# I. Nivel ISA: Definición, Importancia y Fundamentos

## 1. El Nivel ISA como Interfaz

El **Nivel ISA (Instruction Set Architecture)**, a menudo referido simplemente como la **arquitectura de la computadora**, constituye el nivel fundamental y original dentro de la máquina multinivel contemporánea. Es la **interfaz** crítica que logra la integración entre el **mundo del software** (compiladores, sistemas operativos) y el **mundo del hardware** (microarquitectura).

**Definición:** El ISA es el **lenguaje de máquina** que el hardware y los compiladores deben entender para comunicarse. Desde la perspectiva del programador que codifica en lenguaje máquina, el ISA define cómo se comporta la máquina y qué recursos posee.

## 2. Propiedades de un ISA Exitoso

Un diseño de arquitectura de conjunto de instrucciones debe considerar aspectos técnicos y económicos para asegurar su longevidad y éxito:

1. **Implementabilidad y Eficiencia:** El ISA debe poder implementarse de forma eficiente y a bajo costo en las tecnologías actuales y futuras. Su diseño debe perdurar por múltiples generaciones de procesadores, siendo económicamente viable.
2. **Traducción Clara para el Compilador:** Debe ofrecer una correspondencia directa y sencilla para el código generado por los compiladores de lenguajes de alto nivel.
3. **Compatibilidad en Reversa (Backward Compatibility):** Es una propiedad económica fundamental. Un ISA debe mantener la capacidad de ejecutar el software diseñado para sus arquitecturas predecesoras. Esto protege la inversión de los clientes en software.

## 3. Modos de Operación del Procesador

Las arquitecturas modernas distinguen al menos dos modos de operación para gestionar la seguridad y el acceso a recursos críticos:

* **Modo Kernel (Modo Privilegiado):** Este es el modo exclusivo para la ejecución del **Sistema Operativo (SO)**. En este modo, el procesador tiene permitido ejecutar **TODAS** las instrucciones disponibles en el ISA, incluidas aquellas que son peligrosas o que controlan el estado global de la máquina (instrucciones de E/S, gestión de memoria).
* **Modo Usuario (Modo Restringido):** Es el modo en el que se ejecutan las **aplicaciones de usuario**. El acceso a ciertas instrucciones de control crítico está **restringido** para evitar que una aplicación comprometa la estabilidad o seguridad del sistema.

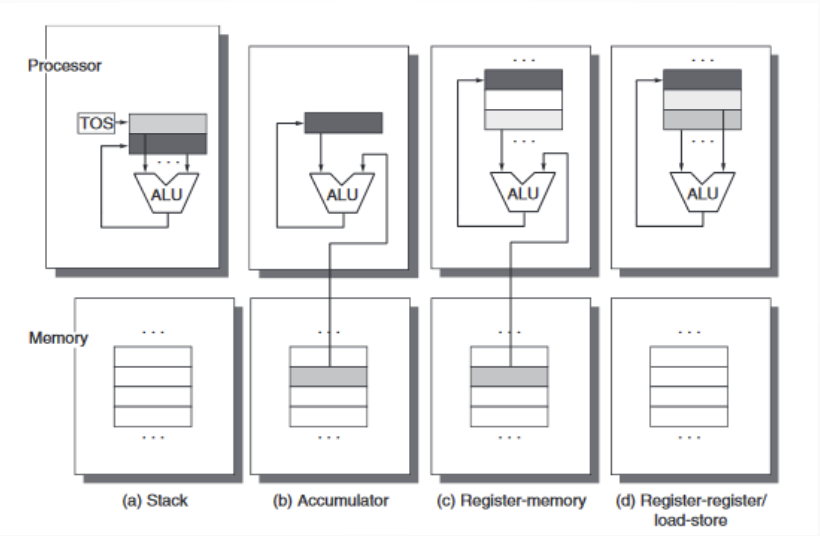
# II. Las Siete Dimensiones del ISA

El diseño y especificación de un ISA se articula a través de la definición de siete dimensiones clave.

## 1. Clase del ISA (Taxonomía)

Esta dimensión define el factor más influyente: **dónde se encuentran los operandos** que las instrucciones manipularán. Se clasifica según el almacenamiento interno principal del procesador:

| Clase de ISA | Almacenamiento Interno | Operandos | Características Clave |
| --- | --- | --- | --- |
| **Pila (Stack)** | Memoria LIFO (Last-In, First-Out). | **Implícitos** (siempre los dos valores en el tope de la pila). | Las instrucciones de ALU no contienen operandos de dirección. El resultado se apila nuevamente. |
| **Acumulador** | Un registro especial (el Acumulador). | Un operando es **siempre el Acumulador**; el otro puede ser una dirección de memoria. | Usado en las primeras computadoras. La gestión de variables es ineficiente. |
| **Registros de Propósito General (GPR)** | Conjunto de registros internos (la solución moderna). | **Explícitos** (se especifican los registros o direcciones de memoria en la instrucción). | Permite al compilador optimizar el uso de registros para variables, expresiones y paso de parámetros. |



### Arquitecturas GPR

El modelo de Arquitectura de Conjunto de Instrucciones (ISA) basado en **Registros de Propósito General (GPR)** es dominante hoy en día por su eficiencia, rendimiento y la capacidad de optimización que ofrece a los compiladores. Las siguientes son las razones concisas y clave de su uso:

### 1. Velocidad Superior (Jerarquía de Memoria)

* **Rapidez Extrema del Hardware:** Los **Registros** son el nivel de almacenamiento más rápido dentro de la jerarquía de memoria de la CPU, construidos con la tecnología más veloz (generalmente *flip-flops*).
* **Acceso en Ciclo Único:** El acceso a un registro se realiza típicamente en **un solo ciclo de reloj** (o menos), mientras que un acceso a la **Memoria Principal (DRAM) puede tomar decenas o cientos de ciclos si la información no está en la caché.**

### 2. Optimización y Eficiencia del Compilador

* **Operandos Explícitos y Flexibles:** A diferencia de las arquitecturas de Pila (operandos implícitos) o Acumulador (un operando fijo), el modelo GPR permite al compilador elegir libremente qué registros usar y cuándo usarlos.
* **Planificación de Recursos:** El compilador puede asignar variables y resultados intermedios a registros específicos (*Register Allocation*). Esto maximiza la permanencia de los datos en el medio más rápido de la CPU y minimiza el tráfico innecesario de LOAD y STORE hacia y desde la memoria.

### 3. Soporte Directo para Variables y Expresiones

* **Almacenamiento de Variables:** Los registros se utilizan como la memoria de trabajo de la CPU, siendo ideales para sostener las variables locales y constantes más utilizadas, especialmente dentro de los bucles (*loops*), donde la velocidad de acceso es vital.

La clase GPR (General-Purpose Register) se subdivide según cuántos de sus operandos pueden ser direcciones de memoria:

| Tipo GPR | Formato (Memoria, Registros) | Descripción | Ventajas | Desventajas |
| --- | --- | --- | --- | --- |
| **Registro-Registro (Load-Store)** | (0, 3) | **RISC** (ej. ARM). Solo las instrucciones LOAD y STORE acceden a memoria. Las operaciones de ALU solo usan registros. | Instrucciones simples, de longitud fija y tiempos de ejecución (CPI) similares. Modelo de código simple. | Los programas son compilados a un **mayor total de instrucciones**; menor densidad de código, programas más grandes. |
| **Registro-Memoria** | (1, 2) | **CISC** (ej. IA-32). Permite que una instrucción aritmética lea un operando directamente desde la memoria. | **Mayor densidad de código** (menos instrucciones totales). El dato es accedido sin un LOAD previo. El formato de instrucción tiende a ser fácil de decodificar | Un operando se **destruye** (el resultado reemplaza al operando destino). La variación en el CPI (Ciclos Por Instrucción) es alta, dependiendo de la ubicación del operando. |
| **Memoria-Memoria** | (2, 2) o (3, 3) | Permite que dos o tres operandos provengan directamente de la memoria. Arquitectura obsoleta. | Código muy compacto. No requiere registros para almacenamiento temporal. | **Mucha variación en el CPI**. El alto número de accesos a memoria por instrucción causa **cuellos de botella**. |

## 2. El Modelo de Memoria

Define cómo se organiza y accede a la memoria desde el punto de vista del ISA. Es decir, define como una dirección de memoria será interpretada y como es especificada.

* **Tamaño de Celda (Byte):** La unidad más pequeña y fundamental de memoria que puede ser direccionada. Hoy en día, esta unidad es universalmente **8 bits (1 Byte)**. Las palabras de datos que manipulan los procesadores modernos suelen ser de 4, 8 o más bytes.
* **Endianness (Ordenamiento de Bytes):** Define el orden en que los bytes de una palabra (ej., un entero de 32 bits) se almacenan en la memoria.
  + **Little Endian:** La dirección de memoria inicial contiene el **byte menos significativo** de la palabra. (Ejemplo: Intel Core i7).
  + **Big Endian:** La dirección de memoria inicial contiene el **byte más significativo** de la palabra.
  + **Bi-endianness:** La arquitectura puede ser configurada (generalmente al inicio) para operar en modo *big-endian* o *little-endian* (Ejemplo: ARM v7).
* **Alineamiento de Memoria:** Es la restricción por la cual los accesos a datos que son más grandes que un byte (por ejemplo, una palabra de 4 bytes) deben comenzar en una dirección que sea un **múltiplo de su tamaño**. La alineación simplifica el diseño del *hardware* y mejora el rendimiento. Los accesos **desalineados** pueden no estar permitidos, requerir múltiples referencias alineadas o degradar severamente el rendimiento.

## 3. Modos de Direccionamiento

Especifican la forma en que se localiza la dirección de un operando en la instrucción.

| Modo | Descripción | Uso Típico |
| --- | --- | --- |
| **Registro** | El operando se encuentra directamente en un registro del procesador. La instrucción sólo necesita especificar el número de registro. | Variables locales muy usadas. |
| **Inmediato** | El operando es un valor constante que está codificado **dentro del campo de operando** de la instrucción. | Constantes, valores pequeños. |
| **Directo** | El campo de dirección de la instrucción contiene la **dirección de memoria física** exacta donde se encuentra el operando. | Variables globales. |
| **Registro Indirecto** | El campo de dirección especifica un **registro** que a su vez contiene la **dirección de memoria** del operando. | Punteros. |
| **Basado/Desplazamiento** | La dirección del operando se calcula sumando el contenido de un **Registro Base** con un valor de **Desplazamiento** (Offset) constante incluido en la instrucción. | Acceso a estructuras de datos o elementos de *arrays* (arreglo[i]). |
| **Indexado (Based-Indexed)** | Similar al basado, pero la dirección final se calcula usando **dos registros** (un registro base y un registro índice) o un registro base, un índice y un desplazamiento. | Operaciones que involucran matrices (arrays bidimensionales). |
| **Pila (Stack)** | Es un modo **implícito**, exclusivo de las arquitecturas de pila. Los operandos son siempre los **dos valores en el tope** de la pila (LIFO). El resultado se "empuja" de vuelta a la pila. | Arquitecturas de pila. |

## 4. Tipos y Tamaños de Datos

Define las representaciones de datos que el hardware del ISA soporta directamente, lo que permite operaciones de alto rendimiento. Hay formas infinitas de tratar todos tipos de datos a nivel software, incluyendo tipos de datos no soportados por el hardware.

| Categoría | Tipo de Dato | Representación Común | Propósito |
| --- | --- | --- | --- |
| **Enteros** | Integers | 8, 16, 32, 64 bits. **Signados** (en Complemento a Dos) o **No-signados**. | Números enteros. |
| **Fraccionados** | Floating-Point | 32 (float), 64 (double), 128 (quadruple) bits. Generalmente bajo el estándar **IEEE 754**. | Números reales con alta precisión y rango. |
| **Decimal** | BCD (Binary-Coded Decimal) | Cada dígito decimal se codifica en 4 bits (Packed) u 8 bits (Unpacked). | Usado para aritmética exacta, crucial en aplicaciones financieras donde la precisión es obligatoria. |
| **Caracteres** | Character | 7 bits (ASCII) o 16 bits (Unicode). | Representación de texto. |
| **Cadena de caracteres** | String | Variable. Contiene un símbolo especial para indicar el final de la cadena. | Representación de texto |
| **Punteros** | Pointer | 32 o 64 bits (dependiendo del bus de direccionamiento). | Direcciones de memoria. |
| **Booleano** | Boolean | Puede usar 1 bit o la palabra completa (ej. 8, 32 o 64 bits) para representar Falso (0) o Verdadero (no-cero). | Valores lógicos. En arreglos lógicos se usa el *bit map*. |

## 5. Tipos de Operaciones

Definen las operaciones que el ISA expone al programador.

* **Aritméticas y Lógicas (ALU):** Las operaciones básicas sobre enteros: Suma (ADD), Resta (SUB), Multiplicación (MUL), División (DIV), operaciones lógicas (AND, OR, NOT, XOR), y operaciones de **desplazamiento/rotación** (SHIFT, ROTATE).

Nota: Aparte de el concepto lógico que llevan el AND y el OR, estas dos operaciones tienen unos usos de suma importancia:

* Masking (AND): se usa cuando querés extraer una porción de bits de una palabra.
* Packing (OR): se usa cuando querés empaquetar una porción de bits a una palabra.
* **Transferencia de Datos:** Mueven (duplican) datos entre los registros y la memoria.
  + **LOAD:** Transfiere datos de **Memoria a Registro**.
  + **STORE:** Transfiere datos de **Registro a Memoria**.
  + **MOVE:** Transfiere datos entre **Registros** o desde un **Inmediato a Registro**.
* **Control de Flujo:** Alteran el orden secuencial de las instrucciones (cambian el Contador de Programa - PC).
* **Punto Flotante:** Operaciones especializadas (suma, resta, multiplicación, etc.) definidas para los números fraccionados (IEEE 754).
* **Sistema:** Instrucciones privilegiadas utilizadas por el sistema operativo, como las de gestión de memoria virtual o llamadas al sistema (*System Calls*).
* **Especializadas:** Incluyen operaciones para datos decimales (BCD), manipulación de cadenas (*strings*) y gráficos (operaciones de píxeles y vértices).

**Instrucciones de entrada/salida**

Hay tres esquemas de instrucciones para tratar entradas y salidas desde dispositivos del sistema.

* **E/S programada con espera activa**: el procesador contiene instrucciones que inician una comunicación con un dispositivo y quedan en un bucle esperando la respuesta. Usada en sistemas que tienen que reaccionar inmediatamente a un cambio externo (real-time emebeded systems).
* **E/S controlada por interrupciones**: las instrucciones inician el dispositivo de E/S y le dicen que genere una interrupción cuando esté listo. Aunque es considerado un mejor esquema que el anterior, si tiene sus problemas, dispositivos que generan muchas interrupciones puede ser costoso para el procesador.
* **E/S DMA (direct memory access – acceso directo a memoria)**: este esquema es parecido al de E/S programada con espera activa, pero pateamos la responsabilidad a otro lado. En vez que el procesador se haga cargo del dispositivo, hay un controlador llamado DMA que se hace cargo. El controlador tienen 4 registros.

## 6. Tipos de Control de Flujo

Mecanismos que permiten al procesador desviarse de la ejecución secuencial:

| Tipo de Control | Descripción Detallada |
| --- | --- |
| **Saltos (Branch/Jump)** | **Jump** es un salto incondicional, mientras que **Branch** es un salto condicional. El *branch* se basa en el estado de los **Códigos de Condición (Flags)** del procesador para decidir si se salta a una nueva dirección o se continúa la secuencia.  Opciones de ramificaciones condicionales: |
| **Llamadas y Retornos a Procedimientos** | Son saltos especializados para invocar funciones/subrutinas. Antes de saltar, el sistema debe **guardar el estado de la máquina** (principalmente el valor actual del **PC**) en la pila. El Return recupera el estado guardado para volver al punto de interrupción.  Hay dos convenciones para guardar valores de registros:   * Dentro del procedimiento que hace el llamado (caller saving):el procedimiento que va a realizar la llamada es el responsable de guardar los registros que necesita conservar en la pila cuyo valor le será útil *después* de que la función llamada regrese. * Dentro del procedimiento llamado (callee saving):, el procedimiento que es llamado es el responsable de guardar los registros que va a utilizar.   Cuando una instrucción llama a un procedimiento, la dirección de retorno es guardada y el llamado usa la instrucción de retorno para transferir de vuelta el control. En una corutina, cada procedimiento llama al otro y continua una instrucción adelante desde la última llamada. Usa instrucciones diferentes a las típicas CALL y RETURN. |
| **Interrupciones y Traps** | Son eventos que fuerzan la detención inmediata del programa en curso para atender una rutina de servicio de interrupción (ISR). Las interrupciones son causadas por eventos **externos** (E/S, temporizador), y los *traps* son causados por eventos **internos** provocados por un programa en ejecución (errores de división por cero, violaciones de acceso).  Las trampas funcionan de forma síncrona con tu programa. En términos de reproducibilidad la trampa siempre ocurrirá en el mismo lugar del programa. En cambio, las interrupciones son consideradas asíncronas. |
| **Transparencia de Interrupciones** | Una interrupción es **transparente** si el sistema puede guardar por completo el estado del programa (todos los registros, PC) antes de ejecutar la ISR, y luego **restaurarlo** con precisión para que el programa interrumpido continúe su ejecución sin "darse cuenta" de la interrupción. |
| **Jerarquía de Prioridades** | En sistemas con múltiples dispositivos, las interrupciones se gestionan por prioridad. Si hay una interrupción activa, otra interrupción solo será procesada (interrumpiendo a la primera) si el dispositivo es de **mayor prioridad**. Esto evita que dispositivos críticos, sensibles al tiempo, esperen demasiado. |

El control de flujo (saltos y ramificaciones) necesita saber a qué dirección ir después. El ISA usa dos métodos para calcular esta dirección de destino:

### 1. Relativo al PC (PC-Relative) 📏

* **Mecanismo:** La instrucción no almacena la dirección absoluta del destino, sino la **distancia (desplazamiento)** que hay que sumar al Contador de Programa (PC) actual. Se usa para **saltos locales y fijos** (bucles, *if-else*).
* **Ventaja clave:** Permite el **código independiente posicional** (*position independence*). El código funciona aunque se cargue en cualquier parte de la memoria.

### 2. Indirecto (Vía Registros o Pila) 🔄

* **Mecanismo:** La dirección de destino se almacena en una ubicación que puede cambiar:
  + **Registros:** Usados para **punteros a funciones** o **tablas de salto**.
  + **Pila:** Usada exclusivamente para la **dirección de retorno** de las funciones (el RETURN).

Se usa para **saltos dinámicos**, donde el destino no se conoce hasta que el programa se está ejecutando.

* **Ventaja clave:** Permite implementar funciones complejas del lenguaje de alto nivel, como las llamadas a funciones y los saltos basados en el contenido de una variable.

## 7. Formato de Instrucción

Define la estructura binaria (la codificación) de las instrucciones que el procesador debe decodificar y ejecutar.

* **Componentes:**
  + **Opcode (Código de Operación):** El código que especifica la operación a realizar (ej. ADD, LOAD).
  + **Operando(s):** Información adicional que indica la ubicación de los datos a manipular (registros o direcciones de memoria).
* **Ortogonalidad:** Es la propiedad por la cual **cada operación** de la ISA soporta **cada modo de direccionamiento**. Las arquitecturas *Load-Store* modernas, como ARM, no son ortogonales, ya que LOAD/STORE son las únicas que pueden usar modos de direccionamiento de memoria.
* **Longitud de Instrucción:**
  + **Longitud Fija (RISC):** Todas las instrucciones tienen el mismo número de bits (ej. 32 bits). Esto simplifica el *hardware* de decodificación y permite que el procesador sepa dónde comienza la siguiente instrucción, facilitando el paralelismo.
  + **Longitud Variable (CISC):** Las instrucciones pueden tener diferentes números de bytes. Esto permite una mayor densidad de código (instrucciones muy compactas) a costa de un *hardware* de decodificación mucho más complejo.
* **VLIW (Very Long Instruction Word):** Un formato especial donde el **compilador** (o un pre-procesador) es responsable de agrupar en una sola instrucción muy larga **múltiples operaciones** que pueden ejecutarse en paralelo. Esto **transfiere la complejidad de la planificación del *hardware* al *software***, resultando en una microarquitectura más simple y económica (ej. Transmeta Crusoe, que emulaba x86 con un VLIW).

## 1. Intel Core i7 (ISA IA-32 / x86-64) - Arquitectura CISC

El Core i7 implementa la ISA **IA-32** (32 bits, desde el 80386) y su extensión de 64 bits, **x86-64**.

| Dimensión | Características Detalladas |
| --- | --- |
| **Clase de ISA** | **CISC (Complex Instruction Set Computer)**. Utiliza la clase **Registro-Memoria**, permitiendo que muchas operaciones de ALU accedan directamente a operandos en la memoria. |
| **Compatibilidad** | Mantiene una estricta **compatibilidad en reversa** hasta la arquitectura original de 16 bits (8086/8088) de los años 70, lo que impone una carga de diseño heredada. |
| **Modos de Operación** | **Protected Mode:** El modo operativo moderno con 4 niveles de privilegio (Nivel 0 para Kernel, Nivel 1 y 2 Device Service ,Nivel 3 para Usuario). **Real Mode** (funciona como un 8086) y **Virtual 8086 Mode** (permite que un SO moderno ejecute programas de 16 bits). |
| **Modelo de Memoria** | Utiliza un modelo de memoria **segmentado** heredado con palabras de 32 bits. Almacena los datos en formato **Little Endian** (el byte menos significativo va en la dirección más baja).  El procesador no trabaja con direcciones de memoria física directamente. En su lugar, utiliza un sistema de dos partes llamado **Direccionamiento Segmentado** o **Direcciones Virtuales de 48 bits**:   * **Segmento (16 bits):** Funciona como una clave o "punto de partida" grande. * **Offset (Desplazamiento de 32 bits):** Funciona como la distancia desde ese punto de partida.   Esta dirección virtual de 48 bits **no es la dirección real** en el chip de memoria. La CPU debe pasarla a un componente llamado **MMU (Unidad de Gestión de Memoria)**, que actúa como un traductor y la convierte en una dirección de 36 bits. |
| **Formato de Instrucción** | **Irregular, complejo y de longitud variable** (hasta 15 bytes).    Las instrucciones tienen **prefijos (PREFIX)** opcionales que modifican su comportamiento: REP (para repetición de instrucciones de *string*), LOCK (para reservar el bus y asegurar atomicidad), o prefijos para forzar el modo de operación (8, 16, o 32 bits).  El campo MODE define el modo de direccionamiento. Estos 8 bits están divididos entres grupos. A veces los primeros 3 bits (R/M) son utilizados como una extensión para un opcode de 1 byte, permitiendo uno de 11 bits.  REG siempre especifica un registro, mientras que la combinación de MOD y R/M especifica el otro operando en combinación con el modo de direccionamiento.  ● MOD 00 indica Register Indirect y Direct  ● MOD 01 y 10 indican Displacement.  ● MOD 11 indica Register, de 32 bits para instrucciones de palabras y 8 bits para instrucciones de bytes.  A veces siguiente al MODE, esta SIB. Este especifica un factor de escala (S), y dos registros (I y B). Cuando presente, la dirección efectiva es computada con lo siguiente:  ● S indica un factor de 1, 2, 4 o 8  ● El registro indice (I) es multiplicado por el factor de S.  ● El resultado es sumado al registro base (B).  ● Y finalmente sumándolo, si es necesario, a un desplazamiento de 8 bits o 32 bits. |
| **Operaciones** | Las operaciones diádicas (dos operandos) reemplazan uno de los operandos (destino), lo que es característico del diseño **Registro-Memoria**. |
| **Tipos de Datos** | El IA-32 del Core i7 soporta: Enteros signados (codificado en complemento-a-dos),Enteros no-signados, BDC y punto flotante (codificado en IEEE 754). También soporta caracteres y cadenas de caracteres. |
| **Modos de direccionamiento** | Immediate, Direct, Register, Register Indirect, Displacement, Indexed-Base |

## 2. ARM v7 (OMAP4430) - Arquitectura RISC

El procesador ARM Cortex A9, usado en el SoC OMAP4430, implementa la arquitectura **ARM v7**.

| Dimensión | Características Detalladas |
| --- | --- |
| **Clase de ISA** | **RISC (Reduced Instruction Set Computer)**. Utiliza la clase **Registro-Registro (Load-Store)**. Esto significa que las operaciones aritméticas y lógicas sólo operan sobre registros, y solo las instrucciones LOAD/STORE interactúan con la memoria. |
| **Modelo de Memoria** | **Simple:** Un solo espacio de direccionamiento de 32 bits, con un límite máximo de 4 GB (problema que se abordó usando memoria FLASH como principal). Es **Bi-endianness** (configurable para Little o Big Endian), aunque la memoria del sistema es fija en Little Endian. **Memoria Alineada** requerida debido a las instrucciones de longitud fija. |
| **Formato de Instrucción** | **Limpio y de longitud fija** (32 bits). También incluye el **Thumb ISA**, un conjunto de instrucciones de **16 bits** que ofrecen **mayor densidad de código** (programas más pequeños) con algunas restricciones operativas. Los diseñadores hicieron todo lo posible para utilizar cada bit disponible. Por está razón, cada tipo de instrucción tiene un formato específico.  Se destaca el campo COND, presente en todas las instrucciones. Este se llama el campo de condición y define si la instrucción se va a completar o no, dependiendo de las condiciones del sistema. |
| **Operaciones** | Las instrucciones aritméticas vienen en **dos formas**: una que **modifica** los códigos de condición (flags) y otra que **no los modifica**. Esta dualidad permite al compilador una **mayor optimización** al reordenar las instrucciones sin afectar el flujo condicional. Las booleanas son análogas a las aritméticas.  Las instrucciones de LOAD/STORE vienen en solo tres sabores, para leer/escribir 1, 2, o 4 Bytes.  Las instrucciones de desplazamiento son las mínimas, un shift a la izquierda (lógico) y dos a las  derecha (lógico y aritmético). Una sola instrucción de desplazamiento circular.  Instrucciones de control de flujo incluyen una instrucción que salto condicional que maneja  todas las condiciones, y dos para llamada de procedimientos. |
| **Modos de direccionamiento** | Register, Immediate, Displacement, Register Indirect, PC-Relative. Los 3 últimos para instrucciones de tipo LOAD/STORE y saltos. |
| **Tipo de datos** | Soporta enteros signados, enteros no-signados y punto flotante. No tienen soporte de hardware para caracteres o strings. |