CISC vs RISC: DE LA GUERRA HISTÓRICA A LA CONVERGENCIA MODERNA

Antes, tenemos que saber que es ISA?

INSTRUCTION SET ARCHITECTURE (ISA): EL LENGUAJE DE LAS MÁQUINAS

PARTE I: FUNDAMENTOS DE ISA

Instruction Set Architecture (ISA) es la interfaz abstracta entre el hardware y el software de bajo nivel.

Define el conjunto de instrucciones que un procesador puede ejecutar, los tipos de datos que maneja, sus registros, los modos de direccionamiento de memoria y el modelo de programación visible para el programador.

Analogía clave:
- ISA = Idioma (español, inglés, mandarín)
- Microarquitectura = Cómo piensa el cerebro (neuronas, sinapsis)
- Implementación física = La boca que habla (cuerdas vocales, lengua)

Un mismo idioma (ISA) puede hablarse por diferentes personas (microarquitecturas) con diferentes cuerdas vocales (tecnologías de fabricación).

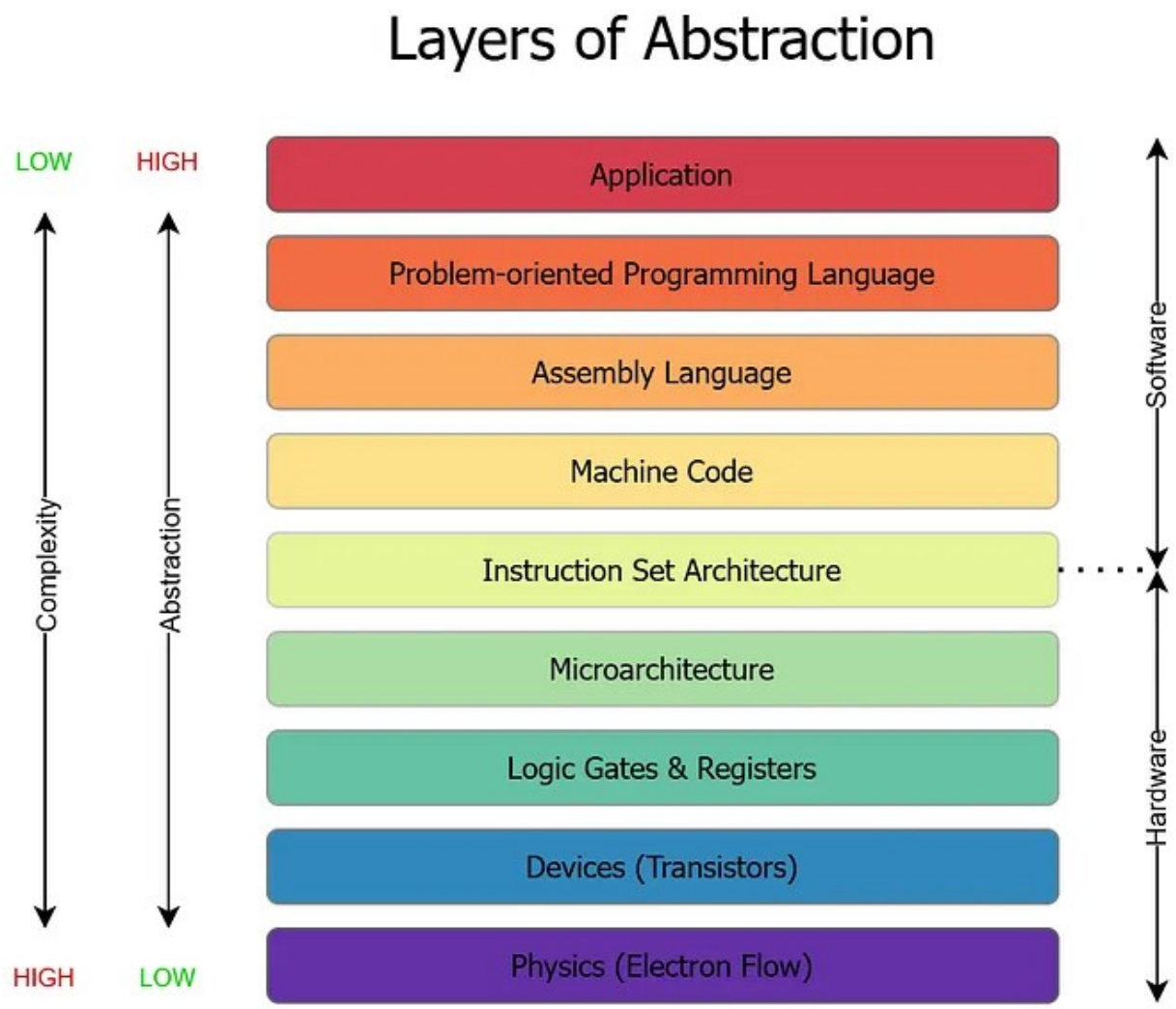
1. Los tres niveles de abstracción (crítico para ingenieros de sistemas)

Nivel	Nombre	¿Qué define?	Ejemplo de cambio
1	Arquitectura del computador	Atributos visibles al programador	ISA (x86 vs ARM)
2	Organización del computador	Unidades funcionales y sus interconexiones	¿Hay caché L3? ¿Cuántos núcleos? ¿Pipeline de 20 etapas?
3	Hardware real	Circuitos, tecnología de fabricación	7nm vs 3nm, frecuencia de reloj, voltaje

Ejemplo práctico:

- Dos procesadores pueden tener la misma ISA x86 (nivel 1)
- Pero diferente organización: uno tiene 8 núcleos y caché L3 de 32MB, otro 4 núcleos y 16MB (nivel 2)
- Y diferente hardware: uno fabricado en 14nm, otro en 3nm (nivel 3)

Consecuencia: Un programa compilado para x86 corre en ambos, pero uno consume 100W y el otro 15W.



https://medium.com/@hannah.scherz.23/layers-of-abstraction-in-computer-systems-3ec127619570

Que es el Pipeline?

La Analogía Esencial: La Línea de Ensamblaje : Imagina una fábrica de autos donde cada coche tarda 10 horas en construirse, pero hay 10 estaciones de trabajo. Si cada estación trabaja 1 hora y pasa el auto a la siguiente, ¿cuánto tarda en salir el primer auto? 10 horas. ¿Y el segundo? 11 horas (no 20). ¿Y el décimo? 19 horas (no 100).

El pipeline es exactamente eso: dividir una tarea larga en etapas pequeñas que se ejecutan en paralelo para diferentes datos/instrucciones.

Pipeline en Procesadores: De la Idea a la Ejecución

¿Por qué existe?

Una instrucción típica pasa por 5 etapas básicas (modelo clásico):

Etapa	Sigla	Qué hace	Analogía fábrica
Fetch	IF	Buscar la instrucción en memoria	Traer el chasis a la línea
Decode	ID	Decodificar qué operación es	Leer el plano del auto
Execute	EX	Realizar la operación (ALU)	Instalar el motor
Memory	MEM	Acceder a memoria si es necesario	Pintar el auto
Writeback	WB	Guardar resultado en registros	Entregar llaves al cliente

- Sin pipeline (secuencial): 5 instrucciones × 5 ciclos = 25 ciclos
- Con pipeline (5 etapas): 5 instrucciones = 9 ciclos (5 + 4)

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

2. Componentes de una ISA

Tipos de datos

Define qué puede entender el procesador:
- Enteros: 8, 16, 32, 64 bits (byte, halfword, word, doubleword)
- Punto flotante: IEEE 754 (single, double precision)
- Vectoriales: SIMD (NEON en ARM, SSE/AVX en x86)
- Direcciones: Tamaño del espacio de memoria direccionable (32-bit = 4GB, 64-bit = exabytes teóricos)

Registros

Memoria ultra-rápida dentro del procesador:

ISA	Registros generales	Tamaño	Características
x86-64	16 (RAX, RBX, RCX, RDX, R8-R15...)	64-bit	Históricamente pocos (acumuladores)

ARM64	31 (X0-X30)	64-bit	Diseño limpio, load/store arquitectura
RISC-V	32 (X0-X31)	32/64-bit	X0 siempre es cero (simplifica hardware)

X0 siempre = 0 en RISC-V: ¿Por qué? Para evitar una instrucción separada de "cargar cero". Ahorra hardware y energía.

3. Modos de direccionamiento

Cómo se especifica la ubicación de los operandos:

Modo	**Descripción**	**Ejemplo**
Inmediato	El dato está en la instrucción misma	`ADD R1, #5` (suma 5 a R1)
Directo	La instrucción contiene la dirección de memoria	`LOAD R1, 0x1000`
Indirecto por registro	El registro contiene la dirección	`LOAD R1, [R2]`
Indexado	Dirección base + desplazamiento	`LOAD R1, [R2 + #4]`
Con auto-incremento	Después de cargar, el puntero avanza	`LOAD R1, [R2]+` (útil para arreglos)

Impacto en rendimiento: Más modos = instrucciones más flexibles pero más complejas de decodificar (tensión CISC vs RISC).

4 Conjunto de instrucciones

Las "palabras" que el procesador entiende. Categorías universales:

Categoría	**Función**	**Ejemplos**
Aritmético-lógicas	Operaciones matemáticas y booleanas	ADD, SUB, AND, OR, XOR
Transferencia de datos	Movimiento entre registros y memoria	LOAD, STORE, MOV, PUSH, POP
Control de flujo	Saltos, llamadas a funciones, retornos	JMP, CALL, RET, BEQ (branch if equal)
Sistema	Privilegios, interrupciones, caché	SYSCALL, CLI (clear interrupts), WFI (wait for interrupt)

Las ISAs Dominantes (Y la que Viene)

x86-64: El imperio CISC

Historia: Intel 8086 (1978) → 80286 → 80386 (32-bit) → AMD64 (2003, extensión 64-bit) → hoy

Características arquitectónicas:
- Carga/acumulador: Operaciones típicamente entre registro y memoria (no solo registro-registro)
- Instrucciones de longitud variable: De 1 byte a 15 bytes (¡complejo de decodificar!)
- Registros especializados: Historia de acumuladores (AX), índices (SI, DI), punteros (SP, BP)
- Compatibilidad heredada: Puede ejecutar código de 1980 en un procesador 2024
- Propietario (Intel/AMD)

Ejemplo de instrucción x86:
asm
```
; Sumar 5 a una variable en memoria, resultado en registro
MOV EAX, [variable]    ; Cargar desde memoria a registro (5 bytes)
ADD EAX, 5             ; Sumar inmediato (3 bytes)
MOV [resultado], EAX   ; Guardar en memoria (5 bytes)
; Total: 13 bytes, 3 instrucciones
```

Dominio de mercado:
- PCs de escritorio y laptops (90%+)
- Servidores empresariales (compartiendo con ARM)
- Consolas de videojuegos (Xbox, PlayStation usan variantes x86/AMD)

Origen del nombre x86
El nombre x86 tiene su origen en la convención de nomenclatura de la línea de procesadores más exitosa de Intel. Comenzó con el microprocesador Intel 8086, presentado en 1978, el primer procesador con arquitectura x86. Los procesadores posteriores, que eran avances del 8086 y mantenían la compatibilidad con versiones anteriores, llevaban números como 80186, 80286, 80386 y 80486. Dado que todos estos procesadores terminaban en 86, se adoptó el término x86 para referirse a toda la familia de estos procesadores y a su arquitectura de conjunto de instrucciones compartida. La x representa los números cambiantes al principio. Aunque Intel pasó posteriormente a utilizar nombres de marca como Pentium, la designación x86 se mantuvo.

Origen del nombre AMD64
Se llama AMD64 porque AMD diseñó esta extensión de arquitectura de 64 bits para los procesadores x86, lanzándola a principios de los 2000s como una forma evolutiva de añadir capacidades de 64 bits a la arquitectura x86 existente, manteniendo la compatibilidad con 32 bits. Aunque Intel terminó usando esta tecnología, el nombre original perduró.

Aquí los puntos claves:

- Creador de la Tecnología: Fue inventada por AMD, no por Intel, lo que le dio el nombre al conjunto de instrucciones.
- Compatibilidad: A diferencia de la propuesta inicial de 64 bits de Intel (IA-64/Itanium), la AMD64 permitió a las computadoras ejecutar software antiguo de 32 bits (x86) sin problemas.
- El Estándar Industrial: Aunque Intel la licenció y la llama oficialmente "Intel 64" o x86-64, el término amd64 se convirtió en el nombre técnico estándar, especialmente en sistemas Linux, para referirse a la arquitectura de 64 bits de escritorio.
- Evolución: AMD tomó el estándar x86 (8086, 386) y lo llevó a 64 bits, convirtiéndose en el estándar universal para computadoras de escritorio y portátiles modernas.

ARM64 (AArch64): El conquistador RISC

Historia: Acorn RISC Machine (1985) → ARM7 (móviles) → ARMv7 (32-bit) → ARMv8-A (64-bit, 2011) → ARMv9 (2021) → hoy

Características arquitectónicas:
- Arquitectura Load/Store pura: Solo LOAD/STORE acceden a memoria. Operaciones aritméticas solo entre registros.
- Instrucciones de longitud fija: 32 bits (ARM64 puede tener algunas de 64, pero dominan 32)
- 32 registros generales: X0-X30 (X31 es SP (Stack Pointer - Apunta al tope de la pila) o XZR (Zero Register - Siempre devuelve **0**) según contexto.
- Ejecución condicional: Muchas instrucciones pueden ejecutarse condicionalmente sin salto (ahorra predicción de saltos)
- Propietario (ARM Holdings)

Ejecución Condicional : El Problema: Los Saltos (Branches) Son Costosos

En computación clásica, cuando queremos hacer algo condicionalmente:
c

```
if (a > b) {
   c = a + 1;  // Solo ejecutar si a > b
}
```

En x86 (y la mayoría de procesadores):

asm

```
; x86-64: Debe usar salto condicional

CMP  RAX, RBX       ; Compara a y b, setea flags
JLE  etiqueta       ; Jump if Less or Equal (salta si a <= b)
                    ; ¡PREDICCIÓN DE SALTO! El procesador adivina si saltará
                    ; Si adivina mal: penalización de 15-20 ciclos (pipeline flush)
ADD  RCX, 1         ; c = a + 1 (solo si no saltó)
etiqueta:
        ; continúa…
```

El problema: Los saltos rompen el pipeline. El procesador debe:
- Predecir si saltará o no (branch prediction)
- Ejecutar especulativamente por el camino predicho
- Desechar el trabajo si predijo mal (misprediction penalty)

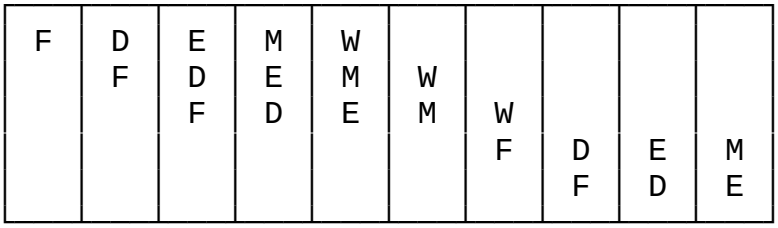
La Solución ARM: Ejecución Condicional (Predicados)
ARM permite añadir una condición a casi cualquier instrucción. La instrucción se ejecuta solo si la condición se cumple, sin necesidad de saltar.

asm

; ARM64: Ejecución condicional sin salto
CMP X0, X1   ; Compara a (X0) y b (X1), setea flags
ADDGT X2, X0, #1   ; Suma SOLO SI X0 > X1 (Greater Than)
; Si condición falsa: la instrucción se convierte en NOP
; ¡Sin salto! El pipeline nunca se interrumpe

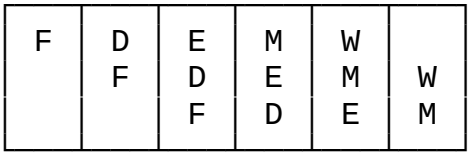
Visualización:

PIPELINE x86 (con salto):



↑ Penalización: 4-5 ciclos perdidos

PIPELINE ARM (condicional):



↑ Sin penalización. El pipeline fluye.

Las Banderas de Condición (Condition Codes)

ARM usa 4 bits de condición en la instrucción (16 posibilidades):

Código	Significado	Condición (flags)
EQ	Equal (igual)	Z == 1
NE	Not Equal (no igual)	Z == 0
GT	Greater Than (mayor, con signo)	Z == 0 && N == V
GE	Greater or Equal (mayor o igual, con signo)	N == V
LT	Less Than (menor, con signo)	N != V
LE	Less or Equal (menor o igual, con signo)	Z == 1 || N != V
HI	Higher (mayor, sin signo)	C == 1 && Z == 0
HS	Higher or Same (mayor o igual, sin signo)	C == 1
LO	Lower (menor, sin signo)	C == 0
LS	Lower or Same (menor o igual, sin signo)	C == 0 || Z == 1
MI	Minus (negativo)	N == 1
PL	Plus (positivo o cero)	N == 0
VS	Overflow Set (desbordamiento)	V == 1
VC	Overflow Clear (sin desbordamiento)	V == 0
AL	Always (siempre)	Siempre verdadero (default)

Flags (NZCV):
Negative: Resultado negativo
Zero: Resultado cero
Carry: Acarreo
o**V**erflow: Desbordamiento con signo

XZR (Zero Register)

Cuando X31 aparece como operando fuente (el valor que lees), el hardware ignora el contenido real y devuelve 0.

asm

// Ejemplos donde X31 = XZR (lectura de cero)

ADD X0, X1, X31 // X0 = X1 + 0 → Equivalente a MOV X0, X1
// X31 aquí es XZR, siempre vale 0

SUB X2, X3, X31 // X2 = X3 - 0 → Equivalente a MOV X2, X3

CMP X0, X31 // Comparar X0 con 0 → Equivalente a CMP X0, #0
// Ahorra bits en la instrucción (no necesita campo inmediato)

¿Por qué es útil?

- Ahorra instrucciones: No necesitas MOV X0, #0 (cargar cero inmediato), usas X31 directamente.
- Decodificación simple: El hardware siempre sabe que X31 = 0 en lectura, no necesita buscar en el banco de registros.
- Energía: Menos conmutación de circuitos.

SP (Stack Pointer)

Cuando X31 aparece como operando destino (donde escribes el resultado), el hardware escribe en el Stack Pointer, no en el registro X31.

asm

// Ejemplos donde X31 = SP (escritura en pila)

SUB X31, X31, #32 // SP = SP - 32 → Reserva 32 bytes en la pila (push)
// Reserva espacio para variables locales

¿Por qué combinarlos?

- Ahorra un número de registro: No necesitas dedicar X30 o X29 solo para SP.
- Codificación eficiente: Las instrucciones de pila son frecuentes; tener SP "gratis" en X31 optimiza el uso del espacio de instrucción.

Ejemplo de instrucción ARM64:
asm

```
// Sumar 5 a una variable en memoria, resultado en registro
LDR X0, [X1, #offset]    // Cargar: X0 = memoria[X1 + offset] (4 bytes)
ADD X0, X0, #5           // Sumar inmediato (4 bytes)
STR X0, [X2, #offset]    // Almacenar: memoria[X2 + offset] = X0 (4 bytes)
// Total: 12 bytes, 3 instrucciones, todas 4 bytes de largo
```

Offset = Desplazamiento / Distancia desde un punto de referencia
En arquitectura de computadores, el offset es un valor numérico (positivo o negativo) que se suma a una dirección base para calcular una dirección final de memoria.

Dirección Final = Dirección Base + Offset

Ejemplo:

Base (X1) = 0x1000
Offset = 0x20 (32 en decimal)
Resultado = 0x1020

Ventaja de longitud fija:
El decodificador sabe dónde termina una instrucción y empieza la siguiente sin analizar. En x86, debe decodificar para saber la longitud (consume energía y complejidad).

Dominio de mercado:
- Smartphones y tablets (99%)
- Servidores cloud creciente (AWS Graviton, Azure Cobalt, Google Axion)
- Laptops (Apple Silicon, Snapdragon X Elite)
- IoT, embebidos, automotriz

RISC-V: El disruptor abierto

Historia: Proyecto UC Berkeley (2010) → Fundación RISC-V (2015) → adopción masiva 2020+

Filosofía radical:
- ISA abierta y libre de regalías: Cualquiera puede implementar sin pagar licencias (vs ARM que cobra por core)
- Modularidad: ISA base mínima + extensiones estándar (M=multiplicación, A=atómica, F=flotante, D=double, C=compressed...)
- Diseño limpio sin legado: Aprendió de 40 años de errores de x86 y ARM

Características arquitectónicas:
- 32 registros (X0-X31): X0 siempre es cero (hardware simplificado)
- Load/Store puro: Como ARM
- Instrucciones de longitud fija: 32 bits (o 16 en extensión C)
- Sin modo de ejecución condicional: Simplifica pipeline (a diferencia de ARM)

Ejemplo de instrucción RISC-V:
asm

```
# Sumar 5 a una variable en memoria, resultado en registro
lw   t0, 0(a0)      # Load word: t0 = memoria[a0] (4 bytes)
addi t0, t0, 5      # Add immediate: t0 = t0 + 5 (4 bytes)
sw   t0, 0(a1)      # Store word: memoria[a1] = t0 (4 bytes)
# Total: 12 bytes, 3 instrucciones
# Nota: t0, a0, a1 son alias de X5, X10, X11 (convención ABI)
```

Dominio emergente – Donde se usa actualmente:
- Controladores de almacenamiento (SSD)
- GPUs y aceleradores de IA (NVIDIA usa RISC-V para controladores internos)
- Microcontroladores IoT de bajo costo
- Supercomputación (RISC-V Vector Extension compite con AVX-512)
- Educación: Cada vez más universidades enseñan arquitectura con RISC-V (open source, visible, sin secretos)

https://www.youtube.com/watch?v=zHjPev8AuB8

https://www.youtube.com/watch?v=1i-P9k_zJno

https://riscv.org/

Actividades Prácticas

Actividad 1: Traductor de ISAs

Objetivo: Entender que el mismo algoritmo se expresa diferente en cada ISA.
Usar: https://godbolt.org/

Instrucción común: `*C = A + B*` (sumar dos variables, guardar en tercera)

Escribir en pseudocódigo C (referencia)
Traducir a x86-64 ensamblador
Traducir a ARM64 ensamblador
Traducir a RISC-V ensamblador

c
```
// C (referencia)
#include <stdio.h>

int main() {
    // Declaración de variables
    int A = 10;      // Primer operando
    int B = 25;      // Segundo operando
    int C;           // Resultado

    // Operación: C = A + B
    C = A + B;

    // Mostrar resultado
    printf("A = %d\n", A);
    printf("B = %d\n", B);
    printf("C = A + B = %d\n", C);

    return 0;
}
```

asm
```
; x86-64 (AT&T syntax)
movl  A(%rip), %eax    ; Cargar A en EAX
addl  B(%rip), %eax    ; Sumar B a EAX
movl  %eax, C(%rip)    ; Guardar en C
```

asm
```
// ARM64
LDR W0, [X1, #A]       // W0 = A (X1 es puntero base)
LDR W2, [X1, #B]       // W2 = B
```

```
ADD W0, W0, W2          // W0 = W0 + W2
STR W0, [X1, #C]        // C = W0


asm
# RISC-V
lw   t0, 0(a0)         # t0 = A (a0 apunta a A)
lw   t1, 4(a0)         # t1 = B (4 bytes offset)
add  t2, t0, t1        # t2 = t0 + t1
sw   t2, 8(a0)         # C = t2 (8 bytes offset)
```

Preguntas:
- ¿Cuál tiene más instrucciones?
- ¿Cuál tiene instrucciones más largas en bytes? (x86 variable, otros fijas)

Actividad 2: Debate "¿Qué ISA para qué proyecto?"

Crear 4 grupos y debatir, cual es la mejor ISA para:

Escenario	**Grupo A**	**Grupo B**	**Grupo C**	**Grupo D**
Smartwatch médico - (batería 1 semana, costo $50)				
Servidor cloud masivo (millones de instancias)				
Consola de videojuegos (rendimiento extremo, retrocompatibilidad)				

A considerar :

- Eficiencia energética (móviles/IoT)
- Ecosistema de software (servidores/gaming)
- Costo de licenciamiento (RISC-V gratis)
- Control del diseño (Apple prefiere ARM para controlar todo)

RISC y CISC

PARTE I: ORIGENES Y FUNDAMENTOS

1. El problema de los años 50: La torre de Babel computacional

En los últimos años de la década de 1950, los procesadores se desarrollaban de forma completamente independiente. Esto significaba que un programa escrito para una máquina no podía ejecutarse en otra sin reescribirlo por completo. Era una situación insostenible para una industria en crecimiento.

IBM tomó la iniciativa. Reunió a sus mejores investigadores para crear una arquitectura que permitiera que el mismo software funcionara en múltiples computadoras, dando nacimiento a dos filosofías que dominarían la computación durante décadas.

2. La operación más sencilla para un procesador

Para entender la diferencia entre CISC y RISC, primero debemos comprender qué es una ISA (Instruction Set Architecture).

Una ISA es el "lenguaje" que habla un procesador. Está compuesta por:
- Tipos de datos que maneja
- Registros y sus tamaños
- Buffers y memoria
- Errores que puede manejar
- Y lo más importante: el conjunto de instrucciones

La analogía del huevo frito

Imagina que programas una receta de huevo frito. Un paso podría ser:

Paso 6: Verterlo con cuidado sobre el aceite caliente

Esta es una instrucción compleja. Pero podríamos descomponerla en instrucciones más simples:

- Paso 6.1: Colocar el huevo partido sobre la sartén
- Paso 6.2: Acercar el huevo a un par de centímetros del aceite
- Paso 6.3: Mover verticalmente el huevo
- Paso 6.4: Verter el contenido hasta vaciar
- Paso 6.5: Retirar el huevo vacío
- Paso 6.6: Desechar la cáscara

Misma tarea, dos enfoques:
- Instrucción única compleja (CISC)
- Múltiples instrucciones simples (RISC)

La elección entre una u otra define la arquitectura del procesador.
https://www.youtube.com/watch?v=zHjPev8AuB8

3. CISC: La búsqueda de lo completo

CISC = Complex Instruction Set Computing

En 1964, IBM lanzó el System/360, el primer procesador CISC comercial. Su filosofía era clara: ofrecer instrucciones poderosas que hicieran mucho trabajo en una sola operación.

Características de CISC:
- Instrucciones complejas: Una sola instrucción puede realizar múltiples operaciones de bajo nivel
- Programas compactos: Menos líneas de código máquina
- Pocos accesos a memoria: Crítico en una época donde la memoria era escasa y lenta
- Hardware complejo: Decodificadores elaborados para interpretar instrucciones variadas

Ejemplo histórico con el huevo:
En CISC, el Paso 6 completo sería una sola instrucción del procesador.

Exponentes históricos:
- IBM System/360 (1964)
- PDP-11, VAX
- Motorola 68000
- Intel 4004, 8086, 80286
- Hoy: Intel Core, AMD Ryzen (x86-64)

Dato clave: Prácticamente cualquier ordenador de sobremesa o portátil desde los años 80 ha utilizado procesadores x86 (CISC).

4. RISC: La simpleza como virtud

RISC = Reduced Instruction Set Computing
A finales de los 70, los científicos de IBM notaron algo sorprendente: los programadores no usaban las instrucciones complejas de CISC. En su lugar, combinaban instrucciones más simples. Esto llevó a John Cocke a diseñar el IBM 801 (1975), considerado el primer procesador RISC de la historia.

Características de RISC:
- Instrucciones simples: Todas se ejecutan en un solo ciclo de reloj (idealmente)
- Muchas instrucciones por programa: El código es más largo
- Más accesos a memoria: Se necesitan más operaciones de carga/almacenamiento
- Hardware simple: Más barato de fabricar, menor consumo energético

Ejemplo histórico con el huevo:
En RISC, el Paso 6 se descompone en los pasos 6.1 a 6.6, cada uno como instrucción separada.

Exponentes históricos:
- IBM 801 (1975)
- MIPS, SPARC, PowerPC
- Hoy: ARM (el estándar mundial en dispositivos móviles)

Dato clave* ARM domina el 99% del mercado de smartphones y tablets, y ahora invade laptops y servidores.

5. Evolución y convergencia: Las líneas se difuminan

Tanto CISC como RISC han evolucionado enormemente desde su creación, adoptando mejoras del rival:

Era	CISC (x86)	RISC (ARM)
1980s	Instrucciones complejas, pocos registros	Instrucciones simples, muchos registros
1990s	Pentium Pro: decodificación interna a micro-ops (RISC)	ARM7: optimización para bajo consumo
2000s	Multi-núcleo, hyperthreading	Cortex-A: SoCs para smartphones
2010s	Integración GPU, eficiencia mejorada	big.LITTLE (núcleos grandes + pequeños)
2020s	Chiplets/tiles, NPUs integrados	Computación de 64 bits, servidores cloud

El secreto de Intel y AMD:

Desde el Pentium Pro (1995), los procesadores x86 no ejecutan x86 directamente. Internamente:

1. Reciben instrucción CISC compleja
2. La descomponen en micro-ops (micro-operaciones tipo RISC)
3. Ejecutan estas micro-ops en núcleos internos eficientes

Resultado: Tu Intel "CISC" es internamente RISC. Mantiene compatibilidad con software antiguo, pero gana eficiencia.

ARM: del RISC puro al "RISC pragmático"

Los ARM modernos (ARMv8, ARMv9) incluyen instrucciones complejas:
- NEON/SVE: Procesamiento vectorial (SIMD)
- Criptografía: Instrucciones dedicadas de seguridad
- big.LITTLE: Núcleos de diferente potencia en el mismo chip

PARTE II: LA BATALLA ACTUAL 2020-2024

6. El cambio de paradigma: Del chip al SoC

La verdadera revolución no es CISC vs RISC, sino cómo se integran en un System on a Chip (SoC).

Característica	PC Tradicional (2000s)	SoC Moderno (2024)
CPU	Chip separado	Integrada en SoC
GPU	Tarjeta dedicada o chip separado	Integrada en SoC
Memoria	Módulos DIMM separados	Integrada o muy cercana (unified memory)
Controladores	Chipset en placa madre	Dentro del SoC
Consumo	65-250W	5-45W
Tamaño	Caja de PC completa	Tamaño de una uña

Tres arquitecturas, tres filosofías (2024)

Característica	Intel Core Ultra (CISC)	Apple Silicon M4 (RISC)	Snapdragon X Elite (RISC)
ISA	x86-64	ARMv9	ARMv9
Fabricación	Intel 4 / TSMC 5nm	TSMC 3nm	TSMC 4nm
Diseño	Tiles/Chiplets	SoC monolítico	SoC monolítico
CPU	6P+8E núcleos	4P+6E núcleos	12 núcleos Oryon
GPU	Intel Arc (tiled)	Apple GPU (unified)	Adreno (unified)
NPU (IA)	11 TOPS	38 TOPS	45 TOPS
Memoria	DDR5 separada	LPDDR5 unified	LPDDR5x unified
Consumo	15-45W	15-30W	12-28W
Sistema	Windows/Linux nativo	macOS/iOS nativo	Windows on ARM (emulación x86)
Precio	$1,200-$2,500	$1,200-$4,000	$900-$1,800

TOPS = Trillions of Operations Per Second

Análisis de cada arquitectura:

Intel (CISC evolucionado):

- Mantiene compatibilidad con 40 años de software x86
- Adopta técnicas "RISC-like": chiplets, eficiencia, NPU integrado
- Desventaja: Arquitectura acumulada, mayor consumo que ARM

AMD (CISC resiliente y flexible):

- Arquitectura chiplet revolucionaria: Separa CCD - Core Complex Die - (núcleos CPU en 4nm) de IOD (controladores en 6nm)
- Ventaja de modularidad: Mismo diseño base escala desde laptops (15W) hasta servidores EPYC (400W+)
- 3D V-Cache: Único en el mercado. Apila caché L3 verticalmente (hasta 128MB extra) para gaming y servidores
- Líder en gaming (Ryzen 7 7800X3D = "rey del gaming" 2024) y servidores cloud (EPYC domina AWS/Azure/Google)
- NPU Ryzen AI (XDNA): Llegó tarde (2023) pero evoluciona rápido (16→50 TOPS)
- Mejor precio-rendimiento que Intel en la mayoría de segmentos; único competidor x86 viable
- Desventaja: Sin unified memory (como Apple/Qualcomm), eficiencia en ultrabooks inferior a ARM

Apple (RISC dominante):

- Control total de hardware y software
- Unified Memory: CPU y GPU comparten RAM física sin copias lentas
- Líder indiscutible en rendimiento por watt

Qualcomm (RISC invadiendo):

- Primera apuesta seria de ARM en laptops Windows
- Mayor NPU del mercado (IA local)
- Desafío: Emulación de software x86 legacy con penalización de rendimiento

https://hardzone.es/noticias/procesadores/intel-amd-cpu-x86-apple-m1-cinebench/

7. El equipo de procesamiento: CPU, GPU y NPU

En los SoCs modernos, no hay un solo "cerebro". Hay tres especialistas:

Componente	Función	Analogía	Tarea típica
CPU	Decisiones complejas, secuenciales	El capitán del equipo	Abrir apps, lógica de negocio, sistema operativo
GPU	Paralelismo masivo de datos	El ejército de obreros	Gráficos, video, cálculos científicos (GPGPU)
NPU	Operaciones de redes neuronales	El especialista en IA	Reconocimiento facial, asistentes de voz, generación de imágenes

GPGPU: La GPU ayuda a la CPU

GPGPU = General-Purpose computing on GPU

La GPU no solo sirve para gráficos. Como tiene miles de núcleos simples, es ideal para:

- Entrenar modelos de IA (antes de que existieran NPUs masivamente)
- Minería de criptomonedas (históricamente)
- Simulaciones científicas (fluidos, moléculas, clima)
- Procesamiento de video (transcodificación)

Regla: La GPU gana cuando la tarea es "haz esto mismo a miles de datos simultáneamente". Pierde cuando hay muchas decisiones "if/then" secuenciales.

8. Por qué esto importa para un ingeniero de sistemas

Escenario 1: Desarrollo móvil
- El 99% de smartphones usan ARM (RISC)
- Optimizar para ARM significa apps que no drenan batería
- Conocer la jerarquía de memoria del SoC mejora el rendimiento

Escenario 2: Cloud y servidores
- AWS Graviton (ARM): 40% mejor precio-rendimiento que x86 para cargas específicas
- Compilar para ARM puede reducir costos operativos significativamente
- Azure y Google Cloud también ofrecen instancias ARM

Escenario 3: Edge computing e IoT
- Raspberry Pi, NVIDIA Jetson, dispositivos médicos: todos ARM
- Un sistema CISC sería inviable por consumo y tamaño
- El NPU permite IA local sin conexión a internet

Escenario 4: Inteligencia Artificial
- Optimizar un modelo para NPU vs GPU vs CPU requiere entender la arquitectura
- Los NPUs solo existen en SoCs modernos (ARM principalmente)
- La eficiencia energética determina si una IA corre en el dispositivo o en la nube

Lo ultimo

https://computerhoy.20minutos.es/tecnologia/cientifico-91-anos-promete-putin-procesador-200-veces-potente-chips-occidentales-1462593

https://www.larazon.es/tecnologia-consumo/tecnologia/los-nuevos-chips-de-ia-de-china-prometen-ser-hasta-100-veces-mas-rapidos-que-las-gpu-de-nvidia_20251221699418921817b41eb64cf295.html

ACTIVIDADES

Actividad 1: Detective de arquitecturas

Identificar CISC vs RISC en dispositivos cotidianos.

Instrucciones

Paso 1 – Individual:

Complete la tabla con sus dispositivos:

Dispositivo	**Procesador exacto**	**¿CISC o RISC?**	**¿SoC?**	**NPU/GPU integrado**
Celular				
Laptop				
Tablet/otro				

Pistas de identificación:

- iPhone: Siempre ARM (Apple A-series o M-series) = RISC
- Android: Snapdragon (ARM), Exynos (ARM), Google Tensor (ARM) = RISC
- Windows: Intel Core/AMD Ryzen = CISC; Snapdragon X = RISC
- Mac: M1/M2/M3/M4 = RISC; Intel antiguo = CISC

Paso 2 - Parejas :
Comparen resultados y respondan:

1. ¿Por qué sus celulares usan RISC y no CISC?
2. ¿Sus laptops usan la misma arquitectura que sus celulares? ¿Por qué sí/no?
3. ¿Qué sacrificios implica usar ARM en una laptop Windows (Snapdragon) versus Intel?

Actividad 2: Guerra de arquitecturas

Continuamos con los 4 grupos, cada uno defiende una arquitectura:

Grupo	**Arquitectura**	**Característica "killer"**
A	Intel Core Ultra	Compatibilidad x86 universal, ecosistema maduro
B	Apple Silicon M4	Eficiencia extrema, unified memory, control total HW/SW
C	Snapdragon X Elite	Mayor NPU del mercado, eficiencia en Windows
D	AMD Ryzen 8000	Rendimiento bruto, gaming, precio competitivo

Tareas por grupo:

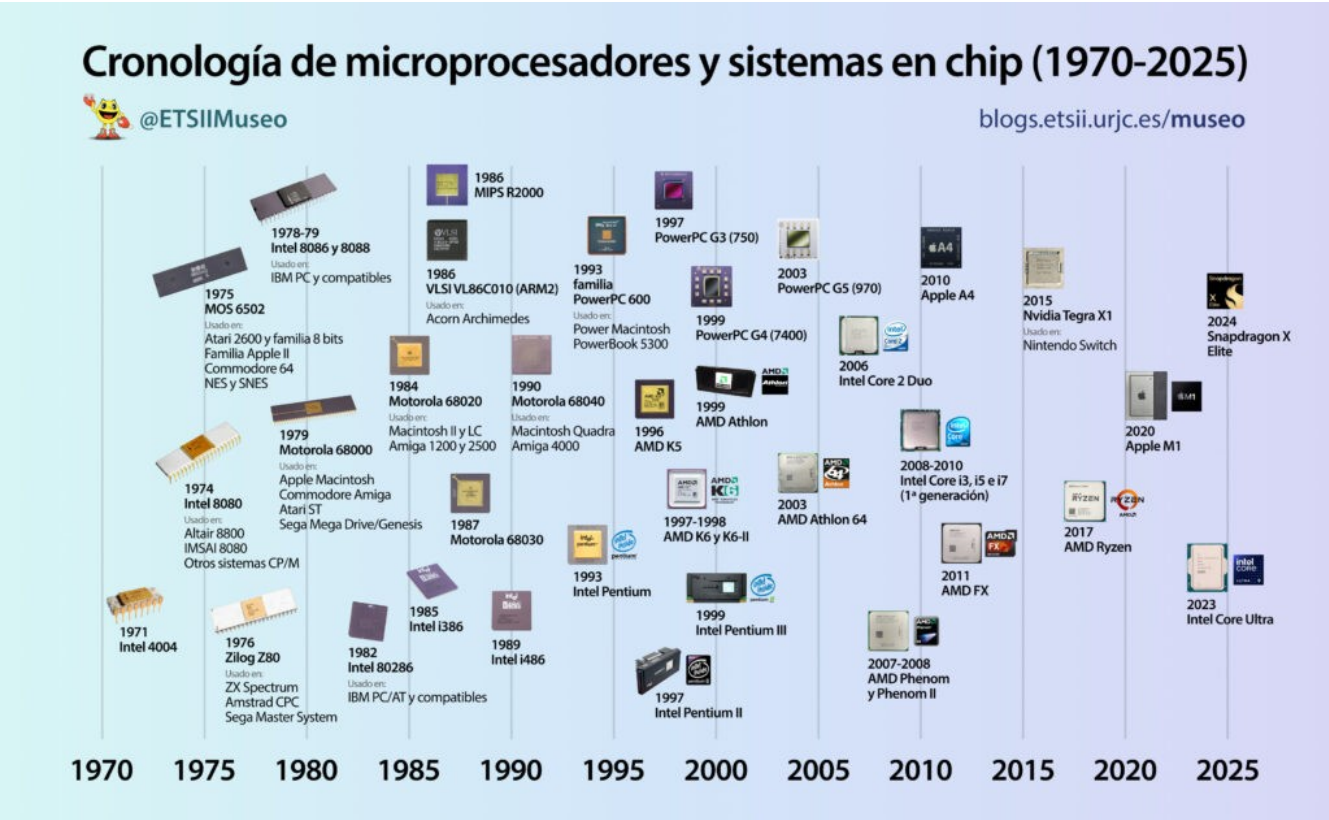
1. Investigar datos técnicos reales (usar celulares)
2. Identificar: ISA (CISC/RISC), tecnología (nm), consumo (W), TOPS de NPU
3. Preparar argumento para 3 escenarios:
    1. Un data center (miles de servidores)
    2. Un estudiante de ingeniería (laptop diaria)
    3. Un dispositivo IoT médico portátil

Debate:
Cada grupo expone 2 minutos. Luego, preguntas del docente:

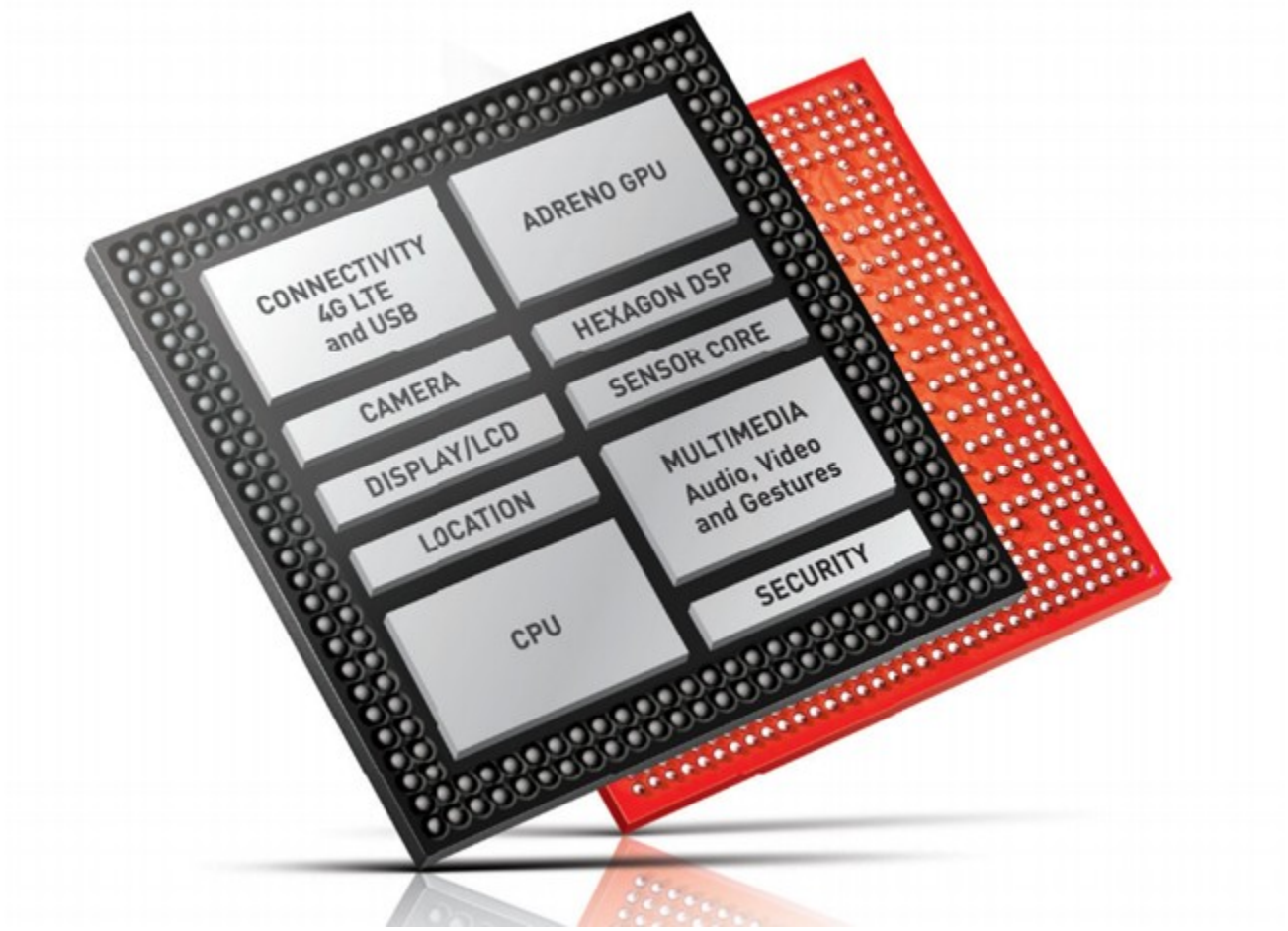
1. ¿Qué arquitectura elegirían para su proyecto de grado y por qué?
2. ¿Es la compatibilidad con software antiguo más importante que la eficiencia energética?

GLOSARIO TÉCNICO

Término	Definición	Contexto CISC vs RISC
ISA	Instruction Set Architecture. El "lenguaje máquina" del procesador.	x86 = CISC; ARM = RISC
Micro-op	Micro-operación. Instrucción RISC interna resultante de decodificar CISC.	Intel/AMD usan internamente desde 1995
SoC	System on a Chip. Integración de CPU, GPU, NPU, controladores en un chip.	Dominante en ARM; adoptado por Intel/AMD
NPU	Neural Processing Unit. Procesador especializado en operaciones de IA.	Presente en todos los SoCs 2024
TOPS	Trillions of Operations Per Second. Medida de rendimiento en IA.	Snapdragon X Elite: 45 TOPS; Apple M4: 38 TOPS
TDP	Thermal Design Power. Consumo máximo de energía bajo carga.	RISC suele tener menor TDP que CISC
big.LITTLE	Arquitectura híbrida con núcleos potentes (big) y eficientes (LITTLE).	Inventado por ARM; adoptado por Intel (P-cores/E-cores)
Chiplets/Tiles	Diseño con diferentes partes del procesador en chips separados interconectados.	Intel Meteor Lake; AMD Zen
Unified Memory	RAM compartida físicamente entre CPU y GPU sin copias entre memorias.	Característica clave de SoCs ARM
GPGPU	General-Purpose computing on GPU. Usar GPU para cálculos no gráficos.	IA, simulaciones científicas, minería
Emulación	Ejecutar software de una arquitectura en otra con traducción en tiempo real.	Windows on ARM emula x86 con penalización



https://blogs.etsii.urjc.es/museo/historia/microprocesadores-cerebro-informatica-moderna/



REFERENCIAS BIBLIOGRÁFICAS

Historia y fundamentos
- Stallings, W. (2006). *Organización y Arquitectura de Computadores* (7ª ed.). Pearson Prentice Hall.
- Tanenbaum, A. S. *Organización de Computadoras. Un Enfoque Estructurado* (4ª ed.). Pearson Educación.
- Hennessy, J. L., & Patterson, D. A. *Arquitectura de Computadores: Un enfoque cuantitativo*. McGraw-Hill.

Actualización 2020-2024
- Apple Inc. (2024). *Apple Platform Security Guide* – Arquitectura Apple Silicon.
- Qualcomm Technologies. (2023-2024). *Snapdragon X Elite Technical Documentation*.
- Intel Corporation. (2024). *Intel Core Ultra Processor Architecture* (Meteor Lake/Arrow Lake).
- AnandTech, Tom's Hardware. Reviews comparativas 2023-2024: Intel Core Ultra vs Apple M3 vs Snapdragon X Elite.

La arquitectura de computadores no es historia. Cada vez que optimizan una app para que no consuma batería, deciden si su software corre en la nube o localmente, o eligen entre una laptop Intel o ARM, están aplicando estos conceptos. La "batalla" CISC vs RISC continúa, ahora en el contexto de la IA y la sostenibilidad energética.