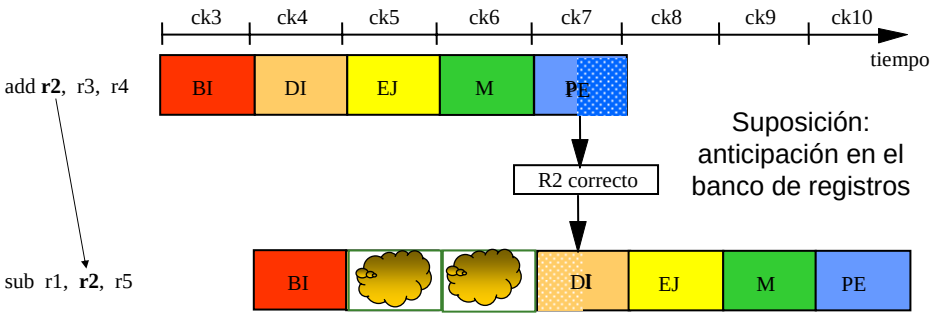
Tema 3: Anticipación

- Eliminar detenciones en presencia de riesgos de datos
- Eliminar detenciones en presencia de riesgos de control
- Tratamiento de las excepciones
- Tratamiento de las instrucciones multiciclo
- Ejemplos de procesadores segmentados y supersegmentados

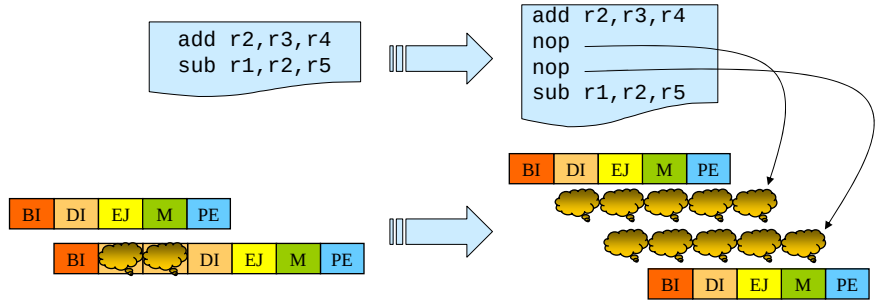
3.1Eliminar detenciones en presencia de riesgos de datos

- Nos centraremos en los riesgos RAW (dependencias verdaderas)
  - ✓ Los riesgos WAR y WAW no pueden ocurrir con la implementación actual del DLX
  - ✓ Los riesgos WAW si pueden aparecer con instrucciones multiciclo (ver sec. 3.4)
- En el tema anterior hemos resuelto los riegos RAW mediante dentenciones
  - ✓ Pérdida de ciclos de reloj → pérdida de rendimiento



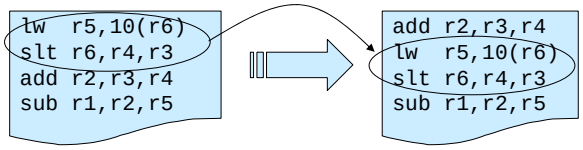
3.1.1 Clasificación de soluciones
Soluciones software (1)

- Insertar instrucciones nop (no-operación)
  - ✓ Es equivalente a la detención de la instrucción dependiente
    - Ventaja: ahorras implementar un HW para detectar riesgos y provocar la detención
    - Inconvenientes:
      - Sigues perdiendo tantos ciclos como instrucciones nop insertes (ya que no hacen nada útil)
      - Ahora es el compilador (o el programador) el que tiene que detectarlos para insertar nops



3.1.1 Clasificación de soluciones
Soluciones software (2)

- Planificación SW: reordenar el código
  - ✓ Consiste en insertar instrucciones del código entre las instrucciones que provocan riesgos
    - Las instrucciones se pueden cambiar de orden si no dependen entre si
    - Ventaja: si lo consigues → ya no pierdes ciclos de reloj
    - Inconveniente: el compilador (o programador) tiene que hacer el trabajo
  - ✓ Ejemplo de planificación SW:

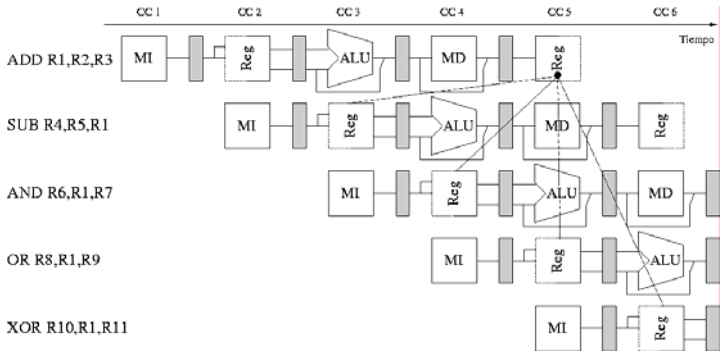


  - ✓ La instrucción add no depende de las instrucciones lw y slt
    - Se pueden cambiar de orden con el add sin modificar la semántica del programa
  - ✓ La instrucción slt sí depende de lw así que no se pueden cambiar de orden
- El compilador no siempre encuentra instr. independientes para insertar
  - ✓ En ese caso y a falta de soluciones HW, hay que introducir nops

3.1.1 Clasificación de soluciones
Soluciones hardware

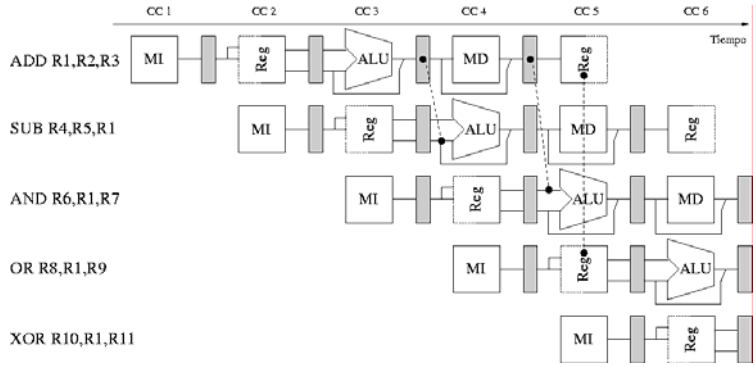
- Es necesario detectar el riesgo por HW → lógica de detección de riesgos
- Dos alternativas
  - ✓ Detención:
    - Detener la ejecución uno o más ciclos hasta que se resuelva el riesgo
    - Pérdida de ciclos de reloj
    - Implica la implementación de una lógica para insertar burbujas (≅ nops hardware)
  - ✓ Anticipación
    - Implementar adelantamientos o cortocircuitos (forwarding o bypassing)
    - Anticipar el resultado que produce una etapa en una instrucción a otra etapa que consume ese resultado para otra instrucción
      - No hay por qué esperar hasta que un resultado se escribe en el banco de registros
      - En cuanto está disponible se puede adelantar (anticipar) a la etapa que lo necesita
    - Reduce la pérdida de ciclos
    - Etapas productoras de datos en el DLX: EJ y M
    - Etapas consumidoras de datos en el DLX: EJ y M (HW de detección de 0 incluido en EJ)
    - Cortocircuitos implementados: EJ-EJ, M-EJ y M-M (EJ-M en instrucciones multiciclo)

3.1.2 Anticipación
Ejemplo sin anticipación:



- ✓ Todas las instrucciones tienen dependencia verdadera con la primera (add)
- ✓ Sólo sub y and están lo suficientemente cerca para provocar riesgo RAW
- ✓ El riesgo de or se evita por anticipación en el banco de registros
- ✓ El xor también depende de add, pero esta más lejos y no provoca RAW

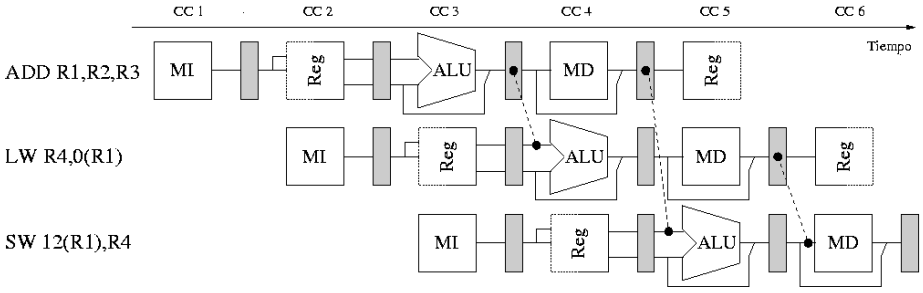
3.1.2 Anticipación
Ejemplo con anticipación:



- Las salidas de una unidad funcional son realimentadas como entradas a la misma unidad o a unidades previas
- La lógica de control de adelantamientos detecta si la fuente de una operación viene del banco de registros o de la salida de una unidad en el ciclo anterior

3.1.2 Anticipación
Generalización de los cortocircuitos

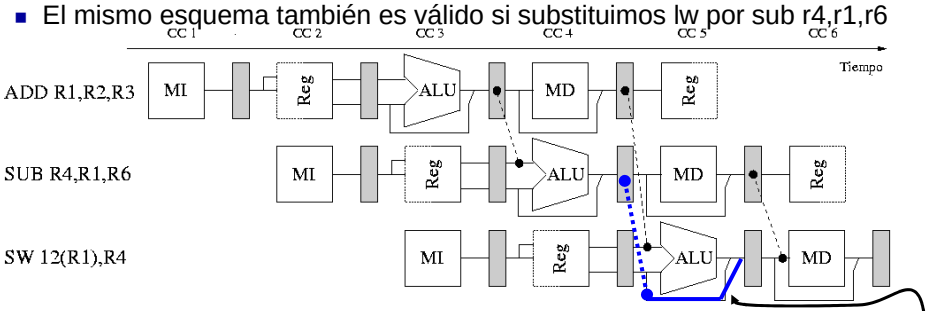
- Cortocircuitos implementados: EJ-EJ, M-EJ y M-M



- Importante:
  - Los datos siempre viajan del final de un ciclo al principio del siguiente
    - Un resultado puede ir del final del CC4 al principio del CC5
    - No puedes ir del final del CC4 al principio del CC3 (al pasado) o del CC6 (al futuro)!!

3.1.2 Anticipación
Generalización de los cortocircuitos

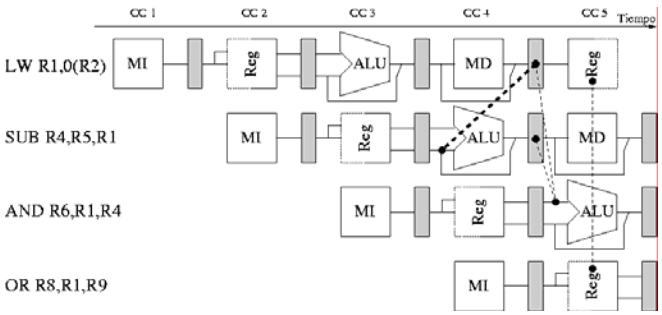
- El mismo esquema también es válido si substituimos lw por sub r4,r1,r6



- ✓ Aunque ahora r4 no se genera en M (sino en EJ) al final del CC5 r4 se ha propagado al registro de segmentación M/PE
- ✓ No es posible un cortocircuito EJ-M para r4 en este segundo ejemplo!!
- ✓ Otra posibilidad (si no está implementado el cortocircuito M-M):
  - r4 se anticipa al final del CC4 por el cortocircuito EJ-EJ y luego en el CC5 pasa a la etapa M

3.1.2 Anticipación
Situaciones sin resolver

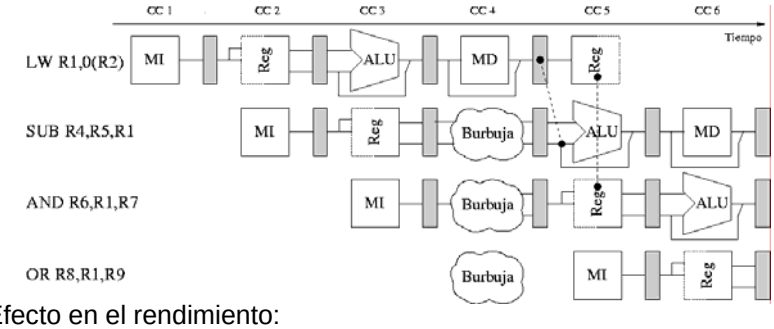
- Los cortocircuitos evitan una gran cantidad de pérdida de ciclos
- Pero aún con cortocircuitos a veces hay que introducir burbujas
- Ejemplo: lw de un dato seguido de una instr. que usa ese dato



El dato generado por lw al final del CC4 no puede viajar al principio del mismo CC4 (al pasado!)

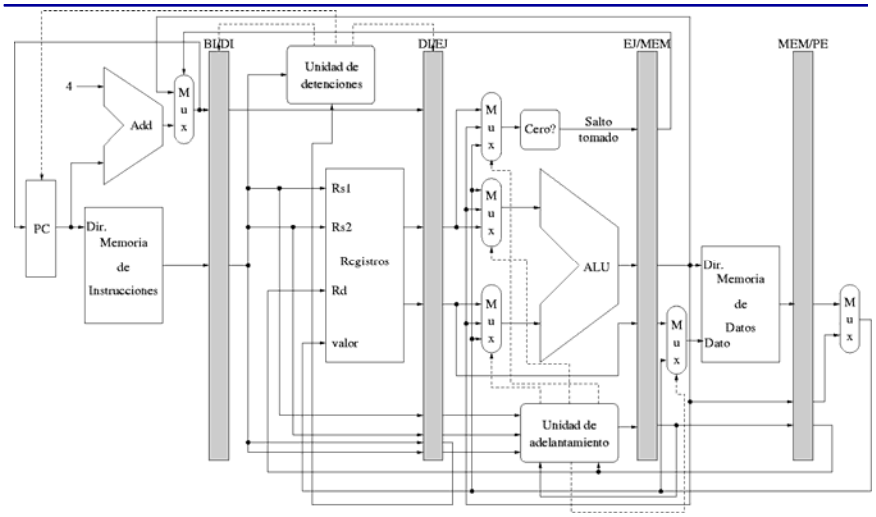
3.1.2 Anticipación
Situaciones sin resolver

- Aunque existan cortocircuitos, en este caso hay que aplicar detención:

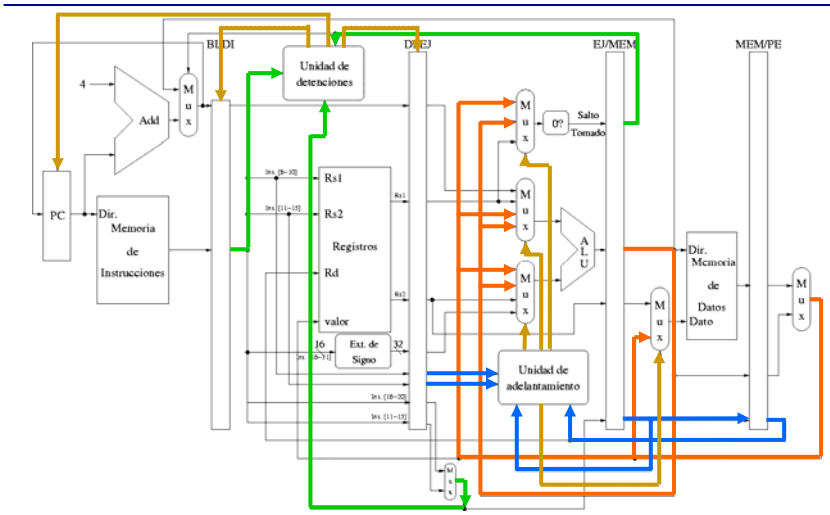


- Efecto en el rendimiento:
  - ✓ Supongamos un programa con un 30% de lw. El 50% de éstos estan seguidos de instrucciones que consumen el dato producido por el lw (RAW)
  - ✓ $CPI_{instrucción\ siguiente\ al\ lw}$=0.5x1ck + 0.5x2ck=1.5; CPI=0.3x1.5 + 0.7x1=1.15
  - ✓ Pérdida de un 15% de rendimiento→ **Solución: planificación SW**

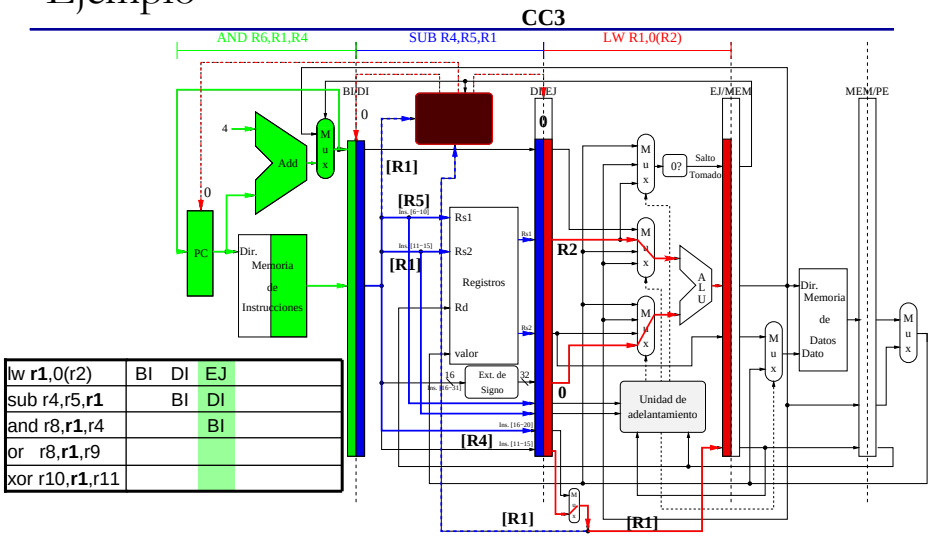
3.1.3 Unidades de anticipación y detención



3.1.3 Unidades de anticipación y detención

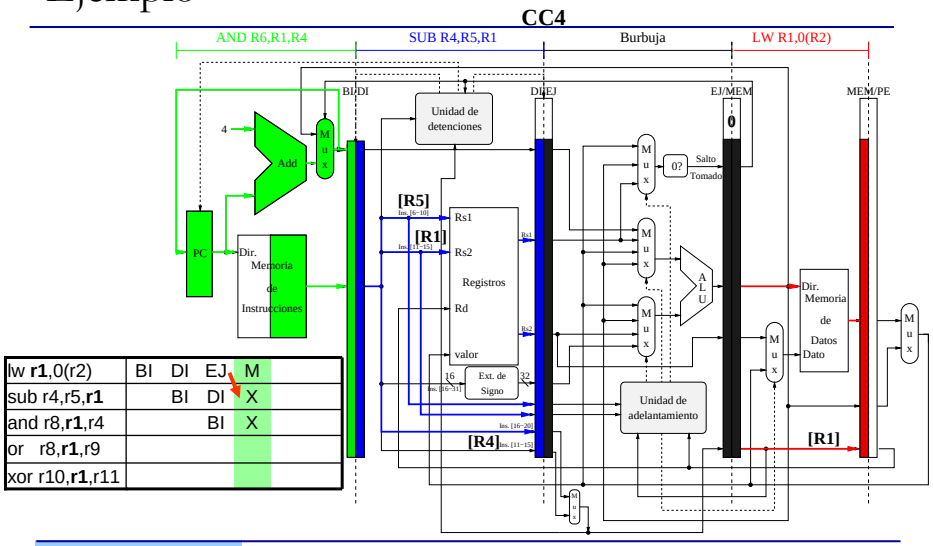


3.1.3 Unidades de anticipación y detención Ejemplo

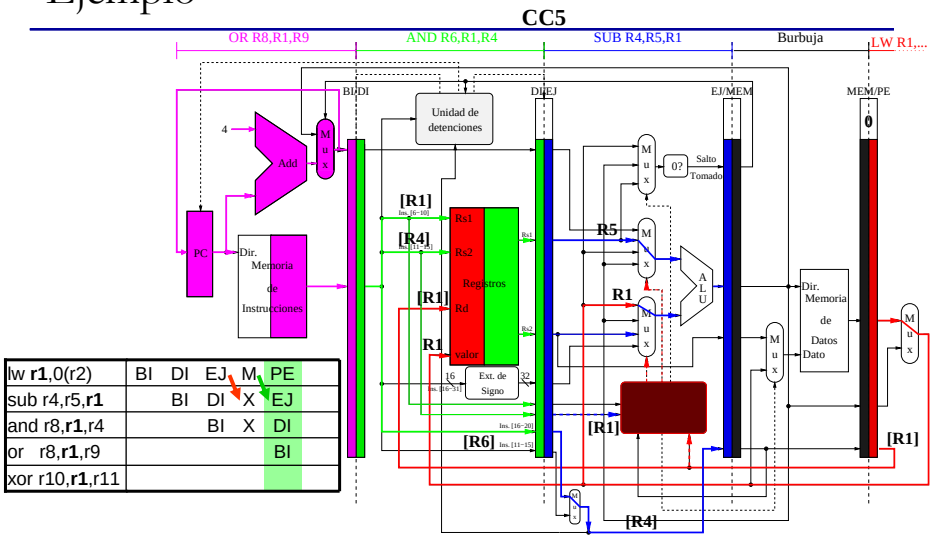


lw **r1**,0(r2)	BI	DI	EJ
sub r4,r5,**r1**		BI	DI
and r8,**r1**,r4			BI
or r8,**r1**,r9			
xor r10,**r1**,r11			

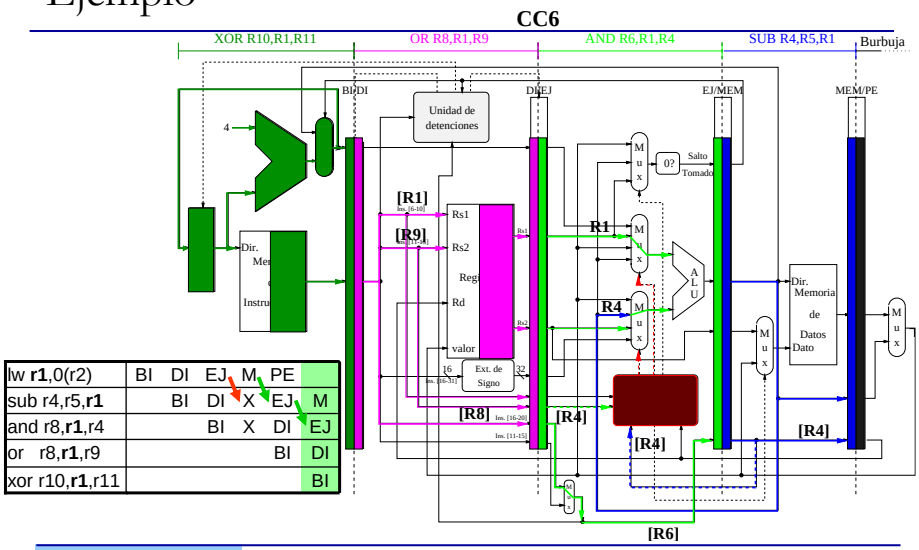
3.1.3 Unidades de anticipación y detención Ejemplo



3.1.3 Unidades de anticipación y detención Ejemplo



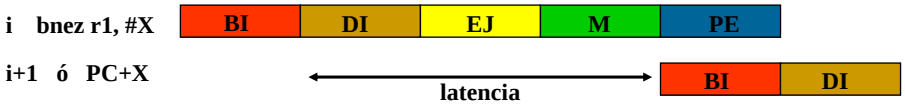
3.1.3 Unidades de anticipación y detención Ejemplo



lw **r1**,0(r2)	BI	DI	EJ	M	PE	
sub r4,r5,**r1**		BI	DI	X	EJ	M
and r8,**r1**,r4			BI	X	DI	EJ
or r8,**r1**,r9					BI	DI
xor r10,**r1**,r11						BI

3.2 Eliminar detenciones en presencia de riesgos de control

- Coste importante de las paradas por dependencias de control
- Instrucción de salto: Bcc R_f, #X ⇨ if (R_f op 0) then PC+X
- Latencia del salto:
  - ✓ Ciclos adicionales entre la BI del salto y su resolución
  - ✓ Ejemplo en la implementación actual del DLX:



- En el DLX (visto hasta ahora) el salto se resuelve en M
  - ✓ Hasta que no se resuelve el salto no podemos buscar la siguiente instrucción
- Llamamos huecos de retardo a los ciclos de latencia (3 en el DLX)
- Bloque básico:
  - ✓ Secuencia de inst. terminada por una inst. que puede modificar el PC

3.2.1 Importancia de los riesgos de control

- Códigos enteros (simbólicos)
  - ✓ 14-24% de saltos condicionales
  - ✓ 1-8% de saltos incondicionales
- Códigos punto flotante (numéricos)
  - ✓ 3-12% de saltos condicionales
  - ✓ 0-2% de saltos incondicionales
- Hay 3.7 veces más saltos condicionales hacia delante que atrás
- Frecuencia de saltos tomados
  - ✓ Saltos hacia atrás tomados un 85%
  - ✓ Saltos hacia delante tomados un 60%
  - ✓ Saltos tomados: el 67% de las veces
- En nuestros ejemplos supondremos a lo largo de este apartado:
  - ✓ 20% de saltos condicionales. 70% de saltos efectivos. El resto: ideal.

SPEC92

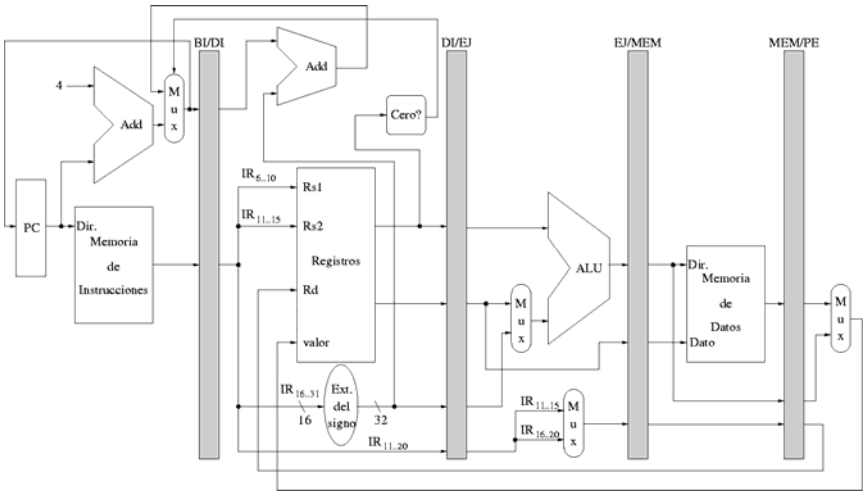
5 códigos simbólicos y 5 códigos numéricos

3.2.1 Importancia de los riesgos de control

- Perdida de rendimiento en el DLX con latencia de salto 3:
  - ✓ La instrucción siguiente al salto tiene CPI_{salto}=4 (se insertan tres burbujas)
  - ✓ Suponiendo un 20% de saltos condicionales:

    CPI=0.2x4 + 0.8x1=1.6 → 60% peor que el ideal (CPI=1)
  - ✓ Pérdida de rendimiento más importante que la debida a riesgos de datos
- Primera solución
  - ✓ Reducir la latencia de salto:
    - Adelantar lo más posible el cálculo de la dirección y el sentido del salto en el cauce de segment.
  - ✓ En el DLX podemos reducir la latencia de salto con un pequeño coste
    - Adelantar el calculo de la dirección de salto añadiendo un sumador dedicado en DI
    - Hacer la comparación con 0 para determinar el sentido del salto en DI
    - Probablemente aumente el Tck (baje la frecuencia):
      - Si la etapa DI era la más lenta (la que determina el Tck), ahora esta etapa va a necesitar más tiempo aun.
    - La etapa DI ahora consume datos: nuevos cortocircuitos EJ-DI y M-DI

3.2.1 Importancia de los riesgos de control



3.2.1 Importancia de los riesgos de control

- Ejemplos de los nuevos cortocircuitos:

add **r1**,r2,r3	BI	DI	EJ	M	PE	
sub r4,r5,r6		BI	DI	EJ	M	PE
beqz **r1**,label1			BI	DI	EJ	M
xor r2,r3,r5				X	BI	DI

lw **r1**,0(r2)	BI	DI	EJ	M	PE	
sub r4,r5,r6		BI	DI	EJ	M	PE
xor r10,r11,r12			BI	DI	EJ	M
benz **r1**,label2				BI	DI	EJ

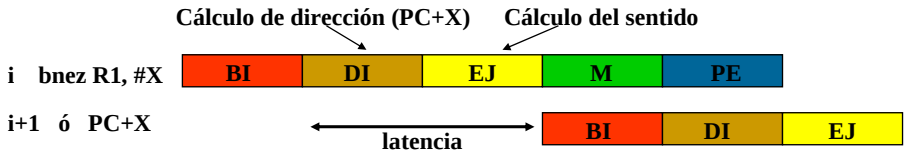
- Aparece un nuevo RAW:
  - ✓ Si el add r1,… fuese un lw r1,… se perderían 2ck en lugar de uno

add **r1**,r2,r3	BI	DI	EJ	M	PE	
benz **r1**,label2		BI	X	DI	EJ	M
xor r10,r11,r12					BI	DI
sub r4,r5,r6						BI

- De todas formas con esta solución hemos mejorado el CPI
  - ✓ La instrucción siguiente al salto tiene CPI_{salto}=2 (se inserta una burbuja)
  - ✓ Suponiendo un 20% de saltos condicionales y sin riesgos de datos:
    CPI=0.2x2 + 0.8x1=1.2 → 20% peor que el ideal (CPI=1)
  - ✓ Mejor que antes con CPI=1.6 pero se puede mejorar significativamente

3.2.1 Importancia de los riesgos de control
Técnicas para mejorar el rendimiento

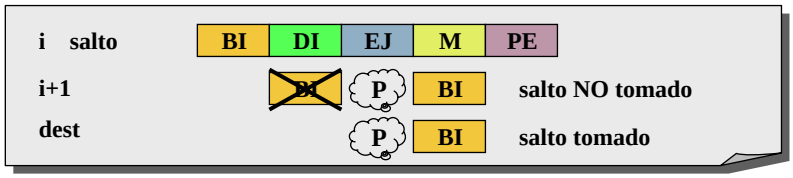
- Salto retardado: el compilador rellena los huecos de retardo
  - ✓ Con ó sin cancelación
- Predicción del sentido
  - ✓ Estática: Salto no tomado/tomado
  - ✓ Semi-estática: Bit en el código de operación
  - ✓ Dinámica: a estudiar en el tema siguiente.
- Para su estudio y comparación entre ellos asumiremos:



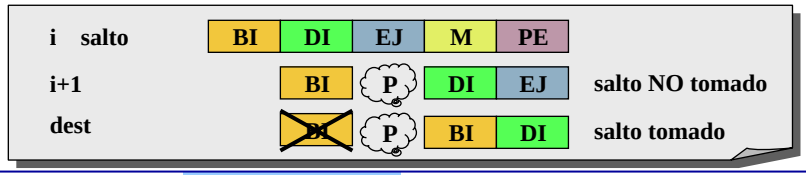
  - ✓ 20% de saltos condicionales. 70% de saltos efectivos. El resto: ideal.

3.2.1 Importancia de los riesgos de control
Rendimiento mínimo: parar siempre

- Estrategia más simple: cada instrucción de salto provoca la pérdida de 2 ck
  - ✓ $CPI_{salto} = 1 + 2 = 3$ $CPI = 0.8x1 + 0.2x3 = 1.4$

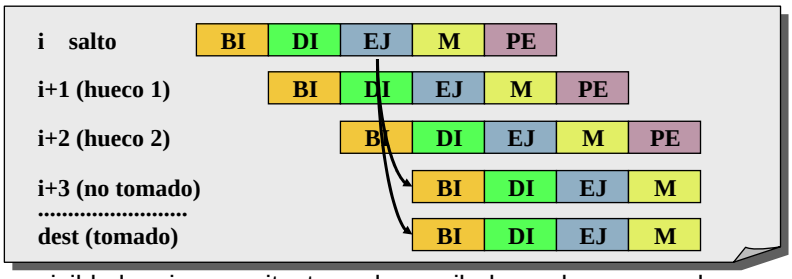


- Mejora evidente: si no salta no cancela BI (sólo pierde 1ck)
  - ✓ $CPI_{salto} = 2x0.3 + 3x0.7 = 2.7$ $CPI = 0.8x1 + 0.2x2.7 = 1.34$



3.2.2 Salto retardado

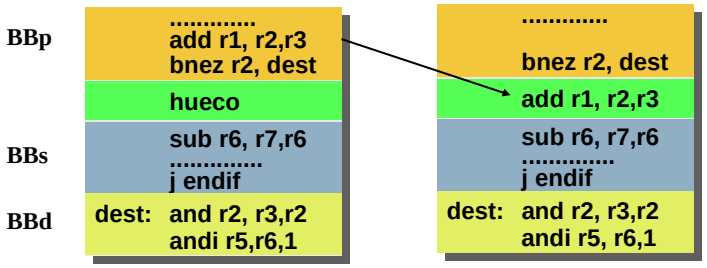
- En vez de parar, el procesador sigue ejecutando instrucciones
- Las instrucciones en los huecos de retardo se ejecutan siempre



- Hace visible la microarquitectura al compilador y al programador
- Buscar instrucciones para rellenar el hueco
  - ✓ Sin modificar la semántica del programa. Si no encuentras: NOP (1ck)
  - ✓ Huecos útiles: 80% el 1° y de ellos, el 25% el 2°.
    - CPIsalto=1 + 0.2x2 + 0.8x0.75x1=2. CPI=0.8x1 + 0.2x2 = 1.2

3.2.2 Salto retardado
Búsqueda de la instrucción de relleno

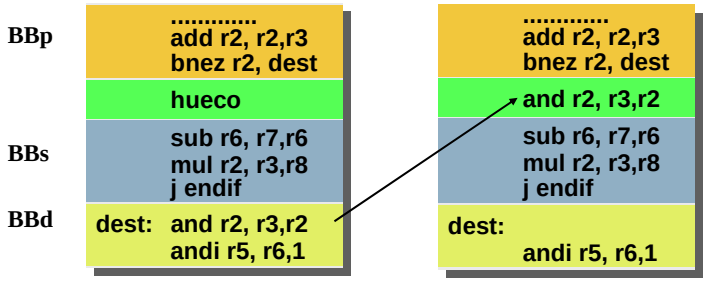
- Tres opciones para buscar la instrucción de relleno
  a) Del propio bloque básico: mejor opción, siempre hace trabajo útil
  b) Del bloque básico destino: si el salto es probable
  c) Del bloque básico siguiente: si el salto no es probable



3.2.2 Salto retardado
Búsqueda de la instrucción de relleno

- Tres opciones para buscar la instrucción de relleno
  a) Del propio bloque básico: mejor opción, siempre hace trabajo útil
  b) Del bloque básico destino: si el salto es probable
  c) Del bloque básico siguiente: si el salto no es probable

No deben modificar la semántica

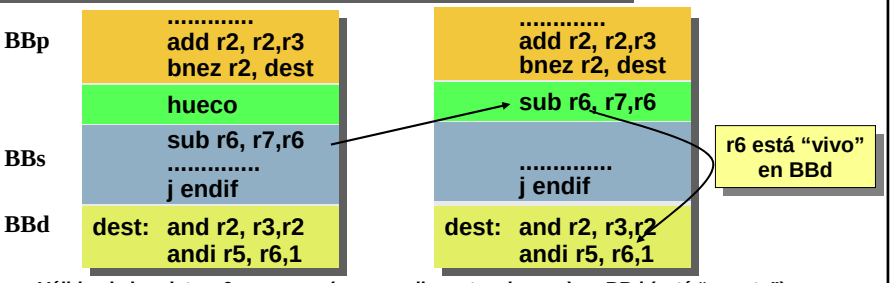


Válido si el registro r2 no se usa (o se escribe antes de usar) en BBs (está "muerto")

3.2.2 Salto retardado
Búsqueda de la instrucción de relleno

- Tres opciones para buscar la instrucción de relleno
  a) Del propio bloque básico: mejor opción, siempre hace trabajo útil
  b) Del bloque básico destino: si el salto es probable
  c) Del bloque básico siguiente: si el salto no es probable

No deben modificar la semántica



Válido si el registro r6 no se usa (o se escribe antes de usar) en BBd (está "muerto")

En este ejemplo vemos que no es posible esta reordenación

3.2.2 Salto retardado
Salto retardado con cancelación

- Limitaciones del salto retardado:
  1. Restricciones para buscar instrucciones de relleno
  2. Habilidad del compilador para predecir el sentido del salto

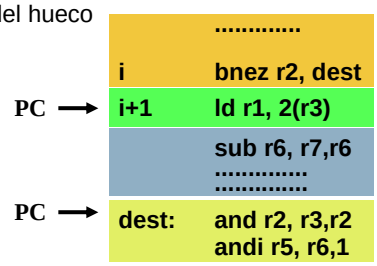
1. Eliminar restricciones en los casos b) y c) : cancelación

- La instrucción de salto contiene información sobre la predicción
  - ✓ Si el salto actúa de la forma predicha la instrucción del hueco se ejecuta
  - ✓ Si no, la instrucción del hueco se cancela: se convierte en un NOP
- Ejemplo (un único hueco de retardo). 20% de saltos cond.
  - ✓ De ellos el 75% son saltos retardados sin cancelación. 100% huecos útiles
  - ✓ El 25% restante son retardados con cancelación y 35% de cancelaciones
    - Sin cancel.: CPI_{salto}=1+0.25x1=1.25, CPI=0.8x1+0.2x1.25=1.05
    - Con cancel.:CPI_{salto}= 1+0.25x0.35x1=1.08, CPI =0.8x1+0.2x1.08 = 1.017

2. Técnicas de predicción estática

3.2.2 Salto retardado
Inconvenientes del salto retardado

- Expone un aspecto de la arquitectura que puede cambiar
- Necesitamos guardar dos PCs en caso de interrupción
  - ✓ Una interrupción ocurre en la instrucción del hueco
  - ✓ El salto ya se ha ejecutado
  - ✓ El salto es efectivo
  - ✓ Las instrucciones i+1 y dest. no son consecutivas
  - ✓ Al retornar de la RTI hay que ejecutar i+1 y luego dest.



- En la práctica sólo se intenta rellenar un hueco
- En arquitecturas supersegmentadas no es suficiente
  - ✓ Ejemplo: Pentium 4 “hipersegmentado” (20 etapas) resuelve el salto en la etapa 17!!.

3.2.3. Predicción estática

- Siempre predice el mismo sentido del salto
- Dos alternativas:

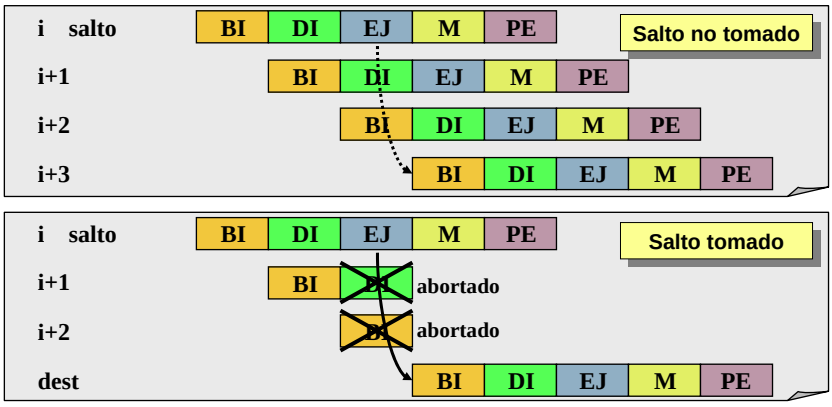
Fijado por el hardware

- Predecir siempre tomado
  - Más probable
  - Si conoces antes la dirección que el sentido
- Predecir siempre NO tomado
  - Más improbable
  - Simple de implementar

Fijado por el compilador

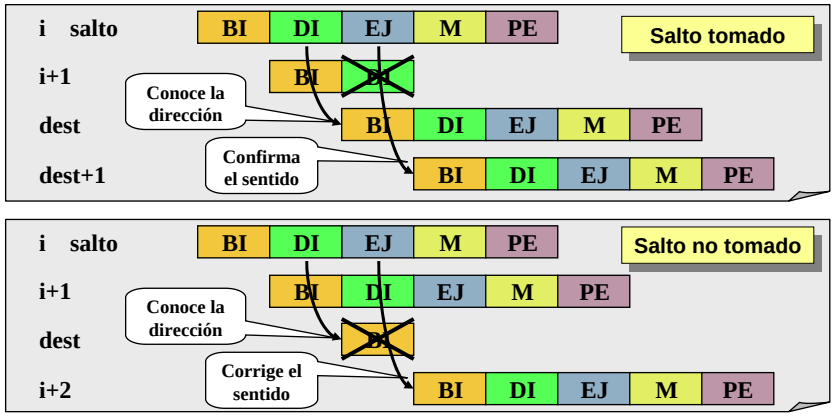
- Un bit en el código de operación
- Distinguir saltos atrás/adelante
  - hacia-atrás-tomado
  - hacia-delante-no-tomado
- Usar perfiles de ejecuciones anteriores
  - con los mismos datos
  - con distintos datos

3.2.3 Predicción estática
Salto NO tomado



- Si aciertas no hay penalización y si no aciertas pierdes 2ck
- Ejemplo: 20% de saltos. 70% de salto tomado.
  - ✓ CPI_{salto} = 0.7x3 + 0.3x1 = 2.4 CPI = 0.8x1 + 0.2x2.4 = 1.28

3.2.3 Predicción estática
Salto tomado



- Si aciertas pierdes un ck y si no también
- Ejemplo: 20% de saltos. 70% de salto tomado.
  - CPI_{salto}= 0.7x2 + 0.3x2 = 2 !! CPI = 0.8x1 + 0.2x2 = 1.2

3.3 Tratamiento de las excepciones
Clasificación de las interrupciones

- Problema:
  - El solapamiento de las instrucciones dificulta el saber si una instrucción puede cambiar el estado de la máquina sin peligro
- Clasificación de las interrupciones
  - Int. SW o SVC (llamadas al supervisor). Síncronas
    - Son realmente llamadas a rutinas del SO: system calls (mediante “trap” o “int”)
    - El vector de interrupción lo proporciona la instrucción
  - Int HW: activan una patilla del micro (int). Asíncronas
    - De E/S: controladores de periféricos, DMA, ...
    - De reset: no enmascarable. Una patilla diferente (reset)
    - De reloj: Permite que el SO se ejecute periódicamente
    - El vector de interrupción lo proporciona el dispositivo que interrumpe (por lo general)
  - Excepciones: generadas dentro del procesador
    - SW: división por cero, overflow, CO ilegal, fallo de página,...
    - HW: sobrecalentamiento, pico de tensión, paridad
    - El vector de int. lo proporciona el procesador

3.3 Tratamiento de las excepciones

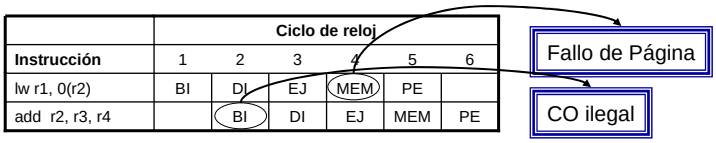
- No hay problema con las interrupciones (HW o SW)
  - ✓ Se pueden tratar como un salto a subrutina
  - ✓ Se dejan terminar las instrucciones anteriores
  - ✓ No se buscan ni ejecutan instrucciones posteriores hasta que se conoce la dirección de salto (a la syscall o RTI)
  - ✓ El retorno (de syscall o RTI) es "equivalente" a un return
- Si puede haber problemas con algunas excepciones:
  - ✓ Las que ocurren en medio de la ejecución de una instrucción
  - ✓ Y además deben ser recomenzables
  - ✓ Ejemplos:
    - Fallo de página en un sistema con memoria virtual
      - El SO debe traer la página del disco duro a memoria y recomenzar la instrucción que falló
    - Excepciones de ejecución punto flotante
      - Por ejemplo, tras una división por 0, el programador puede haber configurado una rutina de tratamiento (manejador de excepción) que cambie el 0 por 1 y reanude la ejecución.

3.3.1 Excepciones precisas Instrucciones enteras

- El tratamiento de las excepciones en un procesador segmentado ha de ser idéntico al de un procesador escalar (no segmentado).
- Excepciones precisas en procesadores segmentados:
  - ✓ Las excepciones han de ser tratadas en el orden de ejecución de las inst.
  - ✓ Las instrucciones anteriores al fallo se completan y las posteriores se reinician
    - Si una instrucción produce excepción, las posteriores no deben modificar el estado (escribir)
- Pasos en el tratamiento de las excepciones:
  1. Forzar una instrucción de TRAP en el ciclo siguiente de búsqueda
  2. Eliminar todas las escrituras para la instrucción que causó la interrupción y las posteriores (las anteriores sí pueden completarse).
  3. Guardar el PC de la instrucción que se interrumpió (o los PCs si salto retardado) y pasar el control a la rutina de gestión de la interrupción.
  4. Una vez completada dicha rutina, recomenzar la ejecución de la instrucción interrumpida (si es un salto, volver a evaluar la condición de salto)

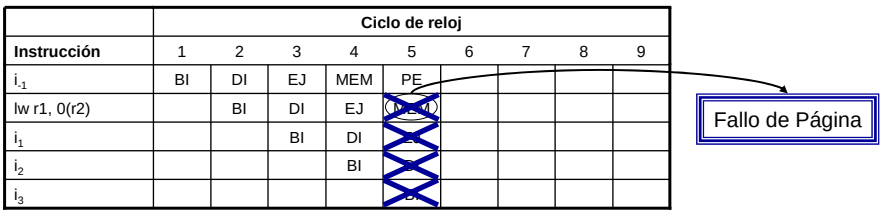
3.3.1 Excepciones precisas
Instrucciones enteras

- En el diseño del DLX:
  - ✓ Las excepciones pueden ocurrir en todas las etapas salvo en PE:
    - BI: fallo de página, dirección no alineada, violación de permisos
    - DI: código de operación ilegal
    - EJ: excepción aritmética, overflow, división por cero
    - MEM: fallo de página, dirección no alineada, violación de permisos
  - ✓ Las escrituras se realizan al final de cauce (en PE o MEM si es un sw)
- Posibilidad: no atender las excepciones en orden
  - ✓ Si lw falla en MEM y add falla en BI → la excepción de add aparece antes!
  - ✓ Solución: Cuando se produce la excepción se apunta con un bit en el registro de segmentación. El bit se propaga y se atiende en PE (en orden)



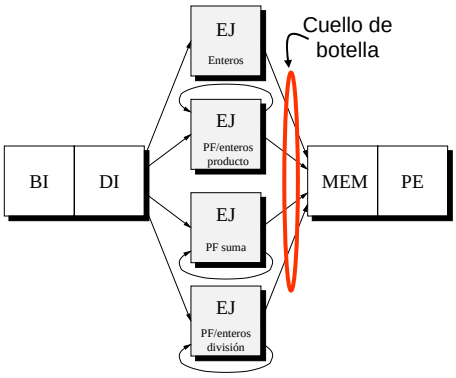
3.3.1 Excepciones precisas
Instrucciones enteras

- Por tanto: en el DLX las excepciones son precisas
  - ✓ Las escrituras se realizan al final de cauce (en PE o MEM si es un sw)
  - ✓ Cuando se produce una excepción en alguna etapa…
    - Ninguna de las instrucciones siguientes en la ejecución ha modificado todavía el estado del procesador
    - Es decir, ninguna de las instrucciones siguientes ha escrito (ni en registros ni en memoria)
    - De esta forma, solo hay que cancelar la instrucción actual y las instrucciones siguientes en cuanto se produzca una excepción
- Ejemplo: fallo de página en un lw en la etapa MEM:



3.4 Tratamiento de las instrucciones multiciclo

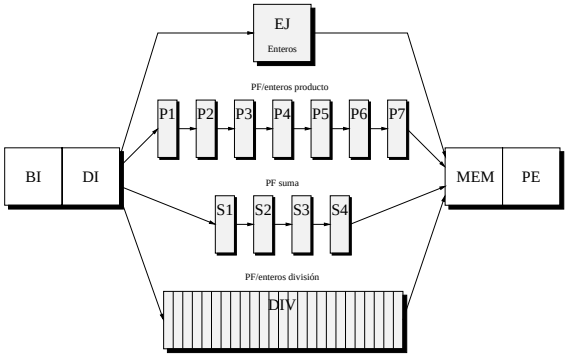
- Ciertas operaciones no pueden ejecutarse en 1 ciclo de reloj
  - ✓ Por ejemplo, operaciones en punto flotante (PF)
- Necesidad de HW adicional para la fase EJ y banco de registros PF



- Número de ciclos de las UF:
  - ✓ ALU enteros (EJ): 1ck
  - ✓ Multiplicador PF/Ent (P1-P7): 7ck
  - ✓ Sumador PF (S1-S4): 4 ck
  - ✓ Divisor PF/Ent (D): 25 ck
- Por tanto: las inst. pueden acabar en orden distinto del orden de comienzo
- Las UFs puede o no estar segmentadas
- El multiplicador y divisor procesan tanto instrucciones enteras como PF.

3.4 Tratamiento de las instrucciones multiciclo Latencia e intervalo de iniciación

- Dos instrucciones consecutivas i y j. Además j depende de i (dep. de datos)
  - ✓ **Latencia**: nº de burbujas necesarias en la ejecución de la instrucción j
- Dos instrucciones consecutivas i y j que usan la misma UF (dep. estructural)
  - ✓ **Intervalo de iniciación**: nº de ciclos necesarios entre la emisión de i y la emisión de j



Inst.	Laten.	Interv. iniciac.
EJ	0	1
Suma	3	1
Multip.	6	1
Divis.	24	25
Load	1	1

3.4 Tratamiento de las instrucciones multiciclo Problemas derivados:

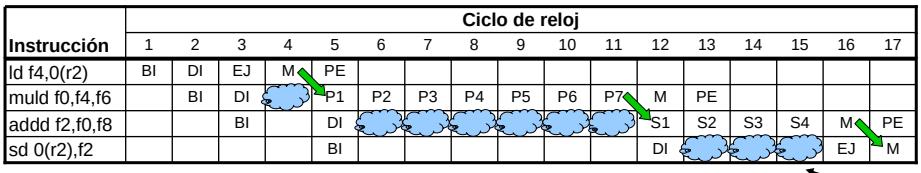
- 1. Riesgos estructurales
  - ✓ Acceso a unidades funcionales no segmentadas
- 2. Mayor penalización y pérdida de rendimiento por riesgos RAW
  - ✓ Debido a la mayor latencia de las unidades funcionales PF
- 3. Problemas derivados de la terminación fuera de orden
  - ✓ 3.a) Riesgos estructurales en el acceso a las etapas MEM y PE
  - ✓ 3.b) Riesgos WAW
  - ✓ 3.c) Problemas para asegurar excepciones precisas

3.4.1 Riesgos estructurales

- Las UFs no segmentadas implican un intervalo de iniciación > 1
- En nuestro ejemplo el divisor no está segmentado y consume 25ck
  - ✓ Intervalo de iniciación = 25
  - ✓ Si quieres evitar el riesgo estructural
    - Planificación SW: separar dos instrucciones de división de forma que existan 25 ck entre ellas
- Soluciones HW
  - ✓ Replicar el número de divisores
    - Si tienes dos divisores: la 2ª div. tiene interv. de iniciación 1, pero la 3ª vuelve a producir riesgo
    - Hacen falta 25 divisores para que “n” divisiones seguidas no produzcan riesgo estructural
  - ✓ Segmentar el divisor
    - Reduce el intervalo de iniciación a 1 → no produce riesgos estructurales
    - La segmentación tiene sus costes: nº de transistores y aumento de la latencia de la división
    - Si la operación de división es poco frecuente la ley de Amdhal recomienda no segmentar

3.4.2 Mayor impacto de los riesgos RAW

- Detección de riesgos RAW en el código
  - ✓ Riesgos RAW entre instrucciones enteras (usan r0-r31)
  - ✓ Riesgos RAW entre instrucciones PF (usan f0-f31)
  - ✓ Ojo a instrucciones que usan registros enteros y PF
    - Movimiento entre registros enteros y PF: `movfp2i r2,f4; movi2fp f6,r0`
    - Load y Store de registros PF: `lf f6,34(r2), sd 8(r5),f10`
- Nº de detenciones mayor y necesidad de nuevos cortocircuitos



Esta burbuja evita el riesgo estructural en el registro de segmentación EJ/MEM

En caso de ampliar los reg. de seg. EJ/MEM y MEM/PE para que puedan pasar dos instr. a la vez se puede quitar esa burbuja y no hay riesgo estructural

3.4.3 Problemas derivados de la terminación fuera de orden

- Ejemplo de operaciones multiciclo independientes
  - ✓ Sufijos F o D para indicar operaciones en simple precisión/doble precisión.

MULD	BI	DI	*P1*	P2	P3	P4	P5	P6	**P7**	MEM	PE	
ADDF		BI	DI	*S1*	S2	S3	**S4**	MEM	PE			
LD			BI	DI	*EJ*	**MEM**	PE					
SF				BI	DI	*EJ*	*MEM*	PE				
DIVF					BI	DI	*D1*	D2	D3	D4	D5	·····

- ✓ Etapa en subrayado: consume dato
- ✓ Etapa en negrita: produce dato

- Problemas derivados de la terminación fuera de orden
  - ✓ A) Conflictos por acceso simultáneo a las etapas MEM y PE
  - ✓ B) Riesgos WAW
  - ✓ C) Problemas para asegurar excepciones precisas

3.4.3 Terminación fuera de orden
A) Acceso simultáneo a MEM y PE

Instrucción	Ciclo de reloj										
	1	2	3	4	5	6	7	8	9	10	11
muld f0,f4,f6	BI	DI	P1	P2	P3	P4	P5	P6	P7	MEM	PE
...		BI	DI	EJ	MEM	PE					
...			BI	DI	EJ	MEM	PE				
addd f2,f4,f6				BI	DI	S1	S2	S3	S4	MEM	PE
...					BI	DI	EJ	MEM	PE		
...						BI	DI	EJ	MEM	PE	
ld f8,0(r2)							BI	DI	EJ	MEM	PE

- A-1) Riesgo estructural en el ck 10 (por el reg. de seg. EJ/M)
  - ✓ Sólo `ld` usa la Mem, pero en el reg. de seg. EJ/M sólo cabe una instrucción
  - ✓ Se puede ampliar EJ/M y M/PE para permitir el paso a más instrucciones
- A-2) Riesgo estructural en el ck 11 (por el banco de registros)
  - ✓ Debido a que el banco de reg. sólo tiene un puerto de lectura
  - ✓ Tres inst. intentan escribir en el mismo ck en tres reg. distintos

3.4.3 Terminación fuera de orden
A-1) Riesgo estructural en reg. de segmentación

- En el registro EJ/MEM se almacena
  - ✓ Valor obtenido de la ALU, dir. del reg. destino y señales de control
- En el registro MEM/PE se almacena
  - ✓ Valor obtenido de la ALU o de MEM, dir del reg. destino y señales de control
- Para permitir que varias instrucciones pasen al mismo tiempo
  - ✓ Extender los reg. de segmentación para almacenar la info de varias inst.
  - ✓ Ejemplo de la transparencia 43 sin riesgo estructural en los reg. de segment.:

Instrucción	Ciclo de reloj																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ld f4,0(r2)	BI	DI	EJ	M	PE												
muld f0,f4,f6		BI	DI		P1	P2	P3	P4	P5	P6	P7	M	PE				
addd f2,f0,f8			BI		DI							S1	S2	S3	S4	M	PE
sd 0(r2),f2					BI							DI			EJ	M	PE

addd no usa la MEM y si usa el banco de registros. sd usa la MEM pero no el banco

3.4.3 Terminación fuera de orden
A-2) Riesgo estructural en PE

- Un solo puerto de escritura: riesgo estructural en el banco de registros
- Solución: detener en DI las instrucciones que produzcan conflicto
  - ✓ Un reg. de desplazamiento indica cuando una inst. va a acceder al banco
  - ✓ Para nuestro DLX basta con un reg. de despl. a la derecha de 27 bits
- Funcionamiento:
  - ✓ Cuando una instrucción se decodifica sabe cuantos ciclos le faltan para llegar a PE
  - ✓ Llamemos d a ese número de ciclos que faltan para llegar a PE
    - d=27 para la división, d=9 para la mult, d=6 para la suma y d=3 para el resto
  - ✓ El registro desplaza a la derecha una posición por cada ciclo de reloj
  - ✓ Cada instrucción que se decodifica chequea el bit d del registro de desplazamiento
    - Si esta a 0, lo pone a 1
    - Si esta a 1, inserta una burbuja y espera al ciclo siguiente

3.4.3 Terminación fuera de orden
A-2) Riesgo estructural en PE

- Ejemplo:
  - ✓ En el ck 2, se decodifica mulf y pone un 1 en la posición 9 (había un 0)
  - ✓ En el ck 3
    - el uno se desplaza a la posición 8
    - or se decod. y escribe un uno en la pos. 3
  - ✓ En el ck 4
    - desplaz. y escritura en 3 del slt
  - ✓ En el ck 5
    - desplaza → hay un 1 en la pos 6
    - mulf llega a PE en 6 ck
    - addf ve que ya hay un 1 en 6 así que espera un ck.

pos / ck	9	8	7	6	5	4	3	2	1
2	1								
3		1					1		
4			1				1	1	
5				1				1	1
6				1	1				1

Instrucción	Ciclo de reloj											
	1	2	3	4	5	6	7	8	9	10	11	12
muld f0,f4,f6	BI	DI	P1	P2	P3	P4	P5	P6	P7	MEM	PE	
or		BI	DI	EJ	MEM	PE						
slt			BI	DI	EJ	MEM	PE					
addd f2,f4,f6				BI	DI		S1	S2	S3	S4	MEM	PE

3.4.3 Terminación fuera de orden
A-2) Riesgo estructural en PE

- Implica implementar el reg. de desplazamiento y la gestión adecuada
- Otra posiblidad: detener las instrucciones conflictivas al final de EJ
  - ✓ En ese punto ya sabes que queda un ck para llegar a PE

Instrucción	Ciclo de reloj											
	1	2	3	4	5	6	7	8	9	10	11	12
muld f0,f4,f6	BI	DI	P1	P2	P3	P4	P5	P6	P7	MEM	PE	
or		BI	DI	EJ	MEM	PE						
slt			BI	DI	EJ	MEM	PE					
addd f2,f4,f6				BI	DI	S1	S2	S3	S4		MEM	PE

- Inconvenientes
  - ✓ Necesidad de establecer prioridades de acceso
    - Por ejemplo: dar mayor prioridad a la unidad con mayor latencia
  - ✓ Ahora tenemos lógica de detenciones en dos etapas del cauce
    - En la etapa DI y en la etapa anterior a MEM

3.4.3 Terminación fuera de orden
B) Riesgos WAW

- Las escrituras en desorden pueden provocar *riesgo WAW*

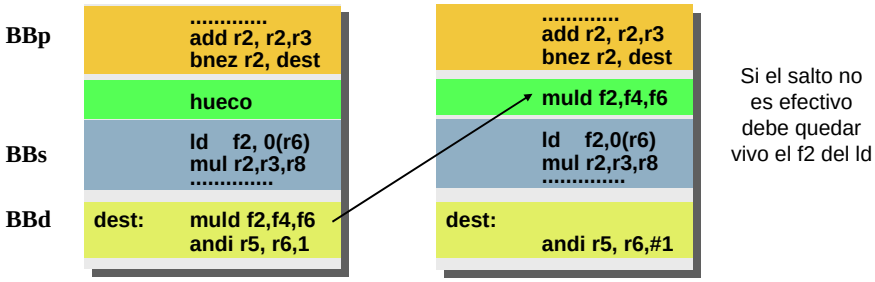
	Ciclo de reloj										
Instrucción	1	2	3	4	5	6	7	8	9	10	11
muld f0,f2,f6	BI	DI	P1	P2	P3	P4	P5	P6	P7	M	PE
ld f6,0(r1)		BI	DI	EJ	M	PE					
addd f0,f4,f8			BI	DI	S1	S2	S3	S4	M	PE	

- Situación poco común: la instrucción muld es inútil
  - ✓ Típicamente, un dato que se escribe en un regist. se usa antes de rescribirlo
  - ✓ Es decir, habría un riesgo RAW que cancelaría el riesgo WAW

	Ciclo de reloj														
Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
muld f0,f2,f6	BI	DI	P1	P2	P3	P4	P5	P6	P7	M	PE				
sd 0(r1),f0		BI	DI						EJ	M	PE				
addd f0,f4,f8			BI						DI	S1	S2	S3	S4	M	PE

3.4.3 Terminación fuera de orden
B) Riesgos WAW

- De todas formas se puede dar WAW por ejemplo en salto retardado:



- Por tanto hay que tratarlos
  - ✓ Detener la instrucción que produce el riesgo (ld en el ejemplo)
  - ✓ Cancelar la escritura de la primera instrucción (muld en el ejemplo)
- Ya que la situación es poco común: cualquier solución es buena.

3.4.3 Terminación fuera de orden
B) Riesgos WAW

- Los riesgos WAR no se producen en el DLX
  - ✓ Aunque tengamos instrucciones multiciclo
  - ✓ Y por tanto instrucciones que acaban en desorden
- No hay riesgos WAR porque
  - ✓ Las lecturas se realizan al principio del cauce
  - ✓ Las escrituras se realizan al final del cauce

	Ciclo de reloj										
Instrucción	1	2	3	4	5	6	7	8	9	10	11
muld f0,f2,f6	BI	DI	P1	P2	P3	P4	P5	P6	P7	M	~~PE~~
ld f6,0(r1)		BI	DI	EJ	M	PE					
addd f0,f4,f8			BI	DI	S1	S2	S3	S4	M	PE	

no riesgo WAR

PE cancelada para evitar WAW con addd

3.4.3 Terminación fuera de orden
C) Asegurar excepciones precisas

- Excepciones precisas
  - ✓ Se atienden en el orden de ejecución de las instrucciones
  - ✓ Las instrucciones anteriores continúan, las posteriores no se completan
- Con instrucciones multiciclo tenemos terminación fuera de orden
  - ✓ Un instrucción PF puede generar una excepción cuando ya han terminado instrucciones posteriores → **Excepciones imprecisas**
  - ✓ Ejemplo:
    - divd puede generar una excepción en el ck10 cuando ya ha terminado las inst. posterior addd.
    - El estado del procesador ha cambiado (el valor de f2) al terminar addd
    - Al recomenzar divd (tras la RTI), addd se vuelve a ejecutar!!

División por cero

Instrucción	Ciclo de reloj										
	1	2	3	4	5	6	7	8	9	10	11
divd f0, f2, f4	BI	DI	D1	D2	D3	D4	D5	D6	D7	D8	...
addd f2, f2, f6		BI	DI	S1	S2	S3	S4	M	PE		

3.4.3 Terminación fuera de orden
C) Asegurar excepciones precisas

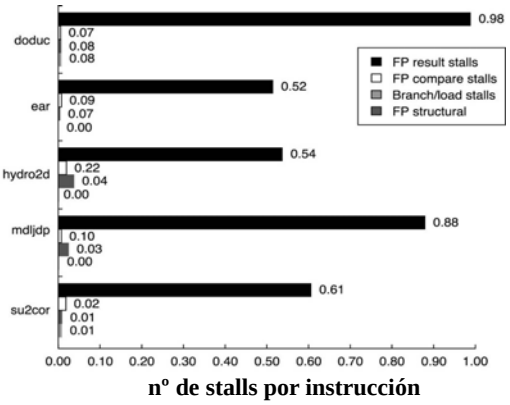
- Soluciones:
  - ✓ Admitir excepciones imprecisas (no es una solución!)
    - Si hay excepciones, el programa puede terminar mal
    - Usado en máquinas antiguas y algunos supercomputadores
    - En máquinas con mem. virtual y unidades PF IEEE-754 no se puede permitir exc. imprecisas
  - ✓ Asegurar la ejecución en un modo preciso:
    - Las inst. emitidas continúan sólo si las anteriores no van a producir excepciones
    - Toda instrucción que potencialmente pueda crear una excepción va a impedir que instrucciones posteriores se completen (aunque no haya dependencia de datos) → Ralentiza el procesador
    - Ejemplos: MIPS R2000, R3000, R4000 y Pentium
  - ✓ Admitir los dos modos de funcionamiento (controlado por un flag o el SO)
    - Modo impreciso (rápido pero inseguro)
    - Modo preciso (lento pero fiable)
    - Ejemplos: Alpha 21064 y 21164, IBM Power-1 y 2 y MIPS R8000

3.4.3 Terminación fuera de orden C) Asegurar excepciones precisas

- Soluciones más modernas: forzar las escrituras en orden
  - ✓ No escribir los resultados hasta que lo hayan hecho las instr. previas
  - ✓ Ninguna inst. modifica el estado mientras existan instr. previas en ejecución
- Métodos:
  - ✓ Cola de valores antiguos o fichero de historia
    - Las instrucciones sí escriben en los registros y memoria
    - Pero se mantiene una copia de lo que tenían en una cola de valores antiguos
    - En caso de excepción de una instr. previa → se recupera el estado de la cola de valores ant.
    - Ejemplo: CYBER 180/990
  - ✓ Cola de valores nuevos o fichero de futuro
    - Las escrituras no se escriben ni en registros ni en memoria, sino en una cola de valores nuevos
    - Cuando las instr. previas han terminado sin excepción se actualizan los registros y la mem.
    - Ejemplos: IBM-Motorola-Apple PowerPC 620, MIPS R10000.
- En los procesadores actuales:
  - ✓ El problema se resuelve gracias al buffer de reordenación. Ver tema 5.

3.4.4 Pérdida de rendimiento por las operaciones multiciclo (ejemplo del DLX)

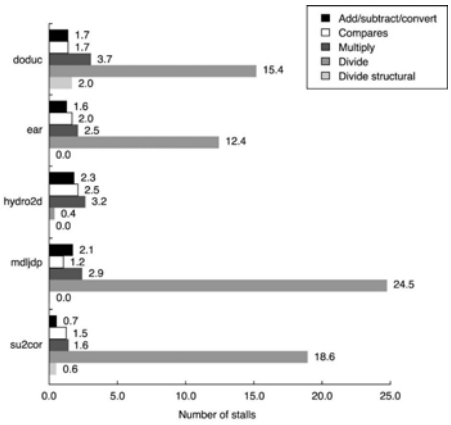
- Tipos de detenciones
  - ✓ RAW por instrucción PF
  - ✓ RAW por comparación (salto condicional)
  - ✓ RAW por load
  - ✓ Riesgo de control
  - ✓ Riesgo estructural
- 5 benchmarks del SPEC89fp
- El nº de detenciones por instr. varía de 0.65 a 1.21ck. De media 0.87ck/inst.
- Riesgos más costosos
  - ✓ RAW PF (0.71ck/inst)
  - ✓ RAW por comp. (0.1ck/inst)



Fuente: Comput. Architecture. Hennessy y Patterson

3.4.4 Pérdida de rendimiento por las operaciones multiciclo

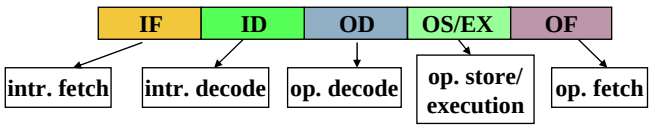
- Número de ciclos de detención para cada tipo de operación PF
  - ✓ El número de detenciones por riesgos RAW depende de la latencia de cada U.F. :
    - la suma pierde de media 1.7 ck
      - 56% de su latencia (3ck)
    - la multiplicación 2.8 ciclos
      - 46% de su latencia (6ck)
    - la división 14.2 ciclos.
      - 59% de su latencia (24ck)
  - ✓ Los riesgos estructurales provocados por la división son bajos (xq hay pocas divisiones)



Fuente: Comput. Architecture. Hennessy y Patterson

3.5 Ejemplos
Primeros procesadores RISC

- RISC I y II. Dave Patterson. Univ. Berkeley
  - ✓ RISC I → ´80; 44.000 TRTs; 8MHz; 2 etapas (BI,EJ)
  - ✓ RISC II → ´80; 41.000 TRTs; 12MHz; 3 etapas (BI,EJ,PE). Salto retardado.
- MIPS. John Hennessy. Univ. Stanford.
  - ✓ MIPS →Microprocessor without Interlocked Pipeline Stages
  - ✓ MIPS → ´81; 24.000 TRTs; 4MHz; 5 etapas
  - ✓ MIPS X → '83;150.000 TRTs; 20MHz; 5 etapas



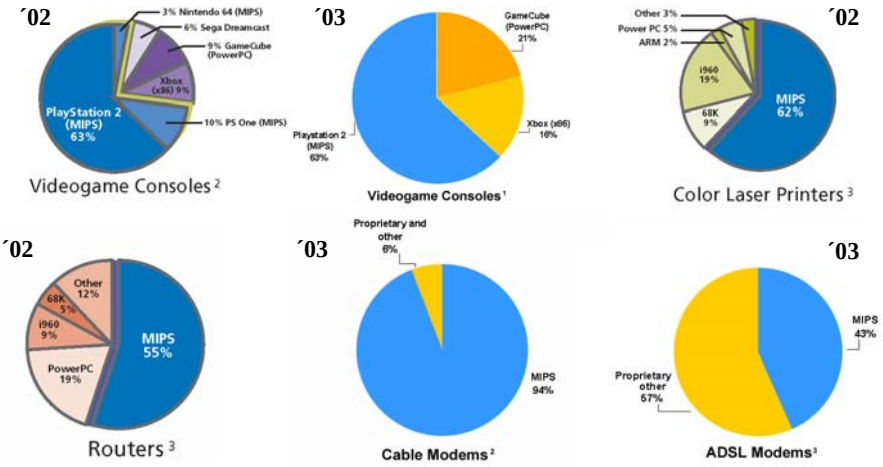
- Operand decode: lectura de operandos del banco de registros
- Operand store: almacenamiento en memoria de un resultado (store)
- Operand fetch: lectura de memoria (load)

3.5 Ejemplos
Descendientes de RISC y MIPS

- Empresa SPARC (Scalable Processor ARChitecture) fundada en el ´89
  - ✓ Arquitectura SPARC V7 basada en el RISC de Dave Patterson
  - ✓ SPARC vende las arquitecturas (V7 de 32b, V8 de 32b y V9 de 64bits)
  - ✓ Existen varios fabricantes: Sun, Fujitsu, Cypress, BIT, TI, LSI, Toshiba,…
  - ✓ Los detalles de implementación dependen del fabricante
    - Por ejemplo: frecuencias en el rango 50-80MHz para el SUN SuperSPARC (V8) del ´93
    - Cuatro etapas (IF, ID, EX, WB). También es superescalar de factor 3 (a estudiar en tema 5)
  - ✓ Más detalles en www.sparc.com
- Empresa MIPS (fundada por John Hennessy en 1984)
  - ✓ Venden las arquitecturas pero no las fabrica (venden la propiedad intelectual)
  - ✓ Algunos fabricantes: AMD, NEC, Philips, Sony, Toshiba, LSI
    - R2000 → ´85; 10.000TRTs; 12MHz; 5 etapas
    - R3000 → ´88; 115.000TRTs; 25MHz; 5 etapas (IF, RD, ALU, MEM, WB)
    - R4000 → ´92; 1.350.000TRTs; 100-200MHz; 8 etapas. Cache de inst. y de datos: 16KB+16KB
  - ✓ SGI (Silicon Graphics Inc.) compra MIPS en 1992
  - ✓ Actualmente vende las arquitecturas MIPS32 y MIPS64

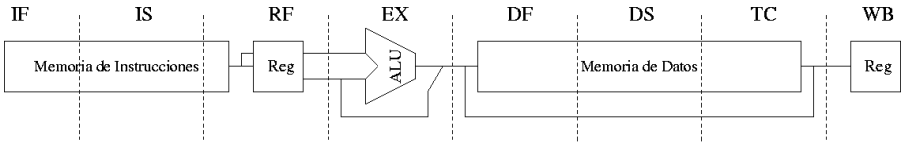
3.5 Ejemplos
Cuota de mercado de procesadores MIPS

Fuente: www.mips.com



3.5 Ejemplos
Procesador supersegmentado: R4000

- Uno de los procesadores de la arquitectura de la Nintendo 64
- Aparece en el '92. Primer procesador de 64 bits. F=100-200MHz
  - ✓ A 150MHz → SPEC92int=59 y SPEC92fp=61
  - ✓ Procesador coetáneo → i486, 33MHz, SPEC92int=29 y SPEC92fp=14
- Procesador con 8 etapas en el cauce de segmentación



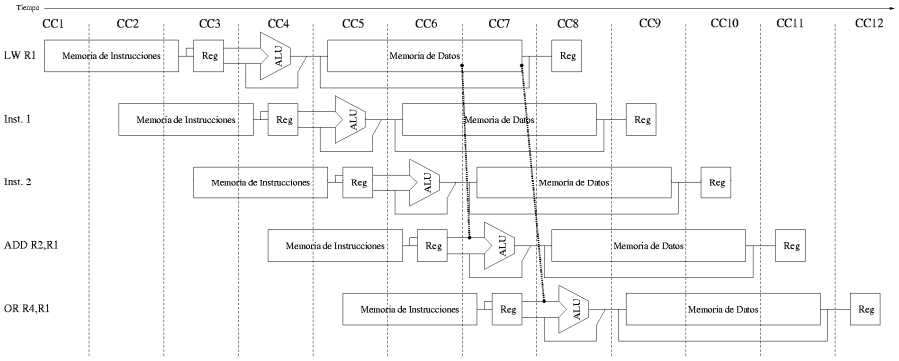
- Accesos a memoria ocupan varios ck, pero están segmentados
- Mayor profundidad en la segmentación implica:
  - ✓ Altas frecuencias de reloj
  - ✓ Más HW para adelantamientos y más retardos por cargas y saltos

3.5 Ejemplos
Procesador supersegmentado: R4000

- Etapas:
  - ✓ **IF**: Primera mitad de búsqueda de instr. Inicia el acceso a cache de instrucciones. Actualiza PC
  - ✓ **IS**: Segunda mitad de BI. Completa el acceso a cache.
  - ✓ **RF**: Comprobación de acierto en cache. Decodificación y lectura de registros. Chequeo de riesgos
  - ✓ **EX**: Ejecución: operación en ALU, cálculo de dirección efectiva, evaluación de condición de salto y de destino de salto. El salto se resuelve por tanto en esta etapa.
  - ✓ **DF**: Inicio de la búsqueda de datos
  - ✓ **DS**: Segunda mitad de búsqueda de datos
  - ✓ **TC**: Chequeo de etiquetas. Determina si hay acierto en cache.
  - ✓ **WB**: Postescritura

3.5 Ejemplos
Procesador supersegmentado: R4000

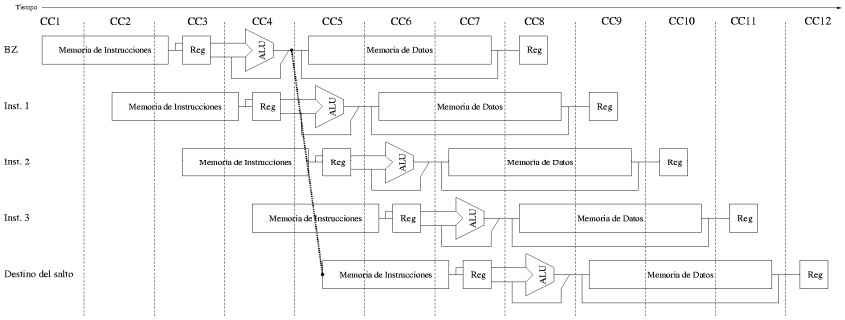
- Las cargas tienen dos ciclos de latencia



- Necesidad de adelantar el resultado de una carga a instrucciones emitidas tres o cuatro ciclos más tarde

3.5 Ejemplos
Procesador supersegmentado: R4000

- La dirección de salto y el sentido se conocen en EX
  - ✓ Latencia de salto de tres ciclos
- Estrategia:
  - ✓ Salto retardado para el primer ciclo
  - ✓ Predicción estática de salto no tomado para lo dos ciclos restantes



3.5 Ejemplos
Procesador supersegmentado: R4000

- Salto no efectivo: no pierdes ningún ciclo si el hueco es útil

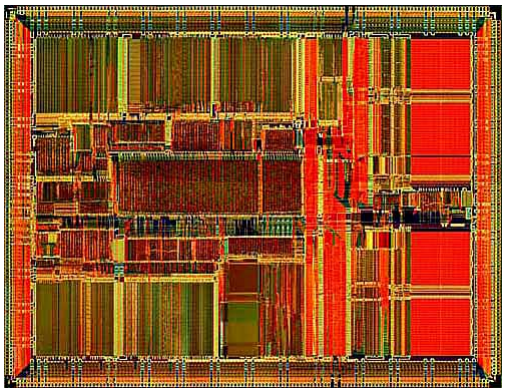
Instrucción	Ciclo de reloj											
	1	2	3	4	5	6	7	8	9	10	11	12
salto	IF	IS	RF	EX	DF	DS	TC	WB				
hueco relleno		IF	IS	RF	EX	DF	DS	TC	WB			
i+2			IF	IS	RF	EX	DF	DS	TC	WB		
i+3				IF	IS	RF	EX	DF	DS	TC	WB	

- Salto efectivo: pierdes 2 ck (más otro si el hueco no es útil)

Instrucción	Ciclo de reloj											
	1	2	3	4	5	6	7	8	9	10	11	12
salto	IF	IS	RF	EX	DF	DS	TC	WB				
hueco relleno		IF	IS	RF	EX	DF	DS	TC	WB			
burbuja			IF	IS	abort							
burbuja				IF	abort							
destino					IF	IS	RF	EX	DF	DS	TC	WB

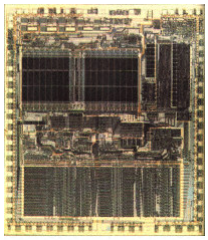
3.5 Ejemplos
Procesador supersegmentado: R4000

- 1.350.000TRTs; 100-200MHz
- 184 mm^2

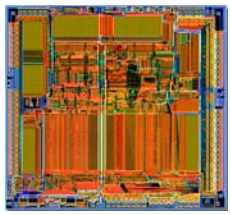


3.5 Ejemplos
Procesadores de Motorola

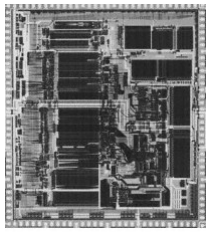
68000: ´79
8-20MHz,
68.000TRTs,
de 16 bits



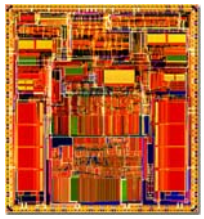
68020: ´84
12-33MHz, 1.5µ
103.000TRTs
de 32 bits



68030: ´87
16-50MHz, 1.2µ
270.000 TRTs
de 32 bits



68040: ´89
25-66MHz, 0.8µ
1.200.000 TRTs
de 32 bits



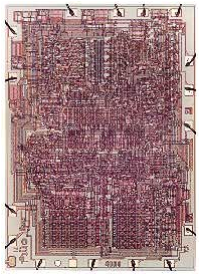
Fuentes: http://micro.magnet.fsu.edu/chipshots/motorola/
http://www.computerhistory.org/ http://www.mot.com

BI DI EA M EJ PE

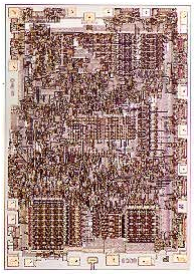
• **El 68060 y los PowerPC ya son superescalares**

3.5 Ejemplos
Procesadores no segmentados de Intel

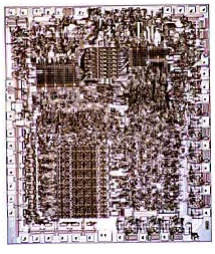
4004: ´71;
0.4MHz, 10µ
2.300 TRTs,
de 4 bits



8008: ´72;
0.8MHZ, 10µ
3.500 TRTs,
de 8 bits

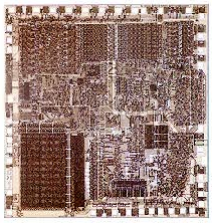
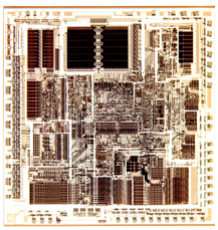
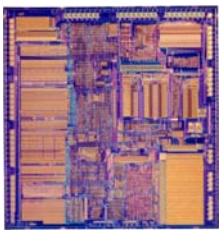
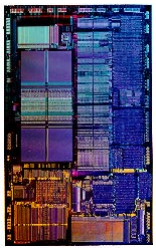


8080: ´74;
2MHz, 6µ
6.500 TRTs,
de 8 bits



Fuente: http://www.intel.com/intel/intelis/museum/exhibits/

3.5 Ejemplos
Procesadores segmentados de Intel

8086: ´78; 4.77-10MHz, 3µ, 29.000 TRTs, de 16b	80286: ´82; 6-12MHz, 1.5µ, 134.000 TRTs, de 16 bits	80386: ´85; 16-33MHz, 1.5-1µ, 275.000 TRTs, de 32 bits	80486: ´89; 25-50MHz, 1-0.8µ, 1.200.000 TRTs, de 32 bits
			
BI EJ	BI EJ	BI DI EJ PE	BI DI EJ M PE

Riesgos de datos | Riesgos de control | Excepciones | Multiciclo | Ejemplos

3.5 Ejemplos

- Desde el Pentium en adelante ya son superescalares
- A estudiar en los temas 4 y 5
  - ✓ Técnicas mejoradas de planificación SW
  - ✓ Técnicas de planificación HW
    - Implementaciones de arquitecturas con ejecución fuera de orden
  - ✓ Técnicas mejoradas para reducir riesgos de control
    - Predicción de salto dinámica
    - Buffer de destino de salto e instrucciones predicadas
  - ✓ Técnicas para reducir el CPI por debajo de 1
    - Procesadores superescalares
    - Procesadores VLIW
  - ✓ Ejemplos de procesadores modernos que implementan todo esto.

Riesgos de datos | Riesgos de control | Excepciones | Multiciclo | Ejemplos

Bibliografía

- J.L HENNESSY, D. PATTERSON. Computer Architecture: a quantitative approach. Ed. Morgan Kaufman,
  - Segunda Edición 1996.
    - Capítulo 3
  - Tercera Edición 2003
    - Apéndice A1
- D.A.PATTERSON, J.L.HENNESSY. Estructura y diseño de computadores. Ed. Reverté. 2000.
  - Capítulo 6