2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores
Tema 4

Lección 13. Procesamiento VLIW

Material elaborado por los profesores responsables de la asignatura:
Julio Ortega – Mancia Anguita

ugr | Universidad de Granada

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

ATC
Departamento de Arquitectura y Tecnología de Computadores
UNIVERSIDAD DE GRANADA

Bibliografía

- Fundamental
  - Capítulo 5. J. Ortega, M. Anguita, A. Prieto. *Arquitectura de Computadores*. Thomson, 2005. ESIIT/C.1 ORT arq
- Complementaria
  - Sima and T. Fountain, and P. Kacsuk. *Advanced Computer Architectures: A Design Space Approach*. Addison Wesley, 1997. ESIIT/C.1 SIM adv

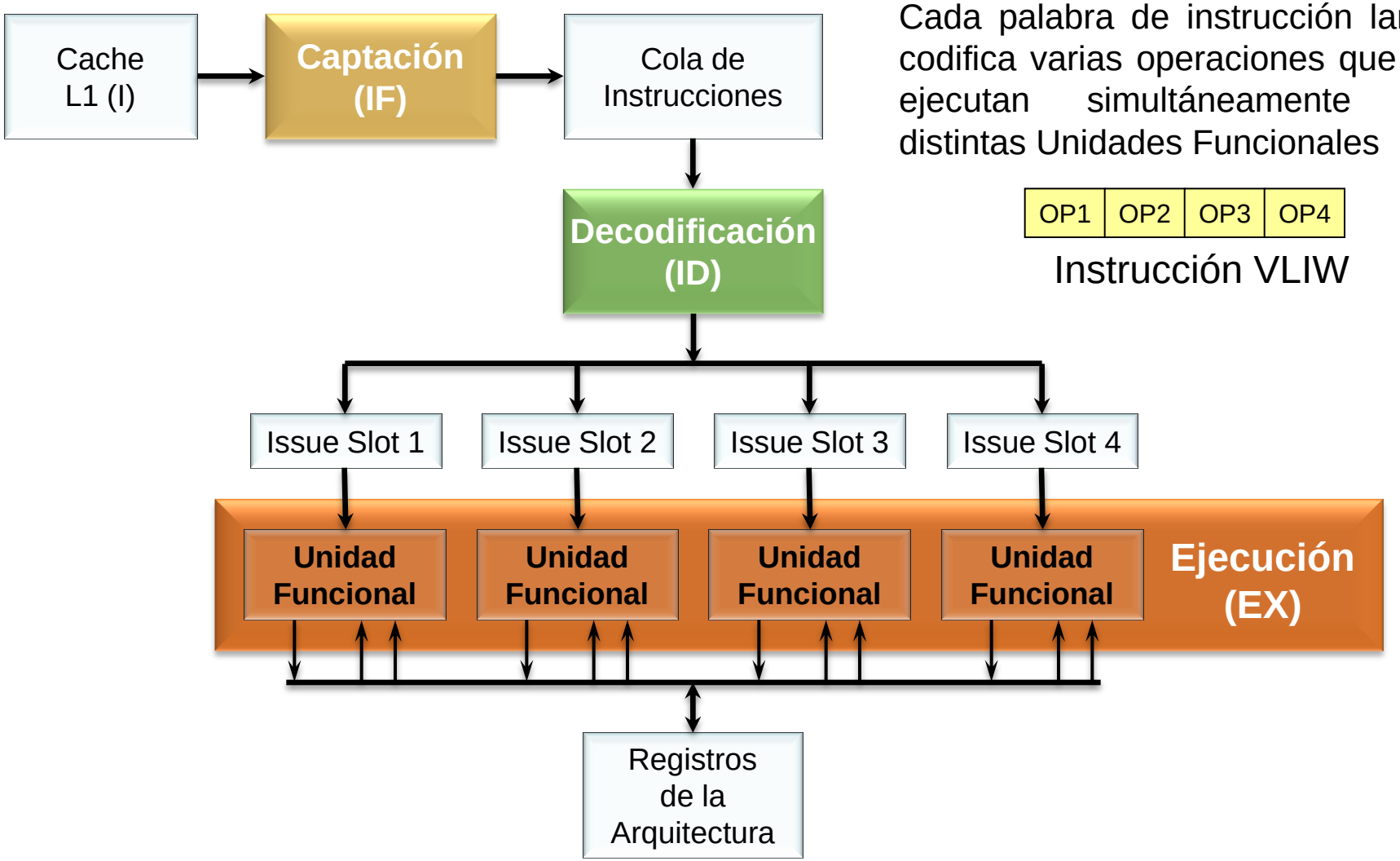
Contenido de la Lección 13

- Características generales y motivación (ILP hardware vs. ILP software)
- Planificación estática
- Procesamiento especulativo

Características generales de los procesadores VLIW I

- 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)

Características generales de los procesadores VLIW II



Cada palabra de instrucción larga codifica varias operaciones que se ejecutan simultáneamente en distintas Unidades Funcionales

OP1	OP2	OP3	OP4

Instrucción VLIW

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

Contenido de la Lección 13

- Características generales y motivación (ILP hardware vs. ILP software)
- Planificación estática
- Procesamiento especulativo

El papel del Compilador

Planificación estática

Necesita asistencia del compilador, que puede realizar renombrados, reorganizaciones de código, etc., para mejorar el uso de los recursos disponibles, el esquema de predicción de saltos,…

Planificación dinámica

Requiere menos asistencia del compilador pero más coste hardware. Facilita la portabilidad del código entre la misma familia de procesadores

VLIW **Superescalar**

El compilador construye paquetes de instrucciones (*ventanas de emisión*) sin dependencias, de forma que el procesador no necesita comprobarlas explícitamente.

Planificación Estática I


```
for ( i = 1000  ;  i > 0  ;  i = i – 1 )
     x[i] = x[i] + s;
```

```
loop:   ld      f0, 0(r1)
        addd    f4, f0, f2
        sd      f4, 0(r1)
        subui   r1, r1, #8
        bne     r1, loop
```

Existen dependencias RAW entre cada dos instrucciones consecutivas, y además existe una instrucción de salto que controla el final del bucle: Parece que no se puede aprovechar mucho ILP

Planificación Estática II

Suponiendo que hay suficientes unidades funcionales para la suma, y que cuando hay dependencias de tipo RAW (las de tipo WAW y WAR las puede evitar el compilador mediante renombrado) los retardos introducidos son:

Instrucción que produce el resultado	Instrucción que usa el resultado	Latencia
Operación FP	Operación FP	3
Operación ALU	Store	2
Load	Operación FP	1
Load	Store	0

	Opción 1 (Sin desenrollar)	Opción 2 (Sin desenrollar)	Ciclos
loop	ld f0,0(r1)	ld f0,0(r1)	1
	----------	subui r1,r1,#8	2
	addd f4,f0,f2	addd f4,f0,f2	3
	----------	---------	4
	----------	bne r1,loop	5
	sd f4,0(r1)	sd f4,8(r1)	6
	subui r1,r1,#8		7
	----------		8
	bne r1,loop		9
	----------		10

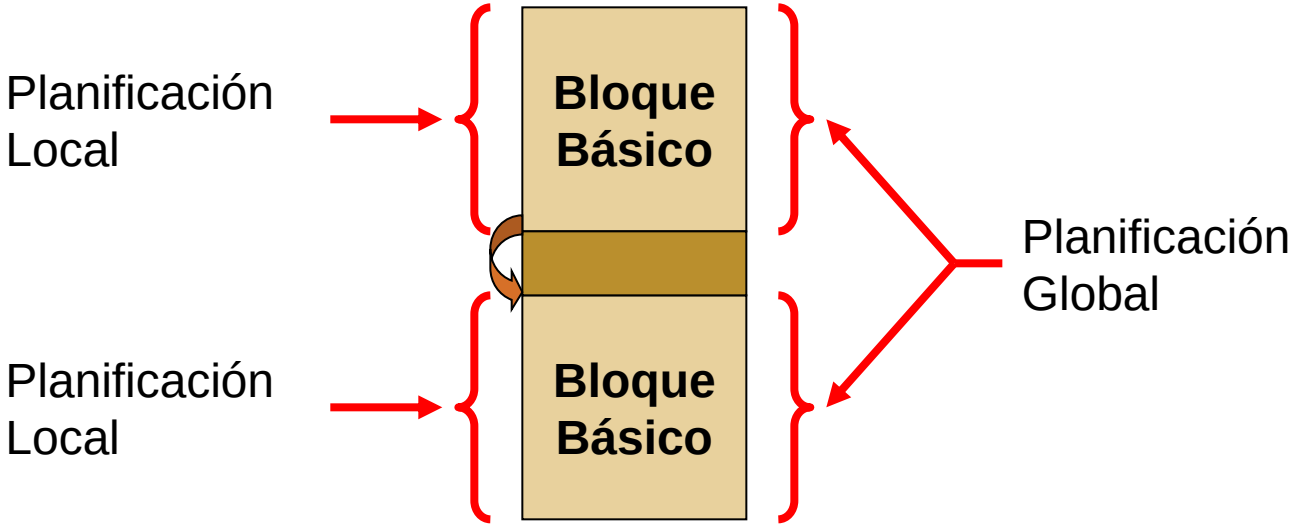
Planificación Estática III

	Instrucción Entera	Instrucción FP	Load/Store	Ciclo
loop	subui r1, r1, #8		ld f0, 0(r1)	1
				2
		addd f4, f0, f2		3
				4
	bne r1, loop			5
			sd f4, 0(r1)	6
	Slot 1	**Slot 2**	**Slot 3**	

La arquitectura VLIW no ganaría nada frente a la opción 2 del bucle sin desenrollar. Además, se necesitarían 12 palabras en el código VLIW frente a las 5 que se utilizaría en la codificación escalar.

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 del código transformado 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 con Desenrollado de Bucles I


```
for ( i = 1000  ;  i > 0  ;  i = i – 1 )
      x[i] = x[i] + s;
```

```
loop:   ld      f0, 0(r1)
        addd    f4, f0, f2
        sd      f4, 0(r1)
        subui   r1, r1, #8
        bne     r1, loop
```

Existen dependencias RAW entre cada dos instrucciones consecutivas, y además existe una instrucción de salto que controla el final del bucle:
Parece que no se puede aprovechar mucho ILP

```
loop:   ld      f0, 0(r1)
        ld      f6, –8(r1)
        ld      f10, –16(r1)
        ld      f14, –24(r1)
        ld      f18, –32(r1)
        addd    f4, f0, f2
        addd    f8, f6, f2
        addd    f12, 10, f2
        addd    f16, f14, f2
        addd    f20, f18, f2
        sd      f4, 0(r1)
        sd      f8, –8(r1)
        sd      f12, –16(r1)
        sd      f16, –24(r1)
        sd      f20, –32(r1)
        subui   r1, r1, #40
        bne     r1, loop
```

El desenrollado pone de manifiesto un mayor paralelismo ILP

Planificación Estática con Desenrollado de Bucles II

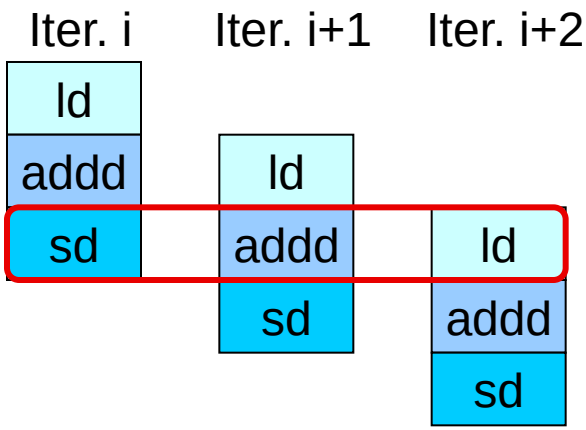
	Instrucción Entera	Instrucción FP	Load/Store	Ciclo
loop			ld f0, 0(r1)	1
			ld f6, 0(r1)	2
		addd f4, f0, f2	ld f10, –16(r1)	3
		addd f8, f6, f2	ld f14, –24(r1)	4
		addd f12, f10, f2	ld f18, –32(r1)	5
		addd f16, f14, f2	sd f4, 0(r1)	6
	subui r1, r1, #40	addd f20, f18, f2	sd f8, –8(r1)	7
			sd f12, 24(r1)	8
	bne r1, loop		sd f16, 16(r1)	9
			sd f20, 8(r1)	10
	Slot 1	**Slot 2**	**Slot 3**	

Se tardarían 10x(1000/5)=2000 ciclos, frente a los 10x1000=10000 ó 6x1000=6000 ciclos del bucle sin desenrollar.

Planificación Estática con Segmentación Software I


```
for ( i = 1000  ;  i > 0  ;  i = i – 1 )
      x[i] = x[i] + s;
```

```
loop:   ld      f0, 0(r1)
        addd    f4, f0, f2
        sd      f4, 0(r1)
        subui   r1, r1, #8
        bne     r1, loop
```




```
loop:   sd      f4, 16(r1)
        addd    f4, f0, f2
        ld      f0, 0(r1)
        subui   r1, r1,#8
        bne     r1, loop
```

```
Iter. i:    ld      f0, 16(r1)
            addd    f4, f0, f2
            sd      f4, 16(r1)

Iter. i+1:  ld      f0, 8(r1)
            addd    f4, f0, f2
            sd      f4, 8(r1)

Iter. i+2:  ld      f0, 0(r1)
            addd    f4, f0, f2
            sd      f4, 0(r1)
```

Planificación Estática con Segmentación Software II

	Instrucción Entera	Instrucción FP	Load/Store	Ciclo
loop		addd f4,f0,f2	sd f4,16(r1)	1
	subui r1,r1,#8		ld f0,0(r1)	2
	----------	----------	----------	3
	bne r1,loop			4
	----------	----------	----------	5
	Slot 1	**Slot 2**	**Slot 3**	

- Se tardarían 5x1000=5000 ciclos (algunos más si se consideran las instrucciones previas al inicio del bucle con segmentación software y las posteriores al final del bucle).
- Si se desenrolla este bucle ya segmentado, se pueden mejorar mucho más las prestaciones.

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 y
  - Especulación

Instrucciones de Ejecución Condicional


```
if ( A == 0 )
    S = T;
```

```
    bnez r1, L
    add r2, r3, 0 ; r3=T
L:
```

```
cmovz r2, r3, r1
```

Si se verifica la condición en el último operando (r1=0 en este caso) se produce el movimiento entre los otros dos operandos (r2 ← r3)

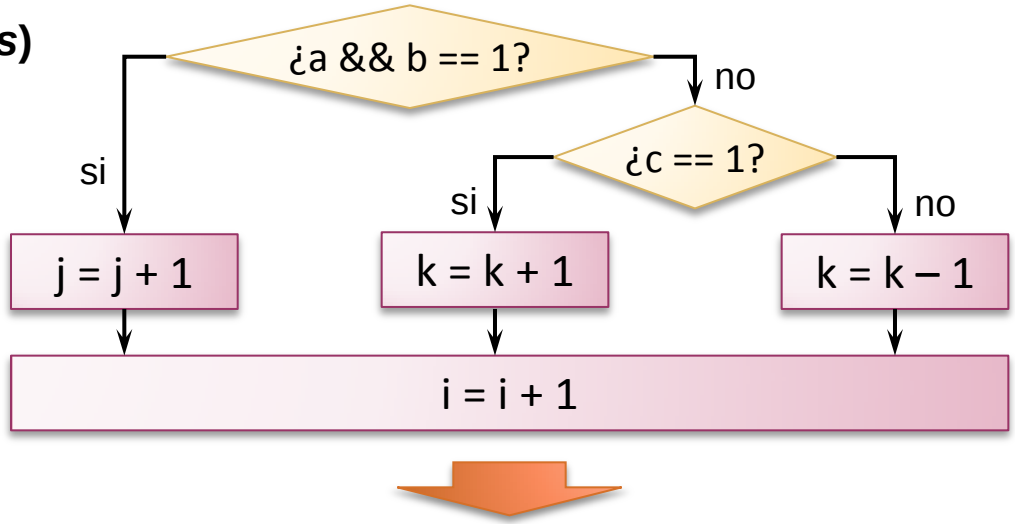
```
if ( B < 0 )
    A= – B;
else
    A = B;
```

```
r2 ← r1
r2 ← – r1  ( condicional a que r1 < 0 )
```

Instrucciones con Predicado I

Ejemplo de Ejecución VLIW (con dos *slots*)

```
if ( a && b ) j = j + 1;
else if (c) k = k + 1;
       else k = k – 1;
i = i + 1;
```



Slot 1	Slot 2
[1]	[2]
[3]	[4]
[5]	[10]
[7]	[6]
[8]	[9]

```
[1]          P1, P2 = cmp (a==0)
[2]          P3 = cmp (a!=a)
[3]          P4 = cmp (a!=a)
[4]          P5 = cmp (a!=a)
[5]  <P2>    P1, P3 = cmp (b==0)     (Si a=1)
[6]  <P3>    add j, j, 1             (Si a=1 y b=1)
[7]  <P1>    P4, P5 = cmp (c!=0)     (Si a=0 ó b=0)
[8]  <P4>    add k, k, 1             (Si c=1)
[9]  <P5>    sub k, k, 1             (Si c=0)
[10]         add i, i, 1
```

Se eliminan todos los saltos.
Las dependencias de control pasan a ser dependencias de datos

Instrucciones con Predicado II

- 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).

Contenido de la Lección 13

- Características generales y motivación (ILP hardware vs. ILP software)
- Planificación estática
- Procesamiento especulativo

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.

Slot 1	Slot 2
lw r1,40(r2)	add r3,r4,r5
	add r6,r3,r7
beqz r10,loop	
lw r8,0(r10)	
lw r9,0(r8)	

Slot 1	Slot 2
lw r1,40(r2)	add r3,r4,r5
lwc r8,0(r10),r10	add r6,r3,r7
beqz r10,loop	
lw r9,0(r8)	

Se produce el lw r8,0(r10) si r10!=0

Se aprovecha el slot de acceso a memoria. Si **beqz** da lugar al salto se ha ejecutado una instrucción de forma innecesaria. **Otro problema son las excepciones**

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 al centinela.
    - Si se han ejecutado instrucciones que dependen del LOAD habrá que repetir todas 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

Para ampliar …

- Páginas Web:
  - http://www.research.ibm.com/vliw (Investigación sobre VLIW en IBM).
- Artículos de Revistas:
  - Fisher, J.A.: "Very Long Instruction Word Architectures and ELI-512". Proc. 10th Symp. On Computer Architecture, pp.140-150, 1983. (Aparece el término VLIW).
  - Rau, B.R.; et al.: "The Cydra 5 departamental supercomputer: design philosophies, decisions, and trade-offs". IEEE Computers, pp.22-34, 22:1, 1989.

Para ampliar …

- Páginas Web:
  - http://www.cs.ucsd.edu/classes/sp00/cse231/mdes.pdf
    (artículo sobre compiladores para VLIW y EPIC)
  - http://www.compilerconnection.com/index.html
    (página bastante completa con información relativa a compiladores: compañías, investigación, grupos de noticias,…)
- Artículos de Revistas:
  - Lam, M.: “Software Pipelining: An effective scheduling technique for VLIW processors”. Proc. ACM SIGPLAN Conf. Prog. Lang. Design and Implementation, pp.318-328, 1988.
  - Wilson, R.P.; Lam, M.: “Efficient context-sensitive pointer analysis for C programs”. Proc. ACM SIGPLAN Conf. Prog. Lang. Design and Implementation, pp.1-12, 1995.
  - Mahlke, S.A., et al.: “Sentinel Scheduling for VLIW and superscalar processors”. Proc. 5th Conf. On Architectural Support for Prog. Languages and Operating Systems, pp.238-247, 1992.