Estructura de Computadores

Segundo curso

Turno de tarde

@Iván Castilla Rodríguez 2024-2025

Tema 2. El funcionamiento de un computador. Instrucciones y juego de instrucciones

Módulo I. Estructura interna del procesador y buses de interconexión

@Iván Castilla Rodríguez 2024-2025

Ciclo de Instrucción de un Computador Von Neumann

Descripción interna de un Computador/Procesador

- Elementos internos a la CPU
- Función en el Ciclo de Instrucción

Otras Arquitecturas: Harvard

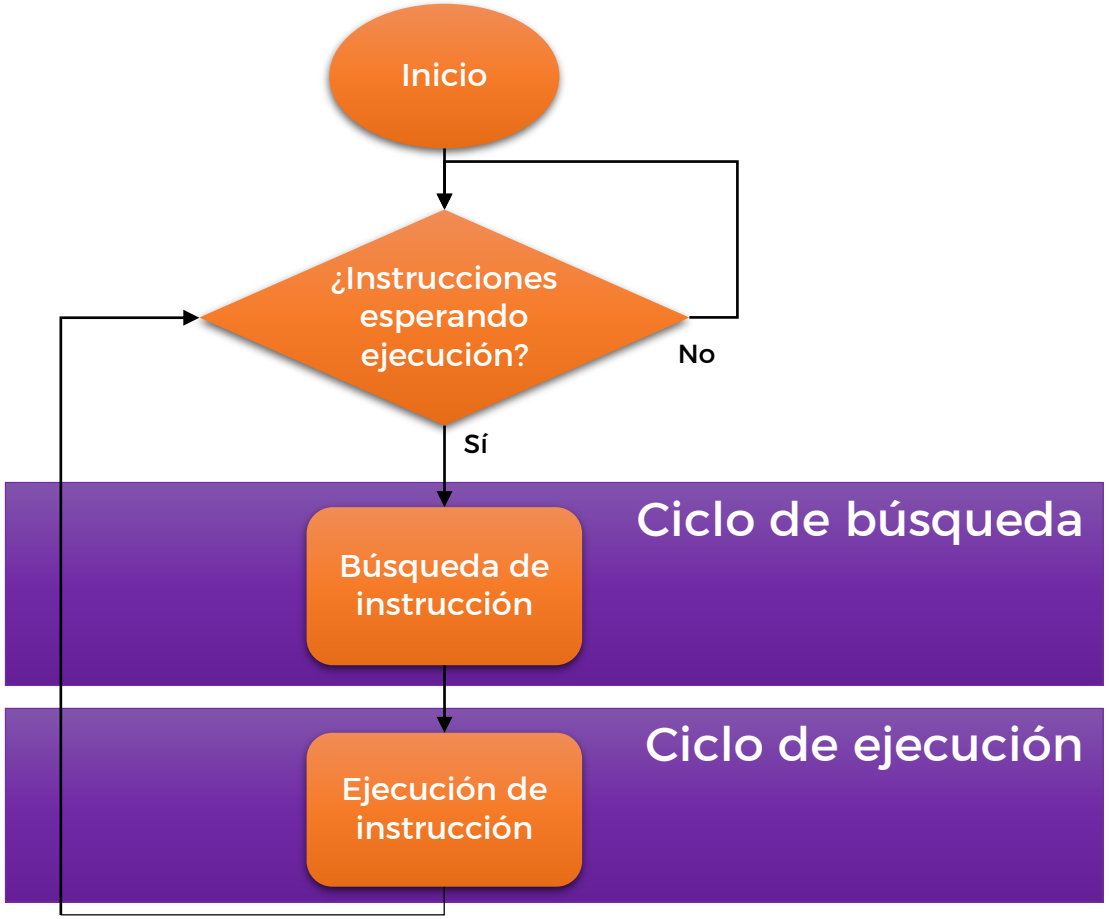
Instrucciones

- Definición
- Lenguaje máquina y lenguaje ensamblador
- Tipos Generales
- Características
- Estructura y Codificación
- Modos de direccionamiento

Prestaciones de un Computador

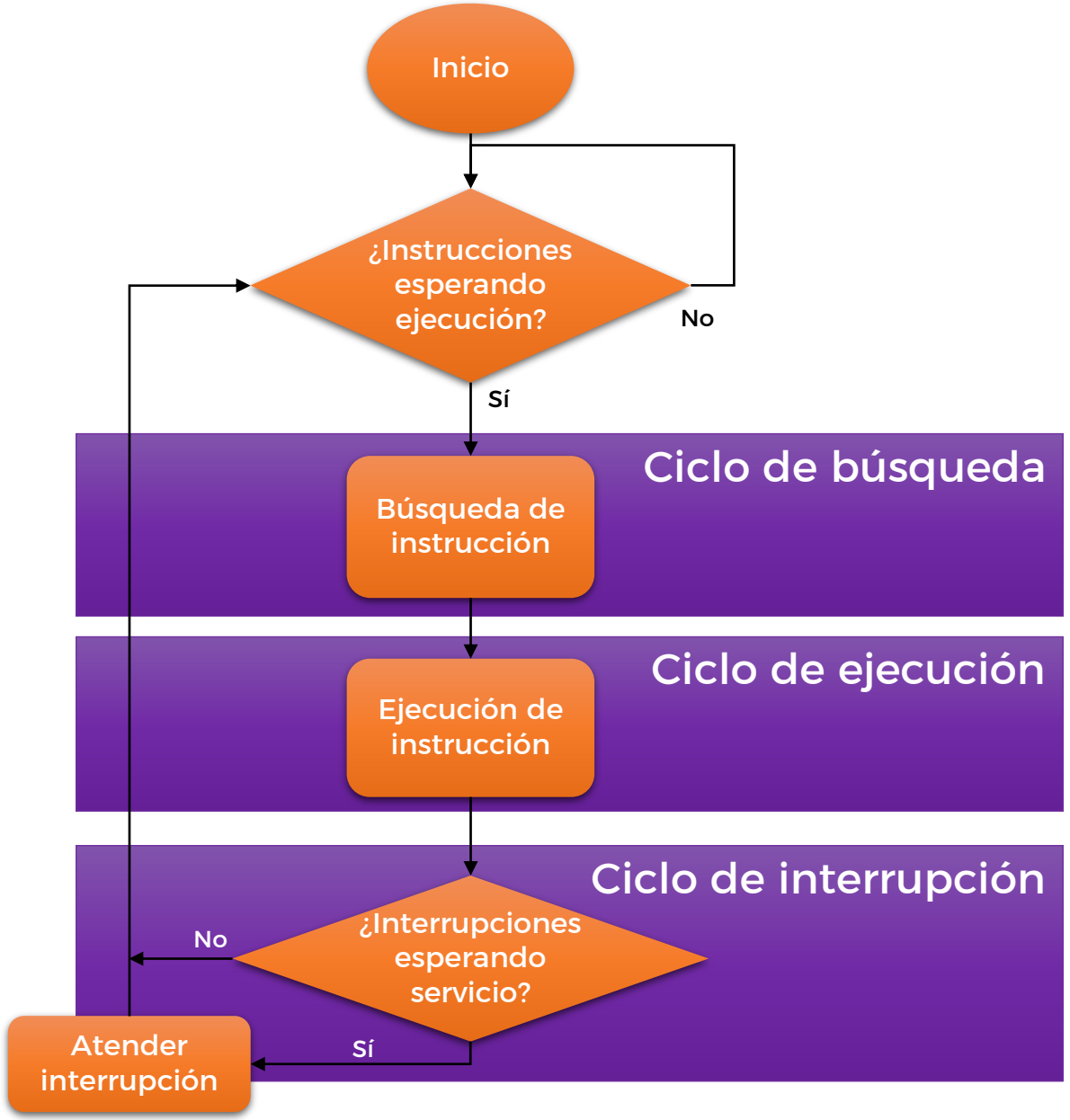
- Rendimiento de un Procesador
- Introducción al Paralelismo

Ciclo de instrucción



¿Es esto todo?

Ciclo de instrucción: interrupciones

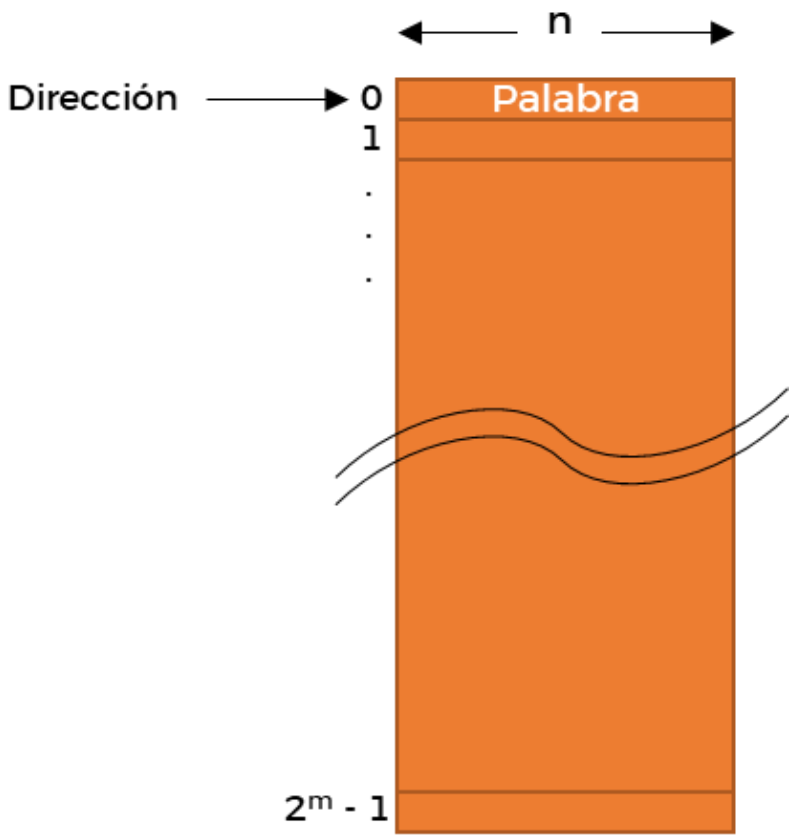


Pregunta de examen

Una interrupción detiene

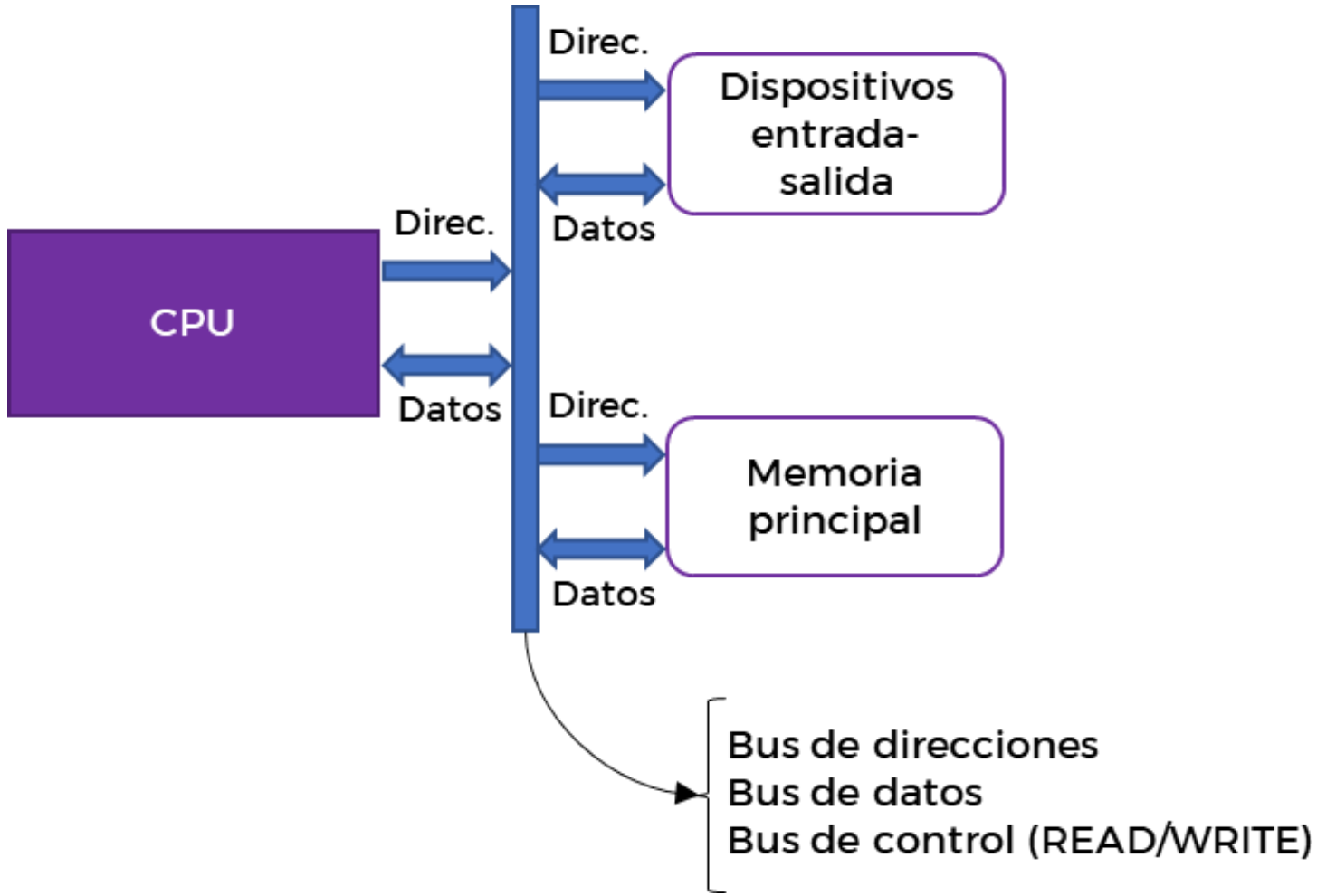
a) La instrucción que se está ejecutando
b) El flujo normal de ejecución de instrucciones
c) El funcionamiento de la máquina

Descripción interna del procesador: la memoria principal

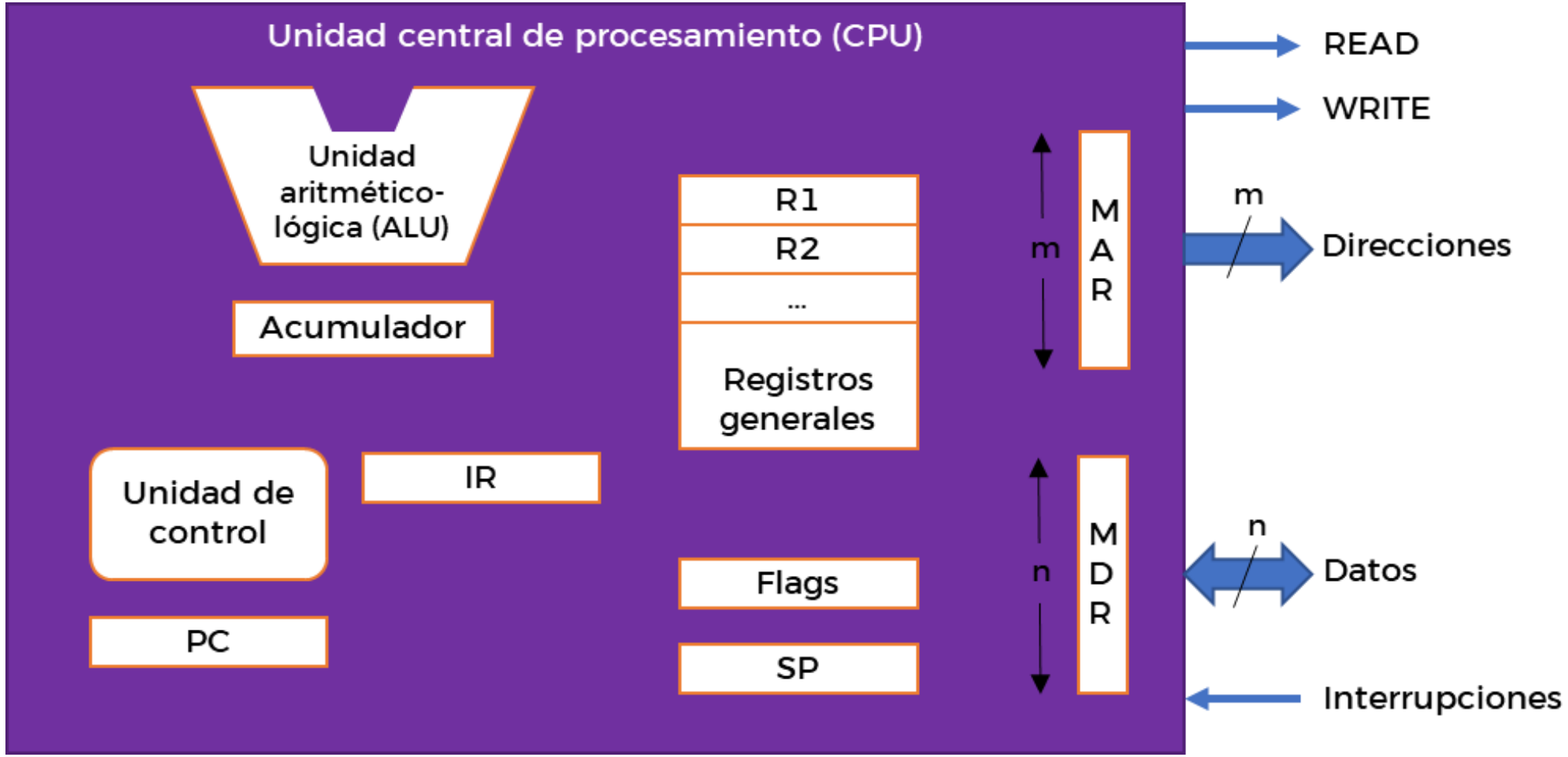


Espacio de direcciones: 2^m palabras

Descripción interna del procesador: los buses



Descripción interna del procesador: el “camino de datos”



Búsqueda y ejecución de una instrucción en el camino de datos

Inicialización

- PC ← dirección de la primera instrucción del programa

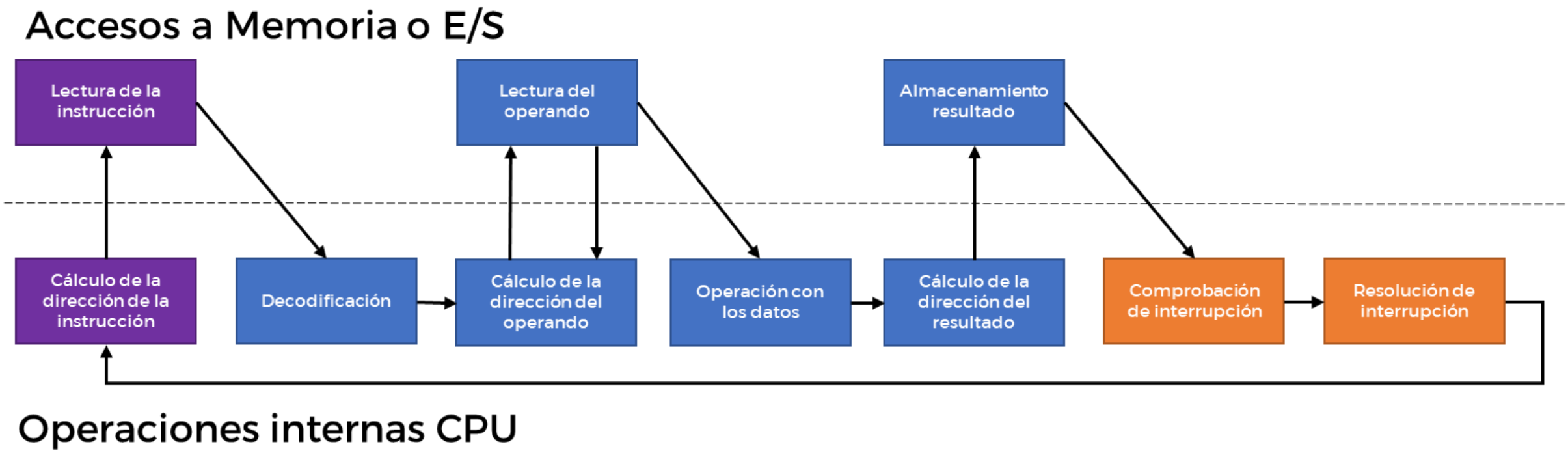
Ciclo de búsqueda

- MAR ← PC
- Operación de lectura (READ)
- Tras un breve tiempo: MDR ← instrucción
- IR ← MDR

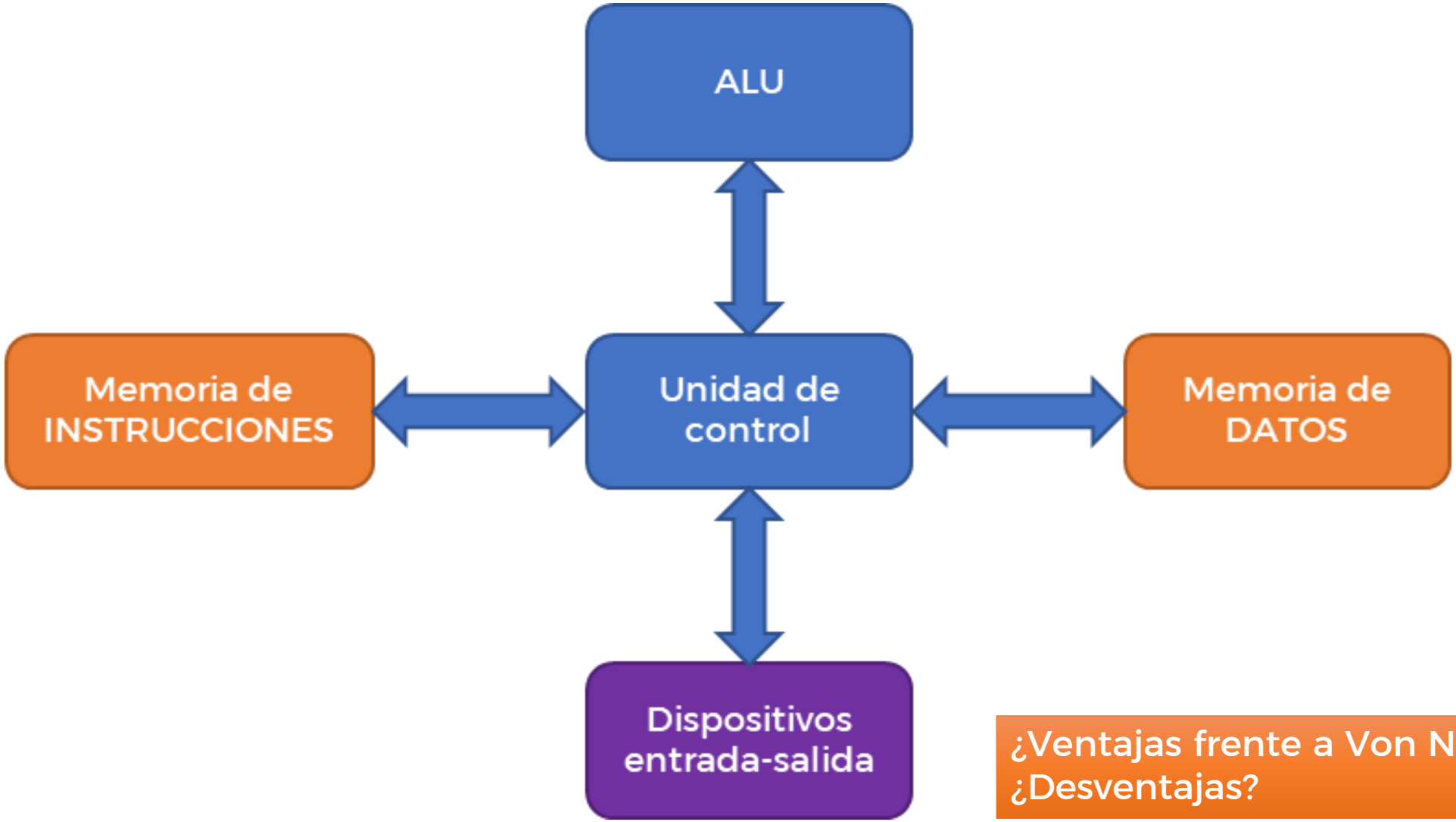
Ciclo de ejecución

- UC decodifica la instrucción
- UC descompone la ejecución de la instrucción en pasos o micro-operaciones
- PC ← PC + 1 (o el que corresponda)
- Volver al ciclo de búsqueda

Ciclo de instrucción (ampliado)



Otras arquitecturas: Harvard



¿Ventajas frente a Von Neumann?
¿Desventajas?

Instrucciones

Acciones elementales que puede ejecutar un computador

Requerimiento para realizar una determinada acción

¡**Indivisibles** desde la perspectiva del programador!

- Codificadas en binario: lenguaje máquina
- Programadores usan **ensamblador** (mnemónicos)

¿Son muy diferentes los repertorios de instrucciones de diferentes computadores?

Tipos de instrucciones

Transferencia

- De datos
- Hacia/desde Entrada/Salida

Operaciones

- Aritméticas
- Lógicas
- Punto Flotante

Control de Flujo de Ejecución (Saltos)

- Saltos o bifurcaciones incondicionales
- Saltos o bifurcaciones condicionales
- Saltos/retornos a/de subrutina

Otras

- Vectoriales
- Cadenas de caracteres
- Gráficos

Una instrucción debe contener toda la información necesaria para su ejecución

Independientemente de su posición en memoria

Independientemente de otras instrucciones

Estructura y codificación de las instrucciones

- Longitud
  - Fija
  - Variable

En general, más sencillo manejar longitudes fijas y relacionadas con la longitud de la palabra en memoria

¿Son mejores las instrucciones cortas o largas?

1. Según el número de operandos explícitos por instrucción

- Máquina de 3 direcciones

  - ADD R1, R2, R3 // R1 ← R2 + R3

- Máquina de 2 direcciones

  - ADD R1, R2 // R1 ← R1 + R2

- Máquina de 1 dirección

  - ADD R1 // ACC ← ACC + R1

- Máquina de 0 direcciones

  - ADD // <tope de la pila> ← <tope de la pila> + <tope de la pila – 1>

¿Tiene sentido que una máquina de 0 direcciones incluya instrucciones con algún operando?

2. Según el número máximo de operandos de memoria que puede usar la ALU

   - Arquitecturas registro-registro: 0/3 operandos de memoria. Ejemplos: MIPS, PowerPC, SPARC
   - Arquitecturas registro-memoria: 1/2 operandos de memoria. Ejempos: 8086, Motorola 68000
   - Arquitectura memoria-memoria: 2/2 o 3/3 operandos de memoria. Ejemplos: VAX.

3. Según la longitud del opcode

- Fija
- Variable

Técnicas de codificación de opcodes con longitud variable

- Codificación por expansión

Ningún opcode puede ser prefijo de otro

Técnicas de codificación de opcodes con longitud variable. Codificación por expansión

- Queremos construir un repertorio de instrucciones con las siguientes características:
  - 15 instrucciones de 3 operandos
  - 14 instrucciones de 2 operandos
  - 31 instrucciones de 1 operando
  - 16 instrucciones de 0 operandos

  - 16 registros y sólo operamos con registros en las instrucciones (4 bits por operando)

Técnicas de codificación de opcodes con longitud variable. Codificación por expansión

0	0	0	0	x	x	x	x	y	y	y	y	z	z	z	z	15 (3 operandos)
…																
1	1	1	0	x	x	x	x	y	y	y	y	z	z	z	z	
1	1	1	1	0	0	0	0	x	x	x	x	y	y	y	y	14 (2 operandos)
…																
1	1	1	1	1	1	0	1	x	x	x	x	y	y	y	y	
1	1	1	1	1	1	1	0	0	0	0	0	x	x	x	x	31 (1 operando)
…																
1	1	1	1	1	1	1	0	1	1	1	1	x	x	x	x	
1	1	1	1	1	1	1	1	0	0	0	0	x	x	x	x	
…																
1	1	1	1	1	1	1	1	1	1	1	0	x	x	x	x	
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	16 (0 operandos)
…																
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

Técnicas de codificación de opcodes con longitud variable

- Codificación Huffman
  - Asigna opcodes más cortos a las instrucciones más frecuentes

Ningún opcode puede ser prefijo de otro

¿Qué efecto (o ventaja) tiene esto?

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

1. Construir tabla de instrucciones en orden decreciente por frecuencia
2. Sumar las dos últimas posiciones y reordenar el resultado sin perder de vista de dónde vino
3. Repetir “2” hasta tener sólo dos elementos
4. Codificar estos dos elementos con un 0 y un 1 de forma aleatoria
5. Comenzar la “marcha atrás”. Si una frecuencia venía de la suma de otras dos, añadir un nuevo bit a la codificación de la instrucción. En caso contrario, dejar el código como está.
6. Repetir “5” hasta volver al principio.

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

- Queremos construir un repertorio de 8 instrucciones, sabiendo que su frecuencia relativa en los programas que se ejecutan es la siguiente:

Instrucción	Frecuencia de uso
LOAD	0,25
STORE	0,25
ADD	0,125
AND	0,125
NOT	0,0625
RSHIFT	0,0625
JUMP	0,0625
HALT	0,0625

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Inicialización		Iteración 1	
LOAD	0,25		0,25	
STORE	0,25		0,25	
ADD	0,125		0,125	
AND	0,125		0,125	
NOT	0,0625		0,125	
RSHIFT	0,0625		0,0625	
JUMP	0,0625		0,0625	
HALT	0,0625			

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 1		Iteración 2	
LOAD	0,25		0,25	
STORE	0,25		0,25	
ADD	0,125		0,125	
AND	0,125		0,125	
NOT	0,125		0,125	
RSHIFT	0,0625		0,125	
JUMP	0,0625			
HALT				

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 2		Iteración 3	
LOAD	0,25		0,25	
STORE	0,25		0,25	
ADD	0,125		0,25	
AND	0,125		0,125	
NOT	0,125		0,125	
RSHIFT	0,125			
JUMP				
HALT				

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 3		Iteración 4	
LOAD	0,25		0,25	
STORE	0,25		0,25	
ADD	0,25		0,25	
AND	0,125		0,25	
NOT	0,125			
RSHIFT				
JUMP				
HALT				

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 4		Iteración 5	
LOAD	0,25		0,5	
STORE	0,25		0,25	
ADD	0,25		0,25	
AND	0,25			
NOT				
RSHIFT				
JUMP				
HALT				

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 5		Iteración 6	
LOAD	0,5		0,5	
STORE	0,25		0,5	
ADD	0,25			
AND				
NOT				
RSHIFT				
JUMP				
HALT				

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 5		Iteración 6	
LOAD	0,5		0,5	1
STORE	0,25		0,5	0
ADD	0,25			
AND				
NOT				
RSHIFT				
JUMP				
HALT				

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 5		Iteración 6	
LOAD	0,5	1	0,5	1
STORE	0,25	01	0,5	0
ADD	0,25	00		
AND				
NOT				
RSHIFT				
JUMP				
HALT				

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 4		Iteración 5	
LOAD	0,25	01	0,5	1
STORE	0,25	00	0,25	01
ADD	0,25	11	0,25	00
AND	0,25	10		
NOT				
RSHIFT				
JUMP				
HALT				

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 3		Iteración 4	
LOAD	0,25	01	0,25	01
STORE	0,25	00	0,25	00
ADD	0,25	11	0,25	11
AND	0,125	101	0,25	10
NOT	0,125	100		
RSHIFT				
JUMP				
HALT				

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 2		Iteración 3	
LOAD	0,25	01	0,25	01
STORE	0,25	00	0,25	00
ADD	0,125	101	0,25	11
AND	0,125	100	0,125	101
NOT	0,125	111	0,125	100
RSHIFT	0,125	110		
JUMP				
HALT				

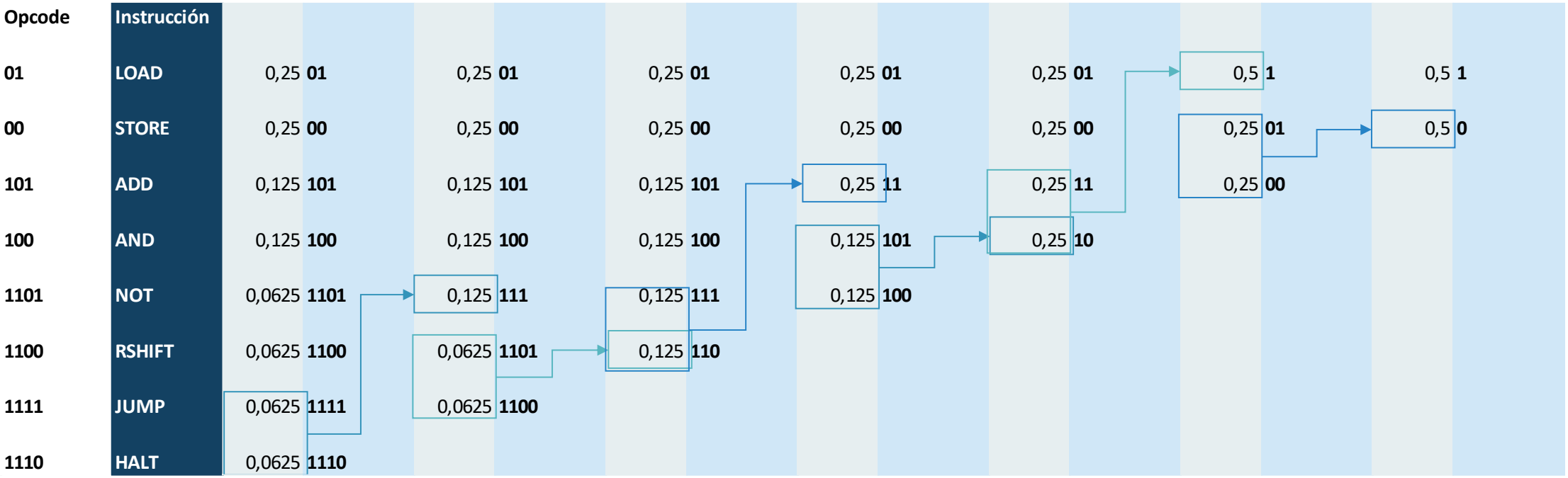
Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Iteración 1		Iteración 2	
LOAD	0,25	01	0,25	01
STORE	0,25	00	0,25	00
ADD	0,125	101	0,125	101
AND	0,125	100	0,125	100
NOT	0,125	111	0,125	111
RSHIFT	0,0625	1101	0,125	110
JUMP	0,0625	1100		
HALT				

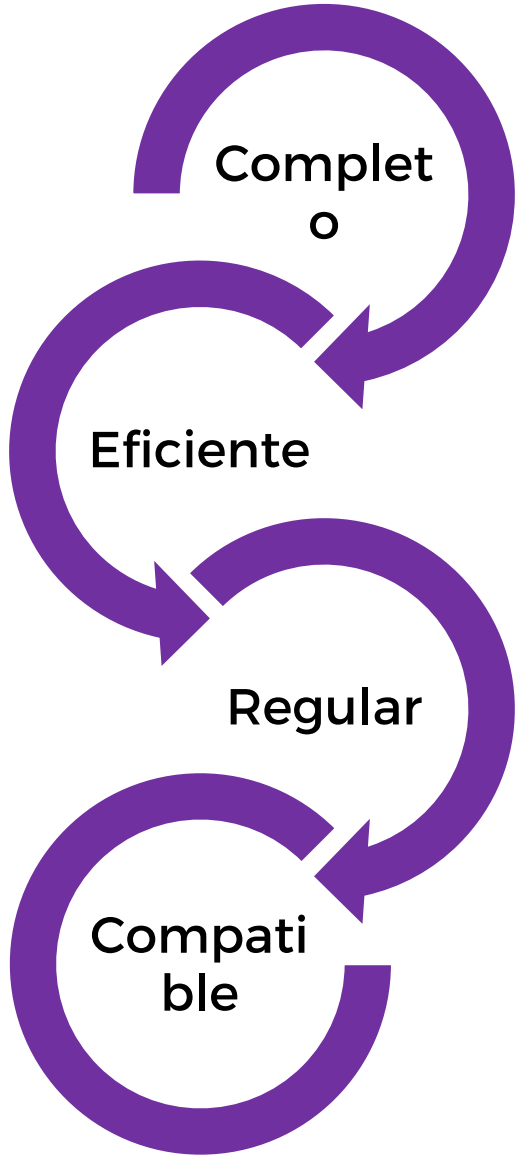
Técnicas de codificación de opcodes con longitud variable. Codificación Huffman

Instrucción	Inicialización		Iteración 1	
LOAD	0,25	01	0,25	01
STORE	0,25	00	0,25	00
ADD	0,125	101	0,125	101
AND	0,125	100	0,125	100
NOT	0,0625	1101	0,125	111
RSHIFT	0,0625	1100	0,0625	1101
JUMP	0,0625	1111	0,0625	1100
HALT	0,0625	1110		

Técnicas de codificación de opcodes con longitud variable. Codificación Huffman



Condiciones deseables de una codificación de un repertorio



Modos de direccionamiento

Modo de direccionamiento

- Abstracción que se emplea para especificar un operando en una instrucción, junto con el procedimiento asociado que permite obtener la dirección donde está almacenado el dato y, como consecuencia, el dato.

Los modos de direccionamiento permiten:

- Utilizar codificaciones más sintéticas que ocupen menos espacio para acceder a un mayor espacio de direcciones
- Simplificar el acceso a estructuras complejas en memoria

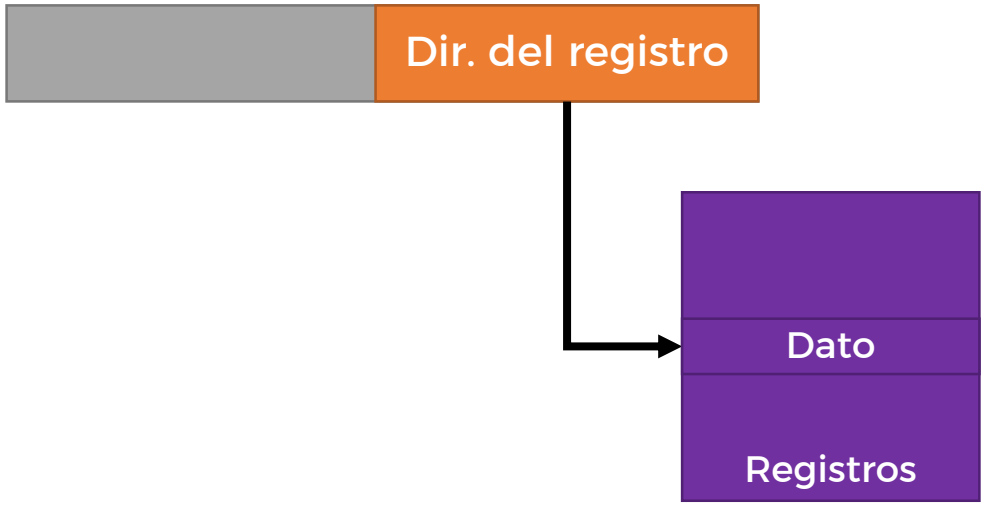
Modos de direccionamiento: notación

- #Inm: Valor inmediato
- A, B...: Dirección de memoria
- R_i: Registros
- (...): Indirección
- []: Direccionamiento indexado

Direccionamiento inmediato

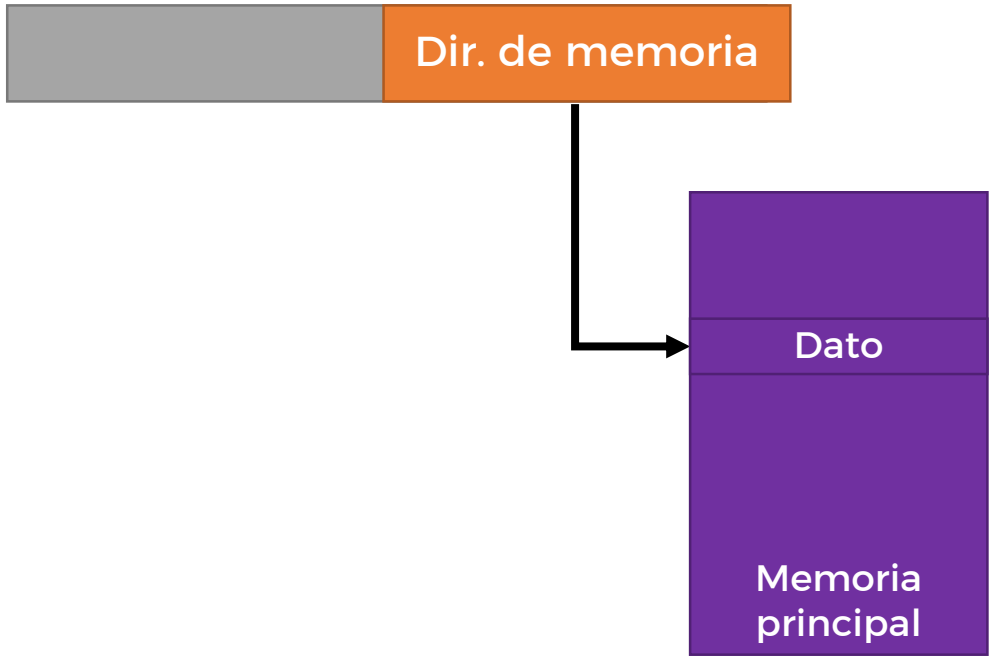
- ADD #5
- Dato: 5

Direccionamiento directo de registro



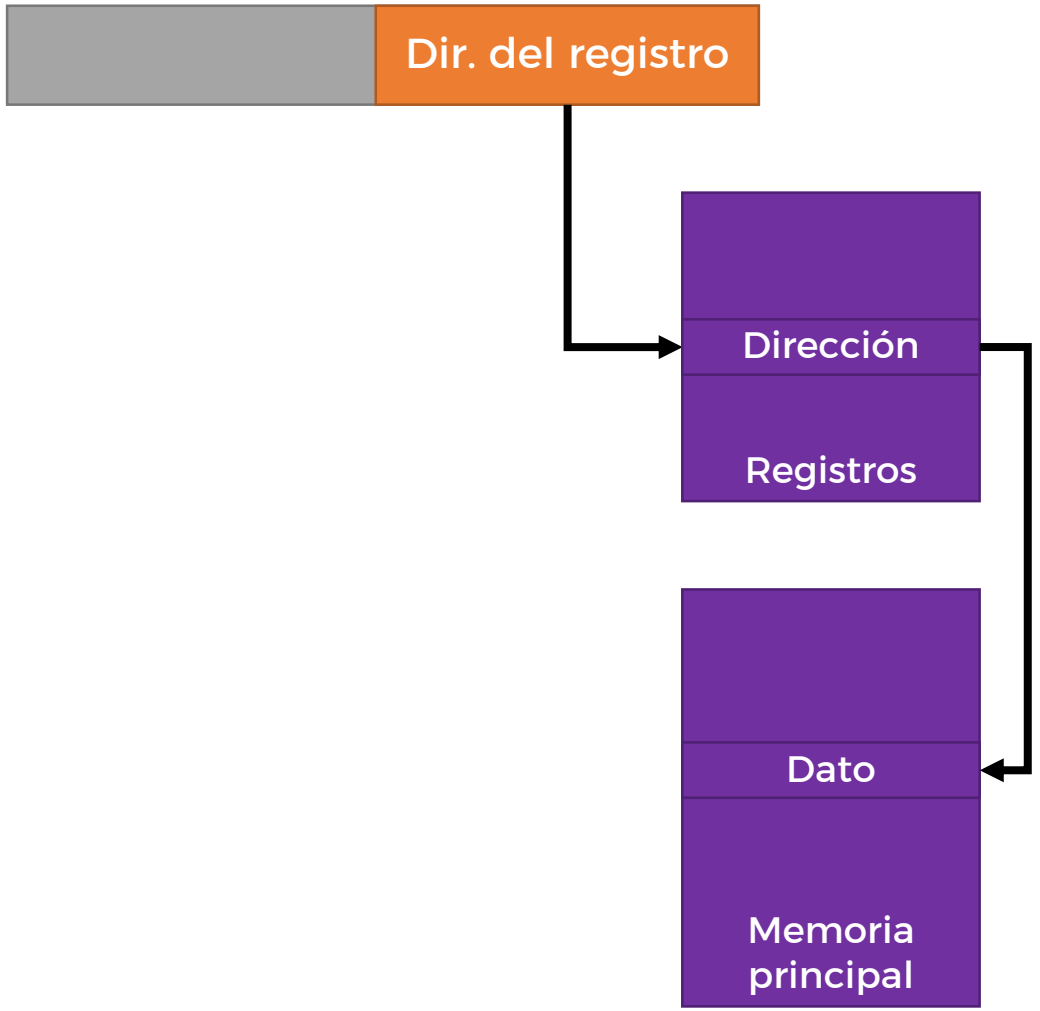
- ADD R1
- Dato: Contenido de R1

Direccionamiento directo de memoria



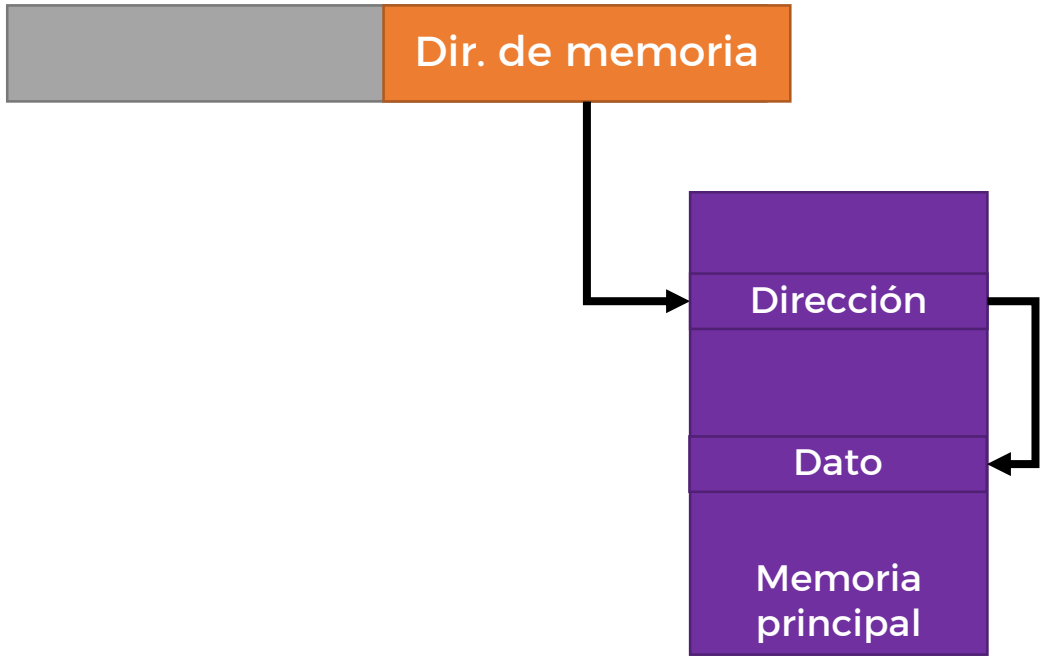
- ADD A
- Dirección efectiva: A
- Dato: Contenido en MEM[A]

Direccionamiento indirecto de registro



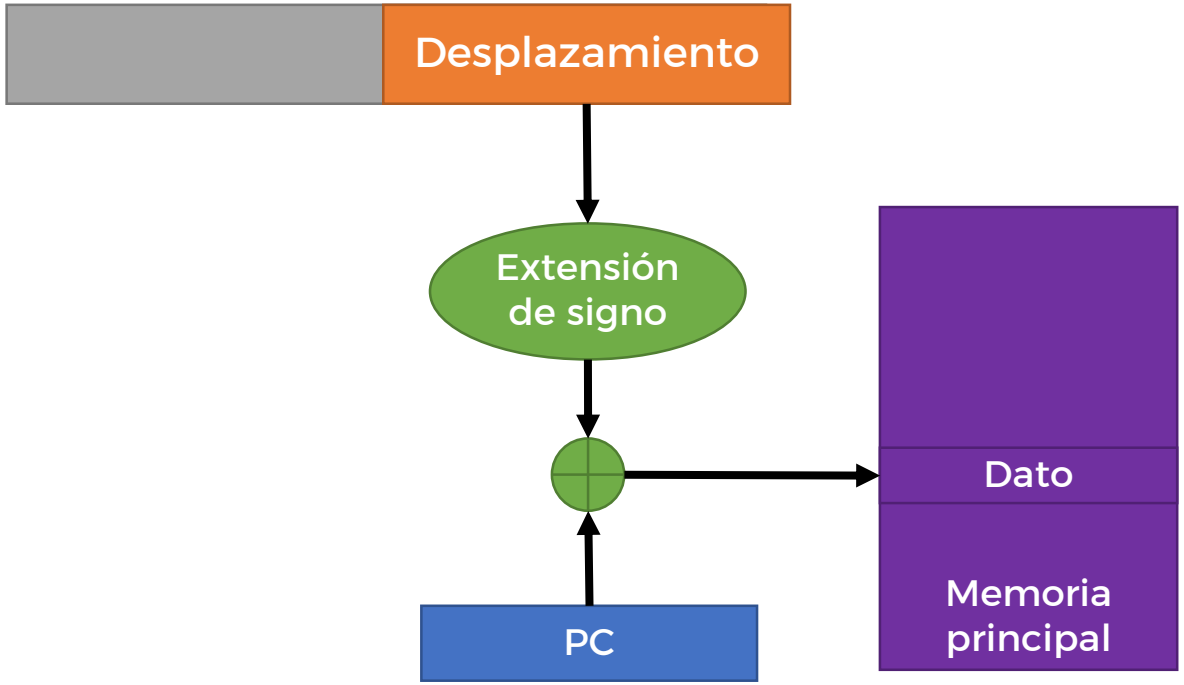
- ADD (R1)
- Dirección efectiva: R1
- Dato: Contenido en MEM[R1]

Direccionamiento indirecto de memoria



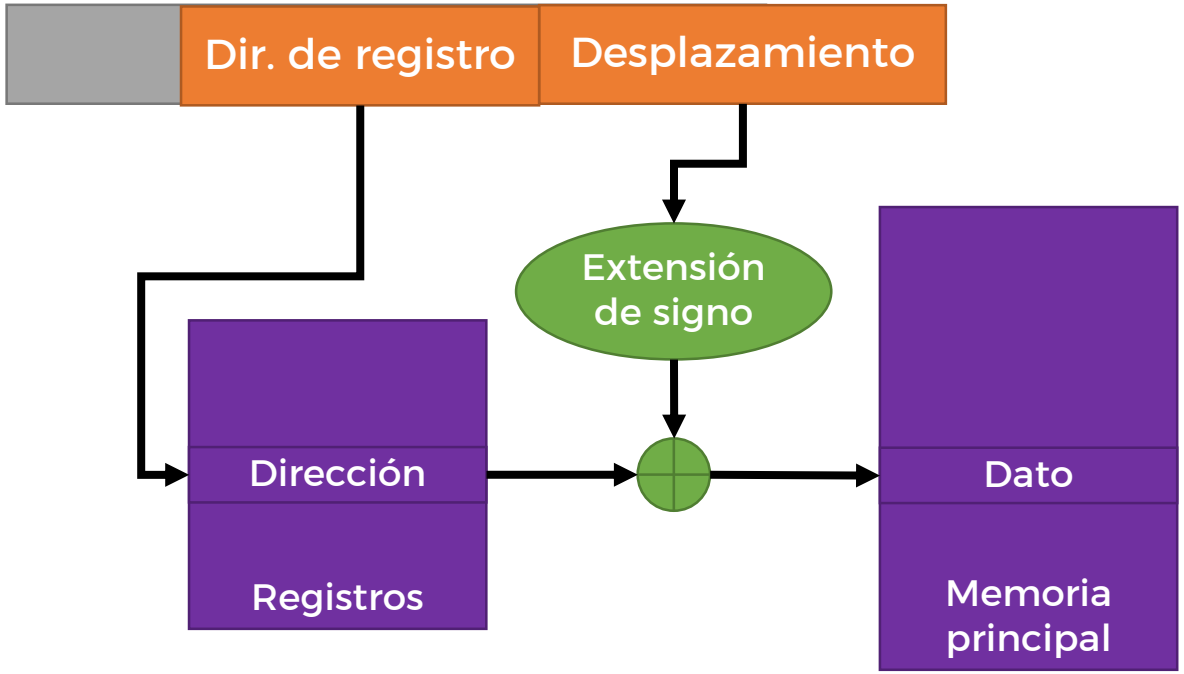
- ADD (A)
- Dirección efectiva: MEM[A]
- Dato: Contenido en MEM[MEM[A]]

Direccionamiento relativo (o relativo a PC)



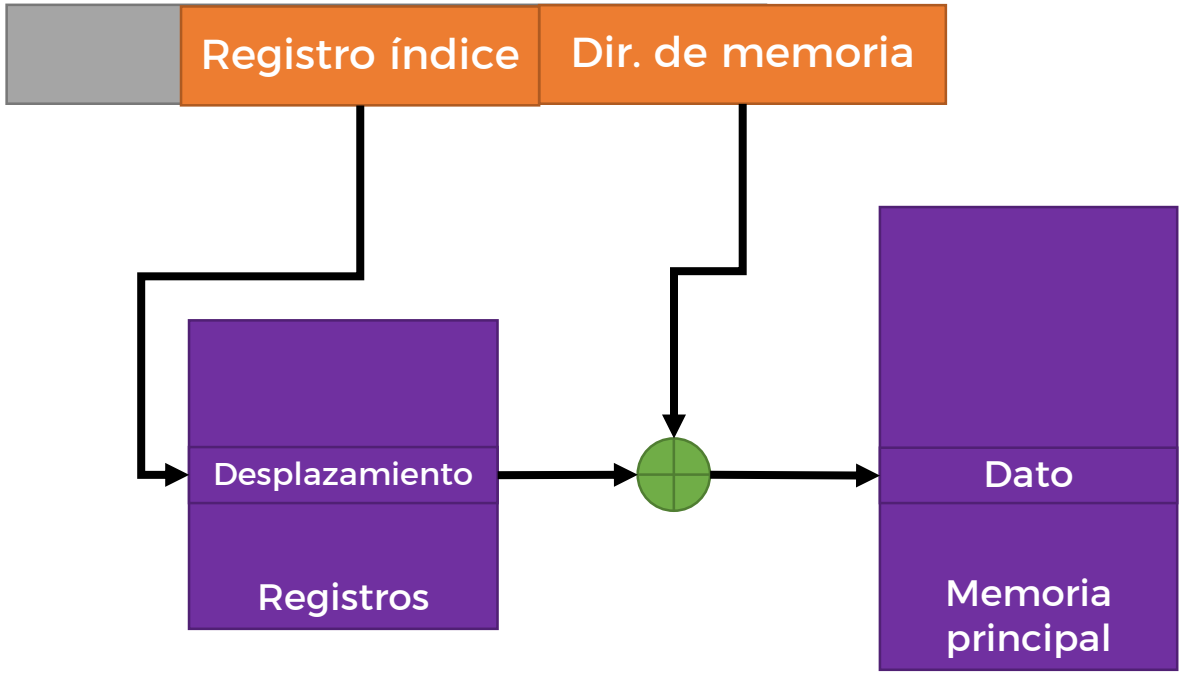
- BEQ 10
- Dirección efectiva: PC + 10
- Dato: Instrucción en MEM[PC + 10]

Direccionamiento relativo a registro base



- LW 2(R1) ó LW (R1, #2)
- Dirección efectiva: R1 + 2
- Dato: Contenido de MEM[R1 + 2]

Direccionamiento indexado



- LW A[R1]
- Dirección efectiva: A + R1
- Dato: MEM[A + R1]

- Tipo de direccionamiento **implícito**
  - Se asume que hay un registro que apunta a la cima o tope de la pila (SP: Stack Pointer)
  - PUSH A
    - Dirección efectiva: SP + 1
    - Dato: MEM[A]
  - POP X
    - Dirección efectiva: SP (se actualiza a SP – 1 tras coger el dato)
    - Dato: MEM[SP] (que se guarda en MEM[X])

Ventajas y desventajas de los modos de direccionamiento (1/2)

Modo de direccionamiento	Ventajas	Desventajas
Implícito	Rápido	Muy limitado
Inmediato	Rápido	Poco rango
Directo de registro	Rápido El campo de dirección es muy pequeño	Espacio direccionable limitado al nº de registros No conviene abusar salvo que se vayan a realizar operaciones repetidas sobre el mismo registro
Directo de memoria	Facilidad de implementación Espacio de direcciones disponible	Sólo se puede acceder al espacio de direcciones que nos quepa en el nº de bits del operando Lento (Memoria)
Indirecto de registro	La dirección de memoria se trata como un dato, lo que permite operar con ella Permite el paso por referencia a subrutinas	Más lento (requiere una indirección)

Ventajas y desventajas de los modos de direccionamiento (2/2)

Modo de direccionamiento	Ventajas	Desventajas
Indirecto de memoria	La dirección de memoria se trata como un dato, lo que permite operar con ella Permite el paso por referencia a subrutinas Permite un mayor espacio de direccionamiento	Más lento (requiere dos cargas desde memoria)
Relativo	Permite reubicación de programas Ahorra bits al ser implícito el PC	
Relativo a registro	Permite acceder a estructuras que se encuentran en posiciones no fijas en memoria.	Más lento (suma y memoria)
Indexado	Permite acceder a elementos diferentes de una estructura fija en memoria	Más lento (suma y memoria) Ocupa más que el anterior Si incluye post-pre incrementos, simplifica el software pero complica el hardware
Pila	Permite acceder a elementos en una pila Permite instrucciones muy compactas En máquinas específicas suele estar muy optimizado	

¿Qué hace a un computador mejor que otro?

Rendimiento de un computador

Rendimiento computacional

Capacidad

Coste

Consumo energético

Tamaño, robustez, durabilidad...

Tipos de computadores

Microcontrolador

Dispositivos móviles personales (SoC)

PC

Portátil

Servidor

Warehouse-scale computers

Criticidad según tipo de computador

	Coste	Consumo energético	Rendimiento computacional	Tamaño
Microcontrolador				
Dispositivos móviles (*SoC*)				
PC				
Portátil				
Servidor				
Warehouse-scale computers				

Criticidad según tipo de computador

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Internet of things/ embedded
Price of system	$100–$1000	$300–$2500	$5000–$10,000,000	$100,000–$200,000,000	$10–$100,000
Price of microprocessor	$10–$100	$50–$500	$200–$2000	$50–$250	$0.01–$100
Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance

Figure 1.2 A summary of the five mainstream computing classes and their system characteristics. Sales in 2015 included about 1.6 billion PMDs (90% cell phones), 275 million desktop PCs, and 15 million servers. The total number of embedded processors sold was nearly 19 billion. In total, 14.8 billion ARM-technology-based chips were shipped in 2015. Note the wide range in system price for servers and embedded systems, which go from USB keys to network routers. For servers, this range arises from the need for very large-scale multiprocessor systems for high-end transaction processing.

Fuente: Hennessy JL, Patterson DA. Computer Architecture, Sixth Edition: A Quantitative Approach. 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2018.

- Al menos dos formas de medirlo
  - Tiempo de ejecución (o tiempo de respuesta, o latencia) de un programa
  - Productividad (*throughput*): cantidad de trabajo realizada en un tiempo determinado

Rendimiento computacional: tiempo de ejecución

- $\text{Tiempo de un programa} = \text{Número de ciclos de un programa} \times \frac{\text{Tiempo}}{\text{Ciclo}}$

Asumimos un tiempo/ciclo constante.
¿Es esto siempre así?

Rendimiento computacional: tiempo de ejecución

- Ciclos por instrucción (CPI) $= \frac{\text{Ciclos del programa}}{\text{Nº de instrucciones del programa}}$
  - En promedio
  - También se usa la medida inversa $IPC = \frac{1}{CPI}$

Rendimiento computacional: tiempo de ejecución

- $\text{Tiempo de un programa} = \frac{\text{Número de instrucciones}}{\text{Programa}} \times \frac{\text{Número de ciclos}}{\text{Instrucción}} \times \frac{\text{Tiempo}}{\text{Ciclo}}$

Rendimiento computacional: tiempo de ejecución

$$\text{Tiempo de un programa} = \frac{\text{Número de instrucciones}}{\text{Programa}} \times \frac{\text{Número de ciclos}}{\text{Instrucción}} \times \frac{\text{Tiempo}}{\text{Ciclo}}$$

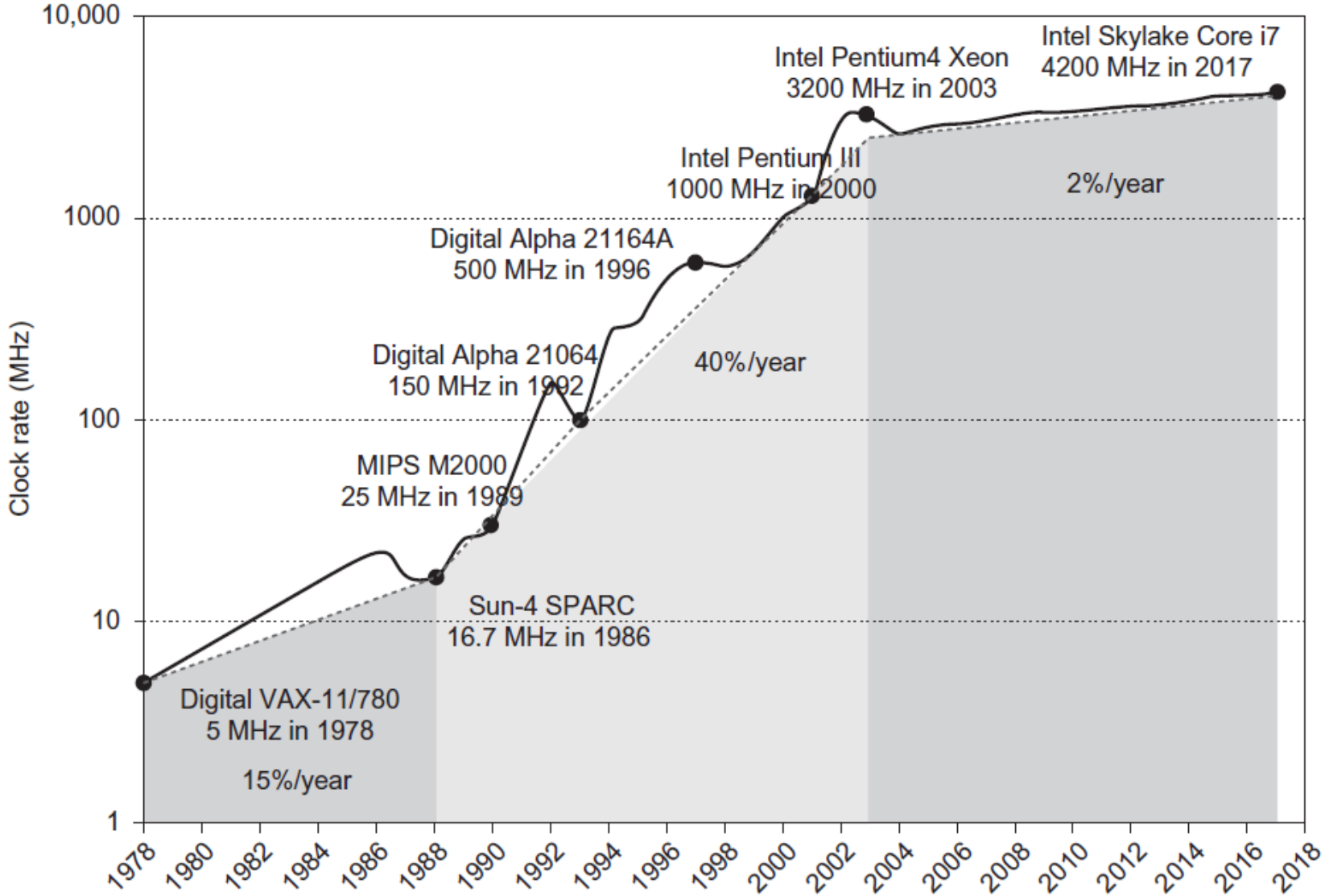
Supongan los siguientes niveles de abstracción de un computador:

- Algoritmo
- Lenguaje de programación / compilador
- Arquitectura del Repertorio de Instrucciones
- Microarquitectura o estructura
- “Tecnología”

¿Qué niveles afectan a qué términos de esa ecuación?

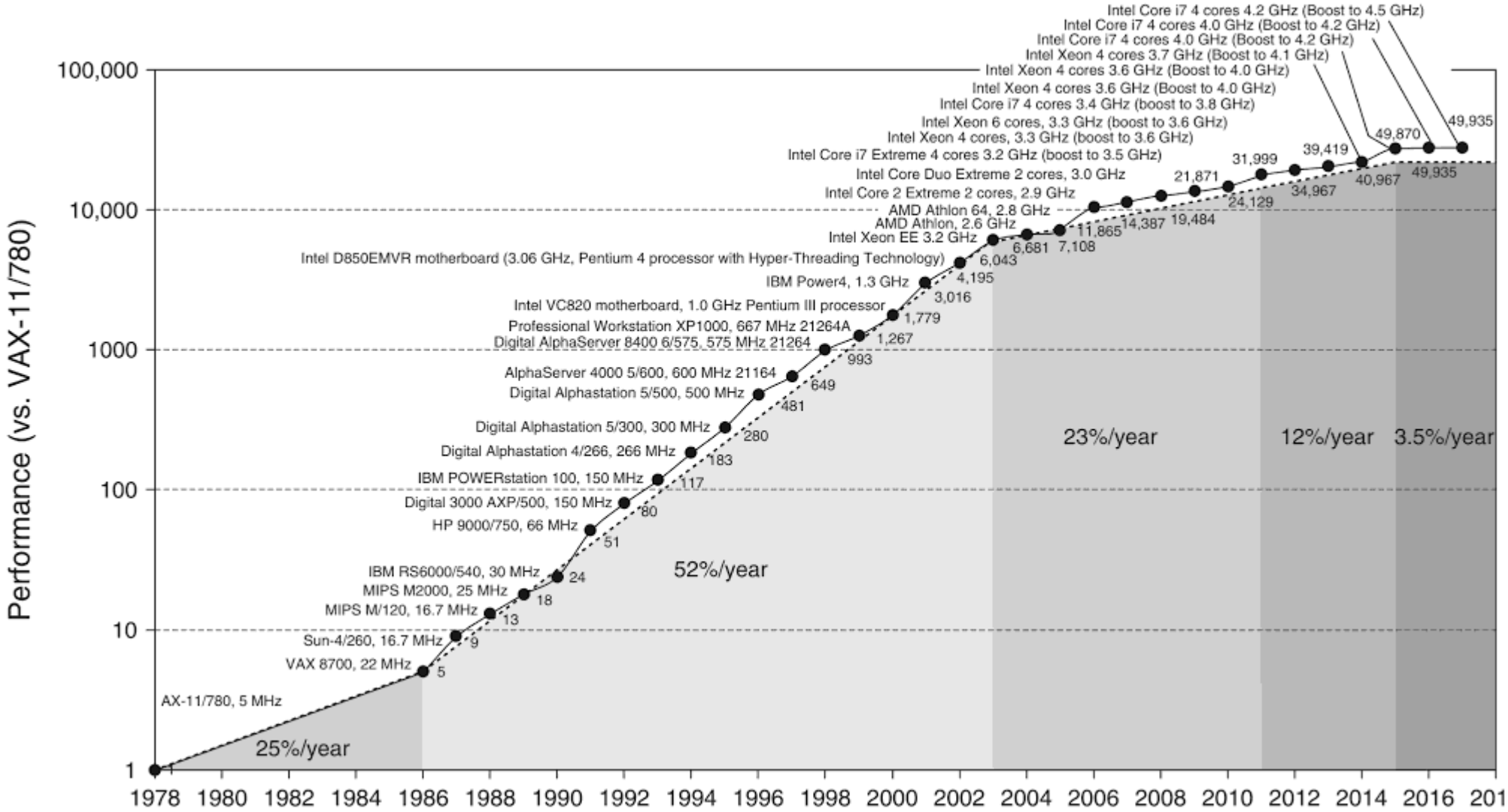
¿Cómo podemos mejorar el rendimiento computacional de un computador?

Evolución de las frecuencias de reloj en los microprocesadores



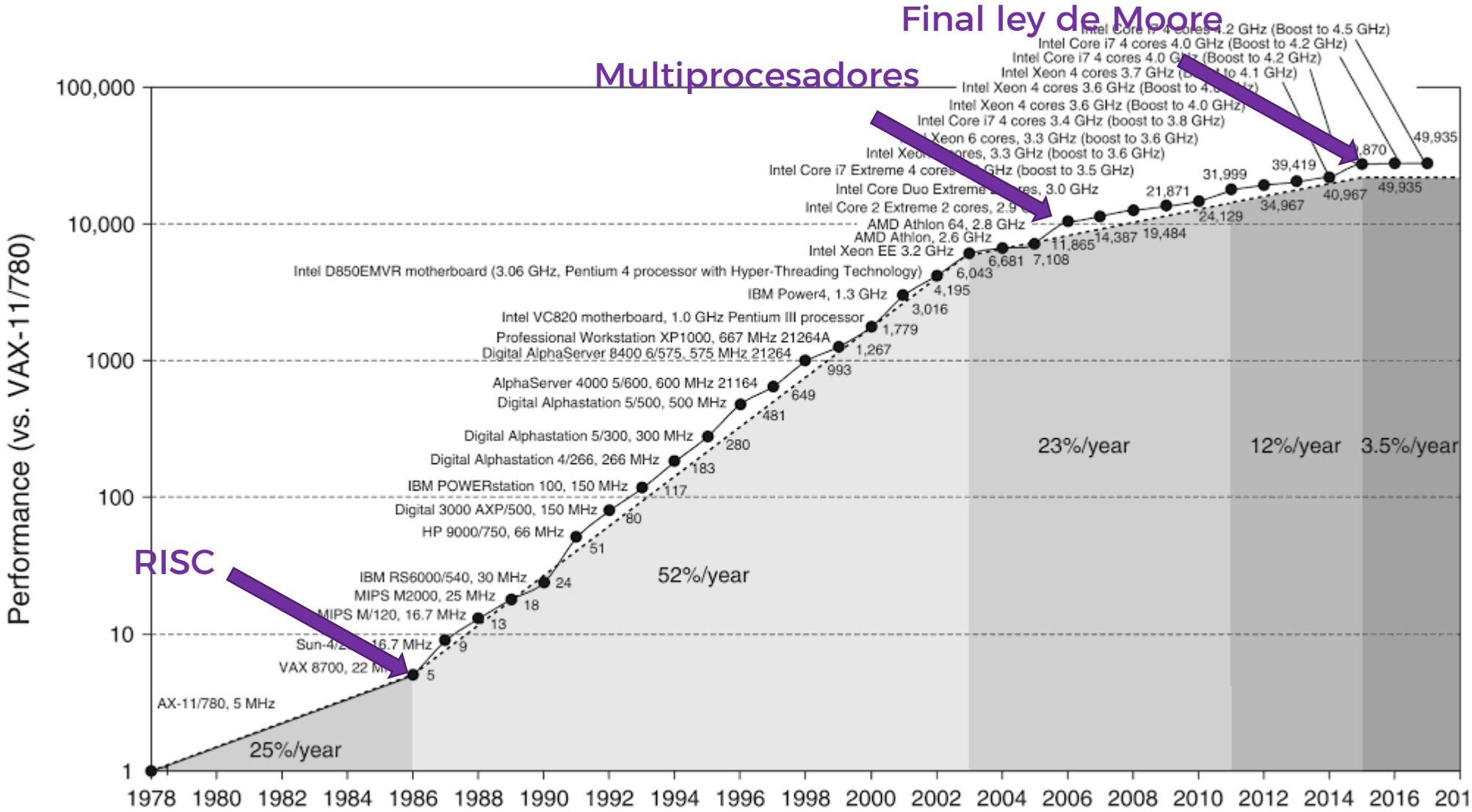
Fuente: Hennessy JL, Patterson DA. Computer Architecture, Sixth Edition: A Quantitative Approach. 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2018.

Evolución del rendimiento de los procesadores



Fuente: Hennessy JL, Patterson DA. Computer Architecture, Fifth Edition: A Quantitative Approach. 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2011.

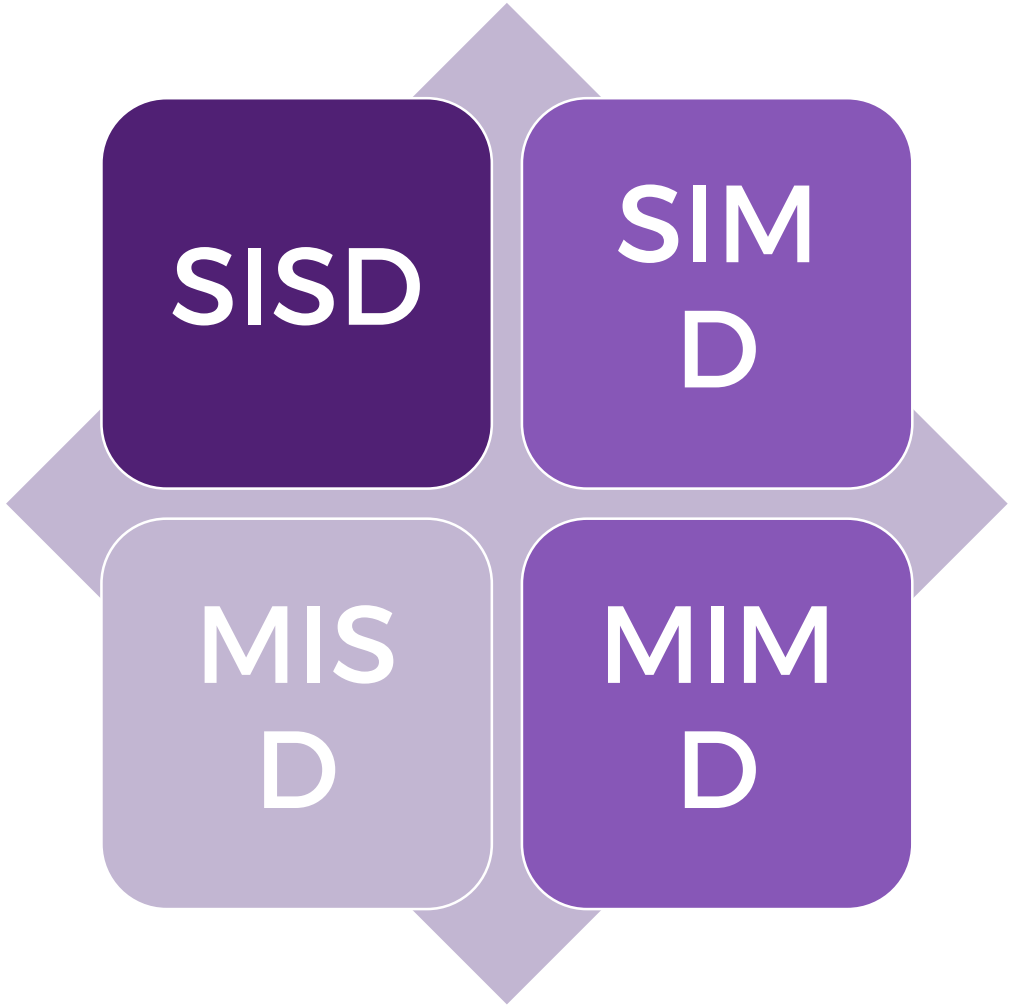
Evolución del rendimiento de los procesadores



Fuente: Hennessy JL, Patterson DA. Computer Architecture, Fifth Edition: A Quantitative Approach. 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2011.

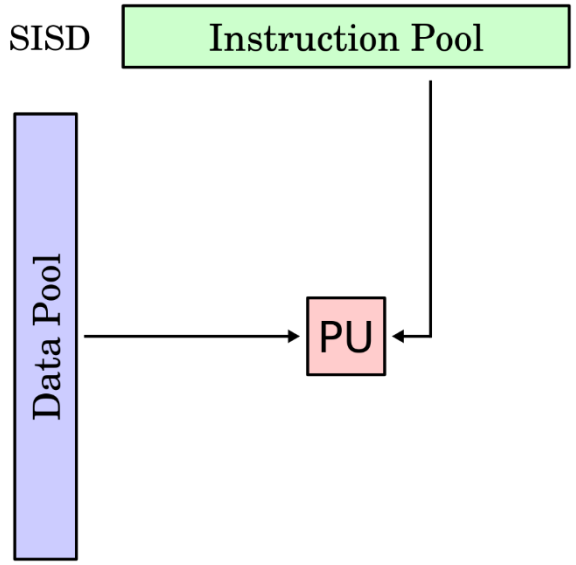
Mejoras del rendimiento computacional: paralelismo y clasificación de Flynn

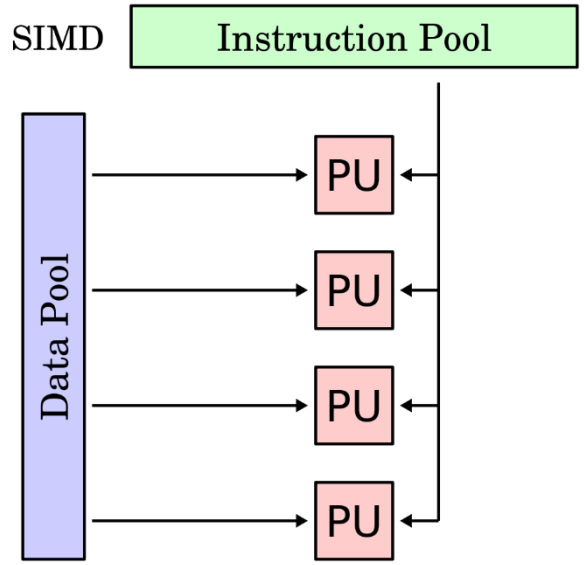
S: Single
M: Multiple

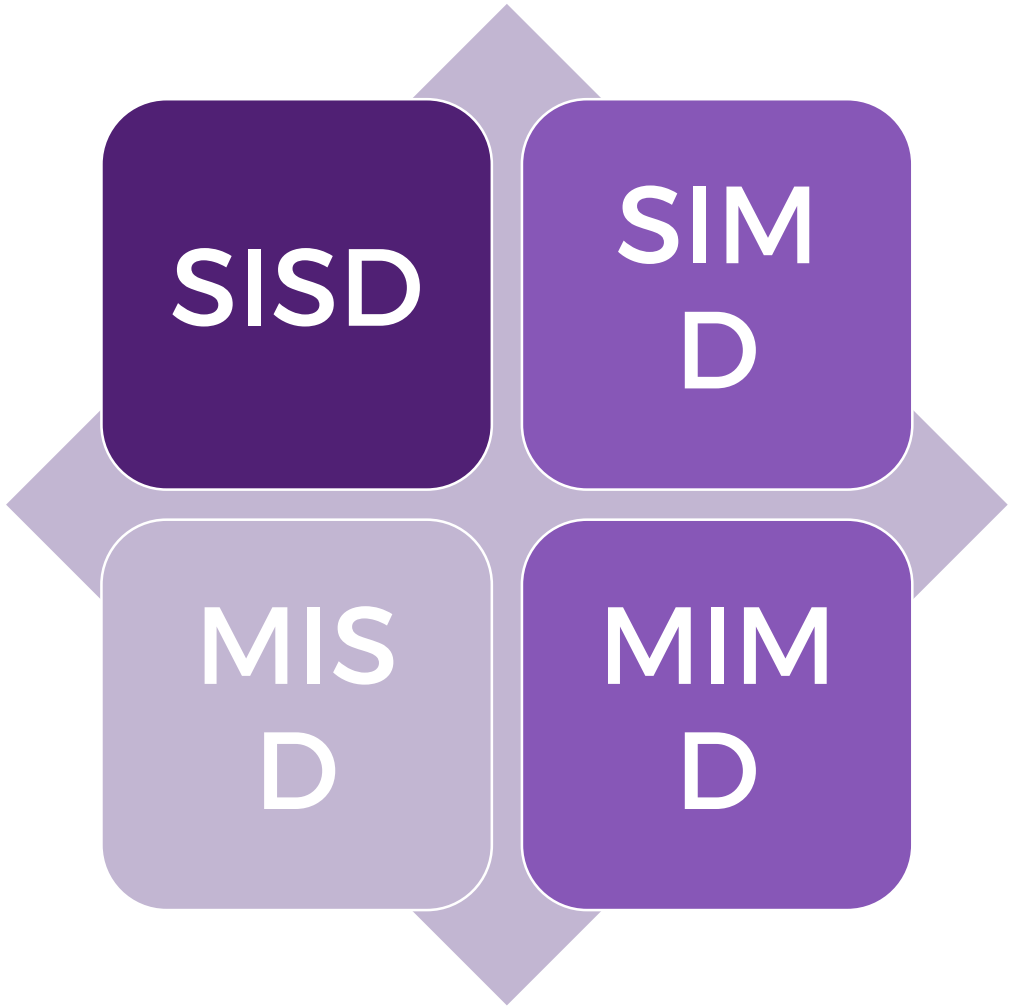


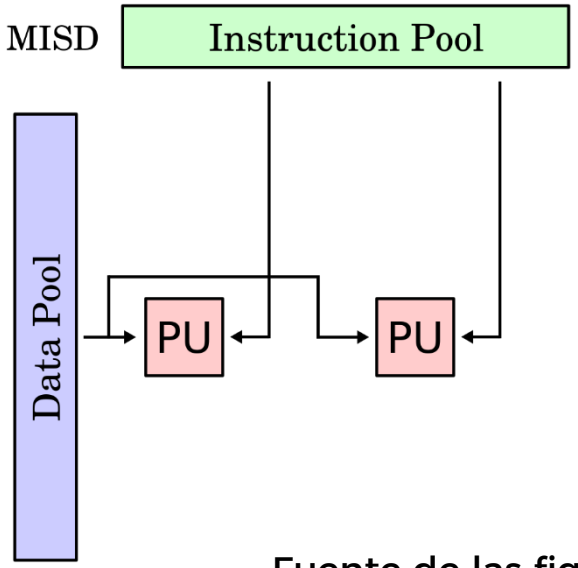
I: Instructions
D: Data

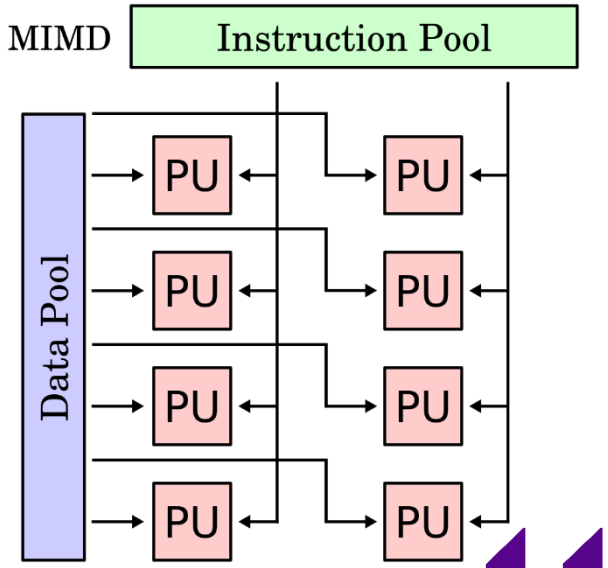
Mejoras del rendimiento computacional: paralelismo y clasificación de Flynn









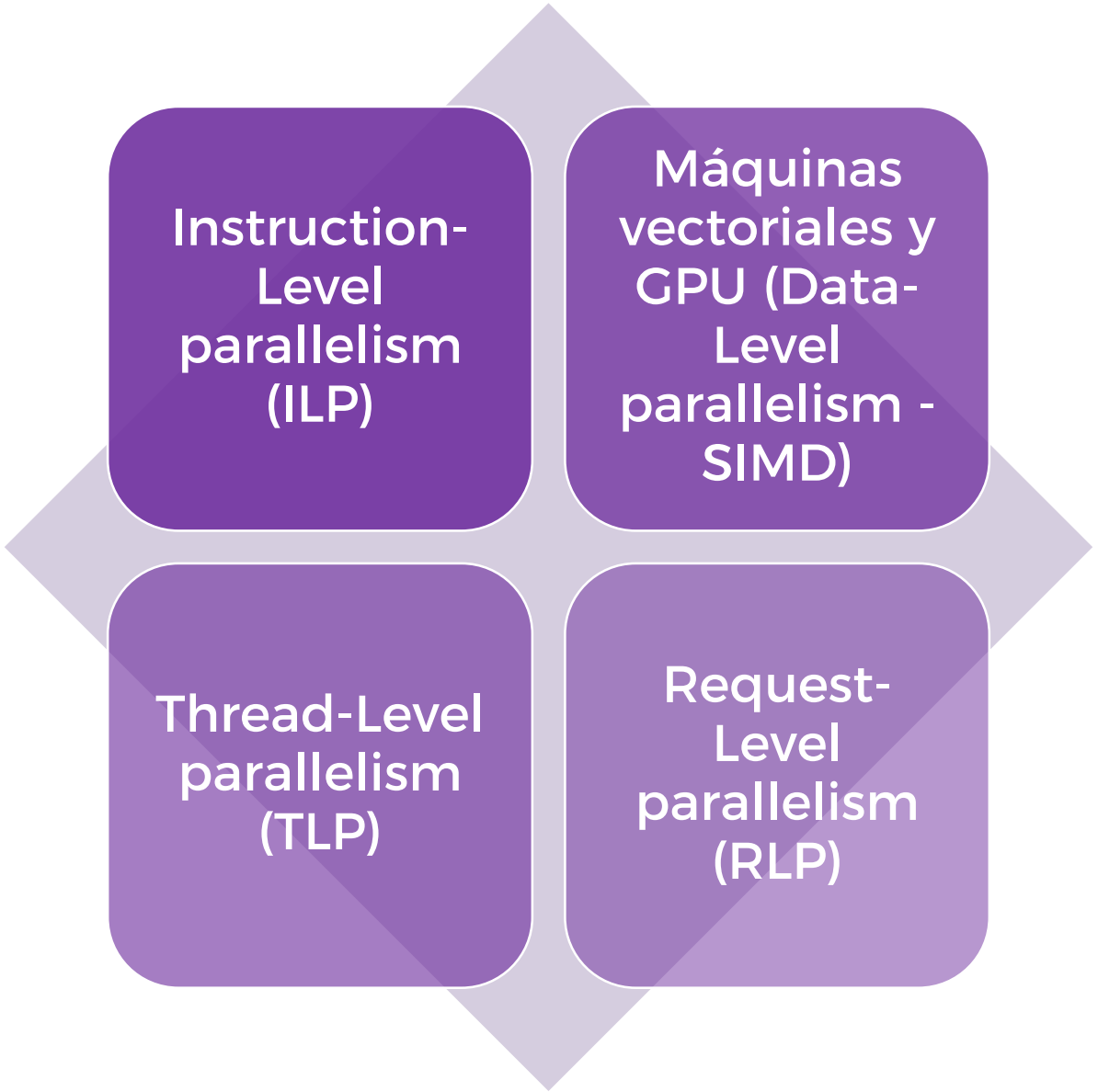


Fuente de las figuras: De I, Cburnett, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2233537

Data-Level parallelism

Task-Level parallelism

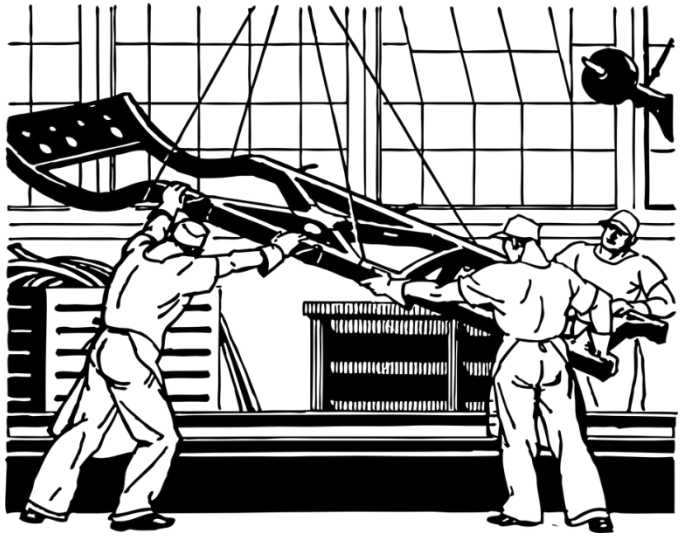
Paralelismo en el hardware

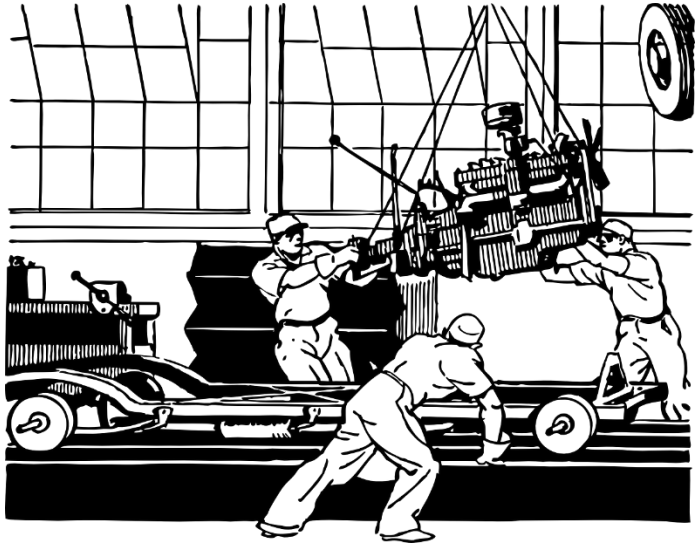


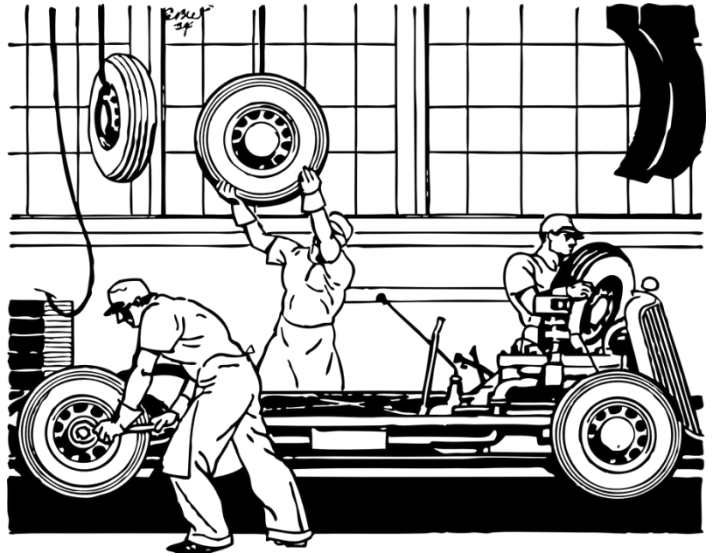
¿En qué aspectos debemos fijarnos para poder explotar el paralelismo a nivel de instrucción?

- Determinar las relaciones de dependencia entre estas instrucciones.
- Adecuar los recursos hardware a la ejecución de varias instrucciones en paralelo.
- Diseñar estrategias que permitan determinar cuándo una instrucción está lista para ejecutarse.
- Diseñar técnicas para pasar valores de una a otra instrucción

Segmentación (*pipelining*)

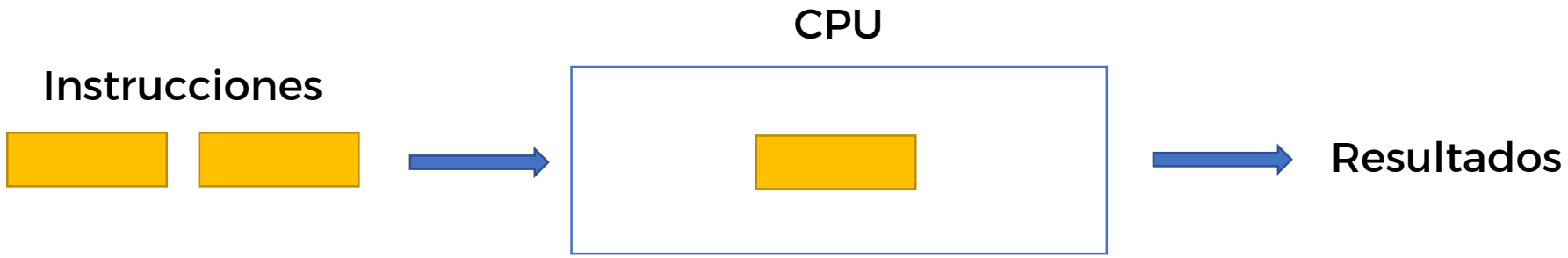




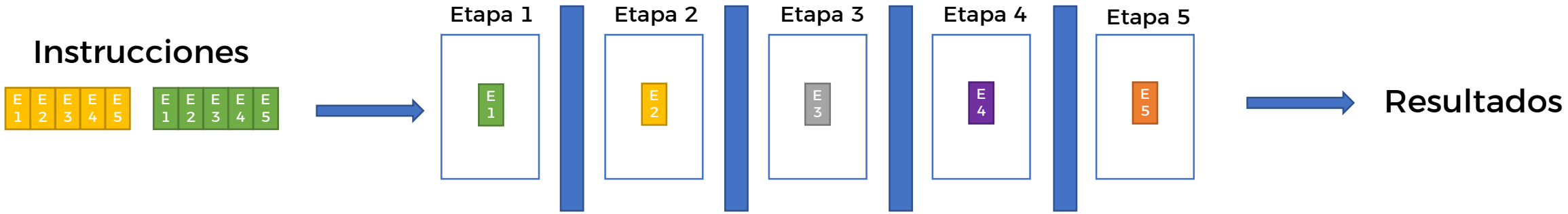


Segmentación: principios básicos

Ejecución no segmentada



Ejecución segmentada



- Todas las instrucciones atraviesan todas las etapas
- Dos etapas diferentes no comparten recursos
- La latencia o tiempo de propagación de una etapa a la siguiente es siempre igual
- Lo que ocurre con una instrucción en una etapa no debe depender de lo que ocurra con otras instrucciones en otras etapas

Si suponemos que cada etapa dura un ciclo de reloj, ¿cuál sería el mejor CPI que se podría obtener en una máquina segmentada?

¿En qué medida mejoraría esto la ejecución de un programa?

¿Reduce la segmentación el tiempo de ejecución de las instrucciones?

Paralelismo a nivel de instrucción: más allá de la segmentación

¿Cómo mejoramos el rendimiento (todavía más) explotando el paralelismo a nivel de instrucción (ILP)?

Paralelismo a nivel de instrucción: más allá de la segmentación

- Aumentar la profundidad del pipeline
- Emitiendo más de una instrucción por ciclo

¿Cómo conseguimos hacer esto?

Paralelismo a nivel de instrucción: aproximaciones

Hardware

- Planificación dinámica
- Superescalares

Software

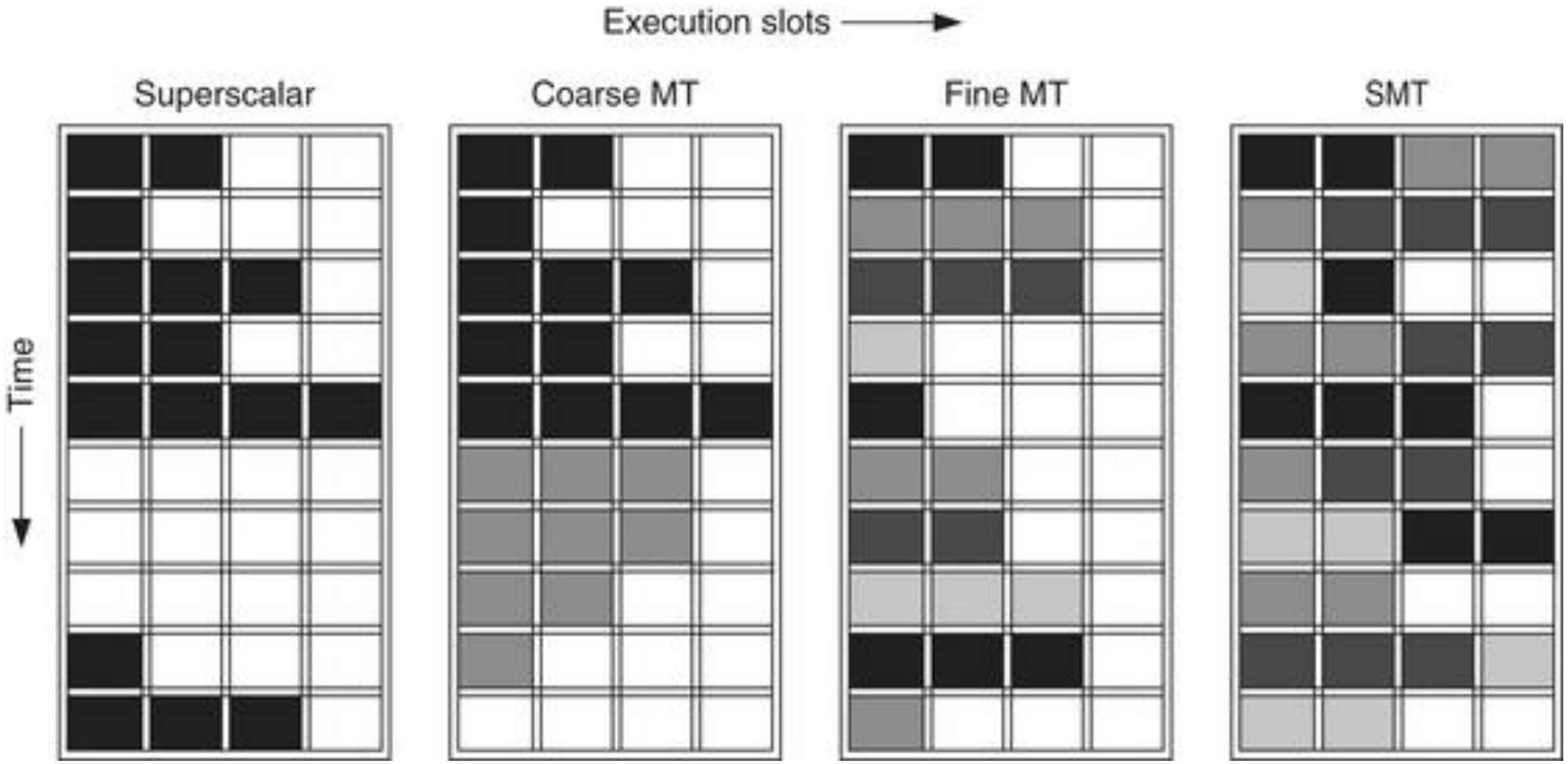
- Planificación estática
- VLIW/EPIC

Multithreading

- Grano fino
- Grano grueso

Simultaneous Multithreading (SMT)

Comparación de diferentes tipos de paralelismo



Fuente: Hennessy JL, Patterson DA. Computer Architecture, Fifth Edition: A Quantitative Approach. 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2011.