Fundamentos de los sistemas informáticos

Arquitectura básica, procesamiento de instrucciones y codificación de datos

Josep Garcia

Sistemas Informáticos · 1 DAM

Curso 2026-2027Curso 2026-2027 — Sincronizado: 22/10/2026 11:19

Índice general

Introducción . . . 2
Objetivos . . . 3
Relación entre bloques, resultados de aprendizaje y criterios de evaluación . . . 3
1.1. Datos, información y operaciones . . . 5
1.2. Hardware, software, datos y usuarios . . . 6
1.3. Entrada, proceso, almacenamiento y salida . . . 7
1.4. Ejercicios de aplicación . . . 7
2.1. Arquitectura de Von Neumann . . . 9
2.2. Arquitectura Harvard . . . 10
2.3. Ejercicios de aplicación . . . 12
3.1. Buses y comunicación interna . . . 13
3.2. Bus de datos, direcciones y control . . . 14
3.3. Comunicación serie y paralela . . . 15
3.4. Entrada/salida y periféricos desde el punto de vista funcional . . . 15
3.5. Ejercicios de aplicación . . . 17
4.1. Función general de la CPU . . . 19
4.2. Componentes funcionales de la CPU . . . 19
4.2.1. Unidad de control, reloj y ALU . . . 20
4.2.2. Registros internos . . . 21
4.2.3. Bus interno de la CPU . . . 22
4.3. Ejercicios de aplicación . . . 24
5.1. De la CPU al almacenamiento: una jerarquía . . . 25
5.2. RAM: memoria principal, direcciones, bits y palabras . . . 26
5.3. Memoria caché: acelerar el acceso a RAM . . . 28
5.4. Almacenamiento secundario: HDD, SSD y flash . . . 29
5.5. Firmware y memorias no volátiles . . . 30
5.6. Ejercicios de aplicación . . . 31
6.1. Conjunto de instrucciones . . . 32
6.1.1. RISC y CISC . . . 32
6.2. Opcode y operandos . . . 33
6.3. Ciclo de instrucción . . . 34
6.4. Ejemplo guiado de ejecución . . . 34
6.5. Ejercicios de aplicación . . . 36
7.1. Sistemas de numeración: decimal, binario y hexadecimal . . . 37
7.2. Bits, bytes y valores representables . . . 37
7.3. Conversión básica entre decimal, binario y hexadecimal . . . 38

7.4. Unidades de información: SI e IEC . . . 40
7.5. Texto: ASCII, Unicode y UTF-8 . . . 41
7.6. Lógica booleana y tablas de verdad . . . 42
7.7. Ejercicios de aplicación . . . 43
8.1. Firmware BIOS/UEFI . . . 45
8.2. POST y comprobaciones iniciales . . . 45
8.3. Dispositivo de arranque y carga del sistema operativo . . . 46
8.4. Verificación inicial desde el sistema operativo . . . 46
8.5. Secuencia completa de puesta en marcha . . . 47
8.6. Ejercicios de aplicación . . . 48
9.1. Actividades de consolidación AULES . . . 50
9.1.1. Ejercicios de consolidación . . . 50
9.1.2. Mapa conceptual de la unidad . . . 51
9.1.3. Reto final: construye un mapa funcional del sistema . . . 53
9.2. Criterios de corrección . . . 54
10.1. Un sistema informático en una tienda . . . 55
10.2. Dos formas de organizar un ordenador . . . 55
10.3. ¿Cómo llega un dato desde la RAM hasta la CPU? . . . 56
10.4. Encuentra los errores sobre la CPU . . . 56
10.5. ¿Dónde se guarda cada cosa? . . . 56
10.6. El documento que desapareció . . . 57
10.7. Verdadero o falso, con explicación . . . 57
10.8. Un mismo número, tres representaciones . . . 57
10.9. Del teclado a la pantalla . . . 58
10.10. Cada registro tiene su trabajo . . . 58
10.11. ¿Qué ocurre si el dato no está en caché? . . . 59
10.12. Entrada, salida o ambas . . . 59
10.13. Revisa unas conversiones . . . 59
Soluciones . . . 60

Introducción

Vivimos rodeados de sistemas informáticos: ordenadores, teléfonos móviles, servidores, videoconsolas, relojes inteligentes, cajeros automáticos o dispositivos integrados en vehículos y electrodomésticos. Aunque pueden tener formas muy distintas, todos comparten una idea común: reciben datos, los procesan siguiendo instrucciones y producen información útil. En esta unidad vamos a estudiar esa base común, es decir, los elementos funcionales que permiten que un sistema informático trabaje.

Un sistema informático no es solo una máquina. Está formado por hardware, software, datos y personas usuarias. - El hardware aporta los componentes físicos. - El software indica qué operaciones deben realizarse. - Los datos son la materia prima que se procesa. - Las personas interpretan los resultados y deciden para qué se utiliza el sistema.

Comprender esta relación es fundamental antes de estudiar sistemas operativos, redes, almacenamiento o administración de equipos.

Para entender cómo funciona un ordenador por dentro, trabajaremos con una visión funcional. No nos centraremos todavía en montar componentes ni en identificar conectores físicos, ya que eso se desarrollará con más detalle en la UD02. En esta unidad veremos cómo se organizan la CPU, la memoria, los buses y los dispositivos de entrada/salida, cómo se ejecutan las instrucciones y qué ocurre durante la puesta en marcha inicial de un equipo.

También estudiaremos cómo representa la información un ordenador. Aunque las personas usamos números, letras, imágenes o sonidos, internamente el sistema trabaja con señales que se interpretan como ceros y unos. Por eso revisaremos el sistema binario, las unidades de información, la representación de datos y los fundamentos de la lógica booleana. El objetivo es que puedas explicar no solo qué componentes existen, sino cómo cooperan para transformar datos en información.

Objetivos

Al finalizar esta unidad, serás capaz de:

- Identificar los elementos funcionales de un sistema informático y explicar la relación entre hardware, software, datos y personas usuarias.
- Diferenciar las arquitecturas de Von Neumann y Harvard, reconociendo cómo organizan la memoria, la CPU y los mecanismos de comunicación.
- Describir la función de la CPU, la unidad de control, la ALU, los registros y el reloj dentro de la ejecución de instrucciones.
- Clasificar los principales tipos de memoria y comparar sus características básicas, prestaciones y función dentro del sistema.
- Explicar cómo se comunican los componentes de un ordenador mediante buses y dispositivos de entrada/salida, sin entrar todavía en el montaje físico.
- Interpretar el ciclo básico de instrucción de una CPU: búsqueda, decodificación, ejecución y almacenamiento del resultado.
- Explicar cómo se codifican cantidades y texto mediante binario, hexadecimal, unidades de información y UTF-8, relacionándolo con el funcionamiento del sistema.
- Verificar de forma básica el proceso de puesta en marcha de un equipo, identificando el papel de BIOS/UEFI, POST y las primeras comprobaciones del sistema.

Relación entre bloques, resultados de aprendizaje y criterios de evaluación

Cada bloque aporta una parte del recorrido de la unidad. Los criterios se comprueban especialmente en las actividades y prácticas indicadas durante el desarrollo y en la consolidación final.

Bloque	Qué se trabaja	RA y criterio de evaluación relacionado
01. El sistema informático	Elementos funcionales y relación entre hardware, software, datos y personas usuarias.	**RA2.a** — Identificación de los elementos funcionales de un sistema informático.
02. Arquitecturas	Organización de CPU, memoria, entrada/salida y buses en los modelos Von Neumann y Harvard.	**RA2.a** — Identificación de los elementos funcionales de un sistema informático.
03. Comunicación interna	Componentes físicos, buses e interconexión básica entre CPU, memoria y entrada/salida.	**RA1.a** — Reconocimiento de componentes físicos y sus mecanismos de interconexión.**RA2.a** — Identificación de los elementos funcionales de un sistema informático.
04. La CPU	Componentes funcionales de la CPU y su relación con el resto del sistema.	**RA1.a** — Reconocimiento de componentes físicos y sus mecanismos de interconexión.**RA2.a** — Identificación de los elementos funcionales de un sistema informático.
05. Memoria y almacenamiento	Tipos de memoria, prestaciones, volatilidad y función en el conjunto del sistema.	**RA1.b** — Clasificación de tipos de memorias, características, prestaciones y función.
06. Instrucciones	Conjunto de instrucciones y ciclo de búsqueda, decodificación, ejecución y almacenamiento.	**RA2.a** — Identificación de los elementos funcionales de un sistema informático.
07. Representación de la información	Binario, hexadecimal, unidades de información, UTF-8 y lógica booleana básica.	**Saberes de apoyo** — Facilitan interpretar datos e instrucciones; no constituyen una evidencia directa de un criterio de esta unidad.

Bloque	Qué se trabaja	RA y criterio de evaluación relacionado
08. Puesta en marcha	Firmware, POST, dispositivo de arranque y comprobaciones iniciales.	**RA1.c** — Verificación del proceso de puesta en marcha de un equipo.**RA2.a** — Identificación de los elementos funcionales implicados en el arranque.
09. Actividades de consolidación	Entrega integrada de Aules: ejercicios, mapa conceptual y mapa funcional.	**RA1.b** y **RA2.a** — Evidencia calificable mediante una nota única. **RA1.a** como apoyo; **RA1.c** requiere verificación observada en el bloque 08.

10. Actividades Repaso | Trece ejercicios formativos de elementos funcionales, CPU, buses, memoria, periféricos y conversiones. | **RA2.a** y **RA1.b**; **RA1.a** como repaso conceptual y apoyo, sin verificación física. Conversiones: **saberes de apoyo**. Sin nota. |

También puedes consultar la presentación de clase.

Antes de estudiar componentes concretos, conviene entender qué problema resuelve un sistema informático: transformar datos en información útil mediante operaciones automáticas.

1.1. Datos, información y operaciones

Para entender qué hace un sistema informático, conviene diferenciar entre dato e información.

Un **dato** es una representación formal de algo. Por ejemplo, `30` es un dato: por sí solo no sabemos si representa grados, euros, años, alumnos o kilómetros. La **información** aparece cuando ese dato se interpreta dentro de un contexto. Si sabemos que `30` representa una temperatura en grados Celsius, podemos concluir que hace calor.

Entre el dato y la información hay una **operación** o regla de interpretación. Por ejemplo:

```
Dato: 30
Regla: si la temperatura es mayor que 23 °C, entonces hace calor.
Información: hace calor.
```

Los sistemas informáticos trabajan aplicando operaciones sobre datos. Algunas operaciones son aritméticas, como sumar o multiplicar. Otras son lógicas, como comparar dos valores o decidir si una condición se cumple. Gracias a esta capacidad, un ordenador puede transformar datos de entrada en resultados útiles.

1.2. Hardware, software, datos y usuarios

Un sistema informático es un conjunto de elementos que trabajan de forma coordinada para tratar información de manera automática. No se trata solo de un ordenador aislado, sino de una combinación de componentes físicos, programas, datos y personas que permiten resolver problemas muy distintos.

Una calculadora sencilla, por ejemplo, está pensada para hacer operaciones concretas. Un sistema informático, en cambio, puede ejecutar programas diferentes: escribir documentos, navegar por Internet, editar imágenes, gestionar una base de datos o ejecutar un videojuego. La clave está en que el comportamiento del sistema depende del software que se ejecuta sobre el hardware disponible.

En un sistema informático podemos distinguir cuatro elementos principales:

- **Hardware:** conjunto de componentes físicos del sistema, como procesador, memoria, almacenamiento, pantalla, teclado o interfaces de comunicación.
- **Software:** conjunto de programas e instrucciones que indican al hardware qué debe hacer.
- **Datos:** representaciones de hechos, valores o mensajes que el sistema almacena, procesa o transmite.
- **Personas usuarias:** quienes utilizan, administran, diseñan o mantienen el sistema para alcanzar un objetivo.

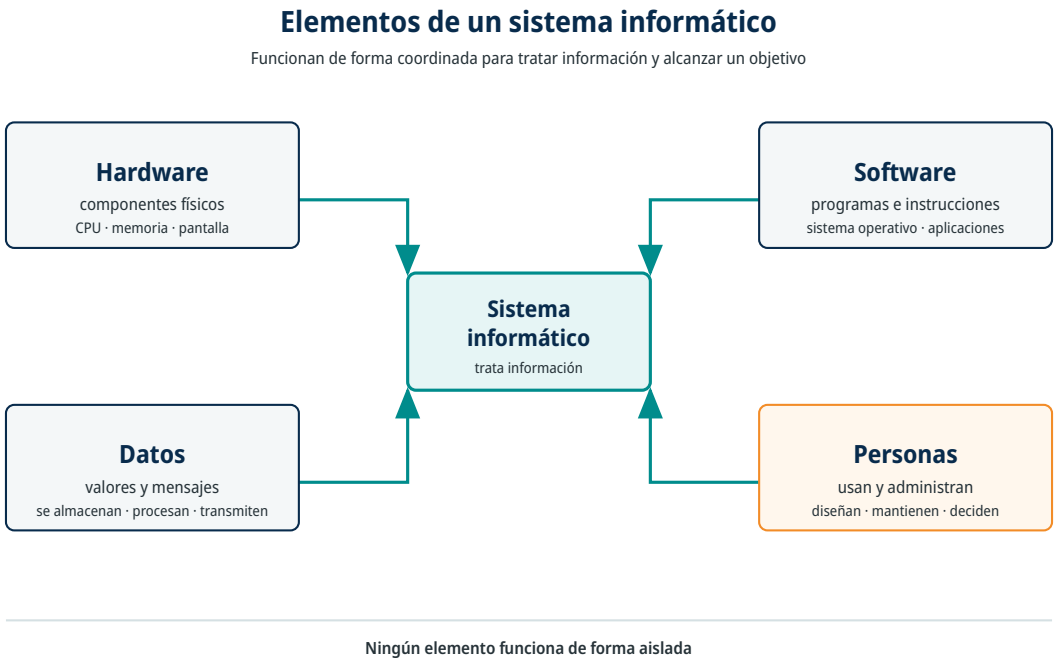


Figura 1: Relación entre hardware, software, datos y personas usuarias en un sistema informático



Un sistema informático no es útil solo por tener buenos componentes. Su valor aparece cuando hardware, software, datos y personas trabajan de forma coherente.

1.3. Entrada, proceso, almacenamiento y salida

Una forma sencilla de entender cualquier sistema informático es observar el recorrido de la información:

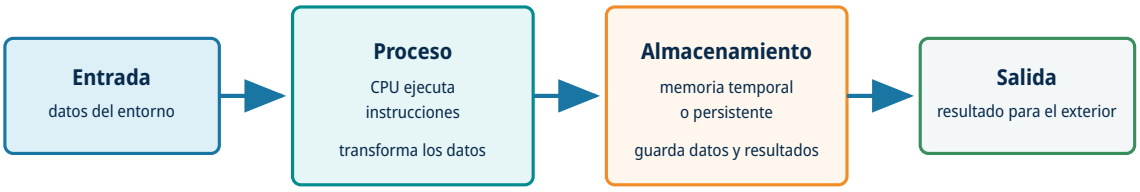
1. **Entrada:** el sistema recibe datos desde una persona, otro equipo, un sensor, un archivo o un periférico.
2. **Proceso:** la CPU ejecuta instrucciones que transforman esos datos.
3. **Almacenamiento:** los datos y resultados se guardan temporalmente en memoria o de forma persistente en almacenamiento secundario.
4. **Salida:** el sistema muestra, transmite o deja disponible el resultado.

Este esquema aparece una y otra vez a lo largo de la unidad. Primero veremos cómo se organiza el ordenador para permitir ese recorrido; después estudiaremos qué papel cumplen la CPU, la memoria, los buses y la representación binaria.

Antes de seguir, conviene adelantar una idea que retomaremos con detalle en el apartado 7: todo dato que circula por estos bloques se representa mediante bits, valores `0` y 1. El sistema hexadecimal permite escribir grupos de cuatro bits de una forma más breve; por eso aparecerá más adelante al hablar de direcciones, registros y memoria.

Recorrido de la información

El sistema transforma datos de entrada en resultados de salida



Ejemplo: pulsar una tecla → procesar la orden → guardar o actualizar → mostrar el resultado



Si puedes explicar un ejemplo cotidiano siguiendo entrada, proceso, almacenamiento y salida, ya tienes el hilo principal de la unidad.

1.4. Ejercicios de aplicación



Los ejercicios se organizan en aplicación guiada, práctica y ampliación, según se indica a continuación.

Para trabajar este bloque, resuelve los ejercicios 1 a 3 con guía del docente y contraste colectivo; utiliza los ejercicios 4 y 5 como aplicación con apoyo de los apuntes; reserva los ejercicios 6 a 8 para trabajo autónomo o de ampliación.

1. **Dato o información.** Clasifica como dato o información e indica qué contexto falta o aporta significado: `27`, `27 GB libres`, `ERROR 404` y `La página solicitada no existe`.
2. **Relaciona conceptos.** Clasifica CPU, navegador web, memoria RAM, contraseña, persona administradora y SSD como hardware, software, dato o persona usuaria.
3. **Tipo test.** Elige la opción correcta y justifica brevemente:
   - ¿Cuál es software? a) teclado, b) sistema operativo, c) memoria RAM, d) monitor.
   - ¿Qué fase representa guardar un documento en un SSD? a) entrada, b) proceso, c) almacenamiento, d) salida.
   - ¿Cuándo un dato se convierte en información? a) cuando ocupa más memoria, b) cuando se interpreta en un contexto, c) cuando se guarda en RAM, d) cuando se escribe en binario.
4. **El viaje de una pulsación.** Abre un editor de texto, pulsa la tecla A, observa el carácter en pantalla y guarda el archivo como `nota.txt`. Dibuja seis flechas y explica qué papel tienen teclado, sistema operativo o editor, CPU, RAM, SSD y pantalla. Indica en qué momento el texto deja de estar solo en memoria temporal.

   Posible recorrido:

   1. El **teclado** detecta la pulsación y envía una señal que representa la tecla A.
   2. El **sistema operativo**, mediante el controlador del teclado, recibe e interpreta esa señal.
   3. El **editor de texto** recibe el evento y solicita procesarlo.
   4. La **CPU** ejecuta las instrucciones necesarias y la **RAM** conserva temporalmente el carácter dentro del documento abierto.
   5. La **pantalla** muestra la A como resultado visible para la persona usuaria.
   6. Al guardar, el editor y el sistema operativo escriben el contenido en el **SSD** con el nombre `nota.txt`.

   El texto deja de estar solo en memoria temporal cuando la escritura en el SSD se completa: entonces puede recuperarse después de cerrar el editor o apagar el equipo.
5. **Caja negra: préstamo de portátiles.** Un sistema registra préstamos de aula. Observa estas situaciones:

   ```
   Alumno A17 + portátil P03 + 09:15 -> "Préstamo registrado"
   Alumno A17 + portátil P03 + 09:40 -> "Equipo no disponible"
   Alumno X00 + portátil P07 + 09:50 -> "Identificación no válida"
   ```

   Imagina que eres quien debe explicar al equipo de desarrollo cómo funciona esta aplicación. No necesitas programarla ni diseñar una base de datos: debes describir el recorrido de la información.

   Completa un esquema como este:

Parte del sistema	Qué debes indicar
Entrada	Los datos que se introducen en cada intento de préstamo.
Proceso	Las comprobaciones que hace el sistema antes de decidir: por ejemplo, si reconoce al alumno o si el portátil se puede prestar.
Almacenamiento	La información que debe recordar para seguir funcionando mañana, incluso después de apagar el equipo.
Salida	El mensaje o resultado que recibe la persona que solicita el portátil.

Después, explica con una frase por qué cambian los mensajes en el segundo y el tercer caso. Fíjate en que el primer intento cambia el estado de `P03`: deja de estar disponible y pasa a estar prestado. Esa información no puede olvidarse al cerrar la aplicación; debe quedar guardada para poder comprobar el siguiente préstamo.

6. **Ampliación: sistema virtualizado.** Una empresa utiliza una máquina virtual con 4 CPU virtuales, 8 GB de RAM virtual y un disco virtual de 100 GB. Explica qué elementos cumplen la función de CPU, memoria y almacenamiento, qué significa que sean virtuales y qué componente administra esos recursos sobre el hardware físico.
7. **Ampliación: control de acceso de aula.** Diseña el funcionamiento de un sistema que permite entrar en un aula informática mediante una tarjeta de estudiante. Tu entrega debe incluir una tabla con dos datos de entrada, tres comprobaciones que realiza el sistema, los datos que guarda, las salidas o acciones posibles y el papel de la persona administradora. Después describe el recorrido completo de una entrada autorizada en seis pasos y explica un caso en que un dato incorrecto produzca una decisión de acceso equivocada.
8. **Ampliación: incidencia con evidencias.** Una aplicación confirma en pantalla que una compra se ha realizado, pero el pedido no aparece en el historial al volver a abrirla. Formula una hipótesis para entrada, proceso, almacenamiento y salida. Para cada una, indica una evidencia que pedirías antes de atribuir el error a ese punto.

Una vez entendido el recorrido general de la información, necesitamos un modelo para explicar cómo se organizan los bloques internos del ordenador.

2.1. Arquitectura de Von Neumann

La arquitectura de Von Neumann es un modelo teórico que describe cómo se organiza un ordenador programable. Su importancia está en la idea de **programa almacenado**: las instrucciones del programa

y los datos se guardan en memoria, de modo que el mismo hardware puede ejecutar tareas diferentes cambiando el programa cargado.

Según este modelo, un ordenador se organiza alrededor de cuatro bloques funcionales:

- **CPU:** interpreta y ejecuta instrucciones.
- **Memoria principal:** almacena instrucciones y datos mientras se están usando.
- **Entrada/salida:** permite comunicarse con el exterior.
- **Buses:** transportan información entre los bloques anteriores.

La idea básica es sencilla: la CPU obtiene una instrucción desde memoria, la interpreta, ejecuta la operación necesaria y guarda o comunica el resultado. Este proceso se repite continuamente mientras el sistema está funcionando.

La arquitectura de Von Neumann ha sido la base de la mayoría de ordenadores modernos. Aunque los equipos actuales incorporan muchas mejoras, este modelo sigue siendo útil para entender cómo se relacionan los componentes principales.

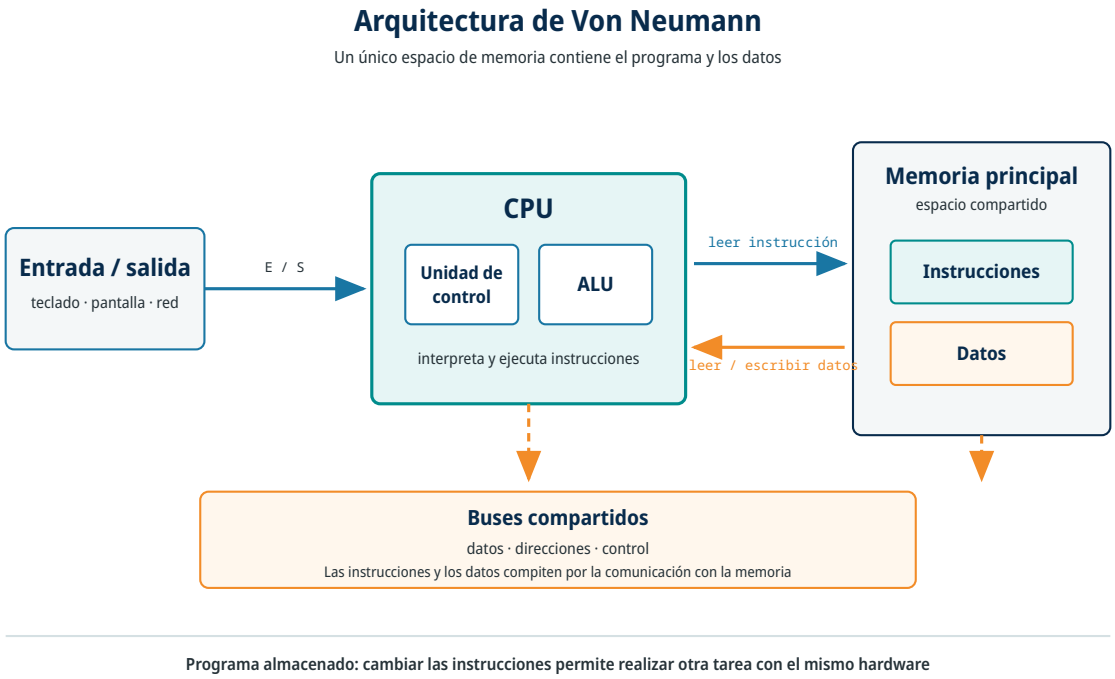


Figura 2: Arquitectura de Von Neumann con programa almacenado, memoria compartida y buses

2.2. Arquitectura Harvard

La arquitectura Harvard es otro modelo de organización de un ordenador. Su diferencia principal respecto a Von Neumann es que separa la memoria de instrucciones y la memoria de datos. Esto permite acceder a una instrucción y a un dato al mismo tiempo, lo que puede mejorar el rendimiento.

La comparación básica es la siguiente:

Aspecto	Von Neumann	Harvard
Memoria	Datos e instrucciones comparten memoria	Datos e instrucciones usan memorias separadas
Buses	Puede compartir canales de comunicación	Puede usar buses separados
Ventaja principal	Diseño más simple y flexible	Mayor velocidad en ciertos usos
Inconveniente principal	Puede aparecer cuello de botella	Diseño más complejo y costoso
Uso típico	Ordenadores de propósito general	Microcontroladores, sistemas embebidos o diseños especializados

En la práctica, muchos procesadores actuales no siguen un modelo puro. Es habitual encontrar diseños híbridos que combinan ideas de ambos modelos, por ejemplo usando una organización general parecida a Von Neumann pero con cachés separadas para instrucciones y datos.

Este enfoque suele denominarse **arquitectura Harvard modificada**. La memoria principal continúa siendo compartida para datos e instrucciones, como en Von Neumann, pero la CPU puede disponer de cachés L1 separadas para cada uno. Así puede obtener una instrucción y acceder a datos con menos interferencias, sin renunciar a una memoria principal común. Esta es la idea que conviene investigar al comparar ambos modelos, no una tercera arquitectura completamente independiente.

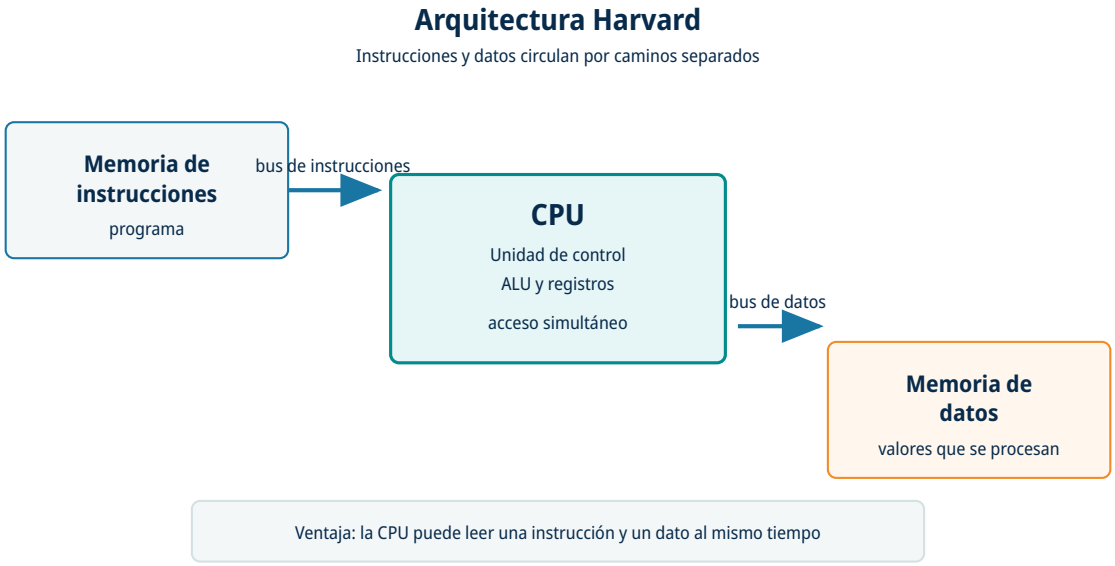


Figura 3: Arquitectura Harvard con memorias y buses separados para instrucciones y datos



Si puedes explicar dónde se guardan datos e instrucciones en cada arquitectura, ya tienes la idea clave para entender por qué se comportan de forma diferente.

2.3. Ejercicios de aplicación

Para trabajar este bloque, resuelve los ejercicios 1 a 3 con guía del docente y contraste colectivo. El ejercicio 4 es de ampliación: resuélvelo de forma autónoma, apoyándote en los apuntes para justificar la decisión de diseño.

1. **Identifica la arquitectura.** Para cada descripción, escribe `Von Neumann` o `Harvard` y añade una palabra clave que justifique la decisión. No te fijes en el nombre del dispositivo: busca si **datos e instrucciones** comparten o separan memoria y caminos de acceso.

   a) Datos e instrucciones comparten memoria.
   b) Existen memorias separadas para datos e instrucciones.
   c) El diseño prioriza la simplicidad y flexibilidad.
   d) Puede accederse simultáneamente a un dato y a una instrucción.

2. **Recorrido de una instrucción.** Imagina que la memoria contiene la instrucción “sumar dos valores”. Redacta una secuencia de tres o cuatro pasos con este orden: 1) la CPU localiza y lee la instrucción en memoria; 2) la interpreta; 3) realiza la operación; 4) deja el resultado en un lugar que pueda usarse después. Puedes usar las palabras *CPU*, *memoria*, *instrucción*, *dato* y *resultado*. Todavía no necesitas nombrar registros concretos.

   Solución: Posible recorrido:

   1. La **CPU** localiza en la **memoria** la siguiente instrucción que debe ejecutar.
   2. Lee esa instrucción y la interpreta para saber qué operación se le está pidiendo, por ejemplo sumar dos valores.
   3. Obtiene los datos necesarios, realiza la operación y genera un resultado.
   4. Guarda el resultado en memoria o en un espacio temporal de la CPU para que la siguiente instrucción pueda utilizarlo.

   El ciclo vuelve a empezar con la siguiente instrucción del programa. En los apartados dedicados a la CPU veremos cómo intervienen registros concretos como el PC o el IR.

3. **Tipo test: el atasco de Von Neumann.** Marca la opción correcta y justifica tu elección con una comparación cotidiana de una sola frase. Piensa en una carretera de un único carril: ¿qué ocurre si deben pasar a la vez un dato y una instrucción por el mismo camino?

   “El posible cuello de botella de Von Neumann aparece principalmente porque…”

   a) la CPU no usa memoria
   b) datos e instrucciones comparten caminos de comunicación
   c) Harvard no tiene CPU
   d) los programas no se guardan.

4. **Ampliación: argumenta una decisión.** Un fabricante diseña dos modelos de controlador para una máquina:

   - **Modelo A:** datos e instrucciones usan la misma memoria y el mismo camino de comunicación. Es más sencillo de construir.
   - **Modelo B:** datos e instrucciones tienen accesos separados. Es más complejo de construir.

   Completa una tabla con estas columnas: `Modelo`, qué `ventaja aporta`, qué `inconveniente tiene` y cuándo `elegirlo`. En cada fila explica la causa: no basta con escribir "A es más simple" o "B es más rápido". Por ejemplo, razona qué sucede cuando la CPU necesita leer una instrucción y un dato a la vez, o por qué añadir caminos separados puede encarecer y complicar el diseño.

Modelo	Qué ventaja aporta	Qué inconveniente tiene	Cuándo elegirlo
A: memoria y camino compartidos			
B: accesos separados			

   Termina recomendando uno de los dos modelos para un dispositivo sencillo que solo realiza unas pocas tareas repetitivas y otro para un sistema donde interesa reducir esperas entre instrucciones y datos. Justifica ambas recomendaciones con una idea técnica de la tabla.

Los bloques de una arquitectura no trabajan aislados. Para que la CPU, la memoria y los dispositivos colaboren, la información debe moverse por canales de comunicación.

3.1. Buses y comunicación interna

Un **bus** es un sistema de comunicación que permite transferir información entre los componentes del ordenador. Conecta bloques como CPU, memoria y entrada/salida para que puedan intercambiar datos, direcciones y señales de control.

Una forma sencilla de imaginarlo es pensar en una carretera interna por la que circula información. Si esa carretera es más ancha o permite más transferencias por segundo, podrá mover más información en menos tiempo.

Dos características importantes de un bus son:

- **Ancho del bus:** cantidad de bits que puede transmitir al mismo tiempo.
- **Frecuencia del bus:** número de transferencias que puede realizar por segundo.

El rendimiento de un bus no depende solo de una de estas características, sino de la combinación entre ambas.

3.2. Bus de datos, direcciones y control

En una arquitectura funcional como la de Von Neumann, se distinguen tres tipos de buses según la información que transportan:

Bus	Qué transporta	Para qué sirve
Bus de datos	Datos e instrucciones	Lleva la información que se lee, escribe o procesa
Bus de direcciones	Direcciones de memoria o de dispositivos	Indica dónde hay que leer o escribir
Bus de control	Señales de coordinación	Indica si se debe leer, escribir, esperar, interrumpir o sincronizar una operación

Por ejemplo, si la CPU necesita leer un dato de memoria, el bus de direcciones indica la posición que se quiere consultar, el bus de control indica que se trata de una lectura y el bus de datos transporta el contenido leído.

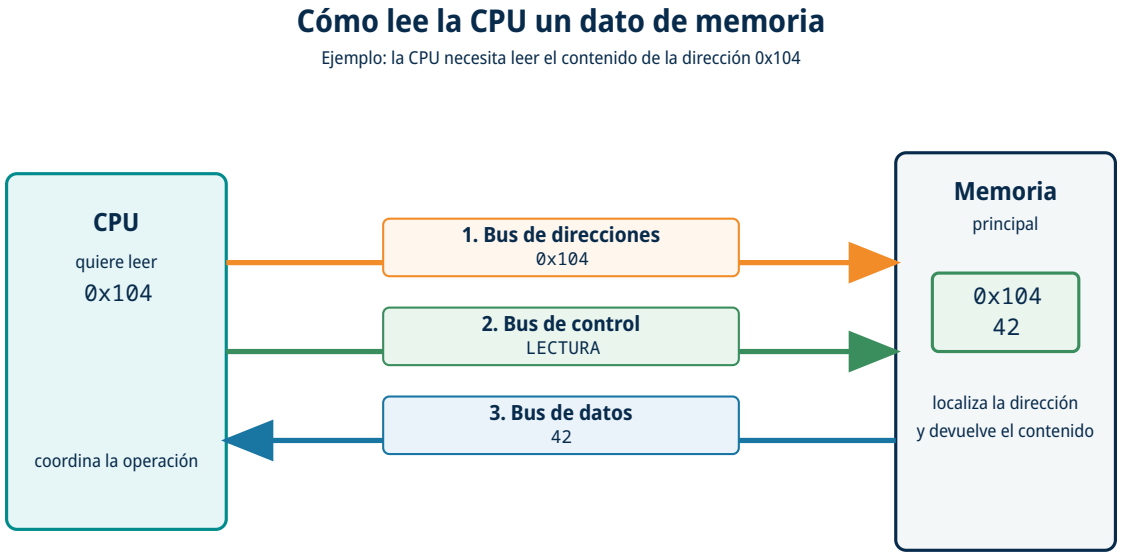


Figura 4: La CPU solicita la dirección 0x104 y recibe el valor 42 mediante los buses del sistema

El proceso puede describirse así:

1. La CPU coloca la dirección `0x104` en el **bus de direcciones** para indicar qué posición de memoria necesita consultar.

2. La CPU envía por el **bus de control** la señal LECTURA, que comunica a la memoria la operación solicitada.
3. La memoria localiza la posición `0x104`, cuyo contenido es el valor 42 en este ejemplo.
4. La memoria coloca 42 en el **bus de datos** y la CPU recibe ese valor para poder utilizarlo en la instrucción que está ejecutando.



Los buses de datos, direcciones y control son un modelo funcional muy útil para entender cómo coopera el sistema. En equipos actuales, muchas conexiones físicas no son buses paralelos compartidos: PCI Express, por ejemplo, utiliza enlaces serie punto a punto para conectar dispositivos de expansión como tarjetas gráficas o unidades NVMe. Estudiaremos su instalación y sus conectores en la UD02.

3.3. Comunicación serie y paralela

Los buses pueden transmitir información de forma **paralela** o **serie**.

En una comunicación **paralela**, varios bits viajan al mismo tiempo por líneas diferentes. Puede parecer más rápido, pero cuando se trabaja a velocidades altas aparecen problemas de sincronización e interferencias.

En una comunicación **serie**, los bits viajan uno detrás de otro. Aunque parezca más lenta, permite alcanzar frecuencias muy altas y simplifica el diseño físico. Por eso muchas tecnologías modernas utilizan comunicaciones serie.



No conviene identificar “paralelo” con “siempre más rápido”. En sistemas modernos, muchas comunicaciones serie superan a buses paralelos antiguos gracias a su mayor frecuencia y mejor control de interferencias.

3.4. Entrada/salida y periféricos desde el punto de vista funcional

Para que un sistema informático sea útil, debe poder comunicarse con el exterior. Esa comunicación se realiza mediante el subsistema de **entrada/salida**, también llamado **E/S**.

Los dispositivos de entrada/salida permiten introducir datos en el sistema, mostrar resultados o intercambiar información con otros equipos. Cuando un dispositivo no forma parte del núcleo funcional del ordenador, formado principalmente por CPU y memoria principal, lo llamamos **periférico**.

Según su función, los periféricos pueden clasificarse así:

Tipo de periférico	Función	Ejemplos
Entrada	Introducen datos o instrucciones en el sistema	Teclado, ratón, micrófono, escáner
Salida	Muestran o transmiten información hacia el exterior	Monitor, impresora, altavoces
Entrada/salida	Pueden enviar y recibir información	Pantalla táctil, unidad de almacenamiento externa
Almacenamiento	Conservan datos para usarlos más tarde	SSD externo, memoria USB
Comunicación	Permiten conectar el equipo con otros sistemas	Tarjeta de red, adaptador Wi-Fi, Bluetooth

Un periférico es mucho más lento que la CPU. Para coordinarse con él, el sistema puede usar **encuesta** (*polling*), cuando la CPU pregunta periódicamente si ya hay una tarea lista; **interrupciones**, cuando el dispositivo avisa a la CPU de que necesita atención; o **DMA** (*acceso directo a memoria*), cuando un controlador transfiere bloques entre el dispositivo y la memoria con poca intervención directa de la CPU. En esta unidad basta con reconocer la función de cada mecanismo; su configuración y uso desde el sistema operativo se verán más adelante.



En esta unidad clasificamos los periféricos por su función. La instalación, conexión física y configuración detallada se trabajarán más adelante.

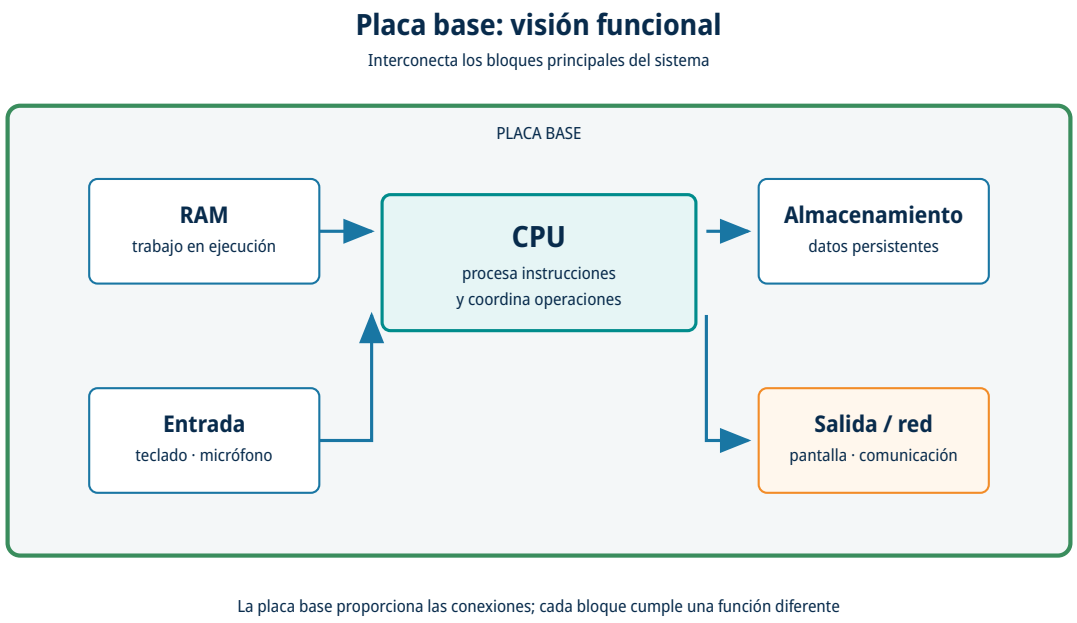


Figura 5: Placa base con CPU, RAM, almacenamiento e interfaces identificables

3.5. Ejercicios de aplicación

Para trabajar este bloque, resuelve los ejercicios 1 a 3 con guía del docente y contraste colectivo. Los ejercicios 4 a 6 requieren aplicar el modelo a una situación más abierta; el 6 puede realizarse como ampliación.

1. **Completa la secuencia.** La CPU quiere leer el contenido de una posición de memoria. Ordena estas acciones:

   a) la memoria entrega el contenido.
   b) la CPU indica que realizará una lectura.
   c) la CPU señala la dirección que necesita
   d) el dato viaja hasta la CPU.

2. **Serie o paralelo.** Clasifica estas afirmaciones como comunicación serie o paralela: a) varios bits usan líneas distintas al mismo tiempo; b) los bits viajan uno detrás de otro; c) puede sufrir desajustes entre líneas a frecuencias altas; d) es la base de muchas conexiones modernas de alta velocidad.

3. **Tipo test.** ¿Cuál es la afirmación correcta? a) Todo bus moderno es paralelo; b) la comunicación serie siempre es más lenta; c) una comunicación serie puede alcanzar gran velocidad; d) el bus de control transporta exclusivamente datos de usuario. Justifica tu elección.

4. **Ampliación: modelo y realidad.** El modelo de buses de datos, direcciones y control permite explicar **qué función cumple cada comunicación**: transportar un valor, señalar dónde se actúa y coordinar una operación. Sin embargo, en un equipo actual no debemos imaginar necesariamente tres grupos de cables paralelos compartidos por todos los componentes.

   PCI Express (PCIe), por ejemplo, utiliza enlaces serie punto a punto: cada dispositivo se comunica con el sistema mediante su propio enlace de alta velocidad. Esta organización cambia la forma física de transmitir la información y evita algunas limitaciones de los buses compartidos tradicionales, pero no elimina la necesidad de transportar datos ni de coordinar operaciones.

   Explica con tus palabras por qué el modelo de buses sigue siendo útil para aprender, aunque PCIe use enlaces serie punto a punto. Tu respuesta debe distinguir entre el **modelo funcional** (qué se comunica) y la **implementación física** (cómo se conecta realmente).

   Posible respuesta: El modelo de buses sigue siendo útil porque permite distinguir la información que necesita intercambiar un sistema: datos, direcciones y señales de control. PCIe no usa necesariamente un bus paralelo compartido como los modelos clásicos, sino enlaces serie independientes entre dispositivos. Aun así, la CPU y los dispositivos siguen necesitando enviar datos y coordinar lecturas, escrituras y transferencias. Por tanto, el modelo funcional ayuda a entender la finalidad de la comunicación, aunque el hardware moderno la implemente de otra manera.

5. **Investigación: la cámara se queda esperando.** Una cámara envía datos con rapidez a un equipo, pero guardarlos tarda mucho. Completa una tabla con tres zonas funcionales que revisarías, una posible causa en cada una y una evidencia que permitiría comprobarla. No atribuyas el problema directamente al SSD: puede estar en la entrada de la cámara, la RAM, la CPU o el almacenamiento.

Zona funcional que revisar	Posible causa	Evidencia que buscarías
Entrada de la cámara	La transferencia llega con errores o con una tasa irregular	Tasa de captura, mensajes de error o prueba con otro cable o puerto

6. **Ampliación: imprimir desde USB.** Diseña un diagrama funcional del recorrido de un documento desde una memoria USB hasta una impresora. Incluye al menos memoria USB, interfaz de entrada/salida, CPU, RAM e impresora. Usa flechas y etiquetas breves para indicar qué sucede en cada paso: lectura del archivo, preparación temporal del documento y envío de datos a la impresora.

Después de ver cómo se conectan los bloques, nos centramos en el componente que ejecuta instrucciones y coordina el procesamiento de datos.

4.1. Función general de la CPU

La **unidad central de proceso** o **CPU** es el componente encargado de interpretar y ejecutar las instrucciones de los programas. Coordina el trabajo del sistema y realiza operaciones sobre los datos que recibe desde la memoria o desde otros componentes.

Se dice que la CPU es el "cerebro" del ordenador. La comparación ayuda, pero conviene matizarla: la CPU no piensa ni comprende la información como una persona. Su trabajo consiste en seguir instrucciones muy simples, una detrás de otra, a gran velocidad.



La potencia de una CPU no depende solo de su velocidad de reloj. También influyen su arquitectura, el número de núcleos, la memoria caché, el conjunto de instrucciones y la forma en que se comunica con el resto del sistema.

4.2. Componentes funcionales de la CPU

Los componentes de la CPU se especializan en coordinar instrucciones, conservar valores temporales, realizar operaciones y comunicar los bloques internos. Entenderlos como un conjunto permite seguir el recorrido de una instrucción sin tratar cada pieza de forma aislada.

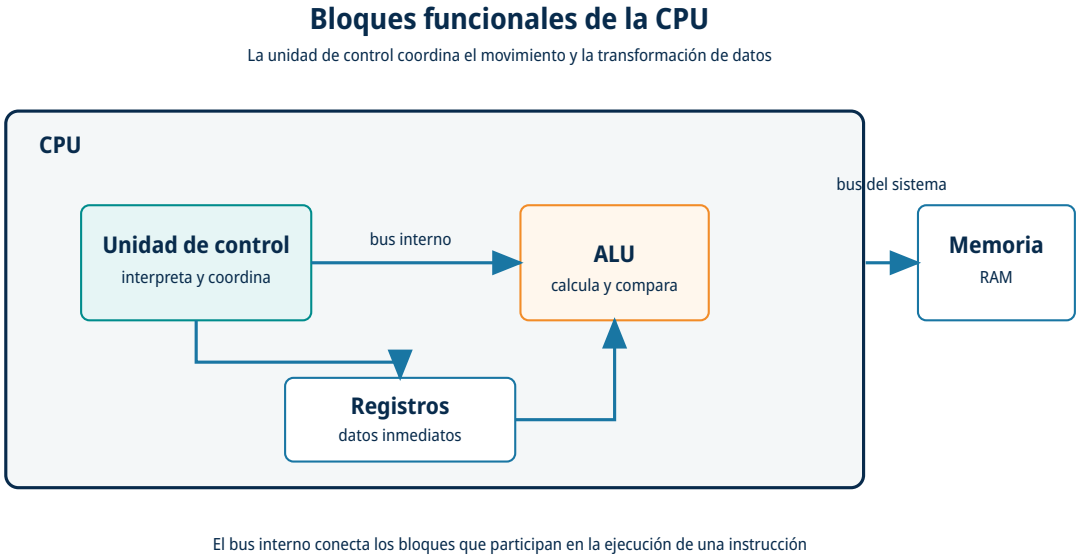


Figura 6: Bloques funcionales internos de una CPU

4.2.1. Unidad de control, reloj y ALU

La **unidad de control** se encarga de dirigir el funcionamiento interno de la CPU. Interpreta las instrucciones y genera las señales necesarias para que cada parte del procesador actúe en el momento adecuado.

El **reloj** marca el ritmo de trabajo. Cada ciclo de reloj permite sincronizar operaciones internas de la CPU. Su frecuencia se mide en hercios, normalmente en gigahercios en los procesadores actuales.

La **unidad aritmético-lógica**, también llamada **ALU** o **ULA**, realiza operaciones matemáticas y lógicas. Por ejemplo:

- sumas y restas.
- comparaciones entre valores.
- operaciones lógicas como AND, OR o NOT.

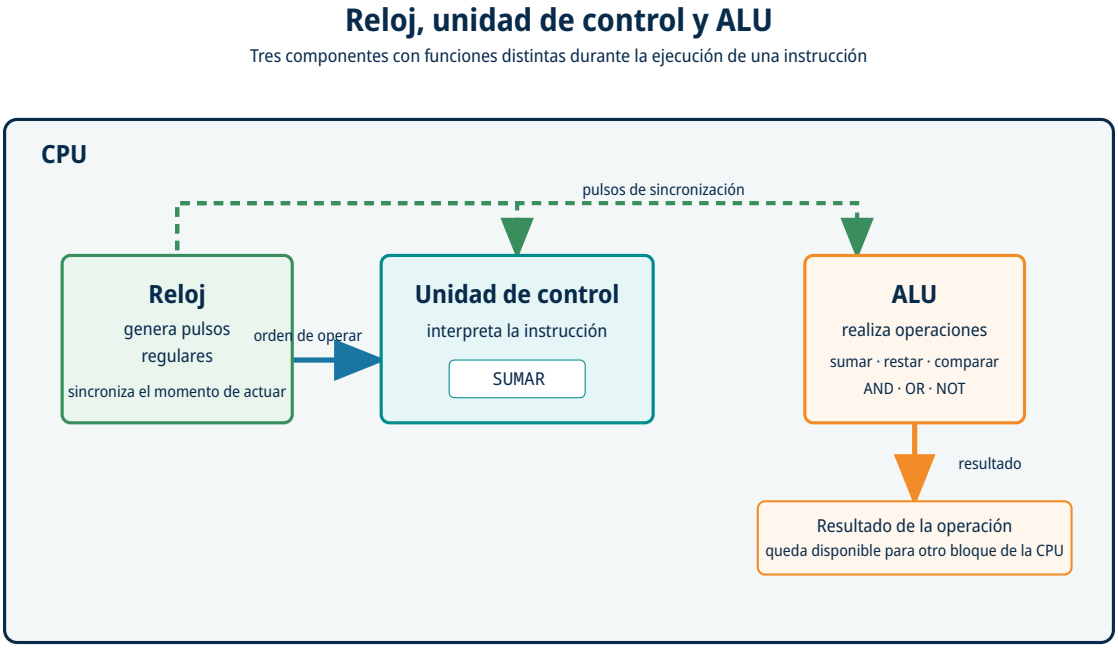


Figura 7: Relación entre reloj, unidad de control y ALU durante la ejecución de una instrucción

Una operación sencilla puede seguir este recorrido:

1. El **reloj** emite un pulso que sincroniza el momento en que pueden cambiar los estados internos de la CPU.
2. La **unidad de control** interpreta la instrucción actual. Por ejemplo, identifica que debe ejecutarse una suma.

3. La unidad de control envía a la **ALU** la orden de realizar esa operación en el instante coordinado por el reloj.
4. La **ALU** calcula o compara los valores que ha recibido y genera un resultado.
5. Ese resultado queda disponible para otro bloque de la CPU, normalmente un registro, desde donde la unidad de control podrá usarlo en el paso siguiente o coordinar su escritura en memoria.

4.2.2. Registros internos

Los **registros** son pequeñas memorias situadas dentro de la CPU. Son muy rápidos, pero tienen una capacidad muy reducida. Se utilizan para guardar temporalmente instrucciones, direcciones, datos o resultados intermedios mientras se ejecuta un programa.

Algunos registros importantes son:

Registro	Función
PC	Contador de programa. Guarda la dirección de la siguiente instrucción que se va a ejecutar.
IR	Registro de instrucción. Guarda la instrucción que se está procesando en ese momento.
MAR	Registro de dirección de memoria. Guarda la dirección de memoria a la que se quiere acceder.
MDR	Registro de datos de memoria. Guarda el dato leído desde memoria o el dato que se va a escribir en ella.
Registros de propósito general	Guardan operandos, resultados parciales o valores temporales.
Flags	Indicadores de estado, por ejemplo si el resultado de una operación ha sido cero, negativo o ha producido acarreo.

Los registros permiten que la CPU trabaje sin tener que acudir constantemente a la memoria principal, que es más lenta.

En el apartado 6 seguiremos el recorrido completo de una instrucción y veremos cómo PC, IR, MAR y MDR intervienen de forma coordinada. De momento, quédate con la idea de que cada registro conserva una pieza concreta de información durante ese recorrido.



> **Predice antes de comprobar.** Si la CPU va a buscar una instrucción en memoria, ¿qué registro debe contener primero la dirección: PC, IR, MAR o MDR? Razona tu respuesta antes de continuar; después podrás contrastarla con el ciclo de instrucción.

4.2.3. Bus interno de la CPU

Los registros, la unidad de control y la ALU necesitan comunicarse entre sí. Para ello utilizan un **bus interno**, es decir, un conjunto de líneas de comunicación por las que circulan datos y señales de control dentro del procesador.

Este bus interno no debe confundirse con los buses generales del sistema, estudiados en el punto 3, que conectan CPU, memoria y entrada/salida.

Ejemplo: una suma entre registros Una operación no utiliza todos los registros disponibles. En este ejemplo, al comienzo solo R1 y R2 contienen los operandos; R3 se reservará para el resultado y R4 no participa. La secuencia muestra cómo la ALU recibe los dos valores y cómo el resultado se guarda después en R3.

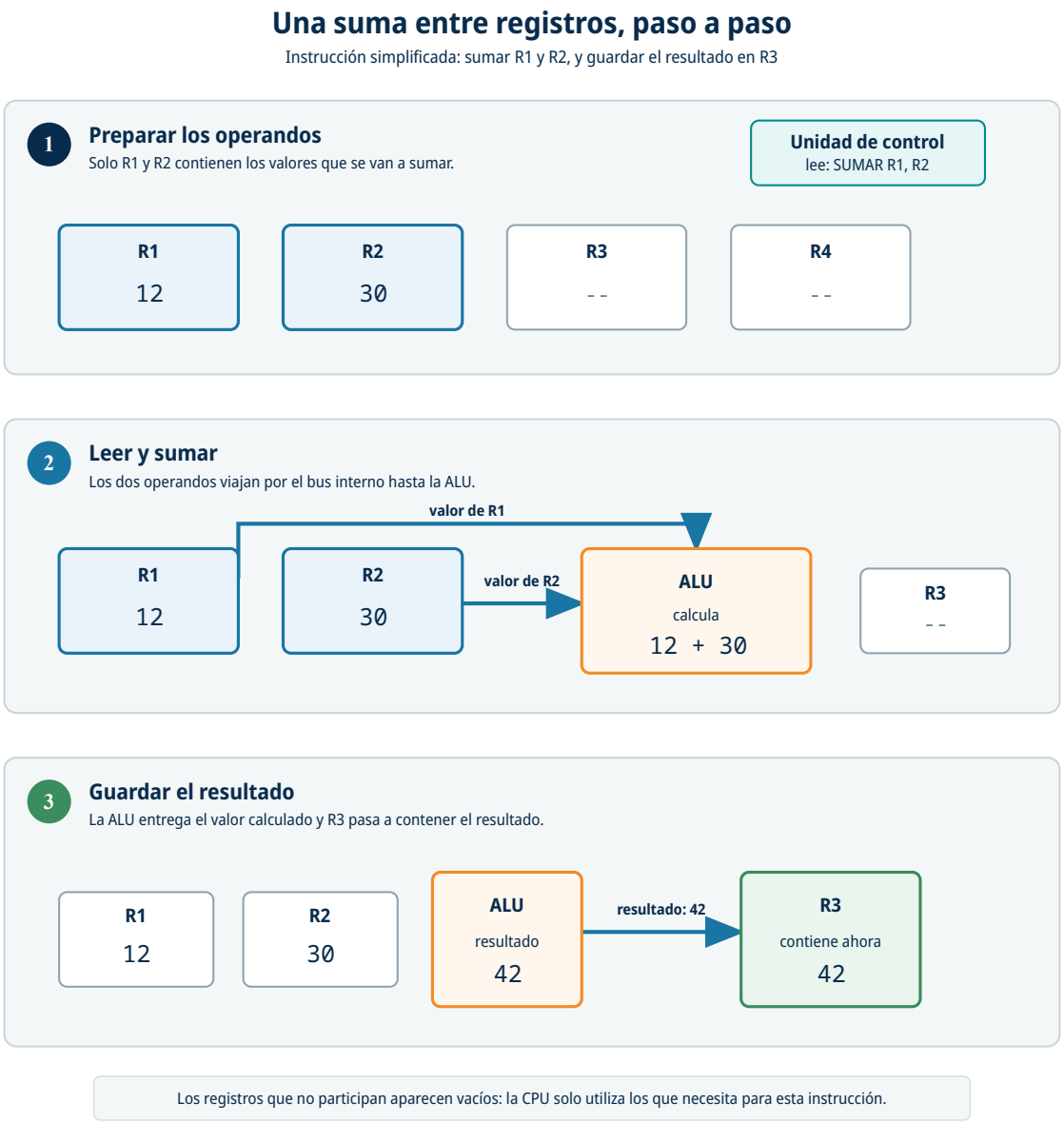


Figura 8: Tres fases de una suma: R1 y R2 contienen los operandos, la ALU los suma y R3 recibe el resultado

Lee los tres paneles del diagrama de arriba abajo:

1. **Preparar los operandos.** R1 contiene el primer operando (12) y R2 el segundo (30). R3 está libre para recibir el resultado, mientras que R4 no interviene en esta suma. La unidad de control interpreta una instrucción simplificada como `SUMAR R1, R2`.
2. **Leer y sumar.** La unidad de control coordina que los valores de R1 y R2 circulen por el bus interno hasta la ALU. La ALU realiza la operación indicada: `12 + 30`, cuyo resultado es 42.

3. **Guardar el resultado.** La ALU entrega 42 y la unidad de control coordina que se copie en R3. A partir de ahí, otra instrucción podrá utilizarlo o escribirlo en memoria si el programa lo necesita.

El dibujo simplifica el recorrido: en un procesador real intervienen más señales y registros, pero la idea esencial se mantiene: **los registros aportan los valores, la ALU opera y otro registro conserva el resultado**.



Si puedes explicar qué papel tienen la unidad de control, la ALU y los registros, ya puedes describir la CPU como un conjunto de bloques funcionales y no solo como "una pieza" del ordenador.

4.3. Ejercicios de aplicación

Para trabajar este bloque, resuelve los ejercicios 1 a 3 con guía del docente y contraste colectivo. Los ejercicios 4 y 5 requieren una explicación más elaborada y pueden realizarse como ampliación.

1. **Verdadero o falso.** Justifica cada respuesta:

   a) la ALU coordina todas las fases de una instrucción
   b) los registros tienen poca capacidad y gran velocidad
   c) el reloj ejecuta programas por sí mismo
   d) el PC guarda la siguiente dirección de ejecución.****

2. **Caso guiado.** Una CPU debe sumar dos valores que ya están en registros. Explica qué papel desempeñan la unidad de control, la ALU y los registros antes, durante y después de la operación.

3. **Diferencia de buses.** Compara en dos frases el bus interno de la CPU con los buses del sistema que conectan CPU, memoria y entrada/salida.

4. **Ampliación: explica un rendimiento.** Dos CPU tienen la misma frecuencia de reloj, pero una completa una tarea antes. Propón tres características, distintas de la frecuencia, que podrían influir y relaciónalas con los bloques estudiados. (Buscar qué es la frecuencia de reloj)

5. **Ampliación: corrige una explicación técnica.** Un compañero escribe: "La ALU busca la siguiente instrucción en la RAM, decide qué operación conviene realizar y guarda por sí sola el resultado". Señala tres errores concretos.

Para ejecutar instrucciones, la CPU necesita leer datos y guardar resultados. Por eso el sistema combina memorias rápidas para trabajar en el momento y almacenamiento persistente para conservar información.

5.1. De la CPU al almacenamiento: una jerarquía

En el bloque anterior se han estudiado los **registros** de la CPU. También almacenan información, pero no son lo mismo que la **memoria RAM** ni tienen la misma función.

Los registros son espacios de almacenamiento mínimos, situados dentro de la CPU. Tienen un nombre y la CPU los utiliza al ejecutar una instrucción: pueden contener un operando, una dirección, la instrucción actual o un resultado inmediato. La RAM, en cambio, es la memoria principal donde se mantienen cargados los programas y los datos con los que se está trabajando.

Un ordenador combina varios niveles para equilibrar velocidad, capacidad y coste. Cuanto más cerca está un nivel de la CPU, más rápido suele ser, pero también más pequeño.

Nivel	Cuándo se utiliza	Qué guarda	Velocidad	Capacidad
Registros	En el instante en que la CPU ejecuta una instrucción	Operandos, direcciones, instrucciones y resultados inmediatos	Muy alta	Muy baja
Caché	Cuando la CPU necesita leer o escribir datos de memoria	Copias rápidas de datos e instrucciones de uso frecuente o cercanos	Alta	Baja/media
RAM	Mientras un programa está abierto o un dato está en uso	Programas, datos, variables y resultados de trabajo	Media	Media/alta
SSD/HDD	Antes de abrir un archivo o programa y al guardarlo	Datos persistentes	Baja comparada con la RAM	Alta

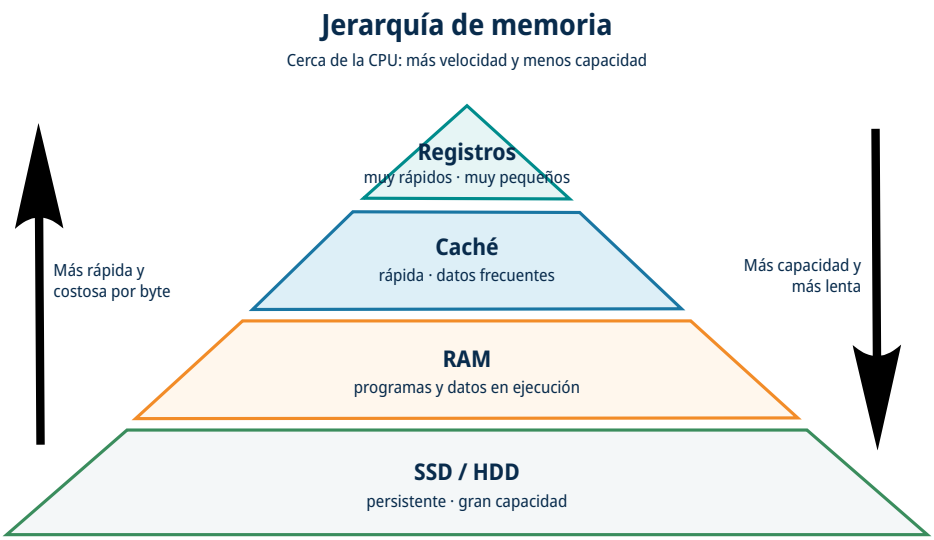


Figura 9: Jerarquía de memoria desde registros hasta almacenamiento secundario

Por ejemplo, al abrir el archivo `practica.py` ocurre, de forma simplificada, lo siguiente:

1. El archivo está guardado en el SSD o HDD.
2. Al abrirlo, el sistema carga el programa y los datos necesarios en la RAM.
3. Cuando la CPU ejecuta una instrucción que necesita un dato, el hardware comprueba primero si existe una copia en la caché.
4. Si el dato no está en caché, se obtiene de la RAM y puede dejarse una copia en caché para próximos accesos.
5. El valor con el que va a operar la ALU se carga en un registro; el resultado inmediato también queda en un registro.
6. Al guardar el archivo, sus cambios se escriben finalmente en el SSD o HDD.



La caché no sustituye a la RAM ni suele gestionarla directamente el programa: el hardware la usa de forma automática como intermediaria rápida. Los registros, en cambio, son el lugar donde la CPU mantiene los valores con los que está operando en ese instante.

5.2. RAM: memoria principal, direcciones, bits y palabras

La **RAM** es la memoria principal de trabajo del sistema. En ella se cargan los programas y datos que se están utilizando en un momento determinado.

Sus características principales son:

- es de **lectura y escritura**;
- permite acceso directo a posiciones de memoria;
- es mucho más rápida que el almacenamiento secundario;
- es **volátil**, es decir, pierde su contenido al apagar o reiniciar el equipo.

Por ejemplo, cuando abres un navegador, el programa y parte de sus datos se cargan en RAM para que la CPU pueda acceder a ellos con rapidez. Si el equipo se apaga, esa información desaparece de la RAM, aunque los archivos guardados sigan estando en el disco.

La RAM se puede imaginar como un conjunto de posiciones numeradas. Cada posición tiene una **dirección de memoria**, que permite localizarla. La CPU no accede a la memoria "por nombre", sino mediante direcciones.

En un ordenador, la información se representa mediante bits, es decir, valores binarios `0` y 1. Los bits suelen agruparse en unidades mayores. Una **palabra de memoria** es un conjunto de bits que el sistema puede leer o escribir como una unidad de trabajo, entendida aquí como el tamaño de dato que la CPU maneja de forma natural en sus registros y operaciones habituales. En ordenadores personales, portátiles y servidores actuales es habitual usar arquitecturas de **64 bits**, por lo que una palabra suele tener 64 bits. En algunas arquitecturas y manuales técnicos, el término "palabra" puede tener un tamaño fijo distinto (por ejemplo, 16 bits en la nomenclatura clásica x86) y no coincidir siempre con el ancho de registro.

Aun así, continúan existiendo sistemas de 32 bits, sobre todo en equipos antiguos o dispositivos especializados. El diagrama usa palabras de 32 bits para que los grupos de bits puedan verse con claridad; el tamaño real depende de la arquitectura.

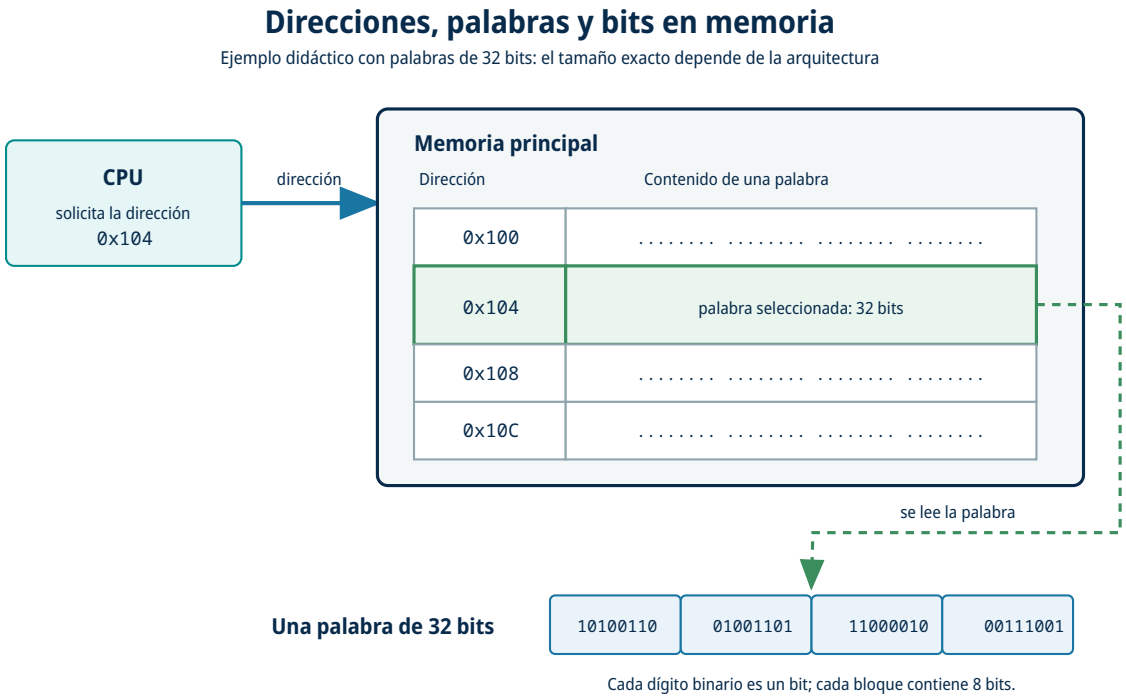


Figura 10: Una CPU solicita una dirección de memoria y obtiene una palabra de 32 bits

5.3. Memoria caché: acelerar el acceso a RAM

La **memoria caché** es una memoria rápida que guarda copias de datos e instrucciones que la CPU utiliza con frecuencia. Su objetivo es reducir el tiempo de espera del procesador.

Suele organizarse en niveles:

- **L1:** muy rápida y pequeña, normalmente asociada a cada núcleo.
- **L2:** algo más grande, pero un poco más lenta.
- **L3:** mayor capacidad y compartida en muchos procesadores, aunque más lenta que L1 y L2.

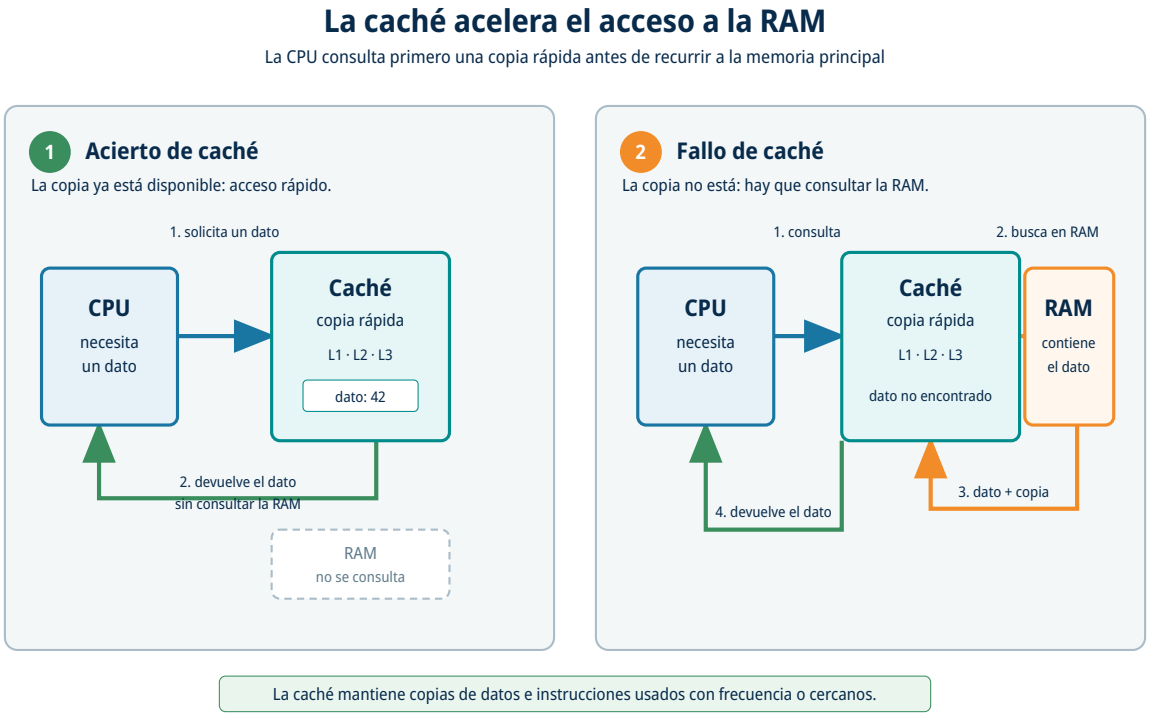


Figura 11: Dos recorridos de acceso a memoria: acierto de caché sin consultar la RAM y fallo de caché con recuperación desde la RAM

La caché mejora el rendimiento porque evita que la CPU tenga que consultar la RAM constantemente.

Esto funciona gracias al **principio de localidad**. La **localidad temporal** indica que un dato o una instrucción usados hace poco probablemente volverán a usarse pronto. La **localidad espacial** indica que, al acceder a una posición de memoria, es probable que se necesiten posiciones cercanas. Por ejemplo, al recorrer una lista de valores, el sistema puede aprovechar que sus elementos suelen estar próximos en memoria.

5.4. Almacenamiento secundario: HDD, SSD y flash

La **memoria secundaria** es el almacenamiento donde se conservan programas, documentos, imágenes, configuraciones y el propio sistema operativo aunque el equipo se apague. A diferencia de la RAM, no se usa normalmente como espacio directo de trabajo de la CPU, sino como lugar de almacenamiento persistente.

Los dispositivos más habituales de memoria secundaria son:

- **HDD** o disco duro magnético: utiliza platos magnéticos y partes mecánicas. Suele ofrecer mucha capacidad a bajo coste, pero es más lento.

- **SSD** o unidad de estado sólido: utiliza memoria flash. Es mucho más rápido que un HDD y no tiene partes mecánicas móviles.
- **Memorias USB y tarjetas de memoria:** también usan memoria flash y se emplean para transporte o almacenamiento externo de datos.

La memoria flash de un SSD no se desgasta por el simple hecho de conservar datos, pero sus celdas soportan un número limitado de ciclos de programación y borrado. Si el sistema escribiera siempre en la misma zona, esa parte del SSD se deterioraría antes que el resto.

Por eso el **controlador** del SSD decide en qué zonas físicas se guardan los datos y reparte las escrituras entre ellas. Esta técnica se denomina **nivelación de desgaste** o *wear leveling*. No elimina el desgaste, pero evita concentrarlo en unos pocos bloques y ayuda a aprovechar de forma más uniforme la vida útil de la unidad.

Cuando abres un programa, normalmente está guardado en la memoria secundaria. Para ejecutarlo, el sistema lo carga en RAM, donde la CPU puede trabajar con él con mucha más rapidez.



No debes confundir memoria RAM con almacenamiento. La RAM se usa para trabajar en el momento; el SSD o el disco duro se usan para conservar datos aunque el equipo se apague.

5.5. Firmware y memorias no volátiles

No toda la memoria se borra al apagar el equipo. Las memorias **no volátiles** conservan su contenido sin alimentación eléctrica.

Algunos ejemplos importantes son:

- **ROM:** memoria de solo lectura, usada históricamente para almacenar firmware.
- **EEPROM:** memoria que puede reprogramarse eléctricamente.
- **Flash:** tipo de EEPROM ampliamente usado en memorias USB, SSD y firmware moderno.
- **Firmware:** programa básico integrado en el hardware, como BIOS o UEFI, que permite iniciar y comprobar el equipo.

En esta unidad nos interesa entender su función general. El detalle de chips, montaje o sustitución física se trabajará más adelante.



Si puedes ordenar registros, caché, RAM y SSD/HDD de mayor a menor velocidad, ya tienes la idea básica de jerarquía de memoria.

5.6. Ejercicios de aplicación

Para trabajar este bloque, resuelve los ejercicios 1 a 5 con guía del docente y contraste colectivo. Los ejercicios 6 y 7 son de aplicación; los ejercicios 8 y 9 pueden realizarse de forma autónoma o como ampliación.

1. **Clasifica memorias.** Clasifica registros, caché, RAM, SSD, HDD, ROM y memoria USB como volátiles o no volátiles.
2. **Respuesta breve.** Explica la diferencia funcional entre memoria RAM y almacenamiento secundario.
3. **Ordena la jerarquía.** Ordena registros, caché, RAM y SSD/HDD de mayor a menor velocidad. Indica también cuál suele tener mayor capacidad.
4. **Tipo test.** Marca la opción correcta:

   a) la RAM conserva los datos al apagar
   b) la caché es más lenta que un HDD
   c) un programa se carga normalmente en RAM para ejecutarse
   d) el firmware es un tipo de memoria volátil. Justifica.

5. **Caso práctico.** Al abrir un editor de vídeo, el programa tarda en iniciarse y después trabaja con muchos datos temporales. Explica qué papel cumplen almacenamiento secundario, RAM y caché en esta situación.
6. **Firmware.** Explica por qué BIOS o UEFI deben almacenarse en una memoria no volátil.
7. **Ampliación: elige una mejora.** Un equipo tarda en abrir programas, pero dispone de suficiente RAM. Compara el posible efecto de sustituir un HDD por un SSD con el de añadir más memoria RAM. Indica qué información necesitarías antes de decidir.
8. **Ampliación: justifica una configuración.** Propón una combinación razonable de almacenamiento y memoria para un aula de programación. Justifica la presencia de RAM, SSD y almacenamiento externo o de red sin citar marcas ni modelos comerciales.
9. **Antes, durante y después.** Una estudiante abre el archivo `practica.py` guardado en el SSD, modifica varias líneas sin guardar y, de repente, el equipo se apaga. Completa la tabla indicando dónde se encontraba la versión más reciente del documento en cada momento y qué ocurre con ella.

Momento	Dónde está la información relevante	Qué ocurre si se apaga el equipo
Antes de abrir el archivo		
Mientras se edita sin guardar		
Después del apagado inesperado		

Ya conocemos la CPU, la memoria y los buses. Ahora podemos unirlos en el proceso central de cualquier ordenador: ejecutar instrucciones una tras otra.

Abrir presentación: Cómo ejecuta instrucciones la CPU

6.1. Conjunto de instrucciones

La CPU no ejecuta órdenes escritas en lenguaje natural. Trabaja con instrucciones codificadas en binario. El **conjunto de instrucciones** de una CPU define qué operaciones puede realizar.

Las instrucciones suelen agruparse en tres tipos básicos:

- **Aritmético-lógicas:** realizan cálculos y comparaciones, como sumar, restar, AND, OR o NOT.
- **Transferencia de datos:** mueven datos entre registros, memoria y dispositivos de entrada/salida.
- **Control de flujo:** cambian el orden normal de ejecución, por ejemplo mediante saltos, llamadas o bucles.

Cada familia de procesadores tiene su propio conjunto de instrucciones, aunque muchas operaciones básicas se repiten en casi todas.

6.1.1. RISC y CISC

Las CPU también se pueden clasificar por la forma en que diseñan su conjunto de instrucciones. Las arquitecturas **RISC** suelen priorizar instrucciones simples y regulares, que se combinan para realizar tareas complejas. Las arquitecturas **CISC** disponen de instrucciones más variadas y algunas pueden expresar operaciones complejas con menos instrucciones.

Aspecto	RISC (*Reduced Instruction Set Computer*)	CISC (*Complex Instruction Set Computer*)
Idea de diseño	Usa un repertorio de instrucciones relativamente simple y regular.	Ofrece un repertorio más amplio, con instrucciones de distinta complejidad.
Una tarea compleja	Suele descomponerse en varias instrucciones sencillas.	Puede expresarse con menos instrucciones, si existe una instrucción que reúna esas acciones.
Acceso a memoria	Tradicionalmente separa la carga y el almacenamiento de los cálculos: primero lleva el dato a un registro, después opera con él.	Puede permitir que algunas instrucciones trabajen directamente con datos de memoria.

Aspecto	RISC (*Reduced Instruction Set Computer*)	CISC (*Complex Instruction Set Computer*)
Formato de las instrucciones	Tiende a usar formatos más uniformes, lo que facilita identificar sus campos.	Puede usar formatos de longitud y estructura más variados.
Ejecución interna	La regularidad favorece que las fases de ejecución sean predecibles.	La CPU puede necesitar traducir una instrucción compleja en operaciones internas más simples.
Ejemplo habitual	ARM y RISC-V.	x86, utilizado por muchos procesadores Intel y AMD.
Consecuencia visible en un programa	Puede haber más instrucciones para expresar la misma tarea.	Puede haber menos instrucciones visibles para expresar esa tarea.

Esta comparación describe tendencias de diseño, no una clasificación rígida ni una medida directa de velocidad. Los procesadores actuales combinan técnicas y, por ejemplo, una CPU x86 puede traducir internamente instrucciones complejas en operaciones más simples.



> **La suma de dos números.** Imagina un robot que debe sumar dos valores guardados en los cajones A y B y dejar el resultado en el cajón C. En un diseño RISC, el robot realiza varios pasos simples: leer A, leer B, sumar y guardar el resultado. En un diseño CISC podría disponer de una orden que indique directamente: "suma A y B y guarda el resultado en C". ¿Qué robot parece más sencillo de fabricar? ¿Cuál podría necesitar menos instrucciones para expresar esta operación?

6.2. Opcode y operandos

+ AMPLIACIÓN

Este apartado se llama **«Opcode y operandos»** porque son las dos piezas que permiten describir qué debe hacer una instrucción y con qué debe hacerlo. El **opcode** responde a la pregunta «¿qué operación debe realizar la CPU?»; los **operandos** responden a «¿sobre qué datos o ubicaciones se realiza?». Sin ambas partes, la CPU no tendría una orden completa que interpretar.

Una instrucción suele incluir dos partes principales:

- **Código de operación** u **opcode:** indica qué operación debe realizar la CPU.
- **Operandos:** indican sobre qué datos debe actuar la instrucción o dónde se encuentran esos datos.

Por ejemplo, una instrucción imaginaria podría representarse así:

```
SUMAR R1, R2
```

En este caso:

- SUMAR sería el código de operación;
- R1 y R2 serían operandos;
- el resultado podría guardarse en un registro definido por la propia instrucción o por la arquitectura.

En una CPU real, esta instrucción no se guarda como texto, sino codificada en bits.

6.3. Ciclo de instrucción

El **ciclo de instrucción** es el proceso que sigue la CPU para ejecutar una instrucción. Se repite continuamente mientras el sistema está funcionando.

Sus fases básicas son:

Fase	Qué ocurre
Búsqueda	La CPU usa el contador de programa (**PC**) para localizar la siguiente instrucción en memoria y cargarla en el registro de instrucción (**IR**).
Decodificación	La unidad de control interpreta el opcode y determina qué componentes deben intervenir.
Ejecución	Se realiza la operación indicada: cálculo en la ALU, lectura de memoria, escritura, comparación, salto, etc.
Almacenamiento	El resultado se guarda en un registro, en memoria o se envía a un dispositivo de salida.

Después de ejecutar una instrucción, la CPU continúa con la siguiente. Si la instrucción era un salto, el valor del PC puede cambiar para continuar en otra dirección de memoria.

6.4. Ejemplo guiado de ejecución

Imagina una instrucción sencilla:

```
SUMAR el contenido de la posición 10 y la posición 11
```

El ciclo podría entenderse así:

1. **Búsqueda:** la CPU trae desde memoria la instrucción que debe ejecutar.
2. **Decodificación:** la unidad de control identifica que se trata de una suma y que necesita leer dos datos.
3. **Ejecución:** la CPU solicita los datos de las posiciones 10 y 11 y la ALU realiza la suma.
4. **Almacenamiento:** el resultado se guarda en un registro o en una posición de memoria.

Este ejemplo simplifica mucho lo que ocurre realmente, pero ayuda a entender la idea principal: ejecutar un programa consiste en repetir este ciclo con muchas instrucciones.

Cuando un programa usa una variable, el valor con el que trabaja debe estar disponible en memoria o en un registro. La CPU lo lee, opera con él y conserva el resultado temporalmente antes de escribirlo donde corresponda. Este puente entre una instrucción y los datos de un programa ayuda a relacionar la arquitectura con el código que se desarrollará en DAM.



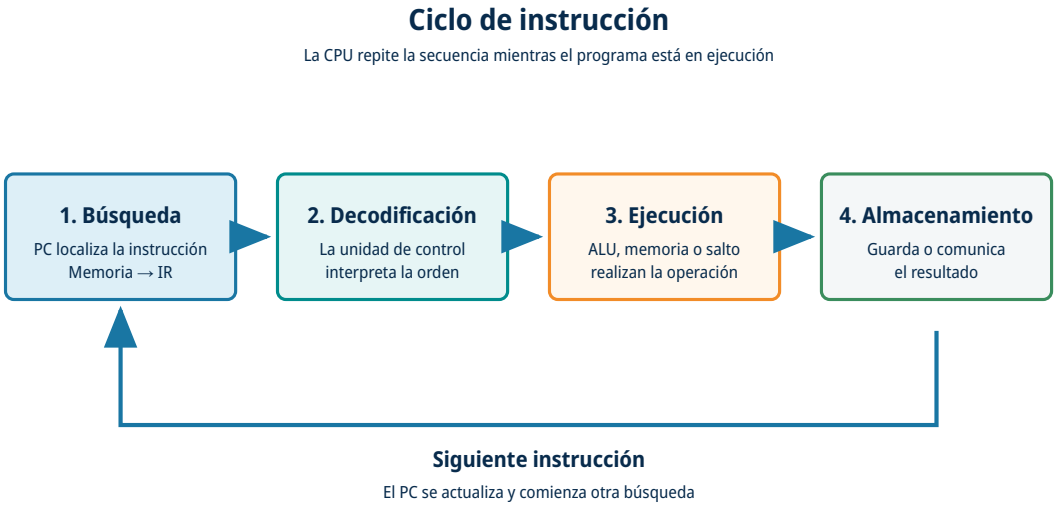
Predice antes de comprobar. Antes de leer la tabla de fases, piensa qué cambiaría si la instrucción fuese un salto: ¿seguiría el PC apuntando a la dirección consecutiva? Explica tu hipótesis y contrástala después con la explicación del ciclo.



Los procesadores modernos usan técnicas como segmentación o predicción de saltos para acelerar la ejecución. En esta unidad nos basta con entender el ciclo básico; esas optimizaciones pertenecen a un nivel más avanzado.



Si puedes relacionar PC, IR, unidad de control, ALU, memoria y buses dentro de las fases del ciclo, ya entiendes el recorrido básico de una instrucción.



6.5. Ejercicios de aplicación

Para trabajar este bloque, resuelve los ejercicios 1 a 3 con guía del docente y contraste colectivo; utiliza los ejercicios 4 a 7 como aplicación con apoyo de los apuntes; reserva los ejercicios 8 a 10 para trabajo autónomo o de ampliación.

1. **Clasifica instrucciones.** Clasifica `SUMAR`, `COPIAR`, `LEER`, `SALTAR` y `COMPARAR` como aritmético-lógicas, de transferencia de datos o de control de flujo.
2. **Opcode y operandos.** En la instrucción `SUMAR R1, R2`, identifica el opcode y los operandos.
3. **Respuesta breve.** Explica por qué una CPU no puede ejecutar directamente una orden escrita en lenguaje natural.
4. **Ordena el ciclo.** Coloca en orden búsqueda, decodificación, ejecución y almacenamiento. Escribe una frase que describa cada fase.
5. **Tipo test.** ¿Qué elemento guarda habitualmente la instrucción que se está procesando? a) PC, b) IR, c) ALU, d) caché. Justifica la respuesta.
6. **RISC y CISC.** Explica con la analogía del robot y los cajones por qué RISC suele combinar varias acciones simples y CISC puede expresar una tarea con menos instrucciones.
7. **Completa una traza.** Para `SUMAR [10], [11] -> R1`, indica qué ocurre en búsqueda, decodificación, ejecución y almacenamiento. Nombra al menos PC, IR, memoria, unidad de control y ALU.
8. **Ampliación: analiza una afirmación.** “Una instrucción CISC siempre se ejecuta más rápido porque hace más cosas”. Indica por qué esa frase es una simplificación y qué otros factores pueden influir.

9. **Ampliación: depura el ciclo.** En una traza, el resultado de una suma es correcto, pero se guarda en una dirección equivocada. Indica en qué fase o elementos buscarías el problema y por qué.
10. **Ampliación: diseña una instrucción.** Inventa una instrucción educativa para comparar dos registros. Define su opcode, sus operandos, el resultado que produce y cómo recorrería las cuatro fases del ciclo de instrucción.

Hasta ahora hemos hablado de datos e instrucciones, pero todavía falta una pregunta clave: ¿cómo se codifica todo eso dentro de la máquina? La respuesta empieza por el sistema binario.

7.1. Sistemas de numeración: decimal, binario y hexadecimal

Un **sistema de numeración** es un conjunto de símbolos y reglas que se utilizan para representar cantidades. El número de símbolos distintos que utiliza recibe el nombre de **base**.

En la vida cotidiana usamos normalmente el sistema **decimal**, de base 10. Sus símbolos son:

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

En informática aparecen otros sistemas porque se adaptan mejor a la forma en que los ordenadores representan la información:

Sistema	Base	Símbolos	Uso habitual
Decimal	10	0-9	Uso humano general
Binario	2	0, 1	Representación interna del ordenador
Hexadecimal	16	0-9, A-F	Direcciones, colores, memoria, depuración



El sistema hexadecimal usa letras porque necesita 16 símbolos. Así, A representa 10, B representa 11, y así sucesivamente hasta F, que representa 15.

7.2. Bits, bytes y valores representables

El **bit** es la unidad mínima de información. Puede representar dos estados posibles:

```
0 o 1
```

Un **byte** u **octeto** está formado por 8 bits:

```
1 byte = 8 bits
```

Con n bits se pueden representar 2^n combinaciones distintas. Si solo se representan números naturales empezando en 0, el valor máximo será:

```
2^n - 1
```

Ejemplos:

Bits	Combinaciones	Rango si son naturales sin signo
1	2	0 a 1
4	16	0 a 15
8	256	0 a 255
16	65 536	0 a 65 535



Si tienes 8 bits, no puedes representar 256 como valor máximo sin signo. Puedes representar 256 combinaciones, pero los valores van de 0 a 255.

7.3. Conversión básica entre decimal, binario y hexadecimal

Para convertir un número decimal entero a binario, se divide sucesivamente entre 2 y se anotan los restos. El resultado se obtiene leyendo los restos desde el último hasta el primero.

Ejemplo: convertir 13 a binario.

```
13 / 2 = 6, resto 1
6 / 2 = 3, resto 0
3 / 2 = 1, resto 1
1 / 2 = 0, resto 1
```

Leyendo los restos al revés:

$$13_{10} = 1101_{2}$$

Para convertir de binario a decimal, se multiplica cada bit por la potencia de 2 que corresponde a su posición, empezando por la derecha con 2^0. Al bit de esa posición (el de menor peso) se le llama **LSB** (*Least Significant Bit*); al bit del extremo opuesto, el de mayor peso, se le llama **MSB** (*Most Significant Bit*). Son términos habituales en documentación técnica.

Ejemplo:

```
1101₂ = (1 · 2^3) + (1 · 2^2) + (0 · 2^1) + (1 · 2^0)
      = 8 + 4 + 0 + 1
      = 13₁₀
```

El sistema hexadecimal es muy útil en informática porque cada dígito hexadecimal equivale exactamente a 4 bits:

```
2^4 = 16
```

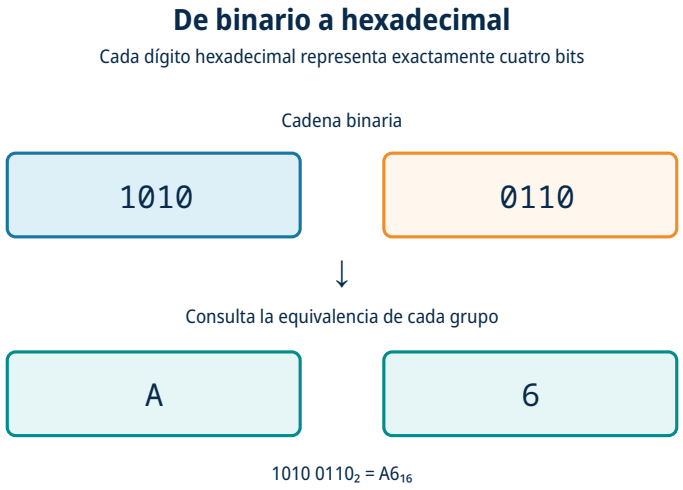
Tabla básica:

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

Ejemplo:

$1101\ 0110_2 = D6_{16}$

Separamos el número binario en grupos de 4 bits, de derecha a izquierda, y convertimos cada grupo a su equivalente hexadecimal.



7.4. Unidades de información: SI e IEC

En informática se usan múltiplos del byte para medir memoria y almacenamiento. Aquí aparece una diferencia importante entre unidades decimales y unidades binarias.

El **Sistema Internacional (SI)** usa potencias de 10:

Unidad SI	Valor
1 kB	1 000 bytes
1 MB	1 000 000 bytes
1 GB	1 000 000 000 bytes

El estándar **IEC** usa potencias de 2:

Unidad IEC	Valor
1 KiB	1 024 bytes
1 MiB	1 048 576 bytes
1 GiB	1 073 741 824 bytes

Por eso puede parecer que un disco tiene menos capacidad cuando lo muestra el sistema operativo. Un fabricante puede anunciar `500 GB` usando unidades decimales, mientras que el sistema puede mostrar una cifra menor en `GiB`.

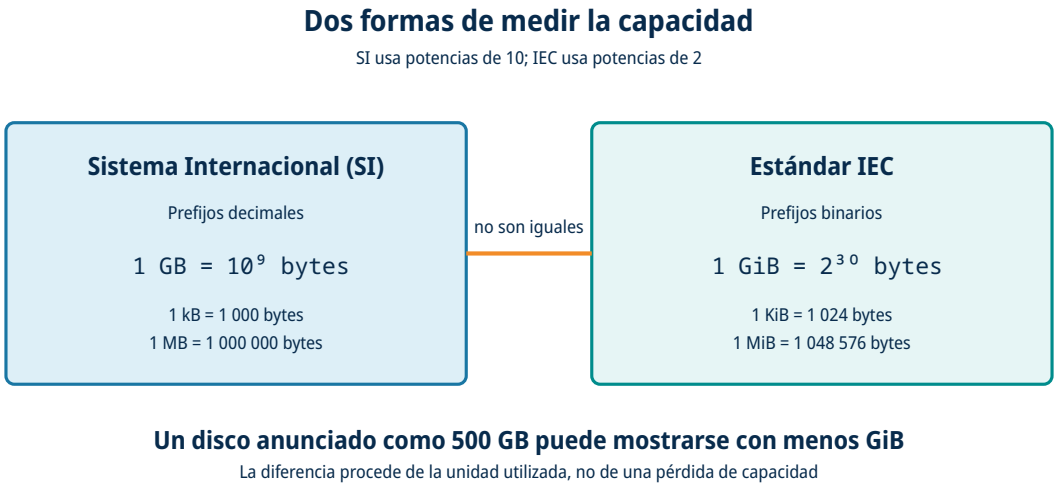


Figura 12: Comparación entre unidades decimales SI y binarias IEC



No es lo mismo kB que kb. La B mayúscula significa byte; la b minúscula significa bit. Un byte son 8 bits.

7.5. Texto: ASCII, Unicode y UTF-8

+ AMPLIACIÓN

Los ordenadores también necesitan representar letras, signos de puntuación, espacios, símbolos y caracteres de distintos idiomas. Para ello utilizan **tablas de codificación**, donde cada carácter se asocia a un número.

Uno de los estándares más conocidos es **ASCII**, que utiliza 7 bits y permite representar 128 caracteres. Incluye letras del alfabeto inglés, números, signos de puntuación y caracteres de control.

Ejemplos:

Código decimal	Carácter
65	A
73	I
97	a

Código decimal	Carácter
32	espacio

El problema de ASCII es que resulta insuficiente para representar todos los idiomas y símbolos actuales. Por eso se utiliza **Unicode**, un estándar diseñado para representar caracteres de prácticamente todos los sistemas de escritura.

La codificación más utilizada de Unicode en la web y en muchos sistemas actuales es **UTF-8**. Es compatible con ASCII en sus primeros caracteres y usa una cantidad variable de bytes según el carácter.

Conviene distinguir ambos conceptos: **Unicode** define qué caracteres existen y les asigna un punto de código; **UTF-8** define cómo se codifican esos puntos de código en una secuencia de bytes. Por eso Unicode no es una simple ampliación de ASCII ni UTF-8 una tabla distinta de caracteres.



La letra A, la letra ñ, un emoji o un carácter chino no ocupan necesariamente el mismo número de bytes en UTF-8.

7.6. Lógica booleana y tablas de verdad

+ AMPLIACIÓN

El **álgebra booleana** trabaja con dos valores:

```
0 = falso
1 = verdadero
```

Esta lógica es fundamental en informática porque los circuitos digitales funcionan con dos estados eléctricos que se interpretan como 0 y 1.

Las operaciones básicas son:

Operación	Significado	Resultado
NOT A	Niega el valor	Vale 1 si A vale 0
A AND B	Conjunción	Vale 1 solo si A y B valen 1
A OR B	Disyunción	Vale 1 si A o B valen 1
A XOR B	Disyunción exclusiva	Vale 1 si A y B son diferentes

Tablas de verdad:

A	NOT A
0	1
1	0

A	B	A AND B	A OR B	A XOR B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Estas operaciones se usan en comparaciones, condiciones, circuitos, permisos, búsquedas y muchas decisiones internas de los sistemas.



Si puedes construir la tabla de verdad de AND, OR y XOR, ya entiendes la base lógica de muchas decisiones que toma un sistema informático.

7.7. Ejercicios de aplicación

Para trabajar este bloque, resuelve los ejercicios 1 a 4 con guía del docente y contraste colectivo; utiliza los ejercicios 5 a 8 como aplicación con apoyo de los apuntes; reserva los ejercicios 9 a 11 para trabajo autónomo o de ampliación.

1. **Identifica la base.** Indica la base de los sistemas decimal, binario y hexadecimal, y escribe qué símbolos usa cada uno.
2. **De decimal a binario.** Convierte mediante divisiones sucesivas entre 2:
   - 18_{10} a binario.
   - 37_{10} a binario.
   - 64_{10} a binario.
   - 125_{10} a binario.
3. **De binario a decimal.** Convierte usando las potencias de 2 de cada posición:
   - 10110_{2} a decimal.
   - 1100101_{2} a decimal.
   - 10000000_{2} a decimal.
   - 11111111_{2} a decimal.

4. **Binario, decimal y hexadecimal.** Completa las conversiones. En las que intervenga binario y hexadecimal, agrupa siempre en bloques de cuatro bits.

Número inicial	Convierte a	Resultado
$1111\ 0010_2$	hexadecimal	
$3C_{16}$	binario	
$1010\ 1111_2$	hexadecimal	
$B7_{16}$	binario	
26_{10}	hexadecimal	
$7D_{16}$	decimal	
255_{10}	hexadecimal	

5. **Valores representables.** Indica cuántas combinaciones se pueden formar con 4, 8 y 16 bits. Señala el valor máximo sin signo en cada caso.
6. **Tipo test.** Marca la afirmación correcta: a) un byte contiene 10 bits; b) kB y kb significan lo mismo; c) un dígito hexadecimal representa cuatro bits; d) UTF-8 solo puede representar letras inglesas. Justifica.
7. **SI e IEC.** Explica por qué un disco anunciado como `500 GB` puede mostrar una capacidad menor cuando el sistema utiliza `GiB`.
8. **Lógica booleana.** Completa los resultados de `1 AND 0`, `1 OR 0`, `1 XOR 1`, `NOT 0` y construye la fila de tabla de verdad correspondiente a `A=1`, `B=0`.
9. **Ampliación: analiza una codificación.** Una aplicación guarda el texto `Año 2026` en UTF-8. Explica por qué no es seguro suponer que todos sus caracteres ocupan el mismo número de bytes.
10. **Ampliación: calcula y justifica.** Un programa usa 12 bits para representar un valor sin signo. Calcula el número de combinaciones y el valor máximo. Explica qué ocurriría si intentara almacenar el siguiente valor.
11. **Ampliación: diseña una regla lógica.** Propón una condición booleana para permitir el acceso a un sistema cuando una persona tiene una cuenta válida y una contraseña correcta, pero no está bloqueada. Define las variables y escribe la expresión usando AND, OR y NOT.

+ AMPLIACIÓN | Todo el punto

Para cerrar el recorrido, volvemos al inicio real de cualquier sesión de trabajo: el momento en que el equipo se enciende y comprueba que puede funcionar.

8.1. Firmware BIOS/UEFI

El **firmware** es software integrado en el hardware. En el arranque del equipo, su función es iniciar y comprobar los componentes básicos antes de cargar el sistema operativo.

Durante muchos años se utilizó **BIOS** como firmware tradicional de los PC. En equipos actuales es habitual encontrar **UEFI**, una evolución más moderna que ofrece más funciones, una interfaz más flexible y mejor soporte para hardware actual.

Aunque BIOS y UEFI no son exactamente lo mismo, en esta unidad nos interesa su papel funcional:

- iniciar el hardware básico;
- realizar comprobaciones iniciales;
- permitir configurar opciones de arranque;
- localizar el dispositivo desde el que se cargará el sistema operativo.



UEFI puede ofrecer opciones avanzadas como **Secure Boot**, soporte para discos grandes mediante particiones GPT o interfaz gráfica. No necesitamos configurar estas opciones ahora; lo importante es entender que forman parte de la fase de firmware, anterior al sistema operativo.

8.2. POST y comprobaciones iniciales

El **POST** (*Power-On Self-Test*) es el conjunto de comprobaciones que realiza el equipo al encenderse. Su objetivo es detectar si los componentes mínimos funcionan correctamente.

Durante el POST se pueden comprobar elementos como:

- CPU;
- memoria RAM;
- teclado u otros dispositivos básicos;
- dispositivos de almacenamiento;
- tarjeta gráfica o salida de vídeo.

Si el POST detecta un problema grave, el equipo puede mostrar un mensaje de error, emitir señales acústicas o detener el arranque. El significado exacto de esos avisos depende del fabricante y del modelo del firmware.



En esta unidad interpretamos el POST como verificación funcional. La reparación física, sustitución de componentes y manipulación interna del equipo se trabajarán en la UD02.

8.3. Dispositivo de arranque y carga del sistema operativo

Después de las comprobaciones iniciales, el firmware busca un **dispositivo de arranque**. Puede ser, por ejemplo:

- una unidad SSD o HDD interna;
- una memoria USB;
- una unidad óptica;
- un arranque por red, si el equipo está preparado para ello.

El orden en que se prueban estos dispositivos se llama **orden de arranque** o *boot order*. Si el primer dispositivo no contiene un sistema arrancable, el firmware prueba con el siguiente.

Cuando encuentra un dispositivo válido, el firmware carga habitualmente un gestor de arranque, como GRUB o Windows Boot Manager, que inicia el sistema operativo. En equipos UEFI, el gestor suele localizarse en la partición del sistema EFI (ESP), normalmente dentro de un disco particionado con GPT. A partir de ese momento, el control pasa progresivamente del firmware al sistema operativo.

8.4. Verificación inicial desde el sistema operativo

Una vez iniciado el sistema operativo, podemos realizar comprobaciones básicas para confirmar que el equipo ha arrancado correctamente y que reconoce sus componentes principales.

Algunas comprobaciones razonables son:

Elemento	Qué comprobar
CPU	Modelo, número de núcleos o arquitectura detectada
RAM	Cantidad de memoria reconocida
Almacenamiento	Unidades disponibles y espacio detectado
Red	Adaptadores de red reconocidos
Sistema	Versión del sistema operativo y estado general

En sistemas GNU/Linux, por ejemplo, se pueden usar comandos como:

```
lscpu
free -h
lsblk
ip addr
uname -a
```

En Windows se puede comprobar información similar desde el Administrador de tareas, Configuración del sistema, Administrador de dispositivos o herramientas como `msinfo32`.



Una verificación inicial no consiste solo en comprobar que "enciende". También implica confirmar que el sistema reconoce CPU, memoria, almacenamiento y dispositivos básicos de forma coherente.

8.5. Secuencia completa de puesta en marcha

Cuando encendemos un equipo, el sistema no empieza ejecutando directamente el sistema operativo. Antes necesita realizar una serie de comprobaciones básicas para verificar que los componentes esenciales están disponibles y que puede continuar el arranque.

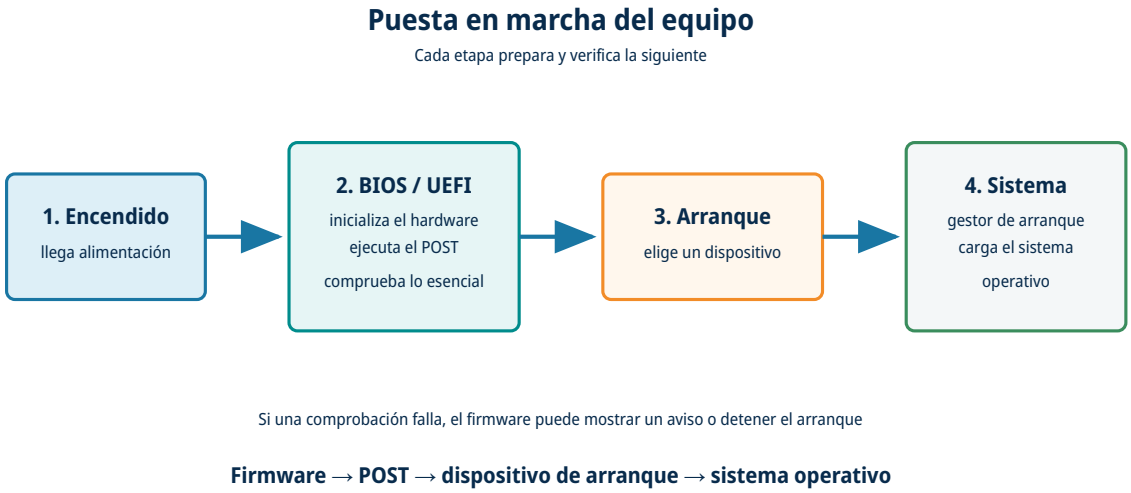
De forma simplificada, el proceso sigue esta secuencia:

1. El equipo recibe alimentación eléctrica.
2. Tras el encendido o el reset inicial, la CPU comienza en un punto de inicio definido por la plataforma y ejecuta el firmware de arranque.
3. El firmware inicializa y comprueba componentes básicos como CPU, memoria y salida de vídeo.
4. El firmware selecciona un dispositivo desde el que iniciar el sistema.
5. Se carga habitualmente un gestor de arranque, que inicia el sistema operativo.
6. El sistema operativo toma el control del equipo.

Este proceso permite pasar de un conjunto de componentes apagados a un sistema capaz de ejecutar programas.



Predice antes de comprobar. Si el POST termina sin errores pero el equipo no encuentra un dispositivo de arranque, ¿qué fases ya han funcionado y cuál debes investigar a continuación? Razona la respuesta antes de revisar el punto 8.3.



8.6. Ejercicios de aplicación

Para trabajar este bloque, resuelve los ejercicios 1 a 3 con guía del docente y contraste colectivo; utiliza los ejercicios 4 a 7 como aplicación con apoyo de los apuntes; reserva los ejercicios 8 a 10 para trabajo autónomo o de ampliación.

1. **Relaciona conceptos.** Completa la segunda columna con firmware, BIOS/UEFI, POST, dispositivo de arranque, gestor de arranque y sistema operativo. Usa cada término una vez.

Función o descripción	Concepto
Programa que inicializa el equipo antes de cargar el SO.	
Ejemplos de firmware de arranque de un PC.	
Comprobaciones iniciales de componentes.	
Medio seleccionado para iniciar el equipo.	
Programa que prepara la carga del SO.	
Administra los recursos y ofrece servicios tras el arranque.	

2. **Ordena la secuencia.** Ordena estas etapas: carga del sistema operativo, POST, selección del dispositivo de arranque, alimentación eléctrica, ejecución del firmware y carga del gestor de arranque.
3. **Respuesta breve.** Explica por qué el firmware debe actuar antes que el sistema operativo.

4. **Verdadero o falso.** Justifica: a) el POST repara automáticamente cualquier componente; b) un equipo puede intentar arrancar desde una memoria USB; c) el sistema operativo toma el control antes que el firmware; d) comprobar que el equipo enciende es suficiente para verificarlo por completo.
5. **Caso guiado.** Un equipo se enciende, muestra el firmware, realiza el POST y después informa de que no encuentra un dispositivo de arranque. Indica qué fases han funcionado y cuál debes investigar a continuación.
6. **Checklist de puesta en marcha.** Con un equipo físico o virtual autorizado, observa el arranque sin modificar el firmware y completa el registro. Anota lo observado y una evidencia identificable: captura, fotografía o registro de observación. No introduzcas contraseñas en las capturas ni reinicies un equipo con trabajo ajeno abierto.

Comprobación	Resultado observado	Evidencia o limitación
Firmware o POST y mensajes de inicio visibles		
Dispositivo desde el que inicia el sistema		
Acceso al sistema operativo		
CPU reconocida		
RAM reconocida		
Unidades de almacenamiento reconocidas		
Adaptador de red y versión del sistema		

   Utiliza las herramientas del apartado 8.4 o sus equivalentes gráficos. Si una fase no es visible, indícalo y explica qué puedes comprobar; no inventes resultados. Entrega la tabla y una conclusión sobre el arranque. Redactar únicamente lo que harías sirve como preparación, pero no demuestra haber verificado la puesta en marcha.
7. **Distingue tareas.** Clasifica estas acciones como comprobación funcional, configuración de firmware o reparación física: consultar la RAM detectada, cambiar el orden de arranque, sustituir un módulo de memoria y comprobar las unidades detectadas desde GNU/Linux.
8. **Ampliación: documenta una incidencia.** Redacta un informe breve de un equipo que no supera el POST. Incluye síntomas observables, qué no debes hacer, datos que recogerías y a quién escalarías la incidencia.
9. **Ampliación: diagnostica con evidencias.** Un equipo inicia el sistema operativo, pero muestra menos memoria RAM de la esperada. Propón una secuencia de comprobaciones no invasivas, desde el sistema y el firmware, antes de manipular el hardware.
10. **Ampliación: plan de verificación.** Diseña un protocolo de puesta en marcha para una máquina virtual recién creada. Debe distinguir qué puedes verificar en la configuración de la máquina virtual, durante el arranque y desde el sistema operativo invitado.

9.1. Actividades de consolidación AULES

Los tres apartados siguientes forman una única entrega de Aules.

9.1.1. Ejercicios de consolidación

1. **Dato o información.** Indica si cada elemento es un dato o información ya interpretada. Justifica brevemente la respuesta.
   - `42`
   - `42 ºC`
   - `La CPU está usando el 85% de carga`
   - 101101_2
2. **Arquitecturas.** Completa la tabla comparando Von Neumann y Harvard.

Aspecto	Von Neumann	Harvard
Memoria		
Buses		
Ventaja principal		
Inconveniente principal		

3. **CPU.** Escribe en la segunda columna el elemento de la CPU que corresponde a cada función. Usa una vez cada uno: PC, IR, MAR, MDR, ALU, unidad de control y flags.

Función	Elemento de la CPU
Coordina las fases del ciclo de instrucción.	
Guarda la dirección de la siguiente instrucción.	
Indican características del resultado, como cero o desbordamiento.	
Guarda temporalmente la instrucción que se está ejecutando.	
Realiza operaciones aritméticas y lógicas.	
Guarda la dirección de memoria que se va a leer o escribir.	
Transporta el dato que entra desde memoria o sale hacia ella.	

4. **Jerarquía de memoria.** Ordena de mayor a menor velocidad: SSD, registros, RAM, caché L1, caché L3. Justifica cada posición usando velocidad relativa, volatilidad, capacidad y una situación de uso.

5. **Buses.** Explica con tus palabras la diferencia entre bus de datos, bus de direcciones y bus de control.
6. **Binario.** Convierte:
   - 13_{10} a binario.
   - 101010_{2} a decimal.
   - $1111\ 0000_{2}$ a hexadecimal.
   - 73_{10} a binario.
   - $1100\ 1011_{2}$ a decimal y a hexadecimal.
   - $9E_{16}$ a binario y a decimal.
7. **Unidades.** Explica la diferencia entre `MB` y `MiB`, y entre `Mb` y `MB`.

9.1.2. Mapa conceptual de la unidad

Antes de elaborar el mapa funcional, construye un **mapa conceptual** de todo lo que has estudiado. Un mapa conceptual organiza ideas y explica cómo se relacionan mediante flechas con palabras de enlace; no es un resumen con cajas aisladas.

Usa el siguiente esquema como base. Incluye los grandes bloques de la unidad y algunas relaciones iniciales, pero tendrás que completarlo con tus propias conexiones y ejemplos.

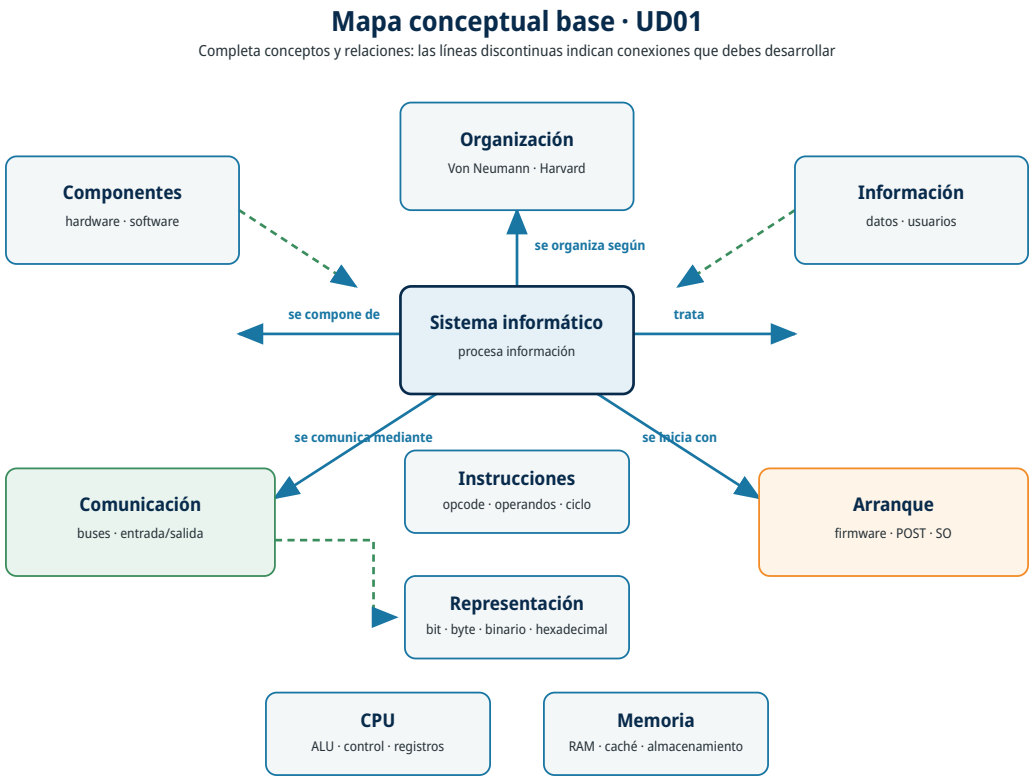


Figura 13: Mapa conceptual base de la UD01: sistema informático, organización, comunicación, CPU, memoria, instrucciones, representación y arranque

Utiliza Excalidraw para crear el mapa. Emplea la imagen solo como referencia: dibuja tu propia versión para poder añadir y modificar elementos.

1. **Copia los bloques principales.** Coloca `Sistema informático` como idea central y distribuye a su alrededor los nueve bloques de la imagen base.
2. **Completa los bloques.** Añade conceptos concretos hasta alcanzar, como mínimo, 16 conceptos en total. Por ejemplo, en CPU puedes incluir ALU y registros; en `Representación` puedes incluir bit y byte.
3. **Etiqueta las relaciones.** Dibuja al menos 12 flechas y escribe sobre cada una una expresión breve que explique el vínculo: “se comunica mediante”, “ejecuta”, “almacena”, “se representa con” o “se inicia con”.
4. **Añade conexiones entre bloques.** No unas todos los conceptos con el centro: incorpora al menos cuatro relaciones entre bloques distintos. Por ejemplo, CPU–memoria, CPU–instrucciones, memoria–representación o firmware–arranque.
5. **Comprueba el resultado.** Revisa que ninguna flecha quede sin etiqueta, que no haya conceptos aislados y que puedas leer una relación completa siguiendo una flecha: “La CPU **ejecuta** instrucciones”.

Exporta tu mapa como PDF o JPG. El mapa debe incluir como mínimo 16 conceptos, 12 flechas etiquetadas y cuatro conexiones entre bloques diferentes.

9.1.3. Reto final: construye un mapa funcional del sistema

Un mapa funcional no es una lista de componentes ni un dibujo de una placa base. Es un diagrama que muestra **qué elementos intervienen** y **qué relación tienen entre sí**. Cada flecha debe expresar una acción o un intercambio: por ejemplo, "envía datos", "lee instrucciones" o "guarda resultados". Créalo en Excalidraw.

Elabora un mapa funcional de un sistema informático a partir de la lámina o fotografía indicada para la actividad. Úsala para reconocer componentes reales, pero representa sus **funciones**, no su posición física.

Sigue estos pasos:

1. **Elige un recorrido sencillo.** Usa este caso: una persona pulsa una tecla y el carácter aparece en pantalla. Ese será el recorrido que explicarás al final.
2. **Dibuja los bloques principales.** Coloca la CPU en el centro. Añade alrededor RAM, almacenamiento secundario, entrada y salida. Escribe un ejemplo en los bloques de entrada y salida: `teclado` y `monitor`.
3. **Añade las relaciones básicas.** Traza flechas entre CPU y RAM; CPU y almacenamiento; entrada y CPU; y CPU y salida. Escribe sobre cada flecha una etiqueta breve que explique la relación.
4. **Completa el contexto de arranque.** Añade un bloque de firmware (BIOS/UEFI) y conéctalo con la CPU y el almacenamiento. Indica que se ejecuta antes del sistema operativo, inicializa el equipo y localiza el sistema que debe cargarse.
5. **Explica la representación.** Añade una nota al mapa: "Los datos y las instrucciones se representan internamente mediante bits". Conecta esa nota, al menos, con RAM y almacenamiento.
6. **Revisa las flechas.** Comprueba que no hay componentes aislados y que cada flecha tiene dirección y significado. Una flecha CPU-RAM puede etiquetarse como "lee y escribe datos e instrucciones"; una flecha CPU-almacenamiento, como "carga programas y guarda resultados".
7. **Redacta el recorrido.** Bajo el mapa, explica en 8 a 12 líneas qué ocurre desde que se pulsa la tecla hasta que aparece el carácter en pantalla. Incluye CPU, RAM, entrada, salida y bits.

El siguiente mapa sirve como modelo de partida. No tienes que copiar su distribución: adapta el diseño al sistema que analices y a tu recorrido de datos.

Ejemplo de partida:

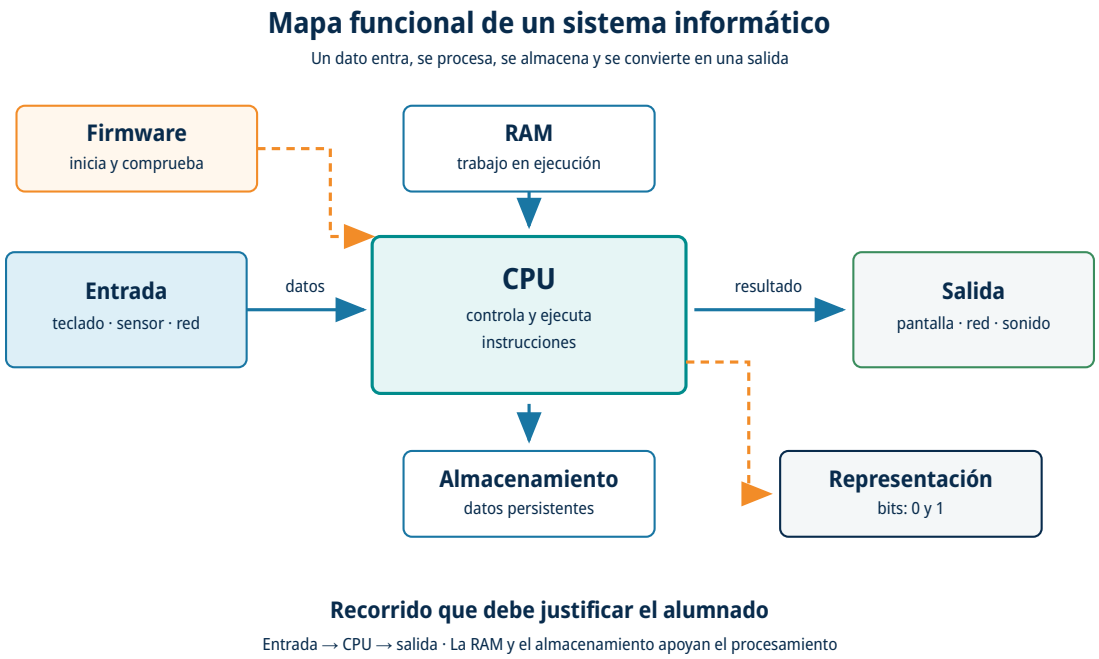


Figura 14: Mapa funcional de un sistema informático con entrada, CPU, memoria, almacenamiento, salida, firmware y representación en bits

Entrega el mapa de Excalidraw exportado como PDF o JPG, junto con la explicación escrita. Deben aparecer CPU, RAM, almacenamiento, entrada, salida, buses o canales de comunicación, firmware y la representación mediante bits.

9.2. Criterios de corrección

Los tres apartados se corrigen como una única entrega y reciben una única calificación. Revisa estos criterios antes de entregarla.

Evidencia	Debe incluir	Para lograr una entrega sólida	Revisa antes de entregar
Actividades cortas	Todas las respuestas y las justificaciones solicitadas.	Emplea con precisión vocabulario como dato, bus, registro, RAM, caché, bit y byte.	No dejes conversiones sin procedimiento ni comparaciones sin justificar.

Evidencia	Debe incluir	Para lograr una entrega sólida	Revisa antes de entregar
Mapa conceptual	Elaboración en Excalidraw, 16 conceptos, 12 flechas etiquetadas y cuatro conexiones entre bloques distintos.	Cada flecha forma una frase legible y correcta: “La CPU ejecuta instrucciones”. Relaciona bloques, no solo conceptos con el nodo central.	Comprueba que no hay flechas sin etiqueta, conceptos aislados ni relaciones contradictorias. Exporta en PDF o JPG.
Mapa funcional final	Elaboración en Excalidraw; CPU, RAM, almacenamiento, entrada, salida, buses o canales, firmware y bits; explicación de 8 a 12 líneas.	Representa flechas con dirección y etiqueta, justifica al menos dos relaciones y explica con claridad el recorrido desde el teclado hasta la pantalla.	Distingue RAM de almacenamiento y firmware de sistema operativo. Exporta en PDF o JPG junto con la explicación.

10.1. Un sistema informático en una tienda

RA/CE: RA2.a.

Una dependienta escanea un producto. El programa consulta su precio, lo muestra en pantalla y guarda la venta.

a) Identifica un ejemplo de hardware, software, datos y persona usuaria en esta situación.

b) Señala una operación de entrada, una de procesamiento, una de almacenamiento y una de salida.

c) Explica la diferencia entre el código leído por el escáner y la información que obtiene la dependienta.

10.2. Dos formas de organizar un ordenador

RA/CE: RA2.a.

El sistema A utiliza una misma memoria y un camino compartido para acceder a datos e instrucciones. El sistema B utiliza memorias y caminos separados para datos e instrucciones.

a) Identifica qué sistema corresponde al modelo de Von Neumann y cuál al de Harvard.

b) Explica qué problema puede aparecer en A cuando necesita buscar una instrucción y acceder a un dato por el mismo camino.

c) Dibuja un esquema sencillo de cada sistema con CPU, memoria o memorias y conexiones.

10.3. ¿Cómo llega un dato desde la RAM hasta la CPU?

RA/CE: RA1.a y RA2.a.

La CPU necesita leer el valor 42, almacenado en la dirección 120 de la RAM.

a) Explica la diferencia entre 120 y 42 en este caso.

b) Indica qué información transporta el bus de direcciones, cuál el de datos y cuál el de control.

c) Describe por pasos cómo se realiza la lectura.

10.4. Encuentra los errores sobre la CPU

RA/CE: RA2.a.

Un compañero escribe:

«La ALU decide qué instrucción se ejecuta después. La unidad de control realiza las sumas. Los registros sirven para guardar archivos cuando apagamos el ordenador. Una CPU de 4 GHz ejecuta siempre cuatro mil millones de instrucciones por segundo».

Localiza los cuatro errores y reescribe cada afirmación correctamente.

10.5. ¿Dónde se guarda cada cosa?

RA/CE: RA1.b.

Para cada elemento —registros, caché, RAM y SSD— indica:

a) Su función principal.

b) Si conserva normalmente su contenido al apagar el equipo.

c) Su posición en una ordenación de menor a mayor rapidez de acceso.

Después, explica por qué un ordenador necesita varios niveles de memoria en lugar de utilizar únicamente un SSD de gran capacidad.

10.6. El documento que desapareció

RA/CE: RA1.b.

Lucía abre un documento guardado en el SSD, escribe durante veinte minutos y se corta la electricidad antes de guardar los cambios. La aplicación no dispone de autoguardado ni de recuperación.

a) ¿Dónde estaba guardado el archivo antes de abrirlo?

b) ¿Qué papel tenía la RAM mientras Lucía trabajaba?

c) ¿Qué espera encontrar al volver a encender el equipo: el documento original, la versión modificada o ninguno? Justifica.

d) ¿Habría evitado perder los cambios tener más RAM? ¿Y un SSD más rápido?

10.7. Verdadero o falso, con explicación

RA/CE: RA1.b y RA2.a; RA1.a como apoyo.

Indica si cada afirmación es verdadera o falsa. Corrige las falsas.

a) Un SSD es una memoria volátil.

b) La caché puede reducir el tiempo que la CPU dedica a esperar datos de la RAM.

c) El bus de direcciones transporta el contenido que se quiere guardar.

d) Un programa guardado en el SSD ya está siendo ejecutado por la CPU.

e) Una pantalla táctil puede actuar como dispositivo de entrada y de salida.

f) Si dos procesadores tienen la misma frecuencia, necesariamente ofrecen el mismo rendimiento.

10.8. Un mismo número, tres representaciones

RA/CE: Saberes de apoyo.

Realiza las conversiones mostrando el procedimiento:

a) 45 en decimal a binario.

b) 10110110 en binario a decimal.

c) 3A en hexadecimal a binario y a decimal.

d) 11110000 en binario a hexadecimal.

Por último, explica por qué un dígito hexadecimal puede representar exactamente cuatro bits.

10.9. Del teclado a la pantalla

RA/CE: RA2.a.

Abres una calculadora, escribes 8 + 5 y pulsas el botón para obtener el resultado.

a) Identifica los dispositivos que intervienen en la entrada y en la salida.

b) Explica qué función desempeñan la RAM y la CPU mientras utilizas la calculadora.

c) Dentro de la CPU, ¿qué papel tienen la unidad de control, la ALU y los registros?

d) Describe el recorrido desde que introduces los números hasta que aparece el resultado en pantalla.

10.10. Cada registro tiene su trabajo

RA/CE: RA2.a.

Relaciona cada registro con su función. Usa cada opción una sola vez.

Registros:

- Contador de programa (PC).
- Registro de instrucción (IR).
- Registro de direcciones de memoria (MAR).
- Registro de datos de memoria (MDR).

Funciones:

A. Guarda la instrucción que se está decodificando o ejecutando. B. Contiene la dirección de la siguiente instrucción que se debe buscar. C. Contiene el dato que se acaba de leer de memoria o que se va a escribir en ella. D. Contiene la dirección de memoria a la que se quiere acceder.

Después, responde:

a) ¿Qué diferencia hay entre guardar una dirección y guardar un dato?

b) ¿Por qué estos registros no sustituyen a la memoria RAM?

10.11. ¿Qué ocurre si el dato no está en caché?

RA/CE: RA1.b.

La CPU necesita un dato. Primero lo busca en la memoria caché, pero no lo encuentra. El dato sí está en la RAM.

a) Explica qué es un fallo de caché.

b) ¿De dónde debe obtenerse el dato en esta situación?

c) ¿Por qué acceder a la RAM suele implicar más espera que encontrar el dato en caché?

d) Un compañero dice: «Si tenemos caché, ya no necesitamos RAM». Explica por qué se equivoca.

10.12. Entrada, salida o ambas

RA/CE: RA1.a y RA2.a.

Clasifica estos dispositivos según su función de entrada, salida o entrada/salida:

- Teclado.
- Monitor sin función táctil.
- Micrófono.
- Altavoces.
- Pantalla táctil.
- Impresora multifunción con escáner.

Para cada uno, indica qué información entra en el ordenador o sale de él.

Después, explica por qué la clasificación depende de la función del dispositivo y no de si se conecta mediante cable o de forma inalámbrica.

10.13. Revisa unas conversiones

RA/CE: Saberes de apoyo.

Un alumno entrega estas respuestas:

A. 1010 en binario equivale a 12 en decimal. B. 1111 en binario equivale a F en hexadecimal. C. 20 en decimal equivale a 10100 en binario. D. 2F en hexadecimal equivale a 25 en decimal. E. 10000000 en binario equivale a 80 en hexadecimal.

a) Indica cuáles son correctas.

b) Corrige las incorrectas mostrando el procedimiento.

c) Explica por qué 10201 no puede ser un número escrito en binario.

Soluciones

Consulta las soluciones de las actividades de repaso.