UT 2. Computadores segmentados
Tema 2.5 Lanzamiento múltiple de instrucciones

J. Flich, P. López, V. Lorente,
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Departamento de Informática de Sistemas y Computadores
Universitat Politècnica de València

UNIVERSIDAD POLITECNICA DE VALENCIA

DISCA

Índice

Bibliografía

John L. Hennessy and David A. Patterson.
Computer Architecture, Fifth Edition: A Quantitative Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5 edition, 2012.

Índice

El tiempo de ejecución de un programa es producto de tres términos:

$$T_e = I \times CPI \times T$$

Segmentación + planificación dinámica de instrucciones $\rightarrow$ CPI ≈ 1

¿Es posible reducir más el T_e?

1. Reducir el número medio de ciclos de reloj por instrucción *CPI*.
   $\rightarrow$ Aumentar las instrucciones lanzadas a ejecución en un ciclo.
   $\Rightarrow$ **Procesadores superescalares**.
2. Reducir el número de instrucciones ejecutadas *I*
   $\rightarrow$ Aumentar el trabajo que realiza cada instrucción máquina.
   $\Rightarrow$ **Procesadores VLIW**.
3. Reducir el periodo de reloj *T*.
   $\rightarrow$ Segmentar el ciclo de instrucción en más etapas.
   $\Rightarrow$ **Procesadores supersegmentados**.

Índice

Concepto de procesador superescalar

Se lanzan m instrucciones en cada ciclo de reloj $\rightarrow$ CPI=$\frac{1}{m}$
$\rightarrow$ Instructions Per Cycle (IPC)=m

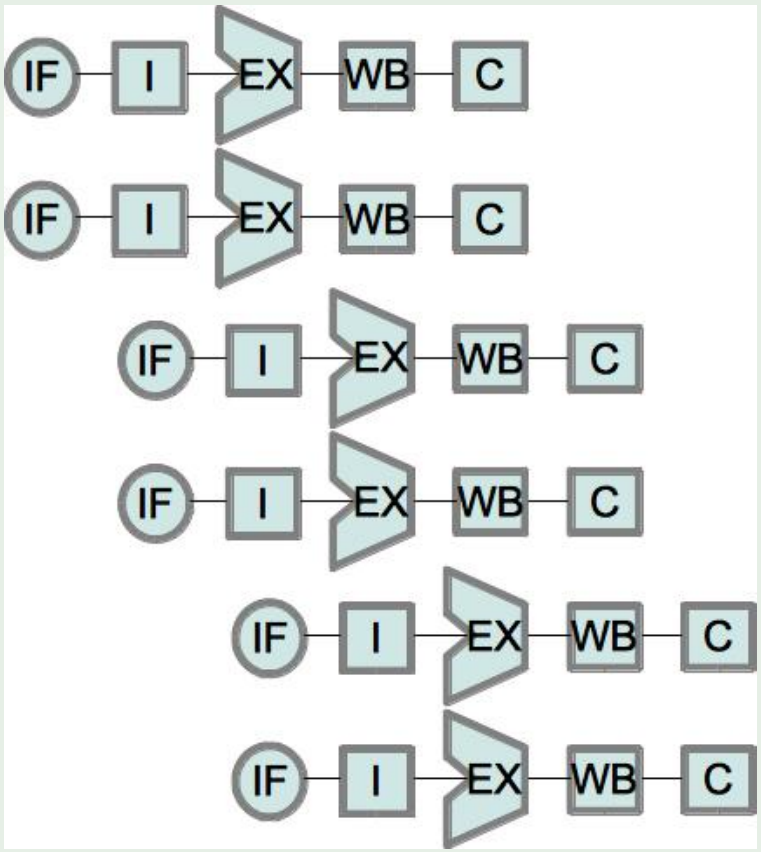
m: Número de **vías** del procesador superescalar (número de instrucciones lanzadas a ejecución en un ciclo).

Comparación con un procesador segmentado

- $T_{\text{segmentado}} = I \times CPI \times T$
- $T_{\text{superescalar}} = I' \times CPI' \times T' = I \times \frac{CPI}{m} \times T$

$\Rightarrow$ Aumentan las prestaciones en un factor m.

Ejemplo con $m = 2$:



Requisitos *hardware*:

- Acceso simultáneo a varias instrucciones (IF) y acceso simultáneo a varios datos en memoria
  - La antememoria debe suministrar m palabras por ciclo de reloj $\rightarrow$ antememoria **más ancha** o compuesta por **múltiples módulos**.
- Decodificación de varias instrucciones (I)
  - Decodificación de m instrucciones.
  - Comprobación de dependencias entre las m instrucciones buscadas simultáneamente y entre éstas y las lanzadas anteriormente.

$\Rightarrow$

  - Decodificador más complejo/rápido.
  - Múltiples ciclos de decodificación.

Requisitos *hardware*: (cont.)

- Lectura de varios operandos (I)
  - Múltiples puertos de lectura en el Banco de registros.
  - Múltiples puertos de lectura en el ROB.
- Ejecución simultánea de varias instrucciones (EX)
  - Varios operadores que trabajen en paralelo.
- Varias instrucciones en WB (WB)
  - Múltiples buses comunes de datos.
  - Múltiples puertos de escritura en el ROB.
- Graduación de varias instrucciones a la vez (C)
  - Varias instrucciones (de la cabeza del ROB) pueden efectuar la fase *Commit*.
  - Múltiples puertos de escritura en el banco de registros.

Implicaciones:

- Aumento de la probabilidad de riesgos → Además de las riesgos entre las instrucciones lanzadas en ciclos distintos, se producen riesgos entre las *m* instrucciones lanzadas en el mismo ciclo de reloj:

Ejemplo:

```
dadd R1,R2,R3    IF I  EX WB C
dadd R13,R4,R5   IF I  EX WB C
dadd R2,R1,R3       IF I  -  EX WB C
dadd R2,R1,R2       IF I  -  -  -  EX WB C
```

Implicaciones: (cont.)

- Aumento de la penalización por riesgos → Un ciclo de parada son *m* instrucciones no lanzadas:

Ejemplo: (4 RS de suma fp)

```
add.d f1,f2,f3 IF I  A1 A2 WB C
add.d f6,f4,f5 IF I  A1 A2 WB C
add.d f2,f1,f3    IF I  -  -  A1 A2 WB C
add.d f2,f1,f2    IF I  -  -  -  -  -  A1 A2 WB C
add.d f3,f3,f5       IF I  I  A1 A2 WB -  -  -  C
add.d f4,f4,f6       IF I  I  -  -  -  A1 A2 WB -  C
inst                    IF IF I  ...
inst                    IF IF I  ...
```

Implicaciones: (cont.)

$\rightarrow$ Enorme importancia de las técnicas para explotar el *ILP*: planificación dinámica de instrucciones, predicción de saltos, especulación, ...

Implicaciones: (cont.)

- Problema adicional de los saltos (1):
  La instrucción destino del salto puede no estar alineada en la cache $\rightarrow$ la etapa IF no entrega las *m* instrucciones en el ciclo posterior al salto:

Ejemplo:

```
inst          IF I  EX WB C
BEQZ R1,L     IF I  EX WB C
L-4:             IF <no sirve>
L:               IF I  EX WB C
```

Implicaciones: (cont.)

- Problema adicional de los saltos (2):
  La instrucción de salto puede no ser la última del grupo de m instrucciones $\rightarrow$ se desecha el resto de instrucciones del grupo:

Ejemplo:

```
BEQZ R1, L   IF I   EX WB C
inst         IF <no sirve>
L-4:            IF <no sirve>
L:              IF I   EX WB C
```

Implicaciones: (cont.)

- No se modifica el juego de instrucciones del procesador segmentado de partida $\rightarrow$ Son **compatibles a nivel binario** con el procesador segmentado de partida.

Procesadores superescalares no uniformes o con restricciones

$\Rightarrow$ Problema: Elevados requisitos *hardware*

Solución: Se imponen limitaciones en cuanto al tipo de instrucciones que se pueden lanzar simultáneamente $\rightarrow$ **procesador superescalar no uniforme**

Implicaciones:

- Aparición de riesgos estructurales.
- El compilador puede ayudar mucho colocando juntas aquellas instrucciones que cumplen los requisitos.
- La compatibilidad a nivel binario puede no ser tan eficiente.

Procesadores superescalares no uniformes o con restricciones (cont.)

Ejemplo

Procesador superescalar de 2 vías, en el que se pueden combinar:

- 2 instrucciones aritméticas enteras.
- 1 instrucción aritmética entera + 1 instrucción en coma flotante.
- 1 instrucción de carga/almacenamiento + 1 instrucción aritmética entera o de coma flotante
- 1 instrucción de salto + 1 instrucción de carga/almacenamiento o 1 instrucción aritmética entera o de coma flotante

$\rightarrow$ sólo una instrucción accede a memoria,
$\rightarrow$ sólo el banco de registros entero necesita tener doble puerto.
$\rightarrow$ no hace falta replicar todos los operadores

Índice

Límites del ILP

- El ILP se ha empleado para aumentar las prestaciones, con la ventaja de ser transparente al programador.
- ¿Cuánto ILP se puede explotar? Importante para el dimensionamiento del hardware y el diseño del compilador.
- Tres análisis:
  - Procesador ideal.
  - Procesador avanzado (2011).
  - Procesador real: Intel Cascade Lake (2019)

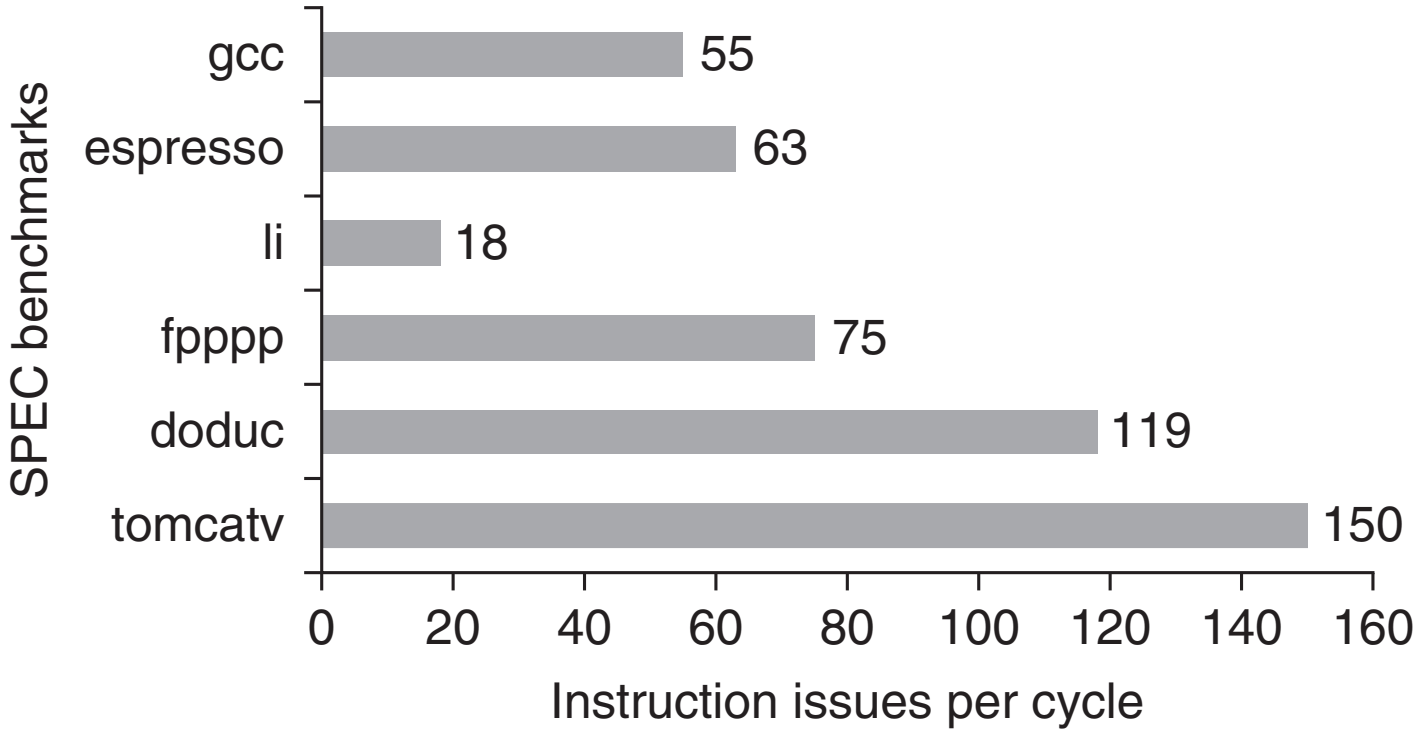
Límites del ILP: Procesador ideal

- Número de instrucciones lanzadas por ciclo ilimitado.
- Renombrado de registros infinito $\rightarrow$ todos los riesgos WAR y WAW solucionados.
- Predictor de salto perfecto.
- Desambiguación de memoria perfecta.
- Cache con tasa de acierto del 100 % $\rightarrow$ 1 ciclo de acceso a memoria.
- Operadores con latencia de 1 ciclo.

$\rightarrow$ el límite viene dado por las dependencias (verdaderas) de datos.

Límites del ILP: Procesador ideal (cont.)

Número medio de instrucciones lanzadas por ciclo:



ILP:

- Enteros (gcc, espresso, li): 18 a 63
- Coma flotante (fpppp, doduc, tomcatv): 75 a 150

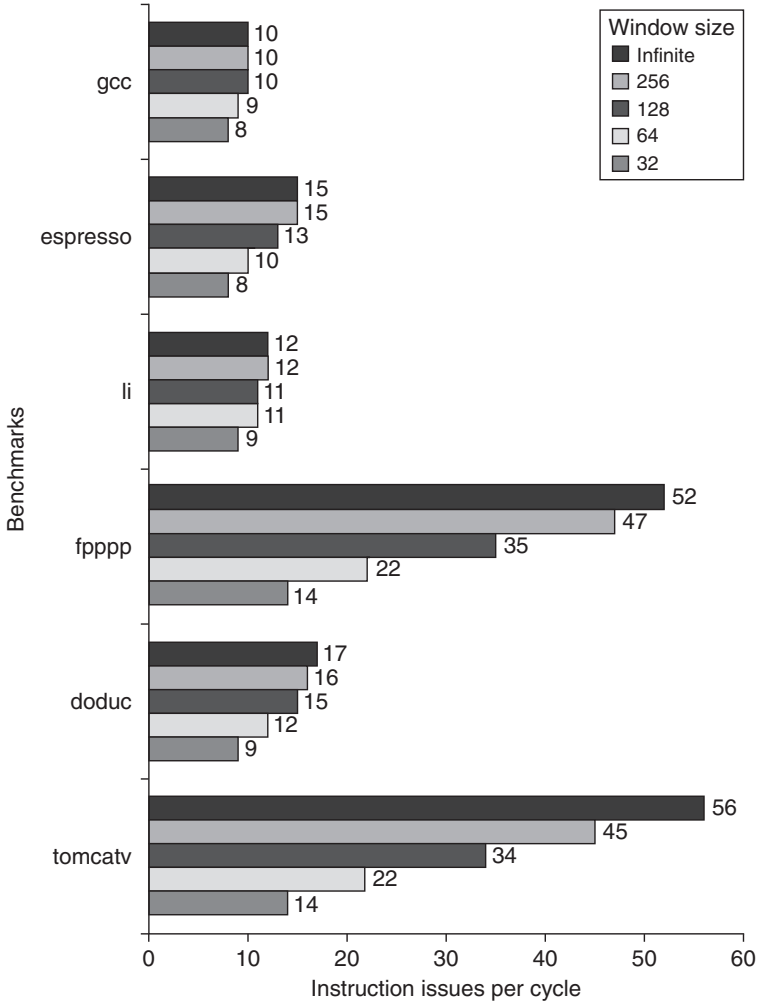
Límites del ILP: Procesador avanzado

Procesador mejorando muy notablemente el hardware disponible en 2011:

- Lanza hasta 64 instrucciones por ciclo, sin restricciones (10X el ancho de *Issue* de un proc. real en 2011).
- Renombrado de registros con 64 registros adicionales enteros + 64 registros adicionales c.f.
- Predictor de salto *tournament predictor* con 1K entradas y buffer de direcciones de retorno de procedimiento de 16 entradas.
- Desambiguación de memoria perfecta.
- Cache con tasa de acierto 100 %.
- Operadores con latencia de 1 ciclo.

Límites del ILP: Procesador avanzado (cont.)

Número medio de instrucciones lanzadas por ciclo (window-size, nº de instrucciones “en vuelo” = tamaño del ROB):



Notable reducción del ILP:

- Enteros: exhiben menos ILP, y les afecta mucho la predicción no ideal de saltos.
- Coma flotante: les afecta mucho la reducción de la ventana de instrucciones.

ILP en procesadores actuales de gama alta: Intel Cascade Lake

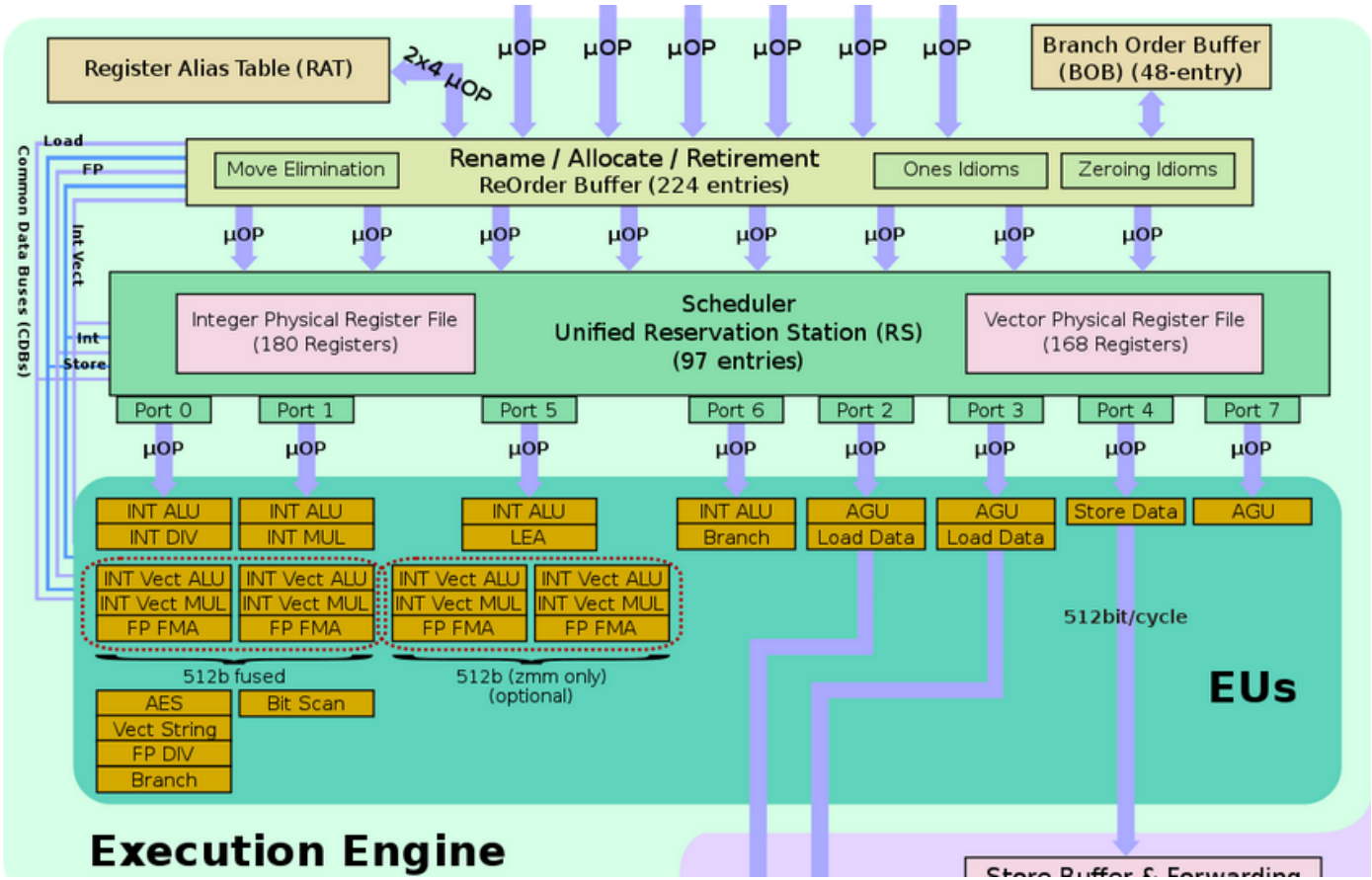
Comercializado en 2019 con el nombre Core X (i7, i9) en los equipos de sobremesa y Xeon en los servidores.

- Vías del procesador superescalar: writeback, commit (4)
- Issue hasta 8 por ciclo (issue ports)
- Estaciones de reserva centralizadas: 97
- Pipeline(s): 14-19 etapas
- Procesador con 56 núcleos
- Jerarquia de cache:
  - L1-D 32 KB 8 vías (4-5 ciclos)
  - L2 1MB 16 vías (14 ciclos)
  - Cache L3 compartida (50-70 ciclos)

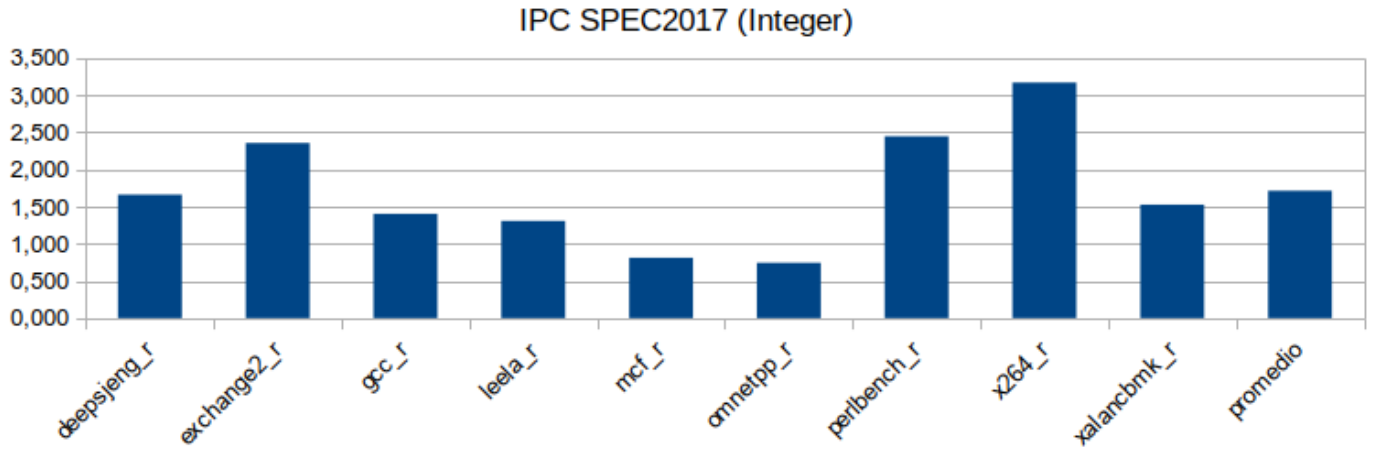
+ Información sobre Cascade Lake

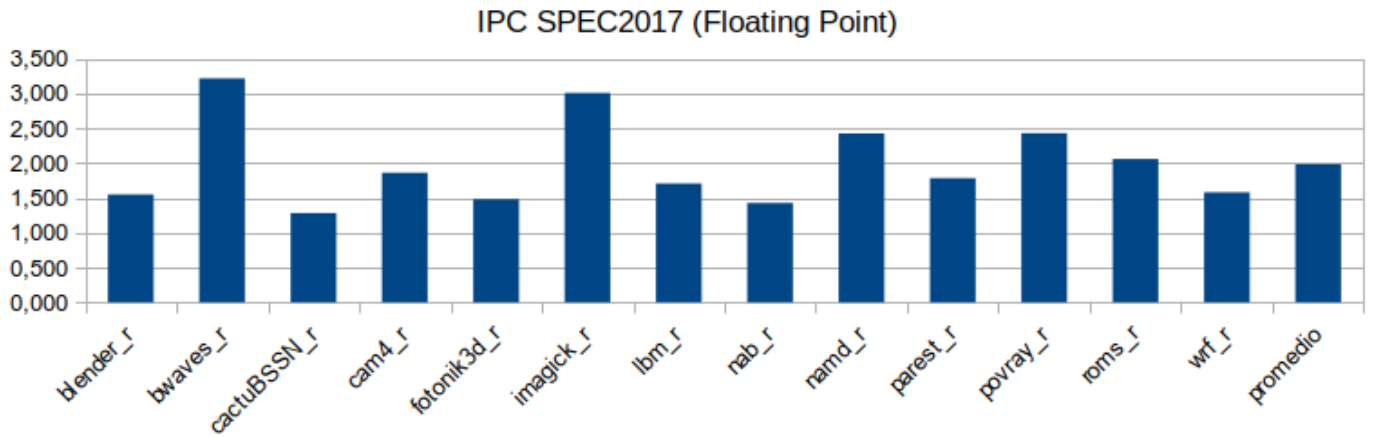
ILP en procesadores actuales de gama alta: Intel Cascade Lake (cont.)

Detalle de la unidad de ejecución.



ILP en procesadores actuales de gama alta: Intel Cascade Lake (cont.)





$\Rightarrow$ IPC máximo 2.5 en la mayoria de las aplicaciones. Necesidad de buscar paralelismo a otro nivel...

ILP vs. TLP

ILP: *Instruction Level Parallelism*
TLP: *Thread Level Parallelism*

El ILP explota paralelismo dentro de un segmento de código lineal o un bucle. El ILP que se puede explotar tiene una cota superior dada por:

- El ILP del programa.
- Las limitaciones del *hardware*.

$\rightarrow$ reducción de las prestaciones efectivas.
$\rightarrow$ baja utilización de las unidades funcionales.

Alternativa: ejecutar en paralelo varios hilos del programa $\rightarrow$ explotar el TLP

- Replicando recursos: **procesadores multinúcleo, multiprocesadores**
- Compartiendo recursos: **procesadores multihilo**.

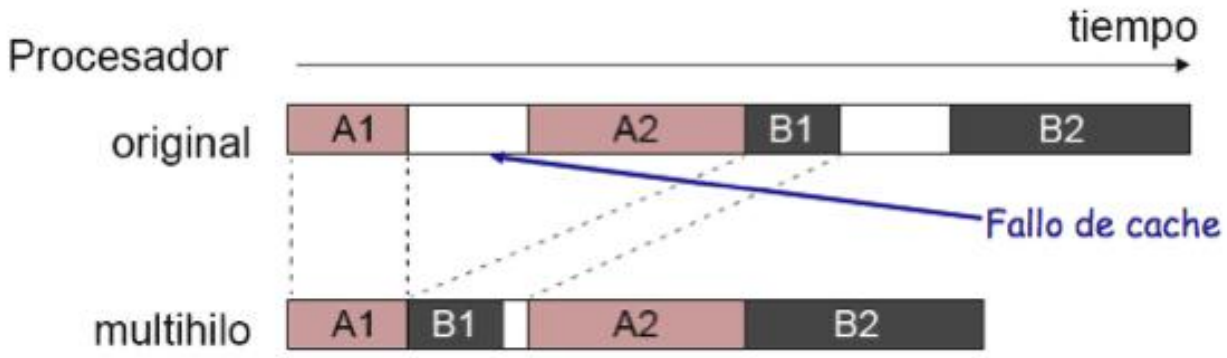
TLP: Thread-Level Parallelism

Thread: proceso (hilo) con sus propias instrucciones y datos.

- TLP representa el uso de múltiples hilos que son inherentemente paralelos.
- Objetivo: Usar múltiples flujos de instrucciones para:
  - Aumentar la productividad de computadores que ejecutan muchos programas diferentes $\rightarrow$ cada hilo es un programa independiente.
  - Reducir el tiempo de ejecución de un programa compuesto por múltiples hilos independientes: multithreaded $\rightarrow$ cada thread es parte de un programa paralelo.
- TLP puede ser más eficaz en coste que ILP.

Idea inicial: Multithreading para tolerar la latencia de la memoria

- En un procesador superescalar, los fallos de cache insertan *stalls*, desaprovechando las unidades funcionales disponibles.
- Con multithreading, en lugar de detener al procesador, se conmuta a otro hilo:



$\rightarrow$ Se incrementa el trabajo procesado por unidad de tiempo
$\rightarrow$ Si los hilos son de la misma aplicación, se reduce el tiempo de ejecución

Recursos necesarios para la ejecución multihilo

- Multithreading: múltiples hilos comparten los recursos del procesador
- El procesador debe mantener el estado de cada hilo:
  - Banco de registros por hilo.
  - Contador de programa (PC) por hilo.
  - Tabla de páginas por hilo.
- La memoria se comparte a través de los mecanismos de memoria virtual (que, de hecho, ya soporta multiprogramación).
- Mecanismos *hardware* muy rápidos para la conmutación entre hilos (mucho más rápido que un cambio de contexto).
- La aplicación debe contener varios hilos, identificados por el compilador (típicamente mediante un programa que soporta la expresión de paralelismo) o por el programador.

Tres aproximaciones para el multithreading

- Multithreading de grano fino.
- Multithreading de grano grueso.
- Multithreading simultaneo (SMT):

Multithreading de grano fino

- Conmuta entre hilos cada ciclo de reloj, entrelazando la ejecución de los diferentes thread.
- Generalmente se hace cíclicamente (*round-robin*), saltándose los hilos bloqueados.
- Necesita que el cambio de hilo sea muy rápido.
- Ventaja: como el tiempo necesario para cambiar de hilo es muy pequeño $\rightarrow$ permite ocultar *stalls* de alta y de baja latencia.
- Inconveniente: como se cambia de hilo tanto si se ha detenido como si no lo ha hecho, retarda la ejecución de cada hilo individual.

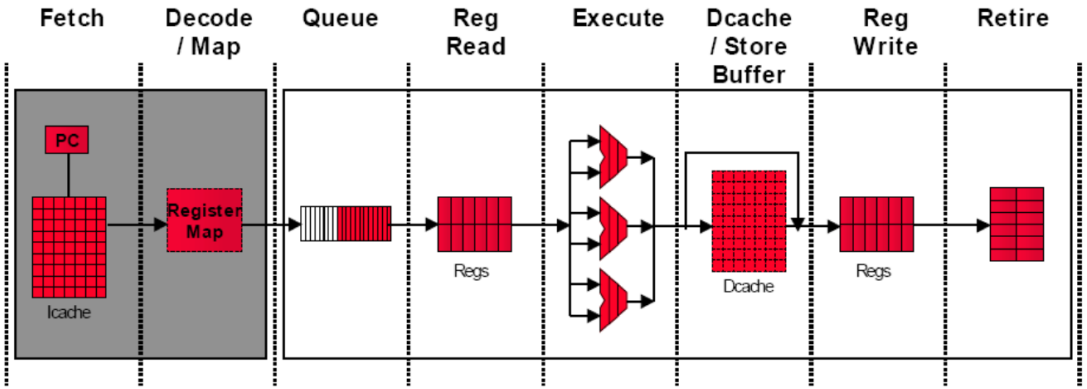
Multithreading de grano grueso

- Conmuta entre hilos solo en caso de *stalls* largos, como fallos de cache L2.
- Ventajas:
  - No necesita conmutación entre hilos muy rápida ya que oculta latencias largas.
  - Habitualmente, se vacía el pipeline y procede a buscar instrucciones del hilo entrante.
  - La conmutación se produce cuando un hilo no puede avanzar, por lo que se evita retrasar en exceso su ejecución.
- Inconveniente: como el tiempo de cambio de hilo requiere múltiples ciclos (por ej. una decena), no elimina la penalización debido a *stalls* cortos.

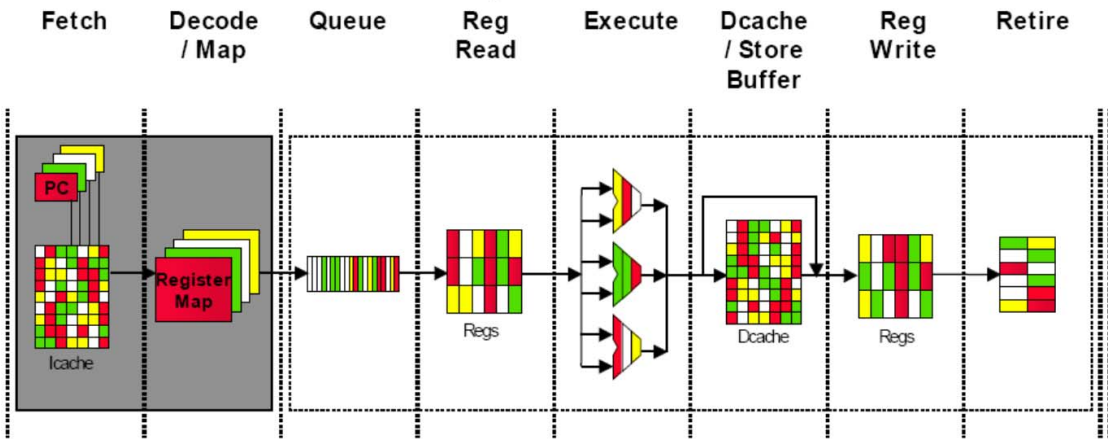
Multithreading simultáneo (SMT)

- Un procesador superescalar ejecuta simultáneamente instrucciones de distintos hilos, tratando de utilizar todos los recursos disponibles.
- Dentro de un procesador superescalar con ejecución fuera de orden ya hay mecanismos para soportar la ejecución de más de un hilo:
  - El renombrado de registros proporciona un identificador único para los operandos de una instrucción, por tanto instrucciones de diferentes hilos se pueden mezclar sin confundir sus operados.
  - La planificación dinámica de instrucciones resuelve las dependencias entre instrucciones de cada hilo.
  - La ejecución fuera de orden permite una utilización eficaz de los recursos.
  - Se requiere un *ROB* por hilo (lógico o físico).

Multithreading simultáneo (SMT) (cont.)



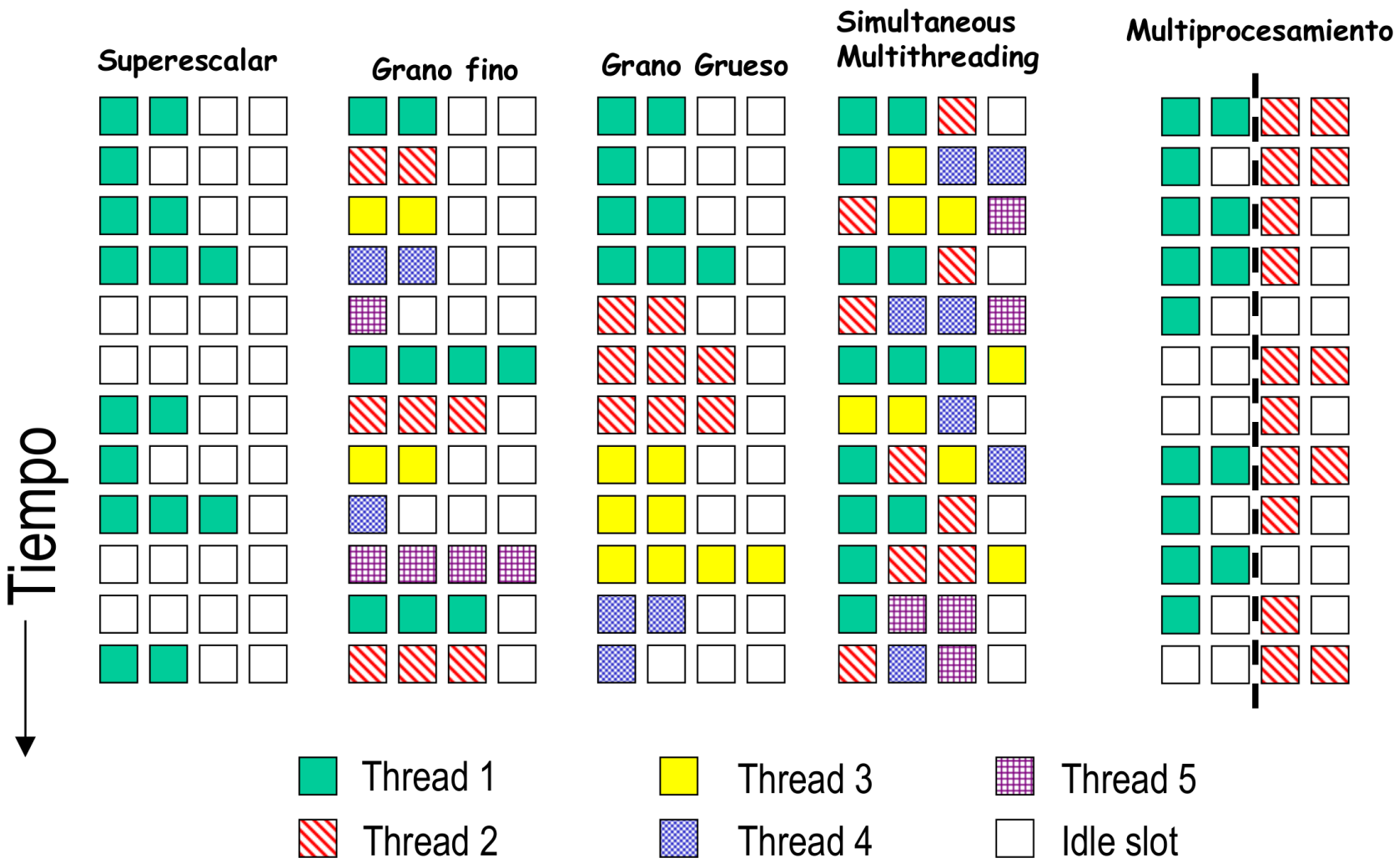
✓ todos los recursos utilizados por un hilo



✓ recursos para distinguir el estado de los hilos

✓ los otros recursos se pueden compartir

Comparación



Índice

Concepto de procesador VLIW

VLIW: *Very Long Instruction Word*

Son procesadores con un formato de instrucción muy largo, capaz de codificar varias (p) operaciones sobre una misma instrucción.

Se aplica planificación estática de instrucciones (compilador).

Ejemplo con $p = 5$:

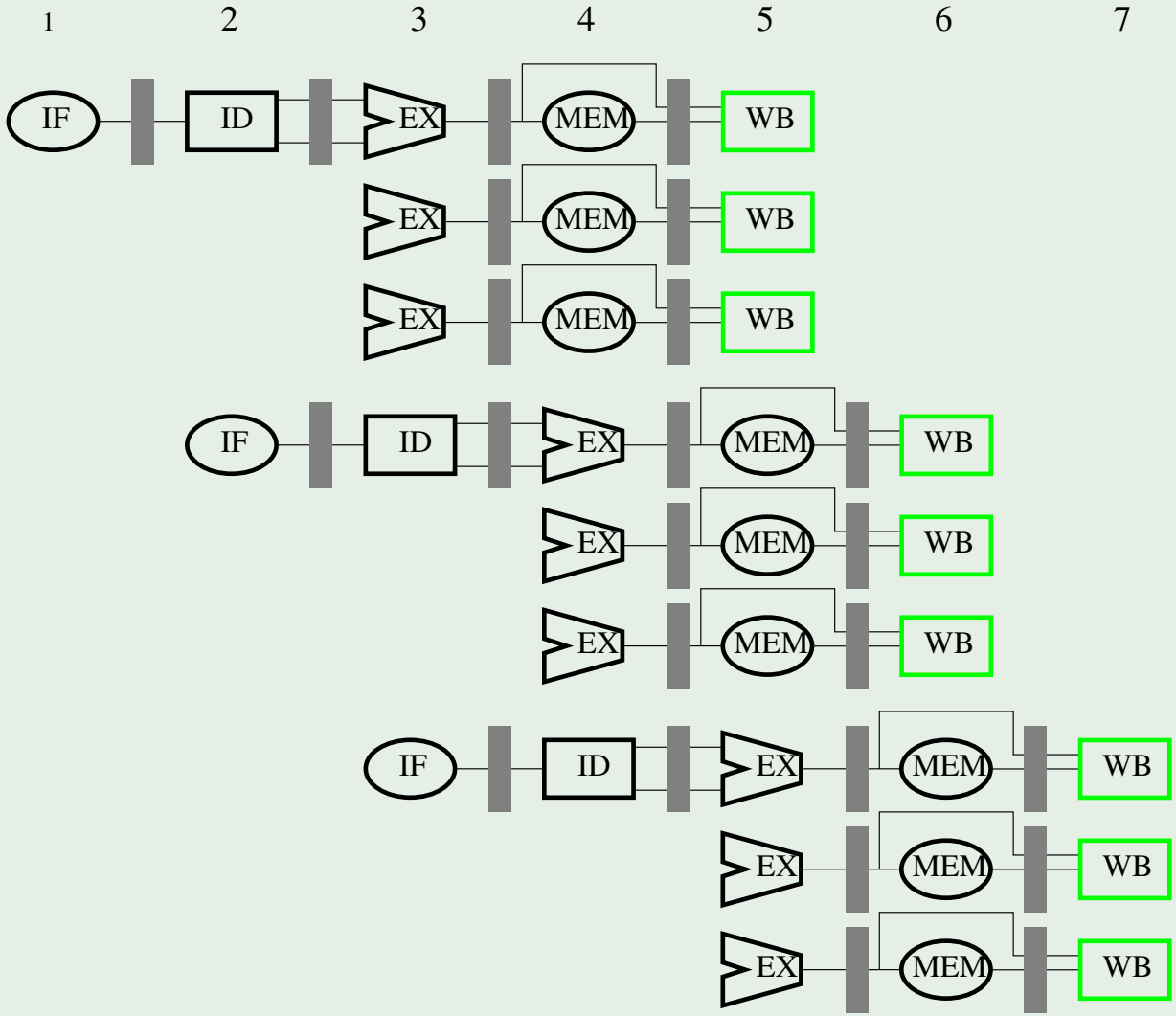
Memoria_1	Memoria_2	CF_1	CF_2	Entera/Salto

Comparación con un procesador segmentado

- $T_{\text{segmentado}} = I \times CPI \times T$
  $T_{\text{VLIW}} = I' \times CPI' \times T' = \frac{I}{p} \times CPI \times T$

$\Rightarrow$ Aumentan las prestaciones en un factor p.

Diagrama i–t con $p = 3$:



Implicaciones:

- El *hardware* no aplica gestión dinámica de instrucciones.
- El compilador extrae el paralelismo a nivel de instrucciones, empaquetando varias operaciones independientes (o NOPs, si no las encuentra) en una misma instrucción $\rightarrow$ técnicas de compilación especiales.
- No son compatibles a nivel binario con las máquina escalar de partida, ni con otras VLIW que tengan distinto *hardware* (distinto número de operadores y/o distintas latencias de cada uno)
- Si el compilador no es muy bueno o el código tiene ILP bajo, el tamaño del código generado es mucho mayor al convencional.

Ejemplo

Código convencional:

```
loop:   L.D F1, 0(R1)
        ADD.D F2,F1,F0
        S.D F2,0(R1)
        DSUB R1,R1,#8
        BNEZ R1, loop
```

Código VLIW para latencias L.D 2 ciclos y ADD.D 3 ciclos; 16 regs c.f.

$Memoria_1$	$Memoria_2$	Aritmética CF_1	Aritmética CF_2	Entera/Salto
L.D F1,0(R1)	L.D F3,-8(R1)			
L.D F5,-16(R1)	L.D F7,-24(R1)			
L.D F9,-32(R1)	L.D F11,-40(R1)	ADD.D F2,F1,F0	ADD.D F4,F3,F0	
L.D F13,-48(R1)		ADD.D F6,F5,F0	ADD.D F8,F7,F0	
		ADD.D F10,F9,F0	ADD.D F12,F11,F0	
S.D F2,0(R1)	S.D F4,-8(R1)	ADD.D F14,F13,F0		
S.D F6,-16(R1)	S.D F8,-24(R1)			
S.D F10,-32(R1)	S.D F12,-40(R1)			DSUB R1,R1,#56
S.D F14,8(R1)				BNEZ R1,loop

Índice

Concepto de procesador supersegmentado

El ciclo de reloj es t veces inferior al del procesador segmentado de partida $\rightarrow$ El número de etapas del ciclo de instrucción es kt $\rightarrow$ CPI=1 (pero el ciclo es inferior!)

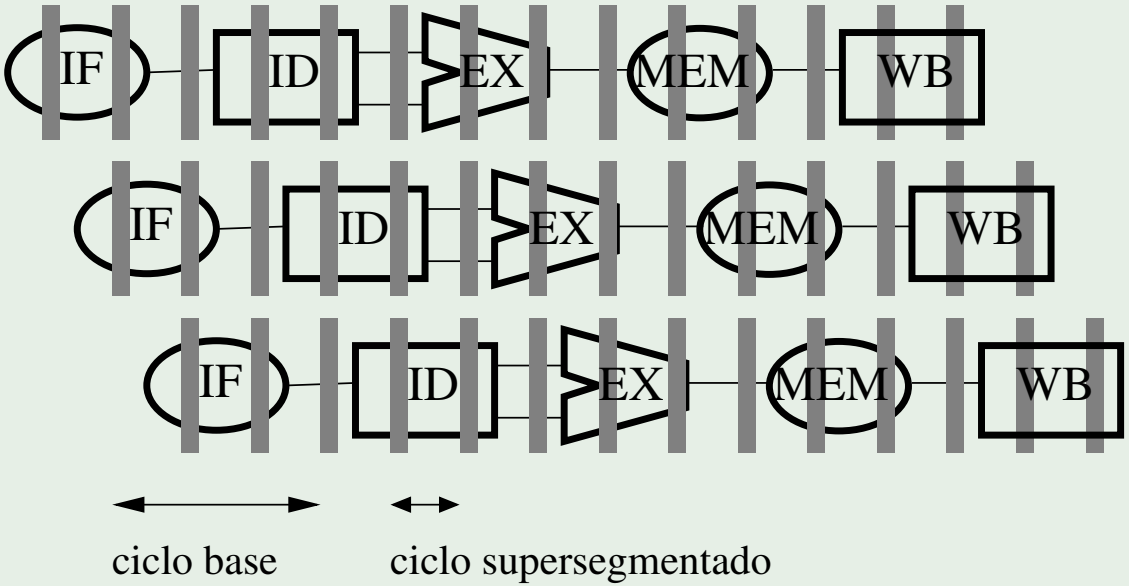
t: **grado** de supersegmentación (número de subetapas en las que se divide cada etapa base).

Comparación con un procesador segmentado

- $T_{\text{segmentado}} = I \times CPI \times T$
  $T_{\text{super-segmentado}} = I' \times CPI' \times T' = I \times CPI \times \frac{T}{t}$

$\Rightarrow$ Aumentan las prestaciones en un factor t.

Ejemplo con $t = 3$:



Implicaciones:

- No necesita replicar unidades de ejecución, pero hay que realizar una segmentación más “compleja” (hay que segmentar los operadores o las memorias en varias etapas)
- Mayor frecuencia de funcionamiento $\rightarrow$ mayor sobrecarga potencial de los registros intermedios y del **desfase del reloj**.
- Realmente, no todas las etapas del procesador segmentado “convencional” tienen por qué super-segmentarse. Sólo las más lentas.
  Ejemplo: MIPS
  IF(10ns) ID(5ns) EX(10ns) M(10ns) WB(5ns)
  $\downarrow$
  IF1(5ns) IF2(5) ID(5) EX1(5) EX2(5) M1(5) M2(5) WB(5)