TEMA 1: ARQUITECTURAS ENCADENADAS AVANZADAS

1.1 Enfoques del paralelismo a nivel de instrucciones: Enfoque superescalar, VLIW, supersegmentado (*superpipeline*).

1.2 Procesadores RISC superescalares: coste hardware y la replicación de unidades funcionales, Políticas de búsqueda, problemas de implementación, Modelos superescalares para el DLX, Descripción de ejemplos reales.

1.3 Paralelismo de instrucciones disponible: programas enteros, científicos y multimedia.

1.4 Ejecución especulativa y predicativa: Especulación hardware: buffer de reordenación, Especulación software, Especulación software con ayuda del hardware, Ejecución predicativa.

Nociones sobre:

1.5 Procesadores CISC encadenados y superescalares.

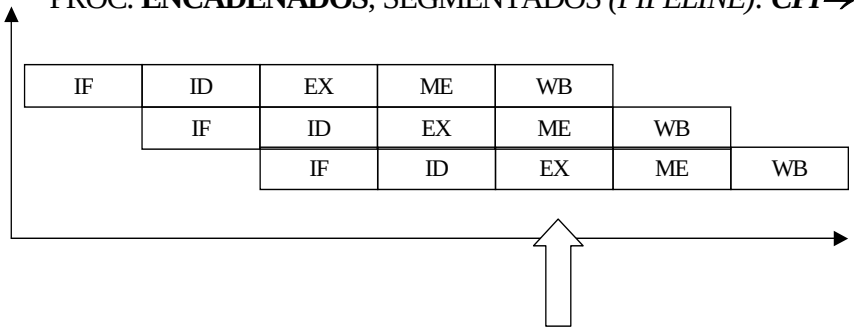
1.6 Organización y gestión de la memoria en procesadores avanzados.

1.7 Organización y gestión de los sistemas de E/S en sistemas avanzados.

1.8 El problema de las excepciones: interrupciones precisas.

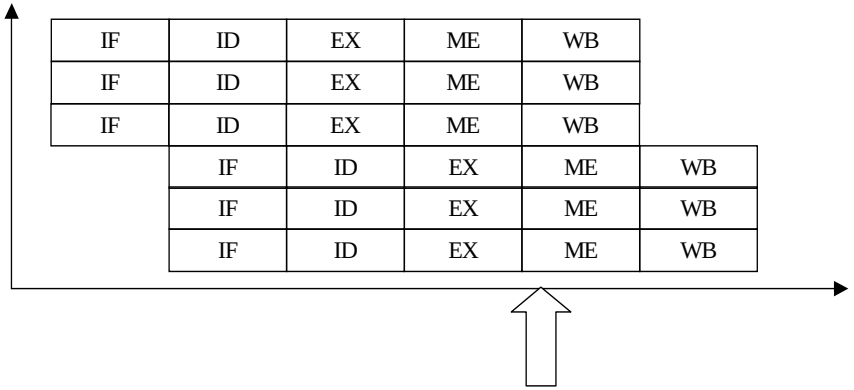
IDEA: Paralelismo a nivel de instrucciones (ILP)

PROC. **ENCADENADOS**, SEGMENTADOS *(PIPELINE): **CPI→1***



k = Nº FASES
k INSTRUCCIONES "EN VUELO"

PROCESADORES *(SUPERESCALARES Y OTROS)*: **CPI<1**



k = Nº FASES. **m** = Nº INSTRUCC A LA VEZ
k×m INSTRUCCIONES "EN VUELO"

OBJETIVOS PRINCIPALES:

- Modelos de máquinas con paralelismo de instrucciones (ILP): ventajas/inconv.
- Gran coste hardware y dificultades de implementación.
- Futuro: Límite del paralelismo de instrucciones (ILP).
- ¿Ayudas del compilador/programador?: Programar teniendo en cuenta la arquit.
- Especulación hardware/software.

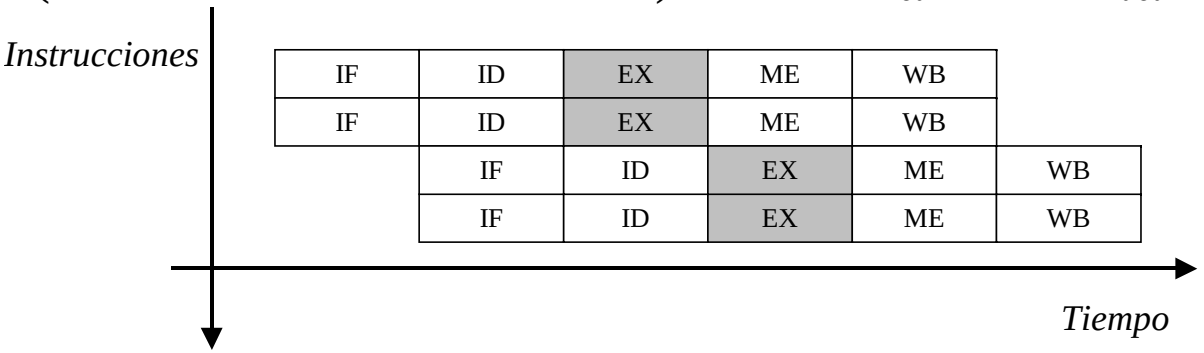
1.1. Enfoques del paralelismo de instrucciones: superescalar, VLIW, supersegmentado.

$$t_{ejecución} = N_{inst} \times CPI \times \tau$$

- Hay tres formas de mejorar este tiempo, reduciendo cada uno de los factores.
- **Los procesadores reales** suelen reducir alguna combinación de ellos.
- **Modelos teóricos** según factor que se reduce:
  - Para reducir el CPI necesitamos emitir más de una instrucción por ciclo de reloj. A estos procesadores se les llama **Superescalares**. Recordemos que a los que emiten una instrucción por ciclo con registros No vectoriales se les llama escalares.
  - Para reducir el número de instrucciones N_{inst}, necesitamos hacer más densas las instrucciones (contengan más información, más bits), de forma que una emisión dé lugar a que trabajen varias unidades funcionales al mismo tiempo. Estos procesadores se denominan **LIW** o **VLIW** ("Very Long Instruction Word").
  - Y la reducción del período de reloj τ sin reducir el tiempo total de cada etapa, y permitiendo que se emita una instrucción por ciclo, se denomina "**superpipeline**", **supersegmentado o superencadenado**. Un procesador superpipeline posee sus etapas supersegmentadas, cada una de las cuales duran varios ciclos.

Definiciones(I): Máquina Superescalar

- **Reducir CPI.** Capaz de emitir más de una instrucc. por ciclo, pero código "normal"
- DEF: ***m* = grado de superescalaridad**. $IPC_{ideal} = m$
- DEF: $IPC = (CPI)^{-1}$ = *Número de instrucciones por ciclo.*
- Luego **idealmente**: $IPC_{ideal} = m>1$, $CPI = 1/m < 1$
  - Bloqueos (de datos, estructurales, control) harán $IPC_{real} <= IPC_{ideal}$

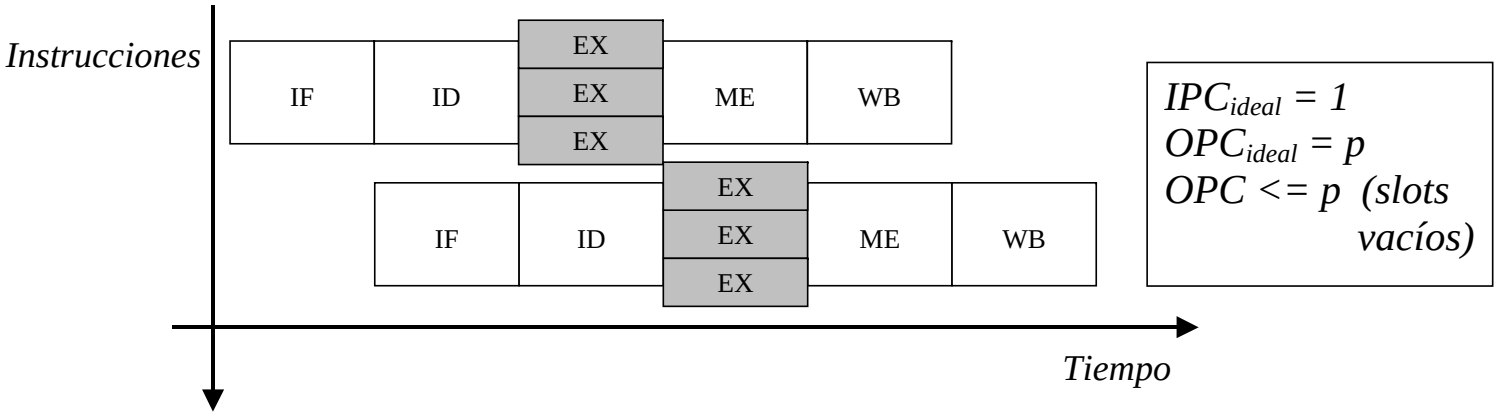


Primeros superescalares: 1991, Intel 960CA y IBM Power 1 (*m*=4)
Procesadores año 2001 y grado superescalaridad *m* (¡Saturado desde 1996 en *m*=3~4!)

MIPS R14000	4	HP PA 8600	4
UltraSPARC III	4	Alpha 21264B	4
Pentium Pro/II/III/4	3	Power PC 7400	3
AMD Athlon	3		

Definiciones(II): Máquina LIW o VLIW

- **Reducir N_{inst}** . "Very Long Instruction Word"
- Instrucciones tienen información para la ejecución de hasta *p* operaciones en paralelo

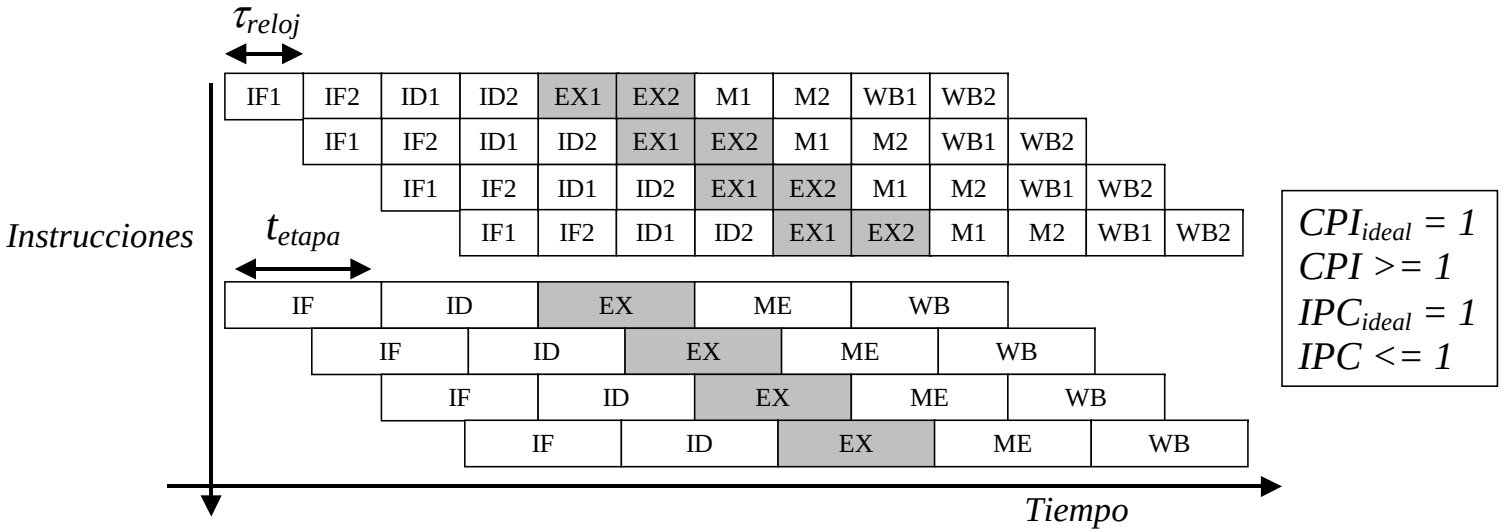


Procesadores VLIW año 2001, ***p slots*** (nº de operaciones por instrucción, concepto similar al grado de superescalaridad *m*) y tamaño de instrucción en bits:

Itanium	3	128	Además puede lanzar *m*=2 instrucc. VLIW por ciclo (grado total 6)
Trimedia	5	variable (código comprimido)	Muy parecido a VLIW "puro" Para sistemas empotrados
Transmeta Crusoe	4	128 & 64	Para sistemas empotrados

Definiciones(III): Máquina súper encadenada, Superpipeline o súper segmentada

- **Reducir** τ: Tiene periodo de reloj τ menor que la duración de una etapa de la cadena típica, es decir, cada etapa necesita varios ciclos de reloj. Pero sólo se puede emitir una instrucción por ciclo de reloj, de manera que las etapas están supersegmentadas.
- **Avance tecnológico** pero no arquitectónico.
- Dos posibles esquemas:



- Cada etapa necesita 2 ciclos de reloj, pero en cada ciclo se emite una instrucción. Realmente existen las fases IF1, IF2, ID1, ID2, ...

- Hay que tener en cuenta que el periodo de reloj ha bajado, pero el tiempo para concluir la etapa, no. Podemos hablar **de dos periodos** (de reloj y etapa).
- La relación entre ambos es el **grado de superencadenamiento *n*** : $\tau_{etapa} = n\ \tau_{reloj}$
- El fabricante siempre hablará de 10 fases y periodo τ_{reloj} ($\nu=1/\tau_{reloj}$). En realidad puede que p.ej. una de las fases originales se divida en 4 fases, otra en 3, otra en 2 y otras en 1. Ej: Pentium P6 (Pentium Pro, PII y similar al PIII): once fases

IFU1	IFU2	IFU3	DEC1	DEC2	RAT	ROB	DIS	EX	RET1	RET2
IF1	IF2	IF3	IS1	IS2	IS3	IS4	IS5	EX1	WB1	WB2

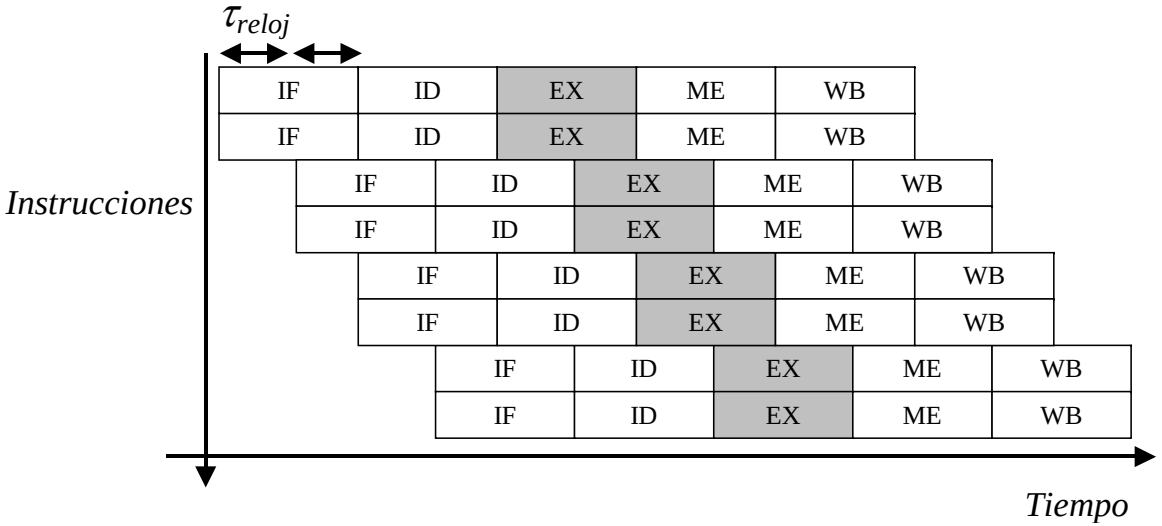
- Así que el grado de superencadenamiento ***n*** **es, en la realidad, aproximado**. Podemos teóricamente hablar de él si lo comparamos con un DLX de 5 fases (planif. estática) o de 4 fases (planif. dinámica). (NOTA: Realmente existe una fase adicional de especulación dinámica, se verá en sig. apartado)

Procesadores año 2001 y grado de superencaden. $n=k/4$ aproximado (k=nº etapas, todos son de planif. dinámica: IF IS EX WB) y frec. MHz. Notar la **relación entre *n* y frecuencia**. Cuidado: el **rendimiento no es proporcional a la frecuencia**.

Procesador 2001	*k*	*k/4*	ν_{reloj}	ν_{etapa}	*Procesador 2001*	*k*	*k/4*	ν_{reloj}	ν_{etapa}
MIPS R14000	6	1.5	400	266	HP PA 8600	7	1.75	552	315
UltraSPARC III	14	3.5	900	257	Alpha 21264B	7	1.75	833	476
Pentium III	11	2.75	1000	363	Power PC 7400	4	1	450	450
Pentium 4	22	5.5	1700	309					

Definiciones(IV): Máquinas Superescalares y Superpipeline

- Uniendo ambos conceptos estas máquinas podrían emitir *m* instrucciones por ciclo de reloj, además de tener las etapas supersegmentadas en grado *n*, reduciéndose el periodo de reloj. Su cronograma quedaría como:



- Los procesadores reales antes mencionados tienen cierto grado *m* de superescalaridad y *n* de superencadenamiento.
- Nº instrucciones "en vuelo": $k \times n \times m$. Ej. P4: $4 \times 5.5 \times 3 = 66$ (en realidad tiene más como veremos más adelante). $k = 4$ = nº fases sin supersegmentar.

Tipos de procesadores con ILP según Arquitectura

- La planificación y emisión determinan en gran manera la arquitectura interna, y el modo en que se extrae paralelismo a nivel de instrucciones ILP.
- Según la Emisión (Issue) y Planificación (Scheduling), vamos a distinguir los siguientes modelos:

Emisión : / *Planificación*	**Estática** (procesador no detecta ninguna dependencia; el compilador lo "prepara" todo)	**Dinámica** (procesador sí detecta dependencias; compilador podría optimizar)
Estática (fase ID)	**VLIW "puro"**	**Superescalar**
Dinámica (similar Algoritmo Tomasulo, fase IS)	**No tiene sentido**	**Superescalar**

Esto no es una clasificación rígida, sino modelos. Existen en la realidad casos intermedios:

- VLIW donde ciertos bits del opcode (introducidos en tiempo de compilación) dan pistas sobre la emisión.
- VLIW donde en la emisión se detectan algunas dependencias, etc.
- Superescalar planif. estática, con emisión tan simple que se acerca a VLIW

1.2 Implementación de Proc con ILP: VLIW "puro"

- Este es el concepto de máquina de emisión múltiple en su **forma más estática.**
- **IDEA: Trasladar complejidad de emisión al compilador**
- Un **compilador deberá ser capaz de agrupar** varias *instrucciones "clásicas"*, en *macroinstrucciones VLIW* para aprovechar los recursos: **EMISIÓN ESTÁTICA**.
- Depende del compilador. No tiene algoritmo de planif. dinámica.
- Además, necesita **técnicas avanzadas de planif. estática,** evitando al máximo las dependencias de datos (VLIW "puro" no detecta riesgos, nunca bloquea la máq.)

  ▹ Trasnmeta Crusoe: Cuatro "*slots*", cada puede ser:

Acc. Memoria	ALU INT/FP/MMX	ALU INT	operando Inm32

Acc. Memoria	ALU INT/FP/MMX	ALU INT	Salto

  ▹ Itanium: Tres "slots" de 41 bits más 5 bits para especificar que UF requiere cada slot y las posibles dependencias. Tiene muchos tipos de instrucciones que resumimos aproximadamente en (VLIW actual):

Acc. Memoria /salto	Cualquiera/ Inm32	Cualquiera excepto acc. mem./Inm32

Los 1º VLIW tenían instr muy rígidas: obligaban a recompilar el código. Los actuales tienen instr más flexibles y complejas: *no necesitan recompilar* (aunque sería mejor).

- **IF-ID:** Sólo habrá hardware para una IF, ID por ciclo (aunque más cantidad de hardw que en un encadenado). En VLIW puro las etapas IF e ID van a ser muy simples, ya que se emitirá lo que haya en la macroinstrucción (no tendremos que hacer comprobaciones de criterios o reglas de emisión, ni de dependencias de datos). Variante: **EPIC** (Explicitly Parallel Instruction Computers). Itanium: 5 bits de la propia instrucción informan de las dependencias (datos+estructurales) que hay entre las propias operaciones de una instrucción VLIW, y entre las de una con la siguiente.
- **EX:** La información pasará a varias UF (en el mismo orden que puso el compilador) para se ejecuten en paralelo. No hay planificación dinámica (in-order-execution).
- **MEM:** El número de accesos a memoria suele ser dos (tienen caché o memoria de doble puerto, o separada en dos bancos). Fallo de caché ⇒ bloqueo total de la máq.
- En **WB** tienen que guardarse más resultados al mismo tiempo. **Nº puertos ↑↑**
- Ventaja: Ahorro hardware (técnicas dinámicas), permite **complicar otros recursos**: 128 registros GPR de 64 bits, 128 reg FP, etc. en Itanium. Pocas fases ⇒ penalidad por fallo de predicción (BTB) más baja. **Menor consumo** ⇒ sist. empotrados
- Problema: **tamaño del código (**⇒ en general el tamaño de los programas es mayor que en RISC superescalares**).**
  - Si un **"slot" no puede ser llenado por el compilador: mete un NOP**.
  - De las técnicas avanzadas de planif. estática, el desenrollado exhaustivo para rellenar los slots, hace crecer mucho el código.

Ejemplo 1: VLIW

- Trabajaremos con este sencillo código en todo el tema y con todos los modelos.
- Supongamos un DLX VLIW puro cuya instr. se compone de:
  - **2 operaciones de acceso a memoria**
  - **2 operaciones FP**
  - **1 operación INT o Salto**

Notar que la **detección de dependencias** de datos (p. ej. RAW) la debe hacer **el compilador**, de forma que nunca dos slots de la misma instrucción tendrán dependencia. Cualquier bloqueo se implementa como **bloqueo software** (afecta a toda la cadena). También el compilador debe **conocer las latencias (problema: recompilación necesaria).**

Supongamos (similar al DLX, duración EX FP=4 ciclos):

Instrucción que genera un dato	**Instrucción que lo usa**	**Nº c bq**
FP ALU	otra FP ALU	3
FP ALU	Store (FP)	2
Load (FP)	FP ALU	1
Load (FP)	Store (FP)	0
Int	Salto	1

Las **UF están totalmente segmentadas** (esto es habitual hoy día, de lo contrario se producirían muchos bloqueos estructurales, dado la cantidad de instrucciones FP que puede emitir por ciclo estas máquinas).

Conflictos estructurales: supongamos que no hay contención, ya que se dota al fichero de registros FP de todos los puertos necesarios (es lo habitual, para que no se produzcan muchos bloqueos). Calcular Nº puertos necesarios.

- Se pide: **Desenrollar el código dado en las iteraciones necesarias para eliminar dependencias, y que al menos 1 macroinstr tenga todos sus slots ocupados.**
- Calcular OPC, y el Nº de ciclos por elemento procesado (al desenrollar, esta es la medida que interesa)
- Porcentaje de slots llenos, relación entre tamaño código original y el del VLIW
- Suponer instrucción de 128 bits y BTB que siempre acierta.

```
for (i=N-1; i>0; i++) x[i] = x[i] +s;
//N es divisible por cualquier número
```

```
// F2←s  ;      Inicialmente:  R1 ← &x[N-1]
  Loop:   LD     F0, 0(R1)
          ADDD F4, F0, F2
          SD     0(R1), F4
          SUBI  R1, R1, 8
          BNEZ  R1, Loop
```