Centro Asociado Palma de Mallorca

Ingeniería de Computadores II
Capítulo 2

Tutor: Antonio Rivero Cuesta

Procesadores Superescalares

Características y Arquitectura Genérica de un Procesador Superescalar

Un procesador basado en segmentaciones de un único cauce tiene ciertas limitaciones en su rendimiento.

En ausencia de detenciones *el número máximo de instrucciones que se pueden emitir por ciclo de reloj es uno*.

Una alternativa para mejorar el rendimiento es reducir la duración del ciclo de reloj para aumentar el número de instrucciones ejecutadas por segundo.

Esto nos lleva a reducir el número de operaciones a realizar por el hardware y a aumentar la profundidad de segmentación para realizar todos los pasos que se hace en el procesamiento de una instrucción.

El aumento en el número de etapas nos lleva a un aumento en el número de buffers que las separan y un aumento en las dependencias entre instrucciones.

Todo esto se traduce en:

- Un aumento de los riesgos.
- Un aumento de las detenciones.
- Burbujas en la segmentación.

Resumiendo:

- Tenemos una pérdida mayor de ciclos.
- Un aumento en la frecuenta del reloj encarece el rendimiento global del procesador.
- Otro factor importante es que cuanto más profunda es la segmentación intervienen más transistores, incrementándose la potencia consumida y el calor a disipar.

La principal diferencia de una segmentación superescalar con una clásica es que por las etapas de la segmentación pueden avanzar varias instrucciones simultáneamente.

Esto implica la existencia de varias unidades funcionales para que sea posible.

Otra característica importante es que las instrucciones se pueden ejecutar fuera de orden, además de una segmentación más ancha.

Las segmentaciones superescalares se caracterizan por tres atributos:

- Paralelismo.
- Diversificación.
- Dinamismo.

Paralelismo

Los procesadores segmentados presentan paralelismo de máquina temporal.

Es decir en un mismo instante de tiempo varias instrucciones se encuentran ejecutándose pero en diferentes etapas.

En un procesador superescalar se dan al mismo tiempo el paralelismo de máquina temporal y el espacial.

El ***paralelismo de máquina espacial*** replica el hardware permitiendo que varias instrucciones se procesen simultáneamente en una misma etapa.

El nivel de paralelismo espacial de un procesador se especifica con el término ancho o grado de la segmentación y especifica el número de instrucciones que se pueden procesar simultáneamente en una etapa.

El coste y la complejidad del hardware aumentan.

Hay que aumentar los puertos de lectura/escritura del fichero de registros para que varias instrucciones puedan acceder a sus operandos simultáneamente.

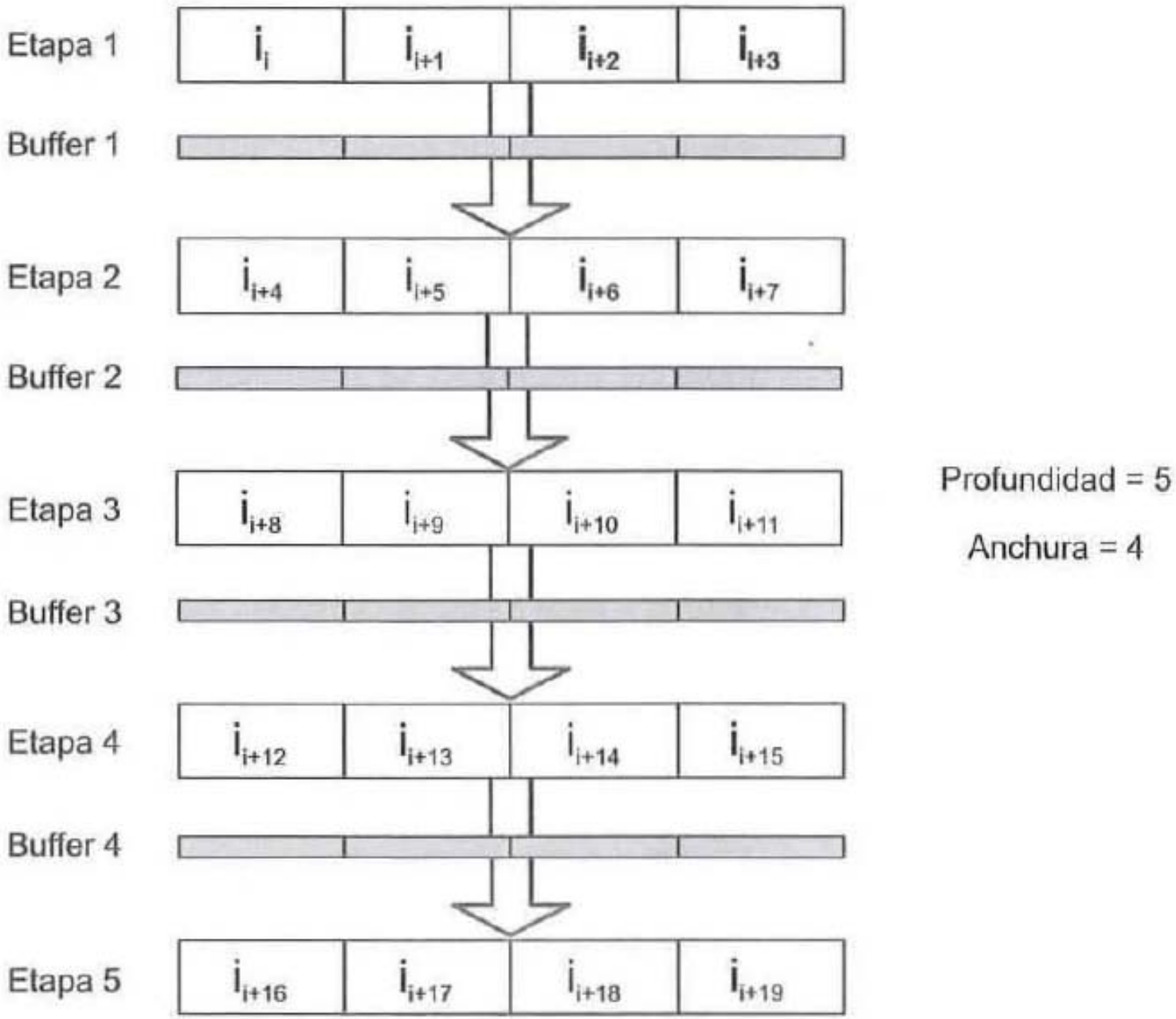
Lo mismo tenemos que hacer con las escrituras/lecturas en las cachés de datos e instrucciones, las unidades aritméticas, los buffers de contención entre etapas, etc..

Aunque un procesador superescalar es una réplica de una segmentación escalar rígida, no resulta sencillo replicar una segmentación escalar ya que tenemos que tener en cuenta varias complicaciones.

Hay que considerar la posibilidad de ejecutar instrucciones fuera de orden y el problema de la dependencia de datos y de memoria.

Los buffers entre etapas pasan a ser buffers complejos multientrada, que permiten pasar a unas instrucciones y a otras no.

Otro elemento es incluir una red compleja de interconexión entre las entradas y las salidas de las unidades funcionales que permitan la resolución de la dependencia de datos.


Etapa 1
i_i
i_{i+1}
i_{i+2}
i_{i+3}
Buffer 1
Etapa 2
i_{i+4}
i_{i+5}
i_{i+6}
i_{i+7}
Buffer 2
Etapa 3
i_{i+8}
i_{i+9}
i_{i+10}
i_{i+11}
Buffer 3
Etapa 4
i_{i+12}
i_{i+13}
i_{i+14}
i_{i+15}
Buffer 4
Etapa 5
i_{i+16}
i_{i+17}
i_{i+18}
i_{i+19}
Profundidad = 5
Anchura = 4

Diversificación

Debido a que los diferentes tipos de instrucción que constituyen un repertorio de instrucciones implican diferentes operaciones, disponer de un único tipo de cauce en la segmentación es ineficiente.

Incluso aunque se trate de una segmentación con varios cauces en paralelo.

Por este motivo los procesadores superescalares incluyen en la etapa de ejecución múltiples unidades funcionales diferentes e independientes.

Siendo habitual la existencia de varias unidades del mismo tipo.

Dinamismo

Las segmentaciones superescalares se etiquetan como dinámicas al permitir la ejecución de instrucciones fuera de orden, respetándose la semántica del código fuente.

Cuenta con los mecanismos necesarios para garantizar que se obtengan los mismos resultados, es decir que se respeta la semántica del código fuente.

Una segmentación dinámica paralela utiliza buffers multientrada que permiten que las instrucciones entren y salgan de los buffers fuera de orden.

La ventaja que aporta la ejecución fuera de orden es que intenta aprovechar al máximo el paralelismo que permiten las instrucciones y el que permite el hardware al disponer de múltiples unidades funcionales.

Esto se traduce en:

- Unas instrucciones pueden adelantar a otras si no hay dependencias falsas, evitando ciclos de detención innecesarios.

- Una reducción de ciclos de detención por dependencias verdaderas de datos y memoria.

Los procesadores superescalares tienen la capacidad para especular.

Pueden realizar predicciones sobre las instrucciones que se ejecutarán tras una instrucción de salto.

Arquitectura de un Procesador Superescalar Genérico

El modelo de segmentación superescalar genérica consta de seis etapas:

- Etapa de lectura de las instrucciones IF (Instruction Fetch).
- Etapa de decodificación ID (Intruction Decoding).
- Etapa de distribución/emisión II (Instruction Issue)
- Etapa de ejecución EX (Execution).
- Etapa de terminación WR (Write-Back Results).
- Etapa de retirada RI (Retirement Instruction).

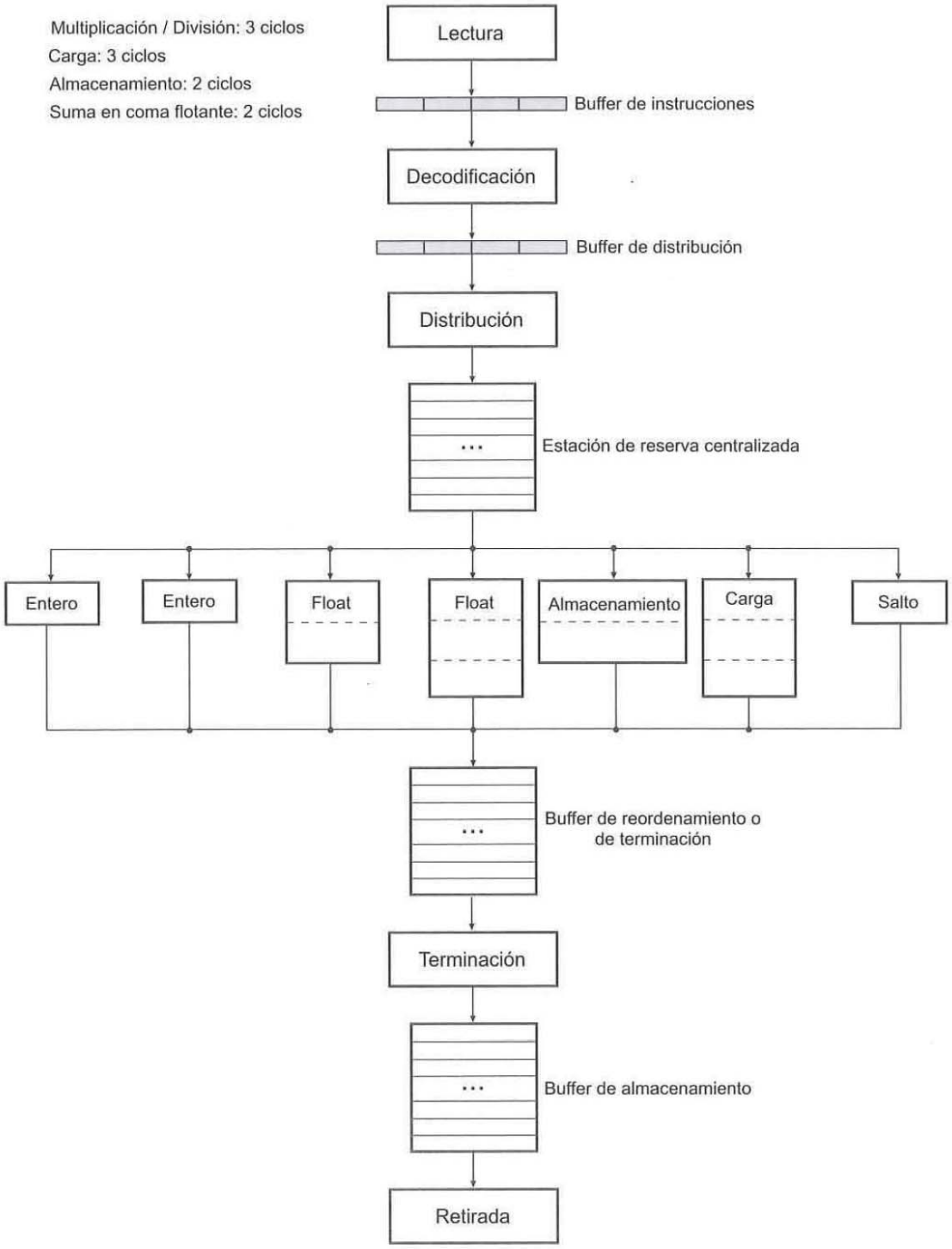


Figura 2.3: Ejemplo de segmentación superescalar genérica.

En una segmentación ***superescalar*** una instrucción ha sido:

- Distribuida.
- Emitida.
- Finalizada.
- Terminada.
- Retirada.

Distribuida (dispatched)

Cuando ha sido enviada a una estación de reserva asociada a una o varias unidades funcionales del mismo tipo.

Emitida (issued)

Sale de la estación de reserva hacia una unidad funcional.

Finalizada (finished)

Cuando abandona la unidad funcional y pasa al buffer de reordenamiento, los resultados se encuentran temporalmente en registros no accesibles al programador.

Terminada (completed)

O terminada arquitectónicamente, cuando ha realizado la escritura de los resultados desde los registros de renombramiento a los registros arquitectónicos, ya son visibles al programador.

Se realiza la actualización del estado del procesador.

Retirada (retired)

Cuando ha realizado la escritura en memoria, si no se necesita escribir en memoria la finalización de una instrucción coincide con su retirada.

Las ***etapas lógicas*** permiten agrupar un conjunto de operaciones que realizan una tarea completa, como por ejemplo la lectura de instrucciones.

En las implementaciones reales las etapas lógicas se encuentran segmentadas, es decir por varias etapas físicas donde cada segmento consume un ciclo de reloj.

Las dependencias de datos falsas, WAR y WAW, se resuelven en los procesadores superescalares recurriendo a un almacenamiento temporal en el que se realiza la escritura que tiene riesgo.

Para esto utilizamos una técnica que se llama ***renombramiento dinámico de registros*** que consiste en utilizar un conjunto de registros ocultos al programador en los que se realizan los almacenamientos temporales.

La técnica consta de dos pasos:

- ***Resolución de riesgos WAW y WAR***: se renombran de forma única los registros arquitectónicos que son objeto de una escritura, se resuelven las dependencias ya que manejan registros diferentes.
- ***Mantenimiento de las dependencias RAW***: se renombran los registros arquitectónicos fuente que corresponden a una escritura previa, el objetivo es mantener las dependencias RAW.

Una vez resueltas las dependencias en la etapa de distribución y las instrucciones se han procesado fuera de orden en la etapa de ejecución, se procede en la fase de terminación a deshacer el renombramiento y a actualizar ordenadamente los registros arquitectónicos.

Respecto a las dependencias entre instrucciones de carga/almacenamiento se presentan las mismas situaciones que con las instrucciones que operan con registros, suponiendo común la posición de memoria tenemos:

- Una carga seguida de un almacenamiento produce una dependencia RAW.
- Un almacenamiento seguida de una carga produce una dependencia WAR.
- Dos almacenamientos seguidos implican una dependencia WAW.

Una solución para respetar las dependencias que puedan existir entre las instrucciones de carga y almacenamiento es su ejecución ordenada, pero no es eficiente.

Si realizamos el renombramiento y la terminación ordenada de los almacenamientos se resuelven las dependencias WAR y WAW, pero no las RAW.

Igual que las operaciones con registros para resolver las dependencias RAW usamos adelantamiento de los operandos.

Otra forma para mejorar el rendimiento es adelantar la ejecución de las cargas.

Que se adelanten resultados o se renombren los registros para evitar dependencias WAR y WAW y aumentar así el rendimiento de las unidades funcionales no garantiza la consistencia semántica del procesador y la memoria.

Para lograr la consistencia una vez deshecho el renombramiento hay que realizar la escritura ordenada de los registros arquitectónicos en la etapa de terminación y de las posiciones de memoria en la etapa de retirada.

Tenemos que tener en cuenta que solo es posible terminar aquellas instrucciones que no sean el resultado de especular con una instrucción de salto.

El buffer de reordenamiento o terminación se convierte en la pieza fundamental para conseguir esta consistencia del procesador, ya que es el sitio en donde se realiza el seguimiento de una instrucción desde que se distribuye hasta que se termina.

Lectura de Instrucciones

La diferencia entre un procesador superescalar y uno escalar en la etapa de lectura es el número de instrucciones que se extraen de la caché de instrucciones en cada ciclo.

En un procesador escalar es una.

En un procesador superescalar está determinado por el ancho de la etapa de lectura.

La caché de instrucciones tiene que proporcionar en un único acceso tantas instrucciones como el ancho de la etapa de lectura.

En los acceso a la caché pueden aparecer fallos de lectura deteniendo el suministro de instrucciones

Los procesadores tienen mecanismos auxiliares de almacenamiento, llamados buffers de instrucciones, entre la etapa de lectura y decodificación.

El objetivo es que aunque la etapa de lectura no suministre instrucciones durante uno o varios ciclos, la etapa de decodificación pueda continuar extrayendo instrucciones del buffer y continuar la tarea sin detenerse.

Los procesadores superescalares cuentan con mayor ancho de banda para la lectura de instrucciones que para la decodificación.

Al conjunto de instrucciones que se extrae simultáneamente de la I-caché se le denomina ***grupo de lectura***.

Aunque el ancho de la memoria caché de instrucciones sea suficiente pueden surgir dos problemas:

- La falta de alineamiento de los grupos de lectura.
- El cambio en el flujo de instrucciones debido a las instrucciones de salto.

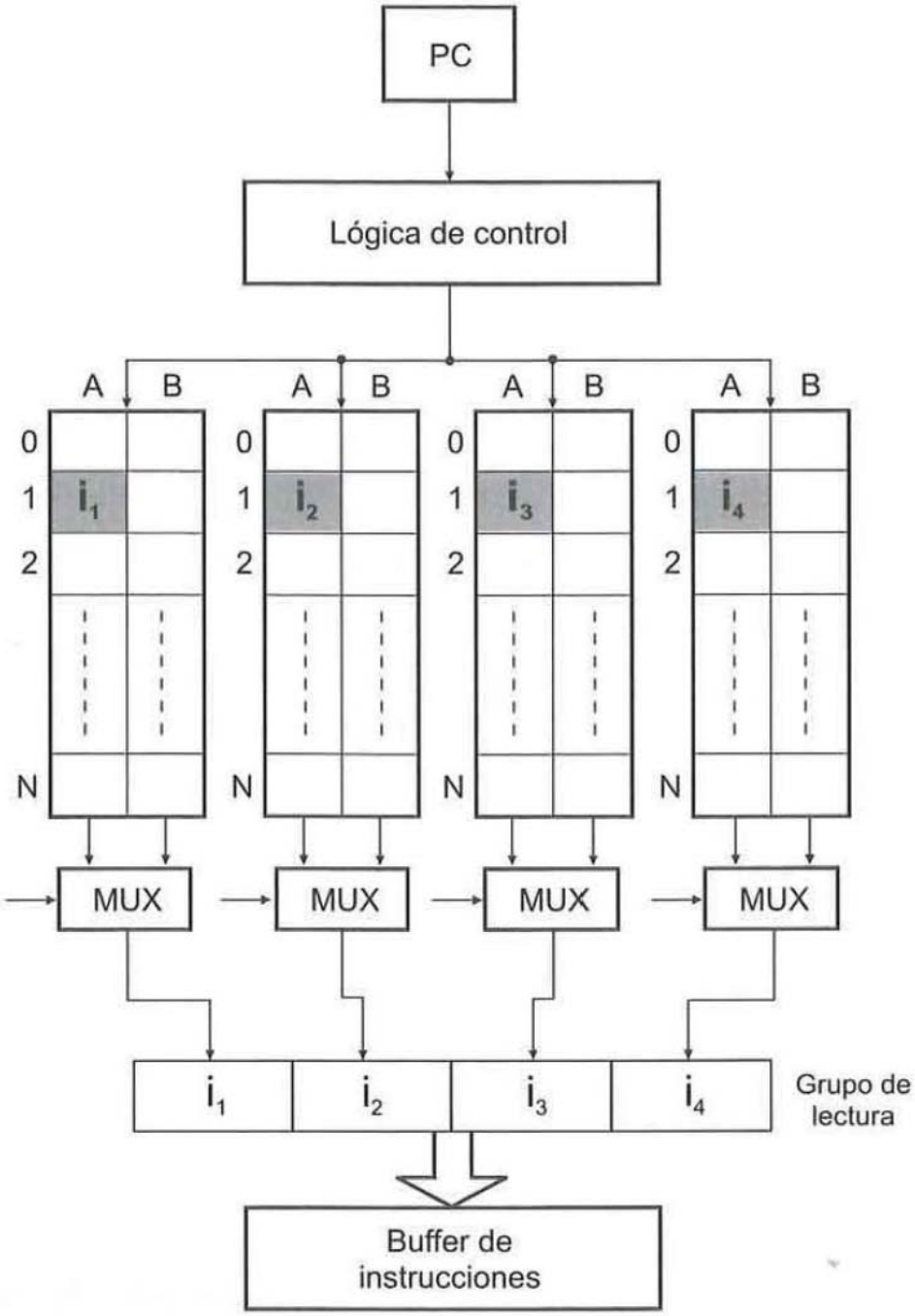


Figura 2.8: Organización de memoria caché en la que el tamaño del bloque físico corresponde al ancho de la etapa de decodificación.

Falta de Alineamiento.

Un alineamiento incorrecto de un grupo de lectura implica que las instrucciones que forman el grupo superan la frontera que delimita la unidad de lectura de una caché.

Implica realizar dos accesos a la caché ya que el grupo de lectura ocupa dos bloques consecutivos.

Nos reduce el ancho de banda como mínimo a la mitad al ser necesarios dos ciclos de reloj para suministrar las instrucciones que forman el grupo de lectura y además.

Puede provocar un fallo de lectura si la segunda parte del grupo de lectura pertenece a un bloque que no está en la caché.

Las máquinas se diseñan con ciertas restricciones de alineamiento para evitar problemas de lectura.

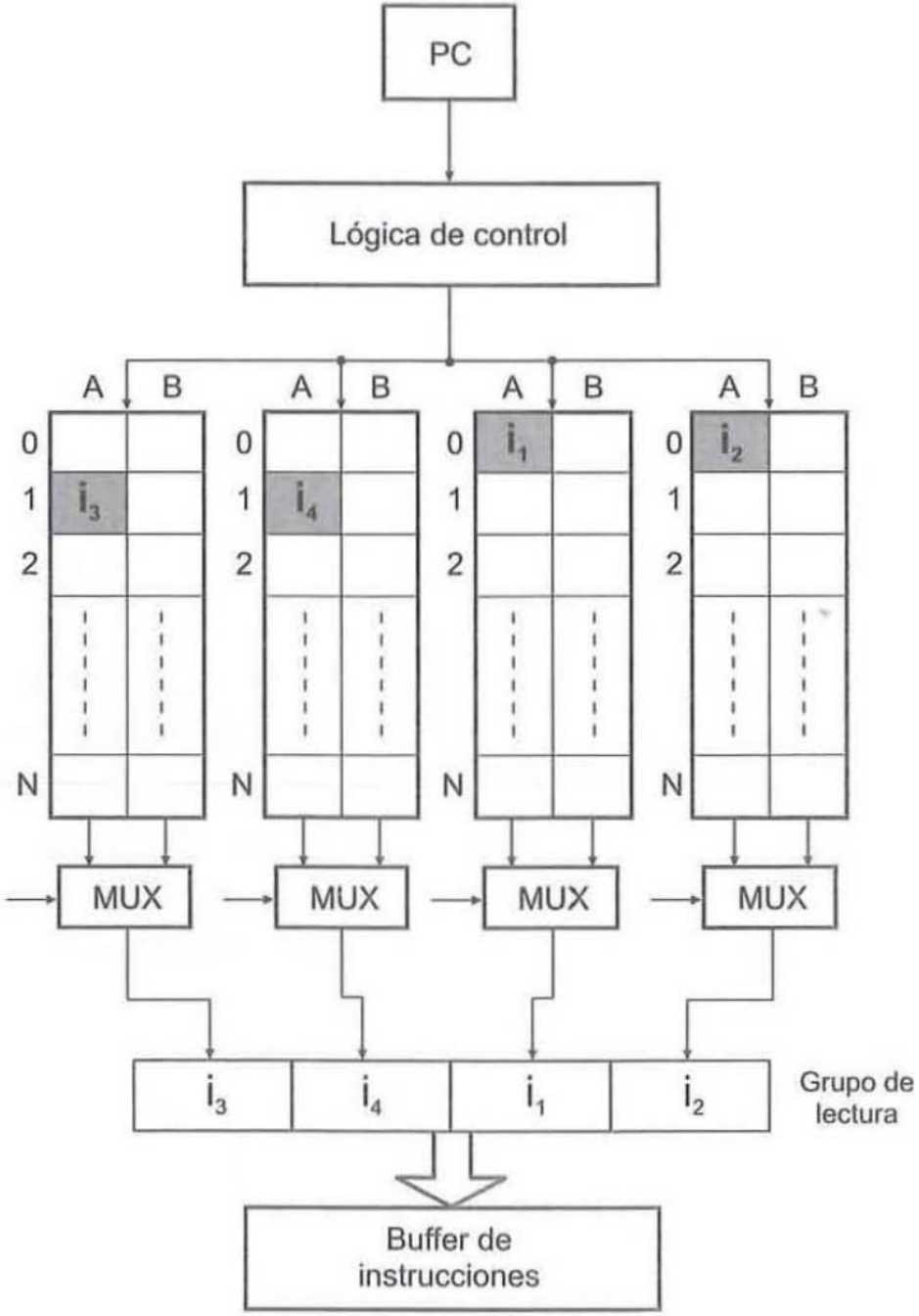


Figura 2.9: Grupo de lectura de cuatro instrucciones desalineado con respecto a la unidad de lectura, provocando la necesidad de dos accesos

Si no hay restricciones de alineación podemos recurrir a utilizar un hardware adicional para realizar la extracción de la unidad de lectura con las instrucciones desordenadas y proceder a la colocación correcta de las instrucciones en el grupo de lectura mediante una *red de alineamiento* o una *red de desplazamiento*.

La ***red de alineamiento*** consiste en reubicar las salidas de la caché mediante multiplexores que conducen las instrucciones leídas a su posición correcta dentro del grupo de lectura.

La ***red de desplazamiento*** recurre a registros de desplazamiento para mover las instrucciones.

El ***prefetching* o *prelectura*** es una técnica utilizada para minimizar el impacto de los fallos de lectura en la caché de instrucciones.

Consiste en disponer de una cola de *prefetch* de instrucciones que usamos cuando hay un fallo de lectura en la caché de instrucciones, si las instrucciones se encuentran en la cola de *prefetch*,

estas se introducen en la segmentación igual que si hubiesen sido extraídas de la I-caché.

Y al mismo tiempo el sector crítico que ha producido el fallo de lectura se trae y se escribe en la I-caché para evitar fallos futuros, en el caso que el grupo de instrucciones no esté tampoco en la cola de prefetch se lanza el fallo al siguiente nivel de la caché, el L2, y se procesa con la máxima prioridad.

Rotura de la Secuencialidad

Puede suceder que una de las instrucciones que forme parte de un grupo sea un salto incondicional o un salto condicional efectivo.

Esto provoca que las instrucciones posteriores del grupo sean inválidas.

Se reduce el ancho de banda de lectura, además de pagar un elevado coste en ciclos de reloj desperdiciados.

En una segmentación de ancho *s*, cada ciclo de parada equivale a no emitir *s* instrucciones o a leer *s* instrucciones NOP.

El ***coste mínimo de la oportunidad perdida*** y se expresa como el producto entre el ancho de la segmentación y el número de ciclos de penalización.

El ***coste mínimo de la oportunidad perdida*** es la cantidad mínima de ciclos que se desperdician como consecuencia de una rotura en la secuencia del código ejecutado y que obliga a anular el procesamiento de las instrucciones que hay en la segmentación, se expresa en ciclos de reloj.

Cuando la instrucción es un ***salto incondicional*** hay que insertar en el PC la dirección de salto dejando inservible el procesamiento de las instrucciones posteriores al salto que ya están en alguna etapa de la segmentación.

Si la instrucción es un ***salto condicional*** hay que esperar a conocer si el salto es efectivo o no y en caso afirmativo calcular la dirección de salto, proceder a su inserción en el PC y anular las instrucciones posteriores que estuviesen en el cauce.

La técnica de salto retardado que se aplica en los procesadores escalares para rellenar huecos en la segmentación no es válida en un procesador superescalar, por la capacidad que tienen los procesadores superescalares para ejecutar instrucciones fuera de orden cuando no existan dependencias que lo impidan.

La solución consiste en la detección temprana de las instrucciones que cambian el flujo de instrucciones para poder aplicar alguna técnica destinada a minimizar el impacto de este tipo de instrucciones.

Una técnica es efectuar una detección integrada con la extracción del grupo de lectura.

Consiste en comenzar el análisis antes de que se extraiga el grupo de lectura de la I-caché.

Para ello se dispone de una pre-etapa de decodificación situada entre la I-caché y el siguiente nivel de memoria, el L2 de la caché.

En esta pre-etapa se analiza la instrucción y como resultado se le concatenan unos bits con diversas indicaciones.

Tratamiento de los Saltos.

El ***principal problema*** que plantea el tratamiento de los saltos no es descubrir que se trata de una instrucción de salto, sino los ciclos que hay que esperar para conocer el resultado de la evaluación de la condición que determina si el salto es efectivo o no y la dirección de destino, en caso de que sea efectivo.

El ***procesamiento de los saltos*** se realiza en una unidad funcional específica, ya que detener todo el cauce hasta saber cuál es la instrucción implica desperdiciar muchos ciclos.

Lo habitual es que el ***tratamiento de los saltos*** se inicie en la etapa IF durante la lectura de la I-caché, cuando las instrucciones han sido extraídas y se confirma en la etapa ID que la instrucción es un salto se puede mantener el tratamiento iniciado o anularlo.

La técnica que se utiliza en los procesadores superescalares para tratar las instrucciones de salto es especular, es decir se realizan predicciones.

Al mismo tiempo que se están leyendo las instrucciones de la I-caché se comienza a predecir su

efectividad y su dirección de destino, sin saber si se trata de un salto, cuestión que se confirma en la etapa ID.

La ***especulación*** también puede realizarse con los saltos incondicionales o bifurcaciones, en este caso la especulación siempre acaba coincidiendo con el resultado del salto.

Estrategias de Predicción Dinámica

Las técnicas que se emplean para predecir el comportamiento de las instrucciones de salto se dividen en dos grandes grupos:

- Estáticas.
- Dinámicas.

Las ***técnicas dinámicas*** se basan en que un salto es una instrucción cuya ejecución se repite con cierta frecuencia, lo que nos permite establecer un patrón basado en su comportamiento.

Las ***técnicas de predicción estática*** logran tasas de predicción de saltos condicionales entre un 70% y 80%.

Las ***técnicas de predicción dinámica*** obtienen tasas de acierto que oscilan entre un 80% y 95%, el inconveniente que tienen estas técnicas es el incremento del coste económico del procesador.

Para realizar una ***especulación completa*** de una instrucción de salto es necesario predecir los dos componentes que producen su procesamiento, ***la dirección de destino*** **y** ***el resultado***, *efectivo o no efectivo*.

Predicción de la dirección de destino de salto mediante BTAC

Para reducir al máximo la penalización por salto mediante la ejecución especulativa de instrucciones debemos saber cuanto antes:

- Si una instrucción es un salto efectivo.
- Su dirección de destino más probable.

Una técnica sencilla para predecir la dirección de destino es recurrir a una BTAC (Brach Target Address Cache), que es una pequeña ***memoria caché asociativa*** en la que se almacenan:

- Las ***direcciones de las instrucciones de saltos efectivos ya ejecutados*** o BIAs (***Brach Instruccion Addres***).

- Las ***direcciones de destino de esos saltos*** o BTAs (***Brach Target Address***).

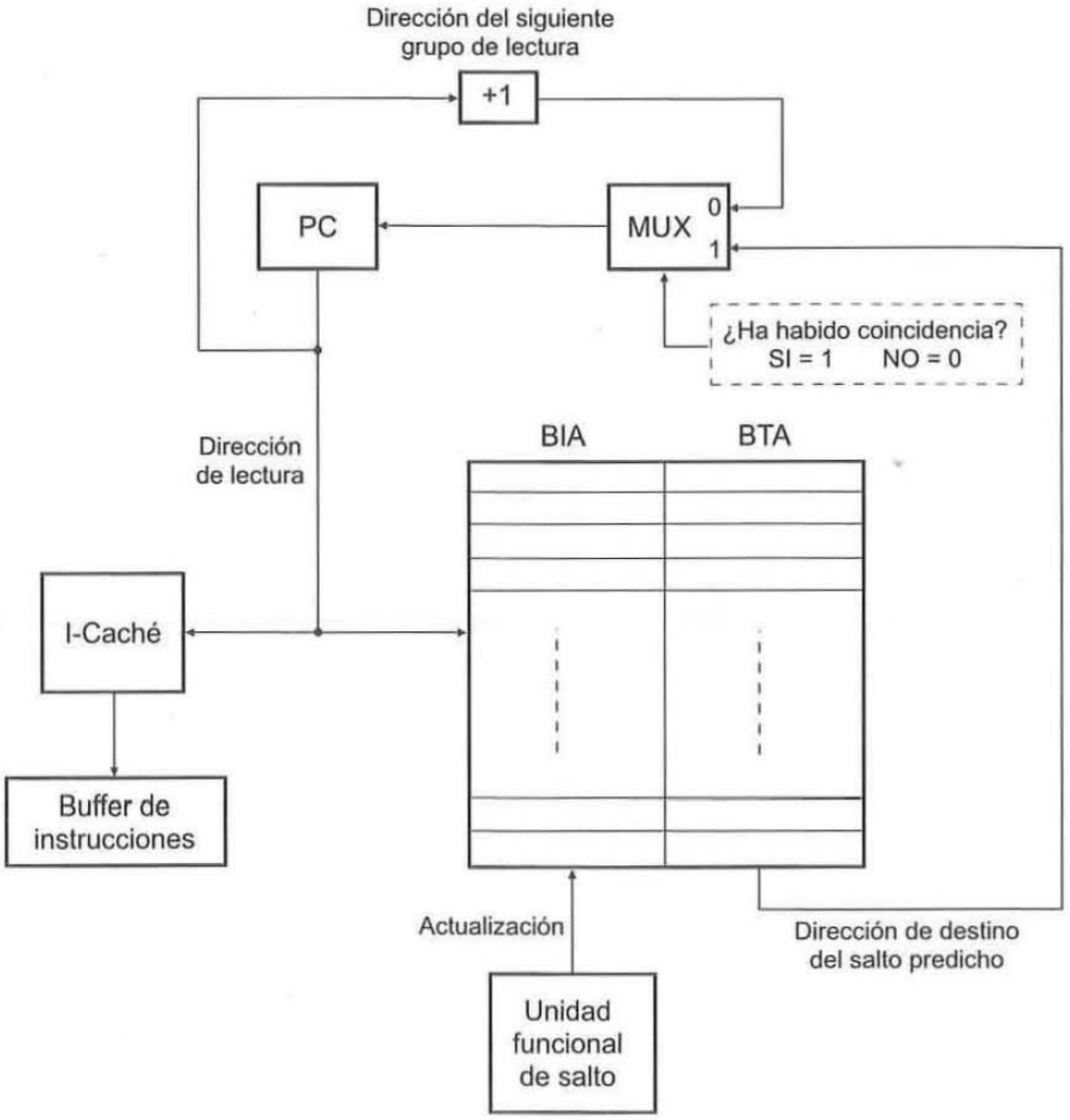
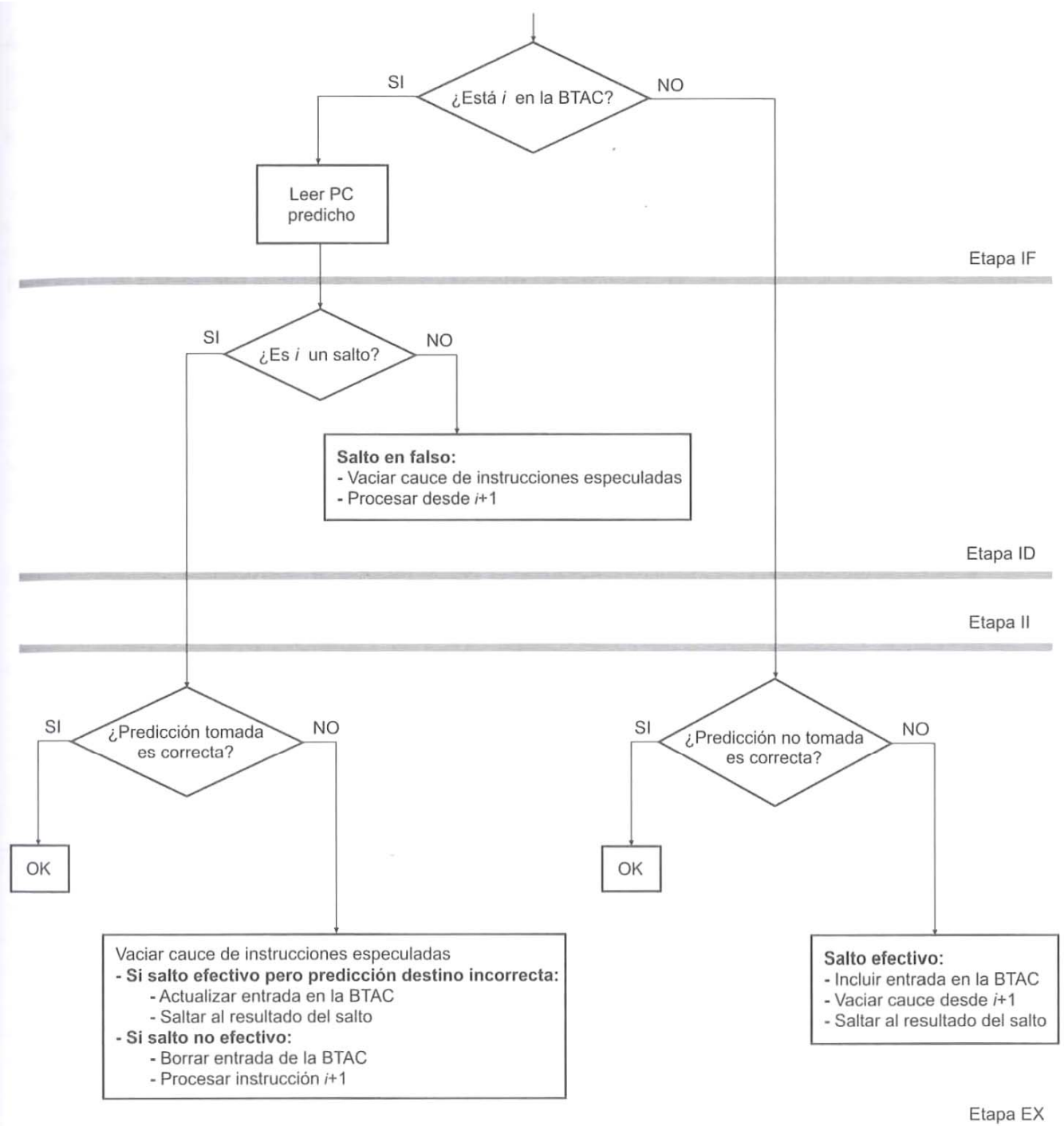


Figura 2.12: Esquema de una BTAC


¿Está i en la BTAC?
SI
NO
Leer PC predicho
Etapa IF
¿Es i un salto?
SI
NO
Salto en falso:
- Vaciar cauce de instrucciones especuladas
- Procesar desde i+1
Etapa ID
Etapa II
¿Predicción tomada es correcta?
SI
NO
OK
¿Predicción no tomada es correcta?
SI
NO
OK
Vaciar cauce de instrucciones especuladas
- Si salto efectivo pero predicción destino incorrecta:
- Actualizar entrada en la BTAC
- Saltar al resultado del salto
- Si salto no efectivo:
- Borrar entrada de la BTAC
- Procesar instrucción i+1
Salto efectivo:
- Incluir entrada en la BTAC
- Vaciar cauce desde i+1
- Saltar al resultado del salto
Etapa EX

El acceso a la BTAC se realiza en paralelo con el acceso a la I-caché utilizando el valor del PC.

Si ninguna de las direcciones que forman el grupo de lectura coincide con alguna de las BIAs que hay en la BTAC es debido a que no hay ninguna instrucción de salto efectivo o se trata de un salto efectivo que nunca antes había sido ejecutado.

Si la BTAC no devuelve ningún valor, por defecto se aplica la técnica de ***predecir como no efectivo***.

Se procede normalmente con la ejecución de las instrucciones que siguen al salto y se deshace su procesamiento en caso de que el salto sea finalmente efectivo.

En este caso las penalizaciones serían:

- 0 ciclos de detención si el salto finalmente no es efectivo, al aplicarse la predicción ***como no efectivo*** el procesamiento se realiza normalmente.
- Ciclos_ID + Ciclos_II + Ciclos_EX_salto de detención si el salto es efectivo suponiendo que el salto se resuelve en la etapa EX y el vaciado del cauce de las instrucciones que seguían al salto. A continuación se actualiza la BTAC con la dirección de la instrucción de salto y el destino real del salto.

Si como predicción por defecto se opta por la técnica ***predecir como efectivo***, no se obtiene ninguna mejora, ya que siempre tenemos que esperar a la resolución real del salto para poder efectuarlo.

Si la dirección de la instrucción que se busca en la BTAC coincide con una de las BIAs se trata de un salto efectivo que ha sido ejecutado con anterioridad y se procede a extraer el valor de BTA que tenga asociado, es decir la dirección de destino asociada a la última vez que fue ejecutada esa instrucción de salto.

Una vez que tenemos la predicción de la dirección de destino lo usamos como nuevo contador de programa.

Se leerá de la I-caché y comenzará la ejecución especulativa de esa instrucción y de las siguientes.

Al mismo tiempo la instrucción de salto debe finalizar su ejecución con el fin de validar que la especulación es correcta.

Se pueden dar dos situaciones al conocer el resultado del salto:

- La ***predicción realizada es incorrecta*** debido a que se trata de un salto no efectivo o a que la dirección de destino especulada no coincidía con la real.
- Si la ***predicción es correcta*** se continúa con el procesamiento de instrucciones, no es necesario modificar la tabla.

Puede surgir el problema de los ***falsos positivos***, la BTAC devuelve una dirección de destino pero en el grupo de lectura no hay realmente ningún salto, esto se descubre en la etapa ID.

Esto es lo que se conoce como ***saltos fantasmas*** *o* ***saltos en falso***, el tratamiento correcto es desestimar la predicción de la BTAC una vez que se confirma que se trata de un falso positivo.

Estas instrucciones que el ***predictor de destinos*** identifica inicialmente como saltos producen una

penalización al tener que expulsar del cauce la instrucción de destino especulada.

Los ***falsos positivos*** son debidos a que el campo BIA de la BTAC no almacena una dirección completa, sólo una parte. Esto provoca que a direcciones distintas se le asocie la misma BIA.

Los ***saltos fantasma*** se minimizan aumentando el número de bits que componen el campo BIA de forma que haya mayor coincidencia con la dirección de la instrucción que se busca en la tabla.

Si la longitud de la BIA llega a igualarse a la longitud de las direcciones, el problema de los falsos positivos deja de existir.

Otra técnica para predecir la dirección de salto es utilizar una **BTIB** (***Brach Target Intruction Buffer***), su estructura es similar a la de la BTAC pero se almacena la instrucción de destino **BTI** (***Brach Target Intruction***) y algunas posteriores en lugar de solo la dirección de salto efectivo BTA.

La BTIB entrega una secuencia de instrucciones al buffer de instrucciones, como si se hubiese extraído de la I-caché y en el PC se coloca la dirección correspondiente a la siguiente instrucción que hay después de la última BTI que forma el paquete.

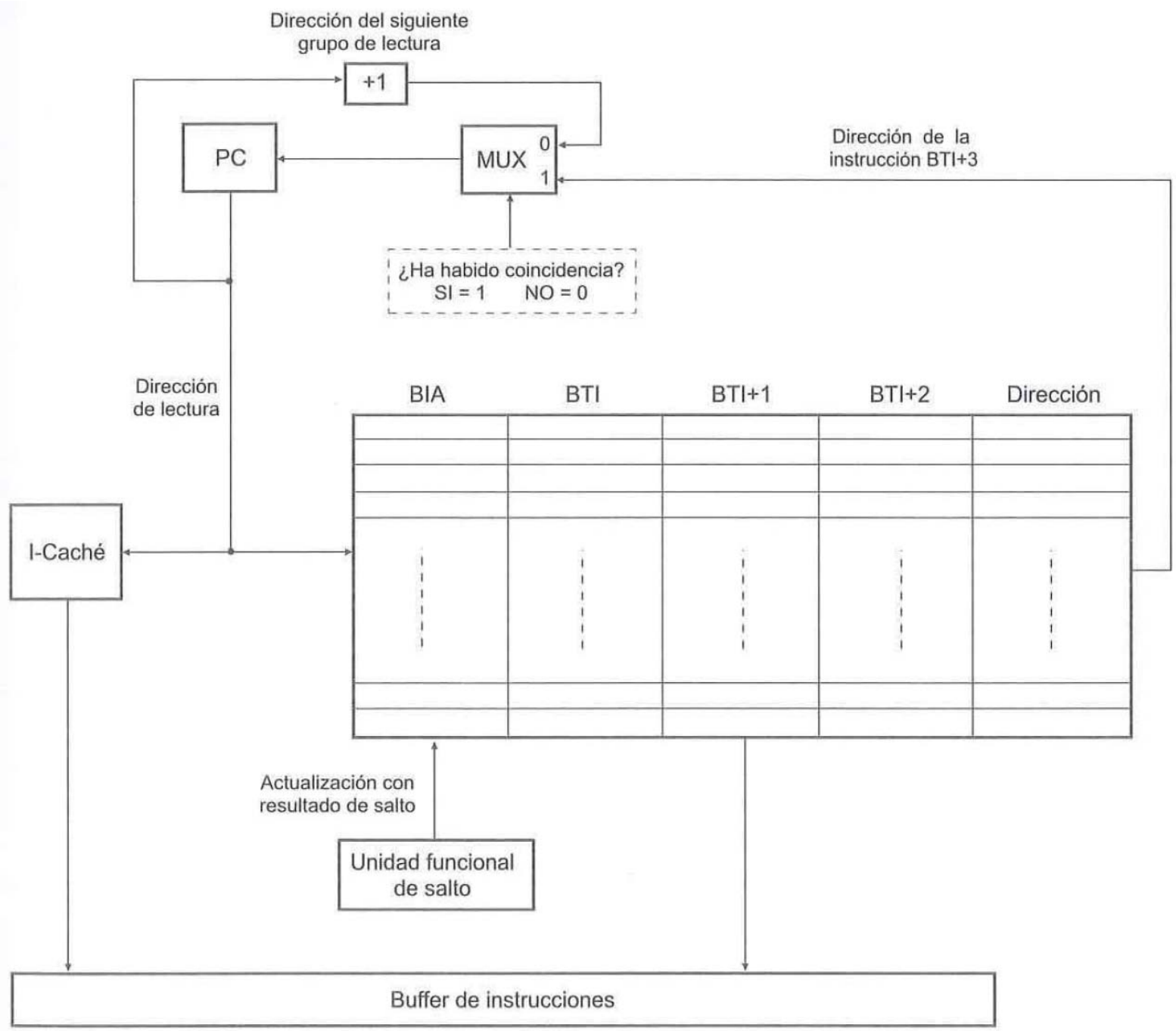


Figura 2.14: Esquema BTIB

Predicción de Destino de Salto Mediante BTB con Historial de Salto

La predicción mediante **BTB** (***Brach Target Buffer***), es una técnica de predicción dinámica similar a la BTAC pero la diferencia es que junto con la dirección de destino predicha, la BTA almacena un conjunto de bits que representan el historial del salto y predicen la efectividad del salto, BH (***Branch History***).

Al mismo tiempo que se extrae el grupo de lectura de la I-caché, se accede a la BTB en busca de alguna de las instrucciones del grupo de lectura.

Si hay un acierto se analizan los bits del historial de salto y se decide si la dirección de destino predicha ha de ser utilizada o no.

La instrucción de salto se continúa procesando para validar el resultado de la especulación.

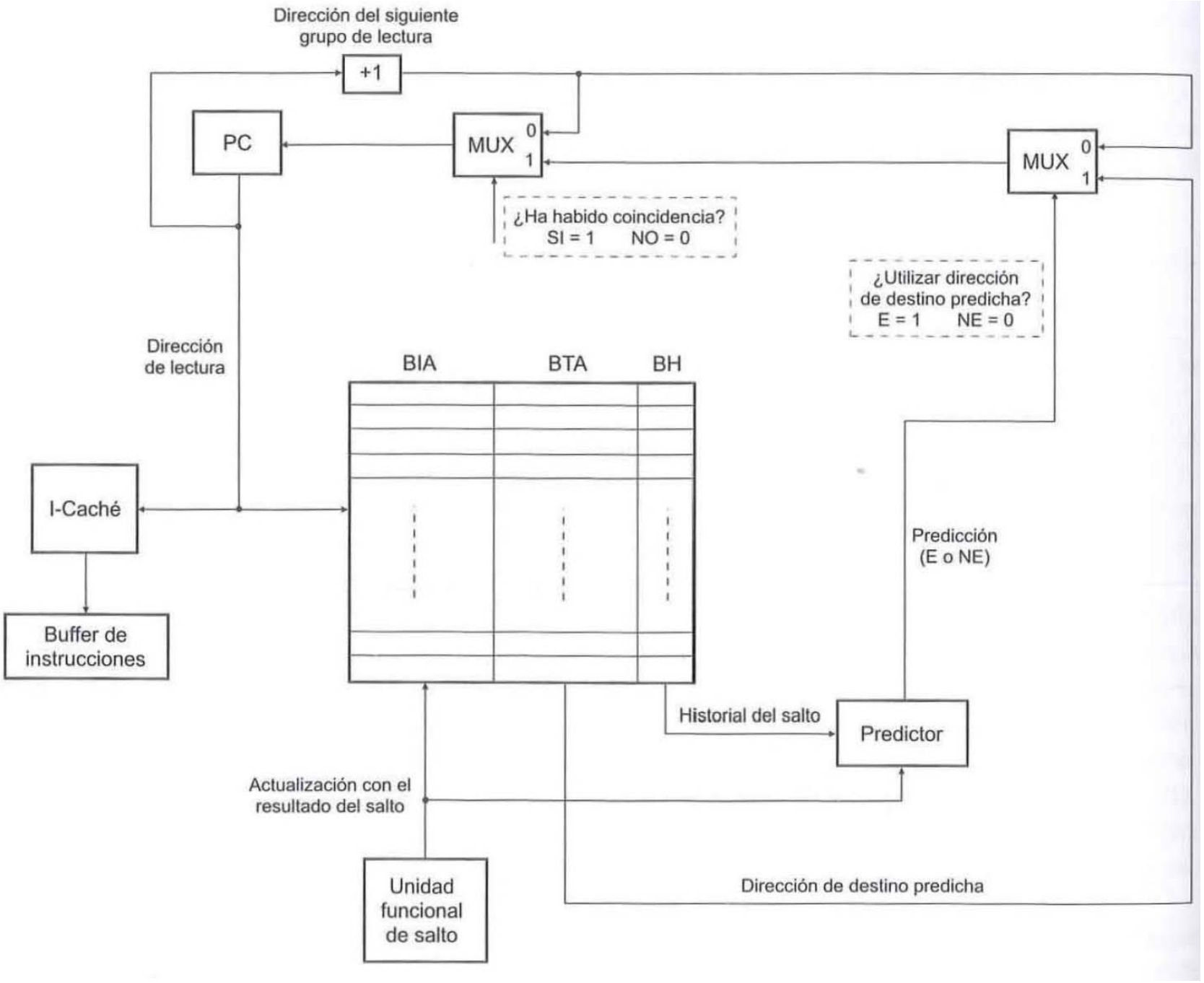


Figura 2.15: Esquema BTB con Historial de Salto

Se pueden dar ***cuatro situaciones***:

- *Se predice como efectivo y no hay error en la predicción* del resultado ni en la dirección de destino, no hay penalización y se actualiza el historial de salto.
- *Se predice como efectivo pero hay algún error en la predicción*, en el resultado, en la dirección o ambos. Se vacía el cauce de las instrucciones especuladas, se actualiza el historial de salto, se actualiza la entrada de la BTB con la dirección

de destino y se comienza a procesar la instrucción indicada por el resultado del salto.

- *Se predice como no efectivo y el salto no lo es*, no hay penalización y se actualiza el historial de salto.
- *Se predice como no efectivo y el salto sí lo es*, se vacía el cauce con las instrucciones especuladas, se salta a la dirección de destino obtenida, se actualiza el historial de salto y se actualiza la entrada de la BTB.

Si ninguna de las instrucciones del grupo de lectura proporciona una coincidencia en la BTB se predice como no efectivo y se pueden plantear dos situaciones:

- El ***salto no es efectivo***: no pasa nada en el cauce.
- El ***salto es efectivo***: se vacía el cauce con las instrucciones especuladas y se salta a la dirección de destino obtenida y se incluye una entrada en la BTB con la dirección de la instrucción de salto, dirección de destino e historial de salto.

Si la instrucción no es un salto no ocurre nada.

Y por último nos falta resolver la eliminación de las entradas en la tabla debido a fallos de capacidad o de conflicto, dependiendo de la organización de la tabla:

- ***Correspondencia directa***, se sustituye la entrada.
- ***Asociatividad***, descartamos la entrada que sea menos eficiente para mejorar el rendimiento. Podemos aplicar el algoritmo LRU, eliminando la entrada que lleva más tiempo sin utilizar o

eliminar la entrada con mayor posibilidad de no ser efectiva según su historial de salto.

Predictor de Smith o Predictor Bimodal

El predictor de Smith, o predictor bimodal, es el algoritmo de predicción dinámica del resultado de salto más sencillo.

Se basa en asociar un contador de saturación de ***k*** bits a cada salto de forma que el bit más significativo del contador indica la predicción para el salto.

- Si el bit es 0, el salto se predice como no efectivo (Not Taken o NT).
- Si el bit es 1, el salto se predice como efectivo (Taken o T).

Los ***k*** bits que forman el contador de saturación constituyen el historial de salto y es la información que se almacena en el campo BH de la BTB, estos bits junto con el resultado actual del salto se utilizan para realizar la predicción de lo que sucederá cuando se ejecute de nuevo ese salto.

En un contador de saturación cuando está en su valor máximo y se incrementa no vuelve a 0, sino que se queda igual, a su vez cuando alcanza el valor 0 y se decrementa su valor continúa siendo 0.

Cuando se invoca un salto se extrae el historial del salto de la BTB, se mira la predicción, el bit más significativo, y se aplica.

Procedemos a actualizar el contador con el resultado real del salto para dejar preparada la próxima predicción:

- Si el ***salto es efectivo***, el contador se incrementa.
- Si el ***salto no es efectivo***, el contador se decrementa.

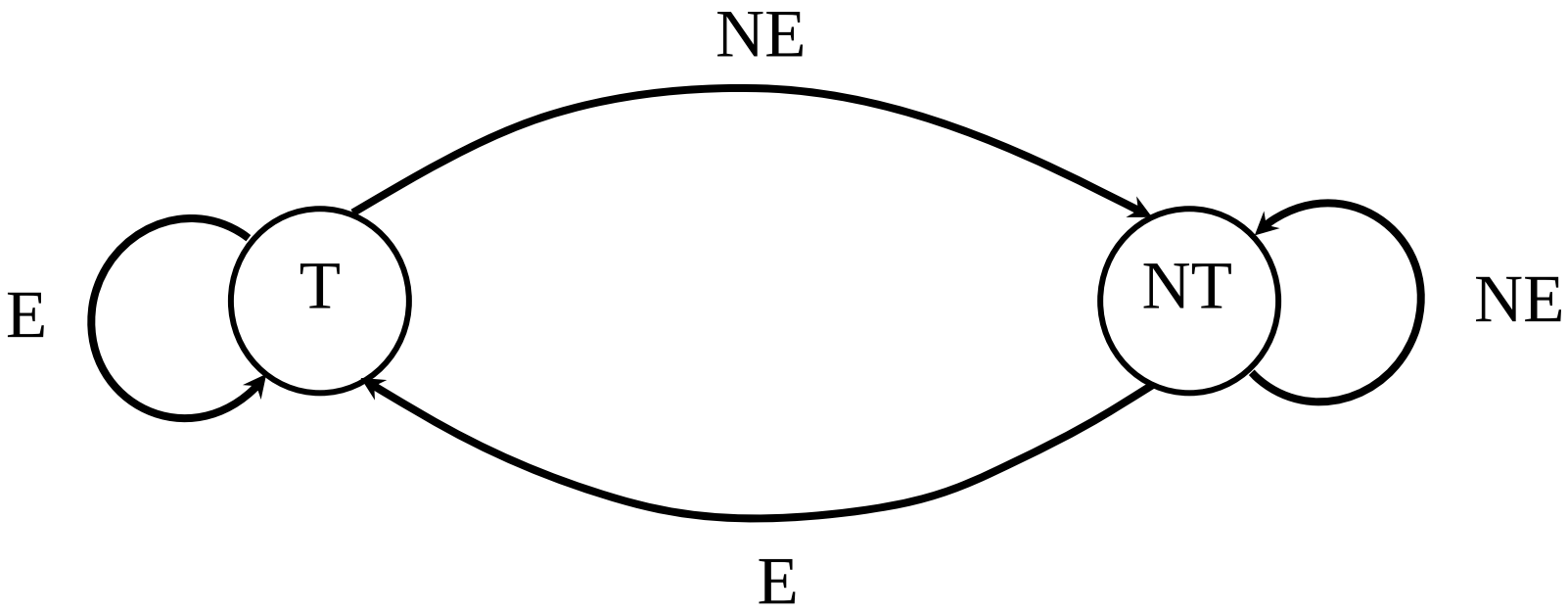
Para valores altos del contador, el salto se predecirá como efectivo y para valores bajos como no efectivo.

Podemos utilizar un contador de 1 bit ($Smith_1$), pero el más utilizado es el de 2 bits ($Smith_2$), el contador de dos bits presenta cuatro posibles estados:

SN (Stongly Not Taken):	00	Salto no efectivo.
WN (Weakly Not Taken):	01	Salto no efectivo.
WT (Weakly Not Taken):	10	Salto efectivo.
ST (Stongly Taken):	11	Salto efectivo.

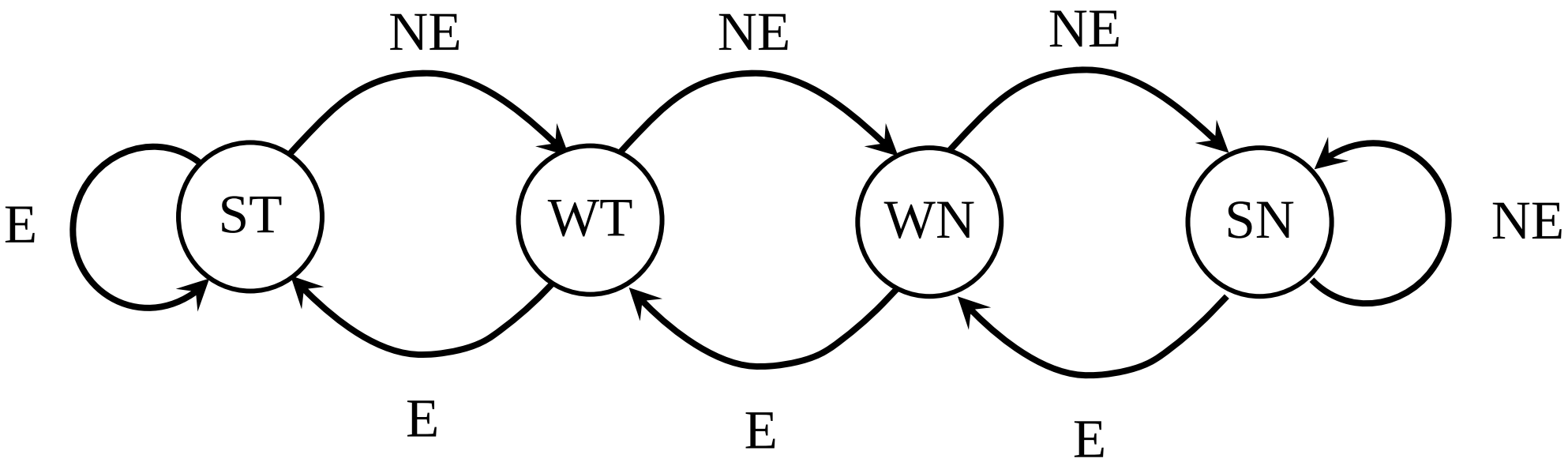
Las transiciones del autómata representan el resultado real del salto.

Los estados representan el historial del salto.



Predicción Smith:
T(Taken): 1
NT(Not Taken): 0

Resultado del salto
E: Efectivo
NE: No Efectivo



Predicción Smith$_2$:
ST (Strongly Taken): 11
WT (Weakly Taken): 10
WN (Weakly Not Taken): 01
SN (Strongly Not Taken): 00

Resultado del salto:
E: Efectivo
NE: No Efectivo

Predictor de dos Niveles Basado en el Historial Global

Estos predictores mantienen en un ***primer nivel*** de información un historial de los últimos saltos ejecutados, ***historial global***, o de las últimas ejecuciones de un salto concreto, ***historial local***.

En un ***segundo nivel***, la información del primer nivel en combinación con la dirección de la instrucción de salto se utiliza para acceder a una tabla que almacena contadores de saturación que son los que determinan la predicción.

Este predictor se basa en un registro en el que se almacena el resultado de los saltos más recientes.

El historial de los últimos ***h*** saltos se almacena en un registro de desplazamiento de ***h*** bits denominado registro del historial de saltos **BHR**, ***Branch History Register***.

Cada vez que se ejecuta un salto se introduce su resultado por el extremo derecho del registro, se desplaza el contenido una posición y se expulsa el resultado más antiguo por el extremo izquierdo.

Si el salto es efectivo se introduce un 1, caso contrario un 0.

Para conocer la predicción de un salto los ***h*** bits del BHR se concatenan con un subconjunto de ***m*** bits obtenido mediante la aplicación de una función hash a la dirección de la instrucción de salto.

La combinación de ***h+m*** bits se utiliza para acceder a una ***tabla de historial de patrones*** **PHT** (***Pattern History Table***).

Esta tabla almacena en cada una de sus 2^{h+m} entradas un contador de saturación de 2 bits.

El ***bit más significativo*** del contador representa la predicción del salto, 1 efectúa el salto y 0 no.

En cuanto evaluamos el salto y conocemos su resultado hay que proceder a la actualización de los dos componentes del predictor.

Se incrementa o decrementa el contador de la PHT y se actualiza el historial del BHR, quedando todo preparado para la próxima predicción.

La aplicación de la función hash es fundamental para reducir la longitud de la dirección de la instrucción a ***m*** bits.

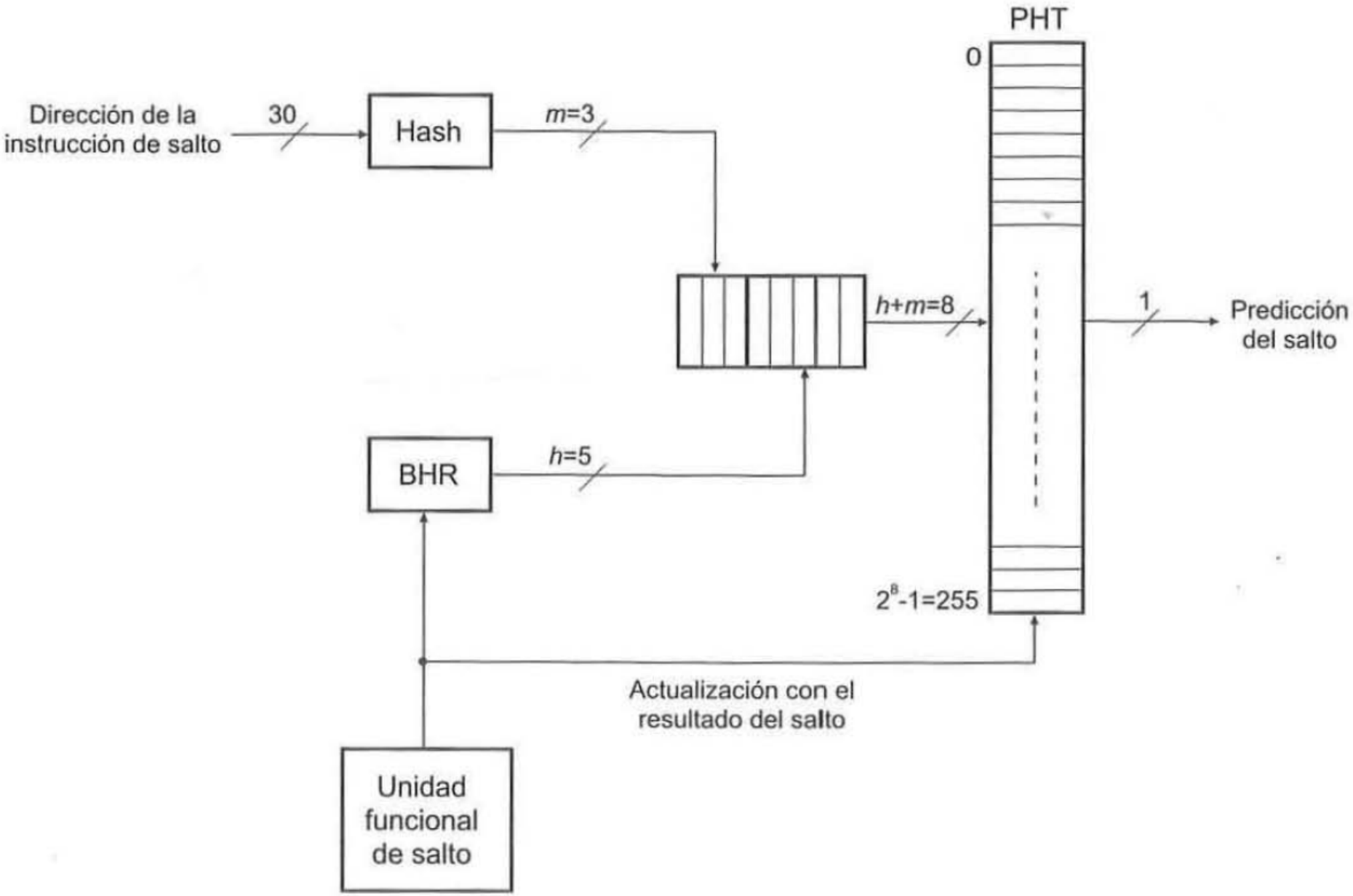


Figura 2.17: Predictor de dos niveles basado en Historial Global

Predictor de Dos Niveles Basado en el Historial Local

Es muy similar al anterior, pero la principal diferencia es que utiliza una tabla BHT en la que se almacena el historial concreto de cada salto en lugar de un único registro BHR que mantiene un historial común para todos los saltos.

Para obtener la predicción de un salto hay que recuperar el historial del salto, el acceso a la BHT se realiza mediante un hashing de la dirección de la instrucción de salto que los reduce a $\boldsymbol{k}$ bits ya que el número de entradas de la BHT es $\boldsymbol{2^k}$.

Concatenamos los ***h*** bits del historial con los *m* bits obtenidos mediante otro hashing de la dirección de la instrucción de salto.

Con los ***h+m*** bits accedemos a la PHT para recuperar el estado del contador de saturación de 2 bits.

La función hash puede ser muy básica y consistir en quedarse con los ***m*** o ***k*** bits menos significativos de la dirección, pudiendo recurrir a funciones más complejas.

En cuanto evaluamos el salto y conocemos su resultado hay que proceder a la actualización de los dos componentes del predictor.

Accedemos al PHT para actualizar el contador de saturación con el resultado, se suma 1 si fue efectivo y se resta 1 en caso contrario.

Se accede al historial del BHT para introducir el resultado del salto, 1 si fue efectivo, 0 en caso contrario, desplazando su contenido.

Quedando actualizados los dos niveles de predictores para la predicción del siguiente salto.

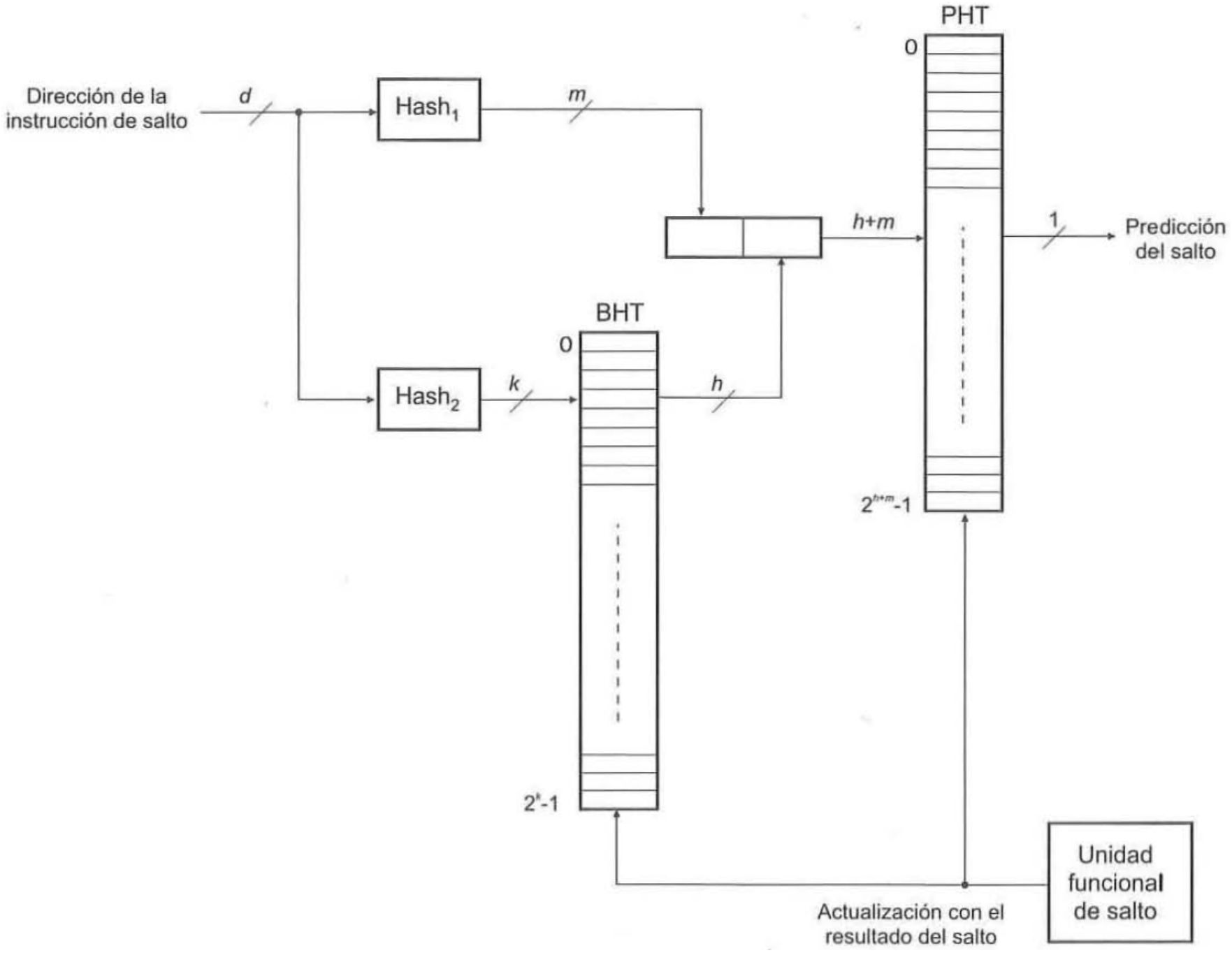


Figura 2.18: Predictor de dos niveles basado en Historial Local

Predictor de Dos Niveles de Índice Compartido Gshare

El ***predictor gshare*** es una variante del predictor de dos niveles de historial global.

Se realiza una función XOR entre los *h* bits superiores de los *m*.

Los *h* bits obtenidos de la XOR se concatenan con los *m*-*h* bits restantes para poder acceder a la PHT.

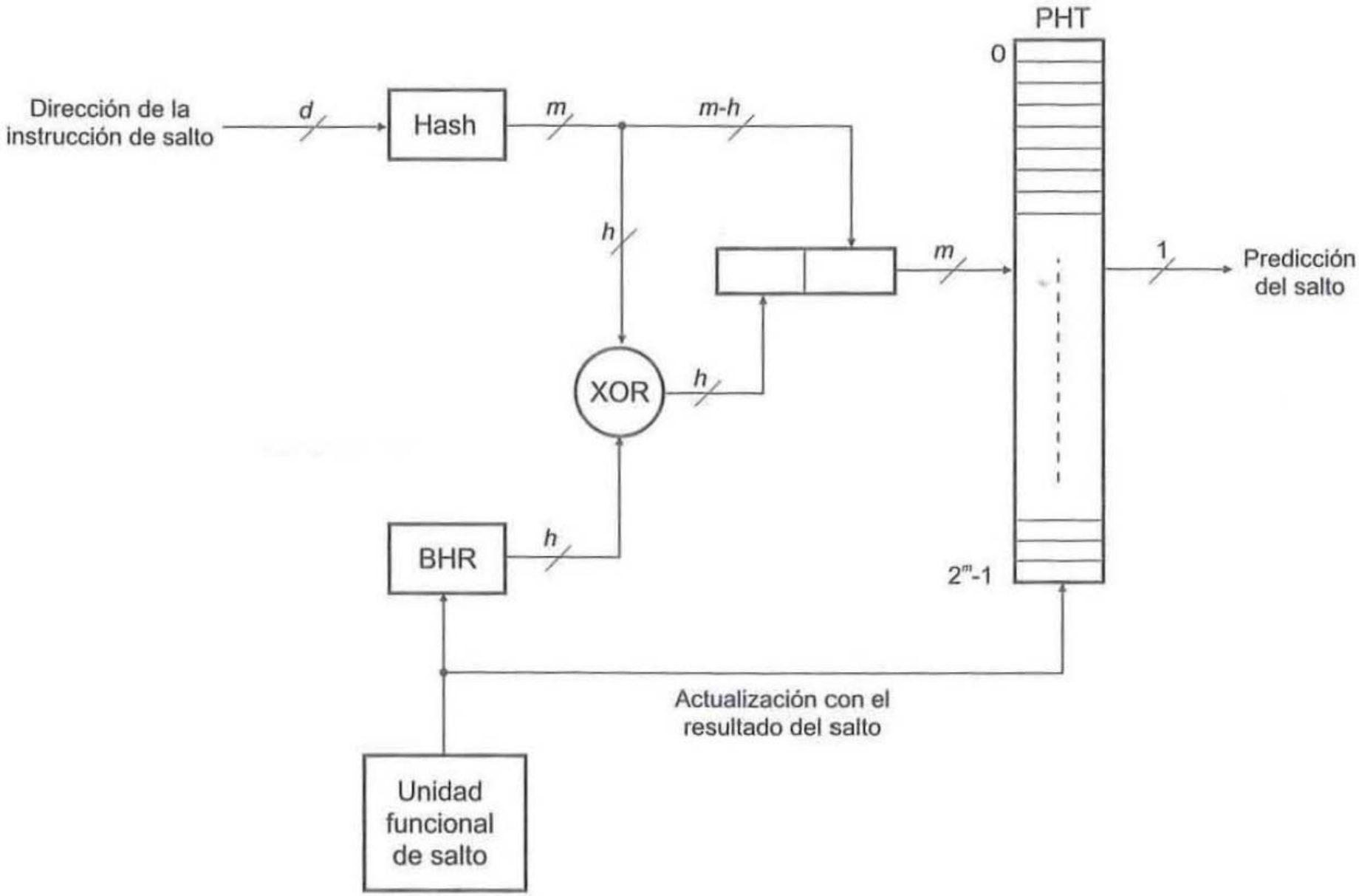


Figura 2.19: Esquema de un predictor *gshare*.

Predictores Híbridos

Los procesadores superescalares recurren a dos predictores que generan un resultado y un selector que se ocupa de decidir cuál de las dos predicciones hay que utilizar, se conoce como ***predicción híbrida***.

Pila de Dirección de Retorno

El ***retorno de subrutina*** es una instrucción especial de salto que no puede predecirse adecuadamente ya que cada vez que se la invoca puede saltar a una dirección completamente diferente.

La BTB generaría predicciones de la dirección de destino con una elevada tasa de fallos.

Las invocaciones a una subrutina se pueden realizar desde distintos lugares de un programa.

Por eso la instrucción de salto que hay al final de la subrutina para devolver el control no tiene una dirección de destino fija.

Esta instrucción de salto nunca obtiene predicciones correctas de la BTB ya que corresponderán a la dirección de retorno de la invocación previa.

El tratamiento correcto de los retornos de subrutina se realiza mediante una ***pila de direcciones de retorno*** **RAS** (***Return Addres Stack***)o ***buffer de pila de retornos*** **RSB (*Return Addres Buffer*)**.

Cuando se invoca una subrutina mediante una instrucción de salto se efectúan tres acciones:

- Se accede a la BTB para obtener la predicción de la dirección de destino.
- Se especula el resultado del salto.
- Se almacena en la RAS la dirección de la instrucción siguiente al salto.

Una vez procesadas las instrucciones de la subrutina, se invoca una instrucción de retorno de subrutina, cuando se detecta que la instrucción es de retorno se accede a la RAS para obtener la dirección correcta de retorno y se desestima la predicción de la BTB.

Las instrucciones procesadas como consecuencia de una especulación incorrecta dada por la BTB son anuladas.

El problema que tiene la RAS es el desbordamiento cuando tratamos subrutinas con muchos anidamientos y la pila no ha sido dimensionada correctamente.

Tratamiento de los Errores en la Predicción de los Saltos

Uno de los problemas de la ejecución especulativa de instrucciones es que el resultado de la predicción del salto no coincida con el resultado verdadero del salto.

En este caso se deshace el procesamiento de todas las instrucciones y se continúa con el procesamiento correcto, esto es la ***recuperación de la secuencia correcta***.

La forma habitual para conocer y validar o anular las secuencias de instrucciones que se están ejecutando de forma especulativa desde que entran en la fase de distribución hasta que son terminadas es etiquetarlas durante todo el tiempo que estén en la segmentación, para conocer su estado en cualquier etapa del cauce.

Dos de esas etiquetas son:

- La especulativa.
- La de validez.

Las etiquetas son campos de uno o más bits que hay en el buffer de reordenamiento, aquí todas las instrucciones tienen una entrada en la que se actualiza su estado desde que son distribuidas y hasta que son terminadas arquitectónicamente, se dice que están ***en vuelo***.

Si la ***etiqueta especulativa*** es de un bit, el valor de 1 identifica la instrucción como ***instrucción especulada***.

Si el procesador permite varias rutas en paralelo, se denominan ***nivel de especulación***, la etiqueta dispondrá de más bits para identificar cada uno de los bloques de instrucciones especuladas.

A continuación hay que especificar la dirección de las instrucciones que se almacenan en una tabla junto con la etiqueta especulativa que se asocia a las instrucciones.

Posteriormente esta etiqueta permite identificar y validar las instrucciones especuladas si la predicción es correcta o realizar la recuperación de la secuencia correcta en caso de error.

La etiqueta ***validez*** permite saber si la instrucción debe o no terminarse arquitectónicamente, es decir si el resultado se escribe en un registro arquitectónico o en la memoria.

Inicialmente todas las instrucciones son válidas, aunque no puede terminarse arquitectónicamente hasta que la etiqueta que la define como especulativa cambie a no especulativa, todos los bits a 0.

En el momento en que se evalúa el salto, si la predicción coincide con el resultado, las etiquetas se cambian de manera que las instrucciones especuladas a ese salto son correctas y terminan arquitectónicamente.

Si la ***predicción es incorrecta*** hay que realizar dos acciones:

- Invalidar las instrucciones especuladas, el bit de validez de todas esas instrucciones pasa a indicar invalidez y no son terminadas arquitectónicamente, no se accede al banco de registros o a la memoria para escribir sus resultados.
- Recuperar la ruta correcta, implica iniciar la lectura de instrucciones desde la dirección de salto correcta.

- Si la predicción incorrecta fue:

  - No realizar el salto se utiliza el resultado del salto como nuevo valor del PC.

  - Realizar el salto, se accede a la tabla en la que se almacenó la dirección de la instrucción de salto y se utiliza para obtener el nuevo valor del PC, el de la siguiente instrucción (*ruta fall-through*).

Decodificación

Tras la extracción simultánea de varias instrucciones el siguiente paso es la decodificación. Es una de las etapas más críticas en un procesador superescalar.

En un procesador RISC superescalar en una etapa de decodificación tiene que decodificar varias instrucciones en paralelo e identificar las dependencias de datos con todas las instrucciones que componen el grupo de lectura y con las que se están ejecutando en las unidades funcionales.

Otra complicación es que serían varias las instrucciones que necesitarían leer todos sus operandos o parte de ellos. También se debe confirmar si hay saltos en falso en el grupo de lectura para anular la ruta especulada y minimizar el impacto del procesamiento erróneo.

Por todo esto los procesadores superescalares reparten estas tareas en dos etapas:

- ***Decodificación***.
- ***Distribución***.

Decodificación

Detecta los saltos en falso.

Realiza la adaptación del formato de las instrucciones a la estructura interna de control y datos del procesador.

Distribución

Se ocupa del renombramiento de los registros.

De la lectura de los operandos.

Del análisis de las dependencias verdaderas.

En los procesadores CISC las instrucciones pueden tener longitudes diferentes.

Hay que analizar el código de operación de cada instrucción para conocer su formato y longitud.

En los RISC la longitud de instrucción es fija.

Sabemos dónde empieza y acaba una instrucción.

Además la extracción del buffer de instrucciones se realiza muy rápido sin importar si es una o varias instrucciones a decodificar.

Los factores que determinan la complejidad de la etapa de decodificación son:

- El tipo arquitectónico del procesador (RISC o CISC).
- Número de instrucciones a decodificar en paralelo (ancho de segmentación).

Se suelen adoptar tres soluciones:

- ***Descomponer la etapa*** de decodificación en varias subetapas o fases, aumentando el número de ciclos de reloj que se emplean.
- Realizar una parte de la decodificación antes que la extracción de instrucciones de la I-caché, esto se conoce como etapa de ***predecodificación o decodificación previa***.
- ***Traducción de instrucciones***, se descompone una instrucción en instrucciones más básicas, o unir varias instrucciones en una única instrucción interna.

Una vez completada la decodificación, las instrucciones pasan al buffer de distribución y a las estaciones de reserva para iniciar su verdadero procesamiento.

Desde los buffers se reparten las instrucciones entre las unidades funcionales del procesador.

Predecodificación

Esta etapa está situada antes de la I-caché de forma que las instrucciones que recibe la I-caché desde la caché de nivel 2 o memoria principal como consecuencia de un fallo de lectura pasan obligatoriamente por esta etapa.

Está constituida por hardware situado antes de la I-caché que realiza una decodificación parcial de las instrucciones, este proceso analiza cada instrucción y la concatena un pequeño conjunto de bits con información sobre ella.

Los ***bits de predecodificación*** son un conjunto de bits que se añaden a cada instrucción cuando son enviados a la I-caché para simplificar las tareas de decodificación y distribución.

Se utilizan posteriormente en la etapa de *fetch* para determinar el tipo de instrucción de salto que hay en el grupo de lectura y proceder o no a su especulación y en la etapa de decodificación para determinar la forma en que se agrupan para su posterior distribución.

Estos bits también identifican las instrucciones que son susceptibles de provocar una excepción.

Inconvenientes de la decodificación previa:

- La necesidad de un mayor ancho de banda.
- El incremento del tamaño de la I-caché.

Otra forma de hacer la predecodificación de instrucciones es situar esta fase entre el buffer interno de fetch que recibe las instrucciones de la I-caché y el buffer de instrucciones que alimenta la etapa de decodificación.

Esta etapa se suele considerar parte de la etapa *fetch* y no de la de decodificación.

La técnica es propia de los CISC y el objetivo de estos bits es:

- Determinar la longitud de las instrucciones.
- Decodificar ciertos prefijos asociados a las instrucciones.
- Señalar determinadas propiedades de las instrucciones a los decodificadores.

La precodificación adelanta parte del trabajo que realiza la etapa de decodificación reduciendo la profundidad de su segmentación.

Una segmentación menos profunda permite que la recuperación de un fallo en la especulación de un salto no encarezca los ciclos de reloj.

Traducción de instrucciones

En la fase de decodificación se realiza la traducción de una instrucción compleja en un conjunto de instrucciones más básicas de tipo RISC.

Estas operaciones básicas se llaman:

- En Intel microoperaciones (micro-ops).
- En PowerPC operaciones internas (IOPs – Internal OPerations).
- En AMD ROPs (Rips Operations).

Esta técnica es propia de arquitecturas CISC y también de las RISC para reducir la complejidad de ciertas instrucciones.

La intención es simplificar el repertorio original de instrucciones para que su procesamiento hardware se pueda realizar directamente en el procesador.

La arquitectura Intel Core complementa la traducción de instrucciones con dos técnicas adicionales:

- ***Macro-fusión***
- ***Micro-fusión***

Macro-fusión

Fusiona ciertos tipos de instrucción en la fase de predecodificación y las envía a uno de los decodificadores para producir una única *micro-ops*, denominada *macro-fused micro-op*.

La macro-fusión permite realizar más trabajo con menos recursos hardware.

Esto hace aumentar el ancho de banda de decodificación, ya que el buffer se vacía más rápido al retirar dos instrucciones por ciclo.

También se produce un incremento virtual del ancho de la segmentación en el núcleo de ejecución dinámica (número de unidades funcionales).

Micro-fusión

Decodifica una instrucción que genera dos *micro-op* y las funde en una *micro-op*.

La nueva *micro-op* solo ocupa una entrada en el buffer de reordenamiento (ROB), al emitirse se desdobla en sus dos componentes originales ejecutándose en paralelo en unidades funcionales diferentes o en serie si es una.

La aplicación más habitual es con instrucciones de almacenamiento ya que implica su desdoblamiento en dos *micro-ops*, una para el cálculo de la operación de destino y otra para la escritura del dato.

Produce un aumento de la capacidad de decodificación y de la eficiencia energética de la arquitectura al emitir más *micro-op* con menos hardware.

En los PowerPC se denomina:

- ***Instrucciones rotas*** a las que se descomponen en dos IOPs.
- ***Instrucciones microcodificadas*** a las que se descomponen en tres o más IOPs.

Distribución

En esta etapa se establece el punto de partida para la ejecución de instrucciones en paralelo y fuera de orden en una segmentación superescalar.

Se reparten las instrucciones según su tipo entre las distintas unidades funcionales para proceder a la ejecución en paralelo.

La distribución es el último componente del *front-end* de un procesador superescalar después de la etapa de *fetch* y de decodificación.

Es el punto de inflexión entre el procesamiento centralizado de las instrucciones y el procesamiento distribuido.

Después de la decodificación, las instrucciones se depositan temporalmente en dos buffers llamados ***buffer de distribución*** o ventana de instrucciones y ***buffer de terminación*** o de reordenamiento.

Para ejecutar una instrucción necesitamos todos los operandos fuente y estar libre una de las unidades funcionales.

Puede ocurrir que los operandos no estén disponibles y haya que esperar por algún resultado de instrucciones que están ejecutándose.

Puede ocurrir que los operandos estén disponibles pero no su unidad funcional o también que todo esté disponible pero que no podamos enviar la instrucción a la unidad funcional debido a que no hay suficientes buses.

Hay un límite en la cantidad de instrucciones por ciclo que se pueden enviar desde la ventana de instrucciones a las unidades funcionales, son los riesgos estructurales.

Este problema lo resolvemos deteniendo la instrucción en la etapa de decodificación hasta que todo esté listo para poder emitirla, esta no es una buena solución ya que reduce el rendimiento de la decodificación y del procesador introduciendo burbujas en la segmentación.

La solución que vamos a utilizar es *desacoplar la etapa de decodificación de la de ejecución* utilizando la ventana de instrucciones.

El ***desacoplo*** consiste en decodificar la instrucción al máximo y no detenerla para que avance hacia la ventana de instrucciones.

En la ventana de instrucciones se deposita la instrucción con los identificadores de los operandos fuente y además se indica mediante un bit de validez por operando si está disponible.

Cuando se cumplan las condiciones necesarias se emitirá la instrucción que está a la espera en la ventana de instrucciones.

Resumiendo, este mecanismo permite distribuir y emitir las instrucciones de forma que se respeten las dependencias verdaderas.

Organización de la Ventana de Instrucciones

Existen varias formas de organizar la ventana de instrucciones:

- ***Estación de reserva centralizada***, es lo que hemos definido hasta ahora como ventana de instrucciones o buffer de instrucciones (Ventana de emisión o cola de emisión). Tiene un hardware de control muy complejo. Los términos distribución y emisión significan lo mismo ya que la asociación de la instrucción a la unidad funcional se produce en el momento del envío.

- ***Estaciones de reserva distribuidas o individuales***, cada unidad funcional dispone de una estación de reserva propia. Un buffer de distribución que recibe las instrucciones de la etapa de decodificiación se ocupa de distribuirlas a las estaciones de reserva individuales según su tipo. Esta estructura aumenta la complejidad de los buses que hay que utilizar para reenviar los resultados de las distintas unidades funcionales a las estaciones de reserva y bancos de registro para emitir nuevas instrucciones. La distribución es el envío desde el buffer de distribución a la estación de reserva individual y la emisión es el

envío desde la estación de reserva individual a la unidad funcional para que se ejecute.

- ***Estaciones de reserva en clústers o compartidas***, las estaciones de reserva reciben las instrucciones del buffer de distribución pero una estación de reserva da servicio a varias unidades funcionales del mismo tipo. La distribución es el envío desde el buffer de distribución a una estación de reserva y la distribución/emisión se produce al enviar la instrucción a una de las unidades funcionales asignada.

La utilización de una configuración depende del equipo de diseño.

- ***Distribución***: asociar una instrucción a una unidad funcional.
- ***Emitir***: enviar la instrucción a la unidad funcional para comenzar la ejecución.

Operativa de una Estación de Reserva Individual

Una estación de reserva es un buffer de almacenamiento con múltiples entradas en donde se almacenan las instrucciones ya decodificadas.

Cada entrada del buffer es un conjunto de bits agrupados por campos y representa una instrucción, está formado por los siguientes campos:

- ***Ocupado*** (O): la entrada está ocupada por una instrucción válida pendiente de emisión.
- ***Código de operación*** (CO): contiene el código de operación de la instrucción.
- ***Operando 1*** (Op1): si el registro correspondiente al primer operando fuente está disponible, este campo almacena el valor del registro o el

identificador. Si no está disponible contiene el identificador del registro.

- ***Válido 1*** (V1): indica si el operando fuente está disponible o no.
- ***Operando 2*** (Op2): similar a Op1
- ***Válido 2*** (V2): similar a V1
- ***Destino*** (D): almacena el resultado de la operación de forma temporal.
- ***Listo*** (L): indica que todos los operandos ya están disponibles y la instrucción puede emitirse a la unidad funcional correspondiente.

El formato de las entradas de las estaciones de reserva es similar tanto si están organizadas de forma individual o compartida, la diferencia surge en las entradas del buffer de distribución que se ocupa de distribuir las instrucciones a las estaciones individuales o agrupadas.

En las estaciones individuales y en las compartidas las entradas del buffer de distribución no disponen de bits de validez ya que se asignan cuando las instrucciones son enviadas desde el buffer a las estaciones.

En una estación de reserva centralizada los bits de validez se establecen al salir de la etapa de decodificación.

El mecanismo de las estaciones de reserva resuelve el problema de las dependencias RAW, esto ocurre al forzar la espera de los operandos pero a la vez permitiendo la ejecución distribuida y fuera de orden de las instrucciones no dependientes.

El hardware de control para la asignación y emisión de instrucciones es bastante complejo.

Las fases por las que pasa una instrucción desde que sale del buffer hasta que se emite a la estación de reserva son: ***distribución, supervisión y emisión***.

Fase de Distribución

Consiste en el envío de una instrucción desde el ***buffer de distribución*** *o* ***ventana de instrucciones*** a la estación de reserva individual que le corresponda según su tipo.

La introducción de instrucciones en la estación de reserva se realiza de forma ordenada por parte de la ***lógica de asignación***.

La lógica de asignación se ocupa de:

- Ubicar correctamente la instrucción recibidas, utiliza un registro de las instrucciones almacenadas y de las entradas que están libres.
- Establece inicialmente los bits de validez.

Si se utiliza una estación de reserva centralizada el origen de las instrucciones es la etapa de decodificación.

Fase de Supervisión

Cuando la instrucción está almacenada en la estación de reserva, comienza la fase de supervisión y termina cuando están los dos operandos fuente disponibles y está lista para ser emitida.

Las instrucciones que tienen algún operando fuente marcado como no disponible se encuentran en espera activa, supervisando los buses de reenvío CDB (Common Data Bus), que es donde cada unidad funcional publica el resultado y el identificador del registro destino en el que almacenar el resultado.

El hardware que realiza la supervisión de los buses se suele denominar ***lógica de activación***.

La lógica de activación compara continuamente los identificadores de los operandos no disponibles de todas las instrucciones que hay en la estación de reserva con los identificadores que se publican en los buses de reenvío.

En cuanto haya una coincidencia de identificadores se cambia el bit de validez del operando fuente correspondiente, se lee el valor del operando del bus de reenvío y si todos los operandos están listos se activa el bit que señala a la instrucción como preparada para ser emitida, *bit L*.

La activación del bit L se conoce como ***activación de la instrucción***.

La complejidad de la lógica de activación es elevada y aumenta con el tamaño y el número de estaciones de reserva.

Fase de Emisión

Cuando la instrucción tiene todos sus operandos disponibles comienza la fase de emisión.

En esta fase las instrucciones se mantienen en espera activa hasta que la ***lógica de selección*** determina la que se puede emitir.

La fase de emisión da paso a la etapa de ejecución.

La lectura de los operandos se realiza en el momento en que la instrucción se emite a la unidad funcional.

Al mismo tiempo el código de la operación y el identificador del registro destino se envían a la unidad funcional desde la estación de reserva y los operandos se leen del fichero de registros y se remiten a la unidad funcional.

Cuando se emite una instrucción se libera la entrada asociada para que se pueda distribuir una nueva instrucción.

Para las instrucciones que pueden provocar algún tipo de interrupción se mantienen en la estación de reserva

hasta que no haya concluido completamente su ejecución.

Si en una estación de reserva individual solo hay una instrucción lista para emitir no hay problema, se envía a la unidad funcional y se marca su entrada como libre para poder ser ocupada por otra instrucción.

El problema se presenta cuando hay varias instrucciones listas para ser emitidas en el mismo ciclo de reloj y la lógica de selección debe decidir cuál hay que emitir.

La lógica de selección es un ***algoritmo de planificación o planificador dinámico***.

El algoritmo más habitual selecciona la instrucción más antigua para emitirla a la espera de que la unidad funcional esté libre y pueda comenzar la ejecución de la instrucción.

Los primeros procesadores superescalares realizaban una ***emisión con bloqueo y ordenada,*** esto implica que las instrucciones salen en orden de la estación de reserva excepto que no tuviese todos sus operandos disponibles debiendo esperar y bloqueando las instrucciones posteriores.

Los procesadores actuales ya realizan ***emisión sin bloqueo y desordenada***, mejorando notablemente su rendimiento.

La emisión de instrucciones desde las estaciones de reserva puede realizarse de forma:

- ***Alineada***, la ventana de distribución no puede enviar nuevas instrucciones a la estación de reserva hasta que no esté completamente vacía.

- ***No alineada***, se pueden distribuir instrucciones desde el buffer de distribución siempre que queden entradas libres en las estaciones de reserva.

Lectura de los Operandos

La primera forma de efectuar la lectura de los operandos se realizaba siempre en el momento en que se emitían las instrucciones desde las estaciones de reserva individuales a las unidades funcionales, se conoce como ***planificación sin lectura de operandos***.

Cuando la instrucción se emite por parte del planificador, todavía no se han extraído los valores de los operandos fuente del fichero de registros.

Cuando una instrucción se distribuye desde la ventana de instrucciones a una estación de reserva individual las entradas contienen únicamente identificadores de los registros fuente. Se analizan las instrucciones en busca de dependencias verdaderas y se marcan en el fichero de registros como inválidos los que resulten ser destino de una operación.

La lógica de asignación se encarga de asignar los bits de validez de los operandos fuente mediante una consulta al fichero de registros.

Consideraremos que la asignación de los bits de validez al fichero de registros y a las entradas a las estaciones de reserva se realizan en la fase de distribución y no en la de decodificación.

Una de las ventajas de este modo de organización es que el ancho de las estaciones de reserva y de los buses de reenvío se reduce mucho.

El inconveniente es que el tiempo desde la emisión de la instrucción hasta que comienza su ejecución es

mayor por el tiempo necesario para extraer los operandos.

La segunda forma de efectuar la lectura de los operandos es cuando la instrucción se distribuye desde el buffer de distribución a las estaciones de reserva, esto es la ***planificación con lectura de operandos***.

Las instrucciones se distribuyen a las estaciones de reserva y los valores de los operandos fuente disponibles en el fichero de registros son leídos y copiados en la estación de reserva.

El código de operación y el identificador de registro se copian directamente en la estación de reserva desde el buffer de distribución y los identificadores de los operandos fuente se envían al fichero de registros.

Si el operando está disponible en el fichero de registros, la estación de reserva recibe el valor y coloca el bit de validez de su entrada a 1.

Si el registro no está disponible se reenvía el identificador a la estación y se coloca el bit de validez a 0.

Tras la emisión de la instrucción la unidad funcional publica en el bus de reenvío el resultado de su operación con el identificador del registro destino, se copia el resultado en el fichero de registros y en algunas entradas de la estación de reserva.

En el fichero de registros se actualiza el valor del registro y se modifica el bit de validez para indicar que ya está disponible y no es el registro destino de ninguna instrucción posterior.

Las estaciones de reserva reemplazan el identificador que presenta alguna coincidencia por el valor del operando y su bit de validez se pone a 1.

El identificador de los registros detecta la disponibilidad de un operando y procede a la activación de las instrucciones.

Renombramiento de Registros

Aunque la ejecución fuera de orden permite maximizar el rendimiento de los procesadores superescalares tiene una serie de inconvenientes:

- La gestión de las dependencias falsas WAR, ***antidependencias*** y
- WAW ***dependencias de salida***.

Si el **ISA** (***Instruction Set Architecture***) tuviese un número infinito de registros no existirían las dependencias falsas, que surgen como consecuencia de reutilizar los registros accesibles por el repertorio de instrucciones para efectuar el almacenamiento temporal de los resultados.

El ***fichero de registros creados*** son los registros accesibles al programador, que son un recurso limitado y que pueden utilizarse por el ISA.

También se conoce como ***fichero de registros arquitectónicos***.

Para generar el código objeto, el compilador utiliza un número ilimitado de registros simbólicos como almacenamiento temporal, manteniendo el máximo de datos en los registros.

De esta forma se minimizan los accesos al sistema de memoria ya que se consumen muchos ciclos.

Para general el código definitivo el compilador realiza una asignación del conjunto infinito de registros simbólicos al conjunto finito de registros arquitectónicos.

El compilador utiliza la siguiente regla, se escribe un nuevo valor en un registro cuando se detecta que el valor almacenado en el registro ya no es necesario para operaciones futuras.

El ***rango de vida de un registro*** es el tiempo durante el que el dato almacenado en un registro es válido, es decir el tiempo entre dos etapas consecutivas.

Dentro del rango de vida se pueden efectuar múltiples lecturas del valor ya que estas operaciones no le quitan validez al dato.

El final y comienzo del rango de vida del mismo registro se produce debido a que se realiza el reciclaje de ese registro.

Los rangos de vida de diferentes registros se pueden intercambiar y mezclar siempre que se respeten las dependencias verdaderas.

La solución adoptada para resolver las dependencias falsas de datos es el ***renombramiento dinámico de los registros de la arquitectura mediante hardware***.

Consiste en utilizar un conjunto de registros auxiliares, invisibles al programador, de forma que se restablezca la correspondencia única entre resultados temporales y registros.

Se conocen como ***registros físicos***, ***registros no creados*** o ***registros de renombramiento***.

Las instrucciones escriben sus resultados en los registros no creados para después deshacer el renombramiento y proceder a la escritura ordenada de los registros arquitectónicos utilizando los valores de los registros no creados.

Este restablecimiento permite eliminar todas las dependencias falsas entre las instrucciones emitidas.

Este proceso consta de dos pasos:

- ***Resolución de los riesgos WAW y WAR***. Se renombran de forma única los operandos destino de las instrucciones y de esta manera se resuelven las dependencias WAW y WAR.

- ***Mantenimiento de las dependencias RAW***, se renombran todos los registros fuente que son objeto de una escritura previa, el objetivo de este paso es respetar las dependencias RAW.

En realidad el procesador realiza el renombramiento instrucción tras instrucción, analizando si los dos operandos fuente, renombrados o no, están disponibles y efectuando el renombramiento del registro destino.

Para realizar esto se recurre a un hardware adicional que trabaja con el fichero de registros arquitectónicos.

La fase de renombramiento se puede realizar en la fase de decodificación o en la de distribución.

El renombramiento dinámico se realiza incluyendo en el procesador un nuevo fichero de registros denominado ***fichero de registros de renombramiento*** **RRF** (*Rename Register File*) o ***buffer de renombramiento***.

Al ***fichero de registros de la arquitectura*** se denominará **ARF** (*Architected Register File*).

Existen tres formas de organizar el **RRF**:

- Como un único fichero de registros formado por la suma del RRF y ARF.
- Como estructura independiente pero accesible desde ARF.
- Como parte del buffer de reordenamiento y accesible desde el ARF.

Organización Independiente del RRF con Acceso Indexado

A continuación se describe la implementación del RRF como una estructura independiente del ARF.

Las entradas del **ARF** se componen de tres campos:

- **Datos**, contiene el valor del registro.
- **Ocupado**, indica si el registro ha sido renombrado.
- **Índice**, apunta a la entrada del RRF.

La estructura de los registros **RRF** tienen tres campos:

- **Datos**, se escribe el resultado de la instrucción que ha causado el renombramiento.
- **Ocupado**, tiene un bit de longitud, comprueba si el registro está siendo utilizado por instrucciones pendientes de ejecución y no pueden liberarse.
- **Válido**, tiene un bit de longitud, indica que aún no se ha realizado la escritura en el RRF.

La lectura de un registro puede encontrarse frente a tres situaciones:

- El registro **ARF** no está pendiente de ninguna escritura.

  Su bit de ***Ocupado*** permanece a 0 indicando que no hay renombramiento.

  Se procede a la lectura del valor almacenado en el campo ***Datos*** de su entrada en el **ARF**.

- El registro **ARF** es destinatario de una escritura por lo que ha sido renombrado.

  Su bit de ***Ocupado*** está a 1.

  El campo ***c*** contiene un puntero a una entrada del RRF.

  El puntero no es otra cosa que el identificador de uno de los registros de renombramiento del RRF.

  Una vez que se accede al RRF se pueden plantear ***dos situaciones*** según el estado del registro de renombramiento:

- **Campo Válido** = 0.

  La actualización del contenido del registro RRF con el resultado de la instrucción que provocó el renombramiento está pendiente.

  El operando no está disponible y se procede a enviar el identificador del registro de renombramiento a la estación de reserva.

- **Campo Válido** = 1.

  El valor del registro de renombramiento con el resultado de la instrucción de escritura se ha actualizado.

  El operando está disponible y se puede extraer el valor del RRF si una instrucción lo necesita como operando fuente.

El ***proceso de renombrar el registro destino*** de una instrucción cuando ésta se distribuye a la estación de reserva individual es el siguiente:

Se accede mediante su identificador a la entrada que le corresponde en el ARF.

Se establece el bit de ***Ocupado*** a l.

Se selecciona un registro de renombramiento del RFF que esté libre y se copia el identificador del registro RRF seleccionado en el campo ***Índice***.

A su vez, el campo ***Ocupado*** del registro seleccionado en el RFF se marca a 1 y el campo ***Válido*** se establece a 0 ya que todavía no se ha realizado la escritura.

La instrucción se está distribuyendo, no se ha llegado a emitir.

El identificador del registro del RRF que ha sido seleccionado se utilizará como identificador de registro destino en la entrada de la estación de reserva individual y se almacenará en la entrada que la instrucción tiene en el buffer de terminación

Así cuando la instrucción sea terminada se sepa qué registro del RRF hay que liberar.

Una vez que una instrucción finaliza su ejecución, se produce la escritura del resultado de la operación en el registro RRF y la colocación del campo ***Válido*** a 1.

Sin embargo, la escritura diferida del valor del RRF al ARF se efectúa cuando la instrucción termina.

Lo que sucederá cuando lo indique el buffer de terminación.

Que puede ser en el siguiente ciclo o muchos ciclos después.

La terminación de las instrucciones se efectúa de forma ordenada puesto que todas las instrucciones al abandonar la etapa de decodificación se insertaron ordenadamente en dos buffers:

- La estación de reserva centralizada
- El buffer de reordenamiento.

Las situaciones que se pueden plantear el realizar el renombramiento en función del estado de la instrucción de escritura que forzó el renombramiento son:

- Instrucción pendiente de escritura.
- Instrucción finalizada.
- Instrucción terminada.

Se explica siguiendo la figura 2.38.

ARF

	Datos	Índice	Ocup
R0	45	2	1
R1	15	1	0
R2	7	0	1
R3			
R15			

RRF

	Datos	Válido	Ocup
Rr0	37	1	1
Rr1	15	1	0
Rr2	10	0	1
Rr3			
Rr7			

Figura 2.38: Organización del RRF, ARF con Acceso Indexado

Instrucción Pendiente de Escritura

El registro R0 del ARF está marcado como ocupado indicando que el registro se encuentra renombrado como consecuencia de una instrucción de

El contenido de ***Datos*** es información no actualizada.

El valor de su índice apunta al registro Rr2 del RRF que es el identificador que estará en las entradas de las estaciones de reserva que corresponden a aquellas instrucciones que tenían R0 como operando fuente y

eran posteriores a la instrucción de escritura que causó el renombramiento de R0.

En el RRF el campo ***Válido*** se encuentra a 0 indicando que el resultado de la operación todavía no se ha obtenido.

El contenido de su campo ***Datos*** tampoco es válido.

El campo ***Ocupado*** está a 1 indicando que la entrada está en uso.

Observe que si ahora una nueva instrucción con R0 como operando fuente se distribuyese a una estación de reserva individual, el identificador R0 se

reemplazaría por el identificador de Rr2 ya que todavía no estaría disponible su valor actualizado.

La instrucción de escritura se encuentra pendiente de emisión o está ejecutándose.

Instrucción Finalizada

El registro R2 está marcado como ocupado lo que indica que una instrucción de escritura lo renombró, utilizando para ello el Rr0 del RRF.

El registro Rr0 tiene su campo ***Válido*** a 1 para señalar que la actualización del registro se ha realizado, lo que es indicativo de la finalización de la ejecución de la instrucción de escritura.

El campo ***Ocupado*** permanece a 1 ya que la actualización del registro R2 del ARF todavía no se ha realizado como consecuencia de que la instrucción de escritura no ha sido terminada.

El valor almacenado en el campo ***Datos*** del R2 no tiene validez ya que está a falta de actualización, es decir, de la escritura del contenido del campo ***Datos*** del Rr0 en el campo ***Datos*** del R2.

Si durante el tiempo que R2 permanece en este estado, una nueva instrucción de lectura con R2 como operando fuente es distribuida, el identificador de R2 será sustituido por el valor de Rr0.

Instrucción Terminada

El registro Rl está marcado como ***no ocupado*** lo que indica que la instrucción de escritura que lo utilizó ya terminó y actualizó el contenido de su campo ***Datos***, deshaciendo el renombramiento.

Aunque el renombramiento de Rl ya no exista, es interesante analizar el estado en que quedó el registro de renombramiento que se le asoció, el Rr1.

Como la instrucción de escritura que lo renombró ya terminó, el contenido del campo ***Datos*** de Rr1 coincide con el de R1.

La indicación de que la instrucción de escritura terminó es que el campo ***Ocupado*** quedó marcado como 0.

El campo ***Válido*** quedó a 1 ya que, previamente, a la terminación, la instrucción finalizó y escribió el resultado en el campo ***Datos*** de Rr1, colocando el bit de válido a l.

Cualquier nueva instrucción que se distribuyese a continuación y tuviese como operando fuente el registro Rl, utilizaría el valor del campo ***Datos*** del ARF.

Organización Independiente del RRF con Acceso Asociativo

Otra forma de organizar el RRF de forma independiente pero introduciendo cambios en su estructura es si se accede de forma asociativa mediante una búsqueda en el RRF del identificador del registro del ARF que provoca el renombramiento.

En este caso el ARF no tiene campo Índice ya que para acceder al registro de renombramiento se utiliza el identificador del registro destino ARF que provoca el renombrado.

ARF

	Datos	Ocup
R0	45	1
R1	15	0
R2	7	1
R3	28	1
R15		

RRF

	Dest	Datos	Válido	Últim	Ocup
Rr0	2	37	1	1	1
Rr1	1	15	1	1	0
Rr2	0	10	0	1	1
Rr3	3	35	1	0	1
Rr4	3	8	0	1	1
Rr7					

Figura 2.39: Organización del RRF como estructura independiente del ARF con Acceso Asociativo

La estructura de los registros RRF tienen cinco campos:

- ***Destino***, se almacena el identificador del registro ARF que provoca el renombramiento y que se utiliza para realizar la búsqueda y validar la coincidencia.

- ***Datos***, se escribe el resultado de la instrucción que ha causado el renombramiento.

- ***Ocupado***, tiene un bit de longitud, comprueba si el registro está siendo utilizado por instrucciones pendientes de ejecución y no pueden liberarse.

- ***Válido***, tiene un bit de longitud, indica que aún no se ha realizado la escritura en el RRF.

- ***Último***, efectúa la labor del campo índice del ARF con acceso indexado, de forma que si está a 1 indica que es el último registro de renombramiento asignado al registro del ARF.

Organización del RRF como parte del Buffer de Reordenamiento

Otro procedimiento para integrar el RRF en el núcleo del procesador es como parte del buffer de reordenamiento o terminación.

El buffer de reordenamiento permite recuperar el orden de las instrucciones y concluir su procesamiento manteniendo la semántica.

Cada instrucción que sale de la etapa de decodificación se almacena simultáneamente en el buffer de distribución y en el buffer de terminación siguiendo el orden secuencial del programa.

De esta forma, tras la etapa de ejecución no se pierde el conocimiento relativo al orden en que deben terminarse arquitectónicamente las instrucciones.

Para incluir el RRF en el buffer de reordenamiento añadimos los campos ***Datos*** y ***Válido*** con la misma función que ya hemos visto.

La diferencia está en el campo ***Índice*** del ARF que contiene un puntero a una de las entradas del buffer de reordenamiento ya que el buffer hace RRF.

Se prescinde del campo ***Ocupado*** ya que este campo existe en el buffer de terminación.

Si se produce un renombramiento de un registro por ser el operando destino de una instrucción, el campo ***Índice*** del registro apuntará a la entrada del buffer de reordenamiento que ocupa la instrucción que renombra el operando destino.

En el instante en que la instrucción termine y se libere su entrada en el buffer de terminación, se accederá con el identificador de esa entrada al ARF y si hubiese coincidencia se actualizará al campo *Datos* y el bit de ***Ocupado*** se establecerá a 0, finalizando el renombramiento.

Al realizar la distribución de una instrucción pueden darse las siguientes situaciones:

- Si el registro no está renombrado, se lee su valor del campo *Datos* del ARF y se utiliza como operando fuente en la estación de reserva.
- Si está renombrado, se sigue el puntero del campo Índice y se accede a una entrada del buffer de reordenamiento, puede ocurrir:

- Si el bit de ***Válido está a 0,*** está a la espera de la finalización de la instrucción que tiene que actualizar el contenido del ***registro de renombramiento***, el campo ***Datos*** de la entrada del buffer.

  Como operando fuente en las estaciones de reserva se utiliza el identificador de la entrada del buffer.

- Si el bit de ***Válido está a 1***, está pendiente de la terminación de la instrucción.

  Como operando fuente para la estación de reserva se utiliza el valor almacenado en el campo Datos del buffer de terminación.

  En el instante en que la instrucción se termina, se actualiza el contenido de ***Datos*** en el ARF con la información almacenada en el buffer.

  El bit de ***Ocupado*** se colocará a 0 tanto en el ARF como en el buffer de reordenamiento para indicar que el renombramiento del registro ha concluido.

La inclusión del RRF en el buffer de terminación añade un campo ***Datos*** y un campo ***Válido*** al buffer de terminación cuando existen instrucciones que no provocan renombramiento de registros.

Lo importante al realizar el diseño del RRF es alcanzar una elevada velocidad de renombramiento.

Así se garantiza que los renombramientos de los registros destino y las lecturas de los registros fuente se realicen en el menor tiempo posible.

Influyendo en el rendimiento de la etapa de distribución, ejecución y terminación.

Para poder distribuir una instrucción son necesarias tres condiciones:

- Una entrada libre en el RRF.
- Una entrada libre en la estación de reserva individual que le corresponda según su tipo.
- Una entrada libre en el ***buffer*** de reordenamiento.

En caso de que falle una de las tres condiciones, el envío de instrucciones desde la estación de reserva centralizada a las estaciones individuales se detiene.

Por ello, es fundamental que el diseño del RRF sea el más adecuado para las características del procesador.

Terminación

Una vez que las instrucciones han completado su etapa de ejecución, se quedan almacenadas en el buffer de reordenamiento a la espera de su terminación arquitectónica.

Una instrucción se considera terminada arquitectónicamente cuando actualiza el estado del procesador manteniendo la ***consistencia***.

Existe ***consistencia del procesador*** cuando las instrucciones concluyen su procesamiento en el mismo orden secuencial en el que se encuentran en el

programa, o lo que es lo mismo en el mismo orden en el que iniciaron el procesamiento.

Mantener la consistencia es fundamental por dos motivos:

- Para garantizar el resultado final del programa, tenemos que respetar la semántica del programa que viene definida por el orden secuencial de las instrucciones.
- Para permitir un tratamiento correcto de las interrupciones, tenemos que conocer el estado del procesador antes de procesar una información.

Estar a la espera de la terminación arquitectónica es estar a la espera de copiar en un registro arquitectónico el resultado temporal almacenado en un registro de renombramiento.

Las instrucciones de almacenamiento necesitan pasar por la etapa de retirada, este es el momento en que escriben sus resultados ordenados en la D-caché.

Una instrucción ha ***finalizado*** cuando abandona la unidad funcional y queda a la espera en el buffer de terminación.

Una instrucción ha ***terminado*** cuando ha actualizado el estado de la máquina. Solamente se pueden terminar las instrucciones que han finalizado y no están marcadas como especulativas o como inválidas en el buffer de terminación.

Una instrucción se ha ***retirado*** cuando ha escrito su resultado en memoria. Este estado sólo pueden alcanzarlo las instrucciones de almacenamiento.

La pieza clave que garantiza la consistencia del procesador es el buffer de reordenamiento o terminación ya que gestiona la terminación ordenada de todas las instrucciones que están en vuelo.

La terminación ordenada asegura que las interrupciones se puedan tratar correctamente.

Cuando una instrucción fuerce una excepción, el buffer de reordenamiento permitirá expulsar del cauce todas las instrucciones posteriores para garantizar la terminación ordenada de las anteriores.

El *buffer de reordenamiento* pone fin a la ejecución fuera de orden de las instrucciones y forma el *back-end* de un procesador superescalar junto con la etapa de retirada.

Este buffer decide cuándo los resultados almacenados temporalmente en el RRF se tienen que escribir en el ARF y puede darse por concluida una instrucción.

El *buffer de reordenamiento* es una estructura que mantiene entradas con el estado de todas y cada una de las instrucciones que hay en vuelo.

La composición de una entrada del buffer de reordenamiento, si tenemos en cuenta que el RRF es una estructura independiente, con los campos más habituales.

Son los siguientes:

- Ocupada (O).
- Emitida (E).
- Finalizada (F).
- Dirección (Dir).
- Registro de destino (Rd).
- Registro de renombramiento (Rr).
- Especulativa (Es).
- Validez (V).

En el caso que el RRF fuese parte del buffer de terminación hay que añadir los campos:

- Datos (D)
- Validez de datos (Vdatos)

Y eliminar el campo Rr, ya que la propia entrada haría de registro de renombramiento.

Si unimos los campos *Ocupada*, *Emitida* y *Finalizada* en uno de tres bits denominado Estado, una instrucción pasará por las siguientes situaciones:

Estado = 100: Instrucción en espera de emisión.

Estado = 110: Instrucción en ejecución.

Estado = 111: Instrucción pendiente de terminación.

Para que una instrucción pueda distribuirse son necesarias tres condiciones:

- Una entrada libre en el RRF,
- Una entrada libre en la estación de reserva individual
- Una entrada libre en el buffer de reordenamiento.

Cuando una ***instrucción es terminada arquitectónicamente*** ocurre lo siguiente:

- Su registro de renombramiento asociado Rr se libera.
- El registro de destino Rd se actualiza con el valor de su registro de renombramiento Rr asociado y se libera si no tiene renombramientos pendientes.
- El campo Ocupado se fija de nuevo a 0.
- El puntero de cola se incrementa para apuntar a la siguiente instrucción a terminar.

El número de instrucciones que pueden terminarse simultáneamente depende de la capacidad del procesador para transferir información desde el RRF al ARF o desde el buffer de terminación al ARF.

El ancho de banda se determina por el número de puertos de escritura del ARF y la capacidad de enrutamiento de datos hacia el ARF.

Tendremos que tener tantos puertos de escritura en el ARF como instrucciones se quieran terminar por ciclo de reloj.

La velocidad pico del procesador es:

- V_{pico} = Máx. instrucciones terminadas por ciclo * Frecuencia del procesador.

Retirada

La etapa de retirada es exclusiva de las ***instrucciones de almacenamiento,*** y es donde estas realizan el acceso a memoria para la escritura de sus resultados.

El objetivo de esta etapa es garantizar la consistencia de memoria, es decir, que las instrucciones de almacenamiento se completen en el orden establecido por el programa, debemos evitar los riesgos WAW y WAR.

El mecanismo que se utiliza en esta etapa para lograrlo es el ***buffer de almacenamiento*** o ***buffer de retirada*** que consta de dos partes:

- Finalización.
- Terminación.

El camino que siguen las instrucciones de carga/almacenamiento en una segmentación superescalar una vez que han sido emitidas consta de tres pasos:

- Cálculo de la dirección de memoria.
- Traducción de la dirección de memoria virtual en memoria física.
- Acceso a memoria.

La sintaxis de las instrucciones de carga y almacenamiento que emplean direccionamiento basado en el desplazamiento es:

LD registro_destino, desplazamiento(registro_base)

SD desplazamiento (registro_base), registro_fuente.

El primer paso en la generación de la dirección de memoria, implica acceder al registro base para leer su valor y sumar este valor de desplazamiento.

Una vez que se dispone de la dirección de memoria hay que realizar la traducción de memoria virtual a memoria principal.

La traducción se realiza consultando la **TLB** (*Translation Lookaside Buffer*), buffer de traducción adelantada, que es donde se mantienen las equivalencias entre direcciones virtuales y direcciones físicas.

Si al realizar el acceso a la TLB, la dirección virtual se encuentra mapeada en la memoria física la TLB

devuelve la dirección física y el acceso se realizará con normalidad.

Si la dirección virtual no se encuentra en la memoria principal, se produce un fallo de página que da pie al levantamiento de una *excepción de fallo de página*.

Recuperada la página y el estado del procesador previo a la instrucción que provocó la excepción, la instrucción de carga/almacenamiento puede volver a emitirse sin que se produzca fallo de página.

El paso final de una instrucción de ***carga/almacenamiento*** es la ***lectura/escritura*** en memoria, es en este paso cuando el procesamiento de las instrucciones de carga y almacenamiento difiere.

Las instrucciones de carga realizan los tres pasos en la etapa de ejecución, en cambio las instrucciones de almacenamiento dejan el acceso a memoria para la etapa de retirada.

Las ***instrucciones de almacenamiento*** acceden a la memoria en la *etapa de retirada* para dar mayor

prioridad al acceso de las instrucciones de carga a la D-caché.

Cuando finaliza una instrucción de almacenamiento el dato a escribir y la dirección de memoria se guardan en la entrada que se asigna a la instrucción en el buffer de almacenamiento y en esa entrada se marca como finalizada.

La actualización del buffer de almacenamiento se realiza al mismo tiempo que la actualización del buffer de reordenamiento.

Cuando la instrucción es terminada por mandato del buffer de reordenamiento, la instrucción de almacenamiento se marca en el buffer de almacenamiento como terminada, quedando lista para escribir en memoria en cuanto el bus de acceso esté libre.

La ***escritura diferida*** por parte de una instrucción de almacenamiento evita una actualización errónea y precipitada de la memoria en el caso de que una instrucción sufra una interrupción.

La ***escritura diferida*** permite dar mayor prioridad a las instrucciones de carga para acceder a la D-caché.

Una *instrucción de almacenamiento* puede estar marcada como especulativa en el buffer de reordenamiento, en el caso que se produzca una predicción incorrecta la instrucción de almacenamiento se invalida en el buffer de reordenamiento y se purga el buffer de almacenamiento.

Como nunca se pueden terminar las instrucciones especulativas, no existe el problema de que existan instrucciones marcadas como terminadas en el buffer de almacenamiento.

Si están marcadas como terminadas no son especulativas.

Los ***riesgos de memoria WAW*** se evitan mediante el buffer de almacenamiento que asegura que las escrituras en la D-caché se realizan respetando el orden en que aparecen en el programa.

Los ***riesgos de memoria WAR*** se evitan gracias a que una instrucción de almacenamiento posterior a una carga en el orden del programa nunca podrá escribir antes que la carga ya que lo evita la escritura diferida.

Mejoras en el Procesamiento de las Instrucciones de carga/almacenamiento

La ***lectura adelantada*** del dato por la carga permite que todas las instrucciones aritmético-lógicas dependientes de ella, directa o indirectamente, tengan el camino libre para su ejecución.

Otro mecanismo que mejora el rendimiento del procesador es el ***reenvío de datos*** desde una instrucción de almacenamiento hacia una de carga que tengan operandos destino y fuente comunes.

Se permite que la carga no necesite acceder a la D-caché para su lectura ya que el almacenamiento le reenviará el dato directamente.

Tanto la terminación adelantada de las cargas como el reenvío son dos técnicas que se pueden aplicar simultáneamente.

Reenvío de datos entre instrucciones de almacenamiento y de carga

El reenvío se puede realizar cuando existe una dependencia de memoria RAW entre una instrucción de almacenamiento (productora) y una instrucción de carga (consumidora).

Para realizar el reenvío es necesario comprobar que las direcciones de memoria de una instrucción de carga y de los almacenamientos pendientes sean coincidentes.

El dato que necesita la carga está en un registro del procesador y no hay que ir a la memoria para obtenerlo.

Los riesgos de datos RAW se resuelven en las estaciones de reserva pero los riesgos de memoria RAW todavía no han sido tratados y son los que hay que analizar para reenviar datos y adelantar cargas.

La coincidencia de direcciones no es posible conocerla cuando las instrucciones se encuentran en la estación de reserva, la comprobación se realiza

cuando las instrucciones carga y almacenamiento han sido emitidas y finalizadas/terminadas.

En el proceso de búsqueda de direcciones coincidentes pueden plantearse dos situaciones:

- Hay coincidencia, hay una dependencia de memoria RAW.
- No hay coincidencia, no hay dependencia de memoria RAW, se realiza el acceso a caché de forma normal.

Puede surgir el problema de una aparición de una carga que presenta varias coincidencias en el buffer de almacenamiento, en este caso el procesador debe contar con el hardware adecuado para saber cuál es el almacenamiento más reciente.

Necesitamos mantener los almacenamientos ordenados según su orden de llegada y utilizar codificadores con prioridades para utilizar la dirección más reciente en caso de múltiples coincidencias.

Otra cuestión importante es el número de puertos necesarios en el buffer de almacenamiento para la lectura de datos.

Hasta ahora con un único puerto en el buffer de almacenamiento para la lectura de datos era suficiente.

Para evitar bloqueos es necesario un segundo puerto de lectura para garantizar que en un mismo ciclo se pueda leer y enviar datos a la caché.

Terminación Adelantada de las Instrucciones de Carga

Adelantar la ejecución de una instrucción de carga frente a varias instrucciones de almacenamiento es más complicado ya que es necesario comprobar que las direcciones de memoria que manejan no son iguales.

Tenemos que comprobar la ausencia de dependencia de memoria RAW teniendo en cuenta la problemática de las *dependencias ambiguas*.

Vamos a suponer que las instrucciones de carga/almacenamiento se emiten en orden por la estación de reserva a las unidades funcionales de carga y almacenamiento.

La carga, mientras que realiza el acceso a la caché de datos puede comprobar la coincidencia de su dirección en el campo *Dirección* de las entradas del buffer de almacenamiento.

Tenemos dos posibles situaciones:

- Hay coincidencia de direcciones, hay dependencia de memoria RAW. No se puede adelantar la carga, se utiliza el dato del almacenamiento coincidente.
- No hay coincidencia de direcciones, no hay dependencia de memoria RAW.

La ventaja de la emisión ordenada de las instrucciones de carga/almacenamiento es que reduce la complejidad del hardware, imprescindible para resolver las dependencias ambiguas.

El inconveniente que reduce el rendimiento.

El empleo de un ***buffer de cargas finalizadas***, es una técnica utilizada para realizar el adelantamiento de cargas permitiendo una emisión desordenada de las cargas y de los almacenamientos.

Esta técnica es una estructura que posibilita que las instrucciones de carga se emitan de forma especulativa sin resolver las dependencias ambiguas que puedan existir con almacenamientos pendientes de ejecución pero previos a la carga en el orden del programa.

Se pueden plantear dos situaciones:

- ***No hay coincidencia***, se permite que la carga continúe normalmente ya que no hay ningún tipo de dependencia de memoria RAW.
- ***Hay coincidencia***, hay dependencia de memoria RAW con un almacenamiento, se anula la carga y todas las instrucciones posteriores para su emisión posterior, no se reenvían los datos ya que pueden quedar almacenamientos anteriores pendientes de ejecución.

Tras finalizar las cargas quedan almacenadas en el buffer de cargas finalizadas, quedan pendientes los almacenamientos que no conocen sus direcciones de acceso a memoria.

Cuando una instrucción de almacenamiento termina tiene que comprobar la existencia de una coincidencia con las cargas que estén en el buffer de cargas finalizadas.

Se pueden plantear dos situaciones:

- ***Hay coincidencia***, la instrucción de carga que depende del almacenamiento se ha ejecutado de forma especulativa. Se ha violado una dependencia de memoria RAW.
- ***No hay coincidencia***, la carga y todas las instrucciones posteriores a ella pueden terminarse ya que no hay ningún tipo de dependencia entre el almacenamiento y la carga.

Tratamiento de Interrupciones

El tratamiento de las interrupciones en un procesador superescalar es más complejo que en un procesador de un único cauce, esto es debido a que la ejecución fuera de orden puede modificar el estado de un proceso en un orden diferente.

Cuando se produce una interrupción, el estado del proceso interrumpido se guarda por el hardware, el software o una combinación de ambos.

El estado del procesador está definido por:

- El contador de programa.
- Los registros arquitectónicos.
- El espacio de memoria asignado.

La precisión de excepción es cuando un procesador es capaz de tratar las interrupciones de forma que se pueda reanudar la ejecución de un programa en las mismas condiciones que había antes de producirse la interrupción.

Es necesario que el procesador mantenga su estado y el de la memoria de forma análoga como si las instrucciones se ejecutasen en orden.

El estado del procesador debe de evolucionar secuencialmente cuando una instrucción termina aunque se haya ejecutado fuera de orden.

Las interrupciones se clasifican en dos grandes grupos:

- ***Interrupciones externas***, se producen por eventos externos a la ejecución del programa.
- ***Excepciones, interrupciones de programa o traps***, suelen producirse en la etapa IF o por las instrucciones desprograma en ejecución.

Las técnicas para el tratamiento de instrucciones que permite la ejecución fuera de orden se pueden agrupar en cuatro categorías:

- Ignorar el problema y no garantizar la precisión de la excepción.
- Permitir que las interrupciones sean algo imprecisas.
- Permitir que una instrucción sea emitida solo cuando se está seguro de que las instrucciones que la preceden terminarán sin causar ninguna interrupción.

- Mantener los resultados de las operaciones en un buffer de almacenamiento temporal hasta que todas las operaciones previas hayan terminado correctamente y se puedan actualizar los registros arquitectónicos sin ningún tipo de riesgo.

Para lograr la precisión de excepción se diseñaron dos técnicas:

- ***El buffer de historia***, es similar al buffer de reordenamiento ya que permite una terminación desordenada de las instrucciones. En sus entradas se almacena el valor de los registros destino al comienzo de la ejecución de las instrucciones (Campo Vp), no al finalizar.
- ***El fichero de futuro***, es una técnica que combina este fichero con el buffer de reordenamiento. Los operandos fuente se leen

del fichero de futuro y al finalizar las instrucciones los resultados se escriben en dos ubicaciones, en el fichero de fututo y en el buffer de terminación.

Las técnicas basadas en el buffer de historia y en el registro de futuro proporcionan precisión de excepción retrasando el almacenamiento definitivo de los resultados mientras que las instrucciones no sean terminadas arquitectónicamente y dadas por válidas.

Excepciones Precisas con Buffer de Reordenamiento

El elemento clave que permite mantener la consistencia del procesador es el ***buffer de reordenamiento*** o ***buffer de terminación***, ya que posibilita que las instrucciones no interrumpidas terminen en el mismo orden en que se sitúan en el programa, almacenando sus resultados definitivos de forma ordenada.

Para saber si una instrucción ha sido interrumpida durante la etapa de ejecución, las entradas del buffer de reordenamiento cuentan con un campo *Int*.

Este campo está a 0 y pasa a valer 1 para indicar que ha sido interrumpida la instrucción.

Cuando no es una instrucción del programa durante la ejecución la que lanza la interrupción es posible realizar un tratamiento del problema diferente en función de dónde suceda la interrupción.

Si la interrupción ocurre en la etapa IF por un fallo de página de memoria virtual al intentar leer una instrucción, no se leen nuevas instrucciones y se terminarían todas las instrucciones ya existentes en el cauce.

Si se produce en la etapa de decodificación debido a un código de operación ilegal o indefinido, se terminaría la ejecución de las instrucciones alojadas en el buffer de distribución y siguientes.

Limitaciones de los Procesadores Superescalares

Hay dos factores que limitan el rendimiento de los procesadores superescalares:

- El grado de paralelismo intrínseco en el flujo de instrucciones del programa.
- La complejidad y el coste de la etapa de distribución.

En la actualidad se diferencian dos grandes líneas de trabajo en el diseño de procesadores:

- Una enfocada a aumentar el ***paralelismo a nivel de instrucción*** (ILP).
  - VLIW (Very Long Instruction Word).
  - EPIC (Explicitly Parallel Instruction Computing)
- Otra orientada a incrementar el ***paralelismo a nivel de hilo*** (Thread-Level Paralelism – TLP).

- Una tercera sería el paralelismo a nivel de proceso, pero se considera más orientada al desarrollo de multiprocesadores.

La tendencia actual para mejorar las prestaciones de un procesador superescalar pasa por su escalabilidad:

- Aumento del ancho y profundidad de las segmentaciones.
- Aumento del tamaño y niveles de la memoria caché.
- Aumento de la frecuencia de reloj.
- Técnicas avanzadas especulativas que mejoran el flujo de control.

- Otra mejora es el procesamiento vectorial, incluyen dentro del procesador una o varias unidades funcionales para procesar instrucciones SIMD.

Para aumentar el paralelismo a nivel de hilo se trabaja en procesadores:

- **SMT** (*Simultaneous MultiThreading*). Permite la ejecución en paralelo de instrucciones pertenecientes a hilos distintos.
- **CMT** (*Chip Multiprocessors*). Coloca varios procesadores similares en un único circuito integrado.