Contenidos

- Orígenes del modelo de la arquitectura Von Neumann
- Arquitectura Von Neumann
  - La Unidad Central de Proceso (CPU)
    - Unidad de control (CU)
    - Unidad Aritmético Lógica (ALU)
    - Juego de instrucciones
  - Memoria principal
    - Jerarquía de la memoria en un ordenador
  - Unidad de Entrada/Salida (E/S)
    - Tipos de dispositivos
  - Buses de comunicación
- Ejecución de programas en simulador de arquitectura Von Neumann
- Lenguajes de programación

Orígenes del modelo de la arquitectura Von Neumann

El ordenador digital

- El ordenador es una **máquina electrónica** que está formado físicamente por diversos componentes que en conjunto pueden ejecutar tareas diversas con suma rapidez bajo el **control de un programa**
- **El programa** de un ordenador determina la secuencia de instrucciones que describen las acciones a realizar por el ordenador para llevar a cabo la tarea para la que el programa fue diseñado
- Un ordenador está compuesto de dos partes esenciales
  - El **hardware** → Los componentes físicos: circuitos electrónicos, cables, teclado, …
  - El **software** → la parte intangible: programas, datos, información, ...
- Si bien un ordenador puede ser de dos tipos, analógico o digital, los ordenadores analógicos se utilizaron en el pasado para propósitos muy específicos, y los más difundidos, utilizados y conocidos son los ordenadores digitales

Máquina de Turing Universal

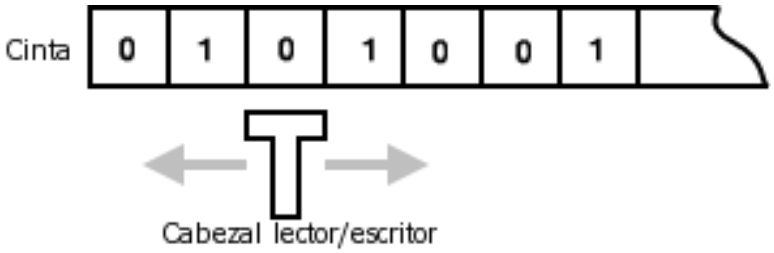
- El modelo básico de arquitectura de un ordenador digital fue propuesto por el matemático y físico **John Von Neumann en el año 1945**, y describe los componentes básicos que este debe tener además de la función a realizar por cada uno
- Es un modelo basado en la **Máquina de Turing Universal** publicado por **Alan Turing en** 1936, que se describía como una **Máquina Programable**, cuya principal característica es la **capacidad de almacenar el programa** a ejecutar y que este se pueda modificar y recargar con otros programas distintos
- Ambos son modelos computacionales, por lo que son **son diseños teóricos** en los que podrían basarse a la hora de construir una máquina computacional real

Funcionamiento de la máquina de Turing (Simplificado)

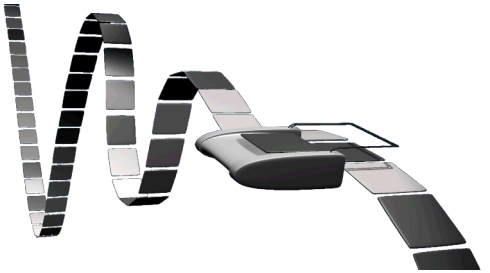
La máquina de Turing está compuesta por:

- Una **cinta** dividida en celdas. Cada celda almacena un carácter de un alfabeto finito
- Un **cabezal** con capacidad de leer y escribir símbolos en la cinta y de moverla a izquierda y derecha
- Un **registro de estado** que almacena el estado de la máquina de Turing. Este conjunto de estados es finito
- Una **tabla de instrucciones**, también finita, que determinan que acción debe hacer la máquina según el estado en el que esté y el valor leído de la cinta. Esta acción puede ser una combinación de las siguientes:
  - Mover la cinta hacia un lado u otro
  - Escribir en la cinta un determinado valor
  - Cambiar el estado interno de la máquina
- A pesar de su simplicidad, una máquina de Turing puede ser **adaptada para simular la lógica de cualquier algoritmo de computador** y es particularmente útil en la explicación de las funciones de una CPU dentro de un computador
- Podemos ver una explicación en el siguiente vídeo:

  https://www.youtube.com/watch?v=iaXLDz_UeYY

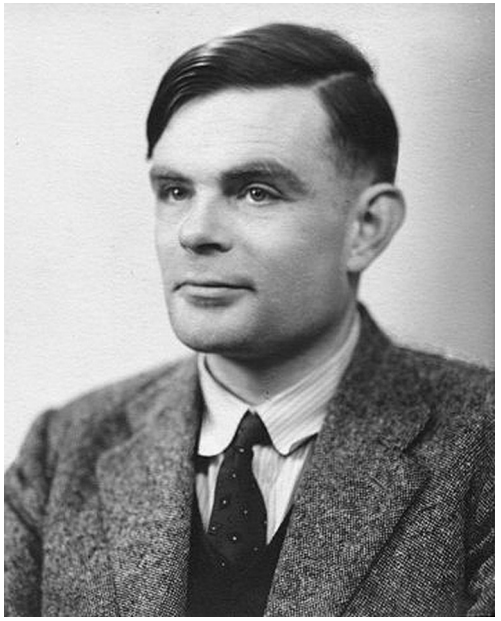


Estado	Símbolo leído	Símbolo escrito	Mov.	Estado sig.
s_1	1	0	R	s_2
s_2	1	1	R	s_2
s_2	0	0	R	s_3
s_3	0	1	L	s_4
s_3	1	1	R	s_3
s_4	1	1	L	s_4
s_4	0	0	L	s_5
s_5	1	1	L	s_5
s_5	0	1	R	s_1



La vida de Alan Turing y su biopic

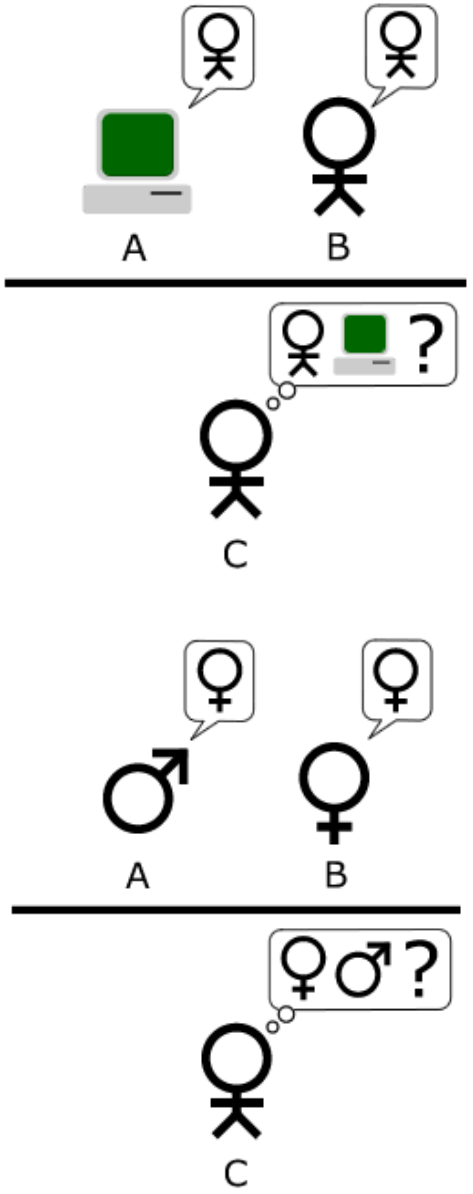
- Turing fue una figura importante en la II guerra mundial, ya que ayudó a descifrar la **máquina enigma**, una máquina de rotores que utilizaba el ejército nazi para cifrar y descifrar mensajes. Esto ayudó en gran medida a inclinar la balanza del lado de los aliados al poder descifrar mensajes de la armada nazi
- En toda la vida de Turing y en ese momento particular se centra el biopic "The imitation game (Descifrando enigma)" (2014), Como pasa en biopics ciertos aspectos de la vida de Turing fueron modificados para hacer la película más emocionante → Polémica en la representación de Turing en Imitation Game (Wikipedia)
- A pesar de su importante papel, Alan Turing fue condenado por homosexualidad en 1952, algo que en aquellos momentos se consideraba delito en el Reino Unido, siendo condenado a la castración química o a ir a prisión, optando por la primera y muriendo dos años después
- El Reino Unido indultó a Alan Turing en 2013, en un gesto que se puede interpretar de disculpa y reconocimiento. Esto fue aprovechado por la productora para empezar ahí la promoción de la película



El test de Turing. El juego de la imitación

- El título **The imitation game** viene de una de las pruebas del test de Turing
- El **Test de Turing** es una prueba de la habilidad de una máquina para exhibir un comportamiento inteligente similar al de un ser humano o indistinguible de este. Para eso un evaluador mantiene una conversación (vía textual con teclado y monitor) con un hombre y una máquina y debe tratar de saber cual es humano y cual es máquina. Una máquina supera el test cuando el evaluador no es capaz de distinguir quien es máquina y quien es humano, aunque en este momento pocas máquinas lo han superado:

  https://www.youtube.com/watch?v=3wLqsRLvV-c
- El juego de imitación es una variante lúdica en la que el interrogador intercambia preguntas a través de notas escritas con un hombre y una mujer con los que no tiene contacto visual. El objetivo es determinar quien es hombre y quien mujer
- Si eres aficionado a la literatura o cine de ciencia ficción tendrás visto que tratar de hacer máquinas que se parezcan lo más posible a humanos es un tema recurrente, pero centrándonos sólo en la prueba de verificación, ¿No encuentras un paralelismo con algún libro o película?

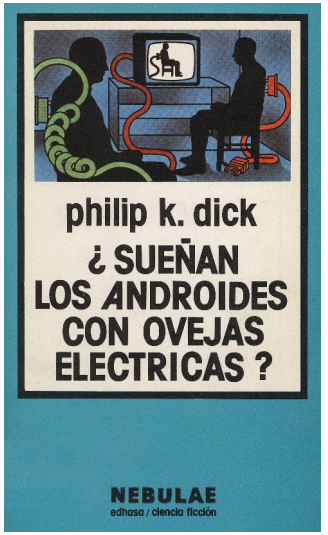


El test de Turing y Blade Runner

- En la película **Blade Runner (1982)** y la novela en la que se basa **"¿Sueñan los androides con ovejas eléctricas?"** de **Philip K. Dick (1968)** se utiliza el **test Voight-Kampff**, un **test ficticio** que permite analizar la empatía del sujeto que se está evaluando con el fin de determinar si se trata de un replicante (androide) o un ser humano. Para ello buscaba una respuesta emocional ante la descripción de diversas situaciones:

  https://www.youtube.com/watch?v=gvJ3eSurq5Y

- Hay un cierto paralelismo con el de Turing, ya que este test evalúa si estamos evaluando una máquina o un ser humano real, sólo que el test Voight-Kampff trata de **medir la respuesta emocional en vez de la inteligencia**, ya que la inteligencia ya está supuestamente superada en este hipotético futuro



Arquitectura Von Neumann

- El modelo de la máquina de Turing sentó la base en la que se basó el matemático húngaro-estadounidense John Von Neumann en la definición de su **Arquitectura Von Neumann** que es un modelo más concreto a la hora de aplicarlo en la construcción real de un ordenador de lo que era el modelo de Turing que era más teórico.
- Su propuesta de máquina programable fue un gran aporte ya que antes de este modelo las arquitecturas de las máquinas con capacidad de procesamiento **sólo permitían ejecutar un programa fijo** para el cual eran específicamente diseñadas, y **cambiarlas** para que ejecutaran otro **requería una reestructuración de la misma**

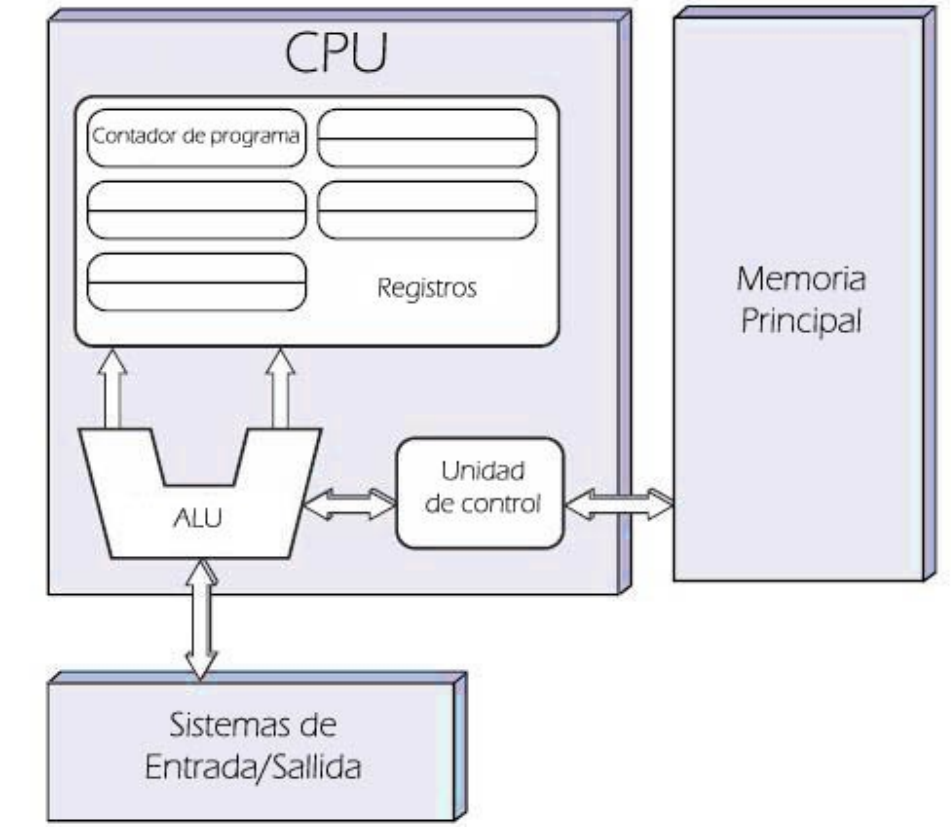
- **El modelo Von Neumann sigue vigente hoy en día y es el más extendido**. Es por esto que ahora nos centraremos en desarrollar más en detalle cada uno de los componentes que lo forman y la interacción entre ellos para la **ejecución de los programas**

Arquitectura Von Neumann

Arquitectura Von Neumann. Componentes I

La arquitectura de Von Neumann propone que un sistema informático deberá estar formado por los siguientes componentes funcionales:

- **CPU o Unidad Central de Proceso**
- **Memoria principal**
- **Unidad de Entrada/Salida**
- **Buses**



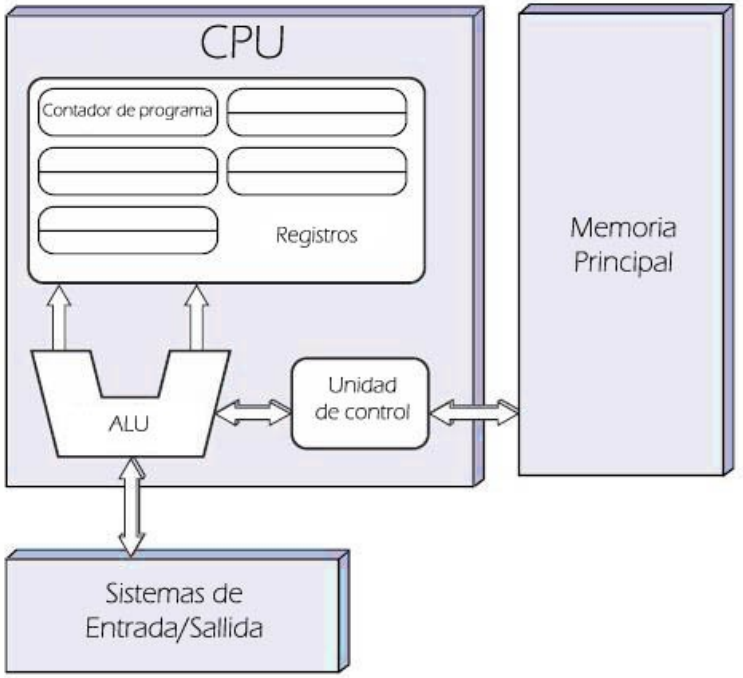
Arquitectura Von Neumann. Componentes II

- **Unidad Central de Proceso (CPU. Del inglés Central Processing Unit)**

  La Unidad Central de proceso (conocida por sus siglas CPU del inglés Central Processing Unit) es el componente encargado de interpretar las instrucciones de los programas informáticos que se ejecutan en el sistema y del procesamiento de sus datos, así como de coordinar las actividades de todo el sistema.

- **Memoria principal**

  Es el componente encargado del almacenamiento de las instrucciones y datos del programa o programas que se ejecutan en el ordenador. Está dividida en celdas de un mismo tamaño y cada una de ellas tiene una dirección única



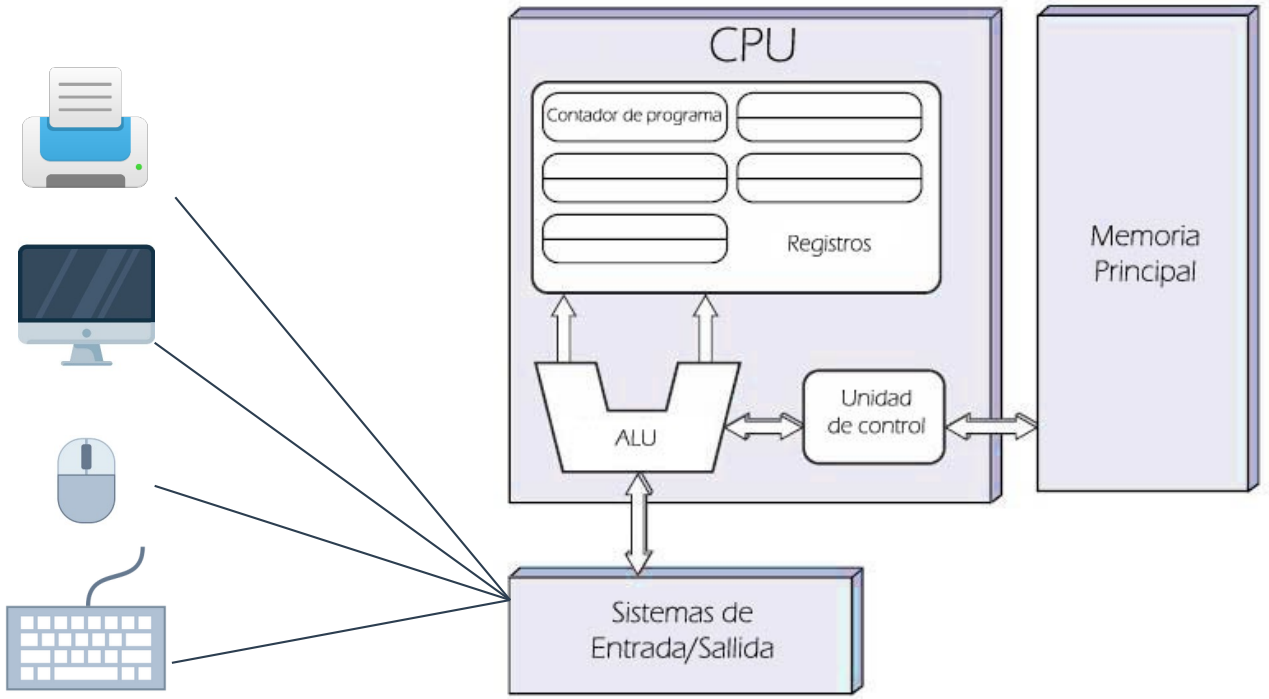
Arquitectura Von Neumann. Componentes III

- **Unidad de Entrada/Salida**

  Es el que permite el intercambio de información con otros dispositivos externos, llamados **periféricos**. Estos periféricos sirven para múltiples usos como **facilitar la interacción humana con el ordenador** (ratones, teclados, impresoras), la comunicación del ordenador con otros (tarjetas de red, módems, …) y disponer de **unidades de el almacenamiento de datos** (Discos duros, unidades de disco, memorias extraíbles, …)

- **Buses**

  Permiten que el resto de componentes se puedan comunicar entre si intercambiándose los datos e instrucciones



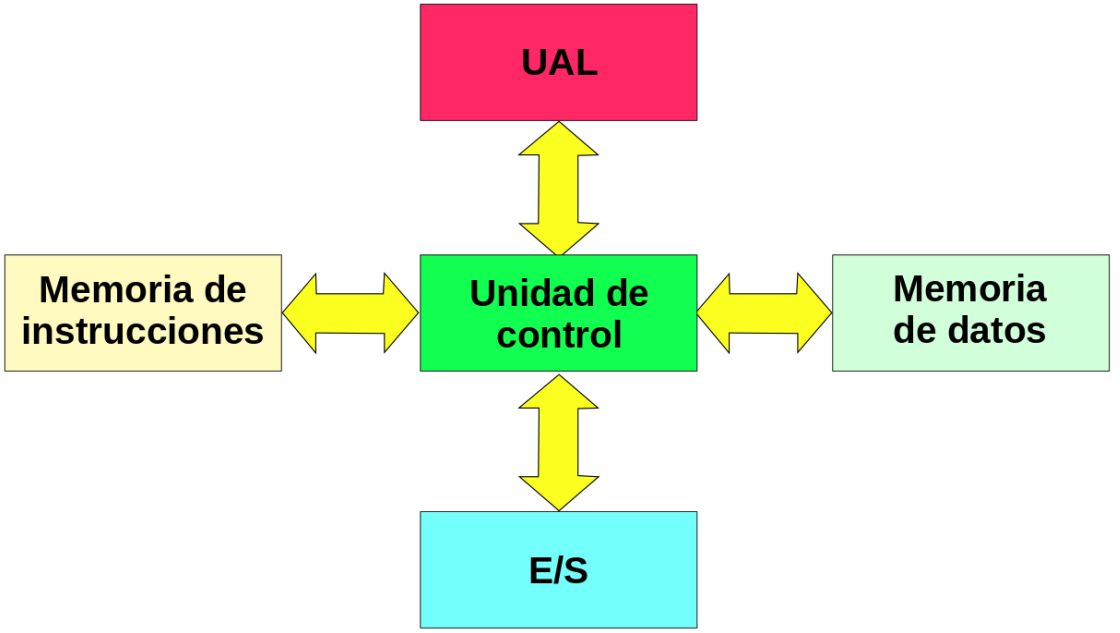
Ejecución de un programa (Sin interrupciones)

De forma simplificada la ejecución de un programa en el modelo de arquitectura Von Neumann tiene lugar de la siguiente forma:

- **Las instrucciones del programa se cargan en la memoria principal**, por lo general en el orden en el que serán ejecutadas y en posiciones contiguas.
- Empezando en la primera instrucción la CPU se encarga de la **ejecución de cada instrucción** a través de un ciclo que básicamente consiste en:
  - **Lectura de la instrucción**. Se obtiene de la memoria la siguiente instrucción del programa a ejecutar de la memoria para ser llevada a la CPU donde es almacenada en un registro interno.
  - **Ejecución de la instrucción**. La instrucción es decodificada e interpretadas y se realizan las operaciones que esta indica. Las acciones de la ejecución pueden variar mucho de unas instrucciones a otras.
  - **Avanzar o saltar a la siguiente instrucción a ejecutar en memoria**.
- El proceso de lectura, ejecución y avance/salto a la siguiente se repite hasta la finalización de la ejecución del programa

Otras arquitecturas

- Sobre la arquitectura Von Neumann nos podemos encontrar variantes encaminadas a optimizar el rendimiento del sistema, basadas en la incorporación de más procesadores, más memorias (cachés o memorias distribuidas) o más buses, pero todas estas soluciones parten de la base definida por Von Neumann.
- Existen también otras arquitecturas como la **arquitectura Harvard**, que tiene como principal característica la **separación de memorias para datos e instrucciones**, o una **arquitectura híbrida**, que es la que suele utilizarse actualmente (**tiene una cache de datos y otra de instrucciones pero una sola memoria**) pero los componentes y funcionamiento general son muy similares a la de Von Neumann.



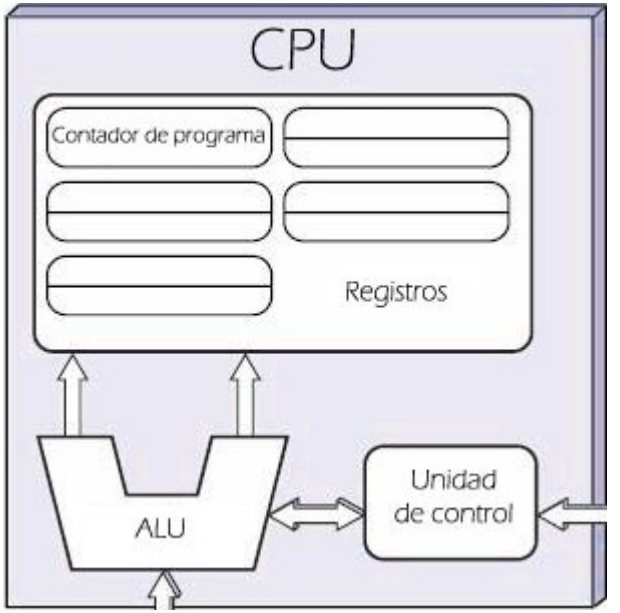
La Unidad Central de Proceso (CPU)

Elementos de la Unidad Central de Proceso (CPU)

La CPU es la encargada de interpretar las instrucciones de los programas informáticos que se ejecutan en el sistema y del procesamiento de sus datos, así como de coordinar las actividades de todo el sistema.

Si bien la forma, el diseño y la implementación de las CPU ha cambiado drásticamente desde los primeros ejemplares su operación fundamental sigue siendo la misma y desde el punto de vista funcional la CPU sigue estando formada por:

- **Unidad de Control (CU del inglés Control Unit)**, encargada de la extracción de las instrucciones de la memoria y de la coordinación de su ejecución
- **Unidad Aritmético Lógica (ALU del inglés Arithmetic Logic Unit)**, encargada de realizar operaciones aritméticas y lógicas cuando la instrucción a ejecutar así lo requiera.
- **Los registros internos de la CPU**, que son elementos auxiliares que permiten el almacenamiento de datos utilizados generalmente en la ejecución de instrucciones.



Componentes CPU. Los registros

Los registros de la CPU son elementos con **capacidad de almacenar datos**. Son por tanto un elemento de memoria que se caracterizan por **su alta velocidad y baja capacidad**.

La capacidad de almacenamiento de los registros de mide en bits, con tamaños que varían entre 8, 16, 32 o 64 bits dependiendo del tipo de información que almacenan y de la CPU.

La CPU los utiliza como apoyo para la ejecución del programa, y entre estos son especialmente significativos con respecto a la ejecución de las instrucciones:

- **El contador de programa (PC)** → PC del inglés Program Counter, **es un registro que almacena** la dirección de memoria de la siguiente instrucción a ejecutar
- **El registro de instrucción (IR)** → Del ingles Instruction Registry, **es un registro que almacena** la instrucción que se está ejecutando en ese momento

En cualquier CPU encontraremos registros adicionales que entre otras cosas se utilizarán para almacenar temporalmente datos leídos de la memoria o resultados intermedios de las operaciones de cálculo.

Componentes CPU. La unidad de control (CU)

Se encarga de extraer de la memoria principal las instrucciones de los programas informáticos y controlar su interpretación y ejecución coordinando para ello a los restantes elementos del ordenador por medio de señales de control

Para realizar el control de la ejecución de las instrucciones de un programa la Unidad de Control se apoya en los siguientes elementos:

- **El decodificador** → Se encarga de interpretar la instrucción leída de la memoria y cargada en el registro IR para su posterior ejecución
- **El reloj** → Genera pulsos eléctricos a intervalos constantes que determinan el momento en el que se ejecutan cada uno de los pasos asociados a una instrucción
- **El secuenciador** → Descompone la instrucción a ejecutar en microórdenes elementales que se procesarán al ritmo marcado por el reloj

Componentes CPU. La Unidad Aritmético Lógica (ALU)

La ALU apoya a la UC **realizando algunas de las operaciones elementales** resultado de la descomposición de la instruccción por parte del secuenciador.

Las operaciones más típicas que realiza una ALU son:

- **Operaciones aritméticas de números enteros** (suma, resta y opcionalmente multiplicación y división)
- **Operaciones lógicas de bits** (NOT, AND, OR y XOR)
- **Operaciones de desplazamiento de bits**, consistentes en mover un número de bits un número determinado de bits a la derecha o a la izquierda
- Algunas ALUs pueden incorporar operaciones más complejas, como las **operaciones aritméticas de números en coma flotante**, pero estas complican su diseño y el coste de las mismas. En algunos casos este tipo de operaciones son realizadas por componentes adicionales de apoyo.

Características relevantes de la CPU

- **Frecuencia de reloj**

  El ordenador funciona en modo síncrono, realizando operaciones en una secuencia ordenada en el tiempo. Para ello necesita contar con un generador de impulsos (el reloj), que marca el momento en que debe ejecutarse cada operación. El número de impulsos generados se mide en ciclos por segundo (típicamente megahercios, MHz, o Gigahercios, GHz), y por tanto **la frecuencia del reloj determina la velocidad en que se ejecutan las instrucciones**. Una operación puede conllevar varios ciclos de reloj para ser ejecutada.

- **Longitud de palabra de datos (N.º de bits)**

  Determina la cantidad de información que es capaz de procesar simultáneamente la CPU en cada pulso de reloj. Se mide en bits y determina el tamaño de los registros y de los buses de comunicación. Una mayor longitud de palabra implica mayor complejidad y circuitería pero mayor potencia de proceso.

- **Juego de instrucciones**

  Cada CPU tiene su propio juego de instrucciones. Para ejecutar un programa escrito en código máquina o ensamblador para una CPU en otra CPU sería necesario reescribirlo de nuevo. Si está escrito en un lenguaje de alto nivel es necesario recompilarlo.

  Por lo general **las instrucciones realizan operaciones muy básicas**, pero **la potencia de un ordenador es la de poder ejecutar muchas a la vez**, por lo que cálculos complejos los resuelve descomponiendo la operación en un conjunto de muchos cálculos sencillos

Juego de instrucciones

Puedes consultar en el siguiente enlace https://es.wikipedia.org/wiki/Anexo:Instrucciones_x86 el juego de instrucciones introducidas con el microprocesador de **Intel 8086 en el año 1979** y que incluyen muchos de los microprocesadores fabricados desde entonces hasta la actualidad.

A este juego de instrucciones de le conoce como x86 y verás que hay varios tipos de juegos de instrucciones:

- De movimiento de datos
- De operaciones aritméticas y lógicas
- Control de flujo de programa
- Control de interrupciones
- …

Ejemplos de programas en ensamblador

Puedes ver algunos ejemplos de programas codificados con el juego de instrucciones x86 en el siguiente enlace. A la codificación de los lenguajes directamente en el juego de instrucciones de le llama codificar en **lenguaje ensamblador**, del que hablaremos más adelante:

```
 ORG 100h
 mov ax, 10 ;AX=10
 mov bx, 00F9h ;BX=0xF9
 inc bx ;BX++
 add ax, 4 ;AX=AX+4
 mov cx,45 ;CX=45
 sub cx,cx ;CX=CX-CX
ret
```

Ejemplo de uso de los registros de propósito general AX, BX Y CX para almacenar datos y hace operaciones aritméticas

```
name "potencia" ;8 chars DOS
org 100h ;counter to 100h
mov cx, num2
mov ax, num1
inicio:
 mov bx,num1
 mul bx ;ax = ax * bx
 loop inicio ;c--

mov num3,ax ;copiamos el resultado
ret
;Variables "db" para byte y "dw" para word
num1 dw 0Ah
num2 dw 03h
num3 dw 0h
```

Ejemplo de programa de cálculo de potencia

https://www.programacion.com.py/escritorio/ensamblador/ejemplos-de-programas-en-ensamblador-8086

Funcionamiento de la CPU

La función de la CPU es ejecutar las instrucciones de los programas en ejecución den el ordenador. Estas instrucciones están almacenadas secuencialmente en posiciones consecutivas de memoria para ser ejecutadas una tras otra.

La tarea de la CPU consistirá en ir extrayendo sucesivamente las instrucciones de la memoria, interpretarlas, extraer de memoria todos los datos implicados en cada operación, enviarlos a la unidad que realiza las operaciones y hallar el resultado.

La ejecución de una instrucción en la CPU sigue las siguientes fases:

- **Carga (Fetch)** → lee de la memoria la instrucción almacenada en la dirección indicada por el contador de programa y la almacena en el registro de instrucciones.
- **Decodificación** → la instrucción es enviada al decodificador que determinar qué instrucción es y por tanto qué se debe hacer.
- **Lectura de operandos** → si la instrucción lo requiere lee de la memoria los datos que sean necesarios para su ejecución.
- **Ejecución** → bajo la supervisión de la UC se ejecutan las acciones necesarias asociadas a la instrucción, como puede ser el caso de las operaciones en la ALU.
- **Escritura de los resultados** en la memoria principal o en los registros tras la ejecución

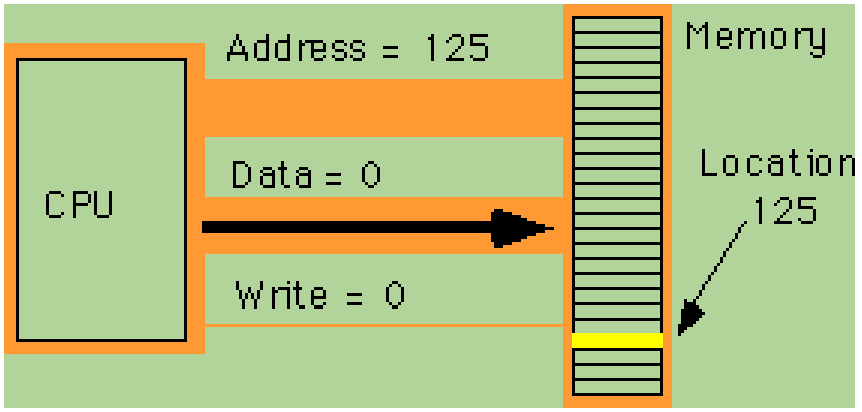
Memoria principal

Memoria principal

- En su definición más amplia memoria es cualquier dispositivo capaz de almacenar información, aunque en la arquitectura Von Neumann la memoria principal se refiere a la que almacena las instrucciones y los datos del programa que ejecuta la CPU
- En un ordenador encontraremos múltiples memorias además de la memoria principal que es la que se considera como imprescindible para el funcionamiento de un ordenador de acuerdo a la arquitectura Von Neumann
- Esto hace que la memoria principal sea el núcleo del sistema de memoria de un sistema informático

Direccionamiento de la memoria

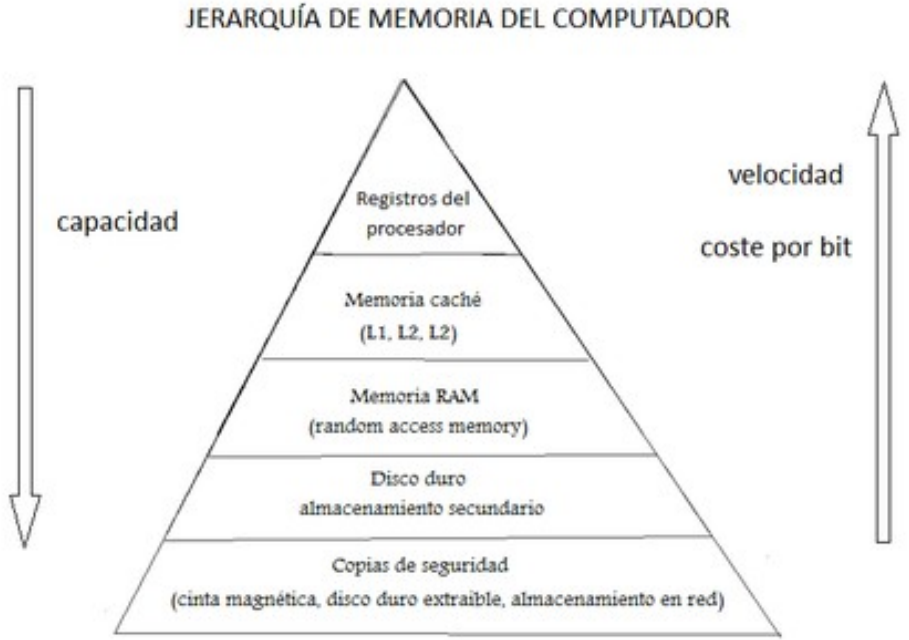
- Una forma común de describir la memoria principal de un ordenador es como una colección de celdas que almacenan datos e instrucciones, en la que cada celda está identificada unívocamente por un número o dirección de memoria.
- Para que la CPU pueda leer o escribir datos de la memoria debe conocer la dirección donde está almacenado el dato a leer o donde lo pretende escribir. La comunicación de la dirección afectada entre la CPU y la memoria se hace través del **bus de direcciones**, que es un bus unidireccional que va sólo de la CPU a la memoria



- Es por esto que **la anchura del bus de direcciones** condiciona la cantidad de direcciones que la CPU es capaz de direccionar, y por tanto el tamaño máximo de memoria que es capaz de utilizar. Actualmente típicamente son de 32 y 64 bits:
  - 32 bits → 2^{32} posiciones de memoria → 4 gigabytes de tamaño máximo
  - 64 bits → 2^{64} posiciones de memoria → 18,4 exabytes de tamaño máximo de memoria (18.400.000.000.000 gigabtyes).

Jerarquía de la memoria en un ordenador

- Si bien la arquitectura Von Neumann se refiere sólo a la memoria principal en un ordenador encontraremos **varios tipos de memorias de distintos tipos según la tecnología que utilizan**.
- **Cada tipo tiene una velocidad de acceso distinta**, lo que es un parámetro crítico en lo que al rendimiento del sistema se refiere. Esto tiene un impacto en su coste de fabricación que implica que **las memorias suelen tener un coste más elevado por unidad de almacenamiento cuanto más rápidas sean**
- Es por esto que con el fin de **buscar un equilibrio entre rendimiento y coste** se establece para la memoria de un ordenador una jerarquía compuesta por varios niveles. Dependiendo del nivel en el que ubiquemos una memoria indicará el uso que se hará de la misma



Descripción de cada nivel de la jerarquía de la memoria

De menor a mayor capacidad y de mayor a menor coste por unidad de almacenamiento se establece la siguiente jerarquía:

- **Registros de la CPU (bits)**: Son registros internos de la CPU, fabricados con semiconductores, de muy poca capacidad de almacenamiento (medido en bits) pero con una velocidad de acceso enormemente alta. El acceso a estas memorias es aleatorio y por palabra.
- **Memoria caché (KB)**: Memorias de acceso aleatorio, muy rápidas pero de poca capacidad que tratan de limitar la cantidad de lecturas que hace el procesador a la memoria principal. Es habitual que estén organizadas en varios niveles y que incluso formen parte del mismo microprocesador.
- **Memoria principal (GB)**: mediana y rápida. Son memorias de semiconductores de acceso por palabra. Existen dos tipos: la memoria ROM, a la que se accede al iniciar el ordenador, y la memoria RAM, donde se almacenan los programas en ejecución y los datos usados por esos programas.
- **Memoria secundaria local (GB/TB)**: Más lenta que la memoria principal, su función principal es la de almacenar los programas y datos que se podrán cargar en la memoria principal para su ejecución (Discos duros mecánicos, SSD, almacenamiento óptico, ...)
- **Memoria secundaria remota (GB/TB)**: copias de seguridad, sistemas de almacenamiento distribuido, servicios web, almacenamiento en la nube, etc. Realmente no ser trata de una memoria física del ordenador, pero conceptualmente podría considerarse como un nuevo nivel de memoria, de acceso más lento que los anteriores, precio cada vez menor y capacidad prácticamente ilimitada.

Unidad de Entrada/Salida (E/S)

Tipos de dispositivos

- Los periféricos son cualquier elemento de un ordenador que se comunican con la CPU a través de la unidad de E/S
- Los hay de muchos tipos distintos pero todos sirven para interactúar de distintas formas con los programas que están cargados memoria principal y en ejecución
- La principal clasificación de estos dispositivos es la que los divide según la dirección del flujo de información del ordenador hacia estos:
  - Los **dispositivos de entrada** recogen algún tipo de información externa al ordenador y este la recoge y la puede utilizar para controlar o cambiar el flujo de ejecución del programa. Ejemplos de estos son: Ratón, teclado, micrófono, ...
  - Los **dispositivos de salida** los utiliza el programa para mostrar algún tipo de resultado. Ejemplos de estos son: Monitor, impresora, altavoces, …
- También los hay mixtos, que son de entrada y salida a la vez, como puede ser una pantalla táctil

Unidad de E/S como interfaz entre CPU y periféricos

- Hay una gran cantidad de dispositivos de E/S que atienden a necesidades muy variadas (Teclados, ratones, monitor, impresora, tarjeta de red, ...) y cada uno de ellos tiene sus propias características de operación
- Es por esto que **resulta difícil que la CPU tenga la capacidad de adaptarse a todos** ellos para controlarlos a todos de forma nativa
- Es por eso que para simplificar la comunicación de la CPU con estos dispositivos la arquitectura propone la **Unidad E/S**, como elemento que **actúa como interfaz de comunicación** y que tiene la capacidad de adaptarse a las particularidades de cada dispositivo
- Esta unidad no sólo se encarga de la gestión de la comunicación a bajo nivel con los dispositivos a través de los conectores y puertos utilizados, sino que contienen la lógica necesaria para controlar los intercambios de información, almacenarla temporalmente (necesario debido a la diferencia de velocidades a las que operan CPU y periféricos), adaptar si es necesario los códigos usados por CPU y periférico y detectar errores.

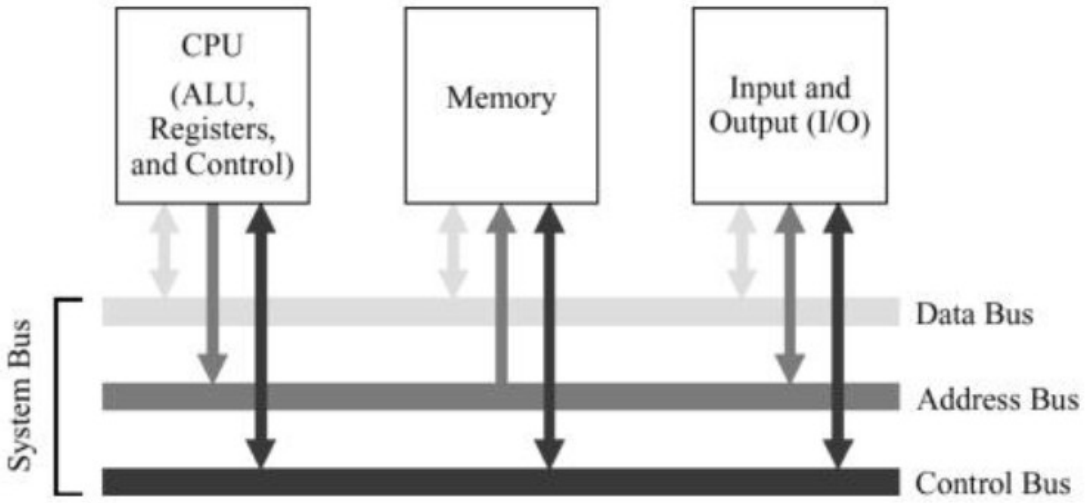
Buses de comunicación

Buses de comunicación

- Los buses son los circuitos (enlaces y conmutadores) encargados de interconectar todos los componentes de un ordenador, permitiendo el envío de datos, direcciones y señales de control entre todos ellos.
- En su implementación en un ordenador real la mayoría de los buses están basados en conductores metálicos por los cuales se trasmiten señales eléctricas.
- La operación básica del bus se denomina ciclo del bus y permite realizar una transferencia elemental entre dos de los dispositivos conectados al bus.

Clasificación de los buses en base a su utilización

- **Bus de datos**: Encargado de transmitir los datos (operandos e instrucciones). Su anchura determina la longitud de la instrucción. Es **bidireccional**, ya que los datos pueden fluir hacia o desde la CPU y cualquier otro elemento del modelo.
- **Bus de direcciones**: Es **unidireccional** (la información se envía siempre de CPU a memoria o a los elementos de E/S), y se utiliza para enviar las direcciones a leer o escribir en la memoria. El número de líneas que lo componen (ancho del bus) determinará la cantidad máxima de memoria que la CPU pueda direccionar y por tanto con la que es capaz de trabajar.
- **Bus de control**: Es un bus **bidireccional** que transmite por un lado las señales de control generadas por la Unidad de Control para la realización de una operación y por otro lado las señales de estado que indicarán en qué modo se encuentran los dispositivos.



Características de los buses

Son múltiples las características de los buses según sus implementaciones, pero hay algunos que aplican de forma genérica a todos y que permite clasificarlos y diferenciarlos:

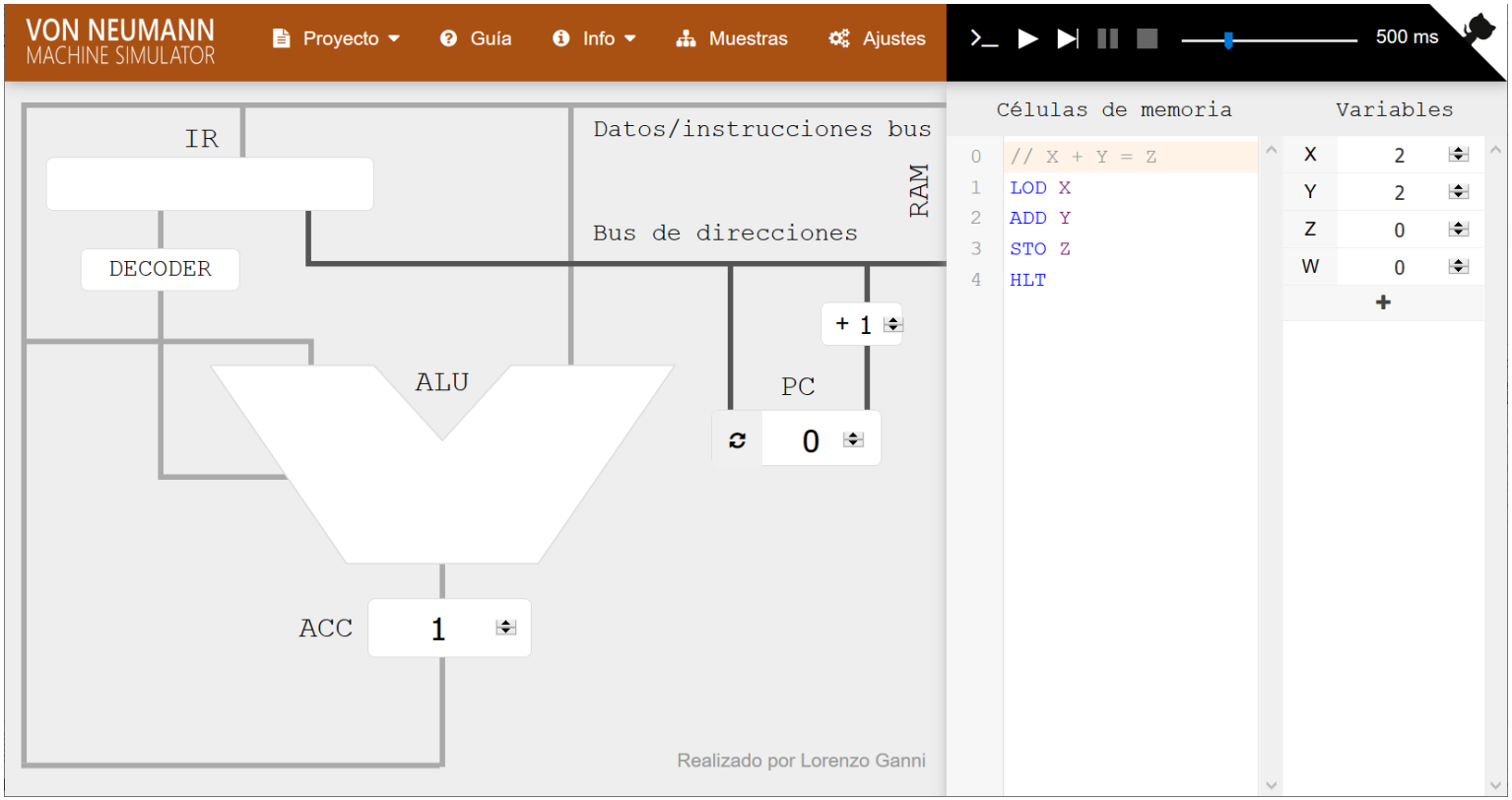
- **Anchura del bus/Grado de paralelismo** → Indica el número de bits que se puede enviar a través del bus en un mismo ciclo. Puede ser:
  - **Serie** (1 bit)
  - **Paralelo** (Varios bits a la vez que agrupados conforman un elemento con significado llamado palabra)
- **Frecuencia** → Que indica el número de ciclos/envíos por segundo que es capaz de realizar por segundo y que se mide en Hertzios, típicamente en Mhz.
- **Velocidad de transmisión** → Que se obtiene a partir de la anchura de bus y su frecuencia, tal como ya vimos

Otros parámetros más concretos nos indican otros parámetros como el material con el que están fabricados, el número de dispositivos que podemos conectar, si la comunicación en síncrona o asíncrona, ...

Ejecución de programas en simulador de arquitectura Von Neumann

Descripción del simulador

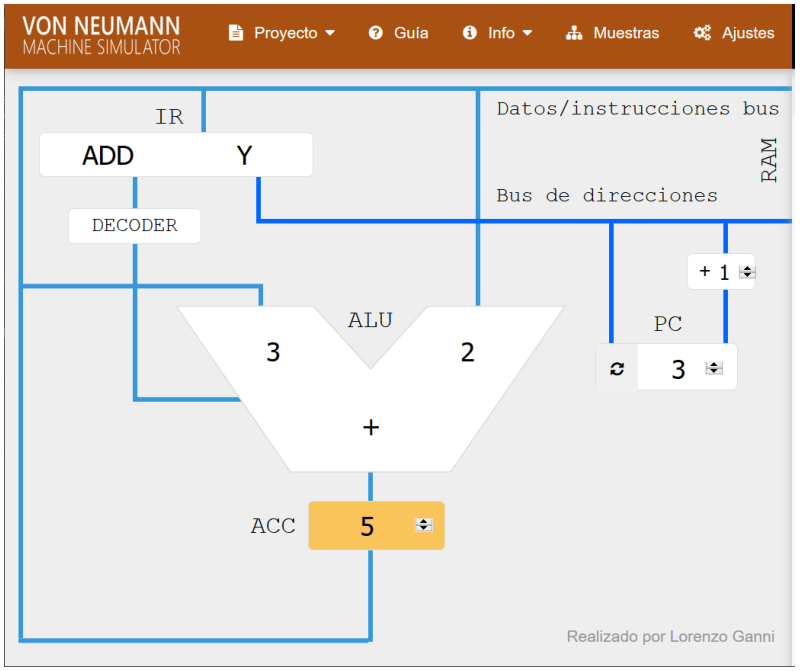
- Para probar el funcionamiento de un ordenador vamos a utilizar un simulador web de máquina según arquitectura Von Neumann → http://vnsimulator.altervista.org/
- Es un simulador con **licencia MIT**, por lo que el código fuente está disponible y si quisiéramos podríamos basarnos en él para crear una nueva versión y distribuirla
- Es un simulador para que el que podemos escribir nuestros propios programas en **un lenguaje similar al ensamblador** para el que define un **juego propio de instrucciones muy simplificadas**



Descripción del simulador (Registros y memoria principal)

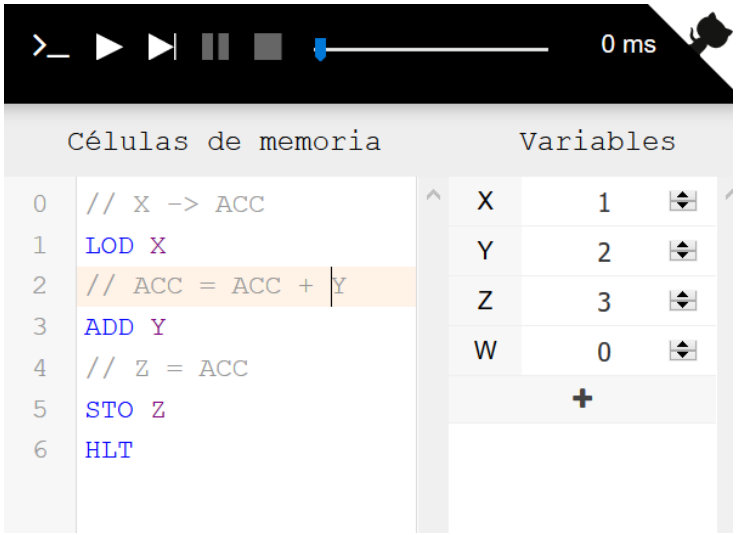
Representa los **registros** más básicos:

- **PC (Program counter)** → Almacena la dirección de memoria que contiene la siguiente instrucción a ejecutar
- **IR (Instrucion Registry)** → Instrucción actualmente en ejecución
- **ACC (Acumulator)** → Almacena los resultados finales e intermedios de los cálculos en la ALU



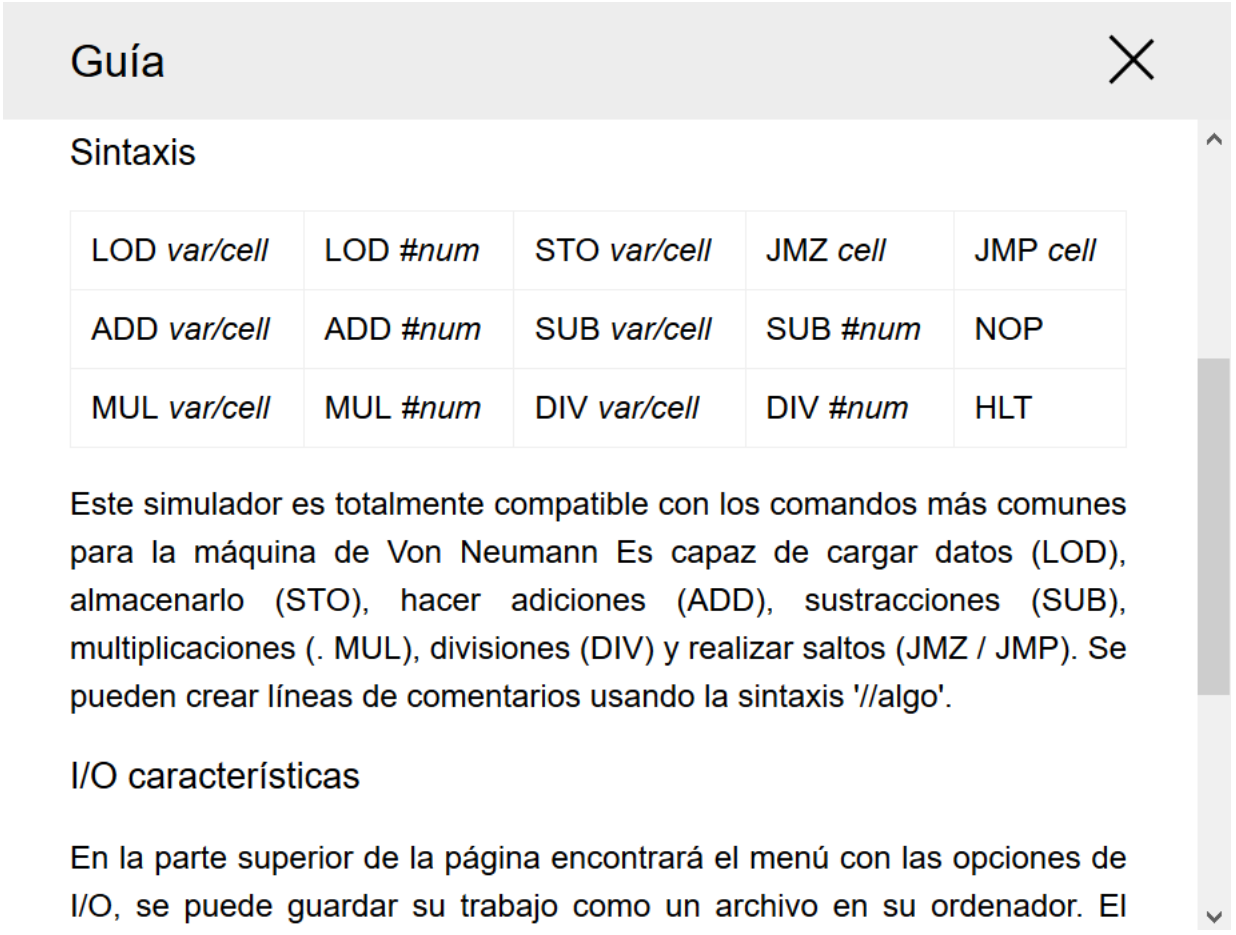
La **memoria principal** aparece dividida en dos partes y las podemos editar manualmente con libertad:

- **Células de programa** → Almacena las instrucciones del programa que ejecuta
- **Variables** → Almacena los datos que utiliza el programa



Descripción del simulador (Juego de instrucciones)

Esta maquina dispone de un juego de instrucciones limitado, idóneo para pensar como a la hora de compilar un programa es necesario descomponer las instrucciones en operaciones muy básicas



Guía

Sintaxis

LOD *var/cell*	LOD *#num*	STO *var/cell*	JMZ *cell*	JMP *cell*
ADD *var/cell*	ADD *#num*	SUB *var/cell*	SUB *#num*	NOP
MUL *var/cell*	MUL *#num*	DIV *var/cell*	DIV *#num*	HLT

Este simulador es totalmente compatible con los comandos más comunes para la máquina de Von Neumann Es capaz de cargar datos (LOD), almacenarlo (STO), hacer adiciones (ADD), sustracciones (SUB), multiplicaciones (. MUL), divisiones (DIV) y realizar saltos (JMZ / JMP). Se pueden crear líneas de comentarios usando la sintaxis '//algo'.

I/O características

En la parte superior de la página encontrará el menú con las opciones de I/O, se puede guardar su trabajo como un archivo en su ordenador. El

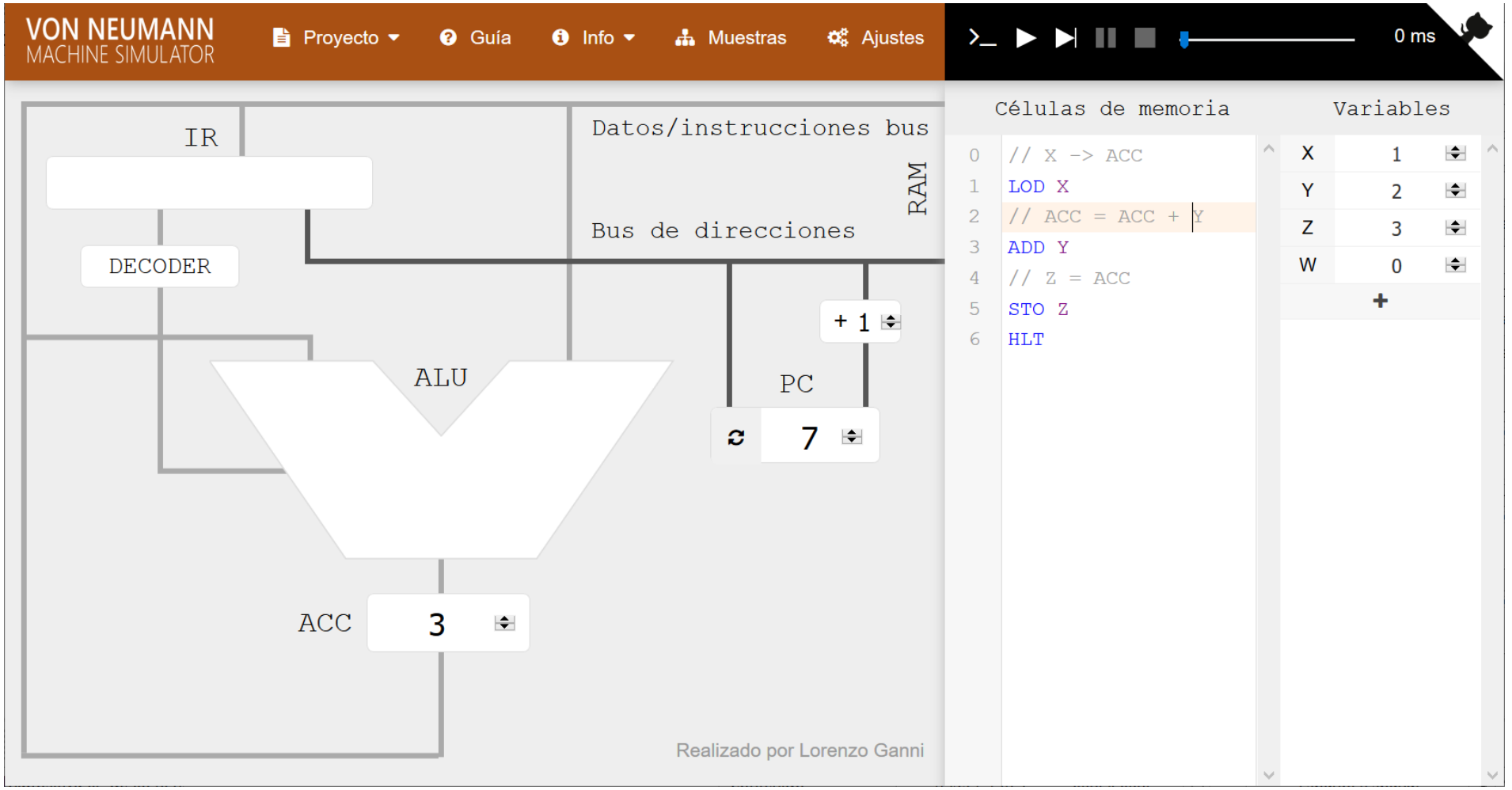
Instrucciones de operaciones aritméticas

Para hacer **operaciones aritméticas**:

- Primero tenemos que cargar el valor del primer operando con la instrucción **LOD (Load)** que es guardado en el registro ACC. Ejemplos:
  - LOD X → Cargamos en ACC el valor de la variable X
  - LOD #3 → Cargamos en ACC el valor 3
- Después se ejecuta la operación aritmética con las instrucciones **ADD, SUB, MUL o SUB** que hará la operación entre lo que está almacenado en ACC y lo que se le pasa a la instrucción. El resultado se guarda en ACC. Ejemplos:
  - ADD Y → Suma Y y ACC y guarda el resultado en ACC (ACC = ACC+Y)
  - SUB #1 → Resta 1 a ACC y guarda el resultado en ACC (ACC = ACC-1)
- Opcionalmente podemos guardar lo que tenemos en ACC en una variable en memoria con la instrucción de carga STO (Store):
  - STO Z → Guarda el resultado en la variable Z

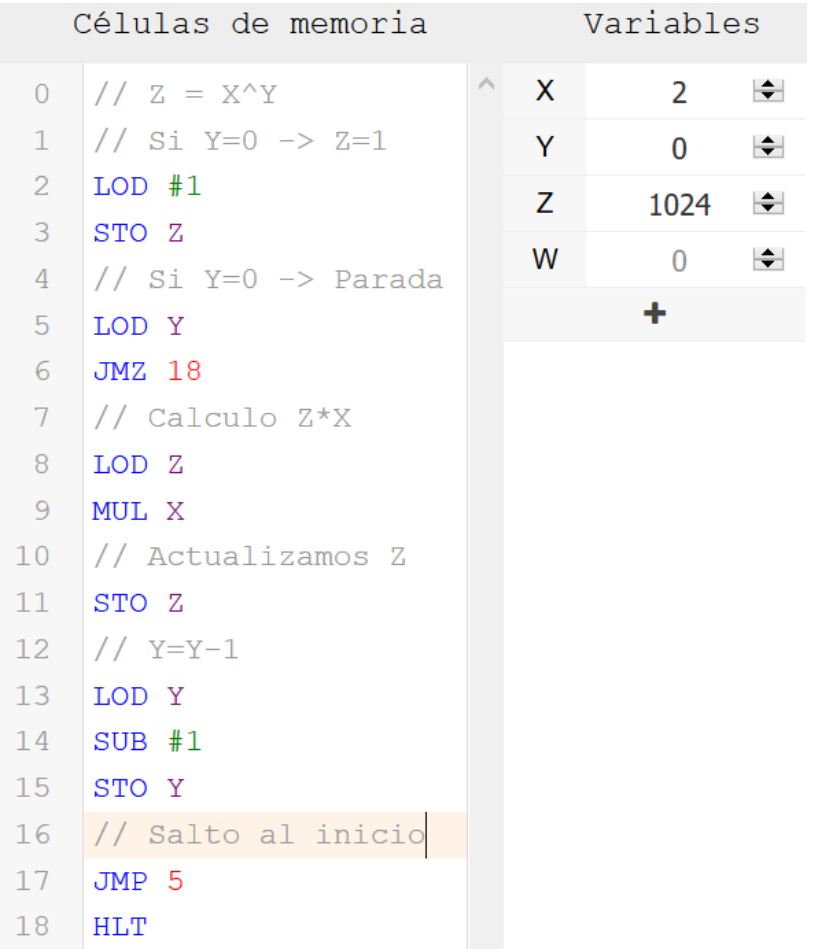
Instrucciones de operaciones aritméticas

Ejemplo de suma de los valores almacenados en las variables X e Y y escritura del resultado en la variable Z:



Instrucciones de salto JMP y JMZ

- En una ejecución normal las instrucciones se ejecutan secuencialmente en el orden en el que están en memoria porque el registro PC se incrementa en 1 para avanzar la lectura a la siguiente dirección.
- Este flujo de ejecución secuencial lo podemos cambiar con las instrucciones de salto:
  - **JMP cell** → Cambia el valor de la dirección de memoria almacenada en el PC por el de cell
  - **JMZ cell** → Realiza el salto sólo si el valor almacenado en el registro ACC es 0
- Las instrucciones de salto son básicas para la realización de bucles que entre otras cosas nos permitirán descomponer operaciones complejas en otras más simples
- Ejemplo: Calculo de la potencia descompuesta en una sucesión de multiplicaciones



Otras instrucciones

- **NOP**

  Es una instrucción que significa **NO-OPERATION**, en la que la CPU no ejecutará ninguna instrucción. El simulador la utiliza cuando se encuentra una dirección de memoria en la que hay un comentario, precedido por //

- **HLT**

  Es una instrucción que significa **HALT**, que sirve para indicar que el programa ha finalizado la ejecución y debe detener la ejecución de instrucciones

Tarea: Codificación de programas en el simulador I

Vamos a utilizar el simulador de máquina Von Neumann para familiarizarnos con el funcionamiento de la ejecución de un programa en un ordenador y el tipo de instrucciones de bajo nivel con las que trabaja una CPU. Para esto tienes que hacer:

- Abre los ejemplos de suma, resta, multiplicación o división y pruébalos para familiarizarte con el funcionamiento del simulador
- Una vez vistos y entendidos estos puedes abrir otros ejemplos más complejos para que trates de entenderlos
- Entendido los rudimentos del funcionamiento del simulador y su limitado lenguaje **vamos a poner en práctica lo que supone tener que codificar con un juego de instrucciones muy simple**.

Tienes que hacer los siguientes programas:

Programa 1:

- Un programa que haga la división de X DIV Y guardando su valor en Z. Además deberá calcular el resto de esa división y guardarlo en la variable W

Programa 2:

- Añadir al programa BASIC_OPERATIONS la capacidad de hacer el resto de una división que hiciste antes
  - BASIC_OPERATIONS hace una operación entre X y Z dependiendo del valor de Y:
    - Si Y es 0 hace una suma
    - Si Y es 1 hace una resta
    - Si Y es 2 hace una multiplicación
    - Si Y es 3 hace una división
  - Modifícalo para que si Y es 4 además pueda calcular el resto de la división entre X y Z
- **Entregarás**:
  - Los archivos con las instrucciones resultado de exportarlos desde la web. Deberán ser totalmente operativos
  - Un documento en el que tendrás que explicar el funcionamiento del primer programa que has hecho. Pondrás una captura del simulador tras la ejecución de cada una de las instrucciones explicando con tus palabras lo que se ha hecho en él (Qué valores ha escrito en cada registro o memoria, operación realizada, los buses utilizados, ...)

Tarea opcional: Codificación de programas en el simulador II

Opcionalmente trata de implementar los siguientes programas:

- **Programa3**: Un programa que multiplique dos números:
  - Para esto **no podrás utilizar la instrucción MUL**, sino que tendrás que hacer la multiplicación como la suma de un número varias veces. Para esto tendrás que **utilizar instrucciones de salto**.
  - El simulador trae un **ejemplo de cálculo de potencias (POWER) que puedes utilizar como referencia**
  - Fíjate que el ejemplo de potencias tiene en cuenta la casuística de que una potencia con exponente 0 siempre es 1. En el caso de una multiplicación un número multiplicado por 0 debe devolver 0
- **Programa 4**: Vamos a seguir añadiendo capacidad de cálculo a nuestro programa BASIC_OPERATIONS incorporando la capacidad de hacer potencias y raíces cuadradas
  - A mayores de las que ya tiene y que añadiste antes para el resto BASIC_OPERATIONS deberá calcular:
    - Si Y es 5 hace la potencia de X^Y
    - Si Y es 6 hace la raíz cuadrada de X
  - Para esto incorpora y adapta las instrucciones del programa de los programas de ejemplo POWER para la potencia y SQUARE_ROOT para la raíz cuadrada

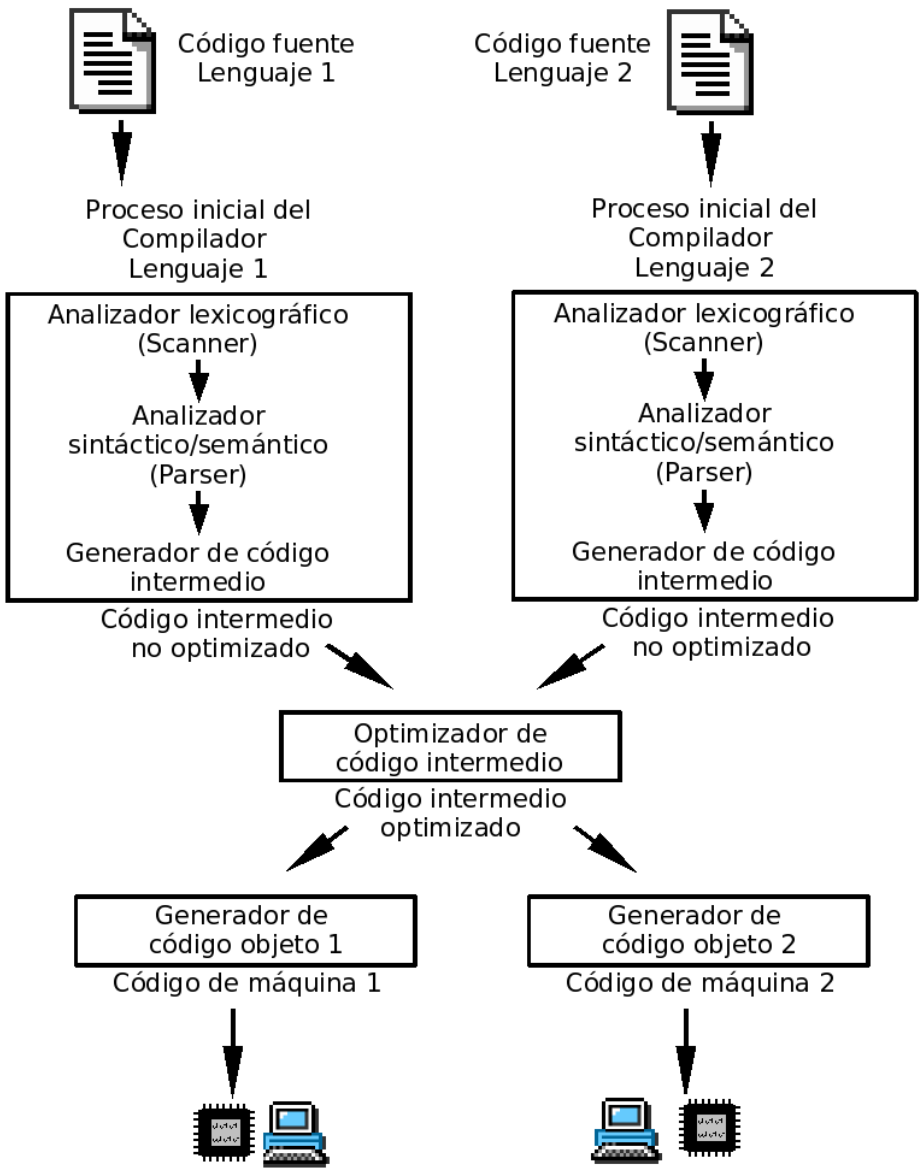
Lenguajes de programación

Código máquina y lenguajes de programación

- Para poder llegar a tener un programa que se carga en memoria necesitamos la figura de un programador, que se encarga de codificar las instrucciones de dicho programa con los que tenemos cierto control sobre el comportamiento del ordenador
- El problema a la hora de codificar un programa es que **el hardware sólo entiende código máquina**, que es un conjunto de códigos directamente interpretable por los circuitos electrónicos del ordenador
- Los problemas del lenguaje máquina son:
  - Que **debe estar escrito en binario, 0's y 1's**,
  - **Las instrucciones que podemos utilizar en lenguaje máquina son muy básicas**, de acuerdo al juego de instrucciones de la CPU, por lo que cualquier cálculo complejo lo tenemos que descomponer en varios cálculos básicos
- Aunque los programadores pueden codificar directamente en este lenguaje máquina lo habitual es utilizar **lenguajes de programación**:
  - Simplifican la escritura de programas utilizando una codificación más intuitiva que el binario
  - Cálculos que para lenguaje máquina son complejos con un lenguaje de programación son básicos

Compiladores

- Los lenguajes de programación por tanto ofrecen una forma más sencilla de escribir programas sin tener que aprender código máquina, pero aún así todo programa debe ser traducido en algún momento a lenguaje máquina
- Esta traducción, que se denomina **pasar de código fuente a código máquina**, es realizada por lo que se denomina un **compilador**
- Cada lenguaje de programación tendrá su propio compilador, ya que cada lenguaje tiene sus propios símbolos y sintaxis que el compilador debe ser capaz de verificar para traducirla a código máquina



Lenguaje de programación de bajo y alto nivel

Hay múltiples formas de tipificar los lenguajes de programación, pero para empezar vamos a centrarnos en una **clasificación basada en su nivel de abstracción con respecto a la máquina**, esto es, cuando más cerca o lejos estamos de codificar directamente en lenguaje máquina

- **Lenguajes de bajo nivel**

  Son lenguajes en los que utilizamos algún tipo de lenguaje mnemónico que es directa o fácilmente traducible a lenguaje máquina

- **Lenguajes de alto nivel**

  Permiten escribir programas utilizando un lenguaje muy parecido al humano que está muy alejado del lenguaje máquina, por lo que resulta mucho más intuitivo para el programador

Visto así podemos pensar que si ya tenemos lenguajes de alto nivel ¿para que queremos los de bajo nivel?, pero veremos que los de bajo nivel aún tienen su utilidad hoy en día

Ventajas y desventajas de lenguajes de alto y bajo nivel

	Lenguajes de bajo nivel	Lenguajes de alto nivel
Ventajas	• Son **más rápidos** porque los programadores pueden aprovechar los recursos de la CPU y la memoria de una forma más eficiente • Prácticamente **no hacen falta compiladores**, por lo que nos ahorramos el proceso de compilación	• Utilizan lenguajes más parecidos al **lenguaje natural**, lo que hace que sean más fáciles de aprender • Son **independientes del hardware** y abstraen al programador de tener que conocer detalles de este • Son más **fáciles de escribir, depurar y mantener** • Es más fácil obtener avances significativos en poco tiempo
Desventajas	• Los programas son muy dependientes de la máquina para los que se han hecho, lo que los hace difícilmente reutilizables en otras máquinas • Son difíciles de escribir, depurar y mantener los que los hace más sensibles a errores • Resultan poco productivos, hay que dedicar mucho tiempo para hacer algún avance significativo • Los programadores deben tener un conocimiento acerca del hardware y la arquitectura del ordenador para el que hacen el programa	• Requieren pasar por un proceso de compilación que en ocasiones puede resultar costoso • En comparativa con los de bajo nivel son menos eficientes en uso de recursos y por tanto más lentos, porque necesitan más memoria y/o CPU • La comunicación con el hardware no es directa, esta se hace de forma indirecta a través de interfaces
Usos	A pesar de sus desventajas se utilizan para desarrollar programas muy críticos que deben ser muy eficientes en lo que a uso de recursos hardware se refiere, como el caso del desarrollo de sistemas operativos o firmware de dispositivos	En el desarrollo de utilidades y aplicaciones para PC, dispositivos móviles, páginas web, ...
Ejemplos	Código máquina, Ensamblador y C	C, C++, Java, Ruby on Rails

Lenguaje C

- En los ejemplos de lenguajes de programación anteriores se tipificó el lenguaje C como de bajo y alto nivel, esto es porque:
  - Dispone de las estructuras típicas e lenguajes de alto nivel
  - Dispone de construcciones del lenguaje que permiten un control a muy bajo nivel
  - Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.
- Además C es un lenguaje que genera un código muy eficiente, lo que lo convierte en un lenguaje idóneo para desarrollar sistemas operativos

Ejemplos en Ensamblador

Vemos algunos ejemplos de instrucciones simples codificadas en ensamblador. En este lenguaje tenemos la capacidad de escribir y leer directamente de los registros de la CPU indicando el nombre de los mismos

```
INC COUNT          ; Increment the memory variable COUNT

MOV TOTAL, 48      ; Transfer the value 48 in the
                   ; memory variable TOTAL

ADD AH, BH         ; Add the content of the
                   ; BH register into the AH register

AND MASK1, 128     ; Perform AND operation on the
                   ; variable MASK1 and 128

ADD MARKS, 10      ; Add 10 to the variable MARKS
MOV AL, 10         ; Transfer the value 10 to the AL register
```

Programa Hola Mundo. Ensamblador vs Java y C++

- El programa Hola mundo suele ser el primer programa que se hace en los tutoriales y manuales de todos los lenguajes de programación. Es un programa sencillo que entre otras cosas sirve para probar que tenemos el entorno de programación bien configurado
- Vamos a ver ejemplos de este programa en un lenguaje de bajo nivel (Ensamblador) y dos de alto nivel (Java y C++)

```
; Hola mundo en Ensamblador
section .text
   global _start      ;must be declared for linker (ld)

_start:               ;tells linker entry point
   mov   edx,len      ;message length
   mov   ecx,msg      ;message to write
   mov   ebx,1        ;file descriptor (stdout)
   mov   eax,4        ;system call number (sys_write)
   int   0x80         ;call kernel

   mov   eax,1        ;system call number (sys_exit)
   int   0x80         ;call kernel

section .data
msg db 'Hola, mundo!', 0xa  ;string to be printed
len equ $ - msg      ;length of the string
```

https://www.tutorialspoint.com/assembly_programming/assembly_basic_syntax.htm

```
// Hola mundo en Java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hola mundo");
  }
}
```

http://holamundo.es/lenguaje/java/

```
// Hola mundo en C++
#include <iostream>
using namespace std;

int main() {
    cout << "Hola Mundo" << endl;
    return 0;
}
```

http://holamundo.es/lenguaje/cpp/