INTERRUPCIONES Y PIPELINE

- Falta de acuerdo en la **terminología**:
  - Preferimos: ***interrupción*** como término genérico.
    - Hennessy y Patterson [2002] usan como genérico ***excepción***.
- **Interrupciones usuales** (también hay variaciones en terminología):
  - Interrupción del **temporizador** (se genera a intervalos regulares).
    - Se usa, por ejemplo, en sistemas de tiempo compartido (la rutina de tratamiento de la interrupción [***interrupt handler***] produce un cambio de contexto).
  - Interrupciones de **entrada/salida**.
    - Informan a la CPU de que un dispositivo de e/s ha terminado una tarea. Distintas acciones posibles de la rutina de interrupción.

- Interrupciones usuales (continuación):
  - **Excepción**.
    - La ejecución de una instrucción produce un error como división por cero, *underflow*, desbordamiento... La rutina de interrupción puede producir un valor predeterminado para esa excepción, o bien generar un mensaje de error y abortar el programa.
  - ***Trap*** (interrupción programada).
    - Es similar a un salto a subrutina, pero una subrutina del sistema operativo, con privilegios no accesibles directamente por el código de usuario.
    - Ejemplo, llamadas a rutinas de entrada/salida del sistema.
  - **Instrucción** de la arquitectura **no implementada** en hardware.
    - La rutina emula la instrucción usando las instrucciones implementadas (ej. típico: operación aritmética compleja).

- Interrupciones usuales (continuación):
  - **Falta de página** en memoria virtual.
    - Se solicita al disco la página. Suele haber un cambio de contexto para que la CPU no esté parada mientras se accede al disco.
  - **Fallo hardware**.
    - Se abortan todos los programas y se ejecuta una rutina de diagnóstico del fallo
  - Otras:
    - Instrucciones de **traza** y **punto de ruptura**.
    - **Violación** de la **protección de memoria**.
    - **Acceso a memoria mal alineado** (la dirección debiera ser múltiplo del tamaño del objeto de memoria al que se quiere acceder).

CARACTERIZACIÓN DE LAS INTERRUPCIONES

- **Síncronas/asíncronas**.
  - Síncronas: causadas por una instrucción particular.
  - Asíncronas: causadas por dispositivos externos y fallos hardware.
    - Podemos terminar las instrucciones ya empezadas antes de saltar a la rutina de interrupción => Más fáciles de manejar.
- **Forzadas/solicitadas** por el programa de usuario.
  - Solicitadas por el usuario: más fáciles de manejar.
- **Enmascarables/no** enmascarables por el programa de usuario.

- **Dentro** de una instrucción/**entre** instrucciones
  - Dentro...: impiden la terminación de la instrucción interrumpida.
    - Más difíciles de tratar porque la instrucción interrumpida deberá ser reiniciada tras volver de la rutina de tratamiento.
- **Con reanudación/terminación** del programa.
  - Evidentemente, las primeras son más difíciles de implementar.

INTERRUPCIONES Y TIPO AL QUE PERTENECEN

	Síncrona/ asíncrona	Forzada/ solicitada	¿Enmas-carable?	Dentro/ entre	¿Con reanu-dación?
Del tempo-rizador	Síncrona	Forzada	Sí	Entre	Sí
De e/s	Asíncrona	Forzada	No	Entre	Sí
Excepción	Síncrona	Forzada	Sí	Dentro	Sí

Interrupción	Síncrona/ asíncrona	Forzada/ solicitada	¿Enmas-carable?	Dentro/ entre	¿Con reanu-dación?
Trap	Síncrona	Solicitada	No	Entre	Sí
Instrucción no implementada	Síncrona	Forzada	No	Entre	Sí
Falta de página	Síncrona	Forzada	No	Dentro	Sí
Fallo hardware	Asíncrona	Forzada	No	Dentro	No
Traza/ *breakpoint*	Síncrona	Solicitada	Sí	Entre	Sí
Violación protección memoria	Síncrona	Forzada	No	Dentro	Sí
Acceso a memoria mal alineado	Síncrona	Forzada	Sí	Dentro	Sí

INTERRUPCIONES PRECISAS

- **Interrupciones precisas**: El **estado guardado** (básicamente PC, registros y memoria) al saltar a la rutina de interrupción se corresponde con el **modelo secuencial** de ejecución, es decir, es el mismo que si no hubiera solapamiento de instrucciones.

- Siguiendo a Smith y Pleszkun, lo que caracteriza a la interrupciones precisas es que en el momento de saltar a la rutina de tratamiento de la interrupción:
  - Las instrucciones anteriores (en el modelo secuencial) a la interrumpida se han ejecutado y han modificado el estado correctamente.
  - Las posteriores, se han cancelado sin que el estado del procesador quede modificado por ellas.
  - La instrucción interrumpida estará en uno de los dos casos anteriores según el tipo de interrupción.

INTERRUPCIONES PRECISAS VS. IMPRECISAS

- **Requisito fundamental de las interrupciones**: Seguridad de que un proceso interrumpido, por una interrupción no catastrófica, sea **capaz de continuar** su ejecución **correctamente**.

- Para cumplir el requisito **es condición suficiente que las interrupciones sean precisas**. Sin embargo, **no es** condición **necesaria**.

- Cantidad de estado guardado antes de saltar a la rutina de tratamiento:
  - Interrupciones **precisas**: el **estado del modelo secuencial** (el que guardaría un procesador no encauzado).
  - Interrupciones **imprecisas**: una **información de estado más compleja**.

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS

- Supondremos:
  - Las instrucciones se emiten en el orden del modelo secuencial.
  - Estudiaremos el caso más complejo: interrupciones dentro de instrucciones y con retorno.

- **Mecanismo básico** para conseguir interrupciones precisas:
  - Las instrucciones sólo modifican irreversiblemente el estado (del modelo secuencial) cuando hay seguridad de que ninguna de las instrucciones emitidas con anterioridad puede provocar interrupción.
    - Así, cuando una instrucción I provoca interrupción, las emitidas después no han modificado el estado.
    - Las emitidas antes se completarán. Si, como consecuencia, se provoca una interrupción, esta interrupción se procesará antes que la provocada por la instrucción I.

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS (cont.)

- Modos de llevar a cabo el mecanismo básico:

  A. Las instrucciones **no modifican el estado** hasta que ninguna de las emitidas con anterioridad pueda provocar interrupción. Dos opciones:

     A1. Las instrucciones **se detienen** para evitar la modificación indeseada del estado (hablamos siempre del estado del modelo secuencial).

     A2. Las instrucciones **no se detienen** pero, en vez de modificar el estado, los resultados de las instrucciones se guardan en algún almacenamiento provisional. Después, cuando haya seguridad de que ninguna de las emitidas con anterioridad pueda provocar interrupción, se actualizará el estado a partir de la información del almacenamiento provisional.

  B. Las instrucciones **modifican el estado** aunque instrucciones emitidas con anterioridad puedan provocar interrupción, pero la modificación del estado es **reversible**.

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS (cont.)

- Se suele distinguir el tratamiento del PC, de los registros y de la memoria.
  - Para el **PC**, se usa siempre el **modo B** porque es imprescindible que cambie en cada ciclo para poder buscar la siguiente instrucción.
  - Para la **memoria** se usa el **modo A** porque deshacer una modificación de memoria es muy lento.

- Modo A1.
  - La manera más simple de saber que las instrucciones emitidas con anterioridad no provocarán interrupción es esperar a que terminen => Terminación en orden y rendimiento bajo.
  - No es necesario esperar a que las instrucciones terminen para saber que no provocarán interrupción. P. ej. MIPS no provoca interrupción después de MEM. En general, hardware adicional puede ayudar a saber que una instrucción no provocará interrupción.
    - Menos esperas y posible terminación fuera de orden.

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS (cont.)

- Modo A2. Técnica ejemplo: **fichero de futuro**.
  - Permite hacer preciso el estado de los **registros**.
  - Dos ficheros de registros:
    - Arquitectural: refleja el estado (del modelo secuencial) de la máquina.
    - De futuro:
      - El que se actualiza cuando las instrucciones producen resultados.
      - Refleja el futuro del arquitectural. Cuando finaliza una instrucción I y ninguna de las anteriores pueda provocar interrupción, el fichero arquitectural se actualiza con los valores de los operandos destino de I almacenados en el fichero de futuro.
  - Cuando hay una interrupción finalizan las instrucciones emitidas antes que la interrumpida y, como consecuencia, en el momento de saltar a la rutina de tratamiento el estado arquitectural es el correcto.

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS (cont.)

- Modo A2. Técnica ejemplo: **buffer de almacenamiento**.
  - Permite hacer preciso el estado de **memoria**.
  - Las instrucciones cuyo operando destino está en memoria (STORE en los RISC) escriben sus resultados en el buffer.
  - Cuando todas las instrucciones emitidas antes de una STORE no puedan provocar interrupción la memoria se actualiza con el contenido del buffer.
  - Las instrucciones de carga leerán simultáneamente en el buffer de almacenamiento y en la caché. Si el dato está en el buffer lo tomarán de ahí.

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS (cont.)

- Modo B. Técnica ejemplo: **buffer de historia**.
  - Es una cola (FIFO) donde, cada vez que la CPU emite una instrucción, crea una entrada para esa instrucción.
  - Cuando la instrucción va a escribir (un registro), el valor antiguo se coloca en la entrada de la cola que se reservó para la instrucción.
  - Las instrucciones abandonan la cola en cuanto han llegado a la cabecera de la misma y han terminado su ejecución.
  - Una instrucción sólo puede provocar interrupción cuando alcanza la cabecera de la cola (así es seguro que ninguna anterior puede provocar interrupción).
  - Cuando hay una interrupción, el buffer de historia permite restaurar el estado anterior a la entrada en el cauce de una instrucción.

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS (cont.)

- Ejemplo con el buffer de historia.
  - Etapas del cauce: **F (fetch), I (issue), E (exe), W (write back)**.
  - Secuencia de instrucciones de punto flotante y diagrama de tiempos:

	1	2	3	4	5	6	7	8	9	10	11	12
F3 := F1*F4	***F***	***I***	***E***	***E***	***E***	***E***	***E***	***E***	***W***			
F4 := F1+F5		***F***	***I***	***E***	***E***	***W***						
F6 := F4*F8			***F***	***I***		***E***	***E***	***E***	***E***	***E***	***E***	***W***

En el ciclo 5 la tercera instrucción espera a que la segunda termine de calcular F4.

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS (cont.)

- Ciclo 2: Se emite F3 := F1*F4. En el buffer aparece la entrada
  [(F3 := F1*F4, ?)]
- Ciclo 3: Se emite F4 := F1+F5. El buffer queda
  [(F4 := F1+F5, ?) (F3 := F1*F4, ?)]
- Ciclo 4: Se emite F6 := F4*F8. El buffer queda
  [(F6 := F4*F8, ?) (F4 := F1+F5, ?) (F3 := F1*F4, ?)]
- Antes del ciclo 6 los valores de los **registros** F1 a F8 eran:
  (27, 1, -21, 13, 0, 88, 3, 0).
- Ciclo 6: F4 := F1+F5 escribe F4 y finaliza.
- Terminado el ciclo 6 el valor de los registros será
  (27, 1, -21, **27**, 0, 88, 3, 0) y el buffer queda
  [(F6 := F4*F8, ?) (F4 := F1+F5, **13**) (F3 := F1*F4, ?)]

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS (cont.)

- Ciclo 9 . Dos casos:
  - F3 := F1*F4 termina sin provocar interrupción.
    - Esta instrucción y F4 := F1+F5 abandonan el buffer puesto que ocupan la cabecera de la cola y han finalizado su ejecución. La cola queda [(F6 := F4*F8, ?)]
  - F3 := F1*F4 provocó interrupción en el ciclo 8.
    - Se recupera el estado que había en el momento de emitirse la instrucción: El 13 del buffer se graba en el registro F4.
    - En general, hay que recorrer el buffer desde la cola hacia la cabecera para recuperar el estado modificado por instrucciones posteriores a la interrumpida.

IMPLEMENTACIÓN DE INTERRUPCIONES PRECISAS (cont.)

- En todos los modos es beneficioso que las instrucciones sólo modifiquen el estado al final:
  - En el modo A1, menos ciclos de detención de las instrucciones.
  - En los modos A2 y B, menos operaciones en los almacenamientos auxiliares.

- El PC antiguo:
  - Puede incorporarse al buffer de historia como un campo más de cada entrada.
  - Puede transmitirse junto con cada instrucción por los *latches* de la CPU encauzada.

INTERRUPCIONES IMPRECISAS

- Justificadas cuando
  - Mejoran el rendimiento.
    - Por ejemplo no daría buen resultado tener que reiniciar desde el principio una instrucción de larga duración (típicamente de punto flotante) que ha quedado interrumpida cuando ya ha sido procesada parcialmente. Mejor reiniciarla en el punto en que quedó.
      - Esto supone aumentar la información de estado (no basta con el estado del modelo secuencial).
  - La interrupción es grave y aborta la ejecución del programa.

- Algunos tipos de interrupciones deben ser precisas.
  - Al menos las relacionadas con la depuración (ejecución paso a paso y puntos de ruptura).

CASO DE BIFURCACIONES RETARDADAS

- La instrucción interrumpida está en un hueco de retardo. **¿Dónde retornar?**
  - Si simplemente se retorna a la instrucción interrumpida, el programa continuará y se habrá ignorado la bifurcación retardada anterior.
  - Por tanto la solución es más compleja. Una posible solución sería:
    - Las instrucciones emitidas antes que la interrumpida se completan.
    - Se guardan en una cola la dirección de la instrucción interrumpida (sólo si hay que retornar a ella) y las direcciones de las instrucciones siguientes que estén en huecos de retardo.
    - Al final de la cola se guarda la dirección calculada por la instrucción de bifurcación.
    - Al retornar de la rutina de interrupción, la etapa FETCH toma las direcciones de las instrucciones de esa cola.
      - Cuando la cola quede vacía el PC se actualiza normalmente.

BIBLIOGRAFÍA SOBRE INTERRUPCIONES PRECISAS

- Hennessy, J. and D. Patterson [2002], *Computer Architecture. A Quantitative Approach, 3rd. Edition,* Morgan Kaufmann.

- Shriver, B. and B. Smith [1998], *The Anatomy of a High-Performance Microprocessor. A Systems Perspective,* IEEE Computer Society Press.

- Smith, J. and A. Pleszkum [1985], "Implementation of Precise Interrupts on Pipelined Processors", en el CD del libro Shriver and Smith [1998].

- Moudgill, M. and S. Vassiliadis [1996], "Precise Interrupts", en el CD del libro Shriver and Smith [1998].

POSIBLES INTERRUPCIONES EN LAS ETAPAS DE MIPS

- IF: Falta de página, dirección mal alineada, violación de la protección de memoria.
- ID: Código de operación ilegal o indefinido.
- EX: Excepción aritmética.
- MEM: Falta de página, dirección mal alineada, violación de la protección de memoria.
- WB: Ninguna.

INTERRUPCIONES PRECISAS EN MIPS

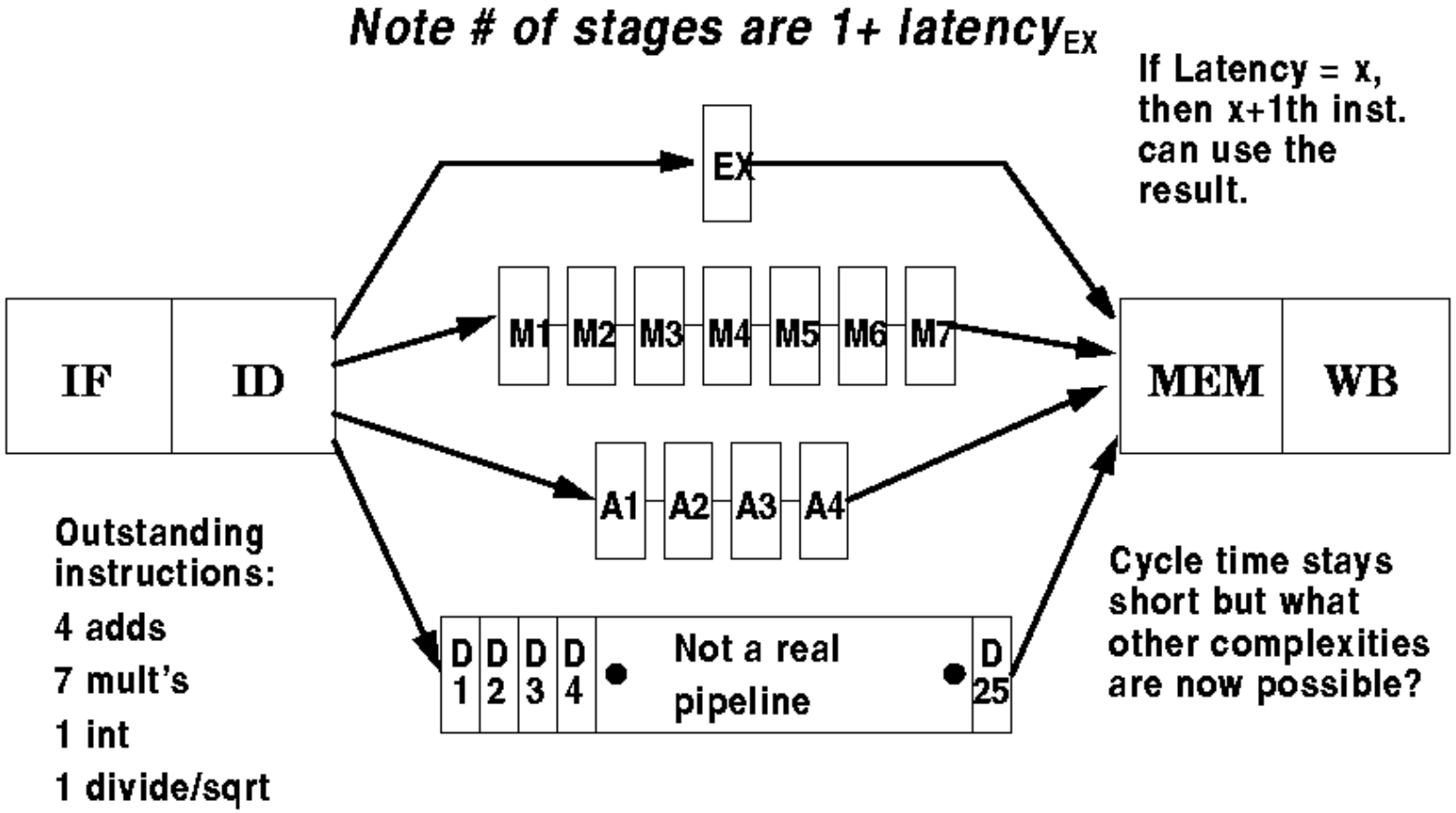
- Las instrucciones no modifican el estado (salvo el PC) hasta MEM o WB =>
  - Cuando se produce una interrupción es seguro que las instrucciones posteriores a la interrumpida no han modificado el estado.
    - Por tanto no hay problema. Las posteriores a la interrumpida, I, se anularán y las anteriores se completarán. Si, como consecuencia, se provoca una interrupción, esta interrupción se procesará antes que la provocada por I.

- Tratamiento del PC.
  - El PC de cada instrucción se lleva con ella de un *latch* al siguiente del cauce => Dirección de retorno accesible por la rutina de tratamiento.
  - Si la instrucción interrumpida está en IF y en ID hay una bifurcación (o salto), una vez que la bifurcación e instrucciones anteriores se completen iremos a la rutina de tratamiento y retornaremos con dos PC: 1º el de la instrucción interrumpida; 2º el calculado por la bifurcación si fue efectiva.

ADAPTACIÓN DEL CAUCE A OPERACIONES MULTICICLO

- **¿Cómo implementar las operaciones de punto flotante?** Opciones:
  - Aumentar el tamaño de ciclo para que en un ciclo la etapa EX pueda calcular cualquier de las operación de punto flotante implementada.
    - **Mala solución**: Las demás instrucciones resultan innecesariamente lentas.
  - Permitir que la etapa **EX** consuma **varios ciclos para las operaciones complejas** y mantener la duración del ciclo.
    - Ejemplo en página siguiente, donde a la etapa EX se añaden, en paralelo las siguientes **unidades funcionales**:
      - Una de división no encauzada (25 ciclos).
      - Una de multiplicación encauzada (7 etapas de 1 ciclo).
      - Una de suma en punto flotante encauzada (4 etapas de 1 ciclo).

MIPS AMPLIADO PARA OPERACIONES DE PUNTO FLOTANTE (MIPS FP)



INTERVALO DE REPETICIÓN Y LATENCIA DE UNA UNIDAD FUNCIONAL

- **Intervalo de iniciación** o repetición de una unidad funcional:
  - Número de **ciclos** que deben transcurrir **entre la emisión de dos instrucciones** que deben ser **procesadas por esa unidad funcional**.
- **Latencia** de una unidad funcional (relativa a la instrucción que utiliza el resultado):
  - **Número mínimo de ciclos intermedios entre** una **instrucción que produce un resultado** en esa unidad funcional **y** una **instrucción que utiliza** ese resultado.
    - Dicha instrucción suele **necesitar el resultado al ser emitida**.
      - Sin embargo, ***store*** necesita el resultado **un ciclo más tarde** (en MEM) cuando se trata del valor a escribir en memoria.

INTERVALOS DE REPETICIÓN Y LATENCIAS EN MIPS FP (con cortocircuito)

Unidad funcional que produce el resultado	Intervalo de iniciación	Latencia (caso general)	Latencia (caso *store* p. anterior)*
Unidad entera (EX)	1	0	0
Memoria de datos (MEM)	1	1	0
Suma/resta FP	1	3	2
Multiplicación (entera o FP)	1	6	5
División (entera o FP)	25	24	23

* (Se supone que no hay problema en que *store* y una instrucción aritmética [o varias] lleguen a MEM a la vez, ya que esta no tiene nada que hacer en MEM)

PROBLEMAS DEBIDOS A LA ADAPTACIÓN DEL CAUCE A OPERACIONES MULTICICLO

- Habrá **más paradas provocadas por** las **dependencias RAW** debido a la **mayor latencia** de las unidades funcionales.
- Surgen **nuevos riesgos estructurales**.
  - Varias instrucciones podrán intentar escribir en el **banco de registros** a la vez.
  - Al no estar encauzada la **u. f. de división** generará colisiones.
- Debido a que las instrucciones pueden llegar a WB **fuera de orden**:
  - Surgen **dependencias WAW** que no existían antes.
  - El **tratamiento** de las **interrupciones** resulta **más complejo**.

DEPENDENCIAS RAW (LDE)

- El número de ciclos de espera debidos a dependencias RAW aumenta considerablemente. Ejemplo:

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L.D F4,0(R2)	IF	ID	EX	*M*	WB												
MUL.D F0,F4,F6		IF	ID	/	**M1**	M2	M3	M4	M5	M6	*M7*	M	WB				
ADD.D F2,F0,F8			IF	/	ID	/	/	/	/	/	/	**A1**	A2	A3	A4	*M*	WB
S.D 0(R2),F2					IF	/	/	/	/	/	/	ID	EX	/	/	/	**M**

Las etapas-ciclo en cursiva generan un dato que se anticipa a la instrucción siguiente en su etapa-ciclo marcado en negrita.

RIESGOS ESTRUCTURALES EN BANCO DE REGISTROS

Instrucción	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0, F4, F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
…		IF	ID	EX	MEM	WB					
…			IF	ID	EX	MEM	WB				
ADD.D F2, F4, F6				IF	ID	A1	A2	A3	A4	MEM	WB
…					IF	ID	EX	MEM	WB		
…						IF	ID	EX	MEM	WB	
L.D F2, 0(R2)							IF	ID	EX	MEM	WB

- Ciclo 10: no hay riesgo estructural porque sólo L.D accede a memoria
- Ciclo 11: tres instrucciones intentan escribir en el banco de registros a la vez.

SOLUCIÓN DE RIESGOS ESTRUCTURALES EN BANCO DE REGISTROS

- Suponemos que el banco de registros sólo tiene un puerto de escritura.
  - Un banco con varios puertos de escritura tiene mayor coste y pocas veces se utilizarán esos puertos a la vez.

- **Opción 1**: **al emitir la instrucción se anota cuantos ciclos transcurrirán hasta que escriba en** el banco de **registros**.
  - Si transcurrirán n ciclos se pone a 1 el bit n-simo de un **registro**.
  - El registro **se desplaza** un bit **a la derecha** en cada ciclo.
  - Instrucción en ID: **no se emite hasta que** el momento en que escribiría si se emitiera no coincida con el momento en que escribirá alguna de las instrucciones emitidas.
    - Es decir, la instrucción no se emite si el bit del registro donde tiene que anotar un 1 está ya a 1.

SOLUCIÓN DE RIESGOS ESTRUCTURALES EN BANCO DE REGISTROS (cont)

- En el ejemplo, la instrucción ADD.D F2, F4, F6 esperará en ID durante el ciclo 6 para llegar a WB en el 12 y no en el 11.
  - L.D F2, 0(R2) esperará un ciclo en ID, para llegar a WB en el ciclo 13.

- **Opción 2: Detener las instrucciones** conflictivas cuando intentan entrar **en MEM** (o en WB).
  - ¿A qué instrucción/u. f. damos **prioridad** para entrar en MEM?.
    - Solución razonable: a la **unidad funcional con mayor latencia**.
      - La de mayor latencia **probablemente causa más detenciones** de instrucciones posteriores **por dependencias RAW**.
  - **Ventaja** de esta opción: es más sencillo detectar el conflicto.
  - **Inconveniente**: complica el control porque las paradas pueden surgir ahora en distintas etapas.

DETECCIÓN DE RIESGOS EN MIPS FP

- Tres **comprobaciones en ID antes de emitir** una instrucción.
  - Comprobación de **riesgos estructurales**.
    - Esperar mientras la unidad funcional necesaria esté ocupada.
      - En MIPS, sólo la unidad funcional de división puede estarlo.
    - Asegurarse de que el banco de registros necesario (GPR o FPR) esté disponible en el ciclo en que se va a necesitar.
  - Comprobación de **dependencias RAW**.
    - Hacer **lista regs. destino** cuyo **valor** esté **pendiente de ser calculado**.
    - **No emitir** la instrucción **hasta que** se sepa que sus registros fuente se habrán borrado de la lista anterior cuando la instrucción los necesite.
  - Comprobación de **dependencias WAW** (son poco frecuentes).
    - No emitir la instrucción mientras alguna instrucción en A1, ..., A4, D, M1, ..., M7 tenga el mismo registro destino.

CAUCES LARGOS E INTERRUPCIONES

- Consideremos la siguiente secuencia de instrucciones

```
DIV.D    F0, F2, F4
ADD.D    F10, F10, F8
SUB.D    F12, F12, F14
```

- No hay dependencias pero…
  - ADD.D y SUB.D terminan antes que DIV.D y...
  - ¿qué ocurre si DIV.D provoca una interrupción?

CAUCES LARGOS E INTERRUPCIONES: ALTERNATIVAS

- Soluciones precisas: las que hemos estudiado (del tipo A1, A2 o B).
- Solución **imprecisa**:
  - **Guardar suficiente información** para que la rutina de tratamiento pueda conocer las instrucciones que estaban en el cauce y sus PC.
  - La **rutina** de tratamiento, tras tratar la interrupción, **completará** las **instrucciones** anteriores a la última instrucción finalizada.
    - Ej.: I_1, ..., I_n, donde, cuando I_1 causa una interrupción, I_2, ..., I_{n-1} no han finalizado pero I_n sí.
      - Tras tratar la interrupción, la rutina completa la ejecución de I_1, ..., I_{n-1} y retorna a la instrucción I_{n+1}.
  - **Inconveniente**: **Complejidad** de completar adecuadamente la ejecución de las instrucciones.
    - Esta solución es viable sólo cuando el máximo número simultáneo de instrucciones de punto flotante en ejecución es muy pequeño (1 ó 2).

DISEÑO DEL JUEGO DE INSTRUCCIONES Y PIPELINE

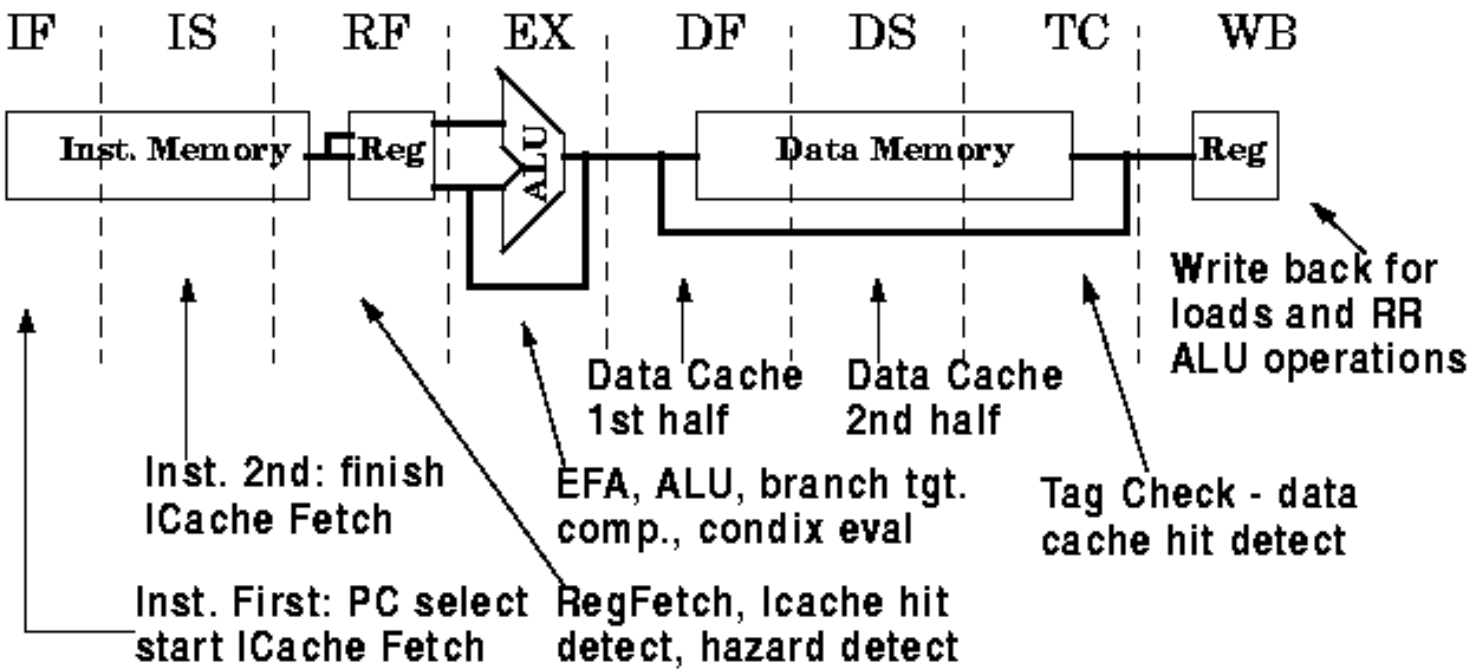
- Juego de instrucciones **complejo** => pipeline **poco eficiente**.
  - Instrucciones de **longitudes y tiempos de ejecución variados**.
    - Aumenta el número de paradas en el pipeline.
    - Complica la detección de riesgos y dificulta el correcto retorno de las interrupciones.
    - **No siempre hay que evitarlo**. **Ej. cachés**. El tiempo en acceder a memoria varía en función de que haya acierto o fallo.
      - La mayoría de las máquinas paralizan el cauce cuando hay un fallo en caché.
      - Las ventajas de las cachés compensan.
  - Modos de **direccionamiento sofisticados** => diversidad de **problemas**.
    - P. ej., los que actualizan registros complican la detección de riesgos y la consecución de interrupciones precisas.

DISEÑO DEL JUEGO DE INSTRUCCIONES Y PIPELINE (cont.)

- **Código automodificable**.
  - Una instrucción puede modificar otra que ya está en el cauce.
    - Se necesita hardware para detectar si esto sucede.
    - Normalmente hay que vaciar el pipeline y modificar la instrucción antes de que vuelva a ser procesada por la CPU.
- **Códigos de condición actualizados implícitamente** por las instrucciones **dificultan** la **planificación** del pipeline.
  - Dificultad/imposibilidad de separar la instrucción que fija la condición de la instrucción de bifurcación => A menudo, la bifurcación tiene que esperar (hay dependencia RAW respecto a los códigos de condición).
  - Dificultad/imposibilidad para colocar instrucciones en los huecos de retardo de la bifurcación.
  - **Soluciones**: a) Un bit en la instrucción determina si actualiza o no los códigos de condición; b) suprimir los códigos de condición.

CAUCE DEL MIPS R4000

- Familia de procesadores **superencauzados**.



RETARDO DE CARGA EN MIPS R4000

- **2 ciclos**.
  - El **valor** está **disponible al final de DS** y puede ser **anticipado** pero...
    - ¡El valor anticipado **no es correcto si** el acceso a caché es un **fallo**!
  - **Si**, en TC, se detecta **fallo** y el valor se anticipó:
    - El cauce se paraliza mientras se resuelve el fallo de caché.
    - Cuando se tiene el dato correcto:
      - La instrucción que estaba en EX vuelve a ser procesada por esa etapa, pero ahora con el nuevo dato.
      - El cauce deja de tenerse parado.

BIFURCACIONES EN MIPS R4000

- **3 ciclos de retardo** en saltos y bifurcaciones.
  - La dirección de salto se conoce **al final de la etapa EX**.
  - En la arquitectura MIPS las **bifurcaciones** son **retardadas** en **1 ciclo**.
  - El R4000 usa una **estrategia** de **predecir salto no efectivo** para los otros 2 ciclos de retardo.
  - Existe una instrucción de **bifurcación con cancelación** (*branch-likely*) que cancela la instrucción ubicada en el primer hueco de retardo cuando no se cumple la condición de salto.
    - Esto facilita al compilador colocar una instrucción en ese hueco de retardo

CAUCE DEL MIPS R4000 FP

- La unidad de punto flotante puede verse de dos maneras:
  - Formada por 3 unidades funcionales (división, producto y suma FP)
  - Formada por 8 etapas, pertenecientes a las uu.ff. que indica la tabla.

Etapa	Unidad funcional	Descripción
A	FP Adder	Suma mantisas
D	FP Divide	Etapa de división
E	FPMultiplier	Comprobación de excepción
M	FPMultiplier	1ª. etapa de multiplicar
N	FPMultiplier	2ª. etapa de multiplicar
R	FP Adder	Redondeo
S	FP Adder	Desplazamiento de operandos
U	las tres	Desempaquetado

OPERACIONES DE PUNTO FLOTANTE EN MIPS R4000

Instrucción FP	Latencia	Intervalo de Iniciación	Secuencia de etapas
Suma, resta	4	3	U, S+A, A+R, R+S
Multiplicación	8	4	U, E+M, M, M, M, N, N+A, R
División	36	35	U,A,R,D^{27},D+A,D+R,D+A,D+R,A,R
Raíz cuadrada	112	111	U, E, $(A+R)^{108}$, A, R
Negado	2	1	U, S
Valor absoluto	2	1	U, S
Comparación FP	3	2	U, A, R