

# Bits, bytes, y enteros

95.57/75.03 Organización del computador

---

**Docentes:** Patricio Moreno y Adeodato Simó

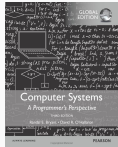
1.<sup>er</sup> cuatrimestre de 2020

Última modificación: Tue Aug 27 14:59:00 2019 -0300

Facultad de Ingeniería (UBA)

# Créditos

Para armar las presentaciones del curso nos basamos en:



R. E. Bryant and D. R. O'Hallaron, *Computer systems: a programmer's perspective*, Third edition, Global edition. Boston Columbus Hoboken Indianapolis New York San Francisco Cape Town: Pearson, 2016.



D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*, RISC-V edition. Cambridge, Massachusetts: Morgan Kaufmann Publishers, an imprint of Elsevier, 2018.



J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. 2019.

# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Tabla de contenidos

---

## 1. Representando la información como bits

### Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Representaciones numéricas

## Sistemas de numeración

- Posicionales

- Decimal:  $2953253037_{10} = 2953253037_{10}$

$$2953253037_{10} = 2 \cdot 10^9 + 9 \cdot 10^8 + 5 \cdot 10^7 + 3 \cdot 10^6 + 2 \cdot 10^5 + 5 \cdot 10^4 + 3 \cdot 10^3 + 3 \cdot 10^1 + 7 \cdot 10^0$$

# Representaciones numéricas

## Sistemas de numeración

- Posicionales

- Decimal:  $2953253037_{10} = 2953253037_{10}$
- Hexadecimal alias *hexa*:  $2953253037_{10} = \text{b}00710\text{ad}_{16}$

$$\text{b}00710\text{ad}_{16} = b \cdot 16^7 + 7 \cdot 16^4 + 1 \cdot 16^3 + a \cdot 16^1 + d \cdot 16^0$$

# Representaciones numéricas

## Sistemas de numeración

### ■ Posicionales

- Decimal:  $2953253037_{10} = 2953253037_{10}$
- Hexadecimal alias *hexa*:  $2953253037_{10} = \text{b}00710\text{ad}_{16}$
- Sexagesimal:  $2953253037_{10} = \text{𐎶 𐎵 𐎴 𐎳 𐎲 𐎱 𐎰 𐎯}_{60}$

$$\text{𐎶 𐎵 𐎴 𐎳 𐎲 𐎱 𐎰 𐎯}_{60} = \text{𐎶} \cdot 60^5 + \text{𐎵} \cdot 60^4 + \text{𐎴} \cdot 60^3 + \text{𐎳} \cdot 60^2 + \text{𐎲} \cdot 60^1 + \text{𐎱} \cdot 60^0$$



# Representaciones numéricas

## Sistemas de numeración

### ■ Posicionales

- Decimal:  $2953253037_{10} = 2953253037_{10}$
- Hexadecimal alias *hexa*:  $2953253037_{10} = \text{b}00710\text{ad}_{16}$
- Sexagesimal:  $2953253037_{10} = \text{𐎶 𐎵 𐎴 𐎳 𐎲 𐎱 𐎰 𐎯 𐎮 𐎭}_{60}$
- Binario:  $2953253037_{10} = 10110000000001110001000010101101_2$

$$10110000000001110001000010101101_{10} = 1 \cdot 2^{31} + 0 \cdot 2^{30} + 1 \cdot 2^{29} + 1 \cdot 2^{28} + 0 \cdot 2^{27}$$

# Representaciones numéricas

## Sistemas de numeración

### ■ Posicionales

- Decimal:  $2953253037_{10} = 2953253037_{10}$
- Hexadecimal alias *hexa*:  $2953253037_{10} = \text{b}00710\text{ad}_{16}$
- Sexagesimal:  $2953253037_{10} = \text{𐎶 𐎵 𐎴 𐎳 𐎲 𐎱 𐎰 𐎯 𐎮 𐎭}_{60}$
- Binario:  $2953253037_{10} = 10110000000001110001000010101101_2$

$$N_b = \sum_{i=-\infty}^{\infty} c_i \cdot b^i$$

# Representaciones numéricas

## Sistemas de numeración

### ■ Posicionales

- Decimal:  $2953253037_{10} = 2953253037_{10}$
- Hexadecimal alias *hexa*:  $2953253037_{10} = \text{b}00710\text{ad}_{16}$
- Sexagesimal:  $2953253037_{10} = \text{𐎶 𐎵 𐎴 𐎳 𐎲 𐎱 𐎰 𐎯 𐎮 𐎭}_{60}$
- Binario:  $2953253037_{10} = 10110000000001110001000010101101_2$

$$N_b = \sum_{i=-\infty}^{\infty} c_i \cdot b^i$$

### ■ No posicionales

- Romano:  $2019_{10} = \text{MMXIX}$

# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

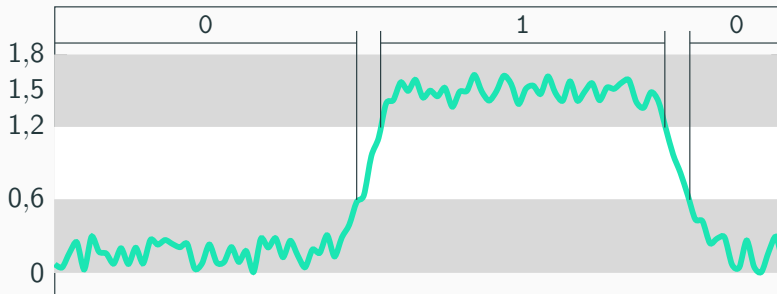
## 5. Code Security

# Todo está compuesto por bits

- Cada bit es 0 o es 1
- Al *encodear*/interpretar los bits de distintas maneras
  - las computadoras determinan qué hacer (instrucciones)
  - representan y manipulan numeros, caracteres, cadenas, etc.
- ¿por qué bits? Por la electrónica subyacente
  - fáciles de almacenar en elementos biestables
  - se transmiten de manera confiable

# Todo está compuesto por bits

- Cada bit es 0 o es 1
- Al *encodear*/interpretar los bits de distintas maneras
  - las computadoras determinan qué hacer (instrucciones)
  - representan y manipulan numeros, caracteres, cadenas, etc.
- ¿por qué bits? Por la electrónica subyacente
  - fáciles de almacenar en elementos biestables
  - se transmiten de manera confiable



# Ejemplos: representaciones en binario

## Representación de números en Base 2

- El número  $13548250_{10}$  se representa como  $110011101011101011011010_2$

# Ejemplos: representaciones en binario

## Representación de números en Base 2

- El número  $13548250_{10}$  se representa como  $110011101011101011011010_2$
- El número  $1,20_{10}$  se representa como  $1.001100110011[0011]..._2$

$$1,20_{10} = 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + 1 \cdot 2^{-7} + 1 \cdot 2^{-8} + \dots$$



# Ejemplos: representaciones en binario

## Representación de números en Base 2

- El número  $13548250_{10}$  se representa como  $110011101011101011011010_2$

- El número  $1,20_{10}$  se representa como  $1.001100110011[0011]..._2$

$$1,20_{10} = 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + 1 \cdot 2^{-7} + 1 \cdot 2^{-8} + \dots$$

- El número  $1,3548250 \times 10^7$  se representa como  $1.10011101011101011011010_2 \times 2^{23}$

# Encoding Bytes

## 1 Byte = 8 bits

- Binario:  $00000000_2$  a  $11111111_2$
- Decimal:  $0_{10}$  a  $255_{10}$
- Hexa:  $00_{16}$  a  $FF_{16}$ 
  - del 0 al 9 ¿y después? ¿113 es 1-13 u 11-3?

# Encoding Bytes

## 1 Byte = 8 bits

- Binario:  $00000000_2$  a  $11111111_2$
- Decimal:  $0_{10}$  a  $255_{10}$
- Hexa:  $00_{16}$  a  $FF_{16}$ 
  - del 0 al 9 ¿y después? ¿113 es 1-13 u 11-3?

| hexa | decimal | binario |
|------|---------|---------|
| 0    | 0       | 0000    |
| 1    | 1       | 0001    |
| 2    | 2       | 0010    |
| 3    | 3       | 0011    |
| 4    | 4       | 0100    |
| 5    | 5       | 0101    |
| 6    | 6       | 0110    |
| 7    | 7       | 0111    |
| 8    | 8       | 1000    |
| 9    | 9       | 1001    |
| A    | 10      | 1010    |
| B    | 11      | 1011    |
| C    | 12      | 1100    |
| D    | 13      | 1101    |
| E    | 14      | 1110    |
| F    | 15      | 1111    |

# Encoding Bytes

## 1 Byte = 8 bits

- Binario:  $00000000_2$  a  $11111111_2$
- Decimal:  $0_{10}$  a  $255_{10}$
- Hexa:  $00_{16}$  a  $FF_{16}$ 
  - del 0 al 9 ¿y después? ¿113 es 1-13 u 11-3?
  - Se utilizan los caracteres '0' a '9' y 'A' a 'F'

| hexa | decimal | binario |
|------|---------|---------|
| 0    | 0       | 0000    |
| 1    | 1       | 0001    |
| 2    | 2       | 0010    |
| 3    | 3       | 0011    |
| 4    | 4       | 0100    |
| 5    | 5       | 0101    |
| 6    | 6       | 0110    |
| 7    | 7       | 0111    |
| 8    | 8       | 1000    |
| 9    | 9       | 1001    |
| A    | 10      | 1010    |
| B    | 11      | 1011    |
| C    | 12      | 1100    |
| D    | 13      | 1101    |
| E    | 14      | 1110    |
| F    | 15      | 1111    |

# Encoding Bytes

## 1 Byte = 8 bits

- Binario:  $00000000_2$  a  $11111111_2$
- Decimal:  $0_{10}$  a  $255_{10}$
- Hexa:  $00_{16}$  a  $FF_{16}$ 
  - del 0 al 9 ¿y después? ¿113 es 1-13 u 11-3?
  - Se utilizan los caracteres '0' a '9' y 'A' a 'F'
- En C, un entero en hexa se escribe de la siguiente manera:
  - $b00710ad_{16} \rightarrow 0xb00710ad$
  - o bien:  $0xB00710AD$

| hexa | decimal | binario |
|------|---------|---------|
| 0    | 0       | 0000    |
| 1    | 1       | 0001    |
| 2    | 2       | 0010    |
| 3    | 3       | 0011    |
| 4    | 4       | 0100    |
| 5    | 5       | 0101    |
| 6    | 6       | 0110    |
| 7    | 7       | 0111    |
| 8    | 8       | 1000    |
| 9    | 9       | 1001    |
| A    | 10      | 1010    |
| B    | 11      | 1011    |
| C    | 12      | 1100    |
| D    | 13      | 1101    |
| E    | 14      | 1110    |
| F    | 15      | 1111    |

# Encoding Bytes

## 1 Byte = 8 bits

- Binario:  $00000000_2$  a  $11111111_2$
- Decimal:  $0_{10}$  a  $255_{10}$
- Hexa:  $00_{16}$  a  $FF_{16}$ 
  - del 0 al 9 ¿y después? ¿113 es 1-13 u 11-3?
  - Se utilizan los caracteres '0' a '9' y 'A' a 'F'
- En C, un entero en hexa se escribe de la siguiente manera:
  - $b00710ad_{16} \rightarrow 0xb00710ad$
  - o bien:  $0xB00710AD$

¿para qué me sirve?

| hexa | decimal | binario |
|------|---------|---------|
| 0    | 0       | 0000    |
| 1    | 1       | 0001    |
| 2    | 2       | 0010    |
| 3    | 3       | 0011    |
| 4    | 4       | 0100    |
| 5    | 5       | 0101    |
| 6    | 6       | 0110    |
| 7    | 7       | 0111    |
| 8    | 8       | 1000    |
| 9    | 9       | 1001    |
| A    | 10      | 1010    |
| B    | 11      | 1011    |
| C    | 12      | 1100    |
| D    | 13      | 1101    |
| E    | 14      | 1110    |
| F    | 15      | 1111    |

# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Organización de la memoria: de a bytes

- Los programas acceden a los *datos* usando *direcciones de memoria*
  - conceptualmente, la ven como un arreglo de **bytes**
    - no lo es, pero sirve pensarlo así

00...0 

|  |  |  |  |  |     |  |  |  |  |
|--|--|--|--|--|-----|--|--|--|--|
|  |  |  |  |  | ... |  |  |  |  |
|--|--|--|--|--|-----|--|--|--|--|

 FF...F



# Organización de la memoria: de a bytes

- Los programas acceden a los *datos* usando *direcciones de memoria*

- conceptualmente, la ven como un arreglo de **bytes**

- no lo es, pero sirve pensarlo así

00...0 

|  |  |  |  |  |     |  |  |  |  |
|--|--|--|--|--|-----|--|--|--|--|
|  |  |  |  |  | ... |  |  |  |  |
|--|--|--|--|--|-----|--|--|--|--|

 FF...F

- una *dirección de memoria* es como un índice en ese arreglo

# Organización de la memoria: de a bytes

- Los programas acceden a los *datos* usando *direcciones de memoria*

- conceptualmente, la ven como un arreglo de **bytes**

- no lo es, pero sirve pensarlo así



- una *dirección de memoria* es como un índice en ese arreglo

- un **puntero** es una variable que guarda *direcciones de memoria*

# Organización de la memoria: de a bytes

- Los programas acceden a los *datos* usando *direcciones de memoria*

- conceptualmente, la ven como un arreglo de **bytes**

- no lo es, pero sirve pensarlo así



- una *dirección de memoria* es como un índice en ese arreglo

- un **puntero** es una variable que guarda *direcciones de memoria*  
`p = 0xFF...D;`

# Organización de la memoria: de a bytes

- Los programas acceden a los *datos* usando *direcciones de memoria*

- conceptualmente, la ven como un arreglo de **bytes**

- no lo es, pero sirve pensarlo así



- una *dirección de memoria* es como un índice en ese arreglo

- un **puntero** es una variable que guarda *direcciones de memoria*

p = 0xFF...D; 

# Organización de la memoria: de a bytes

## ■ Los programas acceden a los *datos* usando *direcciones de memoria*

- conceptualmente, la ven como un arreglo de **bytes**

- no lo es, pero sirve pensarlo así



- una *dirección de memoria* es como un índice en ese arreglo

- un **puntero** es una variable que guarda *direcciones de memoria*

`p = 0xFF...D;`

## ■ Cada *proceso* tiene su espacio de direcciones privado

- piensen un *proceso* como un programa en ejecución
- un programa puede modificar sus datos, pero no los de otro
  - no siempre fue así :-S

# Tamaño de palabras

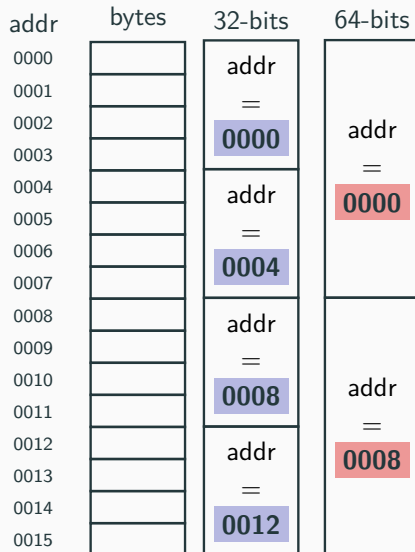
## Todo sistema tiene un tamaño de palabra (*word size*)

- Tamaño nominal de los datos enteros
  - incluyendo las direcciones de memoria
- Máquinas de 32 bits:
  - *word size*: 4 bytes
  - *address space*: limitado a 4GB ( $2^{32}$  bytes)
- Máquinas de 64 bits:
  - *word size*: 8 bytes
  - *address space*: "limitado" a 18+ EB (exabyte) ( $2^{64}$  bytes)
  - 4294967296  
18446744073709552000
- ¿Y los datos de otros tamaños?

# Organización de la memoria: de a palabras

## Las direcciones

- indican las posiciones de **bytes**
- en particular, del primer byte de una palabra
- palabras sucesivas difieren en 4 u 8



# Tamaños de datos

| Tipo (C)          | 32-bit | 64-bit | x86-64 |
|-------------------|--------|--------|--------|
| char              | 1      | 1      | 1      |
| short             | 2      | 2      | 2      |
| int               | 4      | 4      | 4      |
| long              | 4      | 8      | 8      |
| float             | 4      | 4      | 4      |
| double            | 8      | 8      | 8      |
| long double       | –      | –      | 10/16  |
| int *             | 4      | 8      | 8      |
| char *            | 4      | 8      | 8      |
| cualquier puntero | 4      | 8      | 8      |



# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

**Lilliput & Blefuscu**

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

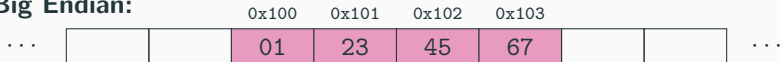
# Byte ordering

¿Cómo se pueden ordenar los bytes de datos multi-byte en memoria?

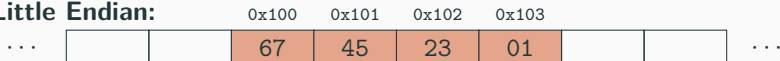
## Supongamos

- La variable `kiwi` almacena el valor de 4-bytes: `0x01234567`
- La posición de memoria dada por `&kiwi` es: `0x100`
- El byte más significativo es `0x01`, el menos significativo es `0x67`

### Big Endian:



### Little Endian:



# Byte ordering

---

**¿Cómo se ordenan los bytes de datos multi-byte en memoria?**

**¡Por convención!**

# Byte ordering

¿Cómo se ordenan los bytes de datos multi-byte en memoria?

¡Por convención!

## Convenciones

- *Big Endian*: las viejas Sun SPARC y iMac PowerPC (hoy son bi-).
  - El byte **menos** significativo tiene la dirección más **alta**.

# Byte ordering

¿Cómo se ordenan los bytes de datos multi-byte en memoria?

¡Por convención!

## Convenciones

- *Big Endian*: las viejas Sun SPARC y iMac PowerPC (hoy son bi-).
  - El byte **menos** significativo tiene la dirección más **alta**.
- *Little Endian*: x86, ADM64/x86-64, los ARM que corren Android, iOS y Windows.
  - El byte **menos** significativo tiene la dirección más **baja**.

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones

```

1 int x = foo();
2 int y = bar();
3 unsigned ux = x;
4 unsigned uy = y;

```

$$\blacksquare x < 0 \quad \Rightarrow \quad ((x*2) < 0)$$

$$\blacksquare ux \geq 0$$

$$\blacksquare x \& 7 == 7 \quad \Rightarrow \quad (x \ll 30) < 0$$

$$\blacksquare ux > -1$$

$$\blacksquare x > y \quad \Rightarrow \quad -x < -y$$

$$\blacksquare x * x \geq 0$$

$$\blacksquare x > 0 \ \&\& \ y > 0 \quad \Rightarrow \quad x + y > 0$$

$$\blacksquare x \geq 0 \quad \Rightarrow \quad -x \leq 0$$

$$\blacksquare x \leq 0 \quad \Rightarrow \quad -x \geq 0$$

# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Álgebra de Boole

## Desarrollada por George Boole en el siglo 19

- Representación algebraica de la lógica
- Codifica los valores de verdad “Verdadero” como 1 y “Falso” como 0

### AND

$A \& B = 1$  cuando  $A = 1$  y  $B = 1$

| $\&$ | 0 | 1 |
|------|---|---|
| 0    | 0 | 0 |
| 1    | 0 | 1 |

### NOT

$\sim A = 1$  cuando  $A = 0$

| $\sim$ |   |
|--------|---|
| 0      | 1 |
| 1      | 0 |

### OR

$A \mid B = 1$  cuando  $A = 1$  o  $B = 1$

| $\mid$ | 0 | 1 |
|--------|---|---|
| 0      | 0 | 1 |
| 1      | 1 | 1 |

### EXclusive-OR (XOR)

$A \wedge B = 1$  cuando  $A = 1$  o  $B = 1$ , pero no ambos

| $\wedge$ | 0 | 1 |
|----------|---|---|
| 0        | 0 | 1 |
| 1        | 1 | 0 |



# Álgebras de Boole en general

## Operan con vectores de bits

Aplican bit a bit

|            |          |            |            |
|------------|----------|------------|------------|
| 01101001   | 01101001 | 01101001   |            |
| & 01010101 | 01010101 | ^ 01010101 | ~ 01010101 |
| -----      | -----    | -----      | -----      |
| 01000001   | 01111101 | 00111100   | 10101010   |

## Aplican todas las propiedades del Álgebra Booleana

- $\langle \{0, 1\}, |, \&, , 0, 1 \rangle$  forman un álgebra
- OR es la operación “suma”
- AND es la operación “producto”
- NOT es la operación “complemento”
- 0 es la identidad para la “suma”
- 1 es la identidad para el “producto”

# Operaciones con bits en C

## Operaciones & (AND), | (OR), ~ (NOT), ^ (XOR)

- Aplican a cualquier dato “entero”
  - `long`, `int`, `short`, `char`, `unsigned`
- Vectores de bits

## Ejemplos

- |  |  |
|--|--|
| ■ $\sim 0x41 \Rightarrow 0xBE$ <ul style="list-style-type: none"><li>■ <math>\sim 01000001_2 \Rightarrow 10111110_2</math></li></ul> | ■ $0x69 \ \& \ 0x55 \Rightarrow 0x41$ <ul style="list-style-type: none"><li>■ <math>01101001_2 \&amp; 01010101_2 \Rightarrow 01000001_2</math></li></ul> |
| ■ $\sim 0x00 \Rightarrow 0xFF$ <ul style="list-style-type: none"><li>■ <math>\sim 00000000_2 \Rightarrow 11111111_2</math></li></ul> | ■ $0x69 \   \ 0x55 \Rightarrow 0x7D$ <ul style="list-style-type: none"><li>■ <math>01101001_2 \   \ 01010101_2 \Rightarrow 01111101_2</math></li></ul>   |

# Contraste: operaciones lógicas en C

## Operaciones `&&` (AND), `||` (OR), `!` (NOT)

- 0 es “Falso”
- Cualquier cosa distintas de cero es “Verdadera”
- Retornan 0 ó 1
- Evaluación mínima: ¡short-circuits!

## Ejemplos

- |   |   |
|---|---|
| ■ <code>!0x41</code> $\Rightarrow$ <code>0x00</code>  | ■ <code>0x69 &amp;&amp; 0x55</code> $\Rightarrow$ <code>0x01</code> |
| ■ <code>!0x00</code> $\Rightarrow$ <code>0x01</code>  | ■ <code>0x69    0x55</code> $\Rightarrow$ <code>0x01</code>         |
| ■ <code>!!0x41</code> $\Rightarrow$ <code>0x01</code> | ■ <code>p &amp;&amp; *p</code> (evita el acceso a punteros nulos)   |

# Desplazamientos: shift

## Left Shift: $u \ll k$

- Desplaza los bits de  $u$  a la izquierda  $k$  posiciones
  - Descarta los  $k$  bits de la izquierda
- Completa con ceros a la derecha

|               |          |
|---------------|----------|
| Argumento $u$ | 01100010 |
| $\ll 3$       | 00010000 |
| Log. $\gg 2$  | 00011000 |
| Arit. $\gg 2$ | 00011000 |

## Right Shift: $u \gg k$

- Desplaza los bits de  $u$  a la derecha
  - Descarta los  $k$  bits de la derecha
- Desplazamiento lógico
  - Completa con ceros a izquierda
- Desplazamiento aritmético
  - Replica el msb  $k$  veces a izquierda

|               |          |
|---------------|----------|
| Argumento $u$ | 10100010 |
| $\ll 3$       | 00010000 |
| Log. $\gg 2$  | 00101000 |
| Arit. $\gg 2$ | 11101000 |

Si  $k < 0$  ó  $k \geq \text{al } word\text{-size} \Rightarrow$  **Undefined Behaviour**

# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Codificación de enteros

## No signado

$$B2U_w(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

short: 2 bytes

```
1 short int x = 16162;
2 short int y = -16162;
```

## Complemento a 2

$$B2T_w(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

|   | Decimal | Hexa  | Binario  |          |
|---|---------|-------|----------|----------|
| x | 16162   | 3F 22 | 00111111 | 00100010 |
| y | -16162  | C0 DE | 11000000 | 11011110 |

## Bit de signo

En complemento a dos, el bit más significativo (MSB) indica el signo

- 0 para positivo
- 1 para negativo

# Codificación de enteros

## No signado

$$B2U_w(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

short: 2 bytes

```
1 short int x = 16162;
2 short int y = -16162;
```

## Complemento a 2

$$B2T_w(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

|   | Decimal | Hexa  | Binario  |          |
|---|---------|-------|----------|----------|
| x | 16162   | 3F 22 | 00111111 | 00100010 |
| y | -16162  | C0 DE | 11000000 | 11011110 |

## Bit de signo

En complemento a dos, el bit más significativo (MSB) indica el signo

- 0 para positivo
- 1 para negativo



# Codificación de enteros

## No signado

$$B2U_w(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

short: 2 bytes

```
1 short int x = 16162;
2 short int y = -16162;
```

## Complemento a 2

$$B2T_w(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

|   | Decimal | Hexa  | Binario           |
|---|---------|-------|-------------------|
| x | 16162   | 3F 22 | 00111111 00100010 |
| y | -16162  | C0 DE | 11000000 11011110 |

## Bit de signo

En complemento a dos, el bit más significativo (MSB) indica el signo

- 0 para positivo
- 1 para negativo

# Codificación de enteros

## No signado

$$B2U_w(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

short: 2 bytes

```
1 short int x = 16162;
2 short int y = -16162;
```

## Complemento a 2

$$B2T_w(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

|   | Decimal | Hexa  | Binario           |
|---|---------|-------|-------------------|
| x | 16162   | 3F 22 | 00111111 00100010 |
| y | -16162  | C0 DE | 11000000 11011110 |

## Bit de signo

En complemento a dos, el bit más significativo (MSB) indica el signo

- 0 para positivo
- 1 para negativo

## Ejemplos $B2U_w(X)$ y $B2T_w(X)$

---

$x = 16162 : 00111111 \ 00100010$

$y = -16162 : 11000000 \ 11011110$

---

| Peso ( $2^i$ ) | 16162 |      | -16162 |        |
|----------------|-------|------|--------|--------|
| 1              | 0     | 0    | 0      | 0      |
| 2              | 1     | 2    | 1      | 2      |
| 4              | 0     | 0    | 1      | 4      |
| 8              | 0     | 0    | 1      | 8      |
| 16             | 0     | 0    | 1      | 16     |
| 32             | 1     | 32   | 0      | 0      |
| 64             | 0     | 0    | 1      | 64     |
| 128            | 0     | 0    | 1      | 128    |
| 256            | 1     | 256  | 0      | 0      |
| 512            | 1     | 512  | 0      | 0      |
| 1024           | 1     | 1024 | 0      | 0      |
| 2048           | 1     | 2048 | 0      | 0      |
| 4096           | 1     | 4096 | 0      | 0      |
| 8192           | 1     | 8192 | 0      | 0      |
| 16384          | 0     | 0    | 1      | 16384  |
| -32768         | 0     | 0    | 1      | -32768 |
| Suma:          | 16162 |      | -16162 |        |

# Rangos numéricos

## Límites: no signados

$$UMin = 0 \quad (000 \dots 0)$$

$$UMax = 2^w - 1 \quad (111 \dots 1)$$

## Límites: complemento a dos

$$TMin = -2^{w-1} \quad (100 \dots 0)$$

$$TMax = 2^{w-1} - 1 \quad (011 \dots 1)$$

## Valores para $w = 16$

|      | Decimal | Hexa  | Binario           |
|------|---------|-------|-------------------|
| UMax | 65535   | FF FF | 11111111 11111111 |
| TMax | 32767   | 7F FF | 01111111 11111111 |
| TMin | -32768  | 80 00 | 10000000 00000000 |
| -1   | -1      | FF FF | 11111111 11111111 |
| 0    | 0       | 00 00 | 00000000 00000000 |

## Valores para distintos tamaño de palabra

|      | tamaño de palabra ( $w$ ) |        |             |                            |
|------|---------------------------|--------|-------------|----------------------------|
|      | 8                         | 16     | 32          | 64                         |
| UMax | 255                       | 65535  | 4294967296  | 18 446 744 073 709 551 615 |
| TMax | 127                       | 32767  | 2147483647  | 9 223 372 036 854 775 807  |
| TMin | -128                      | -32768 | -2147483648 | -9 223 372 036 854 775 808 |

### Observaciones

- $|TMin| = TMax + 1$ 
  - Rango asimétrico
- $UMax = 2 \cdot TMax + 1$

### Programación en C

- `#include <limits.h>`
- Declara constantes
  - `ULONG_MAX`,
  - `LONG_MAX`,
  - `LONG_MIN`,
  - `USHRT_MAX`, etc.
- Los valores son dependientes de la plataforma

# Relación entre signado y no signado

| X    | $B2U(X)$ | $B2T(X)$ |
|------|----------|----------|
| 0000 | 0        | 0        |
| 0001 | 1        | 1        |
| 0010 | 2        | 2        |
| 0011 | 3        | 3        |
| 0100 | 4        | 4        |
| 0101 | 5        | 5        |
| 0110 | 6        | 6        |
| 0111 | 7        | 7        |
| 1000 | 8        | -8       |
| 1001 | 9        | -7       |
| 1010 | 10       | -6       |
| 1011 | 11       | -5       |
| 1100 | 12       | -4       |
| 1101 | 13       | -3       |
| 1110 | 14       | -2       |
| 1111 | 15       | -1       |

## ■ Equivalencia

- Mismo encoding para valores no negativos

## ■ Unicidad

- Cada patrón representa un único entero
- A cada entero representable le corresponde un único patrón

## ■ $\Rightarrow$ se pueden invertir los mapeos

- $U2B(X) = B2U^{-1}(X)$ 
  - Patrón de bits para un entero sin signo
- $T2B(X) = B2T^{-1}(X)$ 
  - Patrón de bits para un entero en complemento a dos

# Conversión de signado a no signado

## C permite convertir enteros signados a no signados

```
1 short int x = 16162;
2 unsigned short int ux = (unsigned short) x;
3 short int y = -16162;
4 unsigned short int uy = (unsigned short) y;
```

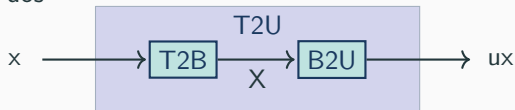
## Resultado

- **NO** cambia la representación *binaria*
- Los valores no negativos se mantienen igual
  - `ux = 16162`
- Los valores negativos cambian a valores (muy) grandes
  - `uy = 49374`

## Conversión de signado a no signado

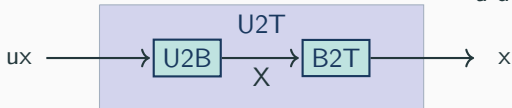
complemento  
a dos

no signado



no signado

complemento  
a dos



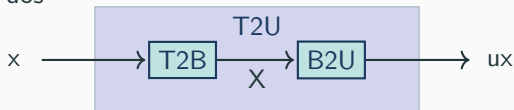
No cambia el patrón de bits: **mantener los bits y reinterpretar**



# Relación entre signado y no signado

complemento  
a dos

no signado



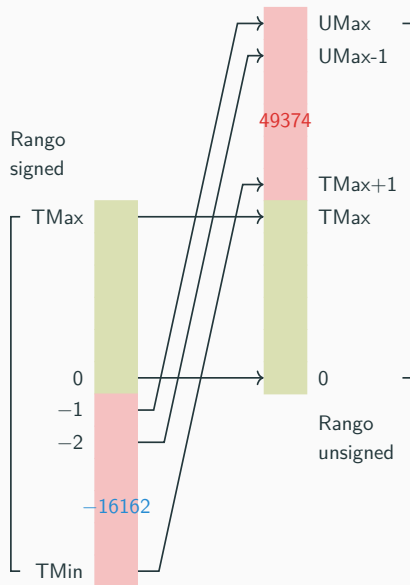
$$ux = \begin{cases} x & x \geq 0 \\ x + 2^w & x < 0 \end{cases}$$

$$+2^{w-1} - -2^{w-1} = 2 \cdot 2^{w-1} = 2^w$$

**Un peso negativo grande se convierte en un peso positivo grande**

# Relación entre signado y no signado

| Peso ( $2^i$ ) | -16162 |        | 49374 |       |
|----------------|--------|--------|-------|-------|
| 1              | 0      | 0      | 0     | 0     |
| 2              | 1      | 2      | 1     | 2     |
| 4              | 1      | 4      | 1     | 4     |
| 8              | 1      | 8      | 1     | 8     |
| 16             | 1      | 16     | 1     | 16    |
| 32             | 0      | 0      | 0     | 0     |
| 64             | 1      | 64     | 1     | 64    |
| 128            | 1      | 128    | 1     | 128   |
| 256            | 0      | 0      | 0     | 0     |
| 512            | 0      | 0      | 0     | 0     |
| 1024           | 0      | 0      | 0     | 0     |
| 2048           | 0      | 0      | 0     | 0     |
| 4096           | 0      | 0      | 0     | 0     |
| 8192           | 0      | 0      | 0     | 0     |
| 16384          | 1      | 16384  | 1     | 16384 |
| $\pm 32768$    | 1      | -32768 | 1     | 32768 |
| Suma:          | -16162 |        | 49374 |       |



# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# En C, ¿cuándo es cuál?

## ■ Constantes

- por omisión, son signadas
- U es el sufijo unsigned  
0U, 4294967259U

## ■ Casting

- Explícito

```
1 int tx, ty;  
2 unsigned ux, uy;  
3 tx = (int) ux;  
4 uy = (unsigned) ty;
```

- Implícito: asignaciones, evaluaciones, llamadas

```
1 tx = ux;  
2 uy = ty;
```

# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  
 $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       |            |
| -1                     | 0                      | <        |            |
| -1                     | 0U                     | >        |            |
| 2147483647             | -2147483647-1          | >        |            |
| 2147483647U            | -2147483647-1          | <        |            |
| -1                     | -2                     | >        |            |
| (unsigned)-1           | -2                     | >        |            |
| 2147483647             | 2147483648U            | <        |            |
| 2147483647             | (int)2147483648U       | >        |            |

# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  
 $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       | unsigned   |
| -1                     | 0                      | <        |            |
| -1                     | 0U                     | >        |            |
| 2147483647             | -2147483647-1          | >        |            |
| 2147483647U            | -2147483647-1          | <        |            |
| -1                     | -2                     | >        |            |
| (unsigned)-1           | -2                     | >        |            |
| 2147483647             | 2147483648U            | <        |            |
| 2147483647             | (int)2147483648U       | >        |            |

# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  
 $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       | unsigned   |
| -1                     | 0                      | <        | signed     |
| -1                     | 0U                     | >        |            |
| 2147483647             | -2147483647-1          | >        |            |
| 2147483647U            | -2147483647-1          | <        |            |
| -1                     | -2                     | >        |            |
| (unsigned)-1           | -2                     | >        |            |
| 2147483647             | 2147483648U            | <        |            |
| 2147483647             | (int)2147483648U       | >        |            |

# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  
 $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       | unsigned   |
| -1                     | 0                      | <        | signed     |
| -1                     | 0U                     | >        | unsigned   |
| 2147483647             | -2147483647-1          | >        |            |
| 2147483647U            | -2147483647-1          | <        |            |
| -1                     | -2                     | >        |            |
| (unsigned)-1           | -2                     | >        |            |
| 2147483647             | 2147483648U            | <        |            |
| 2147483647             | (int)2147483648U       | >        |            |



# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  
 $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       | unsigned   |
| -1                     | 0                      | <        | signed     |
| -1                     | 0U                     | >        | unsigned   |
| 2147483647             | -2147483647-1          | >        | signed     |
| 2147483647U            | -2147483647-1          | <        |            |
| -1                     | -2                     | >        |            |
| (unsigned)-1           | -2                     | >        |            |
| 2147483647             | 2147483648U            | <        |            |
| 2147483647             | (int)2147483648U       | >        |            |

# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  
 $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       | unsigned   |
| -1                     | 0                      | <        | signed     |
| -1                     | 0U                     | >        | unsigned   |
| 2147483647             | -2147483647-1          | >        | signed     |
| 2147483647U            | -2147483647-1          | <        | unsigned   |
| -1                     | -2                     | >        |            |
| (unsigned)-1           | -2                     | >        |            |
| 2147483647             | 2147483648U            | <        |            |
| 2147483647             | (int)2147483648U       | >        |            |

# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  
 $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       | unsigned   |
| -1                     | 0                      | <        | signed     |
| -1                     | 0U                     | >        | unsigned   |
| 2147483647             | -2147483647-1          | >        | signed     |
| 2147483647U            | -2147483647-1          | <        | unsigned   |
| -1                     | -2                     | >        | signed     |
| (unsigned)-1           | -2                     | >        |            |
| 2147483647             | 2147483648U            | <        |            |
| 2147483647             | (int)2147483648U       | >        |            |

# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  
 $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       | unsigned   |
| -1                     | 0                      | <        | signed     |
| -1                     | 0U                     | >        | unsigned   |
| 2147483647             | -2147483647-1          | >        | signed     |
| 2147483647U            | -2147483647-1          | <        | unsigned   |
| -1                     | -2                     | >        | signed     |
| (unsigned)-1           | -2                     | >        | unsigned   |
| 2147483647             | 2147483648U            | <        |            |
| 2147483647             | (int)2147483648U       | >        |            |

# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       | unsigned   |
| -1                     | 0                      | <        | signed     |
| -1                     | 0U                     | >        | unsigned   |
| 2147483647             | -2147483647-1          | >        | signed     |
| 2147483647U            | -2147483647-1          | <        | unsigned   |
| -1                     | -2                     | >        | signed     |
| (unsigned)-1           | -2                     | >        | unsigned   |
| 2147483647             | 2147483648U            | <        | unsigned   |
| 2147483647             | (int)2147483648U       | >        |            |

# En C, ¿cuándo es cuál?

## ■ Evaluación de expresiones

- si hay mezcla de signados y no signados en una expresión **los valores con signo se castean implícitamente a unsigned**
- Ejemplos para  $W = 32$ :  **$TMIN = -2147483648$ ,  $TMAX = 2147483647$**

| Constante <sub>1</sub> | Constante <sub>2</sub> | Relación | Evaluación |
|------------------------|------------------------|----------|------------|
| 0                      | 0U                     | ==       | unsigned   |
| -1                     | 0                      | <        | signed     |
| -1                     | 0U                     | >        | unsigned   |
| 2147483647             | -2147483647-1          | >        | signed     |
| 2147483647U            | -2147483647-1          | <        | unsigned   |
| -1                     | -2                     | >        | signed     |
| (unsigned)-1           | -2                     | >        | unsigned   |
| 2147483647             | 2147483648U            | <        | unsigned   |
| 2147483647             | (int)2147483648U       | >        | signed     |

# Extensión de signo

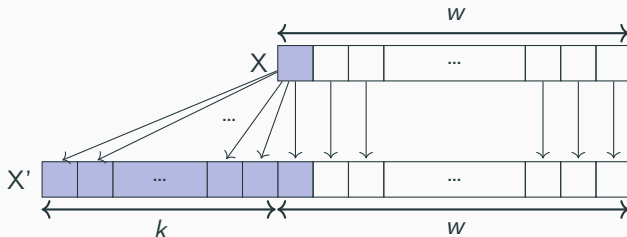
## ■ Queremos:

- Dado un entero *signado* de  $w$ -bits
- Convertirlo a un entero de  $w + k$ -bits con el mismo valor

## ■ Regla:

- Propagar el bit de signo a los nuevos  $k$  bits (MSB)

$$X' = \underbrace{x_{w-1}, \dots, x_{w-1}}_{k \text{ copias del MSB}}, \underbrace{x_{w-1}, x_{w-2}, \dots, x_1, x_0}_{\text{entero original}}$$



## Extensión de signo: ejemplo

```

1 short int x = 16162;
2 int      ix = (int) x;
3 short int y = -16162;
4 int      iy = (int) y;

```

|    | Decimal | Hex         | Binario                             |
|----|---------|-------------|-------------------------------------|
| x  | 16162   | 3F 22       | 00111111 00100010                   |
| ix | 16162   | 00 00 3F 22 | 00000000 00000000 00111111 00100010 |
| y  | -16162  | C0 DE       | 11000000 11011110                   |
| iy | -16162  | FF FF C0 DE | 11111111 11111111 11000000 11011110 |

Se aumenta la cantidad de bits en la representación: expansión ó *up casting*.



# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Suma de enteros no signados

Operandos de  $w$  bits



Suma verdadera  
de  $w + 1$  bits



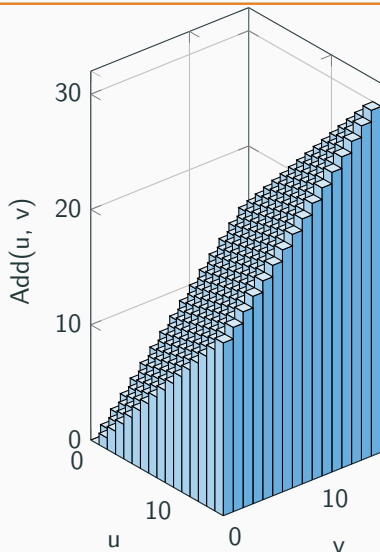
Descarta *carry*:  $w$  bits

$\text{UAdd}_w(u, v)$

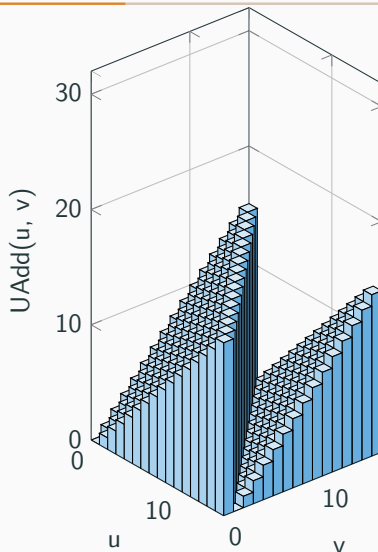


$$\text{UAdd}_w(u, v) = u + v \mod 2^w$$

## Suma de enteros no signados

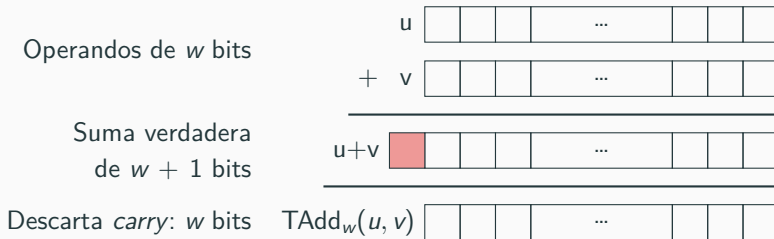


Suma teórica de enteros



Suma real de enteros

# Suma de enteros en complemento a dos



## ■ A nivel binario: comportamiento idéntico a la suma unsigned

```

1 int s, t, u, v;
2 s = (int) ((unsigned) u + (unsigned) v);
3 t = u + v;

```

■ da `t == s`

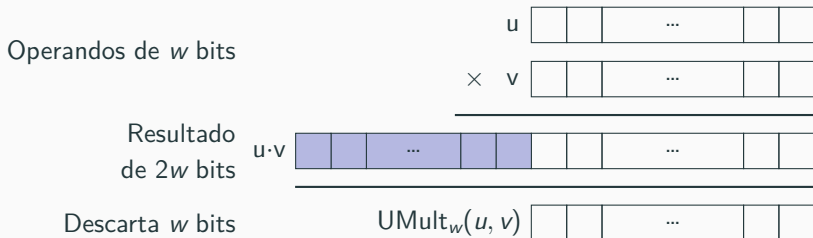
# Suma de enteros signados

**Ejemplo con  $w = 4 \Rightarrow$**

- Requiere  $w + 1$  bits
- Descarta el MSB
- El resultado es en complemento a dos
- *wraps around*
  - Si la suma  $\geq 2^{w-1}$ 
    - Se vuelve negativa
    - No más de una vez
  - Si la suma  $< -2^{w-1}$ 
    - Se vuelve positiva
    - No más de una vez

| u    |     | v    |     | suma  |     | TAdd <sub>4</sub> (u,v) |       |     |
|------|-----|------|-----|-------|-----|-------------------------|-------|-----|
| bin  | dec | bin  | dec | bin   | dec | msb                     | 4-bit | dec |
| 0111 | 7   | 0111 | 7   | 01110 | 14  | 0                       | 1110  | -2  |
| 0111 | 7   | 0110 | 6   | 01101 | 13  | 0                       | 1101  | -3  |
| 0111 | 7   | 0101 | 5   | 01100 | 12  | 0                       | 1100  | -4  |
| 0110 | 6   | 0100 | 4   | 01010 | 10  | 0                       | 1010  | -6  |
| 0011 | 3   | 0110 | 6   | 01001 | 9   | 0                       | 1001  | -7  |
| 0110 | 6   | 0010 | 2   | 01000 | 8   | 0                       | 1000  | -8  |
| 0100 | 4   | 0011 | 3   | 00111 | 7   | 0                       | 0111  | 7   |
| 0000 | 0   | 0110 | 6   | 00110 | 6   | 0                       | 0110  | 6   |
| 0101 | 5   | 0000 | 0   | 00101 | 5   | 0                       | 0101  | 5   |
| 1101 | -3  | 0110 | 6   | 00011 | 3   | 0                       | 0011  | 3   |
| 1111 | -1  | 0010 | 2   | 00001 | 1   | 0                       | 0001  | 1   |
| 1001 | -7  | 0111 | 7   | 00000 | 0   | 0                       | 0000  | 0   |
| 0011 | 3   | 1100 | -4  | 11111 | -1  | 1                       | 1111  | -1  |
| 1000 | -8  | 0101 | 5   | 11101 | -3  | 1                       | 1101  | -3  |
| 0100 | 4   | 1000 | -8  | 11100 | -4  | 1                       | 1100  | -4  |
| 1001 | -7  | 0010 | 2   | 11011 | -5  | 1                       | 1011  | -5  |
| 1000 | -8  | 0001 | 1   | 11001 | -7  | 1                       | 1001  | -7  |
| 1010 | -6  | 1110 | -2  | 11000 | -8  | 1                       | 1000  | -8  |
| 1000 | -8  | 1111 | -1  | 10111 | -9  | 1                       | 0111  | 7   |
| 1010 | -6  | 1100 | -4  | 10110 | -10 | 1                       | 0110  | 6   |
| 1001 | -7  | 1100 | -4  | 10101 | -11 | 1                       | 0101  | 5   |
| 1000 | -8  | 1100 | -4  | 10100 | -12 | 1                       | 0100  | 4   |
| 1000 | -8  | 1001 | -7  | 10001 | -15 | 1                       | 0001  | 1   |
| 1000 | -8  | 1000 | -8  | 10000 | -16 | 1                       | 0000  | 0   |

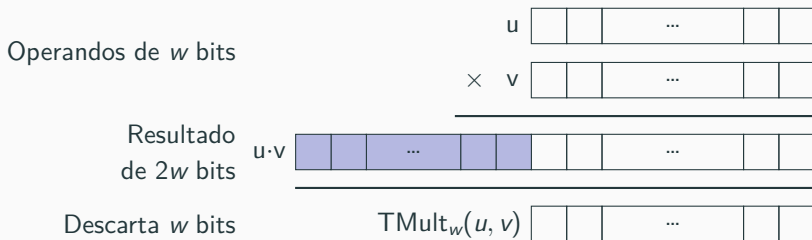
# Multiplicación de enteros



- **Multiplicación estándar**
  - ignora los  $w$  bits superiores
- **Implementa aritmética modular**
  - como ocurre con la suma:

$$\text{UMult}_w(u, v) = u \cdot v \mod 2^w$$

# Multiplicación de enteros



## ■ Multiplicación estándar

- ignora los  $w$  bits superiores
- los bits dentro de los  $w$  superiores pueden diferir
- los bits inferiores son iguales a  $\text{UMult}_w(u, v)$

# Multiplicar por potencias de 2

- Representación numérica

$$N_2 = \sum_{i=0}^{w-1} b_i \cdot 2^i$$

- Multiplicar por  $2^k$

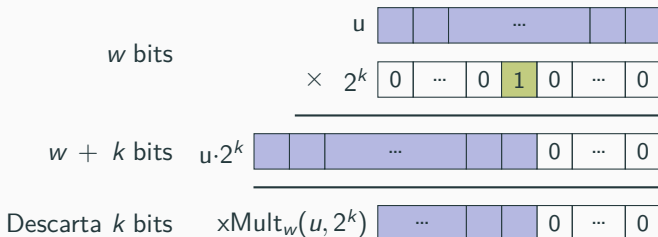
- requiere  $k$  bits más

$$\begin{aligned} N_2 \cdot 2^k &= \left( \sum_{i=0}^{w-1} b_i \cdot 2^i \right) \cdot 2^k \\ &= \sum_{i=0}^{w-1} b_i \cdot 2^{i+k} \\ &= \sum_{i=k}^{w+k-1} b_i \cdot 2^i + \sum_{j=0}^{k-1} 0 \cdot 2^j \end{aligned}$$

- ignora los  $k$  bits superiores



# Multiplicar por potencias de 2: shift



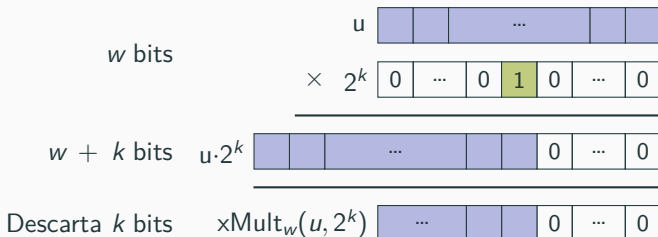
## Operación

- $u \cdot 2^k$  se puede hacer con desplazamientos:  $u \ll k$
- Tanto para signados como para no signados

## Ejemplos

- $u \ll 3 == u * 8$
- $(u \ll 5) - (u \ll 3) == u * 24$
- $64 - 1 \ll 3 * 2 - 1 ==$

# Multiplicar por potencias de 2: shift



## Operación

- $u \cdot 2^k$  se puede hacer con desplazamientos:  $u \ll k$
- Tanto para signados como para no signados

## Ejemplos

- $u \ll 3 == u * 8$
- $(u \ll 5) - (u \ll 3) == u * 24$
- $64 - 1 \ll 3 * 2 - 1 == 2016$

# Dividir por potencias de 2: shift

## Operación

- $\lfloor u/2^k \rfloor$  se puede hacer con desplazamientos:  $u \gg k$
- Tanto para signados como para no signados

## Ejemplos

|        | Teórica  | Real   | Hex   | Binario           |
|--------|----------|--------|-------|-------------------|
| x      | 16162    | 16162  | 3F 22 | 00111111 00100010 |
| x >> 1 | 8081.0   | 8081   | 1F 91 | 00011111 10010001 |
| x >> 2 | 4040.50  | 4040   | 0F C8 | 00001111 11001000 |
| x >> 8 | 63.133   | 63     | 00 3F | 00000000 00111111 |
| y      | -16162   | -16162 | C0 DE | 11000000 11011110 |
| y >> 1 | -8081.0  | -8081  | E0 6F | 11100000 01101111 |
| y >> 2 | -4040.50 | -4041  | F0 37 | 11110000 00110111 |
| y >> 8 | -63.133  | -64    | FF C0 | 11111111 11000000 |

# ¿Cuándo usar unsigned?

- **No usar sólo porque un número es no negativo**

- sin entender las consecuencias
- es fácil cometer errores

```
1 unsigned i;  
2 for (i = cnt-2; i >= 0; i--)  
3     a[i] += a[i+1];
```

- algunos muy sutiles

```
1 #define DELTA sizeof(int)  
2 int i;  
3 for (i = CNT; i-DELTA >= 0; i-= DELTA) ...
```

- **Usar cuando se opera con aritmética modular**

- Por ejemplo: aritmética de precisión arbitraria

- **Usar cuando se utilizan los bits para representar información en grupos**

# Contar hacia abajo usando unsigned

## ■ Manera correcta de usar unsigned en un ciclo

```
1 unsigned i;  
2 for (i = cnt - 2; i < cnt; --i)  
3     a[i] += a[i+1];
```

## ■ el estándar de C garantiza que la suma unsigned es modular:

$0 - 1 \rightarrow \text{UMax}$

## ■ Versión mejorada

```
1 size_t i;  
2 for (i = cnt - 2; i < cnt; --i)  
3     a[i] += a[i+1];
```

## ■ Versión mejorada y concisa

```
1 while (cnt--)  
2     a[cnt] += a[cnt + 1]; /* ?`cuál es el problema? */
```

# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Representación en memoria: enteros

**Decimal:** 16162

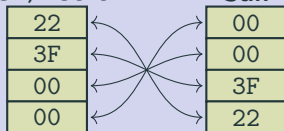
**Binario:** 0011 1111 0010 0010

**Hex:** 3 F 2 2

`int x = 16162;`

**IA32, x86-64**

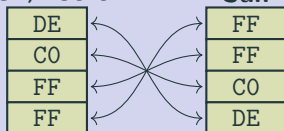
**Sun**



`int x = -16162;`

**IA32, x86-64**

**Sun**

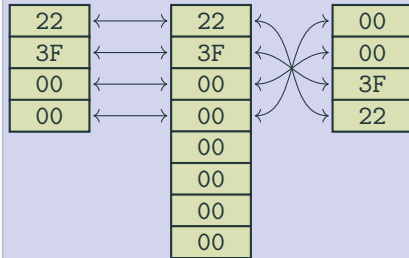


`long int x = 16162;`

**IA32**

**x86-64**

**Sun**



# Representación en memoria: punteros

```
1 int i = -16162;  
2 int *p = &i;
```

| Sun | IA32 | x86-64 |
|-----|------|--------|
| EF  | AC   | 3C     |
| FF  | 28   | 1B     |
| FB  | F5   | FE     |
| 2C  | FF   | 82     |
|     |      | FD     |
|     |      | 7F     |
|     |      | 00     |
|     |      | 00     |



# Representación en memoria: cadenas

## Cadenas en C

- Arreglo de chars
- Encoding ASCII
  - Set de caracteres estándar de 7-bit
  - El carácter '0' tiene el código 0x30
    - Dígito i: código  $0x30 + i$
  - El carácter 'a' tiene el código 0x61
  - El carácter 'A' tiene el código 0x41
- null-terminated (último carácter = 0)

## Compatibilidad

- No hay problema con el endianness

```
1 char s[] = "Orga 95.57";
```

| IA32 |   | Sun |
|------|---|-----|
| 4F   | ↔ | 4F  |
| 72   | ↔ | 72  |
| 67   | ↔ | 67  |
| 61   | ↔ | 61  |
| 20   | ↔ | 20  |
| 39   | ↔ | 39  |
| 35   | ↔ | 35  |
| 2E   | ↔ | 2E  |
| 35   | ↔ | 35  |
| 37   | ↔ | 37  |
| 00   | ↔ | 00  |

# Representación en memoria: programas

## Cada programa se almacena como una secuencia de instrucciones

- *Cada instrucción es una operación*
  - una operación aritmética
  - leer o escribir la memoria
  - un salto condicional
- *Cada instrucción se codifica con una secuencia de bytes*
  - Reduced Instruction Set Computer (RISC)
    - Instrucciones simples y más veloces
    - Requiere más instrucciones para una operación compleja
  - Complex Instruction Set Computer (CISC)
    - Instrucciones complejas y más lentas
    - Requiere menos instrucciones para una operación compleja
- El tipo de instrucciones y su codificación varía de máquina a máquina
  - Código binario no compatible

# Representación en memoria: programas

```

1 int multstore (int x, int y, int *r) {
2     *r = x * y;
3     return *r;
4 }

```

```
$ $CC -std=c99 -o $ARCH-multstore -c -Og multstore.c
```

intel/gcc

|    |
|----|
| 89 |
| F8 |
| 0F |
| AF |
| C6 |
| 89 |
| 02 |
| C3 |

arm/gcc

|    |
|----|
| FB |
| 01 |
| F0 |
| 00 |
| 60 |
| 10 |
| 47 |
| 70 |

mac/gcc

|    |
|----|
| E0 |
| 00 |
| 00 |
| 91 |
| E5 |
| 82 |
| 00 |
| 00 |
| E1 |
| 2F |
| FF |
| 1E |

# Examinando datos en C

## Código para imprimir los datos

```
1 typedef unsigned char *byte_pointer;
2
3 void show_bytes(byte_pointer start, size_t len) {
4     size_t i;
5
6     for (i = 0; i < len; i++)
7         printf("%p\t0x%02x\n", start+i, start[i]);
8     printf("\n");
9 }
```

## Especificadores de printf:

imprimir puntero: %p

imprimir valor hexadecimal: %x

# Examinando datos en C

## Código para imprimir los datos

```
1 typedef unsigned char *byte_pointer;
2
3 void show_bytes(byte_pointer start, size_t len) {
4     size_t i;
5
6     for (i = 0; i < len; i++)
7         printf("%p\t0x%02x\n", start+i, start[i]);
8     printf("\n");
9 }
```

## Especificadores de printf:

imprimir puntero: %p

imprimir valor hexadecimal: %x

Castear *byte\_pointer* a `unsigned char *` nos permite trabajar los datos como un arreglo de bytes

# Ejemplo de ejecución

## Código

```
1 unsigned int manzana = 2953253037;  
2 puts("unsigned int manzana = 2953253037;")  
3 show_bytes((byte_pointer) &manzana, sizeof(unsigned int));
```

## Resultado (GNU/Linux x86-64)

```
unsigned int manzana = 2953253037;  
0x7ffc49e50c24 0xad  
0x7ffc49e50c25 0x10  
0x7ffc49e50c26 0x07  
0x7ffc49e50c27 0xb0
```

## Ejemplo de ejecución

### Código

```
1 unsigned int manzana = 2953253037;  
2 puts("unsigned int manzana = 2953253037;")  
3 show_bytes((byte_pointer) &manzana, sizeof(unsigned int));
```

### Resultado (GNU/Linux x86-64)

```
unsigned int manzana = 2953253037;  
0x7ffc49e50c24 0xad  
0x7ffc49e50c25 0x10  
0x7ffc49e50c26 0x07  
0x7ffc49e50c27 0xb0
```

¿En qué *endianness* está guardado?

# Descifrando código desensamblado

## Desensamblado

- Texto que representa código de máquina binario
- Generado por un programa que lee el código de máquina

## Ejemplo

| Address  | Instruction Code     | Assembly          |
|----------|----------------------|-------------------|
| 8048365: | 5b                   | pop %ebx          |
| 8048366: | 81 c3 ab 12 00 00    | \$0x12ab, %ebx    |
| 804836c: | 83 bb 28 00 00 00 00 | \$0x0, 0x28(%ebx) |

## Descifrando valores

Valor: 0x12ab  
 Pad a 32-bits: 0x000012ab  
 Separar en bytes: 00 00 12 ab  
 Invertir: ab 12 00 00



# Descifrando código desensamblado

## Desensamblado

- Texto que representa código de máquina binario
- Generado por un programa que lee el código de máquina

## Ejemplo

| Address  | Instruction Code     | Assembly          |
|----------|----------------------|-------------------|
| 8048365: | 5b                   | pop %ebx          |
| 8048366: | 81 c3 ab 12 00 00    | \$0x12ab, %ebx    |
| 804836c: | 83 bb 28 00 00 00 00 | \$0x0, 0x28(%ebx) |

## Descifrando valores

Valor: 0x12ab  
 Pad a 32-bits: 0x000012ab  
 Separar en bytes: 00 00 12 ab  
 Invertir: ab 12 00 00

¿En qué *endianness* está guardado?

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

$$1. \ x < 0 \quad \Rightarrow \quad ((x*2) < 0)$$

**Falso:**

$$2. \ ux \geq 0$$

$$3. \ x \ \& \ 7 == 7 \quad \Rightarrow \quad (x \ll 30) < 0$$

$$4. \ ux > -1$$

$$5. \ x > y \quad \Rightarrow \quad -x < -y$$

$$6. \ x * x \geq 0$$

$$7. \ x > 0 \ \&\& \ y > 0 \quad \Rightarrow \quad x + y > 0$$

$$8. \ x \geq 0 \quad \Rightarrow \quad -x \leq 0$$

$$9. \ x \leq 0 \quad \Rightarrow \quad -x \geq 0$$

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

1.  $x < 0 \Rightarrow ((x*2) < 0)$  **Falso:** TMin

2.  $ux \geq 0$

3.  $x \& 7 == 7 \Rightarrow (x \ll 30) < 0$

4.  $ux > -1$

5.  $x > y \Rightarrow -x < -y$

6.  $x * x \geq 0$

7.  $x > 0 \&\& y > 0 \Rightarrow x + y > 0$

8.  $x \geq 0 \Rightarrow -x \leq 0$

9.  $x \leq 0 \Rightarrow -x \geq 0$

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

$$1. \ x < 0 \quad \Rightarrow \quad ((x*2) < 0)$$

**Falso:** TMin

$$2. \ ux \geq 0$$

**Verdadero:**

$$3. \ x \& 7 == 7 \quad \Rightarrow \quad (x < 30) < 0$$

$$4. \ ux > -1$$

$$5. \ x > y \quad \Rightarrow \quad -x < -y$$

$$6. \ x * x \geq 0$$

$$7. \ x > 0 \ \&\& \ y > 0 \quad \Rightarrow \quad x + y > 0$$

$$8. \ x \geq 0 \quad \Rightarrow \quad -x \leq 0$$

$$9. \ x \leq 0 \quad \Rightarrow \quad -x \geq 0$$

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

$$1. \ x < 0 \quad \Rightarrow \quad ((x*2) < 0)$$

**Falso:** TMin

$$2. \ ux \geq 0$$

**Verdadero:**  $0 = \text{UMin}$

$$3. \ x \& 7 == 7 \quad \Rightarrow \quad (x \ll 30) < 0$$

$$4. \ ux > -1$$

$$5. \ x > y \quad \Rightarrow \quad -x < -y$$

$$6. \ x * x \geq 0$$

$$7. \ x > 0 \ \&\& \ y > 0 \quad \Rightarrow \quad x + y > 0$$

$$8. \ x \geq 0 \quad \Rightarrow \quad -x \leq 0$$

$$9. \ x \leq 0 \quad \Rightarrow \quad -x \geq 0$$

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |                   |
|-----------------------|------------------------------|-------------------|-------------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin              |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = \text{UMin}$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> |                   |
| 4. $ux > -1$          |                              |                   |                   |
| 5. $x > y$            | $\Rightarrow -x < -y$        |                   |                   |
| 6. $x * x \geq 0$     |                              |                   |                   |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      |                   |                   |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                   |                   |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |                   |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |                   |
|-----------------------|------------------------------|-------------------|-------------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin              |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = \text{UMin}$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$         |
| 4. $ux > -1$          |                              |                   |                   |
| 5. $x > y$            | $\Rightarrow -x < -y$        |                   |                   |
| 6. $x * x \geq 0$     |                              |                   |                   |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      |                   |                   |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                   |                   |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |                   |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |                   |
|-----------------------|------------------------------|-------------------|-------------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin              |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = \text{UMin}$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$         |
| 4. $ux > -1$          |                              | <b>Falso:</b>     |                   |
| 5. $x > y$            | $\Rightarrow -x < -y$        |                   |                   |
| 6. $x * x \geq 0$     |                              |                   |                   |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      |                   |                   |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                   |                   |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |                   |



# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |                   |
|-----------------------|------------------------------|-------------------|-------------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin              |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = \text{UMin}$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$         |
| 4. $ux > -1$          |                              | <b>Falso:</b>     | 0                 |
| 5. $x > y$            | $\Rightarrow -x < -y$        |                   |                   |
| 6. $x * x \geq 0$     |                              |                   |                   |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      |                   |                   |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                   |                   |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |                   |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |                   |
|-----------------------|------------------------------|-------------------|-------------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin              |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = \text{UMin}$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$         |
| 4. $ux > -1$          |                              | <b>Falso:</b>     | 0                 |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b>     |                   |
| 6. $x * x \geq 0$     |                              |                   |                   |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      |                   |                   |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                   |                   |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |                   |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                                     |
|-----------------------|------------------------------|-------------------------------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b> TMin                  |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> $0 = \text{UMin}$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> $x_1 = 1$         |
| 4. $ux > -1$          |                              | <b>Falso:</b> 0                     |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b> -1, TMin              |
| 6. $x * x \geq 0$     |                              |                                     |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      |                                     |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                                     |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                                     |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                                     |
|-----------------------|------------------------------|-------------------------------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b> TMin                  |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> $0 = \text{UMin}$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> $x_1 = 1$         |
| 4. $ux > -1$          |                              | <b>Falso:</b> 0                     |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b> -1, TMin              |
| 6. $x * x \geq 0$     |                              | <b>Falso:</b>                       |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      |                                     |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                                     |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                                     |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |            |
|-----------------------|------------------------------|-------------------|------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin       |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = UMin$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$  |
| 4. $ux > -1$          |                              | <b>Falso:</b>     | 0          |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b>     | -1, TMin   |
| 6. $x * x \geq 0$     |                              | <b>Falso:</b>     | 49374      |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      |                   |            |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                   |            |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |            |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |                   |
|-----------------------|------------------------------|-------------------|-------------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin              |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = \text{UMin}$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$         |
| 4. $ux > -1$          |                              | <b>Falso:</b>     | 0                 |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b>     | -1, TMin          |
| 6. $x * x \geq 0$     |                              | <b>Falso:</b>     | 49374             |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      | <b>Falso:</b>     |                   |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                   |                   |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |                   |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |            |
|-----------------------|------------------------------|-------------------|------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin       |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = UMin$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$  |
| 4. $ux > -1$          |                              | <b>Falso:</b>     | 0          |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b>     | -1, TMin   |
| 6. $x * x \geq 0$     |                              | <b>Falso:</b>     | 49374      |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      | <b>Falso:</b>     | TMax, TMax |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      |                   |            |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |            |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |            |
|-----------------------|------------------------------|-------------------|------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin       |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = UMin$ |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$  |
| 4. $ux > -1$          |                              | <b>Falso:</b>     | 0          |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b>     | -1, TMin   |
| 6. $x * x \geq 0$     |                              | <b>Falso:</b>     | 49374      |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      | <b>Falso:</b>     | TMax, TMax |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      | <b>Verdadero:</b> |            |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |            |



# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |             |
|-----------------------|------------------------------|-------------------|-------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin        |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = UMin$  |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$   |
| 4. $ux > -1$          |                              | <b>Falso:</b>     | 0           |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b>     | -1, TMin    |
| 6. $x * x \geq 0$     |                              | <b>Falso:</b>     | 49374       |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      | <b>Falso:</b>     | TMax, TMax  |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      | <b>Verdadero:</b> | $-TMax < 0$ |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      |                   |             |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |             |
|-----------------------|------------------------------|-------------------|-------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin        |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = UMin$  |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$   |
| 4. $ux > -1$          |                              | <b>Falso:</b>     | 0           |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b>     | -1, TMin    |
| 6. $x * x \geq 0$     |                              | <b>Falso:</b>     | 49374       |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      | <b>Falso:</b>     | TMax, TMax  |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      | <b>Verdadero:</b> | $-TMax < 0$ |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      | <b>Falso:</b>     |             |

# C Puzzles

- 32-bit *word-size*, complemento a dos
- Argumentar si es cierto, o dar un contraejemplo en las siguientes expresiones ( $x, y$ : int,  $ux, uy$ : unsigned)

|                       |                              |                   |             |
|-----------------------|------------------------------|-------------------|-------------|
| 1. $x < 0$            | $\Rightarrow ((x*2) < 0)$    | <b>Falso:</b>     | TMin        |
| 2. $ux \geq 0$        |                              | <b>Verdadero:</b> | $0 = UMin$  |
| 3. $x \& 7 == 7$      | $\Rightarrow (x \ll 30) < 0$ | <b>Verdadero:</b> | $x_1 = 1$   |
| 4. $ux > -1$          |                              | <b>Falso:</b>     | 0           |
| 5. $x > y$            | $\Rightarrow -x < -y$        | <b>Falso:</b>     | -1, TMin    |
| 6. $x * x \geq 0$     |                              | <b>Falso:</b>     | 49374       |
| 7. $x > 0 \&\& y > 0$ | $\Rightarrow x + y > 0$      | <b>Falso:</b>     | TMax, TMax  |
| 8. $x \geq 0$         | $\Rightarrow -x \leq 0$      | <b>Verdadero:</b> | $-TMax < 0$ |
| 9. $x \leq 0$         | $\Rightarrow -x \geq 0$      | <b>Falso:</b>     | TMin        |

# Tabla de contenidos

---

## 1. Representando la información como bits

Sistemas de numeración

La información se mueve en bits

Organización de la memoria

Lilliput & Blefuscu

## 2. Manipulación de bits

## 3. Representación de enteros

Representaciones signed y unsigned

Uso en C

Aritmética

## 4. Representaciones en memoria

## 5. Code Security

# Code Security

```
1 /* Kernel memory region holding user-accessible data */
2 #define KSIZE 1024
3 char kbuf[KSIZE];
4
5 /* Copy at most maxlen bytes from kernel region to buffer */
6 int copy_from_kernel(void *user_dest, int maxlen) {
7     /* Byte count len is minimum of buffer size and maxlen */
8     int len = KSIZE < maxlen ? KSIZE : maxlen;
9     memcpy(user_dest, kbuf, len);
10    return len;
11 }
```

- ¿Hay alguna vulnerabilidad a la vista?
- Hay *legiones* de personas buscando vulnerabilidades en los programas.. y no siempre para reportarlas y mejorar el software.

## Code Security: uso típico

```
1 /* Kernel memory region holding user-accessible data */
2 #define KSIZE 1024
3 char kbuf[KSIZE];
4
5 /* Copy at most maxlen bytes from kernel region to buffer */
6 int copy_from_kernel(void *user_dest, int maxlen) {
7     /* Byte count len is minimum of buffer size and maxlen */
8     int len = KSIZE < maxlen ? KSIZE : maxlen;
9     memcpy(user_dest, kbuf, len);
10    return len;
11 }
```

```
1 #define MSIZE 528
2 void getstuff(void) {
3     char buf[MSIZE];
4     copy_from_kernel(buf, MSIZE);
5     printf("%s\n", buf);
6 }
```

# Code Security: uso malicioso

```
1 /* Kernel memory region holding user-accessible data */
2 #define KSIZE 1024
3 char kbuf[KSIZE];
4
5 /* Copy at most maxlen bytes from kernel region to buffer */
6 int copy_from_kernel(void *user_dest, int maxlen) {
7     /* Byte count len is minimum of buffer size and maxlen */
8     int len = KSIZE < maxlen ? KSIZE : maxlen;
9     memcpy(user_dest, kbuf, len);
10    return len;
11 }
```

```
1 #define MSIZE 528
2 void getstuff(void) {
3     char buf[MSIZE];
4     copy_from_kernel(buf, -MSIZE);
5     printf("%s\n", buf);
6 }
```

```
1 /* Declaration of library
   function memcpy */
2 void *
3 memcpy(void *dest,
4         void *src,
5         size_t n)
```

# Licencia del estilo de beamer

Obtén el código de este estilo y la presentación demo en

`github.com/pamoreno/mtheme`

El estilo *en sí* está licenciado bajo la Creative Commons Attribution-ShareAlike 4.0 International License. El estilo es una modificación del creado por Matthias Vogelgesang, disponible en

`github.com/matze/mtheme`

