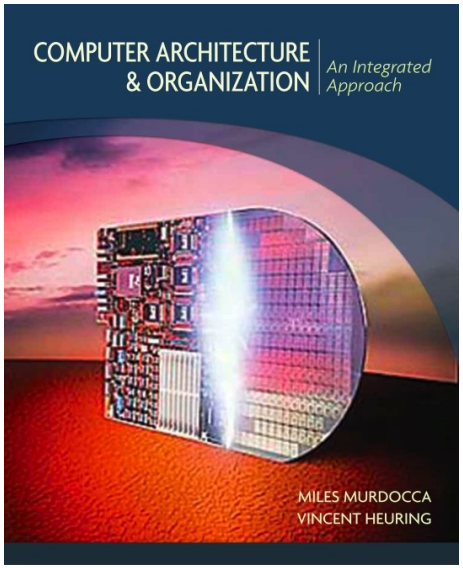
Organización de las Computadoras

Leonardo Giovanini



El conjunto de instrucciones de la arquitectura

Contenidos

2.1 Niveles de abstraccion de una computadora

2.2 El conjunto de instrucciones de la arquitectura

2.3 Elementos del conjunto de instrucciones

- *Modos de operación*
- *Organización de la memoria*
- *Gestión de eventos*
- *Instrucciones*
- *Modos de direccionamiento*

2.3 Arquitectura del procesador

Niveles de abstraccion de una computadora

Niveles de abstraccion

Una computadora tienen diferentes niveles, desde el usuario a los dispositivos.

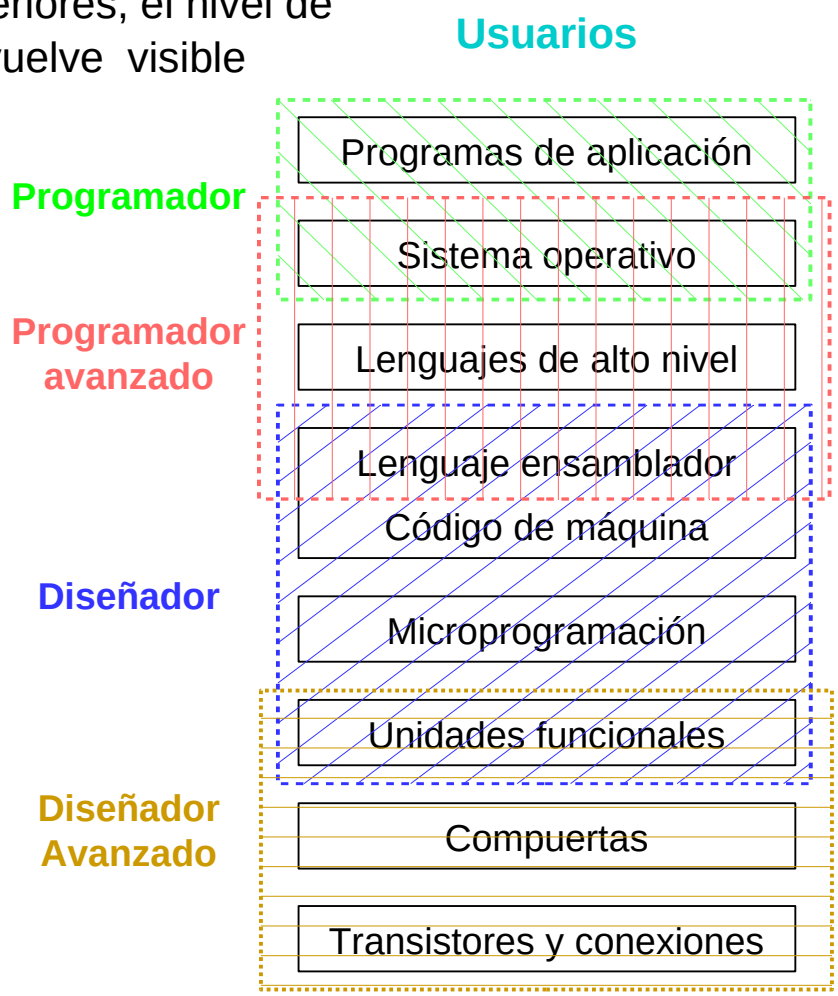
Descendiendo de los niveles superiores a los inferiores, el nivel de abstracción se reduce y la estructura interna se vuelve visible

Los ***lenguajes de alto nivel*** son independientes de la organización y arquitectura de la computadora.

Son utilizados por los programadores para desarrollar **aplicaciones y sistemas operativos**.

El ***Ensamblador*** es un lenguaje de bajo nivel que depende del procesador empleado y su uso requiere de un conocimiento profundo de la **organización y arquitectura de la computadora**.

Es utilizado por los programadores avanzados para desarrollar **software básico** (drivers y compiladores) o **aplicaciones embebidas**.



Niveles de abstraccion

Desarrollo de un programa

Un programa informático es un conjunto de instrucciones que una vez ejecutadas realizarán una o varias tareas en una computadora.

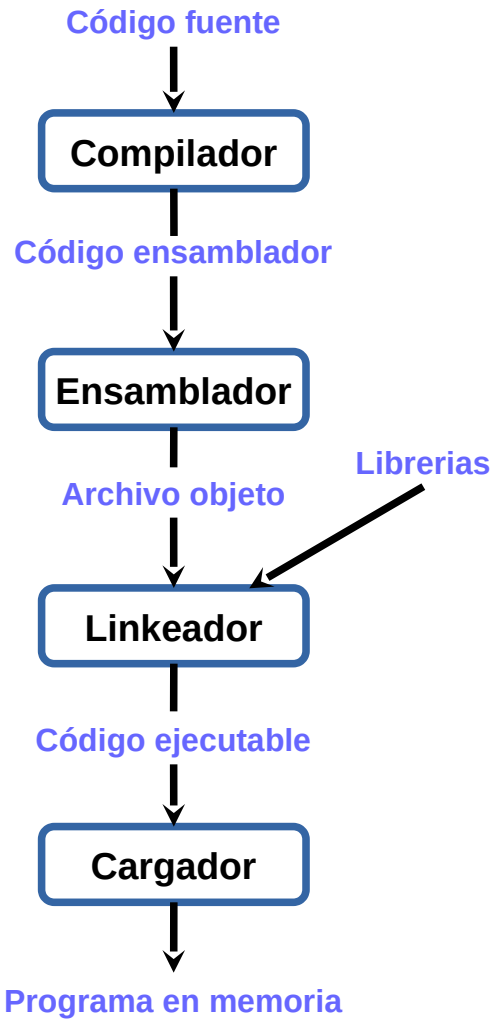
Al conjunto general de programas, se le denomina software, que más genéricamente se refiere al soporte lógico de una computadora digital.

De acuerdo a sus funciones, se clasifican en:

Software de aplicación: es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos. Esto lo diferencia principalmente de otros tipos de programas como los sistemas operativos (que hacen funcionar al ordenador), las utilidades (que realizan tareas de mantenimiento o de uso general), y los lenguajes de programación (con el cual se crean los programas informáticos).

Software de sistema: consiste en programas informáticos que sirven para controlar e interactuar con el hardware y dando soporte a otros programas.

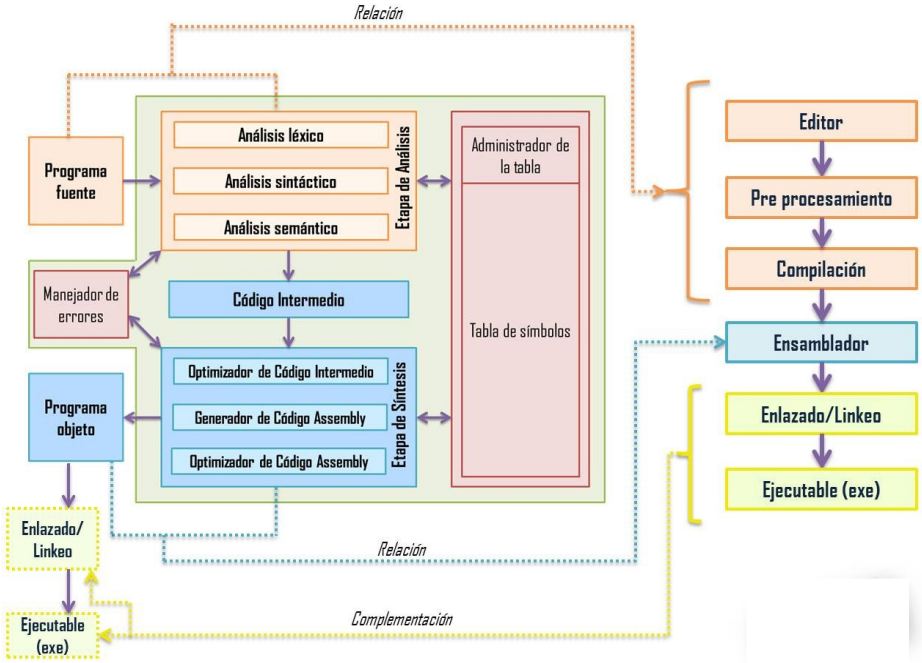
El desarrollo de cualquier programa informático incluye las etapas presentadas en la siguiente figura.



Niveles de abstraccion

Desarrollo de un programa - Compilación

Un **Compilador** traduce un pragrama escrito en ***lenguaje de alto nivel*** (*código fuente*), el cual es *independente de la architectura* de la CPU, en ***lenguaje ensamblador***, el cual *depende de la arquitectura* de CPU utilizada.



Código ensamblador

```
.data
f:
g:
y:

.text
main:
  addi  $sp, $sp, -4  # make stack frame
  sw    $ra, 0($sp)   # store $ra on stack
  addi  $a0, $0, 2    # $a0 = 2
  sw    $a0, f        # f = 2
  addi  $a1, $0, 3    # $a1 = 3
  sw    $a1, g        # g = 3
  jal   sum           # call sum procedure
  sw    $v0, y        # y = sum (f, g)
  lw    $ra, 0($sp)   # restore $ra from stack
  addi  $sp, $sp, 4   # restore stack pointer
  jr    $ra           # return to operating system

sum:
  add   $v0, $a0, $a1 # $v0 = a + b
  jr    $ra           # return to caller
```

Código fuente

```
int f, g, y; // global variables

int main (void)
{
  f = 2;
  g = 3;
  y = sum (f, g);
  return y;
}

int sum (int a, int b) {
  return (a + b);
}
```

Niveles de abstraccion

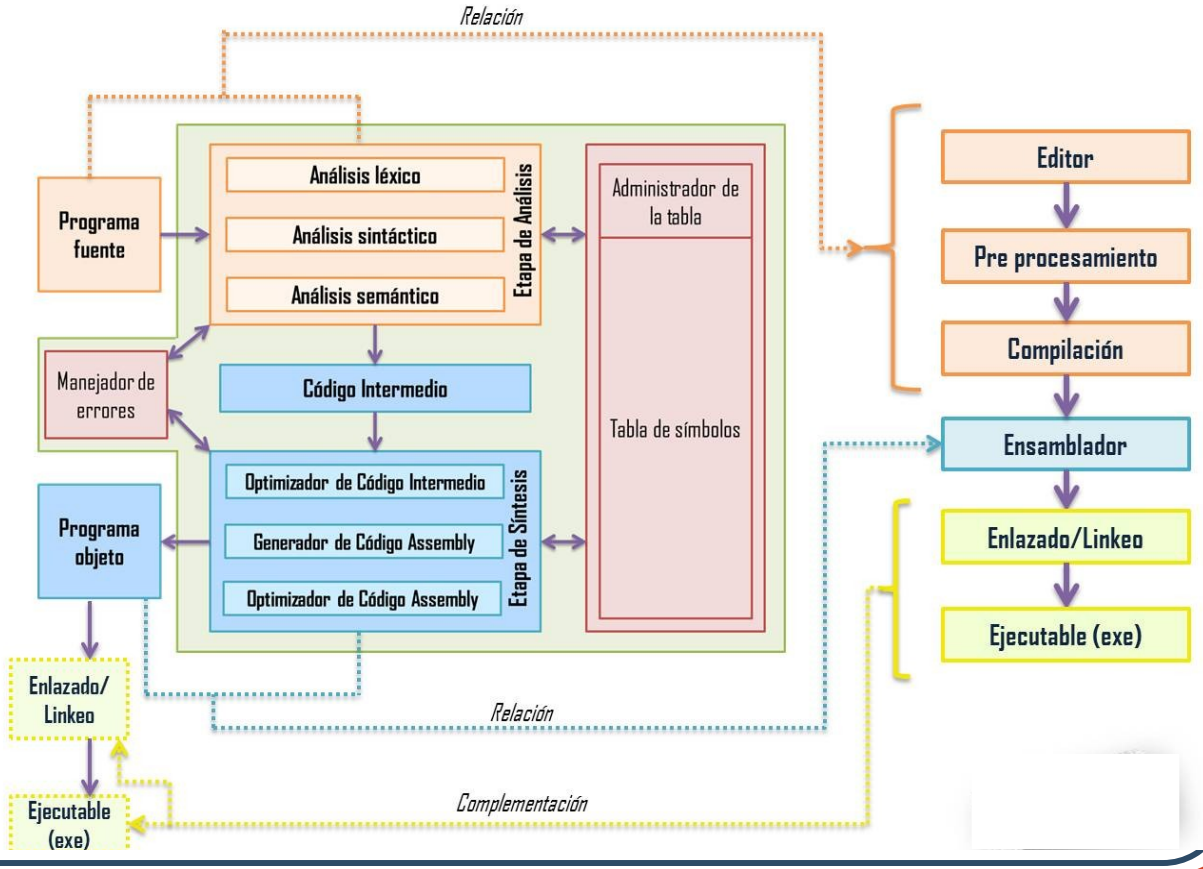
Desarrollo de un programa - Compilación

Un compilador involucra una serie de fases que variará con su complejidad. Generalmente estas fases se agrupan en dos tareas: el análisis del programa fuente y la síntesis del programa objeto.

Análisis: se trata de comprobar la corrección del programa fuente, e incluye las fases de *análisis léxico* (que consiste en la descomposición del programa fuente en componentes léxicos), *análisis sintáctico* (agrupación de los componentes léxicos en frases gramaticales) y *análisis semántico* (comprobación de la validez semántica de las sentencias aceptadas en el análisis sintáctico).

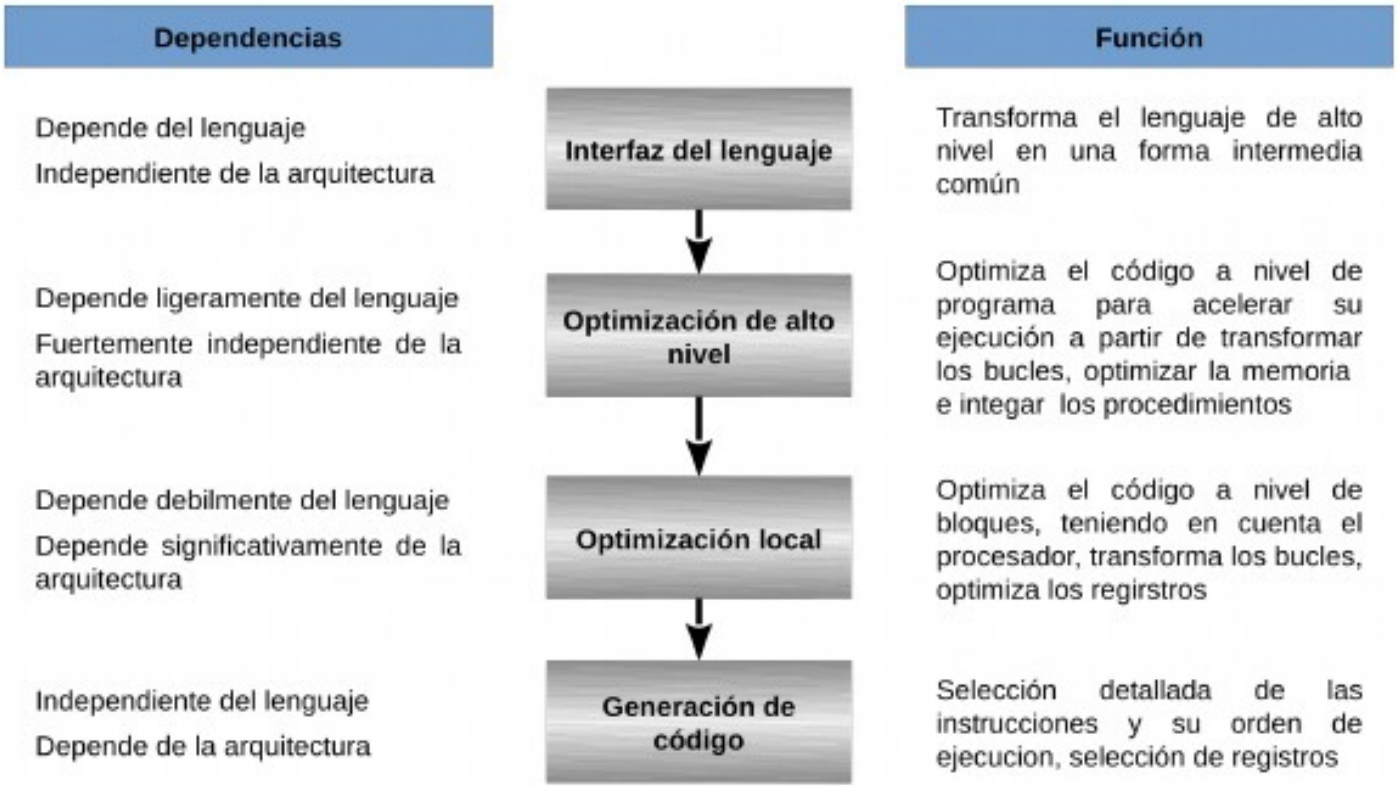
Síntesis: genera la salida expresada en el lenguaje objeto y suele estar formado por una o varias combinaciones de fases de *generación de código* (normalmente se trata de código intermedio o de código objeto) y de *optimización de código* (que busca obtener un código lo más eficiente posible).

Esta división permite que el mismo generador se utilice para crear el código máquina de diferentes lenguajes y que el mismo analizador que sirve para examinar el código para producir código máquina en varias plataformas. Suele incluir la generación y optimización del código dependiente de la máquina.



Niveles de abstraccion

Desarrollo de un programa - Compilación



Niveles de abstraccion

Desarrollo de un programa - Compilación

Interfaz: su función es leer en un programa fuente; comprobar la sintaxis y la semántica; y traducir el programa fuente a una forma intermedia que interprete la mayor parte del funcionamiento específico del idioma del programa.

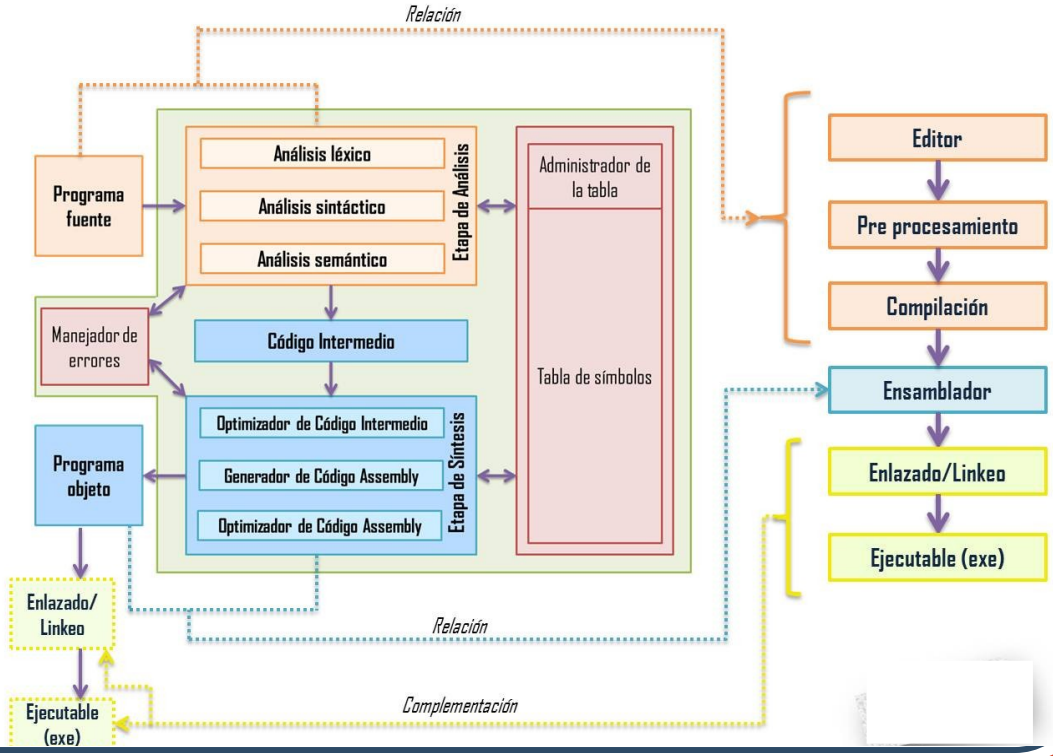
Normalmente se divide en cuatro funciones separadas:

El análisis de léxico (*scanning*), lee caracteres individuales y crea una cadena de palabras reservadas, nombres, operadores y símbolos de puntuación (*tokens*);

El análisis sintáctico (*parsing*), toma el flujo de token, asegura que la sintaxis es correcta y produce una representación de la estructura sintáctica del programa a partir de un *árbol de sintaxis abstracto*;

El análisis semántico (*semantic analysis*), toma el árbol de sintaxis abstracta y comprueba el programa para ver si es semanticamente correcta. Las verificaciones semánticas aseguran que las variables y los tipos se declaren correctamente y que los tipos de operadores y objetos coincidan. Durante este proceso se crea una tabla de símbolos que representa todos los objetos nombrados (clases, variables y funciones).

La generación de la representación intermedia (*code generation*) toma la tabla de símbolos y el árbol de sintaxis abstracta y genera la representación intermedia.



Niveles de abstraccion

Desarrollo de un programa - Compilación

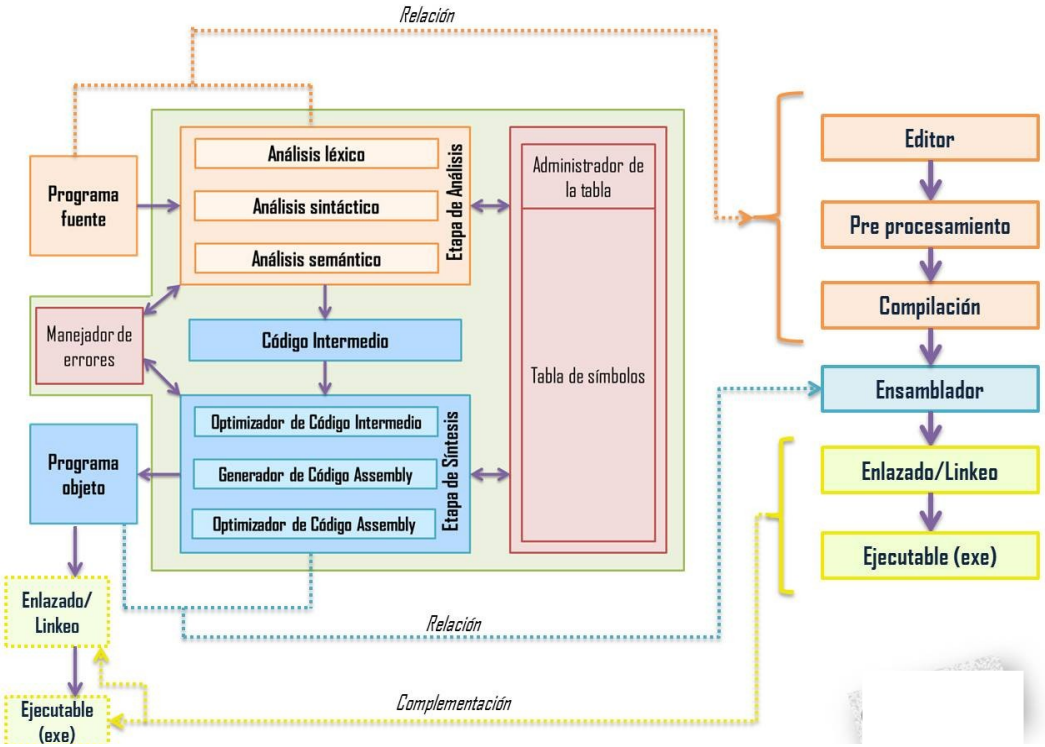
Las optimizaciones de alto nivel son transformaciones que se realizan en algo cercano al nivel de origen. La transformación de alto nivel más común es probablemente el procedimiento en línea, que reemplaza una llamada a una función por el cuerpo de la función, sustituyendo los argumentos de la persona que llama por los parámetros del procedimiento. Otras optimizaciones de alto nivel involucran transformaciones de bucle que pueden reducir la sobrecarga del bucle, mejorar el acceso a la memoria y explotar el hardware de manera más efectiva.

Se realizan tres clases de optimizaciones

La **optimización local** funciona dentro de un solo bloque básico. Un paso de optimización local a menudo se ejecuta como un precursor y sucesor de la optimización global para "limpiar" el código antes y después de la optimización global.

La optimización global funciona a través de múltiples bloques básicos.

La asignación de registros asigna variables a los registros para las regiones del código. Se realizan varias optimizaciones, tanto a nivel local como global, que incluyen la eliminación de subexpresiones comunes, la propagación constante, la propagación de copias, la eliminación de almacén inactivo y la reducción de la resistencia.



Niveles de abstraccion

Desarrollo de un programa - Compilación

Tipo de optimización	Descripción
Nivel alto	*Se realiza a nivel de código fuente* – *Independiente del procesador*
Integración de procedimientos;	Integración de procedimientos;
Nivel local	*Se realiza dentro del código* – *Indepencia débil con el procesador*
Eliminación de expresiones comunes;	Reemplaza instancias iguales del mismo código por una sola copia;
Propagación de constantes;	Reemplaza todas las variables con constantes con constantes;
Reducción de pila;	Reescribe expresiones para minimizar el uso de recursos;
Nivel global	*Se realiza a lo largo de todo del código* – *Depende del procesador*
Eliminación de expresiones comunes;	Reemplaza instancias iguales del mismo código por una sola copia;
Propagación de copias;	Reemplaza todas las instancias de variables con constantes;
Movimientos de código;	Borra código de dentro de bucles que calculan el mismo valor;
Eliminación de variables;	Simplifica y elimina las variables utilizadas dentro de un bucle;
Nivel bajo	*Se realiza en el código Assembler* – *Depende fuerte del procesador*
Reducción fuerte;	Simplifica las operaciones reemplazandolas;
Organización de la segmentación.	Reordenar las intrucciones para mejorar desempeño.

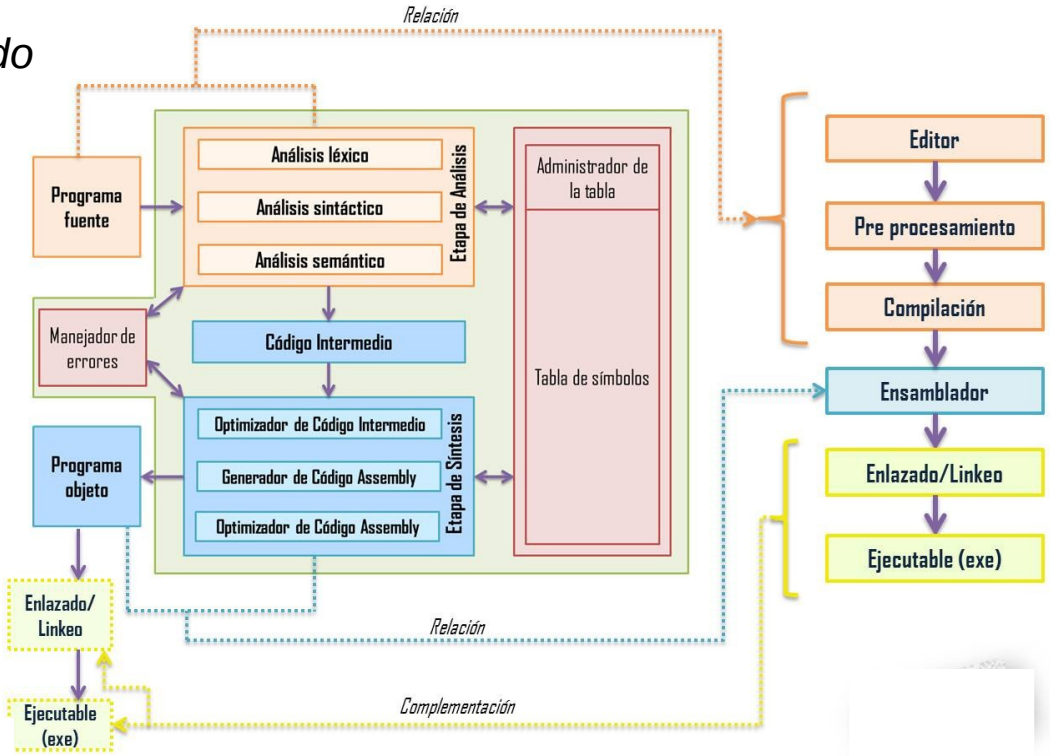
Niveles de abstraccion

Desarrollo de un programa - Ensamblado

El **lenguaje ensamblador** es un lenguaje de programación de bajo nivel que consiste en un conjunto de mnemónicos que representan instrucciones básicas de los procesadores.

Implementa una **representación simbólica** de los códigos de máquina binarios e información necesarias para programar un procesador y es la representación ***directa del código máquina***.

```
0x00400000   main:   addi  $sp, $sp, -4
0x00400004           sw    $ra, 0($sp)
0x00400008           addi  $a0, $0, 2
0x0040000C           sw    $a0, f
0x00400010           addi  $a1, $0, 3
0x00400014           sw    $a1, g
0x00400018           jal   sum
0x0040001C           sw    $v0, y
0x00400020           lw    $ra, 0($sp)
0x00400024           addi  $sp, $sp, 4
0x00400028           jr    $ra
0x0040002C   sum:    add   $v0, $a0, $a1
0x00400030           jr    $ra
```



Un **ensamblador** traduce programa en ***lenguaje ensamblador*** en ***código de máquina*** en dos pasos:

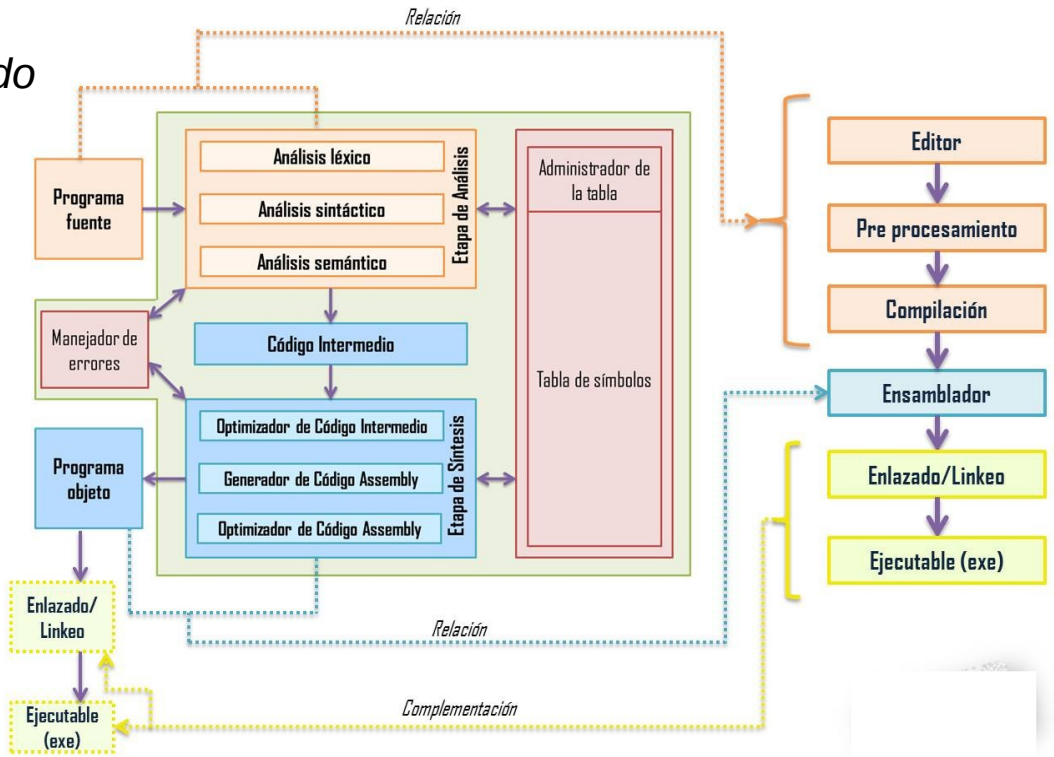
- Asigna una dirección a cada instrucción y busca los símbolos (etiquetas y nombre de variables);
- Produce el código de máquina y la tabla de símbolos.

El código de máquina y la tabla de símbolos son almacenadas en el **archivo objeto**.

Niveles de abstraccion

Desarrollo de un programa - Ensamblado

El **enlazador** organiza los datos e instructiones en un archivo que puede ***ser ejecutado*** por el procesador una vez cargado en memoria.



Text Size	Data Size
0x34 (52 bytes)	0xC (12 bytes)

Address	Instruction
0x00400000	0x23BDFFFC
0x00400004	0xAFBF0000
0x00400008	0x20040002
0x0040000C	0xAF848000
0x00400010	0x20050003
0x00400014	0xAF858004
0x00400018	0x0C10000B
0x0040001C	0xAF828008
0x00400020	0x8FBF0000
0x00400024	0x23BD0004
0x00400028	0x03E00008
0x0040002C	0x00851020
0x00400030	0x03E0008

Address	Data
0x10000000	f
0x10000004	g
0x10000008	y

```
addi $sp, $sp, -4
sw   $ra, 0 ($sp)
addi $a0, $0, 2
sw   $a0, 0x8000 ($gp)
addi $a1, $0, 3
sw   $a1, 0x8004 ($gp)
jal  0x0040002C
sw   $v0, 0x8008 ($gp)
lw   $ra, 0 ($sp)
addi $sp, $sp, -4
jr   $ra
add  $v0, $a0, $a1
jr   $ra
```

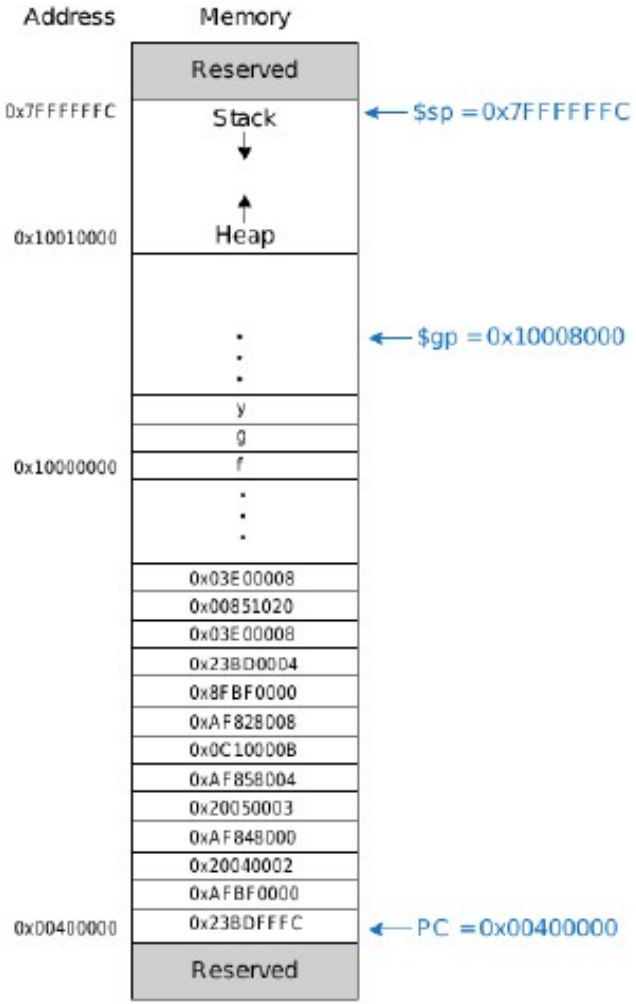
Un enlazador toma los objetos generados en el proceso de compilación, la información de todos los recursos necesarios (biblioteca), quita aquellos recursos que no necesita, y enlaza el código objeto con sus bibliotecas con lo que finalmente produce un fichero ejecutable.

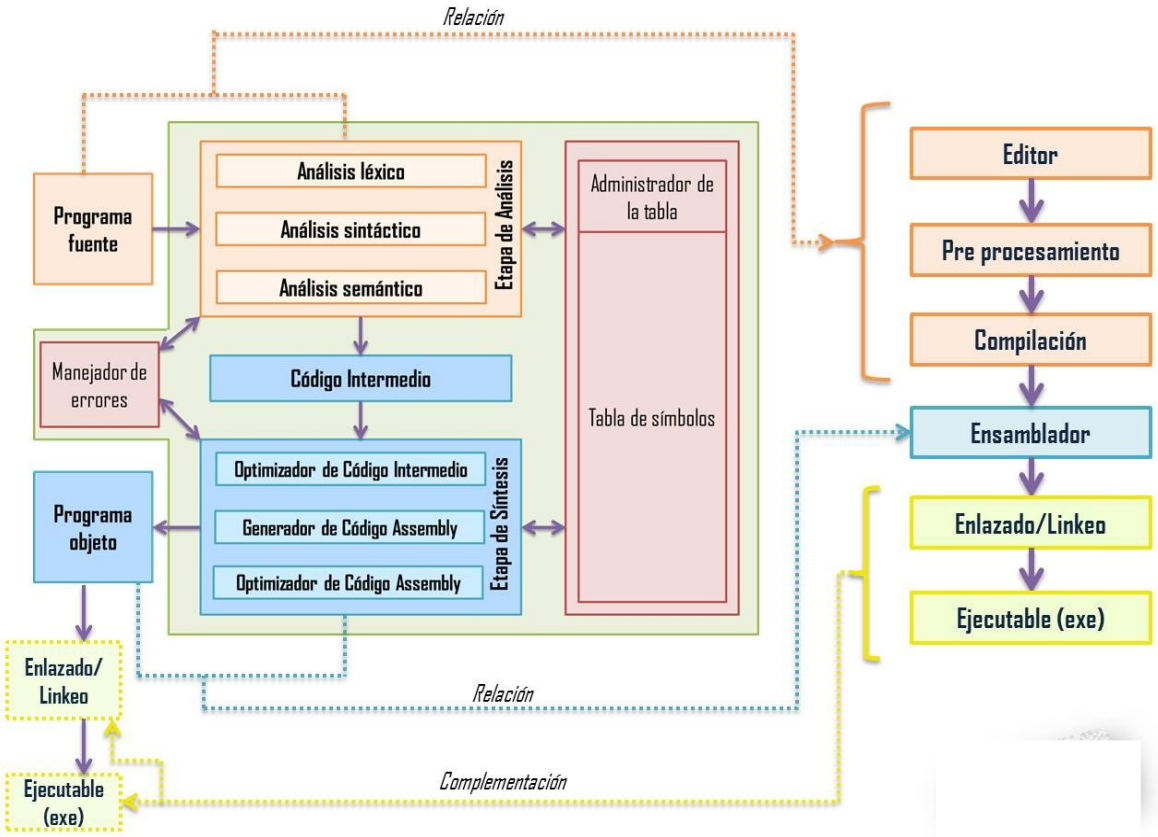
En el caso de los programas enlazados dinámicamente, el enlace entre el programa ejecutable y las bibliotecas se realiza en tiempo de carga o ejecución del programa.

Niveles de abstraccion

Desarrollo de un programa – Carga y Ejecución

El sistema operativo lee el archivo ejecutable del dispositivo de almacenamiento y lo carga en la memoria de la computadora para su ejecución.





La ubicación de cada segmento del programa (código, datos, variables locales y pila) depende de las características técnicas del procesador definidas en el ISA.

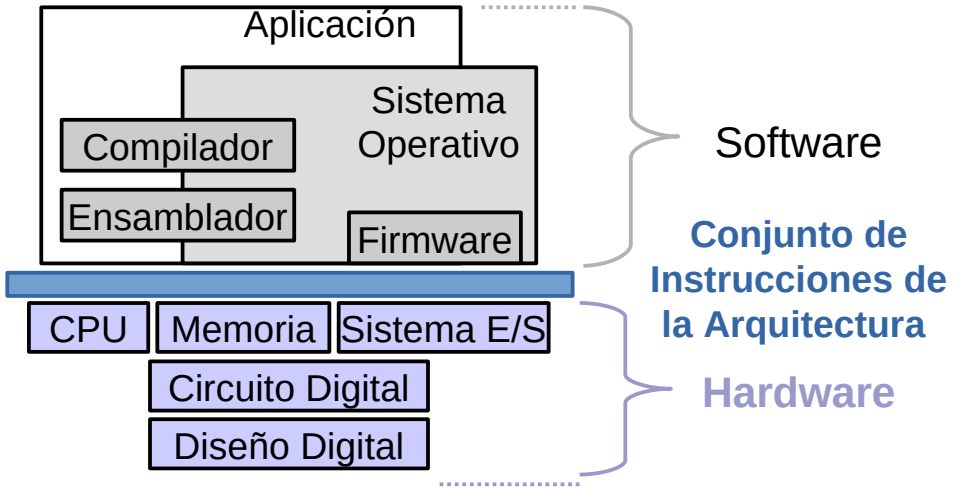
El conjunto de instrucciones de la arquitectura

El conjunto de instrucciones de la arquitectura

El **conjunto de instrucciones de la arquitectura** (ISA) de una computadora es un ***modelo abstracto*** que define toda la información necesaria para ***programar en lenguaje de máquina*** una computadora.

El ISA es la **interfaz entre el hardware y el software**.

La **microarquitectura** es el conjunto de ***técnicas de diseño*** utilizadas para implementar el ISA.



Un ISA puede implementarse con diferentes microarquitecturas que satisfagan **diferentes criterios de diseño** (velocidad de operación, consumo energético, costo, etc.). De este modo, el ISA sirve como la interfaz entre el software y el hardware.

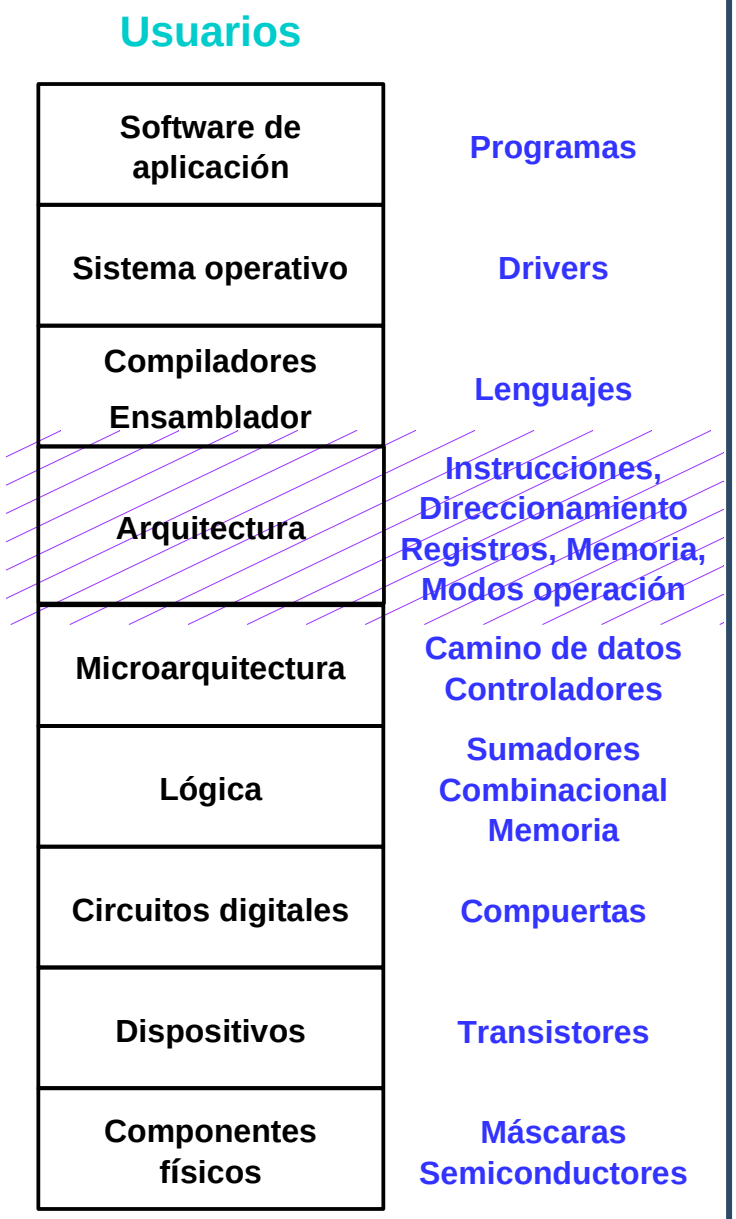
El ISA se describe a través de todos los estados visibles por el programador (registros + memoria, modos de operación) más la semántica de las instrucciones.

La IBM 360 fue la primer familia de computadoras que separó el ISA de la implementación (microarquitectura).

El conjunto de instrucciones de la arquitectura

El **conjunto de instrucciones de la arquitectura** está compuesto por:

- Los **modos de operación** soportados (usuario, supervisor, maquina virtual, etc);
- Los **tipos de datos** soportados (bit, nibble, byte, palabra) y su formato (entero, punto flotante);
- El **conjunto de instrucciones** son las operaciones que puede realizar la CPU (movimientos de datos, aritmeticas, lógicas, control de flujo);
- Los **operandos** utilizados por las instrucciones (registros, memoria y puertos) y los **modos de direccionamiento**, que son los mecanismos para que las instrucciones accedan a los datos;
- La **organización de la memoria** define la ***ubicación*** los datos y programas en la memoria, la ***dirección de reset***, y el **modelo de entrada/salida** utilizado para gestionar los periféricos; y
- El **modelo de gestión de los eventos** ajenos al programa (excepciones e interrupciones) y la ubicación de la información relacionada.

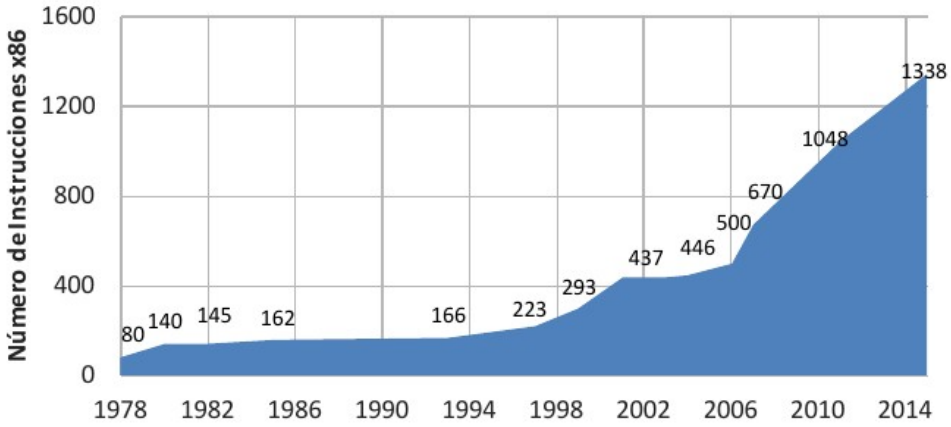


El conjunto de instrucciones de la arquitectura

El **enfoque convencional** en arquitectura de computadoras es desarrollar **ISA incrementales**, en los cuales los nuevos procesadores no solo **implementan las nuevas extensiones**, sino además **todas las instrucciones anteriores**.

El objetivo es **mantener compatibilidad** binaria para que los programas ya compilados y en formato binario anteriores puedan funcionar en los procesadores más recientes.

Dicho requerimiento provocó un incremento en la cantidad y complejidad del ISA.



El **enfoque alternativo** en arquitectura de computadoras es desarrollar **ISA modulares** en los cuales todos los procesadores implementan un **núcleo básico de instrucciones**, el cual provee una plataforna para desarrollar las herramientas de software, y **extensiones opcionales estándar** que el procesador puede incorporar de **acuerdo a las necesidades de cada aplicación**.

El conjunto de instrucciones de la arquitectura

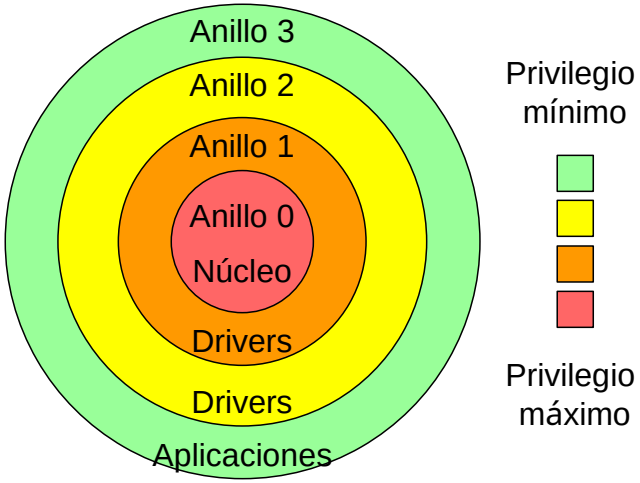
Arquitectura	Bits	Año	Operandos	Tipo	Diseño	Registros	Codificacion	Evaluacion de Salto	Endian -ness	
8080	8	1974	2	R - M	CISC	8	Variable (8 a 24 bts)	Condicion de Registro	Little	Cerrado
Z80	8	1976	2	R - M	CISC	17	Variable (8 a 32 bits)	Condicion de Registro	Little	Cerrado
8051	8	1977	1	R - R	CISC	16	Variable (8 a 128 bits)	Compara y Salta	Little	Cerrado
Microchip	8	1988	1	R - M	RISC	1	Fija (14 bits)	Compara y Salta	---	Cerrado
AVR	8	1997	2	R - R	RISC	32	Variable (16 o 32 bits)	Condicion de Registro	Little	Cerrado
X86	16, 32	1978	2	R - M	CISC	12, 14, 18	Variable (8 a 48 bits)	Condicion de Codigo	Little	Cerrado
MIPS	32	1981	1 - 3	R - R	RISC	32	Fija (32 bits)	Condicion de Registro	Big	Abierto
Open-RISC	32	2010	3	R - R	RISC	16	Fija (32 bits)	Compara y Salta	Big	Abierto
RISC V	32	2010	3	R - R	RISC	32	Variable	Compara y Salta	Little	Abierto
Spark	32	1985	3	R - R	RISC	32	Fija	Codigo de Condicion	Big	Abierto
Transputer	32	1987	1	Stack	MISC	3	Variable (8 a 120 bits)	Compara y Salta	Little	Cerrado

Elementos del conjunto de instrucciones de la arquitectura

El conjunto de instrucciones de la arquitectura

Los modos de operación

Los **modos de operación** de un procesador establecen las **restricciones de recursos** (registros, memoria y puertos) y las **operaciones ejecutables** por los programas al ser ejecutados por el procesador.



Este mecanismo de asignacion de recursos permiten **proteger la funcionalidad** de la computadora frente a fallas y comportamiento malicioso.

Los modos de operación están dispuestos en una **jerarquía** desde los más privilegiados (sin restricciones) hasta el menos privilegiado (recursos e instrucciones restringidas). El ISA proporciona **mecanismos de especiales** (instrucciones ó interrupciones) para permitir a un modo inferior acceder a los recursos del modo superior.

Por ejemplo, sólo al confiable código del núcleo de un sistema operativo tiene **acceso a todos los recursos** de hardware e instrucciones (modo no restringido); cualquier otra cosa (aplicaciones y porciones del sistema operativo cuyas funciones no sean las de supervisión) son ejecutadas en alguno de los modos restringidos, y debe emplear las llamadas al sistema para solicitar al núcleo que lleve a cabo en su nombre cualquier operación que pueda poner en peligro al sistema.

El conjunto de instrucciones de la arquitectura

Los modos de operación

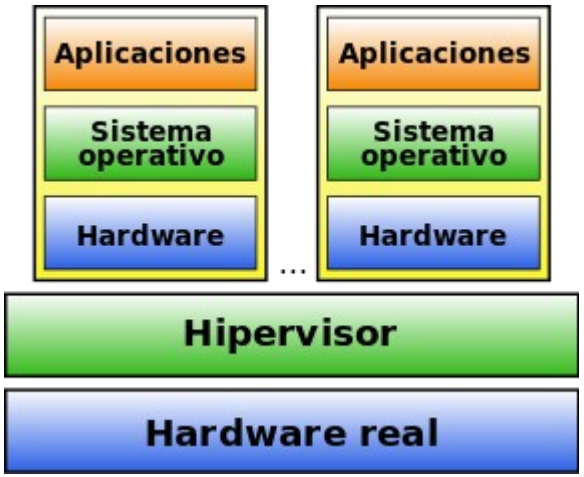
El **modo no restringido**, también llamado **modo supervisor**, permite al procesador realizar cualquier operación permitida por su arquitectura: cualquier instrucción puede ser ejecutada, cualquier operación de E/S y cualquier área de la memoria o registro accedido.

Los **modos restringidos**, también llamados **modos usuario**, inponen restricciones al hardware en las operaciones del procesador. Por lo general, algunas instrucciones no son permitidas (aquellas que incluyen operaciones de gestion avanzada de memoria y E/S que puedan alterar el estado de la computadora), las áreas de la memoria que puede sestán limitadas, entre otras.

La protección de modos puede extenderse a los recursos más allá del hardware del procesador, por ejemplo los **registros del modo de operación actual**, los **registros de memoria virtual**, la **tabla de páginas** y otros datos pueden hacer lo propio con los identificadores de modos de otros recursos.

Las capacidades del procesador en modo usuario son generalmente un subconjunto de aquellas disponibles en el modo supervisor, pero en algunos casos (**emulación de arquitecturas no nativas**) pueden ser significativamente diferentes de aquellas disponibles en el modo supervisor.

Por ejemplo un **modo hipervisor** puede ejecutar **varios programas en modo supervisor**, lo cual sirve de base para la implementación de sistemas de **máquinas virtuales**.



El conjunto de instrucciones de la arquitectura

La organización de la memoria

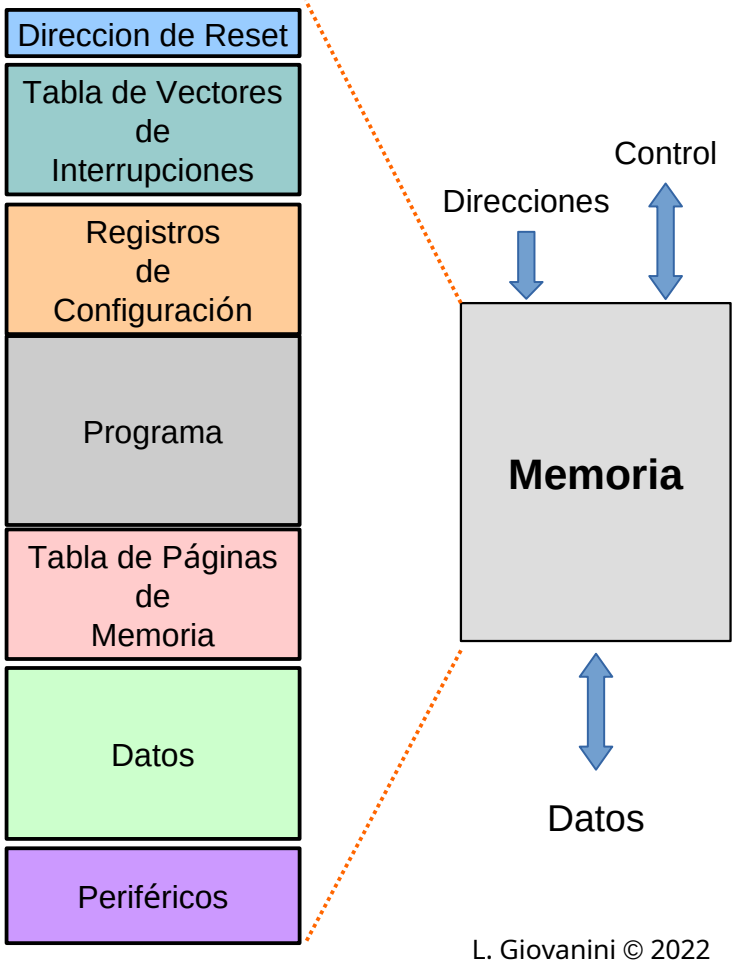
Las posiciones de la memoría están organizadas linealmente en orden consecutivo. Cada una de ella numerada con un número único del espacio de memoria del procesador.

El número que identifica cada palabra se denomina **dirección**.

La organización de la memoria es una clasificación, o subdivisión, de la memoria de la computadora según su uso o la información que almacene.

La memoria está dividida en tres espacios:

- El **espacio de programa** almacena el programa a ejecutar, incluyendo la direccion donde se inicia la ejecucion del programa (***direccion de reset***);
- El **espacio de datos** almacena la información utilizada durante la ejecución del programa (datos, resultados intermedios, etc.) y los periféricos; y
- El **espacio de configuración** almacena la información de configuración del procesador (tablas de páginas, tablas de vectores de interrupciones, registros de configuración y estados, ubicación de los periféricos).



El conjunto de instrucciones de la arquitectura

La organización de la memoria – Tamaño de los datos

Los tamaños de datos más utilizados por los procesadores son

Bit	0			
Nibble	0110			
Byte	10110000			
16-bit word (halfword)	11001001	01000110		
32-bit word	10110100	00110101	10011001	01011000
64-bit word (double)	01011000	01010101	10110000	11110011
	11001110	11101110	01111000	00110101
128-bit word (quad)	01011000	01010101	10110000	11110011
	11001110	11101110	01111000	00110101
	00001011	10100110	11110010	11100110
	10100100	01000100	10100101	01010001

El conjunto de instrucciones de la arquitectura

La organización de la memoria - Formato de almacenamiento

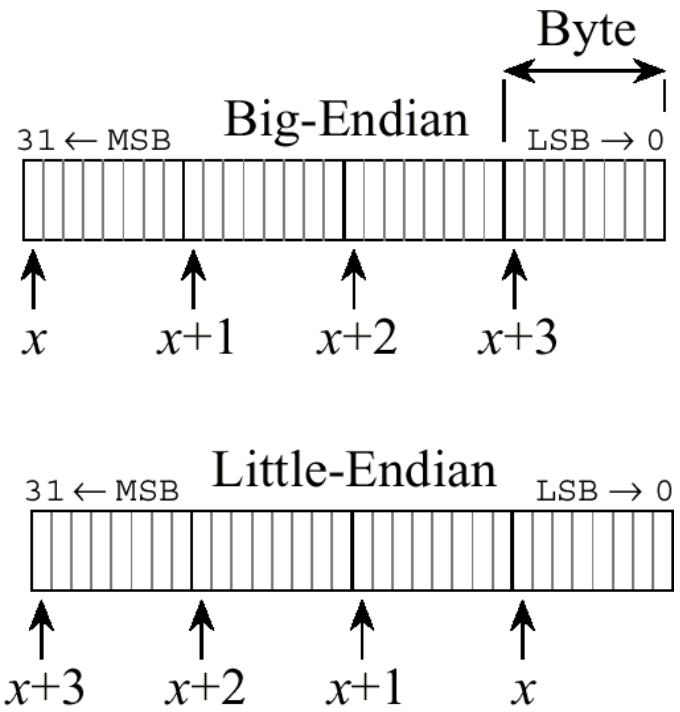
El **dato más pequeño** que puede direccionar un procesador en memoria es **un byte**.

Las palabras cuya **longitud sea mayor que un byte** (multi-byte) son almacenadas como una sequencia de bytes, en la que la dirección de la palabra es la misma que la del byte de la dirección más baja.

Cuando se utilizan ***palabras multibytes***, hay dos formas de almacenar los bytes en la memoria:

Big-endian, donde el byte más significativo se almacena en la dirección más baja, o

Little-endian, donde el byte menos significativo se almacena en la dirección más baja.



El conjunto de instrucciones de la arquitectura

La gestión de eventos

Durante la ejecución de programas suelen ocurrir eventos inesperados, no generados por el programa, que requieren un cambio en en el flujo de instrucciones que recibe el procesador.

Estos eventos pueden ser **externos al procesador** (asíncronos a la ejecución de las instrucciones) o ser **internos al procesador** (síncronos a la ejecución de las instrucciones).

Los mecanismos para atender estos eventos son:

Interrupción: consiste en **suspender temporalmente la ejecución del programa, una vez que se ejecutó la instrucción**, para atender un evento.

Se originan en:

- Un periférico que necesita ser atendido (dato disponible en un puerto serie, etc.); y
- Solicitud de servicios al sistema operativo; entre otras.

Excepción: consiste en **abotar la ejecución de la instrucción** debido a errorres en **procesador mismo** o **procesadores auxiliares** (violación de areas de memoria, error de representación, entre otros).

Se originan en:

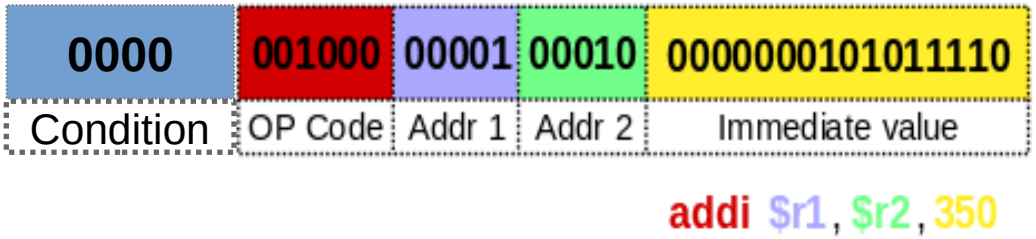
- Códigos de operación incorrectos o indefinidos;
- Errores en la ejecución de instrucciones;
- Instrucción no está permitida en el modo de ejecución actual;
- Violación de areas de memoria o permisos de ejecución;
- Error de representación; entre otros.

El conjunto de instrucciones de la arquitectura

Las instrucciones

El ***lenguaje máquina*** está construido a partir de ***instrucciones*** que especifica la operación a realizar y los operandos necesarios para ejecutarla.

Una instrucción incluye un ***código de operación*** (**OP Code**) que especifica la operación a realizar y ***especificadores de operandos*** (Addr1; Addr2; Immediate Value) que determinan la ubicación de los datos a utilizar por la instrucción.



Las ***instrucciones condicionales*** tienen un campo de predicado que codifica la condición específica para ejecutar o no la operación.

Las instrucciones pueden tener un tamaño:

- **Variable**, lo cual permite conjuntos de instrucciones flexibles y compactos; o
- **Fijo**, lo cual simplifica y hace eficiente su ejecución y paralelización.

Las características que se pretende que tenga un conjunto de instrucciones son:

- **Completo:** que se pueda realizar en tiempo finito cualquier tarea ejecutable;
- **Eficiente:** permita alta velocidad de cálculo y minimize el uso de los recursos;
- **Autocontenidas:** contengan toda la información necesaria para ejecutarse;
- **Independientes:** no dependan de la ejecución de alguna otra instrucción.

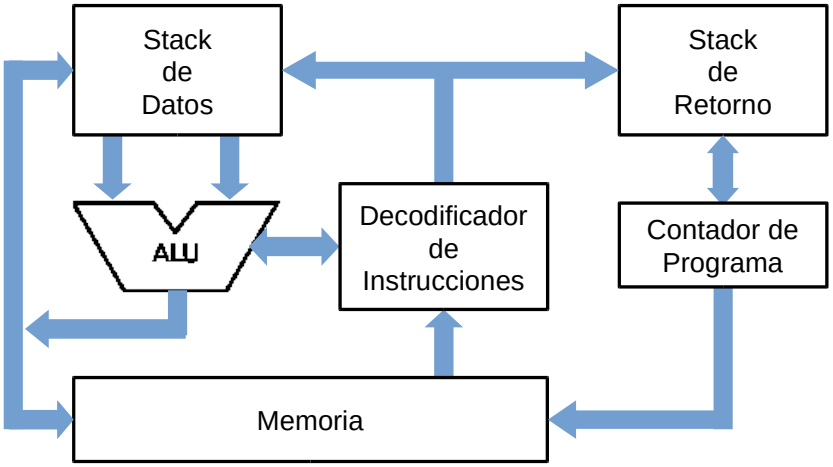
El conjunto de instrucciones de la arquitectura

Las instrucciones – Cantidad de operandos

Según la cantidad de operandos utilizados el ISA se clasifica en:

ISA con **operandos implícitos** (*stack architecture*) es un modelo computacional en el cual la memoria de la computadora toma la forma de pilas. Las instrucciones operan con los datos en el tope de las pilas y los reemplazan por el resultado.

El formato de instrucción de este modelo **no utiliza campo de dirección** por que los operandos estan **implicitos en la instrucción**.



Un ejemplo de código

```
A = B * (C + D * B)
1. Push B
2. Push D
3. *
4. Push C
5. +
6. Push B
7. *
8. Pop A
```

Ventajas: Buena densidad de código; Requerimientos de hardware bajos; Compiladores sencillos;

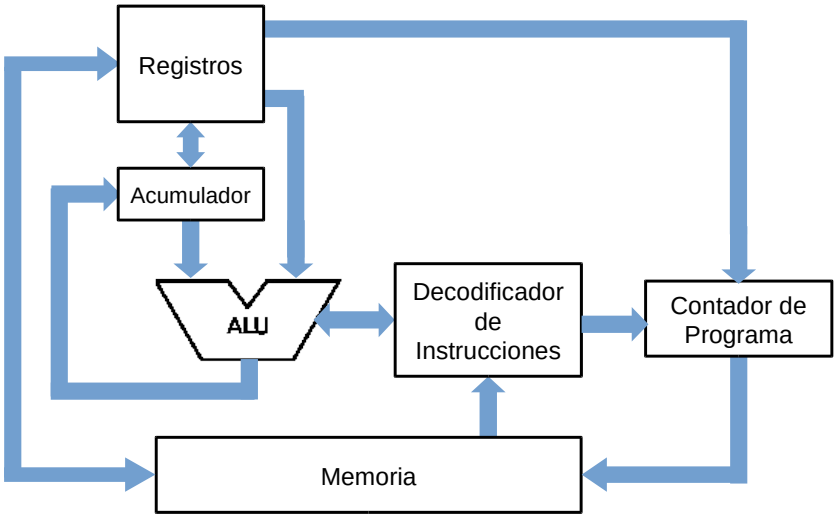
Desventajas: La pila es cuello de botella de ejecución; Dificultad para paralelizar ejecución; Necesidad de instrucciones adicionales para cargar datos; Compiladores óptimos son difíciles de desarrollar.

El conjunto de instrucciones de la arquitectura

Las instrucciones – Cantidad de operandos

ISA con **un operando explícito** (*accumulator architecture*) es un modelo computacional en cual el acumulador se usa implícitamente para procesar todas las instrucciones y almacenar los resultados.

El formato de instrucción de este modelo utiliza **un campo de dirección** por que uno de los operandos y el destino estan implicitos (acumulador).



Un ejemplo de código

$A = B * (C + D * B)$

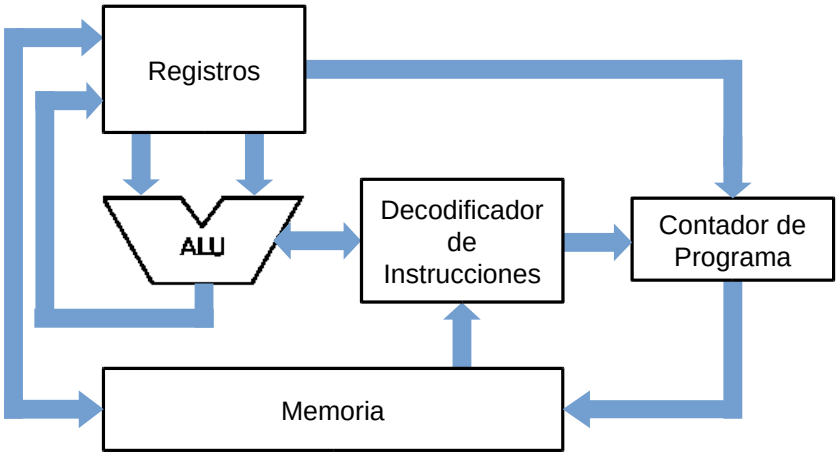
1. Load B
2. Mult D
3. Add C
4. Mult B
5. Store A

Ventajas: Instrucciones cortas y muy flexibles; Programa pequeños; Requerimientos de hardware muy bajos; Fácil de diseñar y comprender;

Desventajas: El tráfico de memoria es elevado; El acumulador es un cuello de botella durante la ejecución; Difícil de paralelizar y segmentar la ejecución de instrucciones.

El conjunto de instrucciones de la arquitectura

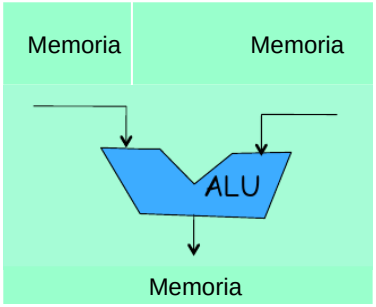
Las instrucciones – Cantidad de operandos



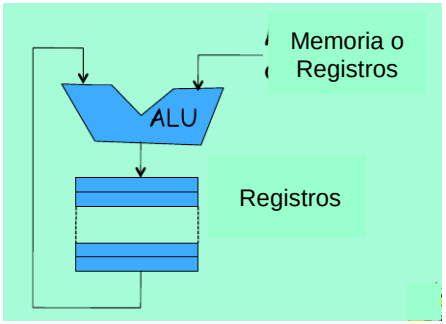
ISA con **registros de propositos generales** (*general purpose register architecture*) es un modelo computacional en cual todos los registros pueden ser utilizados para procesar todas las instrucciones y almacenar los resultados.

El formato de instrucción de este modelo utiliza **tres campos de direcciones**, **dos para los operandos** y uno donde se **almacena el resultado**.

Hay tres variantes de esta arquitectura:

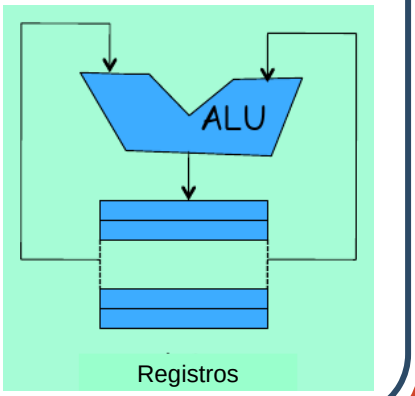


Tres operandos (fuentes y destino) localizados en la **memoria** de la computadora (*Memory-Memory architecture*);



Dos operandos localizados en **registros** (un fuente y destino) y el **otro en la memoria** de la computadora (*Memory-Register architecture*);

Tres operandos (fuentes y destino) localizados en **registros** (*Register-Register architecture*) con instrucciones **Load** y **Store** para **mover datos** entre memoria y registros.



El conjunto de instrucciones de la arquitectura

Las instrucciones – Cantidad de operandos

Ejemplos de códigos para resolver $A = B * (C + D * B)$

```
1. Mult R0, D, B
2. Add R0, R0, C
3. Mult A, B, R0
```

Memoria – Memoria (M-M**)**

```
1. Load R0, B
2. Mult R0, D
3. Add R0, C
4. Mult R0, B
5. Store R0, A
```

Memoria – Registro (M-R**)**

```
1. Load R0, B
2. Load R1, D
3. Mult R2, R0, R1
4. Load R3, C
5. Add R3, R3, R2
6. Mult R4, R0, R0
7. Store R4, A
```

Registro – Registro (R-R**)**

M-M - **Ventajas:** Requiere pocas instructiones; Compiladores fáciles de desarrollar;

Desventajas: Tráfico de memoria muy alto; Número variable de ciclos de reloj.

M-R - **Ventajas:** Formato de instrucción fácil de codificar; Buena densidad de codigo;

Desventajas: Los operandos no son equivalentes; Número variable de ciclos de reloj; Número limitado de registros.

R-R - **Ventajas:** Codificación de instrucciones simple y longitud fija; Número fijo de ciclos de reloj; Facil de paralelizar y segmentar;

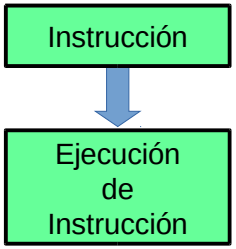
Desventajas: Mayor numero de instrucciones; Depende de la calidad del compilador.

El conjunto de instrucciones de la arquitectura

Las instrucciones – Complejidad de las operaciones

De acuerdo a la complejidad de las operaciones realizadas un ISA puede clasificarse en:

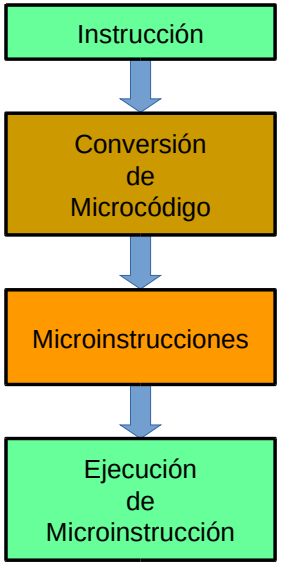
El **conjunto de instrucciones reducido** (RISC) simplifica la microarquitectura al sólo implementar de manera eficiente las **instrucciones más utilizadas**, optimizando el desempeño global.



Los objetivos son paralelizar las instrucciones y reducir los accesos a memoria.

Las características de los diseños RISC son:

- **Codificación uniforme**; lo que permite una decodificación más rápida.
- **Conjunto de registros homogéneo**, cualquier registro es utilizado en cualquier contexto; y
- Modos de direccionamiento y tipos de datos soportados **sencillos**.

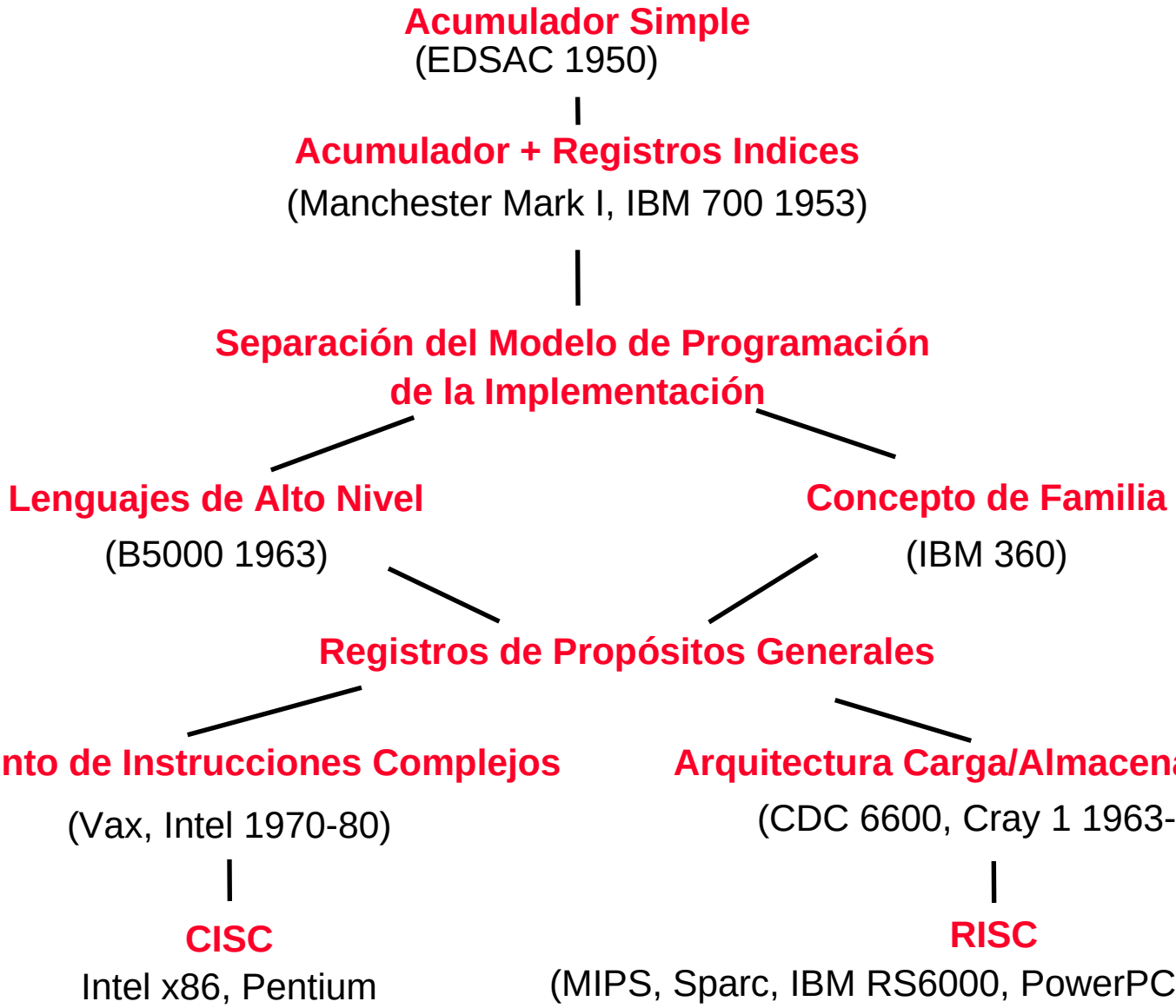


El **conjunto de instrucciones complejas** (CISC) es un modelo de arquitectura de computadores que se caracteriza por ser muy amplio y permitir operaciones complejas entre operandos situados en la memoria o en los registros que hacen que el código sea compacto.

Las características de los diseños CISC son:

- **Codificación no uniforme**, lo que genera códigos compactos pero complica la decodificación de las instrucciones;
- **Ejecución multiciclo** de las instrucciones, lo que dificulta la paralelización de la instrucciones; y
- Instrucciones son implementadas a través de **microprogramación.**

El conjunto de instrucciones de la arquitectura



El conjunto de instrucciones de la arquitectura

Las instrucciones

Las operaciones disponibles en la mayoría de conjuntos son:

- **Movimientos de datos**
  - Establecer un registro a un valor constante;
  - Mover datos desde una posición de memoria a un registro;
  - Leer y escribir datos en periféricos.
- **Operaciones matemáticas y logicas**
  - Sumar, restar, multiplicar o dividir dos registros;
  - Operaciones logicas y comparaciones;
  - Operaciones bit a bit.
- **Control de flujo del programa**
  - Saltar a otra posición del programa;
  - Saltar a otra posición si se cumple cierta condición;
  - Saltar a otra posición, pero salvando la posición actual para poder volver.
- **Control de la CPU y misceláneas**
  - Gestionar las interrupciones y excepciones;
  - Gestionar la información de los modos de operación;
  - Gestionar los recursos del procesador.

El conjunto de instrucciones de la arquitectura

Los modos de direccionamiento

Los modos de direccionamiento de una conjunto de instrucciones de una arquitectura (ISA) definen como las instrucciones identifican los operandos.

Los modos de direccionamiento especifican como calcular la dirección de memoria efectiva de un operando mediante el uso de la información contenida en los registros y/o constantes contenidas dentro de una instrucción.

Los modos de direccionamiento mas utilizados son:

- Implicito
- Directo
- Relativo
- Inmediato
- Indirecto
- Indirecto recursivo

- Condicional
- Salto
- Pila

- Registro
- Registro Indirecto
- Indexado
- Register Base
- Autoincremento / Autodecremento

El conjunto de instrucciones de la arquitectura

Los modos de direccionamiento

Inmediato: El operando está especificado en la instrucción.

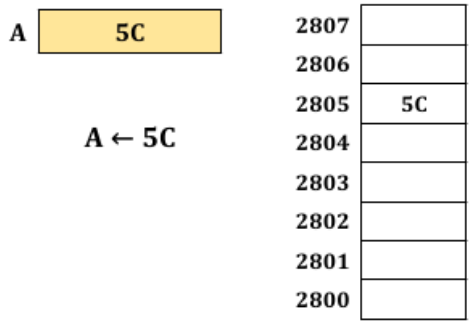
MVI A , 15h $A \leftarrow 15h$

ADI 3Eh $A \leftarrow A + 3Eh$

Directo: La instrucción especifica la dirección del operando.

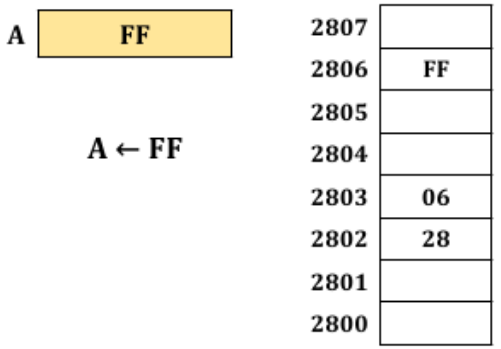
LDA 2805h $A \leftarrow [2805]$

STA 2803h $[2803] \leftarrow A$



Indirecto: La instrucción especifica donde se encuentra la dirección efectiva del operando.

MOV A, 2802h $A \leftarrow [[2802]]$

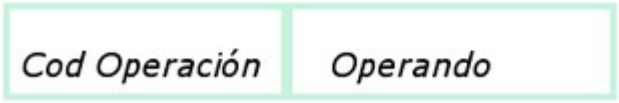


Indirecto recursivo: La instrucción especifica donde encuentra la dirección efectiva del operando la cual tiene un campo para indexación y otro indirecto, de modo que el proceso de direccionamiento podría teóricamente repetirse cualquier número de veces hasta que se encontrara una dirección sin un campo indirecto en la cadena.

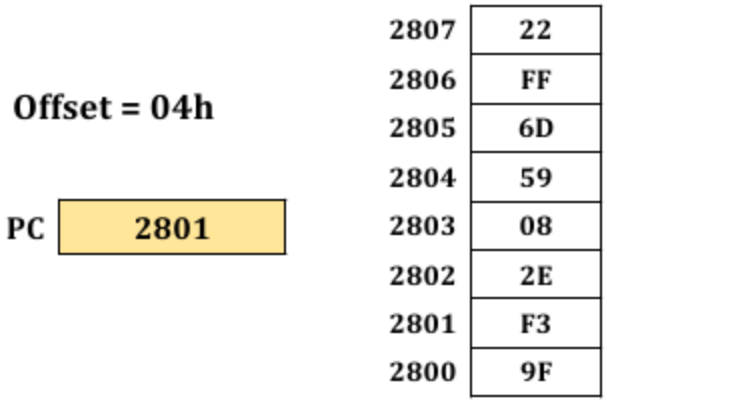
El conjunto de instrucciones de la arquitectura

Los modos de direccionamiento

Implícito: El operando está implícito en la instrucción.



Relativo: La instrucción especifica la dirección efectiva a partir de un desplazamiento relativo al Contador de Programa a partir de un offset provisto en la instrucción.



Effective address of operand = **PC + 01 + offset**
Effective address of operand = **2801 + 01 + 04**
Effective address of operand = **2806h**

El conjunto de instrucciones de la arquitectura

Los modos de direccionamiento

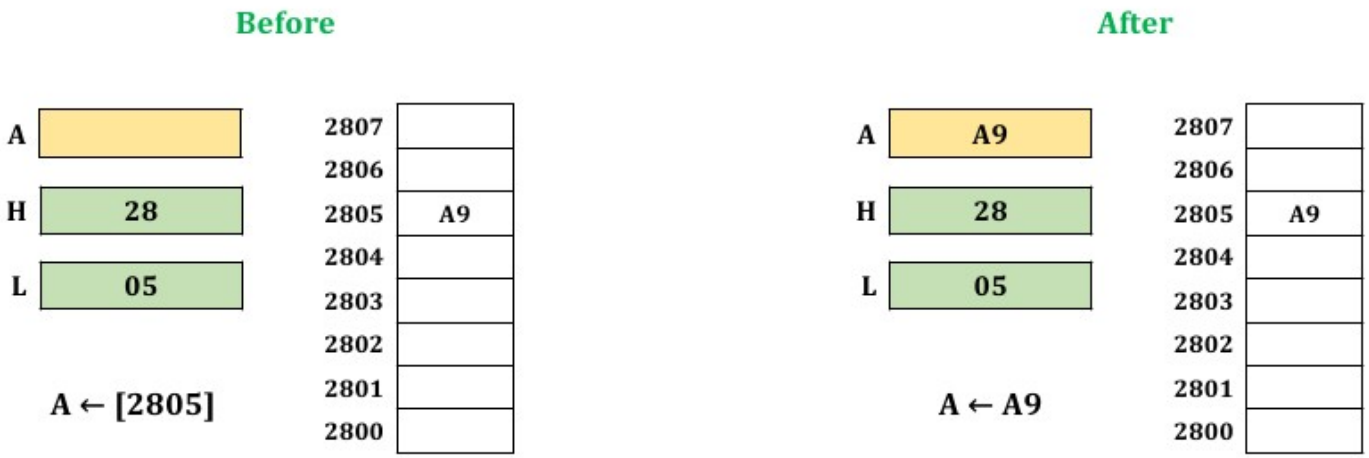
Registro: El operando es especificado en uno de los registro.

MOV C , A $C \leftarrow A$

ADD B $A \leftarrow A + B$

Registro Indirecto: La instrucción especifica el registro en el cual la dirección de memoria está almacenda.

MOV A , M $A \leftarrow [[H][L]]$



El conjunto de instrucciones de la arquitectura

Los modos de direccionamiento

Indexado: La dirección efectiva se obtiene sumandole a un registro índice una base provista por la instrucción.

En este modo de direccionamiento el que varía es el registro índice

Base = 2800h

Effective address of operand = **Base + IX**

2807	22
2806	FF
2805	6D
2804	59
2803	08
2802	2E
2801	F3
2800	9F

IX 0000

2800h + 0000h = 2800h

2807	22
2806	FF
2805	6D
2804	59
2803	08
2802	2E
2801	F3
2800	9F

IX 0001

2800h + 0001h = 2801h

2807	22
2806	FF
2805	6D
2804	59
2803	08
2802	2E
2801	F3
2800	9F

IX 0002

2800h + 0002h = 2802h

2807	22
2806	FF
2805	6D
2804	59
2803	08
2802	2E
2801	F3
2800	9F

IX 0003

2800h + 0003h = 2803h

El conjunto de instrucciones de la arquitectura

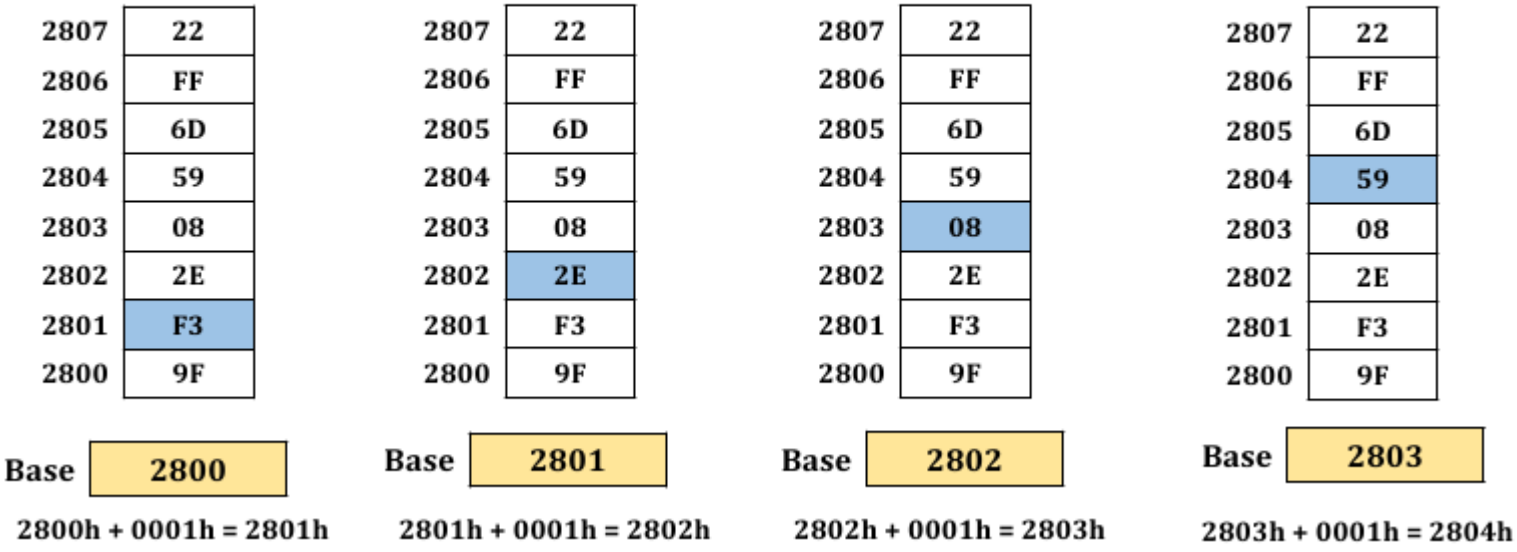
Los modos de direccionamiento

Registro Base: La dirección efectiva se obtiene sumandole a un registro base un offset provisto provisto por la instrucción.

En este modo de direccionamiento el que varía es el registro base.

Offset= 0001h

Effective address of operand = **Base Register + offset**

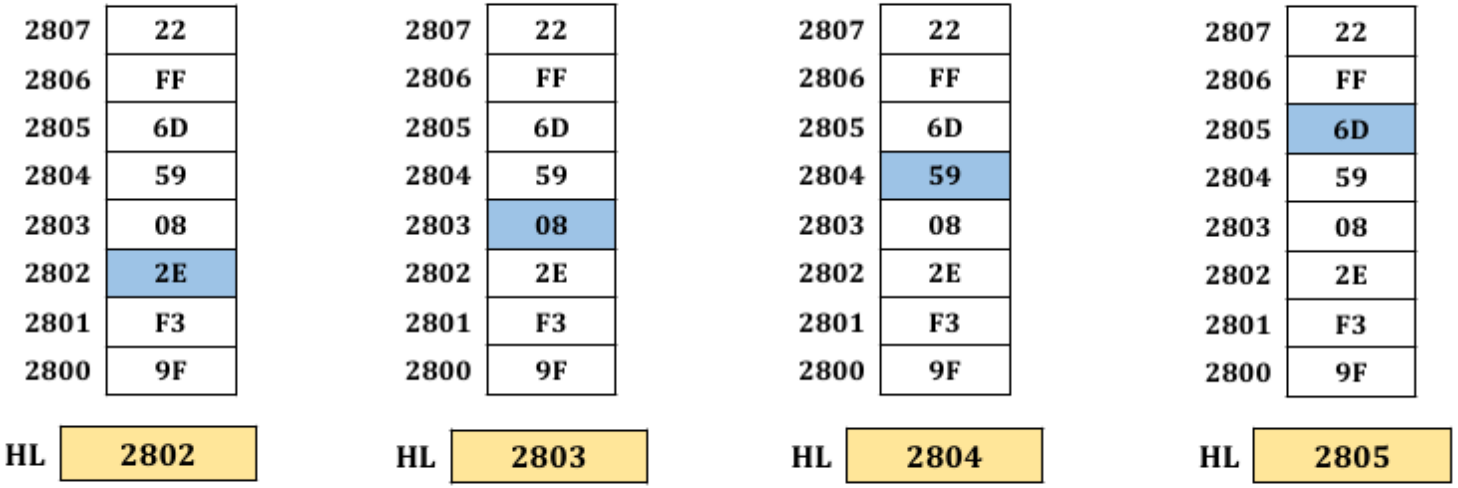


El conjunto de instrucciones de la arquitectura

Los modos de direccionamiento

Autoincremento/Autodecremento: Es similar al modo registro indirecto, pero cada vez que es utilizado el registro se modifica automaticamente.

Al principio HL 2802



El conjunto de instrucciones de la arquitectura

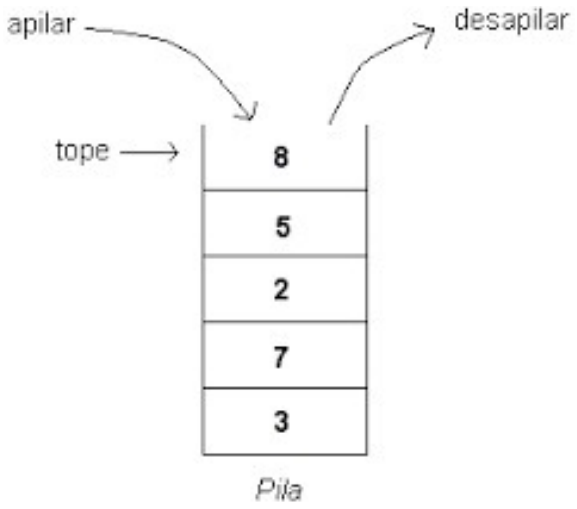
Los modos de direccionamiento

Condicional: Algunos procesadores tienen instrucciones que se ejecutan de manera condiciona o instrucciones de carga condicional.

Una instrucción de comparación o que modifique las banderas debe ejecutarse antes de la ejecución de las instrucciones.

Salto: El direccionamiento de salto o Skip es un caso especial de direccionamien to relativo con el offset fijo a "+1".

Pila: Este direccionamiento se basa en las estructuras tipo LIFO (Pila), las cuales están marcados por el fondo de la pila y un puntero de pila (SP). El puntero de pila apunta a la última posición ocupada. Como es un modo de direccionamiento implícito, solo se utiliza en instrucciones determinadas, las más habituales de las cuales son PUSH (poner un elemento) y POP (sacar un elemento).



Arquitectura del procesador

Arquitectura del procesador

El ***conjunto de instrucciones*** (ISA) es el modelo de programación de un procesador, la manera que es visto por un programador avanzado.

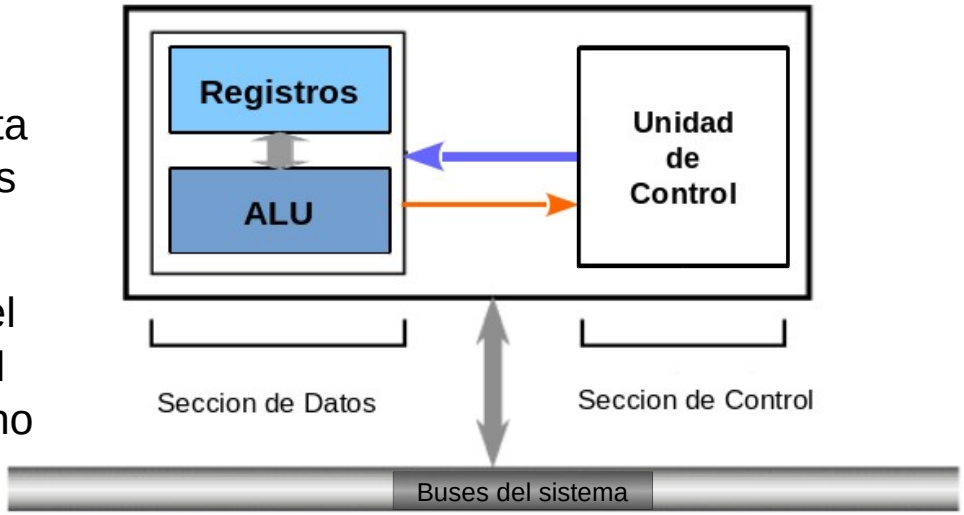
Define el ***modelo de ejecución***, ***registros***, ***modos de direccionamiento***, ***formatos de datos*** y ***mecanismos para atender eventos*** ajenos al programa.

La ***microarquitectura*** describe las ***partes*** del procesador, como se ***interconectan*** y como ***interoperan*** para implementar el ISA.

Se presenta como ***diagramas de bloques*** que describen estas interconexiones.

Estos diagramas suelen incluir al meno bloques:

- **El camino de datos** (*datapath*), es la ruta que recorren las instrucciones, operandos y resultados;
- **La unidad de control** (*control unit*), es el camino que siguen las señales de control que manejan el funcionamiento del camino de datos.

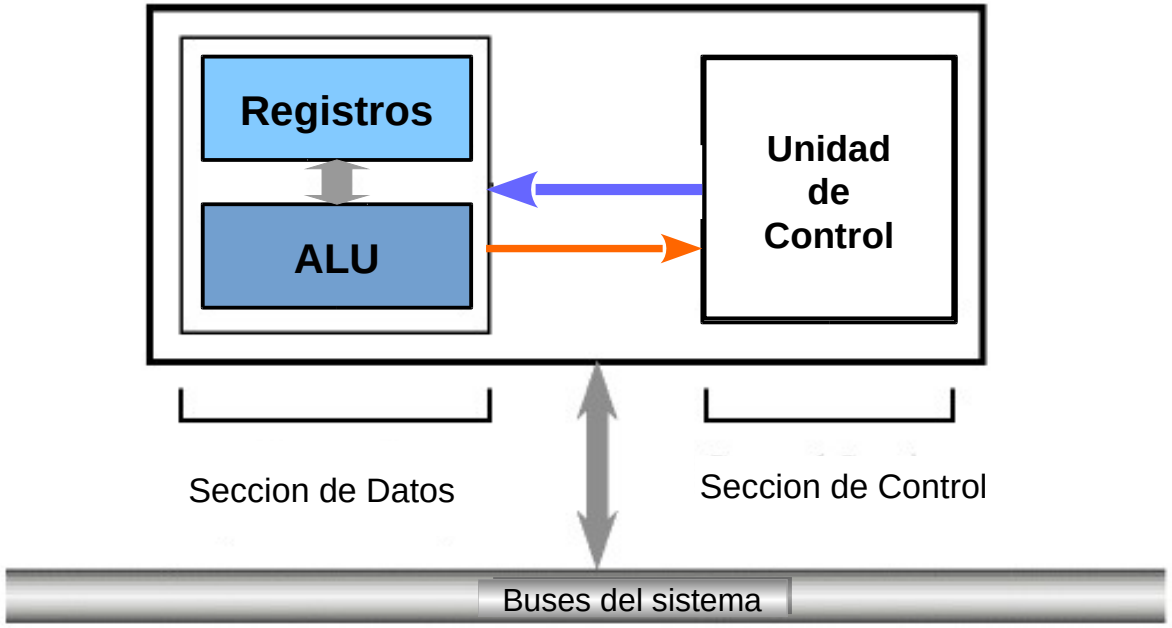


La ***microarquitectura*** y el ***conjunto de instrucciones de la arquitectura*** conforman la **arquitectura de una computadora**.

Arquitectura del procesador

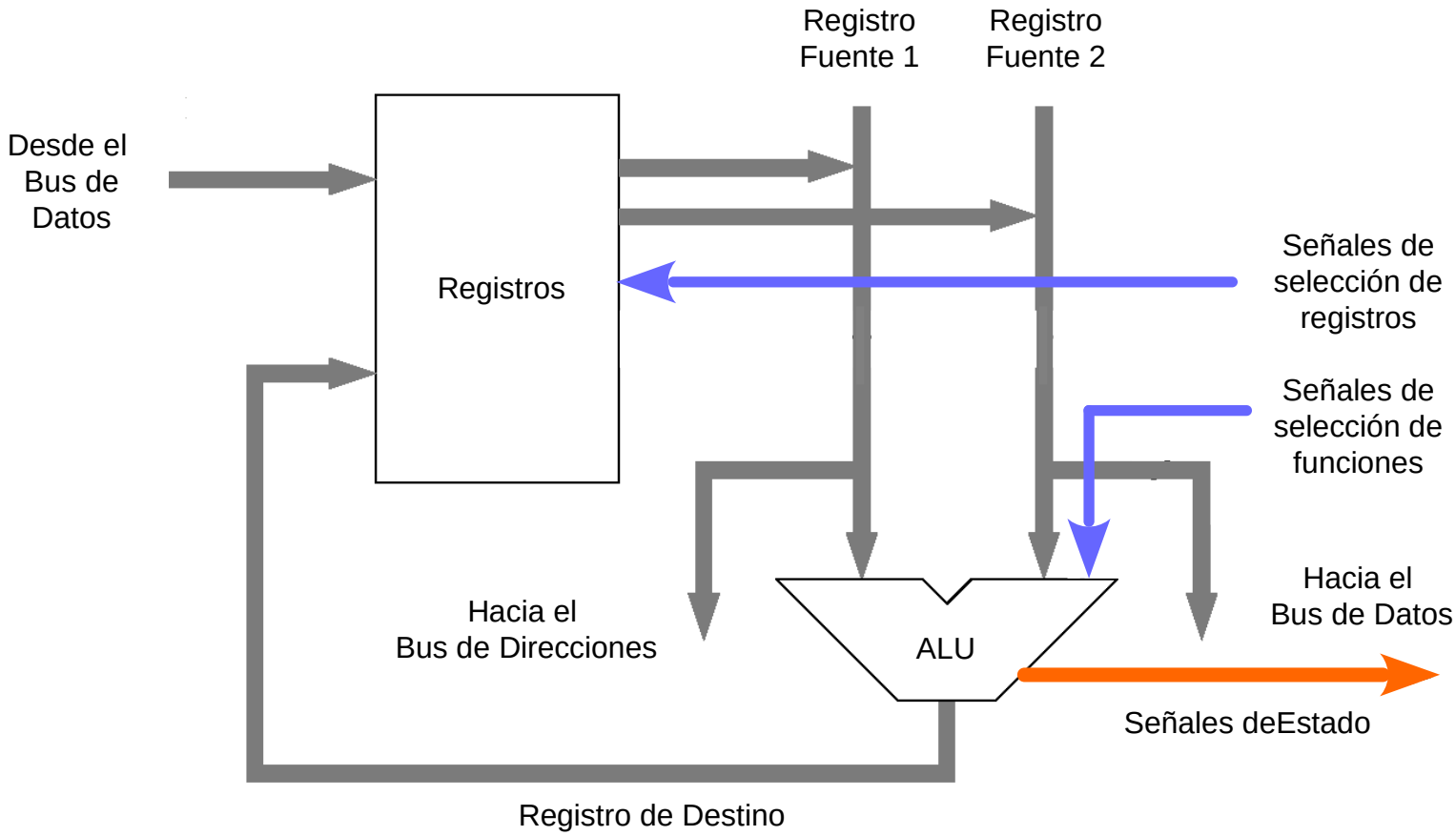
El procesador está compuesto por tres bloques:

- **Almacenamiento de datos** compuesto por los registros;
- **Procesamiento de datos**, compuesta por una Unidad Aritmetico Logica (ALU), y
- **Unidad de Control**, el cual interpreta las instructiones y ejecuta las acciones necesarias para su ejecución.



Arquitectura del procesador

El camino de datos (Datapath)

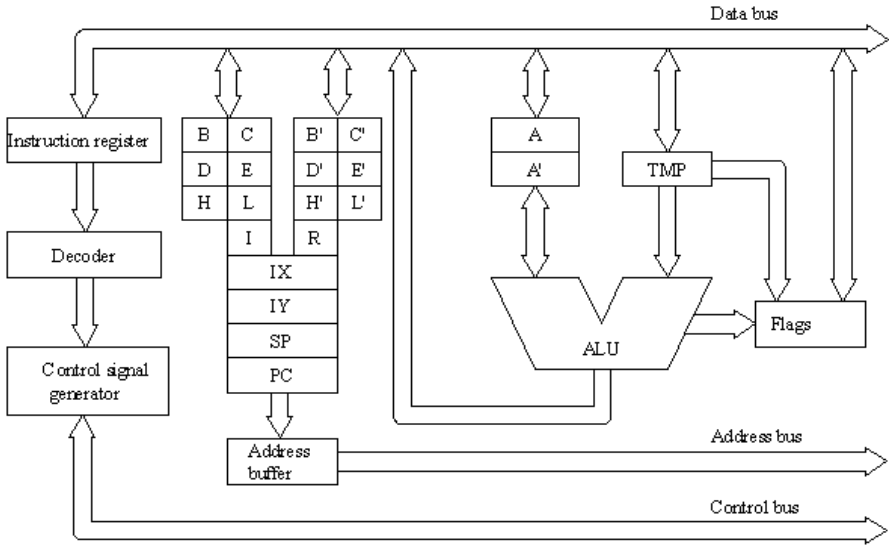


El datapath está conformado por un ***grupo de registros*** (*register file*) y la ***unidad aritmetica y logica*** (ALU).

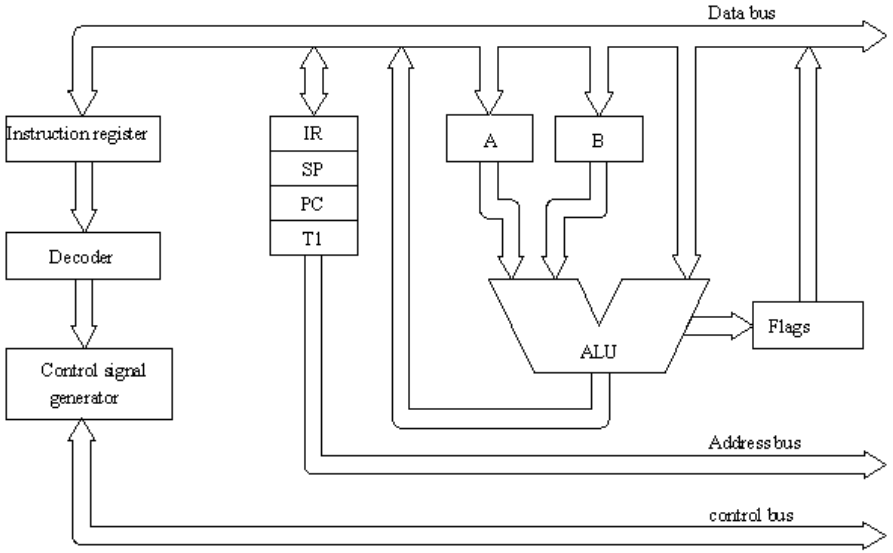
Arquitectura del procesador

Procesadores de 8 bits

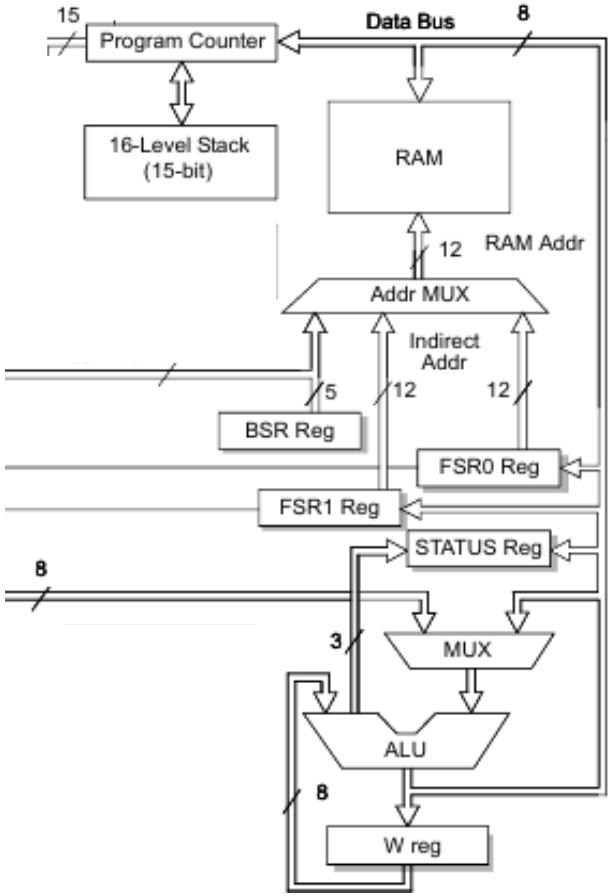
Z80



6800

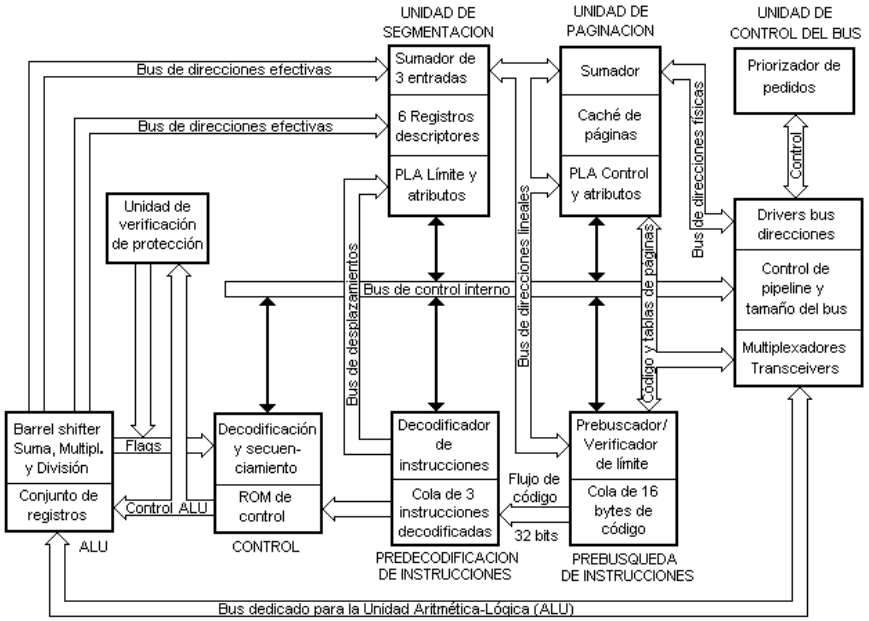


Microchip



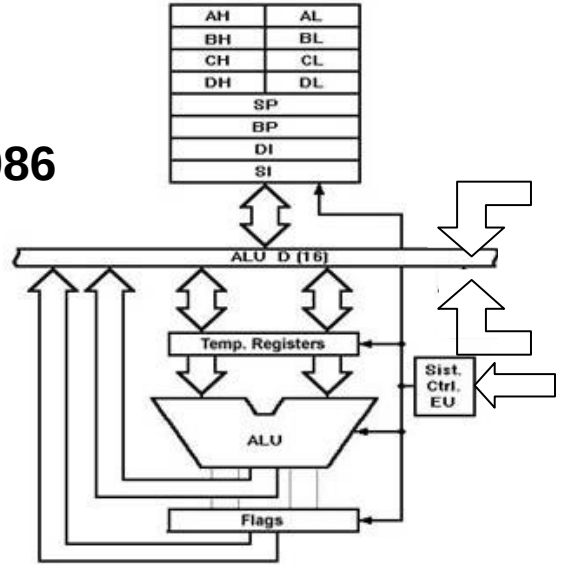
Arquitectura del procesador

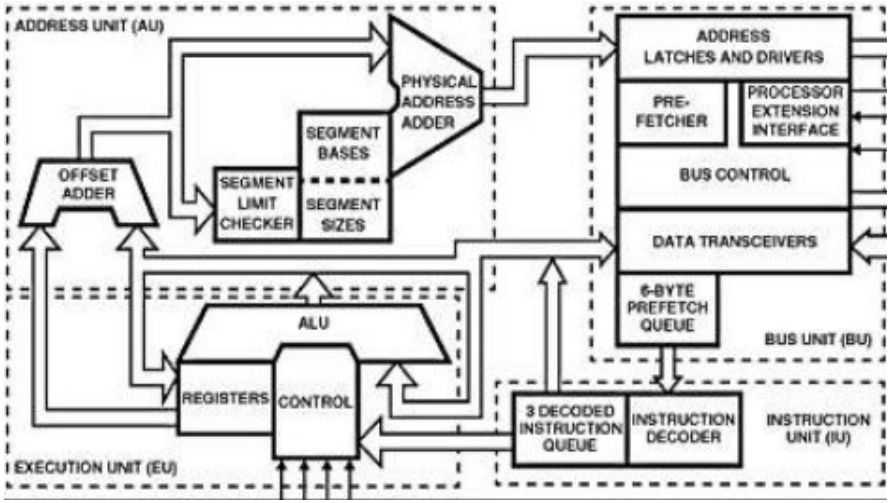
Procesadores 80X86



80386

8086

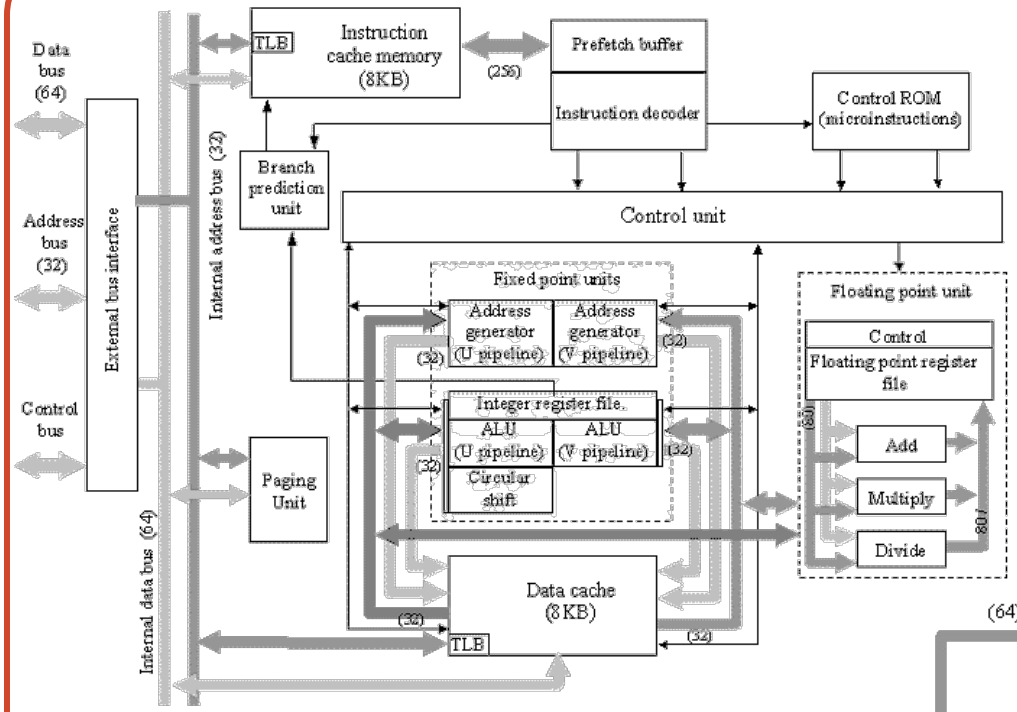




80286

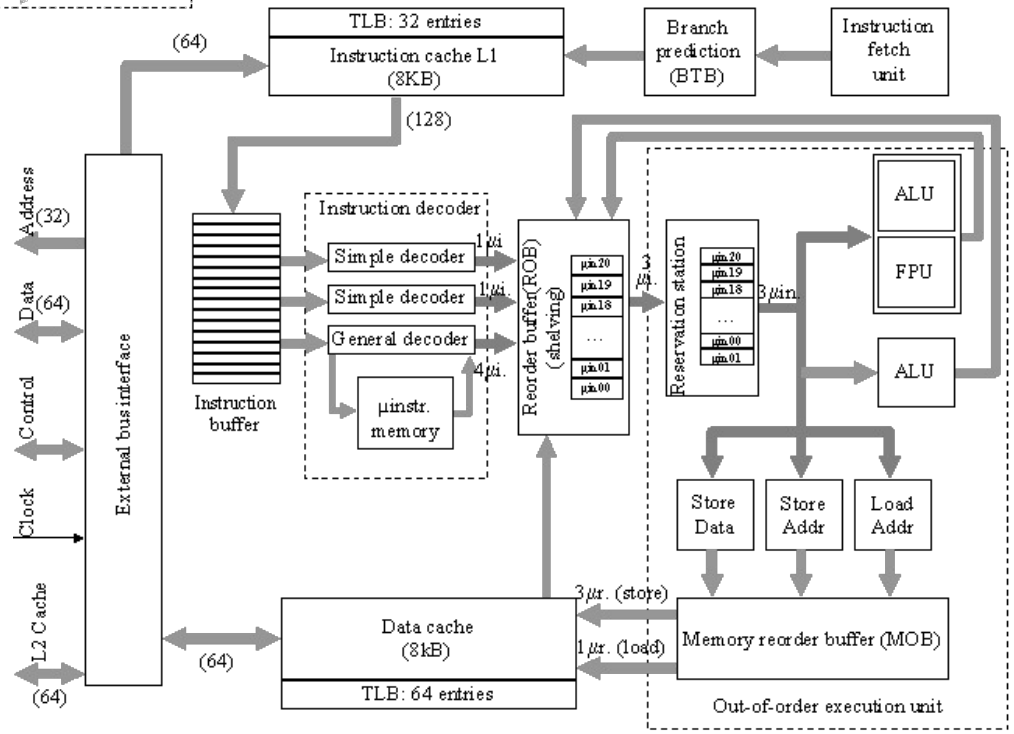
Arquitectura del procesador

Procesadores 80X86



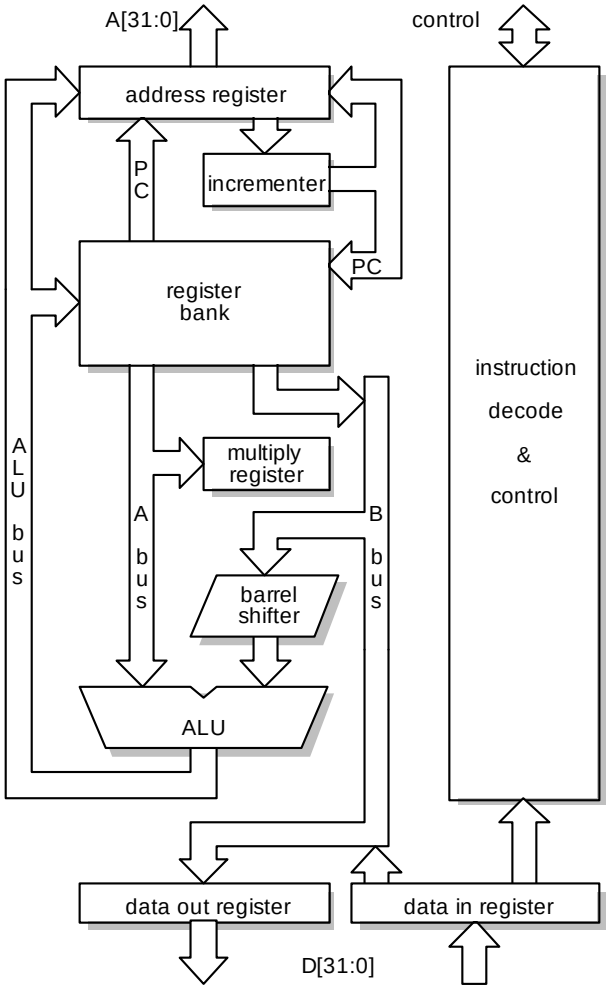
Pentium

Pentium Pro

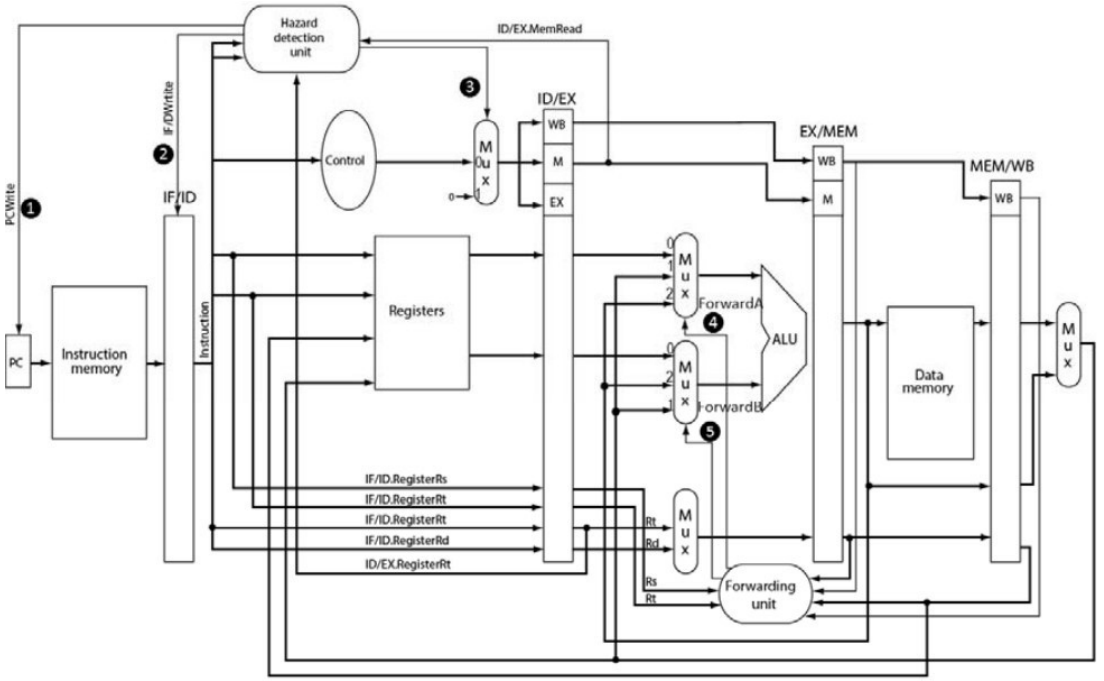


Arquitectura del procesador

Procesadores RISC



ARM

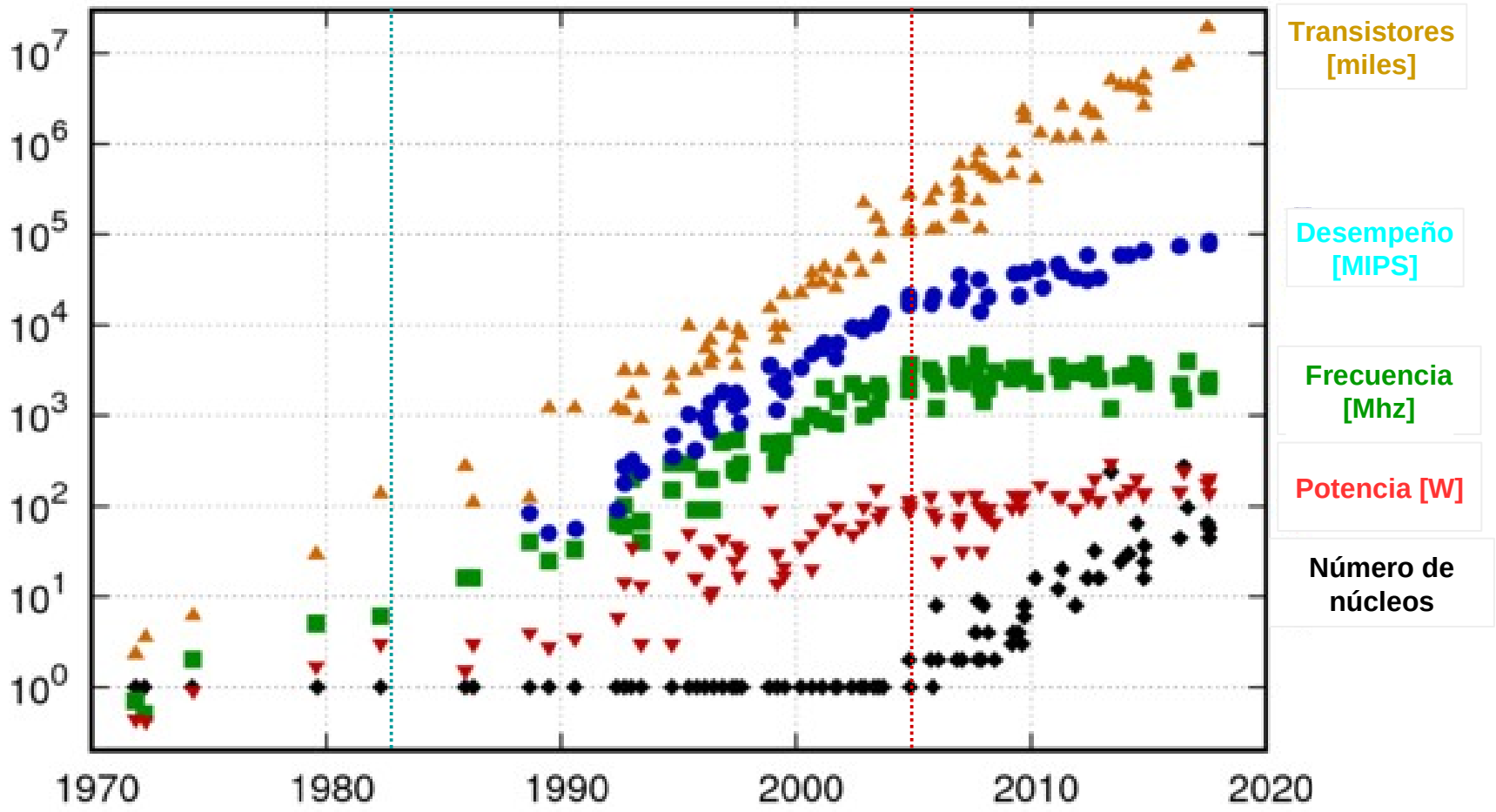


MIPS

Evolución de los procesadores

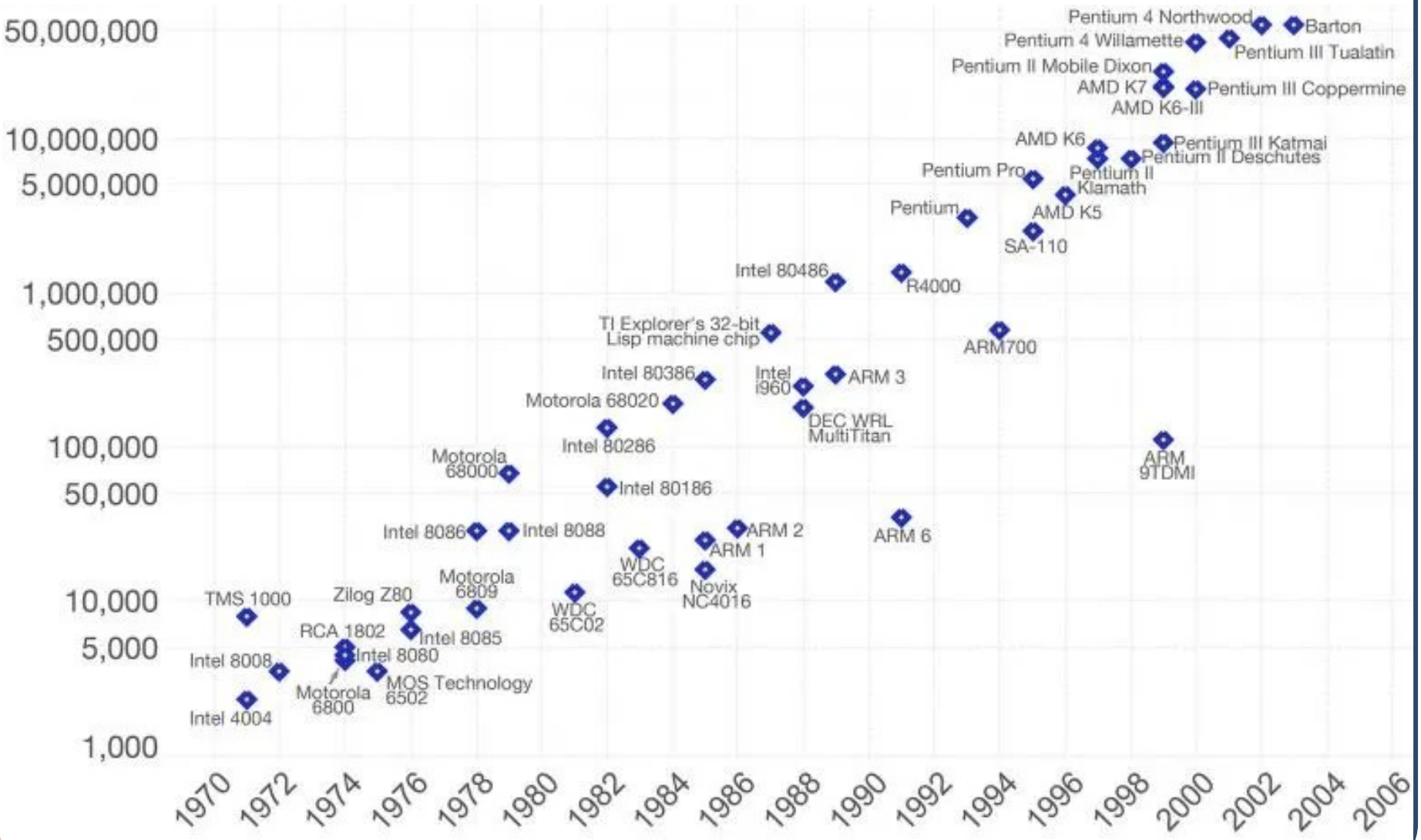
Ley de Moore - el número de transistores en un microprocesador se duplica cada dos años.

Ley de Pollack - el aumento del rendimiento debido a los avances de la microarquitectura es proporcional a la raíz cuadrada del aumento de la complejidad.

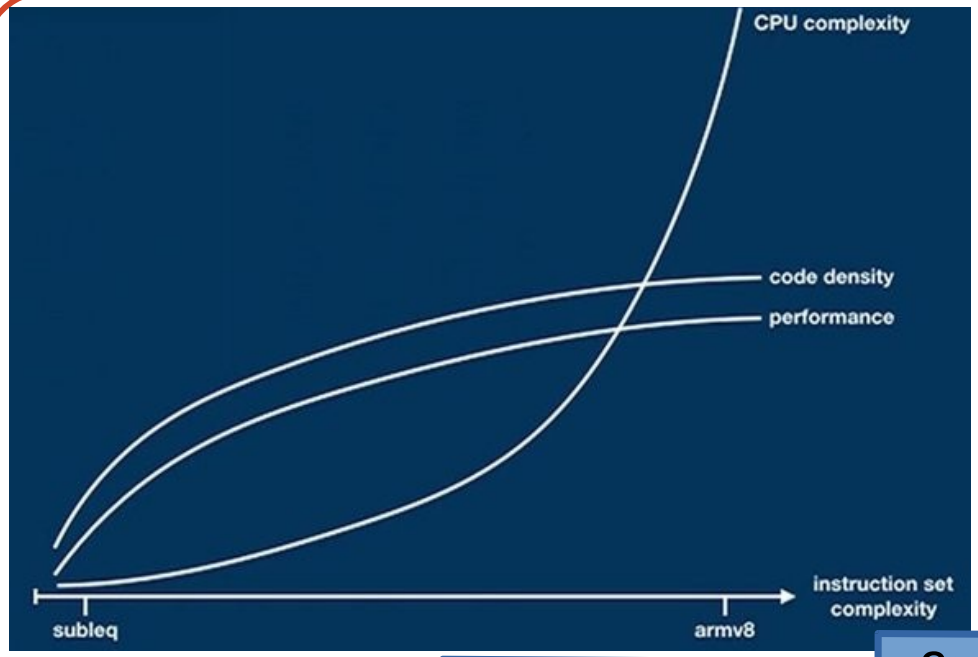


Evolución de los procesadores

Ley de Moore - el número de transistores en un microprocesador se duplica cada dos años.



Evolución de los procesadores



Gran porcentaje de la superficie del integrado es para extraer paralelismo y mitigar el memory wall.

Instrucciones vectorizadas

Cache de Datos de gran ancho de banda

Aceso de memoria de gran ancho de banda

Disponibilidad y confiabilidad de datos mejorada

Prediccion de saltos mejorada

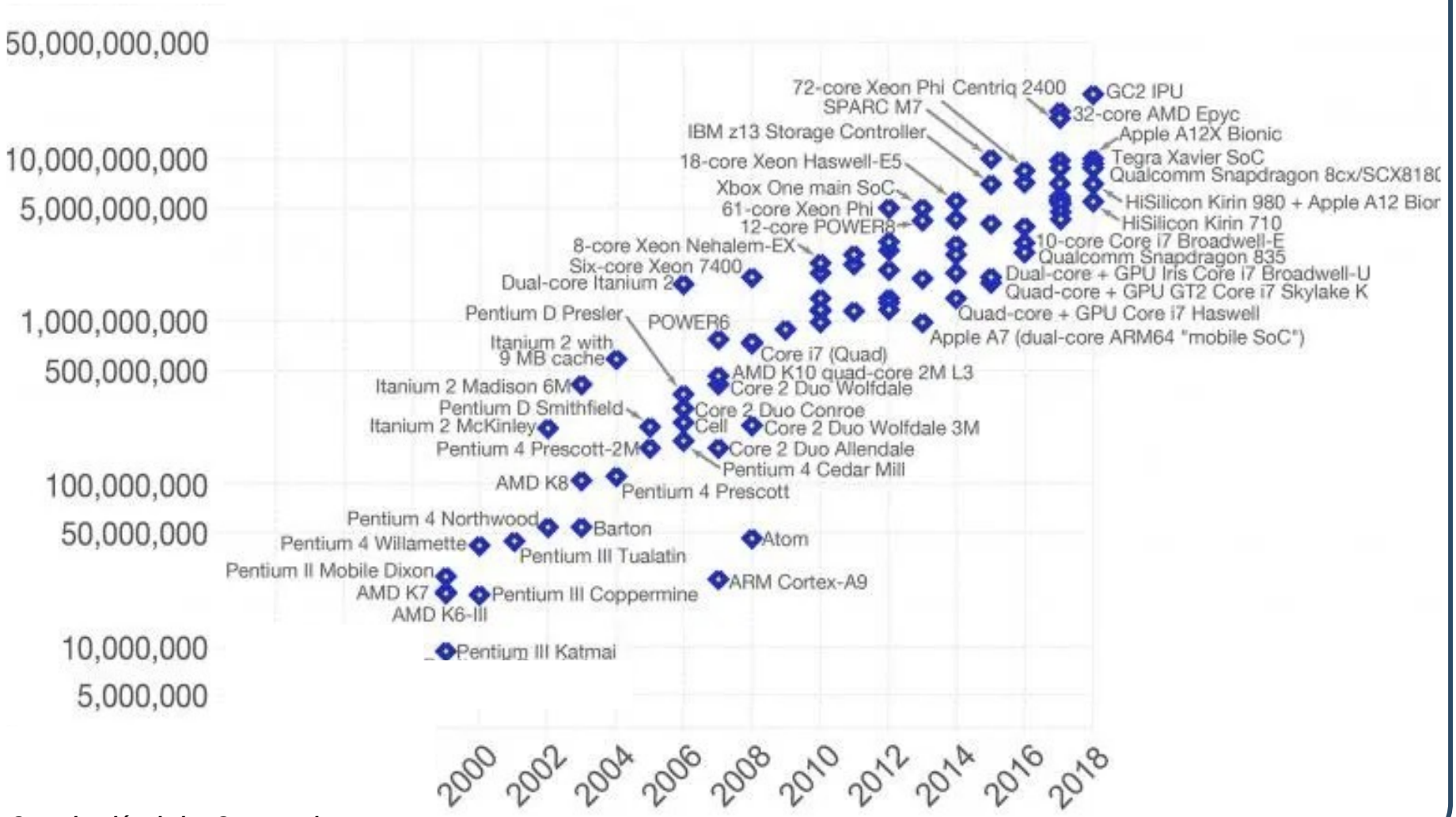
Buffers mas profundos y planificacion mejorada

Decodificacion de alto rendimiento

Busqueda de instrucciones mejorada

Evolución de los procesadores

Ley de Pollack - el aumento del rendimiento debido a los avances de la microarquitectura es proporcional a la raíz cuadrada del aumento de la complejidad.



Evolución de los procesadores

Ley de Pollack - el aumento del rendimiento debido a los avances de la microarquitectura es proporcional a la raíz cuadrada del aumento de la complejidad.