❖Tema 3. Límites ILP. Superescalar. TLP

➢ 3.1 Introducción

Ya hemos visto cómo intentar conseguir un CPI unidad con rutas de datos segmentadas y algoritmos de ejecución elaborados como Tomasulo. Por mucho que avancemos en ese campo nunca seremos capaces de ejecutar más de un Commit por ciclo, lo que implica que no se puede obtener CPI menores que 1.

¿Es posible conseguir $CPI < 1$?

Para poder rebajar el CPI por debajo de 1 es necesario que las instrucciones no estén limitadas en sus etapas a realizar solo una por ciclo. Por ejemplo, en el algoritmo de Tomasulo era posible realizar varios LOAD, ADDD y MULD de forma simultánea en la fase de EXE, sin embargo no se puede hacer FETCH, ISSUE o COMMIT de más una instrucción por ciclo… así es imposible $CPI < 1$.

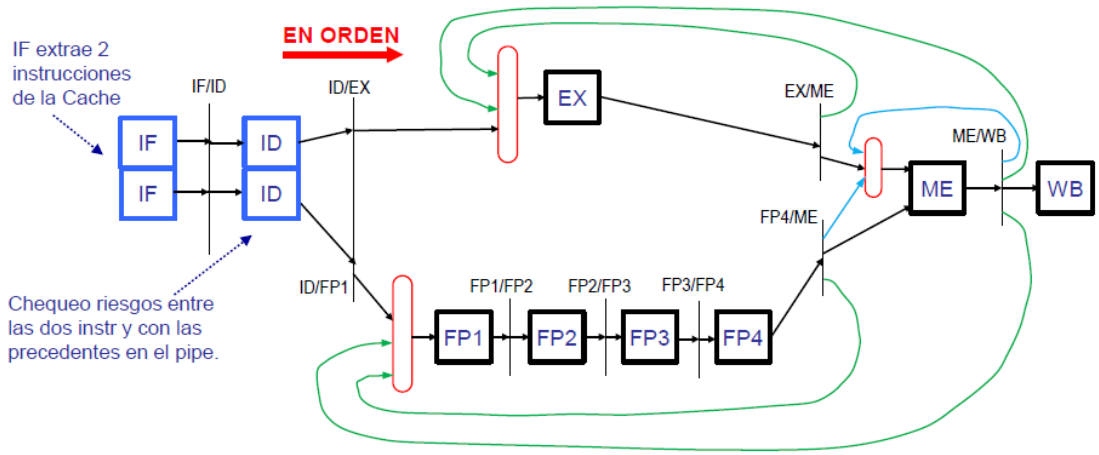
Idea… procesadores Superescalares ->

- Procesador Superescalar con Planificación Estática.
- Procesador Superescalar con Planificación Dinámica.

Idea… procesadores con MultiThreading.

➢ 3.2 Superescalar con Planificación Estática

El procesador duplica muchos recursos para poder lanzar 2 instrucciones a la vez completando las etapas que solo admitían una por ciclo.



- En la etapa ME solamente el camino superior puede acceder realmente a la memoria. El otro la puentea.
- En la etapa WB puede finalizar una instrucción de FP y una ENT si escriben en bancos de registros separados.

- Puede chequear conflictos de varias instrucciones a la vez, pero requiere de una predicción sofisticada.
- Para asegurarse que no hay conflicto de recursos se lanza una instrucción de FP + una instrucción de otro tipo (Load, ENT, Store, Salto).
- Presenta unidades funcionales segmentadas.
- Lanzando 2 instrucciones no puede reducir más de $CPI = 0.5$ y solo si existen suficientes instrucciones de FP.

Un ejemplo de procesador superescalar con planificación estática es el VLIW (Very Long Instruction Word) que tiene las siguientes características:

- Forma "paquetes" de instrucciones de tamaño fijo (por ejemplo 3) en tiempo de compilación. Estos paquetes se lanzan como una superinstrucción cada ciclo.
- La etapa de ID no chequea riesgos.
- El compilador es conocedor del HW disponible en la ruta de datos y empaqueta evitando riesgos de recursos. Si no encuentra suficientes instrucciones rellena con NOP.

Ventajas

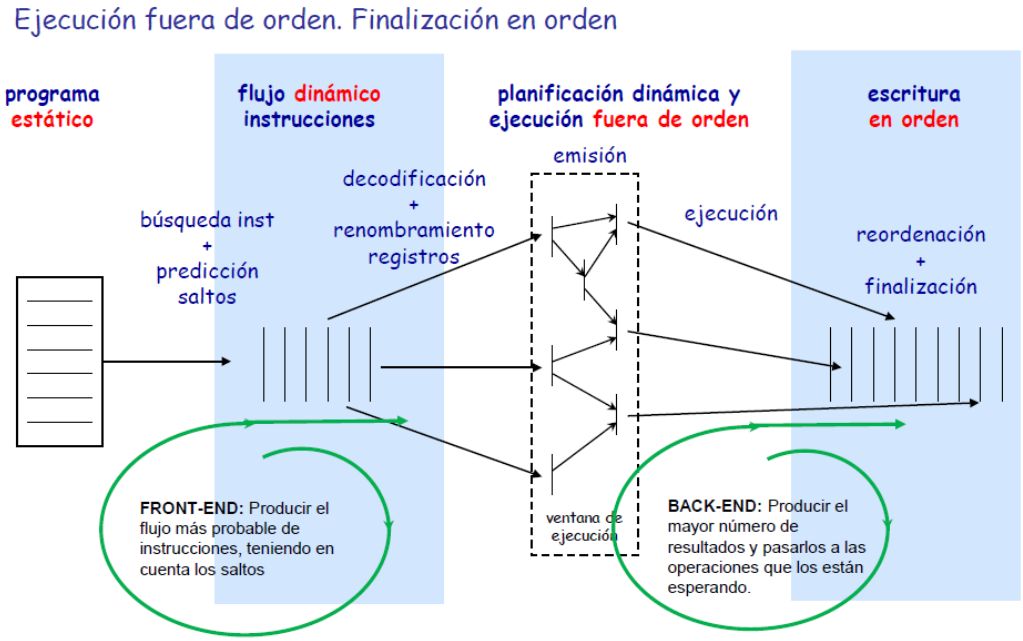
1.) No modifica el código, tiene compatibilidad binaria con los elementos de la ruta.
2.) En VLIW no hay riesgos en ejecución porque todos los riesgos se chequean antes del lanzamiento. Si los hubiese, las instrucciones se bloquean.

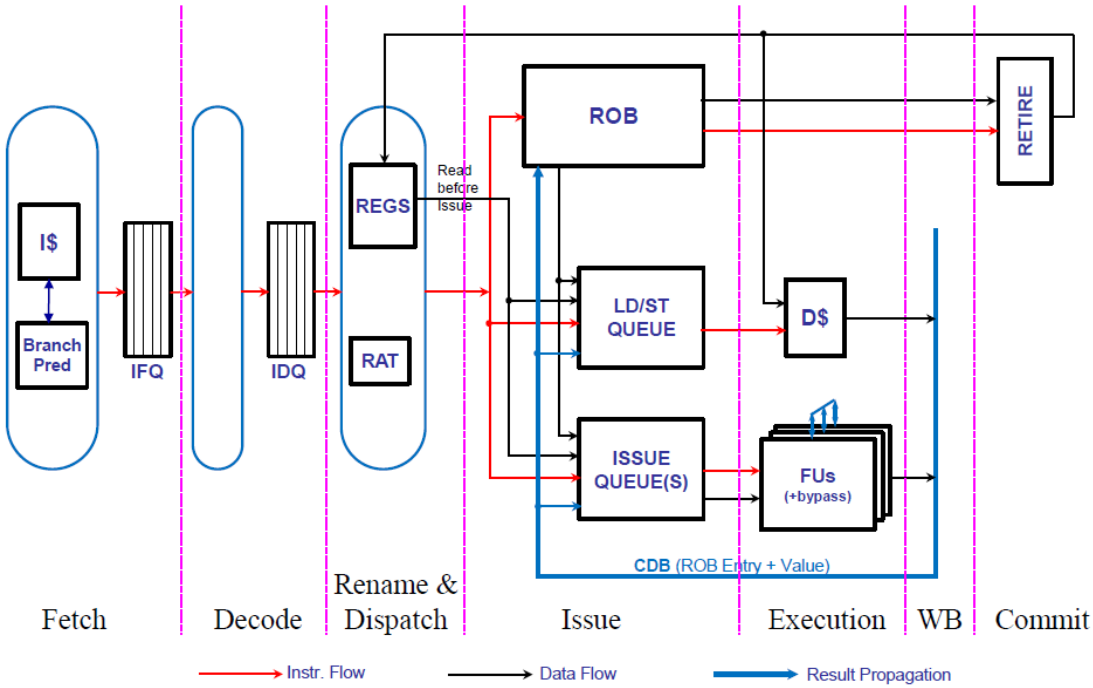
Desventajas

1.) Al mezclar instrucciones solo se consigue mejora de rendimiento si hay suficientes instrucciones para rellenar paquetes o hacer lanzamientos múltiples. Difícil bajar de $CPI = 0.5$.
2.) Se producen bloqueos en el lanzamiento (igual que en un MIPS básico).
3.) La planificación fija no permite adaptarse a cambios en tiempo de ejecución (fallos de caché por ejemplo).
4.) Los códigos deben replanificarse para cada nueva implementación.

➢ 3.3 Superescalar con Planificación Dinámica

La idea del procesador superescalar con planificación dinámica consiste en extraer de un código estático las instrucciones (normalmente varias por ciclo), reordenarlas en varias pilas de ejecución según naturaleza y tipos de operandos y ejecutarlas en paralelo. Una vez ejecutas y resueltas se van finalizando (suele utilizar un ROB igual que en Tomasulo).





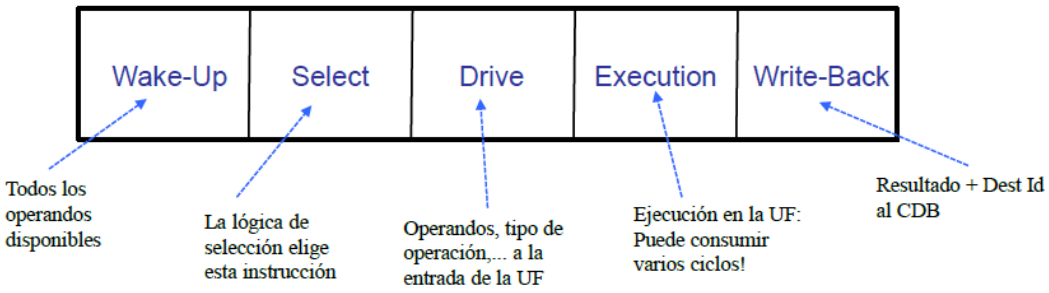
- **Fetch:** Desde la Instruction Cache (I$) se mandan instrucciones del código estático usando predictores. En un solo ciclo se pueden acceder a varias, en un "paquete de instrucciones".

- **Decode:** las instrucciones de la cola se decodifican, se les puede cambiar el formato de instrucción para reducir su peso en bits sobre la ruta. También se pueden resolver varias decodificaciones simultáneamente.

- **Rename:** para anticipar operandos y también evitar riesgos EDE / EDL se renombran registros fuente. Se usa una tabla de renombramiento llamada RAT que veremos después.

- **Dispatch:** es lo que se entendía anteriormente en Tomasulo como "Issue". Se reserva una entrada en el ROB y en las ISSUE queues. Las Queues son algo así como las estaciones de reserva, las instrucciones permanecen ahí antes de ser ejecutadas mientras esperan a que sus operandos estén listos y/o quede una Unidad Funcional libre. Las instrucciones se insertan en el ROB según el orden establecido por el código estático de programa.

- **Issue & Execution & Write Back:** en los procesadores superescalares el Issue se denomina "emisión". Significa que la instrucción está lista para ser ejecutada y se manda a la UF correspondiente. Después comenzará su EXE. Cuando una instrucción está lista para ser ejecutada pasa por una subetapa llamada "Wake-Up" e inicia un proceso hasta realizar su escritura.



La ***ventana de instrucciones*** se denomina al conjunto de instrucciones candidatas a ser lanzadas, es decir, que han realizado la fase de Dispatch, pero no han sido emitidas.

La complejidad de la arquitectura es fuertemente dependiente del tamaño de la ventana de instrucciones. Se necesitan colas de espera con mucho espacio y gran cantidad de comparadores que evalúen constantemente si los operandos están listos.

- **Commit:** definimos aquí un concepto llamado "estado arquitectónico". Este concepto relata el estado de los registros de la arquitectura y el contenido de los operandos en memoria. En el commit se garantiza que el estado arquitectónico se actualiza exactamente en el mismo orden que se haría en un procesador secuencial (como un MIPS). Las instrucciones se retiran del ROB cuando alcanzan la cabecera. Es posible hacer commit también de múltiples instrucciones en un solo ciclo si todas están bien predichas y han ejecutado su Write Back.

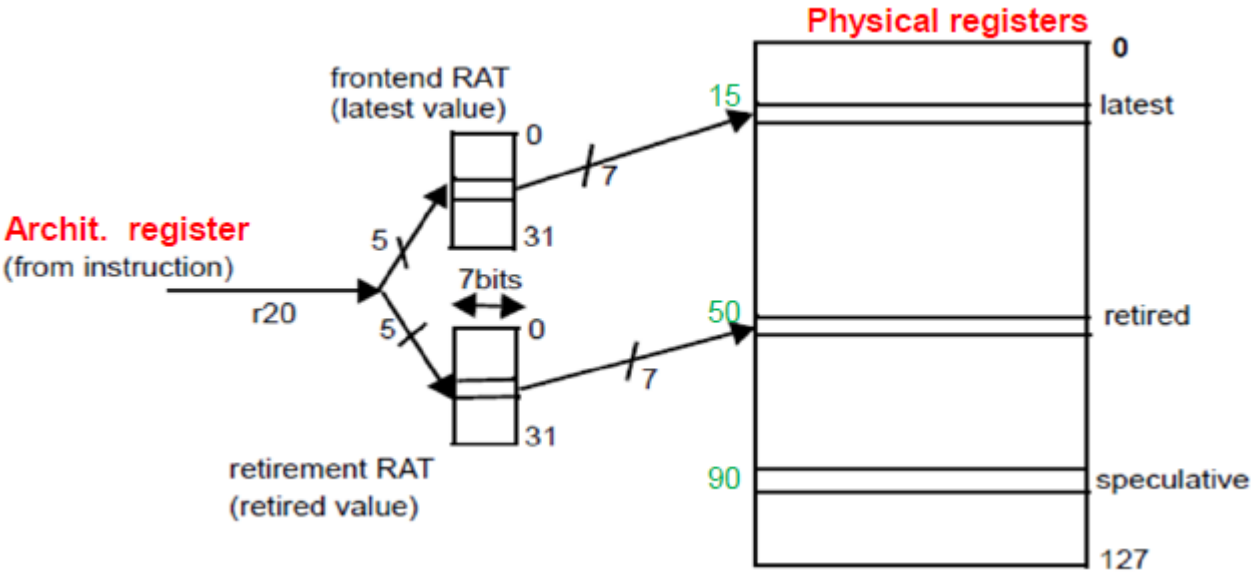
  Para la escritura de los resultados en el banco de registros también se usa la tabla RAT, pero la de salida (debemos ver las dos tablas).

• 3.3.1 Fichero de Registros Unificado. Tablas RAT.

El chip dispone de un banco de registros, que es el habitual, y se denominan Registros Arquitectónicos. En la etapa de Write-Back se escriben los resultados en un banco de registros aparte llamado Registros Físicos que tiene más tamaño que los registros arquitectónicos.

Los resultados que se generan en las Unidades Funcionales o Load no se guardan en el ROB, éste solo mantiene el orden en el que se debe hacer los commit.

Los Registros Arquitectónicos estarán mapeados sobre los Registros Físicos.



Supongamos en este ejemplo que disponemos de 32 RA y de 128 RF. Cuando una instrucción direcciona uno de los 32 RA se indexan dos tablas llamadas Frontend RAT y Retirement RAT. El registro 20 está siendo referenciado. El ancho de las tablas será de 7 bits ya que con ello se indexarán los Registros Físicos que son 128. La Frontend RAT además tiene un bit de Ready para indicar si el contenido del RF al que referencia ya está actualizado y se puede usar como operando de lectura.

Frontend RAT	**Retirement RAT**
Para cada RA indica el número de RF que contiene el valor asignado por última vez como destinatario de una operación. Además tiene 1 bit más para indicar Ready.	Para cada RA indica el número de RF que contiene el valor asignado por última vez como destinatario, pero cuya instrucción ya pasó por la etapa de commit. Por lo tanto es su contenido físico actual. Puede coincidir con la Frontend.

Además de estos dos punteros de "Latest" y "Retired" generados por las dos RAT, pueden haber otras entradas de RF que contengan valores especulados no siendo los últimos. Cualquier valor contenido en un RF es susceptible de ser especulativo salvo el marcado por Retirement RAT.

Ejemplo de comportamiento del renombrado de registros:

```
Cod1 r20, ---, ---
    ....
    (instrucciones zona A)
    ....
Cod2 r20, ---, ---
    ....
    (instrucciones zona B)
    ....
Cod3 r20, ---, ---
    ....
    (instrucciones zona C)
    ....
```

Dispatch Cod1: Hallar un RF libre para asignar a r20.
Sup que se elige RF50 ➔ Asignar: Frontend RAT(20)=50

Rename en zona A: Si r20 aparece como operando, la Frontend RAT indicará que r20 está representado por RF50

Dispatch Cod2: Hallar un RF libre para asignar a r20.
Sup que se elige RF90 ➔ Asignar: Frontend RAT(20)=90

Rename en zona B: Si r20 aparece como operando, la Frontend RAT indicará que r20 está representado por RF90

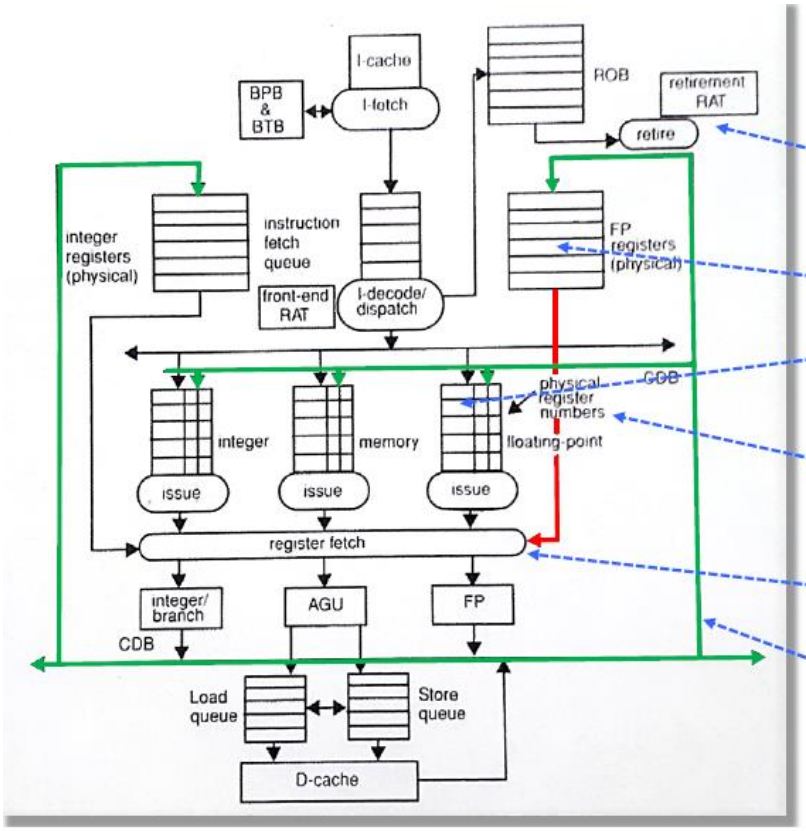
Dispatch Cod3: Hallar un RF libre para asignar a r20.
Sup que se elige RF15 ➔ Asignar: Frontend RAT(20)=15

Rename en zona C: Si r20 aparece como operando, la Frontend RAT indicará que r20 está representado por RF15

- Supongamos que Cod1 hace COMMIT ➔ Asignar: Retirement RAT(20)=50.
  - Todavía puede haber instr en Zona A que precisen RF50 como operando.
  - RF50 representa el valor no especulado de r20
- Supongamos que Cod2 hace COMMIT ➔ todas las instr de Zona A han hecho COMMIT
  - Como Retirement RAT (20) = 50 ➔ RF50 ya no tiene utilidad. Poner RF50 en lista de RF libres.
  - Asignar: Retirement RAT(20) = 90. El valor no especulado de r20 queda representado por RF90

Cualquier instrucción que haga escrituras las realiza en la fase de Commit. Por lo tanto un Commit equivale a actualizar la Retirement RAT.

Cuando una instrucción es enviada a las Issue Queues en la etapa de Dispatch siempre busca en la Frontend RAT los valores que necesita como operandos fuente y la modifica en el valor de operando destino (como hemos visto en el ejemplo).



- Implementado en la mayor parte de los procesadores actuales

Hacer Commit: simplemente actualizar la retirement RAT

Suele combinarse con un fichero de registros unificado

Las Issue Q no almacenan operandos; solo información de si están disponibles en los registros (bit Ready).

La identificación de operandos fuente (Source Id 1 y 2) no se hace usando Nº de ROB, sino Nº de Reg Fisico.

Cuando la instrucción se emite a las UFs se hace la lectura de registros

Los resultados del CBD van acompañados del nº de registro destino → se escriben directamente en registros, no en ROB

➢ 3.4 Límites de ILP.

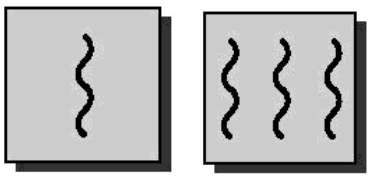
Se pude demostrar que incluso teniendo recursos ilimitados (todos los registros que se quiera, Unidades Funcionales, Memoria, Colas de Instrucciones, etc) hay un límite físico para el rendimiento de los procesadores en el paralelismo a nivel de instrucción

- Se deben mejorar los compiladores y los ISA (Instruction Set Architecture)
- Eliminar los riesgos EDE y EDL referidos a la memoria.
- Eliminar los riesgos LDE en registros y memoria. Predicción.

Para aumentar este rendimiento se buscan nuevos paralelismos en las aplicaciones, de ahí aparece el paralelismos a nivel de hilo, TLP (Thread Level Parallelism).

TLP

Los Threads son unidades de flujo de instrucciones independientes. Como se estudió en Sistemas Operativos un proceso puede tener uno o varios hilos de ejecución.



Cuanto mayor parentesco haya entre hilos más recursos comparten y por ello mejor se puede aprovechar el paralelismo.

El uso de diferentes hilos concurrentes en un procesador nos permite esquivar latencias en memoria o aumentar la densidad de operaciones realizadas en la ventana de ejecución.

En la arquitectura del procesador se denomina MultiThreading a la técnica que explota el TLP.

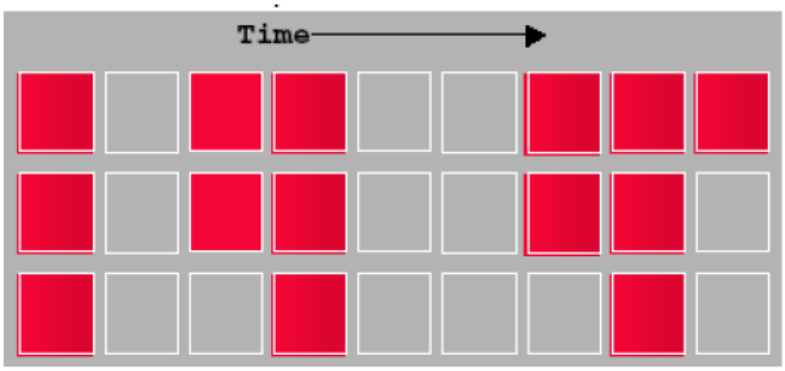
➢ 3.5 MultiThreading

Existe una razón muy poderosa a la hora de la aparición del Multithreading, la latencia de acceso a memoria que se produce en los Load/Store.

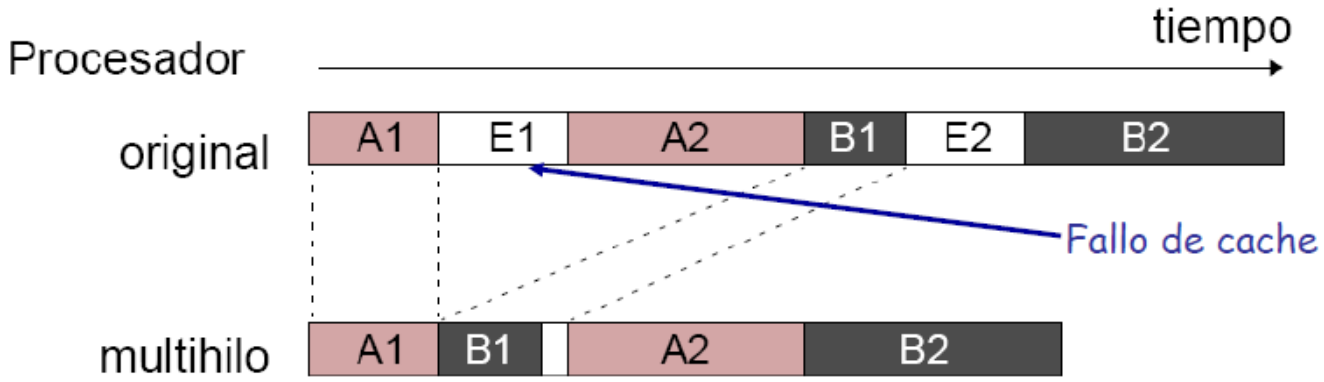
Un procesador Superescalar capaz de realizar 3 operaciones simultáneamente puede estar ralentizado porque los operandos no llegan

❑ Procesador superescalar

Causas de baja utilización de UFs:
- No hay instrucciones que ejecutar (fallo cache)
- Hay instr, pero faltan operandos
- Hay UFs libres, pero no del tipo necesario



Queremos intentar usar esos huecos libres en nuestra ejecución para esquivar las latencias que se producen sobre todo por accesos a memoria.

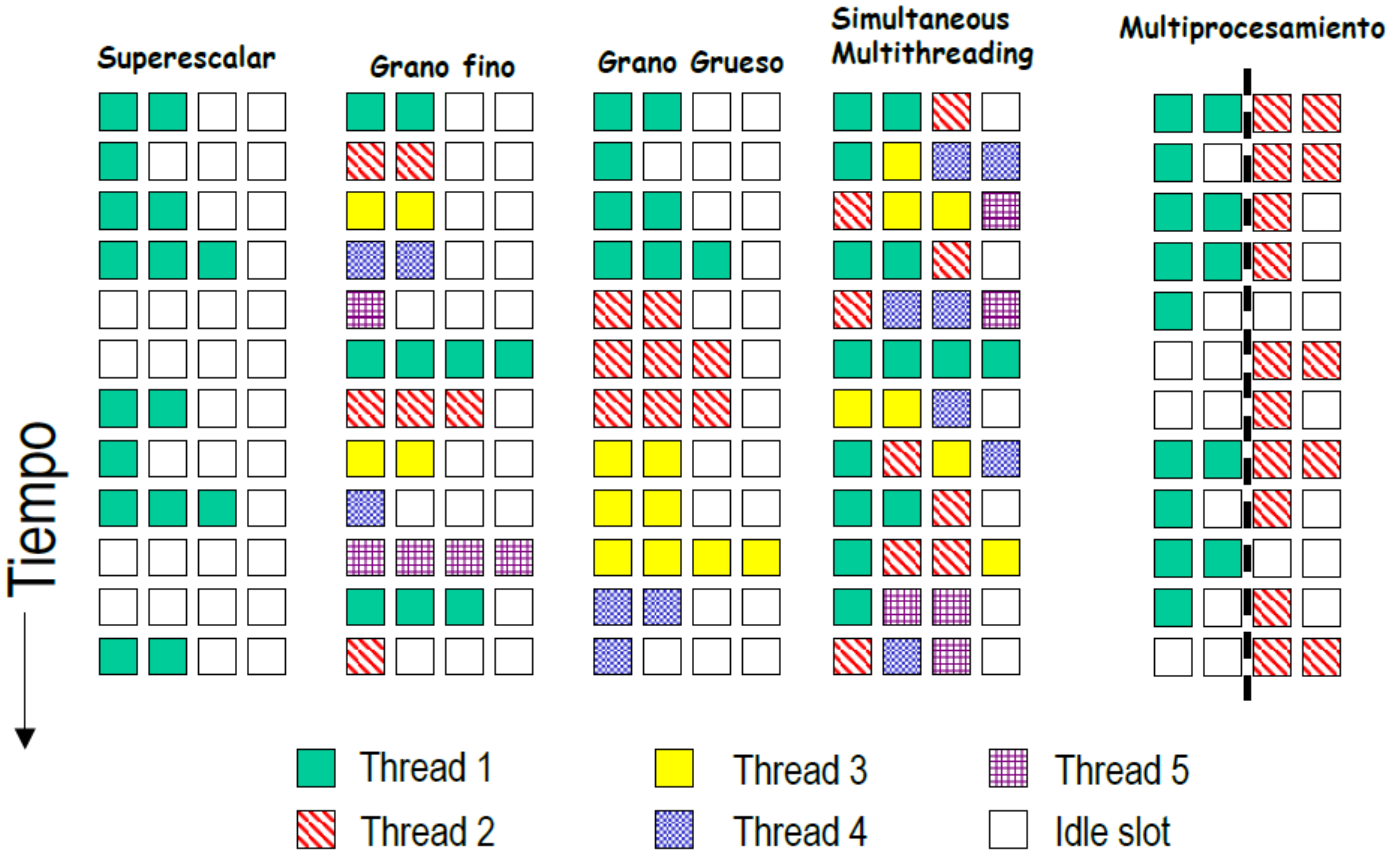


En una ejecución de dos hilos concurrentes A y B podemos entrelazarlos mientras alguno está ocupado en operaciones de larga latencia. En la figura prácticamente se eliminan por completo los tiempos de Espera 1 y 2 (al hacerlos transparentes) y se termina la ejecución antes.

La técnica de Multithreading está lejos de ser una técnica ideal. Tiene costes relativamente grandes a la hora de conmutar entre hilos debido a que el intercambio requiere que se guarde el estado arquitectónico completo del procesador, el PC de cada hilo independiente y otros parámetros.

Si los hilos que se intercambian proceden del mismo proceso siempre es más ligero el cambio. Aunque parezca que la conmutación de hilos pueda tirarnos por tierra el rendimiento global del procesador, se ha comprobado que si está bien diseñado aumenta con respecto a ILP.

- **3.5.1 Técnicas de MultiThreading VS Multiprocesadores.**



Grano Fino: conmuta entre Threads en todos los ciclos, generalmente con la técnica de Round Robin. Los hilos que están bloqueados por una operación de memoria se salen de la cola para no estorbar mientras no disponen de los datos.
->Niagara y Niagara 2.

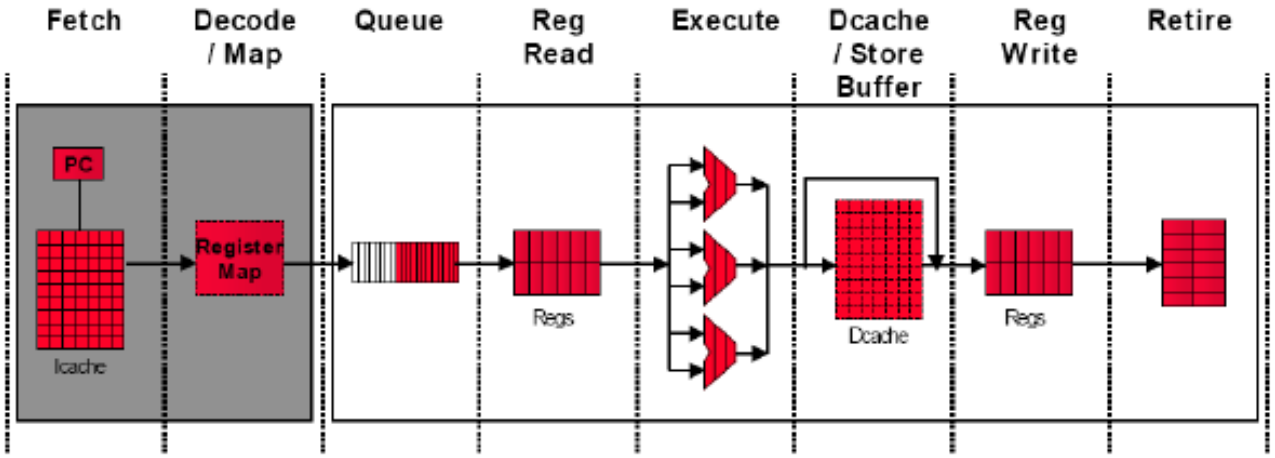
Grano Grueso: conmuta entre Threads solo en caso de latencias muy grandes (como un fallo de Caché L2). Existe también una distribución de los tiempos de uso del procesador según Round Robin.
->IBM AS/400, Itanium2 9000, Sparc64 VI.

SMT: (*Simultaneous MultiThreading*) es la técnica más sofisticada de Multihilo. Mezcla instrucciones de varios hilos a la vez siempre que éstas dispongan de los operandos para ser ejecutadas. Los hilos coexisten en el procesador lo que exige que se mantengan varios PC y se compartan todos los recursos como el banco de registros.
->IBM Power 4, IBM Power 5, IBM Power 8.

Multiprocesadores: el concepto nos indica que hay diferentes cores en el chip completamente independientes. En principio estos cores pueden ser algo más pequeños que un core simple si queremos compararlo en área. Cada uno trata un hilo diferente como si fuera un procesador superescalar independiente. En la práctica podemos hacer que los cores gestionen a su vez multihilo y producir un nuevo aumento del rendimiento. Los Multiprocesadores se verán más adelante.

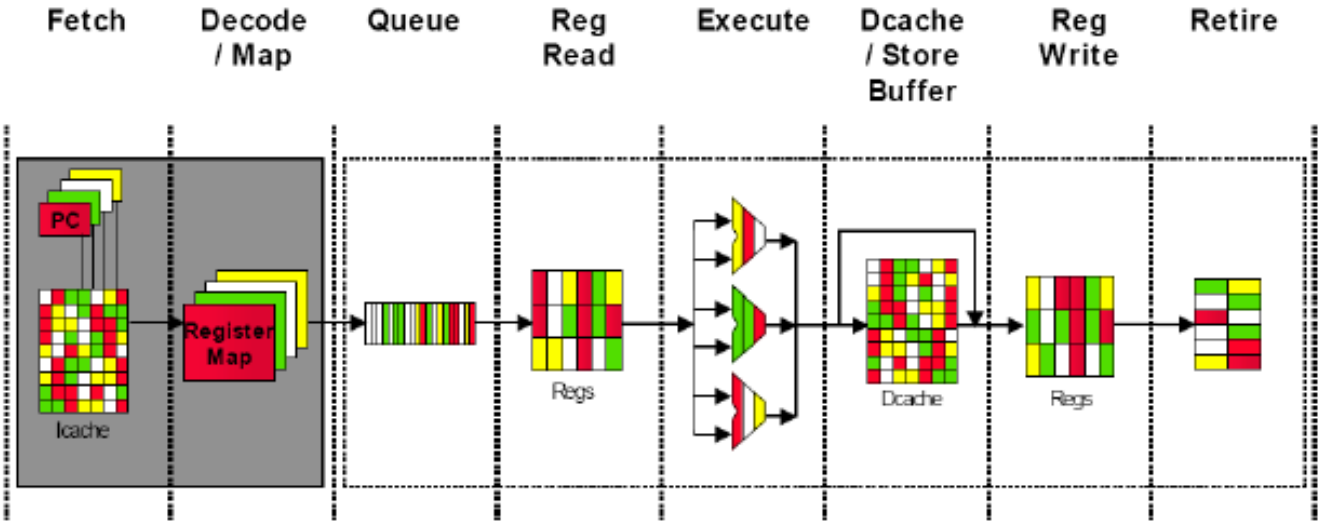
- **3.5.2 SMT**

❑ Un solo hilo: un flujo de instrucciones



✓ todos los recursos utilizados por un hilo

❑ Multihilo: varios flujos de instrucciones



✓ recursos para distinguir el estado de los hilos

✓ los otros recursos se pueden compartir