Curso 0: Arquitectura de Computadores

Carlos Garre

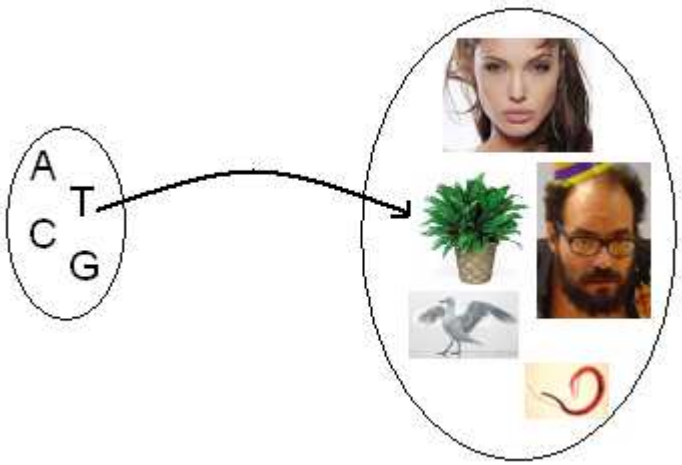
carlos.garre@urjc.es

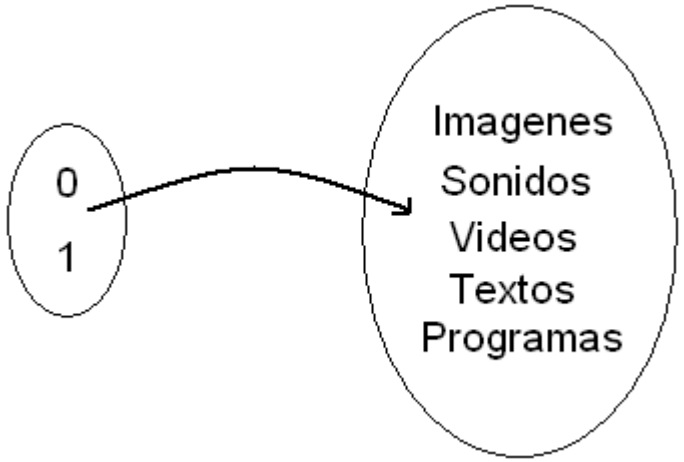
Índice

- **Representación de la Información**
- **Introducción a la Arquitectura de Computadores**
- **Modelo Von Neumann**
- **Jerarquía de Memoria**
- El Procesador
- Dispositivos de Entrada/Salida
- Arquitecturas Avanzadas

Representación de la Información

- Codificación:





- El ordenador codifica toda clase de información utilizando únicamente dos valores:
  - 0: ausencia de corriente eléctrica.
  - 1: corriente eléctrica.

- Es común que trabajemos directamente con información numérica: contabilidad, calificaciones del máster, etc…
- Pero además, el ordenador muchas veces trabaja implícitamente con números:
  - Cuando jugamos a un **juego de ordenador**, éste debe calcular constantemente las coordenadas espaciales de todos los elementos de nuestro escenario 3D, y esas coordenadas se representan como vectores de números reales.
  - Cuando escribimos un **texto en Word**, éste debe llevar un recuento del número de páginas, de la numeración de los capítulos, etc.
  - Cuando **navegamos por Internet**, el ordenador debe controlar las direcciones IP y los puertos TCP y controlar muchos otros números que permiten la conexión entre dos máquinas (números de secuencia, sumas de comprobación, etc).

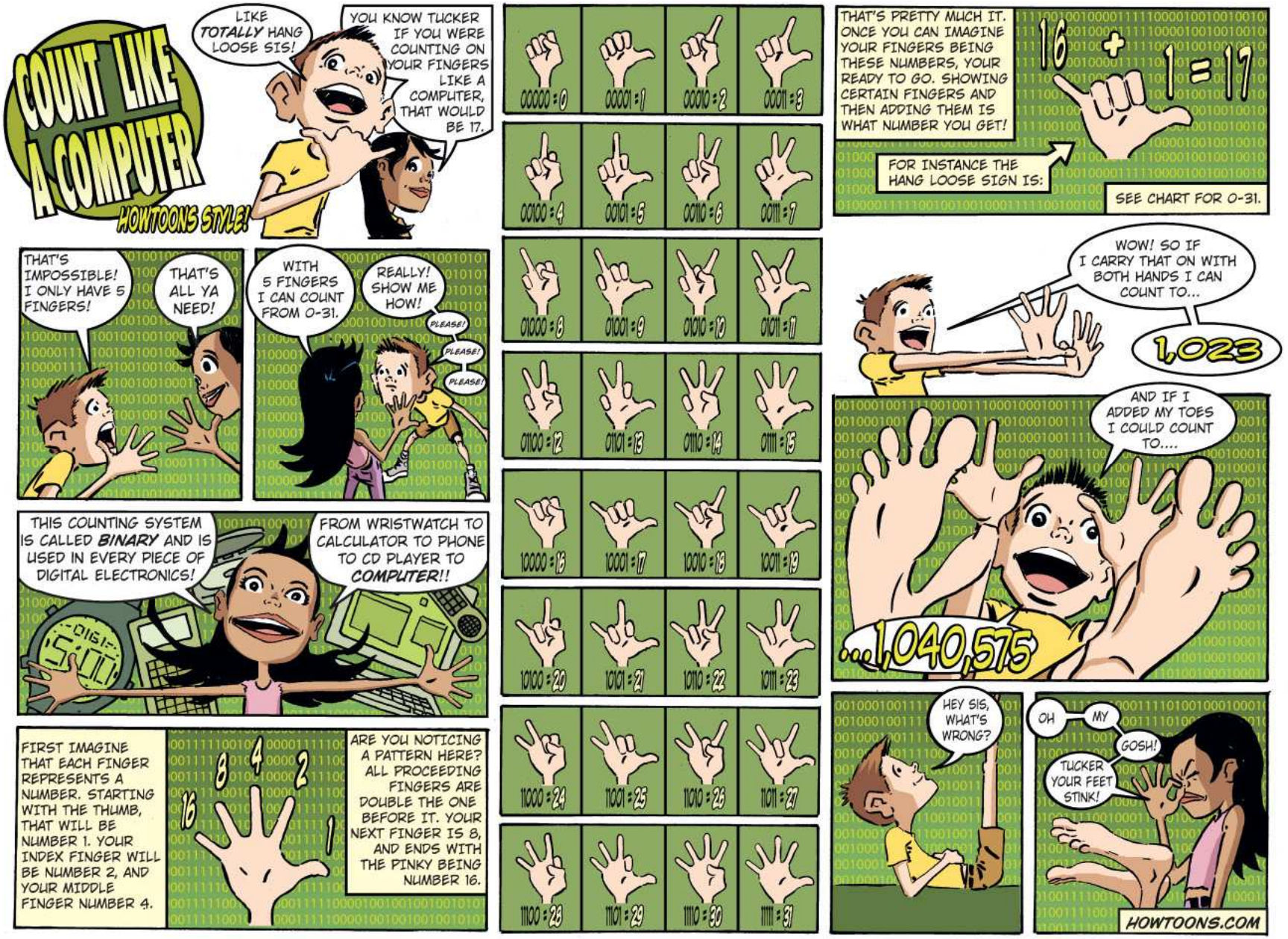
Representación de Números

- La forma de representar (codificar) los números es diferente según el dominio que queramos representar:
  - Números naturales:
    - Binario natural.
    - BCD.
  - Números enteros:
    - Signo y Magnitud.
    - Representación en complementos (Ca1 y Ca2).
    - Representaciones en exceso.
  - Números reales:
    - Coma fija.
    - Coma flotante (estándar IEEE-754).

Contando en binario…

¿Hasta cuánto podemos contar con los dedos de las manos?

Contando en binario…



Conversión de binario a decimal

- **Base decimal:**

  $345_{)10} = 3*10^2 + 4*10^1 + 5*10^0$

- **Base B:**

  $X_{)B} = C_N C_{N-1} \ldots C_2 C_1 C_0 = C_0*B^0 + C_1*B^1 + C_2*B^2 + \ldots + C_{N-1}*B^{N-1} + C_N*B^N$

- **Base binaria (B=2):**

  $10111001101_{)2} = 1*2^0 + 0*2^1 + 1*2^2 + 1*2^3 + 0*2^4 + 0*2^5 + 1*2^6 + 1*2^7 + 1*2^8 + 0*2^9 + 1*2^{10} = 1 + 4 + 8 + 64 + 128 + 256 + 1024 = 1485_{)10}$

Definiciones y conceptos clave

- **Bit:** una cifra en sistema binario. Puede ser 0 o 1.
- **Byte:** secuencia de 8 bits.
- ¿Cuál es el mayor número que podemos representar con un byte?

$$11111111_{)2} = 255_{)10}$$

En general, con una secuencia de N bits podemos representar números desde 0 hasta 2^N-1 (2^N valores diferentes)

Conversión de decimal a binario

- Método de las divisiones sucesivas:

1. Dividimos el número por la base (2), y obtenemos como resultado un cociente y un resto de la división. Anotamos el resto.
2. Cogemos el cociente de la anterior división, y lo dividimos por la base (2). Anotamos el resto.
3. Continuamos el proceso hasta que el cociente es 1. Cuando llegamos a este punto, empezamos a escribir los bits del número convertido siguiendo este orden de izquierda (bit más significativo) a derecha (bit menos significativo):
   - 1ª cifra: último cociente (siempre será 1).
   - 2ª cifra: último resto
   - 3ª cifra: penúltimo resto
   - Y así sucesivamente hasta ubicar el último bit (más a la derecha), que será el primer resto que anotamos.

Conversión de decimal a binario

Ejemplo:

Sea $419)_{10}$ su conversión a binario (base 2) será:

419 / 2
- Resto: **1**
- Cociente: 209

209 / 2
- Resto: **1**
- Cociente: 104

104 / 2
- Resto: **0**
- Cociente: 52

52 / 2
- Resto: **0**
- Cociente: 26

26 / 2
- Resto: **0**
- Cociente: 13

13 / 2
- Resto: **1**
- Cociente: 6

6 / 2
- Resto: **0**
- Cociente: 3

3 / 2
- Resto: **1**
- Cociente: **1**

Obtenemos el resultado simplemente colocando de izquierda a derecha cada uno de los números marcados en rojo desde abajo hacia arriba:

$110100011)_2$

Representación de Números Enteros

- ¿Cómo representar el signo (+/-) de un número si sólo podemos usar ceros y unos?

  ¡Nos inventamos un convenio!

  - **0 = +**
  - **1 = -**

- **Representación en Signo y Magnitud:** por convenio, el primer bit es el signo, y el resto de bits codifican un número natural.

  11101 → Signo: 1 (-), Magnitud: 1101 (13).

$$11101_{)SyM} = -13_{)10}$$

¡Concepto clave! Separación en dos campos: signo y magnitud.

Representación de Números Reales

- ¿Cómo representamos la coma de un número real?

  Separando en dos campos: parte entera y parte fraccionaria.

- *Ejemplo:* 4 bits para parte entera y 4 para parte fraccionaria:

  11011101 = 1101’1101 = 13’13

  Problema: no podemos representar números mayores que 15, mientras que en binario natural con 8 bits podríamos hasta 255.

  Solución: ¿dar más bits a la parte entera?

  11011101 = 110111’01 = 55’25

  Nuevo problema: ¡ahora sólo podemos tener x’0, x’25, x’5, o x’75!

Conclusión:

- Más bits en la parte entera = números más grandes
- Más bits en la parte fraccionaria = números más precisos

- Entonces, ¿cuál es el mejor compromiso?

  Un sistema que te permita mover la coma de izquierda a derecha según convenga para cada número:

  Coma fija vs coma flotante.

- **Notación científica:** forma habitual de representar números muy grandes y/o números con parte fraccionaria:

  Un trillón = 1000000000000000000 = 10^{18}

  Dos milmillonésimas = 0'000000002 = $2*10^{-9}$

- Cualquier número se puede representar en notación científica de diferentes formas:

  14567 = 14567'0*10^0 = 14'567*10^3 = …

  Un trillón = 0'1*10^{20} = …

Representación de Números Reales

- En general:

$$X = M*10^E$$

- Donde:
  - X: número a representar.
  - M: mantisa (número real).
  - E: exponente (número entero).

- En realidad, podemos usar cualquier otra base:

$$X = M*B^E$$

- Ejemplo:

$$14567'0*10^0 = 14567'0*2^0 = 1'4567*10^4 \sim 1'778199*2^{13}$$

Representación de Números Reales

- Representación habitual de números reales en el ordenador:
- Formato estándar de coma flotante (IEEE-754) donde se establece cómo representar la base, la mantisa y el exponente:
  - Base fija = 2 (no hay que representarla).
  - Exponente = 8 bits.
  - Mantisa = 24 bits (incluyendo signo).
- Pero si la mantisa es a su vez un número real, ¿cómo lo representamos? → mantisa **normalizada**.

$$11101 = 11101*2^0 = 1110'1*2^1 = \ldots = 1'1101*2^4$$

- Todo número se puede representar siempre con una mantisa cuya parte entera es 1 (forma normalizada) → sólo necesitamos representar la parte fraccionaria.

Sobre la coma flotante…

- El estándar permite definir además valores especiales para representar ±∞, e incluso valores imposibles (0/0, etc) → **NaN**.
- El estándar define representaciones con mayor número de bits: precisión doble (64 bits), extendida (80 bits)…
- **¡¡¡Importantes consideraciones al usar coma flotante!!!**
  - Hay muchos números que no se pueden representar de forma exacta (ejemplo: 0'7).
  - La precisión es muchísimo mayor entre 0 y 1.
  - Los números grandes no tienen precisión.
  - "¡Las comparaciones son odiosas!"

```
if (x == 0.5)
```

```
if (abs(x – 0'5) < error)
```

Representación de texto

- Byte (B) = una “letra”
- KiloByte (KB) = 1024 bytes = una página
- MegaByte (MB) = 1024 KBs = un libro
- GigaByte (GB) = 1024 MBs = una librería
- TeraByte (TB) = 1024 GBs = una gran biblioteca
- PetaByte (PB) = 1024 TBs = todos los libros de un país
- ExaByte (EB) = 1024 PBs = todos los libros del mundo
- ZettaByte (ZB) = 1024 EBs = …
- YottaByte (YB) = 1024 ZBs = ¿todos los libros de la galaxia?

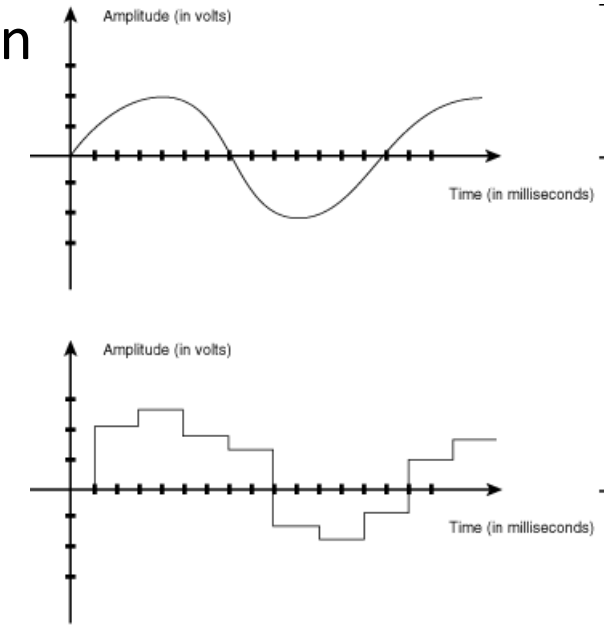
Estándares de representación de texto: ASCII, Unicode…

Representación de imagen

- **Mapas de bits:**
  - Discretización mediante una cuadrícula de **píxeles**.
  - Resolución: número de columnas x número de filas.
  - Habitual en fotografía: JPG, BMP, GIF, ...
- **Representación vectorial:**
  - Formar imagen a base de primitivas: rectas, círculos, ...
  - Facilitan edición, zoom, etc, y reducen tamaño, pero imposible describir imágenes fotorealistas.
  - Fuentes de letras, CorelDraw, ...
- **Codificación de color:**
  - Paletas de colores predefinidos.
  - Colores aditivos (RGB, ...).

Representación de sonido

- Discretización de las variaciones de presión del aire:
  - Eje X = tiempo.
  - Eje Y = amplitud (presión del aire).



- Calidad CD:
  - Frecuencia de muestreo 44100Hz (44100 muestras/segundo).
  - 16 bits por muestra (65536 posibles valores de amplitud).
  - 2 canales: stereo.

Representación de video

- Ejercicio de cálculo de espacio (película en DVD):
  - Resolución imagen: 720x480 píxeles.
  - Color verdadero: 3 bytes por píxel.
  - 30 imágenes por segundo (fps).
  - Duración de la película: 1h30’
  - Sonido en calidad 96KHz y 24 bits, con 6 canales (5.1) y 4 idiomas.

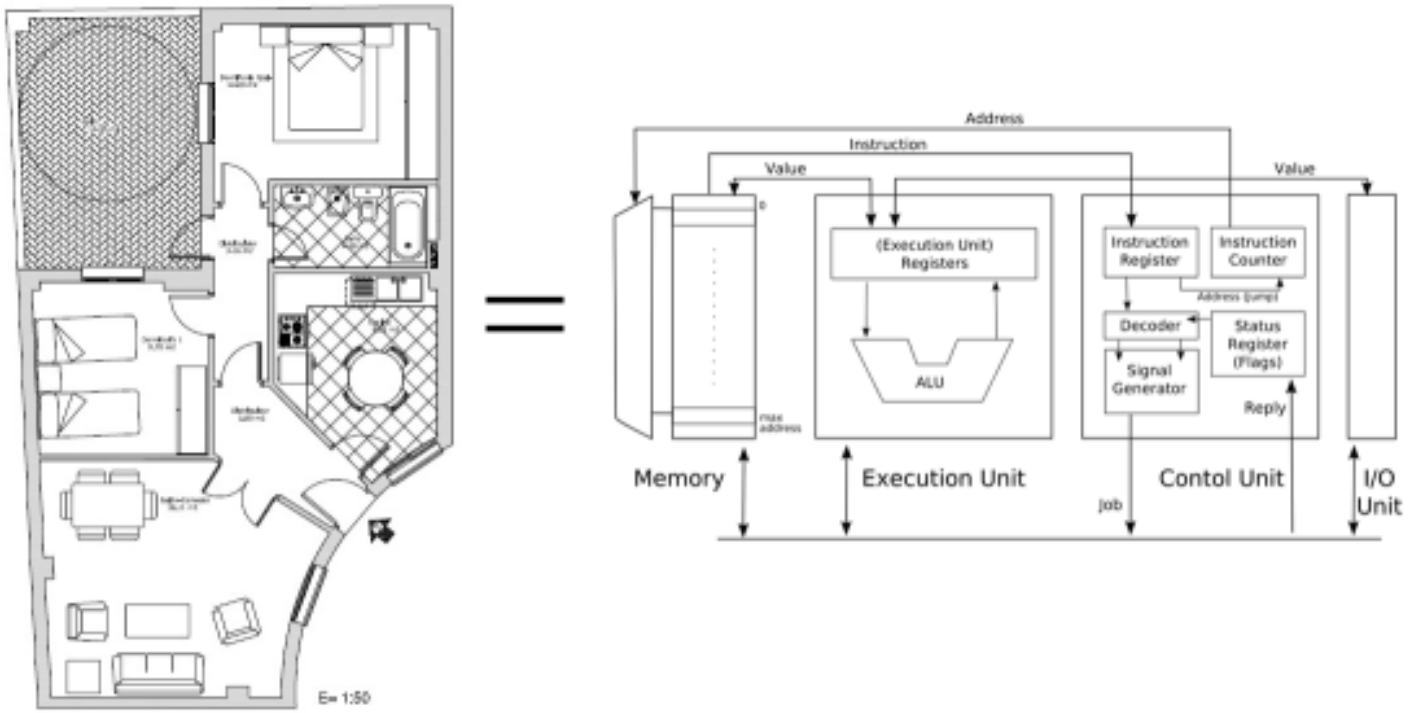
- Cálculo espacio imagen: 720x480x3x30x90x60 ~ 156’4 GBs
- Cálculo espacio sonido: 96000x3x6x4x90x60 ~ 34’7 GBs
- **Total ~ 191 GBs!!!** **(DVD = 4’7GBs)**

Representación de video

- Formatos de representación de vídeo: AVI, MPG, MOV, ... → La mayoría son sólo contenedores con información sobre:
  - Cómo se codifica la imagen (códec de vídeo).
  - Cómo se codifica el sonido (códec de audio).
- Códecs de vídeo:
  - Compresión con pérdida.
  - Keyframes + variaciones entre frames.
  - Se instalan en el Sistema Operativo.
  - DivX, xVid, QuickTime, WMV, ...
- Códecs de audio:
  - Compresión con pérdida.
  - División en bandas de frecuencia ← modelo psicoacústico
  - MP3, Ogg, AAC, WMA...

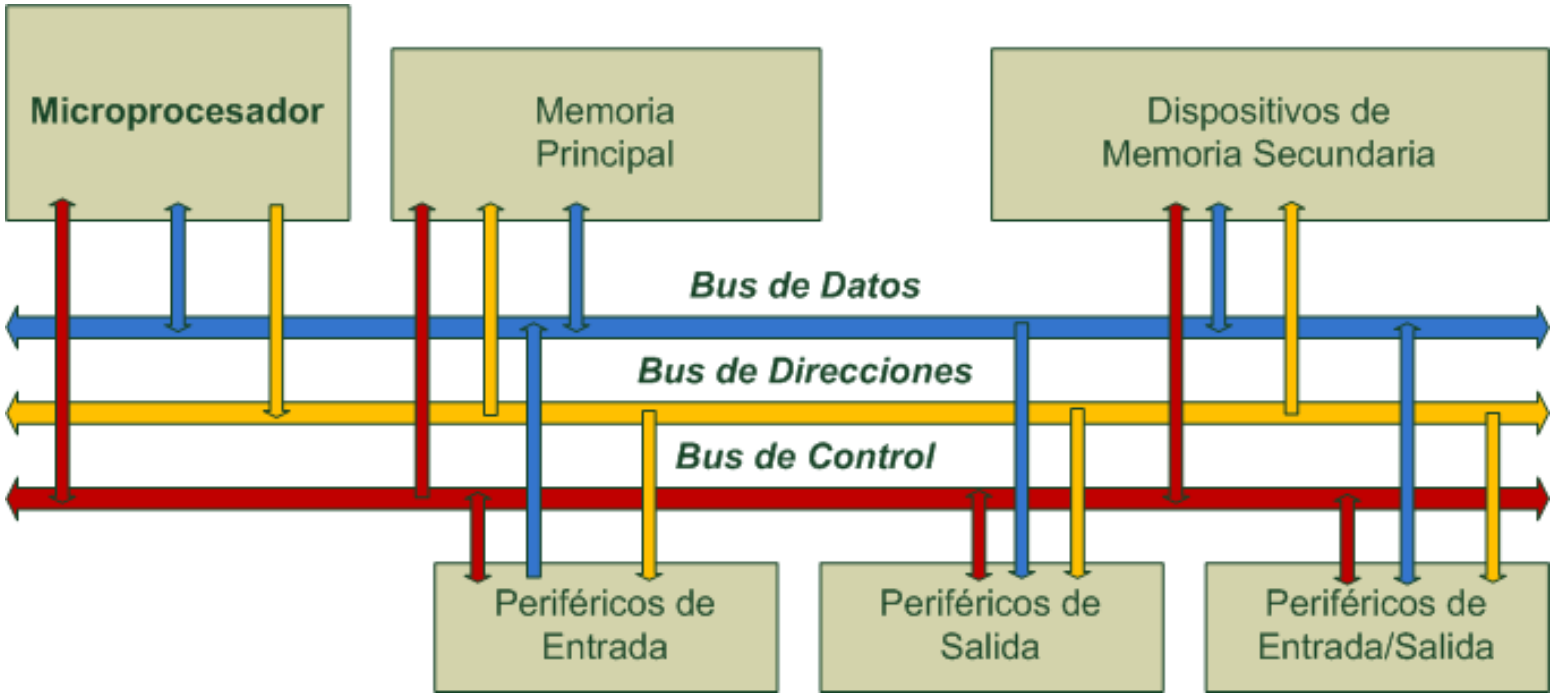
Índice

- Profesores del máster: la mayoría del Departamento de Arquitectura y Tecnología de Computadores.
- **Arquitectura:** estudia la forma en que se deben disponer una serie de elementos básicos (columnas, muros, ...) para el diseño de un edificio de unas determinadas características.
- **Arquitectura de computadores**: estudia la forma en que se deben disponer una serie de elementos básicos (que iremos viendo...) para el diseño de un computador de unas determinadas características.
- **Tecnología de computadores:** estudia el diseño de esos componentes básicos (los arquitectos no se dedican a fabricar ladrillos...).

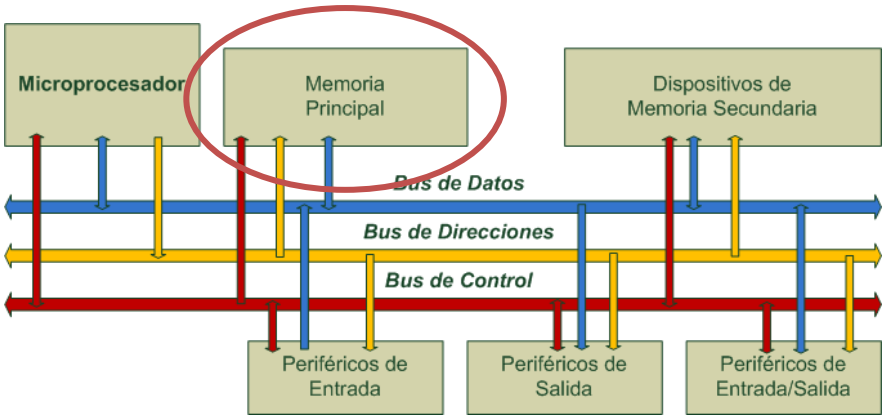

E= 1:50
=
Address
Instruction
Value
Value
(Execution Unit) Registers
ALU
Instruction Register
Instruction Counter
Decoder
Status Register (Flags)
Signal Generator
Reply
Memory
Execution Unit
Contol Unit
I/O Unit
Job

Índice

Modelo Von Neumann



Modelo Von Neumann



Memoria Principal:

- En ella se almacenan los datos e instrucciones necesarios para ejecutar los programas.
- Memoria principal = Datos + Instrucciones.
- Cada posición de memoria tiene asociada una dirección.

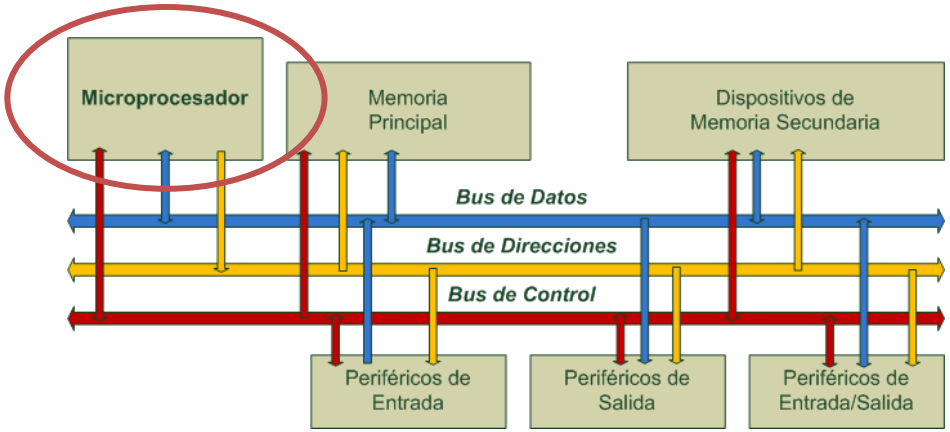
Arquitectura Von Neumann:

- Una única memoria.

Arquitectura Harvard:

- Dos memorias separadas: Datos, Instrucciones

Modelo Von Neumann



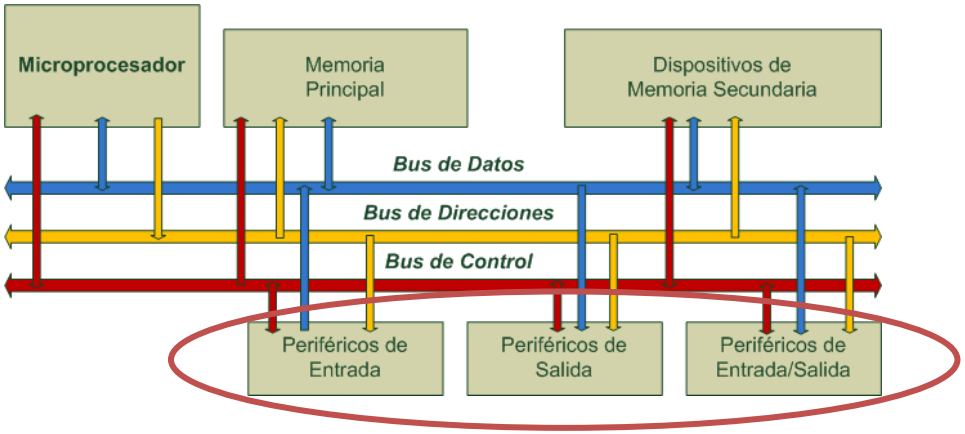
Unidad Central de Proceso (CPU)

- En ella se ejecutan las instrucciones que se leen de la memoria.
- En el caso de un computador, normalmente se trata de un **Microprocesador.**

Microprocesador:

- Unidad central de proceso implementada sobre un circuito integrado formado por millones de transistores.

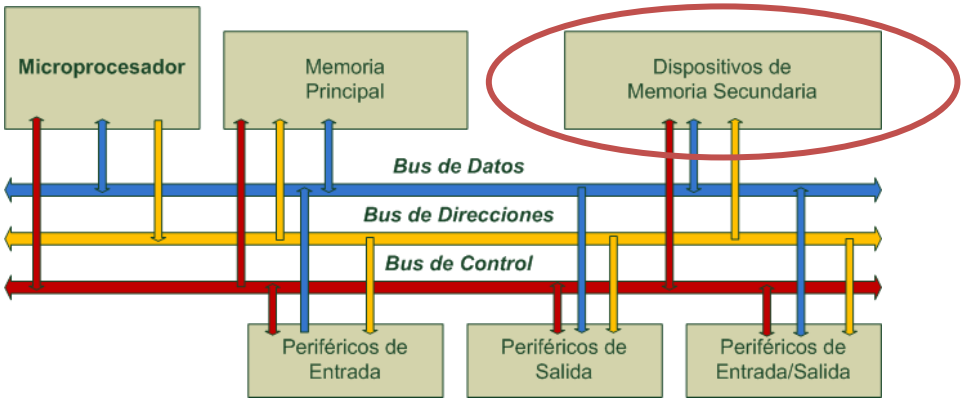
Modelo Von Neumann



Unidad de Entrada/Salida:

- Permite la comunicación entre el microprocesador y los periféricos.
- Periféricos de entrada: ratón, teclado, …
- Periféricos de salida: monitor, impresora, …
- Periféricos de entrada/salida: tarjeta de red, tarjeta de sonido, …

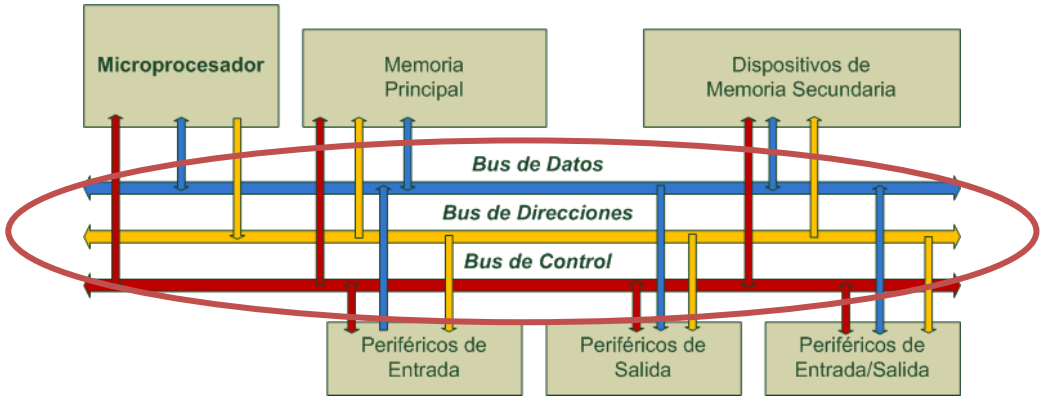
Modelo Von Neumann



Dispositivos de Memoria Secundaria:

- Son periféricos de entrada/salida.
- Permiten almacenar cantidades masivas de datos.
- Almacenan los datos que no se están utilizando en ese momento.
- Diferentes niveles en función de su capacidad/velocidad.
- Ejemplos: disco duro, DVD, CD, pendrive, tarjeta SD, …

Modelo Von Neumann



Bus de Sistema:

- Mecanismo de comunicación entre los componentes del computador.

Bus de Datos

- Circulan los datos que se transfieren entre dos componentes.

Bus de Direcciones

- Localiza la ubicación de los datos dentro de una memoria o un periférico.

Bus de Control

- Transmite las señales de control para coordinar las comunicaciones.

Índice

Jerarquía de Memoria

- **Ejecutar un programa es como leer un libro:**
  - Tenemos una biblioteca llena de libros, de la que escogeremos el que queremos leer.
  - El libro contiene toda la información con la que vamos a "trabajar". Es imposible realizar la tarea de leer el libro sin tener el libro en la mano.
- Para un programador sólo existen dos tipos de memoria:
  - **Memoria principal:** contiene los datos con los que estamos trabajando en **este** momento (**libro**). *En programación: variables.*
  - **Memoria secundaria:** contiene todos los posibles datos con los que podríamos querer trabajar en algún momento (**biblioteca**). *En programación: ficheros en el disco duro.*

Jerarquía de Memoria

- Aunque el programador pueda abstraerse normalmente de esto, en realidad no sólo hay dos memorias. Físicamente hay toda una serie de memorias dispuestas de forma jerárquica que facilitan el acceso a la información de forma eficiente.
- El procesador es como un trabajador que realiza tareas sobre su mesa de trabajo. Imaginemos que es un carpintero…

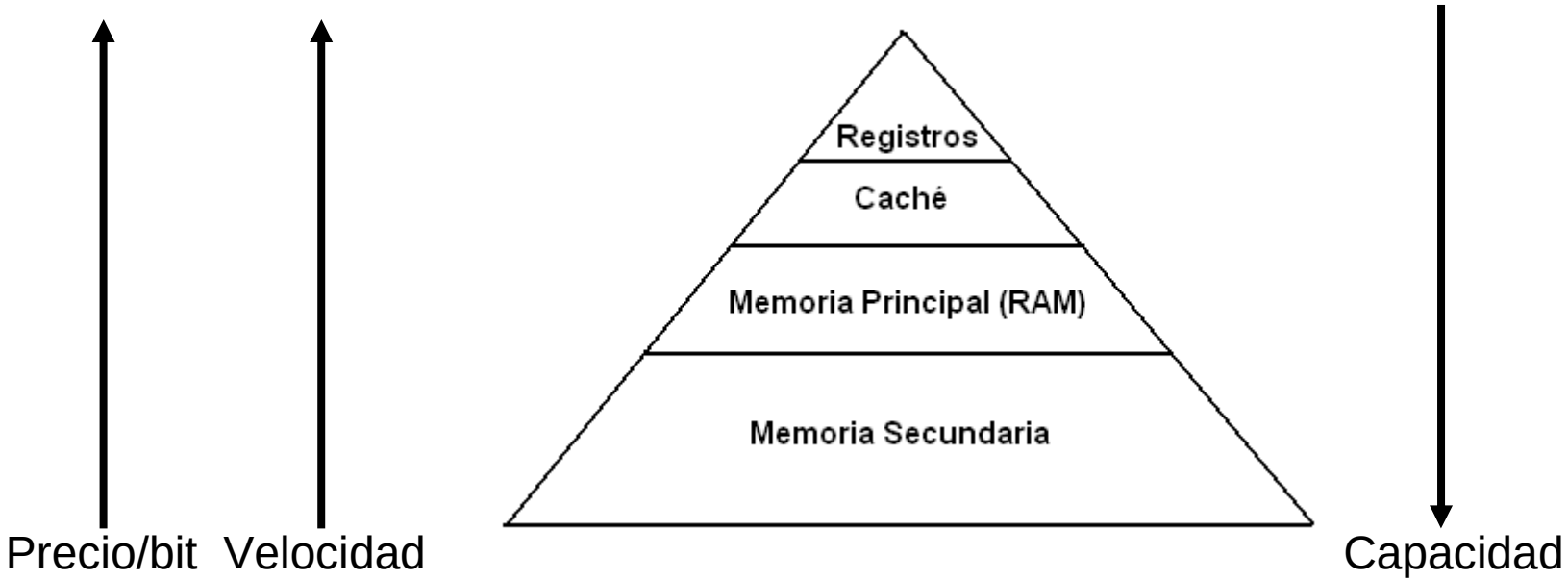
Jerarquía de Memoria

- El carpintero sólo tiene una o dos piezas de madera en su mesa.
- Cuando termina de trabajar con esas piezas, va a la tienda a comprar más madera.
  - Si la tienda tiene las piezas en stock, el carpintero sólo ha perdido un par de horas.
  - Si la tienda no tiene stock, hay que esperar a que las traiga el distribuidor.
    - Si el distribuidor tiene las piezas, perdemos uno o dos días.
    - Si no tiene las piezas, hay que fabricarlas y podemos perder varios días.

¿Cómo evitar estos retrasos? **¡¡Teniendo todas las piezas de la fábrica en la mesa!! ← evidentemente, es imposible…**

Jerarquía de Memoria

- La jerarquía de memoria habitual en los computadores se puede resumir en 4 niveles:



Registros

- Se encuentran físicamente dentro del procesador.
- Contienen los datos con los que el procesador está operando en un momento dado (A = **B** + **C**).
- Habitualmente son muy pocos (entre 1 y 32…).
- *Son las piezas de la mesa de trabajo.*

El programador normalmente no es consciente de que existen los registros. Son gestionados automáticamente por el compilador.

Memoria Principal

- Se encuentra físicamente cerca del procesador, pero es un módulo *hardware* diferente.
- Contienen todos los datos del programa que el procesador está ejecutando: **instrucciones + datos**.
- Es una memoria de acceso aleatorio (RAM): el tiempo de acceso a un dato es independiente de su ubicación.
- Los datos se encuentran en direcciones numeradas.
- El tamaño habitual en la actualidad es de unos pocos GBs.

Las variables que maneja el programador se almacenan en memoria principal.

Memoria Caché

- **Problema**: el acceso a la memoria principal es lento.
- **Solución**: tener un *stock* de piezas haciendo una previsión de las que podremos necesitar.
- La memoria caché es una memoria *inteligente* que predice qué datos puede necesitar el procesador en breve. Los datos que predice los va trayendo desde memoria principal sin que el procesador *se entere* y, cuando el procesador necesita ese dato, lo puede obtener de la caché a gran velocidad.

La caché se gestiona automáticamente a nivel de hardware.

- La caché funciona en base a dos principios:
  - **Localidad temporal:** cuando se accede a un dato es muy probable que ese dato se vuelva a utilizar en breve → la caché mantiene los datos solicitados por la CPU *en stock* durante un tiempo.
  - **Localidad espacial:** cuando se accede a un dato es muy probable que en breve se vaya a acceder a datos consecutivos → cuando la CPU solicita un dato que se encuentra en la dirección N, la caché traerá de memoria principal además las direcciones N+1, N+2…
    - Las instrucciones de un programa se almacenan secuencialmente.
    - Es muy habitual trabajar con datos almacenados en vectores (*arrays*), que se almacenan secuencialmente.

Memoria Caché

- El diseño de una caché implica tomar varias decisiones:
  - Organización: cómo se ubican los datos en la caché en relación a su localización en memoria.
  - Políticas de extracción y escritura: qué datos de memoria copiar a la caché y en qué momento. ← tratamiento de los **fallos**.
  - Política de reemplazo: qué dato borrar de la caché cuando no queda espacio libre.
  - Etc…
- En la actualidad, se utilizan varios niveles de caché (habitualmente 3) siguiendo el mismo principio jerárquico.
- Tamaños habituales en la actualidad (*ejemplo i7 quad*):
  - Caché L1: 4 x 2 x 32 KBs
  - Caché L2: 4 x 256 KBs
  - Caché L3: 8 MBs

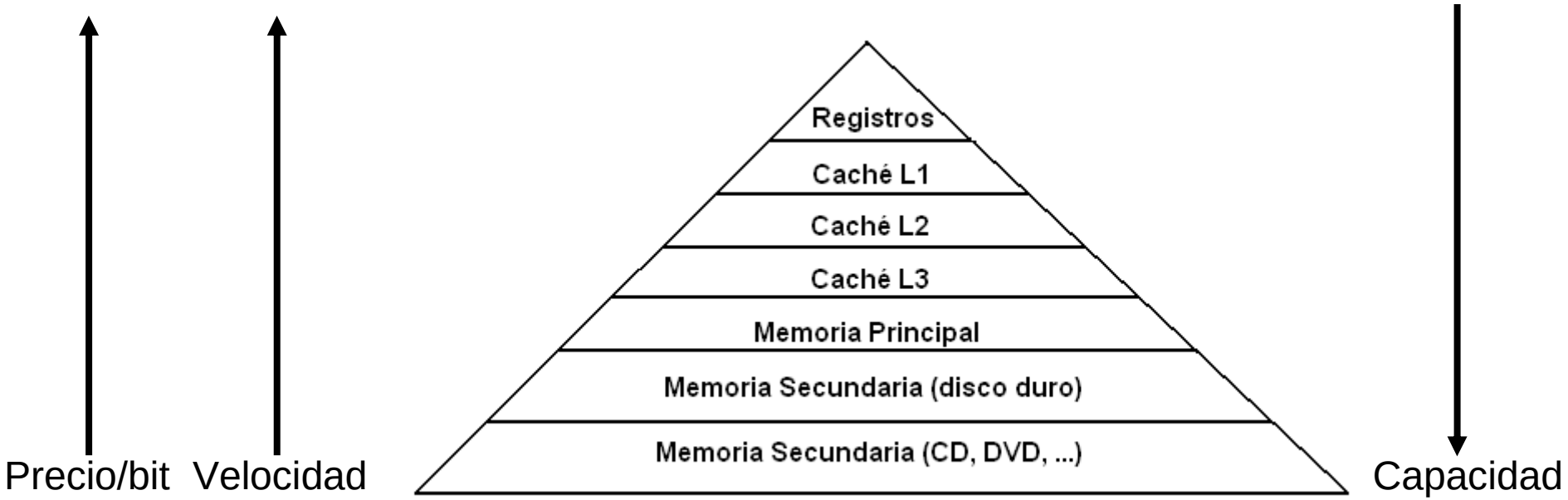
Memoria Secundaria

- Es el almacén masivo en el que se encuentran todos los posibles programas y datos con los que potencialmente podremos trabajar en algún momento.
- Cuando ejecutamos un nuevo programa, hay que moverlo desde memoria secundaria hacia memoria principal.
- También se puede dividir en varios niveles, aunque básicamente tenemos los dispositivos internos (discos duros) y los externos reemplazables (CD, DVD, SD, PenDrive…).
- Tamaños actuales: disco duro (1 TB), CD (700 MBs), DVD (4’7GBs), PenDrive (16 GBs)…

El acceso a memoria secundaria puede ser gestionado automáticamente por el sistema operativo, o bien bajo demanda por el usuario/programador.

Jerarquía de Memoria

- Con todo lo que hemos visto, tenemos una jerarquía de memoria más completa:



Curso 0: Arquitectura de Computadores

Carlos Garre

carlos.garre@urjc.es