

Arquitectura de Computadores



TEMA 5 Lanzamiento múltiple, Límites de ILP, Multithreading

DEPARTAMENTO DE
ARQUITECTURA DE **C**OMPUTADORES
Y **A**UTOMÁTICA

Curso 2010-2011

Contenidos

- o Introducción: $CPI < 1$
- o Lanzamiento múltiple de instrucciones: Superescalar, VLIW
- o Superescalar simple
- o VLIW
- o Superescalar con planificación dinámica
- o Límites de ILP
- o Ejemplo: Implementaciones X86
- o Thread Level Parallelism y Multithreading

- o Bibliografía
 - o Capítulo 3 y 4 de [HePa07]
 - o Capítulos 4 , 6 y 7 de [SiFK97]

□ Introducción

- ¿ Por que limitar a una instrucción por ciclo?
- Objetivo: $CPI < 1$
- Lanzar y ejecutar simultáneamente múltiples instrucciones por ciclo
- ¿Tenemos recursos?
 - Más área de silicio disponible
 - Técnicas para resolver las dependencias de datos (*planificación*)
 - Técnicas para resolver las dependencias de control (*especulación*)

Más ILP: Lanzamiento múltiple

❑ Alternativas

- ❑ Procesador Superscalar con planificación estática
- ❑ Procesador Superscalar con planificación dinámica+(especulación)
- ❑ Procesadores VLIW (very long instruction processor)
 - ✓ Superscalar
 - ✓ Lanza de 1 a 8 instrucciones por ciclo
 - ✓ Reglas de ejecución
 - o Ejecución en orden-planificación estática
 - o Ejecución fuera de orden-planificación dinámica
 - ✓ VLIW
 - ✓ Numero fijo de instrucciones por ciclo
 - ✓ Planificadas estáticamente por el compilador
 - ✓ EPIC (Explicitly Parallel Instruction Computing) Intel/HP

Más ILP: Lanzamiento múltiple


□ Alternativas

Tipo	Forma del issue	Detección de riesgos	Planificación	Ejemplos
Superescalar estático	Dinámico	HW	estática	Embedded MIPS, ARM
Superescalar dinámico	Dinámico	HW	dinámica	ninguno
Superescalar especulativo	Dinámico	HW	Dinámica con especulación	P4, Core2, Power5, SparcVI
VLIW	Estático	Básicamente SW	estática	TI C6x Itanium

Más ILP: Lanzamiento múltiple

□ SUPERESCALAR

Grado 2
2 Vías



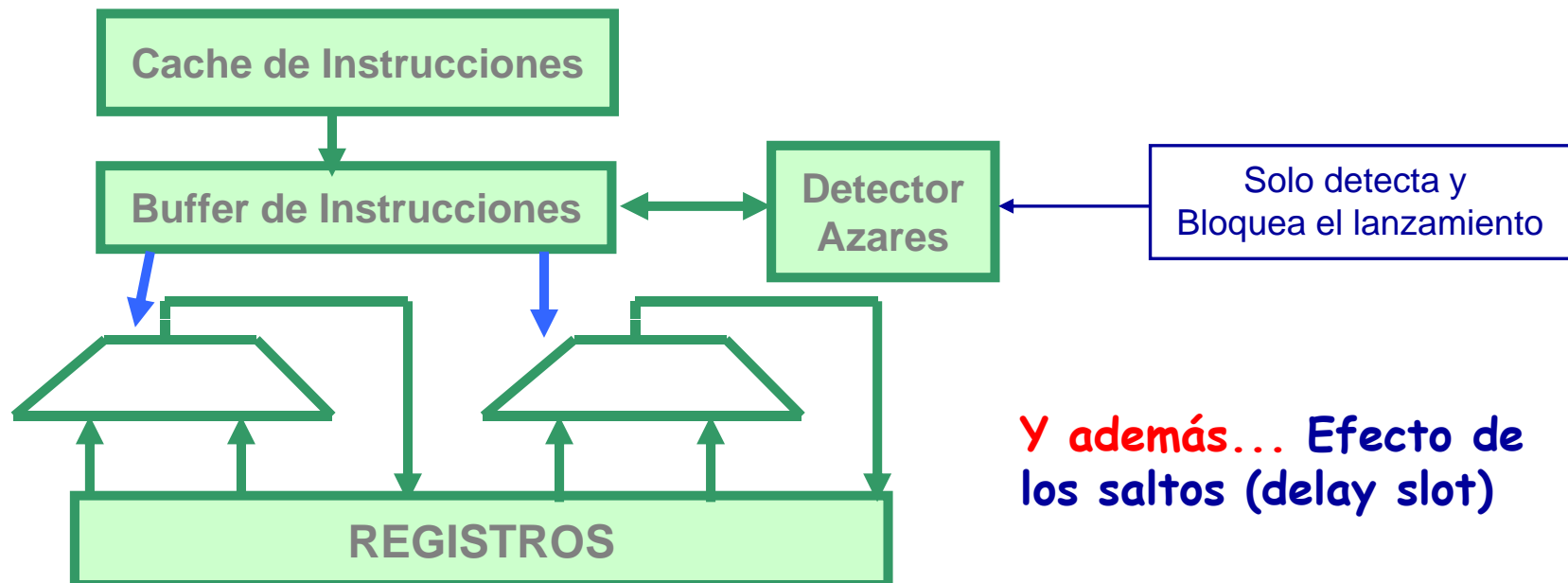
Instruction	1	2	3	4	5	6	7
i	IF	ID	EX	MEM	WB		
i+1	IF	ID	EX	MEM	WB		
i+2		IF	ID	EX	MEM	WB	
i+3		IF	ID	EX	MEM	WB	
i+4			IF	ID	EX	MEM	WB
i+5			IF	ID	EX	MEM	WB

- **Duplicar todos los recursos:**
 - o Puertas bloque de Registros
 - o Fus
 - o Puertas de memoria,...
 - o Control
- **Ideal CPI= 0.5 se reduce por:**
 - o Saltos
 - o LOADs
 - o Dependencias verdaderas LDE
- **Necesita:**
 - o Predicción sofisticada
 - o Tratamiento de LOAD; Cargas especulativas, técnicas de prebúsqueda
- **Más presión sobre la memoria**
- **Efecto incremental de los riesgos**
- **Se puede reducir complejidad con limitaciones (Un acceso a memoria por ciclo)**

Más ILP: Lanzamiento múltiple

❑ SUPERESCALAR Simple (estático, en orden)

- Regla de lanzamiento: Una instrucción FP+ una instrucción de cualquier otro tipo
- Buscar y decodificar dos instrucciones por ciclo (64 bits)
 - Ordenamiento y decodificación
 - Se analizan en orden. Sólo se lanza la 2ª si se ha lanzado la 1ª (conflictos)
- Unidades funcionales segmentadas (una ope. por ciclo) ó múltiples (división, raíz), más puertas en el bloque de registros
- Lanzamiento simple, recursos no conflictivos (diferentes reg y UF,..), excepto
 - Conflictos de recursos; load, store, move FP → más puertas en el bloque de reg.
 - Conflictos de datos LDE → más distancia entre instrucciones.



Más ILP: Lanzamiento múltiple

□ SUPERESCALAR Simple (estático, en orden)

	<u>Instrucción entera</u>	<u>Instrucción FP</u>	<u>Ciclo</u>
Loop:	LD F0,0(R1)		1
	LD F6,-8(R1)		2
	LD F10,-16(R1)	ADDD F4,F0,F2	3
	LD F14,-24(R1)	ADDD F8,F6,F2	4
	LD F18,-32(R1)	ADDD F12,F10,F2	5
	SD 0(R1),F4	ADDD F16,F14,F2	6
	SD -8(R1),F8	ADDD F20,F18,F2	7
	SD -16(R1),F12		8
	SD -24(R1),F16		9
	SUBI R1,R1,#40		10
	BNEZ R1,LOOP		11
	SD 8(R1),F20		12

Separadas por 2 ciclos

- Desarrollo para ejecución superescalar: se desarrolla una iteración más.
12 ciclos, 2.4 ciclos por iteración
- El código máquina está compacto en la memoria

Más ILP: Lanzamiento múltiple

❑ SUPERESCALAR Simple (estático, en orden)

➤ *Ventajas*

- No modifica código. Compatibilidad binaria
- No riesgos en ejecución

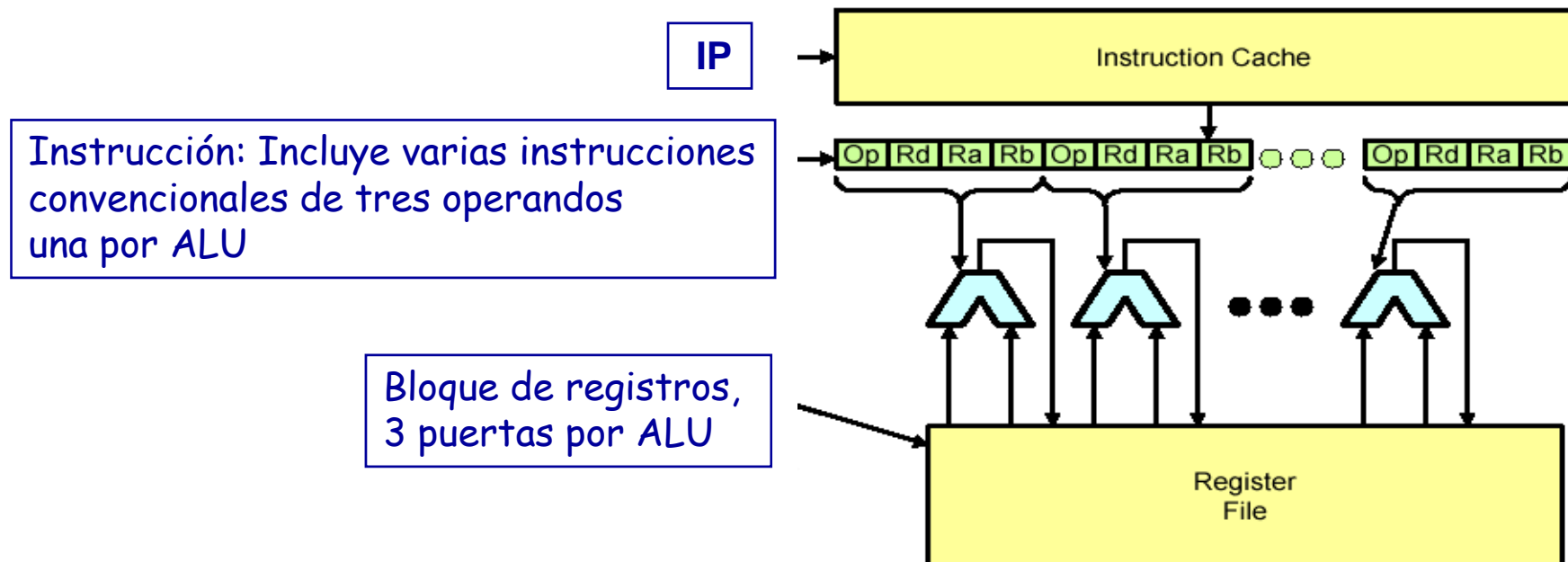
➤ *Desventajas*

- Mezcla de instrucciones. Solo obtiene CPI de 0.5 en programas con 50 % de FP
- Bloqueos en el lanzamiento
- Planificación fija: No puede adaptarse a cambios en ejecución (Fallos de cache)
- Los códigos deben de ser replanificados para cada nueva implementación (eficiencia)

Más ILP: Lanzamiento múltiple

□ VLIW

- El análisis de dependencias en tiempo de compilación
- Muchas operaciones por instrucción (IA64 packet, Tramsmeta molecula)
- Todas las operaciones de una instrucción se ejecutan en paralelo
- Instrucciones con muchos bits
- Muchas operaciones vacías (NOP)



Más ILP: Lanzamiento múltiple

□ VLIW Ejemplo Tema3

```
LOOP    LD      F0,0(R1)
        ADDD    F4,F0,F2
        SD      0(R1),F4
        SUBI    R1,R1,#8
        BNEZ    R1,LOOP
```

- Aplicar técnicas conocidas para minimizar paradas

- Unrolling
- Renombrado de registros



- Latencias de uso: LD a ADD 1 ciclo, ADD a SD 2 ciclos
- Opción: desarrollar 4 iteraciones y planificar: 14 ciclos, 3.5 ciclos por iteración

```
LOOP:   LD      F0, 0(R1)
        LD      F6, -8(R1)
        LD      F10, -16(R1)
        LD      F14, -24(R1)
        ADDD    F4, F0, F2
        ADDD    F8, F6, F2
        ADDD    F12, F10, F2
        ADDD    F16, F14, F2
        SD      0(R1), F4
        SD      -8(R1), F8
        SD      -16(R1), F12
        SUBI    R1, R1, #32
        BNEZ    R1, LOOP
        SD      8(R1), F16; 8-32 = -24
```

Más ILP: Lanzamiento múltiple

□ VLIW

Loop unrolling en VLIW

LOOP: LD F0,0(R1) ; F0 = array element
ADD F4,F0,F2 ; add scalar in F2
SD 0(R1),F4 ; store result
SUBI R1,R1,#8 ; decrement pointer
BNEZ R1, LOOP ; branch if R1!=0

<u>Mem ref 1</u>	<u>Mem ref 2</u>	<u>FP op</u>	<u>FP op</u>	<u>Int op/branch</u>
LD F0,0(R1)	LD F6,-8(R1)			
LD F10,-16(R1)	LD F14,-24(R1)			
LD F18,-32(R1)	LD F22,-40(R1)	ADD F4,F0,F2	ADD F8,F6,F2	
LD F26,-48(R1)		ADD F12,F10,F2	ADD F16,F14,F2	
		ADD F20,F18,F2	ADD F24,F22,F2	
SD 0(R1),F4	SD -8(R1),F8	ADD F28,F26,F2		
SD -16(R1),F12	SD -24(R1),F16			
SD -32(R1),F20	SD -40(R1),F24			SUBI R1,R1,#56
SD 8(R1),F28				BNEZ R1, LOOP

- ✓ 7 iteraciones en 9 ciclos: 1.3 ciclos por iteración
- ✓ 23 operaciones en 45 slots (~50% de ocupación)
- ✓ Muchos registros necesarios

Más ILP: Lanzamiento múltiple

□ VLIW

VENTAJAS

- Hardware muy simple
 - No detecta dependencias
 - Lógica de lanzamiento simple
- Puede explotar paralelismo a todo lo largo del programa

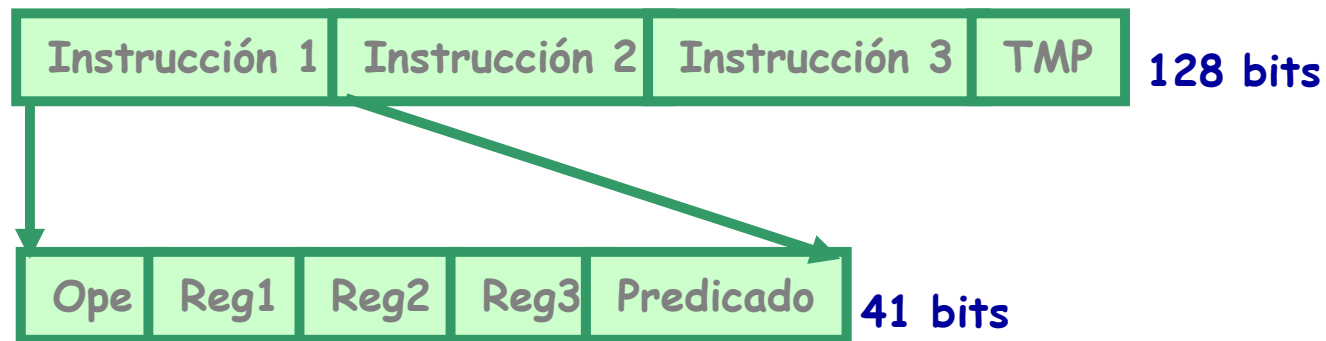
DESVENTAJAS

- Planificación estática; Muy sensible a fallos de cache
- Necesita desenrollado muy agresivo
- Bloque de registros muy complejo en área y tiempo de acceso
- Muchas NOP
 - Poca densidad de código
 - Capacidad y AB de la cache de instrucciones
- Compilador muy complejo
- No binario compatible
- Operación síncrona para todas las operaciones de una instrucción

Más ILP: Lanzamiento múltiple

□EPIC: Explicitly Parallel Instruction Computing IA64

- Instrucciones de 128 bits
 - Operaciones de tres operandos
 - TMP codifica dependencias entre las operaciones
 - 128 registros enteros (64bits), 128 registros FP (82bits)
 - Ejecución predicada. 64 registros de predicado de 1 bit
 - Cargas especulativas
 - Hw para chequeo de dependencias



Primera implementación Itanium (2001), 6 operaciones por ciclo, 10 etapas, 800Mhz

Segunda implementación Itanium2 (2005), 6 operaciones por ciclo, 8 etapas, 1,66Ghz

Más ILP: Lanzamiento múltiple

□ SUPERESCALAR con Planificación Dinámica. Fuera de orden

➤ Un Diseño Simple

- Estaciones de reserva separadas para enteros (+reg) y PF (+reg)
- Lanzar dos instrucciones en orden (ciclo de lanzamiento: partir en dos subciclos)
- Solo FP load causan dependencias entre instrucciones enteras y PF
 - Reemplazar buffer de load con cola. Las lecturas se hacen en orden
 - Ejecución Load: "check" dirección en cola de escritura para evitar LDE
 - Ejecución Store: "check" dirección en cola de lecturas para evitar EDL

➤ Rendimiento del procesador

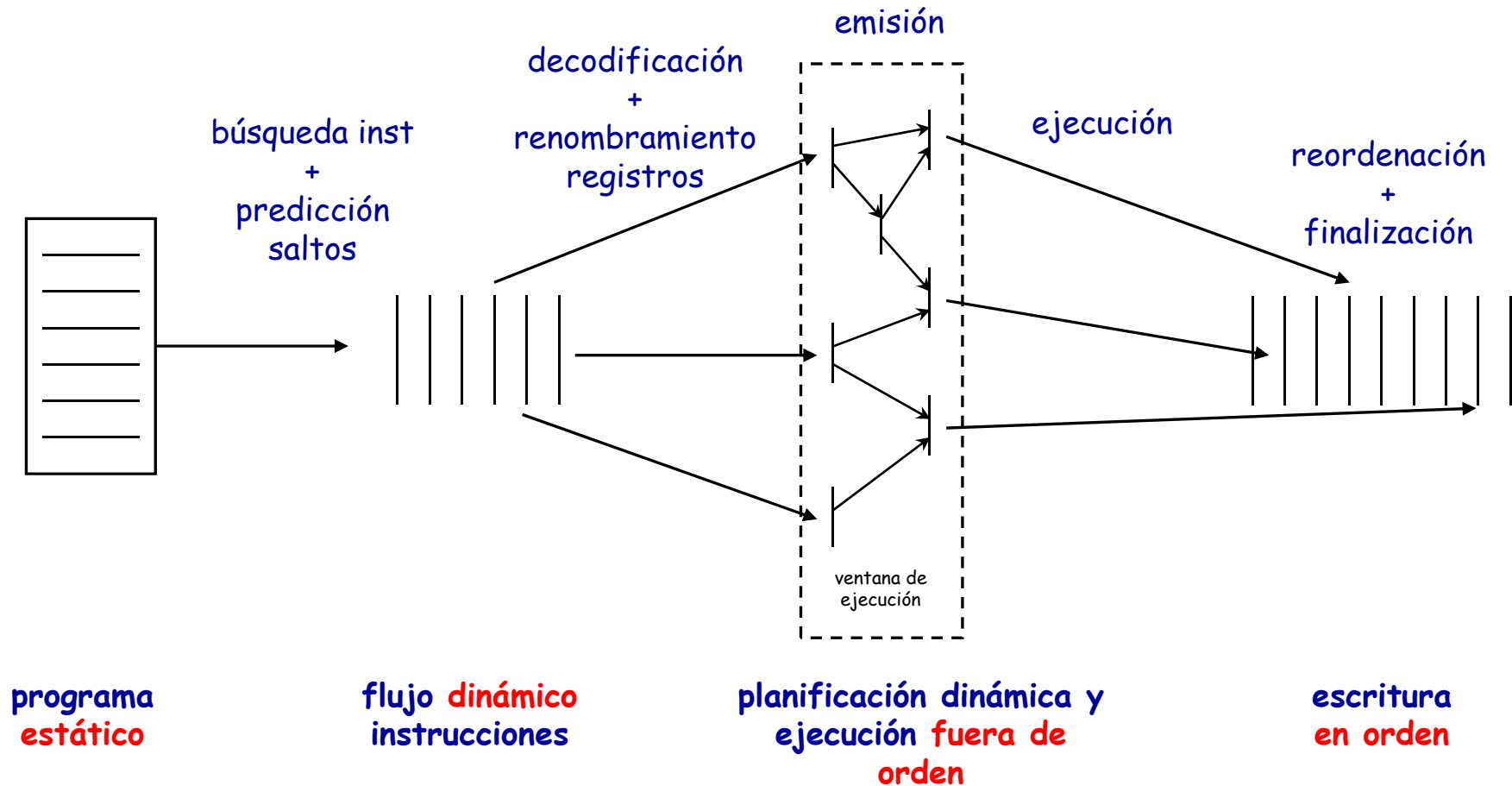
<u>Iteración</u> <u>no.</u>	<u>Instrucción</u>	<u>Lanzada</u>	<u>Ejecutada</u> (número de ciclo)	<u>Escribe resultado</u>
1	LD F0,0(R1)	1	2	4
1	ADDD F4,F0,F2	1	5	8 ←
1	SD 0(R1),F4	2	9	
1	SUBI R1,R1,#8	3	4	5
1	BNEZ R1,LOOP	4	6	
2	LD F0,0(R1)	5	6	8 ←
2	ADDD F4,F0,F2	5	9	12
2	SD 0(R1),F4	6	13	
2	SUBI R1,R1,#8	7	8	9
2	BNEZ R1,LOOP	8	10	

4 ciclos por
iteración

Más ILP: Lanzamiento múltiple

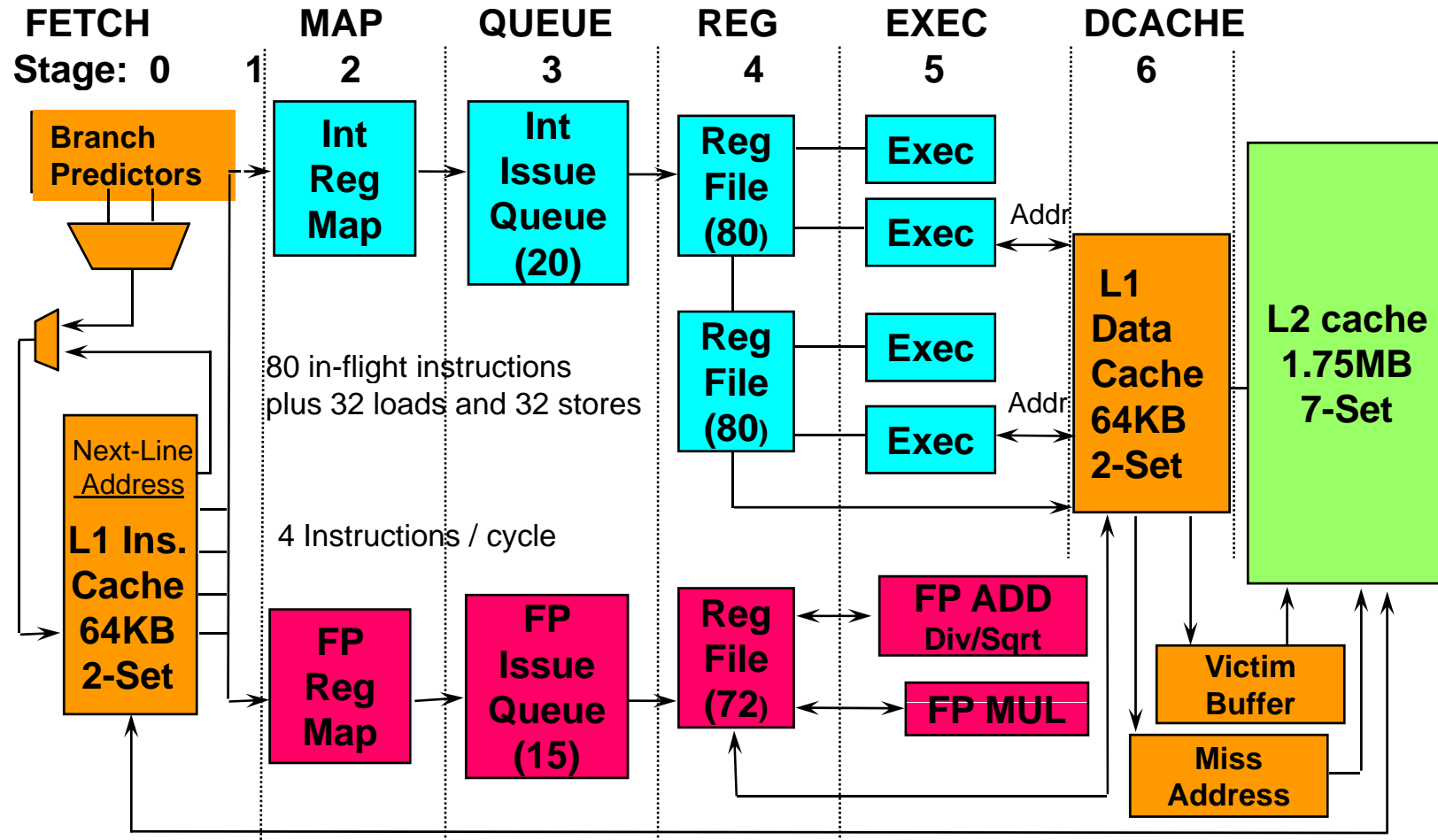
❑ SUPERESCALAR con Planificación Dinámica y Especulación

Ejecución fuera de orden. Finalización en orden



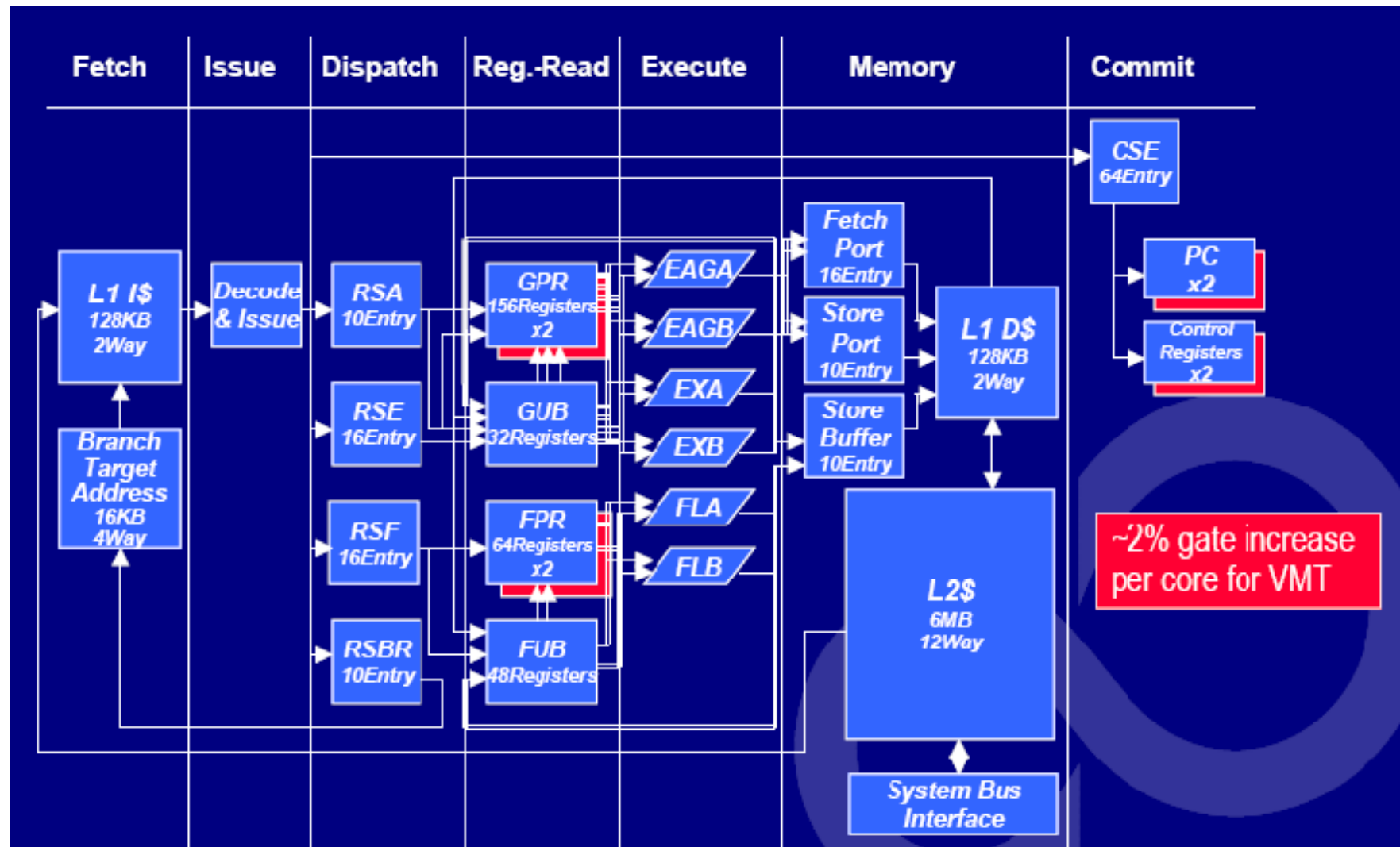
Más ILP: Lanzamiento múltiple

□ EV7 ALPHA 21364 Core (2003)

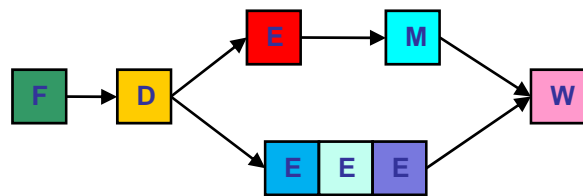


Más ILP: Lanzamiento múltiple

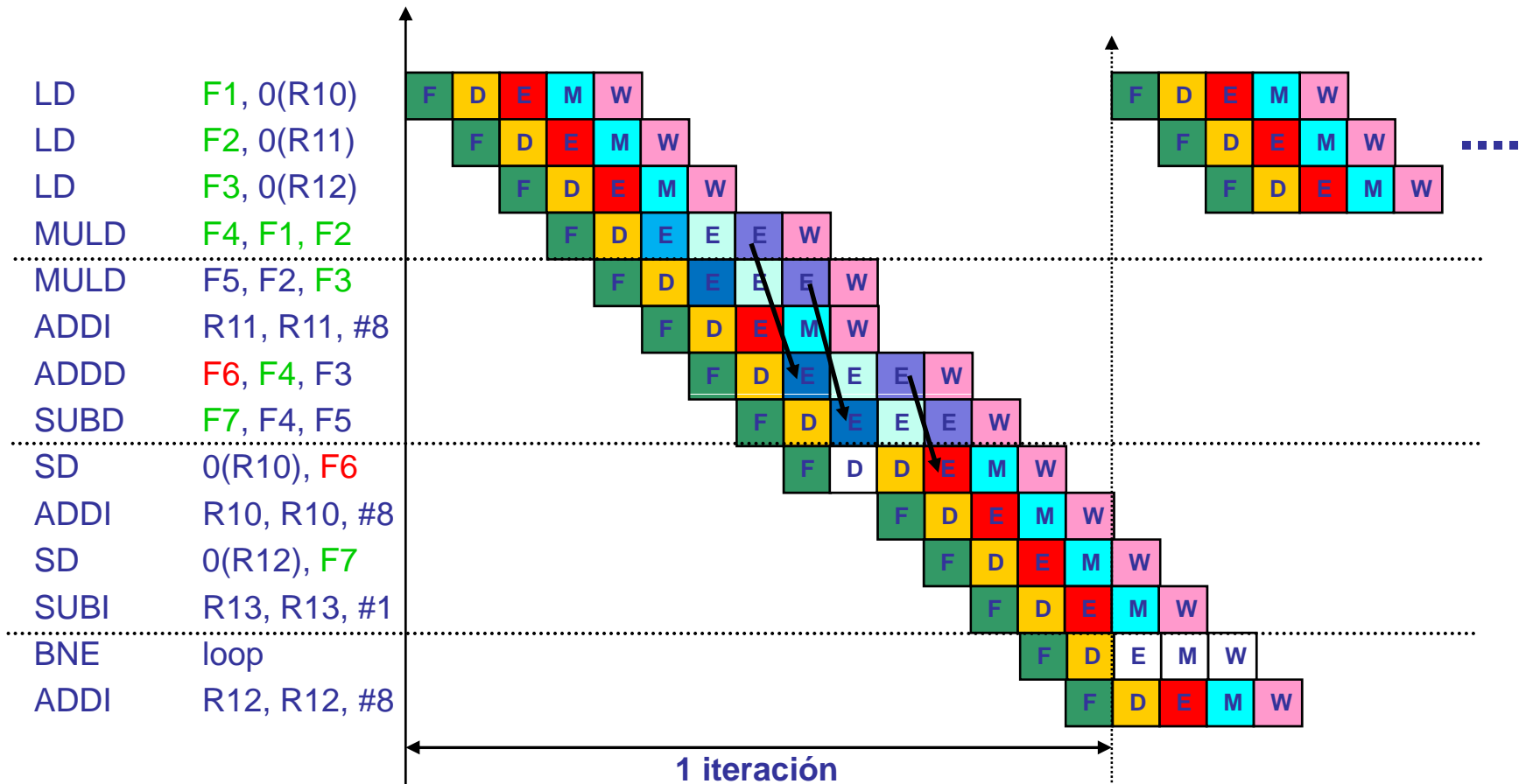
❑ SPARC64 VI (2006/7)



❑ Ejecución escalar con una unidad entera y una de PF

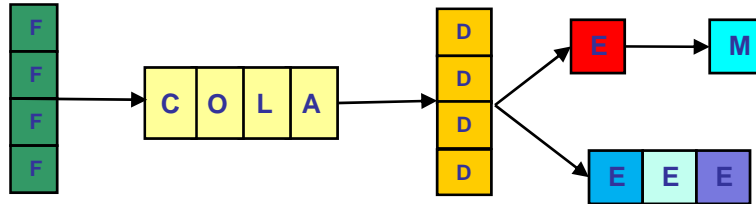


Una instrucción entera y una PF pueden escribir en el mismo ciclo
Los saltos se resuelven en D
Bancos de registros separados para enteros y flotantes



Modelos de Ejecución

Superescalar de 4 vías, Ejecución en orden, Predicción de saltos

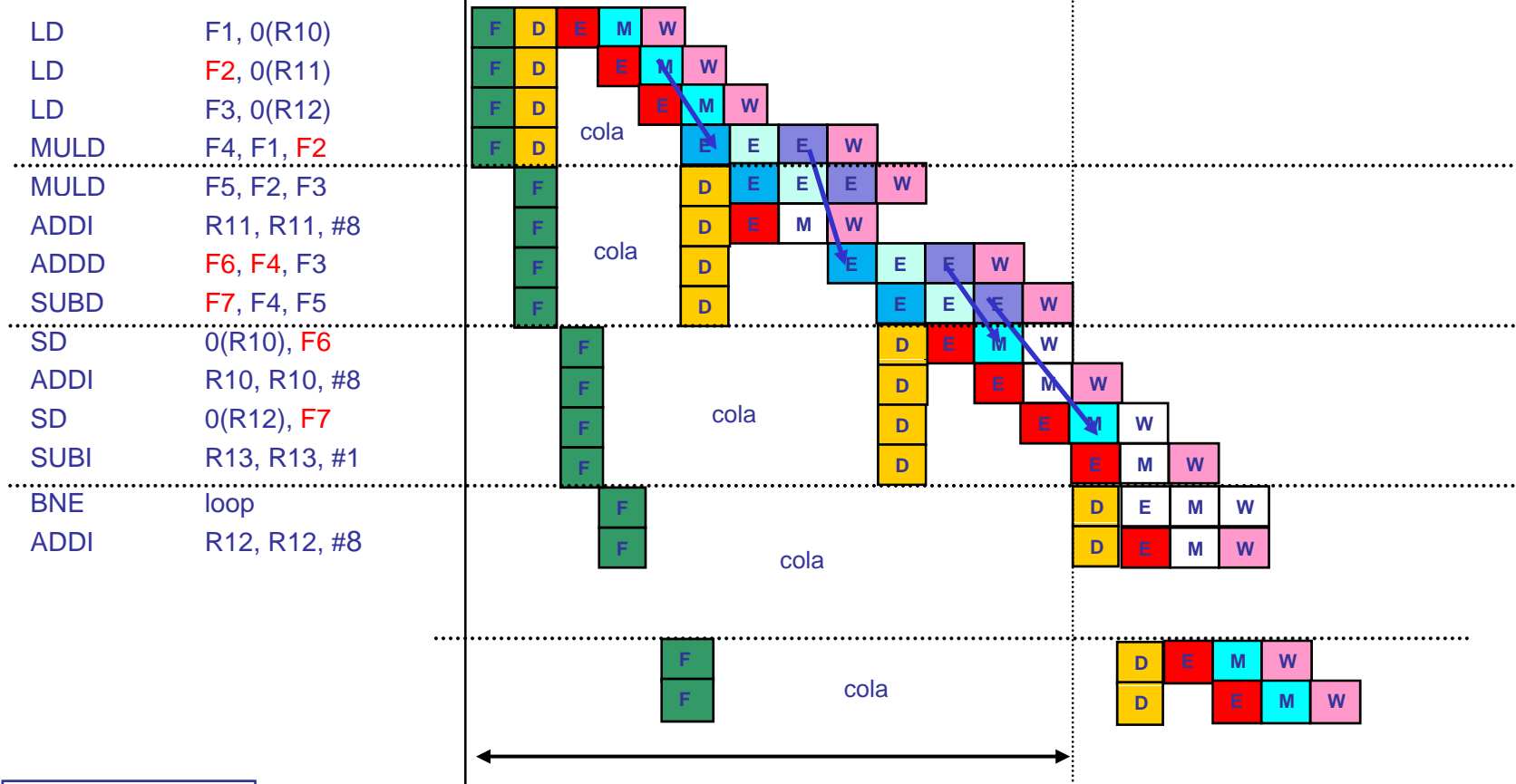


La lectura de operandos se sigue produciendo en la decodificación Cola de instrucciones. Las instrucciones esperan a pasar al decodificador

Hay logica de cortocircuito en la etapa de memoria

Dos instrucciones pueden estar en M pero solo una puede acceder

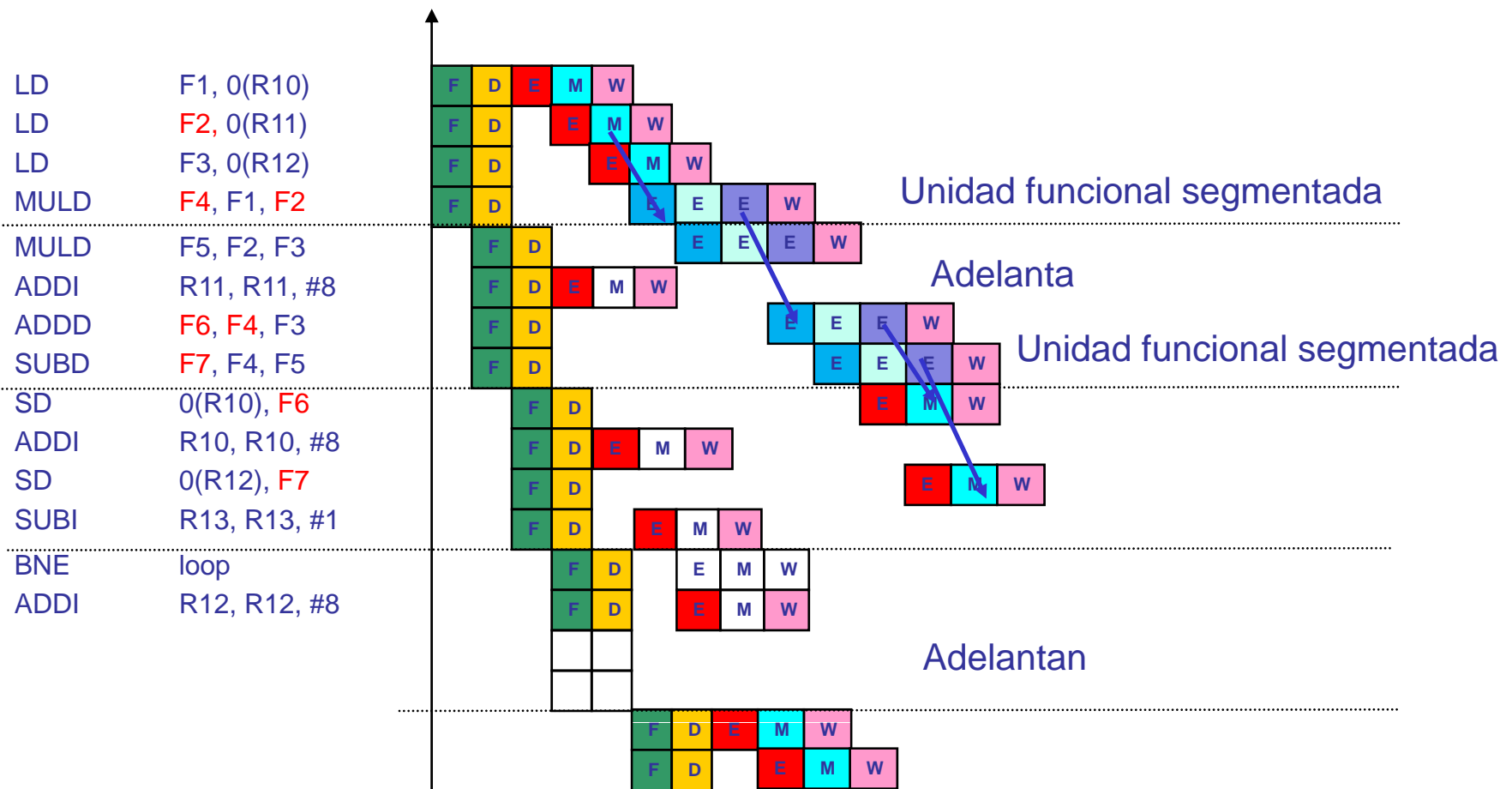
Solo un load por ciclo. Una UF enteros y una de PF



Modelos de Ejecución

Superescalar de 4 vías, Ejecución fuera de orden, Predicción de saltos

Basado en Algoritmo de Tomasulo, las instrucciones esperan en ER
Solo un load por ciclo
Una UF entera y otra PF



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?
 - o Extensiones multimedia:
 - o Intel MMX,SSE,SSE2,SSE3, SSE4
 - o Motorola Altivec, Sparc, SGI, HP

Límites del ILP

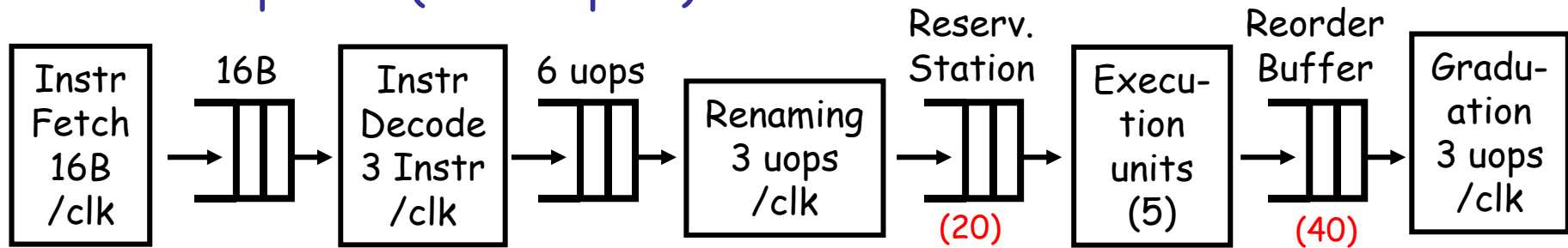
- ❑ ¿Cuanto ILP hay en las aplicaciones?
- ❑ Supongamos un procesador superescalar fuera de orden con especulación y con recursos ilimitados
 - o Infinitos registros para renombrado
 - o Predicción perfecta de saltos
 - o Caches perfectas
 - o Lanzamiento no limitado
 - o Desambiguación de memoria perfecta

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	48 enteros + 40 Fl. Pt.
Predicción de saltos	1K 2-bit	Perfecta	Tournament
Cache	Perfecto	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3
Análisis de Memory Alias	HW disambiguation	Perfecto	Perfecto

□ P6 Pipeline (14 etapas)

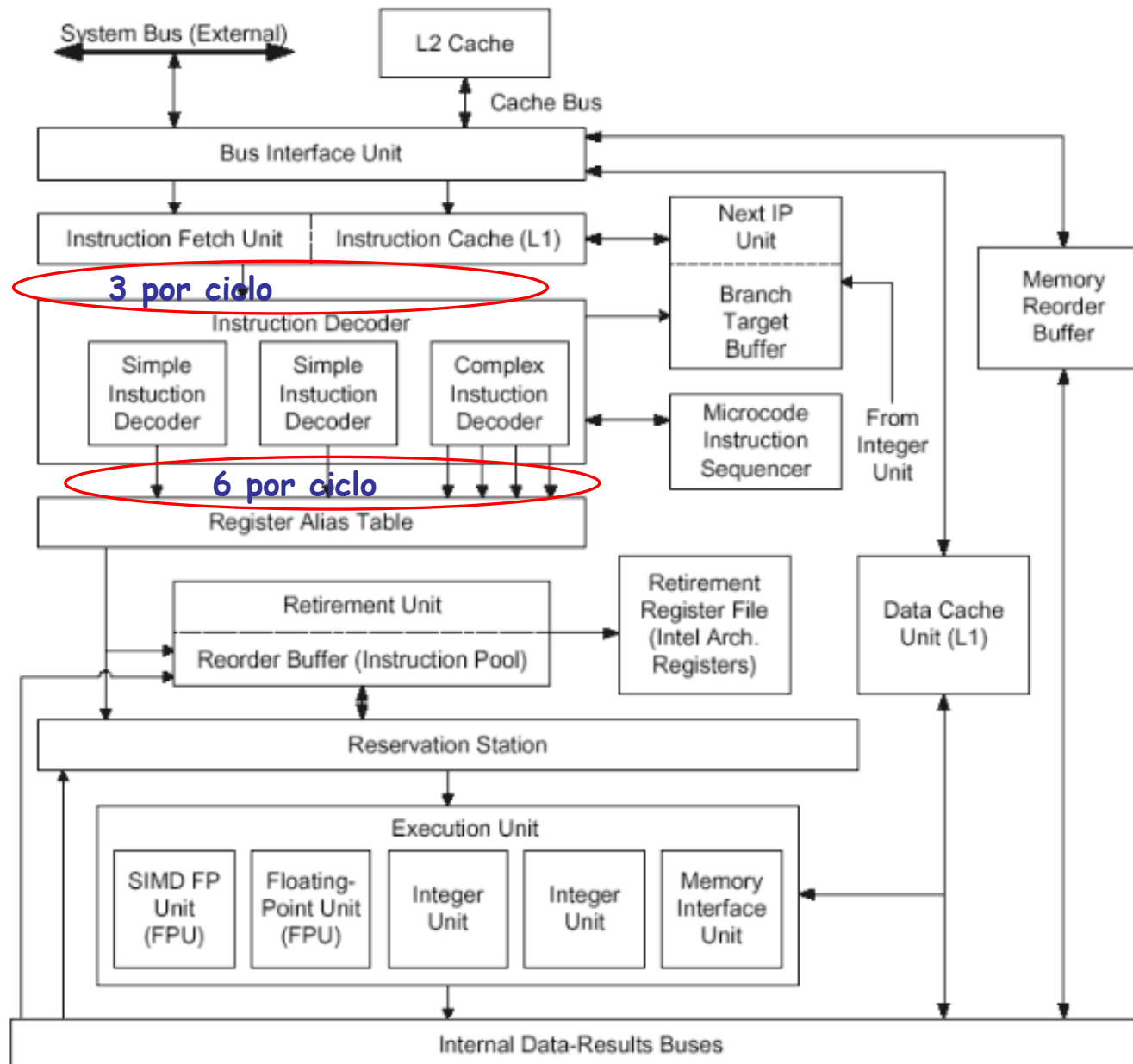


8 etapas para fetch, decodificación y issue en orden

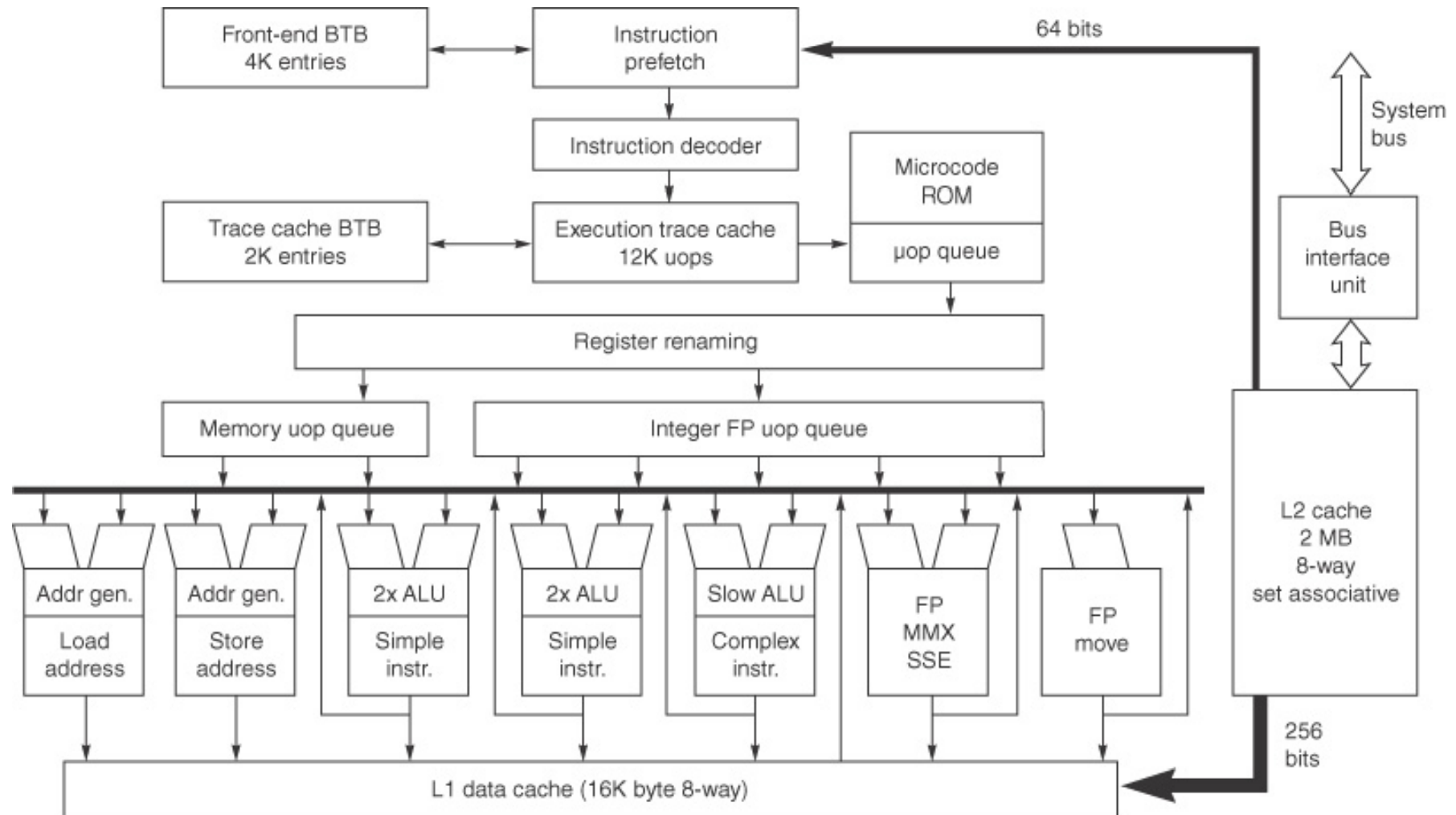
- o 1 ciclo para determinar la longitud de la instrucción 80x86 in + 2 más para generar las microoperaciones

- 3 etapas para ejecución fuera de orden en una de 5 unidades funcionales
- 3 etapas para la finalización de la instrucción (commit)

<u>Parameter</u>	<u>80x86</u>	<u>microops</u>
Max. instructions issued/clock	3	6
Max. instr. complete exec./clock		5
Max. instr. committed/clock		3
Window (Instrs in reorder buffer)	40	
Number of reservations stations	20	
Number of rename registers	40	
No. integer functional units (FUs)	2	
No. floating point FUs	1	
No. SIMD Fl. Pt. FUs	1	
No. memory Fus	1 load + 1 store	



Pentium 4 Microarchitecture



© 2007 Elsevier, Inc. All rights reserved.

- ❑ BTB = Branch Target Buffer (branch predictor)
- ❑ I-TLB = Instruction TLB, Trace Cache = Instruction cache
- ❑ RF = Register File; AGU = Address Generation Unit
- ❑ "Double pumped ALU" means ALU clock rate 2X \Rightarrow 2X ALU F.U.s

❑ Buscar paralelismo de más de un thread

- o Hay mucho paralelismo en algunas aplicaciones (Bases de datos, códigos científicos)
- o Thread Level Parallelism
 - o 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(multiprocesadores, MultiThreading y multi/many cores)
- o Data Level Parallelism: Operaciones idénticas sobre grandes volúmenes de datos (extensiones multimedia y arquitecturas vectoriales)

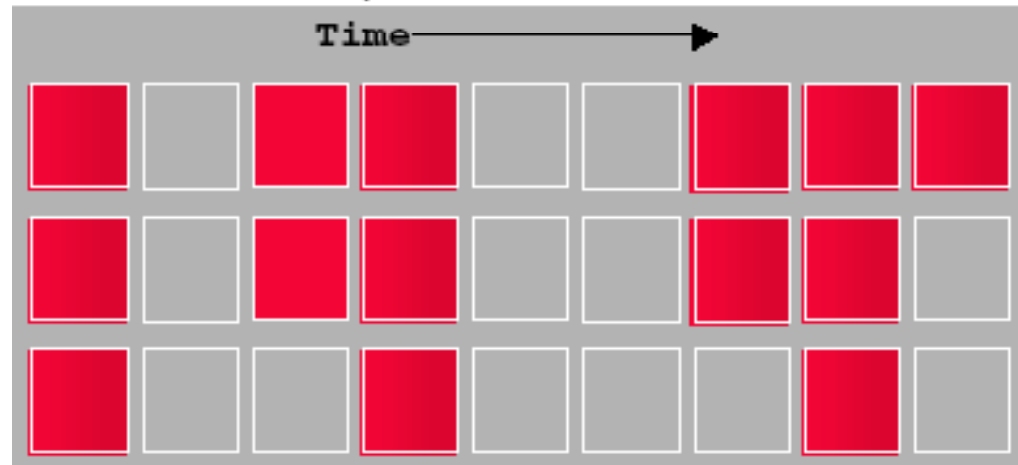
Thread Level Parallelism (TLP) versus ILP

- ❑ 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;
 - o Throughput de computadores que ejecutan muchos programas diferentes.
 - o Reducir el tiempo de ejecución de un programa multi-threaded
- ❑ TLP puede ser más eficaz en coste que ILP

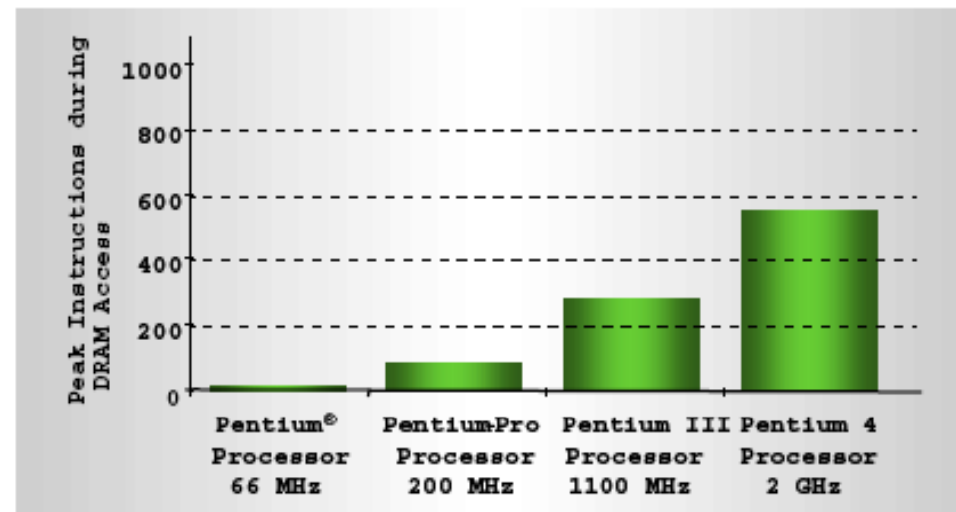
Multithreading

¿ Por que multithreading ?

❑ Procesador superescalar

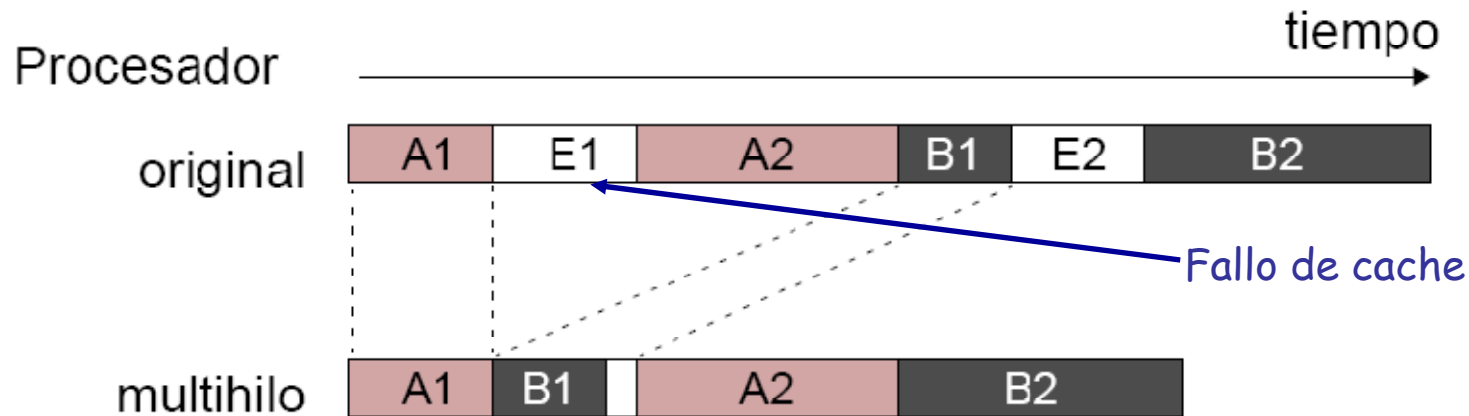


❑ La latencia de memoria crece.
¿ Como soportarla?



Multithreading

□ Multithreading



- 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.

Multithreading

□ Ejecución Multithreaded

- o Multithreading: múltiples threads comparten los recursos del procesador
 - o El procesador debe mantener el estado de cada thread e.g., una copia de bloque de registros, un PC separado, tablas de páginas separadas.
 - o La memoria compartida ya soporta múltiples procesos.
 - o HW para conmutación de thread muy rápido. Mucho mas rápido que entre procesos.
- o ¿Cuándo conmutar?
 - o Cada ciclo conmutar de thread (grano fino)
 - o Cuando un thread debe parar (por ejemplo fallo de cache)
- o HEP (1978), Alewife , M-Machine , Tera-Computer

□ Multithreading de Grano Fino

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

Multithreading

❑ Multithreading Grano Grueso

- o Conmuta entre threads solo en caso de largos stalls, como fallos de cache L2
- o Ventajas
 - o No necesita conmutación entre thread muy rápida.
 - o No retarda cada thread, la conmutación solo se produce cuando un thread no puede avanzar.
- o Desventajas; no elimina pérdidas por stalls cortos. La conmutación es costosa en ciclos.
 - o Como CPU lanza instrucciones de un nuevo thread, el pipeline debe ser vaciado.
 - o El nuevo thread debe llenar el pipe antes de que las instrucciones empiecen a completarse.
- o Ejemplos; IBM AS/400, Montecito (Itanium 9000), Spacr64 VI

Multithreading

❑ Simultaneous Multi-threading

Motivación: Recursos no usados en un procesador superescalar

Un thread, 8 unidades

Ciclo M M FX FX FP FP BR CC

1								
2								
3								
4								
5								
6								
7								
8								
9								

Dos threads, 8 unidades

Ciclo M M FX FX FP FP BR CC

1								
2								
3								
4								
5								
6								
7								
8								
9								

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

Multithreading

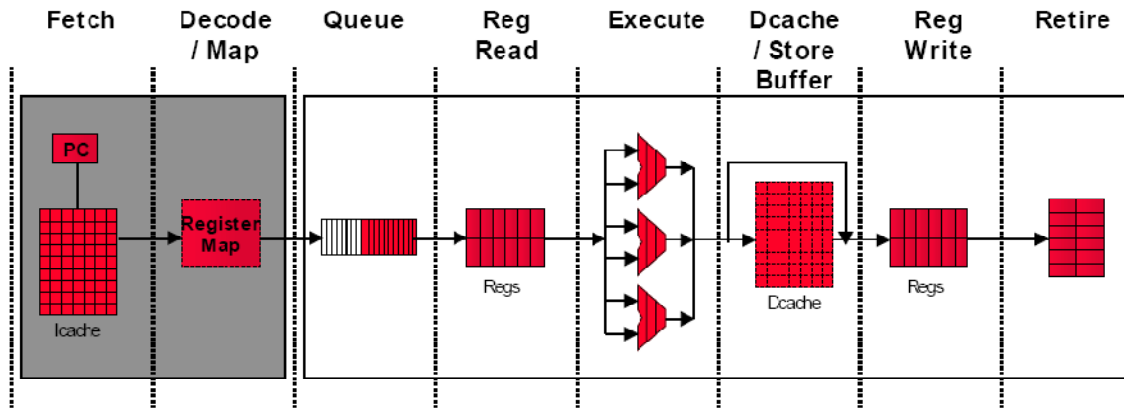
❑ Simultaneous Multithreading (SMT)

- ❑ Simultaneous multithreading (SMT): dentro de un procesador superescalar fuera de orden ya hay mecanismos Hw para soportar la ejecución de más de un thread
 - o Gran numero de registros físicos donde poder mapear los registros arquitectónicos de los diferentes threads
 - o 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
 - o La ejecución fuera de orden permite una utilización eficaz de los recursos.
- ❑ Solo necesitamos sumar una tabla de renombrado por thread y PC separados
 - o Commit independiente se soporta con un ROB por thread (Lógico o físico)
- ❑ Ojo conflictos en la jerarquía de memoria
- ❑ Ejemplos; Pentium4, Power5 y 6, Nehalem (2008)

Multithreading

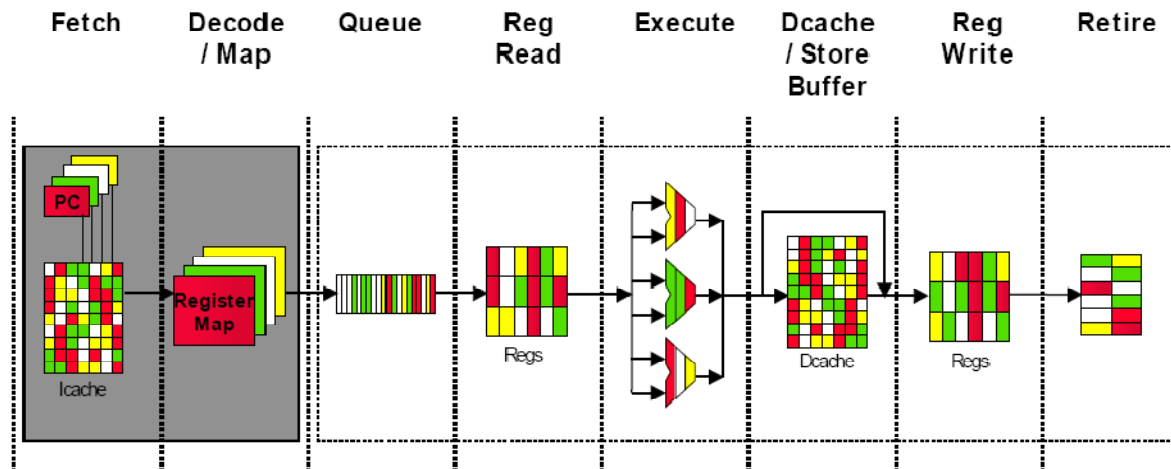
❑ Simultaneous multithreading

❑ Un solo hilo: un flujo de instrucciones



✓ todos los recursos utilizados por un hilo

❑ Multihilo: varios flujos de instrucciones

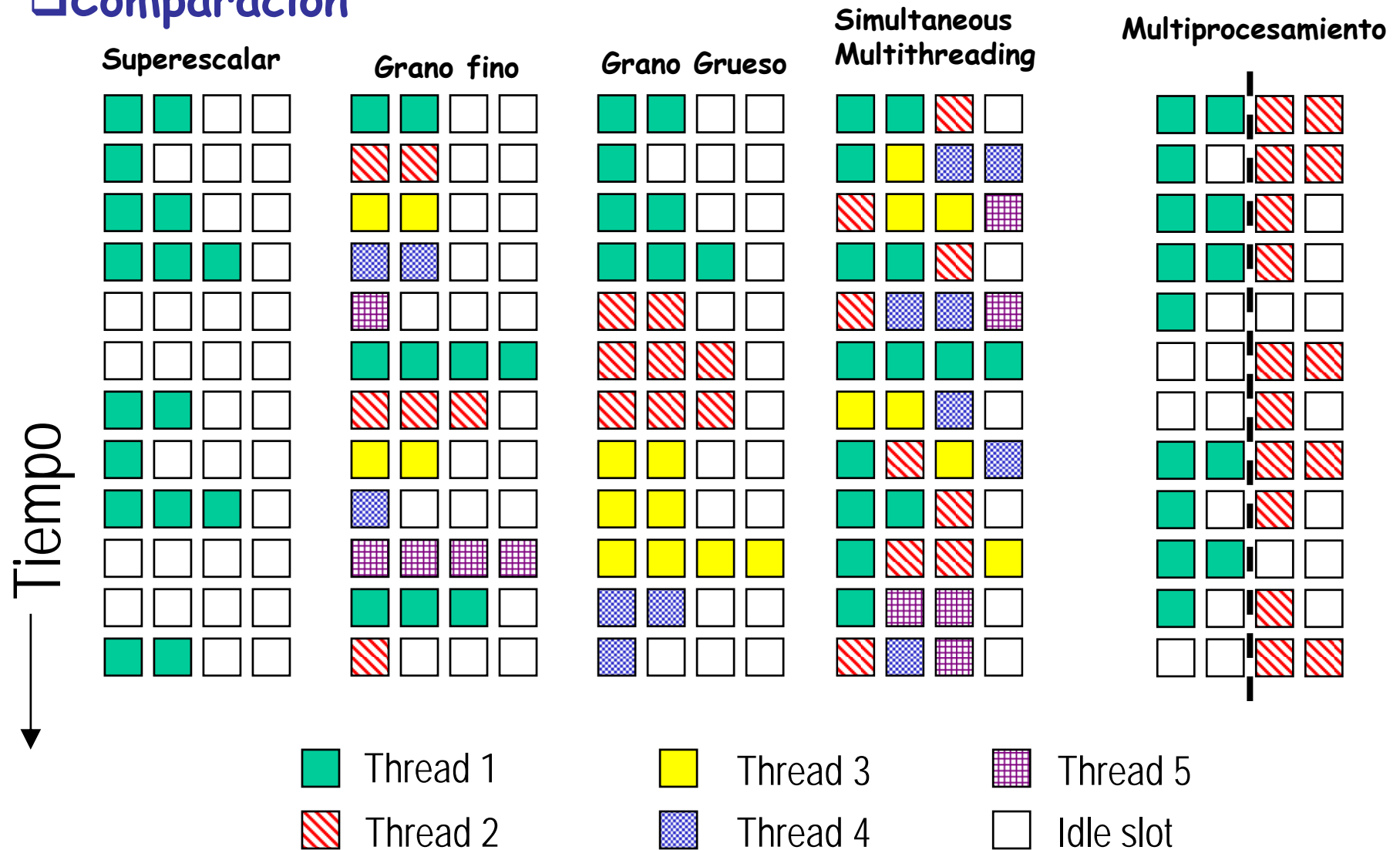


✓ recursos para distinguir el estado de los hilos

✓ los otros recursos se pueden compartir

Multithreading

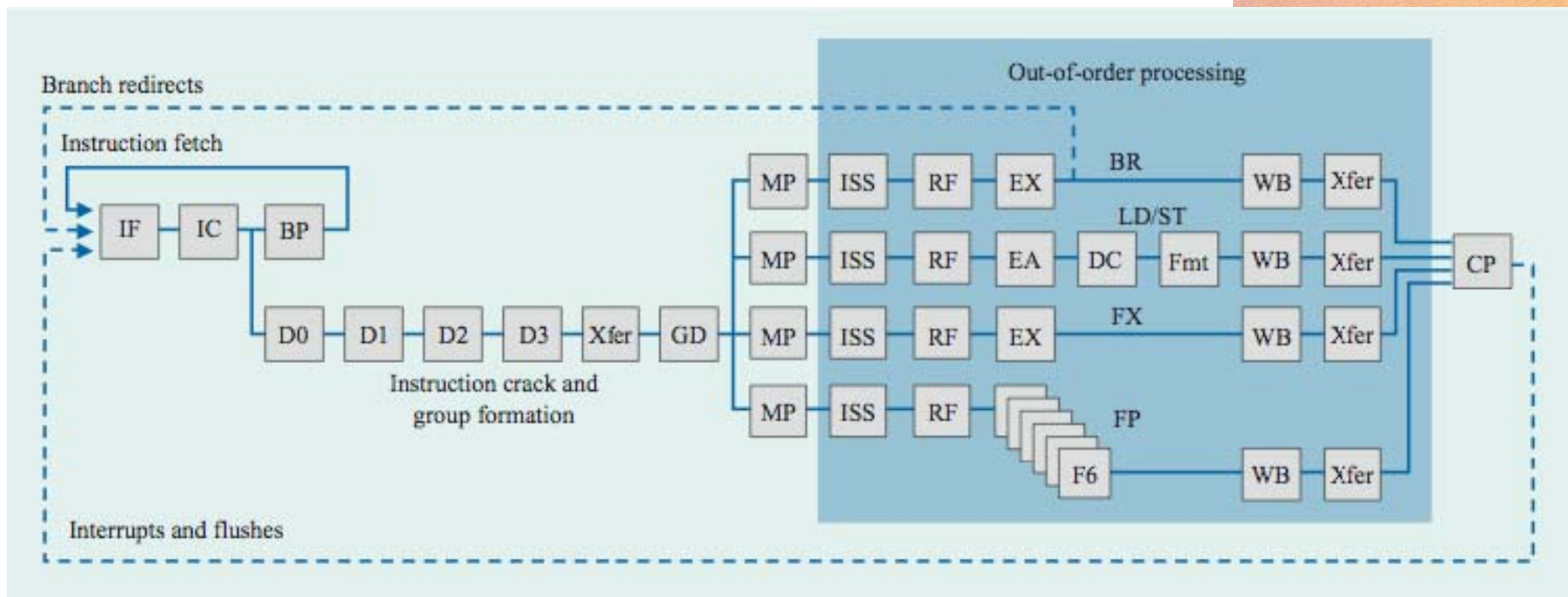
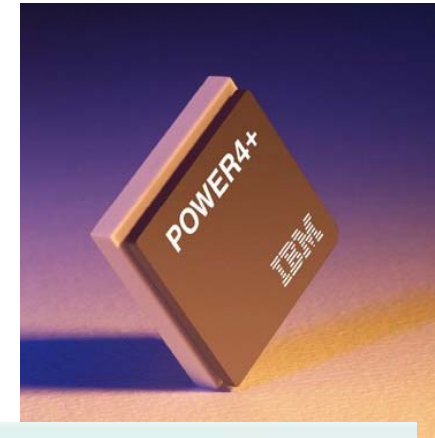
□ Comparación



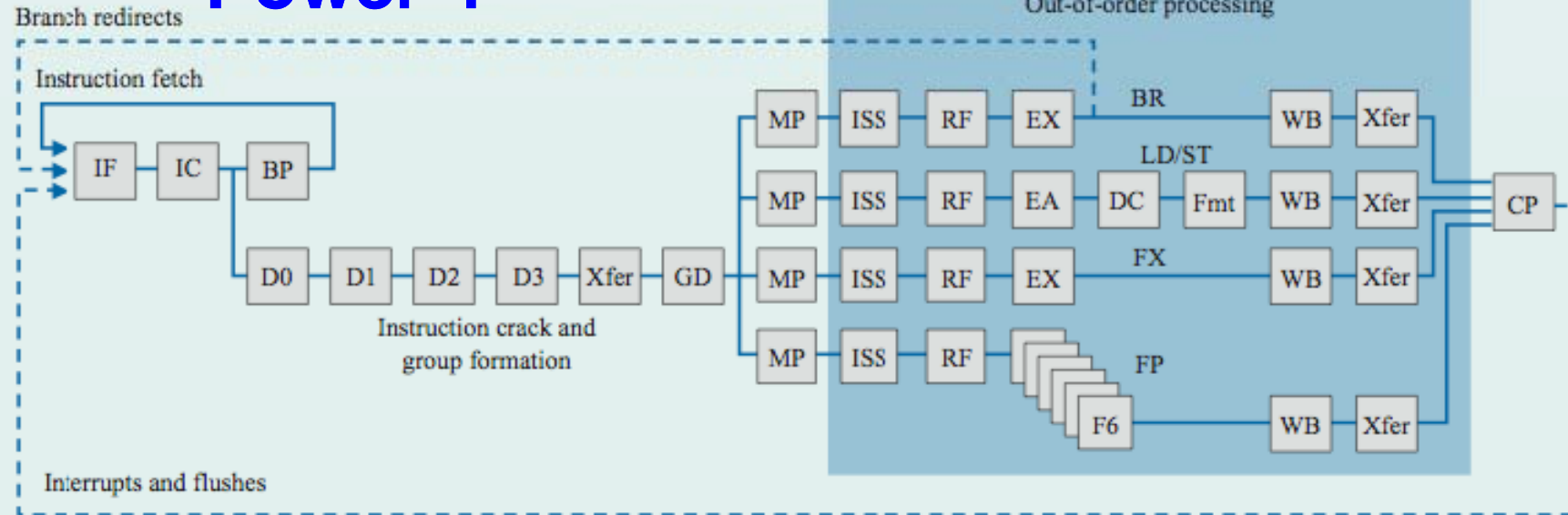
Multithreading

❑ Power 4 (IBM 2000)

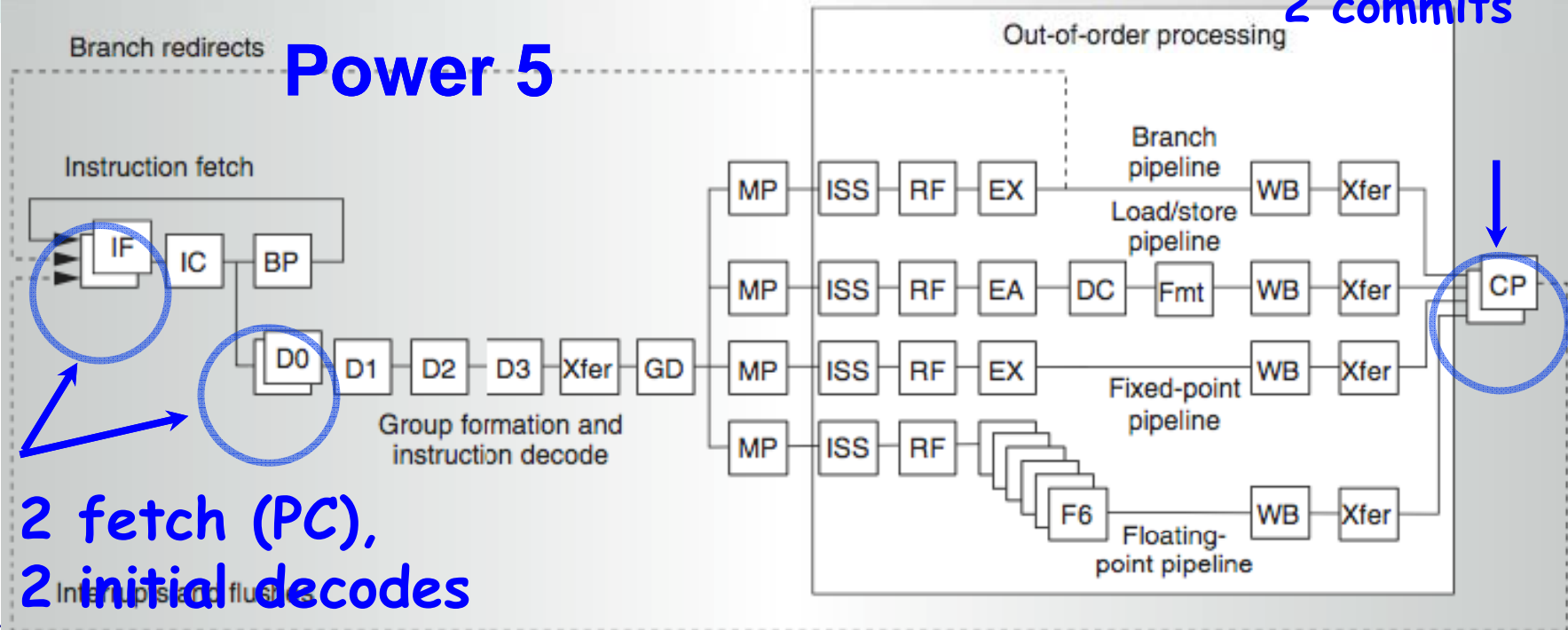
Predecesor Single-threaded del Power 5.
8 unidades de ejecución fuera de orden



Power 4

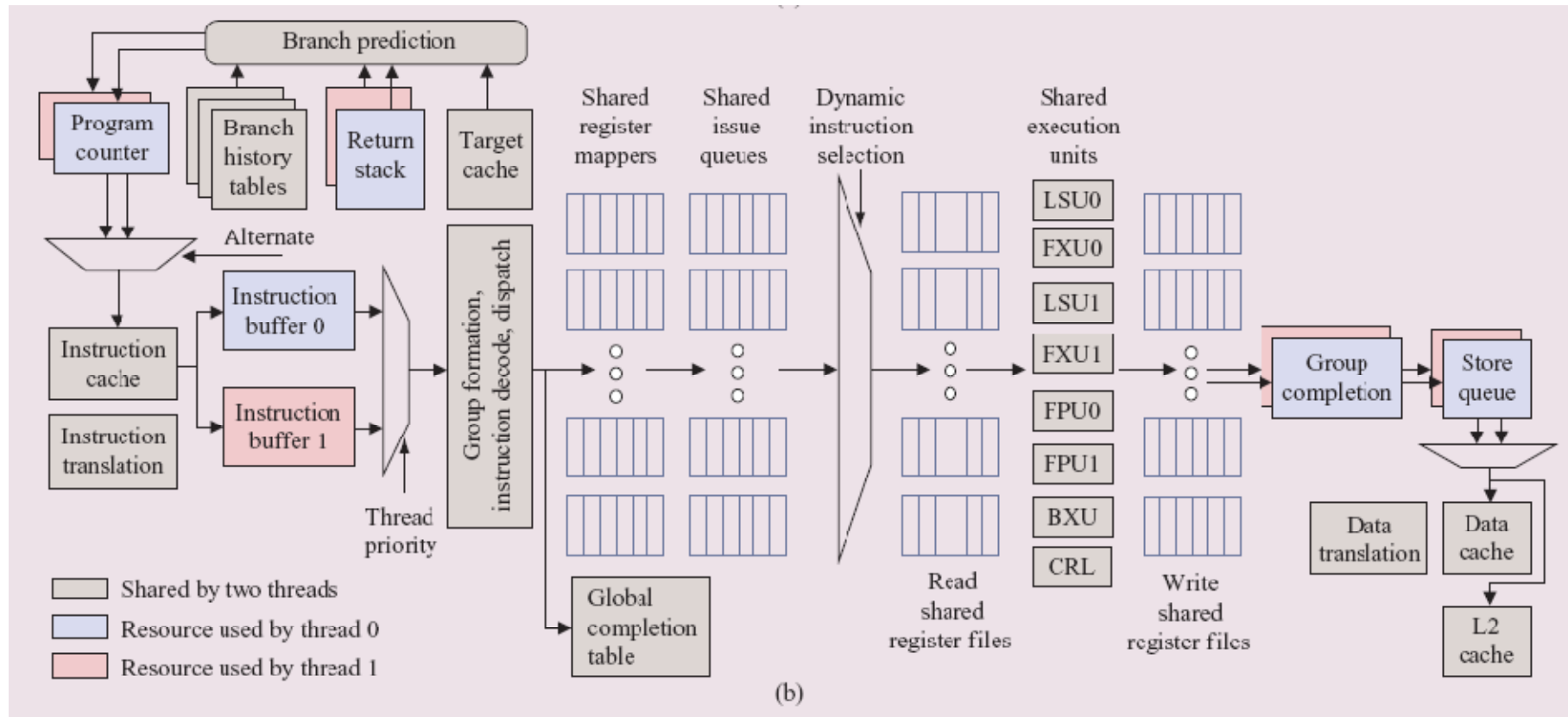


Power 5



Multithreading

❑ Power 5 (IBM 2005)



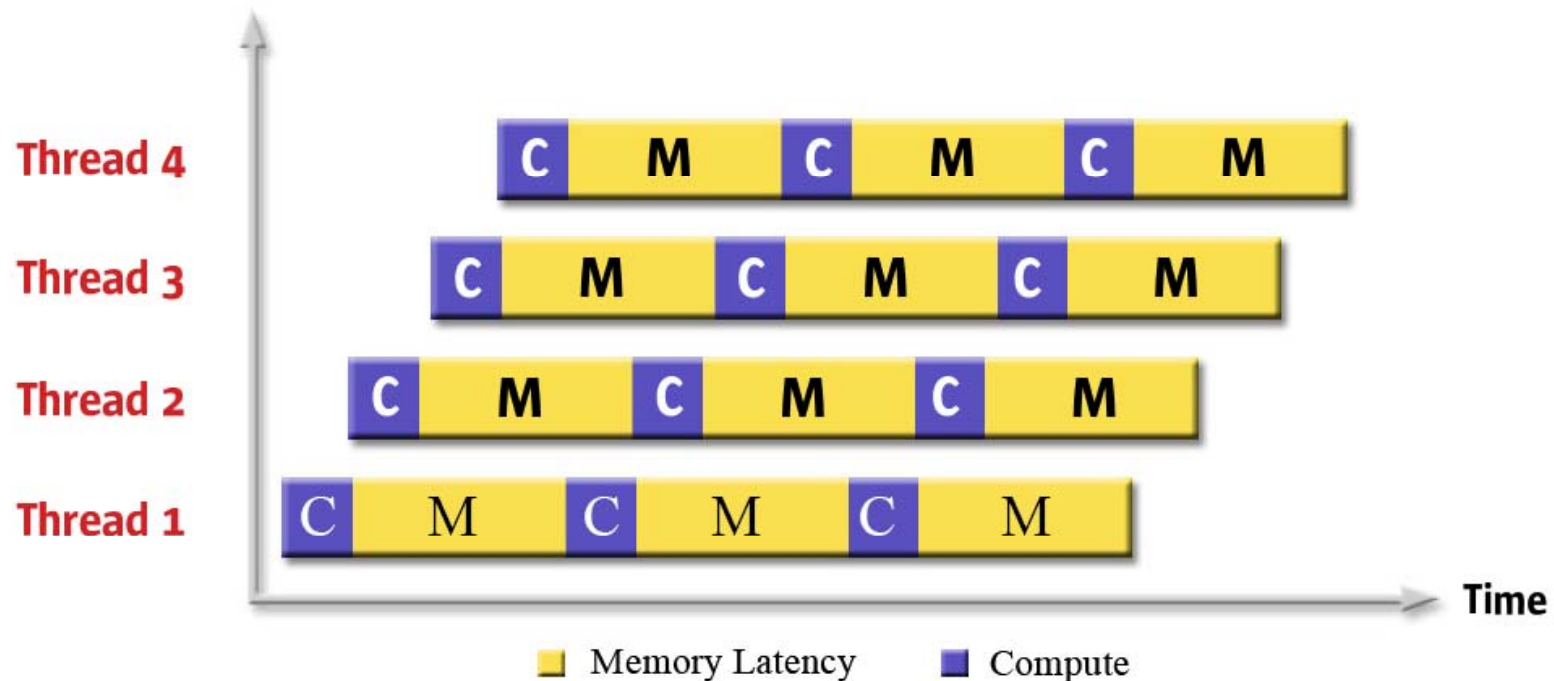
¿Por qué sólo 2 threads? Con 4, los recursos compartidos (registros físicos , cache, AB a memoria) son un cuello de botella.

- ❑ 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 prebusqueda 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

Multiprocesador en un Chip+ Multithreading grano fino

❑ Niagara (SUN 2005)

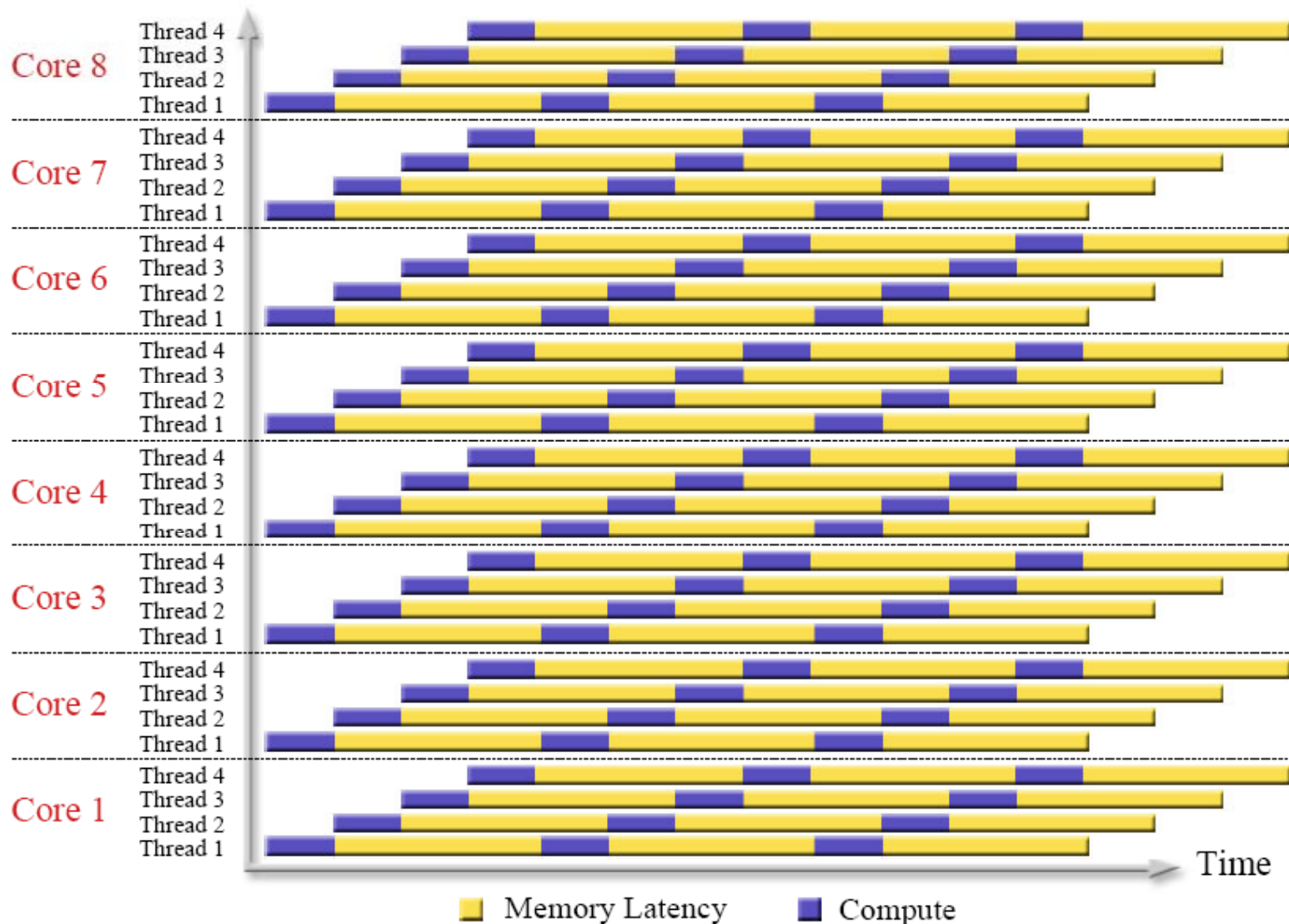
- ❑ Tolerar (soportar) la latencia de memoria mediante hilos concurrentes



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

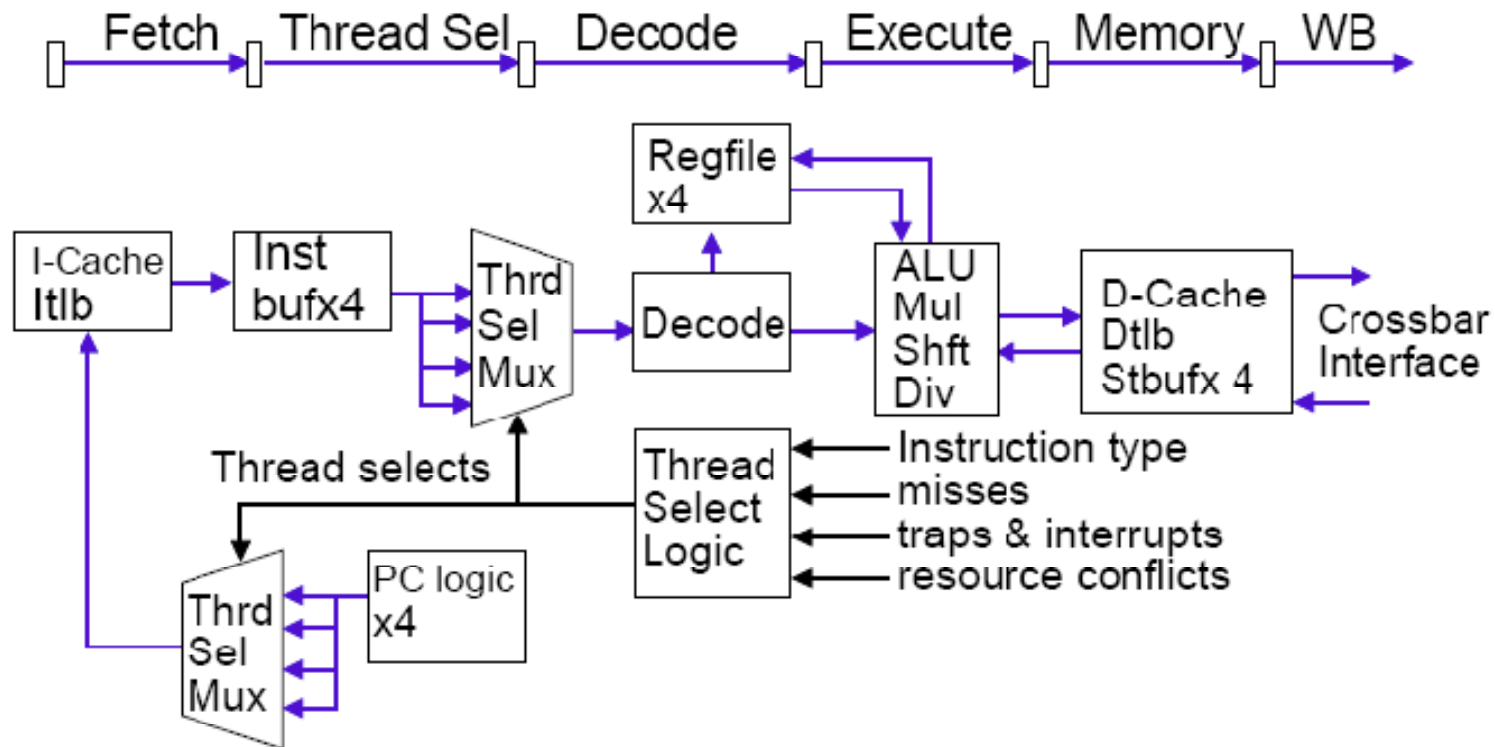
Multiprocesador en un Chip+ Multithreading grano fino

❑ Niagara: Múltiples cores-múltiples thread



Multiprocesador en un Chip+ Multithreading

❑ Niagara UltraSparcT1

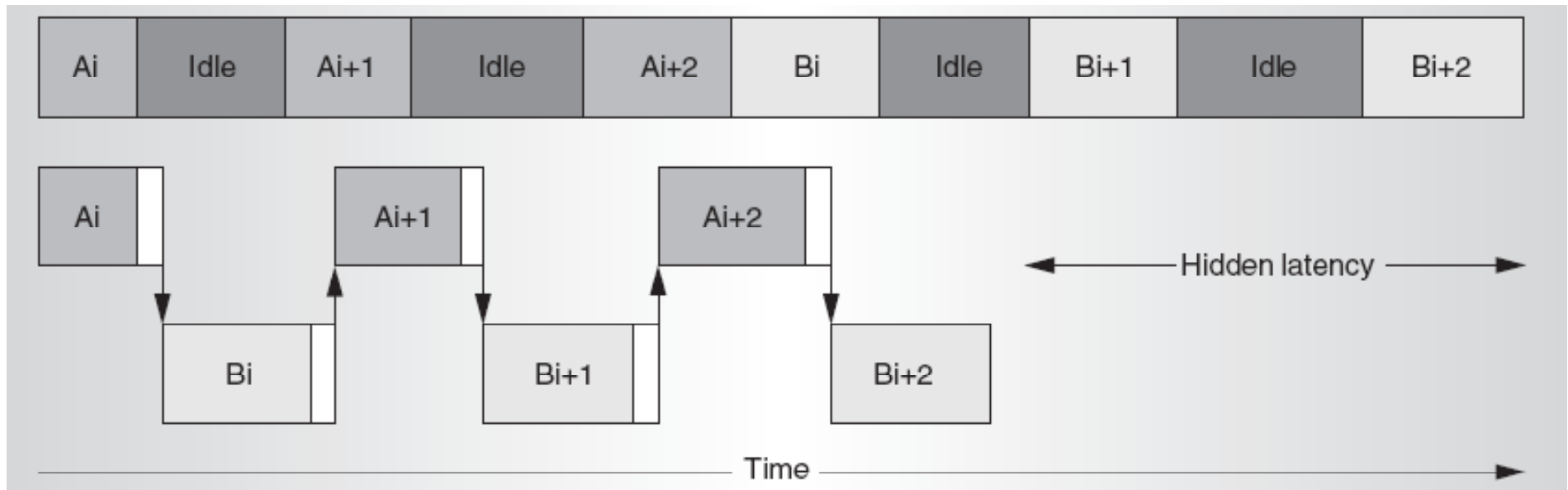


- 6 etapas
- 4 thread independientes
- algunos recursos x4: Banco de registros, contador de programa, store buffer, buffer de instrucciones
- el control de selección de hilo determina el hilo en cada ciclo de reloj
 - ✓ cada ciclo elige un hilo

VLIW EPIC-IA 64 + Multithreading grano grueso

❑ Itanium2 9000 Multi-Threading

- Dinámicamente asigna recursos en función del el uso efectivo a realizar.
 - Un evento de alta latencia determina la cesión de recursos por parte del thread activo.



Rendimiento

¿ Quien es mejor?

Procesador	Microarquitectura	Fetch / Issue / Execute	FU	Clock Rate (GHz)	Transis- -tores Die size	Power
Intel Pentium 4 Extreme	Especulativo con planificación dinámica; Pipe profundo; SMT	3/3/4	7 int. 1 FP	3.8	125 M 122 mm ²	115 W
AMD Athlon 64 FX-57	Especulativo con planificación dinámica.	3/3/4	6 int. 3 FP	2.8	114 M 115 mm ²	104 W
IBM Power5 (1 CPU only)	Especulativo con planificación dinámica; SMT 2 CPU cores/chip	8/4/8	6 int. 2 FP	1.9	200 M 300 mm ² (est.)	80W (est.)
Intel Itanium 2	Planificación estática VLIW-style	6/5/11	9 int. 2 FP	1.6	592 M 423 mm ²	130 W

Rendimiento Conclusiones

- ❑ No hay un claro ganador en todos los aspectos
- ❑ El AMD Athlon gana en SPECInt seguido por el Pentium4, Itanium 2, y Power5
- ❑ Itanium 2 y Power5, tienen similares rendimientos en SPECFP, dominan claramente al Athlon y Pentium 4
- ❑ Itanium 2 es el procesador menos **eficiente** en todas las medidas menos en SPECFP/Watt.
- ❑ Athlon y Pentium 4 usan bien los transistores y el área en términos de eficacia
- ❑ IBM Power5 es el mas eficaz en el uso de la energía sobre los SPECFP y algo menos sobre SPECINT

Conclusiones -Limites del ILP

- ❑ Doblar en ancho de emisión (issue rates) sobre los valores actuales 3-6 instrucciones por ciclo, a digamos 6 a 12 instrucciones requiere en el procesador
 - o de 3 a 4 accesos a cache de datos por ciclo,
 - o Predecir-resolver de 2 a 3 saltos por ciclo,
 - o Renombrar y acceder a mas de 20 registros por ciclo,
 - o 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 mas rendimiento que el consumo.
- ❑ Todas las técnicas de emisión múltiple son poco eficientes desde el punto de vista de la energía.

Conclusiones -Limites del ILP

- ❑ La arquitectura Itanium **no** representa un paso adelante en el incremento el ILP, eliminando los problemas de complejidad y consumo.
- ❑ 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, Fujitsu,...).
- ❑ El balance entre ILP y TLP a nivel de chip no es todavía claro, y probablemente será muy dependiente del tipo de aplicaciones a que se dedique el sistema.