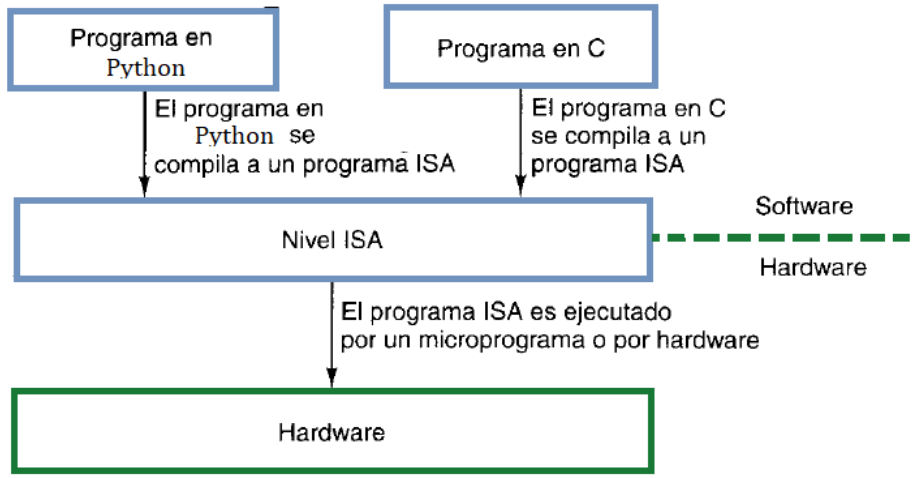
Nivel de Arquitectura del conjunto de Instrucciones

Propiedades del nivel ISA (Instruction Set Architecture)

El nivel ISA es la capa de abstracción en la arquitectura de una computadora que define el conjunto de instrucciones que el procesador puede ejecutar.
Es considerado la interfaz entre el software y el hardware porque permite a los programadores escribir código en ensamblador o lenguaje de alto nivel que después se traducen a instrucciones ejecutables por el procesador.



El nivel ISA es la interfaz entre los compiladores y el hardware.

Factores clave para un buen ISA

1. Un ISA debe definir un conjunto de instrucciones que se pueda implementar con eficiencia en las tecnologías actuales y futuras (crear diseños económicos durante algunas generaciones).
2. Debe ofrecer un objetivo claro para el código compilado

El nivel ISA consiste en:

- **Conjunto de instrucciones**: Operaciones como suma, resta, carga y almacenamiento de datos.
- **Modos de direccionamiento**: Cómo se accede a la memoria y los registros.
- **Registros**: Espacios de almacenamiento temporales dentro del procesador.
- **Formato de instrucciones**: Cómo están estructuradas las instrucciones en bits.
- **Manejo de excepciones e interrupciones**: Mecanismos para eventos inesperados.

Tipos de Arquitecturas ISA son:

x86, ARM, RISC-V y MIPS

TIPOS DE DATOS

Son lo fundamental y están directamente soportados por el hardware o el lenguaje de programación.

- **Enteros (int, short, long, byte)**

  Números sin decimales (Ejemplo: 10, -5, 1000).

- **Flotantes (float, double)**

  Números con decimales (Ejemplo: 3.14, -0.001).

- **Caracteres (char)**

  Un solo carácter (Ejemplo: 'A', '9', '$').

- **Booleanos (bool)**

  Solo puede ser **true** o **false**.

2. Tipos de datos compuestos o estructurados

Son combinaciones de tipos primitivos.

- **Cadenas de texto (string, char[])**

  Conjunto de caracteres ("Hola", "1234").

- **Arreglos (array)**

  Colección de elementos del mismo tipo

  ([1, 2, 3], ['a', 'b', 'c']).

- **Registros (struct, class)**

  Agrupación de diferentes tipos de datos en una sola estructura.

- **Enumeraciones (enum)**

  Conjunto de valores predefinidos (Ejemplo: {Rojo, Verde, Azul}).

3. Tipos de datos abstractos o dinámicos

No están directamente soportados por el hardware, sino que son implementados en software.

- **Listas (list, linked list)**

  Colección de elementos dinámicos.

- **Conjuntos (set)**

  Colección de elementos únicos.

- **Mapas o Diccionarios (map, dictionary)**

  Conjunto de pares clave-valor ({"nombre": "Juan", "edad": 25}).

- **Pilas (stack) y Colas (queue)**

  Estructuras de datos con orden específico (LIFO y FIFO).

4. Tipos de datos personalizados

Son definidos por el usuario utilizando estructuras como **clases, structs o interfaces** en lenguajes de programación orientados a objetos.

Cada lenguaje de programación puede tener variaciones en sus tipos de datos, pero en general, siguen esta clasificación.

Modelos de Memoria

Los **modelos de memoria** son formas en las que una computadora organiza y gestiona la memoria para ejecutar programas. Definen cómo se asigna, accede y comparte la memoria entre diferentes partes del sistema, como el procesador, los programas y los dispositivos.

Los procesadores RISC generalmente utilizan el **modelo Harvard modificado** o una **memoria plana**, dependiendo de la implementación.

Modelo Harvard modificado

- Separa la memoria de datos y la de instrucciones, pero permite cierto intercambio de datos entre ellas.
- Esto mejora la velocidad de acceso sin perder flexibilidad
- Usado en microcontroladores RISC y sistemas embebidos

Modelo de Memoria Plana

- La memoria se trata como un espacio de direcciones único y continuo.
- Común en sistemas RISC modernos como **ARM y RISC-V** en entornos de computación general.

Características en RISC

Los modelos de memoria en RISC siguen principios fundamentales para mejorar el rendimiento:

1) Load/Store (Carga y Almacenamiento)

- Solo los registros pueden operar directamente con la ALU.
- La memoria solo se accede con instrucciones específicas de carga (LOAD) y almacenamiento (STORE).
- Reduce la complejidad del hardware y mejora el rendimiento.

2) Gran uso de registros

- RISC suele tener un banco de registros grande (32 o más).
- Minimiza el acceso a la memoria, que es más lento en comparación con los registros.

3) Pipelining eficiente

- La segmentación del pipeline permite ejecutar múltiples instrucciones simultáneamente.
- Depende de una estructura de memoria que minimice conflictos y latencias.

Modelos de Memoria Virtual en RISC

Los procesadores RISC modernos utilizan memoria virtual y gestión avanzada de memoria:

1) Paginación

- La memoria se divide en páginas de tamaño fijo.
- Usado en sistemas operativos modernos para optimizar la administración de memoria.

2) Memoria Caché

- RISC aprovecha niveles de **caché L1, L2 y L3** para reducir accesos a la RAM.
- ARM y RISC-V utilizan políticas avanzadas de caché para mejorar el rendimiento.

3) Administración de Memoria por el Sistema Operativo

- Utiliza una Unidad de Gestión de Memoria (MMU) para traducir direcciones virtuales a físicas.
- Compatible con modelos de memoria como páginas y segmentación en sistemas multitarea.

FORMATOS DE INSTRUCCIONES

Los formatos de instrucción del RISC-V tienen los siguientes campos:

Código de operación (op): indica el tipo general de instrucción u operación.
Ejemplo: Instrucción de tipo R, operaciones entre registros tiene un opcode (0110011)

Código de función (funct): determina la instrucción concreta dentro del tipo de operación. Operaciones aritméticas.
Ejemplo:
add (suma) tiene un funct3=000 y funct7=0000000
sub (resta) tiene un funct3=000 y funct7=0100000

El funct3 es el mismo porque pertenecen al grupo de operaciones aritméticas (suma/resta).

El funct7 hace la diferencia, una es add y la otra sub.

Operando en registro (rs1, rs2, rd): codifica un número del registro del procesador.

Ejemplo:

rs1 es el registro fuente 1 para el primer operando.
rs2 es el registro fuente 2 para el segundo operando.
rd es el registro destino.

Formato Tipo R (Register)

- Usado para instrucciones aritméticas, lógicas y de manipulación de registros.
- Opera con tres registros: dos operandos y un destino.
- No usa direcciones de memoria ni valores inmediatos.

Opcode (7 bits)	rd (5 bits)	funct3 (3bits)	rs1(5bits)	rs2 (5 bits)	funct7 (7 bits)

add x5, x6, x7 # x5 = x6 + x7

add → Suma registros
x6, x7 → Registros fuente
x5 → Registro destino

Representación binaria de la instrucción, en RISC-V cada registro tiene un número asociado

Registro	Número	Nombre
x0	00000	siempre 0
x1	00001	ra (return address)
x2	00010	sp (stack pointer)
x3	00011	gp (global pointer)
x4	00100	tp (thread pointer)
x5	00101	t0 (temporal 0) registro destino

x6	00110	t1 (temporal 1), registro fuente 1
x7	00111	t2 (temporal 2) registro fuente 2

La **instrucción** (add x5, x6, x7) está codificada en **32 bits** para que el procesador pueda entenderla y ejecutarla.

Se observa que cada campo tiene 5 bits por lo que se pueden representar 32 registros

$(2^5 = 32)$ correspondientes a x0-x31.

Por lo que el binario completo es:

Opcode (7 bits)	rd (5 bits)	funct3 (3bits)	rs1(5bits)	rs2 (5 bits)	funct7(7 bits)

funct7 rs2 rs1 funct3 rd opcode
0000000 00111 00110 000 00101 0110011

Resultado de la operación

Si se dan valores a los registros fuentes:
x6=5
x7=10
después de que se ejecuta la instrucción, el resultado de la operación suma se almacena en el registro (x5) y este valor también se representa en 32 bits.

add x5, x6,x7 el resultado es: x5=5+10=15

la representación binaria del resultado 15 en 32 bits es:

00000000000000000000000000001111

Formato Tipo I (Immediate)

- Usado para instrucciones con valores inmediatos, accesos a memoria y saltos condicionales.
- Tiene un registro fuente, un registro destino y un valor inmediato.

imm: valor inmediato (número constante)
rs1: Registro fuente 1 (primer operando)
funct3: Determina el tipo de operación (addi, jalr, lw…)
rd: Registro destino
opcode: identifica la categoría general de la instrucción.

addi x5, x6, 10   # x5 = x6 + 10

addi → Suma inmediata
x6 → Registro fuente
x5 → Registro destino
10 → Valor inmediato

El código para operaciones con inmediatos es:

0010011 por lo tanto, la representación queda:

imm	rs1	funct3	rd	opcode
000000000010	00110	000	00101	0010011

Formato Tipo J (Jump o Salto)

- Usado para instrucciones de salto incondicional.
- Contiene una dirección de destino de 26 bits, permitiendo saltos lejanos.

jal x1, etiqueta # Salto largo a "etiqueta" y guarda la dirección de retorno en el registro x1

jal-jump and link

La instrucción **jal x1, 10** realiza un salto incondicional a la dirección calculada sumando **10** al contador de programa (PC), y guarda la dirección de retorno en el registro x1

jal x1, 10 realiza un salto de 10 bytes (en términos de dirección de memoria)

Existen otros de instrucción

Formato Tipo S (Store - Almacenamiento)

- Usado para instrucciones de almacenamiento en memoria (store).
- Similar al formato I, pero en lugar de un registro de destino, usa dos registros fuente y un desplazamiento.

sw x5, 20(x6) # Guarda el valor de x5 en la dirección x6 + 20

w (Store Word) almacena una palabra en memoria.

x5 → Registro fuente (dato a almacenar).

x6 → Registro base.

20 → Desplazamiento en memoria.

Formato Tipo B (Branch - Saltos Condicionales)

- Utilizado para *saltos condicionales* en arquitecturas como *RISC-V* y *MIPS.*
- Usa dos registros fuente y un desplazamiento en vez de un inmediato.

beq x5, x6, etiqueta # Si x5 == x6, salta a "etiqueta"

beq (Branch if Equal**)** compara x5 y x6, si son iguales se realiza el salto

Formato Tipo C (Compacto)

- Usado en RISC-V (RVC) y ARM (Thumb) para reducir el tamaño de las instrucciones.
- Generalmente de 16 bits en lugar de 32 bits, optimizando la memoria en sistemas embebidos.

RISC-V
c.add x5, x6 # Versión compacta de add x5, x5, x6
En Arm
mov r0, r1 # Mueve el valor de r1 a r0 en formato compacto