



Tema 1: Introducción a la Arquitectura de Computadores. Análisis de prestaciones.

Arquitectura de Computadores

Grado en Ingeniería Informática

Francisco José Quesada Real

Curso 2021/2022

(Adaptación de diapositivas de Mercedes Rodríguez García)

Índice

01

Introducción a la arquitectura de computadores

1. Definición de arquitectura de computadores
2. Niveles de descripción de un computador
3. Ámbito de estudio
4. Objeto de estudio

02

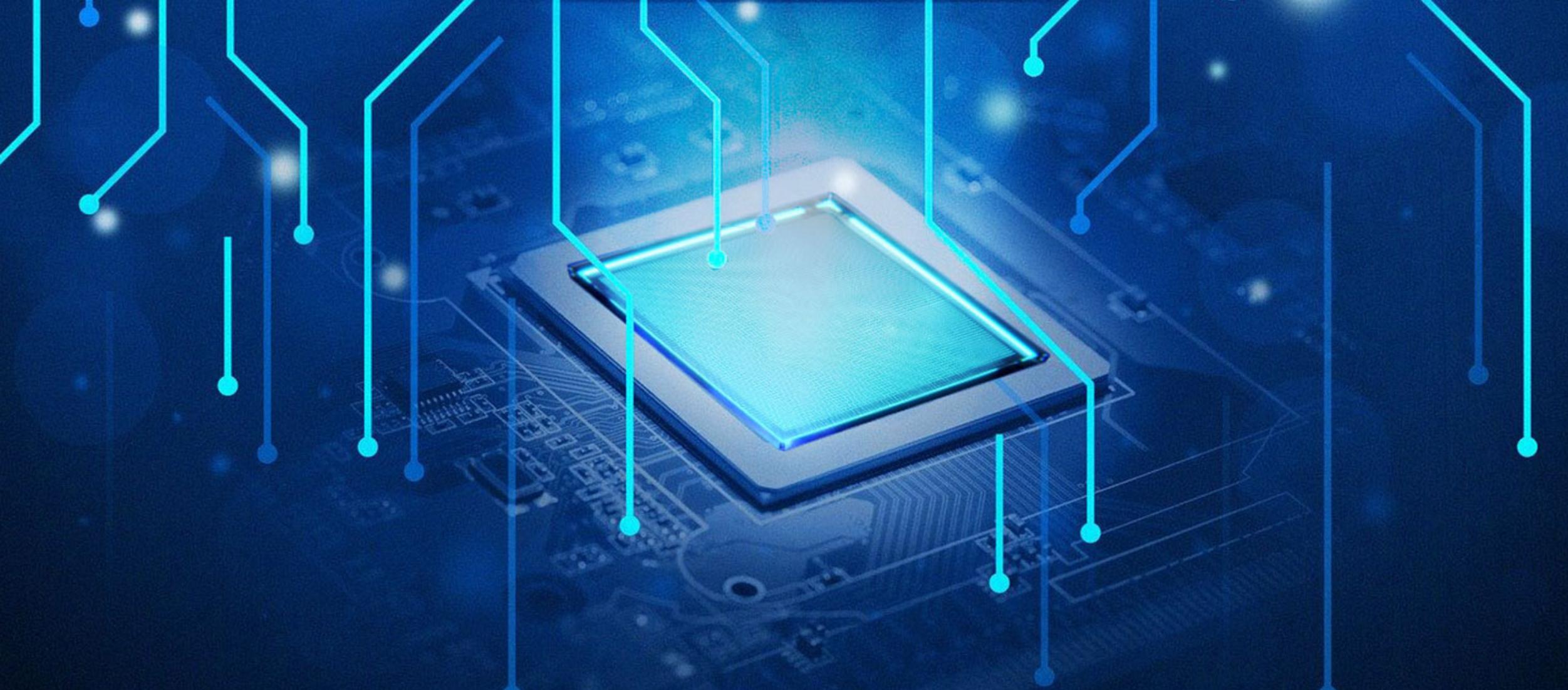
Clasificación de arquitecturas

1. Taxonomía de Flynn
 1. Computadores SISD
 2. Computadores SIMD
 3. Computadores MIMD
 4. Computadores MISD
2. Paralelismo a nivel de instrucción (ILP)
 1. Procesador no segmentado
 2. Procesador segmentado
 3. Procesador superescalar
 4. Procesador supersegmentado

03

Evaluación de prestaciones

1. Medidas de rendimiento
 1. Tiempo de CPU
 2. MIPS
 3. FLOPS
2. Benchmark
3. Ley de Amdahl



1.- Introducción a la Arquitectura de Computadores

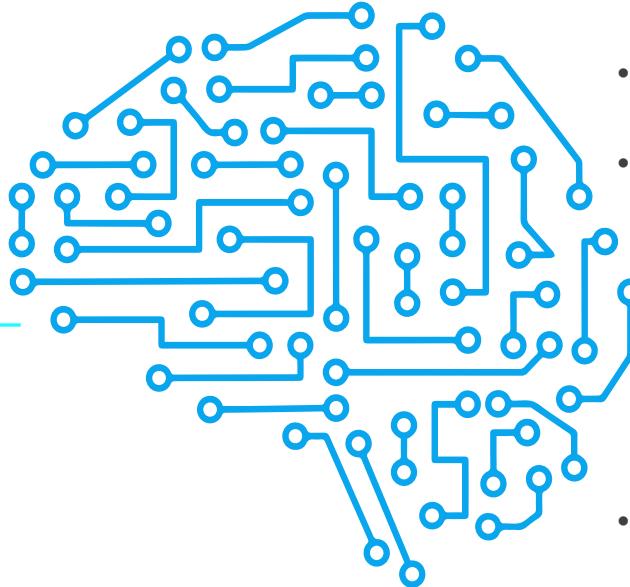
• 1.1 ¿Definición de Arquitectura de Computadores?

Instruction Set Architecture (ISA)

- Instrucciones disponibles
- Formato de instrucción
- Registros
- Codificación de datos (Complemento a 2, coma flotante...)
- Modos de direccionamiento

ISA

Diseño de la organización y del hardware para cumplir los objetivos y los requerimientos funcionales.



Organización (microarquitectura)

- Incluye los aspectos de alto nivel del diseño de un computador (memoria del sistema, diseño de la CPU)
- Define a nivel de esquemático de circuito:
 - Organización interna
 - Elementos funcionales (ALUs, cachés, camino de datos...)
 - Técnicas de mejora del rendimiento (pipeline, superescalar, vectorial...)

Hardware

- Especificaciones del ordenador (lógica de diseño, tecnología del ordenador)
- Arquitectura de sistemas -> Integración de CPU con el resto de elementos esenciales
- Tecnología de fabricación -> Implementación de diseños de forma física

Organización + Hardware

1.1 Niveles de descripción de un computador



- Una **descripción** del computador en **niveles** permite abordar su estudio de forma estructurada y ordenada.
- Interpretación del esquema. El **nivel más bajo ofrece recursos al nivel más alto**.
- Cada nivel debe estar **implementado con la máxima eficiencia**
- Sobre el nivel 7: en el desarrollo de **compiladores** hay que **tener** muy **en cuenta la arquitectura** del computador para aprovechar todos sus características.

1.2 Ámbito de estudio



1.3 Objeto de estudio

Diseño

Sistemas que alcancen los requisitos establecidos por el mercado (precio, consumo y prestaciones)

Evaluación

Características del computador para identificar posibles cuellos de botella

Diseño

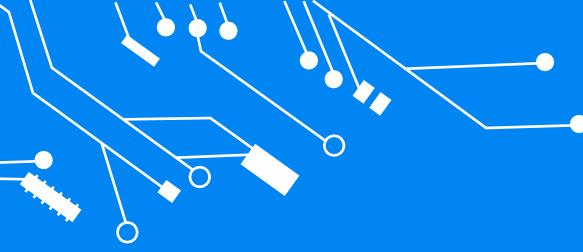


Aprovechamiento

Evaluación

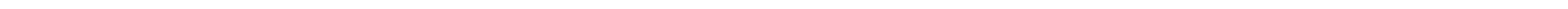
Aprovechamiento

Prestaciones del computador para escribir aplicaciones, compiladores y sistemas operativos eficaces



1.4 Objeto de estudio

1.4.1. Tipos de computadores



Servidores

Rendimiento, disponibilidad, escalabilidad, energía.

Dispositivos móviles personales

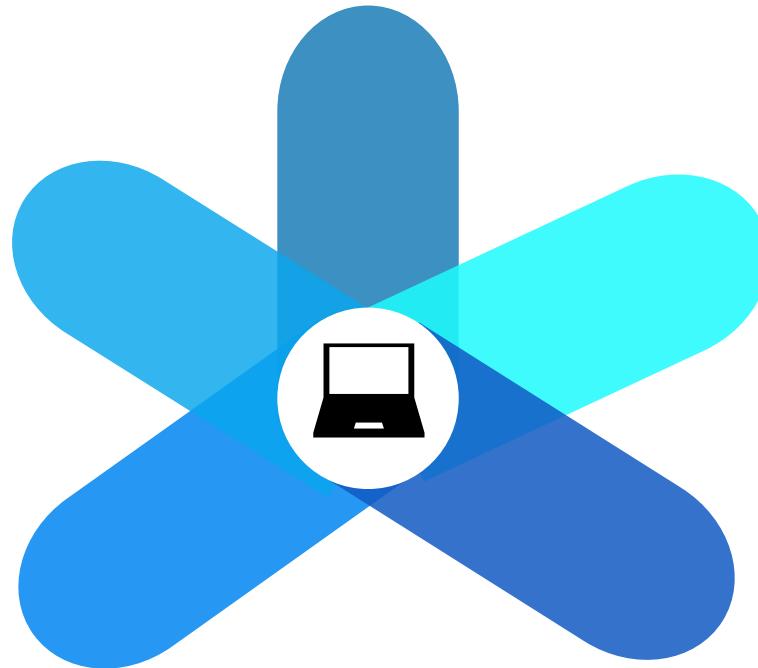
Coste, energía, rendimiento multimedia, capacidad de respuesta.

Sobremesa / portátiles

Precio/rendimiento, energía, rendimiento gráficos.

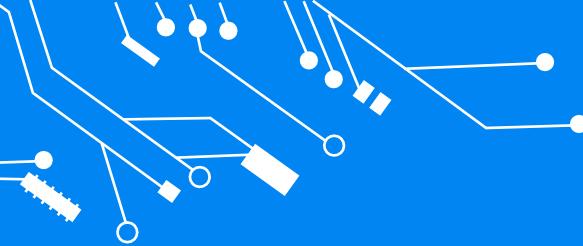
IoT / embebidos

Precio, energía, aplicación-rendimiento específico.



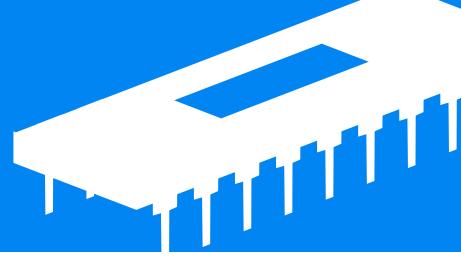
Clusters / warehouse

Precio/rendimiento, desempeño, proporción de energía.



1.4 Objeto de estudio

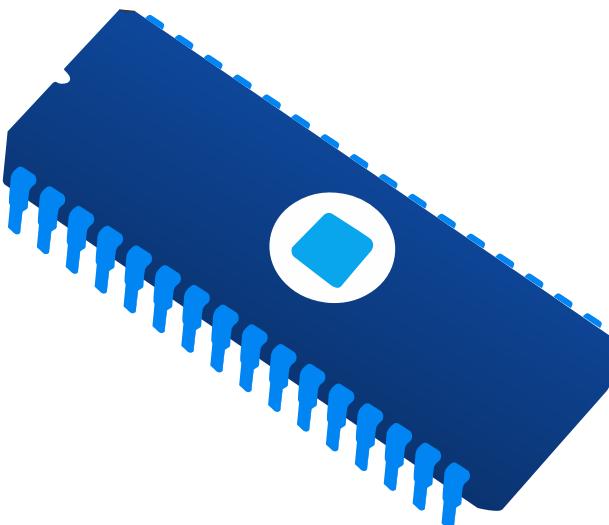
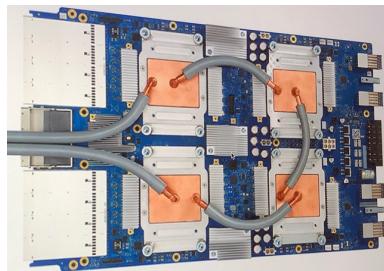
1.4.2. Evolución



Escala de Dennard

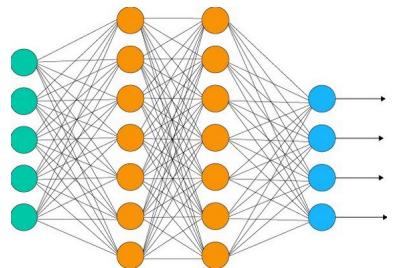
“A medida que los transistores reducen su tamaño físico, la densidad de potencia se mantiene constante”

- Finaliza en 2004
- Uso de microprocesadores con múltiples procesadores (cores)
- Se suspenden proyectos enfocados al alto rendimiento de procesadores de un core
- Se pasa de trabajar exclusivamente en paralelismo a nivel de instrucción (ILP) a otros paralelismos (DLP, TLP y RLP)

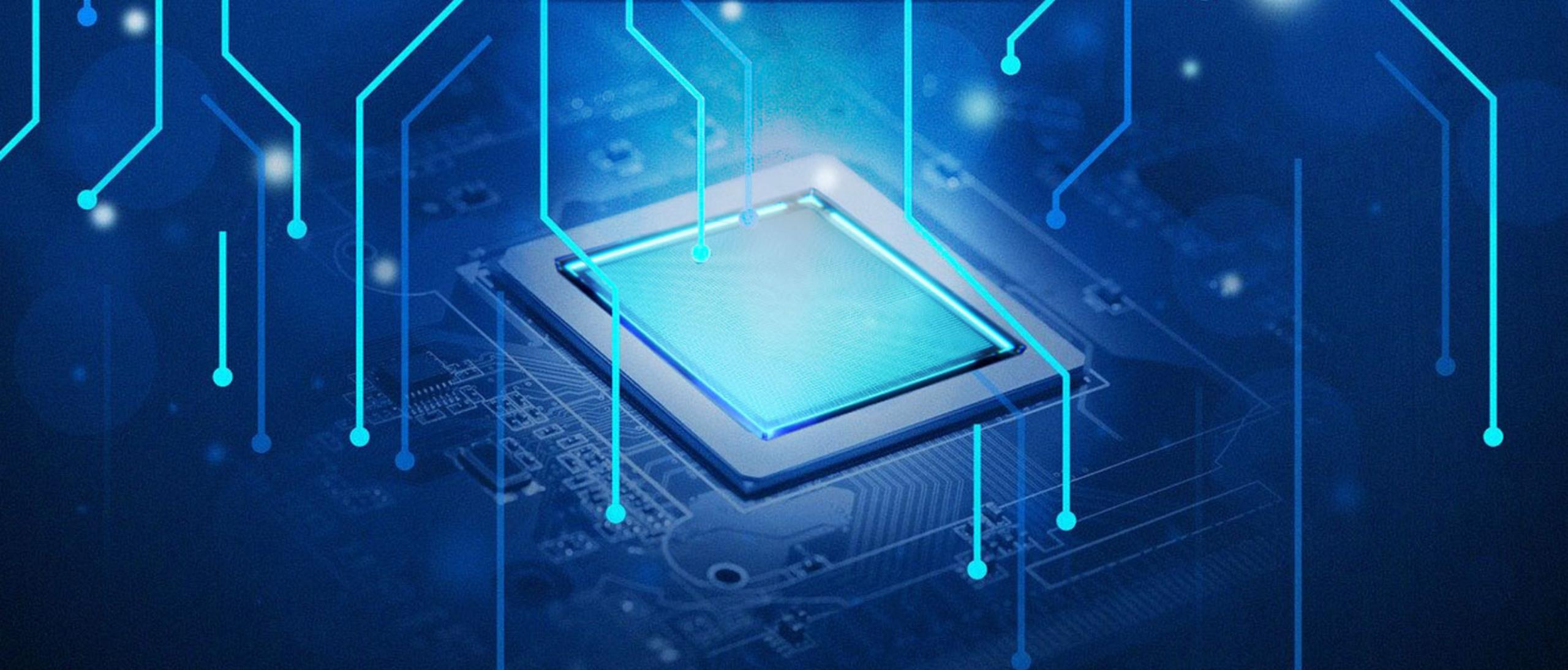


Ley de Moore

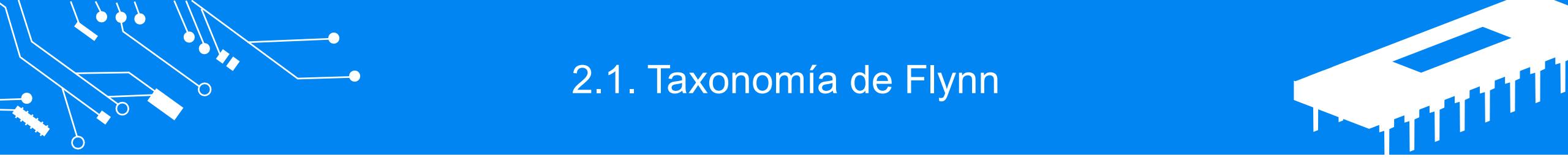
“El número de transistores por chip se doblará cada 2 años”



La única forma de mejorar energía-rendimiento-coste es la especialización



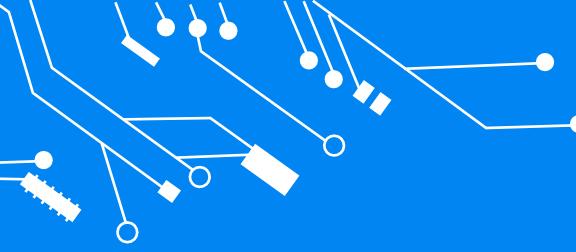
2.- Clasificación de Arquitecturas



2.1. Taxonomía de Flynn

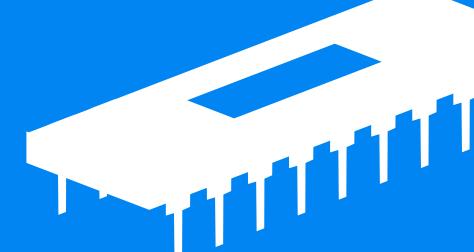
- **Clasificación de arquitecturas de computadores** propuesta por Michael J. Flynn en 1972 (Stanford University)
- Divide los computadores en 4 clases según el **número de flujos de instrucciones y flujos de datos** que pueden **procesarse simultáneamente**

	Single Data	Multiple Data
Single Instruction	SISD	SIMD
Multiple Instructions	MISD	MIMD



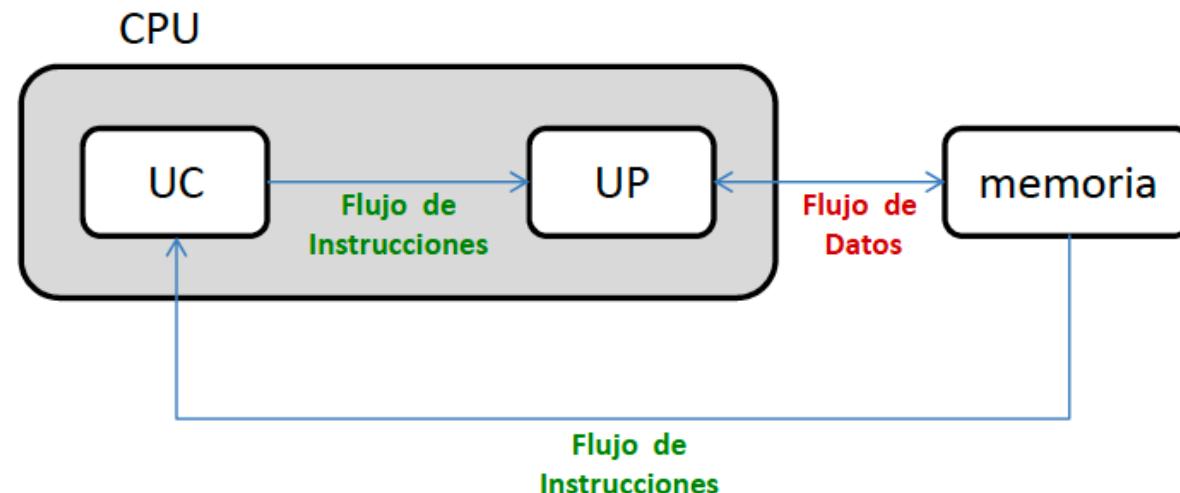
2.1. Taxonomía de Flynn

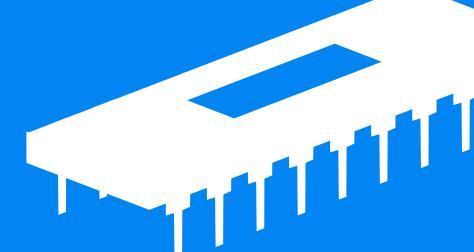
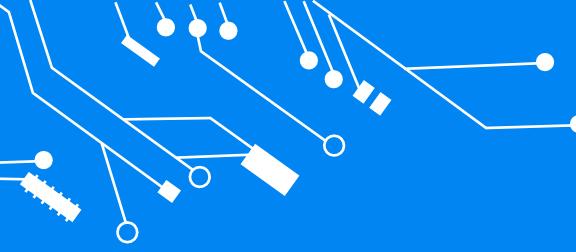
2.1.1 Computadores SISD



Características de un **computador SISD**:

- Sólo tiene **un** procesador (**CPU**)
- El procesador sólo tiene **una** unidad de control (**UC**)
- El procesador sólo tiene **una** unidad de procesamiento (**UP**)



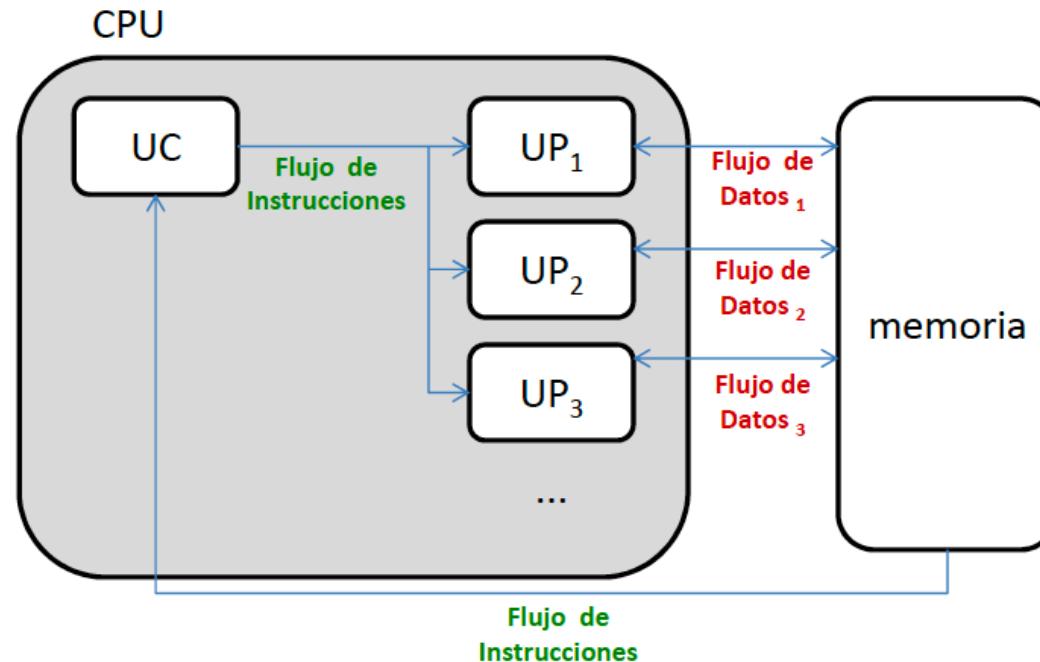


2.1. Taxonomía de Flynn

2.1.2 Computadores SIMD

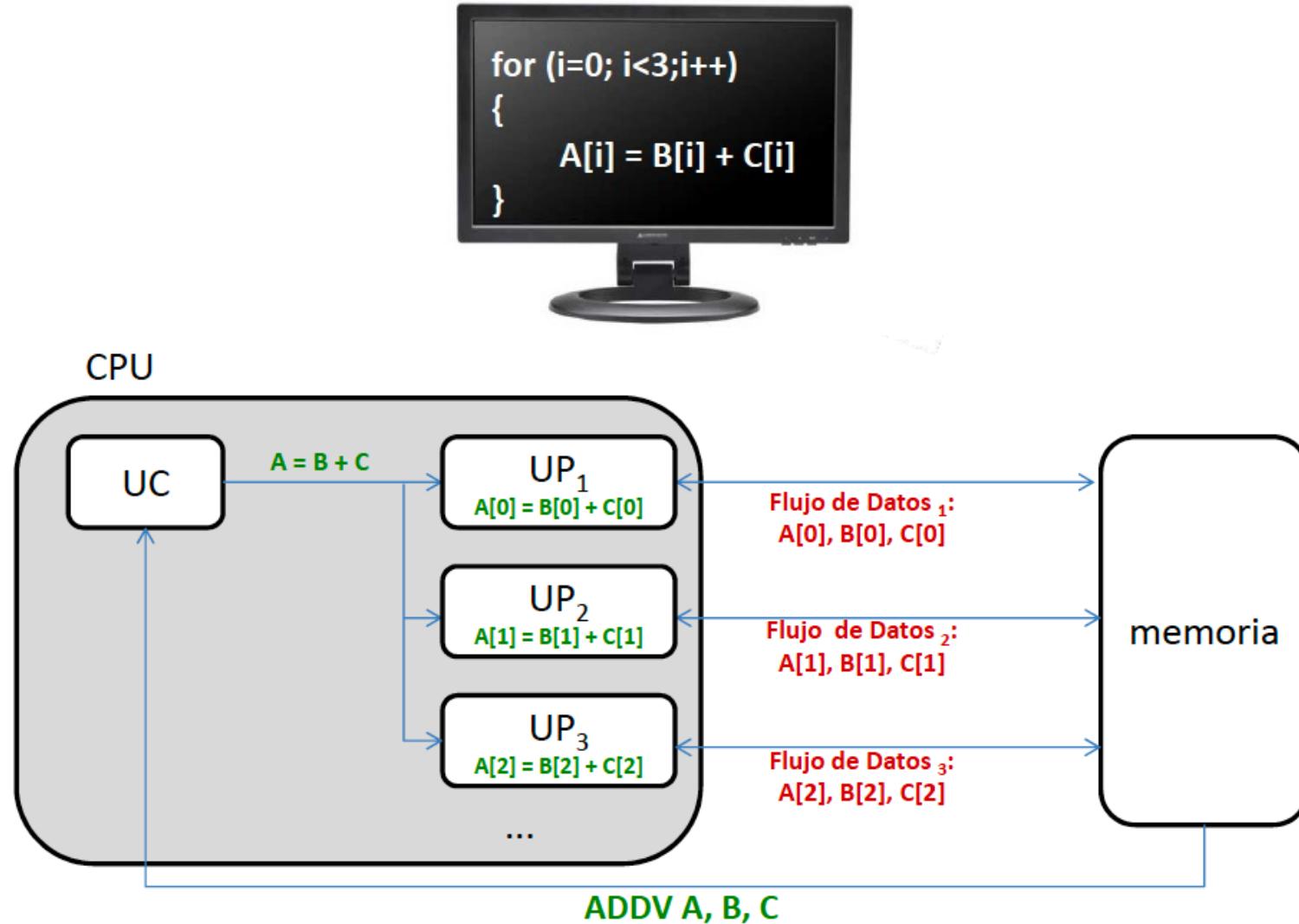
Características de un **computador SIMD**:

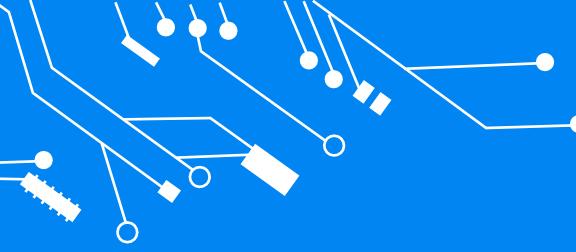
- Sólo tiene **un** procesador (**CPU**)
- El procesador sólo tiene **una** unidad de control (**UC**)
- El procesador tiene **múltiples** unidades de procesamiento (**UP**)



2.1. Taxonomía de Flynn

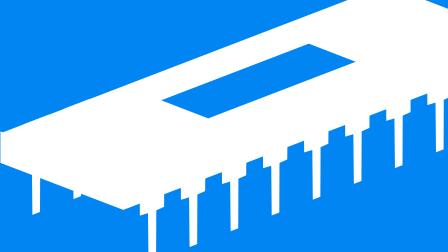
2.1.2 Computadores SIMD



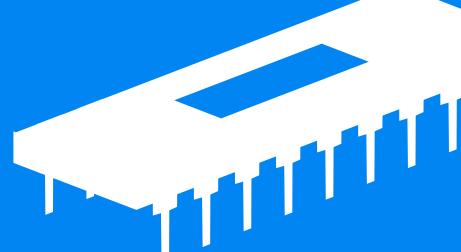
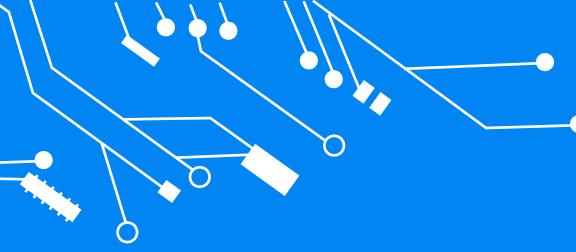


2.1. Taxonomía de Flynn

2.1.2 Computadores SIMD



- Un procesador SIMD es capaz de **ejecutar la misma instrucción sobre muchos datos de forma simultánea**
- Para ello, el procesador dispone de varias unidades de procesamiento (UP) y de un conjunto de **instrucciones vectoriales**
 - Ej. addv A, B, C suma los elementos del vector B con los elementos del vector C
- Una operación vectorial “equivale” a un **bucle completo** que procesa los N elementos del vector
- ¡OJO! Los **compiladores** para procesadores vectoriales analizan si las instrucciones situadas dentro de los bucles pueden ser ejecutadas en paralelo y genera código objeto con instrucciones vectoriales. Cuanto mejor vectorice el código, mejor será el rendimiento

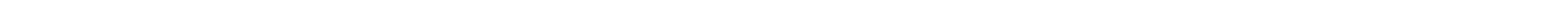
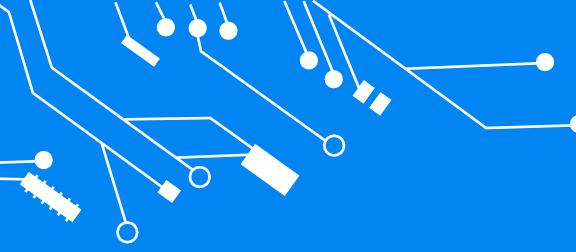


2.1. Taxonomía de Flynn

2.1.2 Computadores SIMD

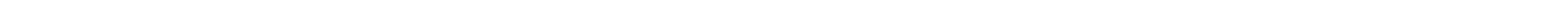
Tipos de computadores SIMD:

- Procesadores de **arrays**. (ej. El antiguo Cray-1 año 1976)
- Procesadores con **extensiones SIMD**, también denominadas extensiones multimedia. (ej. Algunos Pentium, núcleos del Corei7, etc)
- Procesadores **vectoriales** (ej. Procesador Cell año 2005)



2.1. Taxonomía de Flynn

2.1.3 Computadores MISD



Características de un **computador MISD**:

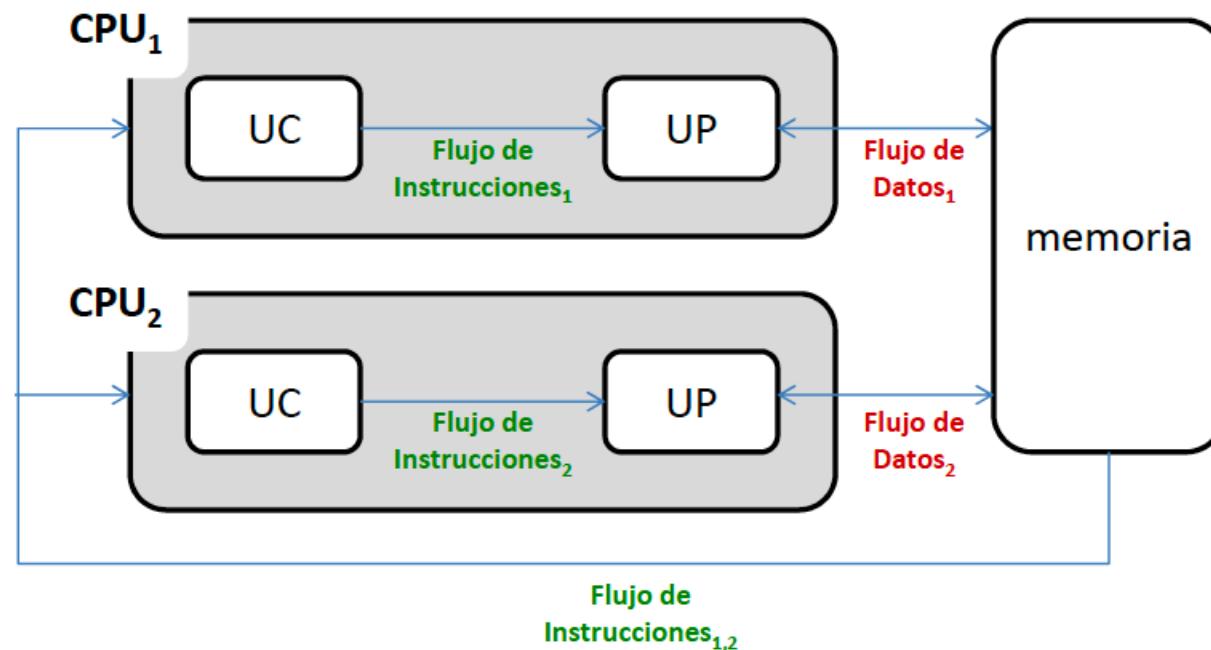
- Constituyen una clase de computadores cuyo comportamiento **se puede implementar con** una arquitectura **MIMD**
- Por el motivo anterior, **no existen** computadores MISD específicos en el **mercado**
- USO: Un ejemplo sería un conjunto de equipos que trata de factorizar un número primo muy grande utilizando diferentes algoritmos

2.1. Taxonomía de Flynn

2.1.4 Computadores MIMD

Características de un **computador MIMD**:

- Tiene **varios** procesadores (**CPU**)
- **Cada procesador** tienen su unidad de control (**UC**) y su unidad de procesamiento (**UP**)

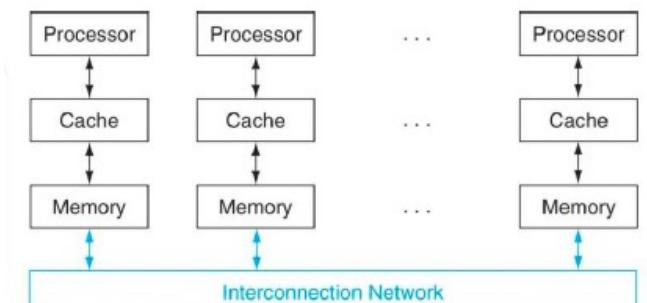
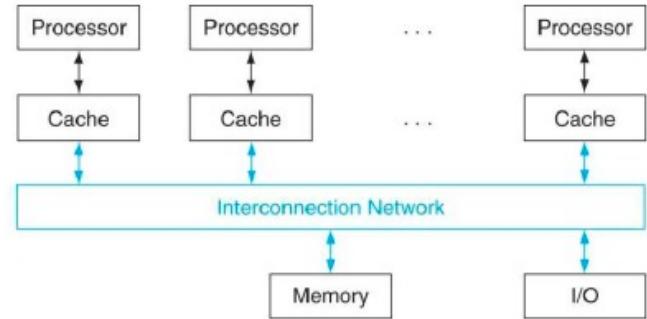


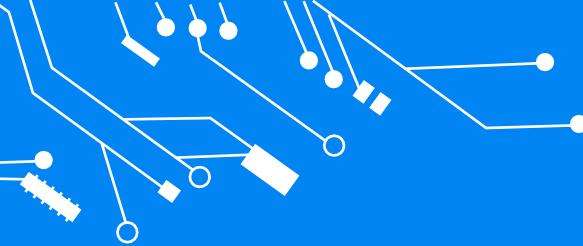
2.1. Taxonomía de Flynn

2.1.4 Computadores MIMD

Tipos de computadores MIMD:

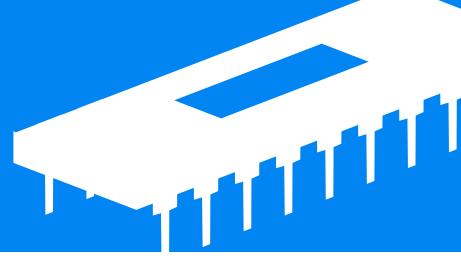
- **Memoria compartida (multiprocesadores)**
 - Existe un único **espacio de direcciones** que es **compartido por todos** los procesadores
 - Normalmente está formado por **una única computadora**
 - Ejemplo: **procesador multinúcleo**
- **Memoria distribuida (multicomputadores)**
 - **Cada procesador tiene su propio espacio de direcciones**
 - **Conjunto de computadoras** conectadas entre sí por una **red de comunicaciones**. Percibido por el usuario como un solo sistema
 - Ejemplos: **cluster, grid**





2.1. Taxonomía de Flynn

2.1.4 Computadores MIMD



¿Qué es una supercomputadora?

<http://en.wikipedia.org/wiki/Supercomputer>

¿Cuál fue la major supercomputadora en 2020?

<http://es.wikipedia.org/wiki/TOP500>

¿Qué es un GRID?

- http://es.wikipedia.org/wiki/Computaci%C3%B3n_grid
- http://en.wikipedia.org/wiki/Grid_computing
- <http://www.gridcomputing.com/>

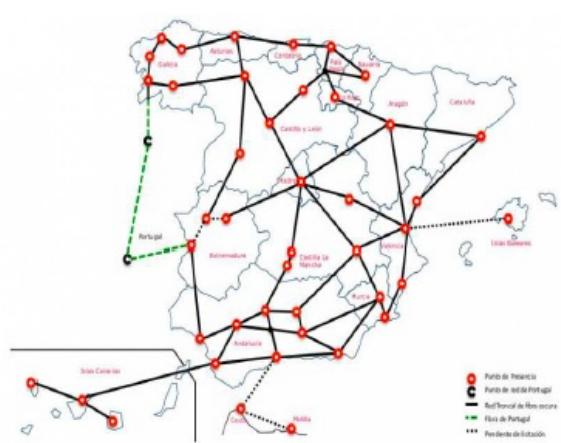
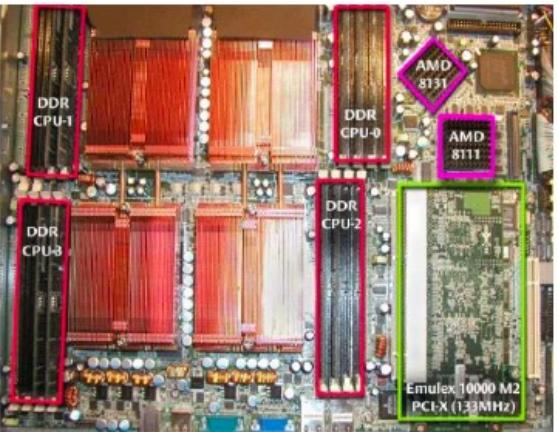


¿Hay alguna española en el TOP500?

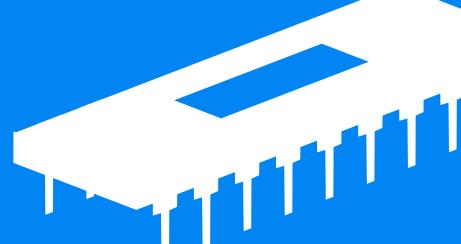
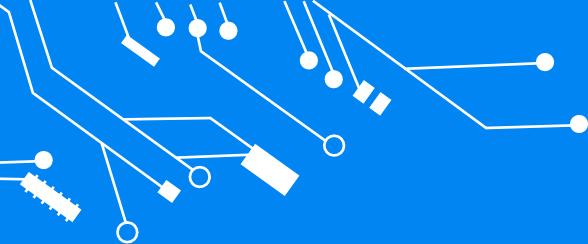
<http://www.top500.org/>

2.1. Taxonomía de Flynn

2.1.4 Computadores MIMD



El supercomputador de la UCA “CAI 2”
<http://supercomputacion.uca.es>



2.2. Paralelismo a nivel de instrucción (ILP)

Si la ejecución de una instrucción la dividimos (segmentamos) en varias etapas y conseguimos que cada etapa se lleve a cabo en una sección concreta del procesador sin interferir con las demás secciones,

¿Qué podríamos conseguir?

2.2. Paralelismo a nivel de instrucción (ILP)

Pipelining

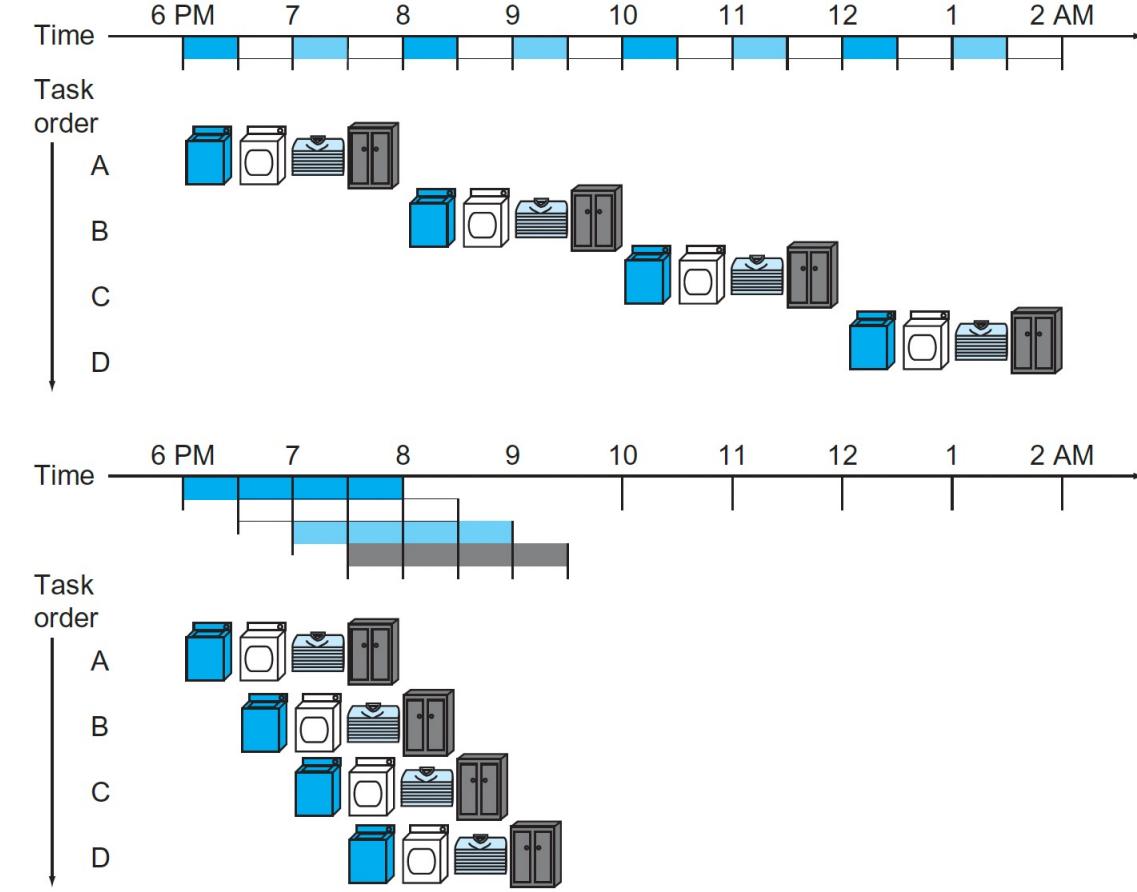
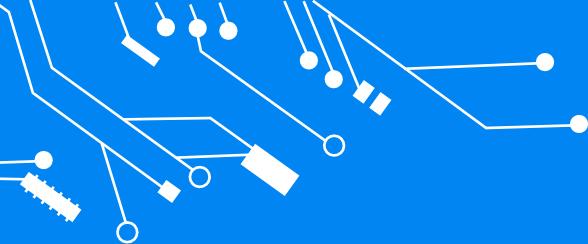
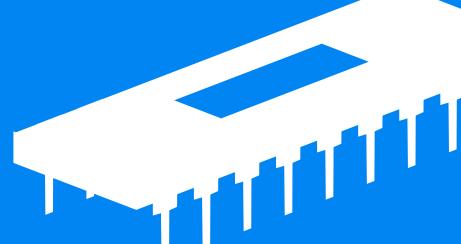


Diagrama extraído [Patt14]



2.2. Paralelismo a nivel de instrucción (ILP)



E T A P A S

IF (Instruction Fetch): captar de memoria la instrucción a ejecutar.

Camino a memoria

ID (Instruction Decode): decodificar la instrucción.

U.C.

EX (Execution): ejecución de la instrucción (calcular una dirección de memoria, realizar una operación aritmético/lógica con los operandos o calcular la dirección de un salto).

A.L.U.

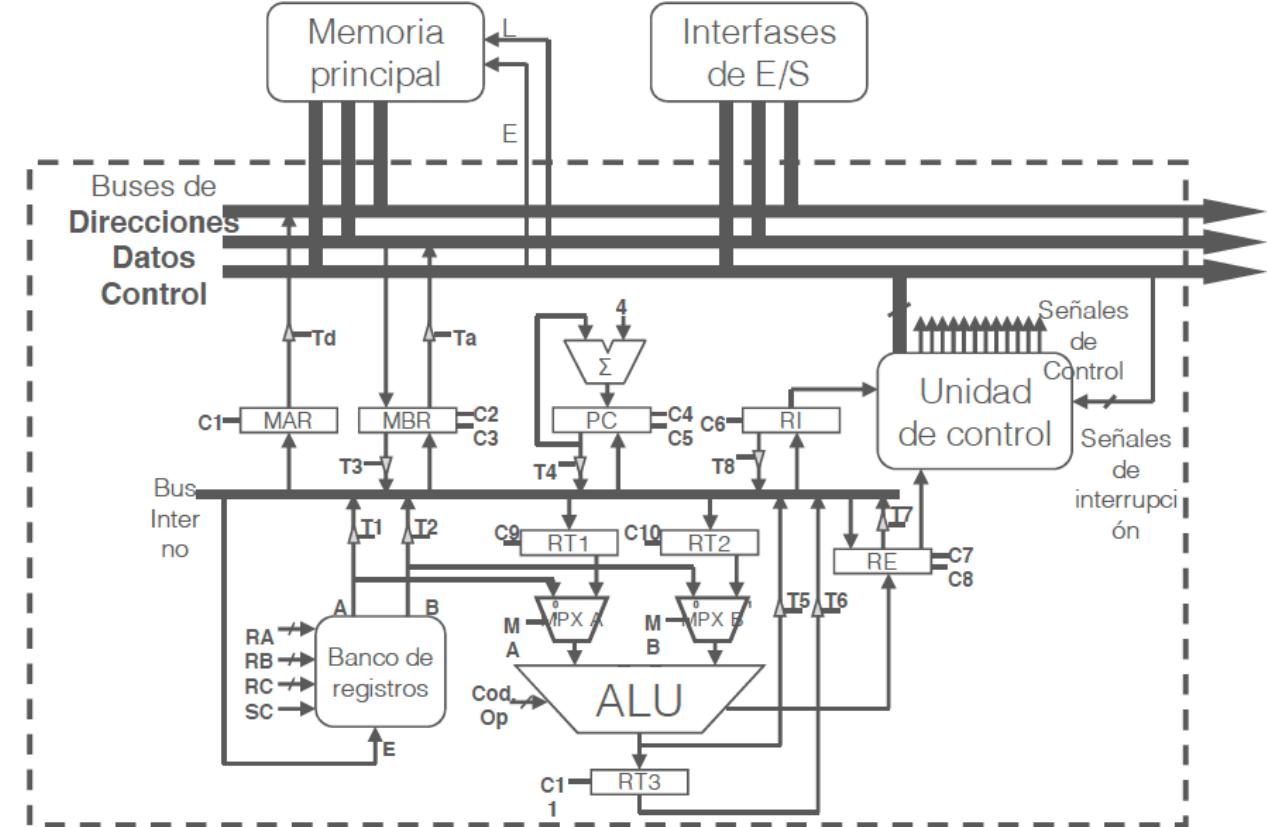
MEM (Memory): acceso a memoria para leer/escribir un dato (esta etapa sólo tiene lugar en instrucciones de carga/almacenamiento).

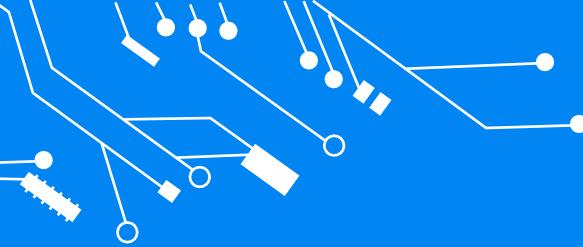
Camino a memoria

WB (WriteBack): escribir el resultado en los registros.

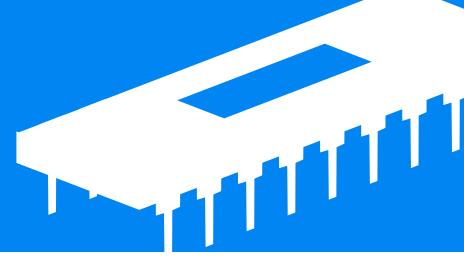
Camino a registros

2.2. Paralelismo a nivel de instrucción (ILP)





2.2. Paralelismo a nivel de instrucción (ILP)



- Para conseguir el paralelismo a nivel de instrucción hay que implementar en el procesador un camino de datos segmentado (**pipeline**).
- El camino de datos se segmenta en “zonas”. Cada zona se dedica a ejecutar una etapa concreta de la instrucción.



Instrucción 1

Instrucción 2

Instrucción 3

Instrucción 4

Instrucción 5

Instrucción 6

...

2.2. Paralelismo a nivel de instrucción (ILP)

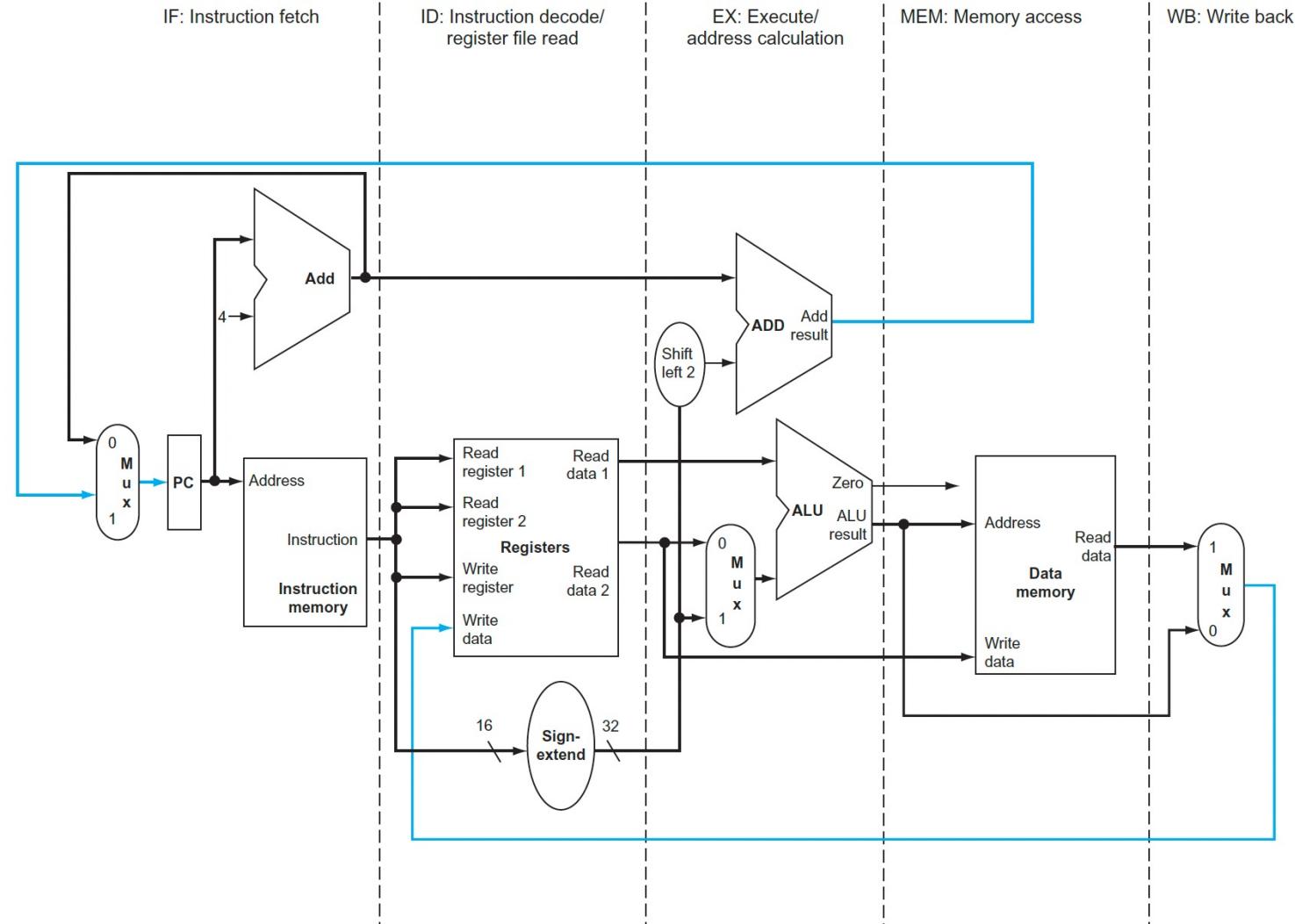
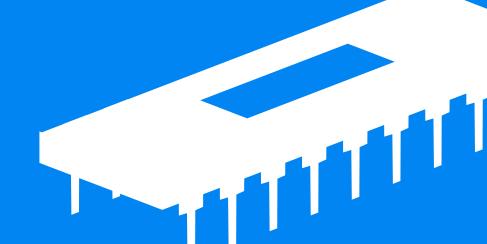
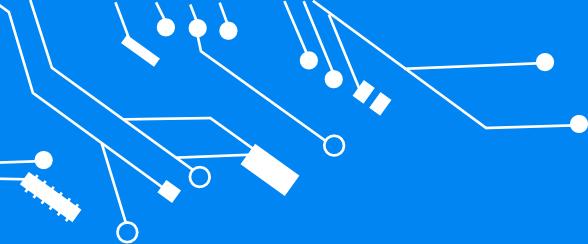


Diagrama extraído [Patt14]



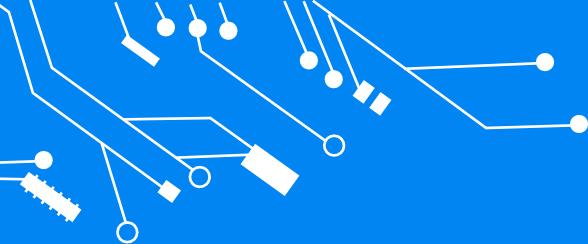
2.2. Paralelismo a nivel de instrucción (ILP)

Ejemplo de dependencia en el pipeline

$$\begin{aligned} a &= b + e; \\ c &= b + f; \end{aligned}$$

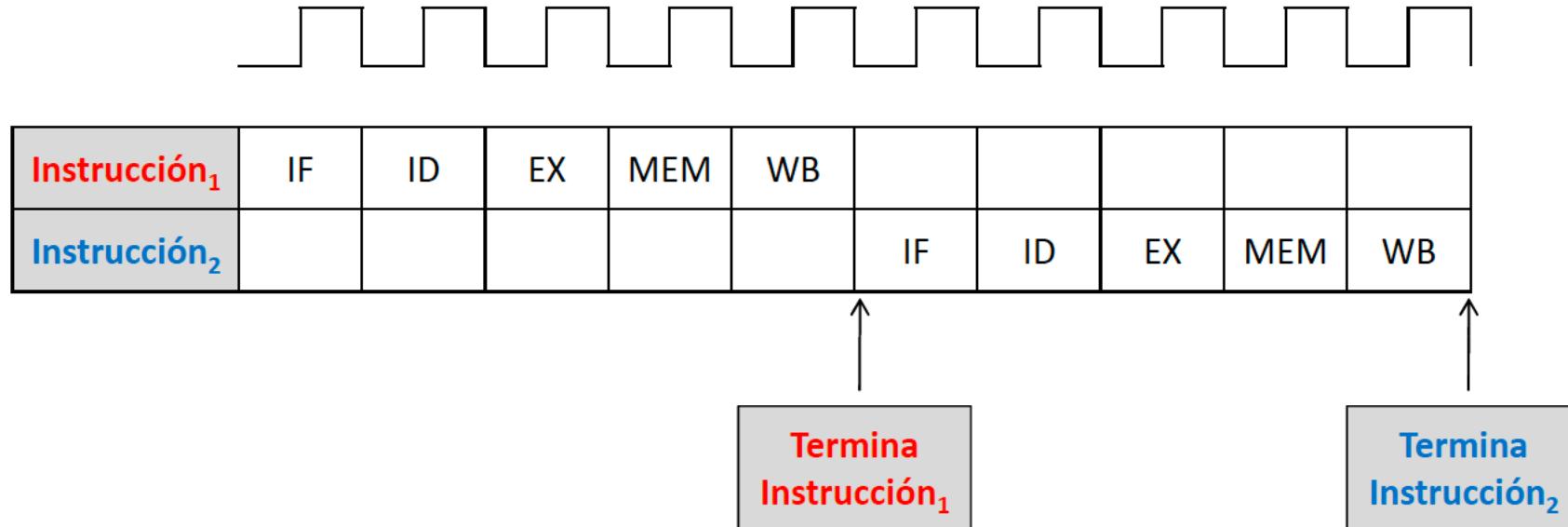
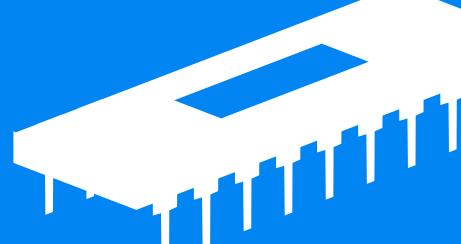
lw	\$t1, 0(\$t0)
lw	\$t2, 4(\$t0)
add	\$t3, \$t1,\$t2
sw	\$t3, 12(\$t0)
lw	\$t4, 8(\$t0)
add	\$t5, \$t1,\$t4
sw	\$t5, 16(\$t0)

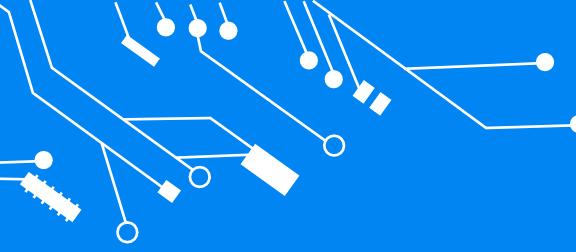
lw	\$t1, 0(\$t0)
lw	\$t2, 4(\$t0)
lw	\$t4, 8(\$t0)
add	\$t3, \$t1,\$t2
sw	\$t3, 12(\$t0)
add	\$t5, \$t1,\$t4
sw	\$t5, 16(\$t0)



2.2. Paralelismo a nivel de instrucción (ILP)

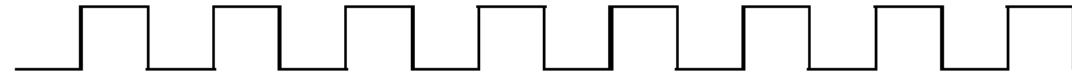
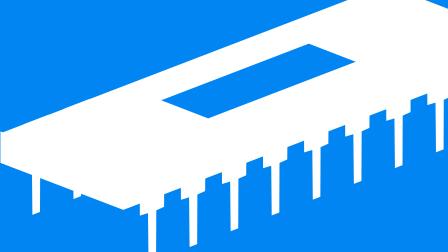
2.2.1 Procesador no segmentado



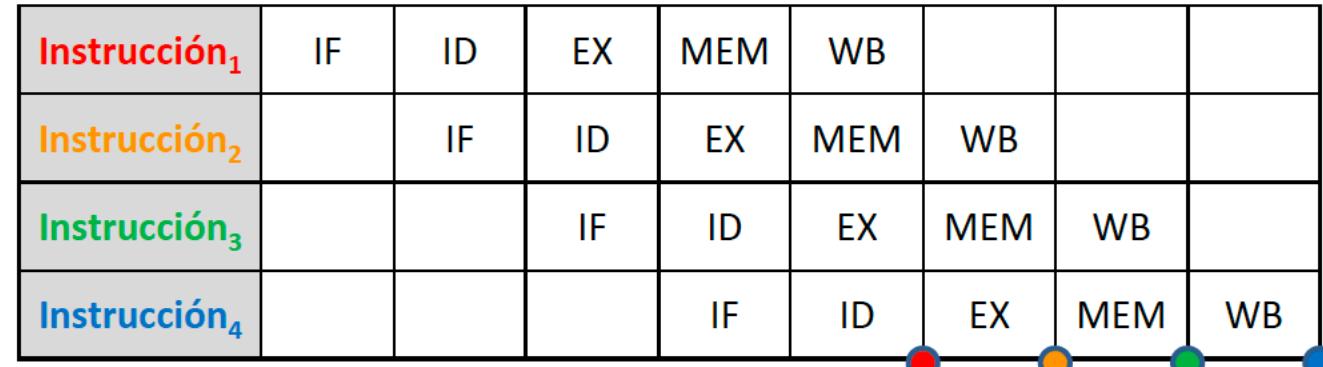


2.2. Paralelismo a nivel de instrucción (ILP)

2.2.2 Procesador segmentado



Instrucción ₁	IF	ID	EX	MEM	WB			
Instrucción ₂		IF	ID	EX	MEM	WB		
Instrucción ₃			IF	ID	EX	MEM	WB	
Instrucción ₄				IF	ID	EX	MEM	WB

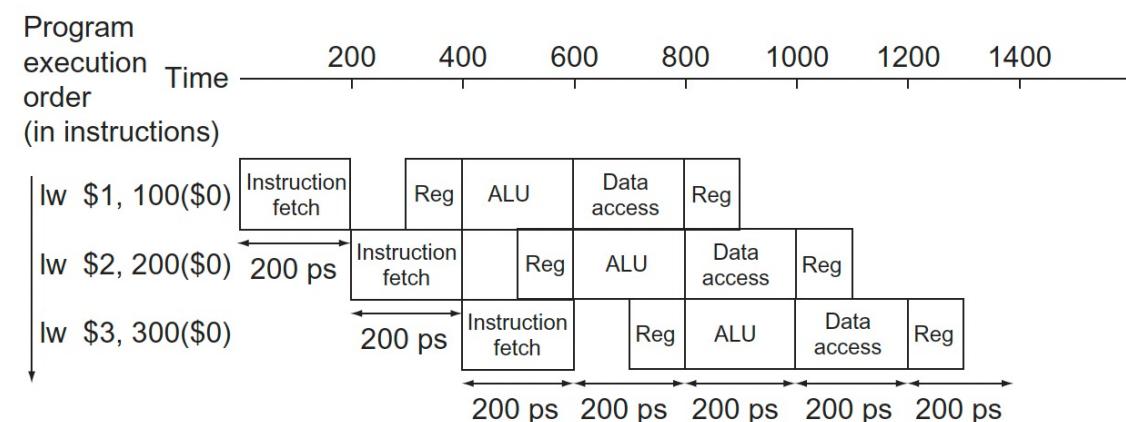
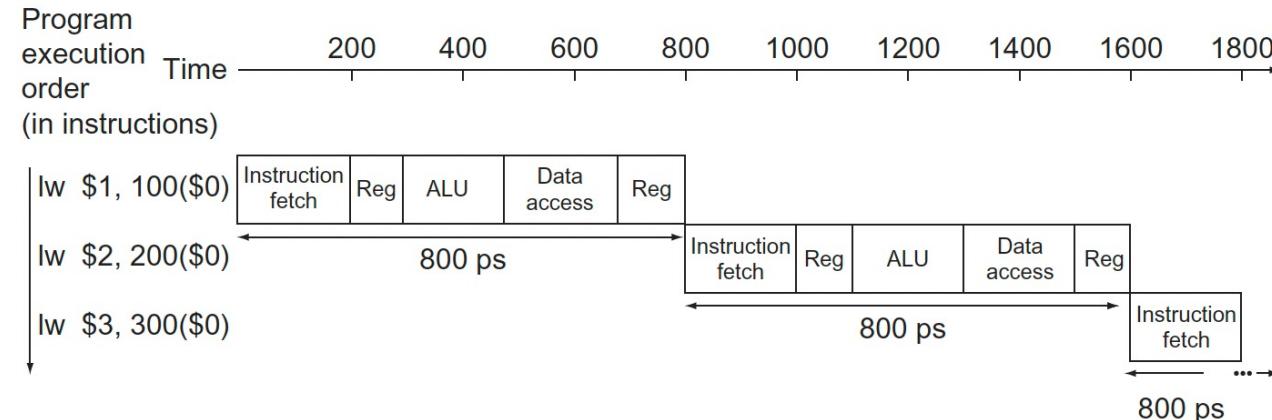


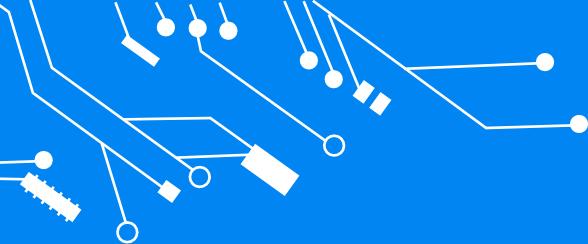
Termina
Instrucción₁

2.2. Paralelismo a nivel de instrucción (ILP)

2.2.2 Procesador segmentado

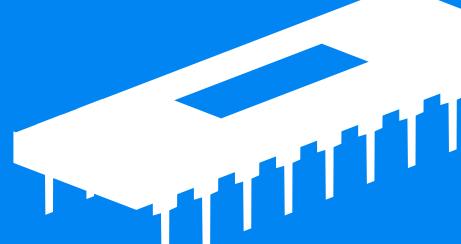
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps



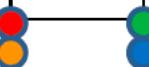


2.2. Paralelismo a nivel de instrucción (ILP)

2.2.3 Procesador superescalar



Instrucción ₁	IF	ID	EX	MEM	WB	
Instrucción ₂	IF	ID	EX	MEM	WB	
Instrucción ₃		IF	ID	EX	MEM	WB
Instrucción ₄		IF	ID	EX	MEM	WB



A diagram illustrating the execution of four instructions (Instrucción₁ to Instrucción₄) through six stages: IF, ID, EX, MEM, WB, and a final stage. The final stage contains three colored circles (red, orange, green) under the IF, ID, and EX columns, indicating parallel execution.

- En cada etapa se ejecuta más de una instrucción. El número máximo de instrucciones que se pueden ejecutar en una etapa se denomina **grado**.
- Para conseguir esta concurrencia hay que replicar elementos del procesador.
- En cada ciclo de reloj se inicia la ejecución de varias instrucciones.

2.2. Paralelismo a nivel de instrucción (ILP)

2.2.3 Procesador superescalar

Procesador superescalar

- Las instrucciones se captan y tratan individualmente

VS

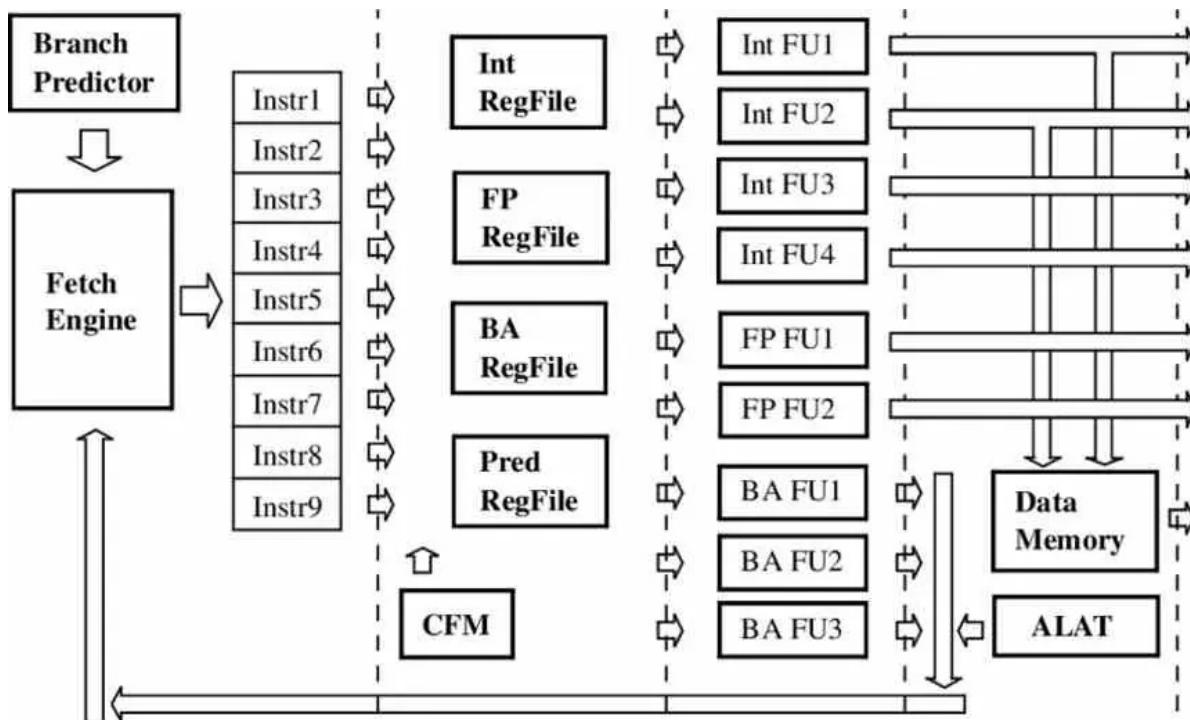


Figura. Ejemplo de procesador VLIW

Procesador VLIW

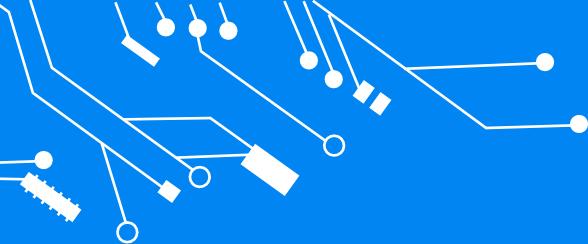
- Agrupa varias instrucciones en una sola y las envía en conjunto a la diferentes unidades que hay disponibles en el procesador
- Dependen del compilador para generar el binario

Ventajas

- Hardware de decodificación más simple
- Más espacio en el chip para unidades de ejecución
- Más espacio para colocar una mayor cantidad de registros

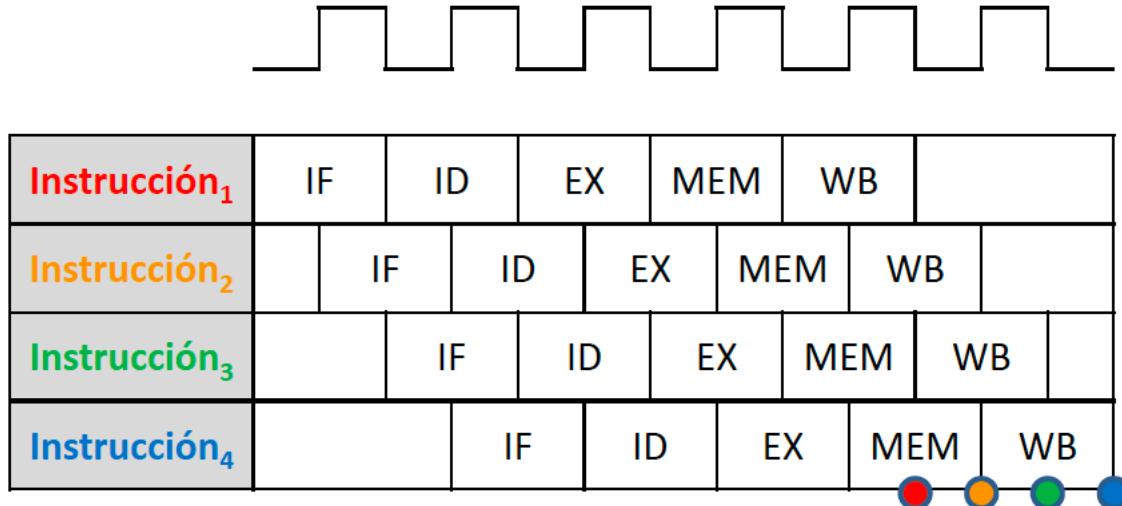
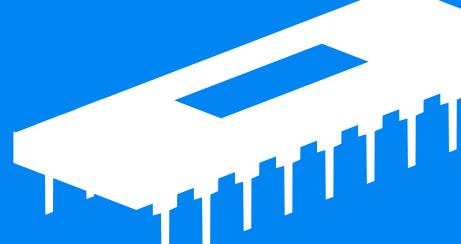
Desventajas

- Requiere un compilador más complejo
- Mayor desaprovechamiento de las unidades de ejecución



2.2. Paralelismo a nivel de instrucción (ILP)

2.2.4 Procesador supersegmentado



- Cada etapa se segmenta en varias subetapas, dando como resultado un pipeline de **numerosas etapas** (algunos procesadores tienen hasta 31 etapas como el Pentium 4 Prescott, año 2004).
- Como consecuencia, se puede ejecutar en paralelo más instrucciones de lo habitual sin tener que replicar elementos del procesador.

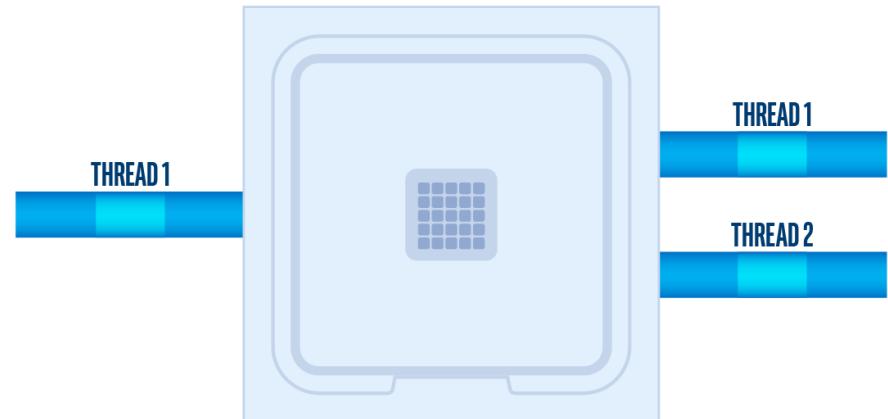
2.3. Paralelismo a nivel de instrucción (ILP) vs Paralelismo a nivel de procesos (PLP)

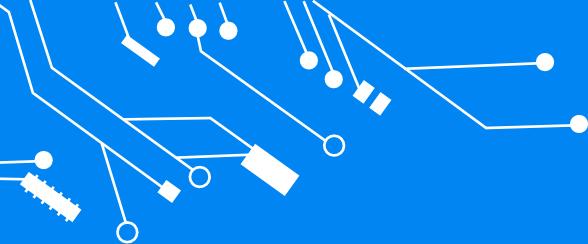
Multiproceso

- Forma de paralelización para procesamiento simultáneo
- Los programas con hilos dividen su trabajo en múltiples hilos de software
- Los hilos se procesan en paralelo por distintos núcleos de la CPU

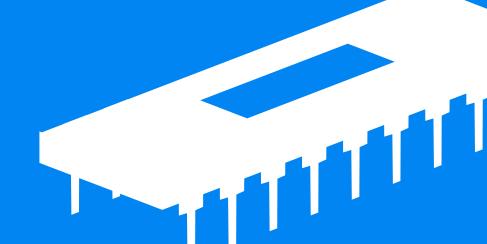
HyperThreading

- En cada núcleo se ejecuta más de un hilo
- La CPU expone 2 contextos de ejecución por núcleo físico
 - Un núcleo físico pasa a funcionar con 2 “núcleos lógicos”
- Mejor rendimiento al aprovechar tiempo de inactividad



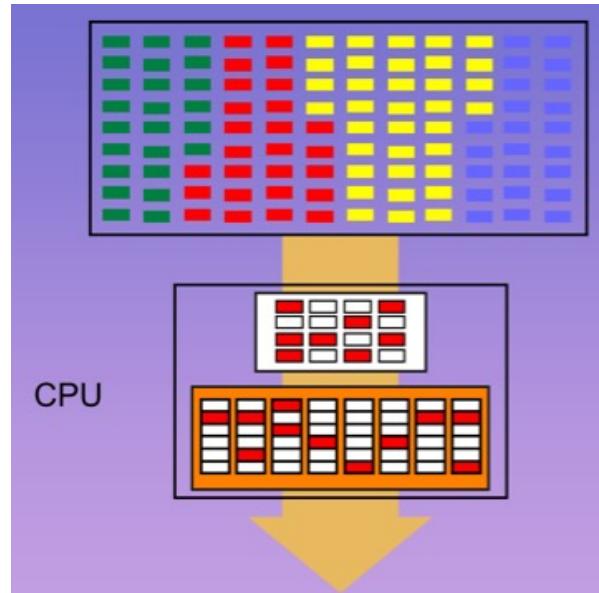


2.3. Paralelismo a nivel de instrucción (ILP) vs Paralelismo a nivel de procesos (PLP)



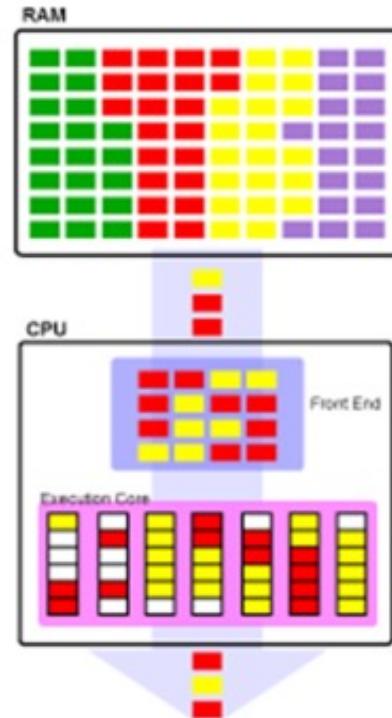
ILP

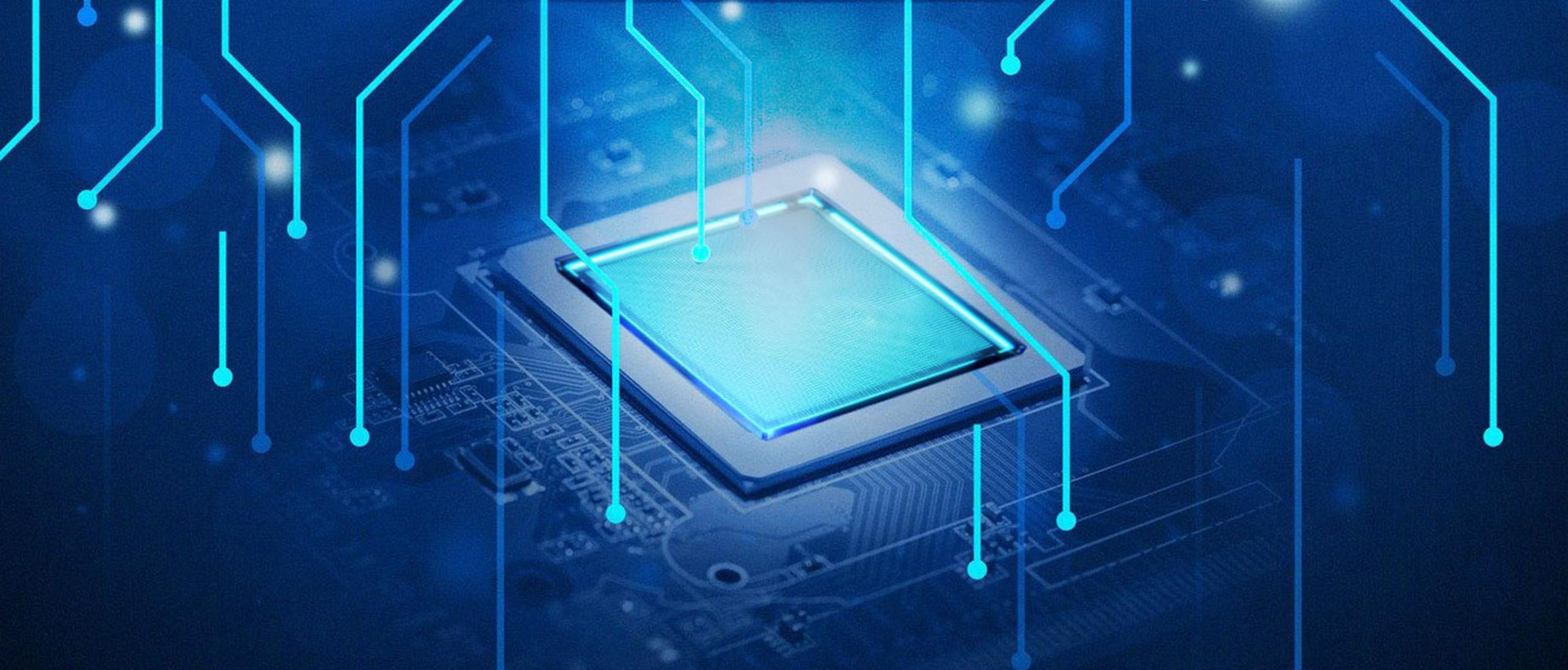
- Reducción de tiempo de ejecución de un programa solapando la ejecución de instrucciones
- Costoso
- Mayor complejidad en la lógica de control



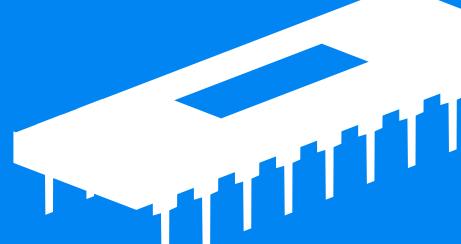
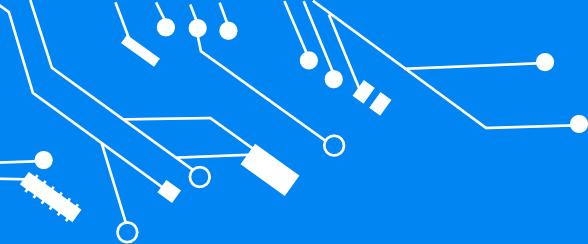
PLP

- Ejecución de programas independientes / distintas partes de un programa, simultáneamente utilizando diferentes fuentes de ejecución
- Mayor rendimiento al aprovechar tiempo de inactividad
- Requiere mayor complejidad hardware

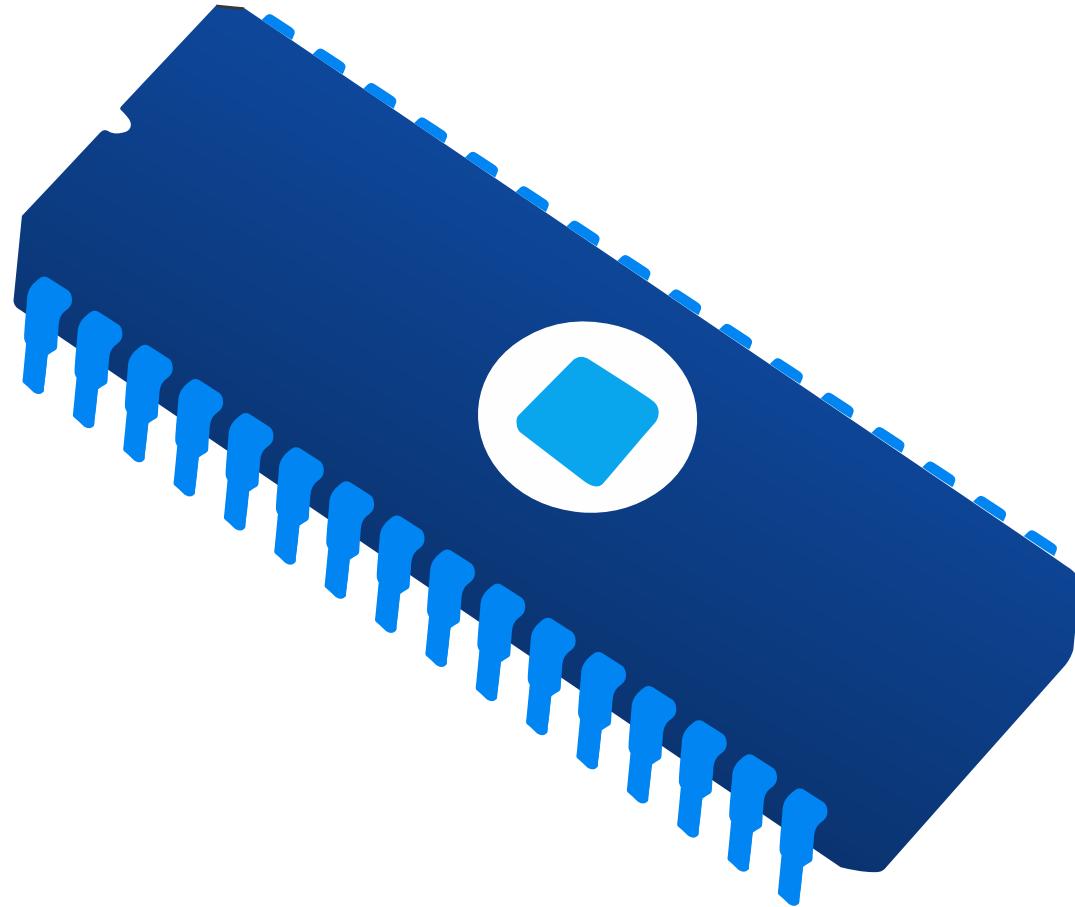




3.- Evaluación de prestaciones



3.1. Medidas de rendimiento



1. Tiempo de CPU
2. MIPS
3. FLOPS

3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU

- Tiempo que tarda la CPU en ejecutar un programa determinado.

Número total de instrucciones máquina que ejecuta el programa.

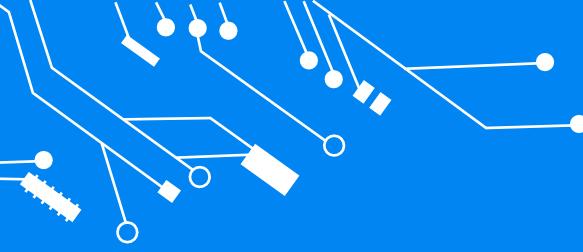
$$T_{CPU} = NI \times CPI_{global} \times T = \frac{NI \times CPI_{global}}{F}$$

Periodo del reloj del procesador, es decir, el tiempo que dura un ciclo.

Número medio de ciclos en los que se ejecuta una instrucción máquina del programa

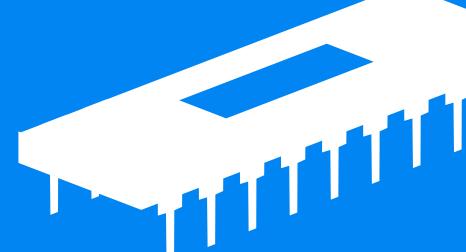
$$CPI_{global} = \frac{\sum_{i=1}^n NI_i \times CPI_i}{NI}$$

donde i identifica los distintos tipos de instrucción.



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



$$T_{CPU} = CPP \times T = \frac{CPP}{F}$$

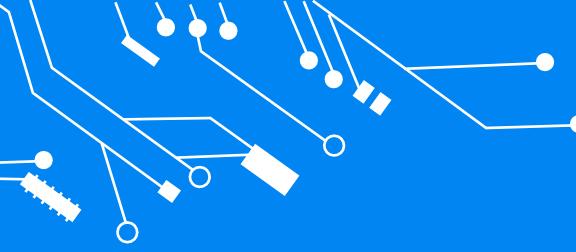
Periodo del reloj del procesador, es decir, el tiempo que dura un ciclo.

Número de ciclos que tarda en ejecutarse un programa.

Frecuencia

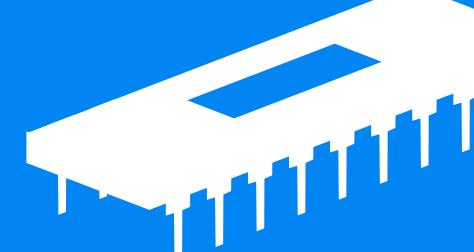
$$CPP = \sum_{i=1}^n NI_i \times CPI_i$$

donde i identifica los distintos tipos de instrucción.



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU

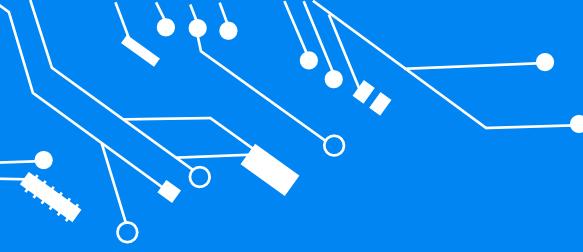


Prestaciones: relaciona las prestaciones y el tiempo de ejecución de un computador.

$$Prestaciones = \frac{1}{T_{CPU}}$$

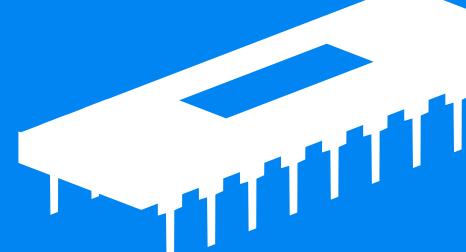
Relación de prestaciones: relaciona las prestaciones de dos máquinas diferentes.

$$\frac{Prestaciones_A}{Prestaciones_B} = \frac{T_{CPUB}}{T_{CPUA}} = n$$



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU

**Problema 1**

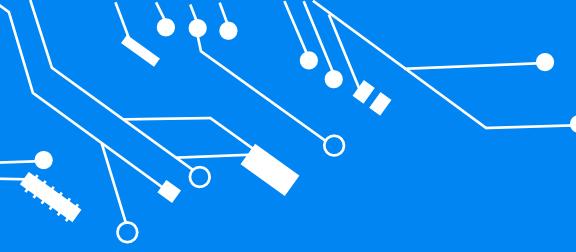
Un mismo programa se ejecuta en dos computadores diferentes,

1.- ¿Qué CPU ejecuta el programa más rápido?

2.- ¿Cuánto más rápida es?

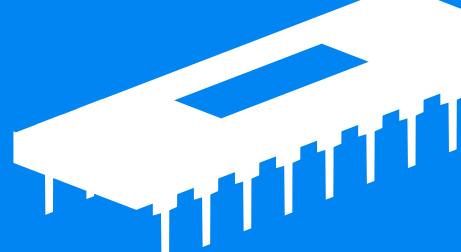
	T	CPI
CPU _A	250 ps	2
CPU _B	500 ps	1,2

Se supone que el programa ejecuta el mismo número de instrucciones máquina en las dos CPUs.



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



	T	CPI
CPU _A	250 ps	2
CPU _B	500 ps	1,2

Sabemos que cada máquina ejecuta el mismo número de instrucciones (I)

1.- *Calcular el número de ciclos de cada máquina*

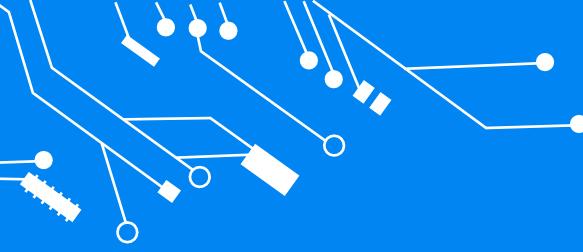
$$\text{Ciclos de reloj de } \text{CPU}_A = I \times 2.0$$

$$\text{Ciclos de reloj de } \text{CPU}_B = I \times 1.2$$

2.- *Calcular el tiempo de CPU para cada máquina*

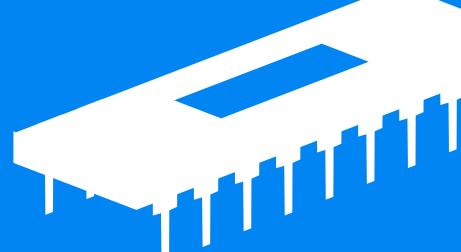
$$\text{Tiempo de } \text{CPU}_A = \text{Ciclos de reloj de } \text{CPU}_A \times \text{Tiempo de ciclo}_A = I \times 2.0 \times 250\text{ps} = 500 \times I\text{ps}$$

$$\text{Tiempo de } \text{CPU}_B = \text{Ciclos de reloj de } \text{CPU}_B \times \text{Tiempo de ciclo}_B = I \times 1.2 \times 500\text{ps} = 600 \times I\text{ps}$$



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



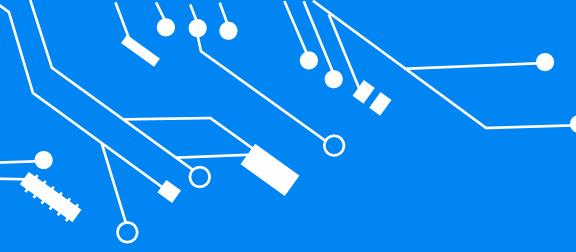
	T	CPI
CPU _A	250 ps	2
CPU _B	500 ps	1,2

$$\frac{\text{Prestaciones}_A}{\text{Prestaciones}_B} = \frac{T_{CPUB}}{T_{CPUA}} = n$$

$$\frac{\text{Prestaciones CPU}_A}{\text{Prestaciones CPU}_B} = \frac{\text{Tiempo ejecución}_B}{\text{Tiempo ejecución}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

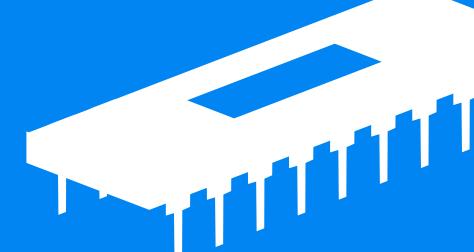
CONCLUSIÓN:

Para este programa, la máquina A es 1.2 veces más rápida que la máquina B



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



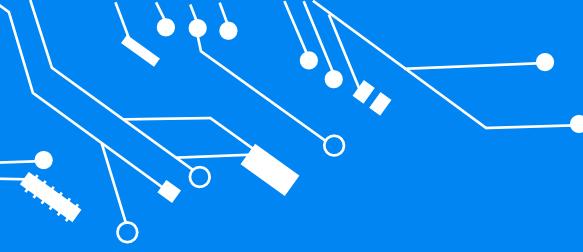
Problema 2

Un mismo programa se ejecuta en dos máquinas diferentes. Sabemos que la máquina B necesita 1,2 veces más ciclos de reloj que la máquina A para ejecutarlo.

1.- ¿Qué frecuencia de reloj tendría que tener la máquina B?

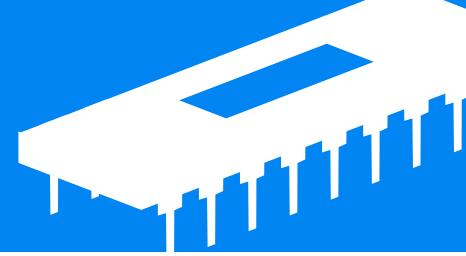
	F	T _{CPU}
CPU _A	2 GHz	10 s
CPU _B	?	6 s

-Problema obtenido de [PATT11]-



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



	F	T _{CPU}
CPU _A	2 GHz	10 s
CPU _B	?	6 s

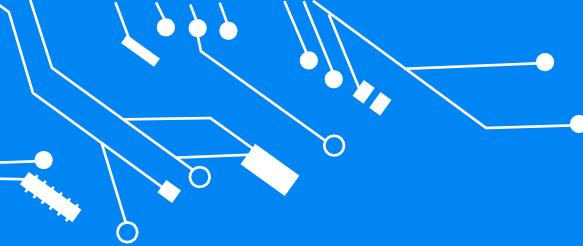
La máquina B necesita **1,2** veces más ciclos de reloj que A para ejecutar el programa

1.- *Calcular los ciclos de reloj requeridos por la máquina A*

$$\text{Tiempo de CPU}_A = \frac{\text{Ciclos de reloj de CPU}_A}{\text{Frecuencia de reloj}_A}$$

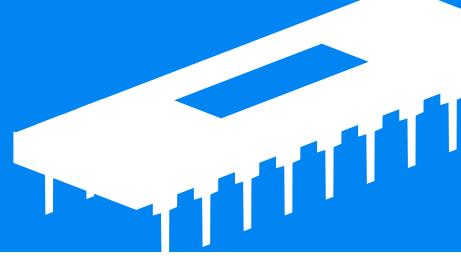
$$10 \text{ segundos} = \frac{\text{Ciclos de reloj de CPU}_A}{2 \times 10^9 \frac{\text{ciclos}}{\text{segundo}}}$$

$$\text{Ciclos de reloj de CPU}_A = 10 \text{ segundos} \times 2 \times 10^9 \frac{\text{ciclos}}{\text{segundo}} = 20 \times 10^9 \text{ ciclos}$$



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



Ciclos de reloj de CPU_A = 20×10^9 ciclos

	F	T _{CPU}
CPU _A	2 GHz	10 s
CPU _B	?	6 s

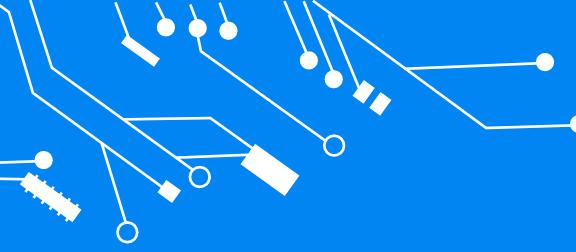
La máquina B necesita **1,2** veces más ciclos de reloj que A para ejecutar el programa

2.- *Calcular los ciclos de reloj requeridos por la máquina B*

$$\text{Tiempo de CPU}_B = \frac{1.2 \times \text{Ciclos de reloj de CPU}_A}{\text{Frecuencia de reloj}_B}$$

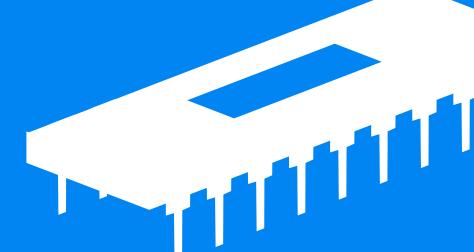
$$6 \text{ segundos} = \frac{1.2 \times 20 \times 10^9 \text{ ciclos}}{\text{Frecuencia de reloj}_B}$$

$$\text{Frecuencia de reloj}_B = \frac{1.2 \times 20 \times 10^9 \text{ ciclos}}{6 \text{ segundos}} = \frac{4 \times 10^9 \text{ ciclos}}{\text{segundo}} = 4 \text{GHz}$$



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



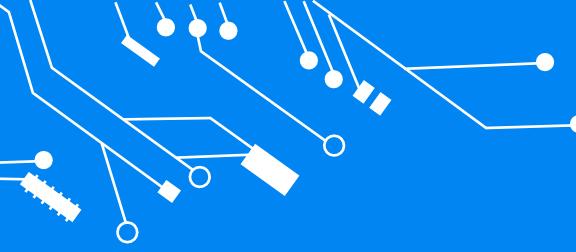
	F	T _{CPU}
CPU _A	2 GHz	10 s
CPU _B	?	6 s

La máquina B necesita **1,2** veces más ciclos de reloj que A para ejecutar el programa

$$\text{Frecuencia de reloj}_B = \frac{1.2 \times 20 \times 10^9 \text{ ciclos}}{6 \text{ segundos}} = \frac{4 \times 10^9 \text{ ciclos}}{\text{segundo}} = 4 \text{GHz}$$

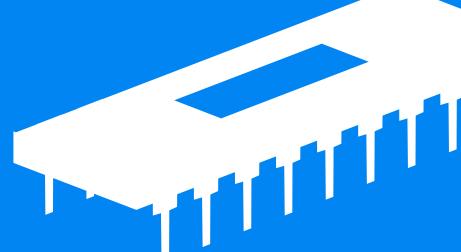
CONCLUSIÓN:

La **máquina B** necesitará el **doble de frecuencia de reloj** de la máquina A para ejecutar el programa en 6 segundos



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



Problema 3

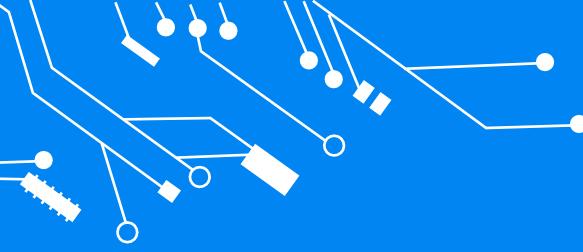
Un diseñador de compiladores tiene que decidir entre dos secuencias de código para una máquina en particular. Esta máquina tiene tres tipos de instrucciones máquina diferentes: A, B, C. Las secuencias requieren el siguiente número de instrucciones:

	A	B	C
Secuencia 1	2	1	2
Secuencia 2	4	1	1

Por otro lado, el CPI para cada tipo de instrucción es el siguiente:

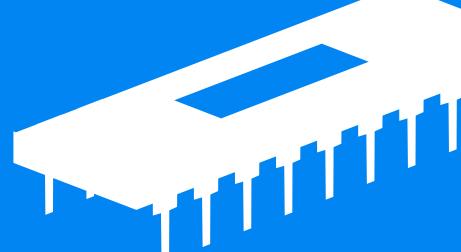
	A	B	C
CPI	1	2	3

- 1.- ¿Qué secuencia de código ejecuta mayor número de instrucciones?
- 2.- ¿Qué secuencia de código es más rápida?
- 3.- ¿Cuál es el CPI_{MEDIO} para cada secuencia?



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



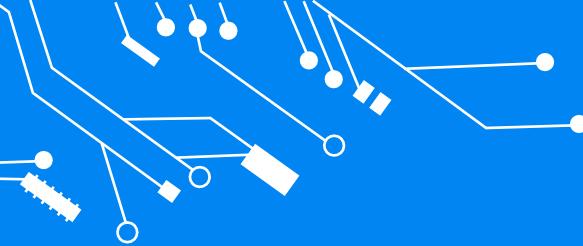
	A	B	C
Secuencia 1	2	1	2
Secuencia 2	4	1	1
	A	B	C
CPI	1	2	3

1.- *Calcular las instrucciones que se ejecutan en cada secuencia*

$$\text{Secuencia}_1 = 2 + 1 + 2 = 5$$

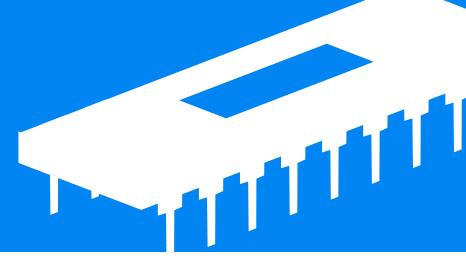
$$\text{Secuencia}_2 = 4 + 1 + 1 = 6$$

La secuencia 1 ejecuta el menor número de instrucciones



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU



$$\text{Secuencia}_1 = 2 + 1 + 2 = 5$$

$$\text{Secuencia}_2 = 4 + 1 + 1 = 6$$

	A	B	C
Secuencia ₁	2	1	2
Secuencia ₂	4	1	1
	A	B	C
CPI	1	2	3

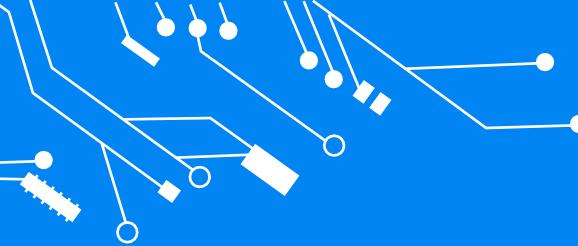
2.- Calcular el número total de ciclos cada secuencia

$$\text{Ciclos de reloj de la CPU} = \sum_{i=1}^n (CPI_i \times C_i)$$

$$\text{Ciclos de reloj de la CPU}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ ciclos}$$

$$\text{Ciclos de reloj de la CPU}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ ciclos}$$

La secuencia 2 es más rápida aunque ejecute una instrucción más, por lo que tiene que tener un CPI menor



3.1. Medidas de rendimiento

3.1.1. Tiempo de CPU

$$\text{Secuencia}_1 = 2 + 1 + 2 = 5$$

$$\text{Secuencia}_2 = 4 + 1 + 1 = 6$$

	A	B	C
Secuencia ₁	2	1	2
Secuencia ₂	4	1	1
	A	B	C
CPI	1	2	3

$$\text{Secuencia}_1 = 10 \text{ ciclos}$$

$$\text{Secuencia}_2 = 9 \text{ ciclos}$$

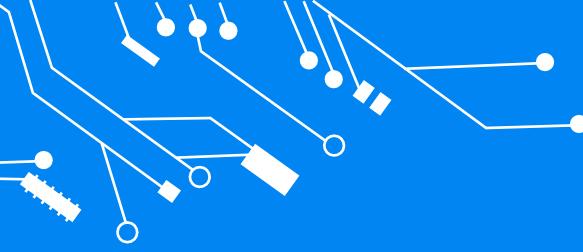
3.- Calcular los CPIs

$$CPI = \frac{\text{Ciclos de reloj de la CPU}}{\text{Número de instrucciones}}$$

$$CPI_1 = \frac{\text{Ciclos de reloj de la CPU}_1}{\text{Número de instrucciones}_1} = \frac{10}{5} = 2$$

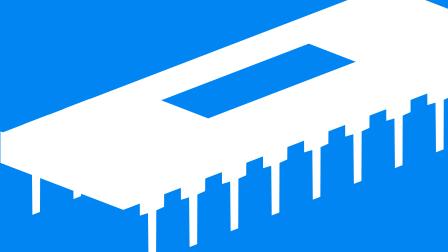
$$CPI_2 = \frac{\text{Ciclos de reloj de la CPU}_2}{\text{Número de instrucciones}_2} = \frac{9}{6} = 1.5$$

La secuencia 2 tiene una CPI menor



3.1. Medidas de rendimiento

3.1.2. MIPS



MIPS = Million Instructions Per Second

$$MIPS = \frac{NI}{T_{CPU} \times 10^6} = \frac{F}{CPI_{global} \times 10^6}$$

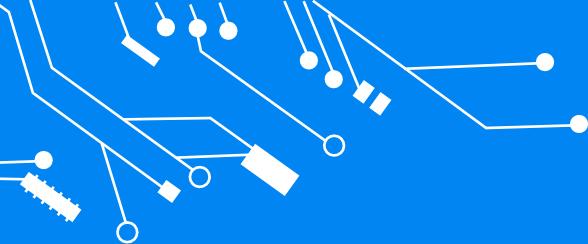
INCONVENIENTE:

Se desea contrastar un programa escrito en lenguaje de alto nivel en dos computadoras de **distinta arquitectura**, por ejemplo, una es MIPS y otra es x86.

En cada computadora se genera un ejecutable con **código máquina** muy **distinto**, específico de cada arquitectura. Decimos que es muy distinto en cuanto a número de instrucciones y número de ciclos que requiere cada instrucción para ejecutarse.

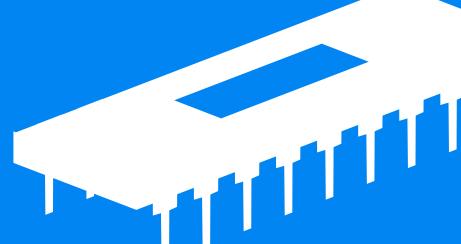
Con la medida MIPS, ¿siempre obtendremos resultados coherentes?

Por tanto, la medida **MIPS sólo** es útil para **comparar procesadores** con el **mismo juego de instrucciones**.



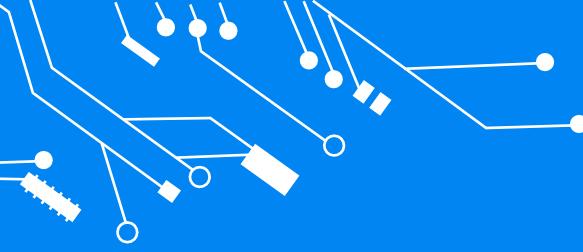
3.1. Medidas de rendimiento

3.1.2. MIPS



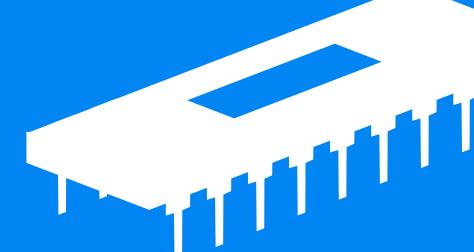
¿Podríamos utilizar el número de ladrillos como métrica para comparar tiempo de construcción de ambas casas?





3.1. Medidas de rendimiento

3.1.3. FLOPS



FLOPS = FLOATING POINT OPERATIONS PER SECOND

IDEA:

Un **programa en computadoras diferentes** puede ejecutar un **número distinto** de **instrucciones máquina**, pero el mismo número de operaciones en punto flotante.

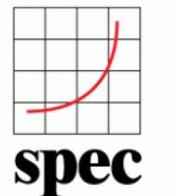
1. Sólo es aplicable a **operaciones en punto flotante**.
2. La **precisión** en punto flotante **no** es la **misma** en todas las máquinas.

Unidades
MFLOPS = 10^6 FLOPS
GFLOPS = 10^9 FLOPS
TFLOPS = 10^{12} FLOPS
PFLOPS = 10^{15} FLOPS

3.2. Benchmark

Herramientas **software** utilizadas para **medir el rendimiento** general del computador o de un componente concreto (CPU, RAM, tarjeta gráfica, etc)

1. **SPEC** (Standard Performance Evaluation Corporation)
Muy utilizado para evaluar **computadoras personales y servidores**.

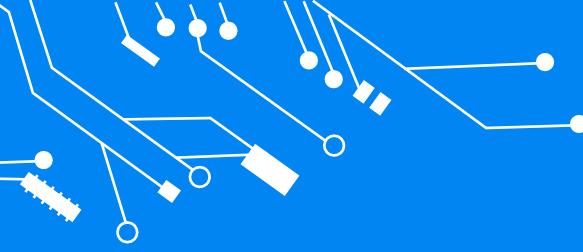


<http://www.spec.org/benchmarks.html>

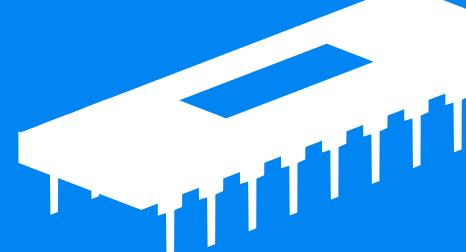
2. **Linpack**
Muy utilizado para evaluar **supercomputadoras** (TOP 500)



<http://www.top500.org/project/linpack>

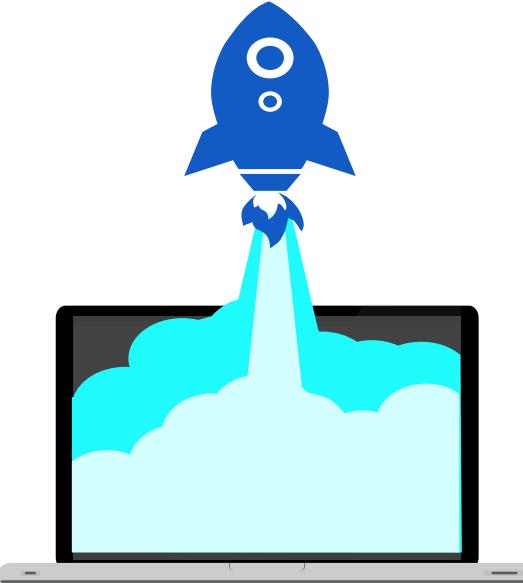


3.3. Ley de Amdahl

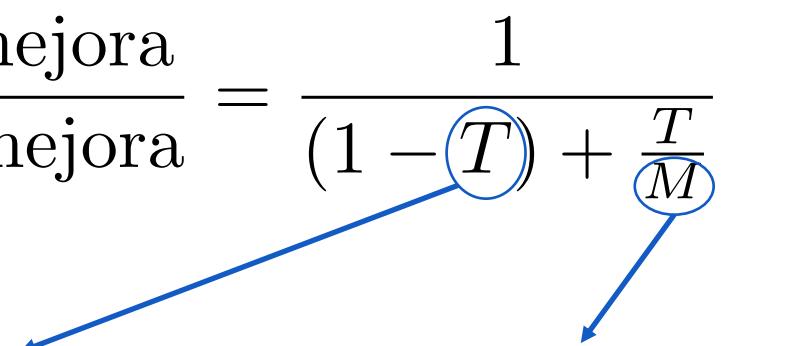


Mide el **incremento de rendimiento del computador** cuando uno de sus **componentes** ha sido **mejorado**.

$$\text{Incremento del rendimiento} = \frac{\text{Tiempo sin mejora}}{\text{Tiempo con mejora}} = \frac{1}{(1 - T) + \frac{T}{M}}$$



Porcentaje de tiempo que el computador utiliza el componente que se ha mejorado



Factor de mejora

3.3. Ley de Amdahl

Problema 4

Supongamos que un programa se ejecuta en 100 segundos en un computador dado, de los cuales 80 segundos se dedican a multiplicaciones.

¿Cuánto debo mejorar la velocidad de la multiplicación si quiero que se ejecute 5 veces más rápido?

$$T_{\text{ejecución tras mejoras}} = \frac{T_{\text{ejecución por la mejora}}}{\text{cantidad de mejora}} + T_{\text{ejecución no afectado}}$$

$$T_{\text{ejecución por la mejora}} = \frac{80 \text{ segundos}}{n} + (100 - 80 \text{ segundos})$$

Como queremos que la ejecución sea 5 veces más rápida el nuevo tiempo debe ser 20 segundos

3.3. Ley de Amdahl

Problema 4

Supongamos que un programa se ejecuta en 100 segundos en un computador dado, de los cuales 80 segundos se dedican a multiplicaciones.

¿Cuánto debo mejorar la velocidad de la multiplicación si quiero que se ejecute 5 veces más rápido?

$$20 \text{ segundos} = \frac{80 \text{ segundos}}{n} + 20 \text{ segundos}$$

$$0 \text{ segundos} = \frac{80 \text{ segundos}}{n}$$

NO es posible mejorar la multiplicación para alcanzar un aumento de prestaciones que implique una ejecución 5 veces más rápida, siempre y cuando la multiplicación suponga un 80% de la carga de trabajo

Bibliografía

Bibliografía

- Patterson, D. A., & Hennessy, J. L. (2018). *Estructura y diseño de computadores*. Reverté.
- Patterson, D. A., & Hennessy, J. L. (2019). *Computer Architecture and Design – A Quantitative Approach*. Morgan Kaufmann - Elsevier
- Patterson, D. A., & Hennessy, J. L. (2014). *Computer Organization and Design – The Hardware/Software Interface*. Morgan Kaufmann - Elsevier

Bibliografía

Links

- <https://www.intel.la/content/www/xl/es/support/articles/000005779/processors.html> (último acceso 09/21)
- <https://www.profesionalreview.com/2020/04/19/em64t-que-es-para-que-sirve/> (último acceso 09/21)
- <https://hardzone.es/reportajes/que-es/vliw/> (último acceso 09/21)
- <https://www.intel.es/content/www/es/es/gaming/resources/hyper-threading.html> (último acceso 09/21)
- <http://en.wikipedia.org/wiki/Supercomputer> (último acceso 09/21)
- http://es.wikipedia.org/wiki/Computaci%C3%B3n_grid (último acceso 09/21)
- http://en.wikipedia.org/wiki/Grid_computing (último acceso 09/21)
- <http://www.gridcomputing.com/> (último acceso 09/21)
- <http://es.wikipedia.org/wiki/TOP500> (último acceso 09/21)
- <http://www.top500.org/> (último acceso 09/21)
- <http://www.spec.org/benchmarks.html> (último acceso 10/21)
- <http://www.top500.org/project/lipack> (último acceso 10/21)