

ARQUITECTURA E INGENIERÍA DE COMPUTADORES

Tema 2.1



Tema 2.1

Concepto de Segmentación

SEGMENTACIÓN

Se basa en **descomponer una tarea en varios subprocessos independientes entre ellos**, de tal manera que puedes hacer varios simultáneamente. *Esto funciona si la tarea a realizar tiene varias fases, separas cada fase y mientras en una fase haces una parte de una tarea, en otros haces otra. Esto reduce el CPI idealmente a 1.*

Periodo de reloj: El tiempo min de reloj es el que hace falta para que se haga la tarea más larga (caso real)

- **Caso ideal:** Mismo retardo en todos módulos. **D**: Retardo del **circuito** original. **k**: **Número de etapas**.

$$\tau = \frac{D}{k}$$

- **Caso real:** Módulos con retardos distintos, registros Inter etapa y desfase del reloj.

t_i : Retardo del **módulo** i. **Tr**: Retardo del **registro** Inter etapa. **Ts**: **Desfase del reloj** entre etapas.

$$\tau = \max_{i=1 \rightarrow k} (t_i) + Tr + Ts \geq \frac{D}{k}$$

Aceleración (Speedup): Aceleración por segmentar en etapas.

$$S = \frac{Tns}{Ts} \approx \frac{nD}{n\tau} = \frac{D}{\tau}$$

Tns: Tiempo para **procesar n datos** en unidad original. **Ts**: Tiempo en la **unidad segmentada**.

- **Caso ideal:** $\tau = D \cdot k \rightarrow S = \frac{D}{\tau} = k$ Nunca tendrás mayor mejora que num etapa.
- **Caso real:** $\tau \geq D \cdot k \rightarrow S = \frac{D}{\tau} \leq k$ De normal tendrás un poco menos siempre.

Productividad: Cantidad de operaciones por unidad de tiempo.

- **Unidad no segmentada:** → 1 resultado cada D segundos
- **Unidad segmentada:** → 1 resultado cada τ segundos = 1 resultado/ciclo de reloj. **CPI = 1 idealmente**.

CICLO Y SEGMENTACIÓN DE INSTRUCCIONES

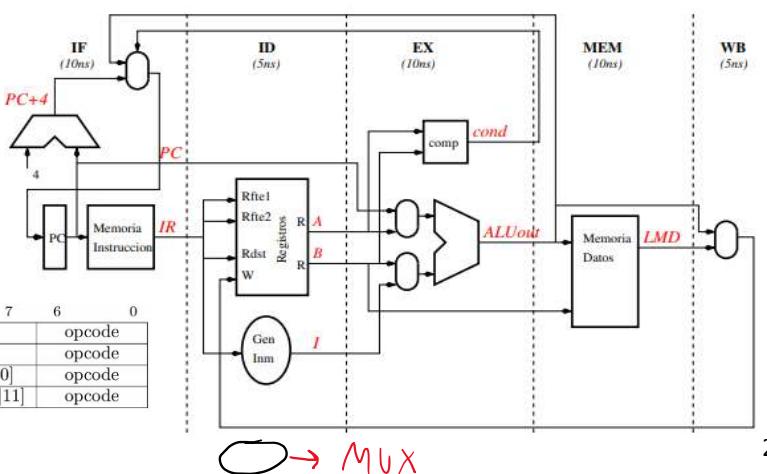
Un ciclo es cada una de las etapas por las que pasa una instrucción para ejecutarse por completo. Cada instrucción es diferente y sigue unos caminos diferentes dentro del circuito segmentado.

Tiempo de ciclo T sin segmentar

Tiempo de la **instrucción más larga** (*pasar por todas las partes*).

Saltos, ual, almac, carga
 $T_{(noSeg)} = \max(25, 30, 35, 40) = 40 \text{ ns}$

R	31	27	26	25	24	20	19	15	14	12	11	7	6	0
I	funct7	rs2		rs1		funct3		rd		opcode				
S		imm[11:0]			rs2	rs1	funct3	rd		opcode				
B		imm[12:10:5]			rs2	rs1	funct3	imm[4:1 11]		opcode				



Tiempo de Ciclo T Segmentado

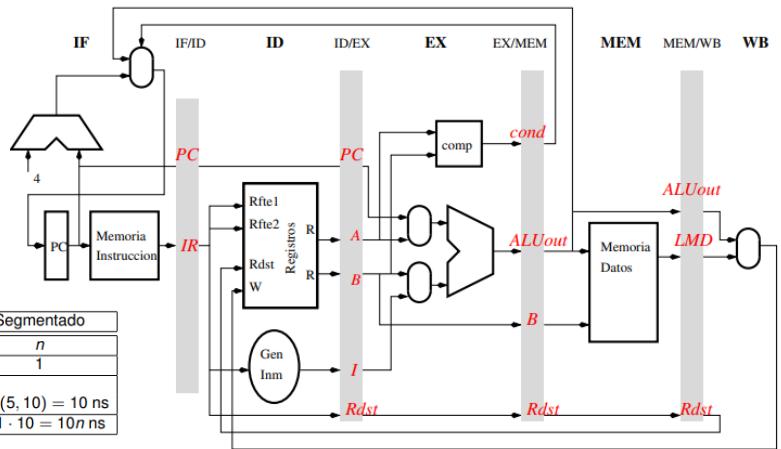
Tiempo de la etapa/ejecución más larga.

$$ID, WB / IF, EX, MEM \\ T = \max(5, 10) = 10 \text{ ns}$$

Aceleración: $S_{\max} = 5$ (ideal) será menor

Antes $T=40$, ahora $T=10 \rightarrow S = 40/10 = 4$

	No segmentado	Segmentado
I	n	n
CPI	1	1
T	$\max(25, 30, 35, 40) = 40 \text{ ns}$	$\max(5, 10) = 10 \text{ ns}$
$T_{ej} = I \cdot CPI \cdot T$	$n \cdot 1 \cdot 40 = 40n \text{ ns}$	$n \cdot 1 \cdot 10 = 10n \text{ ns}$



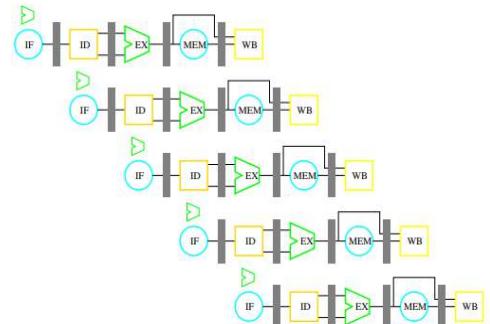
Requisitos Hardware

Del mismo color los segmentos que acceden al mismo recurso.

Azul: IF y MEM podrían dar conflicto si en la **cache** de datos y memoria estuvieran unificadas. *Como de normal están SEPARADAS* ta oke

Verde: Operaciones aritméticas, se usan **dos alu diferentes** y **yau**

Naranja/Amarillo: Naranja lee, Amarillo escribe en los registros, se hace que en la **primera mitad** del ciclo de reloj se **escriba (WB)** y en la **otra mitad se lea (ID)**.



Riesgos: Resultados diferentes de ejecutar en Segmentada y no Segmentada

Tipos de riesgos

- Datos:** El resultado de una instrucción se utiliza como dato en la(s) **instrucción(es) siguiente(s)**.
- Control:** En el flujo de instrucciones **aparece** una instrucción de **salto**. Ya estaba ejecutando cosas y salto.
- Estructurales:** Dos o más **instrucciones** pretenden **utilizar el mismo recurso**. Ej cache de datos unificada.

Solución 1 Ciclos de parada

Inserción de ciclos de parada, Impedir el avance de la instrucción que origina el conflicto y de todas las que le siguen, dejó que la estaba primera acabe mientras retraso a la segunda hasta que la primera salga y quite el posible conflict.

Durante estos ciclos no se buscan nuevas instrucciones (*ciclos de parada o stalls*). Hay una pérdida de prestaciones:

$$CPIs = CPI_{ideal} + \text{Ciclos de parada} = 1 + \frac{\text{stalls}}{\text{instrucción}} > 1$$

Solución 2 Modificación de la ruta de datos

Modificación de la ruta de datos para detectarlos y resolverlos dinámicamente. La modificación puede Reducir el número de ciclos de parada necesarios para resolverlo o Resolver el riesgo completamente (a veces no es posible).

Solución 3 Trabajo por el compilador “instrucciones NOP”

Modificación de la arquitectura del juego de instrucciones, Impedir que aparezca el problema prohibiendo la generación de ciertas secuencias de instrucciones (*que sabes que van a dar error*). Para evitarlas, el compilador puede **insertar instrucciones NOP** o reordenar instrucciones independientes. Se pierde la compatibilidad a nivel binario.

RIESGOS DE DATOS

Ejemplo causa: La 1r instrucción escribe en un registro, pero lo hace en el ciclo 5. La instrucción 2 y 3 *leen* este mismo registro en el ciclo 3 y 4, *antes de que se escriba* → **Inconsistencia**.

Sol 1: Inserción de ciclos de parada (stalls)

Se inserta 1 o 2 ciclos de parada para que avance la instrucción hasta que escriba y ya se pueda leer.

Control: Se han de poner las señales de control antes de la etapa EX para que retengan el flujo como si de una instrucción NOP se tratara. Lo que hace es **Conservar las instrucciones en IF y ID**.

- Señales:** Con las señales **ID.stall** y **ID.nop** se detiene el paso de las instrucciones y se meten las **nop**. Luego el **IF.stall** evita que el PC siga avanzado.

CPI: Ahora el **CPI aumenta** porque hay más ciclos para el mismo número de instrucciones, intentar NO ponerlos.

Sol 2: Cortocircuitos

En verdad el resultado de las **operaciones se sabe nada más salir de la alu** (*fin ciclo 3*), y en verdad la **instrucción que lee** lo usa **antes de entrar a la alu** (*inicio ciclo 4*). Por lo que podemos **hacer un cortocircuito para pasar este dato de uno a otro** (*de MeM a EX*). Es de MeM a EX y no de EX a EX xq el dato ya habrá avanzado una fase

Pero eso es si las instrucciones son consecutivas, si hay otra en medio, el dato que se va a escribir ya va por WB, por lo que también tienes que hacer un cortocircuito entre MeM y WB.

ES ATRAPAR EL DATO PARA DESPUÉS USARLO Y NO SE METER CICLOS EXTRA

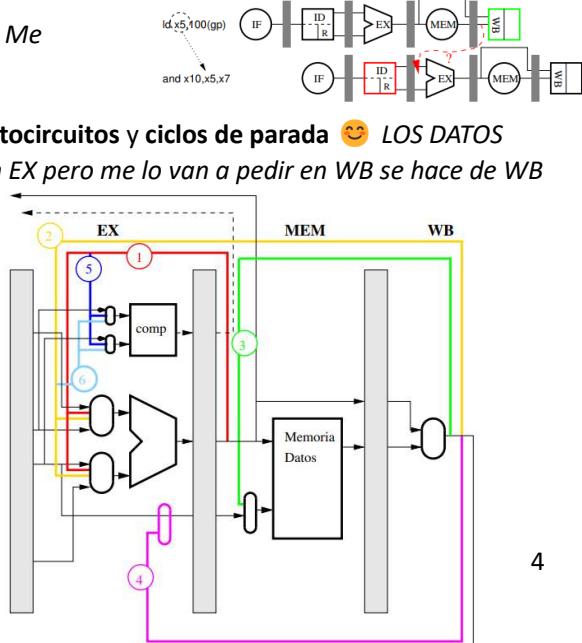
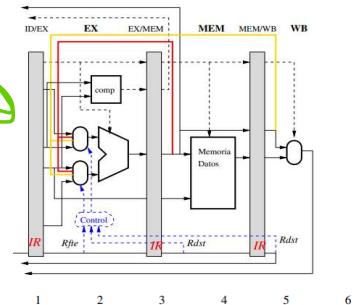
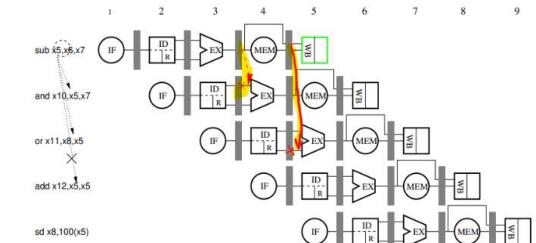
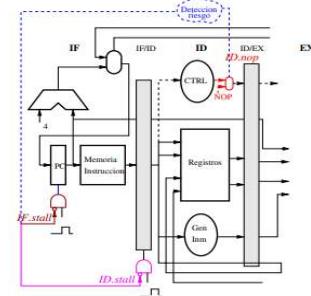
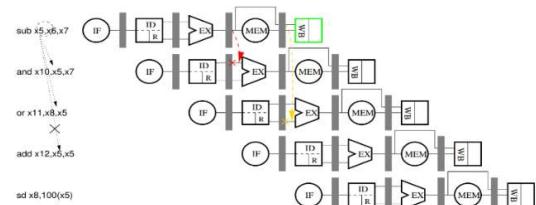
Instrucción consecutiva y dependencia de los loads

Ahora, en vez de **recibir el resultado** en el ciclo 3 lo recibo **en el 4**, por lo que no puedo hacer puente xq aún no sé el dato, sí o sí hace falta otro ciclo extra:

Combinación: Combinamos un **ciclo de parada** y el **cortocircuito**. Me espero a que esté el dato y luego lo paso con cortocircuito.

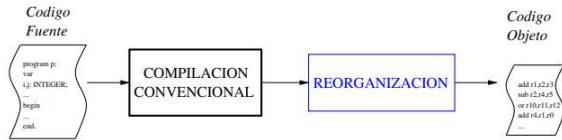
Pues ahora hay hacer eso **para todas las combinaciones de estos cortocircuitos y ciclos de parada** 😊 **LOS DATOS SIEMPRE SE MUEVEN Y SE PASAN EN "VERTICAL". Si o he generado en EX pero me lo van a pedir en WB se hace de WB al otro.**

Tipos de instrucciones	Ejemplo	Cortocircuito	Ciclos de parada	Fig.
UAL - UAL	add x5,x6,x7 sub x8,x5,x9 and x11,x5,x10	MEM a EX WB a EX	0 0	1 2
Carga - UAL	ld x5,100(gp) add x7,x5,x8	WB a EX	1	2
UAL - Carga/Almac.	add x5,x6,x7 ld x6,100(x5) ld x7,200(x5)	MEM a EX WB a EX	0 0	1 2
UAL - Almac.	add x5,x6,x7 sd x5,100(gp) sd x5,200(gp)	WB a MEM WB a EX	0 0	3 4
Carga - Almac.	ld x5,300(gp) sd x5,100(gp) sd x5,200(gp)	WB a MEM WB a EX	0 0	3 4
Carga - Carga/Almac.	ld x5,100(gp) ld x6,100(x5)	WB a EX	1	2
UAL - Salto	slt x5,x6,x7 beq x5,x0,loop	MEM a EX	0	5
UAL - Salto	slt x5,x6,x7 beq x5,x0,loop	WB a EX	0	6
Carga - Salto:	ld x5,100(gp) beq x5,x0,loop	WB a EX	1	6



Sol 3: Reorganización del código

El compilador tiene una fase más después de compilar y hacer todo el trabajo: **Reorganización del código**. Pretende evitar que haya ciclos de parada. *Técnica de la carga retardada*.



Código convencional	Código reorganizado
ld x5,b(gp)	ld x5,b(gp)
ld x6,c(gp)	ld x6,c(gp)
add x7,x5,x6	ld x8,e(gp)
ld x8,e(gp)	add x7,x5,x6
ld x9,f(gp)	ld x9,f(gp)
sub x10,x8,x9	sd x7,a(gp)
sd x7,a(gp)	sub x10,x8,x9
sd x10,d(gp)	sd x10,d(gp)
...	8 ins = 8 ciclos
8 ins + 2 stalls = 10 ciclos	

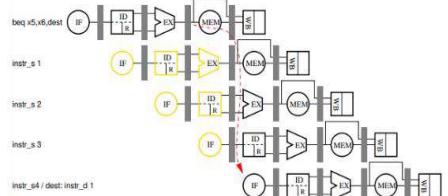
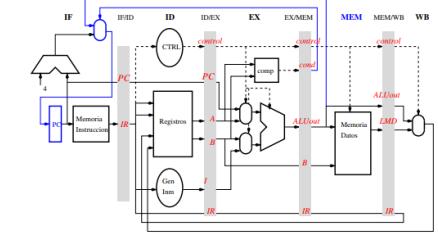
RIESGOS DE CONTROL

Aparecen instrucciones que modifican el contador de programa (PC) y pero esa modificación se hace en etapas posteriores. Inutilizando las instrucciones que ya habían empezado a ejecutarse.

Latencia de salto: Nº de ciclos transcurridos entre la etapa de búsqueda (IF) y la que modifica el PC. Si se hace en **MEM** es 3, en **EX** 2 y en **ID** 1.

Se puede solucionar evitando que entren más instrucciones cuando hay un salto, en concreto 3. Pero esos 3 ciclos de parada te harán ser muy lento.

Predict-not taken: Suponer el **salto no efectivo** de tal forma que las instrucciones buscadas a continuación de la de salto son "válidas". Si, finalmente, el **salto es efectivo**, se abortan las otras instrucciones en curso. **SE ESPERA QUE NO SALTE, SI SALTA YA LO FULMINAS TODO.** (pierdes ciclos en los que ya se había empezado a ejecutar → Ciclos de penalización)



- IMPORTANTE: Estas instrucciones no deben haber modificado el estado.

No se suele usar **PREDICT TAKEN** xq cuando sabes la dirección de memoria a la que saltar, sabes también si saltas o no so te quedas igual

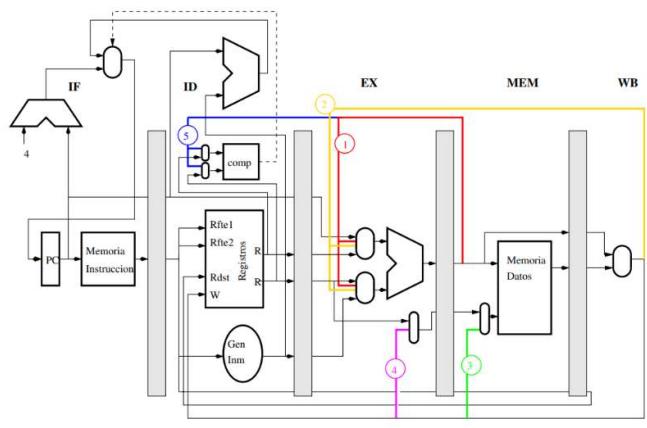
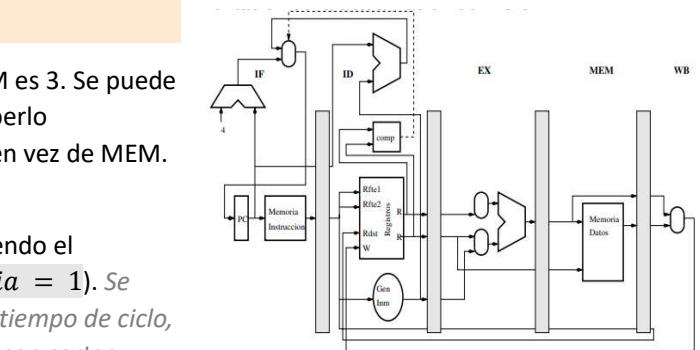
Reducción de latencia de salto

Hemos dicho que el cambio del PC Si se hace en **MEM** es 3. Se puede hacer que en vez de tardar 3 ciclos se tarden 2 en saberlo actualizando y calculando el PC y la condición en **EX** en vez de **MEM**. **Se cablea la salida para tenerlo antes.**

En la solución **se mueve** no solo ha **EX** sino a **ID** poniendo el **comparador y el sumador en la misma fase** (*Latencia = 1*). Se hace xq no llega a tardar tanto como para superar el tiempo de ciclo, no hay problema con hacer tantas cosas en 1 ciclo xq son cortas.

Pues ahora los cortocircuitos no necesitas sus datos en **EX**, los necesitan en **ID**, por lo que hay que cambiar cosas.

Instrucciones	Ejemplo	Cortocircuito	stalls	Fig.
UAL - UAL	add x5,x6,x7 sub x8,x5,x9 and x11,x5,x10	MEM a EX WB a EX	0 0	1 2
Carga - UAL	ld x5,100(gp) add x7,x5,x8	WB a EX	1	2
UAL - Carga/Almac.	add x5,x6,x7 ld x6,100(x5) ld x7,200(x5)	MEM a EX WB a EX	0 0	1 2
UAL - Almac.	add x5,x6,x7 sd x5,100(gp) sd x5,200(gp)	WB a MEM WB a EX	0 0	3 4
Carga - Almac.	ld x5,300(gp) sd x5,100(gp) sd x5,200(gp)	WB a MEM WB a EX	0 0	3 4
Carga - Carga/Almac.	ld x5,100(gp) ld x6,100(x5)	WB a EX	1	2
UAL - Salto	slt x5,x6,x7 beq x5,x0,loop	MEM a EX MEM a ID	0 1	5
UAL - Salto	slt x5,x6,x7 ... beq x5,x0,loop	WB-a-EX MEM a ID	0	5
Carga - Salto	ld x5,100(gp) beq x5,x0,loop	WB-a-EX Por BR	1 2	-
Carga - Salto	ld x5,100(gp) ... beq x5,x0,loop	Por BR	0 1	-



Medir el tiempo de ejecución de un bucle: De oca a oca – 1 → Del primer IF de la primera instrucción del bucle hasta el primer IF de la primera instrucción de la siguiente iteración -1.

El nº de ciclo de la primera IF del segundo bucle – el nº de ciclos de la primera IF del primer bucle

RIESGOS ESTRUCTURALES

El hardware no permite todas las combinaciones posibles de las instrucciones presentes en la unidad si cierto recurso no ha sido replicado suficientemente: Procesador con cache única de instrucciones y datos se usa la cache en MEM y en la lectura de instrucciones en IF. Ambos quieren usar la cache a la vez pero si no está dividida no pueden.

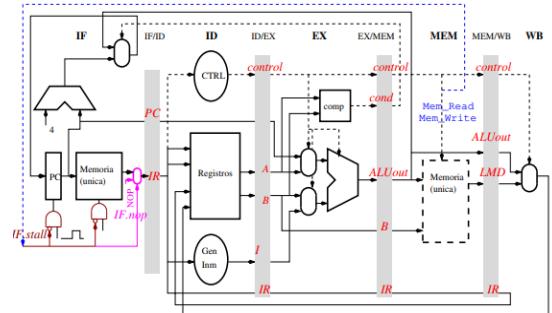
Soluciones

Modificaciones de la ruta de datos: Replicar el recurso para que sea posible esa combinación. En el caso de la caché usar una arquitectura Harvard (cache separada en instrucciones y datos). Sin embargo, esto aumenta el coste y no siempre es posible o tiene sentido.

Inserción de ciclos de parada: Retrasar una de las operaciones que causa el conflicto con stalls. Esto causa una pérdida de prestaciones. Haces que dependan de que tengas o no combinaciones que originan estos riesgos.

- Logica de control:** Cuando una instrucción de Carga/Almacén está en la etapa MEM, No se accede a la memoria de instrucciones, Conservando la instrucción de la etapa IF. Entregando a la etapa ID una instrucción NOP.

Que retrasas todo un ciclo de reloj, bloqueando el acceso a memoria y poniendo un NOP a ID.



EXCEPCIONES

Tienen diferentes nombres: Interrupción, excepción o falta... Un evento que tengo que atender. Hay diferentes tipos:

- Síncrona vs. Asíncrona:** Es síncrona si el evento ocurre en el mismo lugar cada vez que el programa se ejecuta. Si divides entre 0 siempre se hace ahí, pero si se produce en E/S no sabes cuando → (Asíncrona).
- Solicitada por el usuario vs. lanzada hacia el usuario.**
- Enmascarables por el usuario vs. no enmascarables:** Enmascarables son por ejemplo la E/S, y no enmascarables aquellas que sí o sí se han de atender.
- En medio de una instrucción vs. entre instrucciones.**
- Continuar vs. terminar el programa:** Más difícil seguir que terminar.

Si voy a ejecutar instrucciones "i1, i2, i3..." y en i3 me ocurre una excepción, la forma correcta de tratarla sería la Secuencia:

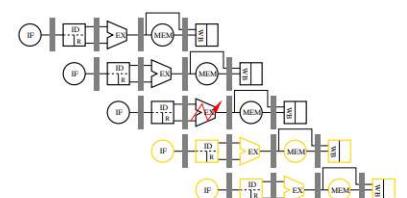
... . i1, i2, i3 - Rutina de servicio - i3, i4, i5, i6 . . .

Excepciones en unidades segmentadas

En el momento en que se produce la excepción, hay varias instrucciones en ejecución posteriores a la que origina la excepción:

Se deja ejecutar la excepción hasta WB y no te cascadas la cabeza pensando en cuales de las de antes cancelas y tal. Nueva secuencia

...i1, i2, i3, i4, i5 - Rutina service - i3, i4, i5, i6...



Excepciones precisas

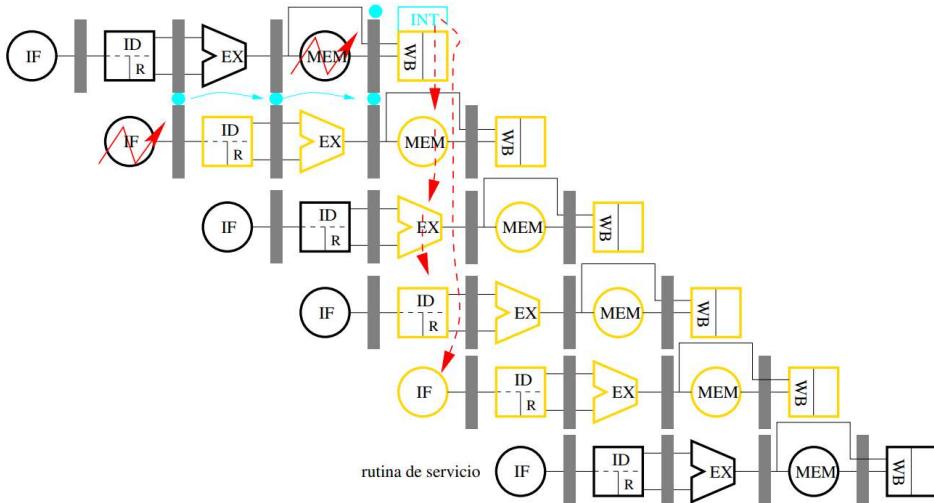
Se dice que un computador soporta un comportamiento preciso frente a las excepciones si:

- Las instrucciones anteriores a la que origina la excepción terminan correctamente.
- La instrucción que origina la excepción y todas las posteriores son abortadas.
- Tras completar la rutina de servicio se puede relanzar el programa comenzando por la instrucción fallida.

Idea: Asegurar que el orden de atención de las excepciones se realiza en el orden natural. Si ha pasado una excepción se deja ejecutar la instrucción hasta WB (no le das que escriba en memoria ni nada pero la dejas seguir) y no te cascás la cabeza pensando en cuáles de las de antes cancelas y tal.

1. En la etapa WB se verifica si la instrucción ha producido una excepción en alguna de las etapas.
 - En caso afirmativo, convierte en NOP las instrucciones posteriores y escribe la dirección de la rutina de servicio en el PC.
2. Se guarda la dirección de la instrucción que origina la excepción.
3. La rutina de servicio toma el control.
4. Cuando finaliza la rutina de servicio, se restaura el PC y continuar con la ejecución desde este punto.

Ejemplo



En este ejemplo se produce una excepción en MEM, se deja seguir la instrucción y se lleva la señal de excepción hasta WB, donde se tratará cambiado el CP y todo el rollo.

Además, como se ha detectado una excepción, las siguientes fases de la instrucción serán NOPs (Para ella solo WB, pero el resto de las instrucciones que ya se estaban ejecutando serán todos los ciclos de NOPs). Evitando que se cambie el estado escribiendo en memoria o en registros.

Por otra parte, en la segunda, se ha producido una excepción en IF, por lo que se extiende la señal de esta y el resto de ciclos son una NOP (incluyendo en los que se ha detectado la excepción en la 1r).

Al haber cambiado ya el CP, la siguiente instrucción que entra es de la rutina de servicio. Esta se ejecuta y al terminar se volvería a la 1r (o si no toca) a la 2n instrucción para seguir ejecutando.

En esta segunda se dejaría que la excepción volviera a ocurrir, repitiendo el ciclo otra ve...

