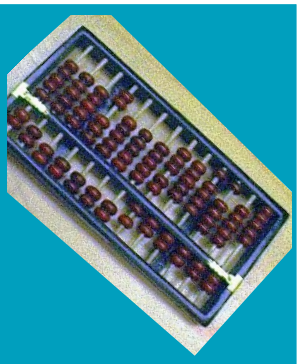
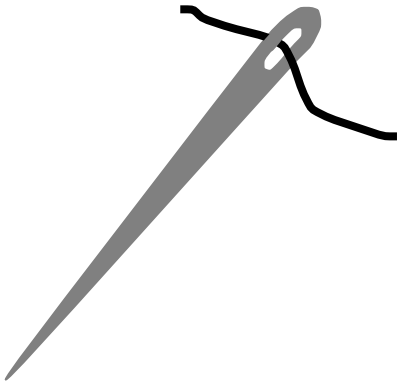


Arquitectura de Computadores



$$T = N \cdot CPI \cdot t_c$$

J.M. Llabería
E. Herrada
A. Olivé

5

PROCESADOR SEGMENTADO MULTICICLO

En el procesador segmentado descrito en el Capítulo 3 y en el Capítulo 4 todas las instrucciones, independientemente del tipo, atraviesan las mismas etapas de la segmentación, ya que hemos unificado la segmentación. Excepto cuando una instrucción se retiene en una etapa para gestionar un riesgo de datos, una instrucción está un ciclo en cada etapa y en el siguiente ciclo pasa a la siguiente etapa.

La unificación de la segmentación para todos los tipos de instrucciones puede dar lugar a dificultades en la implementación y/o ineficiencias debido a riesgos estructurales. Ejemplos de instrucciones que son difíciles de integrar en la sencilla segmentación descrita en los capítulos previos son: a) instrucciones que manipulan números representados en coma flotante o b) ciertas instrucciones de aritmética entera que requieren varios ciclos de ejecución.

En este capítulo se estudia el diseño de procesadores con caminos de datos ramificados o paralelos en función del tipo de instrucción. A este tipo de procesadores los denominaremos procesadores segmentados multiciclo. También se presentan transformaciones de código que permiten incrementar el paralelismo a nivel de instrucción con el objetivo de tolerar u ocultar la latencia productor-uso.

Contenido

INTRODUCCIÓN

En el Capítulo 2 y en el Capítulo 3 se ha unificado la segmentación para todas las instrucciones analizadas. Para ello, se segmentó la ejecución de las instrucciones de acceso a memoria, cuya latencia es de 2 ciclos (ciclos 4 y 5), de forma que estuviera permitida la latencia de inicio igual a uno. Además, se retardó la actualización del banco de registros en instrucciones de aritmética entera (del ciclo 5 al ciclo 6), para eliminar un riesgo estructural producido por el único camino de escritura al banco de registros.

Cuando se utiliza una segmentación unificada para todos los tipos de instrucciones, en una etapa sólo puede haber una instrucción y las instrucciones avanzan de etapa en etapa al ritmo de ciclo de reloj. Entonces, si una instrucción utiliza recursos que no permiten una latencia de inicio igual a 1 se pierde eficiencia.

En la Figura 5.1 se muestra un camino de datos simplificado con una ALU y una unidad funcional para multiplicar números enteros. La latencia de operación de la ALU es 1 ciclo y la latencia de operación del multiplicador son 4 ciclos. El multiplicador no está segmentado y su latencia de inicio son 4 ciclos.

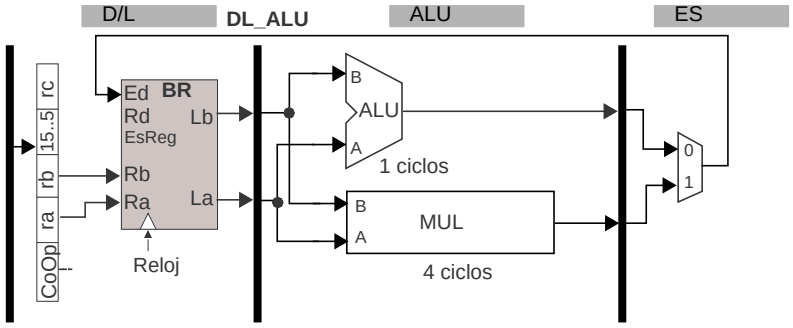


Figura 5.1 *Camino de datos unificado*

En el diseño de la Figura 5.1 una instrucción de multiplicación bloquea la interpretación de instrucciones durante 3 ciclos, ya que el registro de desacoplo DL_ALU almacena durante 4 ciclos los datos que se utilizan en el multiplicador para efectuar el cálculo (Figura 5.2), debido a que el multiplicador no está segmentado.

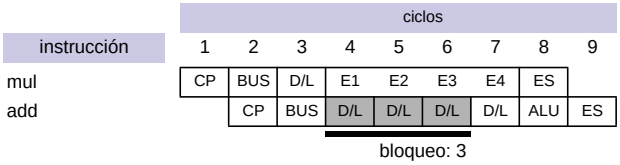


Figura 5.2 *Riesgo estructural en un camino de datos unificado, debido a que el multiplicador no tiene latencia de inicio igual a 1.*

En la Figura 5.3 se muestra un diseño alternativo donde no se bloquea la interpretación de instrucciones, después de iniciar la ejecución de una multiplicación, si las siguientes instrucciones utilizan la ALU. Para ello se utilizan segmentaciones paralelas, una por cada tipo de unidad funcional (ALU, multiplicador). Cada unidad funcional puede tener una latencia de cálculo y/o de inicio distinta. Para tener caminos paralelos se utiliza un registro de desacoplo DL_Ex distinto por cada ramificación. De esta forma se elimina el riesgo estructural que determina una unidad funcional con latencia de inicio mayor que uno.

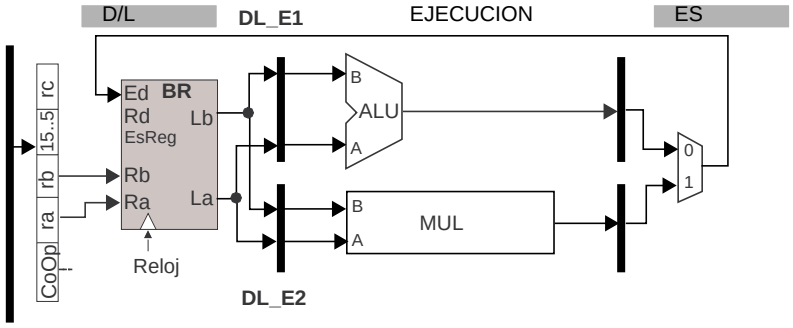


Figura 5.3 *Camino de datos paralelo o ramificado.*

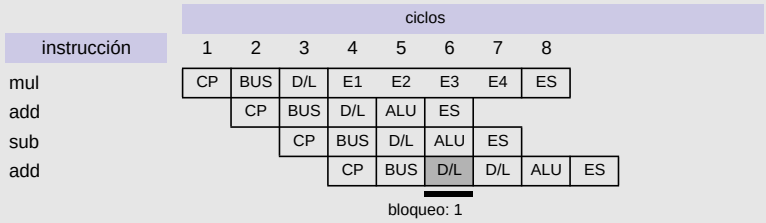
Notemos que en el diseño propuesto aún se comparte el camino de escritura al banco de registros, lo cual da lugar a riesgos estructurales. Sin embargo, ahora el número de ciclos perdidos es 1 por cada instrucción de multiplicación en lugar de los 3 ciclos que se perdían en el diseño de la Figura 5.2.

Ejercicio

Suponiendo el camino de datos de la Figura 5.3, muestre mediante un diagrama temporal la interpretación de la secuencia de instrucciones mul, add, sub, add.

Respuesta

En la siguiente figura se muestra la interpretación de la secuencia de instrucciones. La instrucción de multiplicación ocupa el registro de desacoplo DEC_E2 y permite que en el siguiente y sucesivos ciclos se pueda iniciar la ejecución de instrucciones que utilicen la ALU. Tres ciclos después de iniciar una instrucción de multiplicación hay que bloquear la interpretación de instrucciones, debido a que el camino de escritura al banco de registros es compartido y se produciría un riesgo estructural.



Notemos que las acciones que se efectúan en las etapas CP, BUS y D/L de un procesador segmentado multiciclo son independientes del tipo de instrucción. El paralelismo o ramificaciones en el camino de datos se centra en la fase de ejecución.

La utilización de caminos de datos paralelos en función del tipo de instrucción tiene ventajas. Cada camino puede adecuarse al tipo de instrucción, obteniéndose un hardware más eficiente. La latencia de actualización del banco de registros se adecúa a la latencia de cálculo, lo cual reduce la necesidad de caminos de cortocircuito y una instrucción utiliza todas las etapas que atraviesa. Además, si todos los riesgos entre diferentes tipos de instrucciones se gestionan antes de iniciar la ejecución, entonces, una vez iniciada la ejecución en cada camino individual, no pueden producirse bloqueos debidos a instrucciones en otros caminos. Esto permite distribuir y utilizar un control independiente por cada camino.

En este capítulo se utiliza la interpretación de instrucciones que operan con números representados en coma flotante para mostrar el diseño y compromisos de procesadores segmentados multiciclo. Otras instrucciones que requieren especialización del camino de datos son las de acceso a memoria. La latencia de estas instrucciones es variable, ya que depende del nivel de la jerarquía

de memoria en que se encuentra almacenado el dato al que se accede y esta latencia no se conoce cuando se inicia la ejecución de la instrucción.

El incremento de la frecuencia de funcionamiento de un procesador es una posibilidad ampliamente utilizada para reducir el tiempo de ejecución de los programas. Para ello es de utilidad una mayor especialización del hardware y a ello contribuye la utilización de ramificaciones por clase de instrucción.

Por otro lado, cuando el incremento de frecuencia es agresivo el número de etapas se incrementa, ya que el tiempo disponible en un ciclo para realizar la función lógica es menor. Las fases de interpretación que requieren incrementar el número de etapas, en un procesador multiciclo como el que se describe, son las de acceso a la cache de instrucciones y a la cache de datos. También, en ocasiones, es necesario utilizar más de una etapa en la fase de decodificación y en la fase de lectura o escritura del banco de registro. A estos procesadores se les ha denominado supersegmentados y usualmente utilizan más de 7 etapas para interpretar una instrucción de aritmética entera.

En la descripción que se efectúa en este capítulo sólo tendremos en cuenta el incremento del número de etapas en el acceso a la cache de datos. En la cache de instrucciones seguimos suponiendo que se tarda un ciclo.

Un mayor número de etapas incrementa la penalización de las instrucciones de secuenciamiento y también incrementa la latencia productor-uso. Para tolerar el incremento en la latencia productor-uso es necesario utilizar transformaciones de código que expongan el máximo paralelismo posible a nivel de instrucción. Una de las transformaciones que se describe permite eliminar las dependencias denominadas de nombre, las cuales son las dependencias $D(i) \cap R(i+k) \neq \varnothing$ y $R(i) \cap R(i+k) \neq \varnothing$). Estas dependencias son debidas a que una posición de almacenamiento se actualiza más de una vez. Sin embargo, entre la instrucción origen de la dependencia y la instrucción destino no se transmite información. En particular nos centraremos en las dependencias de nombre debidas al banco de registros. También se mostrará una transformación de código que incrementa el tamaño de los bloques básicos, con la esperanza de que el algoritmo de planificación de instrucciones tenga mayores oportunidades de selección en cada paso del algoritmo.

La metodología para determinar y enumerar los distintos tipos de riesgos y los cortocircuitos ha sido presentada en el Capítulo 3 y en el Capítulo 4. Entonces, este capítulo se centra en los mecanismos hardware utilizados para detectarlos en un procesador multiciclo. Igualmente, en el Capítulo 4 se ha presentado un algoritmo de planificación de instrucciones. Por tanto, en este capítulo nos centraremos en las técnicas de transformación de código para incrementar el paralelismo entre instrucciones. Por todo ello, una parte del capítulo se presenta como ejercicios donde deben aplicarse los conceptos descritos en los capítulos previos en el caso concreto de un procesador multiciclo o utilizar el algoritmo de planificación de instrucciones, después de aplicar alguna de las transformaciones de código que se describen.

INSTRUCCIONES DE COMA FLOTANTE

Las instrucciones de cálculo en coma flotante obtienen sus datos de registros y almacenan el resultado en un registro.

Para mover datos de memoria a registros y viceversa se utilizan las instrucciones de acceso a memoria.

Nosotros supondremos que hay un banco de registros específico para almacenar datos que se interpretan como números en coma flotante.

Las instrucciones de cálculo (CF) utilizan el banco de registros de coma flotante. Sin embargo, las instrucciones de acceso a memoria (FMEM) utilizan el banco de registros enteros para obtener el dato mediante el que se calcula la dirección de memoria y el banco de coma flotante para leer o escribir el dato que se escribe o lee de memoria.

Las instrucciones de secuenciamiento condicional de coma flotante (FIS) utilizan un registro de coma flotante para evaluar una condición y la semántica es igual que en las instrucciones de secuenciamiento condicional de aritmética entera.

El formato de las instrucciones de coma flotante es el mismo que el utilizado para instrucciones de aritmética entera (Figura 5.4). La salvedad es que los campos de identificador de registro deben interpretarse como identificadores de registros de coma flotante en

las operaciones de cálculo y secuenciamiento. En el caso de las instrucciones de acceso a memoria, el identificador de registro ra se refiere al banco de registros de coma flotante y el identificador de registro rb se refiere al banco de registros de enteros.

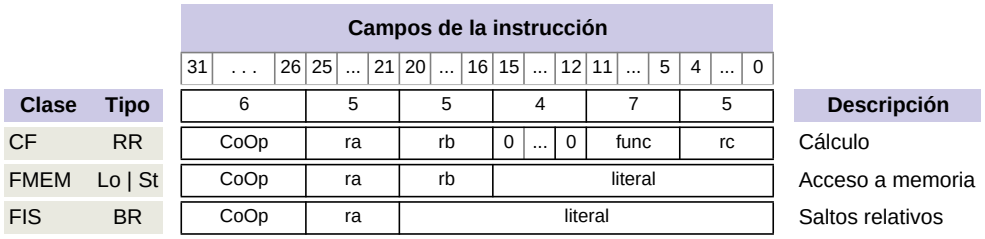


Figura 5.4 *Formato de las instrucciones de lenguaje máquina. Todas las instrucciones son de 32 bit de longitud con un campo de código de operación de 6 bits, ubicado en los bits <31:26> de la instrucción.*

Para mover datos entre bancos de registros no utilizaremos instrucciones específicas. En su lugar, para efectuar el movimiento, utilizaremos una posición de memoria como almacenamiento temporal e instrucciones store y load. Esto es, mediante una instrucción store, el contenido de un registro de un banco de registros se almacena en una posición de memoria. Posteriormente, mediante una instrucción load, que accede a la misma posición de memoria que la instrucción store, se lee el dato que se almacena en el otro banco de registros.

SEGMENTACIÓN EN ETAPAS

En la Figura 5.5 se muestra la segmentación genérica de una instrucción de coma flotante.

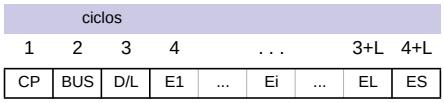


Figura 5.5 *Segmentación genérica de una instrucción de coma flotante. La unidad funcional puede o no estar segmentada.*

El número de ciclos de cálculo (L) y si la unidad funcional está segmentada depende de la operación concreta que efectúe la unidad funcional y de la implementación. La decisión de segmentar una unidad funcional depende de la complejidad y de la frecuencia de utilización en los programas. Las operaciones usuales en coma flotante son sumas algebraicas y multiplicaciones. Notemos que una secuencia de divisiones que utilizan el mismo denominador se puede transformar, por el compilador, en una división y una secuencia de multiplicaciones, como por ejemplo, en la resolución de un sistema de ecuaciones utilizando un método directo.

Las unidades funcionales se caracterizan por la latencia de cálculo o de ejecución y por la latencia de inicio o repetición. En la tabla de la Figura 5.6 se muestran valores típicos de estos parámetros.

OPERACION	LATENCIA DE EJECUCION	LATENCIA DE INICIO
suma algebraica	4	1
multiplicación	4	1
división	20	20

Figura 5.6 *Latencias típicas de ejecución y de inicio en unidades funcionales de coma flotante.*

Los acrónimos utilizados en lenguaje ensamblador para identificar las instrucciones de coma flotante son los mismos que en aritmética entera con el prefijo F. En el caso de los registros, en lugar del prefijo R, antes del ordinal, se utiliza el prefijo F.

En la Figura 5.7 se muestra una segmentación en etapas por tipo de instrucción. Las tres primeras etapas son independientes del tipo de instrucción y el paralelismo o ramificaciones empieza en la etapa posterior a D/L.

Las instrucciones de aritmética entera efectúan el cálculo en la ALU y actualizan el banco de registros de enteros en el siguiente ciclo.

Las instrucciones de acceso a memoria, tanto de enteros como de coma flotante, tienen una latencia de cálculo o ejecución de tres ciclos y en el siguiente ciclo las instrucciones load actualizan el banco de registros. Además, la latencia de inicio es 1 ciclo. Durante el primer ciclo se calcula la dirección efectiva y en los dos ciclos siguientes se accede a la cache, en la cual supondremos que siempre se acierta. En el primer ciclo de acceso a cache

supondremos que se está accediendo al campo etiquetas de la cache y en el segundo ciclo se accede al campo datos del contenedor. Sin embargo, como acrónimos de las etapas se utilizarán M1 y M2. Además, como acrónimo de la etapa en que se efectúa el cálculo de la dirección se utilizará “@”.

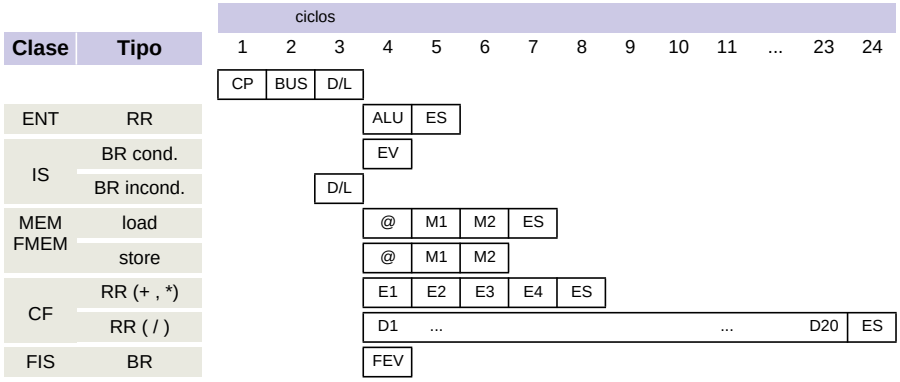


Figura 5.7 *Segmentación por tipo de instrucción.*

Las instrucciones de suma algebraica y multiplicación en coma flotante tienen una latencia de cálculo de 4 ciclos y una latencia de inicio de 1 ciclo. El banco de registros de coma flotante se actualiza en el ciclo que sigue al de finalización del cálculo. La instrucción de división en coma flotante no está segmentada.

Las instrucciones de secuenciamiento incondicional modifican el secuenciamiento en la etapa D/L (ciclo 3) y las instrucciones de secuenciamiento condicional modifican el secuenciamiento en la etapa EV o FEV (ciclo 4).

CAMINO DE DATOS

El camino de datos se corresponde con la segmentación descrita en la sección previa. En la descripción distinguiremos dos partes (Figura 5.8): a) suministro de instrucciones y b) ejecución. En primer lugar describiremos la parte de ejecución. Las instrucciones de secuenciamiento participan en el suministro y por ello se ha tramado parte de la ejecución.

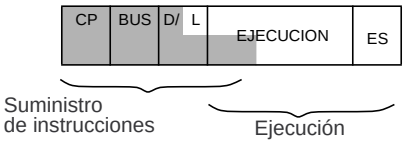


Figura 5.8 *Partes en que se divide la descripción del camino de datos de un procesador multiciclo.*

Cuando se inicia la ejecución de una instrucción por una ramificación hay que indicar a las otras ramificaciones que, al finalizar el ciclo de reloj, el registro de desacoplo de entrada no contiene información válida. Una excepción es la ramificación que efectúa operaciones de división en coma flotante. Si cuando se inicia la ejecución de una instrucción hay en curso una operación de división, entonces, no se envía la señal de operación no válida a la ramificación que efectúa operaciones de división.

Ejecución

En la Figura 5.9 se muestra, desde la etapa D/L, un camino de datos simplificado de un procesador multiciclo. Básicamente se muestra el flujo de información entre etapas excluyendo las líneas de control y otras señales como el identificador del registro de escritura. También obviaremos, en el esquema simplificado del camino de datos, las instrucciones de tipo RI de la clase ENT.

El camino de datos se ramifica en 4 caminos después de la etapa D/L, cada uno de ellos específico para un subconjunto determinado de instrucciones.

En la parte superior de la Figura 5.9 se distingue el camino que utilizan las instrucciones de tipo ENT y de secuenciamiento condicional de aritmética entera. Moviéndose hacia abajo se distingue la ramificación utilizada por las instrucciones tipo MEM, que se utiliza para mover datos entre memoria y los dos bancos de registros. Seguidamente hay un camino en el que se pueden efectuar operaciones de suma y multiplicación de números representados en coma flotante, además de instrucciones de secuenciamiento condicional de coma flotante. Finalmente se dispone de una ramificación para las instrucciones de división en coma flotante. Esta

última ramificación tiene una latencia de inicio igual a la latencia de ejecución y se han etiquetado los ciclos de duración como D1, .., D20.

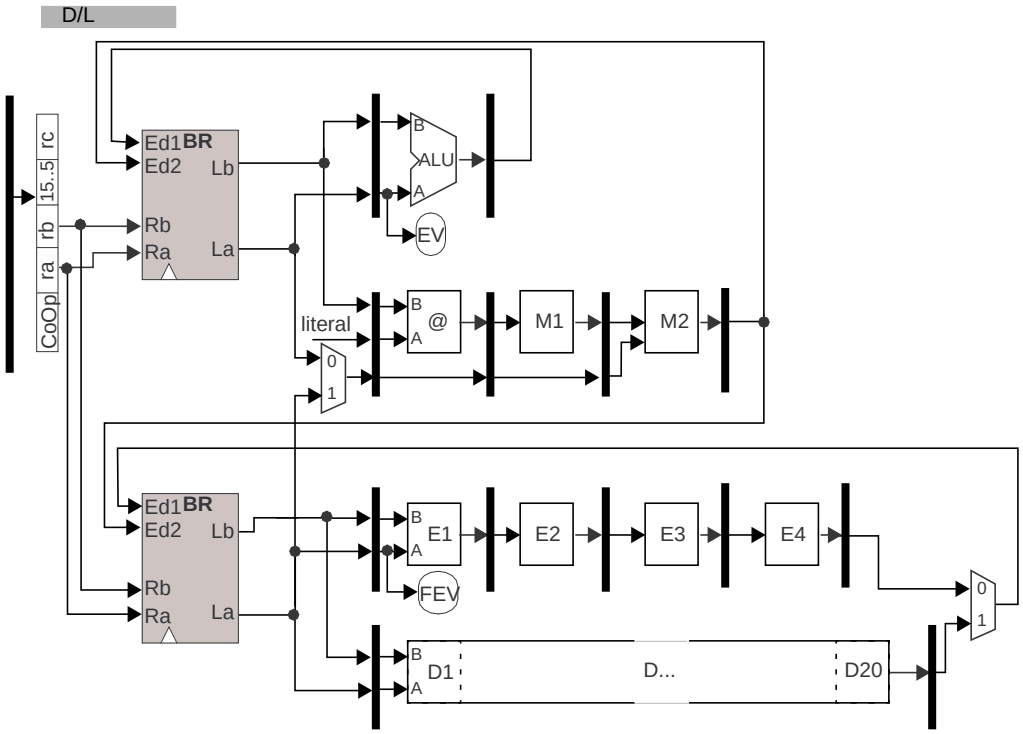


Figura 5.9 *Ramificaciones en el camino de datos.*

Cada uno de los dos bancos de registros tiene dos caminos de escritura. En el banco de registro de enteros uno de los caminos lo utilizan las instrucciones de tipo ENT y el otro lo utiliza la instrucción load de aritmética entera. En el banco de registros de coma flotante un camino de escritura lo utiliza la instrucción load de coma flotante y el otro camino de escritura está compartido por las unidades funcionales de coma flotante.

Cada banco de registros dispone de dos caminos de lectura. Notemos que hacia la ramificación de acceso a memoria se encamina un camino de lectura de cada uno de los bancos de registros. Este camino de lectura lo utiliza una instrucción store para obtener el dato que debe almacenarse en memoria.

Las instrucciones de acceso a memoria de coma flotante utilizan los dos bancos de registros. El banco de registros de aritmética entera se utiliza para leer un valor que sumado al literal de la instrucción formateado se utiliza para calcular la dirección efectiva de acceso a memoria. El banco de registros de coma flotante se utiliza para leer o escribir el valor que se escribe o lee de memoria.

Ejercicio

En el procesador segmentado multiciclo de la Figura 5.9 indique el número de instrucciones que pueden estar en la fase ejecución en un mismo ciclo.

Respuesta

En cada ciclo se inicia la ejecución de una instrucción. Por tanto, en la primera etapa de todas las ramificaciones segmentadas sólo puede haber una instrucción. Igualmente ocurre en las otras etapas. Entonces, el número máximo de instrucciones concurrentes en las ramificaciones segmentadas está determinado por la ramificación segmentada que tiene más etapas. Esta ramificación es la de suma y multiplicación en coma flotante.

Además de las ramificaciones segmentadas hay una ramiffi-cación no segmentada con una latencia de ejecución mayor que la latencia de ejecución máxima de las ramificaciones segmentadas. Por tanto, en esta ramificación puede haber una instrucción que se esté interpretando concurrentemente con instrucciones en las ramificaciones segmentadas.

Entonces, en total puede haber 4 + 1 = 5 instrucciones ejecutándose concurrentemente. En la siguiente figura se muestra la situación descrita.

	ciclos													
instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Fdiv	CP	BUS	D/L	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11
Fadd o Fmul		CP	BUS	D/L	E1	E2	E3	E4	ES					
Fadd o Fmul			CP	BUS	D/L	E1	E2	E3	E4	ES				
Fadd o Fmul				CP	BUS	D/L	E1	E2	E3	E4	ES			
Fadd o Fmul					CP	BUS	D/L	E1	E2	E3	E4	ES		
Fadd o Fmul						CP	BUS	D/L	E1	E2	E3	E4	ES	
Fadd o Fmul							CP	BUS	D/L	E1	E2	E3	E4	ES

Suministro de instrucciones

En la Figura 5.10 se muestran las etapas encargadas del suministro de instrucciones. Destaquemos que las instrucciones de secuenciamiento condicional pueden evaluar el contenido de un registro tanto del banco de registros de enteros como del banco de registros de coma flotante.

Excepto por la evaluación de una condición utilizando números representados en coma flotante, el secuenciamiento de instrucciones es idéntico al descrito en el Capítulo 4. En la etapa D/L se dispone de un sumador para calcular la dirección destino si el salto es tomado. Notemos que se ha ubicado espacialmente el multiplexor MUXBr en la etapa CP para simplificar el trazado de las señales en el dibujo.

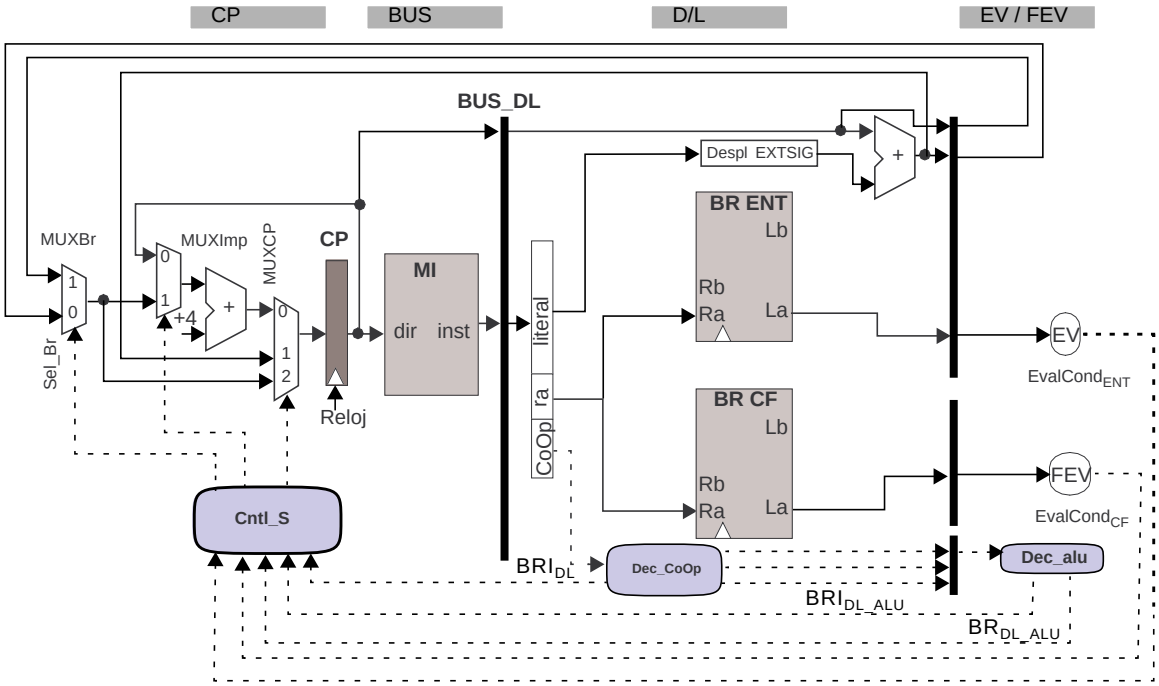


Figura 5.10 *Suministro de instrucciones en un procesador multiciclo.*

Las instrucciones de secuenciamiento incondicional modifican el secuenciamiento cuando están en la etapa D/L y las de secuenciamiento condicional un ciclo más tarde.

En la Figura 5.10 no se muestra la utilización de predicción fija de sentido en instrucciones de secuenciamiento. Su inclusión es mimética a la descrita en el Capítulo 4, teniendo en cuenta que además se evalúan condiciones utilizando números representados en coma flotante.

Gestión de riesgos

De forma semejante al procesador segmentado lineal, en el procesador multiciclo la parte correspondiente al secuenciamiento de instrucciones procesa las instrucciones en orden de programa.

La gestión de riesgos también es semejante. Esto es, cuando se detecta un riesgo en la etapa D/L se emula un funcionamiento serie. En el caso de un riesgo de datos las instrucciones se retienen en las etapas D/L y etapas previas hasta que desaparece el riesgo. En el caso de un riesgo de secuenciamiento, cuando no se utiliza predicción de sentido, se descartan las instrucciones buscadas después de la instrucción de secuenciamiento. Si se utiliza predicción fija de sentido se descartan las instrucciones predichas cuando la predicción es errónea. Entonces, el inicio del procesado de una instrucción en la parte de ejecución también es en orden de programa (Figura 5.11).

Sin embargo, como la latencia de ejecución de las instrucciones depende del tipo de instrucción, las instrucciones pueden finalizar en desorden de programa. Esta circunstancia requiere gestionar los riesgos debidos a dependencias de salida $R(i) \cap R(i+k) \neq \varnothing$.

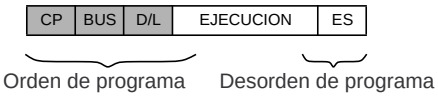


Figura 5.11 *Etapas donde se mantiene el orden de programa y etapa donde se puede modificar el orden de programa.*

Instrucción lista. Una instrucción está lista para iniciar la ejecución cuando no existen riesgos (estructurales y de datos) que lo impidan.

Para contabilizar sin ambigüedad los ciclos perdidos en los diagramas temporales y hacerlo de forma pareja al procesador lineal, extenderemos de forma ficticia los ciclos de interpretación de las instrucciones después de finalizar la interpretación. En

concreto, dada una secuencia de instrucciones elegiremos como latencia de interpretación de todas las instrucciones la de la instrucción con mayor latencia. Entonces, cuando una instrucción tenga una latencia menor que la máxima añadiremos, al final de la interpretación, los ciclos necesarios para obtener la máxima latencia de interpretación. Estos ciclos de extensión no se etiquetan, ya que en ellos no se efectúa ninguna acción. Ahora bien, para identificarlos los tramaremos. Por ejemplo, supongamos una secuencia de instrucciones donde hay instrucciones load, Fmul y add. La latencia máxima de interpretación se corresponde con la instrucción Fmul. Entonces, la latencia de interpretación de las instrucciones load y add se extiende en 1 ciclo y 3 ciclos respectivamente (Figura 5.12). Posteriormente, al analizar los riesgos se describirá con mayor detalle la utilización de esta artimaña para identificar los ciclos perdidos y calcular el CPI de una secuencia de instrucciones.

	ciclos							
instrucción	1	2	3	4	5	6	7	8
add	CP	BUS	D/L	ALU	ES			
load	CP	BUS	D/L	@	M1	M2	ES	
Fmul	CP	BUS	D/L	E1	E2	E3	E4	ES

Figura 5.12 *Extensión de la latencia de interpretación para contabilizar los ciclos perdidos.*

Como hemos comentado, las situaciones de riesgo se gestionan de forma semejante al procesador lineal. En la Figura 5.13 se muestra de forma esquemática el control de la segmentación. En el esquema también se muestra la utilización de caminos de cortocircuito para reducir la latencia productor-uso.

En el diseño del control están separadas la lógica de detección de riesgos de datos y la lógica de control de los cortocircuitos.

- La lógica que determina los bloqueos por riesgos de datos supone la existencia de los cortocircuitos
- La lógica de activación de los cortocircuitos actúa independientemente de las condiciones de bloqueo, ya sean riesgos estructurales o de datos.

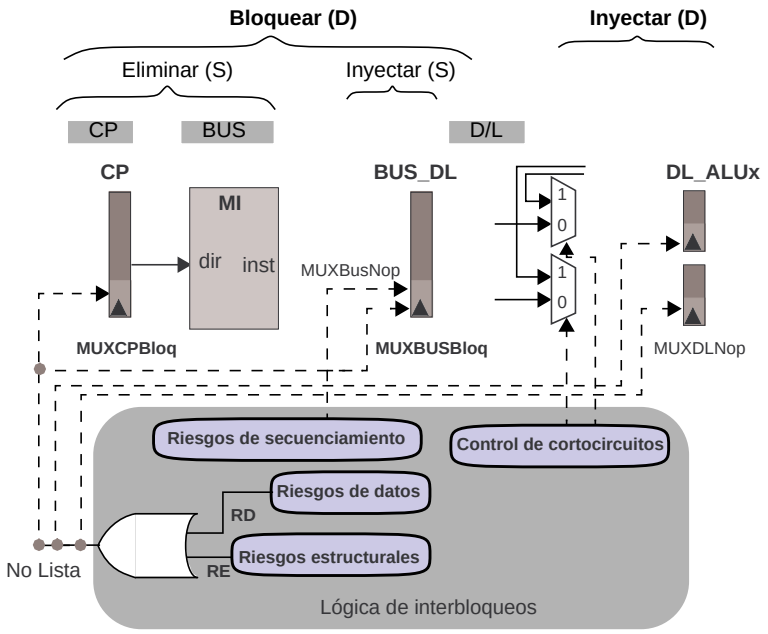


Figura 5.13 *Control de riesgos y cortocircuitos en el procesador segmentado multiciclo. Las etiquetas S y D indican respectivamente riesgo de secuenciamiento y de datos.*

Notemos que después de la etapa D/L hay un registro de desacoplo por cada ramificación y cada uno de ellos se controla de forma independiente. En la Figura 5.13 las señales para cada ramificación se muestran después de la puerta OR, aunque se entiende que se determinan de forma específica para cada ramificación.

En las siguientes secciones se describen los riesgos estructurales y los riesgos de datos. Así mismo, se describe la gestión de dichos riesgos. Los riesgos de secuenciamiento y su gestión ya han sido descritos en el Capítulo 4. Es suficiente ampliar de forma trivial la gestión para el caso de instrucciones de secuenciamiento condicional en coma flotante.

RIESGOS ESTRUCTURALES

En el camino de datos de la Figura 5.9 se pueden producir dos tipos de riesgos estructurales cuando se interpreta una secuencia de instrucciones. Uno de ellos es debido a la unidad funcional de división en coma flotante, que tiene una latencia de iniciación mayor que uno y el otro riesgo estructural es debido al camino de escritura al banco de registros de coma flotante, que comparten las unidades de coma flotante.

Mientras no se indique lo contrario, en los ejercicios, para reducir el tamaño de los diagramas temporales, utilizaremos, en el caso de las unidades funcionales de coma flotante, las latencias que se indican en la tabla de la Figura 5.14.

OPERACION	LATENCIA DE EJECUCION	LATENCIA DE INICIO
suma algebraica	4	1
multiplicación	4	1
división	6	6

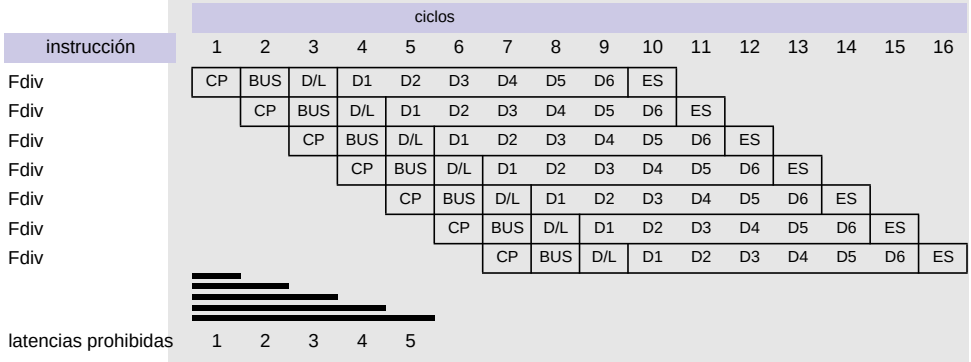
Figura 5.14 *Tabla de latencias de coma flotante utilizada en los ejercicios.*

Ejercicio

Indique las latencias prohibidas si hay que interpretar una secuencia de instrucciones de división en coma flotante.

Respuesta

Como la unidad funcional de división no está segmentada y la latencia de cálculo es 6, las latencias prohibidas son: {1, 2, 3, 4, 5}. En el siguiente diagrama temporal se observan estas latencias.



Ejercicio

Indique los casos en que se produce un riesgo estructural debido al camino compartido de escritura al banco de registros de coma flotante.

Respuesta

La latencia de actualización del banco de registros por parte de las instrucciones de división y suma o multiplicación son 6 y 4 ciclos respectivamente. Entonces, hay una latencia prohibida de valor 2 después de iniciar la ejecución de una instrucción de división. En el siguiente diagrama temporal se observa esta latencia prohibida.

	ciclos											
instrucción	1	2	3	4	5	6	7	8	9	10	11	12
Fdiv	CP	BUS	D/L	D1	D2	D3	D4	D5	D6	ES		
Fmul		CP	BUS	D/L	E1	E2	E3	E4	ES			
Fmul			CP	BUS	D/L	E1	E2	E3	E4	ES		
Fmul				CP	BUS	D/L	E1	E2	E3	E4	ES	
Fmul					CP	BUS	D/L	E1	E2	E3	E4	ES
latencia prohibida	2											

Los dos tipos de riesgos estructurales se gestionan de forma distinta. Para gestionar el riesgo debido a unidades funcionales no segmentadas utilizaremos un vector de bits, que denominaremos vector de ocupadas (VO). Cada posición del vector identifica una unidad funcional no segmentada. El bit correspondiente a una unidad funcional se activa cuando la instrucción ocupa la unidad funcional (primer ciclo de ejecución) y se desactiva en el ciclo previo a la actualización del banco de registros (Figura 5.15). Suponemos que el vector de bits se puede escribir y leer, en este orden, en un ciclo de la señal de reloj. La decisión de utilizar un bit que se activa y desactiva, una vez transcurrida una latencia, se sustenta en el hecho de que el conjunto de valores de latencias prohibidas es consecutivo.

En la etapa D/L se lee del vector VO el bit correspondiente a la unidad funcional que utilizará la instrucción. Si el valor leído es 1 se ha detectado un riesgo estructural.

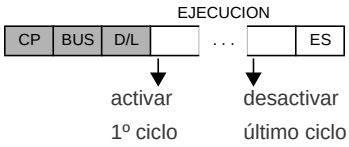


Figura 5.15 *Instantes de activación y desactivación del bit correspondiente a una unidad funcional en el vector VO.*

En el procesador segmentado multiciclo que describimos el vector VO es un bit, ya que sólo hay una unidad funcional no segmentada.

Cuando se detecta un riesgo estructural, debido a una unidad funcional no segmentada, el registro de desacoplo de entrada de la ramificación correspondiente está ocupado por los datos que alimentan a la unidad funcional. En estas condiciones no se debe inyectar una instrucción nop por esta ramificación. Entonces, para observar en los diagramas temporales los ciclos de bloqueo se utiliza una nueva fila por cada ciclo de bloqueo. También, se calculará la latencia de interpretación máxima de las instrucciones que se representan en el diagrama temporal. Entonces, en cada fila, que represente a una instrucción retenida en la etapa D/L, se marcan, después de la etapa D/L, los ciclos necesarios para obtener la máxima latencia de interpretación.

También, se extienden de forma ficticia, después de la etapa ES, los ciclos de interpretación de una instrucción que no tenga la latencia de interpretación máxima. El número de ciclos ficticios que se añaden es la diferencia entre la latencia de interpretación máxima y la latencia de interpretación de la instrucción.

Como hemos comentado previamente, mediante la artimaña descrita todas las instrucciones aparentan tener la misma latencia de interpretación.

Completar la interpretación de una instrucción. Por completar la interpretación de una instrucción entendemos el último ciclo representado en el diagrama temporal, el cual puede ser un ciclo ficticio.

Entonces, los ciclos perdidos se asocian a los ciclos en los cuales no se completa la interpretación de una instrucción.

En la Figura 5.16 se muestra un diagrama temporal donde se detecta un riesgo estructural.

Los ciclos perdidos en la Figura 5.16 son el 14 y el 15. En estos ciclos no completa la interpretación de ninguna instrucción y se corresponden con el bloqueo de la segunda instrucción de división en la etapa D/L (ciclos 7 y 8).

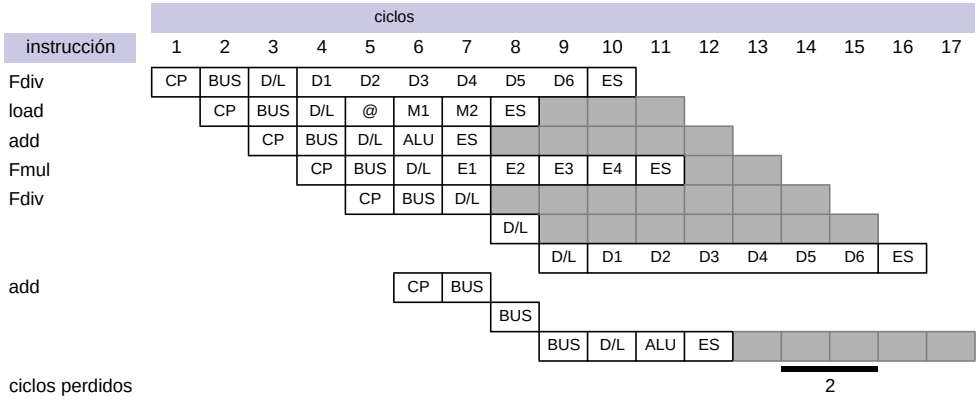


Figura 5.16 *Diagrama temporal donde se muestra la acción de bloqueo cuando se detecta un riesgo estructural debido a una unidad funcional no segmentada.*

En la Figura 5.17 se muestra la misma secuencia de instrucciones que en la Figura 5.16 utilizando un diagrama simplificado. Los ciclos perdidos son los ciclos en que no se completa una instrucción.

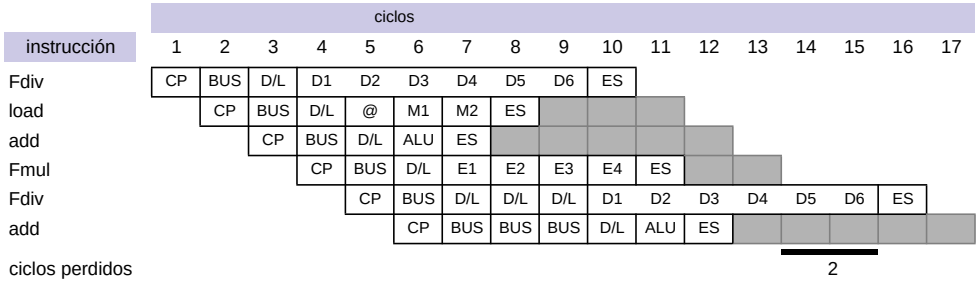


Figura 5.17 *Diagrama temporal simplificado donde se muestra la acción de bloqueo cuando se detecta un riesgo estructural debido a una unidad funcional no segmentada.*

La gestión del riesgo estructural debido al camino de escritura al banco de registros de coma flotante se efectúa de la forma descrita en el Capítulo 2. Cuando se inicia la ejecución de una instrucción, que puede tener un riesgo estructural con una instrucción más jóven, se activa una señal que permite contabilizar una latencia prohibida. En la etapa D/L se observa la salida del circuito de latencias prohibidas. Entonces, en función del tipo de instrucción que ocupa la etapa D/L se decide que prosigue la interpretación o que se bloquea.

Cuando se detecte un riesgo estructural debido al camino de escritura al banco de registros hay que bloquearse en la etapa D/L e inyectar una instrucción nop por la ramificación por la cual se iniciaría la instrucción si no existiera el riesgo estructural.

Ejercicio

En un procesador multiciclo, además de varias unidades funcionales con latencia de inicio igual a uno, se dispone de 3 unidades funcionales no segmentadas (UF1, UF2, UF3) con latencias de 4, 5 y 10 ciclos. Cada una de las unidades funcionales no segmentadas tiene un camino de escritura independiente al banco de registros. Indique el tamaño del vector de ocupadas (VO) y el número de caminos de lectura y escritura a este vector. Así mismo diseñe la lógica que indica la situación de riesgo estructural.

Respuesta

El tamaño del vector VO es de tres entradas, una entrada para cada una de las unidades funcionales no segmentadas.

En el procesador multiciclo se inicia una instrucción por ciclo. Entonces, el número de caminos de lectura necesarios al vector VO es uno y se utiliza en la etapa D/L.

En el procesador multiciclo descrito, en el mismo ciclo, puede finalizar una instrucción en cada UF no segmentada, ya que las latencias de ejecución son distintas. Entonces, el número de caminos de escritura necesario es tres y cada unidad funcional utiliza su camino de escritura al vector VO en el primero y en el último ciclo de cálculo.

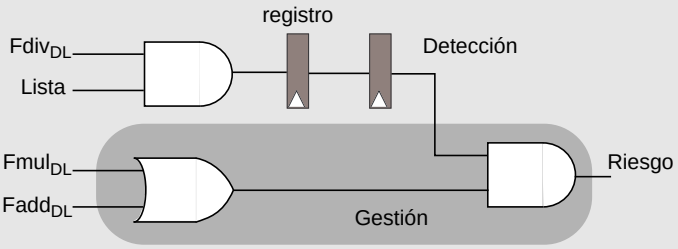
Ejercicio

Diseñe, para el procesador multiciclo descrito en este capítulo, la lógica que gestione el riesgo estructural debido al camino de escritura al banco de registros de coma flotante. Suponga que en la etapa D/L el decodificador suministra las señales $Fadd_{DL}$, $Fmul_{DL}$ y $Fdiv_{DL}$ para indicar, respectivamente, si la etapa está ocupada por una instrucción de suma, multiplicación o de división. También, en la etapa D/L se dispone de la señal Lista que indica cuando una instrucción puede iniciar la ejecución. Elija una

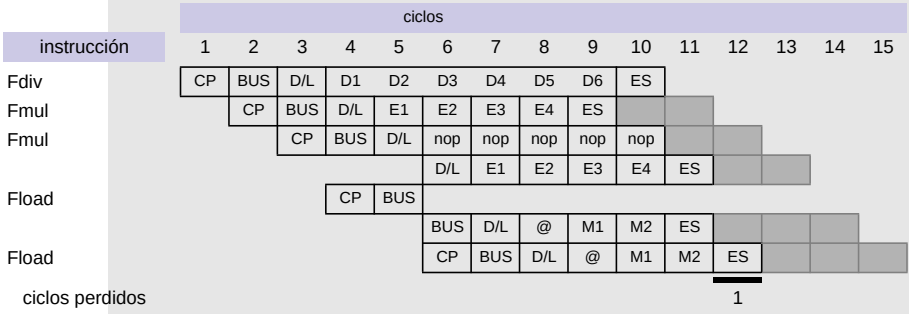
secuencia de instrucciones donde se produzca el riesgo estructural y muestre la inyección de instrucciones nop en el camino de datos cuando se produce una situación de bloqueo.

Respuesta

En la siguiente figura se muestra la lógica de detección y gestión del riesgo estructural. Cuando se inicia la ejecución de una instrucción de división en coma flotante se empieza a contabilizar la latencia prohibida, la cual es 2. Entonces, cuando en la etapa D/L hay una instrucción de suma o multiplicación en coma flotante y se detecta una latencia prohibida hay que bloquear el inicio de ejecución de instrucciones.



En la siguiente figura se muestra la interpretación de una secuencia de instrucciones. Se observa que en el ciclo 5 se detecta un riesgo estructural debido al camino de escritura al banco de registros de coma flotante. La instrucción Fmul se retiene durante 1 ciclo en la etapa D/L y en el siguiente ciclo puede proseguir la interpretación ya que no se detecta ningún riesgo estructural. El ciclo perdido se contabiliza en el ciclo en el que completa la instrucción nop inyectada.



CORTOCIRCUITOS Y RIESGOS DE DATOS

Como hemos comentado: a) la lógica que determina los bloqueos por riesgos de datos supone la existencia de los cortocircuitos y b) la lógica de activación de los cortocircuitos actúa independientemente de las condiciones de bloqueo, ya sean riesgos estructurales o de datos (Figura 5.18).

En primer lugar se describen los cortocircuitos utilizados. Posteriormente se analizan los riesgos de datos suponiendo que existen cortocircuitos.

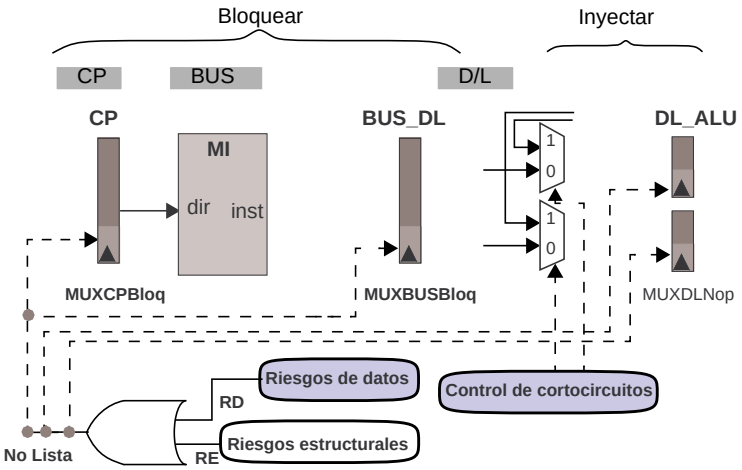


Figura 5.18 *Lógica de interbloqueos: control de los bloqueos por riesgo y de los caminos de cortocircuito.*

De forma semejante al procesador lineal, cuando se detecte una situación de riesgo de datos se retiene a la instrucción que ocupa la etapa D/L y a las instrucciones que ocupan las etapas previas en las etapas que ocupan. Adicionalmente inyectaremos una instrucción nop por la ramificación por la cual se iniciaría la instrucción si no existiera el riesgo de datos.

Cortocircuitos

El banco de registros se actualiza en el ciclo siguiente al de finalización de un cálculo y mediante caminos de cortocircuito se reduce la latencia productor-uso en un ciclo. La etapa donde se dispone del dato calculado por una unidad funcional se muestra en la tabla de la Figura 5.19. A estas etapas las denominamos

productoras y el dato está disponible antes de finalizar el ciclo de reloj. Suponemos que la comunicación se efectúa hacia el final del ciclo de reloj (parte derecha de la Figura 5.19).

Como etapa destino de un cortocircuito sólo consideraremos la etapa D/L. Por tanto, para proseguir la interpretación, una instrucción debe disponer en la etapa D/L de todos los datos que se requieren en la fase de ejecución.

UNIDAD FUNCIONAL	ETAPA o CICLO
ALU	ALU
MD	M2
CF +, x	E4
CF /	ciclo D20

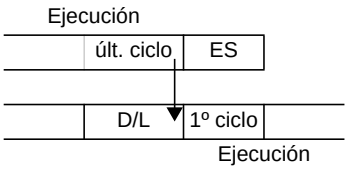


Figura 5.19 *Unidades funcionales productoras e instante de comunicación.*

Utilizando la metodología descrita en el Capítulo 4 se obtiene una enumeración detallada de los cortocircuitos utilizados en el procesador multiciclo. En la Figura 5.20 se muestra una síntesis de las posibles comunicaciones entre las etapas, por tipo de instrucción, que permiten reducir la latencia productor-uso en un ciclo.

Como ejemplo de comunicación describimos los cortocircuitos desde memoria en la Figura 5.20. La línea etiquetada como M2 tiene su origen en la etapa M2 y su destino es la etapa D/L. En la etapa D/L los destinos son cada uno de los caminos que transportan los datos fuente a las unidades funcionales.

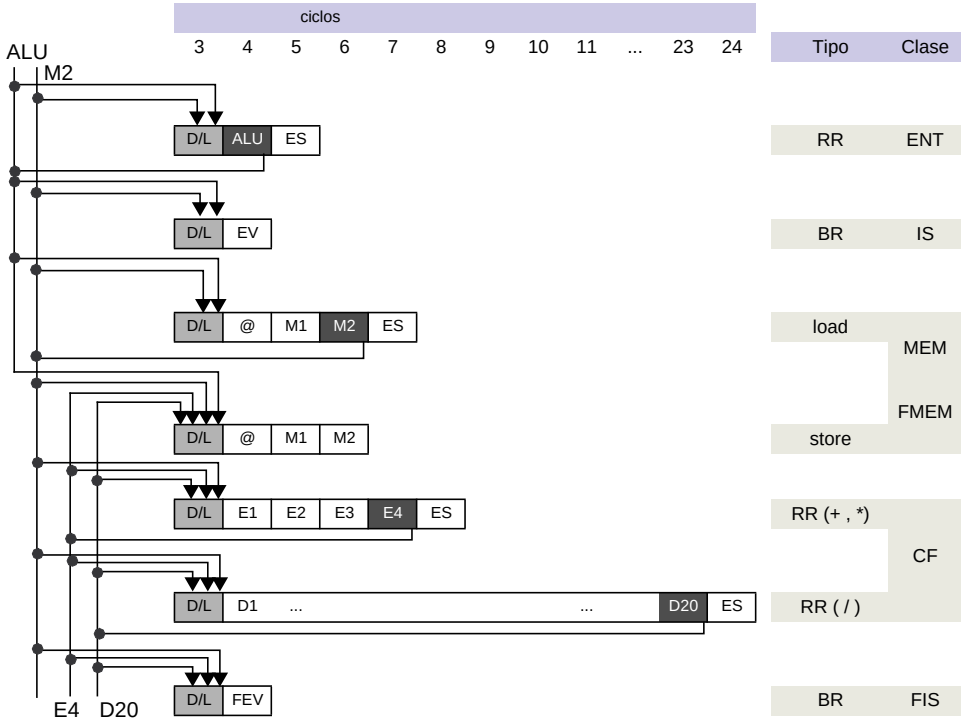


Figura 5.20 *Comunicación entre etapas productoras y consumidora por tipo de instrucción.*

Riesgos de datos

En lo referente al flujo de instrucciones de acceso a memoria no se pueden producir riesgos de datos, ya que las instrucciones actualizan o leen memoria en el mismo ciclo, relativo al inicio de interpretación de la instrucción.

De igual forma que en el procesador segmentado lineal, el riesgo de datos debido a una dependencia $D(i) \cap R(i+k) \neq \varnothing$ no se puede producir, ya que todas las instrucciones leen los datos en el mismo ciclo respecto el inicio de interpretación (etapa D/L) y la actualización del banco de registros por parte de una instrucción (i+k) siempre es posterior a la lectura del banco de registros por parte de una instrucción más vieja (i). Además, el mecanismo de gestión

de riesgos bloquea en la etapa D/L a una instrucción que detecta un riesgo y a las instrucciones más jóvenes en las etapas previas a la etapa D/L.

Seguidamente describimos los riesgos de datos debidos a registros.

Riesgos R(i)∩D(i+k)≠ ∅

Se puede producir un riesgo lectura después de escritura, debido a una dependencia de datos $R(i) \cap D(i+k) \neq \varnothing$, suponiendo que se utilizan caminos de cortocircuito, cuando la latencia de cálculo o ejecución de la instrucción productora es mayor que 1 ciclo.

En la Figura 5.21 se muestra un riesgo $R(i) \cap D(i+k) \neq \varnothing$. La instrucción load tiene una latencia de ejecución de 3 ciclos. Por tanto, si la siguiente instrucción utiliza el dato leído por la instrucción load esta debe retenerse 2 ciclos en la etapa D/L.

Para observar los ciclos de bloqueo, cada vez que se retenga una instrucción en la etapa D/L por un riesgo de datos se inyecta una instrucción nop por la ramificación que utilizaría la instrucción retenida. En el diagrama temporal de la Figura 5.21 se muestra este hecho utilizando una nueva fila para representar a la instrucción retenida. En la nueva fila se inicia la representación, desde la etapa D/L, en el ciclo siguiente al de retención. En la fila previa se muestra la inyección de la instrucción nop. Adicionalmente, se añaden ciclos ficticios de la misma forma que se ha descrito para los riesgos estructurales. Observemos que, debido a los ciclos ficticios que se añaden, podemos decir que en la Figura 5.21 completa una instrucción en cada ciclo. Identificamos el ciclo perdido cuando la instrucción que completa es una instrucción nop inyectada.

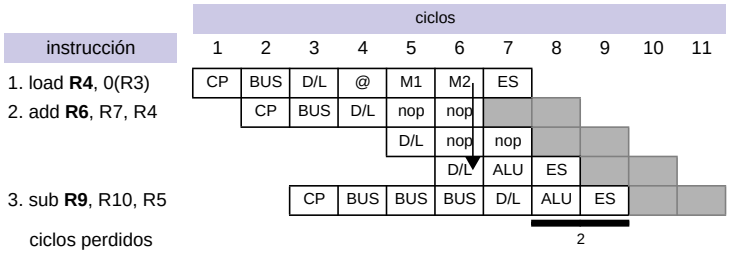


Figura 5.21 *Riesgo de datos lectura después de escritura.*

En la Figura 5.22 se muestra el diagrama simplificado de la secuencia de instrucciones utilizada en la Figura 5.21. Los ciclos perdidos se corresponden con los ciclos en que no se completa una instrucción.

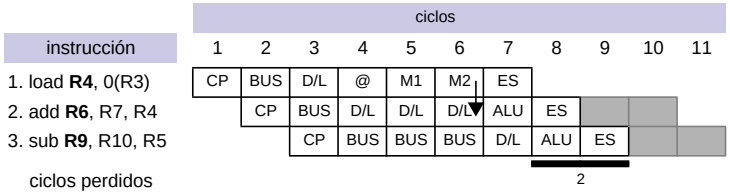


Figura 5.22 *Diagrama temporal simplificado cuando se produce un riesgo de datos lectura después de escritura.*

Ejercicio

Suponga que una instrucción productora tiene una latencia de ejecución L. Indique los ciclos perdidos si la primera instrucción consumidora está a distancia 1, 2, 3, 4, . . ., L-1, L, L+1, L+2. Suponga también, cuando la distancia es mayor que uno, que las instrucciones que hay entre la instrucción productora y la instrucción consumidora no detectan riesgos.

Respuesta

Los ciclos perdidos se calculan como la latencia de ejecución de la instrucción productora menos la distancia a la que se encuentra la instrucción consumidora.

	ciclos perdidos
DISTANCIA	LATENCIA - DISTANCIA
1	L -1
2	L - 2
. . .	. . .
i	L - i
. . .	. . .
L-1	L - (L - 1) = 1
L	L - L = 0

Cuando la DISTANCIA es mayor o igual que la LATENCIA la dependencia no produce ningún riesgo y no se pierde ningún ciclo.

Riesgos R(i)∩R(i+k)≠ ∅

En un procesador segmentado multiciclo se pueden producir riesgos de datos $R(i) \cap R(i+k) \neq \varnothing$ debido al banco de registros, ya que las instrucciones actualizan el banco de registros en ciclos distintos, contando desde el inicio de interpretación de la instrucción.

En la Figura 5.23 se muestra un riesgo $R(i) \cap R(i+k) \neq \varnothing$. Las dos primeras instrucciones actualizan el mismo registro y una instrucción más joven utiliza el contenido del registro como dato.

instrucción	ciclos 1	2	3	4	5	6	7	8	9
1. load **R4**, 0(R3)	CP	BUS	D/L	@	M1	M2	ES		
2. add **R4**, R7, R6		CP	BUS	D/L	ALU	ES			
3. sub **R6**, R9, R2			CP	BUS	D/L	ALU	ES		
4. or **R1**, R10, R1				CP	BUS	D/L	ALU	ES	
5. sub **R2**, R11, R4					CP	BUS	D/L	ALU	ES

Figura 5.23 *Riesgo de datos escritura después de escritura.*

El programador espera que la 5ª instrucción lea el valor calculado por la 2ª instrucción. Sin embargo, la segmentación modifica el orden especificado por el programador para actualizar el registro R4, en primer lugar escribe la 2ª instrucción y un ciclo después la 1ª instrucción. Entonces, la 5ª instrucción utiliza como dato el valor calculado por la 1ª instrucción, lo cual es incorrecto.

Notemos también que si la 4ª instrucción fuera “or R1, R10, R4” leería el valor producido por la 2ª instrucción, lo cual es correcto.

Ejercicio

Muestre las situaciones de riesgo $R(i) \cap R(i+k) \neq \varnothing$ que pueden producirse entre instrucciones de coma flotante que escriben en el mismo registro. Suponga que las instrucciones que se interpretan entre la instrucción origen de la dependencia y la instrucción destino no detectan riesgos.

Respuesta

Para que se produzca un riesgo de datos $R(i) \cap R(i+k) \neq \varnothing$ la instrucción más vieja (origen) tiene que tener una latencia de ejecución mayor. Sean L_V y L_J las latencias de ejecución de la instrucción vieja y joven respectivamente y sea D la distancia entre las instrucciones. Si se cumple la siguiente expresión $D \leq L_V - L_J$ hay un riesgo $R(i) \cap R(i+k) \neq \varnothing$.

En el siguiente diagrama temporal se muestra la instrucción Fdiv seguida de instrucciones Fload. Se produce un riesgo en las distancias {1, 2, 3}.

instrucción	ciclos 1	2	3	4	5	6	7	8	9	10
1. Fdiv **F4**,F3, F2	CP	BUS	D/L	D1	D2	D3	D4	D5	D6	ES
2. Fload **F4**, 0(R6)		CP	BUS	D/L	@	M1	M2	ES		
3. Fload **F4**, 0 (R3)			CP	BUS	D/L	@	M1	M2	ES	
4. Fload **F4**, 0 (R7)				CP	BUS	D/L	@	M1	M2	ES

Cuando una instrucción Fdiv es seguida por instrucciones Fmul o Fadd se produce riesgo en el conjunto de distancias {1, 2}. Cuando una instrucción Fmul o Fadd es seguida por instrucciones Fload el riesgo se produce a distancia 1.

Actualizar dos veces un registro sin mediar una utilización del dato calculado por la instrucción más vieja es una situación poco usual. Esto es, entre dos instrucciones que actualizan el mismo registro usualmente existe una instrucción que utiliza el dato calculado por la primera instrucción y por tanto, antes de detectar la dependencia $R(i) \cap R(i+k) \neq \varnothing$ se detecta una dependencia $R(i) \cap D(i+k) \neq \varnothing$. Entonces, como la gestión del riesgo lectura después de escritura requiere bloquear la interpretación de instrucciones no se produce el riesgo $R(i) \cap R(i+k) \neq \varnothing$. En la Figura 5.24 se muestra un ejemplo. La 1ª instrucción es una instrucción de multiplicación en aritmética entera, con latencia de ejecución de 5 ciclos. El camino de datos para ejecutar la instrucción no se ha mostrado en la Figura 5.9. Entre la 1ª y 3ª instrucción hay una dependencia de datos lectura después de escritura. También hay una dependencia $R(i) \cap R(i+k) \neq \varnothing$ entre la 1ª y la 4ª instrucción. La dependencia lectura después de escritura determina un riesgo y se pierde un ciclo. La dependencia $R(i) \cap R(i+k) \neq \varnothing$ no produce ningún riesgo.

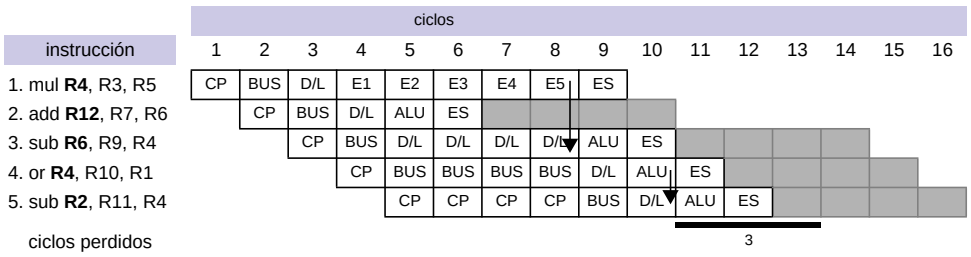


Figura 5.24 *Dependencia escritura después de escritura que no produce un riesgo.*

En la Figura 5.25 se muestra una secuencia de instrucciones donde, en función del sentido del salto, se produce un riesgo $R(i) \cap R(i+k) \neq \varnothing$, ya que suponemos que la latencia de la instrucción Fdiv es 8 ciclos. En concreto, cuando se sigue en secuencia se produce un riesgo $R(i) \cap R(i+k) \neq \varnothing$.

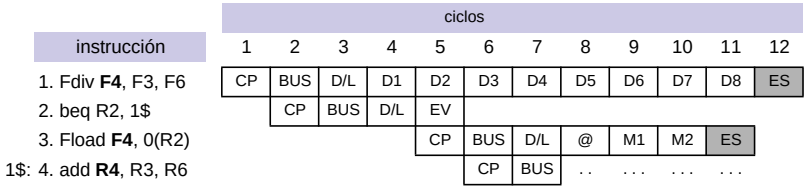


Figura 5.25 *Riesgo de datos escritura después de escritura en función del sentido del salto.*

La solución que adoptaremos para gestionar el riesgo $R(i) \cap R(i+k) \neq \varnothing$ es conservadora. Se bloquea la interpretación de la instrucción que ocupa la etapa D/L y de las instrucciones que ocupan las etapas previas, si existe en fase de ejecución una instrucción que actualiza el mismo registro destino que la instrucción que ocupa la etapa D/L. La instrucción que ocupa la etapa D/L se retiene en esta etapa hasta que la instrucción fuente del riesgo está en el ciclo previo al de escritura (Figura 5.26).

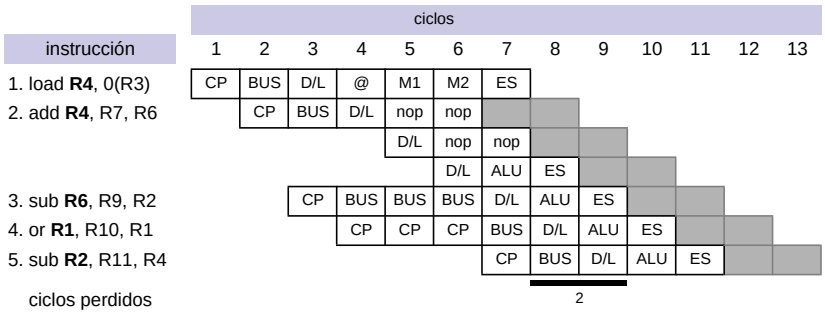


Figura 5.26 *Bloqueo cuando se detecta un riesgo de datos escritura después de escritura.*

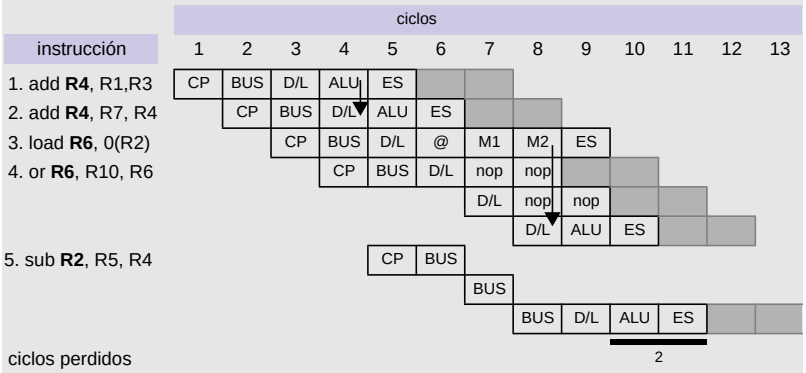
Ejercicio

Utilice un diagrama temporal para mostrar la interpretación de la siguiente secuencia de instrucciones.

1. add **R4**, R1,R3
2. add **R4**, R7, R4
3. load **R6**, 0(R2)
4. or **R6**, R10, R6
5. sub **R2**, R5, R4

Respuesta

En la siguiente figura se muestra el diagrama temporal.



La 1ª y 2ª instrucción actualizan el mismo registro. Además la 2ª instrucción tiene una dependencia $R(i) \cap D(i+k) \neq \varnothing$ con la 1ª instrucción. Sin embargo, como la latencia de ejecución de la 1ª instrucción es 1 ciclo no se detecta ningún riesgo de datos.

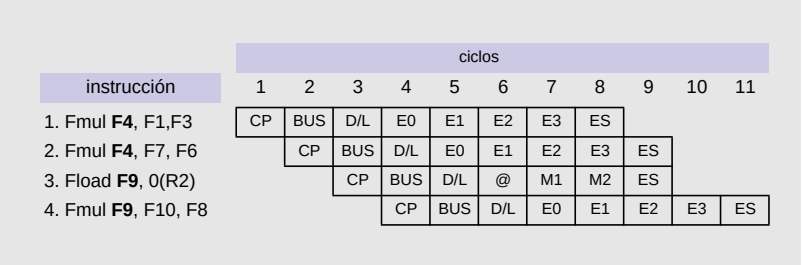
La 4ª instrucción utiliza el mismo registro destino que la 3ª instrucción. Sin embargo, la 4ª instrucción también utiliza el registro R6 como registro fuente. Se detectan riesgos de datos $R(i) \cap D(i+k) \neq \varnothing$ y $R(i) \cap R(i+k) \neq \varnothing$ al interpretar la 4ª instrucción, debido a que la 3ª instrucción tiene una latencia de ejecución de 3 ciclos. En el ciclo 8 desaparece el riesgo $R(i) \cap D(i+k) \neq \varnothing$ y no se detecta el riesgo $R(i) \cap R(i+k) \neq \varnothing$, ya que la 3ª instrucción actualiza el banco de registros en el siguiente ciclo.

Ejercicio

Elija una secuencia de instrucciones que muestre que la solución adoptada para gestionar el riesgo $R(i) \cap R(i+k) \neq \varnothing$ es conservadora.

Respuesta

En la siguiente figura se muestra una secuencia de cuatro instrucciones donde se observa que no es necesario retener a la 2ª instrucción en la etapa D/L, ya que la latencia de ejecución de la 1ª instrucción es igual y por tanto actualiza el banco de registros un ciclo antes. Igualmente no es necesario retener a la 4ª instrucción ya que su latencia es mayor que la de la 3ª instrucción.



CÁLCULO DEL CPI

En un procesador lineal todas las instrucciones tienen la misma latencia de interpretación. Entonces, para calcular el CPI de una secuencia de instrucciones aislada suponíamos que antes y después de la secuencia se estaban interpretando instrucciones. Además, la evolución temporal de la secuencia no estaba afectada por la interpretación de instrucciones previas. De esta forma, en el cálculo del CPI no se tenía en cuenta la latencia de interpretación de la primera instrucción de la secuencia de instrucciones (carga de la segmentación) y sólo se observaba la finalización de las instrucciones.

En un procesador multiciclo las instrucciones pueden tener distinta latencia de interpretación, debido a que la latencia de ejecución es distinta. En este contexto la utilización de ciclos ficticios en los diagramas temporales permite identificar los ciclos perdidos y calcular el CPI de la misma forma que en el procesador lineal. Por un lado, mediante la utilización de los ciclos ficticios, la latencia de ejecución de la instrucción con mayor latencia se imputa a los ciclos de carga de la segmentación (Figura 5.27). Por otro lado, los ciclos perdidos imputables a una instrucción no quedan enmascarados debido a que instrucciones más viejas tienen latencias de ejecución mayores (Figura 5.28).

En la Figura 5.27 se muestra una secuencia de instrucciones que se interpreta sin riesgos. La latencia de interpretación de las instrucciones es distinta y actualizan el estado del procesador en desorden. Utilizando la artimaña de añadir ciclos ficticios observamos que en cada ciclo completa una instrucción. Por tanto, el CPI es igual a uno.

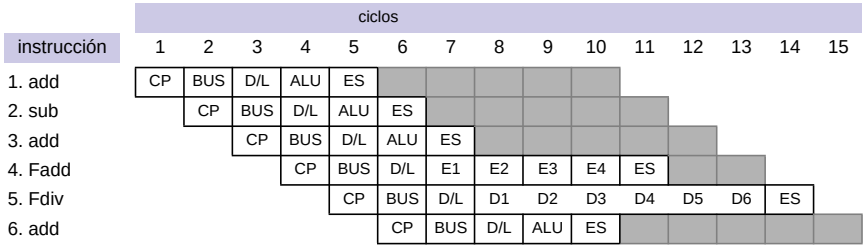


Figura 5.27 *Artimaña de añadir ciclos ficticios para calcular el CPI.*

En la Figura 5.28 se muestra como el añadir ciclos ficticios nos permite identificar de forma sencilla los casos en que la latencia de ejecución de una instrucción consumidora es menor que la latencia de ejecución de la instrucción productora y se han perdido ciclos (parte izquierda de la figura). En la parte derecha de la Figura 5.28 se muestra el caso de instrucciones con la misma latencia.

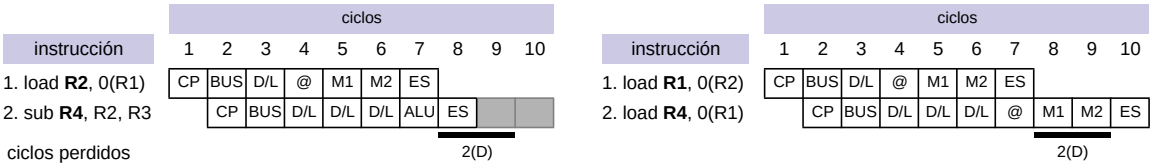


Figura 5.28 *Identificación de ciclos perdidos.*

INSTRUCCIÓN PREPARADA

En esta sección se describe la lógica utilizada para detectar los riesgos de datos debidos a registros.

Dato disponible. El dato está almacenado en el banco de registros o se puede obtener mediante un cortocircuito.

Instrucción preparada. Una instrucción está preparada para iniciar la ejecución cuando sus datos están disponibles y está libre de otros riesgos de datos.

Recordemos que una **instrucción está lista** para iniciar la ejecución cuando no existen riesgos que lo impidan (datos y estructurales).

La lógica que utilizaremos para determinar la disponibilidad de los datos requeridos por una instrucción es sustancialmente diferente de la lógica utilizada en el Capítulo 4. Se dejan de utilizar comparadores para comparar identificadores de registros y se utilizan vectores de bit. El cambio es debido a que el número de comparadores necesarios es proporcional, en el mejor de los casos, a la mayor de las latencia de ejecución. En cambio, el número de entradas del vector de bits es igual al número de registros lógicos. Por otro lado, también es necesario detectar los riesgos $R(i) \cap R(i+k) \neq \varnothing$.

Nosotros utilizaremos, por cada banco de registros, un vector de bits que denominaremos vector de marca (VM). Cada bit de un vector VM está asociado a un registro lógico e indica la disponibilidad o no del contenido del registro. Esto es, si la instrucción más jóven, que utilizaba este registro como registro destino, que es más vieja que la instrucción que está en la etapa D/L ya ha almacenado el valor en el registro, o ya se conoce el valor y se puede obtener mediante un cortocircuito. Cuando sea necesario, para especificar el banco de registro al que se refiere un vector VM utilizaremos los subíndices EN (enteros) y CF (coma flotante).

El valor 1 en un bit de un vector VM indica que el contenido del registro se está obteniendo y el valor cero indica que el valor está disponible.

En la Figura 5.29 se muestran las operaciones de actualización del vector VM. La entrada del vector VM correspondiente al registro destino de la instrucción se activa (1) en el primer ciclo de ejecución de una instrucción y se desactiva (0) en el último ciclo de ejecución.

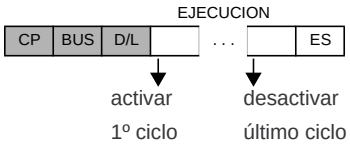


Figura 5.29 *Instantes de activación y desactivación del bit correspondiente al registro destino de una instrucción en el vector VM.*

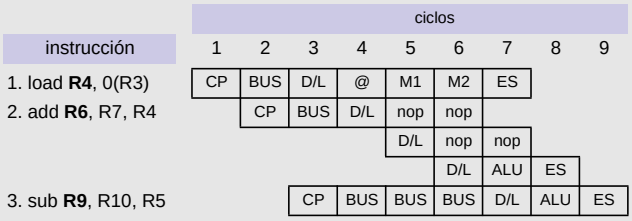
En estas condiciones, si la latencia productor-uso de una instrucción es igual a uno se produce una situación peculiar. En el mismo ciclo hay que indicar que el contenido del registro se está

calculando, pero a la vez hay que indicar que el valor está disponible al final del ciclo. Por tanto, las instrucciones con latencia productor-uso igual a uno no actualizan el vector VM.

Supondremos que en un ciclo de la señal de reloj se puede, en este orden, escribir y leer una entrada del vector VM.

Ejercicio

Dado el siguiente diagrama temporal indique el contenido del vector VM_{ENT} en los ciclos 4, 5, 6, 7, 8 y 9. Utilizando el vector VM justifique los ciclos de bloqueo.



Respuesta

En la siguiente tabla se muestran los bits del vector VM correspondientes a los registros utilizados en la secuencia de instrucciones que se muestra en el diagrama temporal.

VM_{ENT}	ciclos					
registro	4	5	6	7	8	9
R3	0	0	0	0	0	0
R4	1	1	0	0	0	0
R5	0	0	0	0	0	0
R6	0	0	0	0	0	0
R7	0	0	0	0	0	0
R10	0	0	0	0	0	0

En el ciclo 4 se inicia la ejecución de la 1ª instrucción. Por tanto, se activa, en el vector VM, el bit correspondiente al registro R4. Este bit se desactiva en el ciclo 6. En los ciclos 7 y 8 se inicia la ejecución de instrucciones de latencia igual a uno y no se actualiza el vector VM.

Como el bit correspondiente al registro R4 tiene el valor uno en los ciclos 4 y 5, la 2ª instrucción no tiene los operandos disponibles y se bloquea la interpretación de instrucciones en estos dos ciclos.

Ejercicio

Indique el ciclo de actualización de un vector VM cuando el procesador multiciclo no dispone de cortocircuitos.

Respuesta

El ciclo de activación es el mismo. Todas las instrucciones que actualizan un registro destino, incluidas las de latencia de ejecución igual a uno, activan el bit correspondiente al registro destino en un vector VM.

Sin embargo, el ciclo de desactivación se retarda un ciclo. El bit correspondiente al registro destino se desactiva al inicio del ciclo de escritura.

Riesgo de datos lectura después de escritura

Se produce un riesgo $R(i) \cap D(i+k) \neq \varnothing$ cuando en la etapa D/L una instrucción no puede disponer de un dato debido a que aún no ha sido producido.

Para conocer si los datos están disponibles se lee el vector VM, en la etapa D/L, utilizando los identificadores de registro fuente de la instrucción (Figura 5.30).

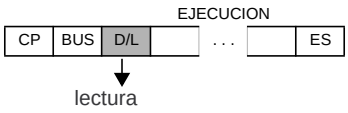


Figura 5.30 *Instante de lectura del vector VM.*

En la Figura 5.31 se muestra la lógica que determina si una instrucción tiene los datos disponibles. Un valor igual a 1 en una entrada del vector VM indica que el registro no está disponible. Entonces, el valor 1 en la salida del circuito indica que los datos están disponibles. Se utilizan los campos de la instrucción (ra y rb) para leer la posición correspondiente de los vectores VM.

Dependiendo del tipo de instrucción los campos ra y rb no se interpretan como identificadores de registro. Por ello, se utilizan las señales de validación, que indican si los campos ra y rb deben interpretarse como identificadores de registro y si estos campos son identificadores del banco de registros de enteros o de coma flotante (ENT_{val}, CF_{val}). Estas señales de validación están disponibles en la etapa D/L y las suministra el decodificador.

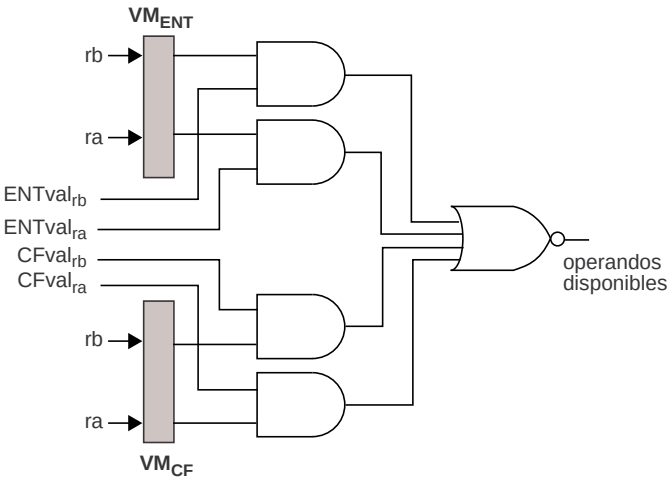


Figura 5.31 *Lógica que determina la disponibilidad de los datos, almacenados en el banco de registros, utilizados por una instrucción.*

Ejercicio

El decodificador suministra las señales BR, BRI, LD, ST, RR, ENT y CF cuya interpretación se indica en la siguiente tabla.

Señal	Interpretación	Señal	Interpretación
BR	instrucción de secuenciamiento	ST	instrucción store o Fstore
BRI	instrucción de secuenciamiento incondicional	ENT	instrucción de aritmética entera
LD	instrucción load o Fload	CF	instrucción de coma flotante
RR	instrucción de cálculo		

Diseñe la lógica que evalúa las señales de validación utilizadas en la Figura 5.31.

Respuesta

El formato de las instrucciones nos indica como deben interpretarse los campos ra y rb. En la siguiente tabla se relacionan los distintos tipos de instrucciones. En la parte izquierda se relacionan los tipos de aritmética entera (ENT) y en la parte derecha los tipos de coma flotante (CF). En este último caso se indica también si el identificador de registro se refiere al banco de registros de

aritmética entera (BR ENT) o de coma flotante (BR CF). Cuando un campo está tramado no se interpreta como identificador de registro.

ENT

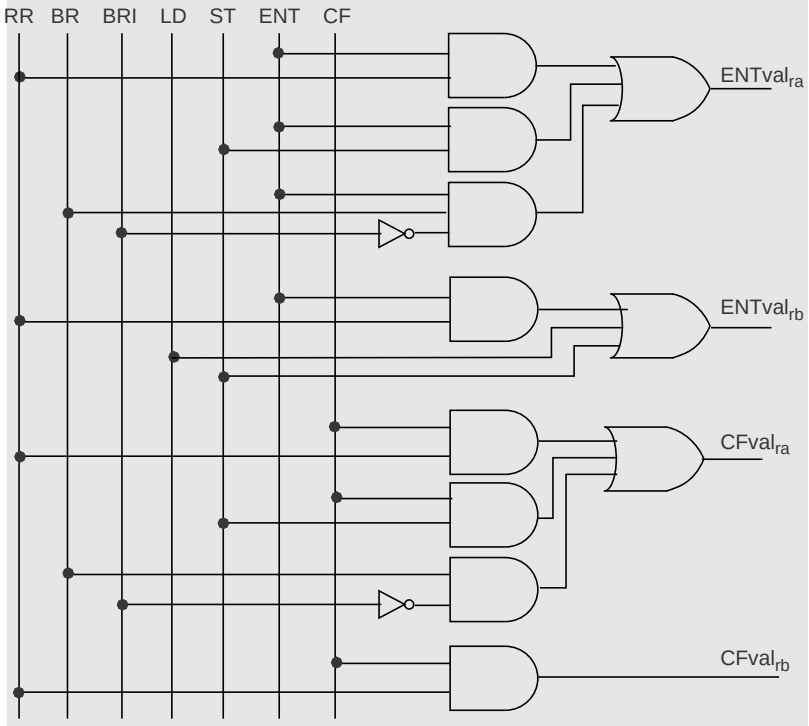
	BR ENT					
	25	...	21	20	...	16
Tipo	ra			rb		
RR	X			X		
Load				X		
Store	X			X		
BR	X					
BRI						
	$ENTval_{ra}$			$ENTval_{rb}$		

CF

	BR ENT						BR CF					
	25	...	21	20	...	16	25	...	21	20	...	16
Tipo	ra			rb			ra			rb		
RR							X			X		
Load				X								
Store				X			X					
BR							X					
BRI												
				$ENTval_{rb}$			$CFval_{ra}$			$CFval_{rb}$		

En la última fila de la tabla se indica la señal de validación en la que debe tenerse en cuenta las marcas de la columna.

A partir de la tabla previa se determinan las señales de validación que se muestran en la siguiente figura.



Por ejemplo, la señal de validación $ENTval_{ra}$ se activa cuando se cumple alguno de los siguientes casos: a) la instrucción es de tipo RR y es ENT, b) la instrucción es de tipo store y es ENT y c) la instrucción es de tipo secuenciamiento condicional y es ENT.

Riesgo de datos escritura después de escritura

Se puede producir un riesgo de datos $R(i) \cap R(i+k) \neq \varnothing$ cuando dos instrucciones en curso de interpretación actualizan el mismo registro. Para gestionar el riesgo hemos elegido una solución conservadora, bloquear la interpretación de instrucciones cuando una instrucción más vieja actualiza el mismo registro que la instrucción que ocupa la etapa D/L.

Para detectar un riesgo de datos $R(i) \cap R(i+k) \neq \varnothing$ se lee el vector VM utilizando el identificador de registro destino de la instrucción que está en la etapa D/L. Si el valor leído es un 1 existe una instrucción interpretándose que actualiza el mismo registro. En la Figura 5.32 se muestra la lógica para detectar el riesgo. Debemos recordar que el identificador de registro destino es el campo ra o rc en función del tipo de instrucción. Entonces, se utilizan los campos ra y rc para leer dos bits de los vectores VM. Posteriormente se selecciona un bit utilizando las señales MuxDE y MuxDF. Las señales de validación indican si el campo ra o rc debe interpretarse como un identificador de registro destino.

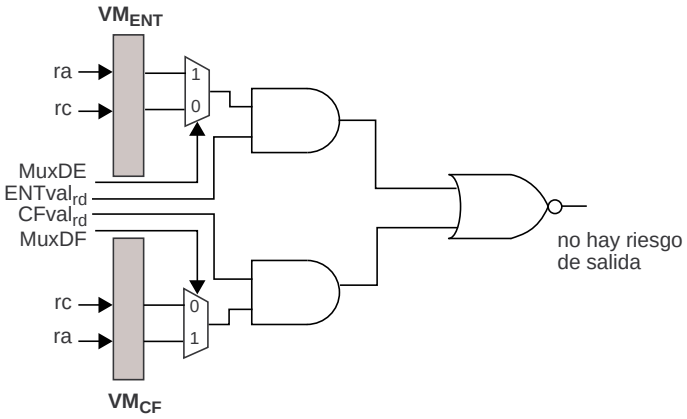
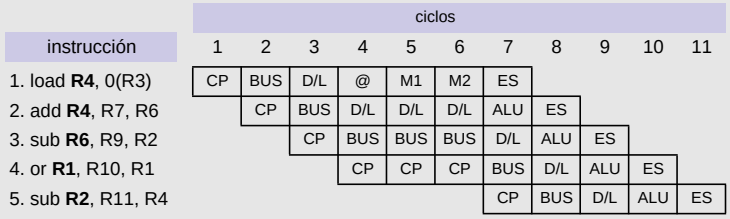


Figura 5.32 *Lógica para detectar riesgos de datos escritura después de escritura.*

Ejercicio

Dado el siguiente diagrama temporal indique el contenido del vector VM_{ENT} en los ciclos 4, 5, 6, 7, 8 y 9. Utilizando el vector VM justifique los ciclos de bloqueo.



instrucción	ciclos 1	2	3	4	5	6	7	8	9	10	11
1. load **R4**, 0(R3)	CP	BUS	D/L	@	M1	M2	ES				
2. add **R4**, R7, R6		CP	BUS	D/L	D/L	D/L	ALU	ES			
3. sub **R6**, R9, R2			CP	BUS	BUS	BUS	D/L	ALU	ES		
4. or **R1**, R10, R1				CP	CP	CP	BUS	D/L	ALU	ES	
5. sub **R2**, R11, R4							CP	BUS	D/L	ALU	ES

Respuesta

En la siguiente tabla se muestran los bits del vector VM correspondientes a los registros destino utilizados en la secuencia de instrucciones que se muestra en el diagrama temporal.

VM_{ENT} registro	ciclos 4	5	6	7	8	9
R1	0	0	0	0	0	0
R2	0	0	0	0	0	0
R4	1	1	0	0	0	0
R6	0	0	0	0	0	0

En el ciclo 4 se inicia la ejecución de la 1ª instrucción. Por tanto, se activa, en el vector VM, el bit correspondiente al registro R4. Este bit se desactiva en el ciclo 6. En los ciclos 7, 8 y 9 se inicia la ejecución de instrucciones de latencia igual a uno y no se actualiza el vector VM.

En los ciclos 4 y 5 al leer el vector VM detectamos que una instrucción más vieja escribe en el mismo registro. Entonces, hay que bloquear la interpretación de instrucciones para gestionar el riesgo escritura después de escritura.

Ejercicio

El decodificador suministra las señales LD, RR, ENT y CF cuya interpretación se indica en la siguiente tabla.

Señal	Interpretación	Señal	Interpretación
LD	instrucción load o Fload	ENT	instrucción de aritmética entera
RR	instrucción de cálculo	CF	instrucción de coma flotante

Diseñe la lógica que evalúa las señales de control de los multiplexores de la Figura 5.32. Así mismo, diseñe la lógica que evalúa las señales de validación utilizadas en la misma figura.

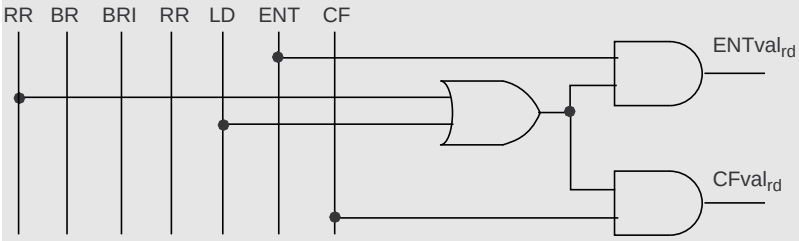
Respuesta

En la siguiente tabla se muestran los tipos de instrucciones en que los campos ra o rc se interpretan como registro destino.

	ENT		CF	
	25 ... 21	4 ... 0	25 ... 21	4 ... 0
Tipo	ra	rc	ra	rc
RR		X		X
Load	X		X	

La señal de control de los multiplexores es LD (MuxDe y MuxDF). Cuando esta señal está activada el multiplexor tiene en su salida el bit correspondiente a la lectura efectuada con el campo ra.

A partir de la tabla previa se obtienen las señales de validación que se muestran en la siguiente figura.



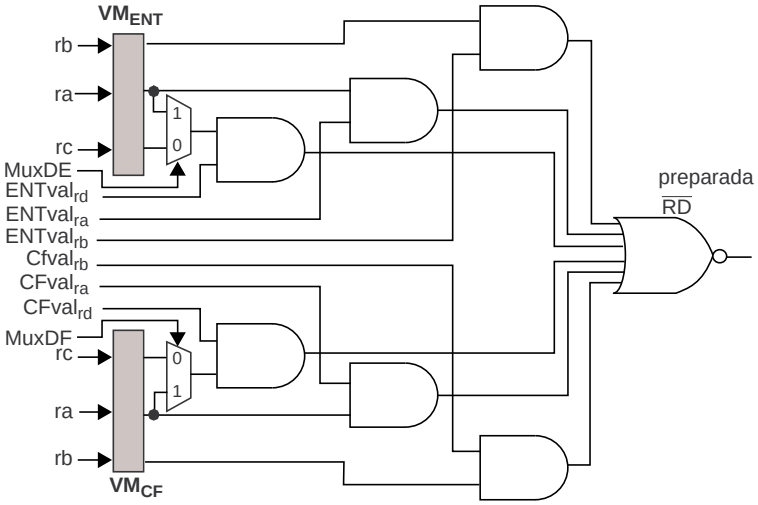
Por ejemplo, la señal ENTval$_{rd}$ se activa cuando se cumple que la instrucción es de aritmética entera (ENT) y es una instrucción load (LD) o una instrucción tipo RR.

Ejercicio

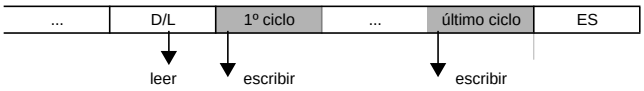
Diseñe la lógica que determina si una instrucción está preparada (señal complementaria de RD en la Figura 5.13 y en la Figura 5.18). Esto es, si está libre de riesgos de datos. Para ello utilice la lógica diseñada para detectar riesgos lectura después de escritura y escritura después de escritura. Así mismo, indique el número de caminos de lectura y escritura en los vectores VM.

Respuesta

En la siguiente figura se muestra la lógica de detección de riesgo de datos. El valor 1 en la salida de la lógica indica que no hay riesgo de datos. Esto es, la instrucción está preparada.



El vector VM se lee en la etapa D/L para determinar si los datos están disponibles. Cuando se inicia la ejecución de una instrucción se escribe en un vector VM para indicar que se está calculando el dato que se almacenará en el registro destino. En el último ciclo de ejecución de una instrucción se escribe en un vector VM para indicar que el dato está disponible.



Una instrucción puede tener 2 registros fuente y un registro destino.

En la etapa D/L y en el 1º ciclo de ejecución sólo puede haber una instrucción. Entonces, cada vector VM requiere 3 caminos de lectura para leer, si es necesario, los bits correspondientes a los 2 registros fuente y el bit correspondiente al registro destino.

Las instrucciones que utilizan la ALU tienen latencia de ejecución igual a uno y no actualizan el vector VM_{ENT}. Las únicas instrucciones que actualizan el vector VM_{ENT} son las instrucciones load. Por tanto, como puede haber una instrucción load en la 1ª etapa de ejecución y otra instrucción load más vieja en la etapa M2 son necesarios 2 caminos de escritura al vector VM_{ENT}.

Todas las instrucciones de coma flotante actualizan el vector VM_{CF}, ya que tienen una latencia de ejecución mayor que uno. En un ciclo determinado sólo puede haber una instrucción en el último ciclo de ejecución de cualquiera de las dos ramificaciones de coma flotante que ejecutan instrucciones de tipo RR, ya que las dos ramificaciones comparten el camino de escritura al banco de registros.

Por otro lado, en el mismo ciclo que una instrucción de tipo RR de coma flotante está en el último ciclo de ejecución, una instrucción Fload puede estar en la etapa M2.

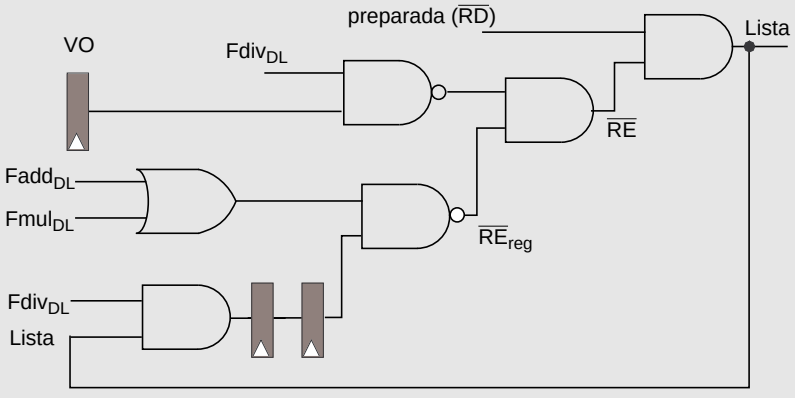
Por tanto, como puede haber una instrucción de coma flotante en el 1º ciclo de ejecución y puede haber dos instrucciones en el último ciclo de ejecución son necesarios 3 caminos de escritura al vector VM_{CF}.

Ejercicio

Diseñe la lógica que permite determinar que una instrucción está lista para iniciar la fase de ejecución. La información disponible es: a) la señal $\overline{RD}$ que indica si una instrucción está preparada, b) el vector VO, las señales $Fadd_{DL}$, $Fmul_{DL}$ y $Fdiv_{DL}$ que indican respectivamente que en la etapa D/L hay una instrucción de suma, multiplicación o división de coma flotante.

Respuesta

Una instrucción está lista para iniciar la fase de ejecución cuando está libre de riesgos estructurales y de datos.



La señal $\overline{RE}_{reg}$ que indica que no hay riesgo en la utilización del camino de escritura al banco de registros de coma flotante. La señal $\overline{RE}$ indica que no hay ningún riesgo estructural. Esto es: a) si la instrucción que ocupa la etapa D/L es una división la ramifi-

cación que efectúa divisiones de coma flotante no debe estar ocupada y b) el ciclo actual no se corresponde con una latencia prohibida debida al camino de escritura compartido al banco de registros.

CONTROL DE LOS CORTOCIRCUITOS

En el procesador multiciclo que se está presentando, un camino de cortocircuito permite reducir la latencia productor-uso en 1 ciclo. Esto es, mediante un camino de cortocircuito se puede disponer del valor un ciclo antes de que se actualice el banco de registros.

En la Figura 5.33 se muestra parte del camino de datos del procesador multiciclo con los caminos de cortocircuito. El destino de los cortocircuitos se ha ubicado, en la etapa D/L, después de la lectura del banco de registros.

Para acceder al valor suministrado por un cortocircuito hay que controlar los multiplexores que permiten seleccionar entre el valor leído del banco de registros y el valor transportado por el camino de cortocircuito.

La etapa origen de un cortocircuito es la última etapa o ciclo de cálculo de una ramificación del camino de datos. La etapa destino de un cortocircuito es la etapa D/L. En esta etapa se distinguen 8 cortocircuitos, 4 de ellos ($XDLMA_B$, $XDLAA_B$, $XDLMA_A$, $XDLAA_A$) en las ramificaciones de ALU y memoria y los 4 restantes ($XDLM1_B$, $XDLCF1_B$, $XDLM1_A$, $XDLCF1_A$) en las ramificaciones de coma flotante. Notemos que las ramificaciones de coma flotante comparten el mismo camino de cortocircuito (multiplexor MUXCO). Ello es posible porque el control de riesgos estructurales prohibe que las dos ramificaciones actualicen el banco de registros en el mismo ciclo y el control del riesgo estructural se efectúa reteniendo a la instrucción en la etapa D/L.

La señal de control del multiplexor MUXCO es la misma señal de control del multiplexor MUXBR, pero se utiliza en el ciclo anterior. Esta señal se ha generado en la etapa D/L teniendo en cuenta el tipo de operación.

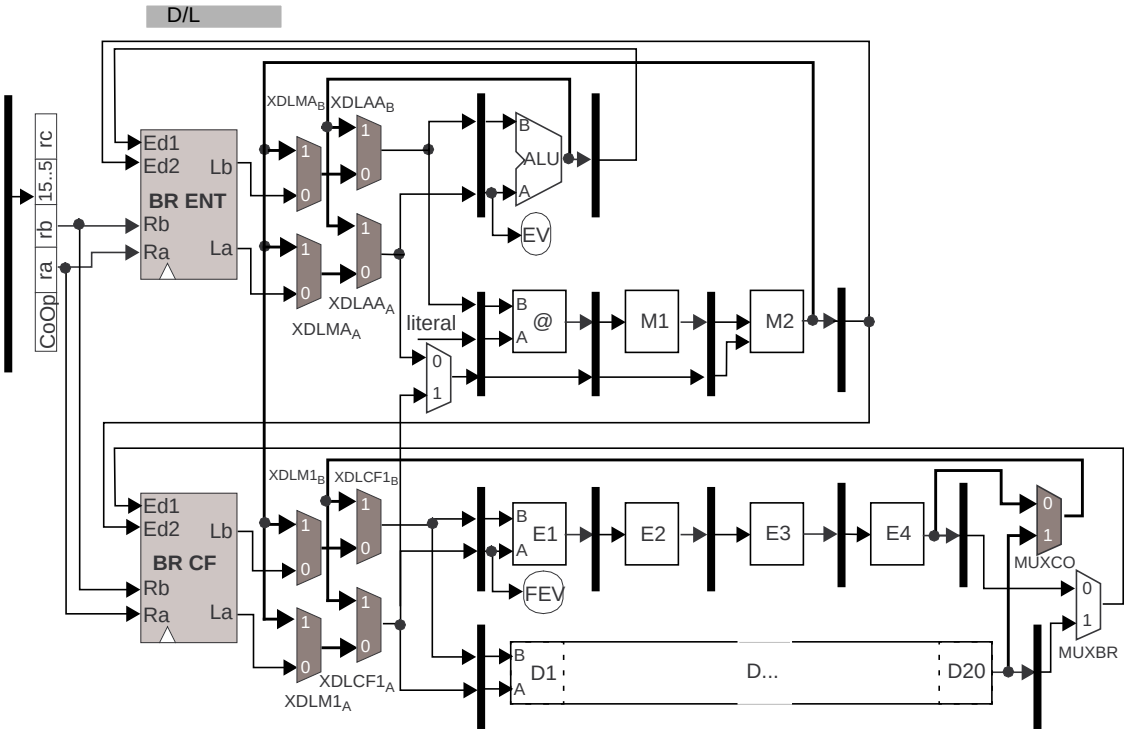


Figura 5.33 *Camino de datos de un procesador multiciclo con cortocircuitos.*

El control de un multiplexor de cortocircuito se determina comparando el identificador de registro destino de la instrucción que está en la etapa origen del cortocircuito con el identificador del registro fuente de la instrucción que está en la etapa D/L. Supondremos que los identificadores de registro destino se propagan por las etapas en cada ramificación o en el caso de una unidad funcional no segmentada se almacenan en una estructura lógica que permite contabilizar ciclos. Entonces, estos identificadores de registro se utilizan en la etapa o ciclo previo a la escritura en el banco de registros para controlar los multiplexores de cortocircuito. Posteriormente, estos mismos identificadores de registro destino se utilizan en la etapa de escritura en el banco de registros para identificar el registro que se actualiza. En la Figura 5.34 se muestra un esquema general de los instantes en que se utiliza el identificador de registro.

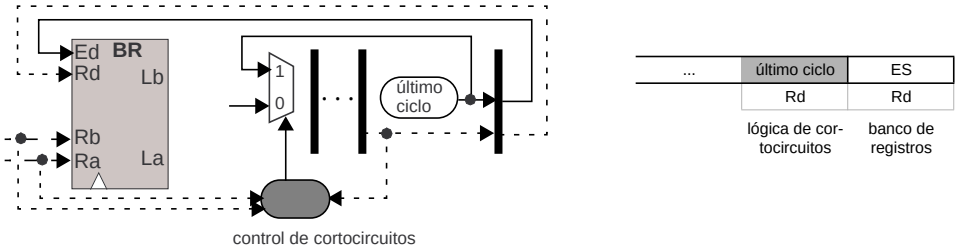


Figura 5.34 *Instantes en que se utiliza un identificador de registro destino. En trazo discontinuo se muestran los buses que transmiten identificadores de registro (Ra, Rb, Rd). En trazo continuo se muestran los buses que transmiten datos (ED).*

En un ciclo puede finalizar la ejecución de más de una instrucción. Sin embargo, por la forma de gestionar los riesgos $R(i) \cap R(i+k) \neq \varnothing$, no puede ocurrir que en el mismo ciclo finalice la ejecución de dos instrucciones que actualizan el mismo registro. En la Figura 5.35 se muestra una situación donde dos instrucciones escriben en el registro R4 y una tercera instrucción, más jóven, utiliza como dato fuente el valor almacenado en R4. La gestión del riesgo $R(i) \cap R(i+k) \neq \varnothing$ determina que la 2ª instrucción esté retenida en la etapa D/L hasta el último ciclo de ejecución de la 1ª instrucción. Por tanto, no es necesario considerar la edad de las instrucciones en la activación de los cortocircuitos. Esto es, cuál de las instrucciones en las etapas fuente de cortocircuito es más jóven.

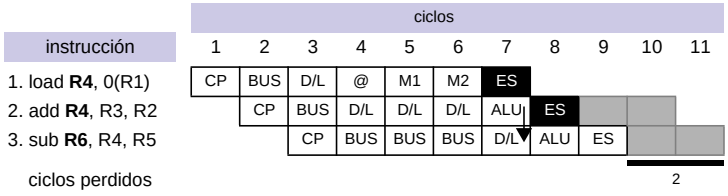


Figura 5.35 *Gestión de un riesgo escritura después de escritura y consumo del dato por una tercera instrucción.*

En la Figura 5.36 se muestra el circuito de control de los multiplexores de cortocircuito. Se comparan los posibles identificadores de registro fuente de la instrucción que ocupa la etapa D/L (campos ra y rb) con los posibles identificadores de registro

destino en la última etapa o último ciclo de cálculo en cada ramificación. Los resultados de los comparadores se validan con señales que indican si la información utilizada en la comparación se refiere a un identificador de registro. Las señales Ra_{DL} y Rb_{DL} se refieren a los campos ra y rb de la instrucción que ocupa la etapa D/L.

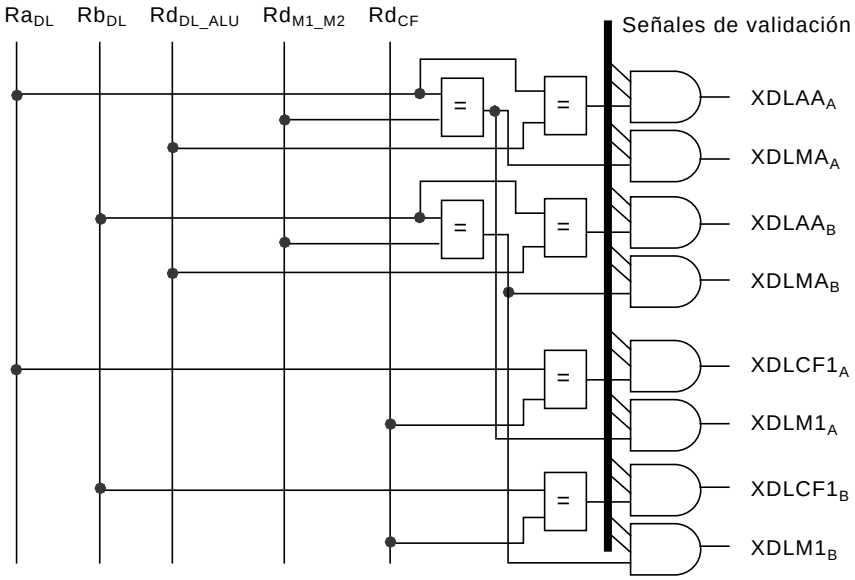


Figura 5.36 *Lógica de control de los cortocircuitos.*

Las señales etiquetadas como Rd_{DL_ALU}, Rd_{M1_M2} y Rd_{CF} se refieren a la información suministrada como identificador de registro destino desde cada ramificación. Notemos que se utiliza una única señal para las instrucciones de cálculo de coma flotante, ya que sólo una ramificación puede utilizar el cortocircuito en un ciclo determinado. Notemos también que la salida de la comparación de los campos ra y rb con la señal Rd_{M1_M2} se utiliza en dos puntos del circuito, ya que la comparación es la misma independientemente de si la instrucción productora (load) es de aritmética entera o coma flotante.

Las señales de salida de los comparadores junto con las señales de validación se utilizan para controlar los multiplexores $XDLMA_A$, $XDLMA_B$, $XDLM1_A$ y $XDLM1_B$.

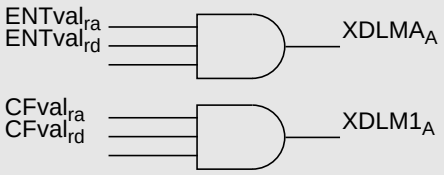
En la Figura 5.36 se identifican de forma agrupada todas las señales de validación. Cada comparación requiere dos señales de validación, una para el identificador de registro fuente y otra para el identificador de registro destino. Las señales de validación de los registros fuente son las mismas que se utilizan en la Figura 5.31.

Ejercicio

Las señales de validación que se utilizan en la lógica de la Figura 5.36 son: $ENTval_{ra}$, $ENTval_{rb}$, $ENTval_{rd}$, $CFval_{ra}$, $CFval_{rb}$ y $CFval_{rd}$. Indique cuáles de estas señales se utilizan para obtener las señales de control de los multiplexores $XDLMA_A$ y $XDLM1_A$.

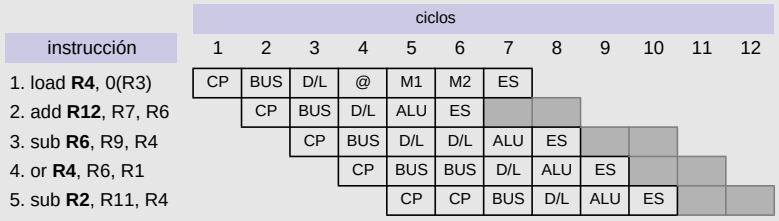
Respuesta

En la siguiente figura se muestran las señales de validación utilizadas.



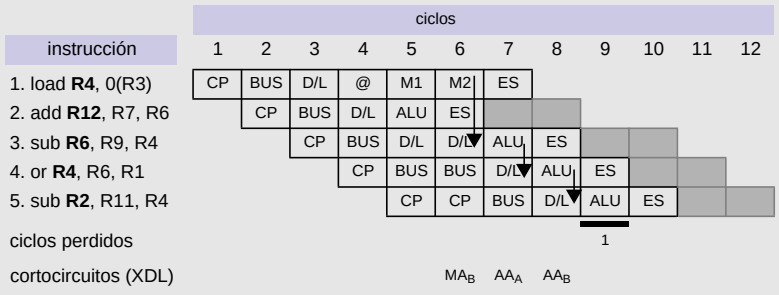
Ejercicio

Dado el siguiente diagrama temporal indique el ciclo en que se utiliza un cortocircuito y el multiplexor de la Figura 5.33 implicado.



Respuesta

En la fila inferior del diagrama temporal se muestran los cortocircuitos utilizados en cada ciclo.



En los ciclos 6, 7 y 8 se utilizan cortocircuitos. En el ciclo 6 el origen del cortocircuito es la ramificación de memoria y en los otros ciclos el origen es la ALU.

Red de cortocircuitos. Es el conjunto de multiplexores que permiten disponer de un dato antes de actualizar el banco de registros.

Bus de resultados. Cada uno de los buses que transportan datos hasta los multiplexores de cortocircuito.

Bus de identificador de registro o de etiqueta. Bus que transporta el identificador de registro destino o etiqueta y que se utiliza para controlar los caminos de cortocircuito.

En la Figura 5.37 se muestra de forma esquemática parte del camino de datos de un procesador utilizando los términos descritos previamente. Los buses de datos se muestran en trazo continuo y los buses de etiquetas en trazo discontinuo. Observemos que no se explícita la segmentación en las ramificaciones. Unicamente se muestran los registros de desacoplo de entrada en cada ramificación.

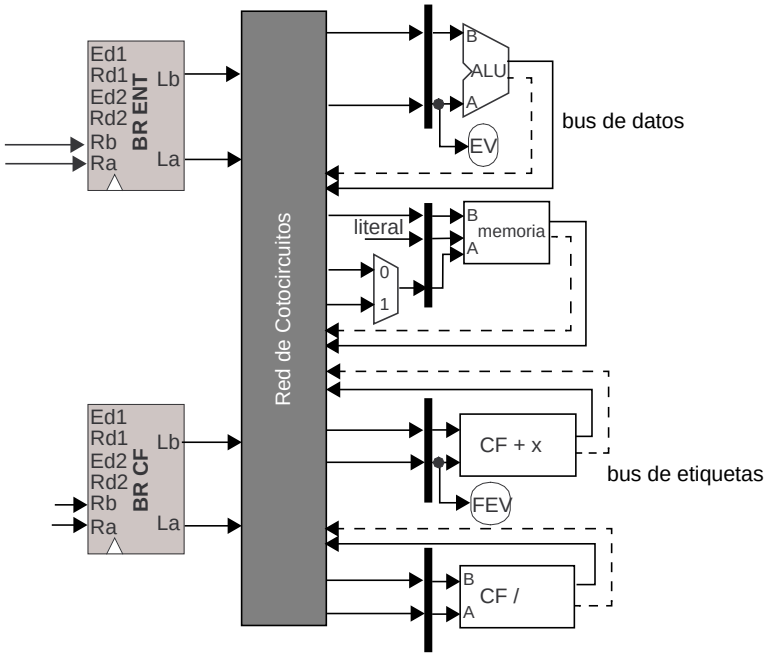


Figura 5.37 *Red de cortocircuitos y buses de datos (trazo continuo) y etiquetas (trazo discontinuo).*

PARALELISMO A NIVEL DE INSTRUCCIÓN

El algoritmo de planificación de instrucciones que se ha presentado en el Capítulo 4 utiliza como punto de partida bloques básicos. Esto es, una secuencia consecutiva de instrucciones que no incluye ninguna instrucción de secuenciamiento y si la incluye, la instrucción de secuenciamiento es la última en la secuencia de instrucciones.

El objetivo del algoritmo de planificación de instrucciones es tolerar u ocultar la latencia de ejecución de las instrucciones ubicando entre una instrucción productora y una instrucción consumidora instrucciones independientes. Esto es, instrucciones que no utilicen como dato el resultado producido por la instrucción productora.

En el procesador lineal con cortocircuitos sólo hay que tolerar la latencia productor-uso de las instrucciones load, ya que todas las otras instrucciones tienen una latencia productor-uso igual a uno y por tanto, en este último caso, una instrucción puede utilizar el dato calculado por la instrucción previa. En resumen, en el procesador lineal con cortocircuitos sólo hay que ubicar una instrucción independiente entre una instrucción load y una instrucción que consuma el dato obtenido de memoria por la instrucción load.

En el procesador multiciclo sólo las instrucciones que utilizan la ALU tienen latencia productor-uso igual a uno. Todas las otras instrucciones que actualizan un banco de registros tienen latencia productor-uso mayor que uno. Entonces, existe mayor demanda de instrucciones independientes o paralelismo a nivel de instrucción para tolerar la latencia de ejecución de las instrucciones.

El tamaño o número de instrucciones de los bloques básicos es usualmente pequeño y las dependencias entre las instrucciones limitan las posibilidades de selección en un paso del algoritmo de planificación de instrucciones. Seguidamente utilizamos un ejemplo para observar esta limitación. En la parte izquierda de la Figura 5.38 se muestra un código en un lenguaje de alto nivel y en la parte derecha una traducción en lenguaje ensamblador. Las operaciones aritméticas son en coma flotante. El registro F2 almacena el contenido de la variable S y el control del bucle se efectúa almacenando en el registro R5 la dirección de la posición de memoria siguiente al último elemento del vector A.

```
do I =1, N
   A(I) = A(I) + S
enddo
```

```
1$:  a. Fload F0, 0(R1)      load A(I)
     b. Fadd F4, F0, F2      tmp = A(I) + S
     c. Fstore F4, 0(R1)     store tmp
     d. add R1, R1, #8       índice del vector A
     e. cmpeq R2, R1, R5     comparar dirección
     f. beq R2, 1$
```

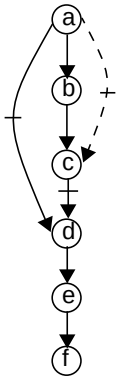


Figura 5.38 *Suma de un escalar a los elementos de un vector. El arco con línea de trazos indica dependencia debida a posiciones de memoria.*

Una interpretación del bucle en el procesador multiciclo descrito en este capítulo tarda 13 ciclos por iteración: se interpretan 6 instrucciones y se pierden 5 ciclos por riesgos de datos y 2 ciclos por riesgos de secuenciamiento. Los riesgos de datos se producen entre la 1ª y la 2ª instrucción, en el que se pierden 2 ciclos, y entre la 2ª y la 3ª instrucción, en el que se pierden 3 ciclos. Como se efectúa una suma de coma flotante, se tardan 13 ciclos por suma de coma flotante.

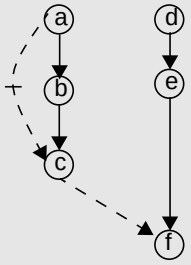
El algoritmo de planificación de instrucciones descrito en el Capítulo 4 utiliza como entrada el cuerpo del bucle. A la vista del grafo de dependencias, mostrado en la parte derecha de la Figura 5.38, no existe paralelismo a nivel de instrucción y por tanto, no existen grados de libertad para planificar las instrucciones con el objetivo de tolerar la latencia de ejecución de las instrucciones.

Ejercicio

En el grafo de dependencias de la Figura 5.38 el orden que determinan las antidependencias debidas al registro R1 puede eliminarse gracias a que el modo de direccionamiento en las instrucciones de acceso a memoria es sumar el contenido de un registro con un literal especificado en la instrucción y a que el valor con que se incrementa el registro R1 es conocido y constante.

Por ejemplo, la instrucción add puede ubicarse entre la instrucción Fload y la instrucción Fstore. Entonces, el literal de la instrucción Fload es cero y el literal de la instrucción Fstore es -8. Teniendo en cuenta esta posibilidad, en la parte derecha de la siguiente figura se muestra un grafo de dependencias en el que no se especifican las antidependencias debidas al registro R1.

1$: a. Fload **F0**, 0(R1)
b. Fadd **F4**, F0, F2
c. Fstore F4, 0(R1)
d. add **R1**, R1, #8
e. cmpeq **R2**, R1, R5
f. beq R2, 1$



El algoritmo de planificación de instrucciones utiliza como entrada el grafo de dependencias de la parte derecha de la figura anterior. Después de ordenar las instrucciones se determina el valor de los literales de las instrucciones FLoad y Fstore. Si no se ha modificado el orden relativo original no se modifica el literal. En caso contrario, se utiliza como literal el valor opuesto que se suma al contenido del registro R1.

Notemos que al cambiar la posición relativa de la instrucción add respecto de otras instrucciones, una antidependencia se convierte en una dependencia verdadera y viceversa. Por ejemplo, en el orden de programa original hay una antidependencia entre la instrucción c y la instrucción d. Si la instrucción d se ubica antes de la instrucción c, cambiando el valor de los literales, habrá una dependencia verdadera entre la instrucción d y la instrucción c.

Proponga tres secuencias de instrucciones reordenadas que tarden 11 ciclos por iteración.

Respuesta

En la siguiente figura se muestran 3 secuencia de instrucciones reordenadas.

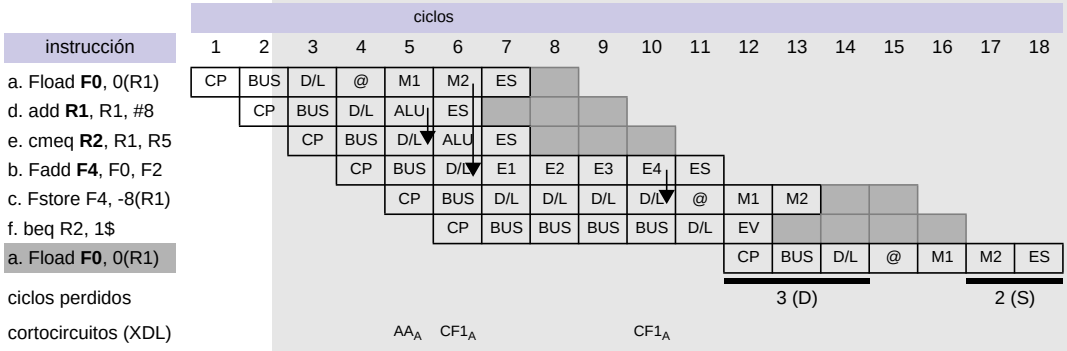
1$: a. Fload **F0**, 0(R1)
d. add **R1**, R1, #8
e. cmpeq **R2**, R1, R5
b. Fadd **F4**, F0, F2
c. Fstore F4, -8(R1)
f. beq R2, 1$

1$: a. Fload **F0**, 0(R1)
d. add **R1**, R1, #8
b. Fadd **F4**, F0, F2
e. cmpeq **R2**, R1, R5
c. Fstore F4, -8(R1)
f. beq R2, 1$

1$: a. Fload **F0**, 0(R1)
b. Fadd **F4**, F0, F2
d. add **R1**, R1, #8
e. cmpeq **R2**, R1, R5
c. Fstore F4, -8(R1)
f. beq R2, 1$

La interpretación de la primera secuencia de instrucciones en el procesador multiciclo se muestra en el siguiente diagrama temporal. El número de ciclos perdidos es 3 por riesgos de datos y 2 por riesgos de secuenciamiento. Entonces, el número de ciclos por iteración es 11.

Las otras dos ordenaciones de la secuencia de instrucciones tardan el mismo número de ciclos por iteración.



Un riesgo que no se detectaba en el procesador lineal y que se detecta en el procesador multiciclo es el debido a las dependencias de salida. Ello es debido a que, en general, las instrucciones tienen latencias de ejecución distintas. La eliminación de este riesgo también requiere paralelismo a nivel de instrucción.

Las antidependencias y dependencias de salida restringen el paralelismo a nivel de instrucción. Sin embargo, estas dependencias no son inherentes al cálculo que se quiere efectuar y son debidas a que se escribe más de una vez en una posición de almacenamiento. Por ejemplo, en la secuencia de instrucciones que se muestra en la parte izquierda de la Figura 5.39 el algoritmo de planificación no puede reordenar las instrucciones debido a las antidependencias y dependencias de salida determinadas por el registro R2. En la parte derecha de la Figura 5.39 se muestra una secuencia de instrucciones que efectúa los mismos cálculos, pero no existen antidependencias y dependencias de salida. Ello es debido a que el registro R2 no se utiliza como registro destino en la 3ª instrucción. En esta secuencia de instrucciones existe paralelismo a nivel de instrucción y el algoritmo de planificación de instrucciones puede tomar decisiones sobre el orden de las instrucciones.

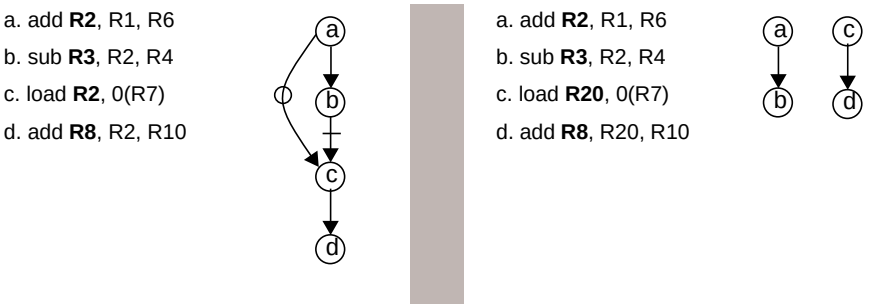


Figura 5.39 *Las antidependencias y dependencias de salida limitan el paralelismo a nivel de instrucción.*

En los siguientes apartados se muestra como no restringir o como incrementar el paralelismo a nivel de instrucción. En primer lugar se describen las dependencias de nombre, las cuales reducen las oportunidades de reordenación de instrucciones. Seguidamente se muestra la transformación desenrollar, la cual permite incrementar el número de instrucciones que analiza el algoritmo de planificación de instrucciones, con el objetivo de incrementar las oportunidades de reordenación de instrucciones.

Dependencias de nombre

Las antidependencias y las dependencias de salida se denominan dependencias de nombre o falsas dependencias, ya que entre la instrucción origen de la dependencia y la instrucción destino no existe flujo de información y son debidas a la actualización repetida de una posición de almacenamiento.

Si en una posición de almacenamiento sólo se escribe una vez no se producen dependencias de nombre. Un lenguaje donde sólo se puede escribir una vez en una posición de almacenamiento se denomina lenguaje de asignación única.

En este apartado nos centraremos en las dependencias de nombre debidas a registros.

Dos de las últimas fases típicas de un compilador son la generación de código y la asignación de registros. Cuando el código se interpreta en un procesador segmentado, a estas fases hay que añadir la fase de planificación de instrucciones, que tiene como objetivo reducir los bloqueos debidos a riesgos de datos.

En la fase de generación de código se crean las secuencias de instrucciones de lenguaje máquina que permiten efectuar el cálculo especificado en un lenguaje de alto nivel. Estas instrucciones especifican registros fuente y destino. El número de registros que se utiliza es ilimitado y se denominan registros virtuales. Un registro se escribe una sola vez y el código generado es de asignación única.

En la fase de asignación de registros se establece una correspondencia entre los registros virtuales y los registros lógicos o de la arquitectura. Como la Arquitectura del Conjunto de Instrucciones (ACI) define un número limitado de registros la asignación de registros lógicos da lugar a dependencias de nombre.

En la fase de asignación de registros se intenta acceder el menor número de veces a memoria y cuando un registro almacena un valor que no se volverá a utilizar, se le asigna otro valor. La acción de escribir en un registro se denomina definición de un registro y una acción de lectura se denomina uso del registro.

La duración entre la definición de un registro y el último uso de un registro se denomina tiempo de vida del valor (o del registro). El objetivo de la fase de asignación de registros es intentar asignar un mismo registro lógico a valores cuyo tiempo de vida no se solapa. Si ello no es posible se utilizan posiciones de memoria para almacenar temporalmente (store) los valores calculados y posteriormente, se recuperan (load) cuando se necesitan (spill code).

En un procesador segmentado se utiliza la planificación de instrucciones para reducir los ciclos perdidos por riesgos de datos. Entonces, es importante que la asignación de registros lógicos no limite la capacidad del algoritmo de planificación de instrucciones.

Las fases de asignación de registros y planificación de instrucciones pueden efectuarse de forma conjunta o separada. Nosotros, por simplicidad, supondremos que se efectuan de forma separada y que la planificación de instrucciones se efectúa antes de la asignación de registros.

Cuando se solicite eliminar las dependencias de nombre debidas a registros supondremos un número ilimitado de registros, aunque una ACI concreto tiene un número limitado de registros. Notemos que cuando se utiliza una ACI con un pequeño número de registros los códigos tendrán muchas dependencias de nombre.

En la parte izquierda de la Figura 5.40 se muestra un ejemplo donde hay dependencias de nombre debidas al registro R1. Para eliminar estas dependencias se sustituye una definición del registro y todos sus usos por otro registro no utilizado en la secuencia de instrucciones (parte derecha de la figura). En el ejemplo se ha sustituido la segunda definición y uso del registro R1 por el registro R10.

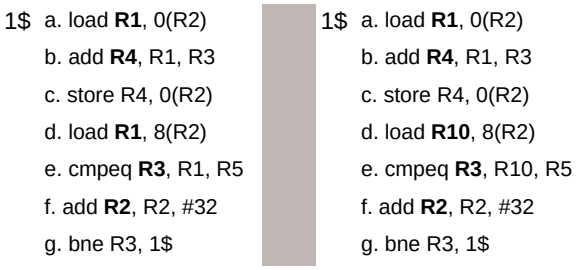


Figura 5.40 *Eliminación de dependencias de nombre debidas a registros.*

Transformación desenrollar

El número de instrucciones de un bloque básico y las dependencias entre las instrucciones limitan las posibilidades de selección en un paso del algoritmo de planificación de instrucciones.

Una construcción utilizada típicamente en los programas son las estructuras iterativas. Nosotros nos centraremos en las construcciones que iteran un número conocido de veces antes de empezar a iterar. A estas construcciones las denominaremos bucles. y al conjunto de sentencias que abraza un bucle lo denominaremos cuerpo del bucle. Entre los bucles nos restringiremos a los casos en que el cuerpo del bucle no contenga construcciones condicionales.

La transformación desenrollar es un método sencillo de transformación de código pero útil para incrementar el tamaño de los bloques básicos, lo cual puede incrementar los grados de libertad del algoritmo de planificación para ocultar (soportar) la latencia productor-uso de las instrucciones.

La transformación desenrollar es simplemente replicar varias veces el cuerpo de un bucle y eliminar las dependencias de nombre debidas a registros. Posteriormente se planifica el nuevo

cuerpo del bucle. El potencial de la transformación se explota cuando las instrucciones load y store acceden a posiciones de memoria distintas y por tanto, el algoritmo de planificación puede modificar el orden de programa.

En la parte izquierda de la Figura 5.41 se muestra, en un lenguaje de alto nivel, la transformación desenrollar aplicada al código de la Figura 5.38. Notemos que se distinguen dos bucles. En el segundo de ellos se ejecuta un número de iteraciones múltiplo de 4 y en el primer bucle se ejecuta el resto de las iteraciones, el cual no es múltiplo de cuatro.

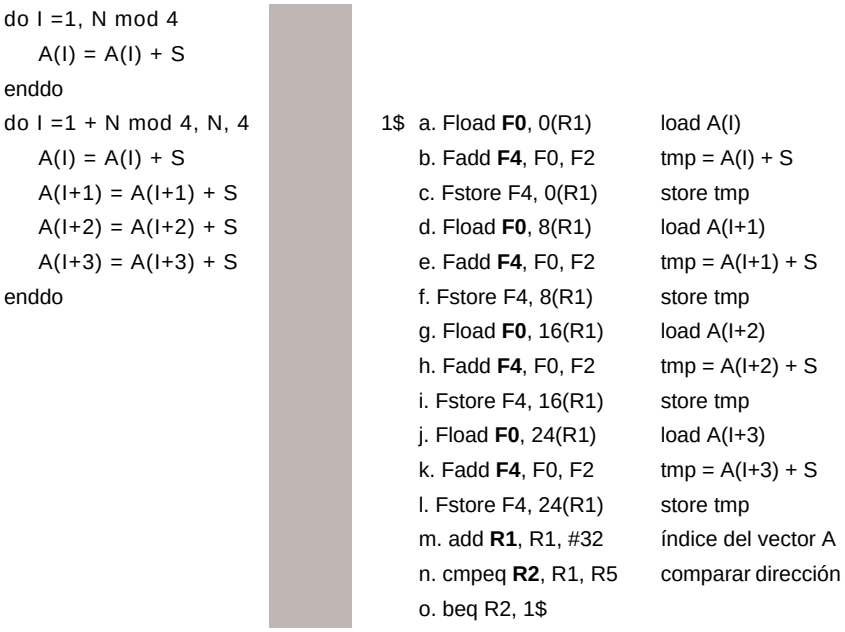


Figura 5.41 *Transformación de código desenrollar.*

En la parte derecha de la Figura 5.41 se muestra una traducción en lenguaje ensamblador del segundo bucle. En primer lugar notemos un efecto beneficioso al desenrollar. El número de instrucciones que se interpretan por cada 4 sumas en coma flotante (Fadd) es menor que en el bucle original. Para ello se ha utilizado el modo de direccionamiento de las instrucciones de acceso a memoria con el objetivo de efectuar sólo una operación add R1, R1, # cada 4 sumas en coma flotante. También se dejan de interpretar 3 instrucciones de comparación y 3 instrucciones de secuenciamiento.

En resumen en el bucle de la derecha de la Figura 5.41, cada 4 sumas en coma flotante, se interpretan menos instrucciones que en el bucle original. Por tanto, se espera reducir el tiempo de ejecución, ya que el número de instrucciones es uno de los factores que se utiliza para evaluar el tiempo de ejecución.

En el ejemplo de la Figura 5.41 el cuerpo del bucle original está replicado 4 veces. Ahora bien, el número de veces que se desenrolla un bucle es dependiente, entre otros, de factores tales como el número de instrucciones del cuerpo del bucle y el número de registros lógicos.

Un efecto de la transformación desenrollar, que puede ser perjudicial, es el incremento del tamaño que ocupa el código o número de instrucciones. Ello puede incrementar el número de fallos de capacidad en la cache de instrucciones.

Ejercicio

Construya el grafo de dependencias de la secuencia de instrucciones de la Figura 5.41. Tenga en cuenta que se puede modificar el valor de los literales en las instrucciones de acceso a memoria. Proponga una planificación de las instrucciones y determine el número de ciclos por suma de coma flotante. Note que entre las instrucciones de acceso a memoria no hay dependencias, ya que entre una instrucción Fstore y una instrucción Fload no se modifica el contenido del registro R1 y el valor del literal en la instrucción Fstore y la instrucción Fload es distinto.

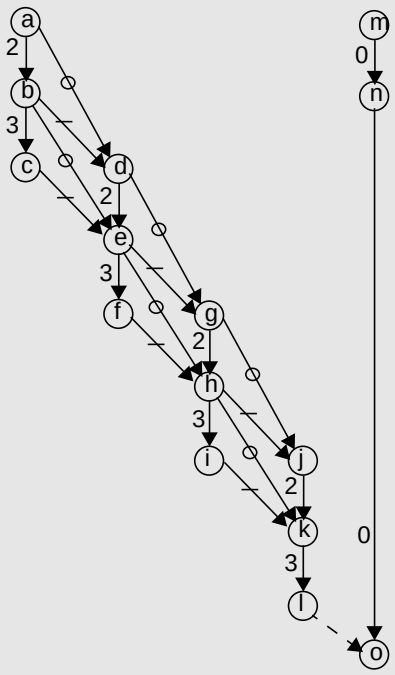
Calcule la ganancia que se obtiene respecto del código original.

Respuesta

En el centro de la siguiente figura se muestra el grafo de dependencias y en la parte derecha de la figura se muestra la secuencia de instrucciones reordenada. Los arcos del grafo de dependencias, en el caso de las dependencias lectura después de escritura, se han etiquetado con la latencia productor uso menos uno. Notemos que en el grafo de dependencias no tenemos en cuenta las dependencias debidas al registro R1 ya que se pueden modificar los literales en las instrucciones de acceso a memoria.

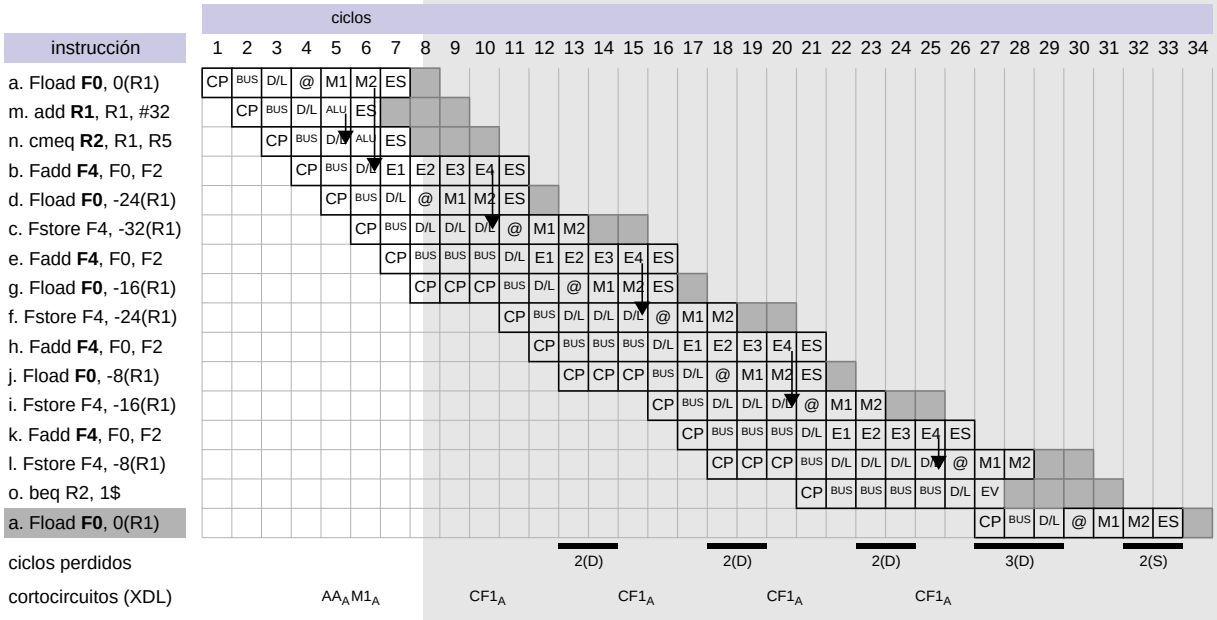
Por claridad en el grafo de dependencias no se muestran las antidependencias debidas a memorias entre los pares de instrucciones (a, c), (d, f), (g, i) y (j, l).

1$ a. Fload **F0**, 0(R1)
b. Fadd **F4**, F0, F2
c. Fstore F4, 0(R1)
d. Fload **F0**, 8(R1)
e. Fadd **F4**, F0, F2
f. Fstore F4, 8(R1)
g. Fload **F0**, 16(R1)
h. Fadd **F4**, F0, F2
i. Fstore F4, 16(R1)
j. Fload **F0**, 24(R1)
k. Fadd **F4**, F0, F2
l. Fstore F4, 24(R1)
m. add **R1**, R1, #32
n. cmpeq **R2**, R1, R5
o. beq R2, 1$



1$ a. Fload **F0**, 0(R1)
m. add **R1**, R1, #32
n. cmpeq **R2**, R1, R5
b. Fadd **F4**, F0, F2
d. Fload **F0**, -24(R1)
c. Fstore F4, -32(R1)
e. Fadd **F4**, F0, F2
g. Fload **F0**, -16(R1)
f. Fstore F4, -24(R1)
h. Fadd **F4**, F0, F2
j. Fload **F0**, -8(R1)
i. Fstore F4, -16(R1)
k. Fadd **F4**, F0, F2
l. Fstore F4, -8(R1)
o. beq R2, 1$

En el siguiente diagrama temporal se observan los ciclos perdidos por riesgos de datos y de secuenciamiento.



El número de ciclos es: 15 + 9 + 2= 26.

Se efectuan 4 sumas de coma flotante. Entonces, el número de ciclos por suma de coma flotante es: 26/4 = 6.5

Cuando se interpreta el bucle original se tardan 13 ciclos por suma de coma flotante. Entonces, la ganancia es: 13 / 6.5 = 2.

El siguiente paso después de desenrollar es eliminar las dependencias de nombre. Nosotros supondremos un número ilimitado de registros lógicos. En la parte izquierda de la Figura 5.42 se muestra el código después de eliminar las dependencias de nombre. En el centro de la figura se muestra el grafo de dependencias, en el cual no se tienen en cuenta las dependencias que determina el registro R1, ya que se pueden modificar los literales en las instrucciones de acceso a memoria. Los arcos del grafo de dependencias, en el caso de las dependencias lectura después de escritura, se han etiquetado con la latencia productor uso menos uno. Por claridad en el grafo de dependencias no se muestran las antidependencias debidas a memorias entre los pares de instrucciones (a, c), (d, f), (g, i) y (j, l).

a. Fload **F0**, 0(R1)
b. Fadd **F4**, F0, F2
c. Fstore F4, 0(R1)
d. Fload **F10**, 8(R1)
e. Fadd **F40**, F10, F2
f. Fstore F40, 8(R1)
g. Fload **F100**, 16(R1)
h. Fadd **F400**, F100, F2
i. Fstore F400, 16(R1)
j. Fload **F1000**, 24(R1)
k. Fadd **F4000**, F1000, F2
l. Fstore F4000, 24(R1)
m. add **R1**, R1, #32
n. cmpeq **R2**, R1, R5
o. beq R2, 1$

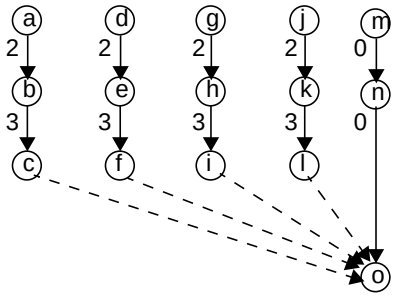


Figura 5.42 *Código ensamblador de un bucle desenrrollado después de eliminar las dependencias de nombre.*

Ejercicio

Planifique las instrucciones del código de la Figura 5.42 y determine los ciclos por suma de coma flotante. En la planificación de instrucciones seleccione siempre cualquier otra instrucción, excepto la instrucción o, antes que la instrucción m o n. Calcule la ganancia que aporta la transformación desenrollar.

Respuesta

Después de planificar las instrucciones se obtiene la secuencia de instrucciones que se muestra seguidamente.

Notemos que en el código planificado se entrelaza la interpretación de instrucciones de las 4 iteraciones del bucle original. Cuando entre las iteraciones del bucle original no hay dependencias (observemos el grafo de dependencias de la Figura 5.42) el código característico, después de planificar las instrucciones, es una secuencia de instrucciones que traen datos de memoria (load), una secuencia de instrucciones que efectúa cálculos (Fadd) y finalmente una secuencia de instrucciones que almacena resultados (store).

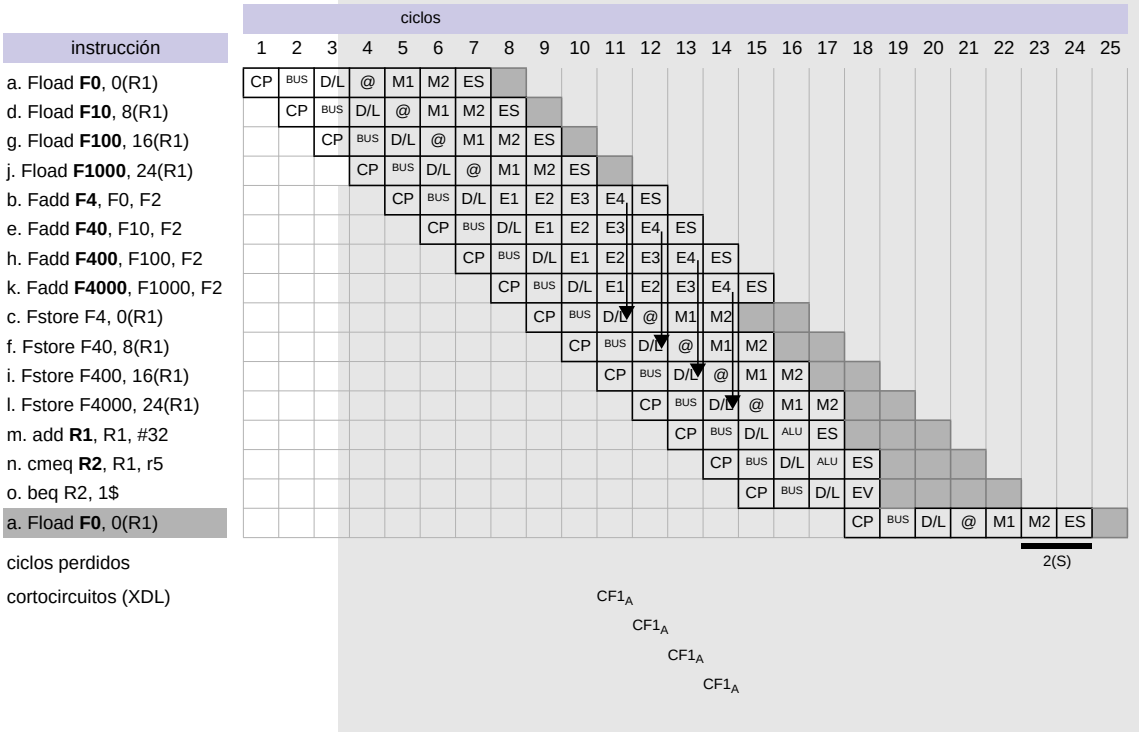
```
1$  a. Fload F0, 0(R1)
    d. Fload F10, 8(R1)
    g. Fload F100, 16(R1)
    j. Fload F1000, 24(R1)
    b. Fadd F4, F0, F2
    e. Fadd F40, F10, F2
    h. Fadd F400, F100, F2
    k. Fadd F4000, F1000, F2
    c. Fstore F4, 0(R1)
    f. Fstore F40, 8(R1)
    i. Fstore F400, 16(R1)
    l. Fstore F4000, 24(R1)
    m. add R1, R1, #32
    n. cmpeq R2, R1, R5
    o. beq R2, 1$
```

En el siguiente diagrama temporal se observan los ciclos perdidos al interpretar la secuencia de instrucciones planificada.

El número de ciclos es: 15 + 2 = 17.

Se efectuan 4 sumas de coma flotante. Entonces, el número de ciclos por suma de coma flotante es: 17/4 = 4.25

Cuando se interpreta el bucle original se tardan 13 ciclos por suma de coma flotante. Entonces, la ganancia es: 13 / 4.25 = 3.06.


ciclos
instrucción
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
a. Fload F0, 0(R1)
d. Fload F10, 8(R1)
g. Fload F100, 16(R1)
j. Fload F1000, 24(R1)
b. Fadd F4, F0, F2
e. Fadd F40, F10, F2
h. Fadd F400, F100, F2
k. Fadd F4000, F1000, F2
c. Fstore F4, 0(R1)
f. Fstore F40, 8(R1)
i. Fstore F400, 16(R1)
l. Fstore F4000, 24(R1)
m. add R1, R1, #32
n. cmeq R2, R1, r5
o. beq R2, 1$
a. Fload F0, 0(R1)
CP BUS D/L @ M1 M2 ES E1 E2 E3 E4 ALU EV
2(S)
ciclos perdidos
cortocircuitos (XDL)
CF1A
CF1A
CF1A
CF1A

EJEMPLOS

En esta sección se utilizan varios ejemplos de código para analizar el rendimiento de un procesador multiciclo. El primero de ellos es la operación SAXPY. Esta operación es el núcleo en la colección de procedimientos de álgebra lineal denominada Linpack.

Otro de los trozos de código es la ordenación lexicográfica de los elementos almacenados en un vector. Por último se analiza un trozo de código que inserta un elemento en una lista ordenada. En cuanto a transformaciones de código se calculan las prestaciones que aporta la transformación desenrollar y la posterior planificación de las instrucciones en el nuevo cuerpo del bucle.

En los ejemplos, cuando se diga procesador descrito en este capítulo entenderemos: procesador multiciclo con cortocircuitos, reducción de latencia efectiva en instrucciones de secuenciamiento incondicional y el control de riesgos descrito en este capítulo. Adicionalmente supondremos predicción fija de sentido, utilizando el signo del literal, en instrucciones de secuenciamiento condicional. El funcionamiento al efectuar predicción de sentido y la recuperación en caso de error de predicción es idéntico al descrito en el Capítulo 4 para el procesador lineal.

Cuando se soliciten ciclos perdidos por riesgos de secuenciamiento se computarán los ciclos perdidos cuando la predicción de sentido es errónea y los ciclos debidos al riesgo de secuenciamiento cuando se predice modificar el secuenciamiento en instrucciones de secuenciamiento condicional o cuando se interpreta una instrucción de secuenciamiento incondicional. A estos últimos ciclos se les denomina ciclos de retardo en la búsqueda de instrucciones.

Operación SAXPY

La operación SAXPY es el producto de un escalar por cada uno de los elementos de un vector para actualizar otro vector y es típica en programas de cálculo numérico. Su expresión en un lenguaje de alto nivel es

```
do I =1, N
    Y(I) = A x X(I) + Y(I)
enddo
```

Una traducción a lenguaje máquina es la siguiente. Los registros r1 y r2 se han inicializado con la dirección base de los vectores X e Y. El valor de la variable A está almacenado en el registro F1 y el registro r3 se ha inicializado con el número de iteraciones.

```
1$:  Fload F2, 0 (r1)      load X(I)
     Fmul F4, F2, F1       multiplicar A por X(I)
     Fload F6, 0 (r2)      load Y(I)
     Fadd F8, F4, F6       sumar A x X(I) a Y (I)
     Fstore F8, 0 (r2)     store Y(I)
     add r1, r1, #8        incrementar índice de X
     add r2, r2, #8        incrementar índice de Y
     sub r3, r3, #1        contador de iteraciones
     bne r3, 1$            iterar si aún no ha finalizado
```

Pregunta 1: *En el procesador segmentado multiciclo descrito en este capítulo, calcule los ciclos de ejecución por iteración. Así mismo, indique los ciclos perdidos por tipo de riesgo y calcule el CPI por iteración.*

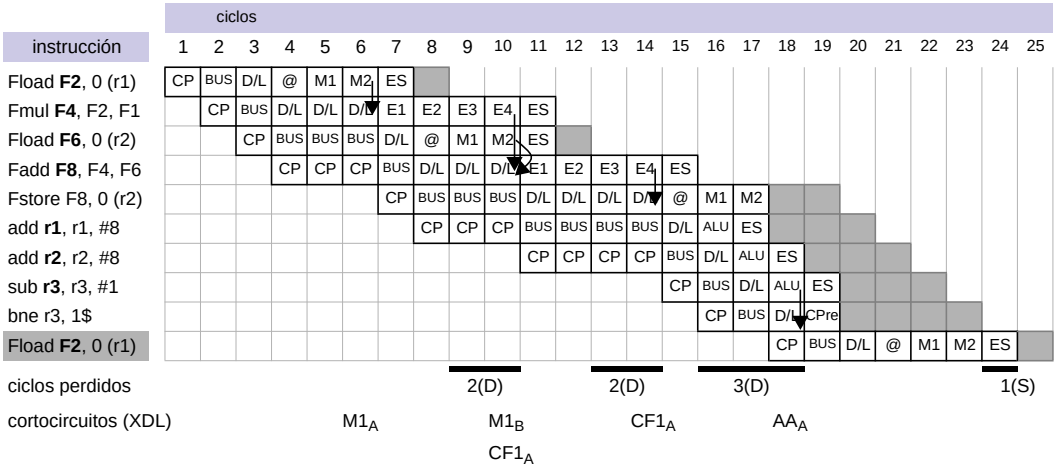
Respuesta: En la siguiente figura se muestra mediante un diagrama temporal la interpretación segmentada de una iteración del código descrito y la 1ª instrucción de la siguiente iteración.

En los ciclos 4 y 5 se detecta un riesgo de datos debido al registro F2. El valor que se almacenará en el registro F2 se obtiene mediante un cortocircuito de memoria a la etapa D/L.

En el ciclo 10, la instrucción Fadd **F8**, F4, F6 obtiene el dato fuente F6 mediante un cortocircuito desde la ramificación de coma flotante que efectúa multiplicaciones a la etapa D/L. El otro dato fuente los obtiene mediante un cortocircuito desde la ramificación de memoria a la etapa D/L. Previamente la instrucción ha estado retenida 2 ciclos en la etapa D/L por riesgos de datos.

La instrucción Fstore permanece 4 ciclos en la etapa D/L debido a riesgos de datos (registro F8). En el ciclo 14 la instrucción Fstore obtiene el dato fuente que está esperando mediante un cortocircuito desde la ramificación de coma flotante que efectúa sumas a la etapa D/L.

En el ciclo 18 la instrucción bne está en la etapa D/L. Obtiene el dato fuente mediante un cortocircuito desde la etapa ALU a la etapa D/L y se predice modificar el secuenciamiento, ya que el signo del literal es negativo. Como se predice que se toma el salto existe un retardo de búsqueda de 1 ciclo (ciclo 18), el cual se observa en el ciclo 24 como ciclo perdido.



Los ciclos perdidos son

	Riesgos	
	datos	secuenciamiento
ciclos perdidos	7	1

Los ciclos de ejecución y el CPI de una iteración son:

En una iteración	
T = número de instrucciones + ciclos perdidos	9 + 8 = 17 ciclos
CPI = T / (número de instrucciones)	17 / 9 = 1.89

Pregunta 2: *Desenrolle una vez el bucle y planifique las instrucciones con el objetivo de soportar la latencia de las instrucciones y reducir el número de ciclos perdidos.*

Respuesta: En la parte derecha de la siguiente figura se muestra el código después de aplicar la transformación de desenrollar, expresado en un lenguaje de alto nivel.

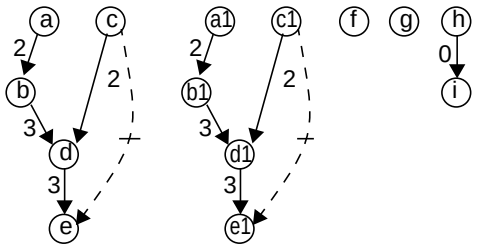
El primer bucle del código transformado itera una vez cuando N no es múltiplo de 2.

original	desenrollado
do I =1, N Y(I) = a x X(I) + Y(I) enddo	do I =1, N mod 2 Y(I) = a x X(I) + Y(I) enddo do I = 1 + N mod 2, N, 2 Y(I) = a x X(I) + Y(I) Y(I+1) = a x X(I+1) + Y(I+1) enddo

En la parte izquierda de la siguiente figura se muestra la secuencia de instrucciones en lenguaje máquina una vez desenrollado el bucle y eliminadas las dependencias de nombre. En la parte derecha se muestra la misma secuencia de instrucciones planificada.

En el centro de la figura se ha representado el grafo de dependencias teniendo en cuenta que se puede modificar el valor de los literales en las instrucciones de acceso a memoria y que se conoce el valor con que se actualiza el registro r1. Los arcos del grafo de dependencias se han etiquetado con la latencia productor-uso menos uno.

1$: a. Fload **F2**, 0 (r1)
b. Fmul **F4**, F2, F1
c. Fload **F6**, 0 (r2)
d. Fadd **F8**, F4, F6
e. Fstore F8, 0 (r2)
a1. Fload **F20**, 8 (r1)
b1. Fmul **F40**, F20, F1
c1. Fload **F60**, 8 (r2)
d1. Fadd **F80**, F40, F60
e1. Fstore F80, 8 (r2)
f. add **r1**, r1, #16
g. add **r2**, r2, #16
h. sub **r3**, r3, #2
i. bne r3, 1$



1$: a. Fload **F2**, 0 (r1)
a1. Fload **F20**, 8 (r1)
c. Fload **F6**, 0 (r2)
b. Fmul **F4**, F2, F1
b1. Fmul **F40**, F20, F1
c1. Fload **F60**, 8 (r2)
f. add **r1**, r1, #16
d. Fadd **F8**, F4, F6
d1. Fadd **F80**, F40, F60
g. add **r2**, r2, #16
h. sub **r3**, r3, #2
e. Fstore F8, -16(r2)
e1. Fstore F80, -8 (r2)
f. bne r3, 1$

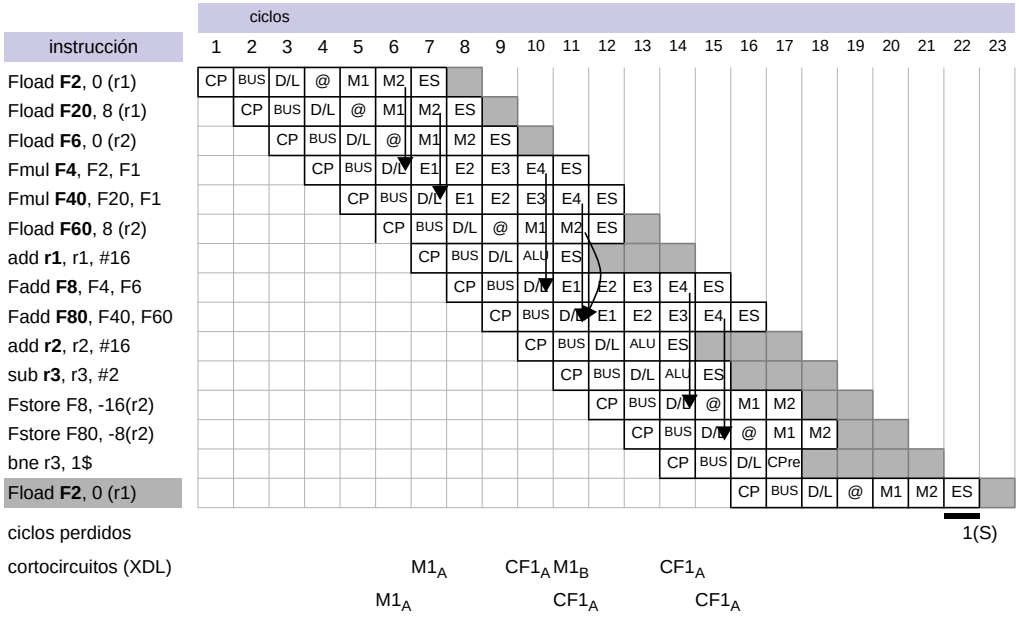
Pregunta 3: *En el procesador segmentado multiciclo descrito en este capítulo, calcule los ciclos de ejecución por iteración del cuerpo del bucle planificado en la pregunta anterior. Así mismo, indique los ciclos perdidos por tipo de riesgo y calcule el CPI por iteración.*

Respuesta: En la siguiente figura se muestra mediante un diagrama temporal la interpretación segmentada de una iteración del código descrito y la 1ª instrucción de la siguiente iteración.

En los ciclos 6 y 7 respectivamente las instrucciones Fmul F4, F2, F1 y Fmul F40, F20, F1 obtienen los datos fuente F2 y F20 mediante el mismo cortocircuito desde la ramificación de memoria a la etapa D/L.

En los ciclos 10 y 11 las instrucciones Fadd obtienen, respectivamente los datos fuente F4 y F40 mediante el mismo cortocircuito desde la ramificación de coma flotante que efectúa multiplicaciones a la etapa D/L. Adicionalmente, en el ciclo 11, la instrucción Fadd F80, F40, F60 obtiene el dato fuente F60 mediante un cortocircuito desde la ramificación de memoria a la etapa D/L.

Las instrucciones Fstore obtienen, en los ciclos 14 y 15 respectivamente, el dato que deben almacenar en memoria mediante el mismo cortocircuito desde la ramificación de coma flotante que efectúa sumas a la etapa D/L.



En el ciclo 16 la instrucción bne está en la etapa D/L y se predice modificar el secuenciamiento, ya que el signo del literal es negativo. Como se predice que se toma el salto existe un retardo de búsqueda de 1 ciclo, el cual se observa en el ciclo 22.

Los ciclos perdidos son

	Riesgos	
	datos	secuenciamiento
ciclos perdidos	0	1

Los ciclos de ejecución y el CPI de una iteración del cuerpo del bucle desenrollado son:

En una iteración	
T = número de instrucciones + ciclos perdidos	14 + 1 = 15 ciclos
CPI = T / (número de instrucciones)	15 / 14 = 1.07

Pregunta 4: *Determine la ganancia que aporta desenrollar el bucle.*

Respuesta: La ganancia se calcula como el cociente de tiempos de ejecución del mismo número de iteraciones del bucle original.

$$Ganancia = \frac{T \times 2}{T_{desenrollar}} = \frac{17 \times 2}{15} = 2.27$$

El hecho de desenrollar aporta una ganancia de un 127%. Parte de la ganancia es debida a que se interpretan 4 instrucciones menos cada dos iteraciones y una de las instrucciones es de secuenciamiento.

Ordenación lexicográfica de los elementos de un vector

Como algoritmo de ordenación utilizaremos el denominado burbuja y nos centraremos en el bucle interno. Este bucle recorre los elementos de un vector y los compara dos a dos. En el caso de estar desordenados efectúa un intercambio del contenido de las posiciones que ha comparado.

En la siguiente figura se muestra el código en alto nivel y la traducción a lenguaje máquina del bucle interno del algoritmo de la burbuja.

```
do I =1, N
    if (A(I) > A(I+1)) then
        temp = A(I)
        A(I) = A(I+1)
        A(I+1) = temp
        cambios = cambios +1
    endif
enddo
```

```
1$:  load r8, (r7)          load A(I)
     load r9, 4 (r7)        load A(I+1)
     cmple r10, r8, r9      A(I) ≤ A(I+1)
     bne r10, 2$            se intercambia si A(I) > A(I+1)
     store r9, (r7)         store A(I)
     store r8, 4 (r7)       store A(I+1)
     add r5, r5, #1         contador de cambios
2$:  add r6, r6, #1         contador de iteraciones
     add r7, r7, #4         índice del vector A
     cmple r11, r6, r4      r6 ≤ r4
     bne r11, 1$            iterar si aún no ha finalizado
```

El bucle externo, el cual no se muestra, utiliza la variable cambios para determinar si ha finalizado la ordenación. Para ello, esta variable se inicializa con el valor cero, antes de iniciar cada ejecución del bucle interno.

El registro r4 se inicializa con el número de iteraciones y el registro r6 con el valor uno. El registro r7 se inicializa con la dirección base del vector A y el registro r5 se inicializa con el valor cero.

En todas la preguntas supondremos que al interpretar la instrucción bne r11,1$ se cumple la condición.

Pregunta 1: *Calcule los ciclos de ejecución por iteración, en los dos posibles supuestos al interpretar la instrucción bne r10,2$. Así mismo, indique los ciclos perdidos por tipo de riesgo y calcule el CPI por iteración, en cada uno de los dos casos.*

Respuesta: Distinguimos las dos posibles situaciones al interpretar la instrucción bne r10,2$.

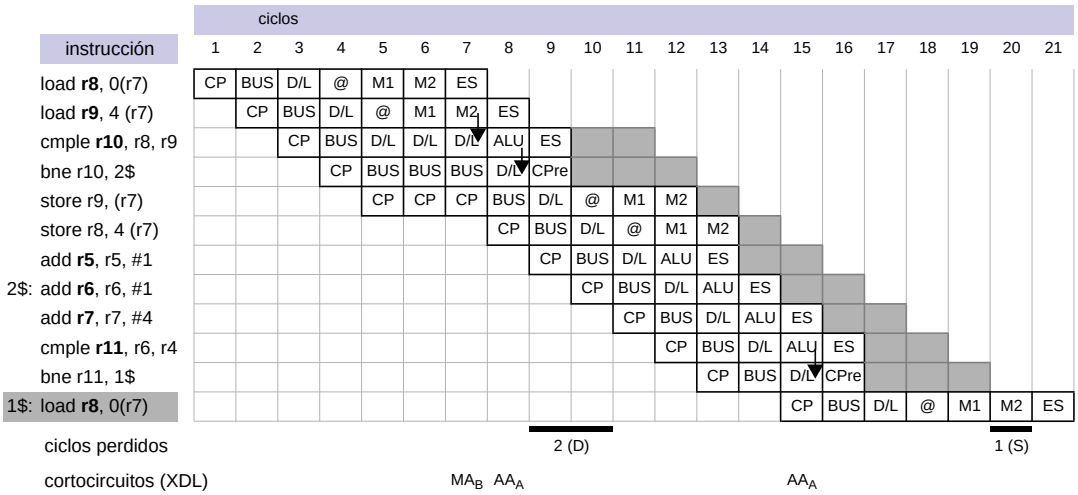
A) Salto no tomado (NT): No se cumple la condición de salto en la instrucción bne r10, 2$.

En la siguiente figura se muestra la interpretación de una iteración del código. Se predice seguir en secuencia, ya que el literal es positivo. Por tanto, se acierta en la predicción de sentido, ya que no se cumple la condición evaluada.

En el ciclo 5 se detectan riesgos de datos debidos a los registros r8 y r9. Un riesgo desaparece en el ciclo 6 y el otro riesgo en el ciclo 7. En el ciclo 7 se lee el registro r8 del banco de registros y se utiliza el cortocircuito de la ramificación de memoria a la etapa D/L para obtener el valor que se almacenará en el registro r9.

En el ciclo 8 se utiliza el cortocircuito desde la etapa ALU a la etapa D/L y se predice seguir en secuencia. En el ciclo 9 se comprueba que la predicción es correcta y por tanto, no se pierden ciclos.

En el ciclo 15 se utiliza un cortocircuito desde la etapa ALU a la etapa D/L para obtener el valor que se almacenará en el registro r11. En el mismo ciclo se predice modificar el secuenciamiento, ya que el literal es negativo y en el ciclo 16 se comprueba que la predicción es correcta. Por tanto, sólo se pierde el ciclo debido al retardo de búsqueda.



Los ciclos perdidos son

	Riesgos	
	datos	secuenciamiento
ciclos perdidos$_{NT}$	2	1

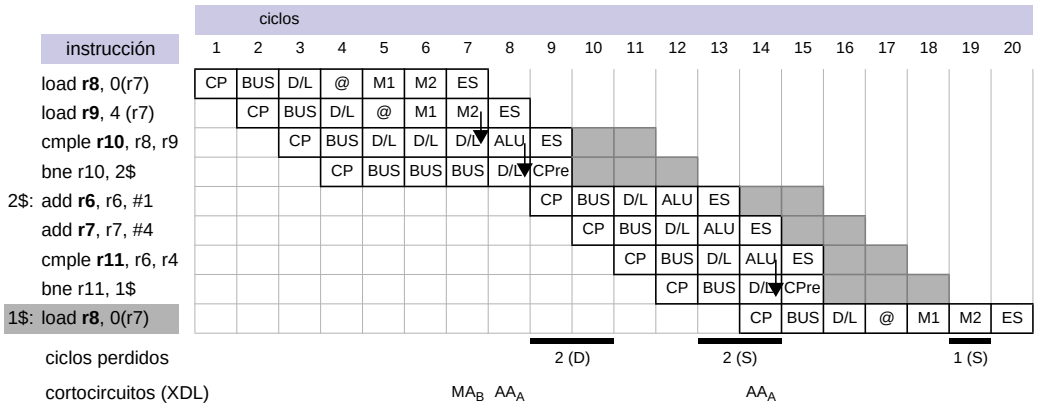
Los ciclos de ejecución y el CPI cuando no se toma el salto son:

En una iteración	salto no tomado
T_{NT} = número de instrucciones + ciclos perdidos	11 + 3 = 14 ciclos
$CPI_{NT} = T_{NT}$ / (número de instrucciones)	14 / 11 = 1.27

B) Salto tomado (T): Se cumple la condición de salto en la instrucción bne r10, 2$.

La diferencia con el caso anterior radica en que la predicción de sentido en la instrucción bne r10, 2$ es errónea.

En la siguiente figura se muestra la interpretación de una iteración del código. La predicción al interpretar la instrucción bne r10, 2$ es seguir en secuencia, ya que el literal es positivo. SIn embargo, como se cumple la condición, se produce un error de predicción de sentido.



Hasta el ciclo 8 el comportamiento es mimético al caso anterior. En el ciclo 9 se detecta un error en la predicción de sentido, se restaura el flujo correcto de instrucciones y se han perdido los ciclos 13 y 14.

En el ciclo 14 se utiliza un cortocircuito desde la etapa ALU a la etapa D/L para obtener el valor que se almacenará en el registro r11. En el mismo ciclo se predice modificar el secuenciamiento, ya que el literal es negativo y en el ciclo 15 se comprueba que la predicción es correcta. Por tanto, sólo se pierde el ciclo debido al retardo de búsqueda.

Los ciclos perdidos son

	Riesgos	
	datos	secuenciamiento
ciclos perdidos$_T$	2	3

Los ciclos de ejecución y el CPI cuando se toma el salto son:

En una iteración	salto tomado
T_T = número de instrucciones + ciclos perdidos	8 + 5 = 13 ciclos
$CPI_T = T_T$ / (número de instrucciones)	13 / 8 = 1.63

Suponga que al ejecutar las N iteraciones del bucle interno del algoritmo de la burbuja se producen un 30% de intercambios.

Pregunta 2: *Determine el CPI medio en el procesador multiciclo.*

Respuesta: Una forma de efectuar el cálculo es dividir los ciclos totales ponderados por el número de instrucciones ponderado.

$$CPI = \frac{\sum f_i \times T_i}{\sum f_i \times N_i}$$

Sustituyendo tenemos

$$CPI = \frac{0.3 \times T_{NT} + 0.7 \times T_T}{0.3 \times N_{NT} + 0.7 \times N_T} = \frac{0.3 \times 14 + 0.7 \times 13}{0.3 \times 11 + 0.7 \times 8} = \frac{4.2 + 9.1}{3.3 + 5.6} = 1.49$$

Inserción de un elemento en una lista ordenada

En la parte izquierda de la siguiente figura se muestra un trozo de código que inserta un elemento en una lista ordenada de mayor a menor. La lista tiene como mínimo dos elementos y el elemento que se inserta es menor que el primero y mayor que el último elemento de la lista. Antes de iterar el bucle la variable q apunta al elemento que se quiere insertar y las variables p y prev apuntan al primer elemento de la lista.

```
While (p!= null)
{    if (p->valor < q->valor)
     {    q->siguiente = p;
          prev->siguiente = q ;
          break
     }
     prev = p;
     p = p->siguiente ;
}
```

```
3$:  load r3, 0(r21)        contenido de p
     cmpeq r6, r3, r7       p!= null
     bne r6, 2$             ¿final de lista?
     load r1, 0(r3)         p->valor
     load r2, 0(r9)         q->valor
     cmple r5, r2, r1       p->valor ≥ q->valor
     bne r5, 1$             insertar si p->valor < q->valor
     load r10, 0(r20)       contenido de prev
     store r3, 8(r9)        q->siguiente = p
     store r9, 8(r10)       prev->siguiente = q
     br 2$                  break
1$:  load r15, 8(r3)        contenido de p->siguiente
     store r3, 0(r20)       prev = p
     store r15, 0(r21)      p = p->siguiente
     br 3$                  iterar
2$:
```

En la parte derecha de la figura se muestra una traducción a lenguaje ensamblador. Los registros r9, r20 y r21 contienen las variables q, prev y p respectivamente. El registro r9 es el contenido de la variable q. El registro r7 contiene el valor cero que se utiliza como codificación de null.

En todas las preguntas supondremos que al interpretar la instrucción bne r6, 2$ no se cumple la condición.

Pregunta 1: *Calcule los ciclos de ejecución en la iteración que se inserta el elemento. Así mismo, indique los ciclos perdidos por tipo de riesgo y calcule el CPI por iteración.*

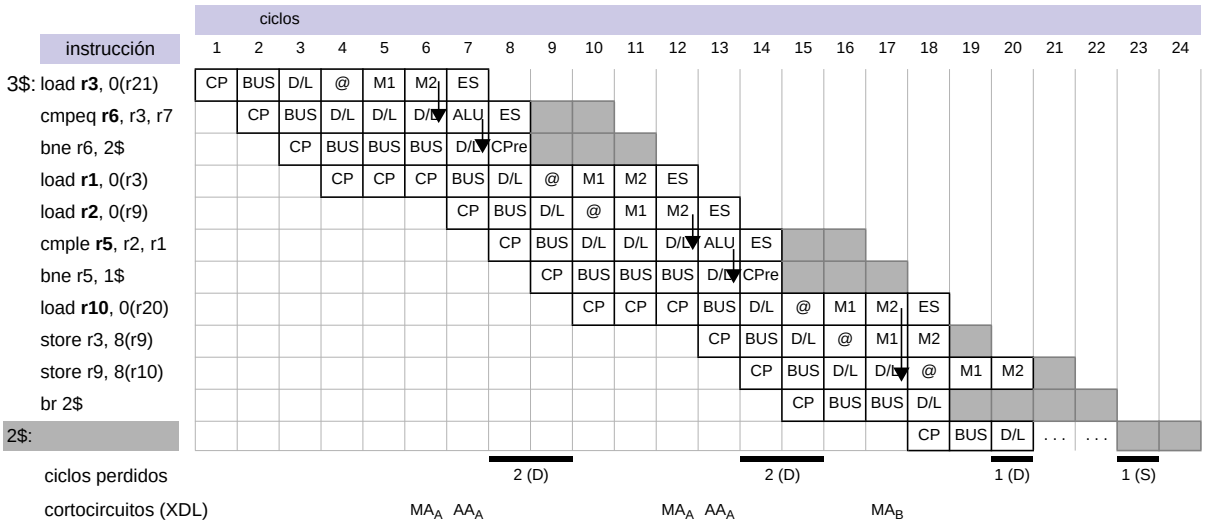
Respuesta: Cuando el elemento se inserta en la lista, al interpretar la instrucción bne r5,1$ no se cumple la condición. En la siguiente figura se muestra mediante un diagrama temporal la interpretación segmentada de este caso.

En el ciclo 4 se detecta un riesgo de datos debido al registro r3. Este riesgo permanece durante el siguiente ciclo y en el ciclo 6 se obtiene el valor, que se almacenará en el registro r3, mediante un cortocircuito desde la ramificación de memoria hasta la etapa D/L. En el ciclo 7 la instrucción bne r6, 2$ obtiene el dato fuente mediante un cortocircuito desde la etapa ALU a la etapa D/L. En este mismo ciclo se predice seguir en secuencia, ya que el signo del literal es positivo. En el ciclo 8 se comprueba que la predicción de sentido ha sido correcta.

En el ciclo 10 se detectan riesgos de datos debidos a los registros r1 y r2. El riesgo debido al registro r2 perdura hasta el ciclo 12, donde se obtiene el valor utilizando un cortocircuito desde la ramificación de memoria a la etapa D/L.

En el ciclo 13 se utiliza un cortocircuito desde la etapa ALU a la etapa D/L para disponer del valor que se almacenará en el registro r5. En este ciclo la instrucción que ocupa la etapa D/L es bne r5, 1$. Se predice seguir en secuencia y en el siguiente ciclo se comprueba que la predicción ha sido correcta. Por tanto, no se pierden ciclos por riesgo de secuenciamiento.

En el ciclo 17 se utiliza un cortocircuito desde la ramificación de memoria a la etapa D/L para obtener el valor que se almacenará en el registro r10.



En el ciclo 18 se modifica el secuenciamiento y se observa la penalización por retardo en la búsqueda en el ciclo 23.

Los ciclos perdidos son

	Riesgos	
	datos	secuenciamiento
ciclos perdidos	5	1

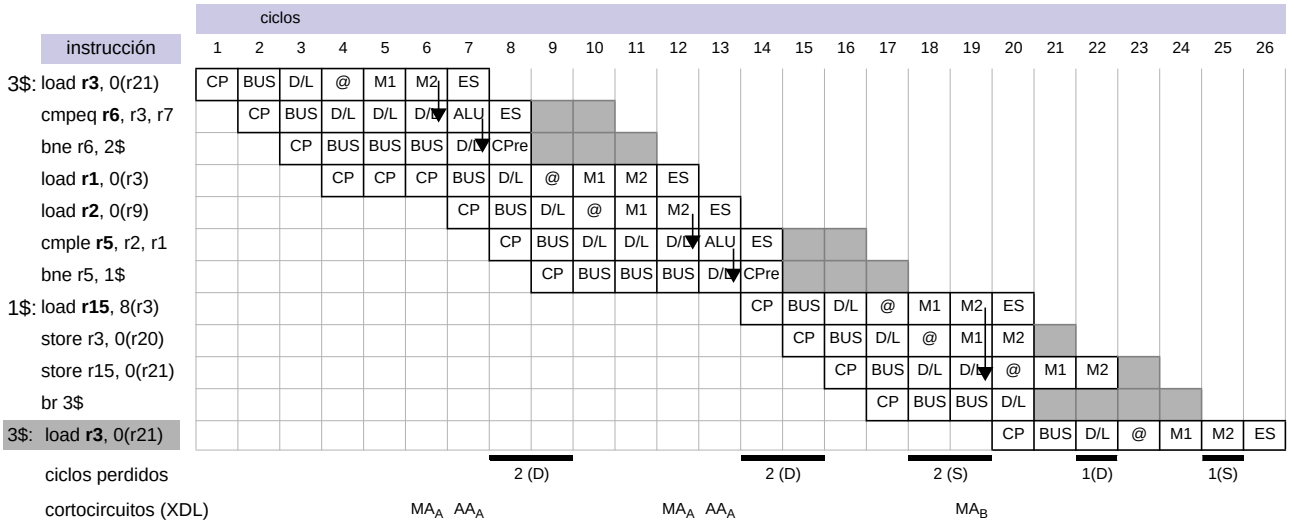
Los ciclos de ejecución y el CPI son:

En una iteración	salto no tomado
T_{NT} = número de instrucciones + ciclos perdidos	11 + 6 = 17 ciclos
$CPI_{NT} = T_{NT}$ / (número de instrucciones)	17 / 11 = 1.55

Pregunta 2: *Calcule los ciclos de ejecución en una iteración en la cual no se inserta el elemento. Así mismo, indique los ciclos perdidos por tipo de riesgo y calcule el CPI por iteración.*

Respuesta: Cuando el elemento no se inserta en la lista, al interpretar la instrucción bne r5,1$ se cumple la condición. Esto es, se modifica el secuenciamiento. En la siguiente figura se muestra mediante un diagrama temporal la interpretación segmentada de este caso.

La diferencia con la pregunta anterior radica en que se modifica el flujo de interpretación a partir de la instrucción de secuenciamiento bne r5, 1$. Entonces, como la predicción es seguir en secuencia se produce un error de predicción y hay que recuperarse.



Analizando a partir de la instrucción bne r5, 1$ tenemos que se detecta un error de predicción de sentido en el ciclo 14. La recuperación se inicia en el mismo ciclo y los ciclos perdidos se observan en los ciclos 18 y 19.

En el ciclo 18 se produce un riesgo debido al registro r15. En el ciclo 19 se obtiene el valor, que se almacenará en el registro r15, mediante un cortocircuito desde la ramificación de memoria a la etapa D/L. En el ciclo 20 se detecta un riesgo de secuenciamiento y la penalización debida a la búsqueda se observa en el ciclo 25.

Los ciclos perdidos son

	Riesgos	
	datos	secuenciamiento
ciclos perdidos$\|_{NT}$	5	3

Los ciclos de ejecución y el CPI son:

En una iteración	salto tomado
T_T = número de instrucciones + ciclos perdidos	11 + 8 = 19 ciclos
$CPI_T = T_T$ / (número de instrucciones)	19 / 11 = 1.73

Antes de empezar la inserción de un elemento el tamaño de la lista son 80 elementos y la inserción se efectúa después de haber recorrido en media un 40% de la lista.

Pregunta 3: *Determine el CPI medio en el procesador multiciclo.*

Respuesta: Para calcular el CPI medio hay que tener en cuenta la proporción de instrucciones en cada uno de los casos respecto del total y la fracción de veces que se recorre la lista y se inserte el elemento y el caso en que no se inserte el elemento.

$$CPI = \frac{N_{NT}}{N} \times CPI_{NT} + \frac{N_T}{N} \times CPI_T$$

donde N_{NT} y N_T son respectivamente el número de instrucciones cuando se inserta el elemento y cuando no se inserta y N es igual a $N = N_{NT} + N_T$, siendo $N_{NT} = 80 \times 0.4 \times 11 = 352$ y $N_T = 11$.

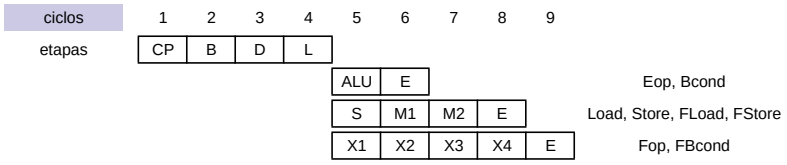
Por tanto,

$$CPI = \frac{80 \times 0.4 \times 11}{363} \times 1.55 + \frac{11}{363} \times 1.73 = 1.55$$

EJERCICIOS 5

Ejercicio 5.1

Un procesador segmentado multiciclo con 1.9 Ghz de frecuencia de reloj está segmentado en las siguientes etapas.



Hay 2 bancos de registros (enteros y coma flotante). Cada banco de registros tiene 2 caminos de lectura y 2 caminos de escritura. En cada banco de registro se puede escribir y leer un mismo registro en un ciclo de reloj.

El camino de datos sólo dispone de cortocircuitos para reducir la latencia efectiva de escritura al banco de registros de enteros.

Los riesgos de datos y de secuenciamiento se detectan en la etapa D. Cuando se detecta un riesgo de datos, se retienen las instrucciones en las etapas D, B y CP hasta que desaparece la condición de riesgo. Sólo las instrucciones con operando fuente entero pueden obtenerlo mediante cortocircuitos durante la etapa L.

Un riesgo de secuenciamiento se resuelve descartando las instrucciones buscadas hasta que se actualiza el contador de programa con la dirección de la instrucción de la siguiente instrucción que debe interpretarse (ciclo 5).

Este procesador ejecuta el programa P:

```
for (i=0; i<N; i++)
{
   p = X[i];
   *p = *p + s;
}
```

```
1$:  Load r1, 0(r0)      ;i1 r1 ← mem[r0+0]
     FLoad  f1, 0(r1)    ;i2 f1 ← mem[r1+0]
     FAdd f3, f1, f2     ;i3 f3 ← f1 + f2
     FStore f3, 0(r1)    ;i4 mem[r1+0] ← f3
     Add r0, r0, #8      ;i5 r0 ← r0 + 8
     Sub r2, r2, #1      ;i6 r2 ← r2 - 1
2$:  bne r2, 1$          ;i7 si (r2≠0) saltar a 1$

     Valores iniciales:
     r0 = dirección del primer elemento del vector X
     r2 = N; f2 = s.
```

Pregunta 1: *Dibuje el grafo de dependencias verdaderas, etiquetando los arcos del grafo con el retardo productor-uso.*

Pregunta 2: *Indique para cada instrucción de una iteración el ciclo en el que el control de la segmentación detecta que esté lista. Calcula el rendimiento en MIPS cuando el procesador ejecuta el programa.*

El compilador desenrolla el cuerpo del bucle 2 veces antes de planificar las instrucciones. Suponga que no hay dependencias de datos debido a posiciones de memoria entre instrucciones de iteraciones distintas.

Pregunta 3: *Muestre el código resultante después de desenrollar, reordenar y renombrar registros y constantes.*

Pregunta 4: *Para cada instrucción de una iteración del código resultante, indique el ciclo en que el control detecta que está lista. Calcule el rendimiento en MIPS del código resultante y la ganancia obtenida.*

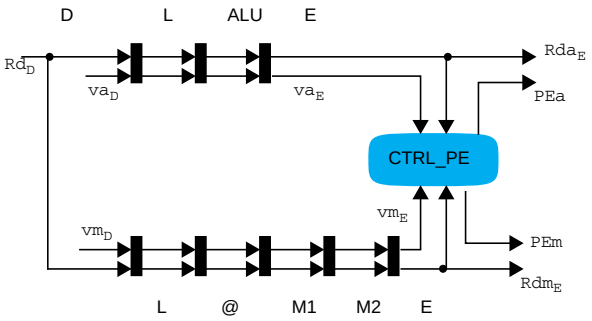
Pregunta 5: *Deduzca el número mínimo de veces que es necesario desenrollar el cuerpo del bucle para que, después de planificarlo, el procesador no detecte riesgos por dependencias de datos.*

Pregunta 6: *Calcule el rendimiento MIPS suponiendo que se desenrolla N veces el cuerpo del bucle (N es el nº de iteraciones).*

Se quiere modificar la gestión de los riesgos WaW (escritura después de escritura). En lugar de bloquear en la etapa D la instrucción destino de la dependencia para forzar que las escrituras al registro común se efectúen en orden, el control de la segmentación permitirá que la instrucción destino inicie la ejecución y anulará la escritura de la instrucción origen de la dependencia. La decisión de inhibir la escritura se toma cuando la instrucción origen está en la etapa E.

Pregunta 7: *Muestre, para cualquier programa, todos los posibles casos en que un riesgo WaW en registros enteros se puede resolver anulando las escrituras.*

Para la gestión de riesgos WaW se propone modificar la circuitería que controla las escrituras al banco de registros. La figura muestra el circuito correspondiente al banco de registros de enteros.



La parte superior propaga el identificador de registro destino asociado a la ramificación ALU, la parte inferior se corresponde con la ramificación MEM. Para cada camino de escritura, desde la etapa D se inyecta el identificador de registro destino (Rd_D) y el bit de validez (va_D y vm_D).

El módulo secuencial CTRL_PE observa en cada ciclo la petición de escritura de cada ramificación (Rda_E-va_E, Rdm_E-vm_E) y calcula las señales de permiso de escritura (PEa, PEm).

Pregunta 8: *Utilizando comparadores, puertas lógicas y registros, diseñe el módulo CTRL_PE.*

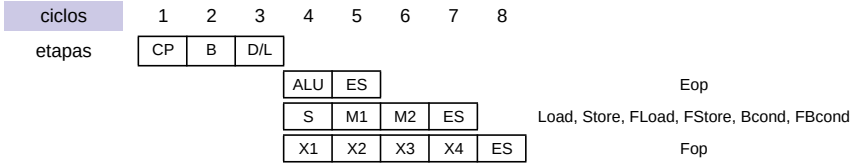
Ejercicio 5.2

Un procesador que interpreta los siguientes tipos de instrucciones

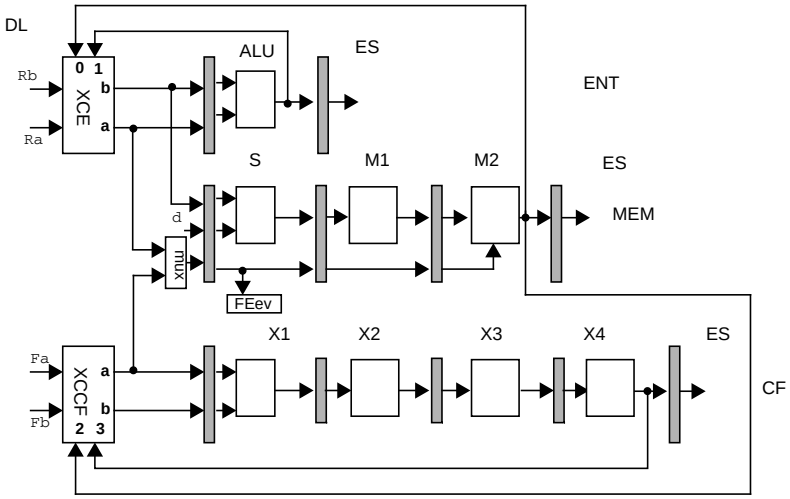
Tipo	Descripción
FOP	Fd = Fa op Fb
FLoad	Fd = M[Rb+d]
FStore	M[Rb+d] = Fa
FBcond	si cond (Fa) PC = PC + d en caso contrario PC = PC+4

Tipo	Descripción
Eop	Rd = Ra op Rb
Load	Rd = M[Rb+d]
Store	M[Rb+d] = Ra
Bcond	si cond RFa) PC = PC + d en caso contrario PC = PC+4

está segmentado en las siguientes etapas.



La figura muestra el camino de datos correspondiente a la parte de ejecución de las instrucciones. Hay 3 ramificaciones: ENT ejecuta las instrucciones de cálculo con enteros; MEM ejecuta las instrucciones de acceso a memoria y los saltos condicionales; CF ejecuta las instrucciones de cálculo en coma flotante. Los módulos etiquetados con XCE y XCCF contienen multiplexores para encaminar datos desde las salidas de las ramificaciones a los registros de entrada de las ramificaciones.



Pregunta 1: *Teniendo en cuenta los recursos del camino de datos de la figura y suponiendo que el procesador inicia la interpretación de una instrucción cada ciclo de reloj. Determine:*
el número máximo de instrucciones que pueden estar simultáneamente en fase de escritura
el numero máximo de instrucciones que pueden estar simultáneamente en fase de ejecución
el número máximo de instrucciones en proceso de interpretación.

Pregunta 2: *Suponiendo que la etapa de búsqueda suministra en cada ciclo 3 instrucciones a la etapa DL (una instrucción para cada ramificación). Calcule:*
el número mínimo de caminos de lectura a cada banco de registros
el número máximo de instrucciones que pueden estar simultáneamente en fase de escritura
el número máximo de instrucciones que pueden estar simultáneamente en fase de ejecución.

Cada banco de registros, que no se muestran en la figura, dispone de 2 caminos de lectura y 2 caminos de escritura. Cada banco permite la escritura y lectura de un mismo registro en un ciclo de reloj.

Los riesgos de datos y de secuenciamiento se detectan en la etapa DL. La detección de un riesgo de datos impide el progreso de las instrucciones en las etapas DL, B y CP hasta que desaparece la condición de riesgo. Un riesgo de secuenciamiento se resuelve descartando las instrucciones buscadas hasta que se actualiza el contador de programa con la dirección de la siguiente instrucción que debe interpretarse (etapa S). El módulo FEev evalúa la condición. En el camino de datos no se muestra el suministro de direcciones a la etapa CP.

Este procesador ejecuta el programa P:

```
for (i=0; i<N; ; i++)
{
   x[i]=x[i]+x[i-1];
}
```

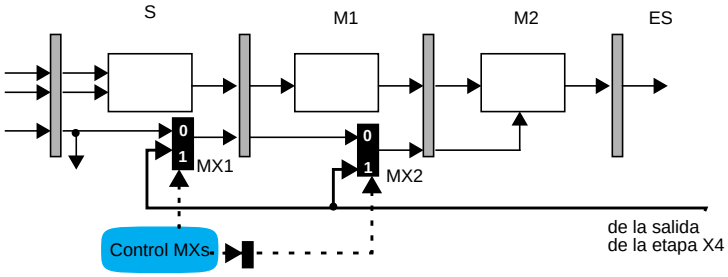
```
1$:  FLoad f0, 0(r0)      ;f0 ← mem[r0+0]
     FLoad f1, -8(r0)     ;f1 ← mem[r0-8]
     FAdd f1, f1, f0      ;f1 ← f1 + f0
     FStore f1, 0(r0)     ;mem[r0+0] ← f1
     Add r0, r0, r1       ;r0 ← r0 + r1
     Sub r2, r2, r3       ;r2 ← r2 - r3
2$:  bne r2, 1$           ;si (r2≠0) saltar a 1$

     Valores iniciales:
     r0 = dirección del primer elemento del vector X
     r1 = 8; r2 = N-1, r3=1.
```

Pregunta 3: *Muestre el cronograma de una iteración del bucle y la primera instrucción de la siguiente iteración. Indique los cortocircuitos utilizados. Un cortocircuito se identifica mediante la pareja* `o-d` *(módulos XCE y XCCF), donde* `o` *puede tomar los siguiente valores (0, 1, 2, 3) y* `d` *puede tomar los siguientes valores (a, b). Calcule el CPI.*

Queremos reducir el retardo productor-consumidor entre instrucciones de cálculo en coma flotante (productor) e instrucciones que actualizan la memoria de datos (consumidor) mediante cortocir-

cuitos. Se modifica el camino de datos original de la forma que se muestra en la figura, donde se observan multiplexores de encaminamiento de datos MX1 y MX2 en la ramificación MEM.



Pregunta 4: *Muestre el cronograma de ejecución de la secuencia de 5 instrucciones, indicando las entradas seleccionadas de los multiplexores MX1 y MX2 en las 3 instrucciones FStore. Indique el retardo productor-consumidor entre instrucciones Fop-FStore en el procesador original y en el nuevo procesador.*

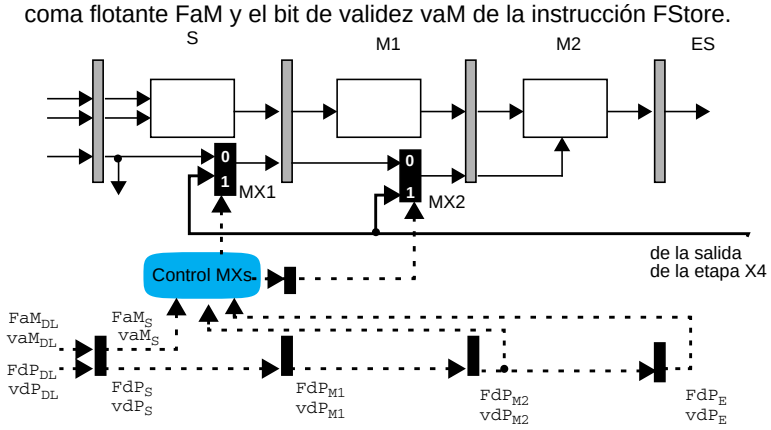
```
FAdd f3,f4,f5
FMul f4,f0,f2
FStore f3,0(r0)
FStore f3,8(r0)
FStore f3,16(r0)
```

Pregunta 5: *Calcule el CPI al ejecutar el programa P en el nuevo procesador.*

Las señales de control de los multiplexores MX1 y MX2 se determinan cuando una instrucción FStore ocupa la etapa S. En este ciclo se localiza la instrucción productora que se ejecuta en la ramificación CF. Para llevar a término esta función:

- Cuando se inicia la ejecución de una instrucción Fop, la etapa DL suministra a la ramificación MEM el identificador de registro destino FdP y el bit de validez vdP de la instrucción Fop. Esta información se propaga por las etapas de la ramificación MEM.
- Cuando se inicia la ejecución de una instrucción FStore, la etapa DL suministra a la ramificación MEM el identificador de registro fuente

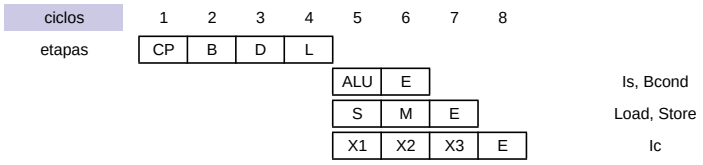
coma flotante FaM y el bit de validez vaM de la instrucción FStore.



Pregunta 6: *Utilizando comparadores y puertas lógicas, diseñe el módulo de control de los multiplexores MX1 yMX2.*

Ejercicio 5.3

Un procesador segmentado multiciclo con 3 ramificaciones está segmentado en las siguientes etapas:

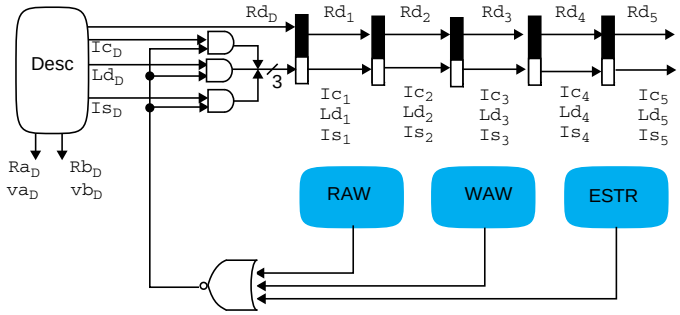


Hay dos tipos de instrucciones aritmético-lógicas con enteros: Is, con latencia de cálculo 1 ciclo; Ic, con latencia de cálculo de 3 ciclos. El Banco de Registros tiene 2 puertos de lectura y 1 puerto de escritura. El Banco permite también la escritura y lectura de un mismo registro en un ciclo de reloj. No hay cortcircuitos.

Los riesgos se detectan en la etapa D. Cuando se detecta un riesgo de datos o estructural, se retienen las instrucciones en las etapas D, B, CP y se inyecta una NOP hacia la etapa L hasta que desaparece la condición del riesgo.

Un riesgo de secuenciamiento se resuelve descartando las instrucciones buscadas hasta que se actualiza el contador de programa con la dirección de la instrucción que sigue a un salto condicional (ciclo 5).

Para gestionar los riesgos estructurales y de datos se utiliza el siguiente esquema:



El módulo Desc genera los identificadores de registro fuente Ra_D, Rb_D (y bits de validez), de registro destino Rd_D y ramificación que calculará el resultado de la instrucción que ocupa la etapa D (Ic_D, Ld_D, Is_D).

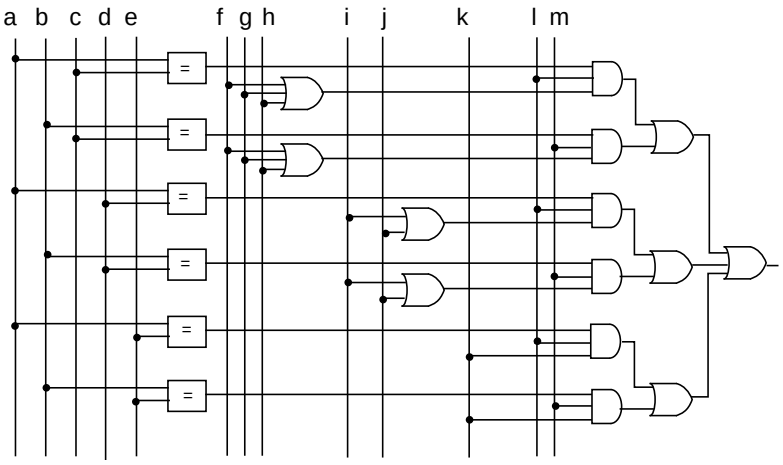
En la parte superior de la figura se muestran 5 elementos de memorización conectados en serie (número máximo de ciclos que una instrucción en la etapa D sin riesgos tarda en actualizar el Banco de Registros). A través de esta cadena de elementos se propaga el identificador de registro destino y la ramificación asociada.

Los módulos RAW, WAW y ESTR detectan si la instrucción que ocupa la etapa D tiene algún riesgo de datos o estructural con alguna instrucción más vieja. Estos módulos contienen sólo lógica combinacional y sus entradas pueden ser cualquier señal almacenada en la cadena de elementos de memorización y las salidas del módulo Desc.

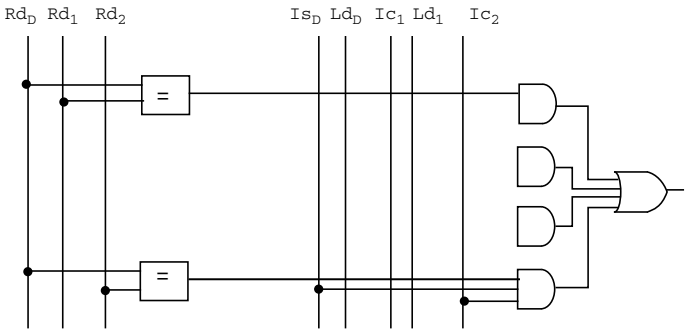
Pregunta 1: *Indique las latencias de inicio que provocan riesgos estructurales. Diseñe el módulo ESTR.*

Pregunta 2: *Suponga una pareja de instrucciones con dependencia verdadera y que las instrucciones que se interpretan entre la instrucción origen de la dependencia y la instrucción destino no detectan riesgos. Indique la distancia mínima entre la pareja de*

instrucciones para que no haya riesgo raw (lectura después de escritura).
Identifique las entradas del módulo RAW.



Pregunta 3: *Suponga una pareja de instrucciones con dependencia de salida y que las instrucciones que se interpretan entre la instrucción origen de la dependencia y la instrucción destino no detectan riesgos. Indique la distancia mínima entre la pareja de instrucciones para que no haya riesgo waw (escritura después de escritura). Complete el diseño del módulo WAW.*



Dado el siguiente programa:

```
for (i=0; i<N; ; i++)
{
   s = s + a * X[i];
}
```

```
for:  Load r1, 0(r0)      ;i1 r1 ← mem[r0+0]
      Add r0, r0, #8      ;i2 r0 ← r0 + 8 (Is)
      Mul r3, r1, r2      ;i3 r3 ← r1 * r2 (Ic)
      Cmpgt r5, r0, r6    ;i4 r5 ← (r0 > r6) (Is)
      Add r4, r3,r 4      ;i5 r4 ← r3 + r4 (Is)
      Beq  r5, for        ;i6 si (r5==0) salta a for
```

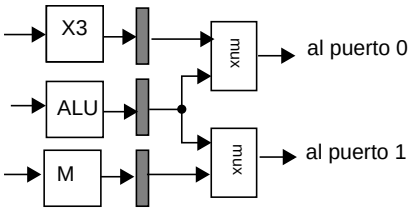
Valores iniciales:

r0 = dirección del primer elemento del vector
r6 = dirección de l'último elemento del vector; r2 = a; r4 = 0.

Pregunta 4: *Dibuje el grafo de dependencias de datos.*

Pregunta 5: *Muestre el cronograma de ejecución de les 6 instrucciones de una iteración y de la primera instrucción de la siguiente iteración. Indique la cantidad de ciclos perdidos por riesgos de datos, estructurales y de secuenciamiento. Calcule el CPI.*

Se añade un segundo puerto de escritura al Banco de Registros. Las salidas de las ramificaciones se conectan a los 2 puertos tal como muestra la figura.



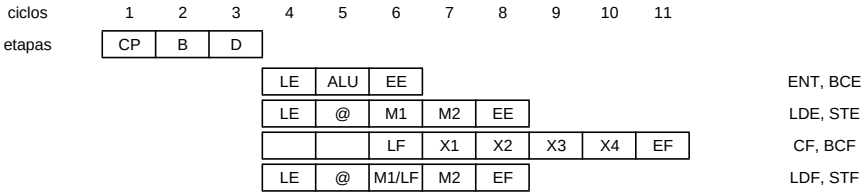
Pregunta 6: *Indique las combinaciones de instrucciones que provocan riesgos estructurales. Rediseñe el módulo ESTR.*

Ejercicio 5.4

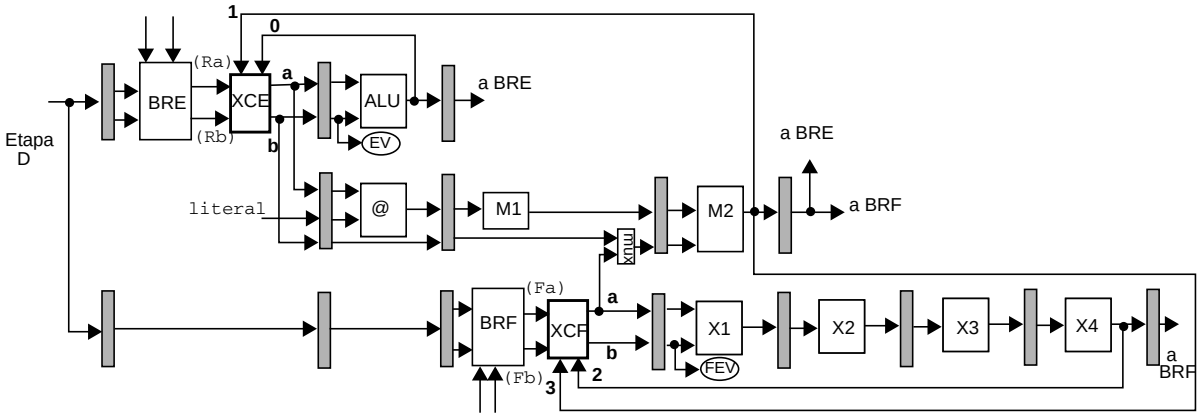
Un procesador segmentado multiciclo interpreta instrucciones del siguiente repertorio:

ENT	Rd = Ra op Rb	CF	Fd = Fa op Fb
LDE	Rd = M[Rb+d]	LDF	Fd = M[Rb+d]
STE	M[Rb+d] = Ra	STF	M[Rb+d] = Fa
BCE	si cond (Rb) entonces CP = CP+d sino CP = CP+4	BCF	si cond (Fb) entonces CP = CP+d sino CP = CP+4

Las instrucciones utilizan las etapas de acuerdo con los siguientes patrones:



El camino de datos correspondiente a la parte de ejecución dispone de 3 ramificaciones, tal como muestra la figura. La ramificación superior (ALU) ejecuta las instrucciones ENT y BCE. La segunda ramificación (MEM) ejecuta las instrucciones de acceso a memoria. La ramificación inferior (PF) ejecuta las instrucciones CF y BCF. El camino de datos dispone de suficientes recursos para que no se produzcan riesgos estructurales.



Cada banco de registros (BRE, BRF) tiene 2 puertos de lectura y 2 de escritura. En cada banco, un registro se puede escribir y leer, en este orden, en un mismo ciclo. Observe que la lectura en el banco BRF se efectúa dos ciclos después de la lectura en el banco BRE. Los módulos XCE y XCF contienen multiplexores para encaminar datos desde las salidas de las ramificaciones hacia los registros de desacoplo de entrada a las ramificaciones y de la etapa M2 (sólo el módulo XCF).

Los riesgos de datos RAW (lectura después de escritura) debido a registros se detectan en la etapa D. Cuando se detecta un riesgo, se bloquean las etapas D, B y CP. El procesador no utiliza ningún mecanismo para gestionar riesgos WAW (escritura después

de escritura) debido a registros. Las instrucciones han de estar ordenadas convenientemente para que las dependencias de salida no produzcan riesgo.

Para interpretar instrucciones de tipo BCE, el procesador utiliza predicción de sentido. En la etapa D se efectúa la predicción, se calcula la dirección destino del salto y se modifica el secuenciamiento si el literal (d) es negativo. La verificación de la predicción se realiza en la etapa ALU y se corrige el secuenciamiento en caso de detectar error. La interpretación de las instrucciones BCF se explica posteriormente.

El procesador ejecuta el siguiente fragmento de código:

```
for (i=1; i<N; i++)
  x[i] = x[i] + x[i-1];
```

```
1$:  Fload   f0,0(r0)          i1
     Fload   f1,-8(r0)         i2
     Fadd    f1,f0,f1          i3
     Fstore  f1,0(r0)          i4
     Add     r0,r0,r3          i5
     Cmpgt   r1,r0,r2          i6
     Beq     r1,1$             i7
```

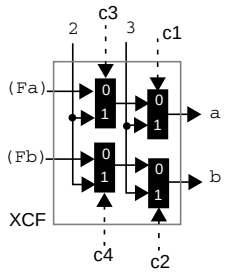
Valores iniciales:
r3=8
r0=dirección del segundo elemento del vector
r2=dirección del último elemento del vector

Pregunta 1: *Muestre el cronograma de ejecución de una iteración y de la primera instrucción de la siguiente iteración. Indique los cortocircuitos utilizados mediante la notación (x,y), donde x e y representan las entradas y salidas, respectivamente, de los módulos XCE y XCF. Calcule el CPI.*

El compilador puede reordenar las instrucciones y modificar los literales de las instrucciones de acceso a memoria para reducir los ciclos perdidos.

Pregunta 2: *Muestre el código planificado y los literales de las instrucciones de acceso a memoria. Para cada instrucción de una iteración y la primera de la siguiente iteración, indique en qué ciclo de reloj la lógica de control detecta que está Lista. Calcule el nuevo CPI y la Ganancia obtenida con la planificación.*

En la figura se muestran los multiplexores de encaminamiento de datos del módulo XCF. Las señales de control de los multiplexores se calculan en la etapa LF.



Pregunta 3: *Utilizando comparadores y puertas lógicas, diseñe el circuito de control del módulo XCF. No considere los bits de validez.*

Para detectar riesgos de datos, la lógica de control utiliza dos vectores de marcas, VME y VMF. El vector VME contiene tantos bits como registros enteros y el vector VMF tantos como registros de coma flotante visibles a nivel de lenguaje máquina. Cada posición de un vector indica si el registro asociado está disponible (valor 0) o si se está calculando (valor 1). La escritura en estos vectores está sincronizada con el flanco de subida de la señal de reloj. En el caso de que el retardo productor-uso sea igual a 1 ciclo, la instrucción productora no actualiza el vector de marcas. Suponga que el contenido de cualquier posición de un vector de marcas se puede escribir y leer, en este orden, en el mismo ciclo.

Pregunta 4: *Para cada tipo de instrucción que modifica el estado en registros, indique en qué flancos de subida del reloj se escriben los vectores de marcas. Deduzca el número de puertos de escritura de los vectores VME y VMF.*

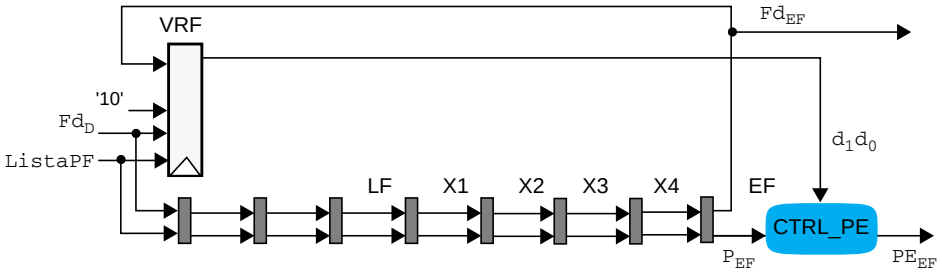
Pregunta 5: *Diseñe la lógica que evalúa si la instrucción que ocupa la etapa D está Lista. No considerare los bits de validez.*

Añadimos un mecanismo que no requiere bloqueo cuando se interpretan concurrentemente instrucciones con dependencia de salida. El mecanismo se basa en que instrucciones que se ejecutan en la misma ramificación no pueden provocar riesgo

WAW. Por tanto, es suficiente en anular la escritura de la instrucción origen de la dependencia cuando se ejecuta en una ramificación distinta de la instrucción destino.

Para cada banco, disponemos de un vector con tantas posiciones como registros lógicos (VRE y VRF). Cada posición indica la ramificación que calculará el resultado en el registro asociado. Supondremos la siguiente codificación para las ramificaciones: 0 (ALU), 1 (MEM), 2 (PF). La escritura en un vector está sincronizada con el flanco de subida del reloj. Un vector se actualiza cuando la lógica de control detecta que la instrucción que ocupa la etapa D está lista. La evaluación de la señal de permiso de escritura se efectúa en la etapa de escritura considerando el contenido de VRE o VRF (d_1,d_0). Suponga que el contenido de cualquier posición de un vector se puede escribir y leer, en este orden, en el mismo ciclo. Observe que el mecanismo descrito anula la escritura de la instrucción origen de la dependencia de salida incluso cuando la instrucción destino no provoca riesgo.

En la figura se muestra el circuito que controla la escritura de la ramificación de coma flotante. Desde la etapa D se inyecta hacia la cadena de registros el identificador del registro lógico que se actualizará (Fd_D).



Pregunta 6: *Diseñe con puertas lógicas el circuito CTR_PE que calcula la señal de permiso de escritura de la ramificación de coma flotante (PE_{EF}).*

La incorporación de este mecanismo para gestionar riesgos WAW requiere modificar el mecanismo que gestiona riesgos RAW, en concreto, la actualización de los vectores de marcas VME y VMF.

Pregunta 7: *Deduzca una secuencia de 3 instrucciones de tipo CF que evidencie la necesidad de modificar el mecanismo de gestión de riesgos RAW.*

Se quiere utilizar también predicción de sentido según el signo del literal para interpretar eficientemente las instrucciones de salto condicional de tipo BCF. La predicción se realiza en la etapa D, modificando el secuenciamiento si el literal es negativo. La verificación de la predicción se efectúa en la etapa X1 y se corrige el secuenciamiento en caso de error.

Pregunta 8: *En caso de error de predicción, deduzca el número máximo de instrucciones que la lógica de control debería descartar. Indique también si es posible que alguna instrucción mal predicha actualizase el estado antes de detectarse el error de predicción. En caso afirmativo, proponga un mecanismo para evitarlo.*

Ejercicio 5.5

Tenemos un procesador segmentado multiciclo con las siguientes características:

- El procesador está dividido en dos partes (entera y coma flotante).
- Cada banco de registros sólo tiene un camino de escritura y 3 caminos de lectura.
- La escritura se efectúa durante la primera mitad del ciclo y la lectura durante la segunda.
- En caso de riesgo estructural o de datos el procesador se bloquea en la etapa de decodificación.
- El procesador no tiene cortocircuitos.
- Se permite escritura en desorden de programa siempre que se respeten las dependencias.
- En caso de riesgo de secuenciamiento el procesador descarta las instrucciones buscadas hasta que se resuelven los saltos.

La parte entera está segmentada en las siguientes etapas:

ETAPA	FUNCIONALIDAD
CP	determinar la dirección de la instrucción
B	búsqueda de la instrucción
D	decodificación y lectura de datos en registros
A	alu, cálculo de la dirección y resolución de los saltos
M	acceso a memoria
E	escritura en el banco de registros de enteros

Este procesador dispone de las siguientes instrucciones de coma flotante con la segmentación que se indica:

ciclos	1	2	3	4	5	6	7	8
fload	CP	B	D	A	M	F		
fstore	CP	B	D	A	M	F		
fadd	CP	B	D	+	+	F		
fmul	CP	B	D	x	x	F		
fmadd	CP	B	D	x	x	+	+	F

Las etapas CP, B, D, A y M son las descritas para la parte entera, mientras que las siguientes etapas son específicas de la parte de coma flotante:

ETAPA	FUNCIONALIDAD
+	sumador de coma flotante
x	multiplicador de coma flotante
F	escritura en el banco de registros de coma flotante

La parte de coma flotante ejecuta las instrucciones aritméticas de coma flotante (fadd, fmul, fmadd), mientras que los accesos a memoria (fload y fstore) se ejecutan en la parte entera, pero también se accede al banco de registros de coma flotante. Las unidades de suma y multiplicación de coma flotante (utilizadas por las instrucciones fadd, fmul y fmadd) no están segmentadas y tardan dos ciclos.

Dado el siguiente bucle:

I1	fload f2 <- A(r1)
I2	fload f3 <- B(r1)
I3	fmul f4 <- f3 * f0
I4	fmadd f5 <- f3 * f1 + f2
I5	fadd f6 <- f2 + f0
I6	fmadd f7 <- f6 * f5 + f4
I7	fstore C(r1) <- f7
I8	add r1 <- r1 + #1
I9	bneq r1, r2, 1$

Pregunta 1: *Rellene un cronograma con la ejecución de una iteración del bucle y la primera instrucción de la siguiente iteración.*

Pregunta 2: *Indique el CPI (ciclos por instrucción) obtenido durante la ejecución del bucle.*

Pregunta 3: *Dibuje las dependencias del bucle en un grafo de dependencias.*

Pregunta 4: *Reescriba el código del bucle para minimizar los bloqueos.*

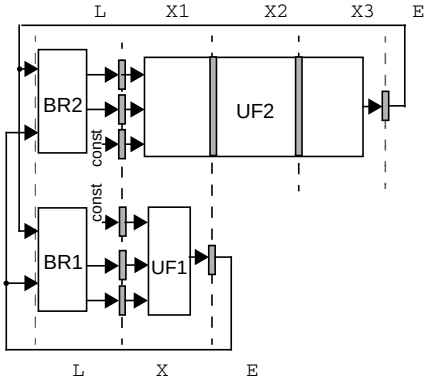
Pregunta 5: *Utilizando el código de la respuesta a la pregunta 4 rellene un cronograma con la ejecución de una iteración del bucle y la primera instrucción de la siguiente iteración.*

Ejercicio 5.6

Tenemos un procesador segmentado multiciclo con las siguientes etapas:

ETAPA	FUNCIONALIDAD
CP	determinar la dirección de la instrucción
B	búsqueda de la instrucción
D	decodificación, detección de riesgos, cálculo de la dirección de destino de salto
L	lectura de datos en registros
X, X1, ..., X3	ejecución
E	escritura en el banco de registros

La figura muestra el camino de datos, que no incluye las etapas CP, B y D. Los registros están replicados en 2 bancos, BR1 y BR2.



Los resultados producidos por las unidades funcionales se escriben simultáneamente en los dos bancos. La escritura ocupa un ciclo de reloj. No hay cortocircuitos. El procesador se bloquea al detectar un riesgo de datos.

UF1 ejecuta operaciones aritmético-lógicas y evalúa condiciones de salto (latencia 1 ciclo de reloj)

UF2 ejecuta operaciones de acceso a memoria (latencia 3 ciclos, latencia de iniciación = 1).

El procesador utiliza predicción fija saltar.

Dado el programa de prueba

```
for (i=0; i<N; ; i++)
{
   A[i] = A[i-1] + A[i] + C;
}
```

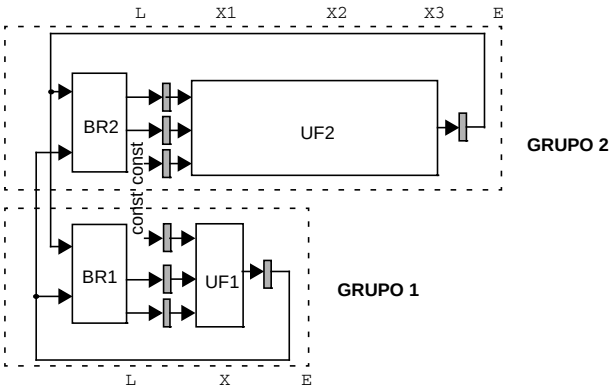
```
1$:  Load  r5, -4(r0)      ;r5 ← mem[r0-4]
     Load   r6, 0(r0)      ;r6 ← mem[r0+0]
     Add r7, r5, r6        ;r7 ← r5 + r6
     Add r7, r7, r8        ;r7 ← r7 + r8
     Store r7, 0(r0)       ;mem[r0+0] ← r7
     Add   r0, r0, #4      ;r0 ← r0 + 4
     Add   r1, r1, #-1     ;r1 ← r1 + (-1)
     Bnez r1, 1$           ;si r1 <> 0 entonces pc ← 1$
```

Valores iniciales:

r0 = dirección del primer elemento del vector A
r1 = N; r8= C

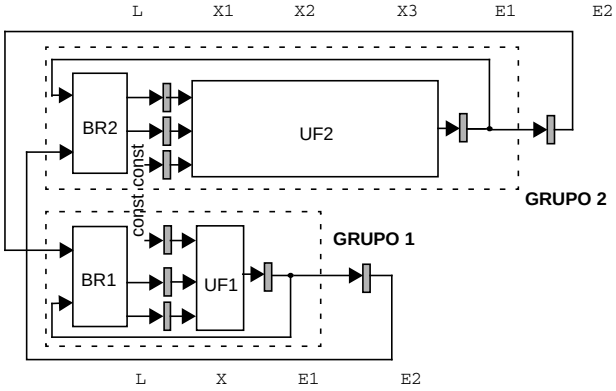
Pregunta 1: *Utilizando el procesador descrito previamente, dibuje el cronograma de una iteración del bucle, indicando claramente los ciclos de bloqueo. Calcule el CPI.*

Los componentes del camino de datos están distribuidos en 2 grupos como se muestra en la figura. El procesador dispone de cortocircuitos, con la restricción que el origen y destino de un cortocircuito deben de estar en el mismo grupo.



Pregunta 2: *Utilizando el procesador que se acaba de describir, muestre en un cronograma una iteración del bucle. Indique en el cronograma los cortocircuitos utilizados y dibújelos en el camino de datos. Calcule el CPI.*

En la siguiente figura se muestra un camino de datos. La etapa de escritura se descompone en 2 etapas: E1 y E2. En la etapa E1 se escribe el resultado en el banco de registros del grupo donde se ejecuta la instrucción y en la etapa E2 se escribe el resultado en el banco de registros del otro grupo. En el camino de datos no hay cortocircuitos.



Pregunta 3: *Utilizando el procesador que se acaba de describir, muestre el cronograma de ejecución de una iteración del bucle, indicando claramente los ciclos de bloqueo. Calcule el CPI.*

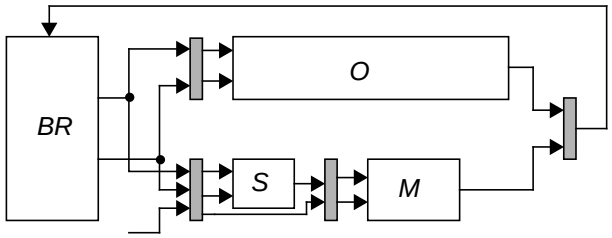
Ejercicio 5.7

Tenemos un procesador segmentado con las siguientes etapas:

ETAPA	FUNCIONALIDAD
CP	determinar la dirección de la instrucción
B	búsqueda de la instrucción
DL	decodificación, detección de riesgos, lectura de datos en registros
O, S, M	ejecución
E	escritura en el banco de registros

A nivel de lenguaje máquina distinguimos dos tipos de instrucciones de cálculo aritmético: `Is` (simples) e `Ic` (complejas). El resto de instrucciones del repertorio son: Load, Store y saltos.

La figura muestra el camino de datos, que no incluye las etapas CP y B.



La unidad funcional *O* efectúa las operaciones de cálculo y no está segmentada. Las operaciones simples `Is` tienen latencia 1 y las complejas `Ic` tienen latencia 3.

La unidad *S* calcula las direcciones efectivas de las instrucciones de acceso a memoria y la dirección destino de salto (latencia 1 ciclo).

La memoria de datos *M* no está segmentada: ejecuta las instrucciones Load con latencia de acceso 1 ciclo y las instrucciones Store con latencia de acceso 2.

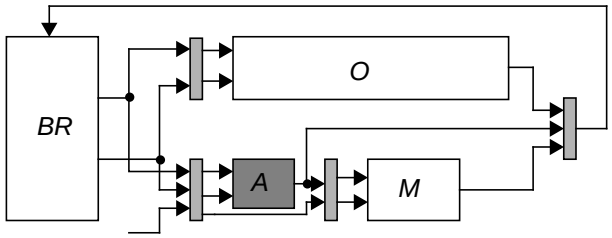
El banco de registros *BR* dispone de 2 caminos de lectura y 1 camino de escritura. El *BR* permite la escritura y después la lectura del mismo registro en un único ciclo.

La detección de riesgos se efectúa en la etapa DL. En caso de riesgo de datos o estructural, el procesador bloquea la búsqueda y decodificación de nuevas instrucciones durante los ciclos necesarios. No hay cortocircuitos.

Las instrucciones de salto condicional tienen latencia 3, es decir: la condición se evalúa en el mismo ciclo en que se calcula la dirección destino y se actualiza el CP con la dirección destino o secuencial. El procesador descarta (anula) las instrucciones buscadas mientras no se conoce el resultado del salto.

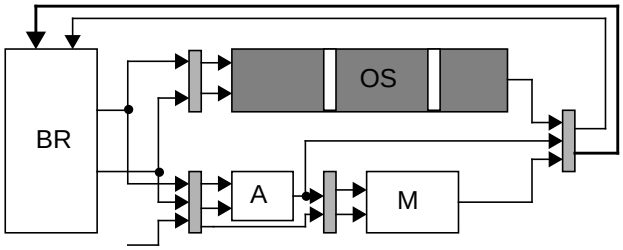
Pregunta 1: *En el procesador descrito anteriormente, determine las combinaciones de instrucciones que dan lugar a riesgo estructural. Así mismo, Para cada caso, indique los tipos de instrucciones, los recursos involucrados y los ciclos de bloqueo necesarios para resolver el riesgo.*

El camino de datos se modifica según se muestra en la siguiente figura: se sustituye la unidad *S* por la unidad *A*, la cual permite también ejecutar instrucciones de cálculo `Is` (latencia 1). La unidad *O* sólo ejecuta las instrucciones `Ic`.



Pregunta 2: *En el procesador descrito anteriormente, determine las combinaciones de instrucciones que dan lugar a riesgo estructural. Así mismo, Para cada caso, indique los tipos de instrucciones, los recursos involucrados y los ciclos de bloqueo necesarios para resolver el riesgo.*

Al camino de datos descrito últimamente se le añade un segundo camino de escritura al banco de registros y se sustituye la unidad *O* por la unidad *OS* (vea la figura). La nueva *OS* sólo ejecuta las instrucciones `Ic` (latencia 3) y está completamente segmentada.



Pregunta 3: *En el procesador descrito anteriormente, determine las combinaciones de instrucciones que dan lugar a riesgo estructural. Así mismo, Para cada caso, indique los tipos de instrucciones, los recursos involucrados y los ciclos de bloqueo necesarios para resolver el riesgo.*

Disponemos del siguiente programa de prueba.

```
for (i=0; i<N; ; i=i+2)
{
   X[i] = X[i]*a ;
   X[i+1] =X[i+1]*a;
}
```

```
1$:  Load r5, 0(r0)        ;r5 ← mem[r0+0]
     Load r6, 4(r0)        ;r6 ← mem[r0+4]
     Addr 1, r1, r2        ;r1 ← r1 + r2 (Is)
     Mul r5, r5, r8        ;r5 ← r5 * r8 (Ic)
     Mul r6,r 6,r 8        ;r6 ← r6 * r8 (Ic)
     Add r0, r0, r3        ;r0 ← r0 + r3 (Is)
     Storer 5, -8(r0)      ;mem[r0-8] ← r5
     Store r6, -4(r0)      ;mem[r0-4] ← r6
     Bnez r1, 1$           ;si r1 <> 0 entonces pc ← 1$

     Valores iniciales:
     r0 = dirección del primer elemento del vector X
     r1 = N; r2 = -2, r3 = 8, r8 = a.
```

Pregunta 4: *Suponiendo que la frecuencia del procesador de la 3ª pregunta es de 1.3 GHz, calcule la velocidad de ejecución en MIPS del programa se prueba. Justifique la respuesta dibujando el cronograma de ejecución de una iteración del bucle del programa de prueba.*

Ejercicio 5.8

Tenemos un procesador segmentado multiciclo con las siguientes etapas.

rc = ra op rb

load: rc = mem(ra + k)
store : mem(ra + k) = rb

La cache de datos tiene una latencia de 2 ciclos y la latencia de iniciación es 2. La detección de riesgos estructurales y riesgos por dependencias de datos a través de registros se efectúa en la etapa D, bloqueando la búsqueda y decodificación. El procesador dispone de todos los cortocircuitos necesarios.

El procesador ejecuta la siguiente secuencia de instrucciones:

A[i] = A[i-1] + A[i]
A[i+1] = A[i] + A[i+1]

```
Load  r5, -4(r0)     ;r5 ← mem[r0-4]
Load   r6, 0(r0)     ;r6 ← mem[r0+0]
Add r7, r5, r6       ;r7 ← r5 + r6
Store r7, 0(r0)      ;mem[r0+0] ← r7
Load r5, 0(r0)       ;r5 ← mem[r0+0]
Load  r6, 4(r0)      ;r6 ← mem[r0+4]
Add   r7, r5, r6     ;r7 ← r5 + r6
Store r7, 4(r0)      ;mem[r0+4] ← r7
```

Valores iniciales:
r0 = dirección del primer elemento del vector A

Pregunta 1: *Muestre un cronograma de ejecución de la secuencia de instrucciones previa en el procesador descrito, indique los ciclos perdidos por riesgos estructurales y por dependencias de datos.*

Para reducir los conflictos estructurales se propone implementar la cache de datos mediante dos unidades CDa y CDb idénticas. Cada unidad tiene una latencia de 2 ciclos y no está segmentada. Una instrucción `Load` utiliza una de las dos unidades; `Loads` consecutivos acceden alternativamente a las unidades CDa y CDb.

Las instrucciones `Store` escriben los resultados simultáneamente en las dos unidades.

Pregunta 2: *Muestre mediante un cronograma las combinaciones de instrucciones que dan lugar a riesgos estructurales.*

Pregunta 3: *Suponga que las instrucciones* `Store` *se ejecutan como las instrucciones* `Load` *(accediendo únicamente a una unidad) y que* `Loads` *y* `Stores` *consecutivos se ejecutan alternativamente en CDa y CDb.*
Razone si se pueden producir riesgos estructurales y riesgos por dependencias de datos a través de memoria.

Ejercicio 5.9

Tenemos un procesador segmentado multiciclo. Se utilizan dos tipos de segmentación, un tipo para todas las instrucciones de coma flotante (donde las operaciones aritméticas requieren 3 etapas y están totalmente segmentadas) y otro tipo para las instrucciones de memoria, de secuenciamiento y de enteros (donde las operaciones aritméticas requieren 2 etapas y están totalmente segmentadas). Las etapas son:

El procesador tiene dos bancos de registros separados (una para instrucciones de coma flotante y otro para las instrucciones de enteros). Las instrucciones de acceso a memoria pueden acceder a los dos bancos de registros.

La escritura de un registro ocupa todo el ciclo. Los riesgos se analizan en la etapa DL. Al detectar un riesgo, el procesador se bloquea. La dirección destino de las instrucciones de salto se calcula en la etapa DL y la condición de salto en la etapa AI2.

Pregunta 1: *¿Porqué motivos se ha de bloquear el procesador? ¿Cuántos ciclos se pierden en cada uno de los casos?*

El procesador ejecuta el siguiente bucle y el número de iteraciones es N:

```
1  1$   F1= Mem[R1 + A]      5  Mem[R1 + B]= F5
2       F2= F1 + F3          6  R3= Mem[R1 + C]
3       F4= Mem[R3 + D]      7  R1= R1 + 1
4       F5= F2 * F4          8  BEQZ R3, 1$
```

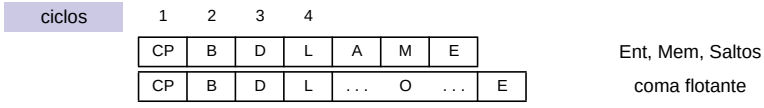
Pregunta 2: *En el procesador descrito previamente (sin cortocircuitos) se pide: a) cronograma de ejecución del código, b) número medio de ciclos por instrucción, c) porcentaje del tiempo total dedicado a resolver riesgos de secuenciamiento, d) porcentaje de tiempo dedicado a resolver riesgos de datos.*

En el procesador descrito se añaden los cortocircuitos necesarios, las instrucciones de secuenciamiento utilizan predicción fija saltar.

Pregunta 3: *Reordene el código para maximizar el rendimiento. Conocemos que inicialmente R1= 0 y que N+B < C. El código sólo se puede reordenar. Muestre en el cronograma de interpretación los cortocircuitos utilizados e indique el número de cortocircuitos distintos que hacen falta para ejecutar el código. Además indique: a) número medio de ciclos por instrucción, b) porcentaje del tiempo total dedicado a resolver riesgos de secuenciamiento, c) porcentaje de tiempo dedicado a resolver riesgos de datos.*

Ejercicio 5.10

Tenemos un procesador segmentado con dos tipos de segmentación. Para ejecutar las instrucciones de coma flotante, (etapa O), el procesador tiene una unidad funcional no segmentada multifuncional. El tiempo de cálculo depende de la operación que se realice (suma, producto, ...).



La funcionalidad del resto de etapas se describe en la siguiente tabla.

ETAPA	FUNCIONALIDAD
CP	determinar la dirección de la instrucción
B	búsqueda de la instrucción
D	decodificación
L	lectura del banco de registros
A	ALU
M	memoria
E	escritura en el banco de registros

Hay dos bancos de registros separados: una para instrucciones de coma flotante y otro para instrucciones enteras. Las instrucciones de acceso a memoria acceden a los dos bancos. Cada banco de registros tiene dos caminos de lectura y un camino de escritura. La escritura de un resultado en el banco de registro requiere todo el ciclo de reloj y las escrituras pueden efectuarse en desorden de programa.

La comprobación de riesgos se realiza en la etapa D. AL detectar un riesgo, el procesador se bloquea. No hay cortorcuitos.

En una instrucción de secuenciamiento condicional, el procesador descarta las instrucciones buscadas hasta que se obtiene el resultado del salto. El cálculo de la dirección destino y la evaluación de la condición se efectúan en la etapa A.

El procesador ejecuta el siguiente programa.

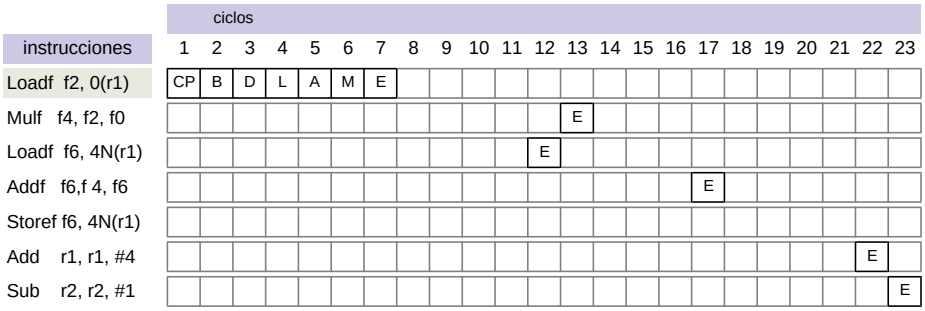
```
DO I=1, N
   Y(I) = A • X(I) + Y(I)
ENDDO
```

```
1$   Loadf  f2, 0(r1)      ; f2 ← mem[r1+0]
     Mulf   f4, f2, f0      ; f4 ¨ f2 • f0
     Loadf  f6, 4N(r1)      ; f6 ¨ mem[r1+4N]
     Addf   f6,f 4, f6      ; f6 ¨ f4 + f6
     Storef f6, 4N(r1)      ; mem[r1+4N] ¨ f6
     Add    r1, r1, #4      ; r1 ¨ r1 + 4
     Sub    r2, r2, #1      ; r2 ¨ r2 - 1
     Bnez   r2, 1$          ; si r2 <> 0 entonces saltar a 1$
```

Valores iniciales:

r1 = dirección del primer elemento del vector X
r2 = N; f0 = A.

En la siguiente figura se muestra, para una iteración del bucle, en qué ciclo cada instrucción escribe el resultado en el banco de registros.



Pregunta 1: *Deduzca el tiempo de cálculo (en ciclos) de las operaciones de suma y producto en coma flotante, completando un cronograma de ejecución. Indique también el CPI.*

Pregunta 2: *En otro procesador con la misma segmentación, conocemos que el tiempo de las operaciones de suma y producto de coma flotante son 1 y 5 ciclos respectivamente. Reordene el código para obtener el máximo rendimiento en este procesador. Muestre un cronograma de ejecución y calcule el CPI que se obtiene.*

Ejercicio 5.11

Un procesador segmentado lineal con 8 etapas.

ciclos	1	2	3	4	5	6	7	8
	CP	B	D	L	@	M1	M2A	E

En la siguiente tabla se describe la funcionalidad de las etapas.

ETAPA	FUNCIONALIDAD
CP	determinar la dirección de la instrucción
B	búsqueda de la instrucción
D	decodificación, detección de riesgos
L	lectura del banco de registros (segundo semiciclo)
@	cálculo de la dirección efectiva, o dirección destino de salto, evaluación de la condición de salto y verificación de la predicción
M1	inicio del acceso a memoria
M2A	final de acceso a memoria, operación aritmético-lógica
E	escritura en el banco de registros (primer semiciclo)

El procesador no dispone de cortocircuitos. La detección de riesgos de datos se efectúa en la etapa D, bloqueando la búsqueda y decodificación de instrucciones cuando se produce un riesgo.

Para la ejecución de las instrucciones de secuenciamiento condicional, el procesador utiliza un mecanismo de predicción fija no saltar. La evaluación de la condición y la verificación de la predicción se efectúan en la etapa @; en caso de error de predicción se anulan las instrucciones de la rama secuencial y se actualiza el CP con la dirección de la dirección destino de salto.

En el programa de prueba el compilador ha desenrollado el cuerpo del bucle dos veces, sin eliminar ninguna instrucción. El número de iteraciones es múltiplo de 8.

```
for (; p!=NULL; p->next)
   {
      ac = ac + p->data;
      p->data = ac;
   }
```

```
1$      Load r1,4(r0)      ;r1 ← mem[r0+4]
        Add r2,r1,r2       ;r2 ← r1 + r2
        Store r2,4(r0)     ;mem[r0+4] ← r2
        Load r0,0(r0)      ;r0 ← mem[r0+0]
        Beqz r0,2$         ;si (r0=0) salta a 2$
        Load r1,4(r0)      ;r1 ← mem[r0+4]
        Add r2,r1,r2       ;r2 ← r1 + r2
        Storer2,4(r0)      ;mem[r0+4] ← r2
        Load r0,0(r0)      ;r0 ← mem[r0+0]
1$ + 9  Bneqz r0,1$        ;si (r0≠0) saltar a 1$
2$
```

Valores iniciales:

r0 = dirección del primer elemento de p
r2 = ac; NULL = 0.

Pregunta 1: *Muestre el cronograma de ejecución de dos iteraciones consecutivas del bucle en alto nivel (es decir, la secuencia de instrucciones 1$, ... , 1$ +9). Indique los ciclos de bloqueo. Calcule el número medio de ciclos por instrucción al ejecutar el programa de prueba.*

Añadimos al procesador los cortocircuitos necesarios para reducir los bloqueos debidos a riesgos de datos.

Pregunta 2: *Muestre el cronograma de ejecución de dos iteraciones consecutivas del bucle en alto nivel (es decir, la secuencia de instrucciones 1$, ..., 1$ +9), indicando los cortocircuitos que se están utilizando. Indique los ciclos de bloqueo. Calcule el número medio de ciclos por instrucción al ejecutar el programa de prueba.*

Pregunta 3: *Calcule el CPI en dos casos: si el compilador no hubiese replicado el cuerpo del bucle (k = 1) y en el caso de que hubiese desenrollado el bucle 8 veces (k = 8) sin eliminar ninguna instrucción. No es necesario mostrar los cronogramas.*

Ejercicio 5.12

Un procesador segmentado multiciclo tiene 5 etapas.

ciclos	1	2	3	4			
	CP	B	D/L	. . .	EX	. . .	E

En la siguiente tabla se describe la funcionalidad de las etapas.

ETAPA	FUNCIONALIDAD
CP	determinar la dirección de la instrucción (1 ciclo)
B	búsqueda de la instrucción (1 ciclo)
D/L	decodificación, detección de riesgos y lectura del banco de registros (1 ciclo)
EX	acceder a memoria, calcular, resolver saltos (duración variable)
E	escritura en el banco de registros

El código a ejecutar es el siguiente bucle:

```
1$   load1 f0, 0(r2)      ; f0 ← mem[r2 + 0]
     Fload2 f4, 0(r3)     ; f4 ← mem[r3 + 0]
     Fop1 f0, f0, f4      ; f0 ← f0 * f4
     Fop2 f2,f 0, f2      ; f2 ← f0 + f2
     Iop1 r2, r 2, #8     ; r2 ← r2 + 8
     Iop2 r3,r 3, #8      ; r3 ← r3 + 8
     Iop3 r5, r4, r2      ; r5 ← r4 - r2
     Bnez r5, 1$          ; si (r5 <> 0) saltar a 1$
```

Valores iniciales:

f2 = 0
r2 = 100; r3 = 1000, r4 = 900.

En este procesador la etapa EX dura

1 ciclo para las instrucciones Iop

1 ciclo para las instrucciones Branch

2 ciclos para las instrucciones Fload

La memoria no está segmentada.

Todas las Fop se ejecutan en una UF segmentada que tiene esta Tabla de Reservas.

	ciclos			
	1	2	3	4
E1	X			X
E2		X	X	
E3	X			

Cualquier riesgo de datos o estructural bloquea al procesador en la etapa DL.

Dispone de todos los cortocircuitos (bypasses) que sean necesarios, hacia los registros de desacoblo de salida de la etapa DL.

Tiene dos bancos de registros separados, uno para enteros (registros `rx`) y otro para coma flotante (registros `fx`). Cada uno de los bancos tiene dos buses de lectura y un bus de escritura.

El orden de las escrituras puede ser cualquiera compatible con el algoritmo y con la cantidad de buses de escritura que hay.

Para cada una de las cuatro preguntas siguientes se pide:

- Representación del cronograma de ejecución, que incluya los bypasses utilizados.
- Señalar claramente los ciclos perdidos.
- Escribir la cantidad de ciclos desde que comienza una iteración hasta que comienza la siguiente.

Pregunta 1: *Se ejecuta exactamente el código del bucle tal como se ha proporcionado. Un salto bloquea al procesador hasta que se resuelve.*

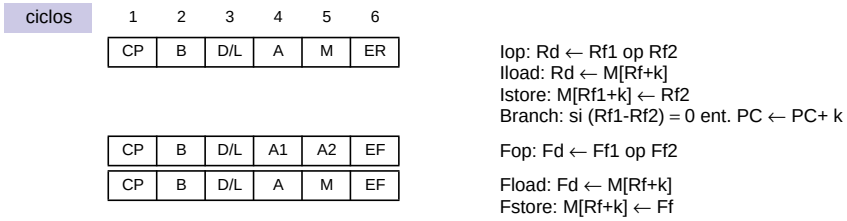
Pregunta 2: *Se reordena el código del bucle para reducir el tiempo de ejecución. Escribir la propuesta. Un salto bloquea al procesador hasta que se resuelve.*

Pregunta 3: *El código está reordenado como en la pregunta anterior, pero además los saltos son ahora "delayed branch". Escriba el código propuesto para reducir el tiempo de ejecución.*

Pregunta 4: *Se desenrollan DOS iteraciones sucesivas del bucle, y se pueden reescribir las constantes y renombrar los registros que convengan. Escriba la propuesta de código reordenado para reducir el tiempo de ejecución. El sentido de un salto se apuesta durante su tercera etapa según el signo del literal, y se acierta siempre que sigamos en el bucle.*

Ejercicio 5.13

Un procesador multiciclo tiene 2 bancos de registros separados, el de enteros (Rx) y el de coma flotante (Fx), y ejecuta las siguientes instrucciones segmentadas:



En la siguiente tabla se describe la funcionalidad de las etapas.

ETAPA	FUNCIONALIDAD
CP	determinar la dirección de la instrucción
B	búsqueda de la instrucción
D/L	decodificación, detección de riesgos y lectura del banco de registros
A	calcular con enteros, evaluar condición y modificar CP si es necesario
M	acceder a memoria de datos si es necesario
ER	escritura en el banco de registros de enteros, si es necesario
A1	primera etapa de calcular coma flotante
A2	segunda y última etapa de calcular coma flotante
EF	escritura en el banco de registros de coma flotante, si es necesario

Este procesador tiene suficientes recursos (unidades de cálculo, buses con los bancos de registros), para no bloquearse nunca por conflicto estructural. Tiene también suficientes cortocircuitos para tratar cualquier riesgo de datos y evitar todos los casos de bloqueo que sea posible. Los cortocircuitos pueden llegar hasta el comienzo de cualquier etapa (A, M, A1) que convenga.

Todas las lecturas de registros fuentes se hacen en la segunda mitad del ciclo correspondiente. Las escrituras de registro destino se hacen en la segunda mitad del ciclo correspondiente EXCEPTO en las instrucciones Fop que escriben en la primera mitad.

Este procesador es capaz de buscar dos instrucciones en un mismo ciclo y decodificar ambas simultáneamente al comienzo del siguiente ciclo, SOLAMENTE cuando sea una pareja de instrucciones consecutivas Iop/Fop y en este orden. En cualquier otro caso se limita a buscar y decodificar una nueva instrucción a cada ciclo.

En cada ciclo que está bloqueado no se busca ninguna instrucción, pero las instrucciones ya decodificadas sin riesgos siguen su camino, incluso las que completan pareja.

Pregunta 1: *Deduzca en qué casos debe este procesador bloquearse por riesgo de datos, a pesar de tener todos los cortocircuitos posibles, y presente un ejemplo de cada caso.*

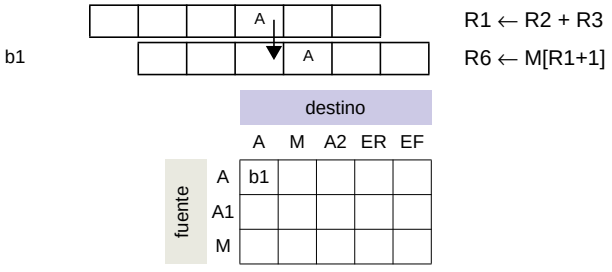
Pregunta 2: *Deduzca si puede haber riesgo en escritura por dependencia de datos, y si es así presente un ejemplo de cada caso que suceda.*

Pregunta 3: *Muestre el cronograma de ejecución en este procesador del código que se adjunta, identificando cada etapa, cada ciclo de bloqueo y cada cortocircuito empleado.*

Fload	`F1←M[R1+3]`
Iload	`R4←M[R2+3]`
Iop	`R3←R4+R3`
Fop	`F3←F3-F1`
Iop	`R5←R5-1`
Fstore	`M[R1-6]←F3`
Iop	`R6←R3+R3`
Fop	`F4←F2+F3`

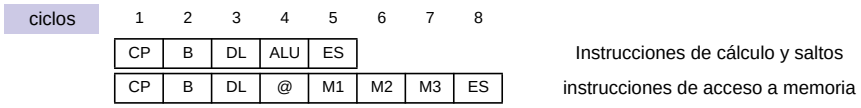
Pregunta 4: *Deduzca cuántas unidades de cálculo tiene este procesador, y decir cuáles son.*

Pregunta 5: *Deduzca los cortocircuitos que tiene este procesador. Exprese la respuesta rellenando la tabla adjunta y presentando un ejemplo para cada uno, tal como se muestra en el caso que se regala.*



Ejercicio 5.14

Disponemos de un procesador segmentado multiciclo con las siguientes etapas.



El repertorio de instrucciones es el utilizado en este capítulo.

La memoria de datos se puede considerar como una unidad funcional segmentada con latencia de iniciación igual a 1 y con una latencia de 3 ciclos de reloj.

El procesador dispone de suficientes recursos para que no se produzcan riesgos estructurales, pero no dispone de cortocircuitos. Cuando se detecta un riesgo de datos (etapa DL), el procesador se bloquea hasta que desaparece el riesgo. La escritura de un resultado en el banco de registros ocupa todo el ciclo de reloj. La escritura en el banco de registros puede ser en desorden de programa.

En caso de salto condicional, el procesador descarta las instrucciones buscadas hasta que se obtiene el resultado del salto (el cálculo de la dirección destino y la evaluación de la condición se efectúan en la etapa ALU).

Este procesador ejecuta el siguiente fragmento de código.

DO I=1, 100

Z(I) = X(I) + Y(I) + Z(I)

ENDDO

```
1$  Load r0,X(r10)       ; r0 ← mem[X+r10]
    Load r1,Y(r10)       ;r1 ← mem[Y+r10]
    Add r2,r0,r1         ;r2 ← r0 + r1
    Load r3,Z(r10)       ;r3 ← mem[Z+r10]
    Add r4,r2,r3         ;r4 ← r2 + r3
    Store Z(r10),r4      ;mem[Z+r10] ← r4
    Sub r10,r10,#1       ;r10 ← r10 - 1
    Bnez r10, 1$         ;si r10 <> 0 entonces
                         saltar a 1$
```

Valores iniciales:

X = 1000, Y = 2000, Z = 3000
r10 = 100.

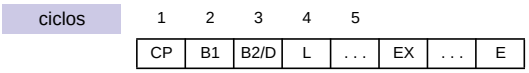
Pregunta 1: *Calcule los ciclos por iteración.*

Pregunta 2: *Reordene el código con el objetivo de mejorar lo máximo posible el rendimiento. Calcule los ciclos por iteración.*

Pregunta 3: *El compilador desenrolla 2 iteraciones el cuerpo del bucle y reordena las instrucciones con el objetivo de mejorar lo máximo posible el rendimiento. Muestre el código resultante. Suponga que el número de registros del procesador es ilimitado. Calcule los ciclos por iteración.*

Ejercicio 5.15

Dispone de un procesador segmentado multiciclo de 6 etapas.



En la siguiente tabla se describe la funcionalidad de las etapas.

ETAPA	FUNCIONALIDAD
CP	determinar la dirección de la instrucción
B1	inicio de la búsqueda de la instrucción
B2/D	finalización de la búsqueda de la instrucción y decodificación
L	detección de riesgos y lectura de datos del banco de registros (segundo semiciclo)
EX	ejecución (durante los ciclos que dure la operación)
E	escritura en el banco de registros (en el primer semiciclo)

Conocemos las siguientes características del procesador:

- puede decodificar una instrucción por ciclo, si no está bloqueado.
- cuando detecta un riesgo, por conflicto en una unidad funcional (UF) o por dependencia de datos, el control bloquea el procesador durante los ciclos que dura el riesgo.
- no dispone de cortocircuitos.
- hay 3 caminos de acceso al banco de registros, 2 de lectura y 1 de escritura.
- la escritura del resultado se efectúa siempre en orden de programa y se mantiene ocupada la UF hasta el ciclo anterior al de escritura en el banco de registros.
- dispone de 2 unidades funcionales: UF1 y UF2. La UF1 está segmentada en 4 etapas; la utilización de las etapas se describe mediante la siguiente tabla de reserva.

	ciclos				
	1	2	3	4	5
E1	X				
E2		X		X	
E3			X		X
E4				X	

- el control de iniciaciones de UF1 siempre sigue la secuencia óptima
- la UF2 no está segmentada y la latencia es de 3 ciclos.

Para estimar las prestaciones de este procesador se utiliza el siguiente código de prueba, donde *op1* es la operación que realiza la UF1 y *op2* la que realiza la UF2:

```
1   F3 ← F1 op1 F2
2   F5 ← F4 op1 F1
3   F6 ← F2 op2 F4
4   F3 ← F2 op1 F7
5   F1 ← F8 op1 F9
6   F5 ← F6 op2 F4
7   F2 ← F2 op2 F6
```

Pregunta 1: *Calcule el número de ciclos que tarda en ejecutarse el código de prueba, representándolo en un cronograma de ejecución. Marque el retardo de escritura con el acrónimo R.*

Pregunta 2: *Repita la 1ª pregunta suponiendo que existen cortocircuitos y que se utilizan sólo si se reducen los ciclos que tarda en ejecutarse el código de prueba.*

Pregunta 3: *Repita la 1ª pregunta suponiendo que no existen cortocircuitos, pero que existe un buffer de resultados que permite que las UF queden libres cuando finalizan una operación y se escribe en este buffer en cualquier orden. La escritura en el banco de registro se mantiene en orden de programa.*

Ejercicio 5.16

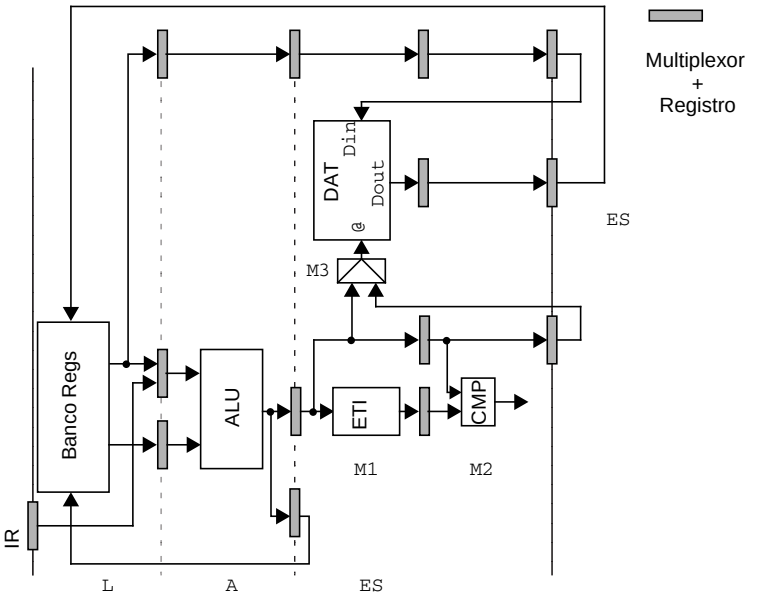
Disponemos de un procesador segmentado multiciclo con las siguientes etapas.

ciclos	1	2	3	4	5	6	7	8	
	CP	B	D	L	A	ES			Instrucciones aritméticas
	CP	B	D	L	A	M1/M3	M2	ES	instrucciones LOAD
	CP	B	D	L	A	M1	M2	M3	instrucciones STORE

donde

ETAPA	FUNCIONALIDAD
CP	determinar la dirección de la instrucción
B	inicio de la búsqueda de la instrucción
D	decodificación, detección de riesgos de datos en registros y estructurales
L	lectura de datos del banco de registros
A	operación aritmética o cálculo de la dirección destino
M1	lectura de etiquetas cache datos
M2	comprobación de acierto/fallo
M3	lectura o escritura del dato en cache datos
ES	escritura en el banco de registros

En la siguiente figura se muestra el camino de datos que no incluye las etapas CP, B y D. El procesador no dispone de cortocircuitos.



Hay 2 caminos de escritura al banco de registros.

La cache de datos es de correspondencia directa y consta de 3 módulos: ETI, donde se almacenan las etiquetas de los bloques; DAT donde se guardan los datos; CMP para comparar las etiquetas y determinar si es acierto o fallo. Las instrucciones Load acceden a ETI y DAT simultáneamente en el ciclo 6. Las instrucciones Store escriben el dato en DAT en el ciclo 8. La comprobación de etiquetas, tanto en Loads como en Stores, se efectúa en el ciclo 7 (supondremos que las instrucciones de acceso a memoria siempre aciertan en la cache).

Los riesgos de datos en registros y riesgos estructurales se detectan en la etapa D, bloqueando la búsqueda y decodificación de instrucciones posteriores.

Para iniciar la ejecución de una instrucción dependiente de un Load se supondrá que el Load acertará en cache.

Pregunta 1: *Indique en qué situaciones se habrá de bloquear el procesador para eliminar los riesgos estructurales y cuántos ciclos se pierden en cada caso.*

Pregunta 2: *Indique en qué situaciones se habrá de bloquear el procesador para eliminar los riesgos de datos en memoria. ¿ En qué etapa es conveniente detectar el riesgo?.*

Se rediseña el procesador con cortocircuitos, los cuales pueden tener cualquier etapa destino.

Supongamos la siguiente secuencia de instrucciones y valores iniciales en registros y memoria.

```
DO I=1, 100
   Z(I) = X(I) + Y(I) + Z(I)
ENDDO
```

```
1$   Store  r7, 4(r29)      ; mem[r29 + 4] ⟵ r7
     Store r8, 8(r29)       ; mem[r29 + 8] ⟵ r8
     Add r7, r0, r1         ; r7 ⟵ r0 + r1
     Load r8, 0(r2)         ; r8 ⟵ mem[r2 + 0]
     Load r6, 0(r3)         ; r6 ⟵ mem[r3 + 0]
     Add r9, r8, r6         ; r9 ⟵ r8 + r6
     Store r5, 0(r7)        ; mem[r7 + 0] ⟵ r5
     Load r4,0(r8)          ; r4 ⟵ mem[r8 + 0]
```

Valores iniciales:

r0 = 0, r1 = 1000, r2 = 2000, r3 = 3000, r29 = 29000
mem[2000] = 1000.

Pregunta 3: *Muestre el cronograma de ejecución de las 8 instrucciones, deduciendo los cortocircuitos utilizados. Identifique cada cortocircuito con una letra (a, b, ...) y añádalo al camino de datos. Considere los riegos de datos debido a registros y los riesgos de datos debido a memoria.*

Ejercicio 5.17

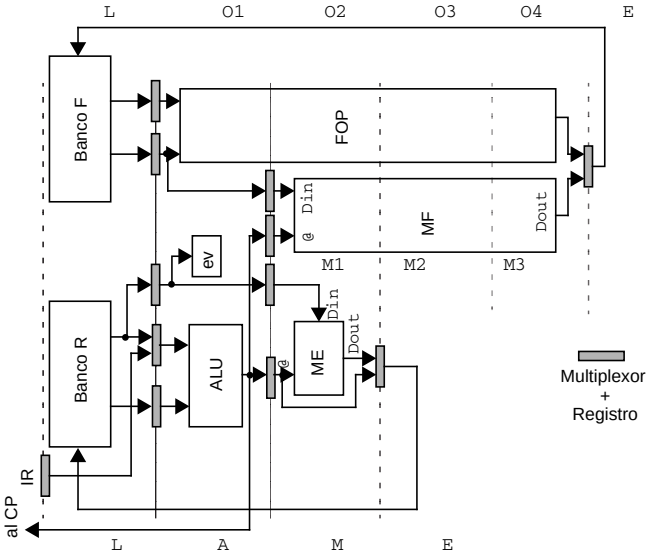
Un procesador segmentado multiciclo ejecuta las siguientes instrucciones.

ciclos	1	2	3	4	5	6	7
	CP	B	D	L	A	M	E

	CP	B	D	L	O1	O2	O3	O4	E
	CP	B	D	L	A	M1	M2	M3	E

Load:	rc = Me(ra+k)
Store:	Me(ra+k) = rb
Op:	rc = ra op rb
Bc:	si (cond rb), cp = cp + k
Fop:	fc = fa op fb
FLoad:	fc = Mf(ra+k)
FStore:	Mf(ra+k) = fb

En las siguiente figura se muestra el camino de datos, que no incluye las etapas CP, búsqueda y decodificación y tampoco muestra los cortocircuitos. Hay 2 bancos de registro; uno de ellos para valores que se interpretan como números enteros y otro para valores que se interpretan como números en coma flotante. La escritura de resultados se efectúa en el primer semiciclo y la lectura en el segundo semiciclo. Hay dos memorias de datos separadas (ME para enteros y MF para coma flotante).



El procesador dispone de los recursos necesarios para que no se produzcan riesgos estructurales.

La detección de riesgos de datos se efectúa en la etapa D, bloqueando el flujo de instrucciones en las tres primeras etapas hasta que desaparece el riesgo.

En los riesgos de secuenciamiento se descartan las instrucciones buscadas hasta que se resuelve el salto en la etapa A.

El procesador ejecuta el siguiente código.

```
for i =1, N
    B(i) = B(i) + A(Index(i))
endfor
```

```
1$   r1 ← Me(r0+0)
     r1 ← r2 + r1
     f2 ← Mf(r1+0)
     f3 ← Mf(r3+0)
     Mf(r3+0) ← f3
     r0 ← r0 + 4
     r3 ← r3 + 8
     r4 ← r4 - 1
     si r4 ≠ 0 entonces ir a 1$
```

Valores iniciales:

r0 = dirección del vector Index
r2 = dirección del vector A
r3 = dirección del vector B
r4 = N.

Pregunta 1: *Muestre el cronograma de ejecución de una iteración del bucle. Indique los ciclos perdidos y los cortocircuitos utilizados, etiquetándolos en el camino de datos. Calcule el número de ciclos por iteración.*

Pregunta 2: *Reordene el código para reducir los ciclos perdidos. Muestre el código resultante y el cronograma de ejecución de una iteración. Calcule el porcentaje de reducción en el tiempo de ejecución respecto del código original.*

Se modifica la ejecución de las instrucciones `FStore`: el procesador se bloquea sólo si el operando entero `rb` que se utiliza para calcular la dirección efectiva no está calculado. Si el operando de coma flotante `fb` no está disponible, pero se puede calcular la dirección efectiva, se inicia la ejecución de la instrucción `FStore`, retardando la operación de escritura en la memoria MF hasta que

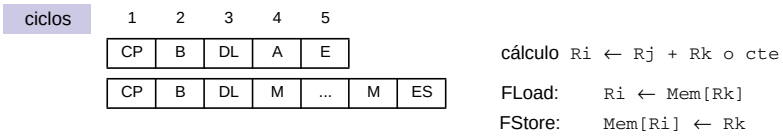
el operando `fb` está calculado. Durante los ciclos de retardo el procesador no puede iniciar la ejecución de ninguna instrucción `FLoad` o `FStore` posterior.

Pregunta 3: *Muestre, mediante un cronograma, si se pueden dar las situaciones de riesgos de datos escritura después de lectura o escritura después de escritura en la ejecución de los 2 siguientes fragmentos de códigos.*

CODIGO 1	CODIGO 2
`f3 ← f2 + f3`	`f3 ← f2 + f3`
`Mf(r3+0) ← f3`	`Mf(r3+0) ← f3`
`f3 ← f0 + f1`	`f3 ← Mf(r2+0)`

Ejercicio 5.18

Un procesador segmentado tiene las siguientes etapas.



La memoria de datos se puede considerar una unidad funcional no segmentada con latencia de operación variable (1 o 3 ciclos).

El procesador dispone de cortocircuitos: ALU/ALU y ALU/M. La escritura en el banco de registros se efectúa en la primera mitad del ciclo, mientras que la lectura se efectúa en la segunda mitad del ciclo.

El procesador se bloquea cuando se detecta un riesgo estructural o un riesgo de datos. El bloqueo inhibe la búsqueda y decodificación de nuevas instrucciones mientras dura el riesgo.

La escritura del resultado en el banco de registros puede ser en desorden de programa. Hay 2 caminos de escritura al banco de registros (desde la ramificación ALU y desde la ramificación M).

Las instrucciones de secuenciamiento condicional se resuelven en la etapa ALU.

Queremos evaluar el rendimiento del procesador con varios mecanismos. Para ello utilizaremos el siguiente programa de prueba, donde el valor que figura a la derecha de las instrucciones Load/Store indica la latencia de la memoria.

```
for i=0 to N-1                  R2 ← N
   a(i+1):=a(i) + a(i+1)        R0 ← A
   b(i) := a(i+1)               R1 ← A +4
   k:=k+2                       R3 ← B
endfor                    1$    R4 ← Mem[R0]          1
                                R5 ← Mem[R1]          3
                                R2 ← R2 - 1
                                R0 ← R0 + 4
                                R1 ← R1 + 4
                                R8 ← R8 + 2
                                R4 ← R4 + R5
                                Mem[R0] ← R4          3
                                Mem[R3] ← R4          3
                                R3 ← R3 + 4
                                si R2 ≠ 0 entonces ir a 1$
```

Pregunta 1: *Determine el tiempo de ejecución del bucle suponiendo que itera N veces. Justifique la respuesta mediante un cronograma con la ejecución de las 2 primeras iteraciones.*

Supongamos que añadimos al procesador una cola FIFO entre las unidades de decodificación y memoria. Esta FIFO tiene capacidad para almacenar 3 instrucciones Load/Store pendientes. En caso de que la memoria no esté disponible cuando se decodifica una instrucción Load/store, se encola la operación.

Pregunta 2: *Indique los campos de una entrada de la cola FIFO.*

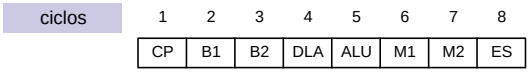
Pregunta 3: *Muestre en un cronograma las 2 primeras iteraciones del bucle. ¿Cuál es el tiempo total de ejecución del bucle?.*

Pregunta 4: *Supongamos que sustituimos la cola FIFO por 2 colas: una cola para Loads y otra cola para Stores. Proponga un algoritmo de gestión de las colas que posibilite una mejora del rendimiento del procesador. La única restricción que se impone es que las escrituras (Stores) a memoria se efectúen en orden de*

programa. Justifique la respuesta mediante un fragmento de código (con sólo instrucciones Load/store) y el cronograma de ejecución asociado.

Ejercicio 5.19

Sea un procesador segmentado lineal de 8 etapas.



donde

ETAPA	FUNCIONALIDAD
CP	determinar el CP
B1, B2	búsqueda de la instrucción (la instrucción se ha leído al final de B2)
DLA	decodificación, comprobación de riesgos, lectura de datos en registros (segundo semiciclo) y cálculo de la dirección destino de saltos
ALU	operaciones aritméticas, cálculo de la dirección de memoria de datos y evaluación de la condición
M1, M2	acceso a memoria (se lee o escribe al final de M2)
ES	escritura del resultado en registro (durante el primer semiciclo)

El procesador dispone de cortocircuitos ALU/ALU y M/ALU. En caso de riesgo de datos el procesador se bloquea hasta que desaparece el riesgo.

Determine, para las cuatro siguientes preguntas, el número de ciclos por iteración, excepto para la última, al ejecutar el código.

```
1$   r1 ← mem[r0+0]          Load
     r2 ← r2 + r1            Add
     mem[r0+0] ← r2          Store
     r0 ← mem[r0+8]          Load
     si r0 <> 0 entonces ir a 1$   Bne
2$   . . .
```

Suponga que el bucle ejecuta varias iteraciones.

Pregunta 1: *Predicción estática fija No Saltar: al decodificar un salto condicional, el procesador sigue buscando instrucciones del camino secuencial; en caso de predicción errónea, el procesador descarta las instrucciones buscadas e inicia una búsqueda por la rama destino del salto.*

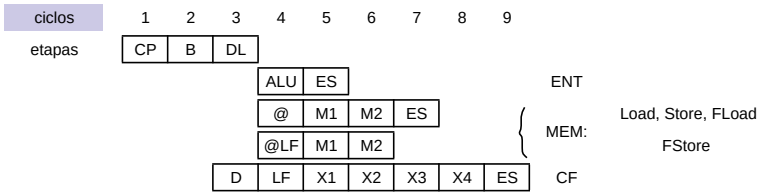
Pregunta 2: *El mecanismo de interpretación de saltos es el mismo que en la 1ª pregunta. El compilador desenrolla 2 iteraciones del cuerpo del bucle. Muestra el nuevo código, donde no es necesario reordenar las instrucciones. Hay que tener en cuenta que el número de iteraciones del bucle no se conoce antes de ejecutarlo.*

Pregunta 3: *El mecanismo de ejecución de saltos es el mismo que en la 1ª pregunta. Analice todas las dependencias del código y reescriba el código original (sin desenrollar) de forma que se optimice el rendimiento.*

Pregunta 4: *Predicción estática fija Saltar: después de calcular la dirección destino del salto (etapa DLA), el procesador inicia la búsqueda a partir del camino no secuencial y descarta las instrucciones del camino secuencial: al verificar la predicción (etapa ALU), si la predicción es errónea se descartan las instrucciones buscadas y se sigue por el camino secuencial.*

Ejercicio 5.20

Un procesador segmentado multiciclo utiliza la siguiente segmentación para las 3 ramificaciones de que dispone: a) aritmética entera (ENT), b) acceso a memoria (MEM) y c) cálculo en coma flotante (CF).



La funcionalidad de las etapas es la siguiente:

ETAPA	FUNCIONALIDAD Y RECURSOS UTILIZADOS
CP	determinar la dirección de la instrucción
B	búsqueda de la instrucción (MI)
DL	decodificación (D), lectura de operandos en registros del banco de enteros (BRent) y cálculo de la dirección destino
D	decodificación (D)
LF	lectura de operandos en registros del banco de coma flotante (BRcf)
@	cálculo de la dirección efectiva (@)
@LF	cálculo de la dirección efectiva y lectura de operandos en registros del banco de coma flotante (@, BRcf)
ALU	operaciones aritmético-lógicas con números enteros y resolución de los saltos (ALU, EV)
M1, M2	acceso a memoria. En M1 se accede al campo etiquetas de la cache (ETIQ) y en M2 al campo datos de la cache (DAT)
X1, ..., X4	cálculo en coma flotante
ES	escritura en el banco de registros de enteros o de coma flotante (BRent, BRcf)

Cada banco de registros tiene 2 caminos de lectura y 2 caminos de escritura. En cada banco de registros se puede escribir y leer, en este orden, un mismo registro en un ciclo de reloj.

El camino de datos sólo dispone de cortocircuitos para reducir la latencia efectiva de escritura al banco de registros y tienen como destino las salidas de la etapa D/L y/o las etapas LF y "@LF" en función del tipo de dato (entero o coma flotante) o de la ramificación destino del dato.

Los riesgos de datos y secuenciamiento se detectan en la etapa DL o D en función de la clase de instrucción. Cuando se detecta un riesgo de datos se retienen las instrucciones en las etapas DL o D, B y CP hasta que desaparece la condición de riesgo. En el caso de riesgo escritura después de escritura el riesgo desaparece en el último ciclo de la instrucción fuente del riesgo. En la etapa D una instrucción se retiene los ciclos indispensables para que pueda obtener el dato mediante un cortocircuito cuyo destino es la etapa LF o "@LF".

Las instrucciones de secuenciamiento actualizan, en la etapa ALU, el registro CP con la dirección de la siguiente instrucción que debe interpretarse y se descartan las instrucciones posteriores al salto.

Como programa de prueba utilizaremos el bucle interno del siguiente código que evalúa el producto de dos matrices.

```
do K = 1, P
    do I = 1, N
        T = A(I, K)
        do J = 1, M
            C(I, J) = C(I, J) + T * B(K, J)
        enddo
    enddo
enddo
```

```
1$:  FLoad F1, 0(r1)      ; bucle interno
     FLoad  F2, 0(r2)
     Fmul F3, F4, F1      ; F4 almacena el contenido de la variable T
     Fadd F5, F3, F2
     FStore F5, 0(r2)
     add r1, r1, #8
     add r2, r2, #8
     sub r3, r3, #1
     bne r3, 1$
```

Pregunta 1: *Muestre en un diagrama temporal la interpretación de una iteración del bucle y la primera instrucción de la siguiente iteración. Indique los ciclos perdidos por riesgos de datos y de secuenciamiento. Así mismo evalúe el CPI.*

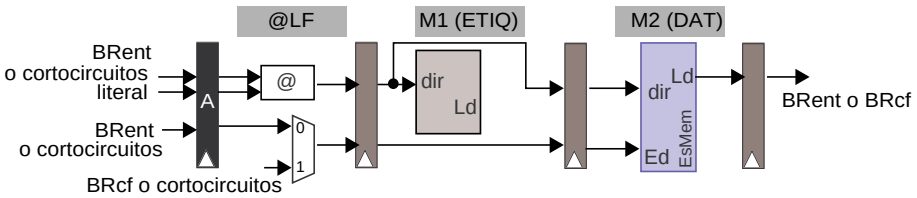
El compilador puede mejorar la planificación de instrucciones para reducir los ciclos perdidos.

Pregunta 2: *¿Cuál es el número de ciclos perdidos por riesgos de datos después de planificar las instrucciones? Justifique la respuesta.*

La instrucción Fstore puede ser una causa de ciclos perdidos. Ahora bien, se ha observado que el dato que debe almacenarse en memoria usualmente no está disponible en el ciclo en que debe iniciarse la ejecución. Sin embargo, el dato utilizado para calcular la dirección está disponible.

La característica descrita y la particularidad de que una instrucción Fstore no es fuente de dependencias debidas a registros puede utilizarse para mejorar las prestaciones del procesador. La idea es seguir interpretando instrucciones que utilicen otra ramificación mientras no se detecten riesgos de datos. Para ello, cuando se detecte un riesgo en una instrucción Fstore, debido al dato que debe almacenarse en memoria, se ocupa la ramificación de memoria hasta que el dato esté disponible, pero no se bloquea la interpretación de instrucciones más jóvenes.

La instrucción Fstore se almacena en el registro de desacoplo de entrada (A) de la ramificación de memoria y permanece en el registro de desacoplo hasta que se puede obtener el dato que debe almacenarse en memoria mediante el cortocircuito disponible en la etapa @LF. Por tanto, el registro de desacoplo actúa como un buffer. En la siguiente figura se muestra el camino de datos simplificado para acceder a memoria.

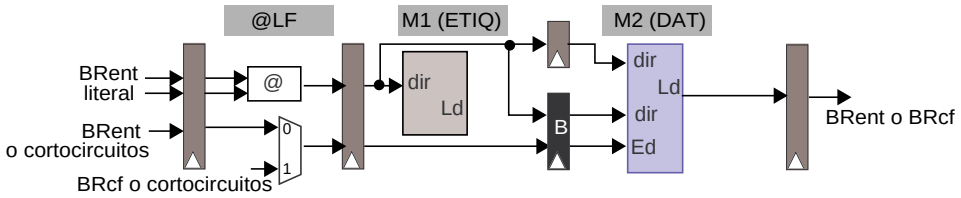


En la siguiente figura se muestra un ejemplo en el cual la instrucción Fstore está esperando en el registro de desacoplo de entrada de la ramificación (A). Esto es, la etapa @LF. Notemos que no pueden iniciarse otras instrucciones de la clase MEM mientras está ocupada la ramificación de memoria.

	ciclos											
instrucción	1	2	3	4	5	6	7	8	9	10	11	12
Fmul **F5**, F6, F7	CP	B	D	LF	X1	X2	X3	X4	ES			
Fstore F5, (r9)		CP	B	DL	@LF	@LF	@LF	@LF	M1	M2		
add **r4**, r4, r5			CP	B	DL	ALU	ES					
xor **r6**, r9, r19				CP	B	DL	ALU	ES				
load **r11**,(r12)					CP	B	DL	DL	@	M1	M2	ES

Pregunta 3: *Utilizando la modificación descrita a nivel de microarquitectura, muestre en un diagrama temporal la interpretación de una iteración del bucle original y la primera instrucción de la siguiente iteración. Indique los ciclos perdidos por riesgos de datos y de secuenciamiento. Así mismo evalúe el CPI.*

Otra mejora a nivel de microarquitectura es permitir que una instrucción Fload o load pueda adelantar a una instrucción Fstore más vieja que está esperando por el dato que se almacena en memoria. Ahora bien, como hay que determinar posibles riesgos de datos debido a memoria, la instrucción Fstore se esperará después de calcular la dirección efectiva. En concreto se esperará en un buffer, denominado buffer de stores (B), ubicado entre la etapa M1 y la etapa M2, el cual se muestra en la siguiente figura.



El buffer de stores (B) dispone de varias entradas y una entrada se asigna en la etapa DL cuando no se detectan riesgos para calcular la dirección efectiva y se libera en la etapa M2 cuando se actualiza el campo DAT. En un ciclo de reloj se puede liberar y asignar, en este orden, una entrada del buffer B.

Supongamos una ráfaga de instrucciones Fstore que tienen disponible en la etapa DL el operando para calcular la dirección, en algún ciclo posterior se captura el dato que debe almacenarse en memoria y no existe ningún ciclo de retardo antes de la etapa M2.

Pregunta 4: *¿Cuántas entradas, en el peor caso, debe tener el buffer de stores para que no se pierdan ciclos al interpretar una ráfaga de instrucciones Fstore?.*

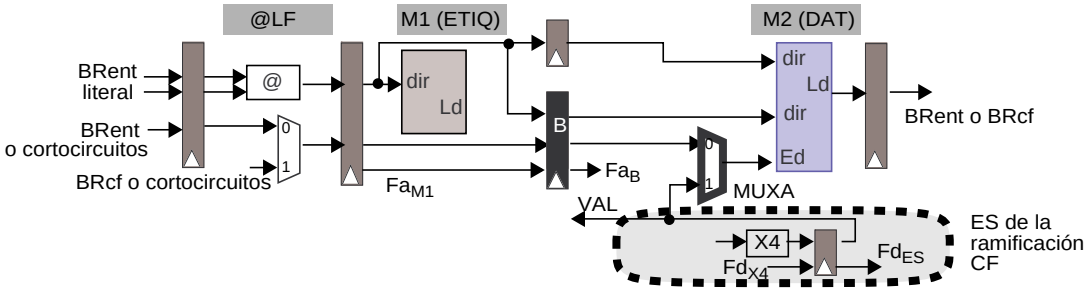
Notemos que después de la etapa M1 existen dos bifurcaciones. La bifurcación superior es utilizada por las instrucciones load y Fload y la bifurcación inferior es utilizada por las instrucciones store y Fstore. Para acceder al campo DAT de la cache se dispone de un camino de lectura y de un camino de escritura. El retardo de actualización del campo DAT requiere todo el ciclo de reloj.

Si una instrucción Fstore en la etapa @LF no ha obtenido el dato que debe almacenar en memoria deberá obtenerlo posteriormente mediante un cortocircuito. Ahora bien, la fuente de este cortocircuito sólo puede ser la etapa de escritura de la ramificación CF. En la siguiente figura se muestra un ejemplo, donde una instrucción load adelanta a una instrucción Fstore y la captura del dato por parte de la instrucción Fstore cuando está en el buffer.

	ciclos								
instrucción	1	2	3	4	5	6	7	8	9
Fmul **F5**, F6, F7	CP	B	D	LF	X1	X2	X3	X4	ES
Fstore F5, 0(r9)		CP	B	DL	@LF	M1	B	B	M2
load **r11**, 0(r12)			CP	B	DL	@	M1	M2	ES

En la siguiente figura se muestra el camino de datos correspondiente a la ramificación de memoria, en la que se puede observar el cortocircuito necesario para que una instrucción Fstore pueda capturar el dato que no está disponible en la etapa @LF (MUXA). Por el camino de datos se transmiten: a) el identificador de registros fuente que almacena el dato que debe almacenarse en memoria (Fa_X), el identificador de registro destino en la ramificación de coma flotante (Fd_X) y el valor que se almacena en el banco

de registros de coma flotante (VAL). También se observan los cables que transmiten el dato que se almacena en el campo DAT (Ed).



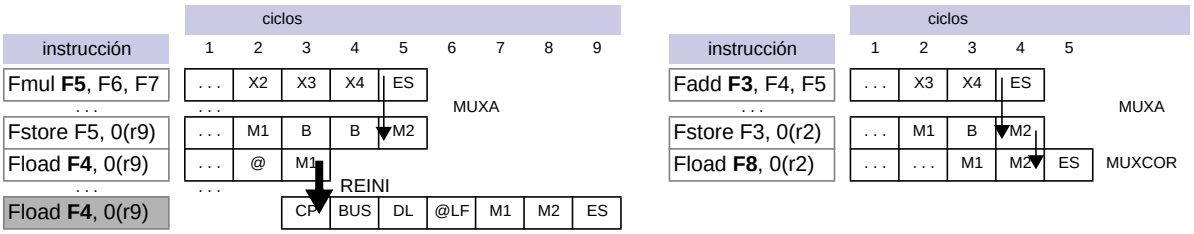
En alguno de los siguientes apartados se solicita diseñar el control de multiplexores. Para ello pueden utilizarse comparadores, puertas lógicas y registros. Para identificar la utilización de una señal en una etapa concreta utilice el acrónimo de la etapa como subíndice. Por otro lado, no se tienen en cuenta las señales de validación.

En las siguientes preguntas supondremos que el buffer de stores (B) sólo tiene una entrada. Por tanto, sólo puede haber una instrucción Fstore en curso de interpretación. Recordemos que al detectar un riesgo escritura después de escritura se suspende la interpretación de la instrucción y siguientes hasta que desaparece el riesgo.

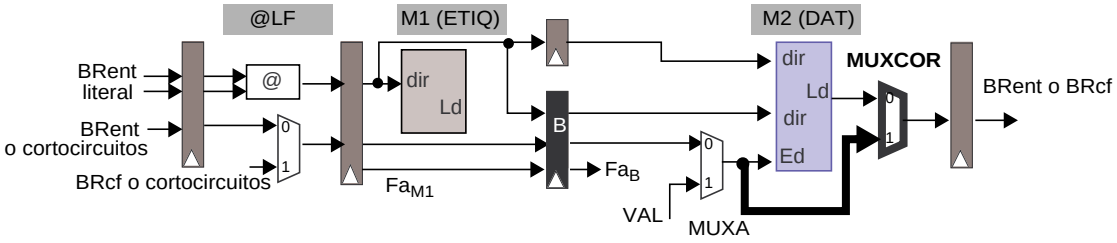
Pregunta 5: *Diseñe el control del multiplexor de cortocircuito MUXA.*

Debido a que una instrucción load o Fload puede adelantar a una instrucción Fstore se pueden producir riesgos de datos debidos a memoria. Cuando se detecta una situación de riesgo se descarta la instrucción load o Fload y todas las instrucciones más jóvenes que se están interpretando. Además, se suministra a la etapa CP la dirección de la instrucción load o Fload. La necesidad de descartar algunas instrucciones determina que la situación de riesgo debe detectarse antes de que instrucciones más jóvenes que el load o Fload que se descarta o anula hayan actualizado el banco de registros o memoria (señal REINI). Por ello, la situación de riesgo hay que detectarla cuando una instrucción load o Fload está en la etapa M1 (parte izquierda de la siguiente figura). Para

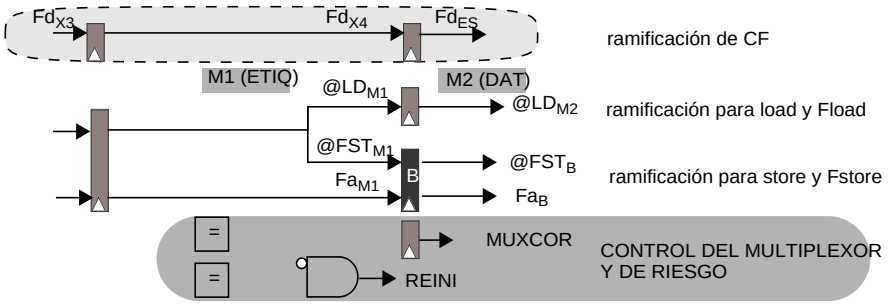
reducir las situaciones de riesgo se puede utilizar el multiplexor MUXCOR (etapa M2). Este multiplexor permite suministrar el dato, que almacena en memoria la instrucción Fstore, a la instrucción load o Fload cuando éste está disponible (parte derecha de la siguiente figura). Recuerde que para acceder al campo DAT de la cache se dispone de un camino de lectura y de un camino de escritura.



En la siguiente figura se muestra el camino de datos correspondiente a la ramificación de memoria, en la que se puede observar el multiplexor MUXCOR en la etapa M2.



Pregunta 6: *Diseñe el control de riesgos de datos debidos a memoria, tanto la señal para controlar el multiplexor MUXCOR como la señal que indica que hay que iniciar la reinterpretación del load o Fload y las siguientes instrucciones (REINI).*



Otra situación que puede producirse es que el bloque referenciado por una instrucción load o Fload deba almacenarse en el mismo contenedor de cache que el bloque referenciado por la instrucción Fstore que está esperando en el buffer de stores (conflicto en cache). Como el Fstore ya ha pasado por la etapa en que se comprueba la etiqueta de cache, si se efectúa el reemplazo inducido por el load o Fload el Fstore actualizaría un bloque incorrecto y además el bloque expulsado no estaría actualizado (escritura retardada). Para resolver esta situación de conflicto se actúa de la misma forma que en el caso de que exista coincidencia entre la dirección de un load o Fload y un Fstore almacenado en el buffer de stores y este último no dispone del dato con el que debe actualizarse memoria.

Suponga una cache de mapeo directo de 32 Kbytes y tamaño de bloque de 32 bytes y que las direcciones que se utilizan para acceder a cache tienen 40 bits.

Pregunta 7: *Diseñe la lógica para detectar una situación de conflicto en cache entre una instrucción Fstore almacenada en el buffer de stores y una instrucción load o Fload más joven. Especifique los bits utilizados de las direcciones.*