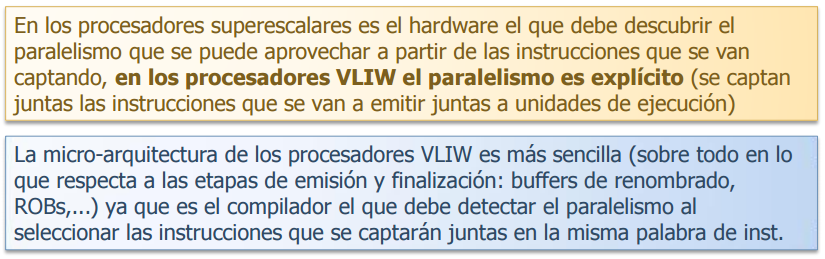
**Lección 11. Microarquitecturas ILP. Cauces Superescalares**

**|Procesadores Superescalares y VLIW |**

Los procesadores Superescalares y VLIW comparten las siguientes características (además del hecho de ser procesadores segmentados):

* Disponen de varias unidades de ejecución
* Pueden ejecutar varias instrucciones simultáneamente en esas unidades de ejecución
* Pueden emitir múltiples instrucciones en paralelo a unidades de ejecución

Pero:



**|Mejora de las Prestaciones de los Procesadores|**

Mejora de la Tecnología de Fabricación de CI basada en el Reducción del→

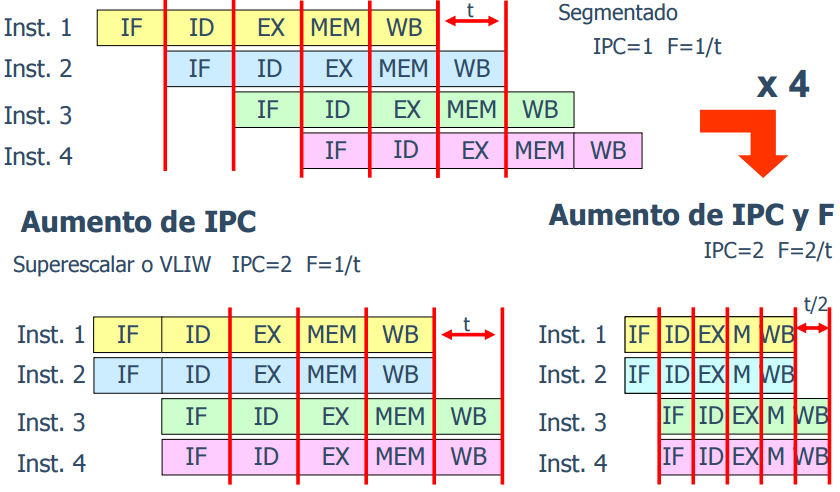
* Silicio tamaño de los transistores + Aumento del tamaño del dado

Esto desemboca en:

* Más transistores por circuito integrado Microarquitecturas más complejas en un solo CI: Paralelismo entre Instrucciones (Procesadores Superescalares)
* Se reduce la longitud de puerta del transistor y con ello Mayores frecuencias de el tiempo de conmutación funcionamiento

Como resultado, si tenemos mas instrucciones IPC y mayores Frecuencias, tenemos mayor velocidad de CPU

**Vcpu=IPC\*F**



**|Paralelismo entre instrucciones (ILP) y paralelismo de la máquina|**

Paralelismo entre Instrucciones

Depende de la frecuencia de las dependencias de datos y control, y del retardo de la operación (tiempo hasta que el resultado de una operación esté disponible).



Paralelismo de la Máquina

Determinado por el número de instrucciones que pueden captarse y ejecutarse al mismo tiempo (número de cauces paralelos) y por la velocidad y los mecanismos que usa el procesador para encontrar las dependencias entre instrucciones

**|Ordenaciones en una secuencia de instrucciones|**

En una secuencia de instrucciones se pueden distinguir tres tipos de ordenaciones:

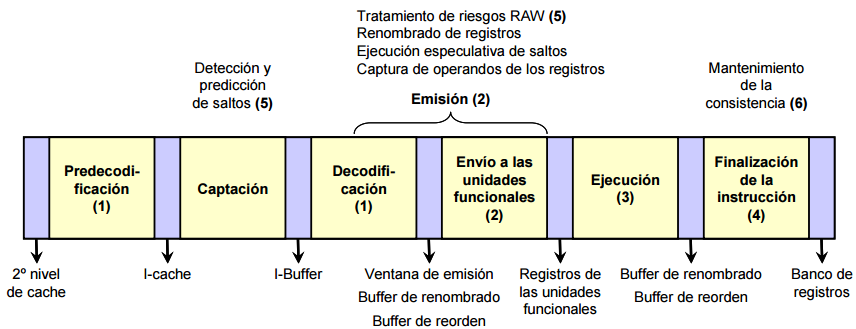
* El orden en que se captan las instrucciones (el orden de las instrucciones en el código)
* El orden en que las instrucciones se ejecutan
* El orden en que las instrucciones cambian los registros y la memoria.

El procesador superescalar debe ser capaz de identificar el paralelismo entre instrucciones (ILP) que exista en el programa y organizar la captación, decodificación y ejecución de instrucciones en paralelo, utilizando eficazmente los recursos existentes (el paralelismo de la máquina).

Cuanto más sofisticado sea un procesador superescalar, menos tiene que ajustarse a la ordenación de las instrucciones según se captan, para la ejecución y modificación de los registros, de cara a mejorar los tiempos de ejecución. La única restricción es que el resultado del programa sea correcto.

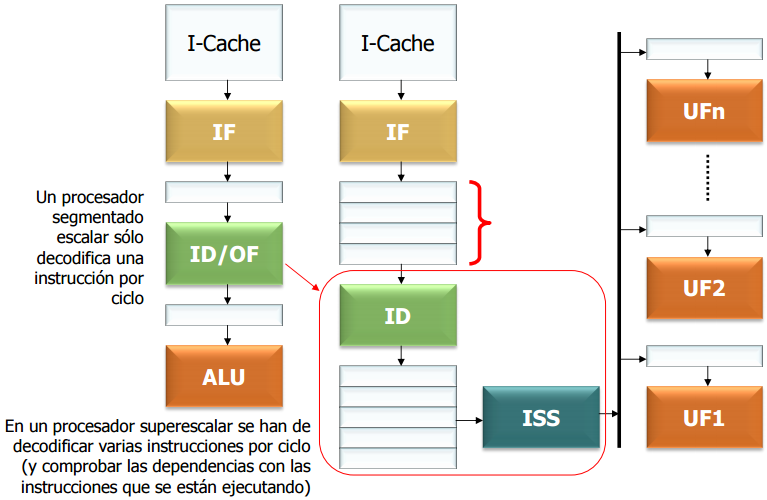
**|Aspectos del Procesamiento Superescalar y Etapas del Cauce|**

1. Predecodificación)◊Decodificación Paralela (Decodificación a mayor velocidad
2. Emisión Paralela de Instrucciones a las Unidades Funcionales (Dependencias)
3. Ejecución Paralela en las distintas Unidades Funcionales, UF, (segmentadas).
4. Finalización del Procesamiento de la Instrucción
5. Detección y predicción de saltos
6. Mantenimiento de la consistencia secuencial (desacoplar la ejecución y la escritura de resultados)

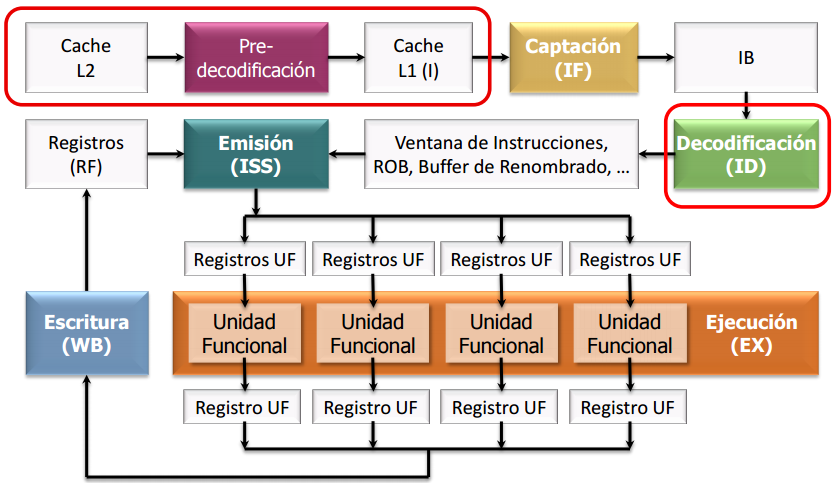


**|Etapas de un procesador Superescalar|**

Captación y Decodificación Paralela



Predecodificación



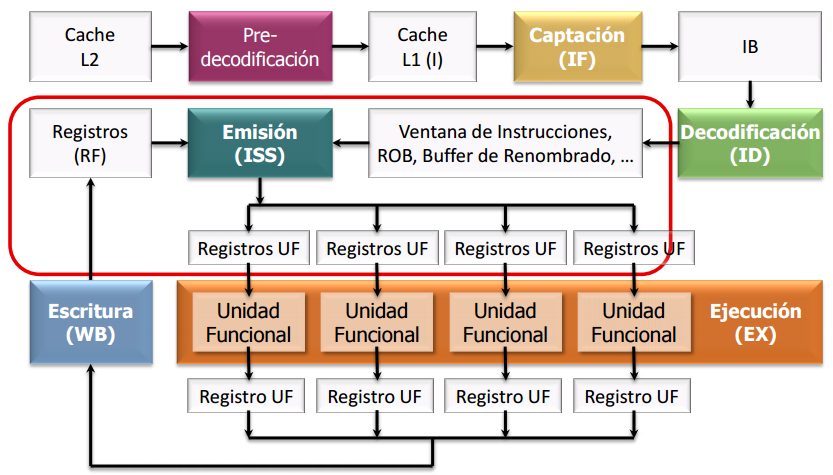
Los bits que se añaden en la etapa de predecodificación (bits de predecodificación) suelen indicar:

* Si es una instrucción de salto o no (se puede empezar su procesamiento antes)
* El tipo de unidad funcional que va a utilizar (se puede emitir más rápidamente si hay cauces para enteros o coma flotante…)
* Si hace referencia a memoria o no

Ejemplos de uso de los bits de predecodificación:

* HP-PA 7200: Se añaden bits por cada par de instrucciones para facilitar la comprobación de si las instrucciones se pueden ejecutar en paralelo en el cauce o hay algún tipo de dependencias (10% de SRAM).
* AMD K6: La lógica de decodificación indica la longitud de la instrucción x86 en bytes permitiendo la localización del final de cada instrucción y se almacena en la cache L1 junto con las instrucciones. En una cache de 64 Kbytes se añaden 20 Kbytes para los bits de decodificación.

Emisión Paralela de Instrucciones



La ventana de instrucciones almacena las instrucciones pendientes

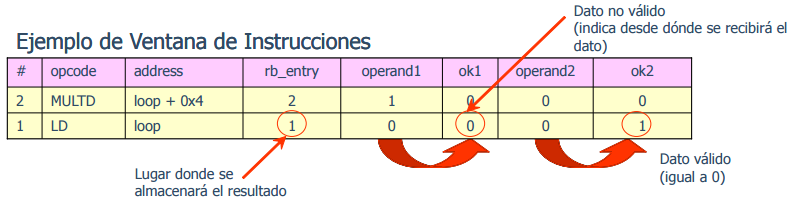
* todas, si la ventana es centralizada, o las de un tipo determinado, si es distribuida

Las instrucciones se cargan en la ventana una vez decodificadas

* Se utiliza un bit para indicar si un operando está disponible (se almacena el valor o se indica el registro desde donde se lee) o no (se almacena la unidad funcional desde donde llegará el operando)

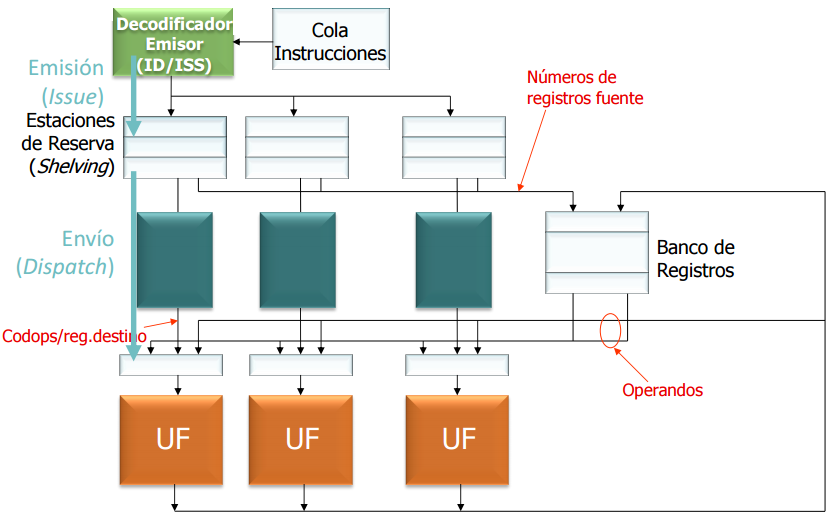
Una instrucción puede ser emitida cuando tiene todos sus operandos disponibles y la unidad funcional donde se procesará.

* Hay diversas posibilidades para el caso en el que varias instrucciones estén disponibles (características de los buses, etc.)



\*Paginas 25,26,27 y 28 de emisión ordenada y desordenada

Estaciones de Reserva



Si no existen limitaciones en el hardware las instrucciones se emiten a las estaciones de reserva (buffers de shelving o consignas) independientemente de las dependencias

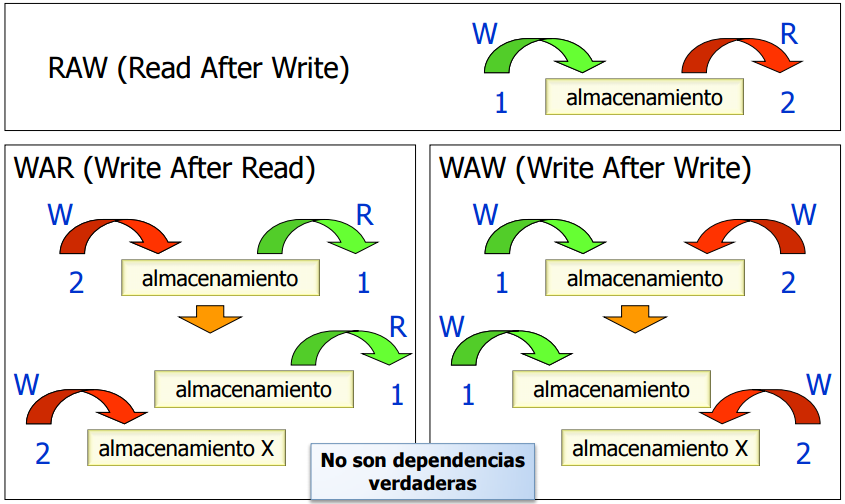
Las instrucciones esperan hasta que se resuelvan las dependencias y se envían a las unidades funcionales una vez comprobada la disponibilidad de la unidad

\*Ejemplos Pagina 31 y 33-40

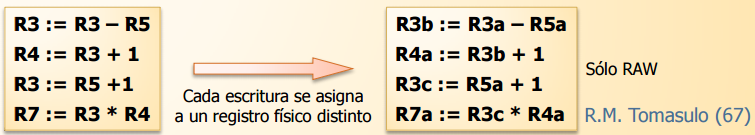
Alternativas para el Envío a las Unidades Funcionales:

* Reglas de Selección: Se determina las instrucciones que pueden enviarse
  + Las instrucciones ejecutables
* Reglas de Arbitraje: Instrucción que se envía si hay varias ejecutables
  + La más antigua entre las ejecutables
* Orden de Envío: Ordenadas, Desordenadas, o Parcialmente ordenadas (ciertas instrucciones no ejecutables bloquean instrucciones de un tipo, pero no de otros)
* Velocidad de Envío: Número de instrucciones que se envían por ciclo Una por ciclo o Varias por Ciclo

**|Renombramiento de registros |**



Técnica para evitar el efecto de las dependencias WAR, o Antidependencias (en la emisión desordenada) y WAW, o Dependencias de Salida (en la ejecución desordenada).



* Implementación Estática: Durante la Compilación
* Implementación Dinámica: Durante la Ejecución (circuitería adicional y registros extra)

Características de los Buffers de Renombrado

* Tipos de Buffers (separados o mezclados con los registros de la arquitectura)
* Número de Buffers de Renombrado
* Mecanismos para acceder a los Buffers (asociativos o indexados)

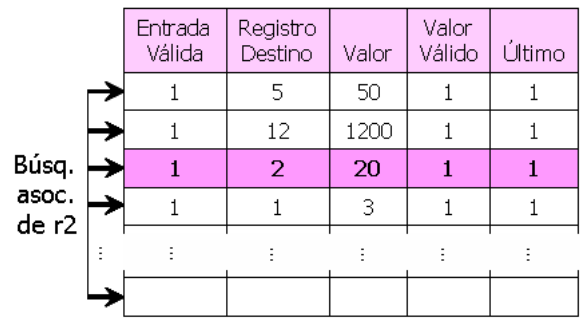
Velocidad del Renombrado

* Máximo número de nombres asignados por ciclo que admite el procesador

Buffers de Renombramiento

Permite varias escrituras pendientes a un mismo registro

El bit de último se utiliza para marcar cual ha sido la más reciente



\* Ejemplo y Tomasulo 45-58

**Lección 12.** **Consistencia del procesador y Procesamiento de Saltos**

**|** **Consistencia. Reordenamiento |**

Consistencia

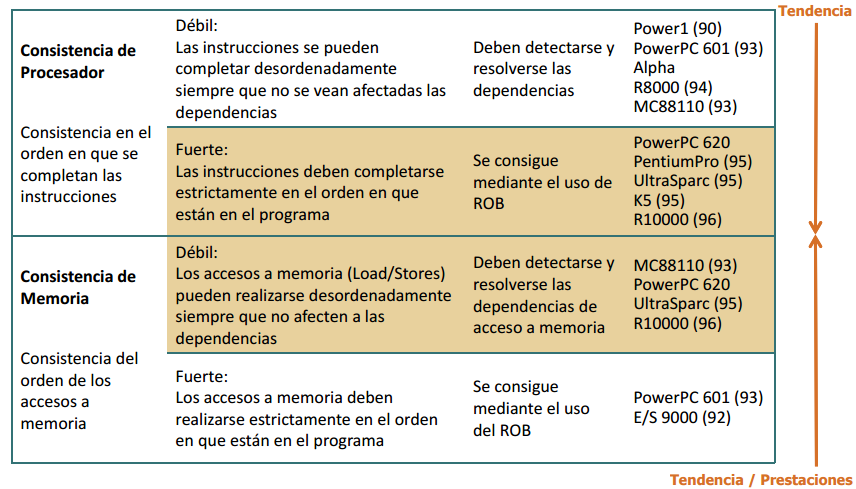
En el Procesamiento de una Instrucción se puede distinguir entre:

* El Final de la Ejecución de la Operación codificada en las instrucciones (Se dispone de los resultados generados por las UF pero no se han modificado los registros de la arquitectura).
* El Final del Procesamiento de la Instrucción o momento en que se Completa la Instrucción (Complete o Commit) (Se escriben los resultados de la Operación en los Registros de la Arquitectura. Si se utiliza un Buffer de Reorden, ROB, se utiliza el término Retirar la Instrucción, Retire, en lugar de Completar)

La Consistencia de un Programa se refiere a:

* El orden en que las instrucciones se completan
* El orden en que se accede a memoria para leer (LOAD) o escribir (STORE)

Cuando se ejecutan instrucciones en paralelo, el orden en que termina (finish) esa ejecución puede variar según el orden que las correspondientes instrucciones tenían en el programa pero debe existir consistencia entre el orden en que se completan las instrucciones y el orden secuencial que tienen en el código de programa.



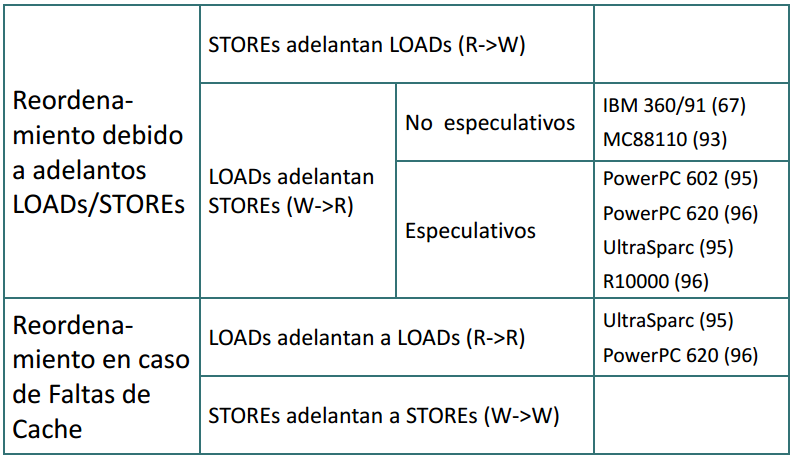
Reordenamiento

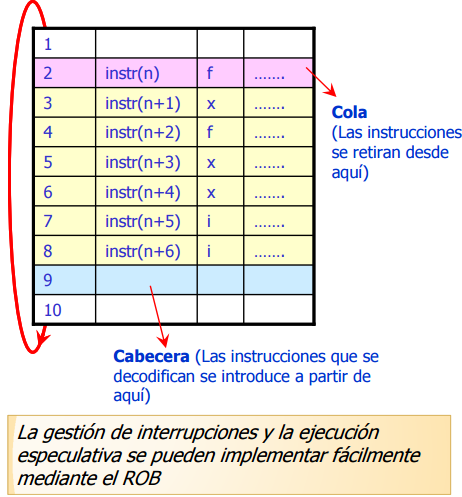
Las instrucciones LOAD y STORE implican cambios en el Procesador y en Memoria

* LOAD: - Cálculo de Dirección en ALU o Unidad de Direcciones - Acceso a Cache - Escritura del Dato en Registro
* STORE: - Cálculo de Dirección en ALU o Unidad de Direcciones - Esperar que esté disponible el dato a almacenar (en ese momento acaba)

La Consistencia de Memoria Débil (reordenación de los accesos a memoria):

* ‘Bypass’ de Loads/Stores: Los Loads pueden adelantarse a los Stores pendientes y viceversa (siempre que no se violen dependencias)
* Permite los Loads y Stores Especulativos: Cuando un Load se adelanta a un Store que le precede antes de que se haya determinado la dirección se habla de Load especulativo. Igual para un Store que se adelanta a un Load o a un Store.
* Permite ocultar las Faltas de Cache: Si se adelanta un acceso a memoria a otro que dio lugar a una falta de cache y accede a Memoria Principal.





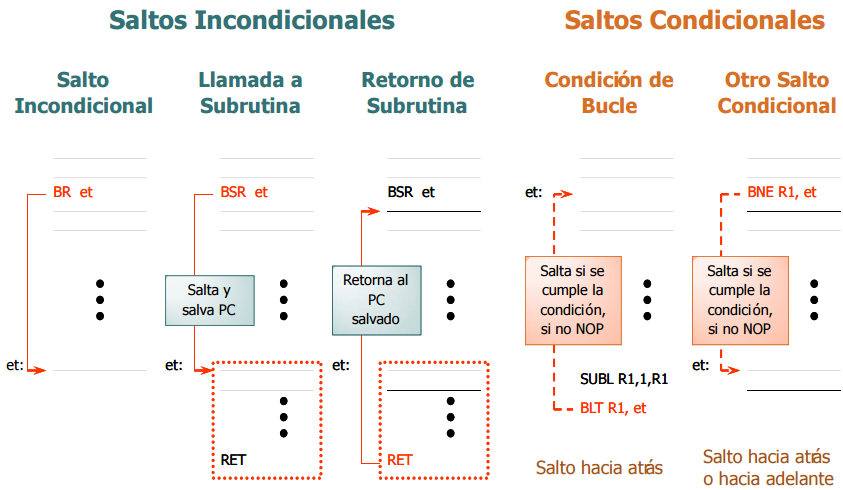
Buffer de Reordenamiento (ROB)

* El puntero de cabecera apunta a la siguiente posición libre y el puntero de cola a la siguiente instrucción a retirar.
* Las instrucciones se introducen en el ROB en orden de programa estricto y pueden estar marcadas como emitidas (issued, i), en ejecución (x), o finalizada su ejecución (f)
* Las instrucciones sólo se pueden retirar (se produce la finalización con la escritura en los registros de la arquitectura) si han finalizado, y todas las que les preceden también.
* La consistencia se mantiene porque sólo las instrucciones que se retiran del ROB se completan (escriben en los registros de la arquitectura) y se retiran en el orden estricto de programa

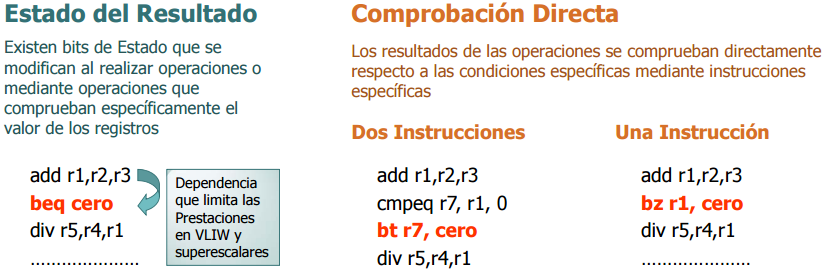
\*Ejemplos 12-15

**|Procesamiento especulativo de saltos|**

Clasificación de los Saltos



Alternativas para la condición de saltos:



Aspectos del Procesamiento de Saltos en un procesador Superescalar

Detección de la Instrucción de Salto

* Cuanto antes se detecte que una instrucción es de salto menor será la posible penalización. Los saltos se detectan usualmente en la fase de decodificación e incluso en la captación (si hay predecodificación) .

Gestión de los Saltos Condicionales no Resueltos

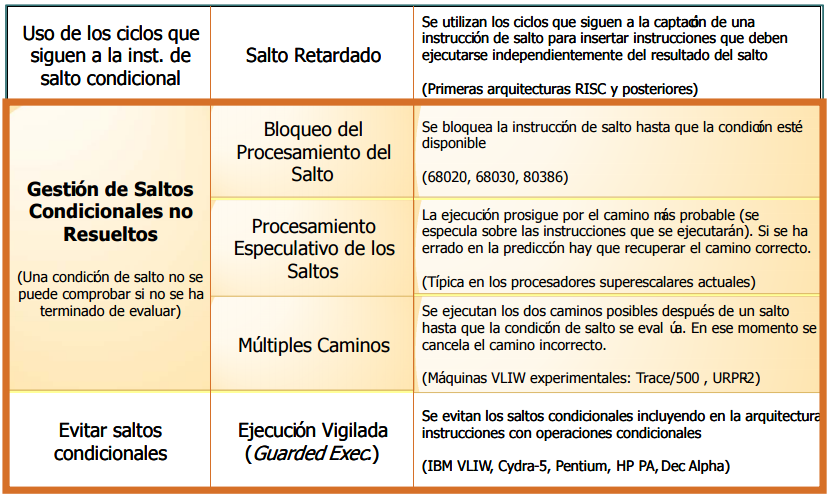
* Si en el momento en que la instrucción de salto evalúa la condición de salto ésta no se haya disponible se dice que el salto o la condición no se ha resuelto. Para resolver este problema se suele utilizar el procesamiento especulativo del salto.

Acceso a las Instrucciones destino del Salto

* Hay que determinar la forma de acceder a la secuencia a la que se produce el salto

El efecto de los saltos en los procesadores superescalares es más pernicioso ya que, al emitirse varias instrucciones por ciclo, prácticamente en cada ciclo puede haber una instrucción de salto.

Gestión de Saltos Condicionales no resueltos



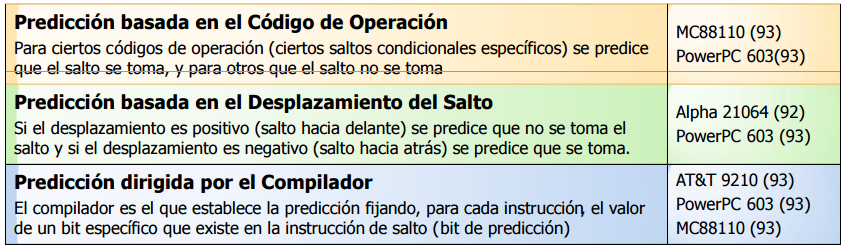
Esquemas de Predicción de Salto

Predicción Fija Se toma siempre la misma decisión: el salto siempre se realiza, ‘taken’, o no, ‘not taken’

Predicción Verdadera (mejores prestaciones)La decisión de si se realiza o no se realiza el salto se toma mediante:

* Predicción Estática: Según los atributos de la instrucción de salto (el código de operación, el desplazamiento, la decisión del compilador)
* Predicción Dinámica: Según el resultado de ejecuciones pasadas de la instrucción (historia de la instrucción de salto)

Predicción Estatica



Predicción Dinamica

La predicción para cada instrucción de salto puede cambiar cada vez que se va a ejecutar ésta según el historial de saltos para dicha instrucción.

El presupuesto básico de la predicción dinámica es que es más probable que el resultado de una instrucción de salto sea similar al que se tuvo en las últimas

Presenta mejores prestaciones de predicción, aunque su implementación es más costosa

Predicción Dinámica Implícita

* No hay bits de historia propiamente dichos sino que se almacena la dirección de la instrucción que se ejecutó después de la instrucción de salto en cuestión

Predicción Dinámica Explícita

* Para cada instrucción de salto existen unos bits específicos que codifican la información de historia de dicha instrucción de salto

Extensión del Procesamiento Especulativo

* Tras la predicción, el procesador continúa ejecutando instrucciones especulativamente hasta que se resuelve la condición.
* El intervalo de tiempo entre el comienzo de la ejecución especulativa y la resolución de la condición puede variar considerablemente y ser bastante largo.
* En los procesadores superescalares, que pueden emitir varias instrucciones por ciclo, pueden aparecer más instrucciones de salto condicional no resueltas durante la ejecución especulativa.
* Si el número de instrucciones que se ejecutan especulativamente es muy elevado y la predicción es incorrecta, la penalización es mayor.

Así, cuanto mejor es el esquema de predicción mayor puede ser el número de instrucciones ejecutadas especulativamente.

* Nivel de Especulación: Número de Instrucciones de Salto Condicional sucesivas que pueden ejecutarse especulativamente (si se permiten varias, hay que guardar varios estados de ejecución)
* Grado de Especulación: Hasta qué etapa se ejecutan las instrucciones que siguen en un camino especulativo después de un salto

Recuperación de Predicción Incorrecta

La recuperación de una predicción incorrecta comprende:

* Descartar los resultados de la ejecución especulativa
* Continuar la ejecución de la secuencia de instrucciones correcta

Recuperación desde un salto efectuado:

* El procesador debe guardar la dirección de la instrucción siguiente a la de salto para utilizarla si la predicción es incorrecta.
* La recuperación es más rápida si no se descartan las instrucciones que se habían pre-captado junto con la de salto

Recuperación cuando no se ha saltado:

* Pre-calcular la dirección de salto y almacenarse para poder recuperarla.
* La recuperación es más rápida si se pre-captan instrucciones de la secuencia que empieza a partir de la dirección a la que se salta.

Acceso a la Secuencia de Salto

Si se detecta una instrucción de salto, se calcula su dirección de destino para acceder a la posición de memoria correspondiente si se produce el salto

Los saltos condicionales efectuados (‘taken’) son más frecuentes que los no efectuados (‘not taken’). Por ello, sería interesante reducir al máximo el tiempo de acceso a la secuencia de instrucciones a partir de la dirección de salto y reducir la penalización para las predicciones incorrectas de los saltos efectuados.

La rapidez de acceso a la secuencia de instrucciones que empieza en la dirección a donde se salta es fundamental para mejorar las prestaciones del esquema de gestión de los saltos condicionales.

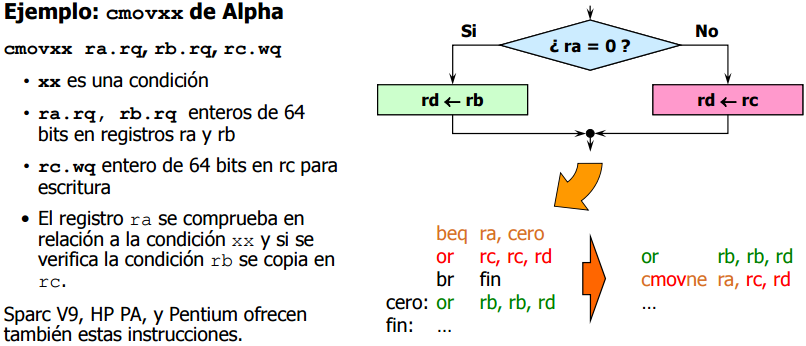
Esquema de cache de direcciones de destino de salto (BTAC)

* Se añade una cache que contiene las direcciones de las instrucciones destino de los saltos, junto con las direcciones de las instrucciones de salto.
* Se leen las direcciones al mismo tiempo que se captan las instrucciones de salto.

Instrucciones de Ejecución Condicional (Guarded Execution)

Se pretende reducir el número de instrucciones de salto incluyendo en el repertorio máquina instrucciones con operaciones condicionales (‘conditional operate instructions’ o ‘guarded instructions’)

Estas instrucciones tienen dos partes: la condición (denominada guardia) y la parte de operación



**Lección 13. Procesamiento VLIW**

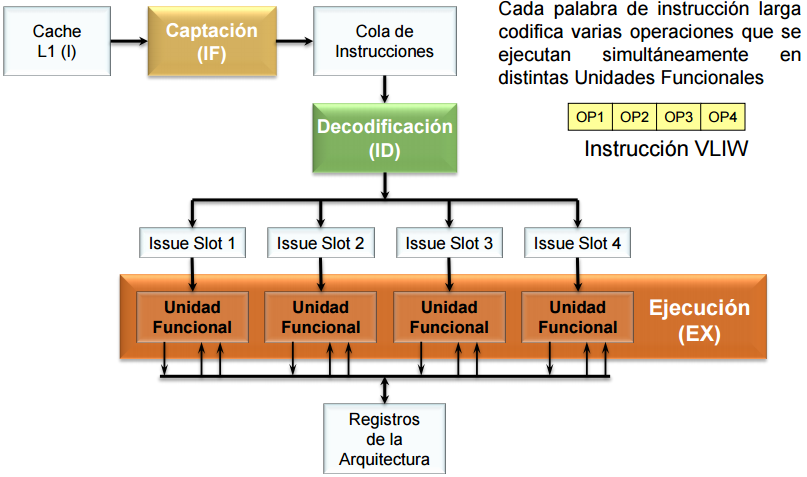
**|** **Características generales de los procesadores VLIW |**

Las arquitectura VLIW utilizan varias unidades funcionales independientes.

En lugar de tratar de enviar varias instrucciones independientes a las unidades funcionales, una arquitectura VLIW empaqueta varias operaciones en una única instrucción (muy larga, Very Long, por ejemplo, entre 112 y 128 bits) u ordena las instrucciones en el paquete de emisión con las mismas restricciones de independencia.

La decisión de qué instrucciones se deben emitir simultáneamente corresponde al compilador (hardware más sencillo que el de un superescalar).

Las ventajas de la aproximación VLIW crecen a medida que se pretende emitir más instrucciones por ciclo (el hardware adicional para un superescalar que emita dos instrucciones por ciclo es relativamente pequeño, pero crece a medida que se pretenden emitir más instrucciones por ciclo)



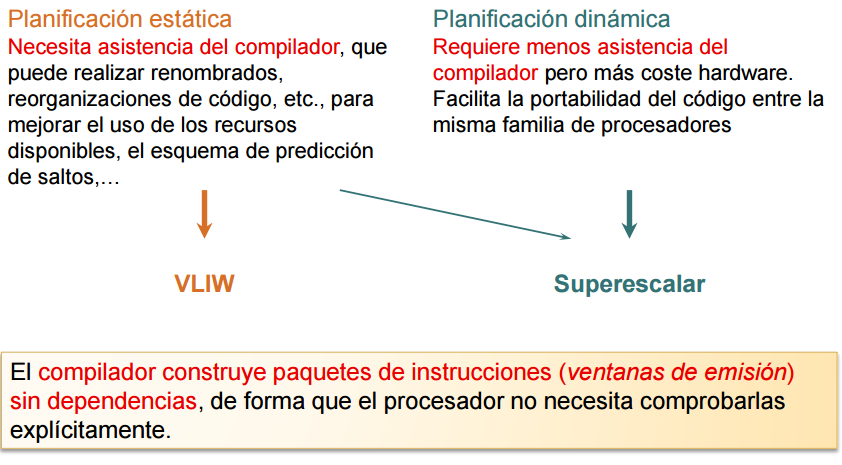
**|Motivación: ILP Hardware vs ILP Software|**

* ILP intensivo en Hardware
  + Capaz de tener en cuenta los eventos que se producen dinámicamente durante la ejecución
  + Mayor portabilidad de los códigos entre plataformas y mejor aprovechamiento de la memoria
* ILP intensivo en Software
  + Capaz de aprovechar la mayor visibilidad del código que tiene el compilador
  + Mayor simplicidad en el hardware y menor consumo de energía

Los procesadores VLIW representan la tendencia hacia el uso de un hardware lo más sencillo posible y la responsabilidad del compilador en la extracción del máximo ILP

**|Planificación Estática|**

El papel del compilador

****

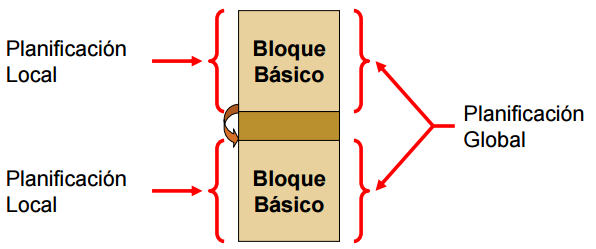
Planificación Estática local y Global

Planificación local:

* actúa sobre un bloque básico (mediante desenrollado de bucles y planificación de las instrucciones del cuerpo aumentado del bucle).

Planificación global:

* actúa considerando bloques de código entre instrucciones de salto.



Planificación Estática Local

Desenrollado de bucles:

* Al desenrollar un bucle se crean bloques básicos más largos, lo que facilita la planificación local de sus sentencias
* Además de disponer de más sentencias, éstas suelen ser independientes, ya que operan sobre diferentes datos

Segmentación software (software pipelining):

* Se reorganizan los bucles de forma que cada iteración contiene instrucciones tomadas de distintas iteraciones del bloque original
* De esta forma se separan las instrucciones dependientes en el bucle original entre diferentes iteraciones del bucle nuevo

Planificación estática global

La planificación global mueve código a través de los saltos condicionales (que no correspondan al control del bucle)

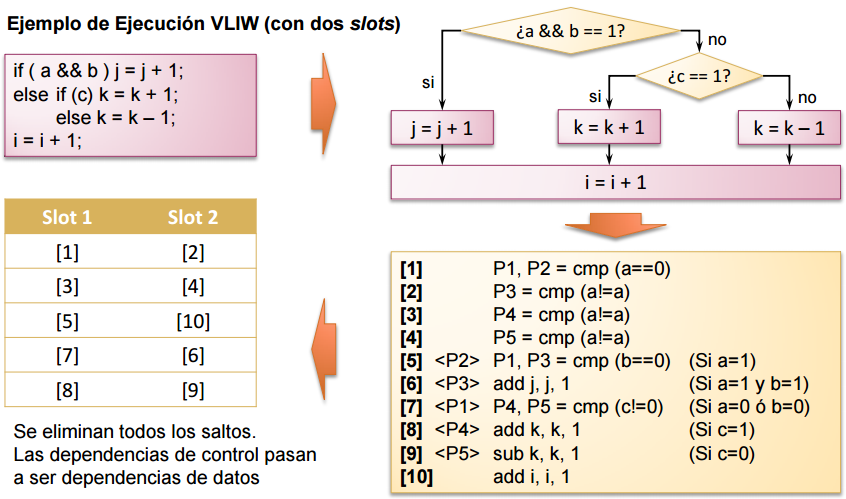
Se parte de una estimación de las frecuencias de ejecución de las posibles alternativas tras una instrucción de salto condicional

Apoyo para facilitar la planificación global:

* Instrucciones con predicado
* Especulación

Instrucciones con Predicado

* Reducen el número de saltos condicionales. Esto es bastante importante, sobre todo si no hay una opción más frecuente que otra en el salto.
* El uso general de predicados es muy útil en planificación global ya que puede eliminar todos los saltos condicionales que no sean de control de bucle (facilita la planificación de traza y la construcción de superbloques)
* Las instrucciones de movimiento condicional de datos son las más utilizadas, aunque pueden ser ineficientes si se dispone sólo de ellas para transformar trozos de código largos que dependen de saltos.
* Instrucciones de movimiento condicional: MIPS, Alpha, SPARC, IA-32 (Pentium)
* Uso generalizado de predicados: IA-64
* Hay arquitecturas que permiten uso general de predicados (full predication): la ejecución de cualquier instrucción está controlada por un predicado. Ej: ARM (es segmentado pero no es VLIW)



**|Procesamiento Especulativo|**

El procesamiento especulativo se basa en la predicción de que determinada instrucción, condición, etc. será muy probable, para adelantar su procesamiento, mejorando las prestaciones del procesador.

El procesamiento especulativo tiene un coste si la predicción que se ha hecho no es correcta. Este coste va desde el correspondiente a haber ejecutado una instrucción que no tendría que haberse ejecutado, hasta la necesidad de incluir código que deshaga el efecto de la operación implementada, vigilar el comportamiento frente a las excepciones, etc.

Uso de Centinelas para Permitir la Especulación de las Referencias a Memoria

Cuando no existe ninguna ambigüedad, el compilador adelanta los LOADs con respecto a los STOREs para reducir la longitud del camino crítico en el código

Cuando existe ambigüedad:

* Se incluye en la arquitectura una instrucción para comprobar los conflictos de direcciones.
* La instrucción se sitúa en la posición original del LOAD (centinela)
* Cuando se ejecuta el LOAD especulativo, el hardware guarda la dirección a la que se ha realizado el acceso.
* Si los sucesivos STOREs no han accedido a esa dirección, la especulación es correcta. En caso contrario, la especulación ha fallado.
* Si la especulación ha fallado:
  + Si la especulación afecta al LOAD solamente, se vuelve a ejecutar cuando se llega alν centinela.
  + Si se han ejecutado instrucciones que dependen del LOAD habrá que repetir todasν esas instrucciones (se necesita mantener información de todas ellas en un trozo de código cuya dirección se incluye en la instrucción centinela)

Especulación Software vs. Hardware

Al final, los procesadores que implementan ILP intensivo en hardware han acabado tomando ideas de la especulación software y viceversa:

* Técnicas Software en ILP intensivo en Hardware
  + Soporte para instrucciones condicionales
  + Precaptación de instrucciones y otros elementos para mejorar el acceso a caches
  + Elementos para la predicción de saltos
  + Soporte especial para los LOADs especulativos
* Técnicas Hardware en ILP intensivo en Software
  + Planificación de instrucciones utilizando campos de marca
  + Predicción dinámica de saltos
  + Soporte para procesamiento de excepciones
  + Hardware para verificar la corrección de LOADs especulativos