Arquitectura de Computadores

TEMA 3

Lanzamiento múltiple, Límites de ILP, Multithreading

DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

Curso 2020-2021

Contenidos

- Introducción: CPI < 1
- Lanzamiento múltiple de instrucciones: Superescalar
  - Superescalar con planificación estática.
  - Superescalar con planificación dinámica (Out-of-Order):
    - Anatomía del pipeline: alternativas
    - Caso de estudio
- Límites de ILP
- Ejemplo: Implementaciones x86, evolución μ-arquitecturas Intel
- Thread Level Parallelism y Multithreading
- Bibliografía
  - Básica:
    - Capítulo 3 de Hennessy & Patterson 5th ed., 2012
    - Capítulos 3-8 de González, Latorre & Magklis, 2011. ISBN: 9781608454532
      Versión electrónica disponible en la Biblioteca de la UCM
  - Complementaria:
    - Capítulo 3 de Dubois, Annavaram & Stenstrom, 2012

Introducción: CPI<1

- ¿Por que limitar a una instrucción por ciclo?
- Objetivo: CPI < 1 ➔ Lanzar y ejecutar simultáneamente múltiples instrucciones por ciclo
- ¿Qué entendemos por **procesador escalar**?
  - No puede ejecutar más de una instrucción en, al menos, una de sus etapas ➔ Su CPI máximo ideal es 1
- **Superescalar**: No existe la anterior restricción.
- ¿Tenemos recursos?
  - Más transistores disponibles (Ley de Moore)
  - Técnicas para resolver las dependencias de datos (planificación)
  - Técnicas para resolver las dependencias de control (especulación)

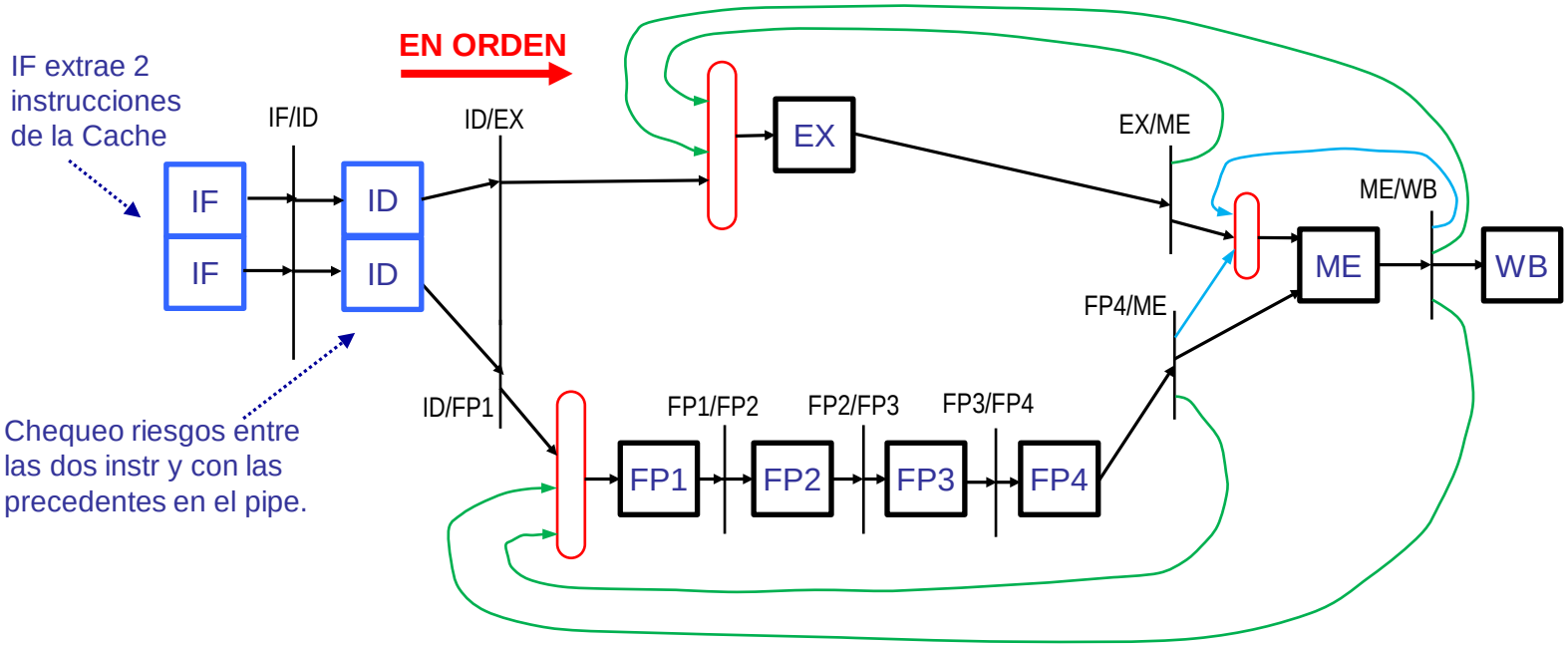
Más ILP: Lanzamiento múltiple

Alternativas

- Procesador Superescalar con planificación estática:
  - En general, chequea riesgos en etapa de ID: Capacidad para chequear conflictos de varias instrucciones a la vez.
  - Si una instrucción presenta conflictos ➔ bloquea instrucción conflictiva y siguientes
  - Implementaciones típicas: 2 instrucciones a la vez (entera, salto, memoria + punto flotante)
  - Caso particular: Very Long Instruction Word (VLIW)
    - La etapa ID no chequea riesgos
    - El compilador conoce el HW disponible en la etapa de ejecución y sus latencias de uso ➔ Forma "paquetes" agrupando varias instrucciones independientes que se descodifican y se lanzan a ejecución simultáneamente.
    - El número de instrucciones en cada paquete (p. ej. 3) es fijo. Si el compilador no encuentra suficientes instrucciones libres de riesgos para formar un paquete, rellena con NOP.
    - Puede adaptarse bien al ámbito de los sistemas empotrados.
    - Ref para estudiantes interesados: Hennesy & Patterson, 5th ed. 2012, Sección 3.7 y Apéndice H.
- Procesador Superescalar con planificación dinámica
  - Analiza dependencias de varias instrucciones a la vez.
    - Si conflictos ➔ manda a colas de espera.
    - Puede mandar a las unidades funcionales (UFs) un número variable de instrucciones por ciclo (en función de la disponibilidad de operandos y de UFs)
  - Decodificación en orden → Ejecución en desorden → Finalización en orden

Superescalar con planificación estática

- Procesador Superescalar con planificación estática
  - Ejemplo: Completando las etapas que solamente admitían una instrucción
    - Las etapas IF e ID pueden procesar dos instrucciones por ciclo
    - Si una es de FP y la otra de tipo ENT pueden enviarse a ejecución en un mismo ciclo: Riesgos?
    - Dependencias entre ENT y FP: instrucciones de LD/ST. Efecto de las dependencias provocadas por Load!
    - Incremento de los problemas provocados por los saltos

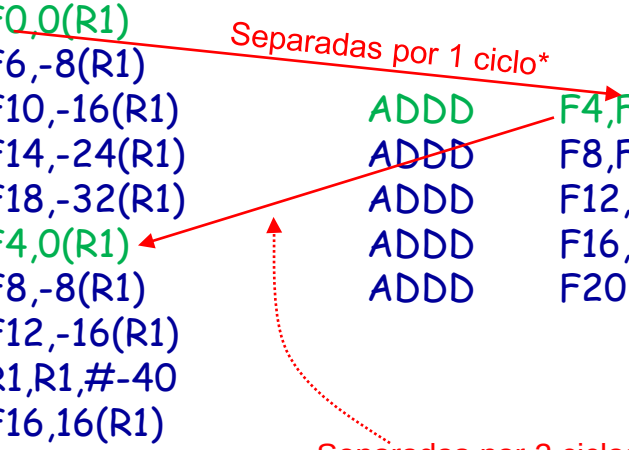


- En la etapa ME solamente el camino superior puede acceder realmente a la memoria. El otro la puentea.
- En la etapa WB puede finalizar una instrucción de FP y una ENT si escriben en bancos de registros separados.

Ejemplo: Suma vector más escalar

- Desarrollo para ejecución superescalar: se desarrolla una iteración más.
  - 5 iteraciones en 12 ciclos ➔ 2.4 ciclos por iteración
  - CPI = 12/17= 0.706 IPC = 1/CPI = 1.417
- El código máquina está compacto en la memoria
  - Instrucciones INT y FP intercaladas. El hw de lanzamiento detecta si puede lanzar una o dos instrucciones.

	Instrucción entera		Instrucción FP		Ciclo
Loop:	L.D	F0,0(R1)			1
	L.D	F6,-8(R1)			2
	L.D	F10,-16(R1)	ADDD	F4,F0,F2	3
	L.D	F14,-24(R1)	ADDD	F8,F6,F2	4
	L.D	F18,-32(R1)	ADDD	F12,F10,F2	5
	S.D	F4,0(R1)	ADDD	F16,F14,F2	6
	S.D	F8,-8(R1)	ADDD	F20,F18,F2	7
	S.D	F12,-16(R1)			8
	DADDIU	R1,R1,#-40			9
	S.D	F16,16(R1)			10
	BNE	R1,R2,Loop			11
	S.D	F20,8(R1)			12



* Recordar: latencias de uso (Tema 2, transp. 13)

Superescalar con planificación estática

➢ *Ventajas*

- No modifica código. Compatibilidad binaria.
- No riesgos en ejecución: si hay riesgo ➔ bloqueo de lanzamiento

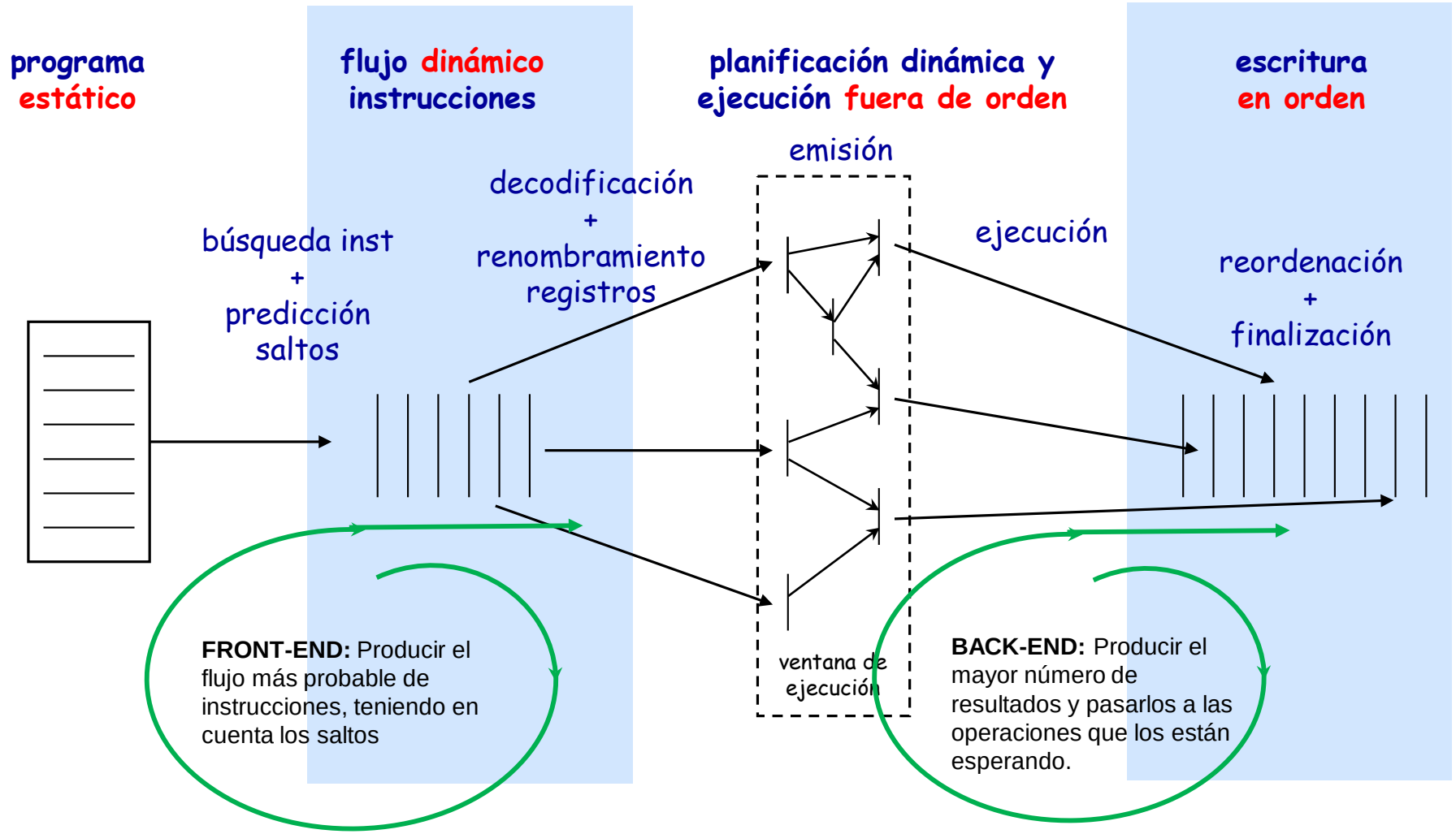
➢ *Desventajas*

- Mezcla de instrucciones. Solo obtiene CPI de 0.5 en programas con 50 % de FP
- Bloqueos en el lanzamiento (igual que en MIPS básico)
- Planificación fija: No puede adaptarse a cambios en ejecución (p.ej. fallos de cache)
- Los códigos deben replanificarse para cada nueva implementación (razones de eficiencia)

Superescalar con planificación dinámica

❑ Idea Básica

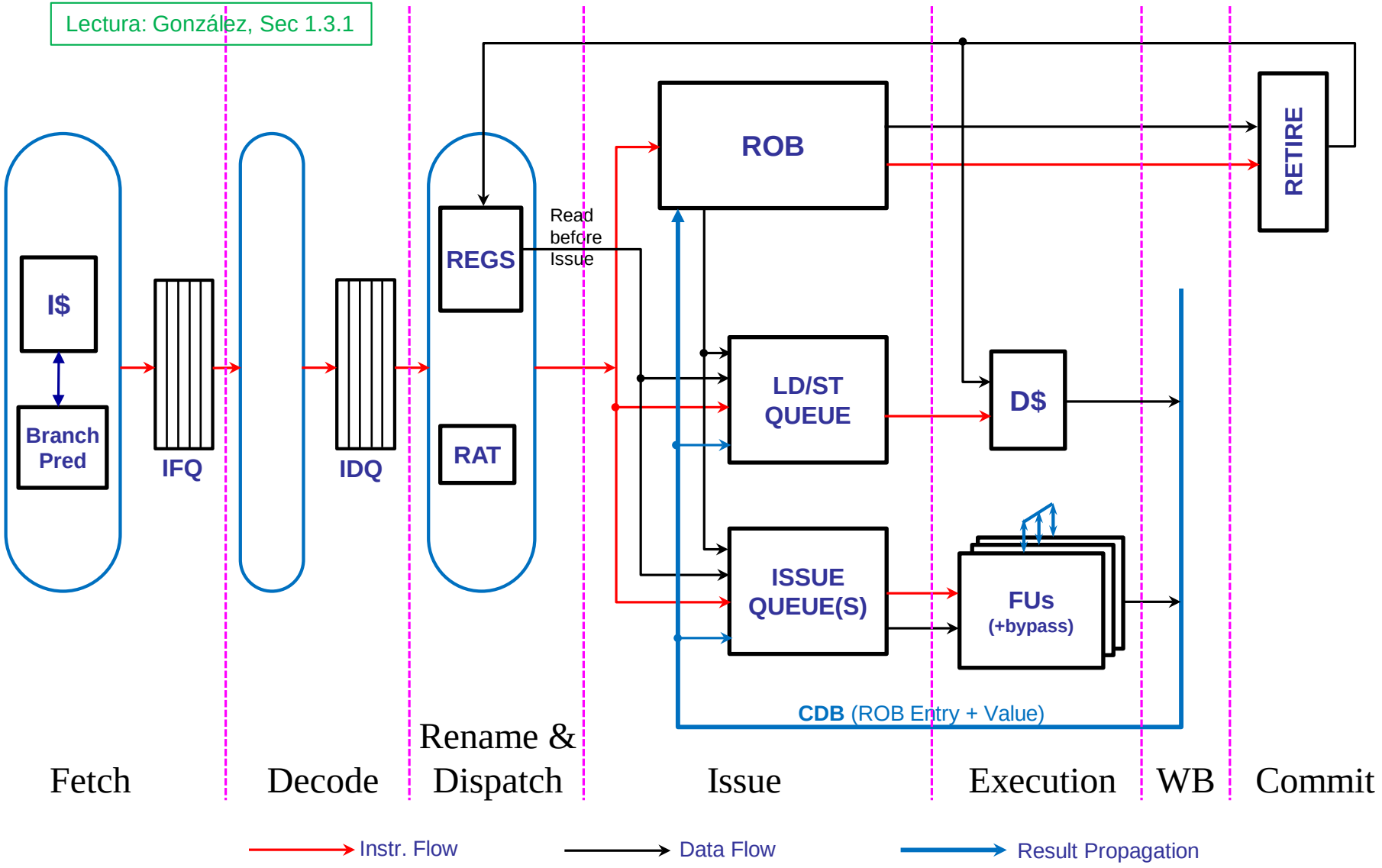
Ejecución fuera de orden. Finalización en orden



Superescalar con planificación dinámica

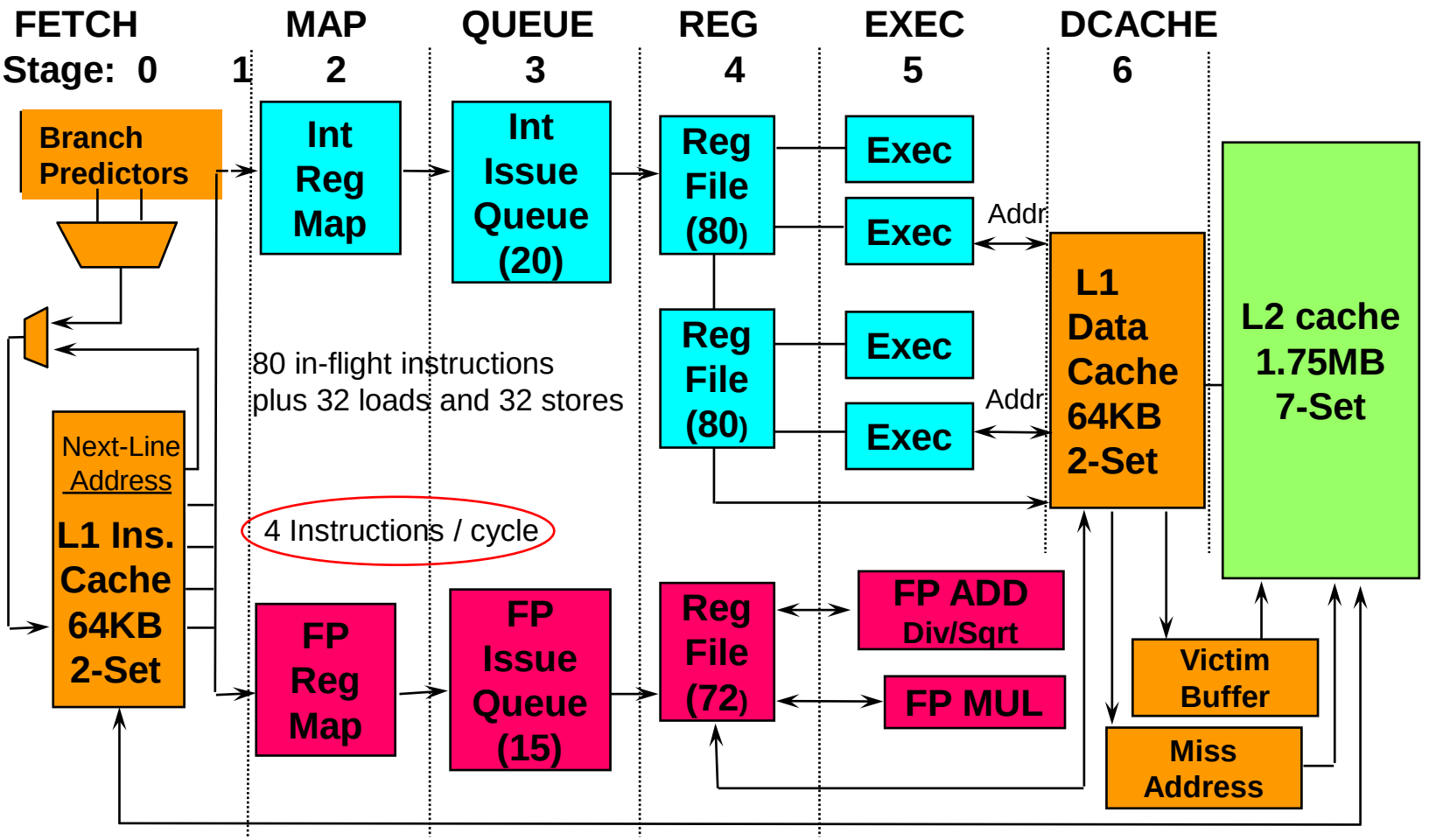
- Esquema general de organización en etapas (varias alternativas posibles)

Lectura: González, Sec 1.3.1



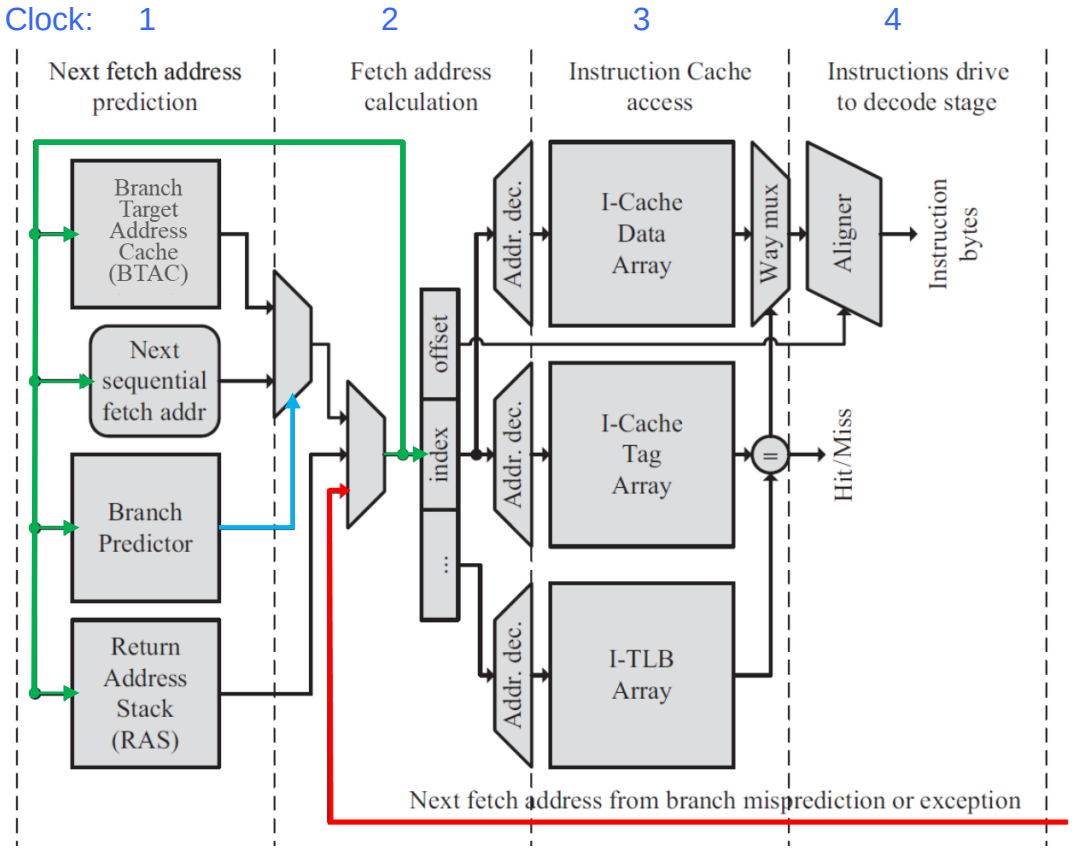
Superescalar con planificación dinámica: Ejemplo

- EV7 ALPHA 21364 Core (2003)



Superescalar con planificación dinámica: Fetch (1)

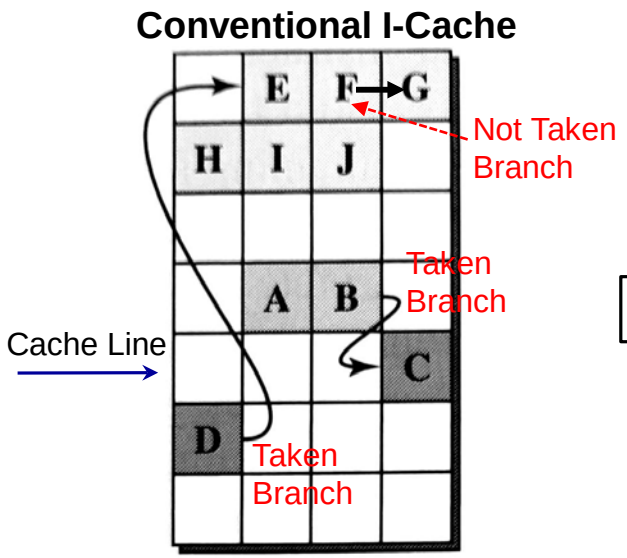
- El diseño mostrado trabaja en cuatro ciclos para sostener altas frecuencias de reloj
  - … Pero cada ciclo puede comenzar una nueva búsqueda
- La dirección enviada a la I-Cache (línea verde) se usa también en los predictores
- Lectura de varias instrucciones por ciclo: sencillo cuando las instrucciones están en la misma línea de cache. Efecto de los saltos!



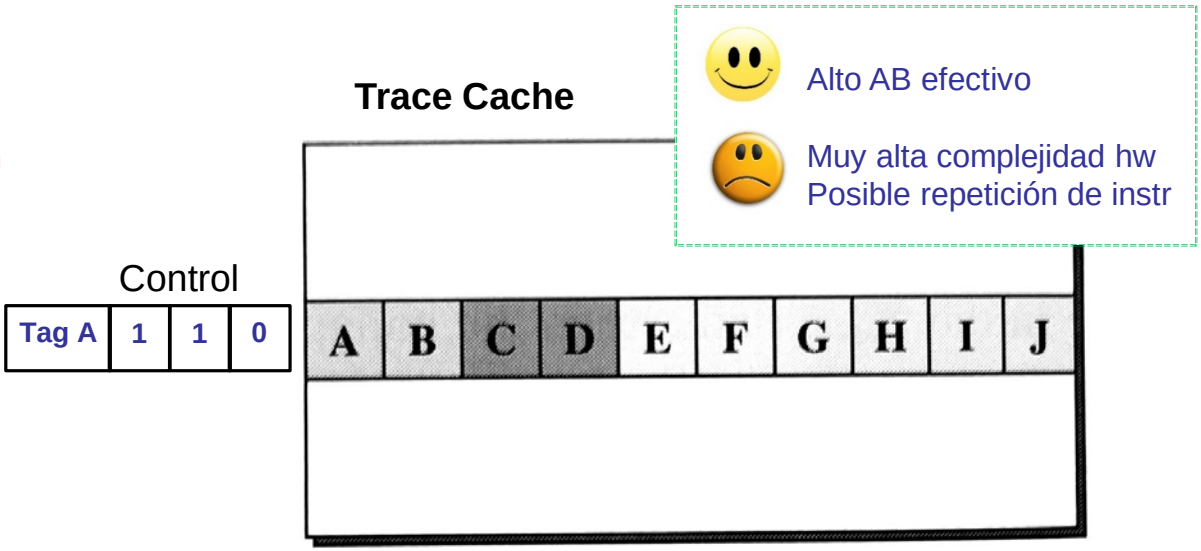
de González: Fig. 3.1

Superescalar con planificación dinámica: Fetch (2)

- Cache convencional. Acceso a grupos de instr consecutivas en memoria (p.ej. cache line) ➔ Un salto tomado (incluso bien predicho) hace inútil el resto del grupo.
- **Idea alternativa: Trace Cache.** Almacenar grupos de instrucciones consecutivas en el flujo dinámico del programa (traza). Usada en Pentium 4.
  - Control (directorio): TAG de la 1ª instr. y comportamiento de las instrucciones de salto (T/NT)
  - El predictor debe predecir varios saltos a la vez
  - CACHE HIT: El TAG de la siguiente dirección a buscar y el comportamiento predicho para los siguientes saltos coincide con Control de una línea de la cache.
  - Problema: Identificar y guardar trazas.



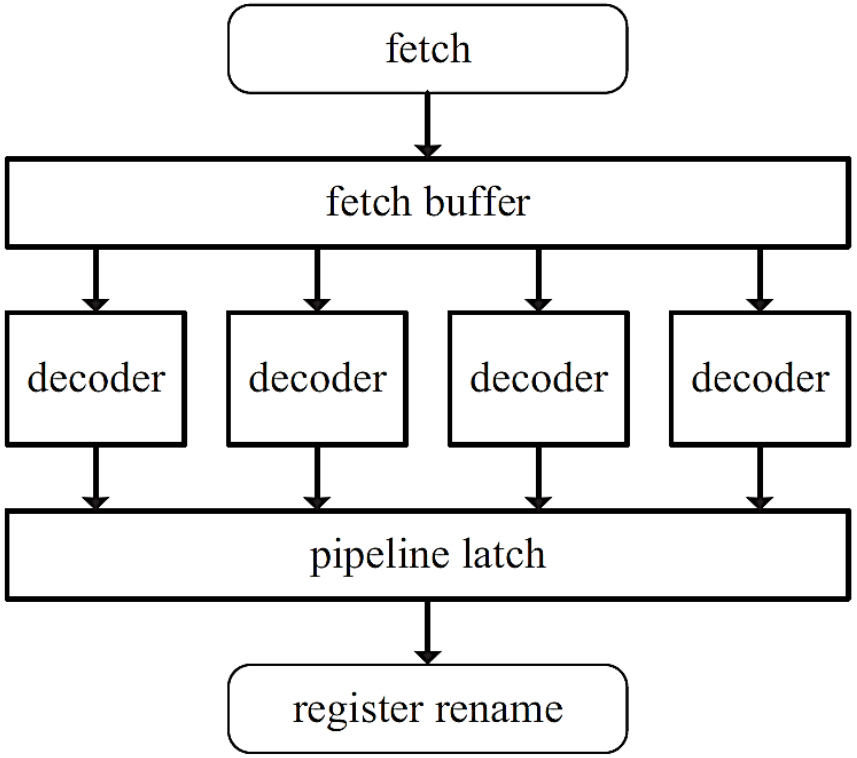
Secuencia de instr **A-J** (con 2 saltos tomados: B y D) en 5 líneas de cache → 5 ciclos



La misma secuencia de instr almacenada en posiciones consecutivas → Acceso posible en 1 ciclo

Superescalar con planificación dinámica: Decode (1)

- Entrada: Cadena de bytes extraídos de la cache
- Salida: Instrucciones separadas + señales control para el pipe.
- Decodificación en RISC: longitud de instrucción fija + pocos formatos ➔ Proceso "relativamente" sencillo
  - Puede consumir un solo ciclo de reloj



Decodifican 4 instrucciones en paralelo, que se pasan a la siguiente etapa

de González: Fig. 4.1

Superescalar con planificación dinámica: Decode (2)

- Segmentación con repertorios de instrucciones complejos (CISC): Caso x86.
  - ¿Como segmentar un repertorio con instrucciones entre 1 y 17 bytes?

    - Dónde empieza cada instrucción? Depende del tipo (Opcode), pero…
    - El Opcode no está siempre en la misma posición
    - El Opcode no siempre tiene la misma anchura
    - Instrucciones que mezclan operandos en registros y memoria

- IDEA! Traducción dinámica de instrucciones:
  - El flujo dinámico de instrucciones x86 se traduce en la etapa decode a un flujo de instrucciones tipo RISC
    - Primeras implementaciones: Intel P6 (Pentium Pro-Pentium III), AMD K5
    - En P6 se traducen las instrucciones originales x86 a "μ-operaciones" de 118 bits
    - Cada instrucción original se traduce a una secuencia de 1 a 4 μ-operaciones
    - Formato μ-op: código + 3 operandos. Modelo: load/store. Muy descodificadas
    - Pero … La instrucciones más complejas son traducidas por una secuencia almacenada en una memoria ROM (8Kx118) (microprograma)
    - Bucles: posibilidad de ejecutar μ-ops repetidamente. Sucesivas iteraciones no requerirían traducción dinámica.

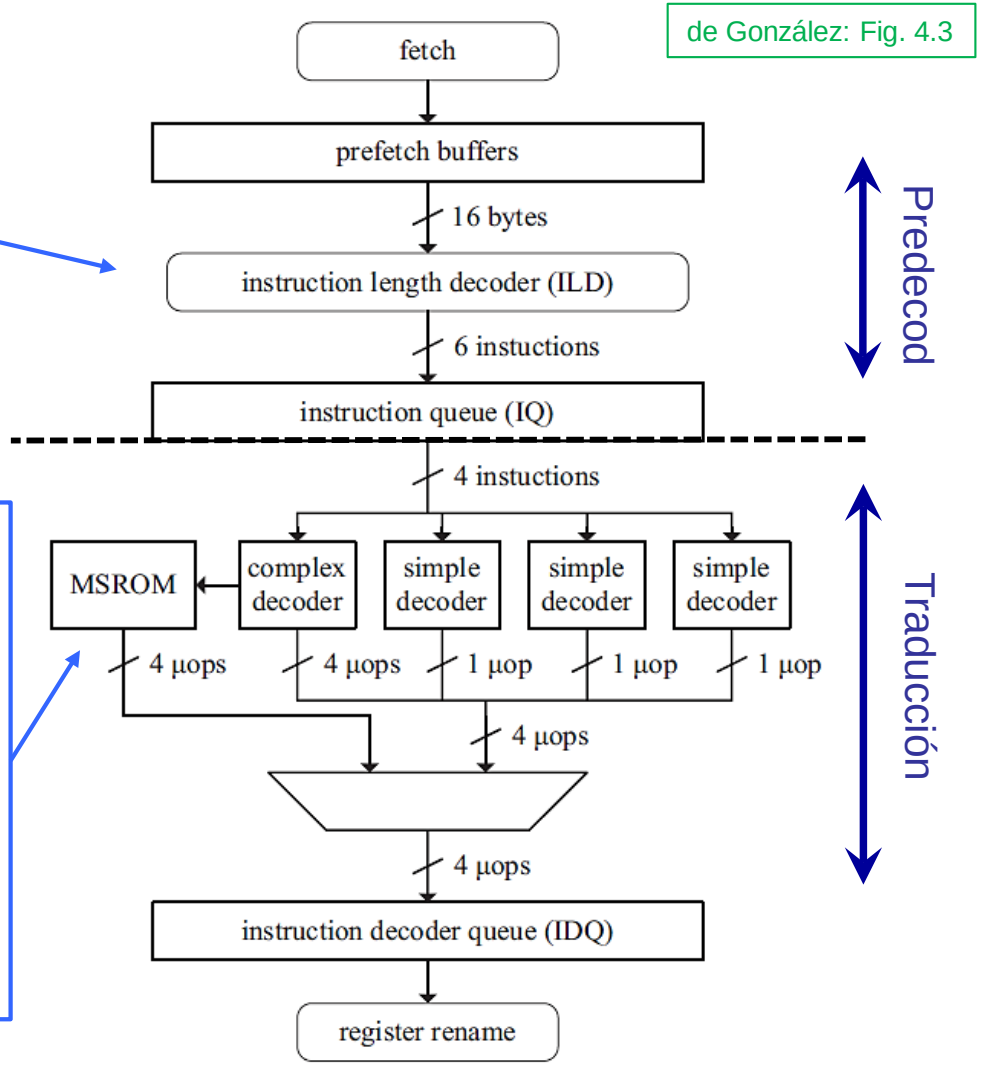
Superescalar con planificación dinámica: Decode (3)

❑ Pipe de decodificación de Intel Nehalem (2008)

de González: Fig. 4.3

Convierte el flujo de bytes en un flujo ordenado de instrucciones (determinando la long de cada una).

En la mayoría de los casos puede trabajar en un solo ciclo. Para algunas instr puede necesitar hasta 6 ciclos.



Toma instrucciones de la IQ y las traduce usando:
- Simple decoders (3): Una instr. x86 → Una µ-op
- Complex decoder: Una instr. x86 → hasta 4 µ-op
- Secuenciador (MSROM): Para instr. x86 más complejas.

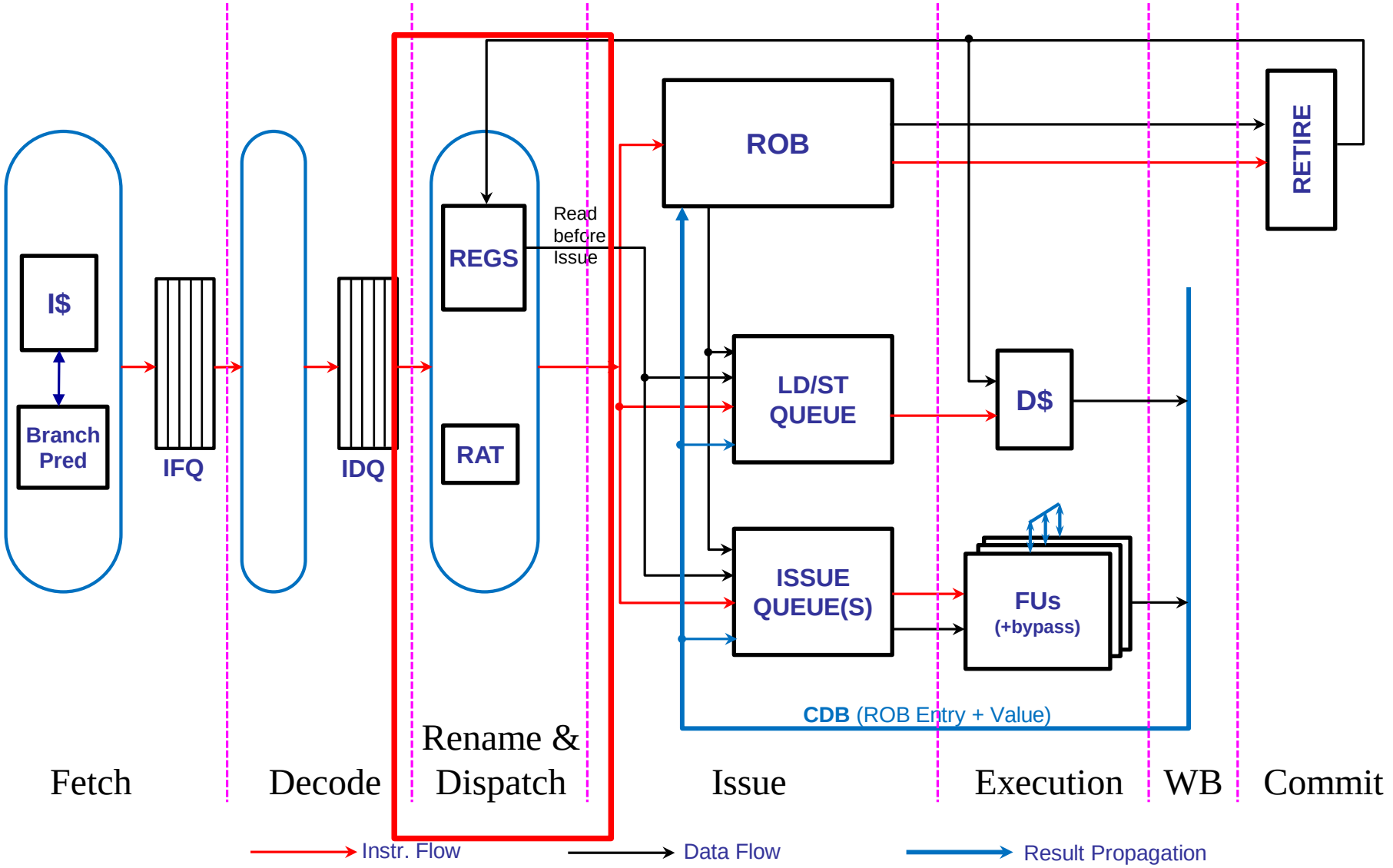
Enfoque adecuado si la mayoría de las instr x86 se traducen en una sola µ-op. Es realista?

↓

Sí: Relación µ-ops / instr x86 en SPEC, típicamente <1,5

Superescalar con planificación dinámica

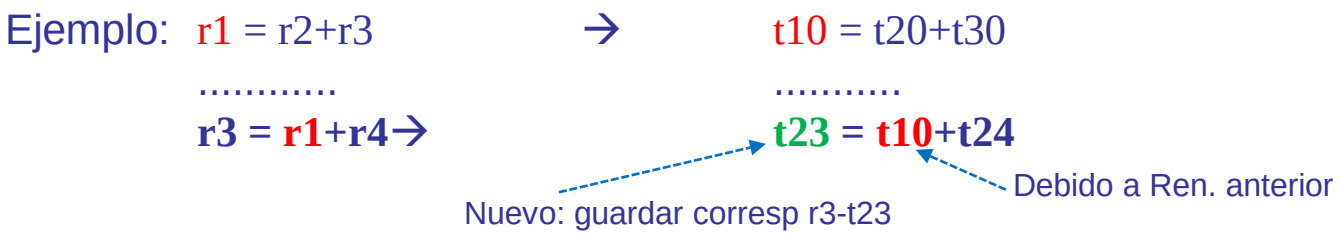
- Esquema general de organización en etapas (varias alternativas posibles)



Superescalar con planificación dinámica: Rename & Dispatch (1)

Rename

- Asignar los elementos de almacenamiento que van a usarse para guardar el resultado (destino) de una instrucción.
  - Además: cambiar los nombres de los registros fuente, teniendo en cuenta los renombramientos de las instr anteriores (tabla)



  - ¿Cuándo podemos volver a reusar un registro (p.ej. t10) para guardar el resultado de otra instr?
- Alternativas de diseño
  - Mediante el ROB: entradas del ROB guardan resultados especulados
  - Mediante un Buffer de Renombramiento
    - Los resultados especulados no se guardan en el ROB
    - Buffer adicional para resultados
  - Fichero de registros unificado
    - Solo existen los registros físicos; parte de ellos representan a los registros arquitectónicos

Superescalar con planificación dinámica: Rename & Dispatch (2)

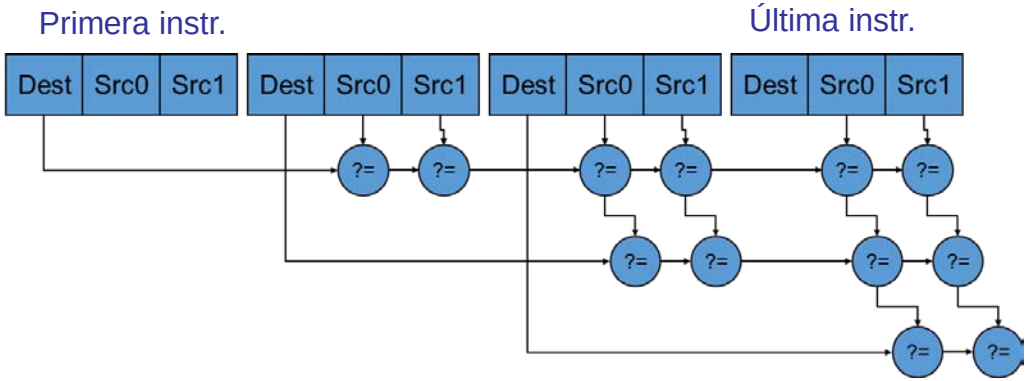
Dispatch

- Reservar entradas en ROB, Issue Queue(s) (y LD/ST Q para las instr de carga/almacenamiento).
  - La Issue Queue se corresponde con el concepto de Estación de Reserva definido por Tomasulo en el IBM 360/91.
  - Aunque en los procesadores modernos esta fase se llama "dispatch", se corresponde con lo que en Tomasulo se denominó "issue".
  - El término "issue" (emisión) se reserva para la fase en que una instrucción se emite hacia la correspondiente UF para comenzar la ejecución.
- Lectura de operandos. Alternativas:
  - Lectura antes de issue (Read before issue). Como Tomasulo
    - Operandos disponibles → copiados a Issue Q. Si no disponibles → copiar identificador del operando (nº de ROB / nº de reg físico).
  - Lectura después de issue (Read after issue)
    - Se pospone la lectura de operandos al momento de la ejecución

Superescalar con planificación dinámica: Rename & Dispatch (3)

- Renombrado de un grupo de instrucciones en paralelo
  - El problema: las tablas de renombrado solamente contienen información sobre las instrucciones de los grupos anteriores
  - ...pero la 2ª instr del grupo debe tener el cuenta el renombrado de la 1ª. Y la 3ª, el de las los dos primeras...
  - Alternativas:
    - Renombrado secuencial: Multiplicar (en la medida de lo posible) la frecuencia de reloj de esta etapa por un factor igual al nº de instr. a renombrar
    - Renombrado combinacional: teniendo en cuenta los operandos y destinos de todas las instr del grupo.

Chequeos necesarios para renombrar un grupo de instrucciones

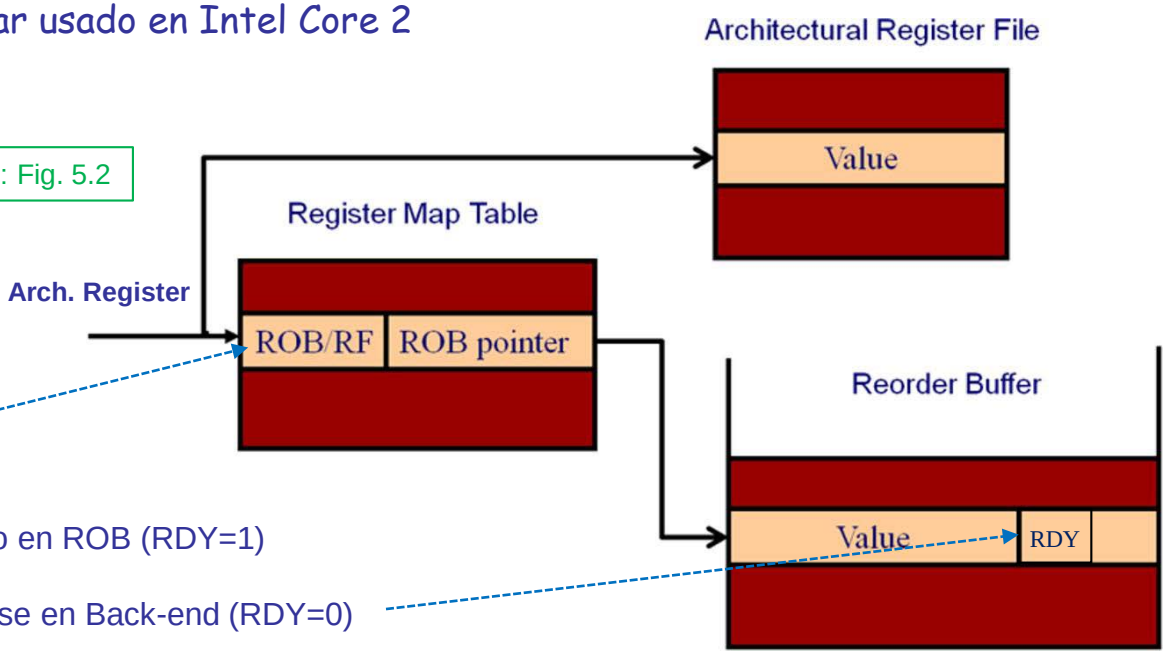


Pensar: ¿Qué ocurre si Src0 de 4ª instr coincide con Dest de 1ª instr. y con Dest de 3ª instr.?

Superescalar con planificación dinámica: Rename & Dispatch (4)

- Renombrado mediante el ROB
  - Se ajusta en líneas generales a lo estudiado en el Tema 2. ¿Cuál es la correspondencia entre las implementaciones?
    - Esquema similar usado en Intel Core 2

de González: Fig. 5.2



Register Map Table

Para cada reg. arquitectónico, indica dónde encontrar su valor más reciente:

ROB/RF
- 1 → Valor en Registro
- 0 →
  - Valor especulado en ROB (RDY=1)
  - Valor calculándose en Back-end (RDY=0)

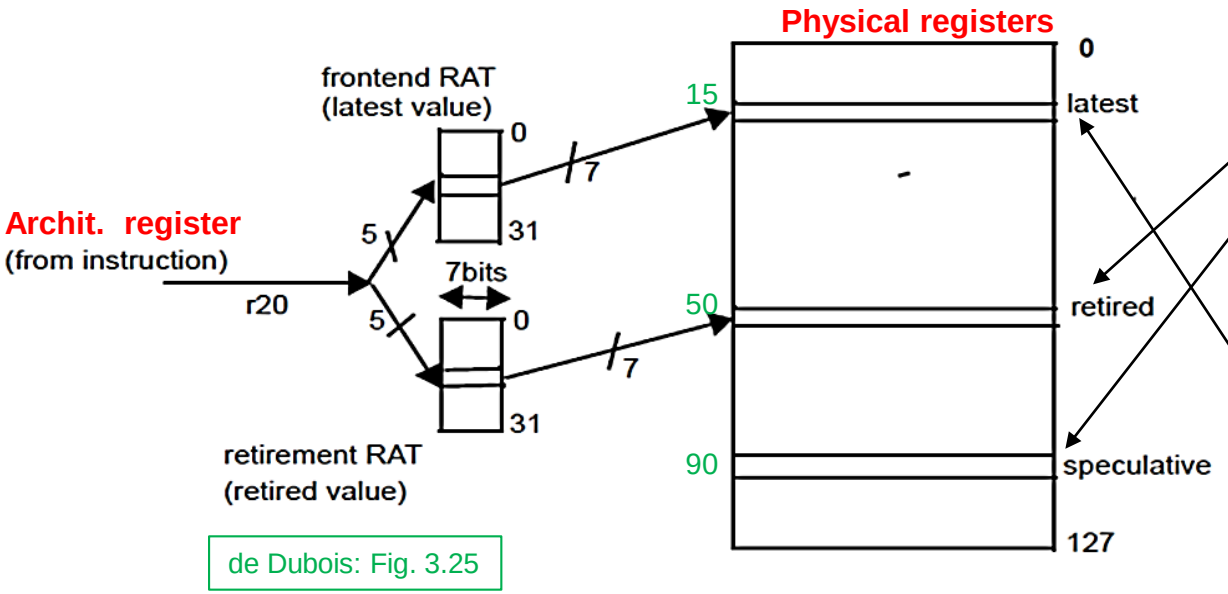
- ALTERNATIVA: Buffer de Renombramiento (esquema usado en IBM Power3)
  - No todas las instrucciones producen un resultado → Desperdicio de espacio en el ROB.
  - Idea: En el ROB se guardan punteros a un conjunto auxiliar de registros (Buffer de Renombramiento) en lugar de valores.
  - Los valores se guardan en el Buffer de Renombramiento
    - Asignar entradas en dispatch, liberar en commit

Superescalar con planificación dinámica: Rename & Dispatch (5)

- Fichero de registros unificado
  - Los resultados, especulados o no, se guardan en un único fichero de registros (registros físicos, RF), con un nº registros > nº registros arquitectónicos (RA).
  - El ROB se limita a mantener el orden de las instr, pero no almacena valores.
    - Ventaja: en la fase commit no hay que copiar resultados del ROB a registros.
    - … pero cada entrada del ROB guarda el nº de RF donde se debe almacenar el resultado de la operación y el nº del correspondiente RA. (**Commit**)
  - Los RA se mapean dinámicamente sobre los RF.
    - En cualquier momento un mismo RA puede estar mapeado sobre varios RF.
  - Debe existir un RF encargado el almacenar el último valor asignado no especulativamente (en commit) a cada RA
  - Un RF quedará libre (reutilizable) cuándo contenga un valor que ya no vaya a ser utilizado: mantener lista de RF libres ¿Cuándo podemos estar seguros?
    - Si RFi se está usando para contener el valor no especulado de RAj y se ejecuta la fase commit de una instrucción cuyo reg destino es tb RAj ➔ el valor que hay RFi ya no se necesita ➔ RFi puede reutilizarse.

Superescalar con planificación dinámica: Rename & Dispatch (6)

- Fichero de registros unificado. Ejemplo: 32 RA y 128 RF
  - **Frontend RAT:** Para cada RA, indica el RF asignado la última vez que dicho RA actuó como destino de una instr. + Bit RDY
  - **Retirement RAT:** Para cada RA, indica el RF que contiene su último valor no especulado (puede ser el mismo)



de Dubois: Fig. 3.25

Ejemplo: **r20** ha actuado como destino de tres instrucciones relevantes:

La 1ª ya ha hecho commit (repr. RF50)

La 2ª es especulada. El RF asignado puede contener un valor necesario para otras instrucciones (repr. RF90)

La 3ª es la última instrucción, por el momento, que tiene a r20 como destino (repr. RF15)

Instr posteriores que usen r20 como operando: hallar en frontend RAT si valor RDY y cuál es el RF asociado

- A cada reg destino se le asigna un RF que esté libre (actualizar la frontend RAT).
  - Este nº de RF actuará como TAG (en lugar del nº del ROB asignado)
- Para los registros operando: Encontrar RF asociado a cada operando en frontend RAT.
- Si valor disponible, mandar a Issue Q (lectura antes de issue). Si no, mandar nº de RF (tag)
- Commit: Significa básicamente actualizar la retirement RAT: Qué fila? Qué pasa con el contenido anterior de esa fila?

Superescalar con planificación dinámica: Rename & Dispatch (7)

Ejemplo(cont.): Supongamos el programa:

```
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

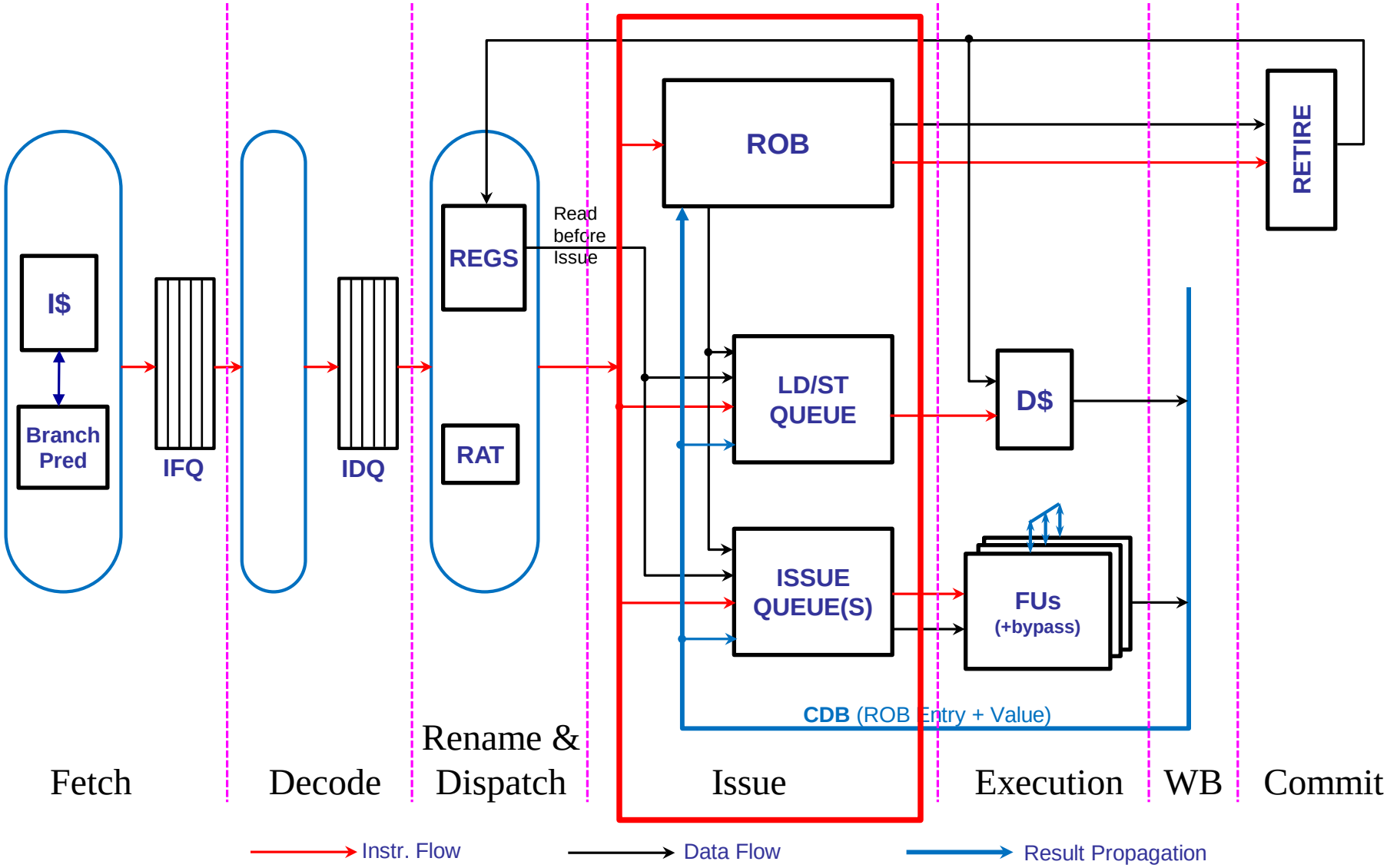
Dispatch Cod3: Hallar un RF libre para asignar a r20.
Sup que se elige RF15 ➔ Asignar: Frontend RAT(20)=15

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

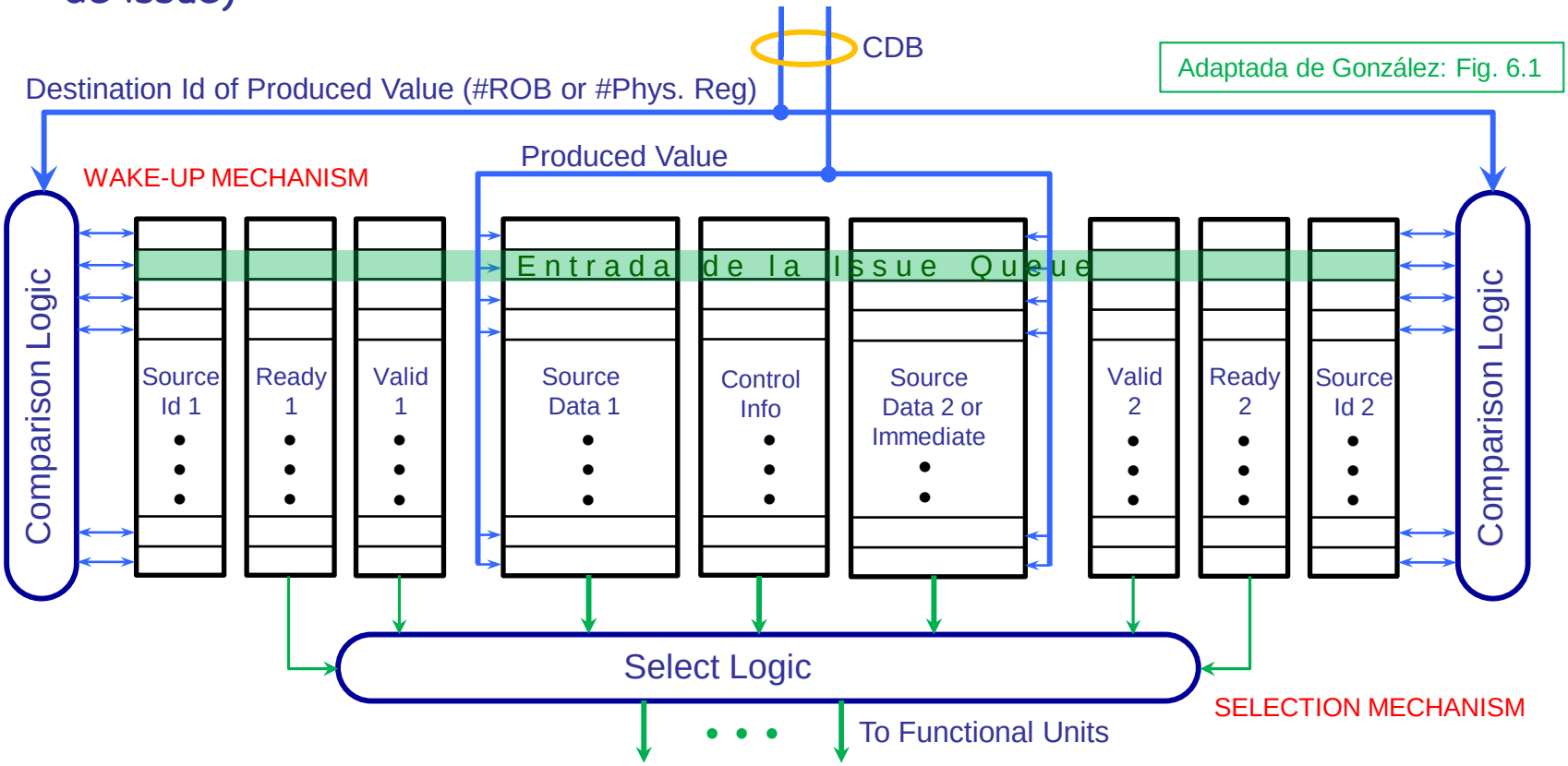
Superescalar con planificación dinámica

- Esquema general de organización en etapas (varias alternativas posibles)



Superescalar con planificación dinámica: Issue (1)

- Esquema general de una Issue Queue (con lectura de operandos antes de issue)

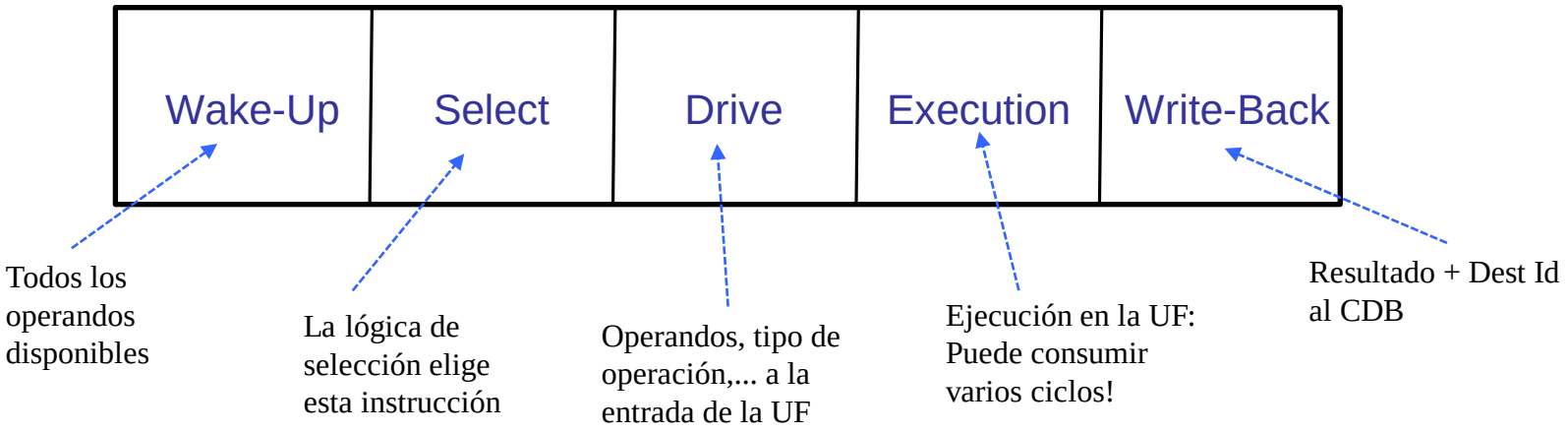


- WAKE-UP: ∀ resultado sobre CDB comparar su Dest Id con Src Id (1 y 2) de todas las entradas de la Issue Q
  - Si coincidencia y Valid bit=1 ➔ Copiar valor y Ready bit←1
- Posibilidad de usar más de un CDB ➔ aumento del nº de comparaciones simultáneas
- Adaptación a "Lectura después de issue (Read After Issue)". Como?

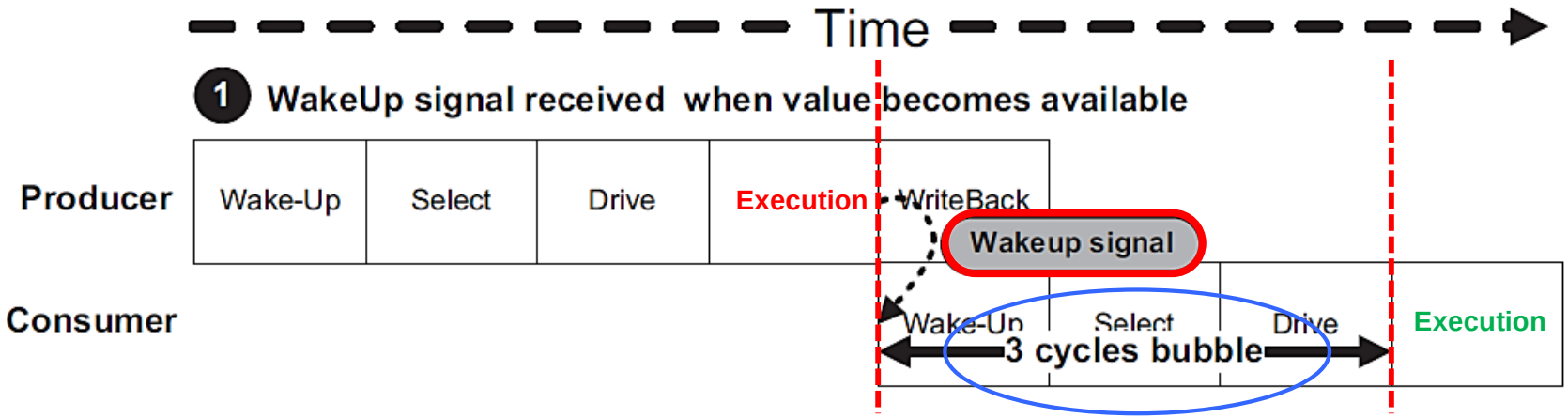
- SELECTION: Tomar un subconjunto de las instrucciones que están listas en la Issue Q y llevarlas hasta la entrada de las UFs (disponibles y adecuadas al tipo de operación)

Superescalar con planificación dinámica: Issue (2)

- De wake-up a write-back...

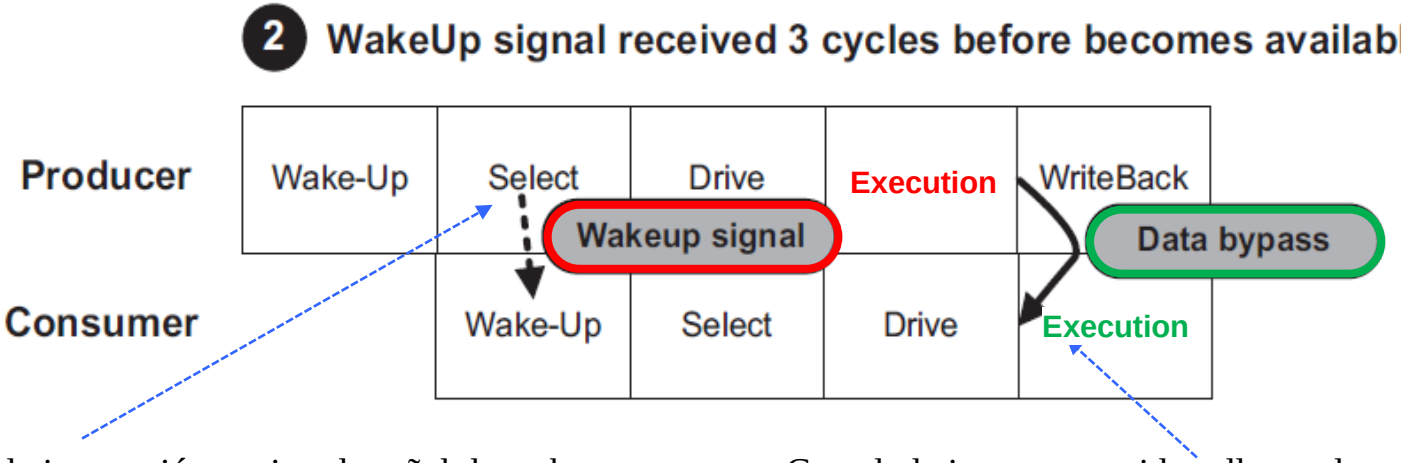


- El problema: penalización en instr próximas con dep LDE



Superescalar con planificación dinámica: Issue (3)

- Alternativa: wake-up anticipado.
  - Las instr aritméticas tienen un tiempo de ejecución conocido de antemano.
  - Despertar la instr consumidora varios ciclos antes (dependiendo de la latencia de la productora)



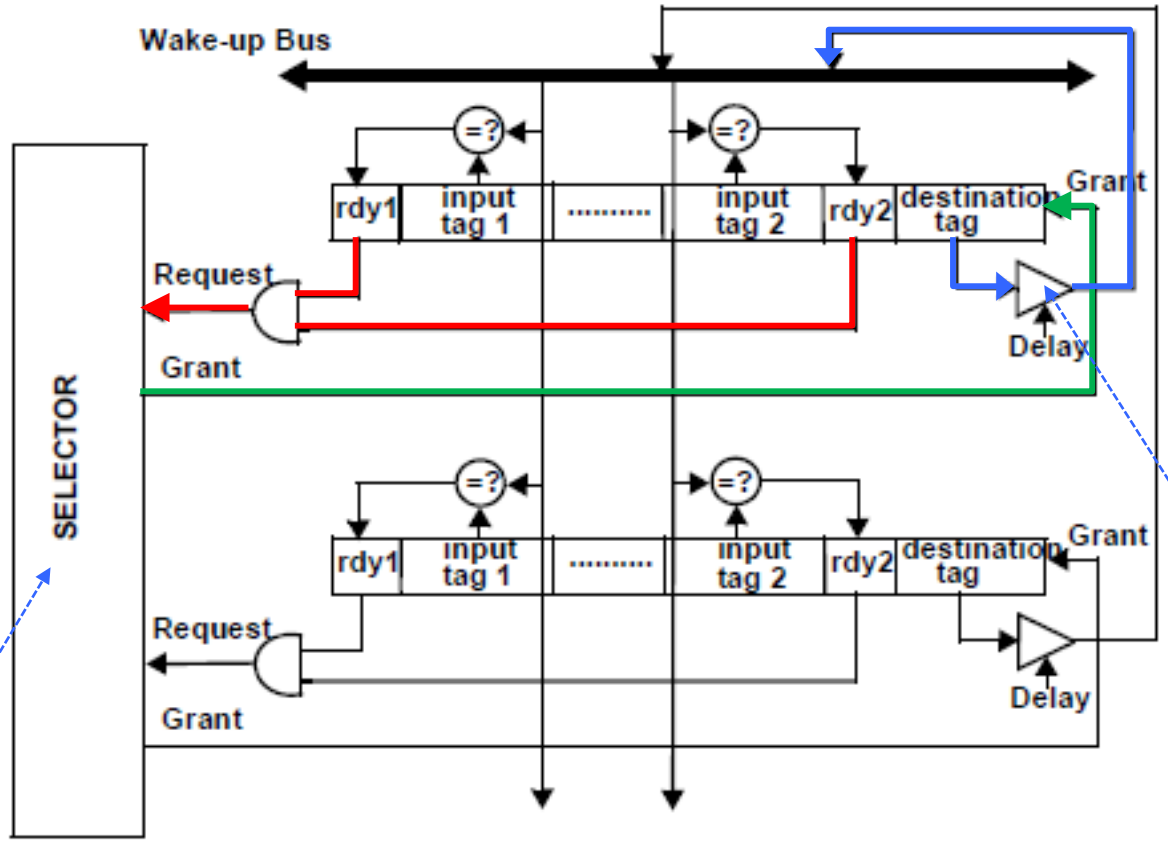
Al seleccionar la instrucción, activar la señal de wake-up (sup: ejecución en 1 ciclo). Si la ejecución consume N ciclos, generar el wake-up N-1 ciclos más tarde

Cuando la instr consumidora llega a la etapa de ejecución puede recibir el operando mediante by-pass desde la salida de la FU productora

  - Cero ciclos de penalización (ejecución "back-to-back") para instr que se ejecutan en un ciclo de reloj.
  - Debe garantizar el timing previsto (disponibilidad de FU, conflictos de CDB...)
  - Problemas cuando el productor es un Load: latencia variable (hit / miss)
    - Necesidad de mecanismo de recuperación

Superescalar con planificación dinámica: Issue (4)

- Mecanismo de selección con Wake-up anticipado
  - Ejemplo: selección de 1 instr en cada ciclo. Se muestran 2 entradas de la Issue Q



de Dubois: Fig. 3.28

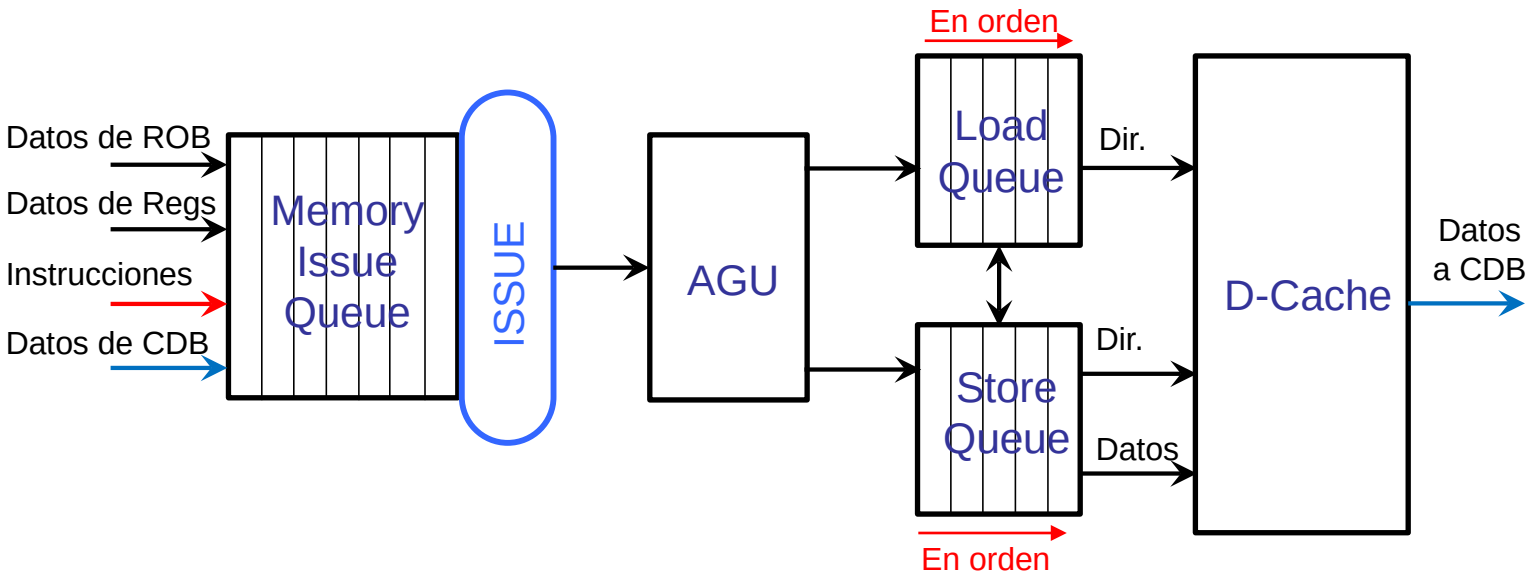
Selecciona una de las instrucciones que han activado la señal "Request", mediante la activación de "Grant".

Tras recibir el Grant, la instr seleccionada genera el Wake-up, con el nº de ciclos de retardo (delay) necesario, teniendo en cuenta la latencia de la instr

Superescalar con planificación dinámica: Issue (5)

Tratamiento de Load y Store

Un posible diseño del bloque genérico LD/ST Q mostrado en la tr. 9



- Memory Issue Q. Cada entrada contiene una instr de LD/ST.
  - La instr permanece en la cola hasta que captura la información necesaria para calcular la dirección (y para ST tb el dato).
  - Las entradas, como en la Issue Q de las op aritméticas, no tienen por qué estar en orden
- AGU: Address Generation Unit. Calcula las dir y las pasa a la entrada correspondiente de la Load o Store Q
- Store Q. Mantiene los ST en orden. A cada ST se le asigna una entrada en orden en la fase de dispactch (en ese momento es posible que no esté disponible la dir ni el dato).
- Load Q. Mantiene los LD en orden. Cada LD se inserta con un TAG que indica la posición en la Store Q del último ST insertado ➔ Permite conocer cuáles son los ST más antiguos que el LD

Superescalar con planificación dinámica: Issue (6)

Tratamiento de Load y Store

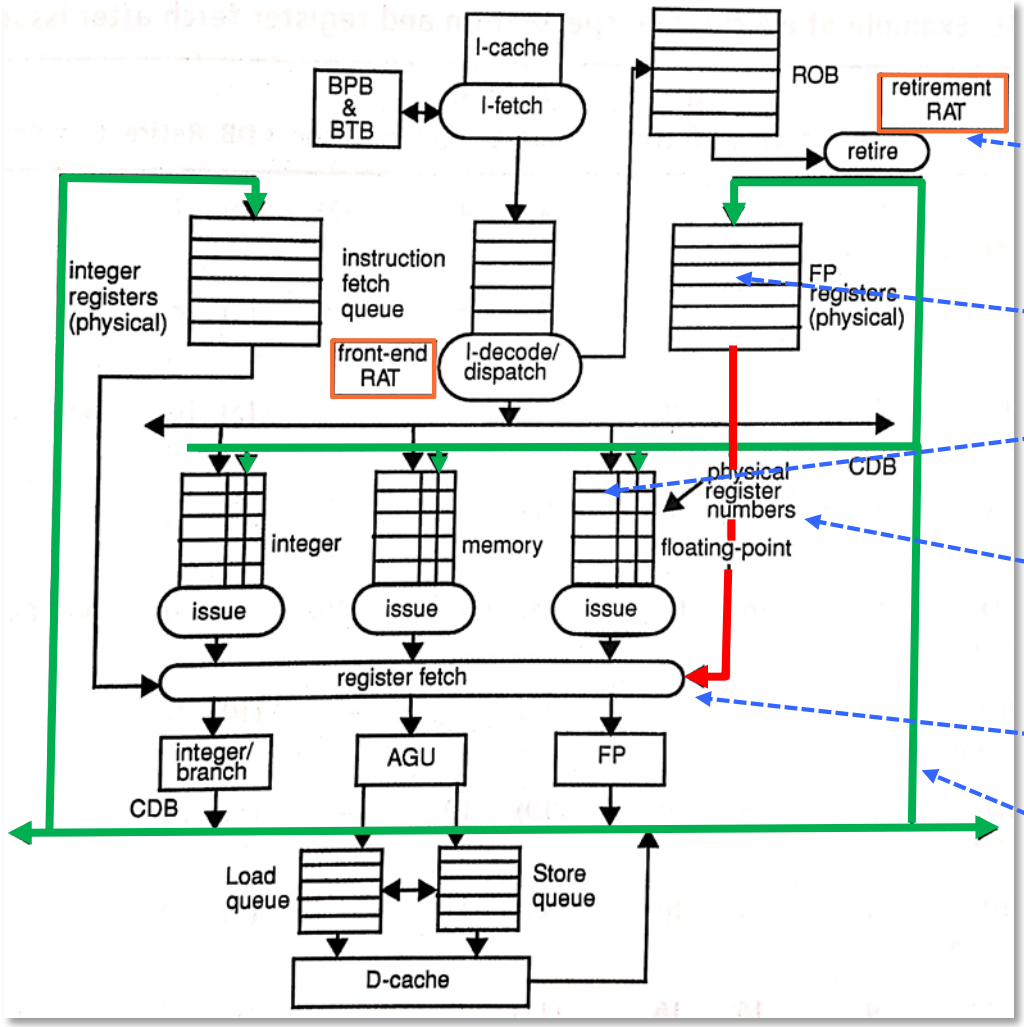
- STORE:
  - Dispatch:
    - Asignar entrada en ROB en orden
    - Buscar entrada libre en Memory Issue Q (almacenar inst + operandos para calcular la dirección + dato; o Tags que permitan obtenerlos más tarde).
    - Asignar entrada en Store Q **en orden**: Contendrá la dir y el dato del ST cuando estén calculados.
  - Issue:
    - Cuando los operandos necesarios están disponibles, se puede mandar la instr a la AGU, y guardar la dir efectiva en la correspondiente entrada de la Store Q.
    - La entrada correspondiente de la Memory Issue Q queda libre.
    - Alternativa: puede ser interesante guardar la dir de un ST en la Store Q, aunque no se disponga todavía del dato. Permite analizar riesgos con los LD más pronto ➔ seguir manteniendo la entrada asignada en la Memory Issue Q.
- LOAD:
  - Dispatch e issue: Tratamiento similar a los ST en cuanto a la dirección

Superescalar con planificación dinámica: Issue (7)

- Desambiguación dinámica de memoria: Comparación de direcciones de LD y ST para evitar riesgos a través de memoria.
  - Un ST en la Store Q solamente puede ejecutar la escritura en la D-Cache si tiene su dato+dirección y está en la cabecera del ROB
    - Todos los LD y ST anteriores han hecho commit ➔ las dependencias WAR y WAW no provocan ningún riesgo.
    - Obviamente, el único ST que puede acceder a la D-cache está tb en la cabecera de la Store Q
  - Un LD puede ejecutar su lectura en la D-Cache si tiene su dirección y está dirección no coincide con la de ningún ST más viejo en la Store Q (búsqueda asociativa)
    - Se garantiza la ausencia de riesgos LDE a través de memoria
- Qué ocurre con un LD cuando existen ST más viejos cuya dirección aún no se conoce?
  - Enfoque conservador: Esperar a conocer esas direcciones
  - Enfoque optimista: **Load especulativo**
    - Fundamento: es raro que un LD dependa de un ST próximo ➔ Apostar a que no va haber coincidencia de direcciones (no riesgo LDE) (Guardar dirección del LD)
    - Mecanismo de recuperación: Si cuando se conoce la dirección de un ST hay coincidencia con un LD ya ejecutado, repetir el LD y las instrucciones siguientes (similar a un salto mal predicho).
    - Mecanismo de prevención: Mantener en Fetch una tabla de predicción de Load dependientes, indexada por el PC, con 1 bit por fila (similar a un predictor de saltos)
      - Load mal especulado ➔ poner a 1 el bit correspondiente en la tabla
      - Sucesivas ejecuciones del mismo Load ➔ bit a 1 ➔ Tratamiento conservador
      - Borrar periódicamente el contenido de la tabla

Superescalar con planificación dinámica: Issue (8)

- Lectura de operandos después de Issue (visión general)
  - Implementado en la mayor parte de los procesadores actuales



de Dubois: Fig. 3.26

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).

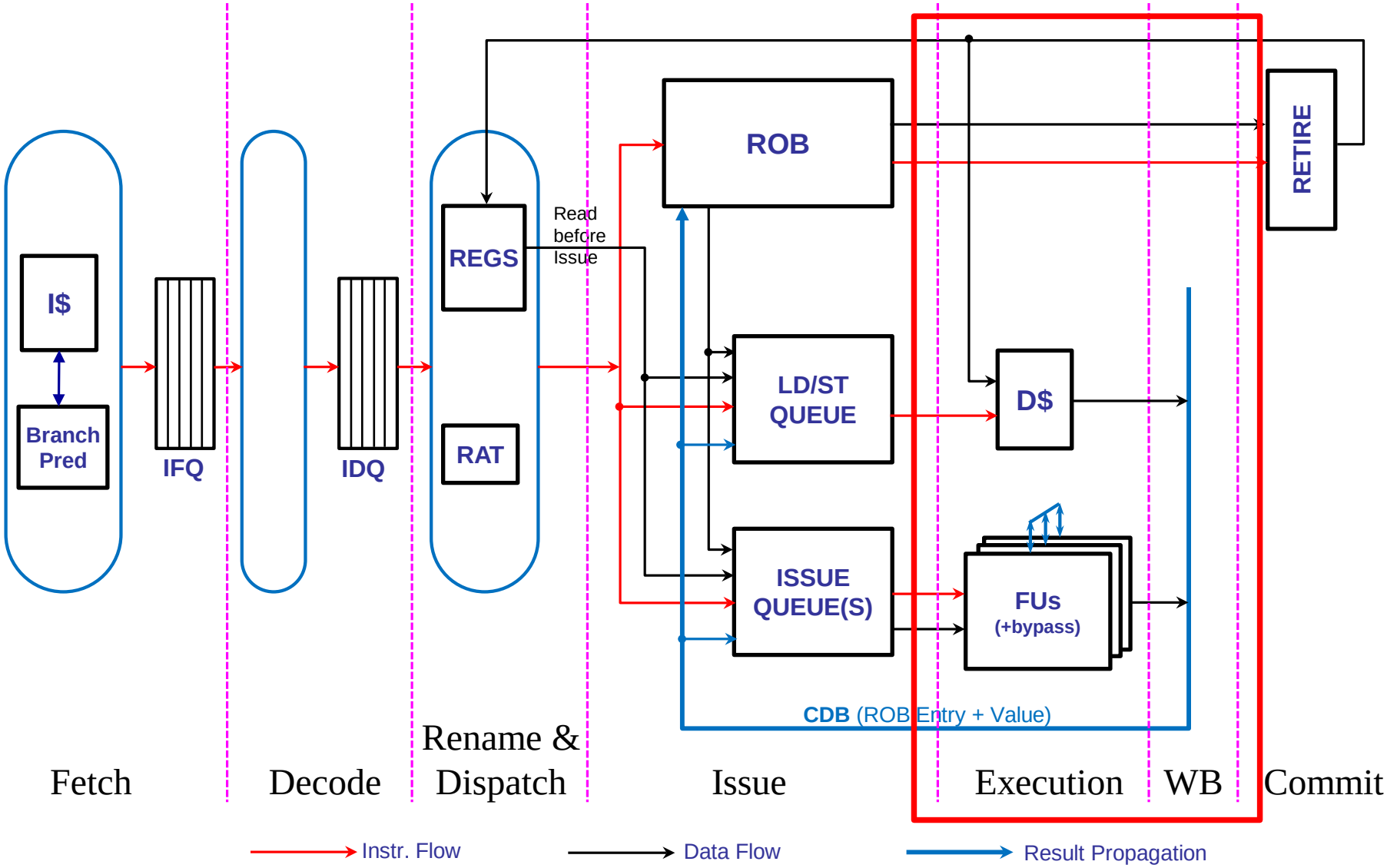
La identificación de operandos fuente (Source Id 1 y 2) no se hace usando Nº de ROB, sino Nº de Reg Fisico.

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**

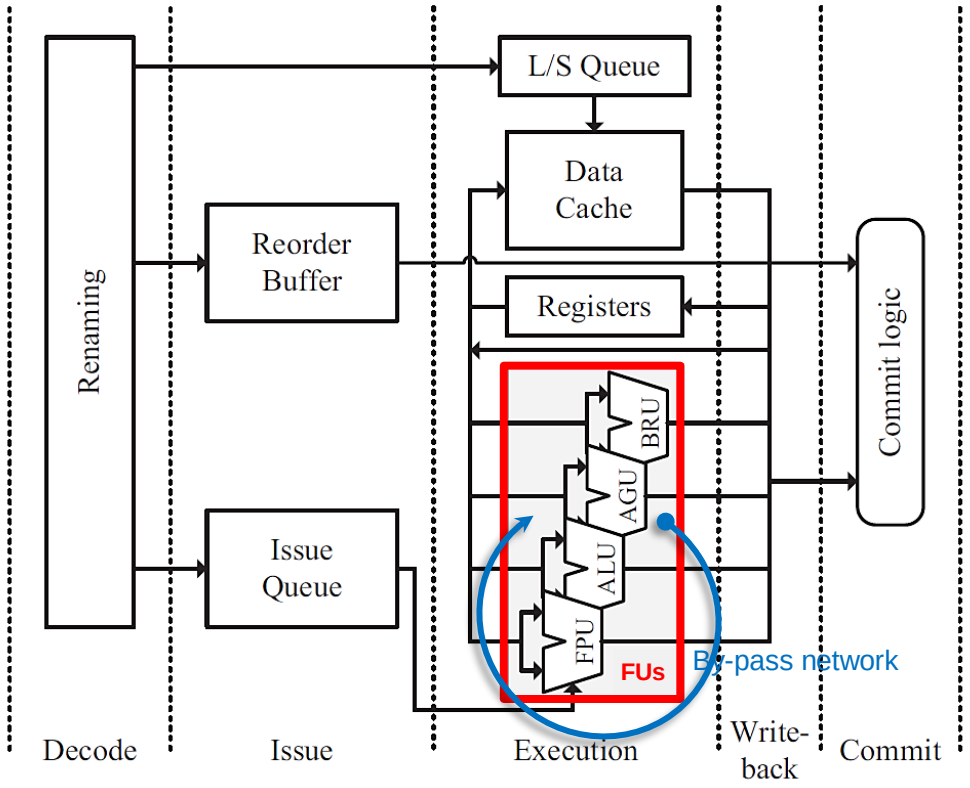
Superescalar con planificación dinámica

- Esquema general de organización en etapas (varias alternativas posibles)



Superescalar con planificación dinámica: Execute&WB (1)

Esquema general de la Unidades Funcionales en su contexto

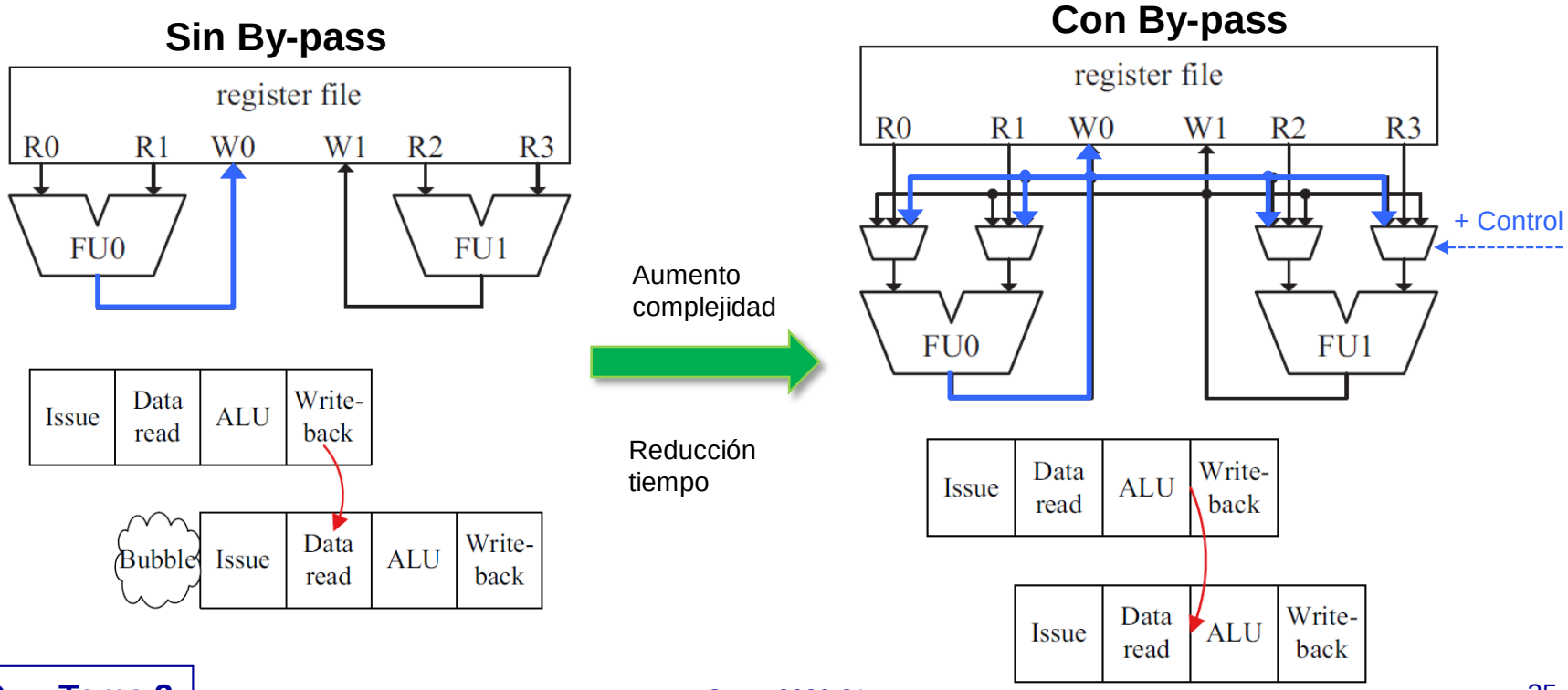


Se asume un fichero de registros unificado

- Las FUs operan sobre los datos que se hacen llegar a sus entradas, junto con la operación a realizar.
- Cuando un resultado está listo (uno o varios ciclos de reloj más tarde) se envía a los elementos de almacenamiento previstos en la instrucción (Write-back) y se realiza el "wake-up" de instrucciones a la espera del resultado.
- El resultado se puede mandar también directamente a las entradas de las FUs que los van a necesitar en el ciclo siguiente, mediante un mecanismo combinacional de anticipación de operandos (by-pass).
  - Concepto similar al estudiado en un procesador segmentado en orden, aunque aumenta la complejidad.

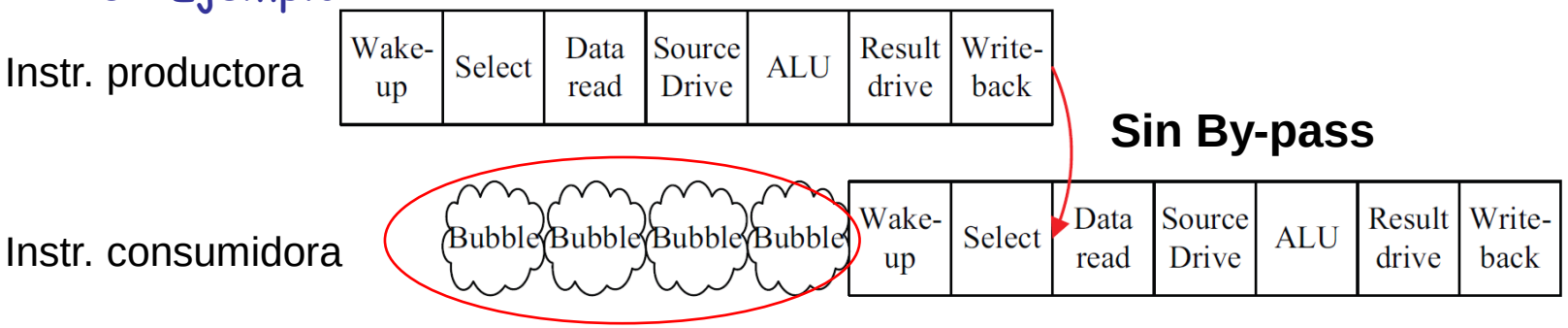
Superescalar con planificación dinámica: Execute&WB (2)

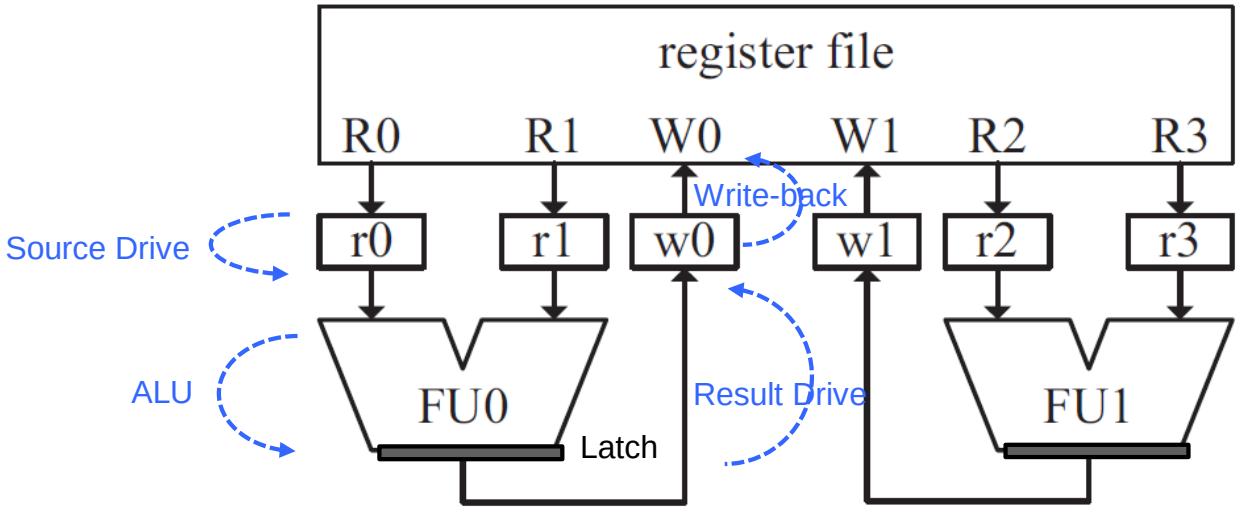
- By-pass: compromiso entre la complejidad combinacional (puede empeorar ciclo de reloj, consumo) y los ciclos de espera entre instrucciones dependientes (mejora IPC).
  - La mayoría de los procesadores actuales usan algún mecanismo de by-pass
  - Hay excepciones importantes (p.ej. IBM POWER5, para disminuir complejidad y aumentar frecuencia de reloj)
- By-pass en un pipeline sencillo.
  - 2 FUs y RF con 4 puertos de lectura / 2 de escritura



Superescalar con planificación dinámica: Execute&WB (3)

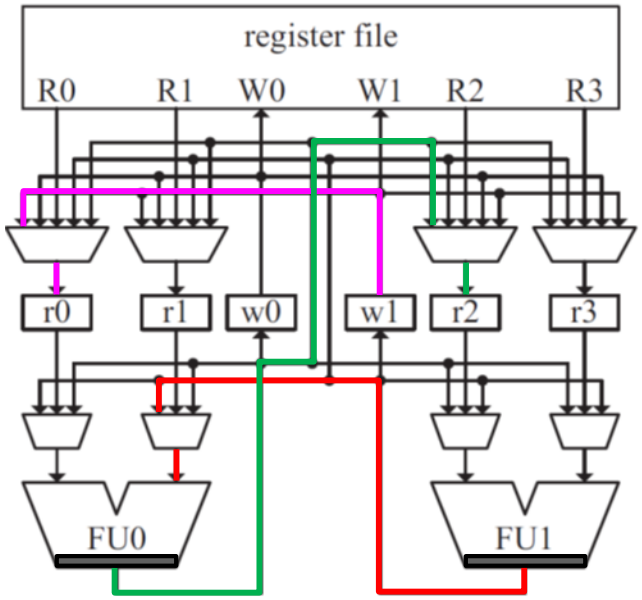
- By-pass en procesadores con alto grado de segmentación. Camino entre RF y FUs segmentado.
  - Más complejidad
  - Sin by-pass: mayor retraso de operaciones dependientes próximas
  - Ejemplo





Superescalar con planificación dinámica: Execute&WB (4)

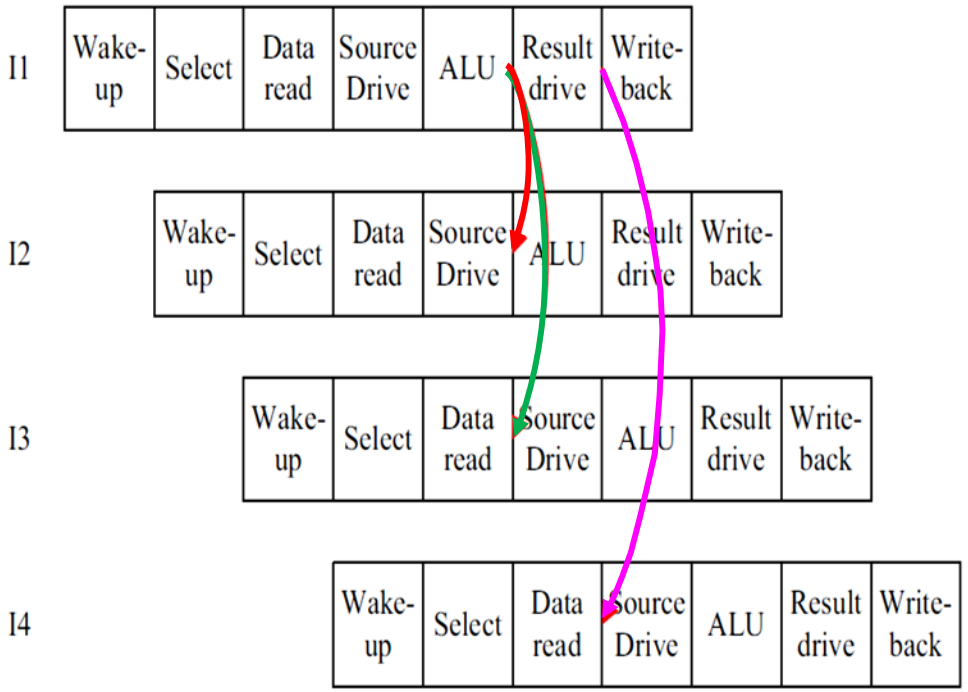
- Ejemplo (cont.): red de by-pass con 2 FUs.
  - Complejidad ↑↑
    - Alternativa. Clustering: By-pass posible solo entre grupos de FUs
  - Paso del resultado de I1 a I2, I3 e I4



Ejemplos de paso de resultados:

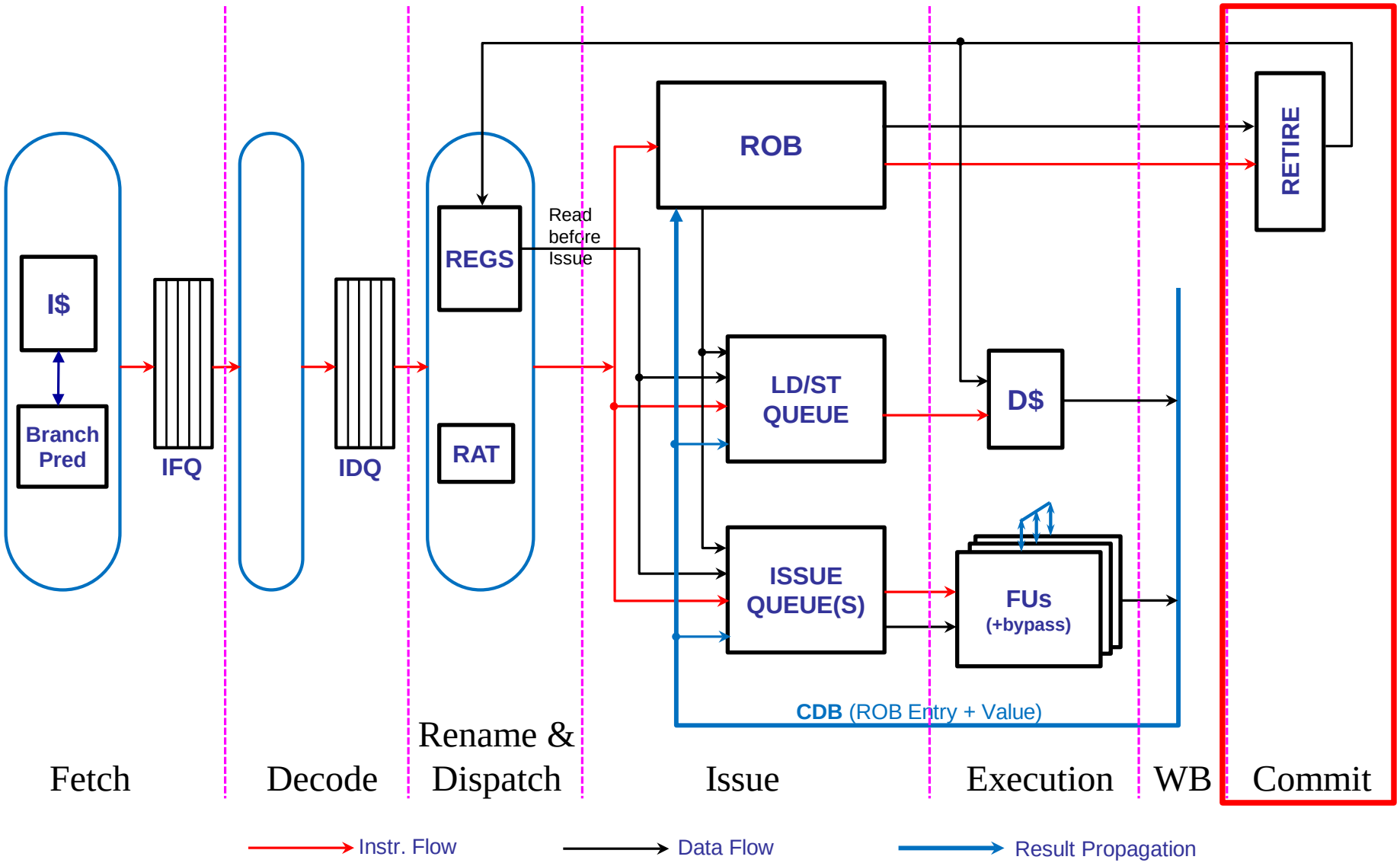
- De I1 a I2 (FU1 → FU0)
- De I1 a I3 (FU0 → r2)
- De I1 a I4 (w1 → r0)

Con By-pass



Superescalar con planificación dinámica

- Esquema general de organización en etapas (varias alternativas posibles)



Superescalar con planificación dinámica: Commit (1)

- Estado arquitectónico: formado por el contenido de la memoria + contenido de los registros utilizables para codificar instrucciones en LM (es decir, registros arquitectónicos o visibles al programador).
- Garantiza que el estado arquitectónico, desde el punto de vista de un programador en LM, se actualiza en el mismo orden que en un procesador totalmente secuencial.
  - Las instrucciones hacen la fase commit (es decir, se retiran) en el mismo orden en que se ejecutan en el flujo dinámico del programa (recordar que en dispatach se insertan en el ROB en dicho orden).
  - Una instrucción I se retira cuando está en la cabecera del ROB: Actualiza el estado arquitectónico del procesador. Ejemplo: instr ADD.D F2,F4,F6
    - Con valores en el ROB: Copiar el valor que hay en cabecera del ROB en F2. El resultado del ADD.D se escribe 2 veces: Una en ROB (en WB), otra en F2 (en commit).
    - Con Fichero de Registros Unificado: Sup que en dispatch el Reg. Arq. F2 quedó representado por RF40 ➔ El contenido de la fila 2 de la Retirement RAT pasa a ser 40. El resultado del ADD.D sólo se ha escrito una vez: en RF40
    - Las instrucciones más antiguas que I ya se han retirado ➔ no pueden activar excepciones: Se han ejecutado completamente
    - Las instrucciones más modernas que I están produciendo / han producido resultados especulativos ➔ no modifican el estado arquitectónico.
- Tratamiento de saltos y excepciones. Como?
- Procesador superescalar: retirada de varias instrucciones en un ciclo.

Superescalar con planificación dinámica: Commit (2)

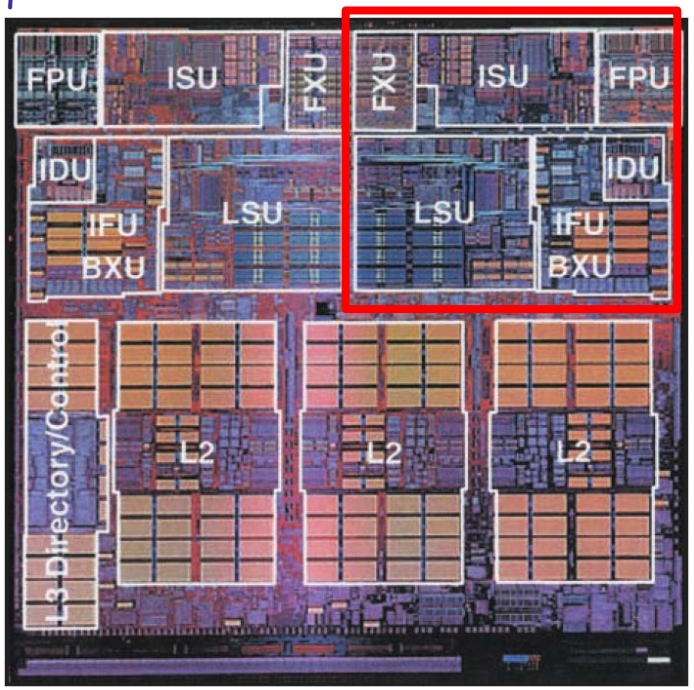
- Si la instrucción I es un salto mal predicho, el contenido del ROB y todas las instrucciones en las colas deben eliminarse
  - EL ROB debe guardar la predicción de los saltos para poder compararlo en commit (o antes) con el comportamiento real

- Suposición: Fichero de Reg Unificado
  - Restaurar la Frontend RAT. Todos los renombramientos hechos corresponden a instr en ROB (a eliminar) o finalizadas (el Reg. Físico asociado al Reg. Arquitectónico está anotado en la Retirement RAT).
  - Solución más simple:
    - La retirement RAT no se ve alterada por el fallo de predicción!! Indica los regs que forman el estado arquitectónico.
    - Los registros indicados en la retirement RAT son lo que deben suministrar los operandos a las nuevas instrucciones que referencien dichos registros ➔ Basta con copiar la retirement RAT sobre la fronted RAT y comenzar a buscar instrucciones por la rama correcta.
    - Problema: Gran número de instrucciones en las colas ➔ muchos ciclos de reloj perdidos, mucha energía gastada en trabajo inútil.
  - Soluciones más ágiles: Anticipar detección fallos mal predichos
    - Ejecutar y finalizar las instr más antiguas, Desechar el salto y todas las instr más jóvenes.
    - Problema: Cómo distinguirlas en las colas de emisión? Como reconstruir la frontend RAT?

Superescalar con planificación dinámica: Commit (3)

- Dispacth y commit de grupos de instrucciones. Caso de estudio: IBM POWER4 (2000)
  - Capacidad para emitir (issue) hasta 8 instr/ciclo (P9 12). 8 UFs en paralelo (P9 16UFs)
  - Tasa de finalización (commit) de 5 instr/ciclo
  - Puede albergar más de 200 instrucciones en proceso de ejecución en un instante dado (P9 256)
  - Varios procesadores posteriores de IBM con arquitecturas similares

Fuente: IBM J. Res. & Dev., January 2002



Micro-fotografía del chip:

- Dos procesadores POWER4 encapsulados en un chip (P9 24 cores)
- Frecuencia: hasta 1.3 GHz (P9 3,5GHz)
- Tecnología: 180 nm (P9 14nm)
- 174 Mtransistores en el chip (P9 8000 Mtran.)
- Interconexiones: 7 capas de metal (cobre) (P9 17capas)
- Cache L2 (~1.5 Mbytes) compartida por los dos procesadores: ocupa más de la mitad del chip (P9 L3 120MB)

Superescalar con planificación dinámica: Commit (4)

POWER4: Diagrama de bloques

Instruction Fetch Address Register

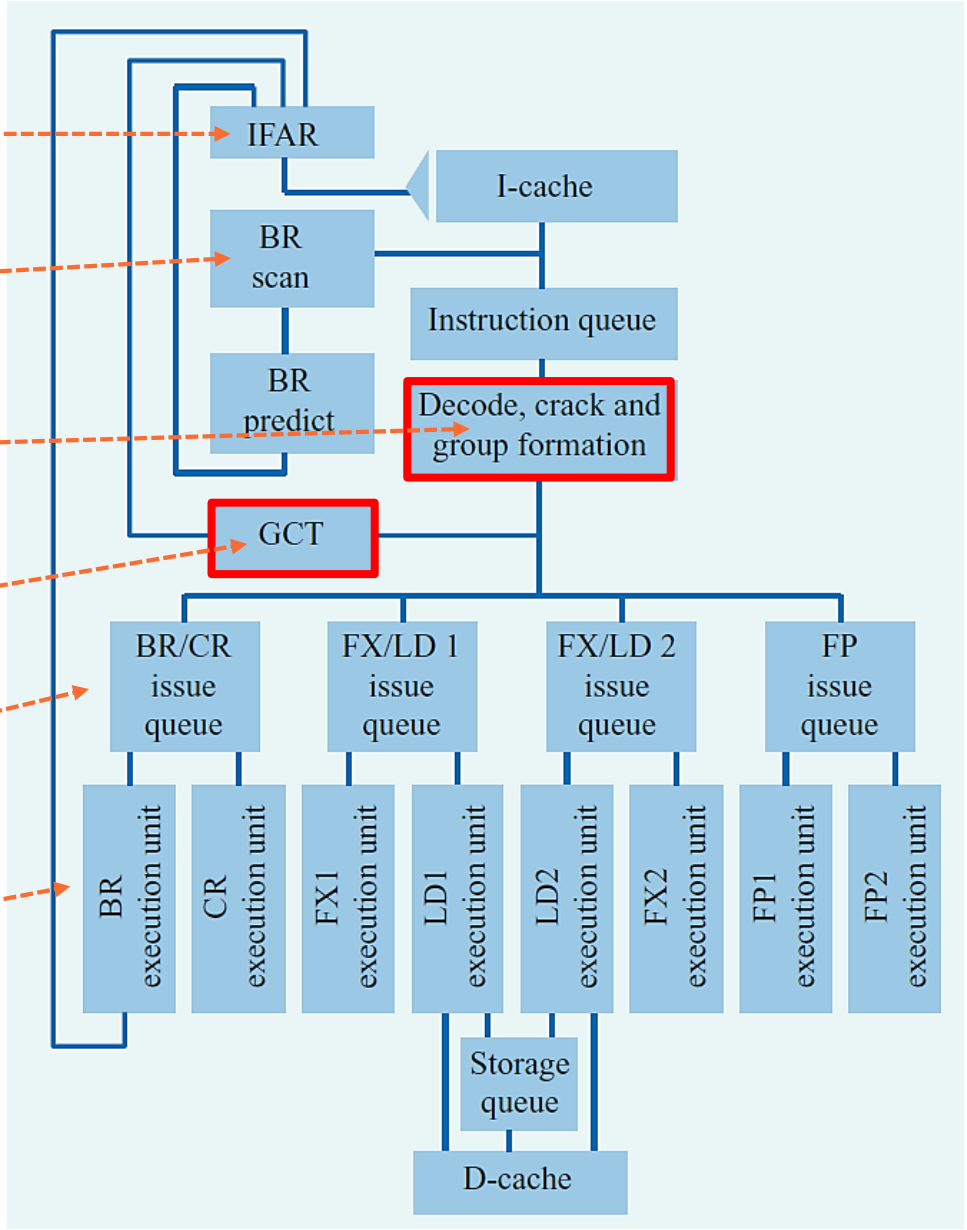
Búsqueda de saltos en el grupo de instr leídas en la I-cache (hasta 2 saltos)

Dispatch de grupos (en orden): Forma grupos de hasta 5 instr, con una de salto como max. Si el salto existe, va en último lugar. Envío a back-end en orden.

Commit de grupos (en orden): **Group Completion Table**. Cada ciclo puede hacer commit de un grupo completo.

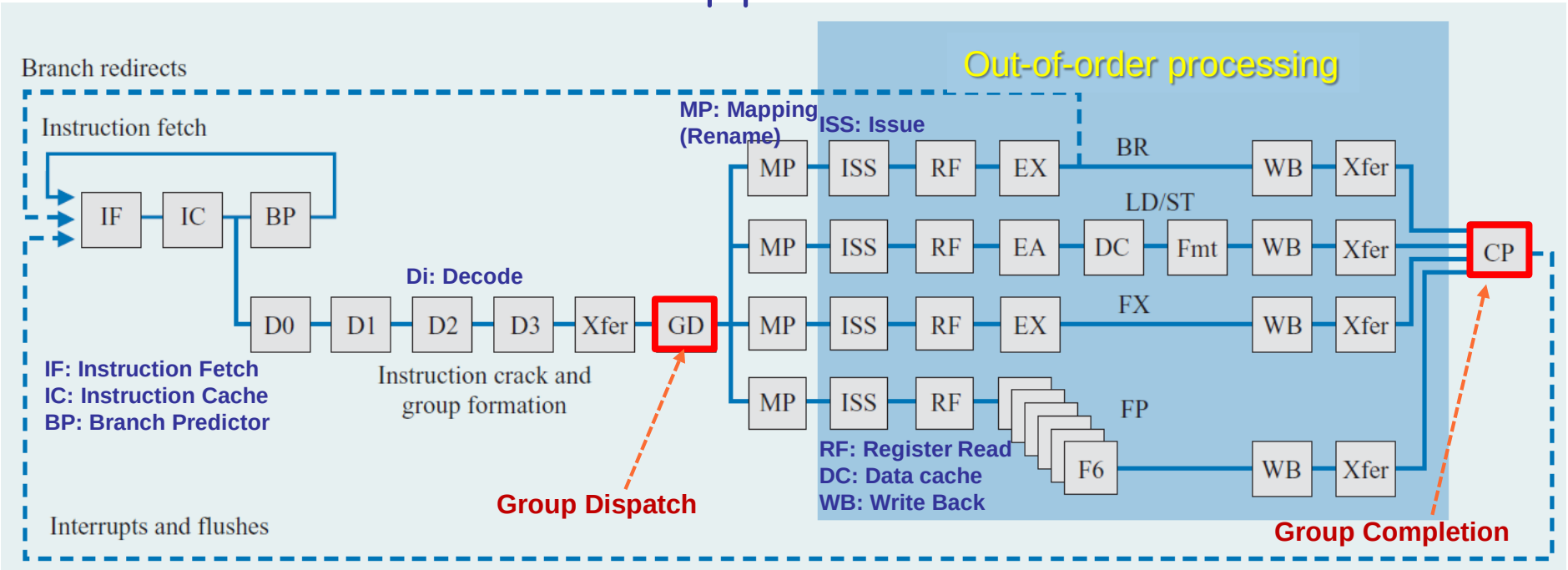
Issue Queues

Unidades Funcionales



Superescalar con planificación dinámica: Commit (5)

❑ POWER4: Estructura del pipeline



Organización de 1 grupo:

IOP1	IOP2	IOP3	IOP4	BR / NOP

- ❑ IOP: operación interna. En general es una instr del LM. Pero algunas instr complejas pueden descomponerse en varias IOPs.
- ❑ Solo hay un salto como máximo y va en la última posición. Si aparece antes, entonces los huecos IOPi se rellenan con NOP y el salto se pone en la última posición.
- ❑ Dispatch de un grupo precisa: 1 entrada libre en GCT, 1 entrada libre para cada instr del grupo en la Issue Q correspondiente, 1 registro libre para renombrar el reg destino de cada instr del grupo, 1 entrada libre en LD o ST Q para cada instr de LD o ST del grupo.
- ❑ La GCT puede alojar hasta 20 grupos. Cuando todas las instr del grupo han finalizado, el grupo entero se retira

Límites del ILP

- El lanzamiento múltiple permite mejorar el rendimiento sin afectar al modelo de programación 
- En los últimos años se ha mantenido el mismo ancho superescalar que tenían los diseños del 1995 
- La diferencia entre rendimiento pico y rendimiento obtenido crece. 

- ¿Cuanto ILP hay en las aplicaciones?

- ¿Necesitamos nuevos mecanismos HW/SW para explotarlo?

  Extensiones multimedia:

  - Intel MMX,SSE,SSE2,SSE3, SSE4
  - Motorola Altivec, Sparc, SGI, HP

Límites del ILP

- **¿Cuanto ILP hay en las aplicaciones?**

Lectura: sección 3.10 de H&P 5th ed.

- **Supongamos un procesador superescalar fuera de orden con especulación y con recursos ilimitados**
  - Infinitos registros para renombrado
  - Predicción perfecta de saltos
  - Caches perfectas
  - Lanzamiento no limitado
  - Desambiguación de memoria perfecta
- **Entonces…**
  - En tal sistema (teórico) la ejecución de instrucciones está solo condicionada por las dependencias LDE.
  - Además, si suponemos que la latencia de las UFs es 1 ¿cómo determinar el IPC?

Límites del ILP

- Comparaciones necesarias en lanzamiento
  - Ejemplo: Lanzar 6 instrucciones/ciclo (check LDE)
    - 1-2 1-3 1-4 1-5 1-6 = 5 instr x 2 operandos = 10 comp
    - 2-3 2-4 2-5 2-6 = 4 instr x 2 operandos = 8 comp
    - ....
    - 5-6 = 1 instr x 2 operandos = 2 comp
    - TOTAL = 30 comp
  - En general: lanzar n instrucciones ➔ n^2-n comp

- Comparaciones necesarias para paso de operandos
  - Ejemplo: pueden acabar 6 instr/ciclo, 80 instr en la ventana, 2 operandos/instr.
    - 6 x 80 x 2 = 960 comparaciones/ciclo
  - En general, comparar con los operandos de todas las instrucciones que están esperando a ser ejecutadas.

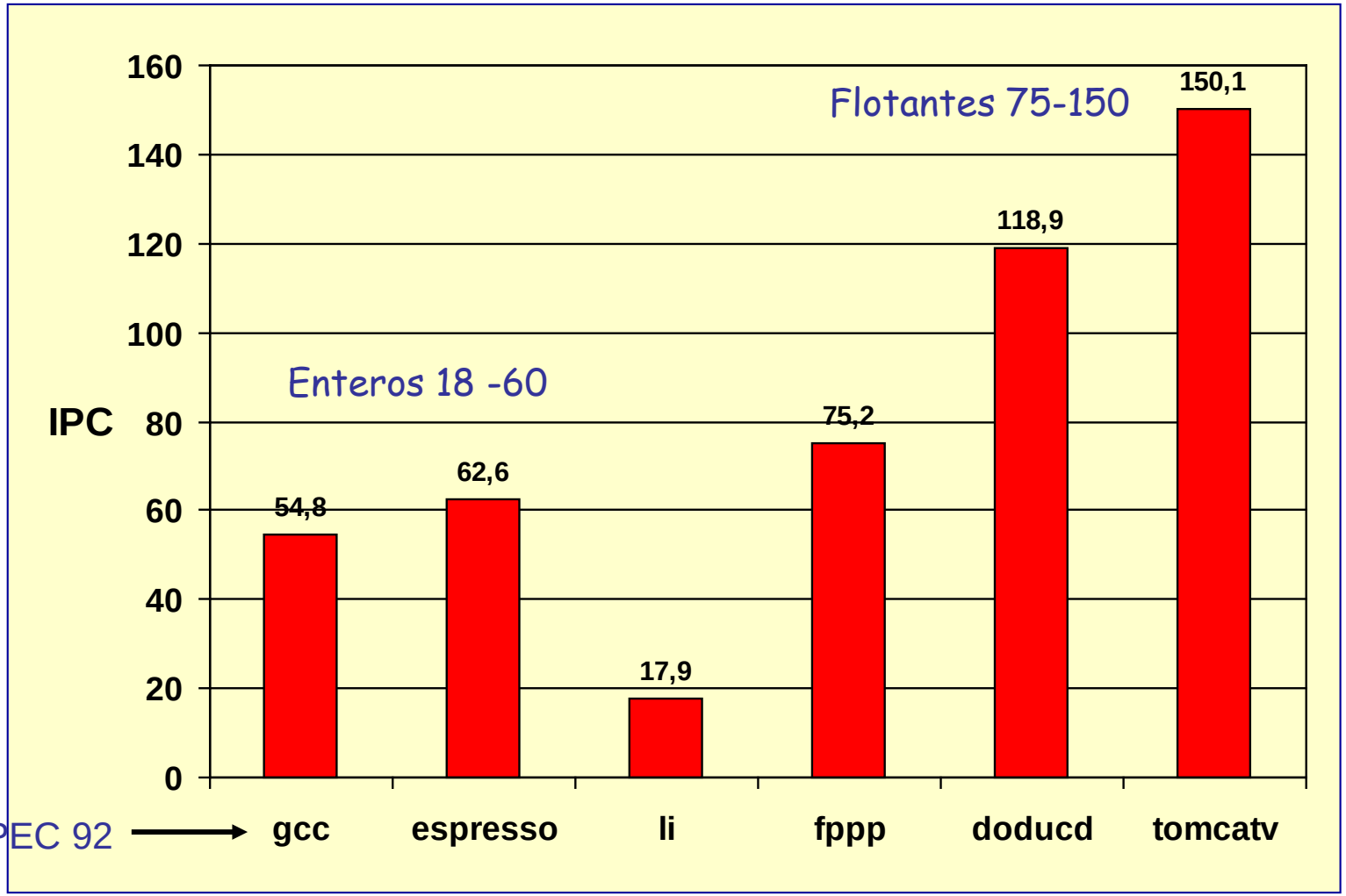
Límites del ILP

- Modelo versus procesador real

	Modelo	Power 5
Instrucciones lanzadas por ciclo	Infinito	4
Ventana de instrucciones	Infinita	200
Registros para renombrado	Infinitos	Extra: 88 integer + 88 Fl. Pt.
Predicción de saltos	Perfecta	2% to 6% de fallos de predicción (Tournament Branch Predictor)
Cache	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3 (off-chip)
Análisis de Memory Alias	Perfecto	

Límites del ILP

❑ Límite superior: Recursos ilimitados



Algunas aplicaciones tienen poco paralelismo (p.ej. Li)

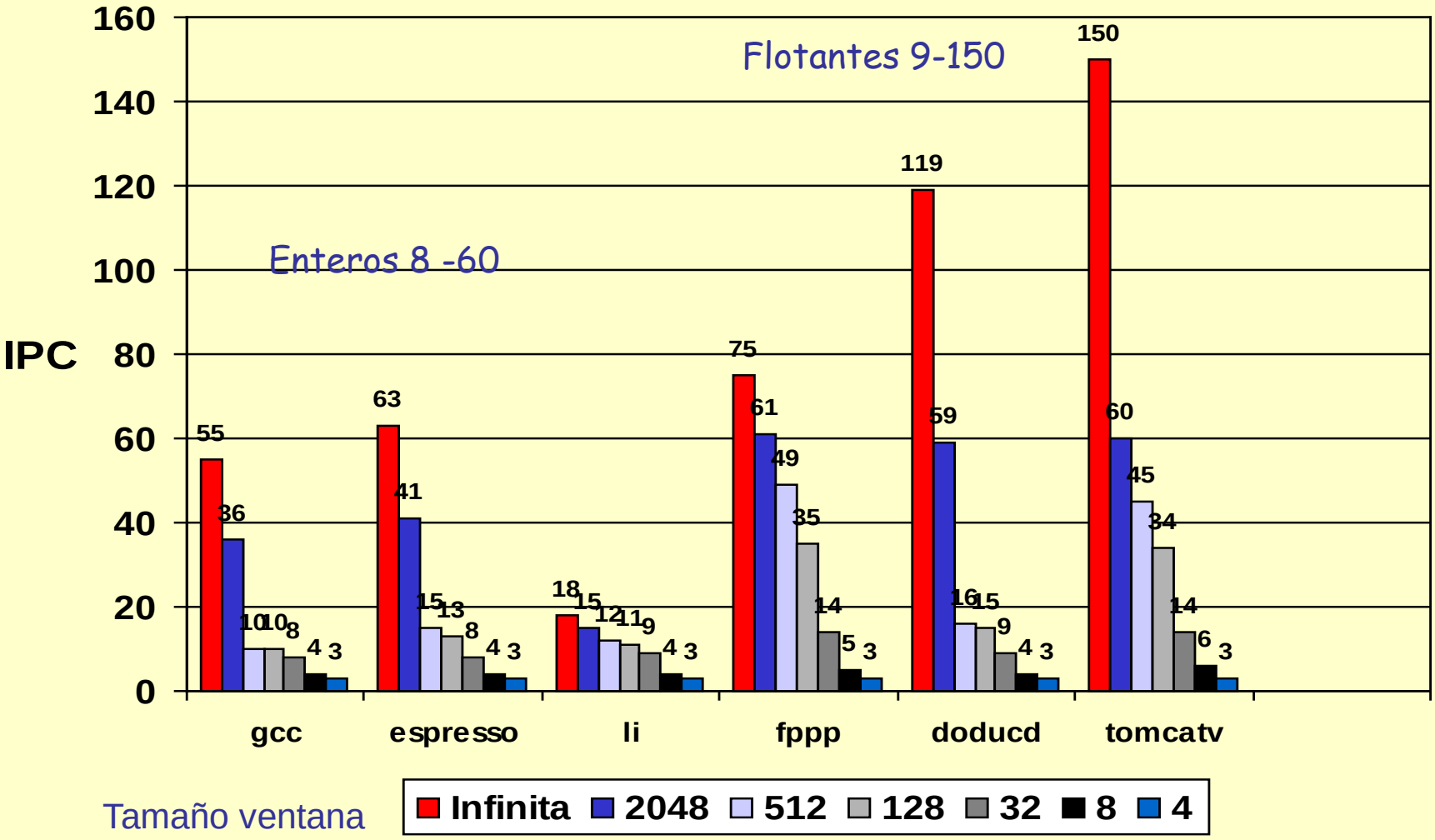
Límites del ILP

- Modelo versus procesador real

	Nuevo Modelo	Modelo	Power 5
Instrucciones lanzadas por ciclo	Infinitas	Infinitas	4
Ventana de instrucciones	Infinito vs. 2K, 512, 128, 32, 8,4	Infinita	200
Registros para renombrado	Infinitos	Infinitos	Extra: 88 integer + 88 Fl. Pt.
Predicción de saltos	Perfecta	Perfecta	Tournament
Cache	Perfecta	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3
Análisis de Memory Alias	Perfecto	Perfecto	

Límites del ILP

- HW más real: Impacto del tamaño de la ventana de instrucciones



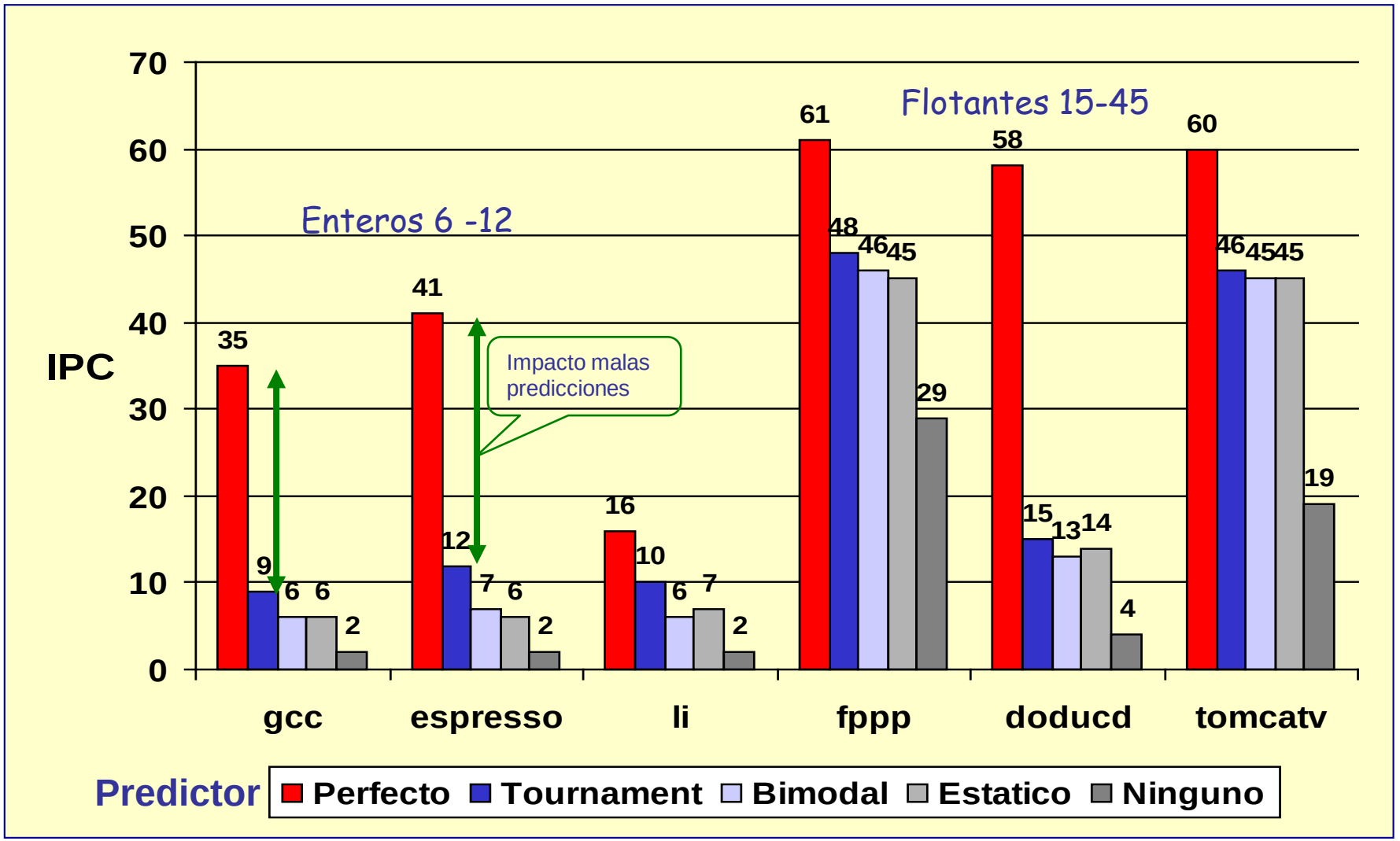
Límites del ILP

- Modelo versus procesador real

	Nuevo Modelo	Modelo	Power 5
Instrucciones lanzadas por ciclo	64 sin restricciones	Infinitas	4
Ventana de instrucciones	2048	Infinita	200
Registros para renombrado	Infinitos	Infinitos	Extra: 88 integer + 88 Fl. Pt.
Predicción de saltos	Perfecto vs. 8K Tournament vs. 512 2-bit vs. profile vs. ninguno	Perfecta	Tournament
Cache	Perfecta	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3
Análisis de Memory Alias	Perfecto	Perfecto	

❑ HW más real: Impacto de los saltos

Ventana de 2048 y lanza 64 por ciclo



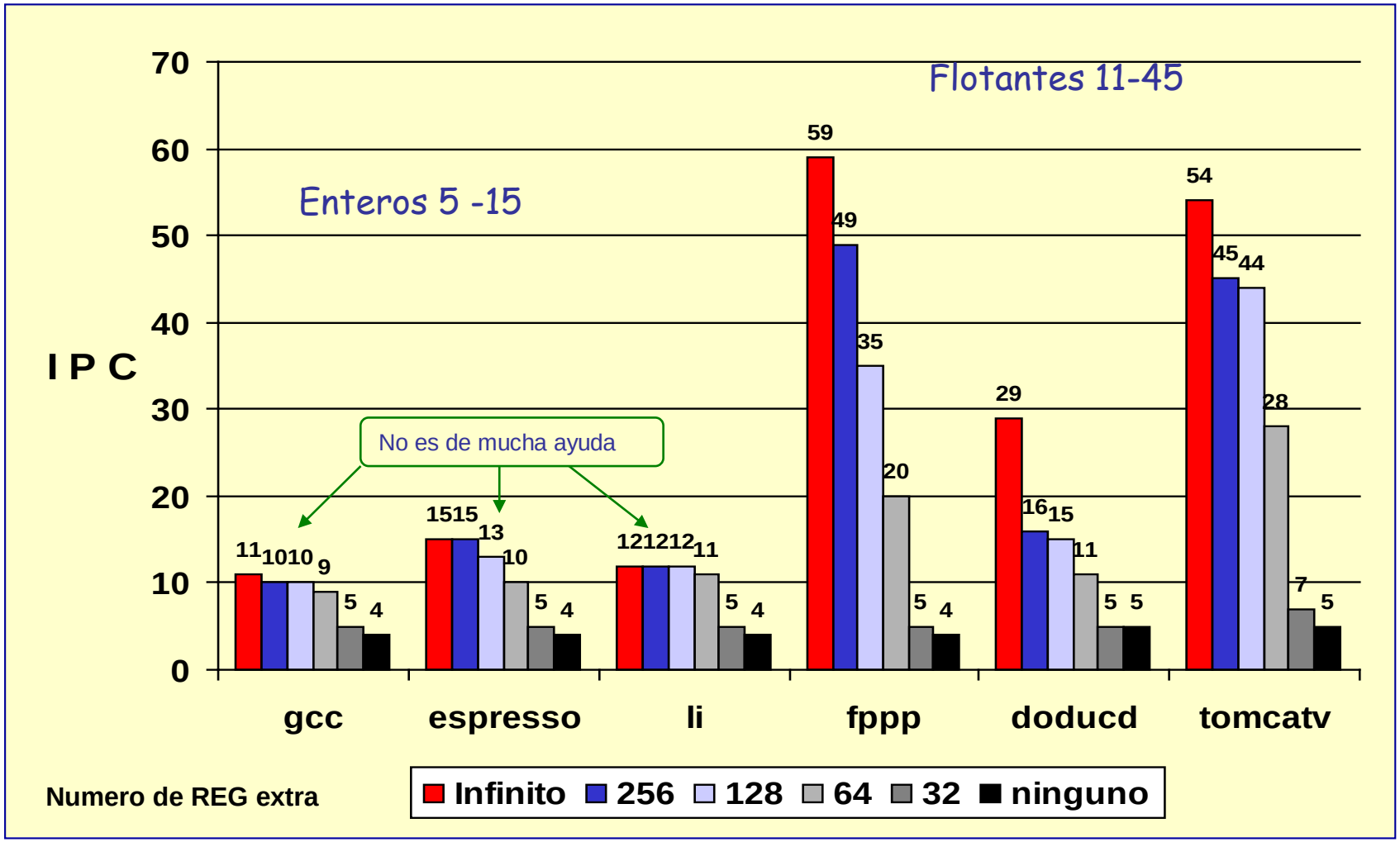
Límites del ILP

- Modelo versus procesador real

	Nuevo Modelo	Modelo	Power 5
Instrucciones lanzadas por ciclo	64 sin restricciones	Infinitas	4
Ventana de instrucciones	2048	Infinita	200
Registros para renombrado	Infinito v. 256, 128, 64, 32, ninguno	Infinitos	Extra: 88 integer + 88 Fl. Pt.
Predicción de saltos	8K Tournament (hibrido)	Perfecta	Tournament
Cache	Perfecta	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3
Análisis de Memory Alias	Perfecto	Perfecto	

❑ HW más real: Impacto del numero de registros

Ventana de 2048 y lanza 64 por ciclo, predictor híbrido 8K entradas



Límites del ILP

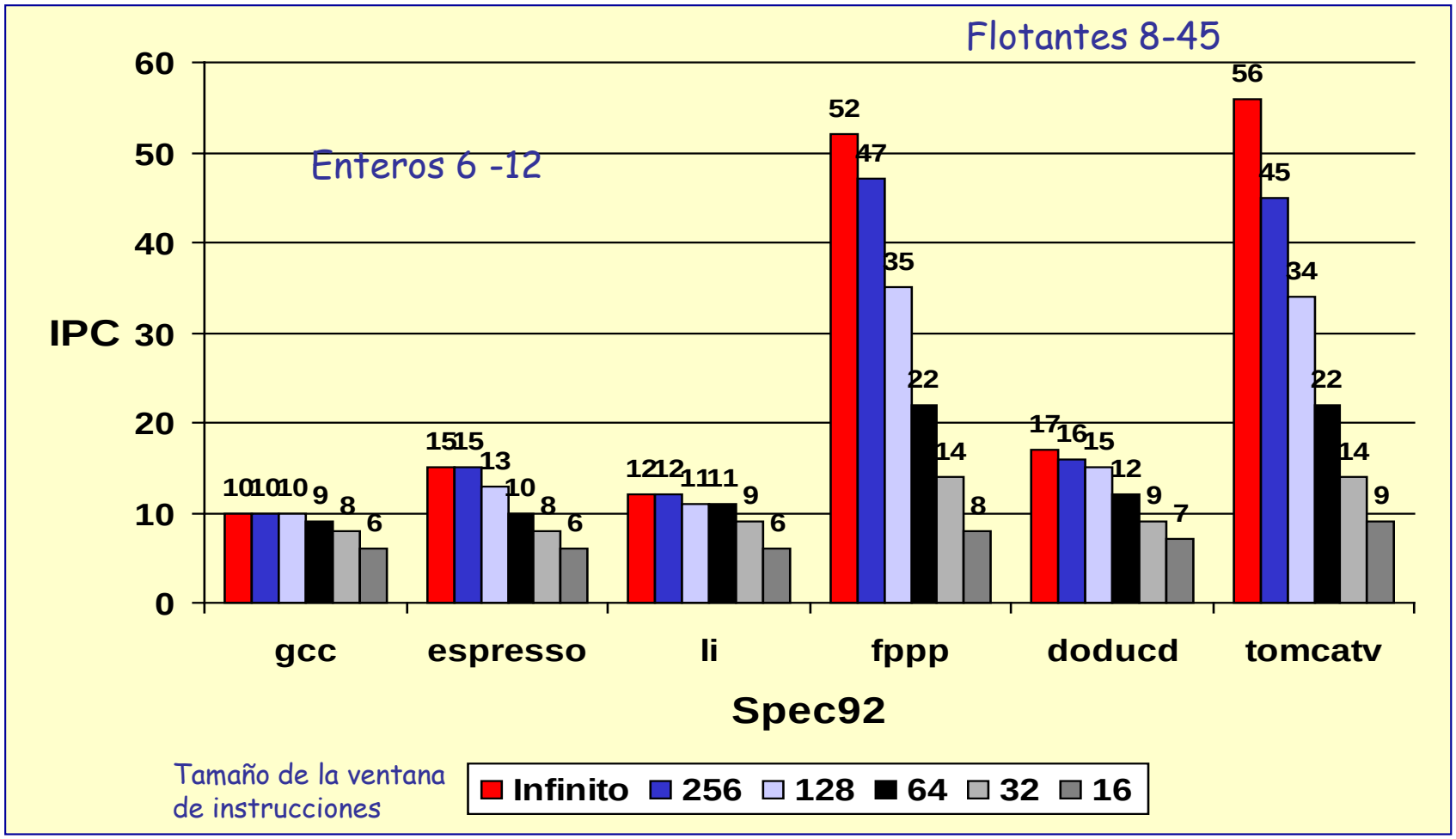
- Modelo versus procesador real

	Nuevo Modelo	Modelo	Power 5
Instrucciones lanzadas por ciclo	64 (sin restricciones)	Infinitas	4
Ventana de instrucciones	Infinito vs. 256, 128, 32, 16	Infinita	200
Registros para renombrado	64 Int + 64 FP	Infinitos	Extra: 88 integer + 88 Fl. Pt.
Predicción de saltos	Tournament 1K entradas	Perfecta	Tournament
Cache	Perfecto	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3
Análisis de Memory Alias	HW disambiguation	Perfecto	

Límites del ILP

HW realizable

64 instrucciones por ciclo sin restricciones, predictor híbrido, predictor de retornos de 16 entradas, Desambiguación perfecta, 64 registros enteros y 64 Fp extra



Ejemplos: evolución procesadores Intel – P6

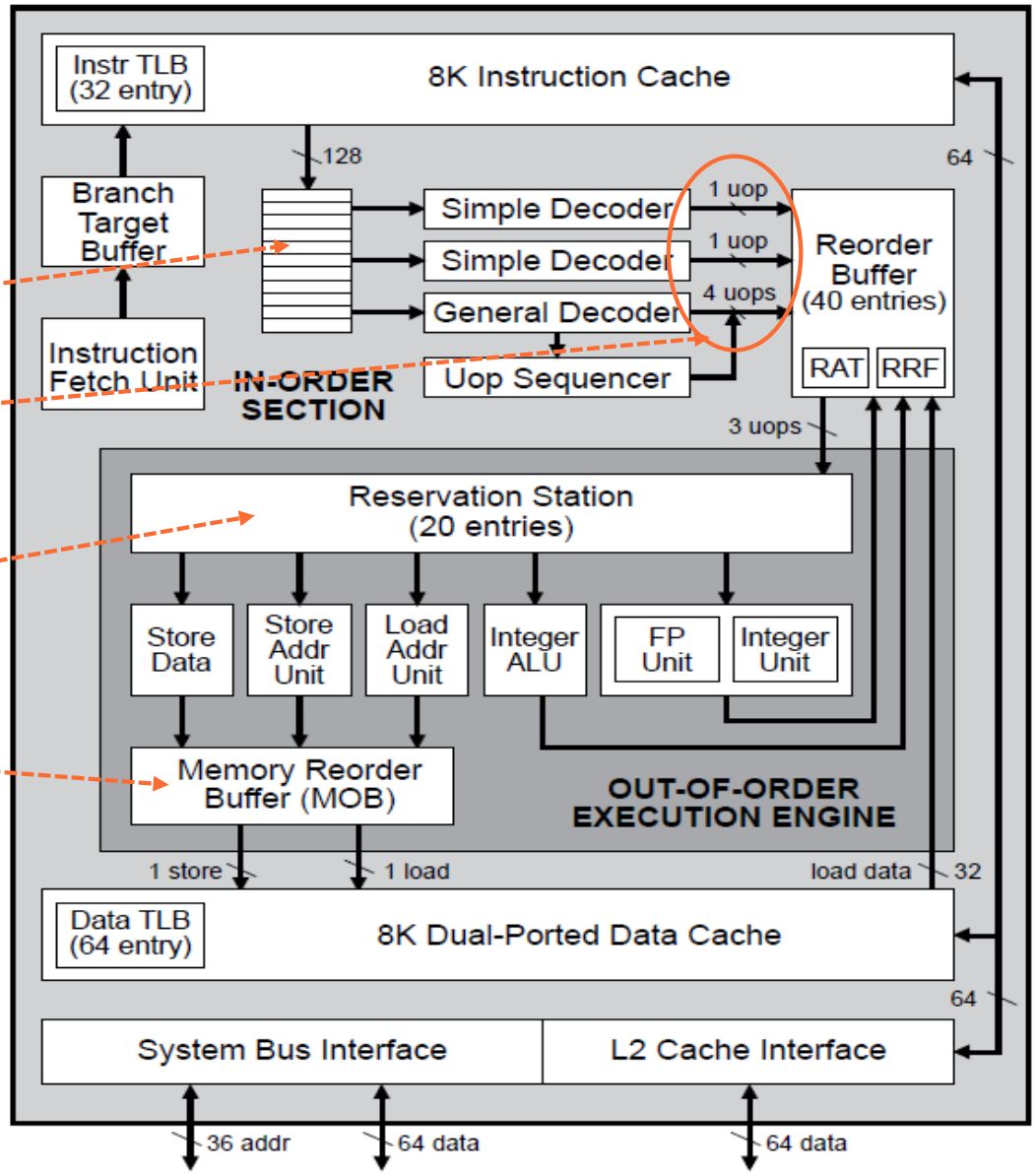
- Micro-Arquitectura usada desde Pentium Pro (1995) hasta Pentium III (1999)

Cola de instrucciones x86

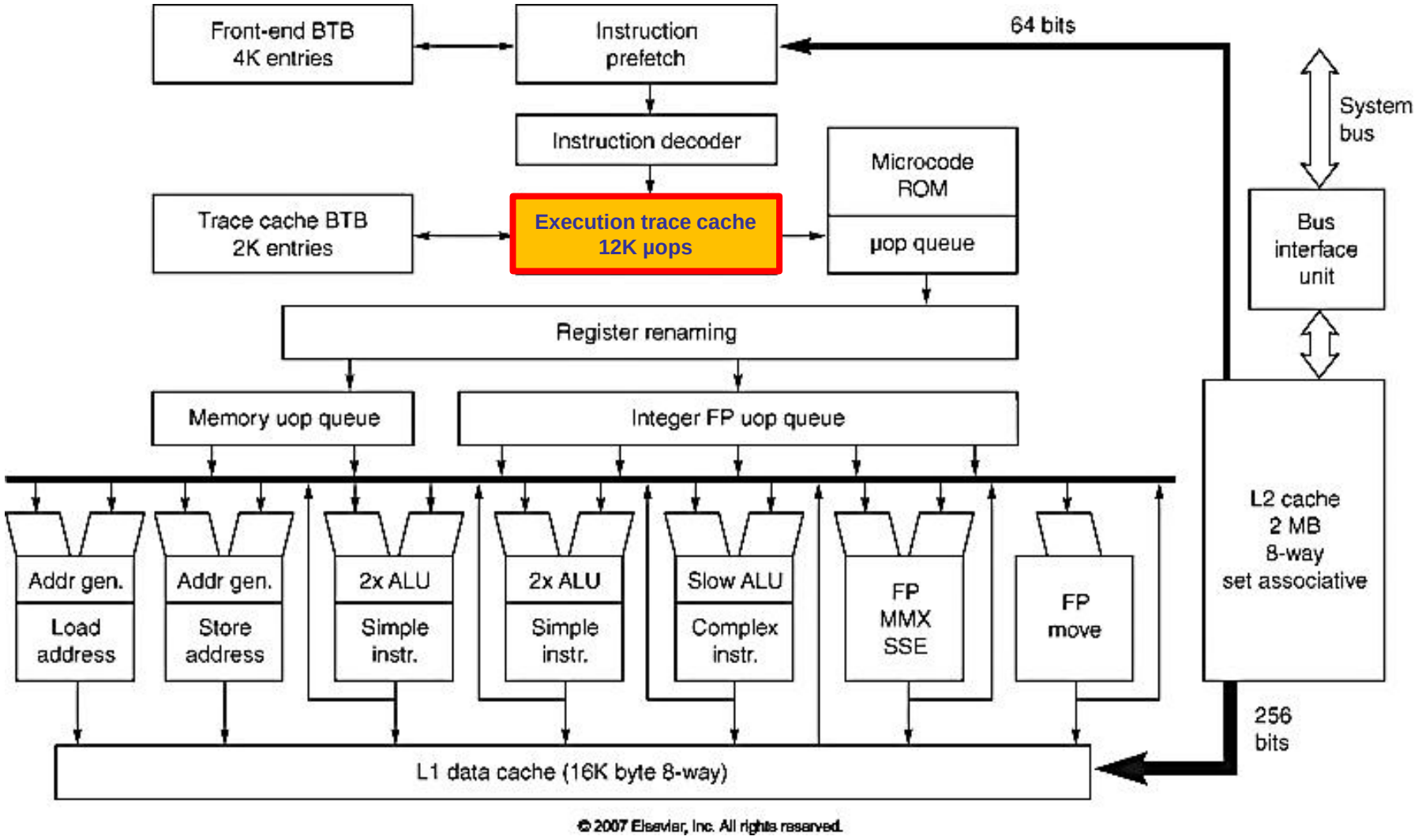
µ-ops descodificadas (2 simple decoder + 1 complex decoder)

Issue Queue unificada

Load Q y Store Q



Ejemplos: evolución procesadores Intel – Pentium 4 (Netburst)



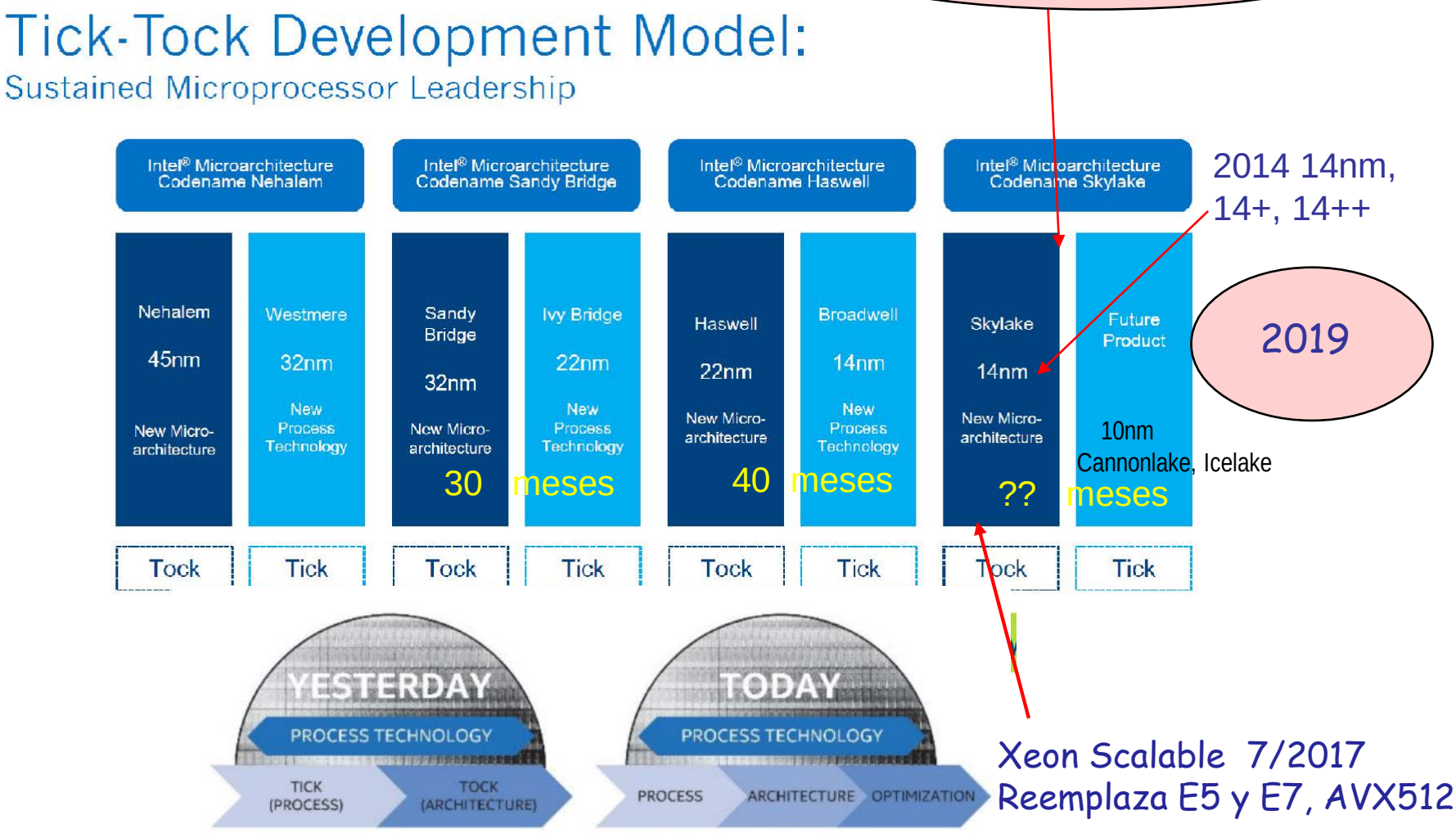
Intel Netburst Microarchitecture

- Traduce instrucciones 80x86 a micro-ops (como P6)
- P4 tiene mejor predicción de saltos (x8) y más FU (7 versus 5)
- La Cache de instrucciones almacena micro-operaciones vs. 80x86 instrucciones. **"trace cache"** (TC), BTB TC 2K entradas
  - En caso de acierto elimina decodificación
- Nuevo bus de memoria: 400(800) MHz vs. 133 MHz (RamBus, DDR, SDRAM) (Bus@1066 Mhz)
- Caches
  - Pentium III: L1I 16KB, L1D 16KB, L2 256 KB
  - Pentium 4: L1I **12K uops**, L1D 16 KB 8-way, L2 2MB 8-way
- Clock :
  - Pentium III 1 GHz v. Pentium 4 1.5 GHz (3.8 Ghz)
  - **14 etapas en pipeline vs. 24 etapas en pipeline (31 etapas)**
- Instrucciones Multimedia: 128 bits vs. 64 bits => 144 instrucciones nuevas.
- Usa RAMBUS DRAM
  - Más AB y misma latencia que SDRAM. Costo 2X-3X vs. SDRAM
- ALUs operan al doble del clock para operaciones simples
- Registros de renombrado: 40 vs. 128; Ventana: 40 v. 126
- BTB: 512 vs. 4096 entradas. Mejora 30% la tasa de malas predicciones

Intel Dual Core, Core2, Quad, Core i…

Tick-Tock a Tick-Tock-Tock-…

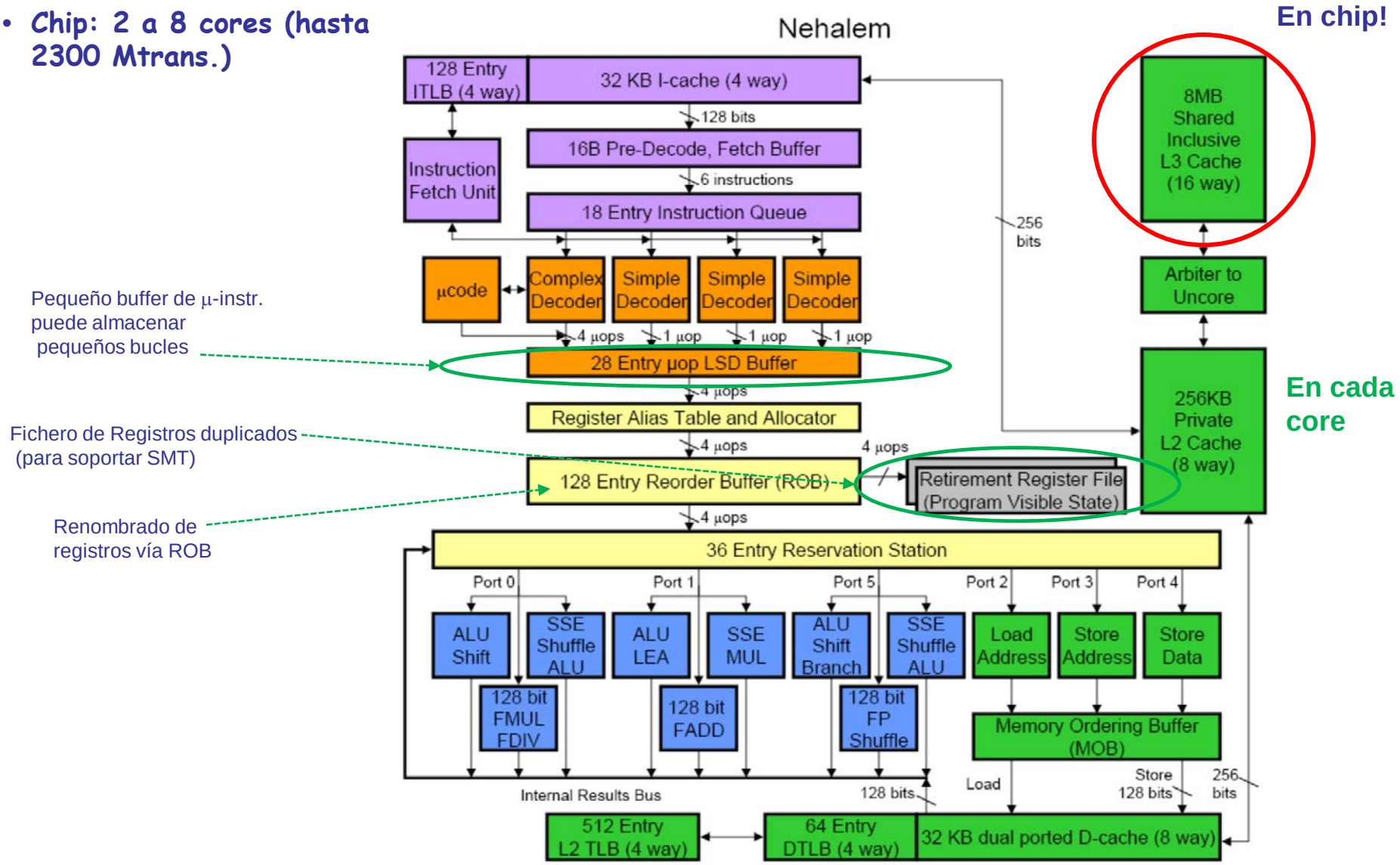
+3 tock Kaby Lake, Coffee Lake
Core i3-5-7 7ªG, 8ªG, 9ºG



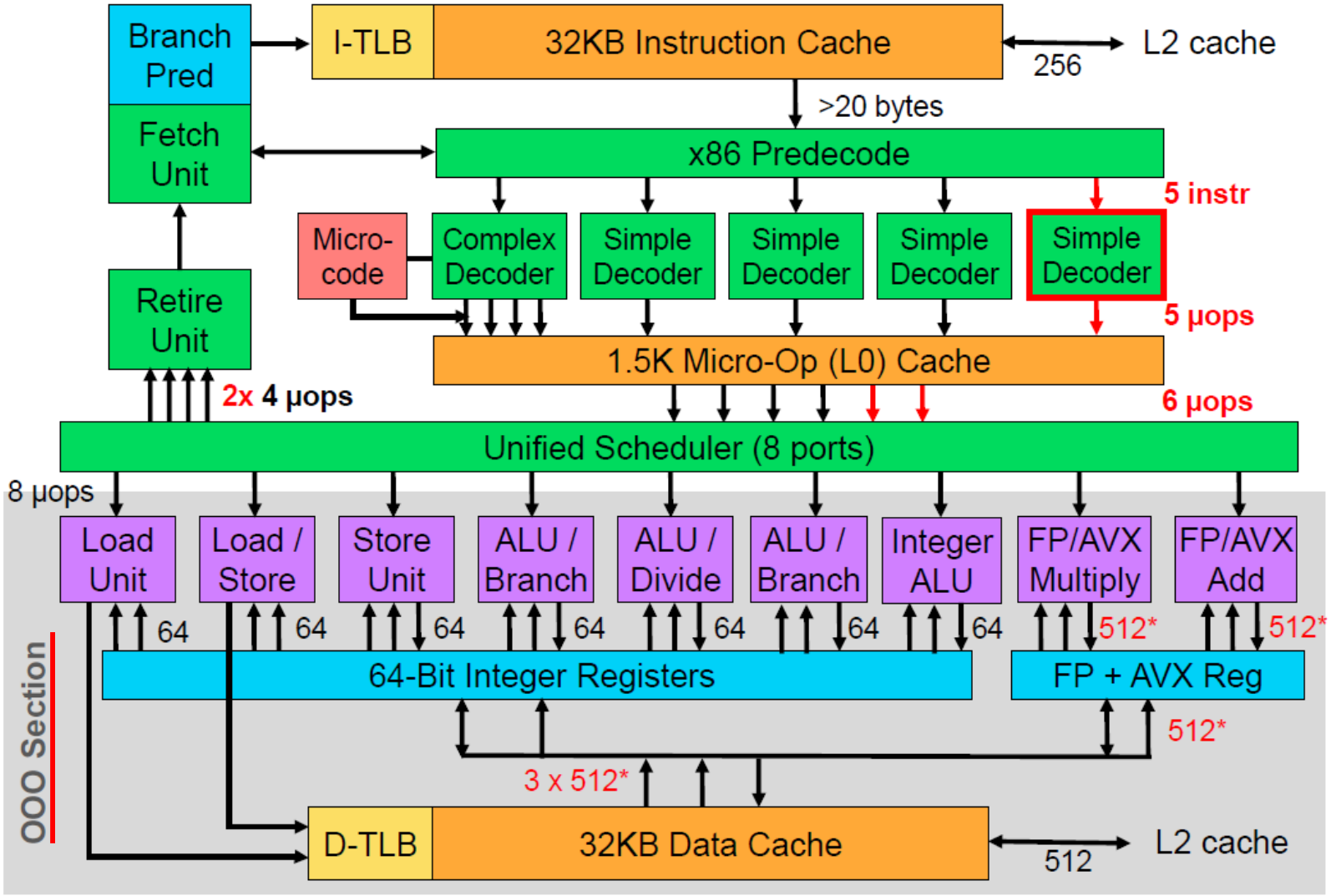
Xeon Scalable 7/2017
Reemplaza E5 y E7, AVX512

Nehalem Microarchitecture

- **Core: 2 threads con SMT**
- **Chip: 2 a 8 cores (hasta 2300 Mtrans.)**



Skylake Microarchitecture



Skylake (2015)/Sunny Cove (2019) Microarchitecture

	Nehalem	Sandy Bridge	Haswell	Skylake
x86 Decoders	4 instr	4 instr	4 instr	5 instr
Max Instr/Cycle	4 ops	6 ops	8 ops	8 ops
Reorder Buffer	128 ops	168 ops	192 ops	224 ops
Load Buffer	48 loads	64 loads	72 loads	72 loads
Store Buffer	32 stores	36 stores	42 stores	56 stores
Scheduler	36 entries	54 entries	60 entries	97 entries
Integer Rename	In ROB	160 regs	168 regs	180 regs
FP Rename	In ROB	144 regs	168 regs	168 regs
Allocation Queue	28/thread	28/thread	56 total	64/thread

Sunny Cove

5 instr

10 ops

352 ops

128 loads

72 stores

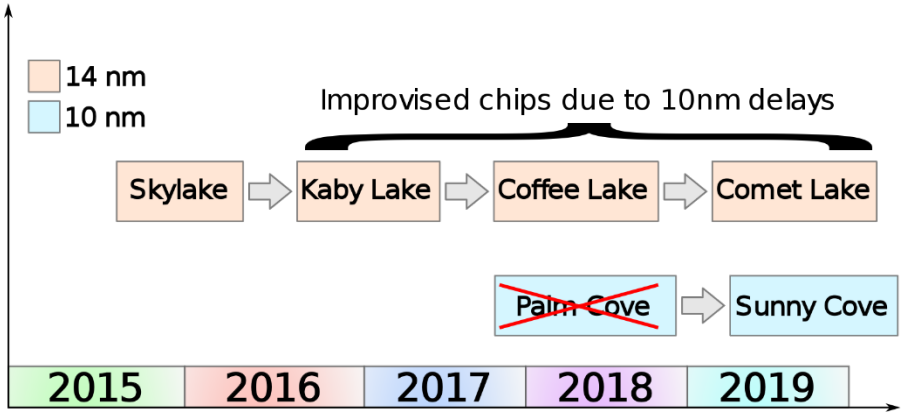
??

Sunny Cove

Large virtual address (57 bits, up from 48 bits)

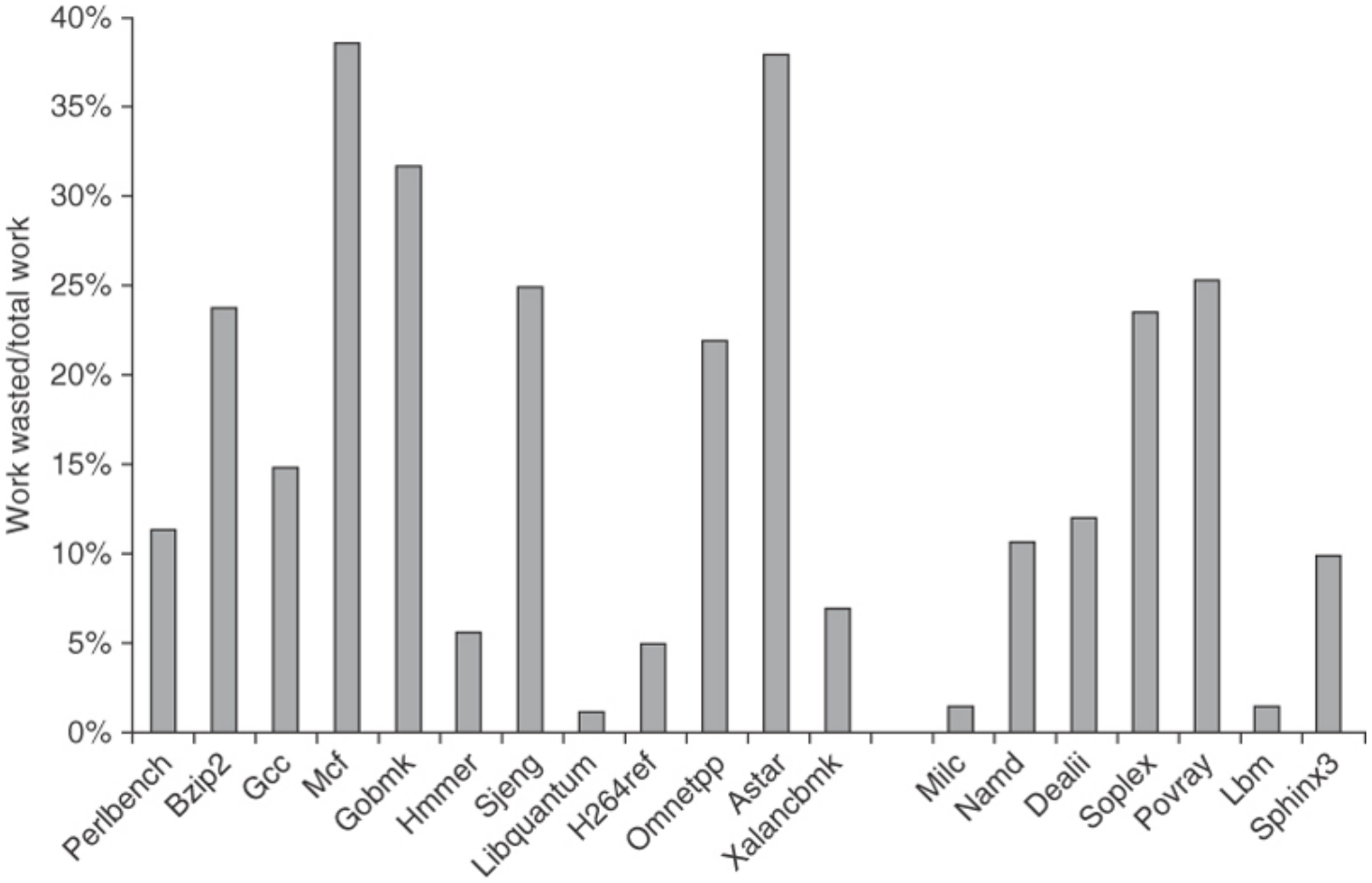
Larger L1 data cache (48 KB, up from 32 KB)

Larger L2 cache (512 KB, up from 256 KB)



Rendimiento I7-920 (Nehalem)

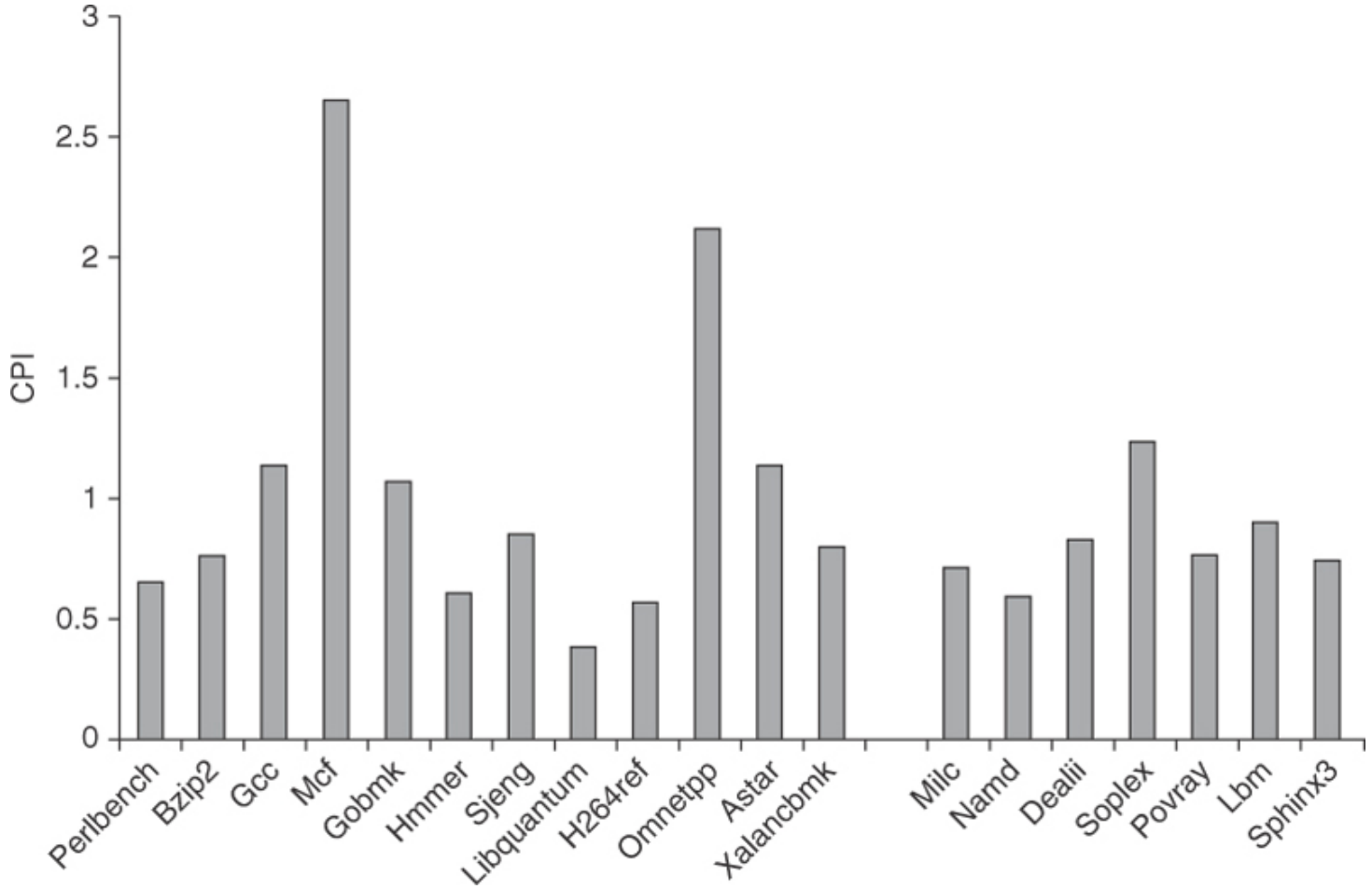
Especulación: porcentaje de μ-operaciones que no finalizan (commit)



Del 1% al 40 % de las instrucciones no finalizan

The data were collected by Professor Lu Peng and Ph.D. student Ying Zhang, both of Louisiana State University.

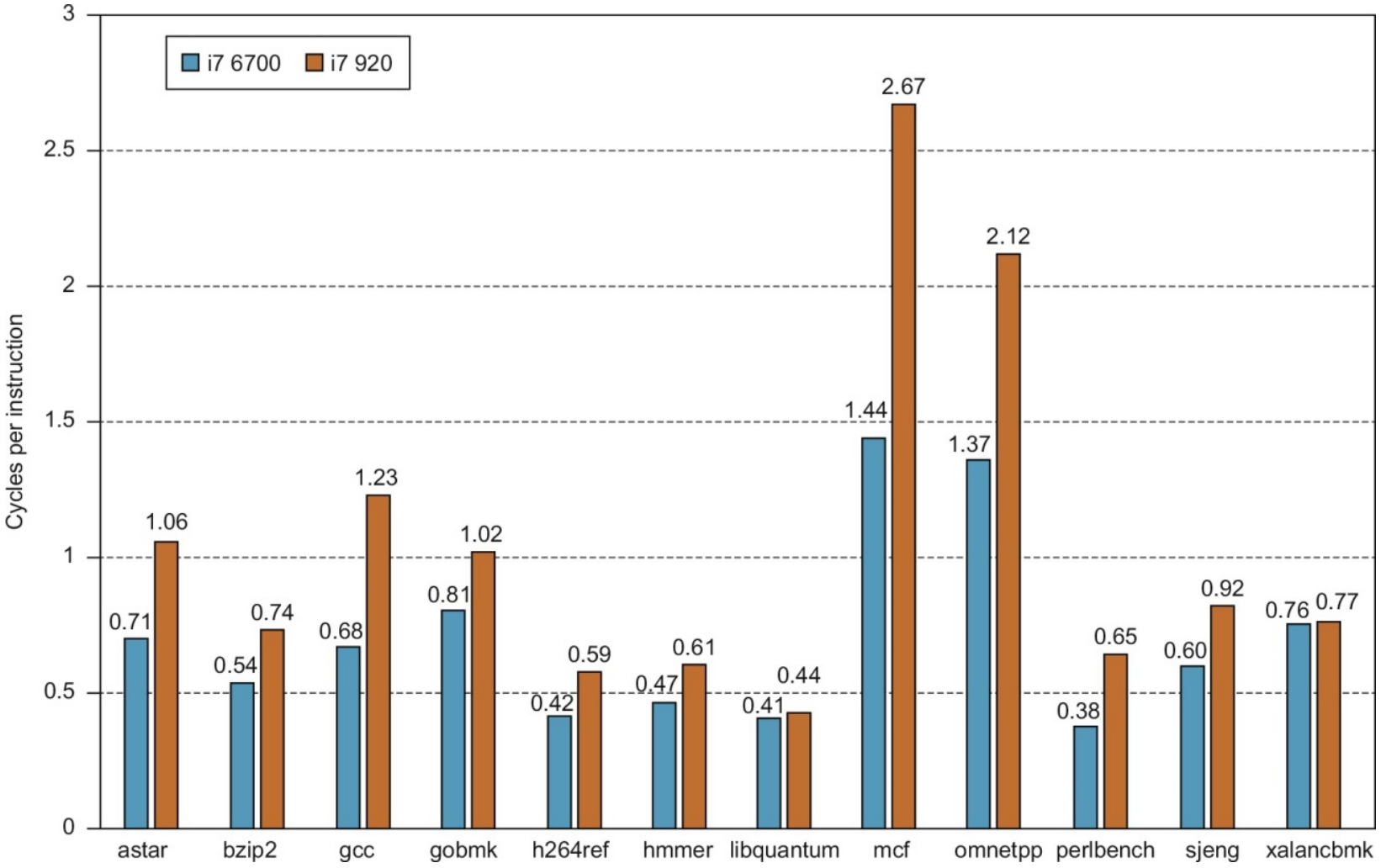
Rendimiento I7-920 (Nehalem)



El CPI para los SPECCPU2006 muestra un CPI promedio de 0,83 (teórico 0,25).
Comportamiento diferente en FP y enteros.
Enteros, CPI entre 0.44 y 2.66. FP entre 0.62 y 1.38.

The data were collected by Professor Lu Peng and Ph.D. student Ying Zhang, both of Louisiana State University.

Rendimiento I7-920 (Nehalem) versus I7-6700 (Skylake)



The data in this section were collected by Professor Lu Peng and PhD student Qun Liu, both of Louisiana State University.

Límites del ILP

- **¿Como superar los limites de este estudio?**
- Mejoras en los compiladores e ISAs
- Eliminar riesgos EDL y EDE en la memoria
- Eliminar riesgos LDE en registros y memoria. Predicción
- Buscar otros paralelismos (thread)

❑ Buscar paralelismo de más de un thread

- Hay mucho paralelismo en algunas aplicaciones (Bases de datos, códigos científicos)
- Thread Level Parallelism
  - Thread: proceso con sus propias instrucciones y datos
    - Cada thread puede ser parte de un programa paralelo de múltiples procesos, o un programa independiente.
    - Cada thread tiene todo el estado (instrucciones, datos, PC, register state, …) necesario para permitir su ejecución
    - Arquitecturas (MultiThreading, multi/many cores, multiprocesadores/computadores paralelos)
- Data Level Parallelism: Operaciones idénticas sobre grandes volúmenes de datos (extensiones multimedia, GPU y arquitecturas vectoriales)

Thread Level Parallelism (TLP) versus ILP

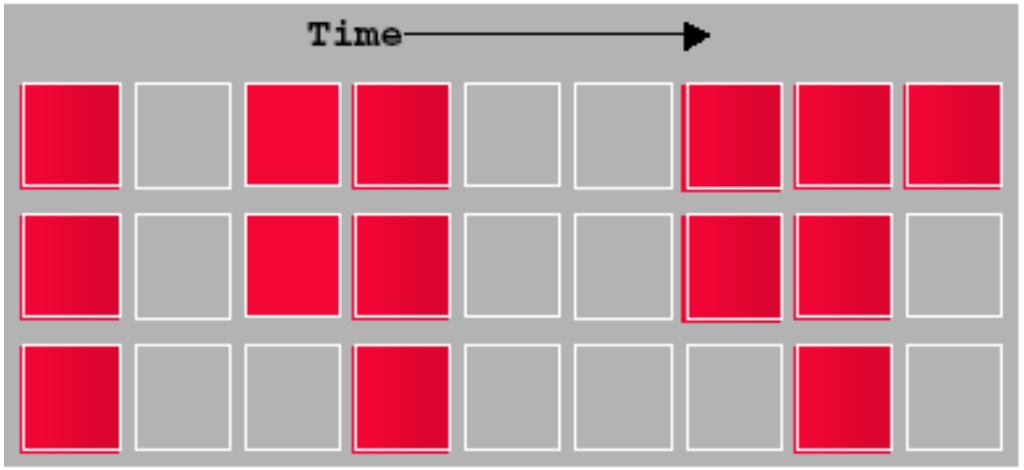
- ILP explota paralelismo implícito dentro de un segmento de código lineal o un bucle
- TLP representa el uso de múltiples thread que son inherentemente paralelos.
- Objetivo: Usar múltiples streams de instrucciones para mejorar:
  - Throughput de computadores que ejecutan muchos programas diferentes.
  - Reducir el tiempo de ejecución de un programa multi-threaded
- TLP puede ser más eficaz en coste que ILP

¿Por qué multithreading?

- Procesador superescalar

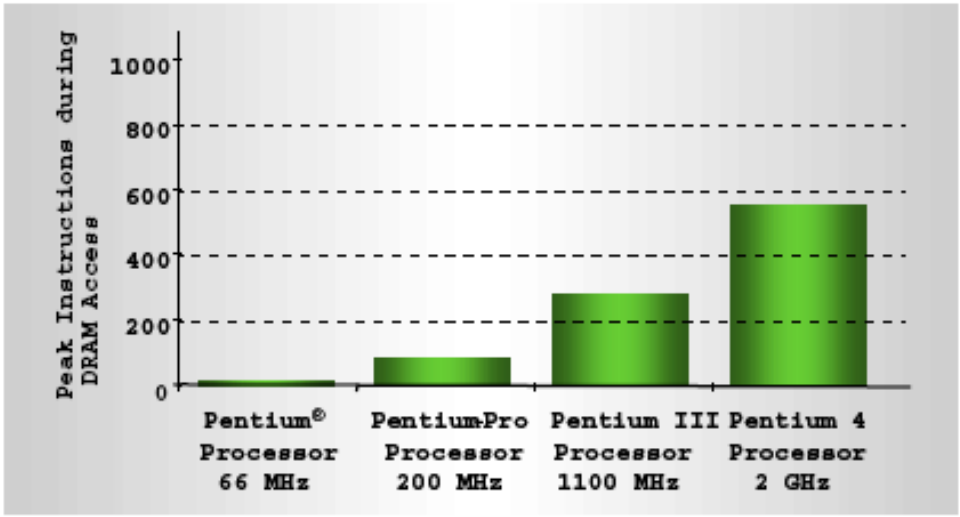
  (P. ej.: Hasta 3 Fus pueden comenzar una op en cada ciclo)

  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



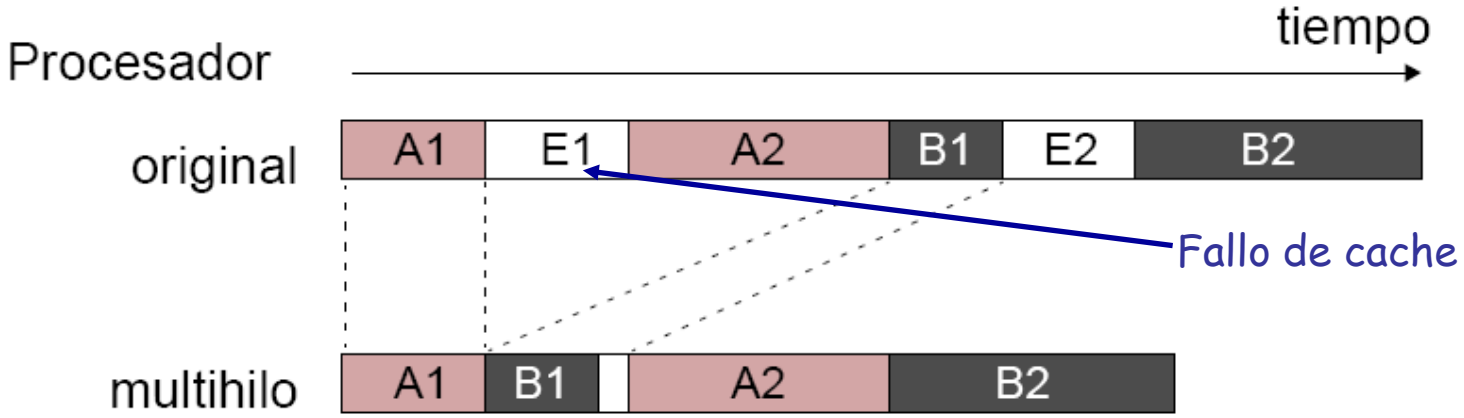
- La latencia de memoria crece.

  ¿Cómo soportarla?



❑ Cómo funciona el Multithreading

❑ Ejemplo fallo de cache



- Incrementar el trabajo procesado por unidad de tiempo
- Si los hilos son del mismo trabajo se reduce el tiempo de ejecución

La técnica multithreading no es ideal y se producen pérdidas de rendimiento.
Por ejemplo, un programa puede ver incrementado su tiempo de ejecución aunque el número de programas ejecutados por unidad de tiempo sea mayor cuando se utiliza multithreading.

❑ Ejecución Multithreaded

- Multithreading: múltiples threads comparten los recursos del procesador
  - El procesador debe mantener el estado de cada thread e.g., una copia de bloque de registros, un PC separado, tablas de páginas separadas.
  - La memoria compartida ya soporta múltiples procesos.
  - HW para conmutación de thread muy rápido. Mucho mas rápido que entre procesos.

Clave

- ¿Cuándo conmutar?
  - Cada ciclo conmutar de thread (grano fino)
  - Cuando un thread debe parar (por ejemplo fallo de cache)

- HEP (1978), Alewife (1990) , M-Machine (1995) , Tera-Computer (1999)

Lectura recomendada

H&P 5th ed., Sección 3.12: págs. 223-226

Multithreading de Grano Fino

- Conmuta entre threads en cada ciclo, entrelazando la ejecución de los diferentes thread.
- Generalmente en modo "round-robin", los threads bloqueados se saltan
- La CPU debe ser capaz de conmutar de thread cada ciclo.
- Ventaja; puede ocultar stalls de alta y baja latencia, cuando un thread esta bloqueado los otros usan los recursos.
- Desventaja; retarda la ejecución de cada thread individual, ya que un thread sin stall es retrasado por reparto de recursos (ciclos) entre threads
- Ejemplo Niagara y Niagara 2 (SUN-Oracle)

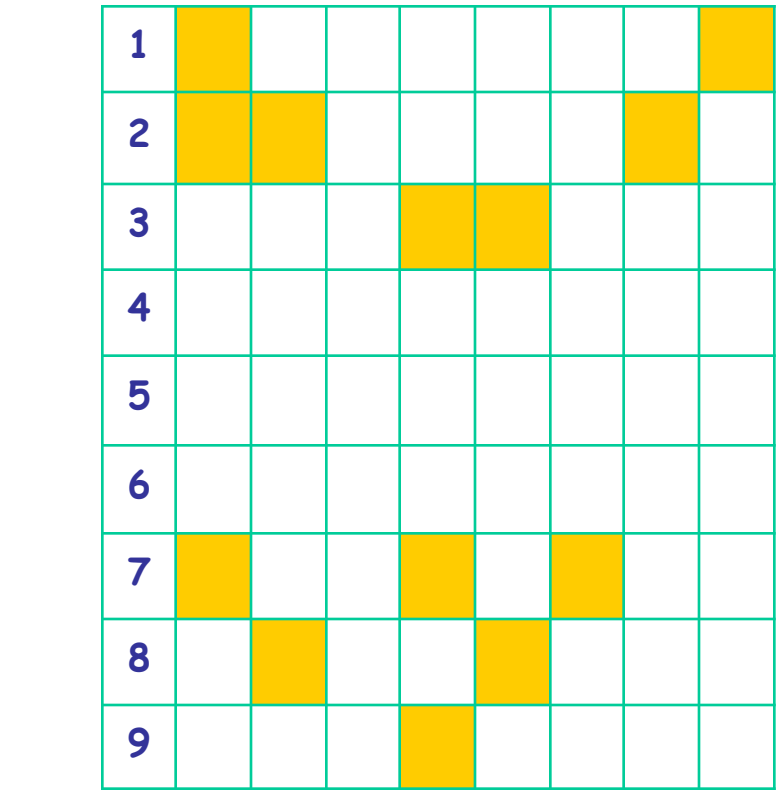
Multithreading Grano Grueso

- Conmuta entre threads solo en caso de largos stalls, como fallos de cache LLC (L3)
- Ventajas
  - No necesita conmutación entre thread muy rápida.
  - No retarda cada thread, la conmutación solo se produce cuando un thread no puede avanzar.
- Desventajas; no elimina perdidas por stalls cortos. La conmutación es costosa en ciclos.
  - Como CPU lanza instrucciones de un nuevo thread, el pipeline debe ser vaciado.
  - El nuevo thread debe llenar el pipe antes de que las instrucciones empiecen a completarse.
- Ejemplos: IBM AS/400, Montecito (Itanium2 9000), Sparc64 VI

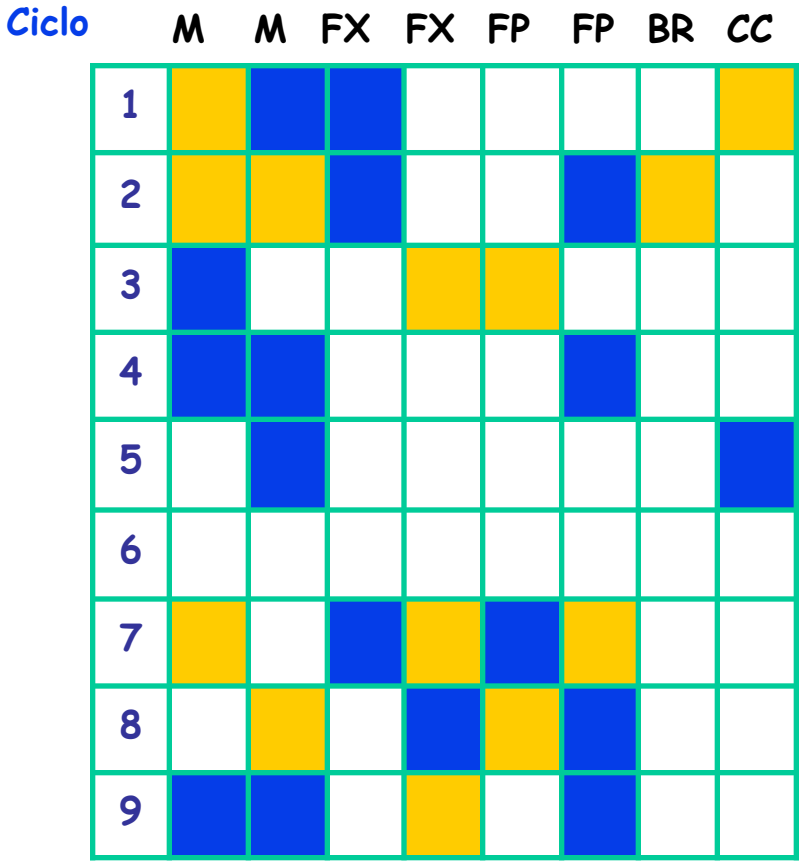
❑ Simultaneous Multithreading

Motivación: Recursos no usados en un procesador superescalar

Un thread, 8 unidades



Dos threads, 8 unidades

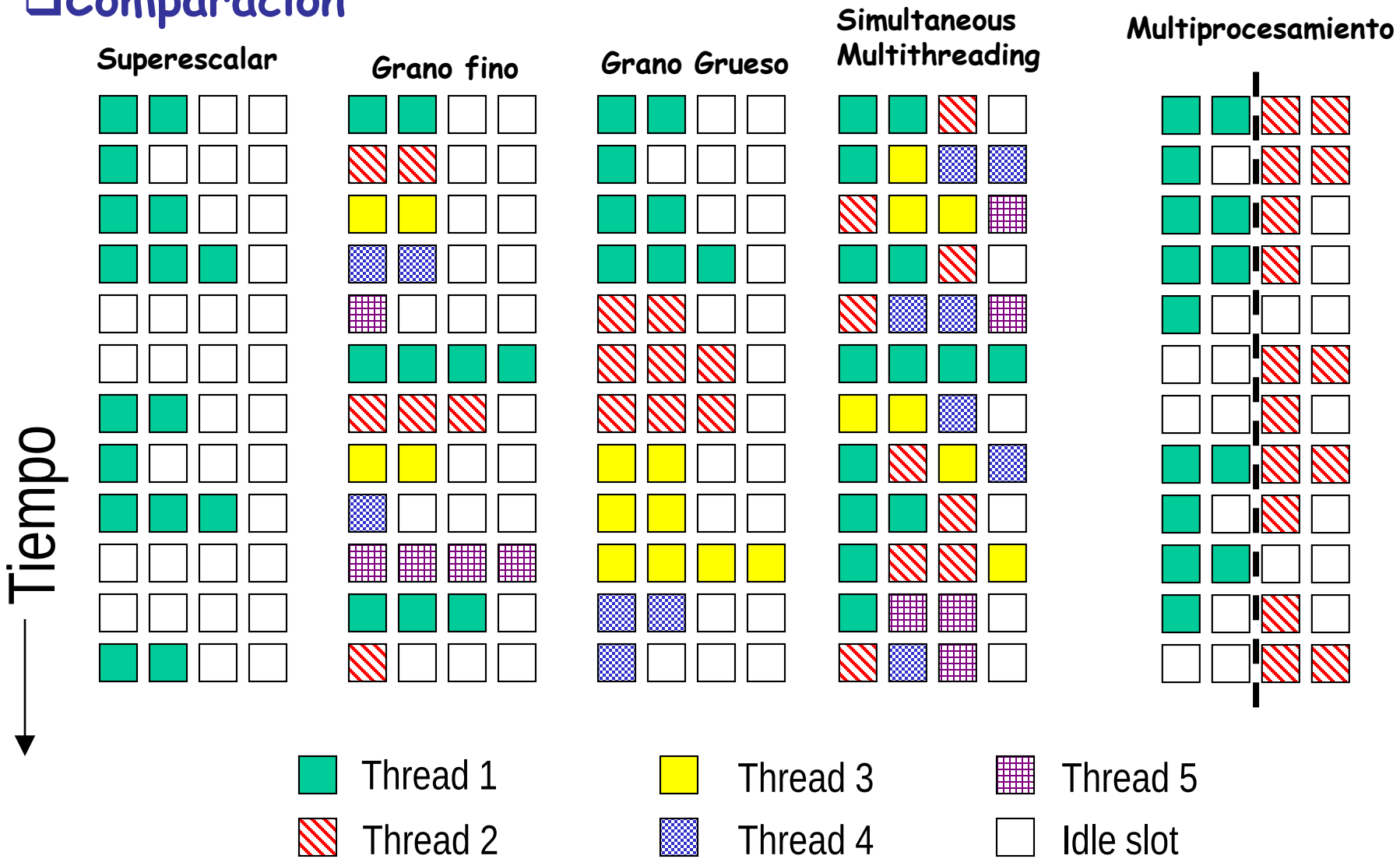


M = Load/Store, FX = Fixed Point, FP = Floating Point, BR = Branch, CC = Condition Codes

Simultaneous Multithreading (SMT)

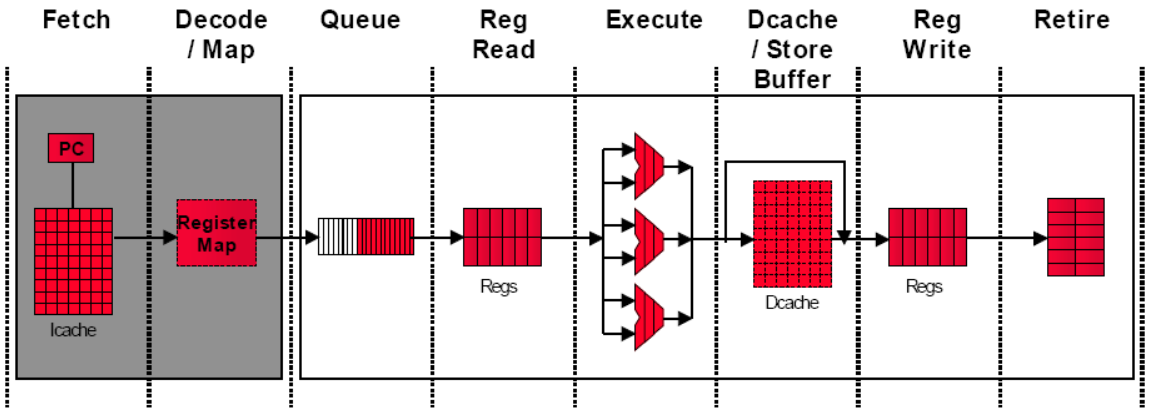
- Simultaneous multithreading (SMT): dentro de un procesador superescalar fuera de orden ya hay mecanismos Hw para soportar la ejecución de más de un thread
  - Gran numero de registros físicos donde poder mapear los registros arquitectónicos de los diferentes threads
  - El renombrado de registros proporciona un identificador único para los operandos de una instrucción, por tanto instrucciones de diferentes thread se pueden mezclar sin confundir sus operados
  - La ejecución fuera de orden permite una utilización eficaz de los recursos.
- Solo necesitamos sumar una tabla de renombrado por thread y PC separados
  - Commit independiente se soporta con un ROB por thread (Lógico o físico)
- Ojo conflictos en la jerarquía de memoria
- Ejemplos: Pentium4, Power5,6,7,8,9 Nehalem (2008), … AMD Zen, SPARC64-XII

❑Comparación



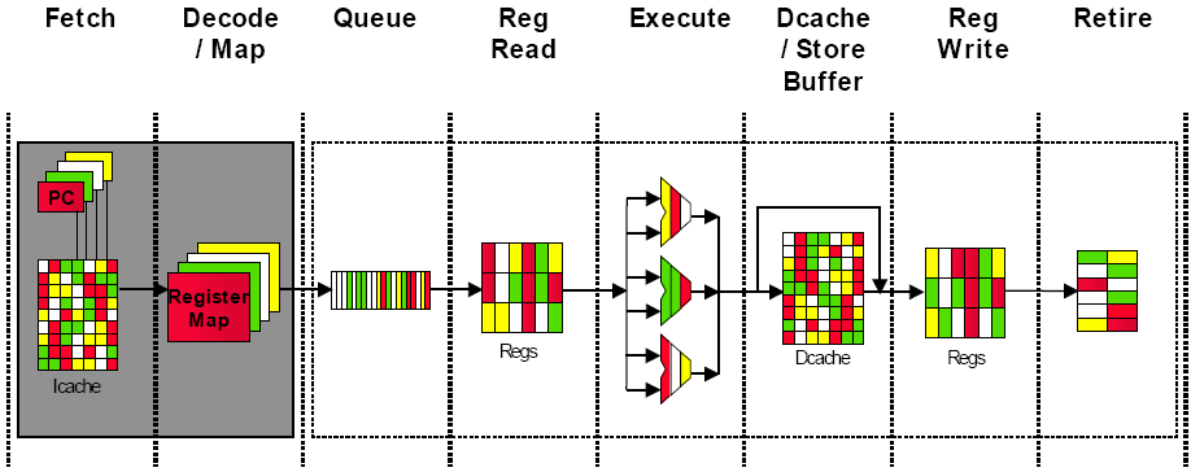
❑ Simultaneous multithreading

- ❑ 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

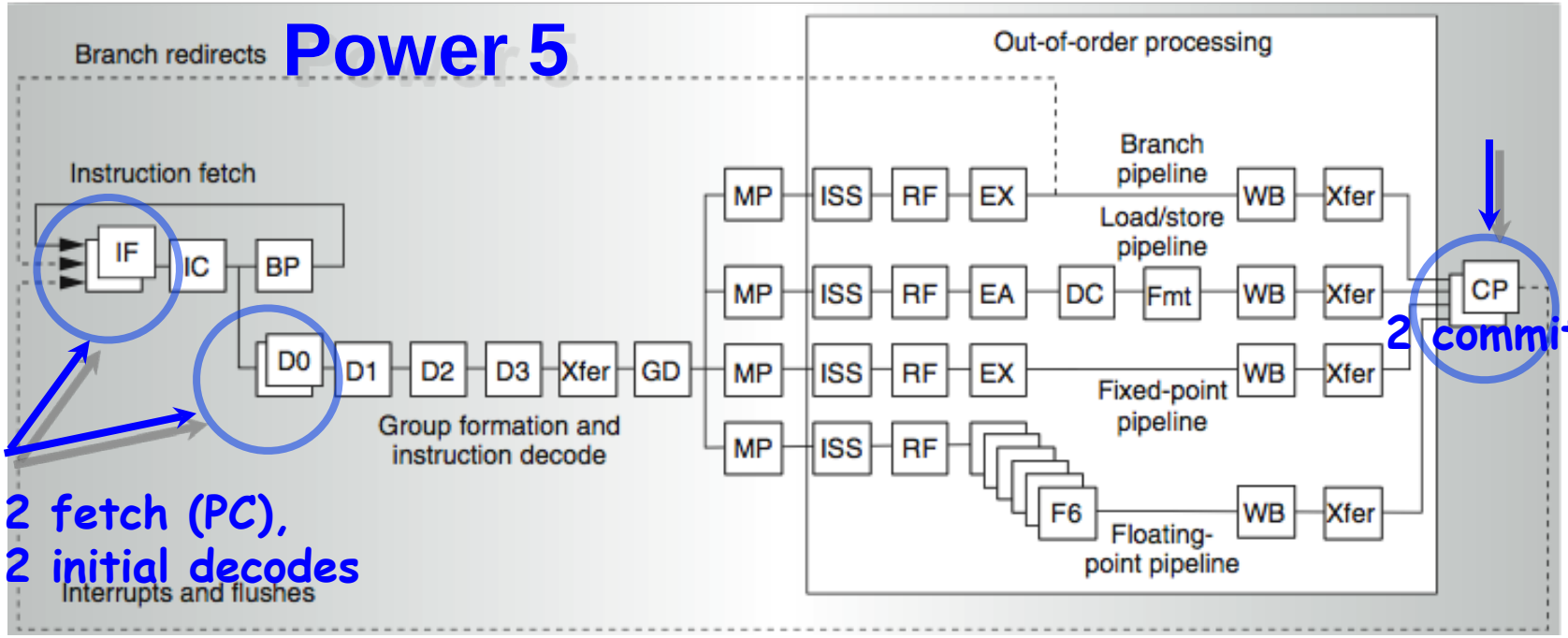
✓ los otros recursos se pueden compartir

Pentium-4 Hyperthreading (2002)

- Primer procesador comercial SMT 2 threads
- Los procesadores lógicos comparten casi todos los recursos del procesador físico.
  - Caches, unidades de ejecución, predictor de saltos
- Overhead de área por hyperthreading ~ 5%
- Cuando un procesador lógico (thread) se detiene el otro puede progresar usando los recursos
  - Ningún thread puede usar todos los recursos (colas) cuando hay 2 threads activos.
- Procesadores ejecutando solo un thread: se ejecuta a la misma velocidad que sin hyperthreading
- Hyperthreading se elimino de los diseños OoO sucesores del Pentium-4 (Pentium-M, Core Duo, Core 2 Duo), fue de nuevo incluido con el Nehalem en 2008 y se ha mantenido en todos los sucesores.

SMT Multithreading

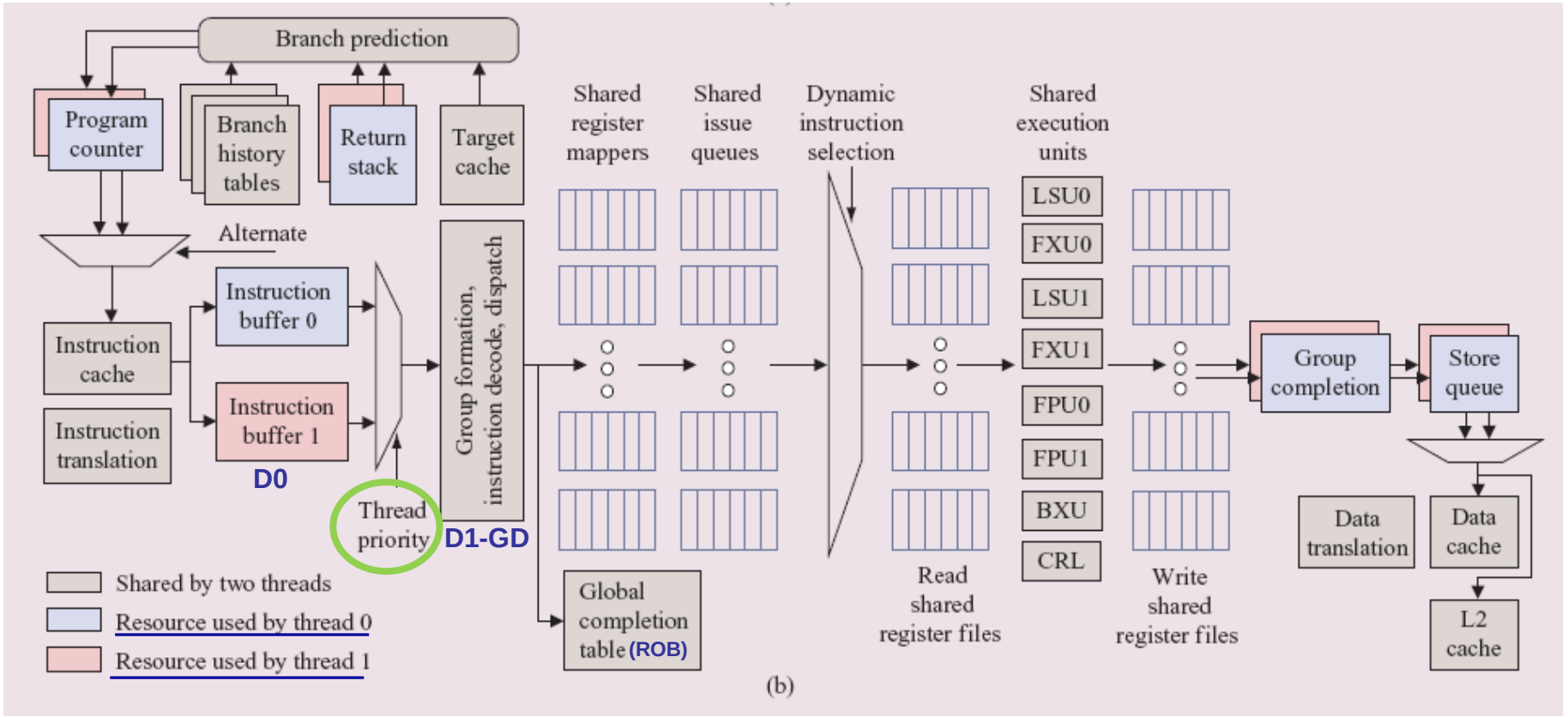
- IBM POWER5: Una adaptación de la arquitectura POWER4 para SMT



Observar: Muy pocos recursos duplicados, en comparación con la arquitectura del POWER4 que hemos estudiado.

Fuente: IBM J. Res. & Dev., Jul./Sep. 2005

❑ IBM POWER5 (2005): una vision más detallada



¿Por qué sólo 2 threads? Con 4, los recursos compartidos (registros físicos, cache, AB a memoria) son un cuello de botella.

❑ Power 5

Balanceo de la carga dinámica

1- Monitorizar

cola de fallos (load en L2)

entradas ocupadas en ROB (GCT)

2 – Quitarle recursos;

Reducir prioridad del hilo,

Inhibir decodificación (L2 miss)

Eliminar instrucciones desde emisión y parar decodificación

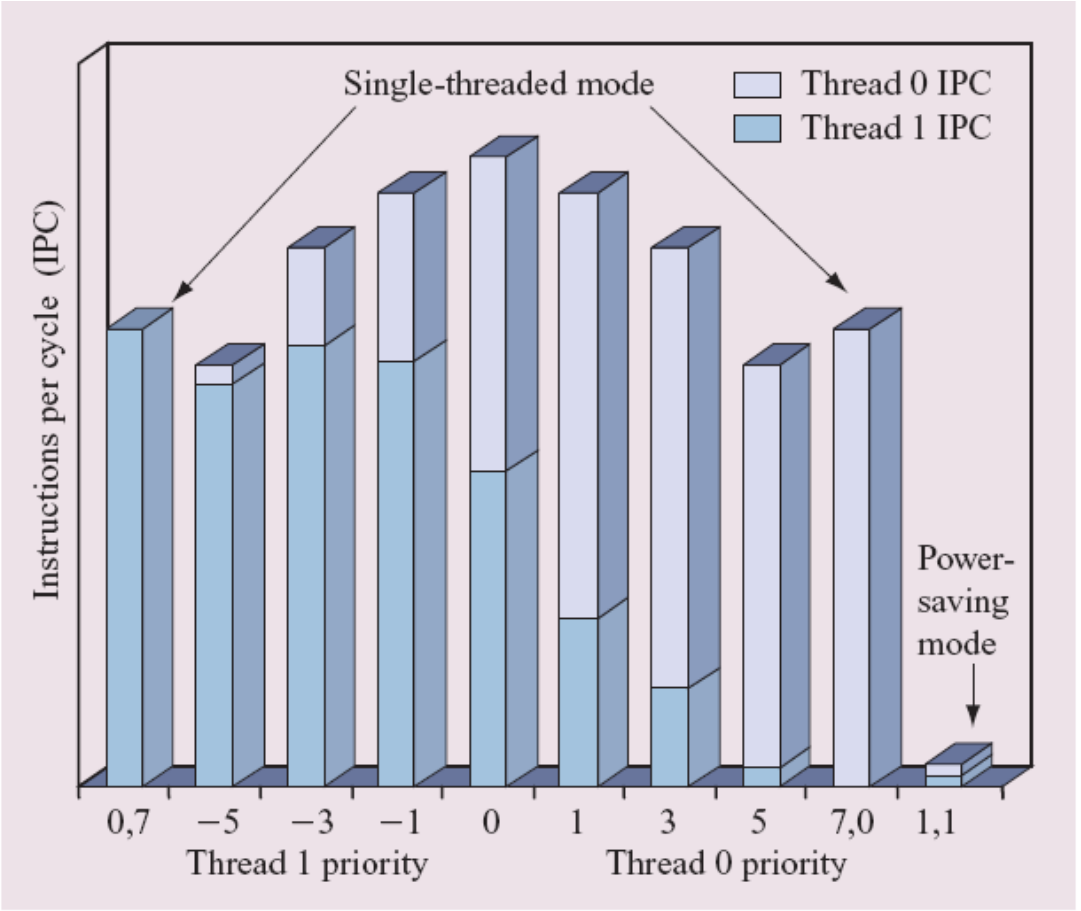
3- Ajustar prioridad del hilo (hardware/software)

Baja en espera activa

Alta en tiempo real

8 niveles de prioridad

Da mas ciclos de decodificación al de mas prioridad



❑ Cambios en Power 5 para soportar SMT

- Incrementar asociatividad de la L1 de instrucciones y del TLB
- Una cola de load/stores por thread
- Incremento de tamaño de la L2 (1.92 vs. 1.44 MB) y L3
- Un buffer de prebúsqueda separado por thread
- Incrementar el numero de registros físicos de 152 a 240
- Incrementar el tamaño de la colas de emisión
- El Power5 core es 24% mayor que el del Power4 para soportar SMT
- Más consumo, pero soportado con DVFS

POWER 8 (2015)

Technology

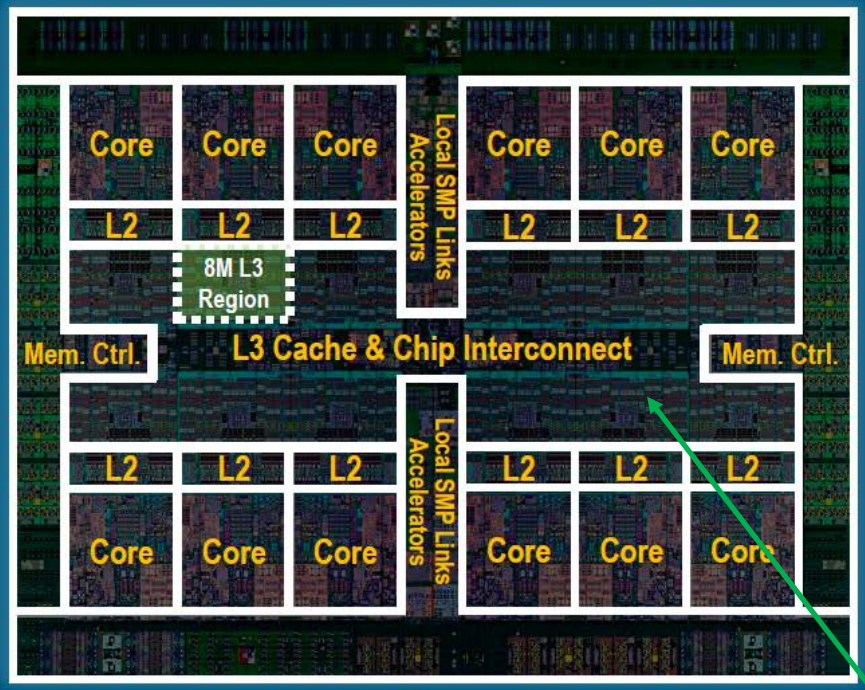
- 22nm SOI, eDRAM, 15 ML 650mm2

Cores

- 12 cores (SMT8)
- 8 dispatch, 10 issue, 16 exec pipe
- 2X internal data flows/queues
- Enhanced prefetching
- 64K data cache, 32K instruction cache

Accelerators

- Crypto & memory expansion
- Transactional Memory
- VMM assist
- Data Move / VM Mobility



Caches

- 512 KB SRAM L2 / core
- 96 MB eDRAM shared L3
- Up to 128 MB eDRAM L4 (off-chip)

Memory

- Up to 230 GB/s sustained bandwidth

Bus Interfaces

- Durable open memory attach interface
- Integrated PCIe Gen3
- SMP Interconnect
- CAPI (Coherent Accelerator Processor Interface)

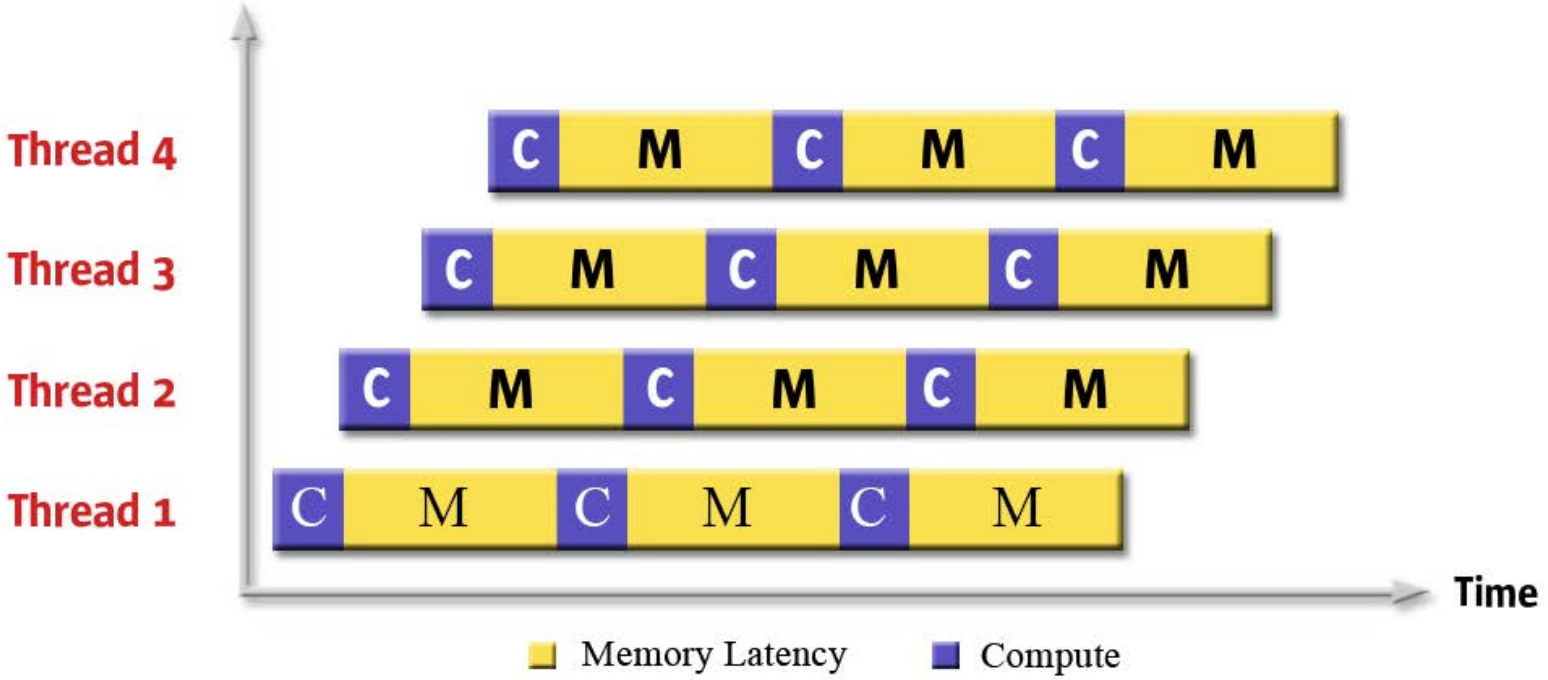
Energy Management

- On-chip Power Management Micro-controller
- Integrated Per-core VRM
- Critical Path Monitors

Área de L3 relativamente pequeña: gracias a uso de "embedded DRAM"

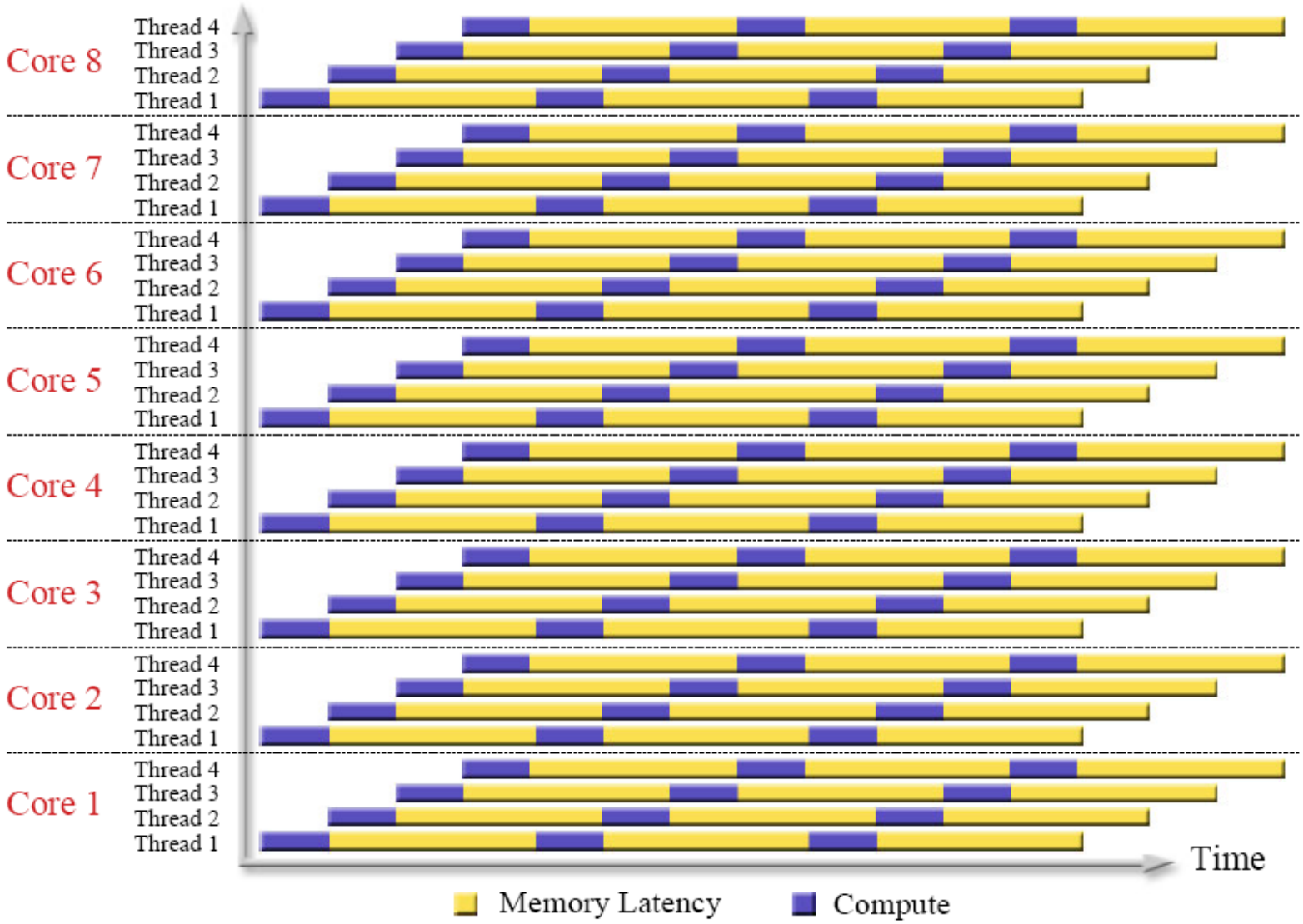
❑ Niagara (SUN 2005)

❑ Tolerar (soportar) la latencia de memoria mediante hilos concurrentes

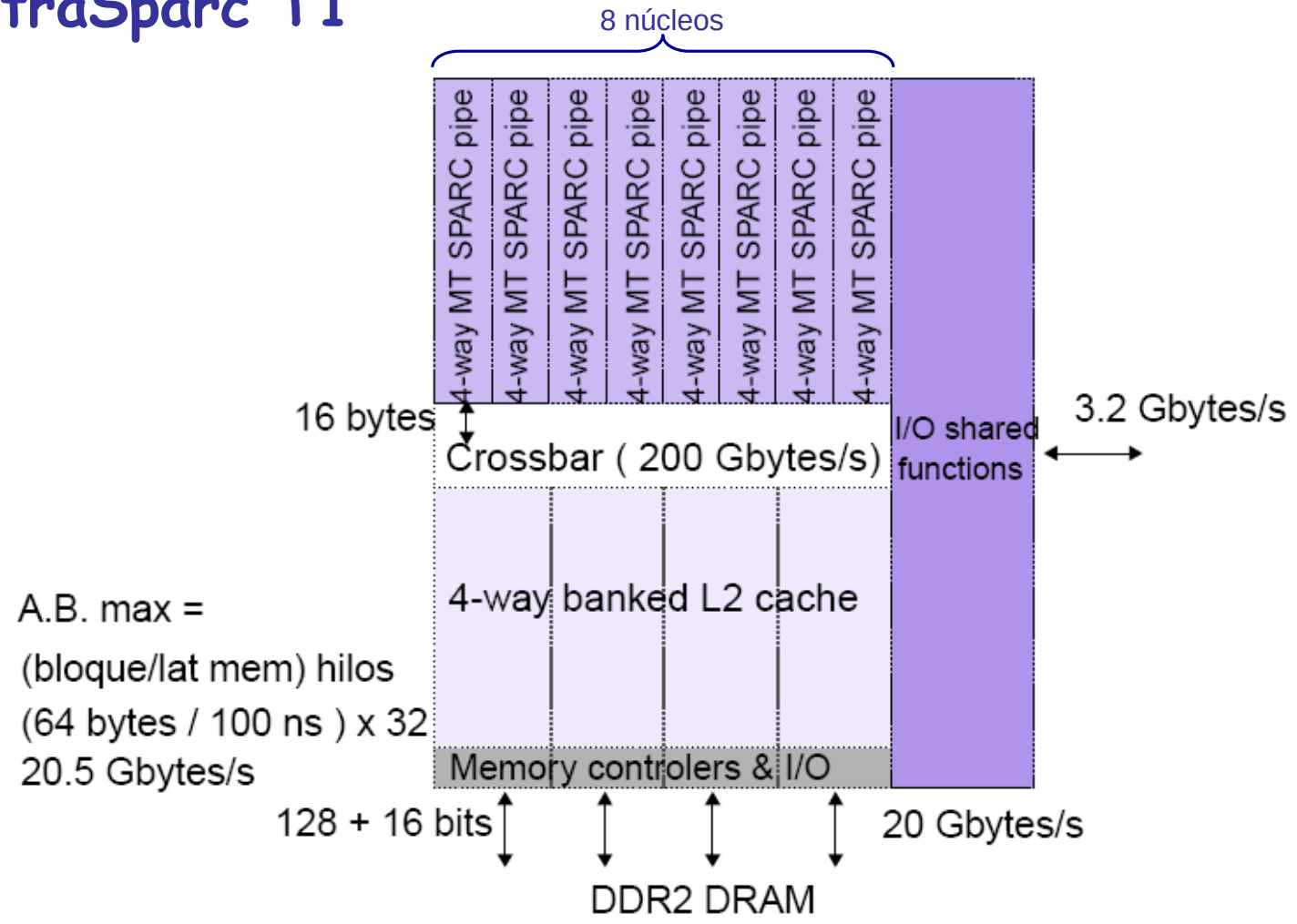


✓ Incrementa la utilización del procesador
✓ Es necesario un gran ancho de banda
- 4 accesos concurrentes a memoria

❑ Niagara: Múltiples cores-múltiples thread

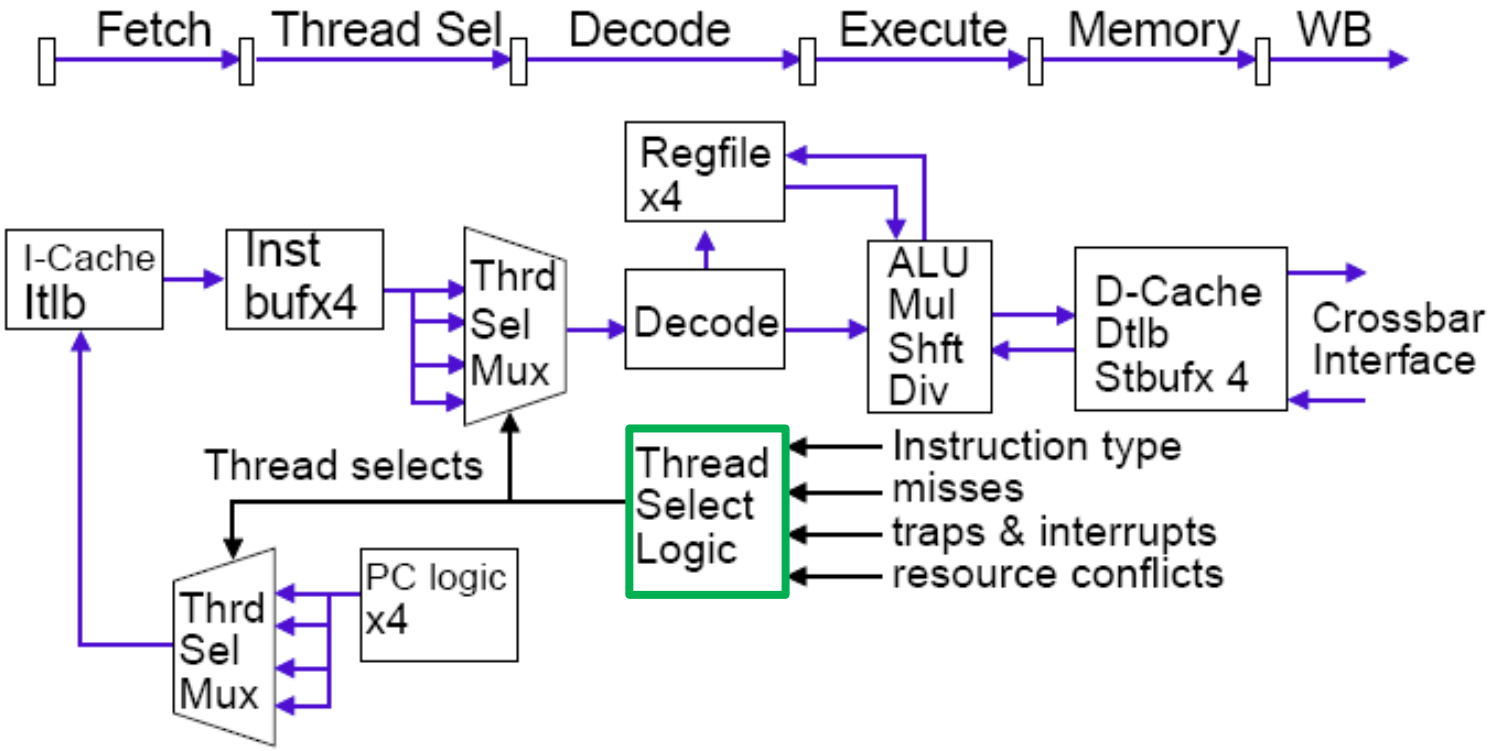


❑ Niagara UltraSparc T1



1Ghz, 1 instrucción por ciclo, 4 thread por core, 60 vatios
I-L1 16Kb(4-Way) / D-L1 8Kb (4-Way), escritura directa / L2, 3Mb(12-Way)
Crossbar no bloqueante, No predictor de saltos, no FP (1 por chip)

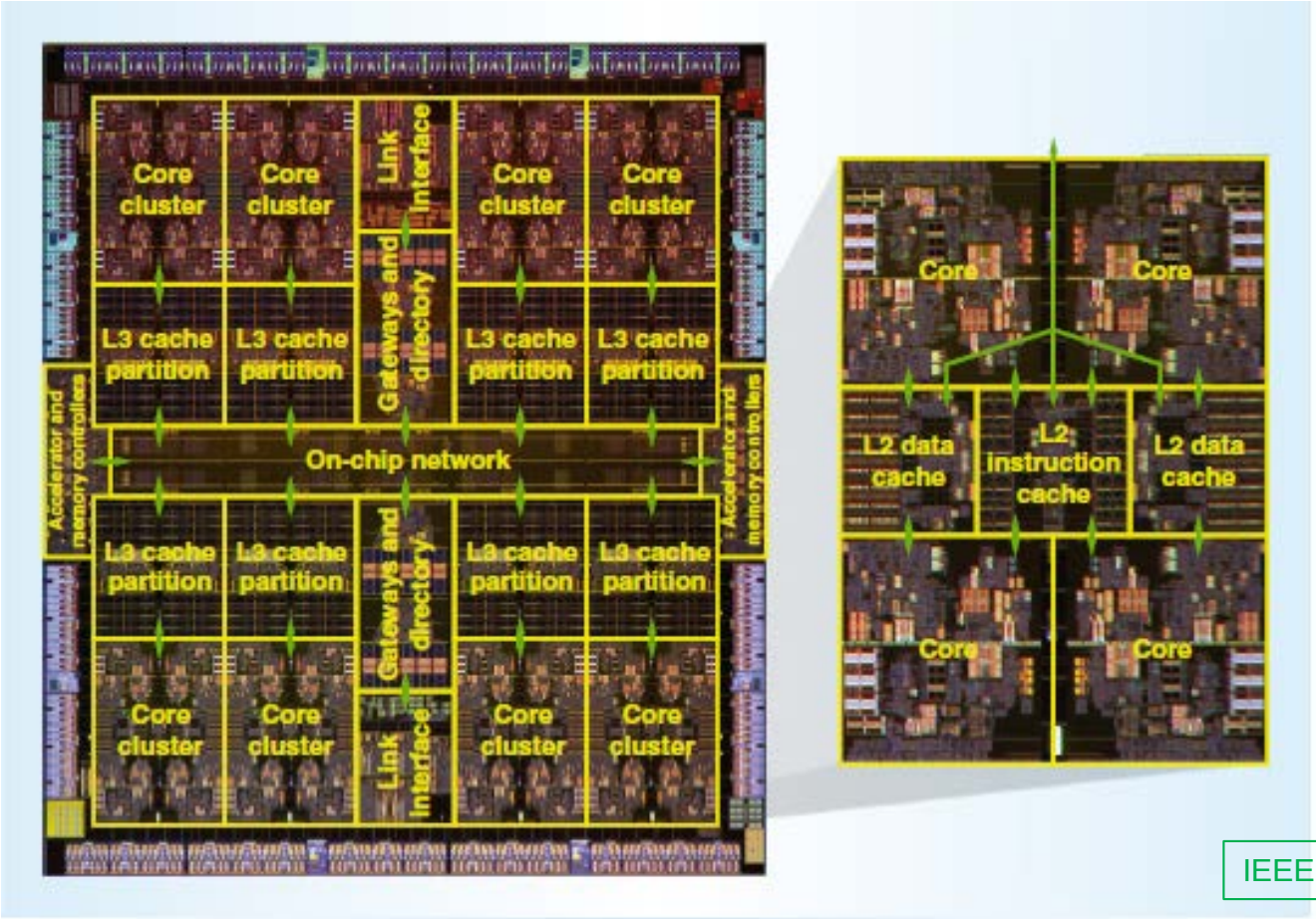
❑ Niagara UltraSparcT1



- 6 etapas
- 4 thread independientes
- algunos recursos x4: Banco de registros, contador de programa, store buffer, buffer de instrucciones
- el control de selección de hilo determina el hilo en cada ciclo de reloj
  - ✓cada ciclo elige un hilo

Evolución Niagara/Sparc

- Oracle Sparc M7
  - 32 cores / 8 threads, 256 threads/chip, 20nm, >10000 Mtrans, 699mm^2, 3.8 Ghz, 64MB L3
  - Hasta 8 socket/nodo: 2048 threads



IEEE Micro, abril 2015

Procesadores Oracle/Sun Niagara

- Orientación datacenters ejecutando servicios web y bases de datos (Oracle), con muchos threads ligeros.
- Muchos cores simples con reducida energía/ operación. Bajo rendimiento por thread

- Niagara-1 [2004], 8 cores, 4 threads/core
- Niagara-2 [2007], 8 cores, 8 threads/core
- Niagara-3 [2009], 16 cores, 8 threads/core
- T4 [2011], 8 cores, 8 threads/core
- T5 [2012], 16 cores, 8 threads/core
- M5 [2012], 6 cores, 8 threads/core
- M6[2013] 12 cores, 8 threads/core
- M7 [2014], 32 cores, 8 threads/core, 4Ghz

	Sparc M7	Sparc M6	Sparc M5
CPUs	32 CPUs	12 CPUs	6 CPUs
CPU Type	SPARC V9 S4	SPARC V9 S3	SPARC V9 S3
CPU Freq (max)	3.8GHz*	3.6GHz	3.6GHz
CPU Threads	8 threads per CPU, 256 threads total	8 threads per CPU, 96 threads total	8 threads per CPU, 48 threads total
CPU Cluster	4 CPUs per cluster	CPUs not clustered	CPUs not clustered
Superscalar Issue	Dual issue, out of order		
Pipeline Depth	16 stages		
L1 Cache (I+D)	16KB + 16KB		
L2 Cache	256KB I-cache per cluster, 256KB D-cache per pair, 6MB total	128KB per CPU, 1.5MB total	128KB per CPU, 768KB total
L3 Cache	8MB per cluster, 64MB total	48MB L3	48MB L3
DRAM Interface	16x 64-bit DDR4-2133/2400/2667	8x 64-bit DDR3-1066	8x 64-bit DDR3-1066
DRAM Bandwidth (peak)	341.3GB/s	68.2GB/s	68.2GB/s
DRAM Bandwidth (measured)	170GB/s (DDR4-2133)	57GB/s (DDR3-1066)	57GB/s (DDR3-1066)
DRAM Capacity	2TB per socket	1TB per socket	1TB per socket
Glueless SMP	Up to 8 sockets		
Coherent SMP	Up to 32 sockets	Up to 96 sockets	Up to 32 sockets
PCI Express	4x8 PCIe (Gen3)	2x8 PCIe (Gen3)	2x8 PCIe (Gen3)
Transistors	>10 billion	4.27 billion	3.9 billion
Die Size	700mm2*	643mm2	511mm2
IC Process	TSMC 20nm HKMG	TSMC 28nm HP	TSMC 28nm HP
Availability	Production 1H15*	Production 2013	Production 2012

- M8 [2017], 32 cores, 8 threads/core, 5Ghz, Quad-issue, out-of-order, 32 on-chip Data Analytics Accelerator (DAX) engines,

Procesadores Oracle/Sun Niagara

- Orientación datacenters ejecutando servicios web y bases de datos (Oracle), con muchos theads ligeros.
- Muchos cores simples con reducida energía/ operación. Bajo rendimiento por thread

- Niagara-1 [2004], 8 cores, 4 threads/core
- Niagara-2 [2007], 8 cores, 8 threads/core
- Niagara-3 [2009], 16 cores, 8 threads/core
- **2010 Oracle compra Sun Microsytems**
- T4 [2011], 8 cores, 8 threads/core
- T5 [2012], 16 cores, 8 threads/core
- M5 [2012], 6 cores, 8 threads/core
- M6[2013] 12 cores, 8 threads/core
- M7 [2014], 32 cores, 8 threads/core, 4Ghz

TABLE 2. SPARC M8, SPARC M7, SPARC M6, AND SPARC T5 PROCESSOR FEATURE COMPARISON.

Feature	SPARC M8 Processor	SPARC M7 Processor	SPARC M6 Processor	SPARC T5 Processor
CPU frequency	5.0 GHz	4.13 GHz	3.6 GHz	3.6 GHz
Out-of-order execution	Yes	Yes	Yes	Yes
Instruction issue width	4	2	2	2
Data/instruction prefetch	Yes	Yes	Yes	Yes
SPARC core	Fifth generation	Fourth generation	Third generation	Third generation
Cores per processor	32	32	12	16
Threads per core	8	8	8	8
Threads per processor	256	256	96	128
Sockets in systems	Up to 8	Up to 16	Up to 32	Up to 8
Memory per processor	Up to 16 DDR4 DIMMs	Up to 16 DDR4 DIMMs	Up to 32 DDR3 DIMMs	Up to 16 DDR3 DIMMs
Caches	32 KB L1 four-way instruction cache 16 KB L1 four-way data cache Shared 256 KB L2 four-way instruction cache (per quad cores) 128 KB L2 eight-way data cache (per core) Shared 64 MB (L3) cache	16 KB L1 four-way instruction cache 16 KB L1 four-way data cache Shared 256 KB L2 four-way instruction cache (per quad cores) Shared 256 KB L2 eight-way data cache (per core pair) Shared 64 MB (L3) cache	16 KB L1 four-way instruction cache 16 KB L1 four-way data cache 128 KB L2 eight-way cache Shared 48 MB L3 twelve-way cache	16 KB L1 four-way instruction cache 16 KB L1 four-way data cache 128 KB L2 eight-way cache Shared 8 MB L3 sixteen-way cache
Large page support[1]	16 GB	16 GB	2 GB	2 GB
Power management granularity	Half of the chip	A quarter of the chip	Entire chip	Entire chip
Technology	20 nm technology	20 nm technology	28 nm technology	28 nm technology

1. Large page support with Oracle Solaris 11.3

- M8 [2017], 32 cores, 8 threads/core, 5Ghz, Quad-issue, out-of-order, 32 on-chip Data Analytics Accelerator (DAX) engines,

Conclusiones -Limites del ILP

- Doblar en ancho de emisión (issue rates) sobre los valores actuales 4-8 instrucciones por ciclo, a digamos 8 a 16 instrucciones requiere en el procesador
  - Mas de 4 accesos a cache de datos por ciclo,
  - Predecir-resolver de 3 a 4 saltos por ciclo,
  - Renombrar y acceder a mas de 24 registros por ciclo,
  - Buscar de la cache de instrucciones de 12 a 24 instrucciones por ciclo.
- La complejidad de implementar estas capacidades implica al menos sacrificar la duración del ciclo e incrementa de forma muy importante el consumo.

Conclusiones -Limites del ILP

- La mayoría de la técnicas que incrementan rendimiento incrementan también el consumo.
- Una técnica es *eficiente en energía* si incrementa más el rendimiento que el consumo.
- Todas las técnicas de emisión múltiple (más allá de cierto punto) son poco eficientes desde el punto de vista de la energía.

Conclusiones -Limites del ILP

- En lugar de seguir explotando el ILP, los diseñadores se han focalizado sobre multiprocesadores en un chip (CMP, multicores,..)

- En el 2000, IBM abrió el campo con el 1º multiprocesador en un chip, el Power4, que contenía 2 procesadores Power3 y una cache L2 compartida. A partir de este punto todos los demás fabricantes han seguido el mismo camino. (Intel, AMD, Sun/Oracle, Fujitsu, y también Apple, Samsung, Qualcomm..).

- La explotación del paralelismo existente en las aplicaciones (a nivel de procesos y/o a nivel de datos) se plantea como la vía adecuada para obtener balances satisfactorios entre velocidad de procesamiento y consumo de energía...

- ...pero las ventajas de lanzar un nº razonable de instrucciones por ciclo, unida a la capacidad de la ejecución especulativa en desorden para adaptarse a eventos imprevisibles en tiempo de compilación (como los fallos de cache L1), hacen que los **procesadores especulativos con un ILP limitado sigan siendo el bloque básico de los diseños multicore.**

Lectura recomendada

H&P 5th ed., Sección 3.15: Concluding Remarks: What's Ahead?