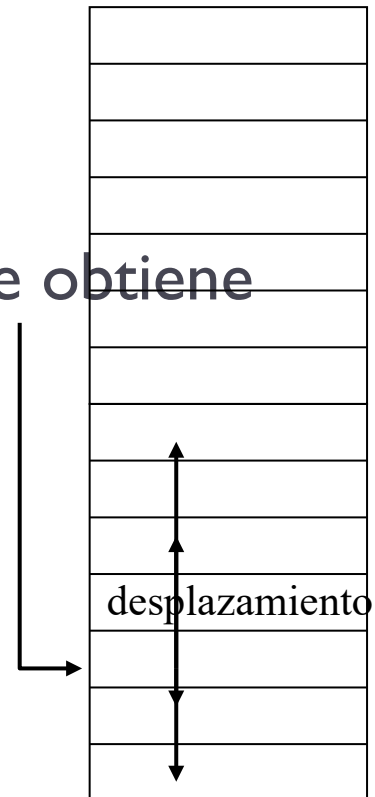
- Direccionamiento **implícito** .El operando no aparece en la instrucción

- **MD inmediato, o literal.** Las instrucciones incluyen el propio dato

Ej.:ADD “dirección destino”, # 1.5 , # 3.3

- **MD relativos.** La posición del dato en memoria se obtiene

- **Sumando,**
  - Encadenando,
  - o realizando otra operación elemental ( $a + \text{shift}(b)$  )
- entre dos (o más) direcciones. base



# Direccionamiento relativos: muchos sabores, pero...

---



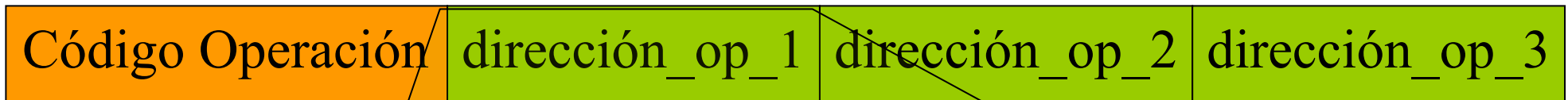
Nombres	Base		Desplazamiento
Relativo, relativo a base	[Registro Base]		número entero (+ o -)
Indexado , Relativo a índice	dirección de memoria		[Registro índice]
Base-Indexado, Indexado	[Registro Base]		[Registro índice]
Indexado a base	[Registro Base]	[Registro índice]	número entero (+ o -)



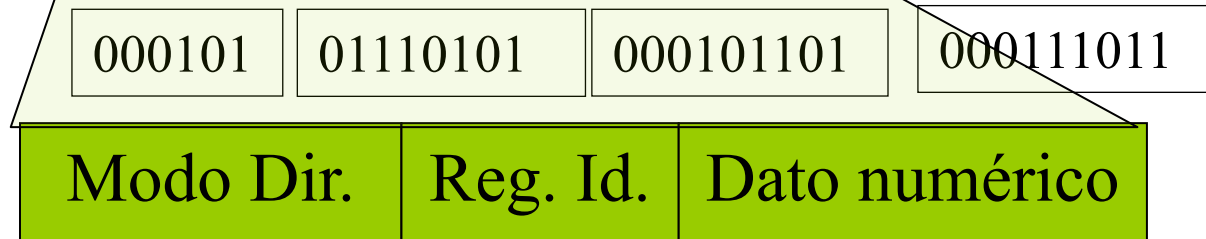
Juntándolo todo:

# La instrucción

---



Ej.:



- ✓ Una dirección
- ✓ Un valor constante
- ✓ Un desplazamiento





# Formato de las Instrucciones

# Longitud de la instrucción

---

- ▶ Fija:

- ▶ Múltiplo de 8 bits. Frecuentemente **32** bits.
- ▶ 8 bits en el procesador de la primera lección

- ▶ Variable:

- ▶ Sólo en procesadores antiguos
- ▶ Mejor aprovechamiento:
  - ▶ Más bits en instrucciones con más operandos



# El campo código de operación

---

- ▶ Indica la operación a realizar en el ciclo de instrucción actual
- ▶ Longitud de la secuencia binaria
  - ▶  $\log_2(\text{n}^\circ \text{ de instrucciones}) \rightarrow$  Si es un campo de longitud fija
  - ▶ Unos **6 bits** en procesadores modernos (64 instrucciones)
  - ▶ Más de 8 bits en procesadores CISC (+256)
- ▶ Codificación arbitraria. Ejemplos:
  - ▶ ADD : 010000
  - ▶ NOP: 000 (instrucción que no hace nada; frecuentemente, 000..0)
  - ▶ En caso de longitud variable, se reservan opcodes cortos a instrucciones con más argumentos (operandos), para disponer de más bits
- ▶ A veces, extensible con subcódigos de operación
  - ▶ La macroinstrucción 000000 en MIPS se extiende con 6 bits adicionales



# Los campos de operandos

---

- ▶ Tantos como requiera la instrucción
- ▶ Normalmente de 0 a 3 campos
- ▶ Cada campo podría precisar de:
  - ▶ Bits que indiquen el modo de direccionamiento
  - ▶ En algunos casos, bits para nombrar un registro:
    - ▶ Si es modo directo por registro
    - ▶ Si es modo indirecto por registro
    - ▶ Si es modo relativo
  - ▶ En algunos casos, un dato (numérico) adicional
    - ▶ Si es directo por memoria (un número natural)
    - ▶ Si es modo relativo (un desplazamiento: un entero)
    - ▶ Si es inmediato (normalmente un número entero)



# 1. El modo de direccionamiento

---

- ▶ Cada operando puede usar un modo diferente
- ▶ Antiguamente:
  - ▶ Se permitían varios modos por operando
  - ▶ Normalmente 1 a 3 bits (2 a 8 modos)
- ▶ Actualmente
  - ▶ En cada operando, un sólo modo es posible
    - ▶ ADD Reg Reg Reg
    - ▶ ADDI Reg Reg Inm
    - ▶ LW Reg Rel
  - ▶ No es necesario este campo en ningún operando
    - ▶ **0 bits**  $\rightarrow \log_2(1)$



## 2. Campo Registro

---

- ▶ Nombre del registro utilizado:
  - ▶ Si es directo por registro
  - ▶ Si es indirecto por registro
  - ▶ Si es relativo (a registro)
- ▶ Tamaño: de 2 a 6 bits
  - ▶ 2 bit(procesador 8086, con registros AX, BC, CX, DX)
  - ▶ **5 bit** En MIPS y DLX con 32 registros



### 3. Campo numérico

---

- ▶ En modos relativo → Desplazamiento
- ▶ En modos inmediatos → Valor
- ▶ En modo directo a memoria → Dirección (en desuso)
- ▶ Casi siempre un natural o entero
- ▶ En MIPS → **16 bit en Compl-2**



# Ejemplo: ADDI en MIPS

---

- ▶ **ADDI Rgdestino, Rgfuelle, Inmediato**
- ▶ 6 bits de opcode
- ▶ **Operando 1: sólo es válido directo x registro**
  - ▶ Sin campo MD
  - ▶ Sin campo numérico
  - ▶ Campo nombre del registro: 5 bit
- ▶ **Operando 2 → igual**
- ▶ **Operando 3 → Sólo es válido modo inmediato**
  - ▶ Entero de 16 bits
- ▶ 000000 RRRRR RRRRR IIIIIIIIIIIIIIIIIIIII
- ▶ 001000 01001 10111 111111111111111100
- ▶ ADDI R23, R9, #-4      ¿¿!!!!????





# Ensamblador $\leftrightarrow$ Binario

---

- ▶ Traducción simple, aunque...
- ▶ A veces, distinto orden en los operandos
- ▶ A veces, usan macros:
  - ▶ Una instrucción asm: 2 instrucciones binarias
- ▶ Uso de símbolos: etiquetas que representan valores binarios
- ▶ ...



---

de **CISC**

Computador de Conjunto de Instrucciones Complejo  
(Complex Instruction Set Computer)

a **RISC**

Computador de Conjunto de Instrucciones Simple  
(Reduced Instruction Set Computer)



# El repertorio de Instrucciones

# Tipos de Operación

---

- ▶ **Movimiento de datos** Registro-memoria, registro-registro  
Acceso a pila  
Entrada/Salida
- ▶ **Aritméticas** Entero  
Flotante  
Desplazamientos aritméticos  
Decimal (BCD, Exceso-3, etc.)
- ▶ **Lógicas** Operaciones lógicas, AND, OR, XOR, NOT  
Desplazamientos lógicos y rotaciones  
Test, Set, Reset, Conteo de bits, etc.
- ▶ **Control** Saltos incondicionales  
Saltos condicionales o bifurcaciones  
Rutinas e interrupciones software  
No clasificadas: NOP, HALT, etc.



# Instrucciones de movimiento de datos

---

- ▶ Movimiento de datos (genérica) MOVE, MOVB, MOVI
- ▶ Si la fuente/destino es la memoria principal: LOAD/STORE ( LD / ST )
- ▶ Otras: EXCHANGE, Extracción de bits , etc.
- ▶ Acceso a la pila PUSH,POP, SWAP



# Instrucciones aritméticas

---

- ▶ Aritmética de punto fijo o punto flotante
- ▶ En binario o decimal (BCD, Ex3)
- ▶ Operaciones básicas: Suma, resta, multiplicación y división
- ▶ Desplazamientos aritméticos
- ▶ Trigonómicas, logaritmos, exponenciales,  $a \cdot b + c$
- ▶ Instrucciones de conversión de datos (BCD, enteros, flotantes)



# Instrucciones lógicas

---

- ▶ Instrucciones binarias: Operación bit a bit
  - ▶ Or, Nor, And, Nand, Xor, Xnor, etc.
- ▶ Instrucciones unarias:
  - ▶ Bit a bit: Not
  - ▶ Sobre los bits: AND, OR, Conteo, Primer\_I, etc.
  - ▶ Desplazamientos lógicos y rotaciones, etc...
- ▶ Instrucciones SET, RESET, etc..



# Instrucciones de control

---

- ▶ **JUMP**

- ▶ **BRANCH**

  - Dependen de condiciones establecidas por instrucciones anteriores (flags) BNZ destino

  - Dependen de condiciones establecidas por instrucciones actuales

    - BEQZ reg, destino

    - BEQ rg1, rg2, destino

- ▶ **CALL y RET**

- ▶ **INT e IRET**

- ▶ **HALT, SU/US, NOP, etc.**





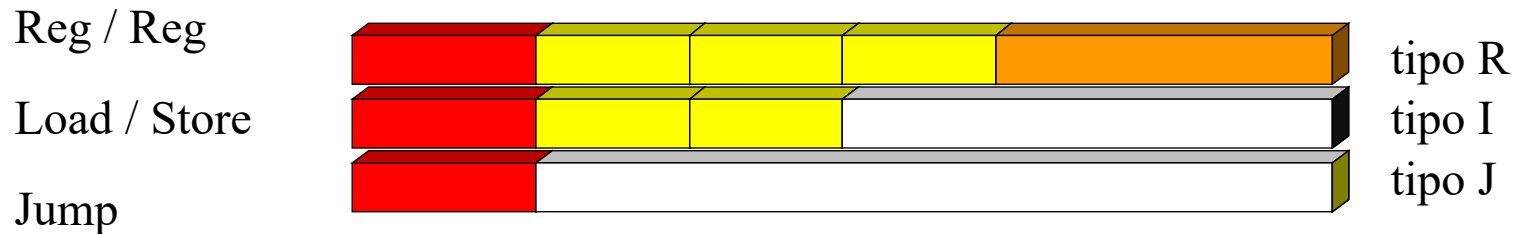
# MIPS y su formato de instrucción

---

- ▶ **Características del computador**
  - ▶ 32 bits (tamaño típico de sus estructuras)
  - ▶ 64 instrucciones (6 bits OpCode). Número elevado de subinstrucciones
- ▶ **La instrucción preestablece los MD de sus operandos**
  - ▶ 0 bits para Modos de Direccionamiento
- ▶ **3 Modos posibles**
  - ▶ Inmediato
  - ▶ Directo a registro
  - ▶ Relativo a registro (único modo que accede a memoria)
- ▶ **3 formatos, a los que se deben ajustar las instrucciones**
  - ▶ Opcode 0 → Formato R (tres campos registros)
  - ▶ Opcodes 2 y 3 → Formato J (1 dato numérico de 26 bits)
  - ▶ Resto → Formato I (2 campos registros + 1 dato numérico)
  - ▶ (Opcodes 0100XX reservados para el futuro)



# Formatos del MIPS



► El resto de instrucciones de ajustan a los 3 formatos:

► Ejemplos: Tipo I

- load R1, 1024(R2)
- store R2, 1023(R1)
- addi R1,R2,#500 (suma R2 con valor inmediato)
- beqz R1, 200 (salta a PC+200 si R1 es cero)

Formato	Bits					
	31	26	25	2	20	1
			1		6	
tipo-R	000000		Rs1		Rs2	Rd
tipo-I	opcode		Rs1		Rd	immediato
tipo-J	opcode		valor			

Instr.	Descripción	Formato	Opcode/Subcode	Operación (estilo-C)
ADD	add	R	0x20	$Rd = Rs1 + Rs2$
ADDI	add immediate	I	0x08	$Rd = Rs1 + \text{extend}(\text{immediate})$
AND	and	R	0x24	$Rd = Rs1 \& Rs2$
ANDI	and immediate	I	0x0c	$Rd = Rs1 \& \text{immediate}$
BEQZ	branch if equal to zero	I	0x04	$PC += (Rs1 == 0 ? \text{extend}(\text{immediate}) : 0)$
BNEZ	branch if not equal to zero	I	0x05	$PC += (Rs1 != 0 ? \text{extend}(\text{immediate}) : 0)$
J	jump	J	0x02	$PC += \text{extend}(\text{value})$
JAL	jump and link	J	0x03	$R31 = PC + 4 ; PC += \text{extend}(\text{value})$
JALR	jump and link register	I	0x13	$R31 = PC + 4 ; PC = Rs1$
JR	jump register	I	0x12	$PC = Rs1$
LHI	load high bits	I	0x0f	$Rd = \text{immediate} \ll 16$
LW	load woRd	I	0x23	$Rd = \text{MEM}[Rs1 + \text{extend}(\text{immediate})]$
OR	or	R	0x25	$Rd = Rs1   Rs2$
ORI	or immediate	I	0x0d	$Rd = Rs1   \text{immediate}$
SEQ	set if equal	R	0x28	$Rd = (Rs1 == Rs2 ? 1 : 0)$
SEQI	set if equal to immediate	I	0x18	$Rd = (Rs1 == \text{extend}(\text{immediate}) ? 1 : 0)$
SLE	set if less than or equal	R	0x2c	$Rd = (Rs1 \leq Rs2 ? 1 : 0)$
SLEI	set if less than or equal to immediate	I	0x1c	$Rd = (Rs1 \leq \text{extend}(\text{immediate}) ? 1 : 0)$
SLL	shift left logical	R	0x04	$Rd = Rs1 \ll (Rs2 \% 8)$
SLLI	shift left logical immediate	I	0x14	$Rd = Rs1 \ll (\text{immediate} \% 8)$
SLT	set if less than	R	0x2a	$Rd = (Rs1 < Rs2 ? 1 : 0)$
SLTI	set if less than immediate	I	0x1a	$Rd = (Rs1 < \text{extend}(\text{immediate}) ? 1 : 0)$
SNE	set if not equal	R	0x29	$Rd = (Rs1 != Rs2 ? 1 : 0)$
SNEI	set if not equal to immediate	I	0x19	$Rd = (Rs1 != \text{extend}(\text{immediate}) ? 1 : 0)$
SRA	shift right arithmetic	R	0x07	as SRL & see below
SRAI	shift right arithmetic immediate	I	0x17	as SRLI & see below
SRL	shift right logical	R	0x06	$Rd = Rs1 \gg (Rs2 \% 8)$
SRLI	shift right logical immediate	I	0x16	$Rd = Rs1 \gg (\text{immediate} \% 8)$
SUB	subtract	R	0x22	$Rd = Rs1 - Rs2$
SUBI	subtract immediate	I	0x0a	$Rd = Rs1 - \text{extend}(\text{immediate})$
SW	store woRd	I	0x2b	$\text{MEM}[Rs1 + \text{extend}(\text{immediate})] = Rd$
XOR	exclusive or	R	0x26	$Rd = Rs1 \wedge Rs2$
XORI	exclusive or immediate	I	0x0e	$Rd = Rs1 \wedge \text{immediate}$