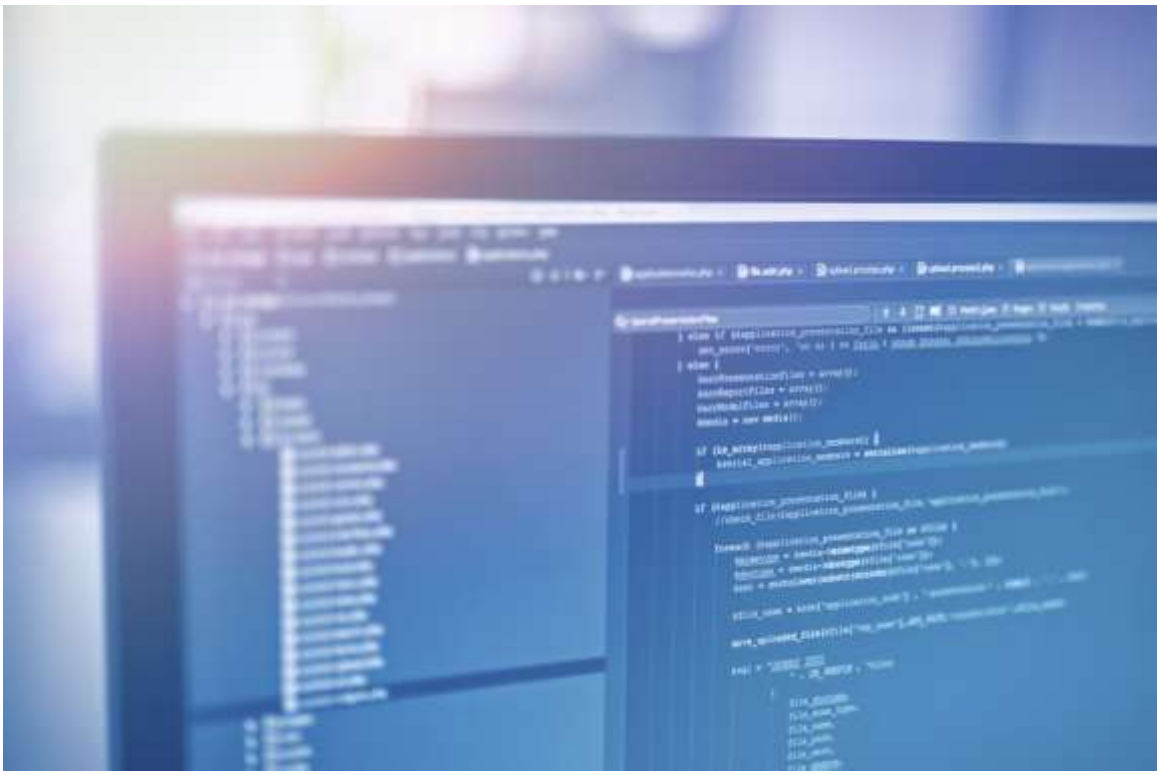
Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

Lenguaje ensamblador

Contenido

Presentación



En este recurso se va a estudiar el **ensamblador**, un lenguaje de programación de bajo nivel y muy dependiente del microprocesador con el que se trabaje.

Al ser un lenguaje de muy bajo nivel, está muy relacionado con el **hardware** y la **arquitectura del microprocesador**. Por eso, el estudio del lenguaje ensamblador y del microprocesador va de la mano.

Lo ligado que el ensamblador está al microprocesador es, a la vez, su punto fuerte y su debilidad. Esta característica hace que programar en ensamblador **genere un código muy óptimo y eficiente**, teniendo el programador la decisión de todo. Por el contrario, aprender ensamblador para un microprocesador no asegura que se pueda programar para otra familia de procesadores, ya que el lenguaje y sus características cambian.

Para estos contenidos se ha decidido utilizar familia de microcontroladores **Atmel "ATMega"** como hardware de estudio, por su **ensamblador simple, mapa de memoria único, arquitectura fácil** y por su bajo coste. Todas estas características facilitan que el alumno tenga su propio microprocesador para hacer las prácticas.

Para los ejemplos se usa el **Genuino Uno** como base. Pero serán compatibles con cualquier Arduino con pequeñas variaciones.

A continuación, se pasa a describir de manera genérica qué es el lenguaje ensamblador.

Objetivos

Los **objetivos** que se pretenden alcanzar en este tema son los siguientes:

- Introducir la **programación ensamblador** analizando sus conceptos desde un punto de vista teórico sin ligarlo a un microprocesador en concreto.
- Comprender el **concepto de programa** así como la secuencia básica de ejecución del mismo.
- Diferenciar y analizar los distintos **tipos de direccionamientos**.
- Conocer los distintos **modelos de ejecución** que originan las diversas tecnologías de microprocesadores.

Código máquina

Un **computador** es una máquina diseñada para **ejecutar programas**, solo sabe ejecutar programas y siempre tiene que estar ejecutando algún programa. Esto implica que nunca se detiene, siempre está en ejecución.

Un **programa** no es más que una **secuencia de instrucciones** simples, indivisibles que **realizan una única operación.** Pero, para poder ejecutar estas instrucciones exige que tanto estas como los datos que se necesiten, residan en la **memoria principal**. Dado que el componente que ejecuta las instrucciones es el microprocesador, surge una necesidad directa para que estas instrucciones se ejecuten. Tienen que viajar desde la memora al **microprocesador**, y una vez ahí, tienen que almacenarse temporalmente para poder ejecutarlas completamente.

Por tanto, la **secuencia básica de ejecución** de cualquier programa se compone de los siguientes pasos, que se repiten sucesivamente.

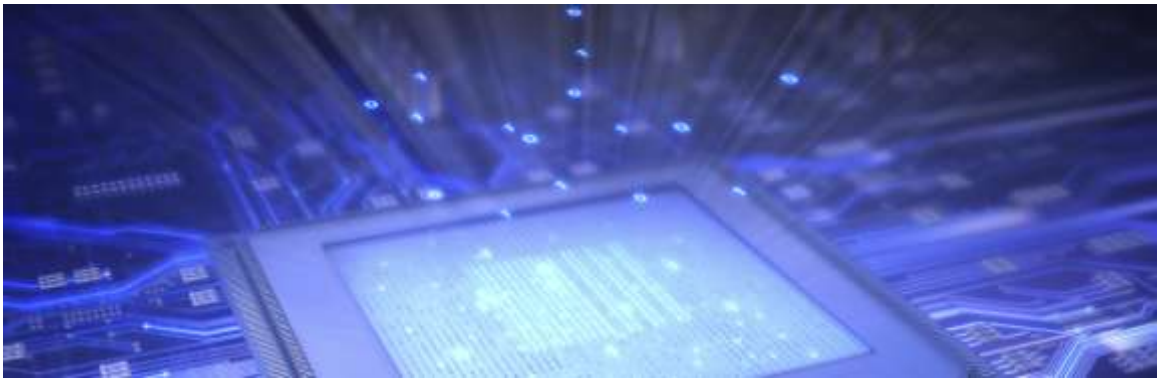
Lectura
Se lee una **instrucción o sentencia** del programa de la **memoria principal**.
Interpretación
La **unidad de control** (uno de los componentes que están dentro del microprocesador) **interpreta** la instrucción leída.
Ejecución
Bajo las órdenes de la unidad de control se **ejecuta la instrucción**, por ejemplo, pidiendo a la ALU (Unidad Aritmética Lógica) que realice una operación de suma.

Pero recordemos:

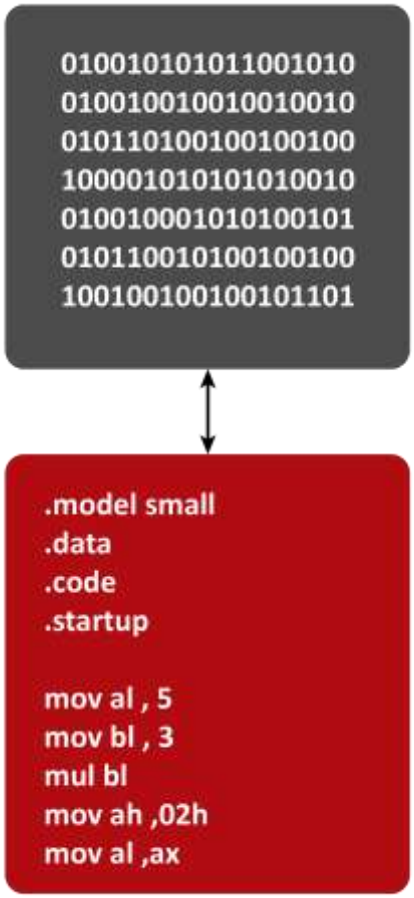
- Los computadores **no son capaces de interpretar** directamente un lenguaje de alto nivel. Por el contrario, solamente "entienden" un lenguaje muy restringido, que se denomina **lenguaje máquina**.
- Además, ¿qué se puede guardar en la memoria? **Bits**, por tanto, **números y solo números**.

Uniendo estas dos afirmaciones se llega a las conclusiones de que el programa se guarda en la memoria **como números** y de que el **lenguaje** que ejecuta un ordenador es "**código máquina**".

Ejemplo	**Juego de instrucciones** • **Instrucción 7E.** Representa la operación A ← M(HL). La instrucción ”7E” (en hexadecimal) perteneciente al microprocesador Z80, esta instrucción transfiere al registro “A” el contenido de la posición de memoria cuya dirección está almacenada en la pareja de registros [H,L]. • **Instrucción 3A3353.** Representa la operación M(0x5333) ← A. Instrucción del Z80 que transfiere a la posición de memoria principal “0x5333” el contenido del registro “A”. • **Instrucción 3C47**. Representa la operación R4 ← R4*R7. Instrucción del IBM 370 que multiplica el contenido de los registros R4 y R7, almacenando el resultado en R4.



Ensamblador



Llegados a este punto, tenemos la verdad más dura para el programador a bajo nivel. Si queremos programar un ordenador, **sin intervención de software intermedio** (compiladores) tenemos que generar nuestro programa como una **secuencia de ceros y unos inmensa,** sin estructura (no hay funciones, objetos, variables, etc.). Para colmo, el lenguaje máquina es diferente para cada tipo o familia de computadores, lo que provoca una incompatibilidad entre ellos.

Esta es una forma muy engorrosa de escribir programas y de manejar información. Por esto, emplearemos una simplificación de esta notación binaria, utilizando **notación hexadecimal** o notación simbólica tipo ensamblador para poder hacer códigos más legibles, de acuerdo al estándar del IEEE: "*Microprocessor Assembly Language Standard IEEE 694*".

¿Qué es Ensamblador?

El ensamblador lo podemos definir como un **lenguaje de programación**, como todo lenguaje de programación, está acompañado de un **software** que permite convertir de este lenguaje a **código máquina**.

Pero, más que un lenguaje de programación, es una **colección de nemotécnicos** que permiten escribir código de una manera más legible por el ser humano. Además, a diferencia del código máquina que solo permite el binario, el ensamblador permite escribir **números en decimal, hexadecimal, octal** y por supuesto **binario**.

En la imagen podemos apreciar la diferencia de lectura de un fragmento de programa escrito en código máquina, frente al mismo código escrito en ensamblador del Intel 286.

Retomando los ejemplos del caso anterior:

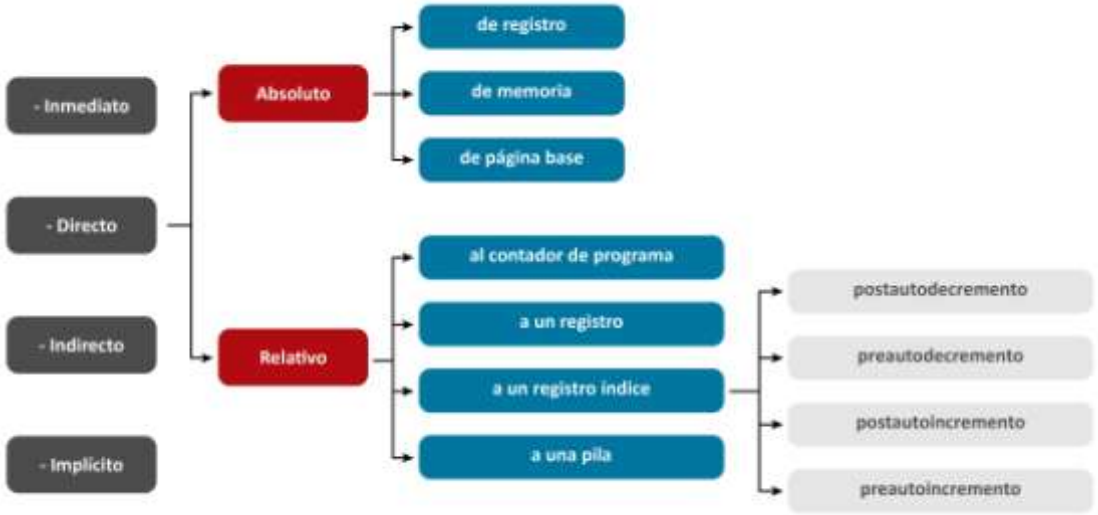
Características de las instrucciones en ensamblador
Las instrucciones en ensamblador suelen cumplir las siguientes características: • Cada instrucción realiza una única y sencilla función. • Emplean un número fijo de operandos, con una representación determinada y concreta. • La codificación de las instrucciones es muy sistemática y rígida. • Las instrucciones son autocontenidas e independientes, es decir, cada instrucción es independiente de la anterior y de la siguiente.
Información que han de contener las instrucciones
Dado que las instrucciones de máquina son autocontenidas, y que deben encadenarse para formar programas, la información que han de contener es: • Operación a realizar (suma, producto, transferencia, etc.). • Identificación de los operandos sobre los que debe realizarse la operación. • Destino del resultado; la identificación del lugar donde debe almacenarse. • Ubicación de la siguiente instrucción, normalmente este valor se auto calcula a partir de la propia instrucción. Si se sabe la longitud de ella, se sabe dónde termina.

Modos de direccionamiento

Las operaciones en ensamblador tienen operandos sobre los que realizar las operaciones. Estos operandos, muchas veces están codificados en la propia instrucción (multiplicar 3 por 5) o son registros (multiplicar R4 por 3). Pero en otros muchos casos, estos datos no son tan fáciles de localizar, porque pueden estar en memoria u otras ubicaciones.

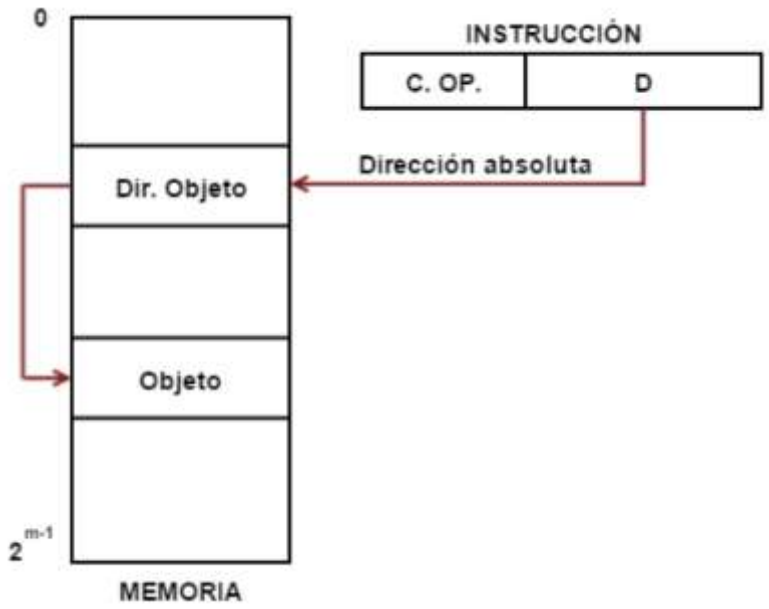
Los **modos de direccionamiento** hacen referencia a **dónde** y de **qué tipo** pueden ser los datos que intervienen en una operación.

Básicamente hay cuatro formas de referenciar un dato, tal como se puede ver en el siguiente esquema:



Inmediato
El operando se encuentra **contenido** en la **propia instrucción**, es un valor que se escribe numéricamente en la instrucción. Ejemplos: • LD A, #0x3A: Cargar el registro A con el valor inmediato 0x3A (58 en decimal). • MUL R2, #0x7C: Multiplicar el contenido del registro 4 por 0x7C y almacenar el resultado en el registro 4.

hace referencia a una posición de memoria.
Indirecto
El direccionamiento indirecto **comienza con un direccionamiento directo** (ya sea absoluto o relativo), pero se diferencia de aquél en que la dirección obtenida **no apunta al objeto deseado sino a su dirección**. Por tanto, para obtener el objeto deseado se requiere un acceso adicional a memoria principal. Ejemplo: • LD R4, [0x44]: Cargar el registro R4 con el contenido de la dirección de memoria que está indicada en la dirección de memoria 0x44.
Implícito
En el direccionamiento implícito, la instrucción **no contiene información sobre la ubicación del parámetro**, porque este está en un lugar predeterminado (registro o posición de memoria) o se sobreentiende su valor. La ventaja del direccionamiento implícito es que **no ocupa espacio en la instrucción**. Por el contrario, su inconveniente es que **restringe la aplicación** de la mencionada operación. Ejemplo: • INC R3: Incrementar el registro R3 en una unidad. En esta instrucción no se especifica el valor a incrementar, se sobreentiende que es en una unidad.
Directo absoluto
El direccionamiento directo absoluto indica que la instrucción contiene una **dirección efectiva sin compactar**. Este tipo de direccionamiento presenta **tres alternativas**: • La información contenida en la instrucción es el **identificador de un registro**. En alguna bibliografía a este modo se le conoce también como direccionamiento de registro. • La información contenida en la instrucción es una **dirección completa de memoria principal**. El número de bits necesarios para ello depende del mapa de direcciones del computador. • La información contenida en la instrucción es una **dirección limitada**, que permite referirse solamente a una parte del mapa de memoria. Este tipo de direccionamiento suele llamarse direccionamiento de página.



Directo relativo

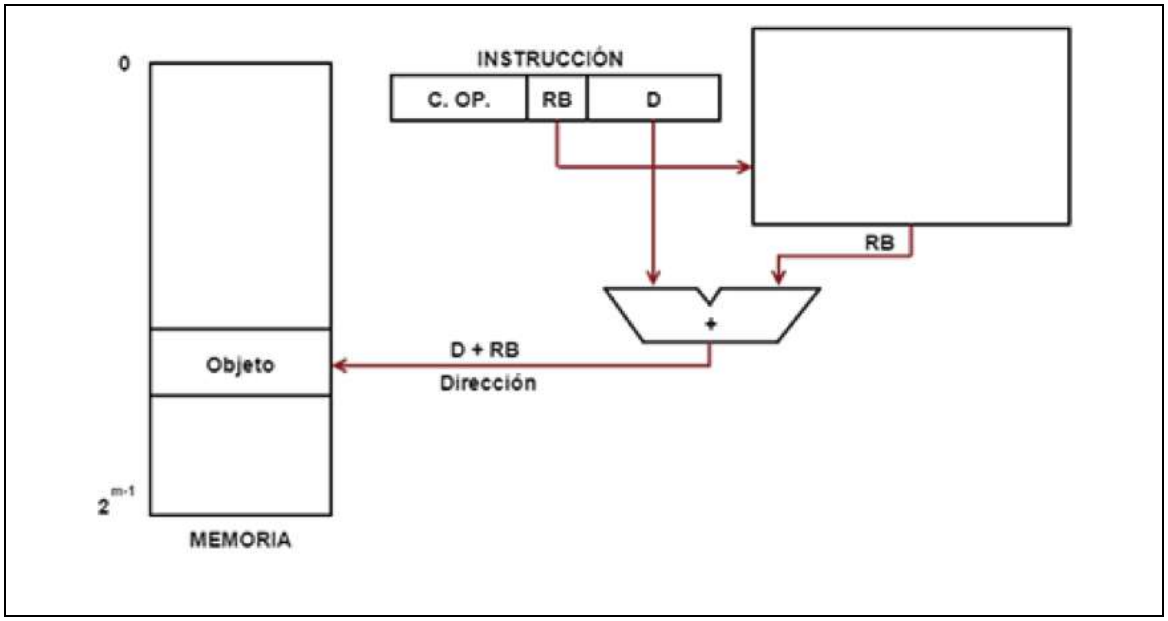
En el direccionamiento directo relativo la instrucción **no contiene la dirección del objeto**, sino un desplazamiento "D" sobre una dirección marcada por un puntero (punto conocido). La dirección se calcula **sumando el desplazamiento "D" al puntero de referencia**.

La mayoría de los procesadores permiten desplazamientos positivos y negativos, de esta forma, se puede alcanzar una zona de memoria principal alrededor de la posición marcada por el puntero.

Hay varios tipos de directo relativo, según el punto de referencia:

Direccionamiento directo relativo al contador de programa PC: El puntero empleado es el contador de programa PC, esto es, el registro que almacena la dirección de siguiente instrucción que se va a ejecutar.

- **Direccionamiento directo relativo a registro base**: Emplea como puntero un registro base RB. La instrucción deberá contener la identificación del registro que se emplea como base.
- **Direccionamiento directo relativo a registro índice**: El registro puntero (que llamarnos registro índice RI) es modificado para ir recorriendo los elementos de una tabla o vector.
- **Direccionamiento a pila**: La máquina deberá disponer de uno o varios registros SP, que realicen la función de puntero de la pila. Cada uno de ellos contiene la dirección de la posición de memoria principal que actúa de cabecera de pila.



Tipos y formatos de instrucciones

Las **instrucciones** siempre se pueden categorizar en este grupo, además, es necesario que un lenguaje de programación (ensamblador incluido) tenga **instrucciones que abarquen todos estos tipos**, si no se corre el riesgo de que el lenguaje no sea completo, y por consiguiente, algunos programas no puedan implementarse con él.

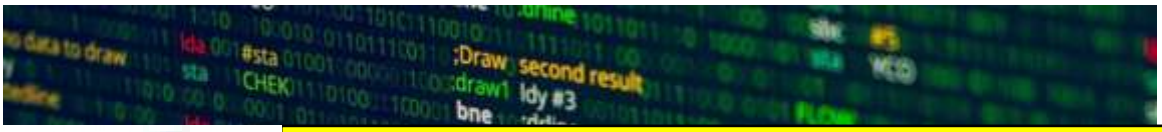
- Movimiento o transferencia de datos [move, load, …]
- Modificación de la secuencia del programa [call, branch, …]
- Aritméticas [add, divide, …]
- Comparación [compare, test, …]
- Lógicas [and, not, …]
- Desplazamiento [shift, rotate, …]
- De bit [bit set, bit test, …]
- De entrada/salida y misceláneas [imput, halt, …]

Aunque las instrucciones de máquina son **cadenas de "unos" y "ceros"**, para identificarlas más fácilmente se emplearán sus nombres más corrientes (en inglés). Los códigos nemónicos empleados por los ensambladores se derivan de estos nombres, por lo que conviene familiarizarse con ellos.

En ensamblador, las instrucciones tienen un **formato muy escrito y definido**, para simplificar su decodificación.

La instrucción en ensamblador (y lo mismo en código maquina) se divide en una **serie de campos**, estando referido cada campo a un tipo de información específico.

Código operación	Indica la operación a realizar (suma, transferencia, carga, etc.). Ejemplo: MOV (mover).
Operandos	Campo de dirección, especifica la dirección de un dato, resultado o instrucción a la que se bifurca. Dependiendo del tipo de direccionamiento, se divide en subcampos. Ejemplo: R1, 0x44



Modelo de ejecución

El **modelo de ejecución** de una **instrucción especifica** los dispositivos en los que están almacenados sus operandos. El modelo de ejecución está estrechamente ligado al **formato de las instrucciones**, pues el direccionamiento implicado en cada modelo es diferente.

Los modelos de ejecución que suelen considerarse son:

Pila
Todas las operaciones hacen referencia a datos que están en la pila, tanto **datos de entrada como de salida**, por lo que las instrucciones **no necesitan ninguna dirección**. Mejor dicho, todos los datos que intervienen en la operación son implícitos, hacen referencia a la cima de la pila. Solo hay dos instrucciones especiales "Load" y "Store" que permiten mover datos de memoria a pila y viceversa, solo estas instrucciones tienen un dato explícito.
Registro - Registro
Todos los operandos **residen necesariamente en registros**. Por lo que las instrucciones solo pueden hacer referencia a registros. Al igual que en el caso anterior, hay dos instrucciones especiales **"Load"** y **"Store"** que permiten mover datos de memoria a registros y viceversa.
Registro - Memoria
Este es un **subconjunto especial** del caso anterior, donde **solo uno de los datos necesarios en la instrucción puede estar en memoria**. Si la instrucción tiene más de un operando, todos los demás tiene que estar forzosamente en registros. Pero hay que recordar, que no es necesario que uno de los operando resida en memoria, solo posible, así, esta máquina también puede funcionar como una maquina registro-registro.
Memoria - Memoria
En este tipo de máquina, **cualquier número de operados puede residir en la memoria**. Este tipo de máquina, no se construye por ser mejor que los modelos anteriores, sino por ser una **necesidad en procesadores de bajo coste**, que, al no tener suficiente conjunto de registros, necesitan que las instrucciones **operen con memoria directamente**. Pero por término general, estos procesadores son muy lentos y se considera una característica de baja eficiencia.

Algunos de los **computadores ultrarrápidos** actuales están basados en **modelos de ejecución de tipo**

RISC vs CISC

Son dos tecnologías para construir microprocesadores genéricos, las siglas significan:

CISC
Complex Instruction Set Computer, Computadora con Conjunto de Instrucciones Complejo.
RISC
Reduced Instruction Set Computer, Computadora con Conjunto de Instrucciones Reducido

Los factores que provocan que aparezcan dos arquitecturas distintas para hacer lo mismo son muy diversos: **coste, facilidad de fabricación, eficiencia, facilidad de programación**, etc. Las dos arquitecturas tienen sus **ventajas y sus desventajas**, para ver sus puntos fuertes, analicemos cuáles son los términos de esas arquitecturas.

En las máquinas **CISC,** se daba un mayor poder al lenguaje (ensamblador), creando implementaciones de instrucciones en código máquina que realizaban tareas complejas, pero que eran muy fáciles de utilizar. Es decir, parte del trabajo de programación se bajaba al hardware, creando instrucciones que realizaban acciones complejas.

Por el contrario, **RISC**, apostó por instrucciones muy simples, que solo realizan una tarea, pero la realizan muy rápido y muy eficientemente.

Tareas complejas
Por ejemplo: **Instrucciones de salto condicional** (el "if") que decrementan una variable y luego realizaban un salto si no esta no llegaba a cero. Esta instrucción es muy cómoda para hacer un bucle tipo "for".

Instrucciones muy simples
En RISC no existe la instrucción de salto condicional, sino que esta hay que implementarla mediante tres instrucciones distintas. Una que decrementa una variable, otra que la compara contra cero y la otra que hace el salto si el resultado es correcto.

en ensamblador. Esto provoca que no sea determinante lo fácil o difícil que es programar un microprocesador, de eso se encarga el compilador, ya que el programador tiene que escribir código en un lenguaje que es el mismo para cualquier arquitectura, y le da lo mismo que ese código escrito en alto leguaje se traduzca a 1000 instrucciones máquina o a 100.000 instrucciones. Lo importante es que el código generado sea **rápido y eficiente**. Y esta característica está mejor implementada en máquinas RISC que en máquinas CISC.

En detalle - INTEL, ¿CISC o RISC?

Intel nace como una máquina **CISC**, de ahí se gran popularidad. Al ser una maquina CISC, es fácil programarla, por tanto, aparece mucho código que es útil, lo que provoca que este procesador se popularice. Mientras las máquinas RISC quedan **relegadas a un entorno muy profesional** y sobre todo **académico**, ya que en estos ambientes se hace software a medida, primando la eficiencia sobre la versatilidad.

Pero según pasa el tiempo, los compiladores mejoran, por lo que Intel se adapta a los nuevos tiempos y en la actualidad es una máquina **RISC**.

Por tanto, la respuesta es: Intel, nace CISC y crece en RISC.

Ejemplos de arquitecturas:

- RISC: PowerPC, DEC Alpha, MIPS, ARM, Pentium, etc.
- CISC: Motorola 68000, Zilog Z80, Intel x86, etc.

Resumen

Los procesadores **ejecutan instrucciones en código máquina**, la ejecución de una instrucción conlleva unos pasos, no es algo inmediato.

El **código máquina** es una **secuencia de números enorme** que dificulta la programación, haciéndola muy compleja y propensa a errores. La solución nos la ofrece el "**ensamblador**".

El ensamblador lo podemos considerar como un lenguaje de programación muy simple y muy dependiente de hardware. Estas características lo hacen muy eficiente pero muy poco portable e incompatible con diferentes familias de microprocesadores.

En ensamblador, los datos que necesita una instrucción pueden estar en distintos orígenes, por lo que hay que especificar dicho origen. A estos posibles orígenes de datos, se le conoce como "**modos de direccionamiento**". No todos los procesadores admiten todos los modos de direccionamiento, de ahí que aparezcan las categorías de máquinas según el direccionamiento, lo que se conoce como "**modelo de ejecución**".

Estos modelos de ejecución dan origen a diferentes familias de máquinas o tecnologías de microprocesadores, donde la división más popular es la de **RISC o CISC**.