Universidad Nacional de Entre Ríos

Tecnicatura universitaria en desarrollo web

Procesadores de alta prestación

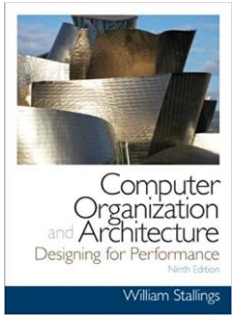
Semana 9 – Arquitectura de computadoras

Docente: José Ruiz {jose.ruiz@uner.edu.ar}

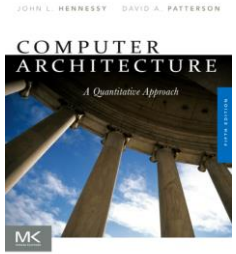
UNER virtual

Esta presentación esta basada en el libro de:

- William Stallings, Computer Organization and Architecture, 9th Edition, 2017.
  - Capítulo 16: “INSTRUCTION-LEVEL PARALLELISM”



- Hennessy-Patterson - Computer Architecture A Quantitative Approach (5th edition)
  - Capítulo 3: “ILP Instruction-Level Parallelism”



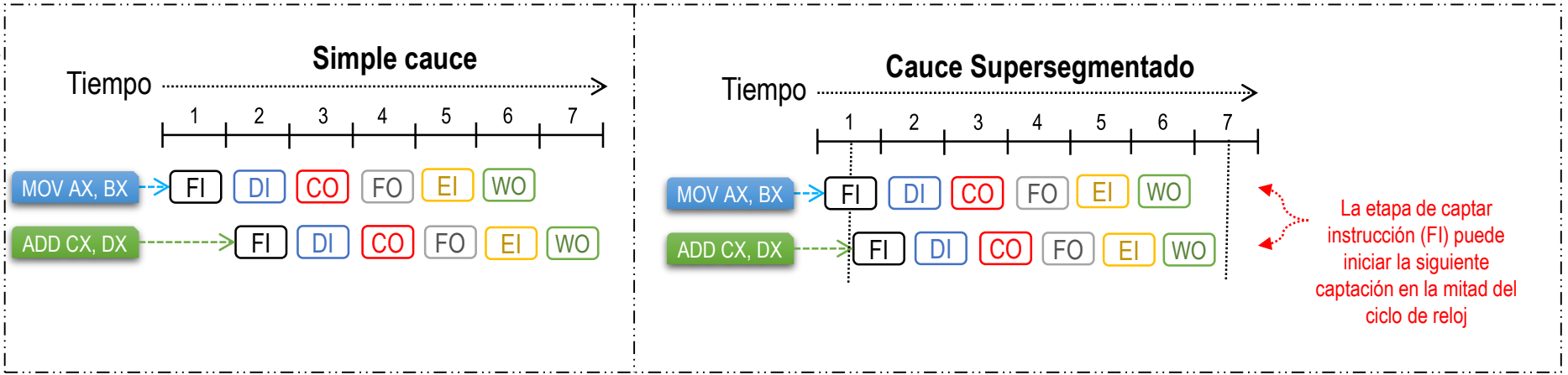
Archivos presentación y ejemplos se alojan en:

https://github.com/ruiz-jose/tudw-arq.git

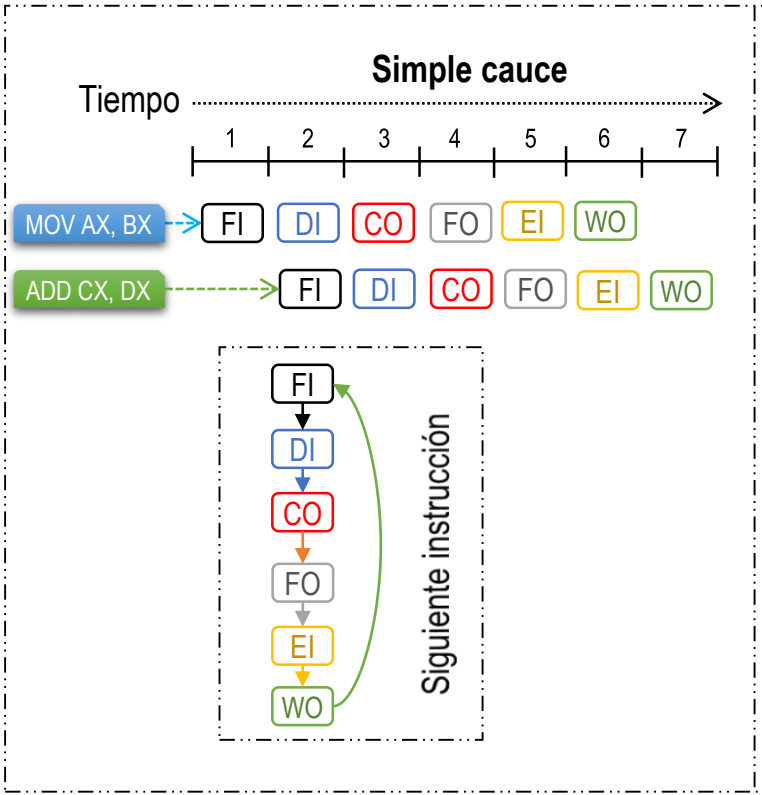
Agenda

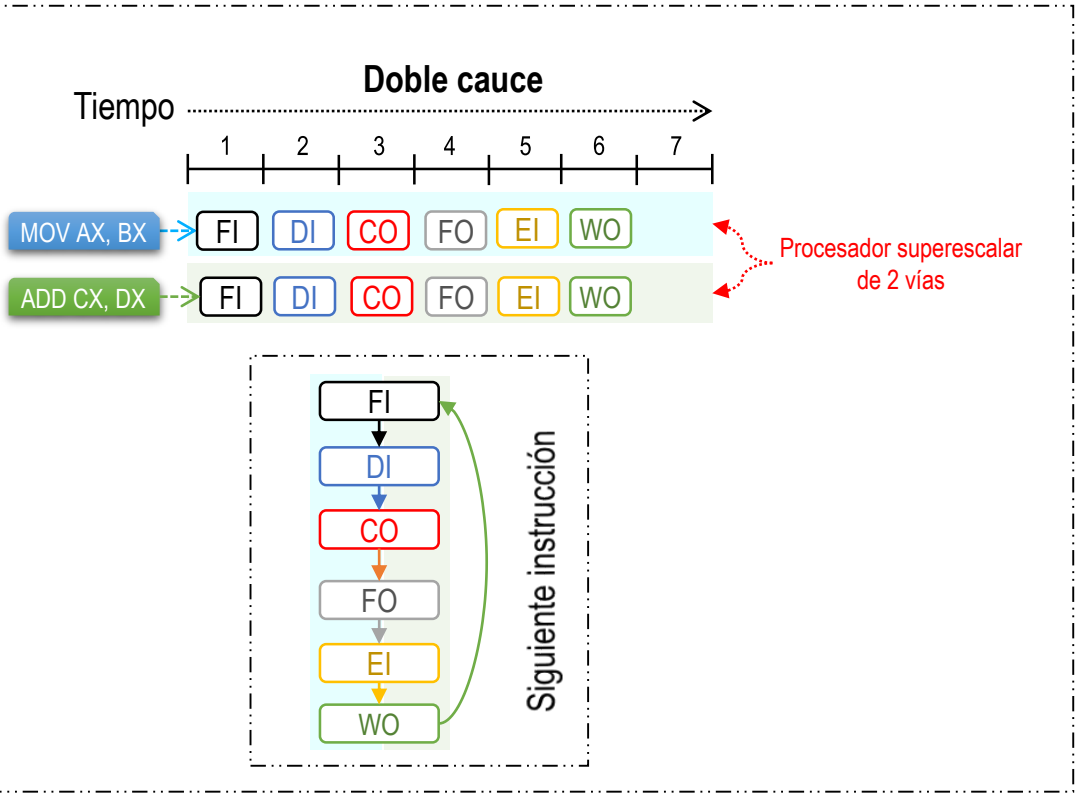
- Paralelismo a nivel de instrucciones (segmentación de instrucciones)
  - Supersegmentado
  - Superescalar
    - Planificación estática y dinámicas
    - Nuevos riesgos
    - Posibles soluciones (ejecución fuera de orden)
- Paralelismo a nivel de datos (SIMD)
- Paralelismo a nivel de procesos (Multi-core)

Existen etapas de la segmentación que no necesitan todo el ciclo de reloj, esas etapas se pueden subdividir en sub-etapas más pequeñas, que resulta en una mayor frecuencia del ciclo de reloj.

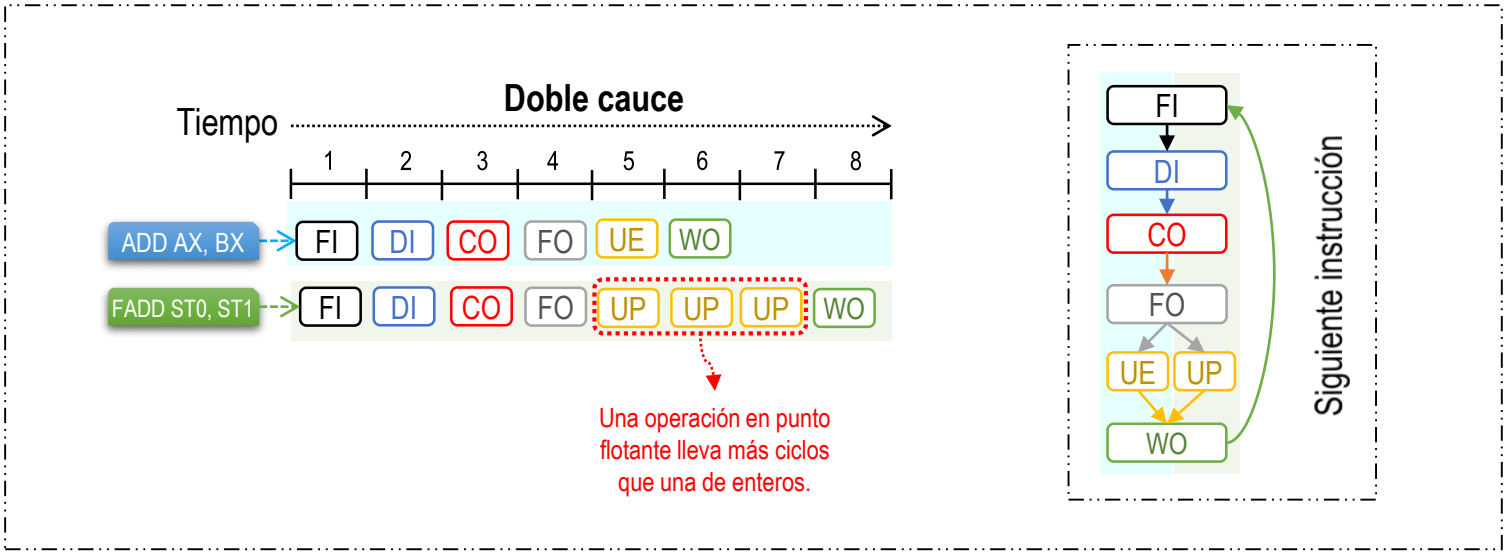


Un procesador superescalar duplica el cauce permitiendo ingresar dos instrucciones en el mismo ciclo de reloj, para ello debe ser capaz de captar y decodificar dos instrucciones a la vez y poseer unidades funcionales independientes.





Los CPU modernos poseen instrucciones para realizar operaciones de números enteros y en punto flotante independientes, para ello poseen unidades de ejecución de enteros (UE) y de punto flotante (UP), que también poseen sus propios conjuntos de registros. Según el tipo de instrucción la CPU utiliza un tipo de unidad de ejecución u otra.



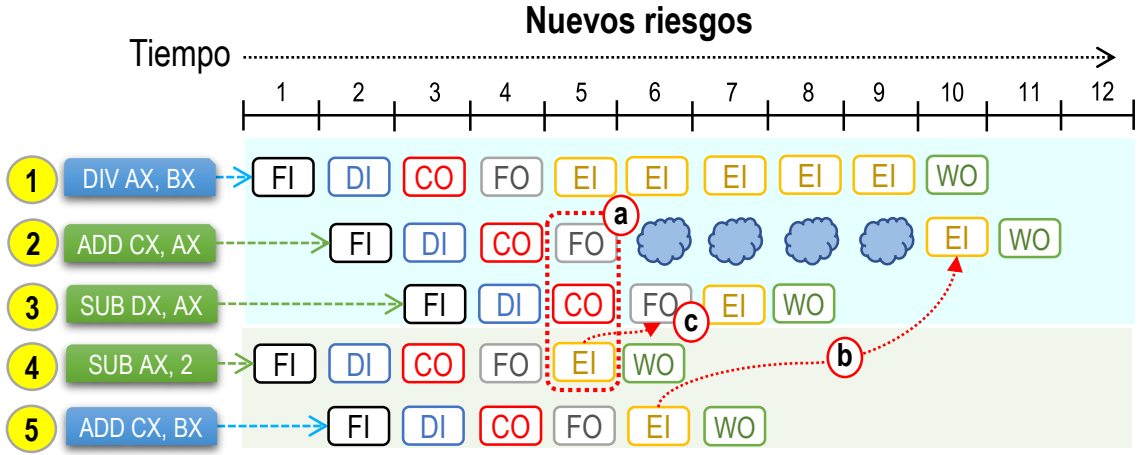
Superescalar – Nuevos riesgos

Aparecen nuevos riesgos entre cauces

a. En un cauce la instrucción 2 está leyendo el operando AX y en el otro cauce la instrucción 4 está escribiendo ese mismo operando, entonces, existe un conflicto estructural.

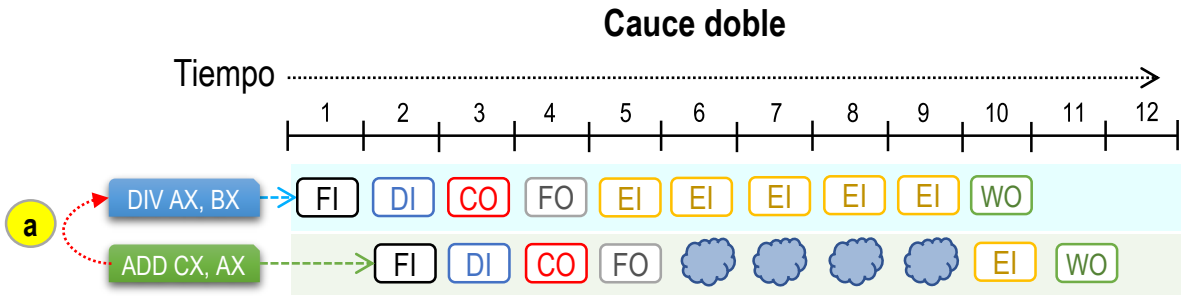
b. La instrucción 5 escribe su resultado en el destino de la instrucción 2, eliminando el resultado de la instrucción 5 y alterando la secuencia que el programador determino para el programa. (**Write After Write (WAW):** escribir después escribir)

c. La instrucción 4 escribe su resultado en un operando de la instrucción 3 antes de pueda leerlo. (**Write After Read (WAR):** escribir después leer)



Superescalar – planificación estática

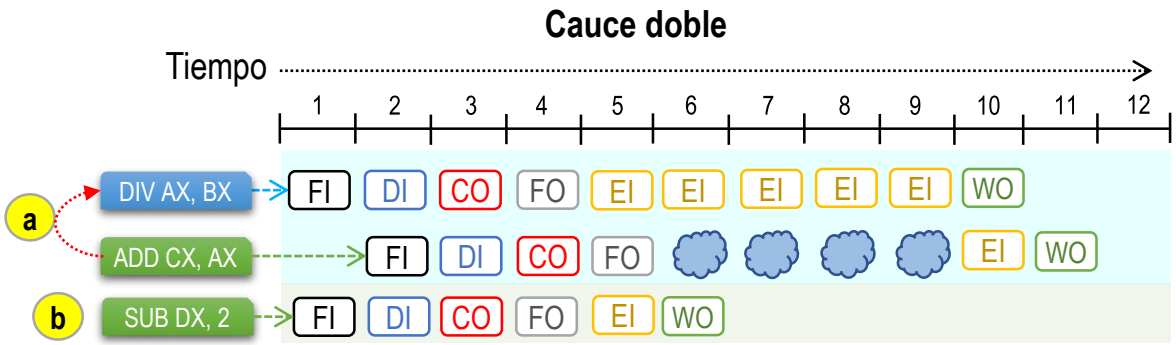
En la CPU **superescalar con planificación estática** el orden en que son captadas las instrucciones es el orden en que son enviadas a los diferentes cauces.



a La instrucción DIV demora varios ciclos y atasca la ejecución de la instrucción ADD ya que utiliza el registro AX, en donde se almacenara el resultado de la instrucción previa.

Superescalar – planificación dinámica (ejecución fuera de orden)

En la **CPU superescalar con planificación dinámica** se puede seleccionar que instrucciones se envían a cada cauce para reducir los riesgos, por ejemplo enviando instrucciones independientes.



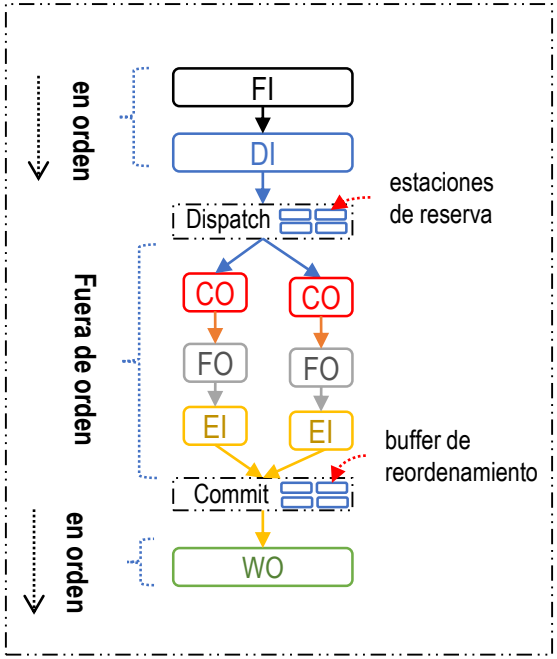
a La instrucción DIV demora varios ciclos y atasca la ejecución de la instrucción ADD ya que utiliza el registro AX, en donde se almacenara el resultado de la instrucción previa.

b La instrucción SUB no guarda dependencias de datos con las previas, y queda esperando que se complete la instrucción responsable del atasco.

Posibles soluciones – Ejecución fuera de orden

Consiste en enviar instrucciones para su ejecución en un orden diferente en el que esta indicado en el código.

- La implementación de la ejecución fuera de orden añade etapas adicionales al ciclo de instrucción, se añaden dos nuevas etapas, las cuales ocurren antes y después de la ejecución de instrucciones.
- El despachador (dispatch) debe evaluar si hay un atasco de datos o estructural y vigilar cuando una unidad de ejecución esta libre, y lo que hace es adelantar la siguiente instrucción en su ejecución, almacenando en un registro interno cual es el orden real de las instrucciones, una vez han sido ejecutadas estas son confirmadas (commit) en el orden original que estaban en el código.
- La etapa de commit se encarga de ordenar las instrucciones en orden de salida. En el caso de los saltos condicionales se ejecutan las dos ramas (tomado y no tomado), y se descarta las ramas de ejecución según la decisión del salto (ejecución especulativa).
- Las estaciones de reserva y el buffer de reodenamiento son registros internos que se utilizan para cada instrucción.
- Renombramiento de registros: es una técnica que abstrae los registros lógicos (AX, BX, CX y DX) de los registros físicos, sirve para reducir los dependencias de datos. En lugar de retrasar la escritura hasta que se completen todas las lecturas, se pueden mantener el valor anterior y el nuevo del registro. Las lecturas que preceden, en el orden del programa, a la escritura del nuevo valor se pueden proporcionar con el valor antiguo, incluso mientras que otras lecturas que siguen a la escritura se proporcionan con el nuevo valor.



Las tres últimas instrucciones son independientes de las tres primeras, pero el CPU no puede terminar hasta que se ejecuten las anteriores.

1. MOV AX, dato1
2. ADD AX, 2
3. MOV dato1, AX
4. MOV AX, dato2
5. ADD AX, 4
6. MOV dato2, AX

R1 y R2 registros físicos

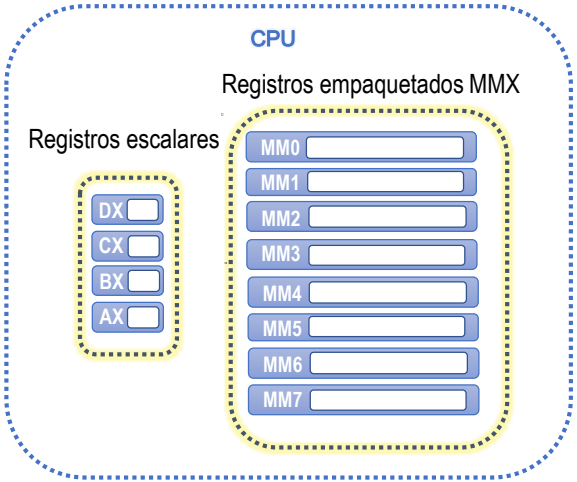
1. MOV R1, dato1
2. ADD R1, 2
3. MOV dato1, R1
4. MOV R2, dato2
5. ADD R2, 4
6. MOV dato2, R2

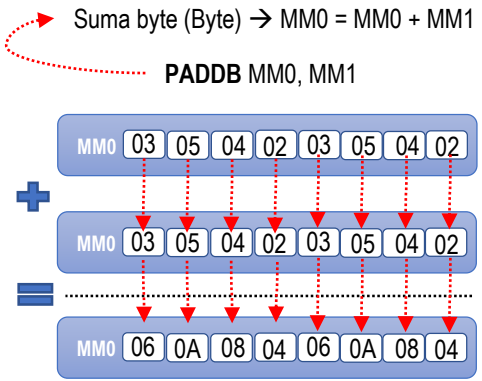
Ahora las tres últimas instrucciones se pueden ejecutar en paralelo con las tres primeras.

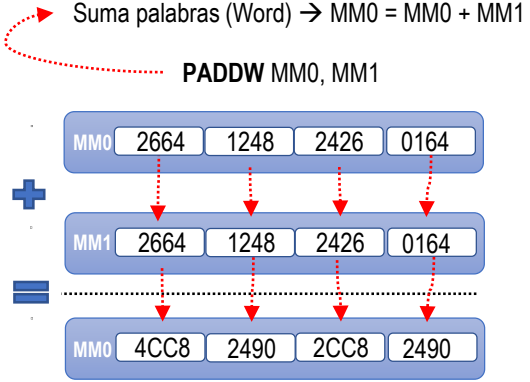
Paralelismo a nivel de datos (SIMD)

Un **procesador vectorial** es capaz procesar múltiples datos con una sola instrucción. Las instrucciones SIMD se usan ampliamente para gráficos 3D y procesamiento de audio/video en aplicaciones multimedia.

Ejemplo de instrucciones MMX (Matrix Math Extensions)







Las operaciones con los registros mmx no poseen aritmética modular como con los otros registros. Las operaciones multimedia no se benefician de este tipo de aritmética. Por ejemplo, considere un algoritmo para aumentar el brillo de una imagen agregando algún valor a cada píxel. Es posible que algunos de los píxeles ya sean casi blancos, con un valor cercano a 255, 255, 255, que es un píxel blanco en los modos de color RGB24 estándar. Si se agrega brillo adicional a estos píxeles, de repente se oscurecerán mucho. Si nuestro valor RGB ya es blanco 255 y agregamos 2 al brillo para hacerlo 257, si tuviéramos aritmética modular terminaríamos con valor 1. Nuestro píxel pasará de muy claro a muy oscuro, haciendo el algoritmo de ajuste de brillo parece incorrecto. Por esta razón exacta, la aritmética de saturación se incorporó en las instrucciones SIMD. La aritmética de saturación establece un valor máximo y mínimo para cada uno de los tipos de datos y, en lugar de recortar el valor, la respuesta se limitará a estos valores. El 255 + 2 en nuestro ejemplo estaría saturado a 255.

Paralelismo a nivel de datos (SIMD)

```
for(int i = 0; i < 8; i++) {
    vec1[i] += vec2[i];
}
```

Traducción a instrucciones vectoriales

```
movq mm0,  vec1 ; cargar 8 bytes en el registro MM0 desde memoria vec1
movq mm1,  vec2  ; cargar 8 bytes en el registro MM1 desde memoria vec2
paddb mm0, mm2 ; Sumar mm0 = mm0 +  mm1
movq vec1, mm0 ; Almacenar el resultado en vec1
```

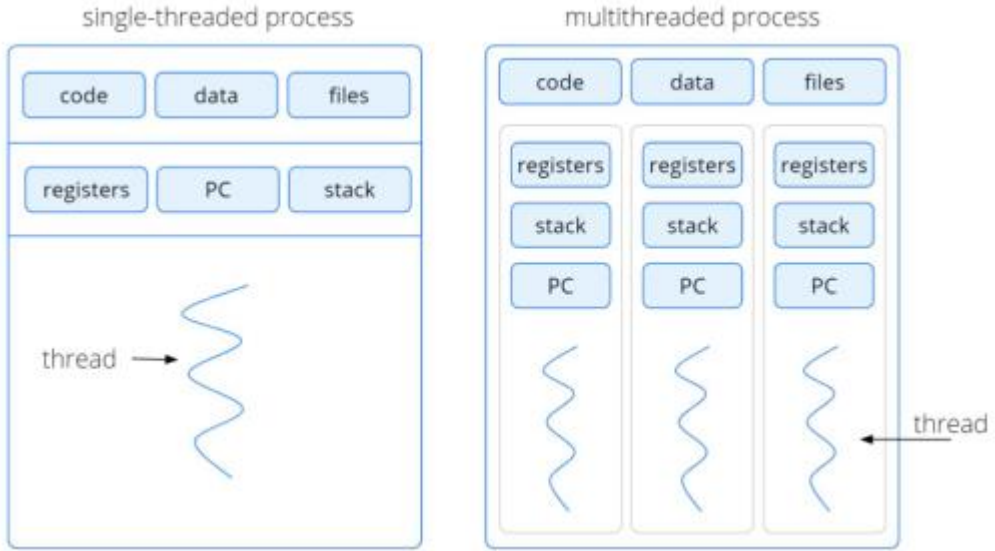
Paralelismo a nivel de procesos

Un proceso tiene un solo hilo de control. Si un proceso tiene múltiples hilos de control, puede realizar más de una tarea a la vez.

1. A=0;
2. B=1;
3. C=2;
4. C=C+1;
5. D:=5;

	Núcleo 1	Núcleo 2
1	A=0	B=1
2	C=2	D=5
3	C:=C+1	

Se necesita preservar la lógica del programa de modo que los resultados aparezcan en el mismo orden en el que ocupan las instrucciones en el programa.



Preguntas?