

# Unidad 2 - Niveles de Abstracción, ISA y Arquitectura

## Niveles de Abstracción

Los niveles de abstracción de la computadora van desde el usuario nivel (alto nivel) hasta los dispositivos que componen a la computadora (bajo nivel). Esto significa que los **lenguajes de alto nivel** están menos atados a la organización y arquitectura de la computadora. Estos son usados principalmente para aplicaciones dirigidas al usuario.

Al contrario, el **lenguaje Ensamblador** (Assembler) depende intrínsecamente del procesador en donde lo usemos, por lo que se necesita un entendimiento profundo de su arquitectura. Este es usado para desarrollar drivers, compiladores y aplicaciones embebidas.

## Software

Lo que denominamos software es el conjunto general de programas, o soporte lógico de la computadora, que a su vez son conjuntos de instrucciones que, al ser ejecutadas, realizan tareas en una computadora.

Podemos clasificar el software en:

- Software de Aplicación: programa diseñado para permitir al usuario realizar uno o más tipos de tareas.
- Software de Sistema: programas para controlar y dar soporte a otros programas y/o para interactuar con el hardware, como driver utilizado para el uso de un periférico.

## Compilación



Niveles de Abstracción

Es un programa que traduce un programa escrito en lenguaje de alto nivel a lenguaje de bajo nivel de acuerdo a las especificaciones del hardware.

Las etapas de la compilación son:

- **Análisis:** se verifica el código fuente antes de traducirlo a código de máquina. Consiste en cuatro etapas:
  - Análisis Léxico (scanning): consiste en identificar las palabras clave, identificadores, números, operadores y símbolos de puntuación (tokens) y crear una sola cadena con ellos.
  - Análisis Sintáctico (parsing): toma ese flujo de tokens y verifica la sintaxis, como que los paréntesis estén balanceados, que las sentencias estén bien formadas. Además se genera una representación de la estructura sintáctica del programa a partir de un árbol abstracto de sintaxis.
  - Análisis Semántico: toma el árbol de sintaxis para comprobar que el programa sea semánticamente correcto. Esto asegura que las variables se declaren correctamente y que los tipos de operadores y objetos coincidan, para así evitar por ejemplo la suma entre cadenas de texto y números o el uso de variables no inicializadas. Acá también se genera una tabla de símbolos que representan los objetos (clases, variables, funciones).
  - Generación de Código: genera la representación intermedia a partir de la tabla de símbolos y el árbol de sintaxis.
- **Síntesis:** genera la salida del compilador, en el lenguaje objeto o lenguaje ejecutable, específico de la máquina objetivo (procesador donde se ejecutará). Generalmente consiste en cuatro sub-etapas:
  - Interfaz del lenguaje: transforma del lenguaje de alto nivel a un lenguaje intermedio. Esta etapa está atada al lenguaje que estemos traduciendo, pero no de la arquitectura.
  - Optimización: transformaciones que mejoran el acceso a memoria y explotan de manera más efectiva el hardware.
  - Generación de código: se elijen las instrucciones y su orden de ejecución, además de los registros. Esta etapa depende 100% de la arquitectura.

## Optimización

Las diferentes **optimizaciones** que se realizan se clasifican en:

- Optimización del alto nivel: se optimiza el código a nivel de origen. Aquí podemos encontrarnos transformaciones en línea (reemplaza llamadas a funciones por el cuerpo de la función) y transformación de bucles, que mejoran el uso de memoria y la integración de otros procedimientos.
- Optimización local: funciona dentro de bloques básicos y suele ejecutarse antes y después de la optimización global. Se optimiza teniendo en cuenta bucles, registros usados y el procesador mismo.

- Optimización global: funciona a través de múltiples bloques básicos.
- Asignación de registros: se asigna variables a los registros dentro de las regiones del código.  
Tanto a nivel local como global, se eliminan subexpresiones comunes, la propagación constante, propagación de copias, eliminación de almacenamiento inactivo y otros.

## Ensamblado

El ensamblador traduce un programa en **lenguaje ensamblador** a código de máquina. Esto lo hace primero asignando una dirección a cada instrucción y buscando los símbolos(etiquetas y nombres de variables), y luego genera el código de máquina y la tabla de símbolos. Estos últimos son almacenados en el **archivo objeto**.

El **lenguaje ensamblador** es un lenguaje de programación de bajo nivel, el cual es la representación simbólica (mnemónicos) directa del código de máquina (binario).

## Enlazador

En enlazador se encarga de organizar los recursos necesarios para la ejecución del programa. Enlaza el código objeto con sus bibliotecas, además de quitar aquellas que no sean necesarias, y genera un fichero ejecutable.

Los programas enlazados dinámicamente realizan este enlace entre el programa y las bibliotecas en tiempo de ejecución.

## Ejecución

El sistema operativo lee el archivo ejecutable desde el dispositivo de almacenamiento y lo carga en la memoria principal para su ejecución. La ubicación del código, los datos, las variables locales y la pila dependerá del **ISA**.

## Conjunto de Instrucciones de la

```
1  int f, g, y; // variables globales
2
3  int main(void) {
4      f = 2;
5      g = 3;
6      y = sum(f, g);
7
8      return y;
9  }
10
11 int sum(int a, int b) {
12     return (a + b);
13 }
```

Código fuente

```
1  .data
2  f: .word
3  g: .word
4  y: .word
5
6  .text
7  main:
8      addi    sp, sp, -4 # stack
9      sw      ra, 0(sp) # almacenar ra en stack
10     addi    a0, x0, 2 # a0 = 2
11     sw      a0, f      # f = 2
12     addi    a1, x0, 3 # a1 = 3
13     sw      a1, g      # g = 3
14     jal     sum        # llamar a subrutina sum
15     sw      s0, y      # y = sum(f, g)
16     lw      ra, 0(sp) # restaurar ra del stack
17     addi    sp, sp, 4 # restaurar puntero del stack
18     jr      ra        # retornar al OS
19
20 sum:
21     add     s0, a0, a1 # s0 = a + b
22     jr      ra        # retornar a donde se llamo
```

Código ensamblador

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	0x03E00008
Address	Data
0x10000000	f
0x10000004	g
0x10000008	y

Programa enlazado

# Arquitectura

---

El conjunto de instrucciones de la arquitectura (ISA) es el modelo abstracto que define la información que necesitamos para programar en el lenguaje de máquina para una computadora, y establece la interfaz entre el hardware y el software.

No hay que confundirlo con la **Microarquitectura**, que son las técnicas de diseño para **implementar** el ISA. Una de las razones por las que el ISA sirve de interfaz justamente es que puede implementarse con diferentes microarquitecturas con distintos criterios de diseño.

El ISA está compuesto por

- **Modos de Operación:** formas en las que se puede interactuar con la máquina (usuario, supervisor, maquina virtual, etc.).
- **Tipos de Datos:** tipos (bit, byte, word) y formatos (entero, flotante, etc.) de datos soportados.
- **Conjunto de Instrucciones:** las operaciones que puede realizar la CPU (aritmética, lógica, control de flujo, movimiento de datos).
- **Operandos y Modos de Direccionamiento:** operandos utilizados por las instrucciones (registros, puertos, direcciones de memoria) y los mecanismos para que las instrucciones accedan a los datos.
- **Organización de la Memoria:** la ubicación de los datos y programas en memoria, dirección de reset, modelo de entrada/salida para gestionar periféricos, etc.
- **Gestión de Eventos:** modelo para gestionar eventos ajenos al programa (**excepciones e interrupciones**) y la ubicación de la información relacionada a ellos.

## Modos de Operación

Los modos de operación establecen las restricciones de **recursos** (registros, memoria y puertos) y de **operaciones** para los programas ejecutados por el procesador. Es un mecanismo para protección frente a **fallas y comportamiento malicioso**.

Existe una jerarquía de modos de operación, de más privilegiados (menos restricciones) a menos privilegiados (recursos e instrucciones restringidas). El ISA puede además proporcionar **mecanismos especiales** por los cuales un modo inferior puede acceder a recursos de uno superior.

### Modo No Restringido/Supervisor

Permite al procesador realizar cualquier operación permitida por la arquitectura. Ejecutar una instrucción, una operación de E/S o leer/escribir cualquier área de la memoria o registro.

### Modos Restringidos

Son también llamados "**modos usuario**" y por lo general imponen restricciones sobre la gestión avanzada de memoria y operaciones de E/S que alteren el estado de la computadora.

## Modo Hypervisor

Puede ejecutar varios programas en modo supervisor, que sirve para implementar sistemas de **máquinas virtuales**.

## Organización de la Memoria

La memoria está organizada de forma lineal, es decir, cada posición de la memoria está en una posición consecutiva de otra. Estas están numeradas con un número único llamado **dirección** correspondiente al espacio de memoria del procesador.

La **organización** de la memoria es la subdivisión de la memoria según su uso e información que almacene. Los espacios en los que se divide son:

- **Espacio de Programa:** almacena el programa a ejecutar, junto con la dirección donde inicia el programa (**dirección de reset**).
- **Espacio de Datos:** almacena la información utilizada durante la ejecución del programa y los periféricos.
- **Espacio de configuración:** almacena la configuración del procesador (tablas de páginas, tablas de vectores de interrupciones, registros de configuración y estados, ubicación de periféricos).

## Tipos de Datos

El dato más pequeño que puede direccionar un procesador en memoria es el **byte**. Las palabras multi-byte (más de un byte) son almacenadas como una secuencia de bytes, donde la dirección de la palabra es la dirección del byte más bajo.

Además, existen dos formas de almacenar palabras multi-byte:

- Big-endian: el byte más significativo se almacena en la dirección más baja.
- Little-endian: el byte menos significativo se almacena en la dirección más baja. Este es el caso de RISC-V

## Gestión de Eventos

Durante la ejecución de los programas pueden ocurrir eventos inesperados (no generados por el programa) que requieren alguna acción. Estos eventos pueden ser externos o internos al procesador,

es decir, asincrónicos o sincrónicos a la ejecución de las instrucciones.

Los mecanismos usados ante estos eventos son las **interrupciones y excepciones**

## Interrupción

Suspende temporalmente la ejecución del programa, una vez ejecutada la instrucción, para atender el evento. Se puede originar por un periférico que solicita ser atendido o por una solicitud de servicios al sistema operativo

## Excepción

Aborta la ejecución de la instrucción por errores en el procesador mismo o procesadores auxiliares (violación de áreas de memoria u otros). Se origina por códigos de operación incorrectos, errores en la ejecución de instrucciones, instrucción no permitida para el modo actual, violación de áreas de memoria, errores de representación (no se qué es eso), u otros.

## Conjunto de Instrucciones

El lenguaje de máquina está compuesto por instrucciones que especifican la operación a realizar y los operandos necesarios para realizarla. Estas instrucciones incluyen un código de operación (OP Code) para indicar la operación a realizar y especificadores de operandos (Addr1, Addr2, Imm) que indican la ubicación de los datos a utilizar en la instrucción.

## Tamaño de las Instrucciones

Las instrucciones puede además tener un **tamaño variable o fijo**. Las instrucciones de tamaño variable permiten mayor flexibilidad y conjuntos más compactos. Las instrucciones de tamaño fijo simplifican el ISA y hacen más eficiente la ejecución y paralelización de instrucciones.

## Características Deseables

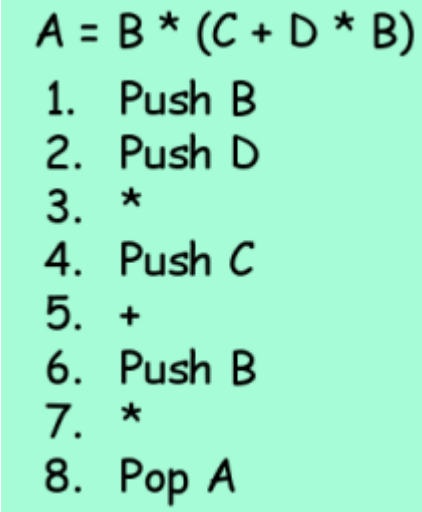
Hay ciertas características deseables en un conjunto de instrucciones:

- **Completo:** que cualquier instrucción pueda ejecutarse en tiempo finito.
- **Eficiente:** que permita alta velocidad de cálculo y minimice el uso de recursos.
- **Autocontenidas:** que contenga toda la información necesaria para ejecutarse.
- **Independientes:** que no depende de la ejecución de alguna otra instrucción.

## Cantidad de Operandos

Dependiendo la cantidad de operandos, el ISA se clasificará en:

- **Operandos Implícitos:** también llamado Stack Architecture, es un modelo computacional donde la memoria toma la forma de pilas. Las instrucciones operan con los datos que se encuentran en el tope de las pilas, que son reemplazados por los resultados. Las instrucciones de este modelo no utilizan campos de dirección, ya que están determinados implícitamente por las pilas. La ventaja de esta arquitectura es que tiene una buena densidad de código, bajos requerimientos de hardware y compiladores sencillos.



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

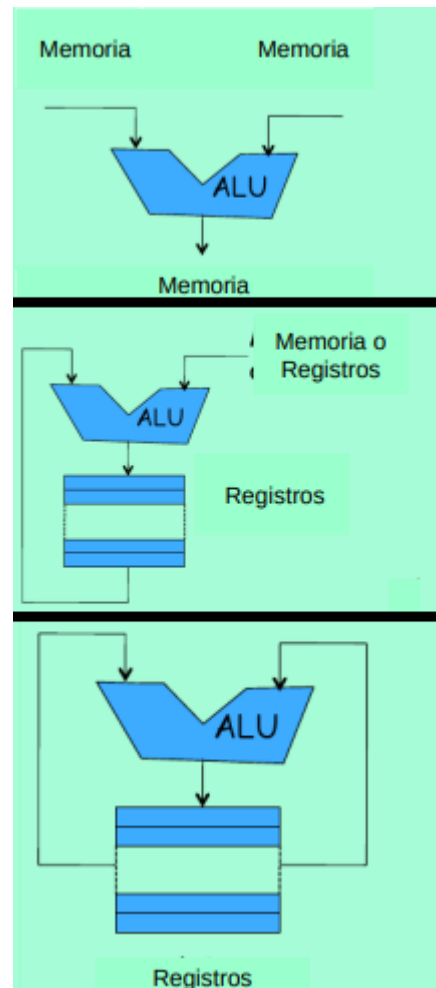
Ejemplo de código en Stack Architecture

- Por otro lado, la pila es el cuello de botella de la ejecución, es difícil de paralelizar instrucciones, se necesitan instrucciones adicionales para cargar datos y los compiladores óptimos son difíciles de desarrollar.
- **Operandos Explícitos:** también llamado Accumulator Architecture, es el modelo en donde se usa un acumulador para procesar todas las instrucciones y almacenar los resultados. En este, el formato de instrucción utiliza un campo de dirección uno de los operandos y el destino del resultado están implícitos (el acumulador). La ventaja que tiene es que las instrucciones son cortas y flexibles, los programas son pequeños, los requerimientos de hardware son bajos y es difícil de diseñar y comprender. La desventaja es que el tráfico de memoria es muy alto, el acumulador se convierte en el cuello de botella y es difícil de paralelizar y segmentar instrucciones.
  - **Registros de Propósitos Generales:** también llamado General Purpose Register Architecture, es un modelo donde todos los registros pueden utilizarse para todas las instrucciones, además de almacenar datos. En este, las instrucciones tienen tres campos de direcciones, dos para operandos y uno para el resultado. Existen tres variantes para esta arquitectura/modelo:
    - Memory-Memory: Tres operandos (dos fuentes y un destino) localizados en **memoria**. Requiere pocas instrucciones, y los compiladores son fáciles de desarrollar, pero genera un alto tráfico de memoria a un número variable de ciclos de reloj.
    - Memory-Register: Dos operandos (un fuente y un destino) localizados en **registros** y otro operando fuente en **memoria**. El formato de instrucción es fácil de codificar con buena densidad de código, pero los operandos no son equivalentes, además de generar un número variable de ciclos de reloj y limitar el número de registros
    - Register-Register: Tres operandos (dos fuentes y un destino) localizados en **registros**. Este modelo requiere instrucciones **Load** y **Store** para mover datos entre memoria y registros. La codificación de la instrucción es simple y de longitud fija, al igual que los ciclos de reloj, además de ser fácil de paralelizar y segmentar; pero requiere un mayor número de instrucciones, además de depender de la calidad del compilador.

# Complejidad de las Operaciones

Un ISA puede clasificarse también según la complejidad de las operaciones que puede realizar

- **Conjunto de Instrucciones Reducido (RISC):** simplifica la microarquitectura implementando de manera eficiente solo las instrucciones más utilizadas, lo cual mejora el rendimiento global y logra un mayor potencial de paralelización y reducción del costo de acceso a memoria. De esta forma, como características principales presenta
  - Codificación Uniforme: que permite una decodificación más rápida
  - Conjunto de Registros Homogéneo: cualquier registro es utilizado en cualquier contexto
  - Modos de Direccionamiento y Tipos de Datos **sencillos**.



Ejemplo de código en Stack Architecture

- **Conjunto de Instrucciones Complejas (CISC):** es un modelo amplio y que permite operaciones complejas entre operandos tanto en memoria como en registros, dando así un código más compacto. Sus características son:
  - Codificación No-Uniforme: complica la decodificación de las instrucciones.
  - Ejecución Multiciclo: dificulta la paralelización de las instrucciones.
  - Instrucciones Microprogramadas.

## Instrucciones más Comunes

Las operaciones que se pueden encontrar en la mayoría de ISAs son:

- **Movimiento de Datos**
  - Establecer un registro a una constante.
  - Mover datos de una posición en memoria a un registro.
  - Leer y escribir datos en periféricos.
- **Operaciones Matemáticas y Lógicas:**
  - Sumar, restar, multiplicar o dividir registros (binarias)



- Operaciones Lógicas y comparaciones
- Operaciones bit a bit
- Control de Flujo
  - Saltar a otra posición del programa.
  - Saltar bajo cierta condición.
  - Salta guardando la posición actual (para poder volver).
- Control de CPU y Otras
  - Gestionar interrupciones y excepciones
  - Gestionar información de los modos de operación
  - Gestionar recursos de CPU

## Modos de Direccionamiento

Los modos de direccionamiento del ISA definen la forma de identificar los operandos. Es decir, especifican la forma de calcular la dirección de memoria de un operando, utilizando información contenida en los registros y constantes dentro de la instrucción. Entre los modos más utilizados se encuentran

- **Inmediato:** El operando está especificado en la instrucción. Por ejemplo: `li a0, 15h` almacenar el inmediato `15h` en el registro `a0`
- **Directo:** la instrucción especifica la dirección del operando. Algo así como `lw a0, (0x10010000)` en caso
- **Indirecto:** la instrucción especifica dónde se encuentra la dirección efectiva del operando. Algo como `lw t0, (a0)` donde el registro `a0` contiene la dirección efectiva del operando
- **Indirecto Recursivo:** la instrucción especifica dónde se encuentra la dirección efectiva del operando, pero ésta tiene un campo de indexación y otro indirecto, por lo que podría repetirse el direccionamiento hasta que se llegara a una dirección sin campo indirecto.
- **Implícito:** El operando está implícito en la instrucción
- **Relativo:** la instrucción especifica la dirección efectiva a partir de un desplazamiento relativo al PC (Program Counter) con un offset especificado en la instrucción.
- **Registro:** El operando es especificado en uno de los registros. Por ejemplo `ADD t0, t1, x0;`  
`ADD t0, t0, t2`
- **Registro Indirecto:** La instrucción especifica el registro en el cual la dirección de memoria está almacenada. Por ejemplo `SW t0, 0(t1)` donde `t1` contiene el valor de la dirección efectiva del operando
- **Indexado:** la dirección efectiva se obtiene sumándole a un registro offset una base provista por la instrucción. No hay ejemplo de esto en RISC V, pero sería como el relativo, pero nosotros

especificamos el registro base en lugar de usar el PC.

- **Registro Base:** la dirección efectiva se obtiene sumándole a un registro base un offset provisto por la instrucción. Lo mismo que el indexado, pero acá lo que provee la instrucción es el offset en lugar de la base.
- **Autoincremento/Autodecremento:** es como el indirecto, pero cada vez que se usa el registro se autoincrementa/autodecrementa automáticamente.
- **Condicional:** existen instrucciones en algunos procesadores que se ejecutan de manera condicional
- **Salto:** el salto o skip es como el relativo pero el offset es siempre +1.
- **Pila:** se basa en la estructura LIFO (Pila justamente), donde el puntero de la pila (Stack Pointer) apunta a la última posición ocupada. Como es implícito (ver modo de direccionamiento implícito), solo se usa en algunas instrucciones, como PUSH o POP.

## Arquitectura del Procesador

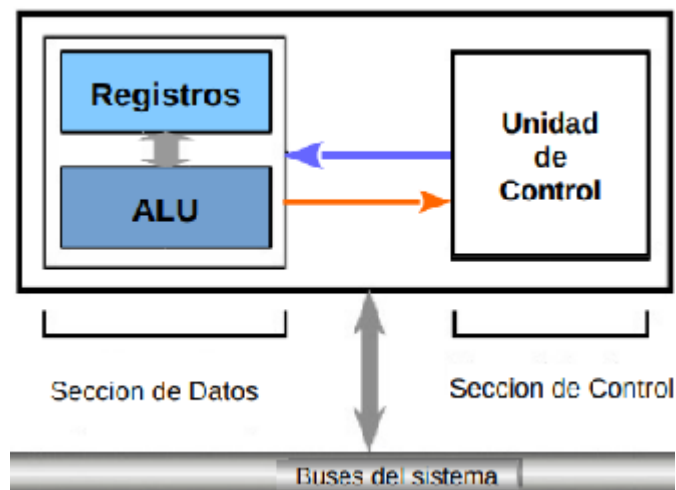
El **Conjunto de Instrucciones** y la **microarquitectura** componen lo que llamamos **arquitectura de una computadora**.

La **microarquitectura** describe las **partes** del procesador, sus interconexiones y cómo interoperan para **implementar** el ISA. Es presentada como diagramas de bloques con estas interconexiones, y suelen incluir al menos:

- **Camino de Datos** (datapath): la ruta que recorren las instrucciones, operandos y resultados. Esta formado por un grupo de registros (register file) y la ALU
- **Unidad de Control** (control unit): el camino que siguen las señales de control, las que manejan el funcionamiento del camino de datos.

El procesador está compuesto por tres bloques:

- **Almacenamiento de Datos:** compuesto por los registros.
- **Procesamiento de Datos:** compuesto por la Unidad Aritmético Lógica (ALU)
- **Unidad de Control:** interpreta las instrucciones y ejecuta las acciones necesarias para su ejecución.



Arquitectura

