

Sistemas Informáticos

Unidad 01. Elementos funcionales de una computadora



I.E.S. SERRA PERENXISA



Autores: Sergi García, Alfredo Oltra

Actualizado Julio 2025



Licencia



Reconocimiento - No comercial - CompartirIgual (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se ha de hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 **Importante**

 **Atención**

 **Interesante**

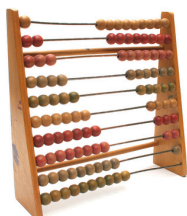
ÍNDICE

1. Evolución histórica	3
1.1 Computadoras no-digitales	3
1.2 Computadoras digitales	3
2. Definición de un sistema informático	5
3. Arquitecturas de computadores	5
3.1 Arquitectura Von Neumann	5
3.2 Arquitectura Harvard	6
4. Functional elements of a computer	7
4.1 CPU (Unidad Central de Proceso)	7
4.2 Unidad de memoria	8
4.3 Otros tipos de memorias	9
4.4 E/S (Entrada/Salida - Dispositivos externos)	9
4.5 Buses	10
4.6 Conjunto de instrucciones	11
5. Ciclo de instrucción en una CPU	11
6. Ejemplo didáctico de un conjunto de instrucciones	13
7. Diseño de CPU: RISC y CISC	15
7.1 Arquitectura RISC (Reduced Instruction Set Computer)	15
7.2 Arquitectura CISC (Complex Instruction Set Computer)	16
7.3 Comparación y evolución: RISC vs CISC	17
8. Bibliografía	17

UNIDAD 01. ELEMENTOS FUNCIONALES DE UNA COMPUTADORA

1. EVOLUCIÓN HISTÓRICA

1.1 Computadoras no-digitales



El ábaco, inventado hacia el año 500 a.C., fue una de las primeras herramientas para realizar cálculos aritméticos de forma rápida. Aunque no era un ordenador en el sentido moderno, marcó el inicio del uso de dispositivos para el procesamiento de datos.

En el siglo XVII, Blaise Pascal creó la Pascalina, una calculadora mecánica capaz de sumar y restar automáticamente. Más tarde, en el siglo XIX, Charles Babbage diseñó la Máquina Analítica, considerada el primer concepto de ordenador programable, aunque nunca llegó a completarse.

Paralelamente, se establecieron fundamentos esenciales para la informática actual: el sistema binario propuesto por Leibniz (siglo XVII), el álgebra de Boole (George Boole, siglo XIX), y la máquina de Turing, descrita por Alan Turing en 1936 como modelo teórico de computación..

Interesante: George Boole definió un sistema algebraico basado solo en dos valores (0 y 1), en lugar de los diez del sistema decimal. Este sistema binario es la base de todos los ordenadores modernos, cuyos componentes físicos funcionan alternando entre dos estados: encendido o apagado.

1.2 Computadoras digitales

El uso de la electricidad permitió construir ordenadores más rápidos, potentes y fiables que los modelos mecánicos anteriores. El primero de este tipo fue el **Z1**, diseñado por el ingeniero alemán **Konrad Zuse** en 1938. Poco después, en 1939, **John Atanasoff** y **Clifford Berry** crearon en EE. UU. el **ABC** (Atanasoff-Berry Computer). Ambos fueron pioneros al usar interruptores eléctricos para representar información: un interruptor **apagado** indicaba un 0, y uno **encendido**, un 1.

Más adelante, durante la Segunda Guerra Mundial, se desarrollaron otros ordenadores programables como el **Colossus** (1943), usado por los británicos para descifrar mensajes secretos, y el **ENIAC** (1946), considerado el primer ordenador de propósito general.

Estos equipos ya no usaban solo relés mecánicos, sino **válvulas de vacío**, dispositivos que controlan el paso de corriente en un tubo sellado al vacío. Gracias a ellas, podían realizar cálculos más rápidamente. Sin embargo, tenían un gran inconveniente: **reprogramarlos requería modificar físicamente su hardware** —por ejemplo, conectando cables o cambiando piezas—, lo que era un proceso lento, complicado y muy costoso.

Interesante: un tubo de vacío es un dispositivo que controla el paso de corriente entre electrodos en un recipiente sin aire (al vacío). En los primeros ordenadores, se usaban para representar los valores 0 y 1, según estuvieran apagados o encendidos.

1.2.1 Computadoras de propósito general

En **1946** se desarrolló el **EDSAC**, el primer ordenador capaz de ejecutar **instrucciones almacenadas internamente**, lo que permitía **reprogramarlo con facilidad**. Hasta entonces, cambiar un programa

requería modificar físicamente el hardware.

En **1949** apareció el **EDVAC**, el **primer ordenador con programas almacenados en memoria** (modelo de von Neumann). Esto marcó un antes y un después: por primera vez, un ordenador podía ejecutar **distintos programas sin modificar su estructura física**. Así nació el concepto de **software**, es decir, instrucciones independientes del hardware que se pueden guardar, compartir y reutilizar.

1.2.2 Transistores y circuitos integrados

Los **tubos de vacío** utilizados en los primeros ordenadores eran voluminosos, frágiles y consumían mucha energía. Por eso, en **1947** se inventó el **transistor**, un componente electrónico mucho más pequeño, eficiente y fiable. Supuso una auténtica revolución tecnológica, permitiendo crear ordenadores más compactos y rápidos.

Sin embargo, **los transistores seguían necesitando conexiones manuales** entre sí, lo que limitaba su miniaturización. Para resolver esto, en **1959** se desarrollaron los **circuitos integrados (IC)**, que permiten **agrupar varios transistores en un único chip**. Este avance fue clave para la creación de ordenadores mucho más pequeños y potentes a partir de los años 60.

1.2.3 Era del ordenador personal (años 70-80)

Durante los años 70 comenzaron a aparecer los primeros ordenadores personales que permitían a usuarios individuales acceder a la informática de manera directa. Entre los modelos más representativos se encuentran el MITS Altair, Apple II, IBM 5100 y Mark-8. Sin embargo, el gran salto llegó con el IBM PC en 1981, cuyo diseño abierto y estándar se convirtió en la base sobre la que se construirían la mayoría de los ordenadores personales posteriores, consolidando la informática como algo accesible para el gran público y no solo para empresas o instituciones.

1.2.4 Internet y la conectividad (años 90-2000)

En la década de los 90, la conexión de los ordenadores personales a Internet empezó a expandirse rápidamente. Esto abrió un mundo de nuevas posibilidades, desde la comunicación instantánea hasta el acceso a información global y el comercio electrónico. A lo largo de los años 2000, el acceso a Internet se popularizó casi de manera universal en los PCs, transformando el modo en que las personas trabajan, estudian y se relacionan.

1.2.5 Actualidad - Informática omnipresente

Hoy, la informática ya no se limita a los ordenadores de escritorio o portátiles. Está integrada en casi todos los aspectos de nuestra vida diaria a través de dispositivos como teléfonos móviles, tabletas, relojes inteligentes, televisores inteligentes, asistentes de voz y electrodomésticos conectados. Esta conectividad permanente ha creado un entorno digital omnipresente que influye en cómo vivimos, trabajamos y nos comunicamos, dando lugar a una sociedad cada vez más interconectada y dependiente de la tecnología.

2. DEFINICIÓN DE UN SISTEMA INFORMÁTICO

¿Qué es un sistema informático? Desde siempre, las personas han querido crear máquinas que les ayuden a resolver problemas. Al principio, cada máquina se diseñaba para una tarea concreta, por ejemplo, un reloj para medir el tiempo o una calculadora para hacer cuentas.

Pero entonces surgió la pregunta: ¿podemos construir una máquina que sirva para resolver diferentes problemas, no solo uno? La respuesta es sí, y esa máquina se llama sistema informático.

Un sistema informático tiene dos partes principales:

- **Hardware:** los componentes físicos, como el teclado, la pantalla, el procesador o el disco duro. Por ejemplo, el teclado es hardware porque lo puedes tocar.
- **Software:** las instrucciones o programas que le dicen al hardware qué hacer, como un juego, un procesador de texto o un navegador de Internet. Por ejemplo, Microsoft Word es software que permite escribir documentos.

Además, algunos expertos consideran otras dos partes importantes:

- **Usuario:** la persona que utiliza el sistema, como tú cuando usas un móvil o un ordenador.
- **Datos:** la información que el sistema procesa, por ejemplo, los documentos que escribes, las fotos que guardas o los vídeos que ves.

En esta unidad vamos a estudiar los elementos funcionales del sistema informático, es decir, los conceptos básicos que forman cualquier ordenador y que, en la realidad, corresponden principalmente al hardware.

3. ARQUITECTURAS DE COMPUTADORES

3.1 Arquitectura Von Neumann

La **arquitectura de Von Neumann**, propuesta en 1946 por el matemático John von Neumann, es un modelo teórico que describe cómo debe estar organizado un ordenador. Este diseño ha sido la base de la mayoría de los ordenadores desde entonces.

Según este modelo, todo ordenador debe tener al menos estos cuatro elementos:

1. **CPU (Unidad Central de Proceso):** Es el “cerebro” del ordenador. Se encarga de interpretar y ejecutar las instrucciones.
2. **Memoria principal:** Almacena tanto los **datos** como las **instrucciones** del programa (algo que diferencia esta arquitectura de otras anteriores).
3. **Buses:** Son los “cables virtuales” que transportan información entre los componentes: datos, direcciones e instrucciones.
4. **Dispositivos de entrada/salida (E/S):** Permiten comunicarse con el exterior: teclado, pantalla, impresora, ratón, etc.

El funcionamiento de un ordenador según la arquitectura de Von Neumann se puede resumir así:

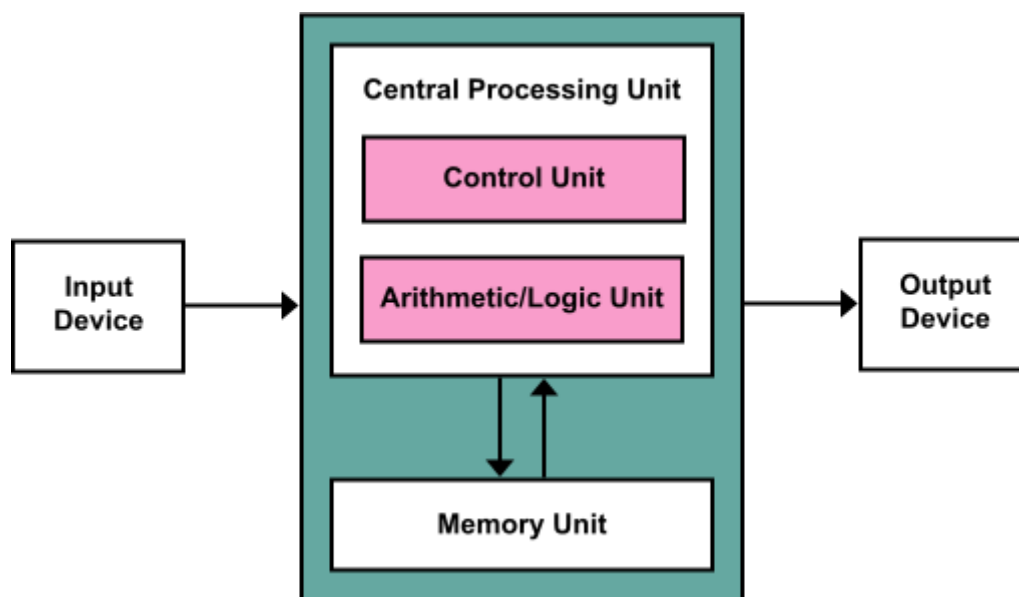
- **Tanto los datos como las instrucciones** están almacenados en la misma memoria.
- La **CPU lee una instrucción**, gracias a los buses, desde la memoria.
- La **unidad de control** de la CPU interpreta esa instrucción y genera las señales necesarias para ejecutarla.
- Si se requiere, la **unidad aritmético-lógica (ALU)** realiza operaciones (suma, resta,

comparaciones...).

- Los **resultados** pueden almacenarse en memoria o enviarse a un dispositivo de salida.
- El proceso se repite con la siguiente instrucción.

Antes de Von Neumann, muchos ordenadores tenían que ser **reconfigurados físicamente** para ejecutar un programa diferente. Con esta arquitectura, basta con **cambiar el contenido de la memoria**, lo que permite ejecutar diferentes programas fácilmente. Esto marca el nacimiento de la **informática programable**.

Hoy en día, aunque los ordenadores modernos han evolucionado, la base de su funcionamiento sigue inspirándose en esta arquitectura. Existen variantes más modernas, como la arquitectura Harvard, pero Von Neumann sigue siendo la más influyente. Además, se han añadido mejoras como la **memoria mapeada**, donde dispositivos de entrada/salida se tratan como si fueran parte de la memoria.



Puedes experimentar con una simulación interactiva de esta arquitectura en el siguiente enlace:

👉 <https://lab.xitrus.es/VonNeumann/>

💬 **Interesante:** Con el tiempo, la arquitectura de Von Neumann ha sido mejorada con técnicas como la E/S mapeada en memoria, que permite tratar dispositivos como posiciones de memoria. También se han incorporado mejoras como la arquitectura Harvard (separación de instrucciones y datos), la segmentación de instrucciones (pipelining), el uso de memoria caché, procesadores multinúcleo y ejecución fuera de orden.

3.2 Arquitectura Harvard

La arquitectura Harvard es otro modelo abstracto que describe cómo puede organizarse un ordenador. Aunque es menos común que la arquitectura de Von Neumann, se emplea en contextos específicos como sistemas embebidos o microcontroladores.

🔍 Diferencias principales con la arquitectura de Von Neumann:

- **Memorias separadas:** En Von Neumann, los datos y las instrucciones se almacenan en la misma memoria, lo que significa que la CPU solo puede acceder a uno de ellos por vez. En Harvard, los datos y las instrucciones están en **memorias diferentes**, lo que permite leer

una instrucción y un dato al mismo tiempo.

- **Buses separados:** En Von Neumann, tanto datos como instrucciones comparten el mismo bus, lo que puede generar cuellos de botella. En Harvard, cada tipo tiene su **propio bus**, lo que mejora la velocidad de acceso y evita conflictos.
- **Espacio de direcciones distinto:** En Von Neumann, los datos y las instrucciones comparten el mismo espacio de direcciones. En Harvard, cada uno tiene su **propio espacio de memoria independiente**.

Aunque Harvard mejora el rendimiento al permitir accesos simultáneos, **su mayor complejidad y coste** (por la duplicación de buses y memorias) limita su uso a sistemas con requisitos específicos, como microcontroladores o procesadores digitales de señal (DSP).

Esquema de Arquitectura Harvard:



! Atención: En la práctica, la mayoría de ordenadores modernos no usan una arquitectura Harvard o Von Neumann puras. Utilizan **modelos híbridos**, que combinan ventajas de ambas para mejorar rendimiento y reducir costes.

4. FUNCTIONAL ELEMENTS OF A COMPUTER


En esta sección describiremos los elementos funcionales de un ordenador basado en la arquitectura de Von Neumann.

4.1 CPU (Unidad Central de Proceso)

La CPU es el núcleo del ordenador. Su función principal es leer las instrucciones almacenadas en memoria y ejecutarlas realizando las operaciones necesarias. Está compuesta por:

- **Registros:** Pequeñas memorias muy rápidas que almacenan datos temporalmente durante la ejecución de instrucciones. Los más importantes son:
 - **Contador de programa (PC):** Guarda la dirección de la siguiente instrucción que se va a ejecutar y se incrementa automáticamente.
 - **Registro de dirección de memoria (MAR):** Contiene la dirección de memoria desde donde se va a leer o escribir información.
 - **Registro de datos de memoria (MDR):** Registro bidireccional que almacena datos leídos de la memoria o datos que serán escritos en ella.
 - **Registro de instrucción (IR):** Guarda temporalmente la instrucción que acaba de ser leída de memoria.
 - **Registros de propósito general:** Almacenan temporalmente operandos o resultados de operaciones.

- **Flags (banderas):** Indicadores simples que pueden estar en “encendido” o “apagado” y que reflejan el estado de operaciones, por ejemplo: la bandera Z se activa si el resultado es cero, la N si es negativo.
- **Bus interno:** Conecta los registros, la unidad de control y la unidad aritmético-lógica, permitiendo la transferencia de datos entre ellos.
- **Unidad de Control (UC):** Es el “cerebro” que dirige el funcionamiento de la CPU, encargándose de interpretar las instrucciones y generar las señales necesarias para su ejecución. Sus componentes clave son:
 - **Decodificador:** Interpreta la instrucción y envía las señales para realizar cada paso del proceso. Por ejemplo, para una instrucción “ADD”, envía señales para leer los operandos de memoria, enviarlos a la ULA, realizar la suma y guardar el resultado.
 - **Reloj (clock):** Marca el ritmo al que se ejecutan las operaciones, sincronizando todos los componentes.
- **Unidad Aritmético-Lógica (ULA):** Esta unidad realiza las operaciones matemáticas (suma, resta, multiplicación...) y lógicas (AND, OR, NOT, etc.) requeridas por las instrucciones. Es la “mano ejecutora” de la CPU.

 **Interesante:** La velocidad de una CPU se mide en Hertz (Hz), que indica cuántas operaciones puede realizar por segundo. Por ejemplo, un procesador de 3 GHz puede realizar 3 mil millones de ciclos por segundo.

4.2 Unidad de memoria

La unidad de memoria es un almacenamiento interno que contiene tanto las instrucciones que la CPU debe ejecutar como los datos o programas que utiliza. Está formada por múltiples celdas, cada una almacenando un único valor binario: 0 o 1 (1 bit).

Un conjunto de celdas de tamaño fijo se denomina **palabra**. Los ordenadores suelen usar palabras de 32 o 64 bits. Cada palabra tiene una dirección única asociada. Al leer información de la memoria, se accede siempre a la palabra completa ubicada en la dirección seleccionada.

Por ejemplo, una memoria de 192 bits se organiza en 6 palabras de 32 bits cada una. Estas palabras se numeran con direcciones del 0 al 5. La primera palabra incluye los bits del 0 al 31, la segunda del 32 al 63, y así sucesivamente.

Normalmente, esta memoria es **RAM** (Memoria de Acceso Aleatorio). La RAM es volátil, lo que significa que necesita estar alimentada con energía para mantener la información; si se corta la alimentación, los datos se pierden. Además, el tiempo que tarda en leer o escribir una palabra es constante, independientemente de su posición en la memoria.

Memorias volátiles relacionadas con la CPU

Además de la RAM, la CPU utiliza varios tipos de memoria volátil esenciales para su funcionamiento:

- **Registros:** Pequeñas celdas de memoria muy rápidas ubicadas dentro de la CPU donde se almacenan datos e instrucciones inmediatas necesarias para realizar operaciones con gran rapidez.
- **Memoria caché:** Una jerarquía de memorias rápidas (niveles L1, L2 y L3) que almacenan datos usados frecuentemente para acelerar el procesamiento y reducir el tiempo de acceso a la RAM.

- **RAM (Memoria de Acceso Aleatorio):** Memoria volátil de mayor tamaño y menor velocidad que la caché, donde se cargan los programas y datos que el sistema está ejecutando en cada momento.

💬 **Interesante:** La velocidad de la memoria se mide por la cantidad de palabras que puede leer o escribir en una unidad de tiempo, y normalmente se expresa en Hertz (Hz), que indica el número de operaciones por segundo.

4.3 Otros tipos de memorias

Otros tipos de memoria presentes en un ordenador son:

- **ROM (Memoria de Solo Lectura):** Es una memoria de solo lectura que no puede borrarse. Se usaba comúnmente en BIOS antiguas para almacenar el firmware de forma permanente.
- **EPROM (Memoria de Solo Lectura Programable y Borrable):** Es una memoria no volátil que es de solo lectura, pero puede borrarse mediante luz ultravioleta y reescribirse. Se usaba en BIOS donde los datos se leen con frecuencia, pero se reescriben raramente.
- **EEPROM (Memoria de Solo Lectura Programable y Borrable Eléctricamente):** Similar a la EPROM, pero puede borrarse eléctricamente y reprogramarse sin necesidad de retirarla. Se usa en memorias BIOS que requieren actualizaciones ocasionales.
- **Memoria Flash:** Evolución de la EEPROM que permite leer y escribir en múltiples posiciones simultáneamente. Se usa habitualmente en memorias USB y es la tecnología más común para almacenar BIOS modernos, especialmente UEFI.
 - **Memoria de firmware (BIOS vs UEFI):**
 - El firmware BIOS tradicional usaba ROM o EEPROM con capacidad limitada (~1 MB) y sin soporte para características modernas.
 - El firmware UEFI actual utiliza memoria Flash NOR (entre 4 y 32 MB), que permite ejecutar código directamente desde la memoria (XIP), ofrece mayor fiabilidad, arquitectura modular y soporta funciones avanzadas como Secure Boot y discos grandes.

4.4 E/S (Entrada/Salida - Dispositivos externos)

Los dispositivos de Entrada/Salida (E/S) son aquellos que permiten al ordenador comunicarse con el exterior. Los dispositivos de **entrada** se usan para introducir información o comandos al sistema, mientras que los dispositivos de **salida** muestran o transmiten información desde el sistema hacia el usuario o hacia otros dispositivos.

Por ejemplo, un ratón, un teclado o un micrófono son dispositivos de entrada, y una pantalla, una impresora o unos altavoces son dispositivos de salida. También existen dispositivos que combinan ambas funciones, actuando como entrada y salida al mismo tiempo, como un disco duro, una pantalla táctil o componentes internos como la tarjeta gráfica, la tarjeta de sonido o un puerto de expansión.

Para distinguir si un dispositivo es periférico o no, se puede usar la referencia a la **arquitectura de Von Neumann**: si el dispositivo forma parte central de la máquina (como la CPU o la memoria principal), no se considera periférico. En cambio, cualquier dispositivo externo o accesorio que conecta la máquina con el mundo exterior se considera un periférico.

Los periféricos suelen conectarse al ordenador mediante diferentes interfaces o buses, como USB, HDMI o PCIe, que permiten la comunicación rápida y eficiente entre el sistema y estos dispositivos.

4.5 Buses

Un **bus** es un conjunto de cables o líneas de comunicación que conectan los distintos componentes de un ordenador para transferir datos, direcciones y señales de control.

El **ancho del bus** indica cuántos bits puede transmitir en una sola operación. Por ejemplo, un bus de 32 bits puede enviar 32 bits al mismo tiempo. Los buses comunes suelen tener anchos de 32, 64 o incluso 128 bits.

La **velocidad del bus** se mide en hercios (Hz), que indican cuántos ciclos o transferencias pueden realizarse por segundo. Sin embargo, no solo la velocidad importa: el ancho del bus también determina la cantidad de información que se transmite en cada ciclo.

Tipos de buses según su forma física:

- **Bus serial:** Transfiere la información bit a bit, a través de un único cable. Aunque pueda parecer lento, el diseño simple y las altas frecuencias permiten que sea muy eficiente. Por eso, hoy en día, buses seriales como USB y SATA superan en rendimiento a muchos buses paralelos.
- **Bus paralelo:** Transfiere varios bits al mismo tiempo, usando varios cables en paralelo. Esto puede generar problemas de interferencia y sincronización, limitando su velocidad real. Por esta razón, aunque sean más anchos, suelen ser menos eficientes que los buses seriales modernos. Ejemplos antiguos son el puerto paralelo y el puerto COM.

Tipos de buses según su ubicación:

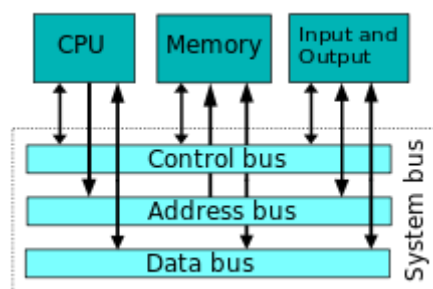
- **Buses internos:** Conectan componentes dentro de un mismo dispositivo, por ejemplo, dentro de la CPU conectan los registros, la unidad aritmético-lógica y la unidad de control.
- **Buses externos:** Conectan dispositivos diferentes, como la CPU con la memoria principal o con dispositivos de entrada/salida.

En la **arquitectura de Von Neumann**, los buses externos se dividen en tres tipos:

- **Bus de control:** Transporta las señales que controlan y sincronizan el funcionamiento de los distintos componentes del ordenador.
- **Bus de direcciones:** Lleva las direcciones de memoria donde se deben leer o escribir datos.
- **Bus de datos:** Transporta la información real, como los datos y las instrucciones que se procesan.

Separar estas funciones en buses distintos, ayuda a organizar el flujo de información y evita conflictos o cuellos de botella, mejorando el rendimiento y la fiabilidad del sistema.

Esquema de buses en Von Neumann:



4.6 Conjunto de instrucciones

La **Unidad Central de Procesamiento (CPU)** dispone de un conjunto definido de instrucciones que puede ejecutar para realizar distintas tareas. Estas instrucciones son comandos codificados que le indican a la CPU qué operaciones realizar.

Las instrucciones se suelen clasificar en tres grandes grupos según su función principal:

- **Operaciones aritmético-lógicas:** Son aquellas que permiten realizar cálculos matemáticos (como suma, resta, multiplicación, división) y operaciones lógicas (como AND, OR, XOR, NOT) sobre datos almacenados en registros o memoria.
- **Operaciones de manejo de datos y memoria:** Permiten transferir datos entre la CPU y la memoria principal o dispositivos de entrada/salida. Incluyen instrucciones para leer (cargar) datos de la memoria o dispositivos, así como para almacenar (guardar) resultados en memoria o enviar datos a dispositivos externos.
- **Operaciones de control de flujo:** Modifican el orden secuencial en que se ejecutan las instrucciones del programa. Esto se logra cambiando el valor del Contador de Programa (Program Counter, PC), lo que permite implementar saltos condicionales o incondicionales, llamadas a funciones, retornos, y bucles.

5. CICLO DE INSTRUCCIÓN EN UNA CPU

El **ciclo de instrucción** es el proceso básico mediante el cual la CPU obtiene, interpreta y ejecuta cada instrucción del programa. Este ciclo se repite continuamente mientras la computadora está encendida y ejecutando código.

Cada instrucción en la CPU está compuesta por un conjunto de bits. La longitud típica de una instrucción está ligada al tamaño de palabra de la CPU (por ejemplo, 16, 32 o 64 bits), que determina cuántos bits puede procesar o manejar la CPU en una operación.

Una instrucción generalmente consta de dos campos principales:

- **Código de operación (Opcode):** Es el código que indica a la CPU qué tipo de operación debe realizar, por ejemplo, sumar, cargar un dato, saltar a otra instrucción, etc.
- **Operandos:** Son los datos específicos o direcciones sobre los cuales se realiza la operación. Estos pueden ser valores inmediatos, registros o direcciones de memoria.

Fases detalladas del ciclo de instrucción

1. Búsqueda (Fetch):

- La CPU utiliza la dirección almacenada en el **Contador de Programa (PC)** para acceder a la siguiente instrucción en la memoria principal.
- La instrucción completa se carga en el **Registro de Instrucción (IR)**.
- El PC se incrementa automáticamente para apuntar a la dirección de la siguiente instrucción, preparándose para la siguiente iteración del ciclo.

2. Decodificación (Decode):

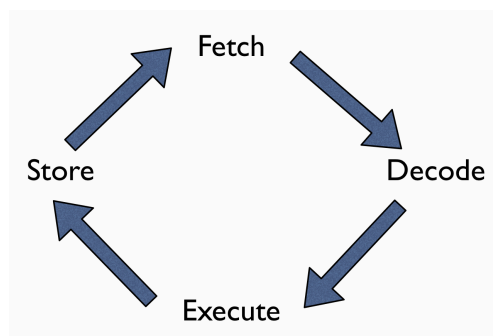
- La unidad de control interpreta el código de operación (opcode) almacenado en el IR.
- Se determina qué tipo de instrucción es y qué componentes de la CPU estarán involucrados en su ejecución.
- Se identifican los operandos necesarios y su ubicación (registros, memoria, dispositivos).

3. Ejecución (Execute):

- La CPU realiza la operación solicitada.
- Por ejemplo, puede tomar datos de registros o memoria, pasarlos a la **Unidad Aritmético-Lógica (ALU)** para efectuar cálculos o comparaciones, o realizar transferencias de datos.
- La unidad de control emite las señales adecuadas para coordinar estos procesos.

4. Almacenamiento (Store):

- El resultado de la operación se escribe en el destino indicado, que puede ser un registro interno de la CPU, una dirección de memoria o un dispositivo externo.
- Si la instrucción implica modificar el flujo de control (como saltos), el PC puede actualizarse con una dirección diferente para continuar la ejecución desde otro punto del programa.



Importancia del ciclo de instrucción

- Este ciclo básico es el motor que permite que la CPU ejecute cualquier programa, desde sistemas operativos hasta aplicaciones de usuario.
- El diseño eficiente del ciclo de instrucción es crucial para el rendimiento de la CPU.
- En arquitecturas modernas, existen técnicas para acelerar este ciclo, como **pipelines**, ejecución fuera de orden y predicción de saltos, que permiten procesar múltiples instrucciones simultáneamente o anticipar pasos futuros.

6. EJEMPLO DIDÁCTICO DE UN CONJUNTO DE INSTRUCCIONES

A continuación, describimos un conjunto de instrucciones ficticio de un ordenador imaginario para entender qué es un conjunto de instrucciones y cómo se codifica una instrucción.

Características del ordenador ficticio:

- Cada instrucción está codificada usando 8 bits.
- Los primeros 2 bits indican la operación a realizar.
- Los siguientes 6 bits almacenan información relacionada con la operación (como direcciones de memoria).
- La memoria tiene $2^6 = 64$ direcciones, cada una con una palabra de 8 bits (1 byte).
- La CPU tiene 3 registros temporales: **R1**, **R2** y **R3**, que almacenan 8 bits cada uno.
- En la memoria:
 - La dirección **000000** contiene la palabra **00001001** (que representa el número 9 en complemento a dos).
 - La dirección **000001** contiene la palabra **00001000** (que representa el número 8 en complemento a dos).

Conjunto de instrucciones:

Código binario	Operación
00	Sumar el contenido de los registros R1 y R2, almacenar el resultado en R3. Los siguientes 6 bits se ignoran.
01	Cargar en R1 el contenido de la dirección de memoria indicada en los 6 bits siguientes.
10	Cargar en R2 el contenido de la dirección de memoria indicada en los 6 bits siguientes.
11	Mostrar en pantalla el contenido del registro R3. Los siguientes 6 bits se ignoran.

Ejemplo de codificación de una instrucción

Si queremos cargar en R2 el valor almacenado en la dirección **000010**, la instrucción se codifica así:
10 000010

- **10** indica la operación de carga en R2.
- **000010** es la dirección de memoria que queremos leer.

Programa sencillo para mostrar el número 17 en pantalla

El siguiente código realiza la suma de los números 9 y 8 almacenados en memoria y muestra el resultado en pantalla.

Instrucción codificada	Descripción
01 000000	Cargar en R1 el contenido de la dirección 000000 (valor 9).
10 000001	Cargar en R2 el contenido de la dirección 000001 (valor 8).
00 000000	Sumar R1 y R2, almacenar el resultado en R3.
11 000000	Mostrar en pantalla el valor de R3 (resultado de la suma).

Ejecución paso a paso del programa

Estado inicial

- Registros R1, R2 y R3 vacíos.
- Memoria con:
 - Dirección 000000 = 9
 - Dirección 000001 = 8

Paso 1: 01 000000 — Cargar en R1 el valor en la dirección 000000

- La CPU interpreta la instrucción.
- Lee el valor 9 en la dirección 000000.
- Guarda 9 en el registro R1.

Estado:

R1 = 9, R2 = vacío, R3 = vacío.

Paso 2: 10 000001 — Cargar en R2 el valor en la dirección 000001

- La CPU interpreta la instrucción.
- Lee el valor 8 en la dirección 000001.
- Guarda 8 en el registro R2.

Estado:

R1 = 9, R2 = 8, R3 = vacío.

Paso 3: 00 000000 — Sumar R1 y R2, almacenar en R3

- La CPU interpreta la instrucción.
- Suma 9 (R1) + 8 (R2) = 17.
- Guarda 17 en el registro R3.

Estado:

R1 = 9, R2 = 8, R3 = 17.

Paso 4: 11 000000 — Mostrar en pantalla el valor de R3

- La CPU interpreta la instrucción.
- Envía el valor 17 de R3 al dispositivo de salida para mostrarlo en pantalla.

Resumen visual del programa

Instrucción	Acción	Registro afectado	Valor almacenado
01 000000	Cargar en R1 desde memoria dirección 000000	R1	9
10 000001	Cargar en R2 desde memoria dirección 000001	R2	8
00 000000	Sumar R1 + R2 y guardar en R3	R3	17
11 000000	Mostrar valor de R3 en pantalla	—	17

7. DISEÑO DE CPU: RISC Y CISC

El diseño de un procesador (CPU) no solo depende de su velocidad o capacidad de memoria, sino también de cómo está construido internamente, especialmente de cómo está diseñado su **conjunto de instrucciones** — es decir, las operaciones que puede realizar directamente la CPU.

Existen dos enfoques fundamentales para diseñar este conjunto de instrucciones:

- **RISC (Reduced Instruction Set Computer)**
- **CISC (Complex Instruction Set Computer)**

Ambos enfoques buscan optimizar el rendimiento del procesador, pero con filosofías diferentes.

7.1 Arquitectura RISC (Reduced Instruction Set Computer)

¿Qué es RISC? Es un diseño que apuesta por reducir la cantidad y complejidad de las instrucciones que la CPU debe ejecutar. La idea es que cada instrucción sea sencilla y rápida, haciendo que la CPU pueda procesarlas eficientemente.

Características principales:

- **Conjunto reducido de instrucciones:** La CPU solo entiende unas pocas instrucciones básicas. Cada una hace una operación simple y clara, por ejemplo, sumar dos números, cargar un dato en un registro, o almacenar un dato en memoria.
- **Instrucciones de tamaño fijo:** Todas las instrucciones tienen la misma longitud (por ejemplo, 32 bits), lo que facilita que la CPU las lea y procese rápidamente y de manera

uniforme.

- **Separación clara entre acceso a memoria y procesamiento:** Solo ciertas instrucciones (load/store) acceden directamente a la memoria, mientras que las operaciones aritméticas o lógicas solo trabajan con registros internos, que son mucho más rápidos.
- **Ejemplo de funcionamiento:** Supón que quieres sumar dos números que están en memoria. En RISC, primero tendrás que cargar cada número en registros separados (dos instrucciones de carga), luego sumar esos registros (una instrucción), y finalmente almacenar el resultado en memoria (otra instrucción). Cada instrucción es simple, pero el conjunto completo permite realizar operaciones complejas.

Ventajas:

- **Mayor velocidad:** Las instrucciones simples suelen poder ejecutarse en un solo ciclo de reloj.
- **Facilidad para implementar técnicas modernas:** Como la ejecución en pipeline (dividir la ejecución de instrucciones en fases) o ejecución fuera de orden.
- **Diseño más sencillo:** La CPU es menos compleja internamente, lo que puede reducir costes y consumo.

7.2 Arquitectura CISC (Complex Instruction Set Computer)

¿Qué es CISC? Este diseño adopta una estrategia diferente: tener un conjunto amplio y muy rico de instrucciones, que pueden realizar tareas complejas en una sola instrucción.

Características principales:

- **Conjunto muy amplio de instrucciones:** La CPU puede ejecutar operaciones complejas con una sola instrucción, por ejemplo, leer un dato, hacer una operación aritmética y escribir el resultado en memoria, todo en un solo paso.
- **Instrucciones de tamaño variable:** Las instrucciones pueden tener diferentes longitudes, dependiendo de la complejidad de la operación, lo que permite optimizar el espacio en memoria.
- **Mayor complejidad del hardware:** La CPU debe poder interpretar y ejecutar muchas instrucciones diferentes, lo que hace que su diseño sea más complicado y pueda limitar la velocidad máxima del reloj.
- **Ejemplo de funcionamiento:** Para sumar dos números en memoria, con una instrucción CISC podrías tener una sola instrucción que diga “suma el contenido de esta dirección de memoria con esta otra y guarda el resultado aquí”. La CPU se encarga de hacer todas esas tareas por dentro.

Ventajas:

- **Programas más compactos:** Como cada instrucción puede hacer mucho, los programas pueden tener menos líneas de código.
- **Facilidad para los programadores y compiladores:** Se simplifica la generación de código porque algunas operaciones complejas son instrucciones nativas.

7.3 Comparación y evolución: RISC vs CISC

- **Rendimiento:** Las CPUs RISC suelen ser más rápidas para ejecutar instrucciones porque cada instrucción es simple y puede completarse en un ciclo de reloj, lo que permite frecuencias mayores y técnicas de paralelismo como pipeline.
- **Tamaño del código y facilidad de programación:** Las CPUs CISC generan programas más compactos, con menos instrucciones para realizar la misma tarea, lo que puede ahorrar memoria y facilitar la escritura del código.
- **Complejidad del hardware:** Las CPUs CISC son más complejas internamente, mientras que las RISC son más simples y eficientes desde el punto de vista del diseño del hardware.

¿Qué ocurre en las CPUs modernas?

Los procesadores actuales combinan lo mejor de ambos mundos. Por ejemplo, las CPU x86 (que son CISC en diseño externo) internamente traducen las instrucciones complejas en secuencias de instrucciones RISC para procesarlas más eficientemente.

Esto permite que los procesadores mantengan compatibilidad con el enorme ecosistema de software legado (programas antiguos para x86), mientras aprovechan las ventajas del diseño RISC para mejorar la velocidad y eficiencia.

¿Por qué no cambiar el conjunto de instrucciones de x86 a RISC directamente?

Porque hacerlo rompería la **compatibilidad binaria**. Esto significa que los programas escritos para procesadores x86 no funcionarían en la nueva arquitectura si cambiase el conjunto de instrucciones, y dado que hay millones de programas, sistemas operativos y aplicaciones que dependen de esa compatibilidad, esto no es viable comercialmente.

Por eso, las CPU modernas traducen internamente las instrucciones CISC a microinstrucciones RISC sin que el software lo note.

8. BIBLIOGRAFÍA

[1] Von Neumann architecture

https://en.wikipedia.org/wiki/Von_Neumann_architecture

To test this architecture, <https://lab.xitrus.es/VonNeumann/>

[2] Harvard architecture

https://en.wikipedia.org/wiki/Harvard_architecture

[3] Modified Harvard architecture

https://en.wikipedia.org/wiki/Modified_Harvard_architecture

[4] Instruction cycle

https://en.wikipedia.org/wiki/Instruction_cycle

[5] RISC

https://en.wikipedia.org/wiki/Reduced_instruction_set_computing

[6] CISC

https://en.wikipedia.org/wiki/Complex_instruction_set_computing

[7] Computer

<https://en.wikipedia.org/wiki/Computer>