Ingeniería de Computadores II

TEMA II
PROCESADORES SUPERESCALARES

Fuente imágenes: Prof. Morillo

2.1. Características de los procesadores superescalares.
2.2. Arquitectura de un procesador superescalar genérico.
2.3. Lectura de instrucciones.
2.4. Decodificación.
2.5. Distribución.
2.6. Terminación.
2.7. Retirada.
2.8. Mejoras en el procesamiento de las instrucciones de carga/almacenamiento.
2.9. Tratamiento de interrupciones.
2.10. Limitaciones de los procesadores superescalares

Diferencias con la segmentacion clasica

- La principal diferencia de una segmentación superescalar con una clásica es
  - Por las etapas de la segmentación pueden avanzar varias instrucciones simultáneamente
    - Esto implica la existencia de varias unidades funcionales para que sea posible.
  - Las instrucciones se pueden ejecutar fuera de orden
  - Una segmentación más ancha.
- Las segmentaciones superescalares se caracterizan por tres atributos
  - Paralelismo
  - Diversificación
  - Dinamismo

2.1. Características de los procesadores superescalares

- Paralelismo
  - Paralelismo de máquina temporal
    - En un mismo instante de tiempo se ejecutan varias instrucciones pero en diferentes etapas
    - Procesadores segmentados (Máquina segmentada clásica)
  - Paralelismo espacial
    - Varias instrucciones se procesan simultáneamente en una misma etapa
      - Por replicación del hardware
    - Ancho o grado de la segmentación
      - El número de instrucciones que se pueden procesar en la misma etapa
- En un procesador superescalar se dan al mismo tiempo el paralelismo de máquina temporal y el espacial
  - Los procesadores superescalares incluyen en la etapa de ejecución múltiples unidades funcionales diferentes e independientes.

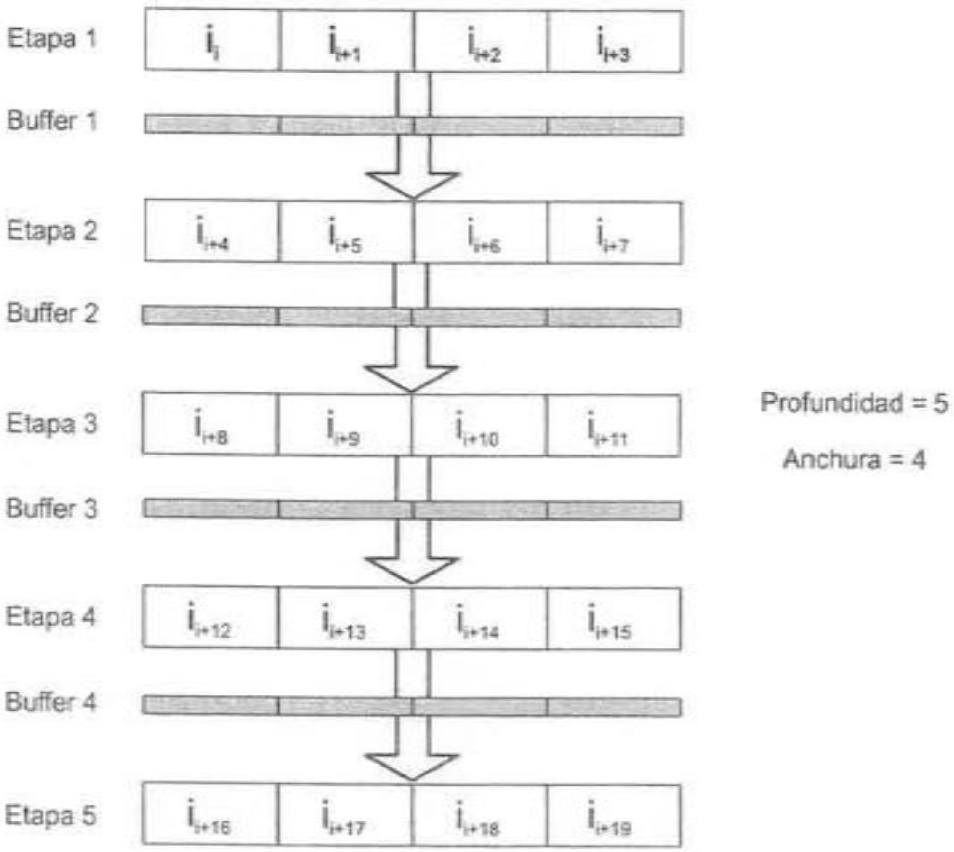


Figura 2.1: Esquema de una segmentación de profundidad 5 y anchura 4. En condiciones ideales pueden concluir su procesamiento de forma simultánea hasta 4 instrucciones por ciclo.

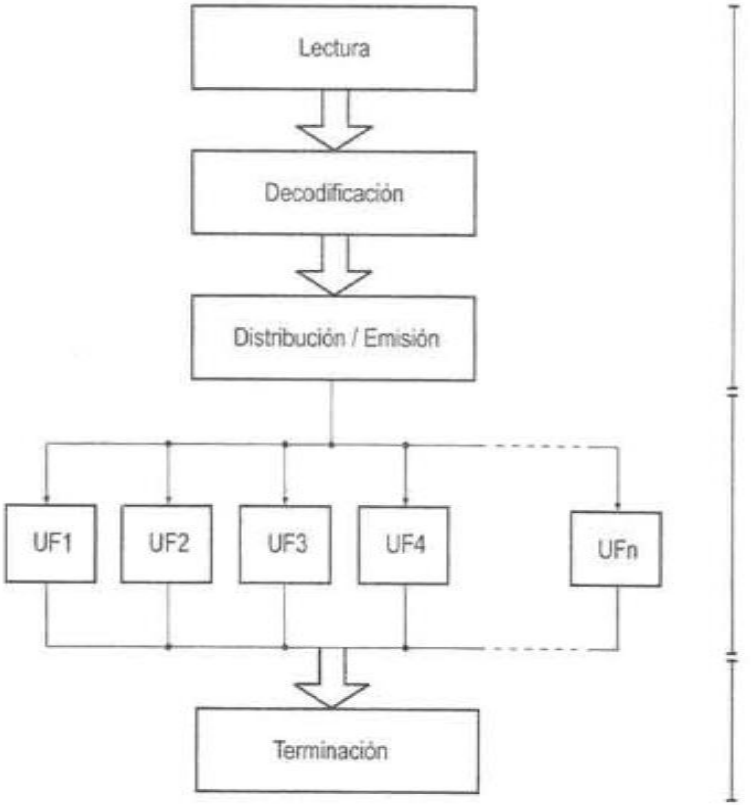
Diversificación

- Debido a los diferentes tipos de instrucciones es insuficiente disponer de un único tipo de cauce de ejecución
- Hay instrucciones que no tiene que pasar por todas las etapas
- La etapa de ejecución posee múltiples unidades funcionales diferentes e independientes

Dinamismo

- Permiten la ejecución de instrucciones fuera de orden
- Cuenta con los mecanismos necesarios para garantizar que se obtengan los mismos resultados,
  - Se respeta la semántica del código fuente.
- Una **segmentación dinámica paralela** utiliza buffers multi entrada 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 (WAR y WAW), evitando ciclos de detención innecesarios.
    - Una reducción de ciclos de detención por dependencias verdaderas de datos y memoria.
- Tienen la capacidad para especular
  - Pueden realizar predicciones sobre las instrucciones que se ejecutarán tras una instrucción de salto.

Esquema procesador superescalar



Ejecución en orden

Ejecución fuera de orden

Ejecución en orden

Figura 2.2: Esquema de bloques de un procesador superescalar segmentado con ejecución fuera de orden.

2.2. Arquitectura de un procesador superescalar genérico

- Consta de seis etapas
  - Lectura (IF – Instruction Fetch)
  - Decodificación (ID – Instruction Decoding)
  - Distribución/emisión (II – Instruction Issue)
  - Ejecución (EX – execution)
  - Terminación (WR– Write-Back Result)
  - Retirada (RI – Retirement Instructions)
    - Comparación con las etapas del procesador segmentado
      - IF (Instruction Fetch): Lectura de la instrucción
      - ID (Instruction Decoding) Decodificación y lectura de los operandos
      - EX (Execution)
      - MEM (Memory Access) Acceso a la cache de datos
      - WB (Write-Back results) Escritura en el fichero de registros

Segmentación Superescalar Genérica

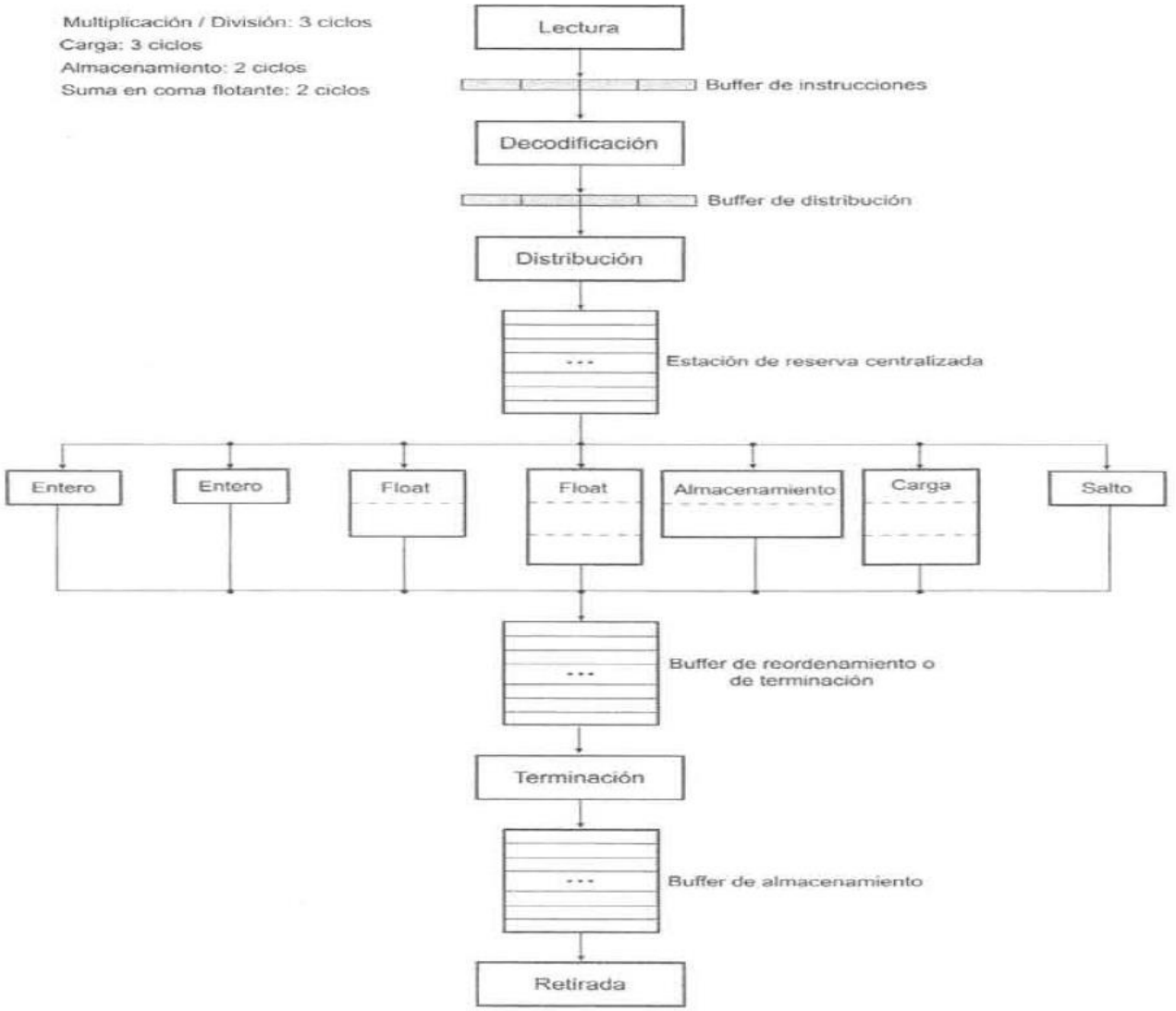


Figura 2.3: Ejemplo de segmentación superescalar genérica.

Estados por los que pasa una instrucción

- 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 registros 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.

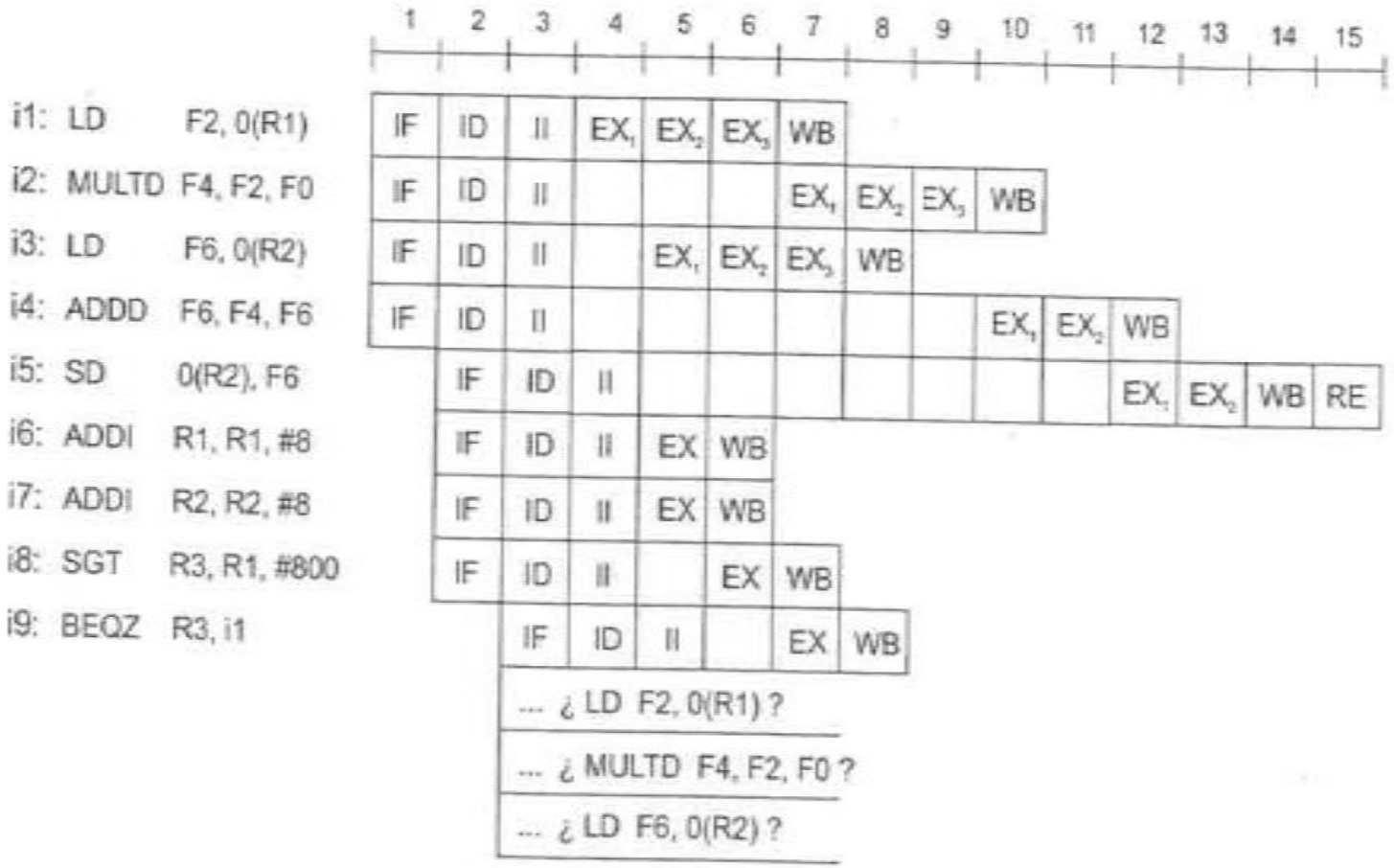


Figura 2.4: Secuencia de la ejecución del bucle DAXPY en la segmentación superescalar de 6 etapas.

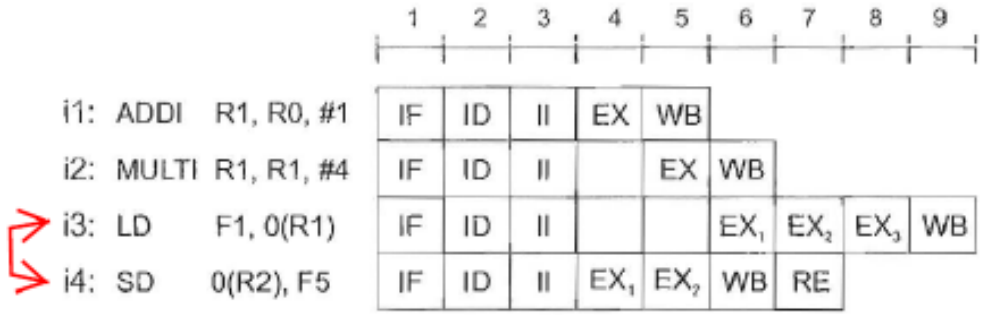


Figura 2.5: Ejecución de una secuencia de instrucciones en la segmentación superescalar de 6 etapas

- Se observa que se puede haber producido la violación de una dependencia WAR ya que el almacenamiento de i4 ha escrito en memoria antes que la instrucción de carga haya realizado la lectura.
- La violación de la dependencia WAR se habrá producido si el contenido de R1 y R2 coinciden.
- Una solución inmediata para forzar la terminación ordenada de las instrucciones es la introducción de ciclos de detención.

Terminación ordenada de instrucciones mediante la introducción de ciclos de detención.

Riesgo WAR y WAW

- Los riesgos WAR y WAW
  - La inserción de ciclos de reloj para forzar la terminación ordenada de las instrucciones y respetar la semántica del programa es conservadora y no permite aprovechar todo el paralelismo que brinda la existencia de múltiples unidades funcionales así como evitar detenciones innecesarias en las unidades.

En el ejemplo no sería necesario detener las instrucciones i3 e i4 hasta que la i2 leyese el contenido de R1 si se reemplazase el registro R1 en las instrucciones i1 e i2 por otro no utilizado.

```
i1: ADD   R1,R1,R2
i2: MULTI R3,R1,#1
i3: ADDI  R1,R2,#1
i4: MULTI R5,R1,#8
```

- Se recurre a un almacenamiento temporal en el que se realiza la escritura que tiene riesgo

Resolución de las dependencias de datos WAR y WAW (falsas)

- Utilizamos una técnica que se llama **Renombramiento Dinámico de Registros**
  - Consiste en utilizar un conjunto de registros ocultos al programador en los que se realizan los almacenamientos temporales.
- La técnica consta de dos pasos
  - 1.- 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.

```
i1: ADD   R1,R1,R2
i2: MULTI R3,R1,#1
i3: ADDI  R1,R2,#1
i4: MULTI R5,R1,#8
```

```
ADD   Rr1,R1,R2
MULTI Rr3,R1,#1
ADDI  Rr2,R2,#1
MULTI Rr5,R1,#8
```

  - 2.- Mantenimiento de las dependencias RAW
    - Se renombran los registros arquitectónicos fuente que corresponden a una escritura previa,
      - El objetivo es mantener las dependencias RAW

```
ADD   Rr1,R1,R2
MULTI Rr3,Rr1,#1
ADDI  Rr2,R2,#1
MULTI Rr5,Rr2,#8
```

Una vez resueltas las dependencias se procede a la fase de terminación a deshacer el renombramiento y a actualizar ordenadamente los registros.

Dependencias entre instrucciones de carga/almacenamiento

- Una carga seguida de un almacenamiento produce una dependencia RAW.

```
SD 0(R1),R5
LD R5,0(R1)
```

- Un almacenamiento seguida de una carga produce una dependencia WAR.

```
LD R3,0(R1)
SD 0(R1),R5
```

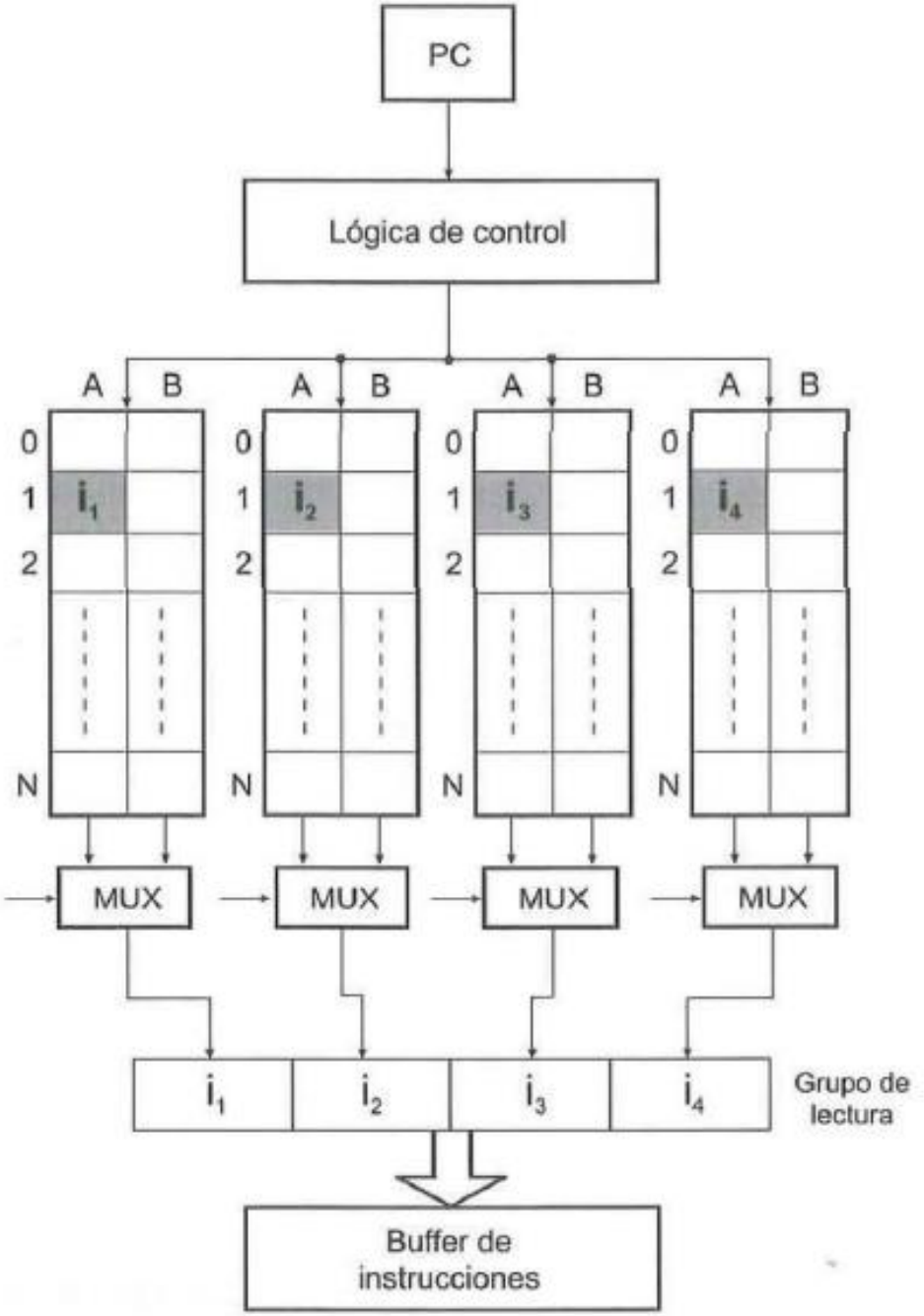
- Dos almacenamientos seguidos implican una dependencia WAW.

```
SD 0(R1),R3
SD 0(R1),R5
```

- 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**.
- Para resolver las dependencias RAW usamos **adelantamiento de los operandos** (Igual que las operaciones con registros)
  - 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
    - Es el sitio en donde se realiza el seguimiento de una instrucción desde que se distribuye hasta que se termina.

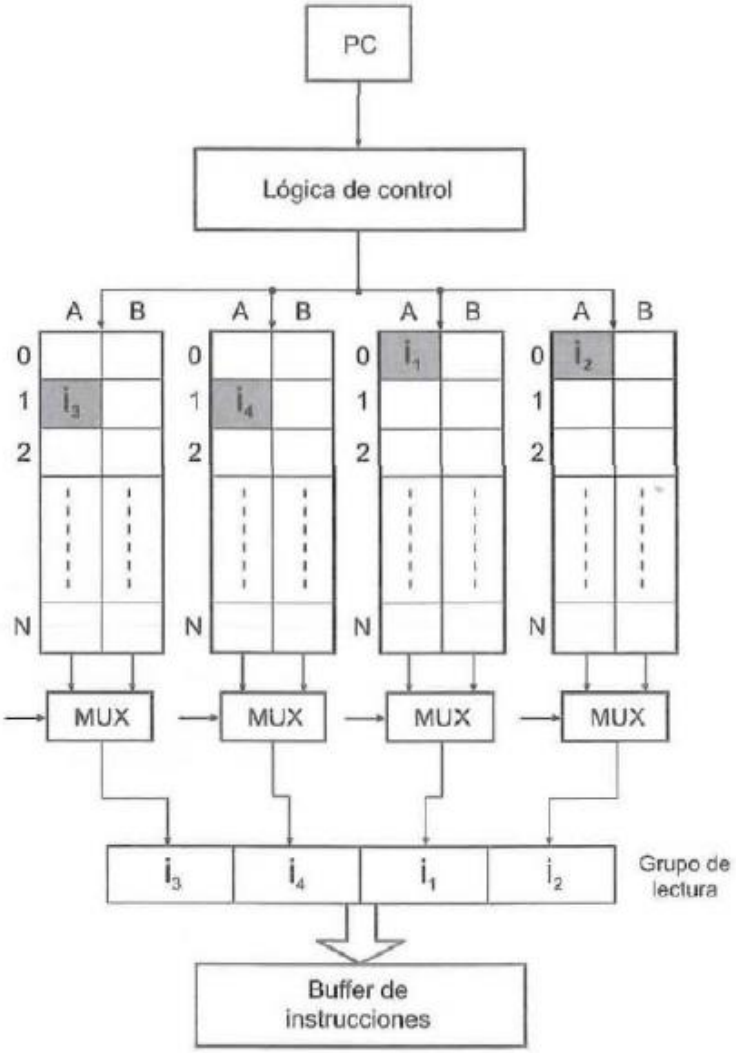
2.5 Lectura de instrucciones

- ¿Cuál es la diferencia, En la etapa de lectura, entre un procesador superescalar y uno escalar ?
  - El número de instrucciones que se extraen de la caché de instrucciones en cada ciclo.
    - En un procesador escalar es una
    - En uno 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.
    - Grupo de lectura
      - El conjunto de instrucciones que se extrae simultáneamente de la I-caché
  - El objetivo de la fase de lectura es garantizar el suministro constante de instrucciones al procesador
  - 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.


PC
Lógica de control
A
B
0
1
2
N
i_1
i_2
i_3
i_4
MUX
Grupo de lectura
Buffer de instrucciones

2.5.1 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.
  - Esto nos lleva a realizar dos accesos a la caché ya que el grupo de lectura ocupa dos bloques consecutivos.
- Las máquinas se diseñan con ciertas restricciones de alineamiento para evitar problemas de lectura.
- Si no hay restricciones de alineación, una solución es recurrir a hardware adicional se procede a la colocación correcta de instrucciones
- Soluciones
  - **La red de alineamiento**
    - Consiste en reubicar las salidas de la caché median multiplexores que conducen las instrucciones leída a su posición correcta dentro del grupo de lectura.
  - **La red de desplazamiento**
    - Recurre a registros de desplazamiento para mover l instrucciones.
  - **Disponer de una cola de prefetching o prelectura**
    - Es una técnica utilizada para minimizar el impacto los fallos de lectura en la caché de instrucciones.



Soluciones:

- Multiplexor
- Registro de desplazamientos

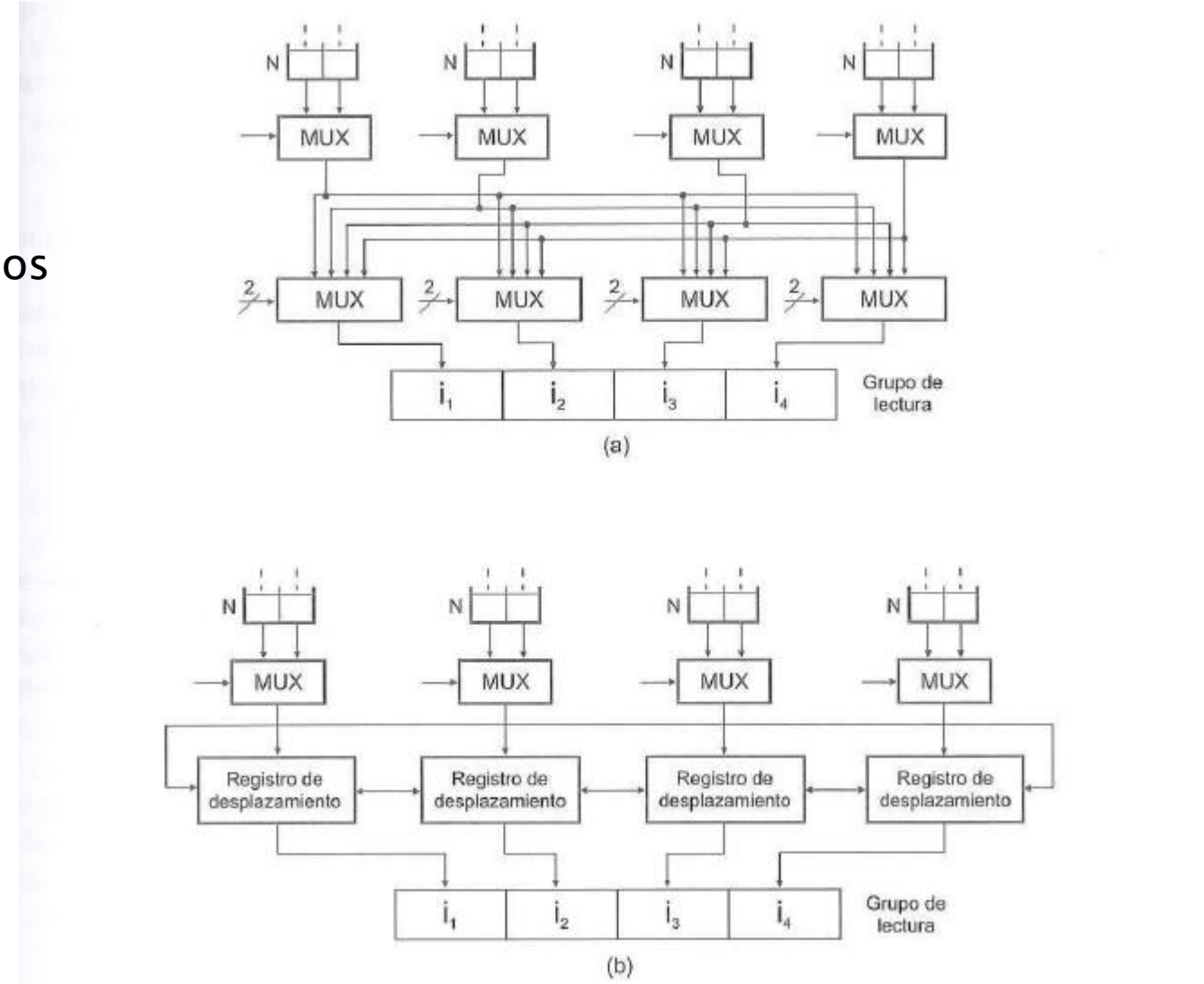


Figura 2.10: Red de alineamiento basada en multiplexores (a) y basada en registros de desplazamiento (b).

2.5.2 Rotura de la secuencialidad en el flujo de instrucciones que forman un grupo de lecturas

- Puede suceder que una de las instrucciones que forme parte de un grupo de lectura sea **un salto incondicional o un salto condicional efectivo**
  - Esto provoca que las instrucciones posteriores del grupo sean inválidas, reduciéndose el ancho de banda de lectura.
- **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, como el producto entre el ancho de la segmentación y el número de ciclos de penalización
    - En una segmentación de ancho **s**, cada ciclo de parada equivale a no emitir **s** instrucciones (o a leer **s** instrucciones de no operación, NOP).

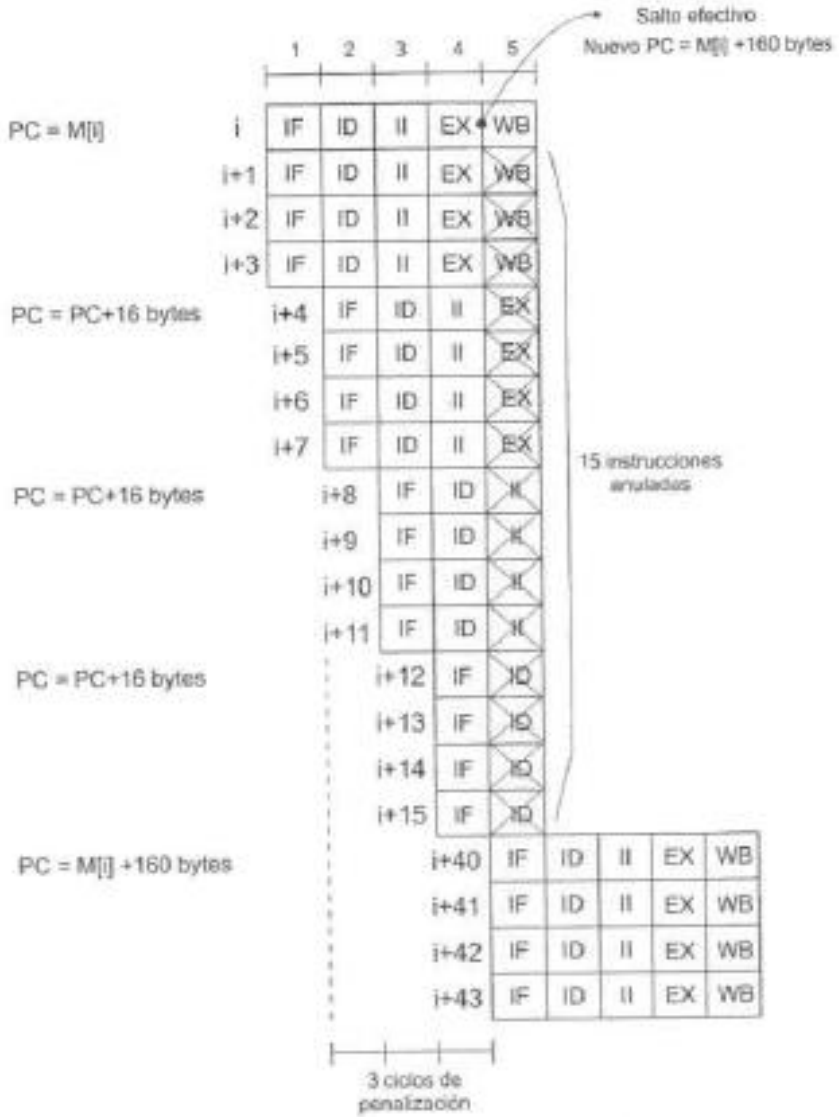


Figura 2.11: El nuevo valor del PC generado por la instrucción de salto en la etapa EX obliga a vaciar el cauce y proceder a la extracción del nuevo grupo de lectura.

2.5.3 Tratamiento de los saltos

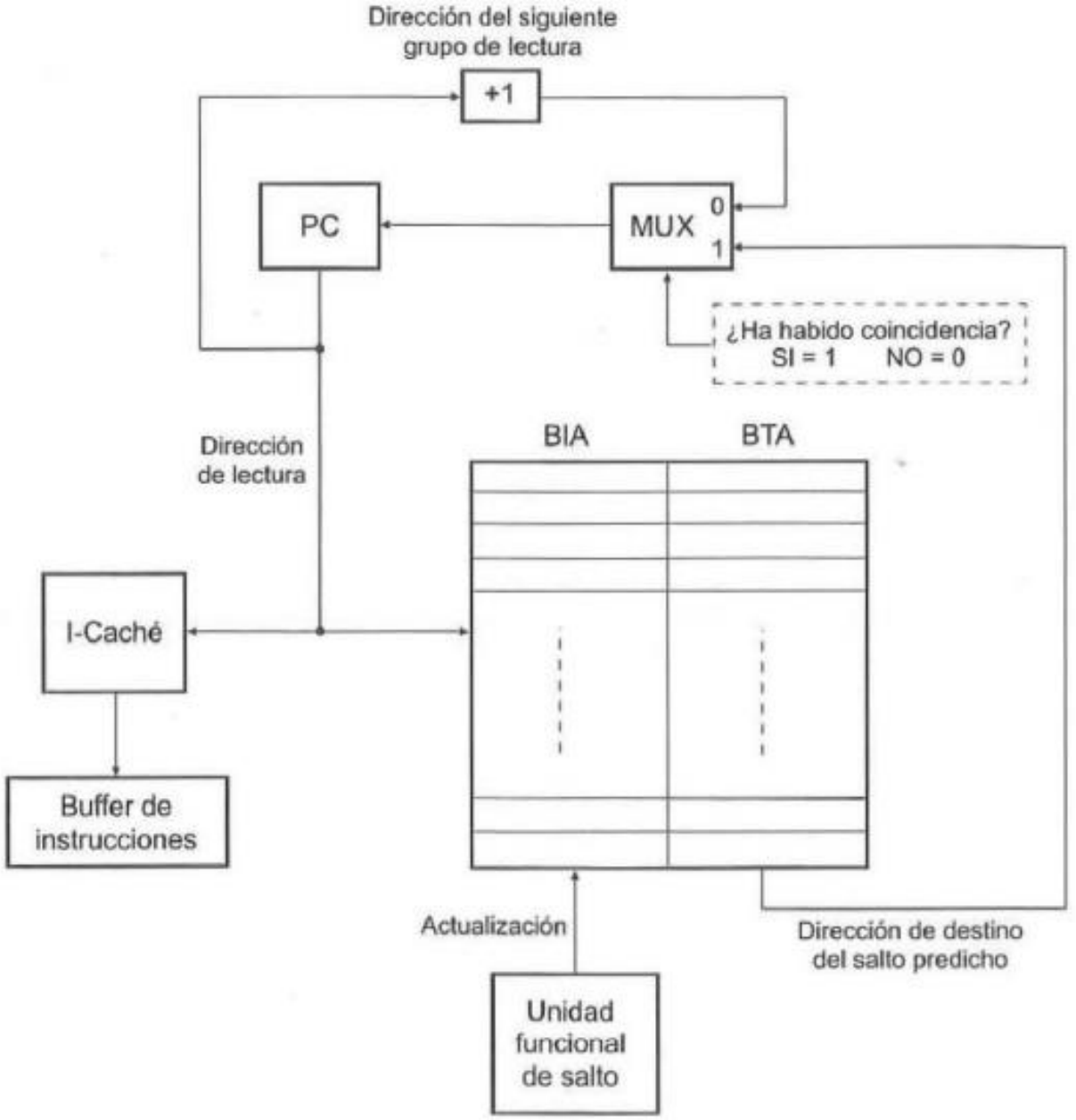
- El principal problema es conocer el resultado de la evaluación de la condición que determina:
  - Si el salto es efectivo o no y
  - La dirección de destino, en caso de que sea efectivo
- La técnica que se emplea en los procesadores superescalares para tratar las instrucciones de salto
  - **Es especular** (realizar predicciones)
- El procesamiento de los saltos se realiza en una unidad funcional específica
  - Si el predictor señala el salto como efectivo, se procede a leer de la I-caché la instrucción de destino y siguientes

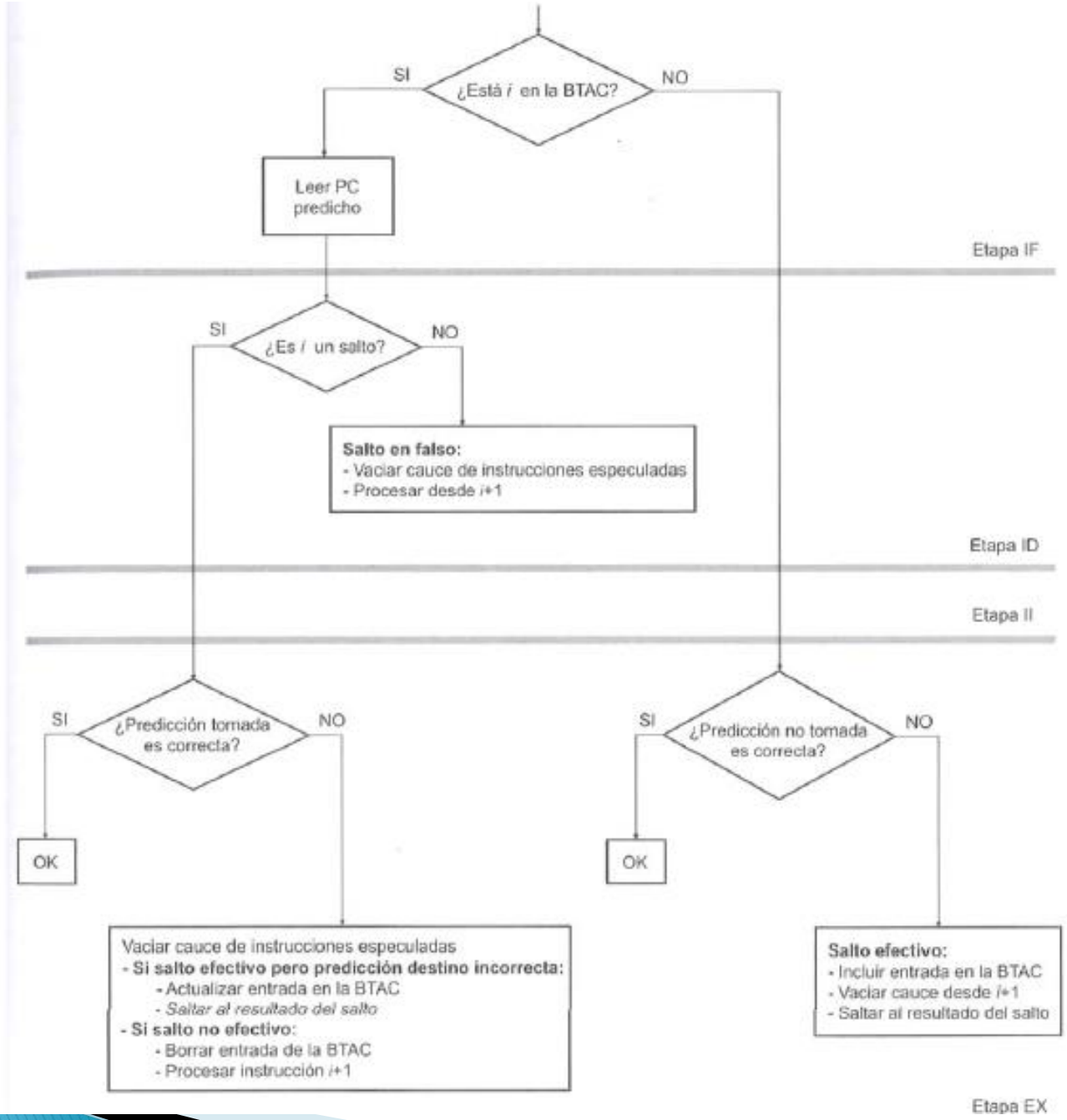
2.5.4 Estrategias de predicción dinámica

- **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.

2.5.4.1. Predicción de la dirección de destino de salto mediante BTAC (*Branch Target Address Cache*)

- Una BTAC, que es una pequeña memoria caché asociativa en la que se almacenan:
  - Las direcciones de las instrucciones de saltos efectivos ya ejecutados o BIAs (*Branch Instruction Address*) y
  - Las direcciones de destino de esos saltos o BTAs (*Branch Target Address*).
- 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.
- 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.
  - 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
    - Si la tasa de fallos es alta Solución:
      - Que la longitud del campo BIA se incremente hasta alcanzar una tasa de saltos fantasma tolerable


Dirección del siguiente
grupo de lectura
+1
PC
MUX
0
1
¿Ha habido coincidencia?
SI = 1 NO = 0
Dirección
de lectura
BIA
BTA
I-Caché
Buffer de
instrucciones
Actualización
Unidad
funcional
de salto
Dirección de destino
del salto predicho


SI
¿Está i en la BTAC?
NO
Leer PC predicho
Etapa IF
SI
¿Es i un salto?
NO
Salto en falso:
- Vaciar cauce de instrucciones especuladas
- Procesar desde i+1
Etapa ID
Etapa II
SI
¿Predicción tomada es correcta?
NO
SI
¿Predicción no tomada es correcta?
NO
OK
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

BTIB(*Branch Target Instruction Buffer*), ENTREGA UN GRUPO DE INSTRUCCIONES que siguen a la de la bifurcación

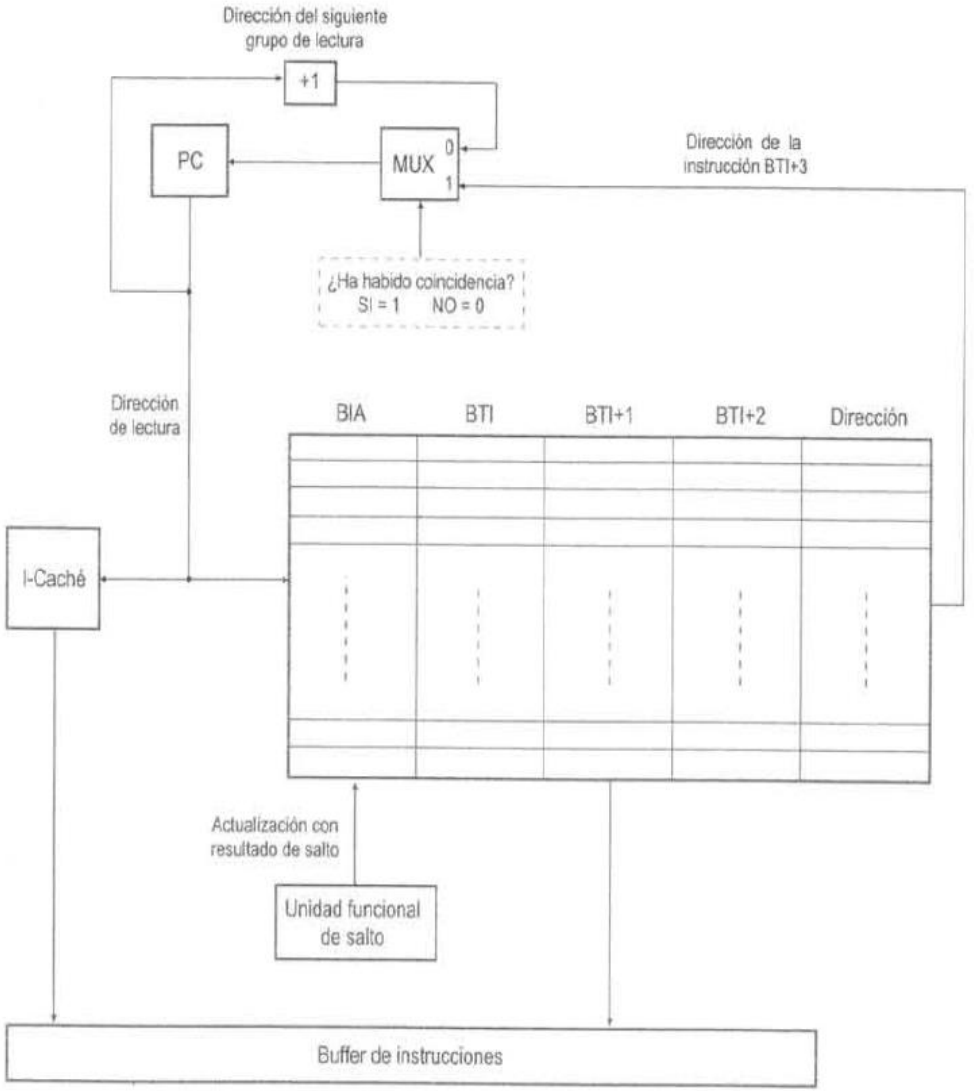


Figura 2.14: Esquema de una BTIB en la que se entregan al buffer de instrucciones paquetes de 3 instrucciones. Esto obliga a modificar el PC para que apunte a la instrucción que sigue a la última que forma el paquete, en este caso, el PC debe apuntar a la dirección de la instrucción BTI+3.

2.5.4.2. Predicción de destino de salto mediante BTB (Branch Target Buffer) con historial de salto

- Similar a la BTAC
  - Se diferencia en 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).

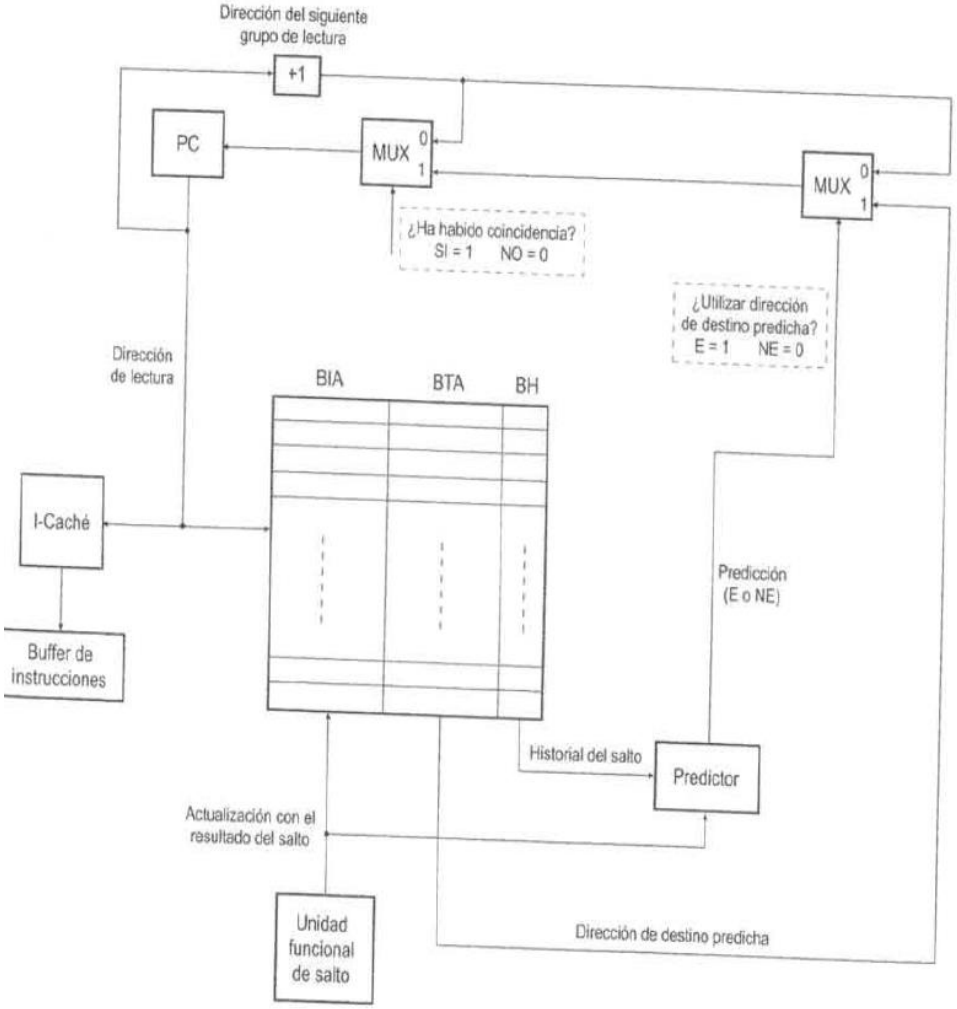
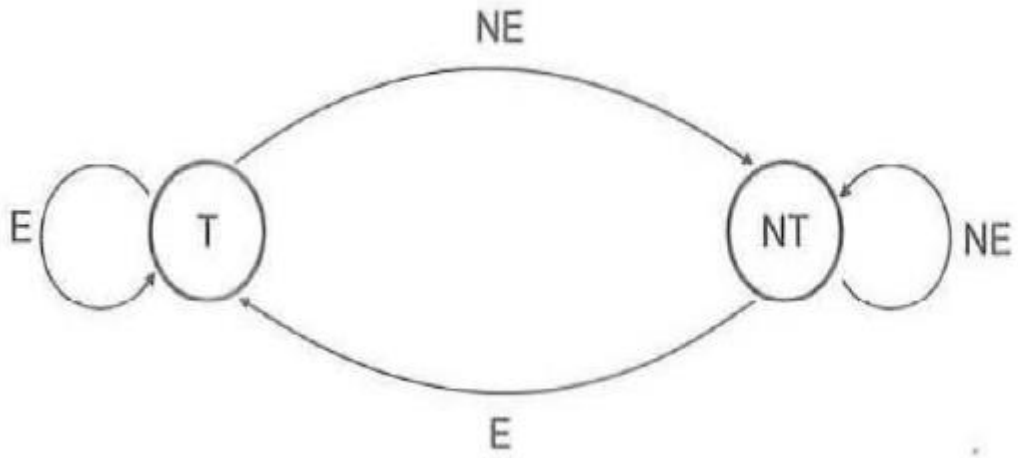


Figura 2.15: Esquema de una BTB con historial de salto.

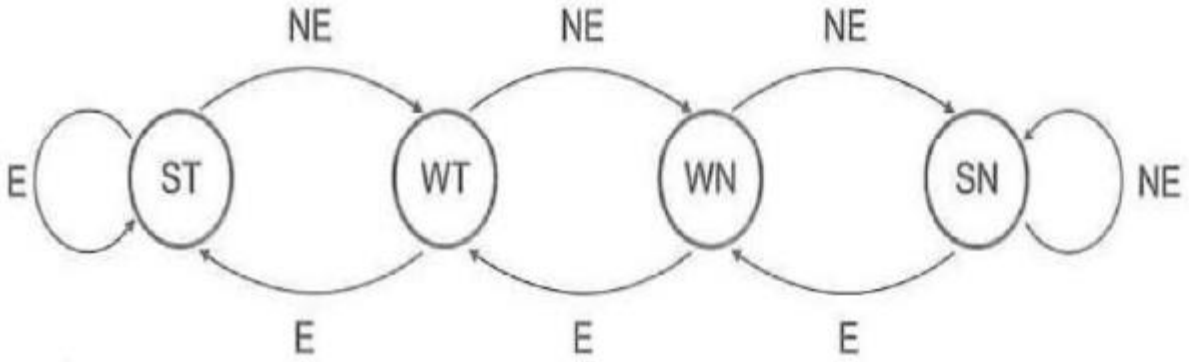
2.5.4.3. Predictor de Smith o predictor bimodal

- 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.
  - Un contador de saturación,
    - Cuando alcanza su valor máximo, permanece en el aunque se incremente
    - Permanece a cero aunque se decremente
  - 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).
- 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.
- El de 2 bits (Smith2), 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.



Predicción $Smith_1$:
T (Taken): 1
NT (Not Taken): 0

Resultado del salto:
E: Efectivo
NE: No Efectivo

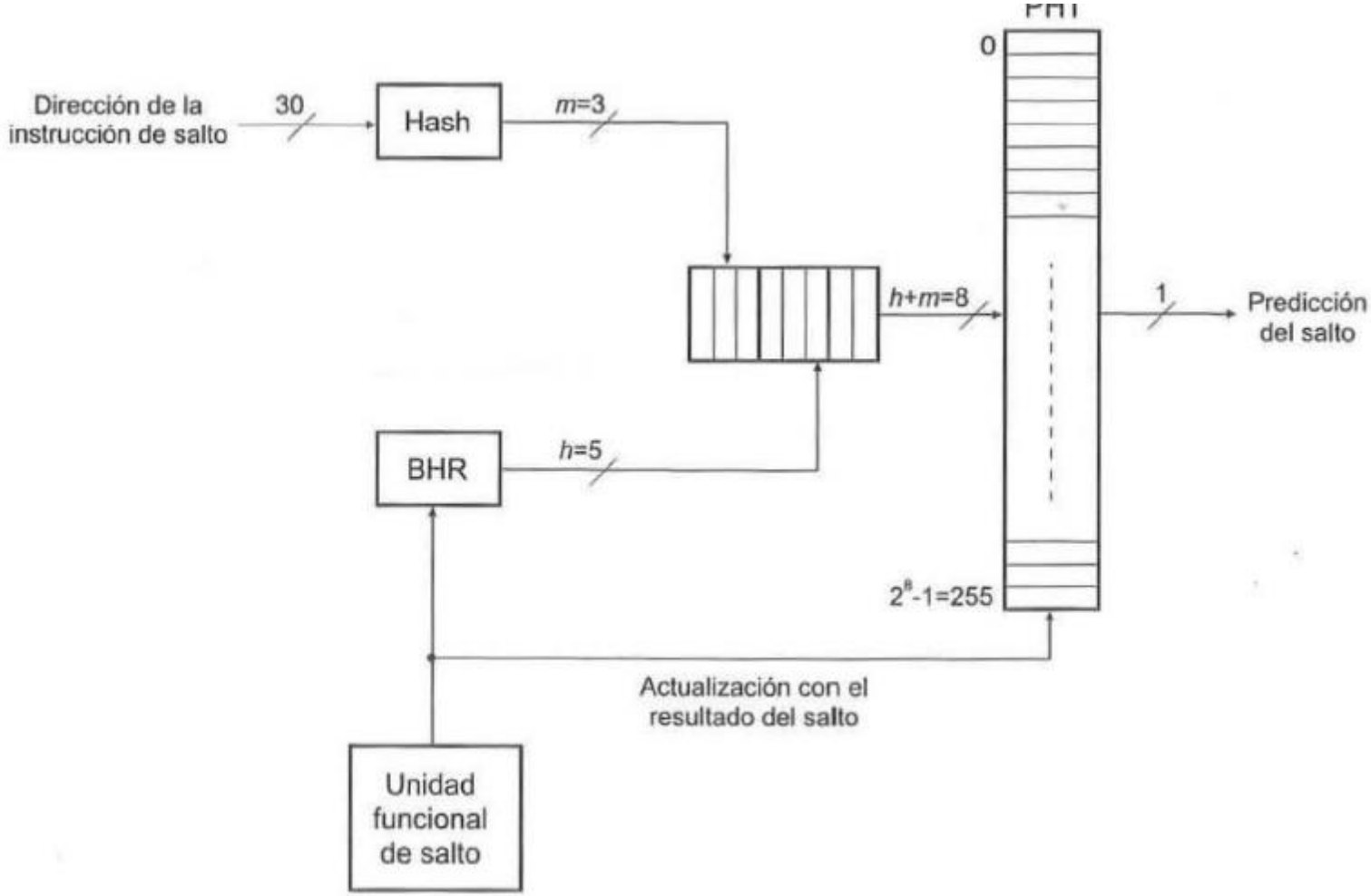


Predicción $Smith_2$:
ST (Strongly Taken): 11
WT (Weakly Taken): 10
WN (Weakly Not Taken): 01
SN (Strongly Not Taken): 00

Resultado del salto:
E: Efectivo
NE: No Efectivo

2.5.4.4. Predictor de dos niveles basado en el historial global

- Mantiene dos niveles
  - Primer nivel: historia global o historia local
  - Segundo nivel: combinación historia primer nivel con dirección de salto, para acceder a la tabla de contadores de saturación (PHT)
- Global
  - 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.
    - BHR=11110000, Si el salto es efectivo se introduce un 1 (BHR=1110001), caso contrario un 0.
  - Poder conocer la predicción para 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.
    - Quedarse con los m bits menos significativos de la dirección.
  - Esta combinación de h + m bits se utiliza para acceder a una tabla de historial de patrones (PHT - Pattern History Table).
  - Una vez que se ha evaluado el salto y se conoce su resultado hay que proceder a la actualización de los dos componentes del predictor:
    - Se incrementa/decrementa el contador de la PHT y se actualiza el historial del BHR.


PHT
0
Dirección de la instrucción de salto
30
Hash
m=3
h+m=8
1
Predicción del salto
BHR
h=5
2^8-1=255
Actualización con el resultado del salto
Unidad funcional de salto

2.5.4.5. Predictor de dos niveles basado en el historial local

- La diferencia, no almacena solo un registro de h bits, sino una tabla con el histórico de bifurcaciones
- 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 k bits ya que el número de entradas de la BHT es 2^k.

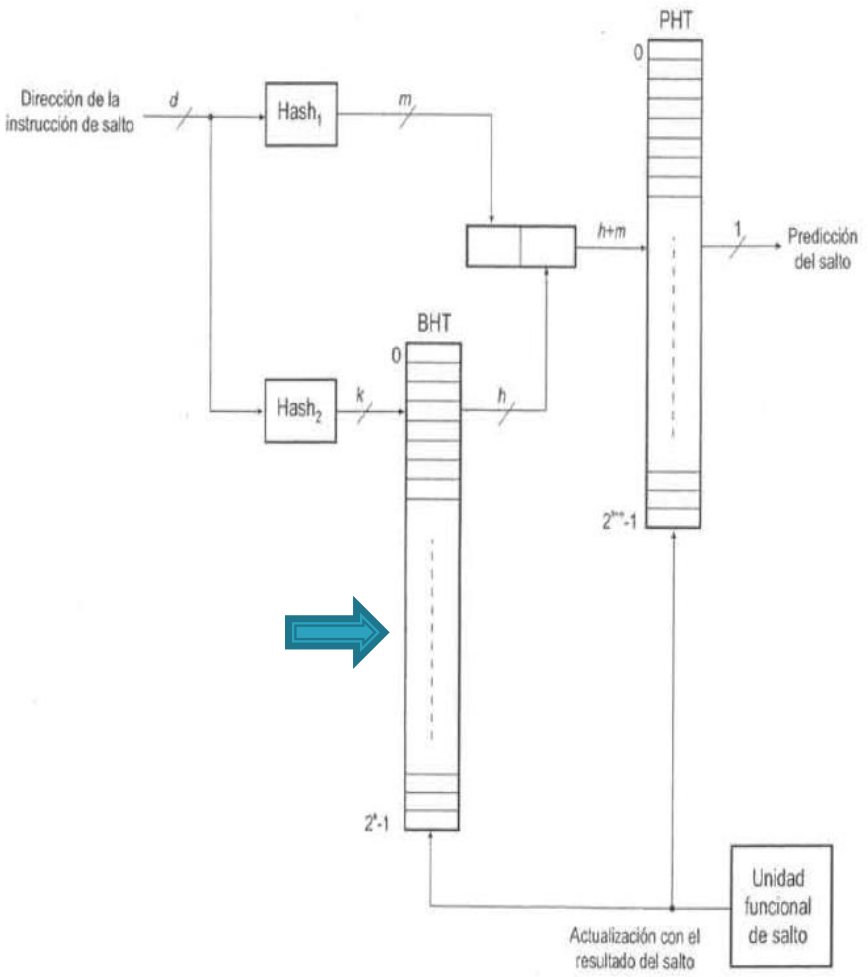
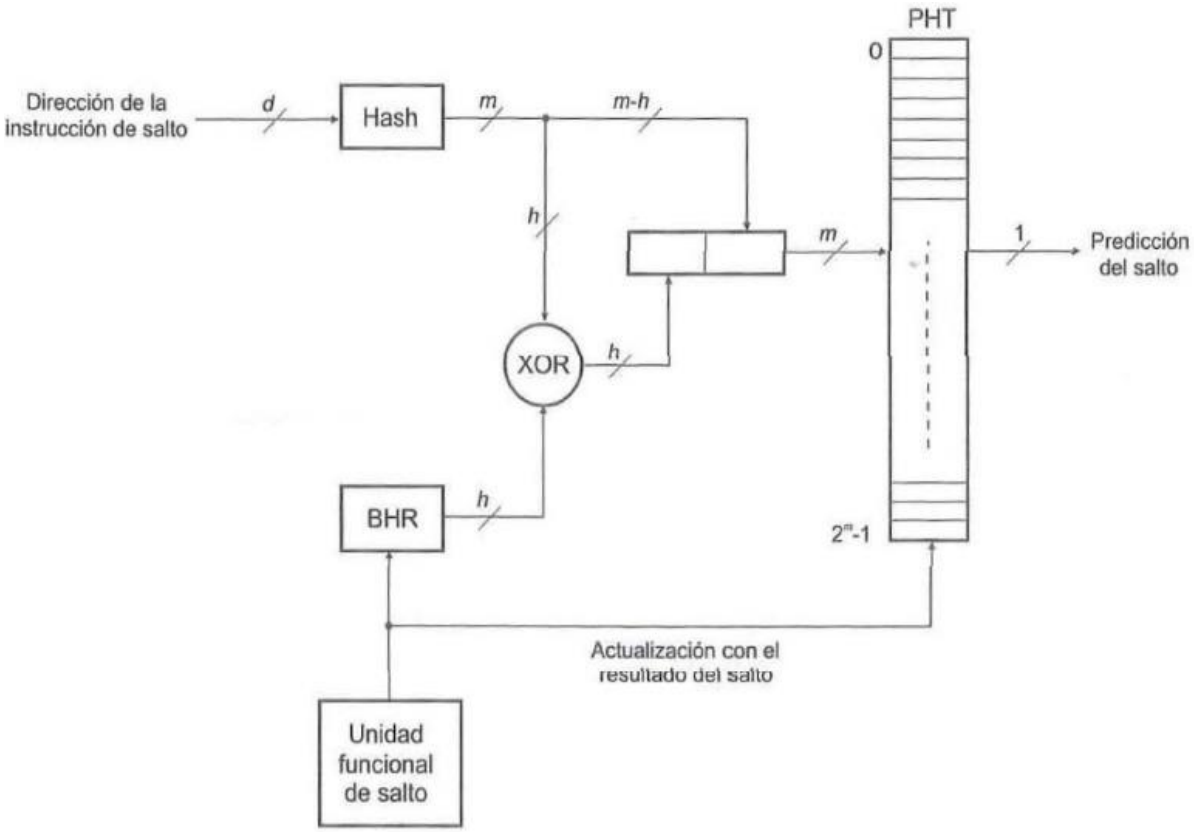


Figura 2.18: Esquema de un predictor de dos niveles basado en el historial local.

2.5.4.6. Predictor de dos niveles de índice compartido 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.



2.5.4.7. Predictores híbridos

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

2.5.5. Pila de dirección de retorno

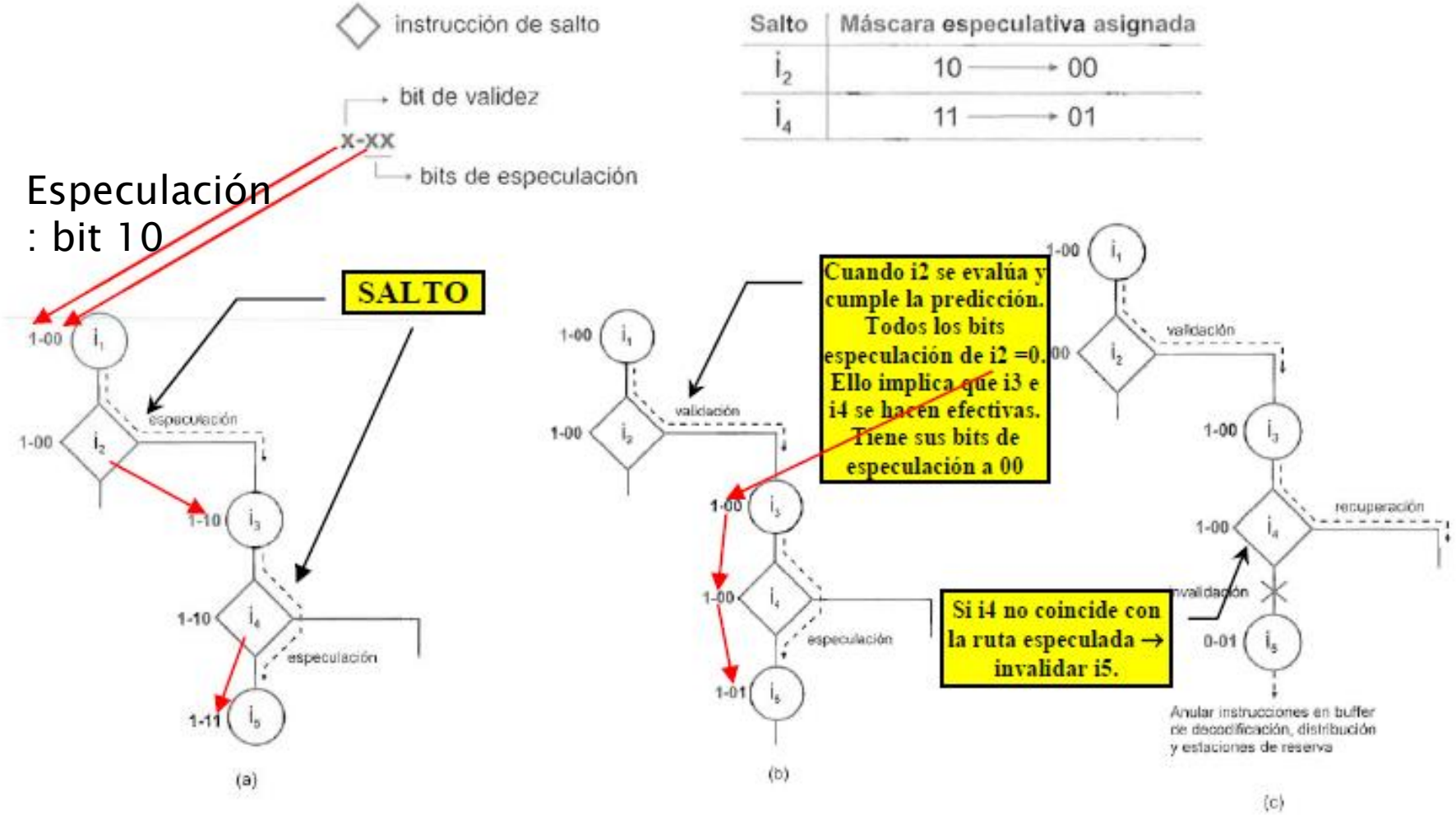
- El retorno de subrutina es una instrucción especial de salto que no puede predecirse adecuadamente ya que cada vez que se la invoca puede saltar a una dirección completamente diferente,
  - La BTB generaría predicciones de la dirección de destino con una elevada tasa de fallos
- El tratamiento correcto de los retornos de subrutina se realiza mediante una pila de direcciones de retorno RAS o buffer de pila de retornos RSB.
- 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.
- El problema que tiene la RAS es el desbordamiento y la pila no ha sido dimensionada correctamente.

2.5.6. Tratamiento de los errores en la predicción de los saltos

- El resultado de la predicción del salto no coincida con el resultado verdadero del salto.
  - Cuanto esto sucede es necesario deshacer el procesamiento de todas las instrucciones incorrectamente ejecutadas debido al error de predicción y continuar con el procesamiento correcto.
    - Esto se conoce como recuperación de la secuencia correcta
- La forma habitual para conocer y validar o anular las secuencias de instrucciones es etiquetarlas.
- Dos de esas etiquetas son:
  - La especulativa.
    - La dirección de toda instrucción de salto sobre la que se realiza la especulación se almacena en una tabla junto con la etiqueta especulativa.
  - La de validez.
    - La etiqueta validez permite saber si la instrucción debe o no terminarse arquitectónicamente.
      - Aunque una instrucción sea válida, no podrá terminarse arquitectónicamente hasta que la etiqueta que la define como especulativa cambie a no especulativa (todos los bits a 0).
- 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.
  - **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*).

Figura 2.21: Un error de predicción conlleva la ejecución incorrecta de instrucciones que es necesario deshacer una vez conocido el resultado del salto.

Especulación : bit 10



2.6. Decodificación

- Los procesadores superescalares reparten estas tareas en dos etapas:
  - **Decodificación**
    - Detecta los saltos en falso y 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 y del análisis de las dependencias verdaderas.
  - 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 (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.
    - La precodificación o decodificación previa
      - Realizar una parte de la decodificación entes que la extracción de instrucciones de la I-caché
    - Traducción de instrucciones,
      - Se descompone una instrucción en instrucciones más básicas, o
      - Unir varias instrucciones en una única instrucción interna.

2.6.1. Predecodificación

- 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.

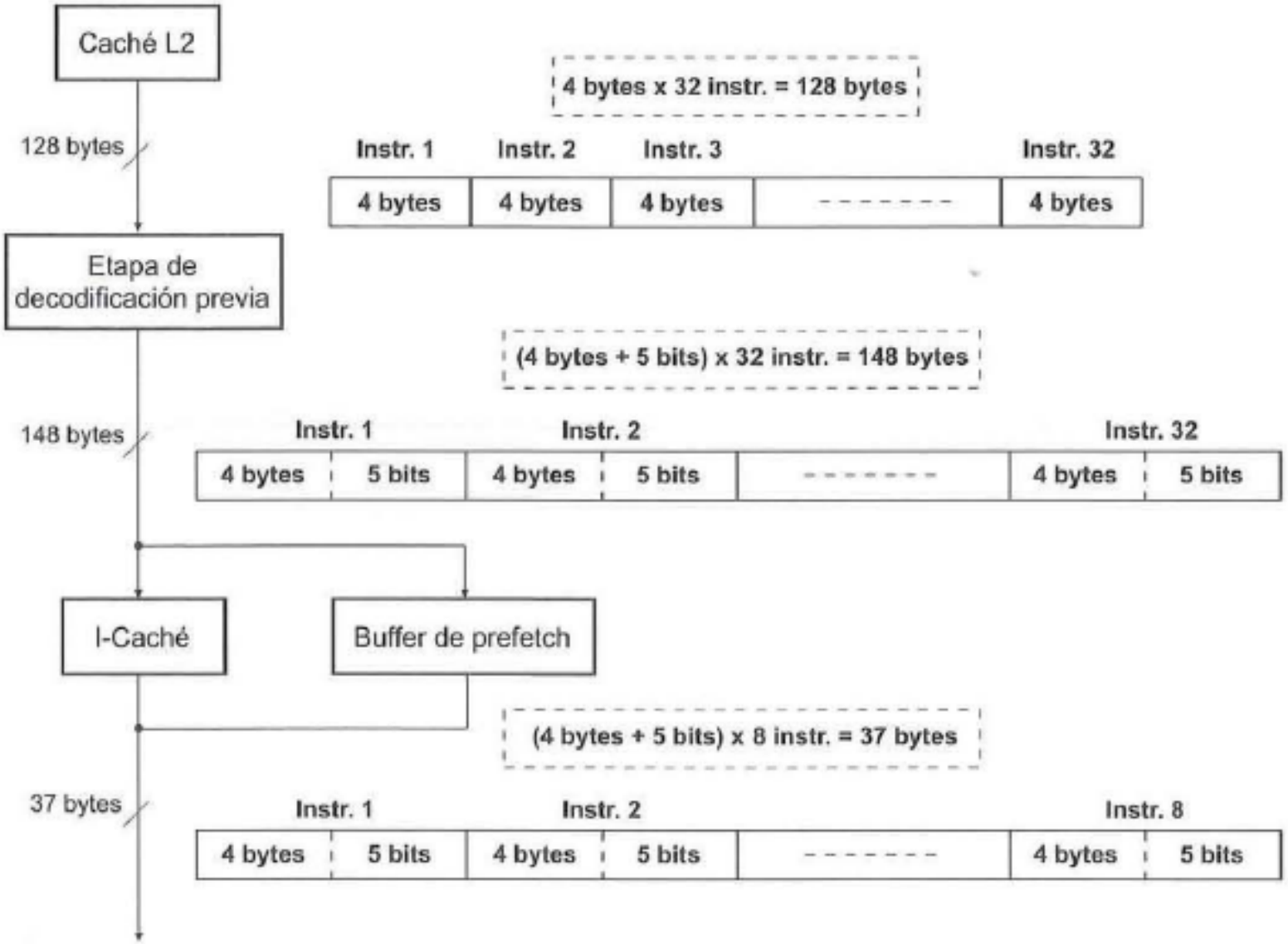


Figura 2.23: Decodificación previa en el PowerPC 970.

AMD Opteron añade 3 bits de predecodificación por cada byte de instrucción y 4 bits adicionales por cada 16 bytes.
1º bit START
2º bit STOP
3º bit FUNCTION:
- Si instrucción sencilla
- Si inst. compleja → descomposición en oper. Sencillas micro-ops.

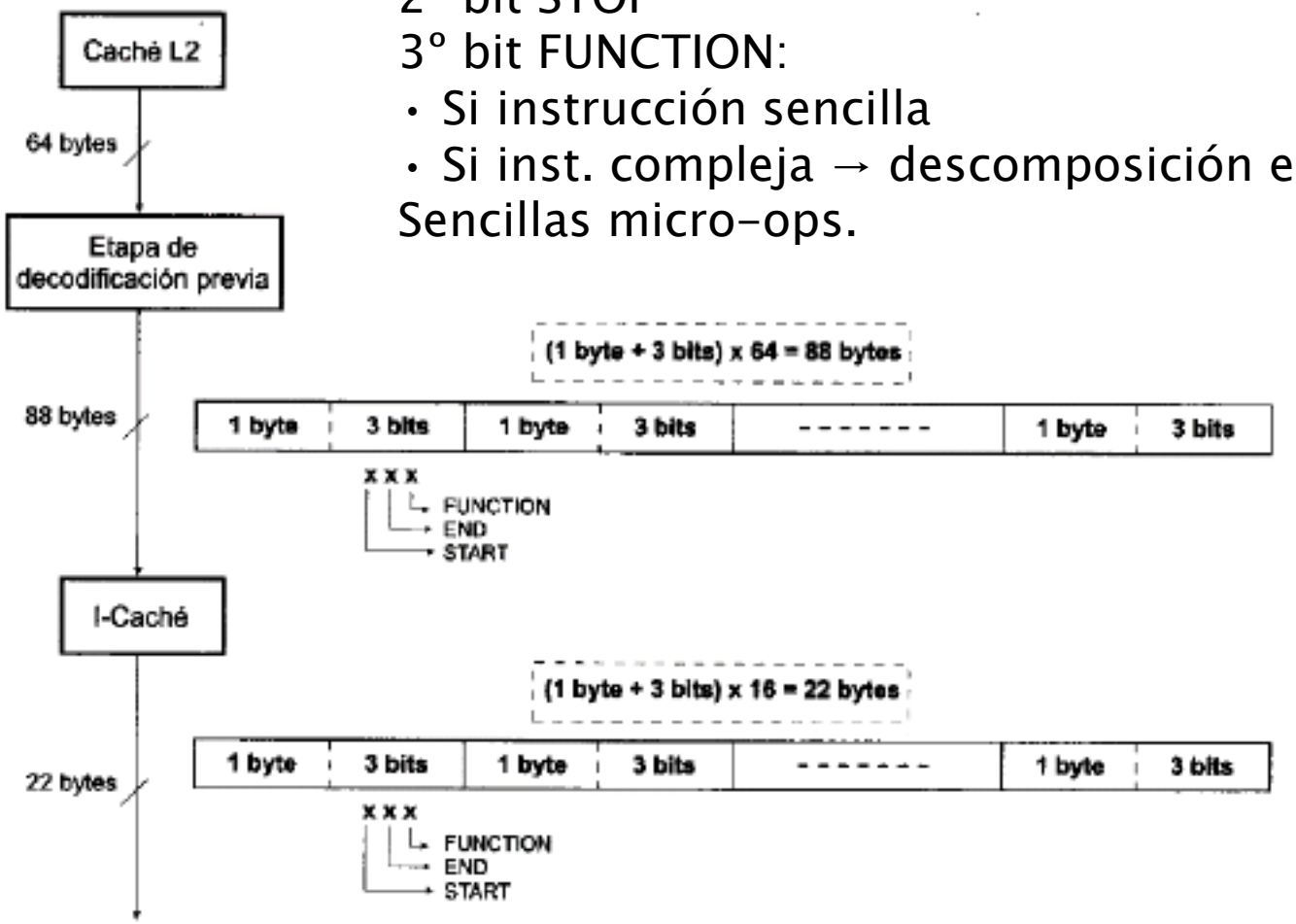


Figura 2.24: Decodificación previa en el AMD Opteron. Por simplificar, no se reflejan los 4 bits que se añaden cada 16 bytes.

2.6.2. Traducción de instrucciones

- Traducción de instrucciones complejas (CICS) en un conjunto de instrucciones más básicas tipo RISC
  - Estas operaciones básicas se llaman:
    - En Intel microoperaciones (micro-ops).
    - En PowerPC operaciones internas (IOPs –Internal OPerations).
    - En AMD ROPs (Rips Operations).

Ejemplo de PowerPc

El D0, puede manipular cualquier instrucción, generando entre una y cuatro micro-ops por ciclo de reloj a partir de una instrucción.
Si es instrucción muy compleja que requiere de cinco o más micro-ops, entonces el D0 la envía al generador de microcódigo que produce secuencias de más de cuatro micro-ops a una velocidad de tres micro-ops por ciclo de reloj.
Macro-fusión → fusionar o unir ciertos tipos de instrucciones en la fase de predecodificacion.
Micro-fusión → decodificar una instrucción que genera dos micro-ops y fundirlas en una única micro-op.

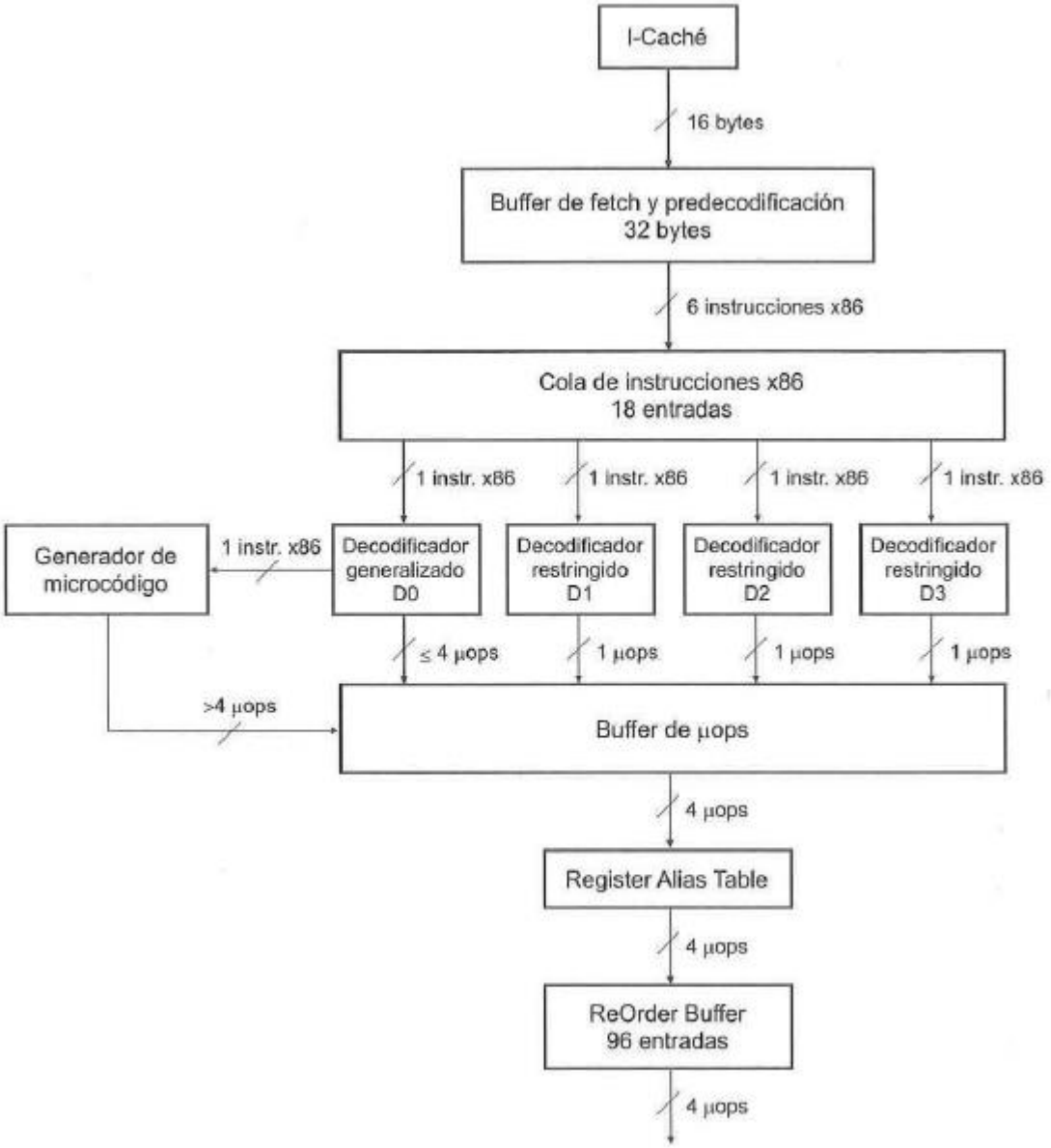


Figura 2.25: Decodificación en la arquitectura Intel Core Microarchitecture.

Ejemplo de Intel core

Cada ciclo de reloj extrae cinco instrucciones del buffer de instrucciones y se comienzan a procesar por las tres etapas D1, D2 y D3 y conocida como in-fine decoding. Cuando una instrucción tiene que ser traducida en más de dos IOPs, el procesador recurre a una extensión de la etapa de decodificación compuesta por un cauce de cuatro subetapas y que realiza una decodificación basada en plantillas (template-based decoding). Esta extensión es capaz de generar hasta 4 IOPs por ciclo de reloj para emular el comportamiento de una instrucción normal.

- **Instrucciones rotas (cracked instruction)** → se descomponen en dos IOPs.
- **Instrucciones microcodificadas (microcoded instructions)** → descomponen en tres o más IOPs.

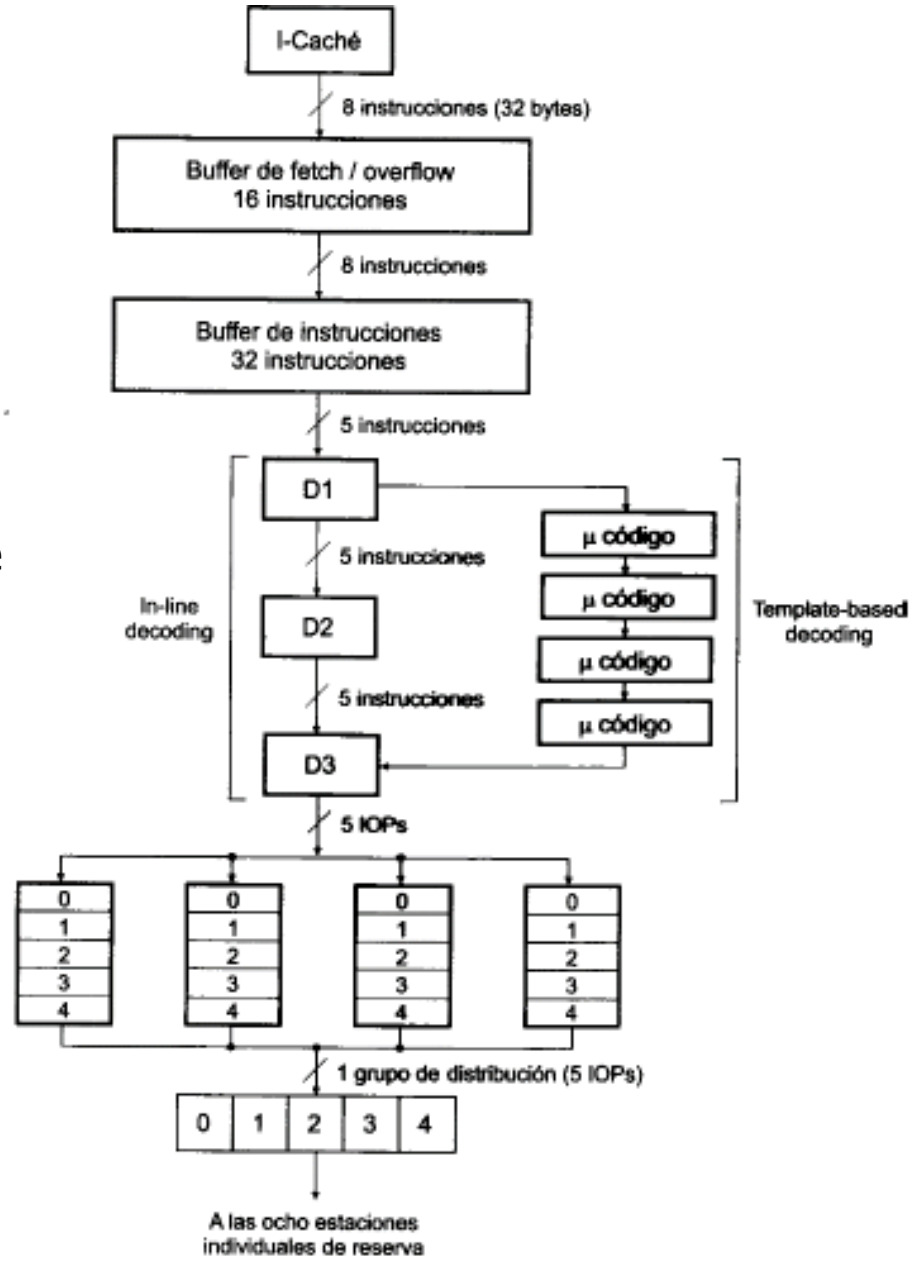


Figura 2.26: Etapa de decodificación del PowerPC 970.

2.7 Distribución

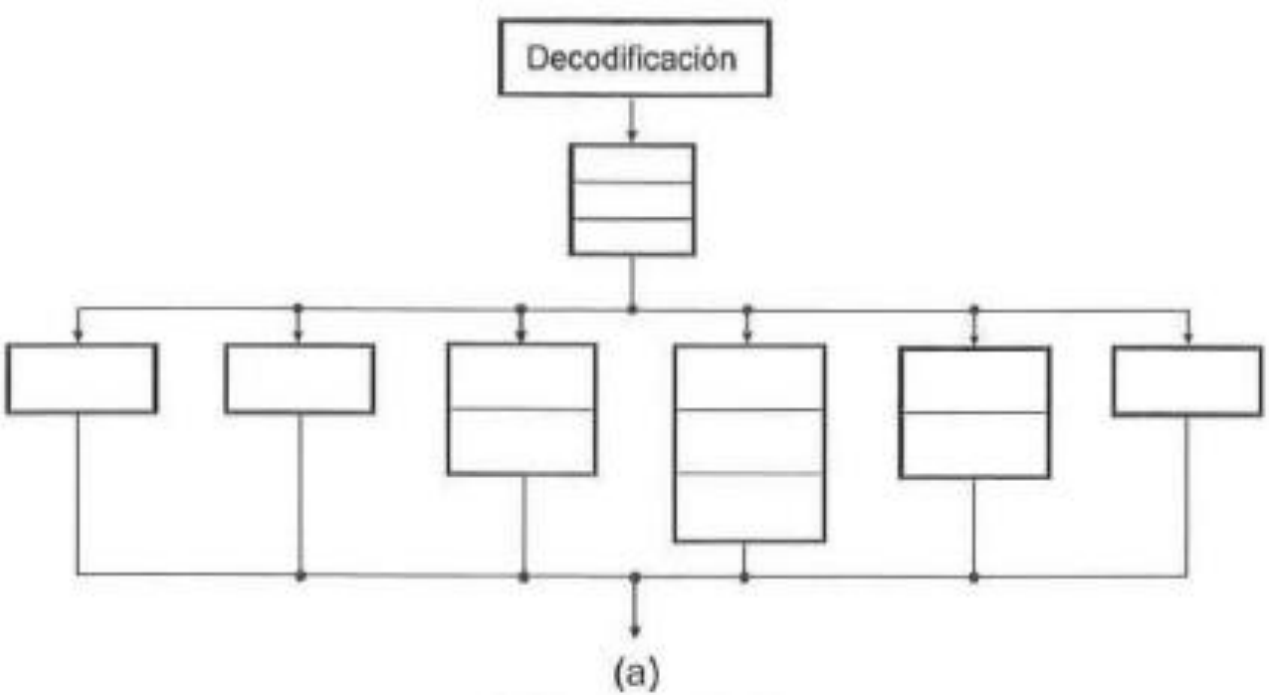
- Se ocupa de repartir las instrucciones según su tipo entre las diferentes unidades funcionales para que se pueda proceder a su ejecución en paralelo y en fuera de orden
- Las instrucciones se depositan en dos buffers
  - **Buffer de distribución o ventana de instrucciones**
    - No conservan el orden del programa
    - Sirve para desacoplar la etapa de decodificación de la de ejecución
    - Para poder ejecutar una instrucción debe tener disponibles todos los operandos y las unidades funcionales y buses que le correspondan
    - Los operandos se almacenan con un bit que indica si está ya disponible o no
  - **Buffer de terminación o de reordenamiento (después de ejecución)**
    - Se almacenan en el mismo orden que el programa
    - Una vez ejecutadas las instrucciones, este buffer reestablece el orden y garantiza la consistencia del procesador y de la memoria
- Solución ante los riesgos estructurales
  - Desacoplar la etapa de decodificación de la de ejecución utilizando para ello 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.

2.7.1. Organización de 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).
- Estación de reserva distribuida o individuales
  - Cada unidad funcional dispone de una estación de reserva propia.
- Estación de reserva en cluster 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.

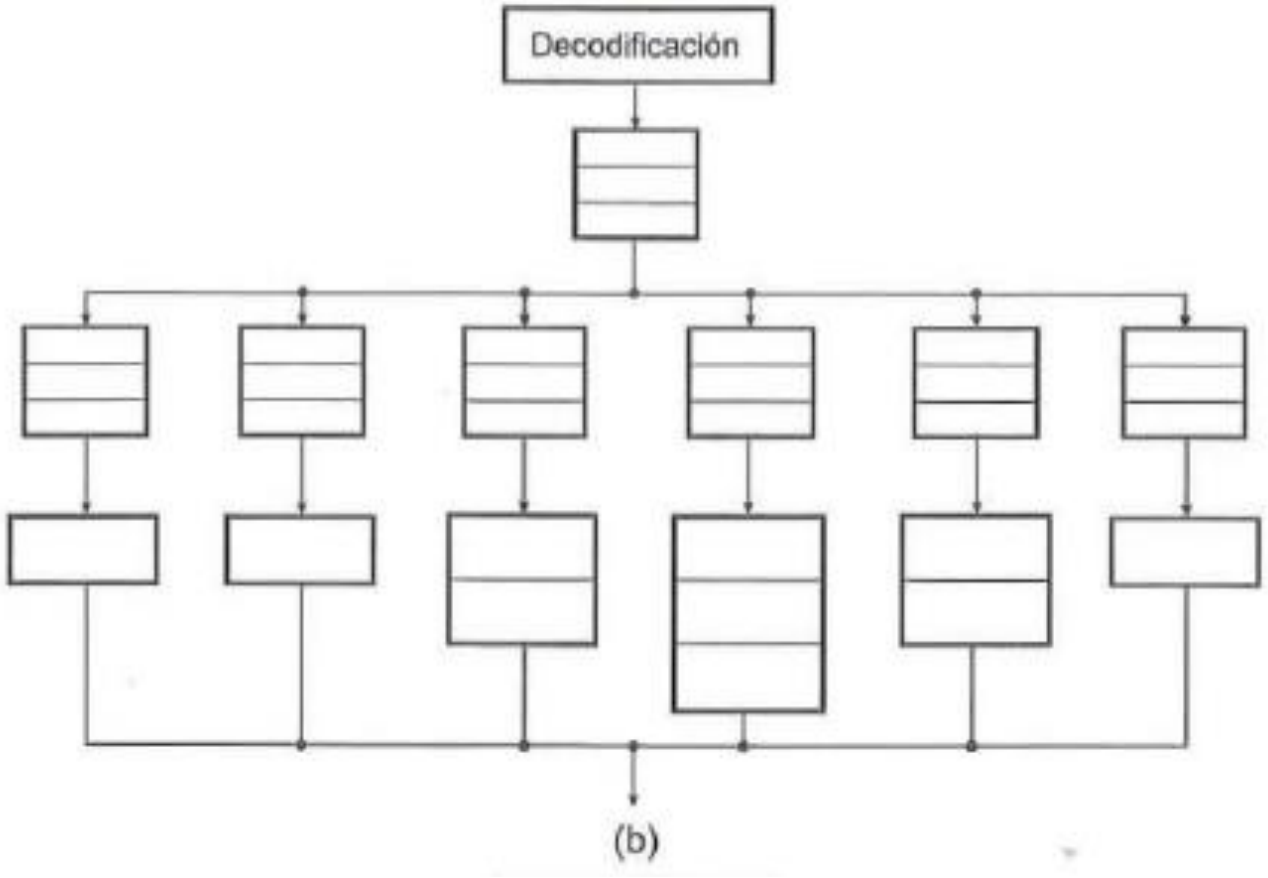
- Distribución
  - Asociar una instrucción a una unidad funcional.
- Emitir
  - Enviar la instrucción a la unidad funcional para comenzar la ejecución

Centralizada



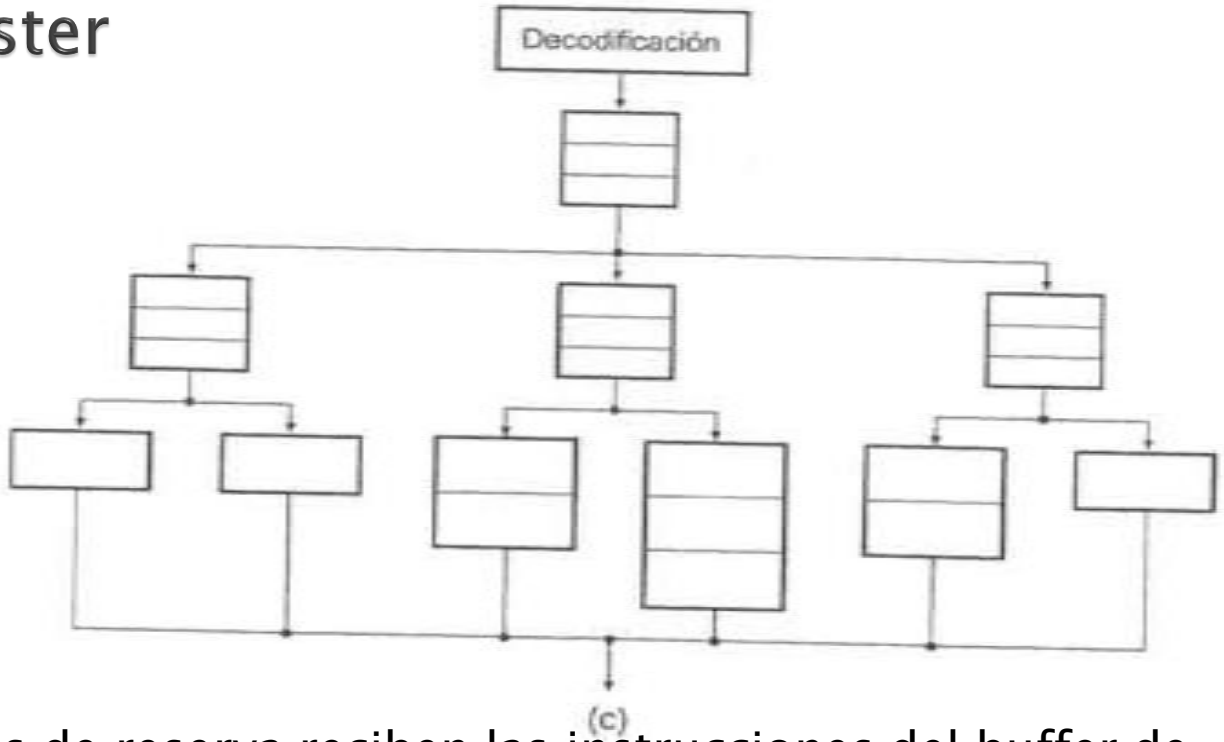
- Es lo que hemos definido hasta ahora como ventana de instrucciones o buffer de instrucciones (Ventana de emisión o cola de emisión).
  - Tiene un hardware de control muy complejo.
  - Los términos distribución y emisión significan lo mismo ya que la asociación de la instrucción a la unidad funcional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

Estación de re distribuida 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.

Estación de reserva en cluster 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

2.7.2. 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 es un conjunto de bits agrupados por campos y que representan una instrucción
  - Ocupado(O)
  - Código de operación (CO)
  - Operando 1 (Op1)
    - Si está disponible contiene el valor
    - Si no está disponible contiene el identificador del registro
  - Válido 1 (V1):
    - Indica si el operando 1 está disponible o no
  - Operando 2 (Op2)
  - Válido 2 (V2)
  - Destino (D)
  - Listo (L)
    - Que todos los operandos están disponibles y la instrucción puede emitirse

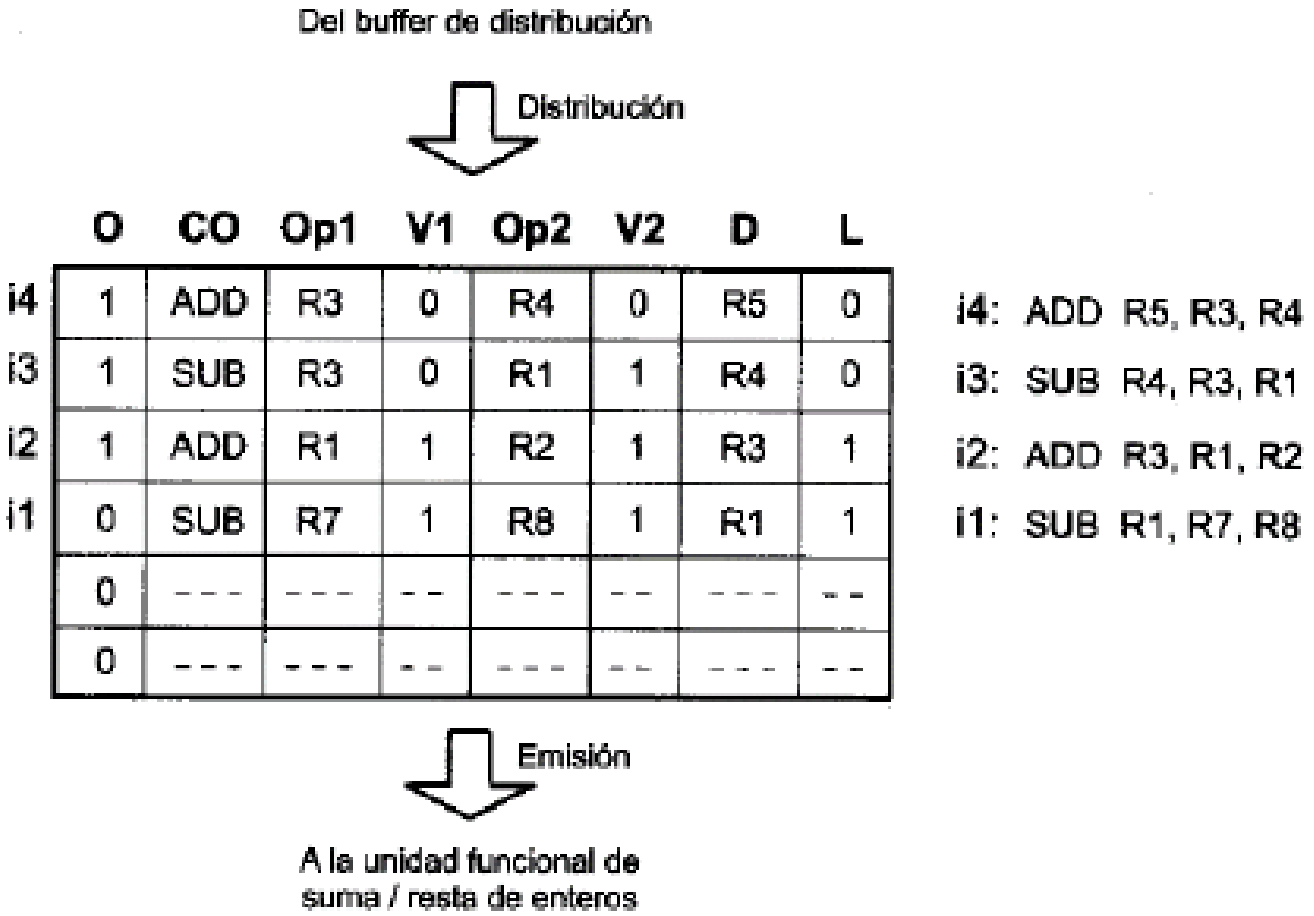


Figura 2.29: Estados de una instrucción en una estación de reserva individual.

EJEMPLO de emisión con estaciones distribuidas

- Premisas
  - El buffer de distribución puede recibir y suministrar hasta 5 instrucciones/ciclo
  - La unidad de suma/resta consume 1 ciclo y la de multiplicación/división 2 ciclos estando segmentada (varias instrucciones en ejecución en el mismo ciclo).
  - En el ciclo en el que se emite una instrucción se empieza su ejecución.

i1: SUB R1, R7, R0

i2: MULT R5, R1, R7

i3: ADD R5, R2, R1

i4: DIV R4, R3, R7

i5: MULT R6, R4, R5

(a) Secuencia de instrucciones

Ciclo i: Recepción de la etapa de decodificación de i1, i2, i3, i4 e i5

Las 5 instrucciones son recibidas

	O	CO	Op1	V1	Op2	V2	D	L
i5	1	MULT	R4	0	R5	0	R6	0
i4	1	DIV	R3	1	R7	1	R4	1
i3	1	ADD	R2	1	R1	0	R5	0
i2	1	MULT	R1	0	R7	1	R5	0
i1	1	SUB	R7	1	R0	1	R1	1

RAW

WAW

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

Ciclo i+1: Distribución a las estaciones de reserva de i1, i2, i3 e i4

Se distribuyen las 4 primeras, cada una a su tipo de unidad funcional

	O	CO	Op1	V1	Op2	V2	D	L
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
i5	1	MULT	R4	0	R5	0	R6	0

	O	CO	Op1	V1	Op2	V2	D	L
i3	1	ADD	R2	1	R1	0	R5	0
i1	1	SUB	R7	1	R0	1	R1	1

	O	CO	Op1	V1	Op2	V2	D	L
i4	1	DIV	R3	1	R7	1	R4	1
i2	1	MULT	R1	0	R7	1	R5	0

Figura 2.30: Ejemplo de emisión de instrucciones entre estaciones de reserva distribuidas (continúa).

Disponibles

Ciclo i+2: Emisión de i1 e i4. Distribución de i5

i1: SUB R1, R7, R0

i2: MULT R5, R1, R7

i3: ADD R5, R2, R1

i4: DIV R4, R3, R7

i5: MULT R6, R4, R5

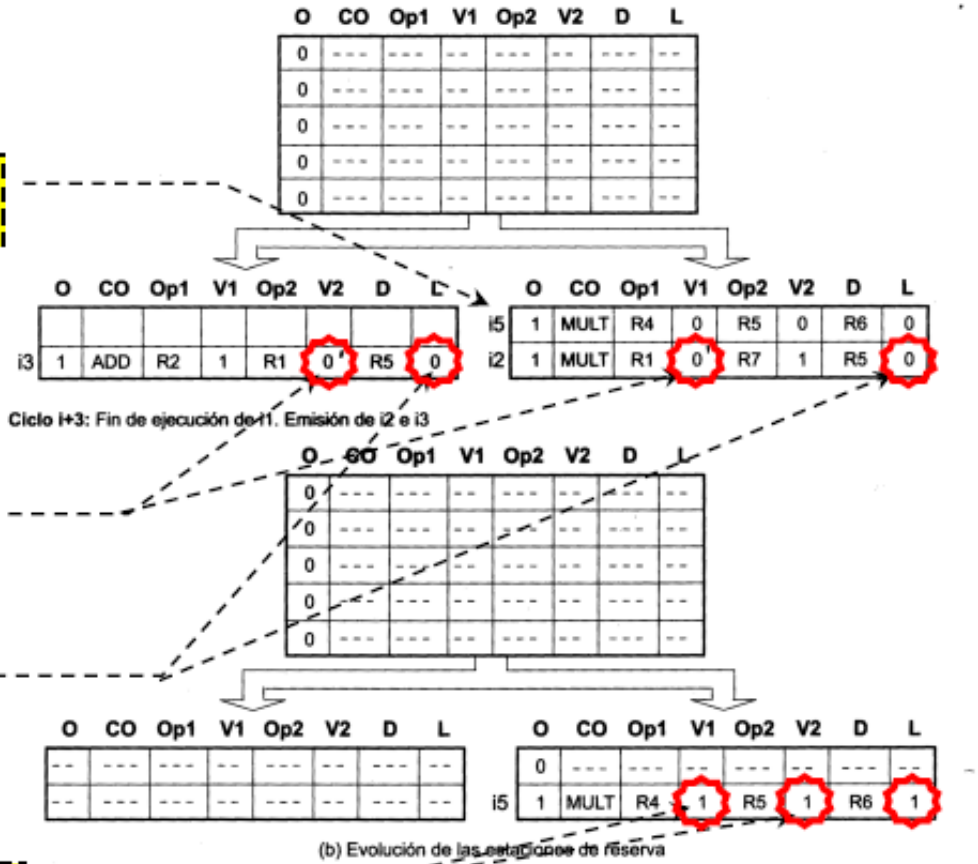
Se emiten i1 e i4 y se distribuye i5

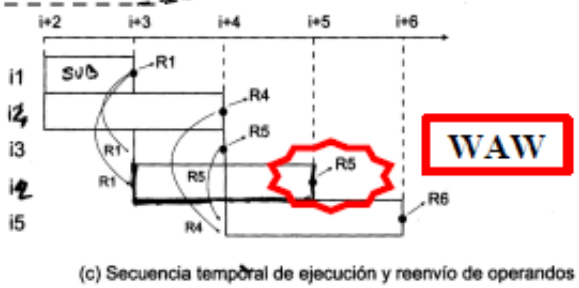
Al terminar i1 (suma/resta 1 ciclo) R1 disponible y se informa a las estaciones

i3 e i2 quedan listas (L=1) y se emiten

Ciclo i+4:
- **Termina i4 ((emitida en i+2) DIV 2 ciclos)**
- **Disponible R4**
- **Termina i3 (ADD 1 ciclo)**
- **Disponible R5**
- **Emisión de i5**

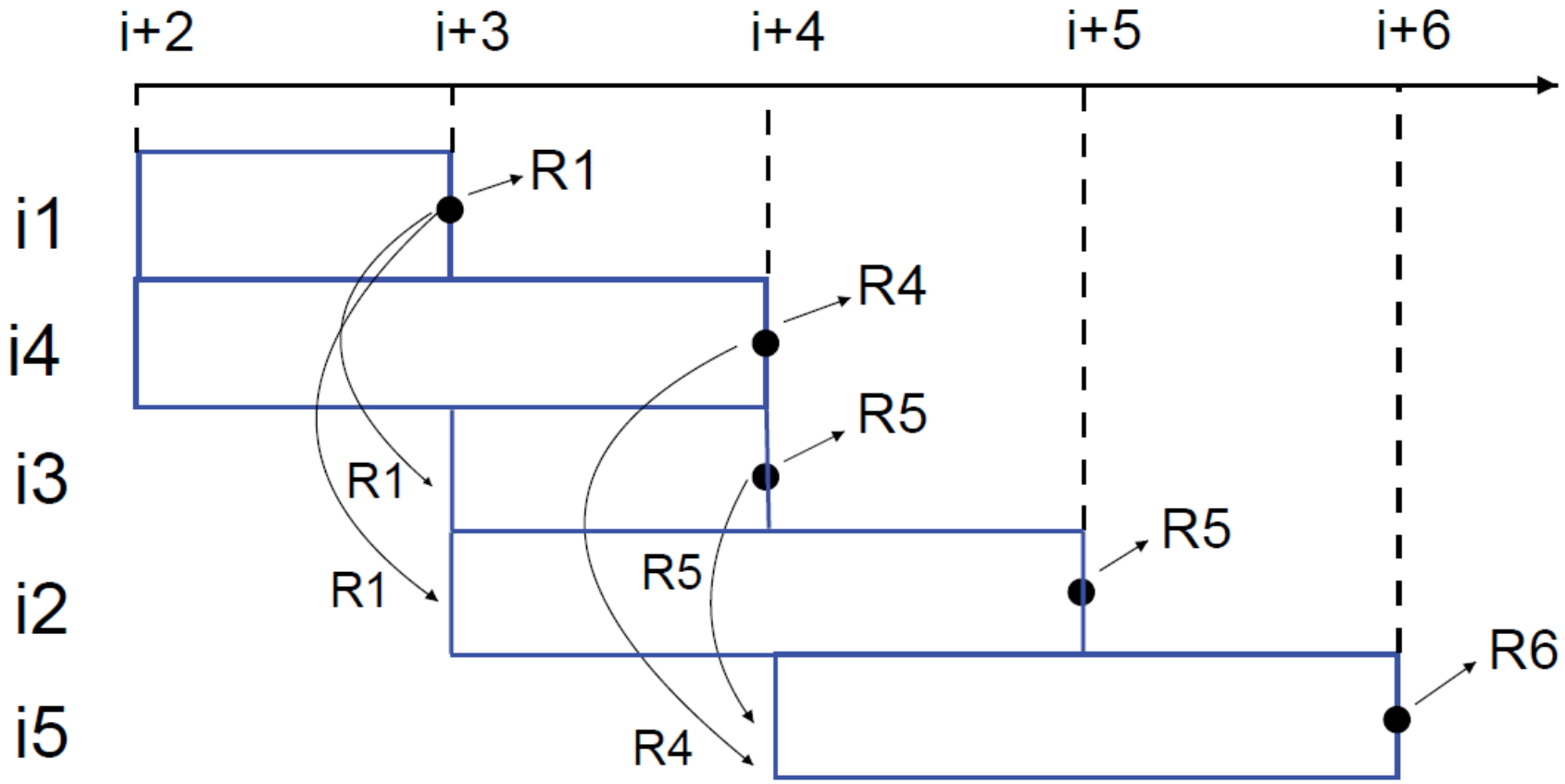
Ciclo i+5: Termina i2
Ciclo i+6: Termina i5





(c) Secuencia temporal de ejecución y reenvío de operandos

Figura 2.30: [Continuación] Ejemplo de emisión de instrucciones entre estaciones de reserva distribuidas.



Las fases por las que pasa una instrucción desde que abandona el buffer de instrucciones hasta que se emite desde la estación de reserva individual son:

- Distribución
- Supervisión
- Emisión

2.7.2.1. Fase de distribución

- Envío de una instrucción desde el buffer de instrucciones a la estación de reserva individual
- La *lógica de asignación*
  - Se ocupa de ubicar correctamente la instrucción recibida para lo que debe llevar un registro de las instrucciones almacenadas y de las entradas que están libres (para ello se dispone de los bits de ocupado en la estación de reserva).

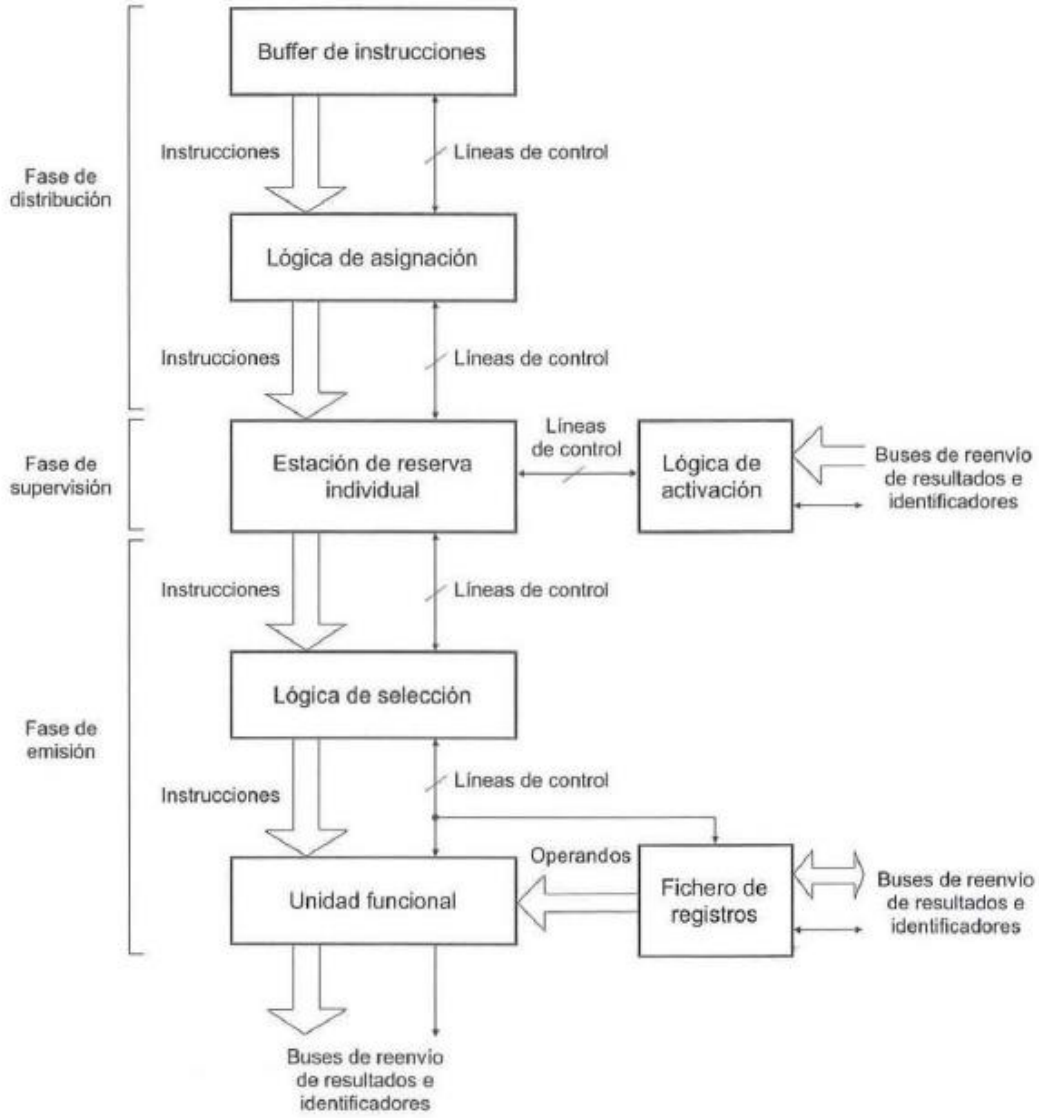


Figura 2.31: Fases de una instrucción en la etapa de distribución.

2.7.2.2. Fase de supervisión

- La fase de supervisión concluye cuando tiene los dos operandos disponibles
  - Revisa en CDB (Common Data Bus)
- **Lógica de activación**
  - El hardware que realiza la supervisión de los buses
  - Durante la espera activa, la *lógica de activación* actualiza los bits de los campos
    - 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.

2.7.2.3. Fase de emisión

- Cuando están todos los operandos disponibles, la **lógica de selección** determina la instrucción que se puede emitir entre todas las disponibles
- El problema surge cuando hay varias instrucciones en condiciones de ser emitidas en el mismo ciclo de reloj.
  - Algoritmo de planificación o planificador dinámico
    - El más habitual: Política de planificación emisión la mas antigua
- 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:
  - Emisión alineada
    - La ventana de distribución no puede enviar nuevas instrucciones a la estación de reserva hasta que no esté completamente vacía
  - Emisión no alineada
    - Puede emitir instrucciones siempre que queden entradas libres en las estaciones de reserva

Ejemplo de emisión

Premisas

- La misma estación de reserva alimenta a la UF suma/resta y multi/división
- La unidad de suma/resta consume 1 ciclo y la de multiplicación/división 2 ciclos estando segmentada.
- El ciclo en el que la UF genera el operando, la estación de reserva actualiza sus bits de validez → el ciclo siguiente se puede emitir la instrucción.

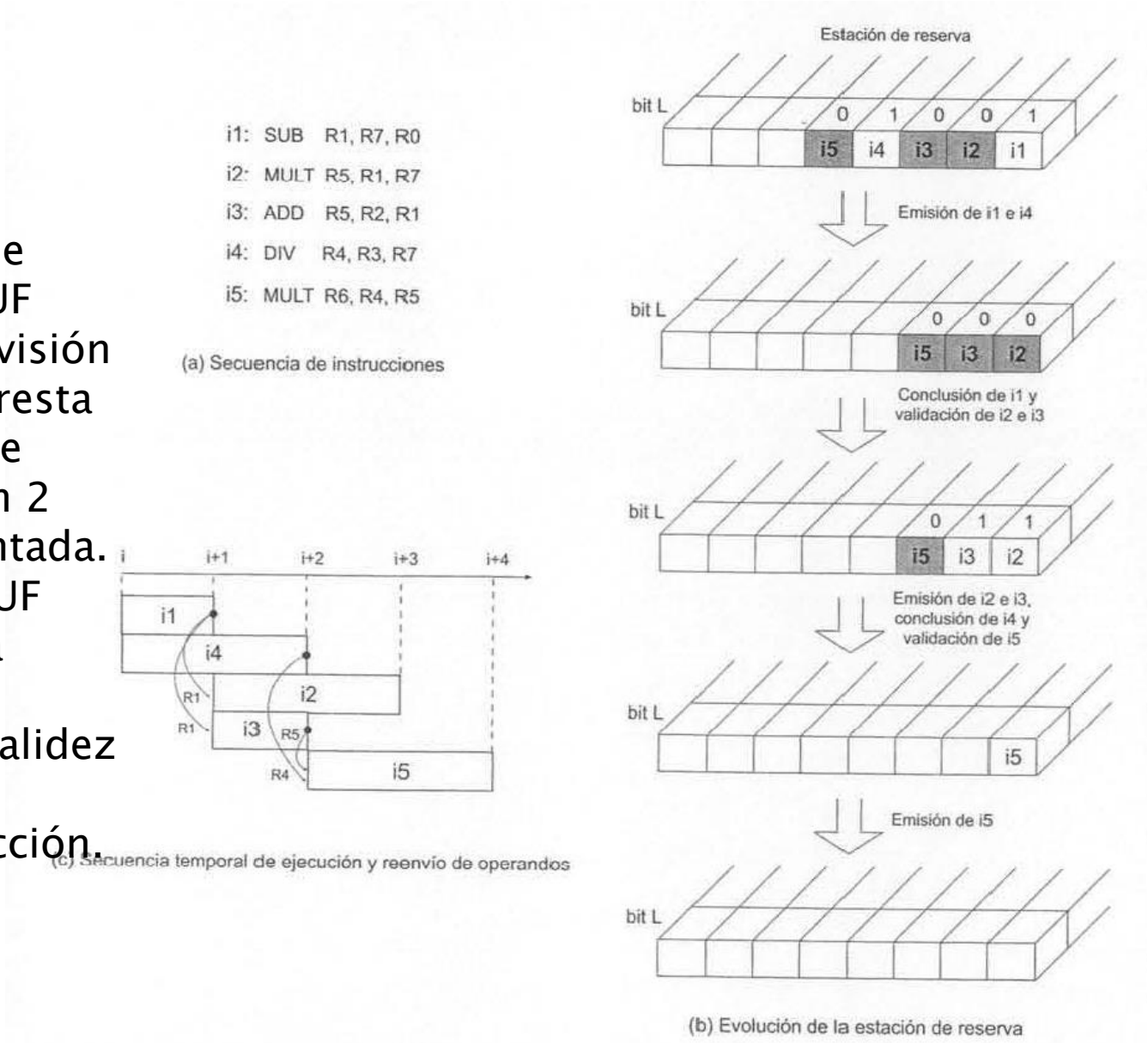


Figura 2.32: Ejemplo de emisión de una secuencia de cinco instrucciones a una estación de reserva individual que alimenta dos unidades funcionales.

i1: ADD R3, R2, R1
i2: ADD R8, R3, R7
i3: MULT R6, R5, R4
i4: ADD R9, R6, R1
i5: MULT R10, R3, R6
i6: ADD R9, R2, R1
i7: ADD R10, R3, R8
i8: MULT R10, R2, R1

(a) Secuencia de instrucciones

RAW

i2, i5, i7 ← R3 — i1

i4, i5 ← R6 — i3

i7 ← R8 — i2

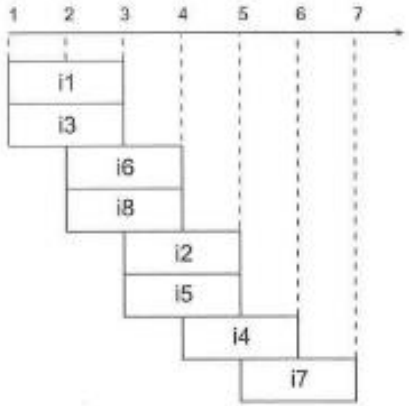
WAW

i6 ← R9 — i4

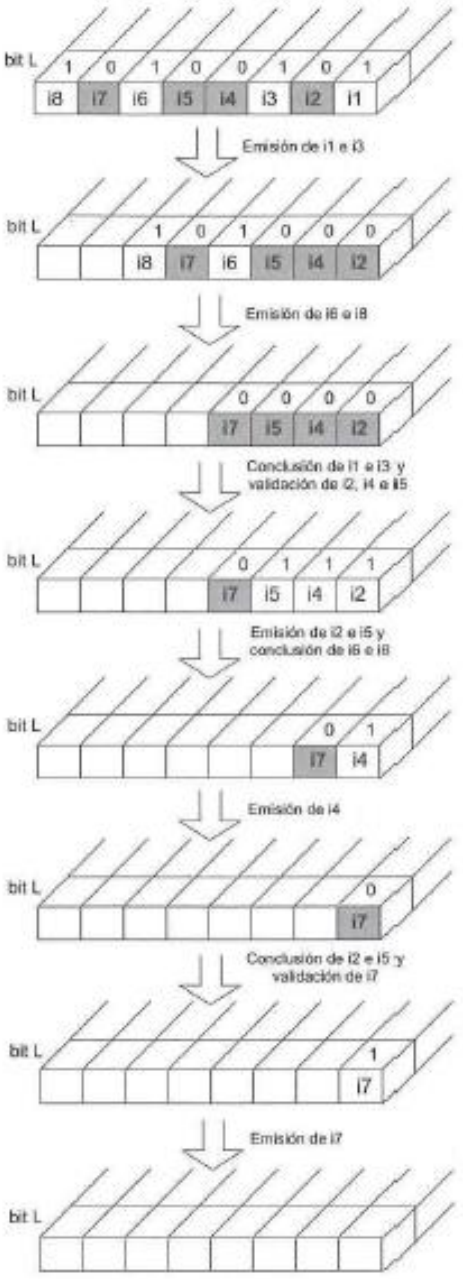
i7, i8 ← R10 — i5

i8 ← R10 — i7

(b) Dependencias



(c) Secuencia temporal de ejecución y reenvío de operandos

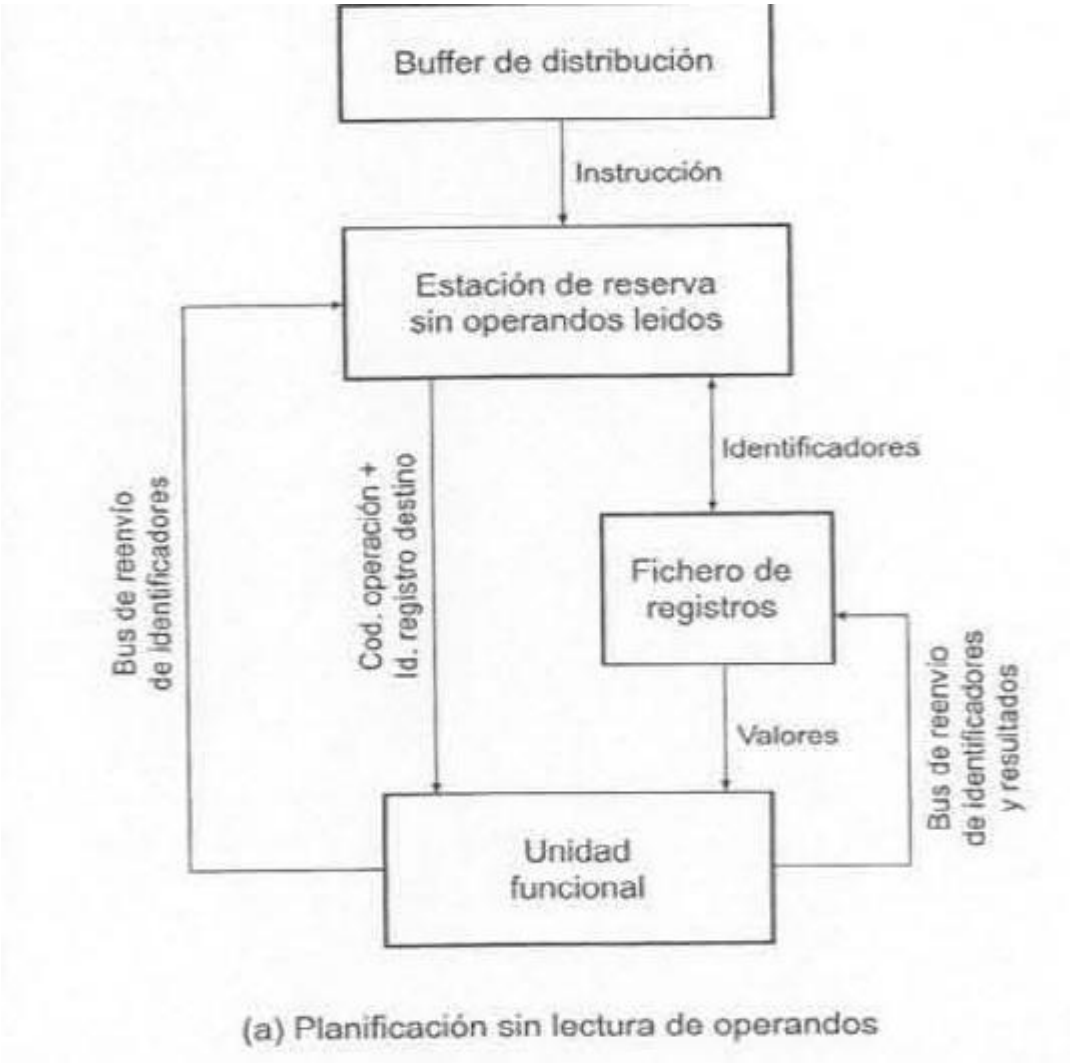


(d) Evolución de la estación de reserva

Figura 2.33: Ejemplo de emisión de una secuencia de 8 instrucciones a una estación de reserva individual que alimenta a 2 unidades funcionales.

2.7.3. Lectura de los operandos

- Planificación sin lectura de operando
  - La lectura de operandos fuentes se hace en el momento de la emisión a las unidades funcionales



(a) Planificación sin lectura de operandos

Planificación con lectura de operandos

- Los operandos se leen cuando la instrucción se distribuye desde el buffer de distribución a las estaciones de reserva

(b) Planificación con lectura de operandos

Planificación con lectura de operandos

- La estación de reserva centralizada distribuye 4 instrucciones/ciclo a las dos estaciones de reserva individuales.
- Unidad funcional de suma/resta (1 ciclo).
- Unidad segmentada de multiplicación/división (2 ciclos).

i1: SUB R1, R7, R0
i2: MULT R5, R1, R7
i3: ADD R5, R2, R1
i4: DIV R4, R3, R7
i5: MULT R6, R4, R5

(a) Secuencia de instrucciones

Ciclo i:

	O	CO	Op1	Op2	D
i5	1	MULT	R4	R5	R6
i4	1	DIV	R3	R7	R4
i3	1	ADD	R2	R1	R5
i2	1	MULT	R1	R7	R5
i1	1	SUB	R7	R0	R1

	Datos	V
R0	0	1
R1	1	1
R2	2	1
R3	35	1
R4	4	1
R5	5	1
R6	6	1
R7	7	1

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

- Se distribuyen i1, i2, i3 e i4.
- i5 queda en espera porque solo se distribuyen 4 inst.
- Se establecen los bits de validez para R1, R4 y R5 como ocupados (0) porque son destinos de oper.
- El resto de Ri se marcan como válidos (1).
- En las estaciones de reserva se ponen en los registros válidos sus valores y sus Vi=1, en los no válidos el Vi=0 y su índice Ri.
- i1 e i4 se marcan como listas.

Ciclo i+1: Distribución de i1, i2, i3 e i4

	O	CO	Op1	Op2	D
	0				
	0				
	0				
	0				
i5	1	MULT	R4	R5	R6

	Datos	V
R0	0	1
R1	1	0
R2	2	1
R3	35	1
R4	4	0
R5	5	0
R6	6	1
R7	7	1

	O	CO	Op1	V1	Op2	V2	D	L
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
i3	1	ADD	2	1	R1	0	R5	0
i1	1	SUB	7	1	0	1	R1	1

	O	CO	Op1	V1	Op2	V2	D	L
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
i4	1	DIV	35	1	7	1	R4	1
i2	1	MULT	R1	0	7	1	R5	0

i1: SUB R1, R7, R0

i2: MULT R5, R1, R7

i3: ADD R5, R2, R1

i4: DIV R4, R3, R7

i5: MULT R6, R4, R5

(a) Secuencia de instrucciones

- Se emiten i1 e i4.
- Se recibe i5.
- Se marca R6 con bit validez a 0 porque es destino de oper.

Ciclo i+2: Se ejecutan i1 e i4

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

	Datos	V
R0	0	1
R1	1	0
R2	2	1
R3	35	1
R4	4	0
R5	5	0
R6	6	0
R7	7	1

	O	CO	Op1	V1	Op2	V2	D	L
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
i3	1	ADD	2	1	R1	0	R5	0

	O	CO	Op1	V1	Op2	V2	D	L
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
i5	1	MULT	R4	0	R5	0	R6	0
i2	1	MULT	R1	0	7	1	R5	0

i1: SUB R1, R7, R0
i2: MULT R5, R1, R7
i3: ADD R5, R2, R1
i4: DIV R4, R3, R7
i5: MULT R6, R4, R5

(a) Secuencia de instrucciones

Final ciclo i+2: Finalización de i1. Quedan disponibles i2 e i3

- Termina i1.
  - R7-R0 = 7-0=7 → R1.
  - Se actualiza el bit de validez de R1 a 1 y el contenido de los registros R1.
  - Se marcan i2 e i3 como listas.

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

	Datos	V
R0	0	1
R1	7	1
R2	2	1
R3	35	1
R4	4	0
R5	5	0
R6	6	0
R7	7	1

	O	CO	Op1	V1	Op2	V2	D	L
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
i3	1	ADD	2	1	7	1	R5	1

	O	CO	Op1	V1	Op2	V2	D	L
	0	---	---	--	---	--	---	--
	0	---	---	--	---	--	---	--
i5	1	MULT	R4	0	R5	0	R6	0
i2	1	MULT	7	1	7	1	R5	1

Figura 2.35: [Continuación] Ejemplo de aplicación de la planificación con lectura de operandos a una secuencia de 5 instrucciones (continúa).

- Se emiten i2 e i3.
- En la UF MULT/DIV hay dos instrucciones. Pero es segmentada y es posible.

Ciclo i+3: Se emiten i2 e i3

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

	Datos	V
R0	0	1
R1	7	1
R2	2	1
R3	35	1
R4	4	0
R5	5	0
R6	6	0
R7	7	1

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
i5 1	MULT	R4	0	R5	0	R6	0

i1: SUB R1, R7, R0
i2: MULT R5, R1, R7
i3: ADD R5, R2, R1
i4: DIV R4, R3, R7
i5: MULT R6, R4, R5

(a) Secuencia de instrucciones

Final ciclo i+3: Finalización de i3 e i4. Queda disponible i5

- Terminan i3 e i4.
  - i3→ (R2+R1→ R5) → 7+2=9 → R5=9.
  - i4→ (R3/R7→R4)→ 35/7=5 → R4=5.
  - Se actualiza i5 en la estación de reserva y queda lista para emisión en el siguiente ciclo.
  - No se puede emitir porque su UF está ocupada con i2 y la terminación de i4.

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

	Datos	V
R0	0	1
R1	7	1
R2	2	1
R3	35	1
R4	5	1
R5	9	1
R6	6	0
R7	7	1

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
i5 1	MULT	5	1	9	1	R6	1

Figura 2.35: [Continuación] Ejemplo de aplicación de la planificación con lectura de operandos a una secuencia de 5 instrucciones (continúa).

- Se emite i5.

Ciclo i+4: Se emite i5

i1: SUB R1, R7, R0
i2: MULT R5, R1, R7
i3: ADD R5, R2, R1
i4: DIV R4, R3, R7
i5: MULT R6, R4, R5

(a) Secuencia de instrucciones

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

	Datos	V
R0	0	1
R1	7	1
R2	2	1
R3	35	1
R4	5	1
R5	9	1
R6	6	0
R7	7	1

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

- Terminan i2.
  - i2→ (R7*R1→ R5) → 7*7=49 → R5=49.

Final ciclo i+4: Finalización de i2

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

	Datos	V
R0	0	1
R1	7	1
R2	2	1
R3	35	1
R4	5	1
R5	49	1
R6	6	0
R7	7	1

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

Figura 2.35: [Continuación] Ejemplo de aplicación de la planificación con lectura de operandos a una secuencia de 5 instrucciones (continúa).

Ciclo i+5: Continúa el procesamiento de i5

- Continua el 2° ciclo de i5.

	O	CO	Op1	Op2	D
	0				
	0				
	0				
	0				
	0				

	Datos	V
R0	0	1
R1	7	1
R2	2	1
R3	35	1
R4	5	1
R5	49	1
R6	6	0
R7	7	1

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

i1: SUB R1, R7, R0
i2: MULT R5, R1, R7
i3: ADD R5, R2, R1
i4: DIV R4, R3, R7
i5: MULT R6, R4, R5

(a) Secuencia de instrucciones

Final ciclo i+5: Finalización de i5

- Terminan i5.
  - i5→ (R4*R5→ R6) → 5*9=45 → R5=45.
  - Se actualizan R5 y su bit de validez (1)

	O	CO	Op1	Op2	D
	0				
	0				

	Datos	V
R0	0	1
R1	7	1
R2	2	1
R3	35	1
R4	5	1
R5	49	1
R6	45	1
R7	7	1

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

O	CO	Op1	V1	Op2	V2	D	L
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--
0	---	---	--	---	--	---	--

- Este el valor que debería dar si no hubiese violación de dependencia WAW.
- Sin violación las instrucciones son:
  - i1: SUB R1,R7,R0 → 7 – 0 = 7 → R1 = 7
  - i2: MULT R5,R1,R7 → 7 * 7 = 49 → R5 = 49
  - i3: ADD R5,R2,R1 → 7+2= 9 → R5=9
  - i4: DIV R4,R3,R7 → 35/7=5 → R4=5
  - i5: MULT R6,R4,R5 → 9*5=45 → R6=45
- Por el orden en el que se ejecutan las instrucciones:
  - Orden real: i1, i3, i4, i2, i5.
  - i1: SUB R1,R7,R0 → 7 – 0 = 7 → R1 = 7
  - i3: ADD R5,R2,R1 → 7+2= 9 → R5=9
  - i4: DIV R4,R3,R7 → 35/7=5 → R4=5
  - i2: MULT R5,R1,R7→ 7*7=49 → R5=49
  - i5: MULT R6,R4,R5 → 5*9=45 → R6=45

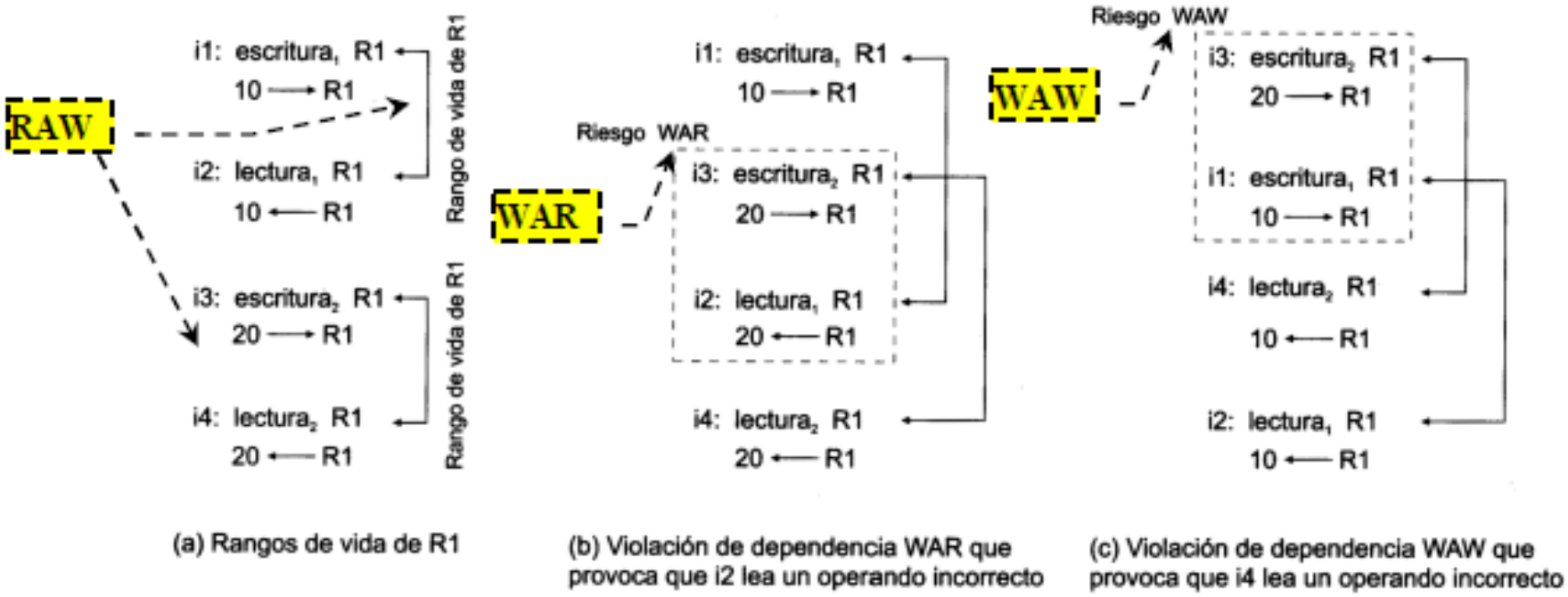
SE EVITARÍA CON LA TÉCNICA DE RENOMBRAMIENTO DE REGISTROS.

Ejemplo de aplicación de la planificación con lectura de operandos a una secuencia

2.7.4. Renombramiento de Registros

- El compilador utiliza un número infinito de registros simbólicos
- El número de registros arquitectónicos son limitados
  - El compilador reutiliza los registros arquitectónicos, escribe un nuevo valor cunado detecta que el valor almacenado ya no es necesario
- Fichero de registros Arquitectonicos (ARF *Architected Register File*)
  - Son los registros accesibles al programador
  - Su número son limitados
- Rango de vida de un registro
  - El tiempo desde que se almacena el valor en el registro hasta que hace uso de ese valor por última vez, antes de reemplazarlo mediante una nueva escritura

```
i1: ADDI  R1,R0,#10
....................
i2: MULTI R2,R1,#40
i3: ADDI  R1,R0,#20
....................
i4: MULTI R3,R1,#80
```



(a) Rangos de vida de R1

(b) Violación de dependencia WAR que provoca que i2 lea un operando incorrecto

(c) Violación de dependencia WAW que provoca que i4 lea un operando incorrecto

Figura 2.36: Rangos de vida de un registro. Los diferentes posibles solapamientos de dos rangos de vida de un mismo registro inducen diferentes tipos de riesgos.

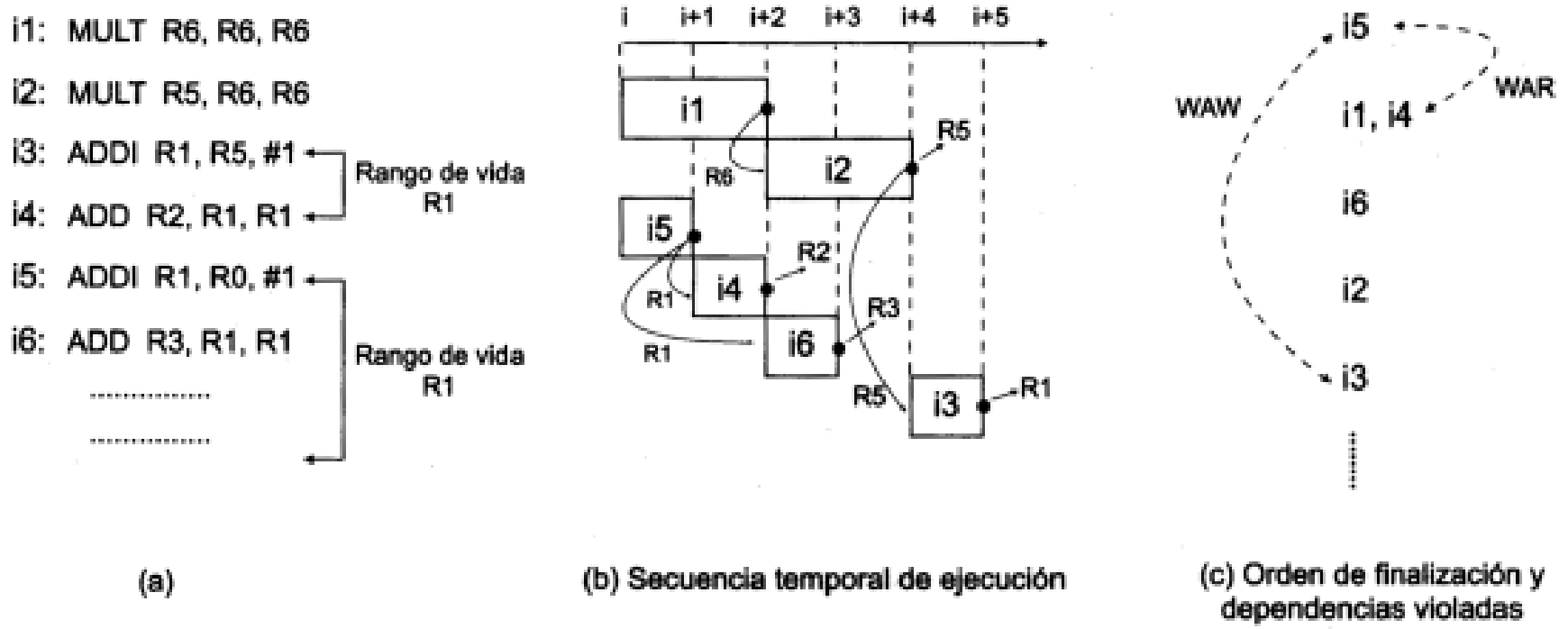


Figura 2.37: Violación de dependencias WAR y WAW como consecuencia de la ejecución fuera de orden.

Una solución inmediata y sencilla

- La ejecución secuencial de las instrucciones y la escritura ordenada de los registros
- Elimina ejecución fuera de orden
  - Bajo rendimiento.

Solución

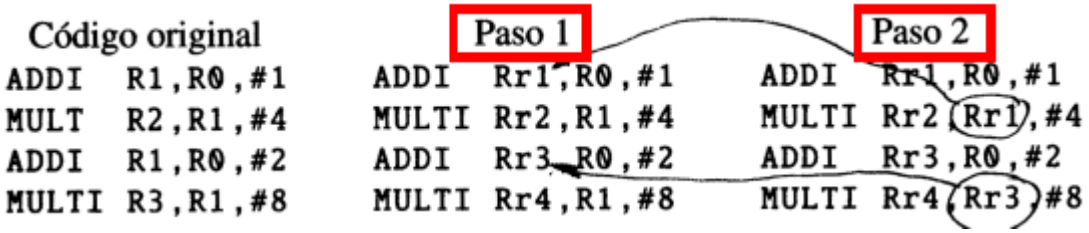
- Renombramiento dinámico de los registros de la arquitectura mediante hardware

Renombramiento dinámico de registros de la arquitectura mediante hardware

- Consiste en utilizar registros auxiliares invisibles al programador de forma que se reestablezca la correspondencia única entre resultados temporales y registros
  - Se conocen como **registros físicos, registros no creados o registros de renombramiento**
  - Las instrucciones escriben los resultados en los registros de renombramientos para posteriormente escribirlos en los arquitectónicos
    - Este restablecimiento permite eliminar todas las dependencias falsas entre las instrucciones emitidas

Pasos del renombramiento dinámico

- Paso 1
  - Resolución de los riesgos WAW y WAR
    - Se renombran de forma única los operandos destinos de la instrucción
- Paso 2
  - Mantenimiento de la dependencia RAW
    - Se renombran todos los registros fuentes que son objeto de una escritura previa utilizando en mismo nombre que se empleo en el paso uno



Renombramiento dinámico

- Ficheros de registro
  - Al fichero de registros de la arquitectura se denominará ARF (Architected Register File).
  - Fichero de Registros de Renombramientos (RRF – *Rename Register File*)
- Existen tres formas de organizar el RRF
  - Un único fichero de registro (ARF + RRF)
  - Como una estructura independiente pero accesible desde el ARF
  - Como un buffer de reordenamiento y accesible desde ARF

2.7.4.1. Organización independiente del RRF con acceso indexado

- 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.

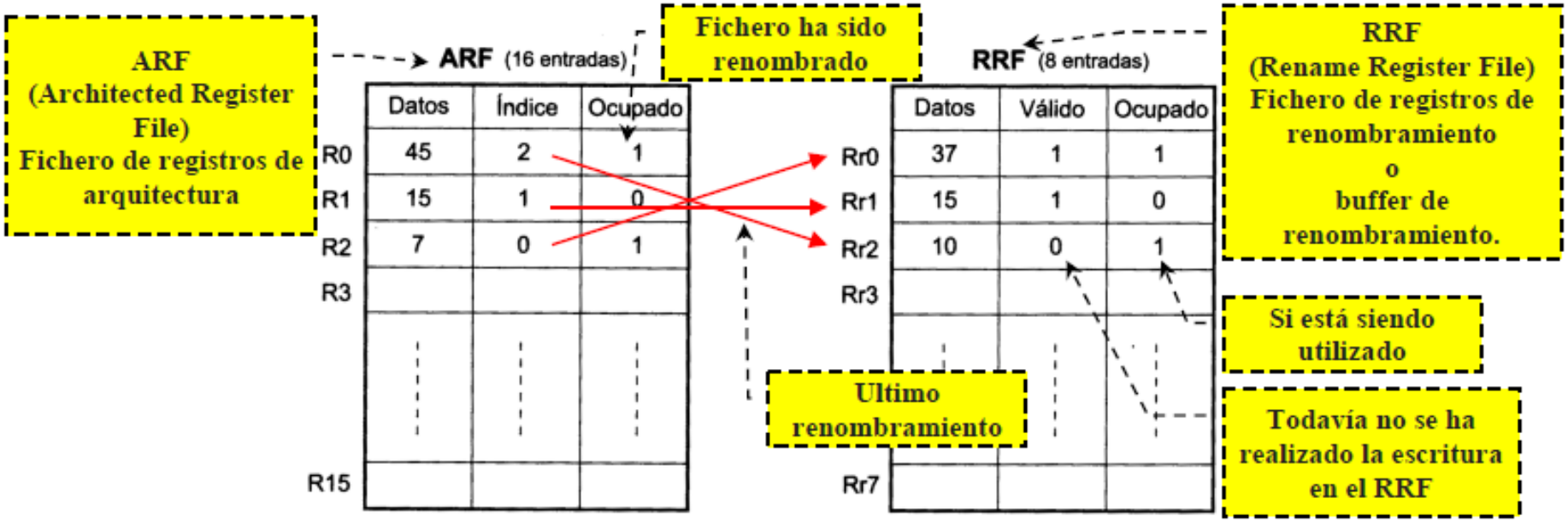


Figura 2.38: Organización del RRF como estructura independiente del ARF con acceso indexado.

Proceso de renombramiento de registros

- 1. Se accede mediante su identificador a la entrada que le corresponde en el ARF, se establece el bit de Ocupado a 1, se selecciona un registro de renombramiento del RRF que esté libre y se copia el identificador del registro RRF seleccionado en el campo Índice.
  - Ocupado en RRF → 1
  - Válido → 0
- 2. Instrucción finaliza, la escritura del resultado de la operación en RRF y bit Válido a 1.
- La escritura diferida del valor del RRF al ARF cuando la instrucción termina

- R0 → Ocupado (ARF) y No válido (RRF) → Pendiente de escritura.
- R2 → Ocupado (ARF) y Válido y Ocupado (RRF) → Instrucción finalizada pero no terminada
- R1 → No ocupado (ARF) → Instrucción terminada

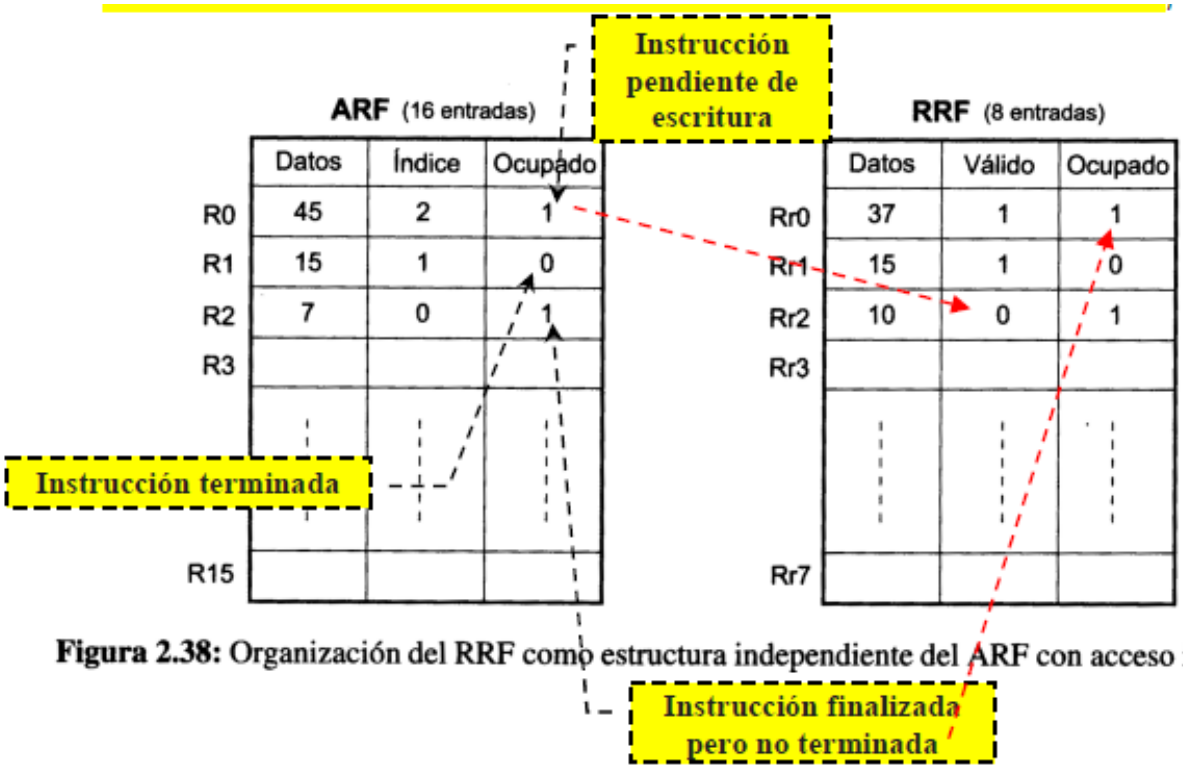


Figura 2.38: Organización del RRF como estructura independiente del ARF con acceso indexado.

2.7.4.2. Organización independiente del RRF con acceso asociativo

- 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
- Ultimo:
  - Si está a uno es el ultimo renombramiento del registro del ARF
    - El último registro de renombramiento pasa de 1 a 0, y el nuevo de 0 a 1
- Destino
  - Almacena el identificador del registro ARF, se utiliza para realizar la búsqueda y validad la coincidencia

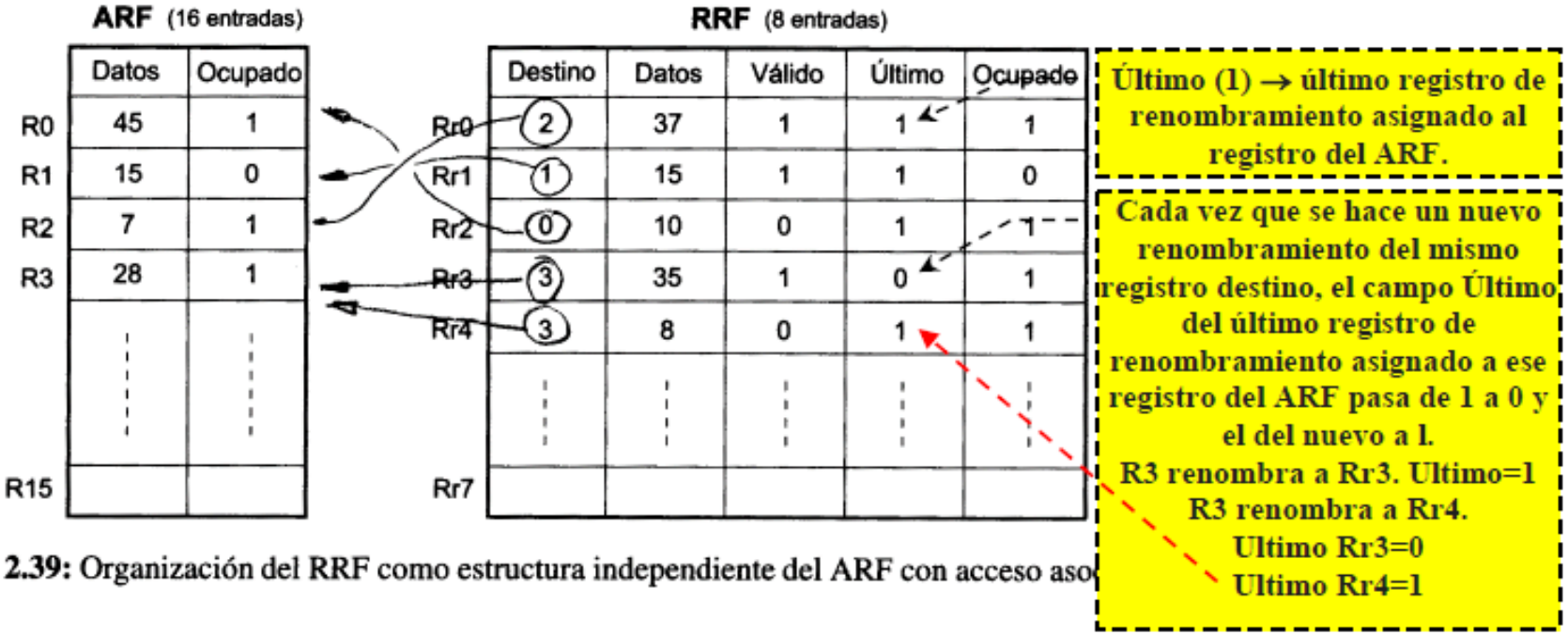


Figura 2.39: Organización del RRF como estructura independiente del ARF con acceso aso

2.7.4.3. Organización del RRF como parte del buffer de reordenamiento

- Como parte del buffer de reordenamiento o terminación
  - Permite recuperar el orden de las instrucciones y concluir su procesamiento manteniendo la consistencia semántica del programa
- Para incluir el RRF en el buffer de reordenamiento añadimos los campos Datos y Válido con la misma función que ya hemos visto
  - Datos
  - Válido
- Si al realizar la distribución de una instrucción se accede al ARF para efectuar la lectura de un operando, las situaciones posibles son:
  - 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. Pueden acontecer dos situaciones:
    - Válido está a 0 es que se está a la espera de la finalización de la instrucción.
    - Válido está a 1 es que se está pendiente de la terminación de la instrucción.

Condiciones para poder distribuir una instrucción

- 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

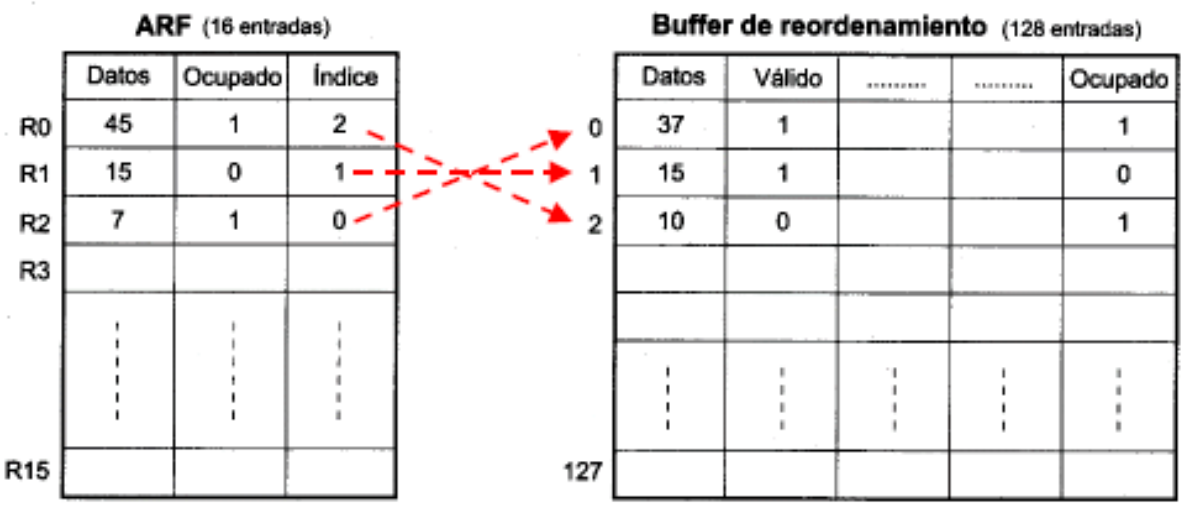


Figura 2.40: Implementación del RRF en el buffer de reordenamiento.

2.7.5. Ejemplo de procesamiento de instrucciones con renombramiento

Los accesos al ARF y al RRF se realizan en el mismo ciclo en que se distribuye la instrucción (el renombramiento no penaliza).

- Se pueden finalizar dos instrucciones simultáneamente.
- Las instrucciones son terminadas en el ciclo siguiente a su finalización.

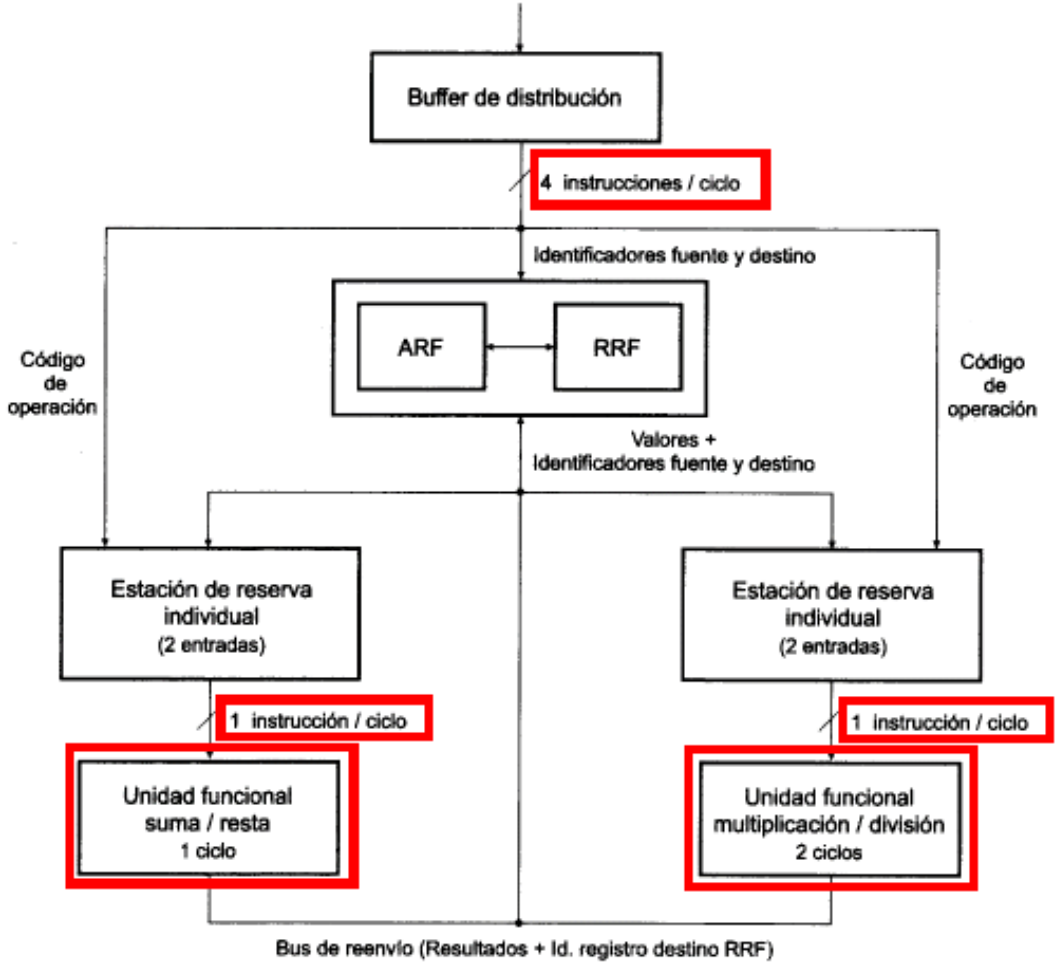
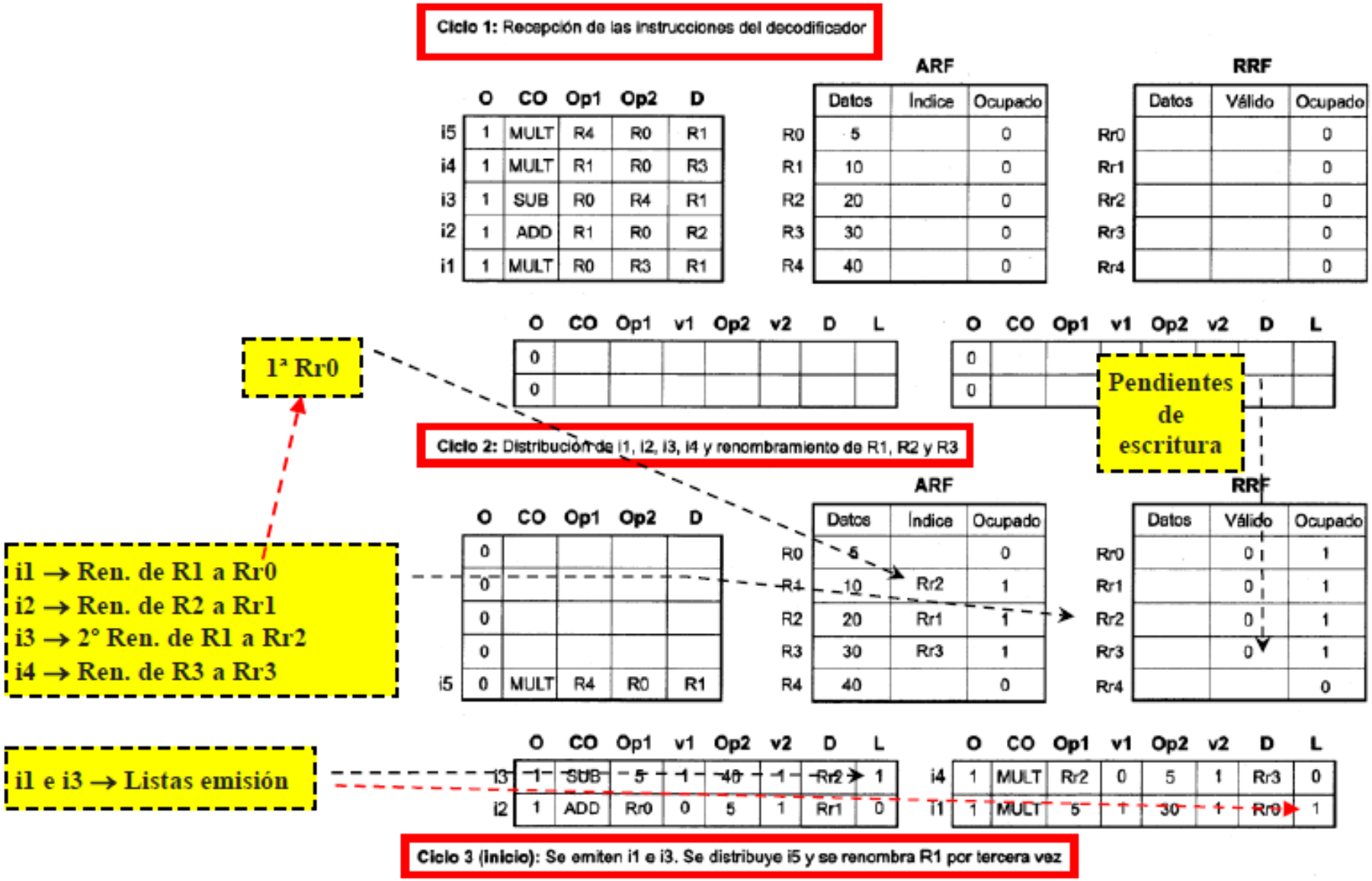


Figura 2.41: Núcleo de ejecución con planificación dinámica con lectura de operandos y RRF independiente con acceso indexado.


Ciclo 1: Recepción de las instrucciones del decodificador
ARF
RRF
O CO Op1 Op2 D
i5 1 MULT R4 R0 R1
i4 1 MULT R1 R0 R3
i3 1 SUB R0 R4 R1
i2 1 ADD R1 R0 R2
i1 1 MULT R0 R3 R1
Datos Índice Ocupado
R0 5 0
R1 10 0
R2 20 0
R3 30 0
R4 40 0
Datos Válido Ocupado
Rr0 0
Rr1 0
Rr2 0
Rr3 0
Rr4 0
O CO Op1 v1 Op2 v2 D L
1ª Rr0
Pendientes de escritura
Ciclo 2: Distribución de i1, i2, i3, i4 y renombramiento de R1, R2 y R3
i1 → Ren. de R1 a Rr0
i2 → Ren. de R2 a Rr1
i3 → 2° Ren. de R1 a Rr2
i4 → Ren. de R3 a Rr3
i5 0 MULT R4 R0 R1
R0 5
R1 10 Rr2 1
R2 20 Rr1 1
R3 30 Rr3 1
R4 40 0
Rr0 0 1
Rr1 0 1
Rr2 0 1
Rr3 0 1
Rr4 0
i1 e i3 → Listas emisión
i3 1 SUB 5 1 40 1 Rr2 1
i2 1 ADD Rr0 0 5 1 Rr1 0
i4 1 MULT Rr2 0 5 1 Rr3 0
i1 1 MULT 5 1 30 1 Rr0 1
Ciclo 3 (Inicio): Se emiten i1 e i3. Se distribuye i5 y se renombra R1 por tercera vez

i1: MULT R1, R0, R3

i2: ADD R2, R1, R0

i3: SUB R1, R0, R4

i4: MULT R3, R1, R0

i5: MULT R1, R4, R0

i5 → 3° Ren. de R1 a Rr4

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

ARF

	Datos	Índice	Ocupado
R0	5		0
R1	10	Rr4	1
R2	20	Rr1	1
R3	30	Rr3	1
R4	40		0

RRF

	Datos	Válido	Ocupado
Rr0		0	1
Rr1		0	1
Rr2		0	1
Rr3		0	1
Rr4		0	1

	O	CO	Op1	v1	Op2	v2	D	L
	0							
i2	1	ADD	Rr0	0	5	1	Rr1	0

	O	CO	Op1	v1	Op2	v2	D	L
i5	1	MULT	40	1	5	1	Rr4	1
i4	1	MULT	Rr2	0	5	1	Rr3	0

I5 → No se emite pq UF ocupada con i1

Ciclo 3 (final): Concluye i3 y se publica su resultado (-35) y su destino (Rr2)

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

ARF

	Datos	Índice	Ocupado
R0	5		0
R1	10	Rr4	1
R2	20	Rr1	1
R3	30	Rr3	1
R4	40		0

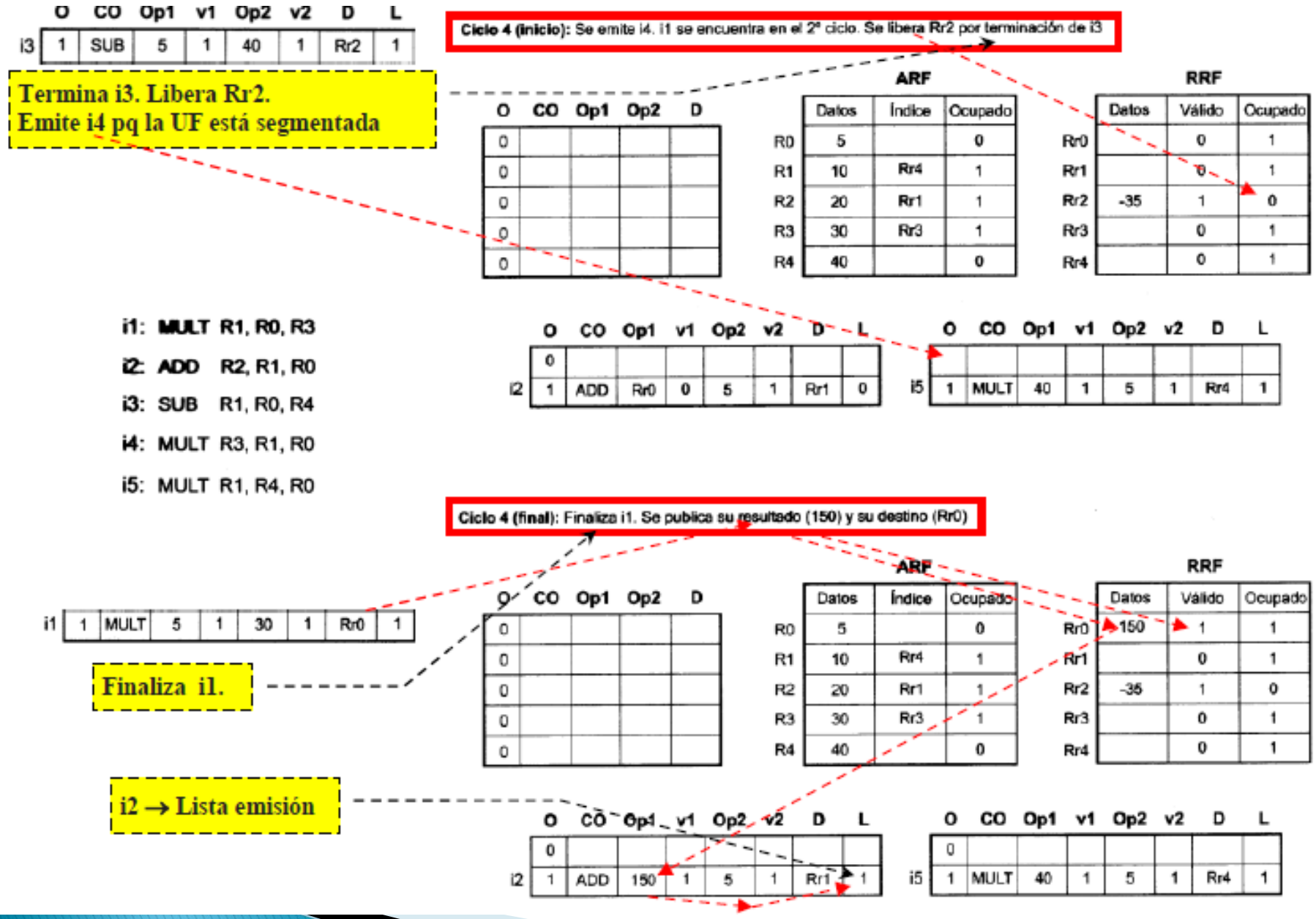
RRF

	Datos	Válido	Ocupado
Rr0		0	1
Rr1		0	1
Rr2	-35	1	1
Rr3		0	1
Rr4		0	1

	O	CO	Op1	v1	Op2	v2	D	L
	0							
i2	1	ADD	Rr0	0	5	1	Rr1	0

	O	CO	Op1	v1	Op2	v2	D	L
i5	1	MULT	40	1	5	1	Rr4	1
i4	1	MULT	-35	1	5	1	Rr3	1

i4 → Lista emisión


O CO Op1 v1 Op2 v2 D L
i3 1 SUB 5 1 40 1 Rr2 1
Termina i3. Libera Rr2.
Emite i4 pq la UF está segmentada
Ciclo 4 (inicio): Se emite i4. i1 se encuentra en el 2º ciclo. Se libera Rr2 por terminación de i3
ARF
RRF
i1: MULT R1, R0, R3
i2: ADD R2, R1, R0
i3: SUB R1, R0, R4
i4: MULT R3, R1, R0
i5: MULT R1, R4, R0
i2 1 ADD Rr0 0 5 1 Rr1 0
i5 1 MULT 40 1 5 1 Rr4 1
Ciclo 4 (final): Finaliza i1. Se publica su resultado (150) y su destino (Rr0)
i1 1 MULT 5 1 30 1 Rr0 1
Finaliza i1.
i2 → Lista emisión
i2 1 ADD 150 1 5 1 Rr1 1
i5 1 MULT 40 1 5 1 Rr4 1

Ciclo 5 (inicio): Se emiten i2 e i5. i4 se encuentra en el 2° ciclo. Se libera Rr0 por terminación de i1

Termina i1. Libera Rr0.

Se puede emitir i5 pq i4 está en el 2° ciclo y la UF es segmentada.

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

ARF

	Datos	Índice	Ocupado
R0	5		0
R1	10	Rr4	1
R2	20	Rr1	1
R3	30	Rr3	1
R4	40		0

RRF

	Datos	Válido	Ocupado
Rr0	150	1	0
Rr1		0	1
Rr2	-35	1	0
Rr3		0	1
Rr4		0	1

O	CO	Op1	v1	Op2	v2	D	L
0							
0							

O	CO	Op1	v1	Op2	v2	D	L
0							
0							

i1: MULT R1, R0, R3

i2: ADD R2, R1, R0

i3: SUB R1, R0, R4

i4: MULT R3, R1, R0

i5: MULT R1, R4, R0

Ciclo 5 (final): Finaliza i4 y se publica su resultado (-175) y su destino (Rr3)
Finaliza i2 y se publica su resultado (155) y su destino (Rr1)

i4	1	MULT	-35	1	5	1	Rr3	1
i2	1	ADD	150	1	5	1	Rr1	1

Finaliza i4. Libera Rr3.
Finaliza i2. Libera Rr1.

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

ARF

	Datos	Índice	Ocupado
R0	5		0
R1	10	Rr4	1
R2	20	Rr1	1
R3	30	Rr3	1
R4	40		0

RRF

	Datos	Válido	Ocupado
Rr0	150	1	0
Rr1	155	1	1
Rr2	-35	1	0
Rr3	-175	1	1
Rr4		0	1

O	CO	Op1	v1	Op2	v2	D	L

O	CO	Op1	v1	Op2	v2	D	L

Ciclo 6 (inicio): Termina i2, se actualiza R2 y se libera Rr1
Termina i4, se actualiza R3 y se libera Rr3
i_5 se encuentra en el 2° ciclo

Termina i2. Se actualiza R2 y se libera Rr1
Termina i4. Se actualiza R3 y se libera Rr3

i4	1	MULT	-35	1	5	1	Rr3	1
i2	1	ADD	150	1	5	1	Rr1	1

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

ARF

	Datos	Índice	Ocupado
R0	5		0
R1	10	Rr4	1
R2	155	Rr1	0
R3	-175	Rr3	0
R4	40		0

RRF

	Datos	Válido	Ocupado
Rr0	150	1	0
Rr1	155	1	0
Rr2	-35	1	0
Rr3	-175	1	0
Rr4		0	1

O	CO	Op1	v1	Op2	v2	D	L

O	CO	Op1	v1	Op2	v2	D	L

i1: **MULT** R1, R0, R3
i2: **ADD** R2, R1, R0
i3: SUB R1, R0, R4
i4: MULT R3, R1, R0
i5: MULT R1, R4, R0

i5	1	MULT	40	1	5	1	Rr4	1

Ciclo 6 (final): Finaliza i5, se publica su resultado (200) y su destino (Rr4)

O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

ARF

	Datos	Índice	Ocupado
R0	5		0
R1	10	Rr4	1
R2	155	Rr1	0
R3	-175	Rr3	0
R4	40		0

RRF

	Datos	Válido	Ocupado
Rr0	150	1	0
Rr1	155	1	0
Rr2	-35	1	0
Rr3	-175	1	0
Rr4	200	1	1

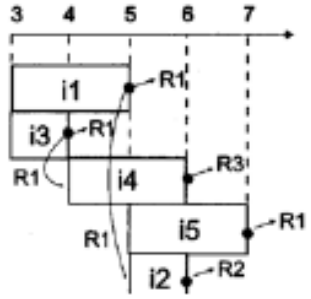
O	CO	Op1	v1	Op2	v2	D	L

O	CO	Op1	v1	Op2	v2	D	L

Ciclo 7 (inicio): Termina i5, se actualiza R1 y se libera Rr4

Termina i5. Se actualiza R1 y se libera Rr4

Secuencia de ejecución



O	CO	Op1	Op2	D
0				
0				
0				
0				
0				

ARF

	Datos	Índice	Ocupado
R0	5		0
R1	200	Rr4	0
R2	155	Rr1	0
R3	-175	Rr3	0
R4	40		0

RRF

	Datos	Válido	Ocupado
Rr0	150	1	0
Rr1	155	1	0
Rr2	-35	1	0
Rr3	-175	1	0
Rr4	200	1	0

O	CO	Op1	v1	Op2	v2	D	L

O	CO	Op1	v1	Op2	v2	D	L

SECUENCIA ORIGINAL Y RESULTADO RENOMBRAMIENTO

Registros antes

R0	5
R1	10
R2	20
R3	30
R4	40

Secuencia instrucciones

i1: MULT R1,R0,R3	R1 ← 150 ← 5*30
i2: ADD R2,R1,R0	R2 ← 155 ← 150+5
i3: SUB R1,R0,R4	R1 ← -35 ← 5-40
i4: MULT R3,R1,R0	R3 ← -175 ← -35*5
i5: MULT R1,R4,R0	R1 ← 200 ← 40*5

Registros después

R0	5
R1	200
R2	155
R3	-175
R4	40

SI NO HUBIESE HABIDO RENOMBRAMIENTO

Registros antes

R0	5
R1	10
R2	20
R3	30
R4	40

Secuencia de ejecución instrucciones

i3: SUB R1,R0,R4	R1 ← -35 ← 5-40
i1: MULT R1,R0,R3	R1 ← 150 ← 5*30
i4: MULT R3,R1,R0	R3 ← 750 ← 150*5
i2: ADD R2,R1,R0	R2 ← 155 ← 150+5
i5: MULT R1,R4,R0	R1 ← 200 ← 40*5

Registros después

R0	5
R1	200
R2	155
R3	750
R4	40

2.8. Terminación

- Consistencia del procesador
  - Cuando las instrucciones concluyen su procesamiento en el mismo orden secuencial en el que se encuentran en el programa, es decir, en el mismo orden en que iniciaron su procesamiento
- Razones para mantener la consistencia
  - Garantizar el resultado final del programa
  - Permitir un tratamiento correcto de las interrupciones
- Las instrucciones que han terminado su etapa de ejecución se almacenan en el buffer de reordenamiento a la espera de su terminación arquitectónica
- Instrucción terminada arquitectónicamente
  - Cuando actualiza el estado del procesador manteniendo la consistencia
- Consistencia del procesador
  - Cuando las instrucciones concluyen su procesamiento en el mismo orden en que se encuentran en el programa
- Una instrucción ha **FINALIZADO**
  - Cuando abandona la unidad funcional y espera en el buffer
- Una Instrucción ha **TERMINADO**
  - Cuando ha actualizado el estado de la máquina
- Una instrucción se ha **RETIRADO**
  - Cuando ha escrito su resultado en memoria (SOLO INSTRUCCIONES DE ALMACENAMIENTO)

Buffer de reordenamiento

- El buffer de reordenamiento es la pieza del procesador que pone fin a la ejecución fuera de orden de las instrucciones.
  - Éste decide cuándo los resultados almacenados temporalmente en el RRF tienen que escribirse en el ARF y una instrucción puede darse por concluida (salvo los almacenamientos).
- Su principio de funcionamiento se basa en una Cola que va a medida que va escribiendo en los registros va incrementando su puntero de inicio.
  - Para ello cuenta con dos punteros, el de inicio y el de final
- 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.

Campos de una entrada del buffer de reordenamiento

- *Ocupada* (0): Es un campo de un bit que indica que la instrucción se ha distribuido. Permanece a 1 hasta que es terminada, momento en que se coloca a 0.
- *Emitida* (E): Campo de un bit que pasa a valer 1 cuando la instrucción inicia su ejecución en una unidad funcional.
- *Finalizada* (F): Campo de un bit que indica que la instrucción ha salido de la unidad funcional y espera ser terminada arquitectónicamente.
- *Dirección* (Dir): Campo que contiene la dirección de memoria de la instrucción como forma de identificarla. Aunque no se ha indicado explícitamente en las secciones previas, la dirección en memoria de una instrucción la acompaña a lo largo de toda la segmentación como etiqueta identificativa. De esta forma, se puede saber en qué estado de procesamiento se encuentra.
- *Registro de destino* (Rd): Es el identificador del registro de destino asociado a la instrucción. Se actualiza una vez que se termine la instrucción para garantizar el correcto tratamiento de las excepciones. Se libera cuando no hay renombramientos pendientes.
- *Registro de renombramiento* (Rr): Es el identificador del registro de renombramiento asociado a la instrucción. De esta forma se sabe el registro de renombramiento que hay que liberar en el RRF.
- *Especulativa* (Es): Campo que identifica a la instrucción como parte de una ruta especulativa. No se pueden terminar aquellas instrucciones marcadas como especulativas.
- *Validez* (V): Campo de un bit para saber si la instrucción puede terminarse o hay que ignorarla y no realizar acción alguna cuando le corresponda su terminación. Desde el instante en que la instrucción entra en el buffer de reordenamiento se considera válida. Sin embargo, posteriormente

a) Campos si se considera RRF como estructura independiente
b) RRF como parte del buffer de reordenamiento
c) Ejemplo de a) RRF independiente

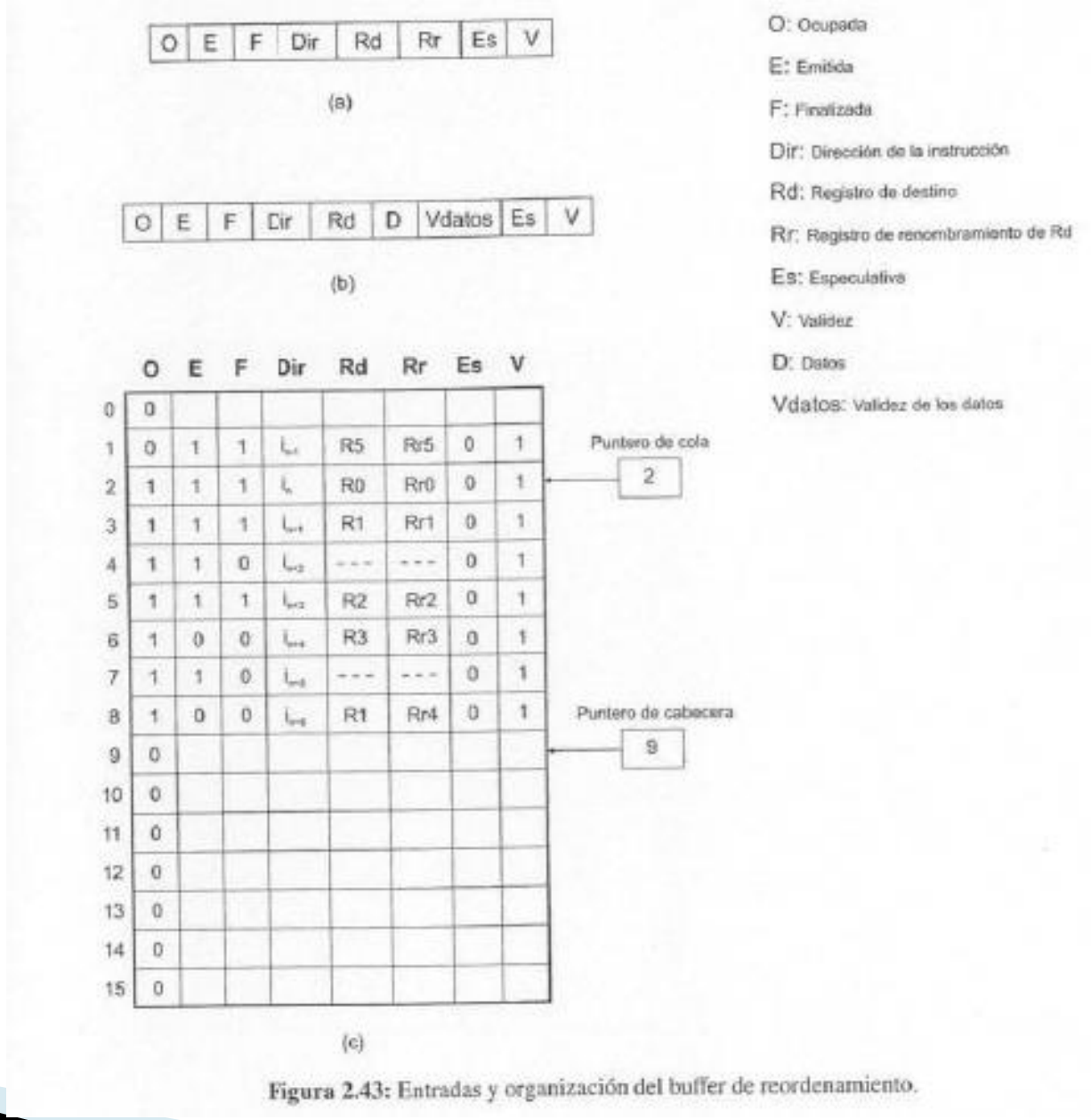
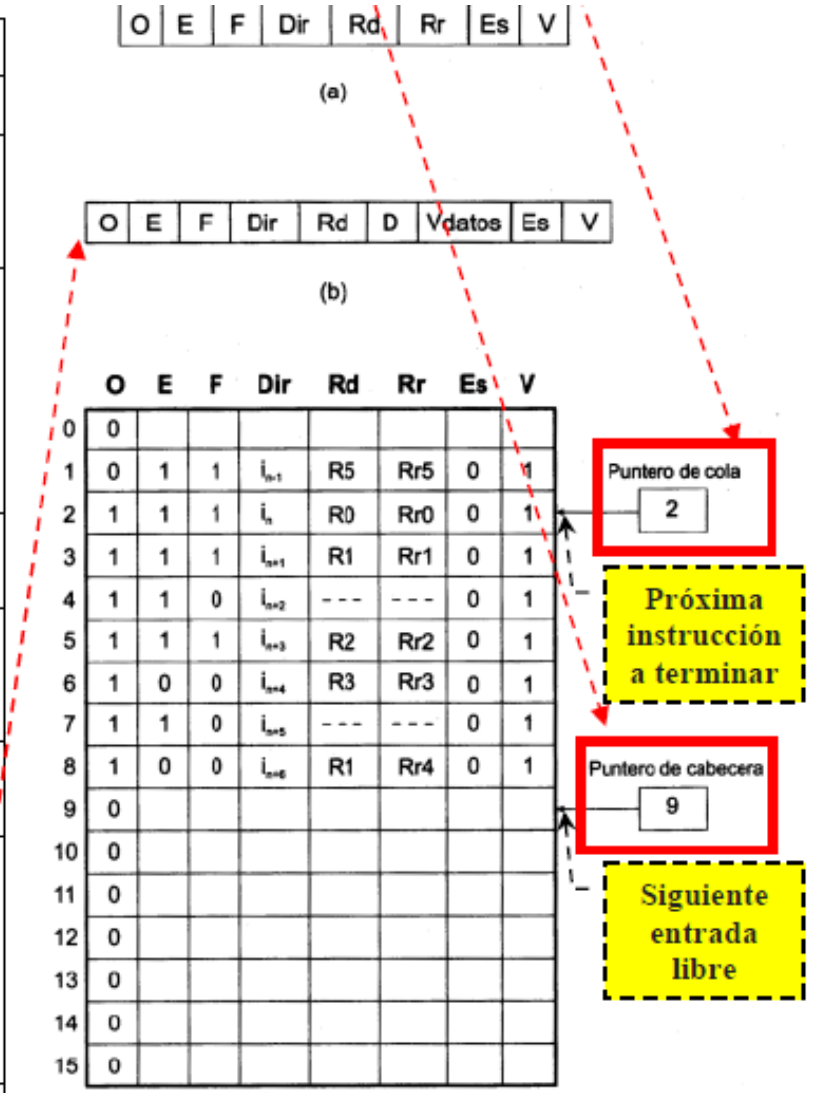


Figura 2.43: Entradas y organización del buffer de reordenamiento.

Ocupada (O):	Instrucción distribuida
Emitida (E):	Inicia ejecución en unidad funcional
Finalizada (F):	Ha salido de la unidad funcional y espera ser terminada arquitectónicamente.
Dirección (Dir):	Contiene la dirección de memoria de la instrucción. La dirección en memoria de una instrucción la acompaña a lo largo de toda la segmentación como etiqueta identificativa.
Registro de destino (Rd):	
Registro de renombramiento (Rr):	
Especulativa (Es):	Identifica a la instrucción como parte de una ruta especulativa.
Validez (V):	Si la instrucción puede terminarse o hay que ignorarla. Cuando la instrucción entra en el buffer de reordenamiento se considera válida. Puede pasar a ser inválida si se trata de una instrucción especulativa.
Si el RRF fuese parte del buffer de terminación	
Datos (D)	**eliminar el campo Rr**
Validez de los datos (Vdatos)	



- Cuando una instrucción es terminada arquitectónicamente sucede lo siguiente:
  - 1. Rr se libera.
  - 2. Rd se actualiza con el valor de su registro de renombramiento Rr asociado y se libera si no tiene renombramientos pendientes.
  - 3. Ocupado → 0.
  - 4. El puntero de cola se incrementa para apuntar a la siguiente instrucción a terminar.
- Como mínimo será necesario disponer de tantos puertos de escritura en el ARF como instrucciones se quieran terminar por ciclo de reloj.
  - Vpico = Máx. instrucciones terminadas por ciclo * Frecuencia procesador

```
i1: MULT R1,R0,R3
i2: ADD  R2,R1,R0
i3: SUB  R1,R0,R4
i4: MULT R3,R1,R0
i5: MULT R1,R4,R0
```

Ciclo 2: Se distribuye i1, i2, i3 e i4

	O	E	F	Dir	Rd	Rr	Es	V
0	1	0	0	i1	R1	Rr0	0	1
1	1	0	0	i2	R2	Rr1	0	1
2	1	0	0	i3	R1	Rr2	0	1
3	1	0	0	i4	R3	Rr3	0	1
4								

Puntero de cola: 0

Puntero de cabecera: 4

Ciclo 3 (inicio): Se emiten i1 e i3. Se distribuye i5

	O	E	F	Dir	Rd	Rr	Es	V
0	1	1	0	i1	R1	Rr0	0	1
1	1	0	0	i2	R2	Rr1	0	1
2	1	1	0	i3	R1	Rr2	0	1
3	1	0	0	i4	R3	Rr3	0	1
4	1	0	0	i5	R1	Rr4	0	1

Puntero de cabecera: 0

Puntero de cola: 0

Ciclo 3 (final): Finaliza i3

	O	E	F	Dir	Rd	Rr	Es	V
0	1	1	0	i1	R1	Rr0	0	1
1	1	0	0	i2	R2	Rr1	0	1
2	1	1	1	i3	R1	Rr2	0	1
3	1	0	0	i4	R3	Rr3	0	1
4	1	0	0	i5	R1	Rr4	0	1

Puntero de cabecera: 0

Puntero de cola: 0

Ciclo 4 (inicio): Se emite i4

	O	E	F	Dir	Rd	Rr	Es	V
0	1	1	0	i1	R1	Rr0	0	1
1	1	0	0	i2	R2	Rr1	0	1
2	1	1	1	i3	R1	Rr2	0	1
3	1	1	0	i4	R3	Rr3	0	1
4	1	0	0	i5	R1	Rr4	0	1

Puntero de cabecera: 0

Puntero de cola: 0

Ciclo 4 (final): Finaliza i1

	O	E	F	Dir	Rd	Rr	Es	V
0	1	1	1	i1	R1	Rr0	0	1
1	1	0	0	i2	R2	Rr1	0	1
2	1	1	1	i3	R1	Rr2	0	1
3	1	1	0	i4	R3	Rr3	0	1
4	1	0	0	i5	R1	Rr4	0	1

Puntero de cabecera: 0

Puntero de cola: 0

Ciclo 5 (inicio): Termina i1, se actualiza R1 y se libera Rr0
No se libera R1 porque tiene dos renombramientos pendientes (Rr2 y Rr4)
Se libera la entrada i, del buffer. Se emiten i2 e i5

	O	E	F	Dir	Rd	Rr	Es	V
0	0	1	1	i1	R1	Rr0	0	1
1	1	1	0	i2	R2	Rr1	0	1
2	1	1	1	i3	R1	Rr2	0	1
3	1	1	0	i4	R3	Rr3	0	1
4	1	1	0	i5	R1	Rr4	0	1

Puntero de cabecera: 0

Puntero de cola: 1

Ciclo 5 (final): Finalizan i2 e i4

	O	E	F	Dir	Rd	Rr	Es	V
0	0	1	1	i1	R1	Rr0	0	1
1	1	1	1	i2	R2	Rr1	0	1
2	1	1	1	i3	R1	Rr2	0	1
3	1	1	1	i4	R3	Rr3	0	1
4	1	1	0	i5	R1	Rr4	0	1

Puntero de cabecera: 0

Puntero de cola: 1

Ciclo 6 (inicio): Termina i2, se actualiza R2 y se libera Rr1
Termina i3, se actualiza R1 y se libera Rr2
No se libera R1 porque tiene un renombramiento pendiente (Rr4)
Se liberan las entradas 1 y 2 del buffer

	O	E	F	Dir	Rd	Rr	Es	V
0	0	1	1	i1	R1	Rr0	0	1
1	0	1	1	i2	R2	Rr1	0	1
2	0	1	1	i3	R1	Rr2	0	1
3	1	1	1	i4	R3	Rr3	0	1
4	1	1	0	i5	R1	Rr4	0	1

Puntero de cabecera: 0
Puntero de cola: 3

```
i1:  MULT  R1,R0,R3
i2:  ADD   R2,R1,R0
i3:  SUB   R1,R0,R4
i4:  MULT  R3,R1,R0
i5:  MULT  R1,R4,R0
```

Ciclo 6 (final): Finaliza i5

	O	E	F	Dir	Rd	Rr	Es	V
0	0	1	1	i1	R1	Rr0	0	1
1	0	1	1	i2	R2	Rr1	0	1
2	0	1	1	i3	R1	Rr2	0	1
3	1	1	1	i4	R3	Rr3	0	1
4	1	1	1	i5	R1	Rr4	0	1

Puntero de cabecera: 0
Puntero de cola: 3

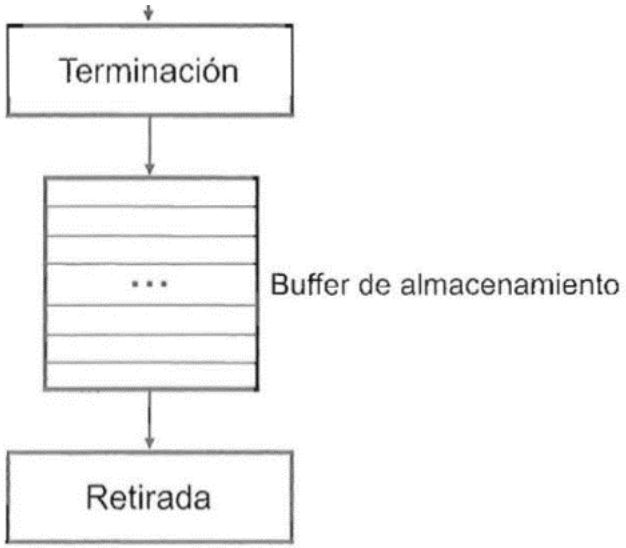
Ciclo 7 (inicio): Termina i4, se actualiza R3 y se libera Rr3
Termina i5, se actualiza R1 y se libera Rr4
Se liberan las entradas 3 y 4 del buffer

	O	E	F	Dir	Rd	Rr	Es	V
0	0	1	1	i1	R1	Rr0	0	1
1	0	1	1	i2	R2	Rr1	0	1
2	0	1	1	i3	R1	Rr2	0	1
3	0	1	1	i4	R3	Rr3	0	1
4	0	1	1	i5	R1	Rr4	0	1

Puntero de cabecera: 0
Puntero de cola: 0

2.9. Retirada

- Es exclusiva de las instrucciones de almacenamiento
  - 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
  - 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 (cuando se almacenan los datos).
- Las instrucciones de carga/almacenamiento (seg. Superescalar) una vez emitidas, consta de tres pasos:
  - Calculo de la dirección de memoria
  - Traducción de la dirección de memoria virtual a memoria física
  - Acceso a memoria
- Las instrucciones de carga realizan los tres pasos en la etapa de ejecución
- **Sintaxis de las instrucciones**




```
LD registro_destino,desplazamiento(registro_base)
SD desplazamiento(registro_base),registro_fuente
```

Pasos de ejecución de las instrucciones de carga y almacenamiento

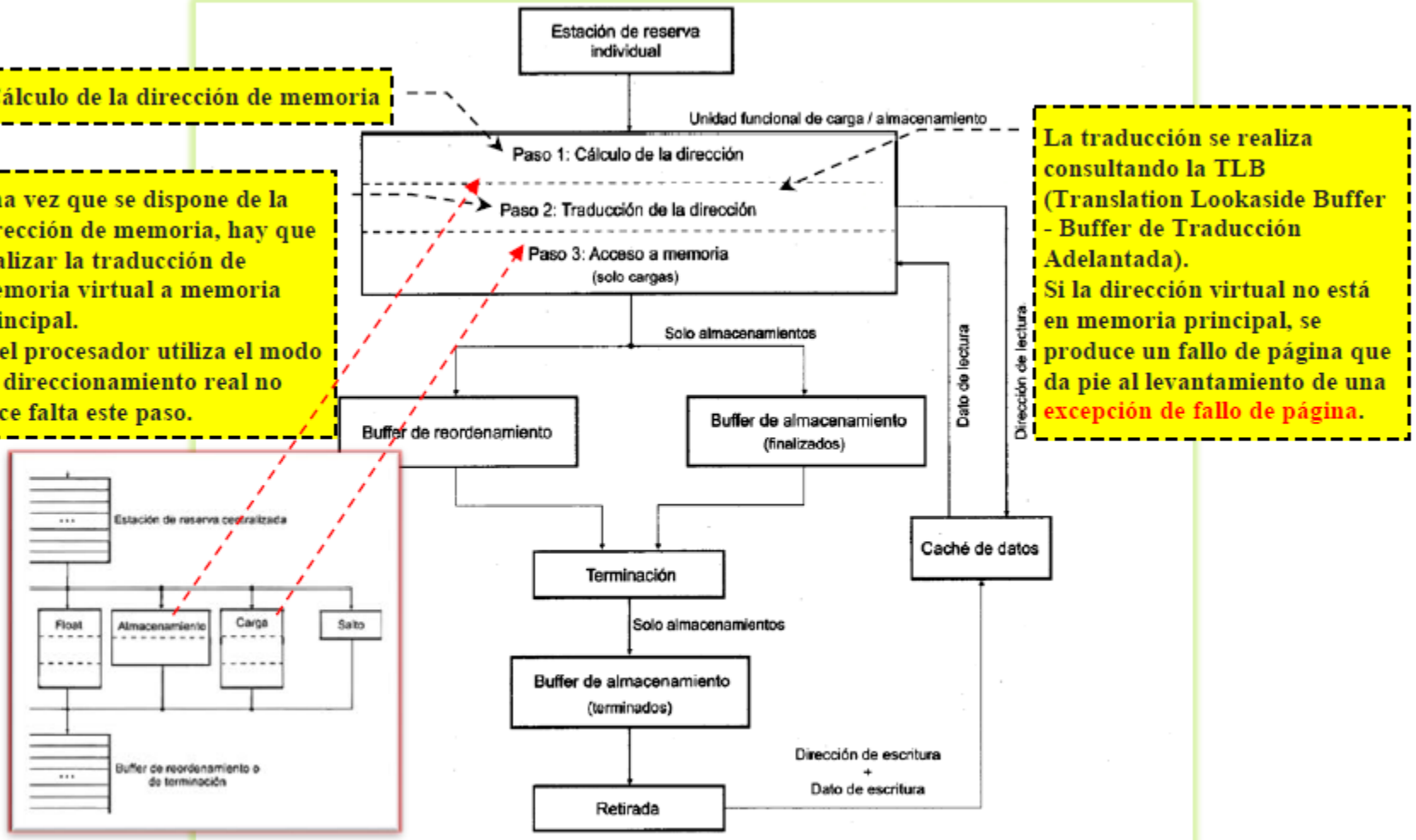


Figura 2.45: Organización de la etapa de ejecución y del *back-end* de la segmentación para el procesamiento diferenciado de las instrucciones de carga y almacenamiento.

Cálculo de la dirección de memoria

Una vez que se dispone de la dirección de memoria, hay que realizar la traducción de memoria virtual a memoria principal.
Si el procesador utiliza el modo de direccionamiento real no hace falta este paso.

La traducción se realiza consultando la TLB (Translation Lookaside Buffer - Buffer de Traducción Adelantada).
Si la dirección virtual no está en memoria principal, se produce un fallo de página que da pie al levantamiento de una excepción de fallo de página.

- La escritura diferida por parte de una instrucción de almacenamiento tras el abandono del buffer de reordenamiento evita una actualización errónea y precipitada de la memoria.

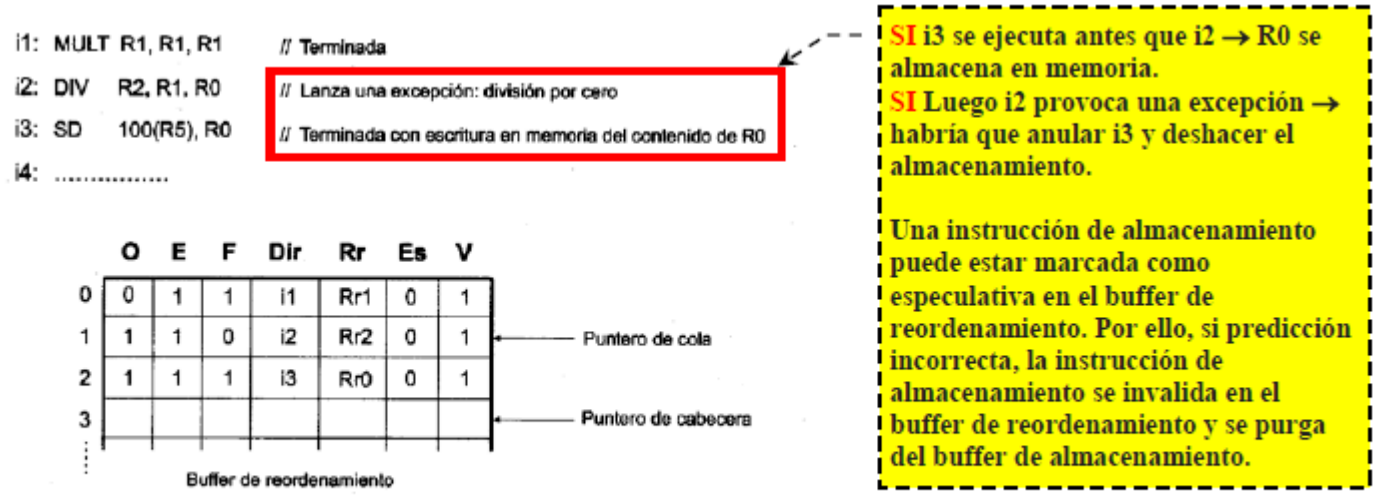


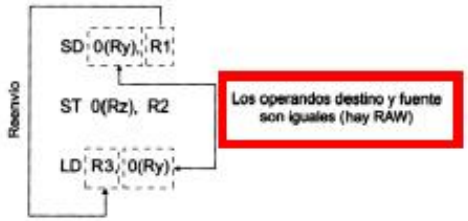
Figura 2.46: Ejemplo de situación anómala si la escritura en memoria se realizase en la fase de ejecución y se produjese una excepción.

El orden en la terminación y retirada de las instrucciones de carga/almacenamiento garantiza las dependencias de memoria tipo WAR y WAW.

2.10. Mejoras en el procesamiento de las instrucciones de carga/almacenamiento

- Instrucciones de carga que escriben en un registro que constituyen un punto de arranque critico para un conjunto de instrucciones aritmetico lógicas que dependen de este operando
- Terminación adelantada de Instrucciones de carga
  - Una mejora notable en el rendimiento de los procesadores superescalares se obtiene permitiendo que esas instrucciones de carga terminen antes que otras instrucciones, sin que se violen dependencias RAW tanto de datos como de memoria.
    - 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
  - Otra mejora del rendimiento de un procesador es el reenvío de datos desde una instrucción de almacenamiento hacia una de carga que tengan operandos destino y fuente comunes.

(a) Situación que permite terminar una carga sin que terminen los adelantamientos



(b) Situación que permite que una instrucción de almacenamiento transfiera un dato a una carga para evitar el acceso a la caché de datos

Figura 2.47: Esquemas de adelantamiento y reenvío de datos entre instrucciones de carga y almacenamiento.

2.10.1. Reenvío de datos entre instrucciones de almacenamiento y de carga (de memoria)

- Se puede realizar cuando existe una dependencia de memoria RAW entre una instrucción de almacenamiento (productora) y una instrucción de carga (consumidora).
- Condición
  - Comprobar que las direcciones de memoria de una instrucción de carga y de los almacenamientos pendientes sean coincidentes.
  - El dato que requiere la carga está en un registro del procesador y no hay que ir a la memoria para obtenerlo
- Dependencia ambigua
  - Las direcciones efectivas de dos instrucciones de carga/almacenamiento son iguales, una vez que han sido emitidas y terminadas/finalizadas

```
i1:  SD   100(R2),R1    % Mem[100+R2] <= R1
.....................
i2:  LD   R3,50(R4)     % R3 <= Mem[50+R4]
```

Si R2=50 y R4=100 → M[150] → RAW

Hay que comprobar coincidencias → Cuando las instrucciones de carga y almacenamiento han sido emitidas y finalizadas/terminadas.

La situación entre i1 e i2 = *dependencia ambigua*

Compara las direcciones de almacenamiento con las de carga

- Si **hay** coincidencia
  - El valor que hay en el campo Datos de la entrada que presenta la coincidencia se coloca en el bus de reenvío junto con el identificador del registro destino RRF de la carga.
  - La carga no necesita acceder a la D-caché.
- Si **no** hay coincidencia,
  - No hay RAW → Acceso a caché datos de forma normal.
- Si hay varias coincidencias, el procesador debe contar con el hardware adecuado para saber cuál es el almacenamiento más reciente

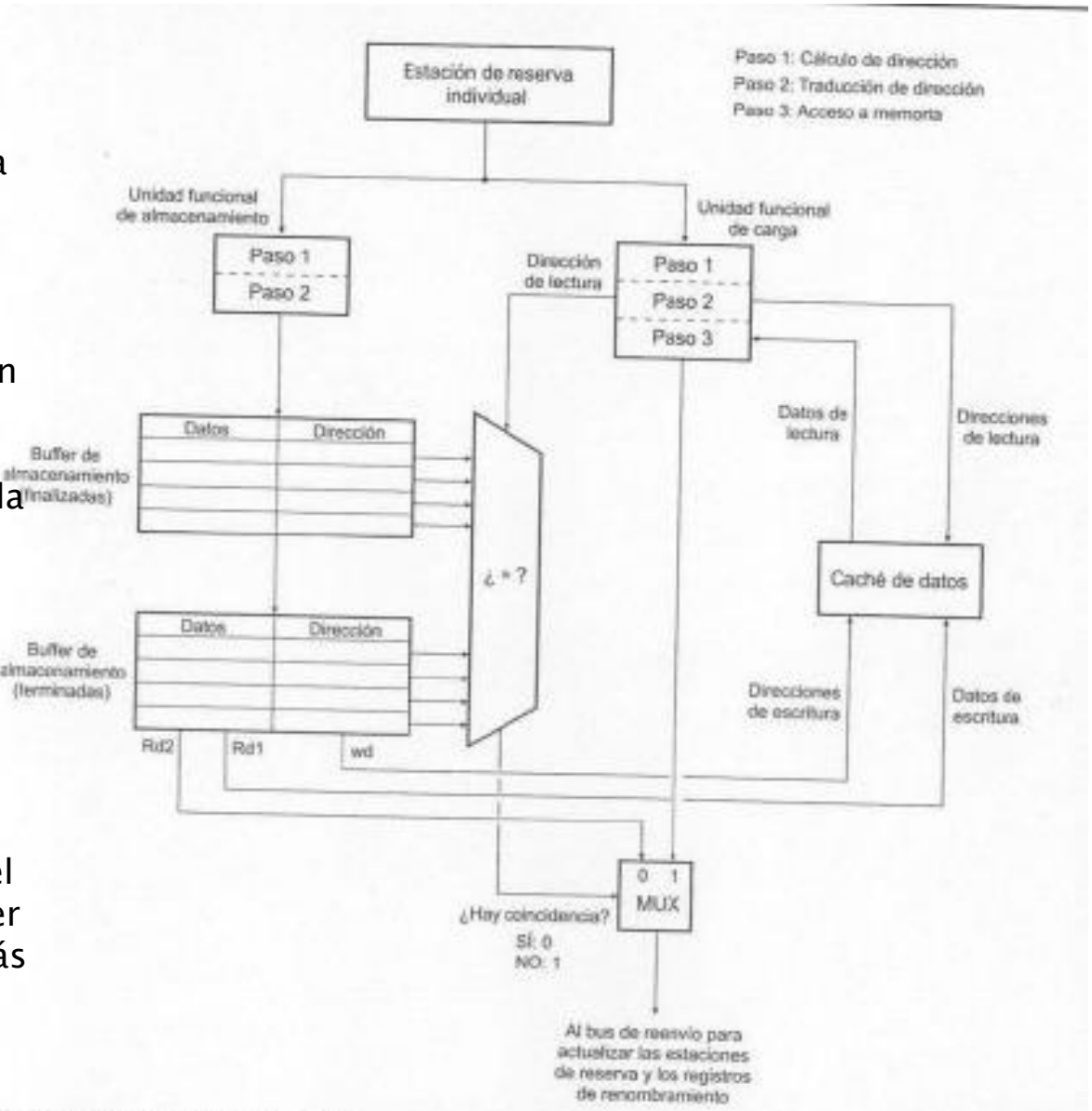


Figura 2.48: Esquema de reenvío de datos entre cargas y almacenamientos en el que se contempla la emisión ordenada de las cargas y los almacenamientos.

2.10.2. Terminación adelantada de las instrucciones de carga

- Adelantar la ejecución de una instrucción de carga frente a varias instrucciones de almacenamiento es mas complicado porque hay que comprobar que las direcciones de memoria que manejan no son iguales.
  - La carga, mientras que realiza el acceso a la caché de datos (tercer paso), podría comprobar la coincidencia de su dirección en el campo Dirección de las entradas del buffer de almacenamiento.
- Dos casos
  - Hay coincidencia de direcciones
    - No se puede adelantar la carga
  - No hay coincidencia de direcciones
    - El dato recuperado de la D-cache es válido y se puede almacenar en el registro destino
- Se emplea la técnica de **buffer de carga finalizada**
  - Es una técnica es la emisión desordenada de las cargas y de los almacenamientos
  - La idea se basa en permitir que, cuando sea posible, las cargas se ejecuten de forma especulativa sin comprobar la existencia de dependencias ambiguas.
  - Las cargas cuando llegan al segundo paso de su unidad funcional comprueban la existencia de coincidencias de dirección con los almacenamientos que, en ese instante, se encuentren en el buffer de almacenamiento.
    - 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.
    - Pueden darse **dos situaciones**:
      - **Hay coincidencia** → Se ha violado una dependencia → Hay que invalidar esa carga y todas las instrucciones posteriores
      - **No hay coincidencia** → La carga y todas las instrucciones posteriores a ella pueden terminarse.

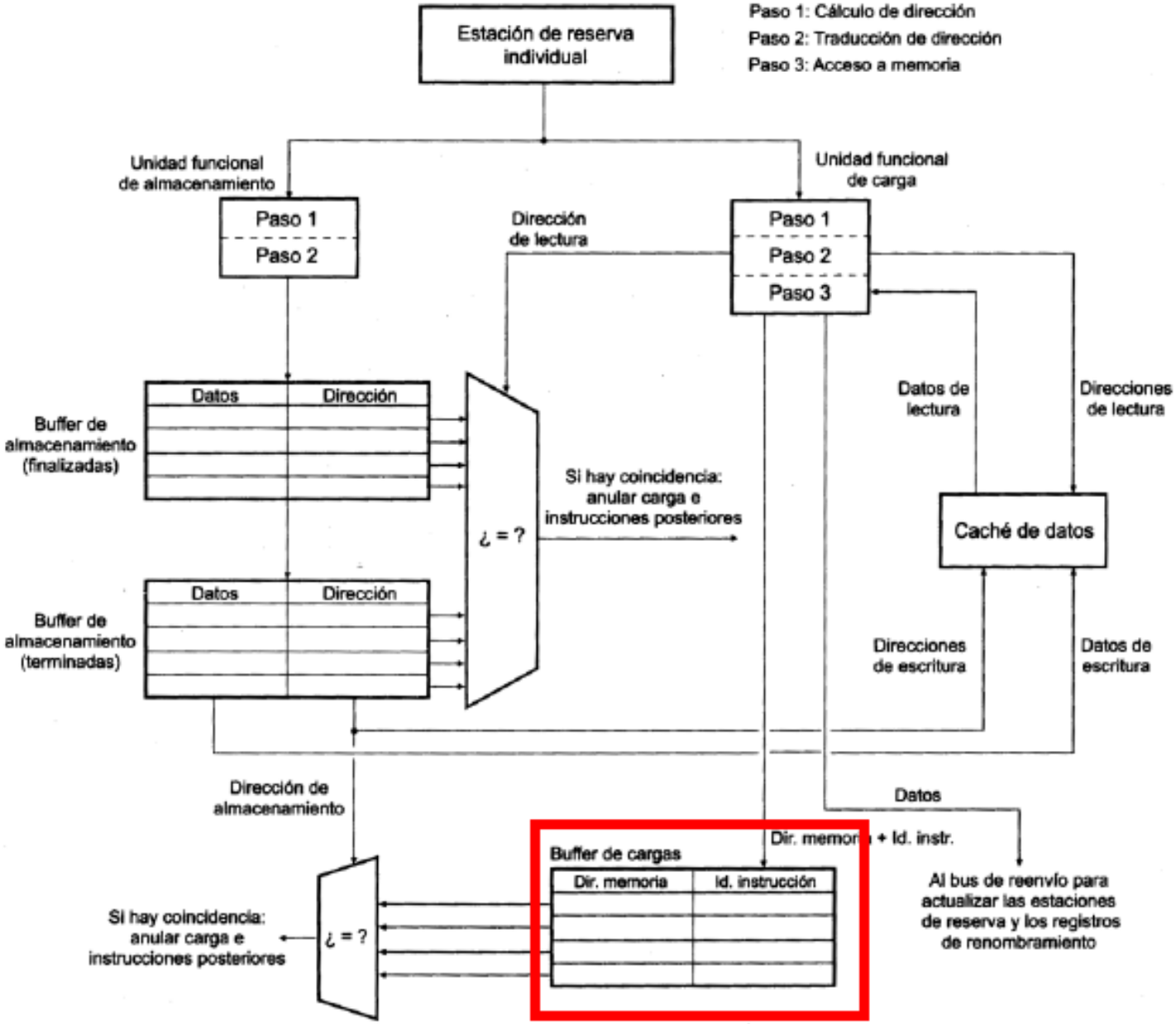


Figura 2.49: Esquema para realizar el adelantamiento de las cargas permitiendo la emisión fuera de orden de las cargas y los almacenamiento.

2.11. Tratamiento de interrupciones

- En los procesadores superescalares, el tratamiento de las interrupciones es más complicado.
- Cuando se produce una interrupción, el estado del procesador es guardado
  - El estado está definido por
    - El contador de programa
    - Los registros arquitectónicos y
    - El espacio de memoria asignado
- **Precisión de excepción** (Interrupciones precisas)
  - Si es capaz de tratar las interrupciones de forma que se respeten las condiciones previas
    - Para tener interrupciones precisas es necesario que el procesador mantenga su estado y el de la memoria, de forma análoga a si las instrucciones se ejecutasen en orden
- El problema consiste en recuperar la información de estado al momento que se produce la interrupción
  - Hay que reconstruir la información al estado como si las instrucciones se hubiesen ejecutado en orden

Clasificación de las instrucciones

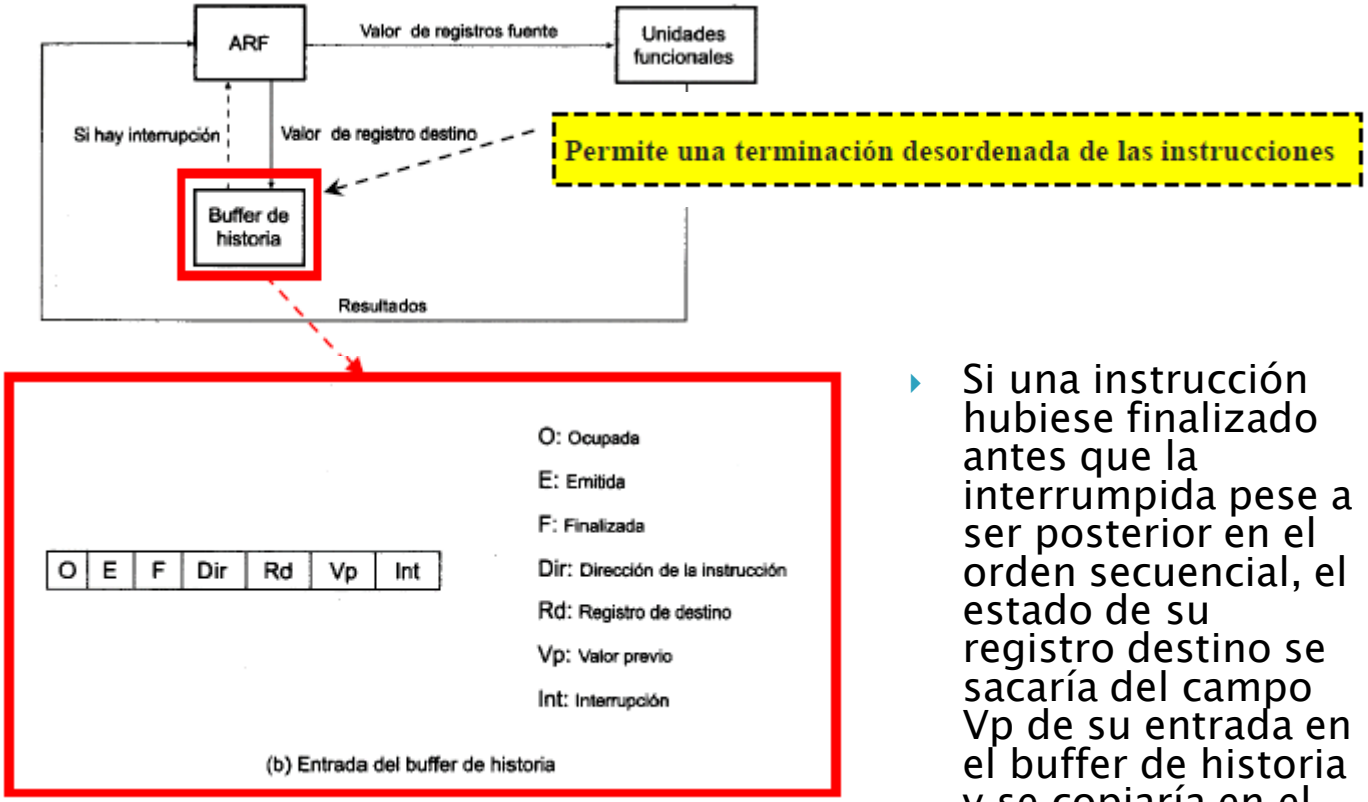
- Interrupciones externas:
  - Se producen por eventos externos a la ejecución del programa
    - Una estrategia habitual de actuación consiste en detener la lectura de nuevas instrucciones, vaciar el cauce terminando arquitectónicamente las instrucciones ya existentes y guardar el estado en que quedó el procesador tras terminar la instrucción previa a la que fue interrumpida
- **Excepciones, interrupciones de programa** o traps
  - Causadas por (divisiones por cero, desbordamientos aritméticos), los fallos de página en el sistema de memoria virtual.
    - El tratamiento es similar al de las interrupciones externas, aunque en algunos casos no se considera la reanudación del programa como una posibilidad.

Técnicas de tratamiento de interrupciones (procesadores que permiten ejecución fuera de orden)

- Ignorar el problema y no garantizar la precisión de excepción.
- Permitir que las interrupciones sean "algo" imprecisas pero guardando suficiente información para que la rutina de tratamiento software pueda recrear una secuencia precisa para recuperarse de la interrupción.
- 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.

Procesamiento de interrupciones basados en buffer de historia

Se almacena el valor de los registros destinos al comienzo de la ejecución (campo Vp valor previo), no al finalizar



- Si una instrucción hubiese finalizado antes que la interrumpida pese a ser posterior en el orden secuencial, el estado de su registro destino se sacaría del campo Vp de su entrada en el buffer de historia y se copiaría en el ARF.

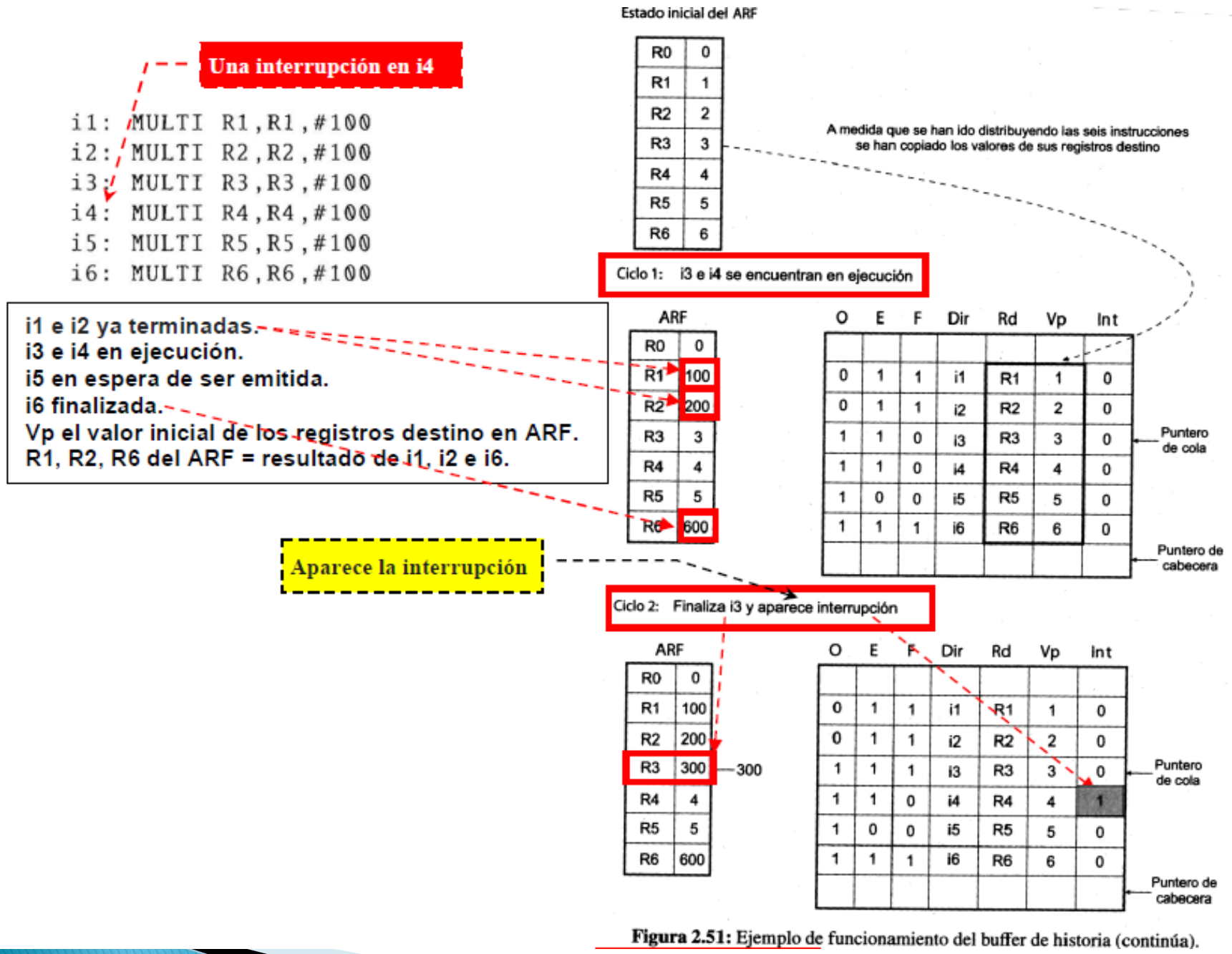


Figura 2.51: Ejemplo de funcionamiento del buffer de historia (continúa).

Ciclo 3: i3 termina e i4 finaliza

ARF

R0	0
R1	100
R2	200
R3	300
R4	400
R5	5
R6	600

400

O	E	F	Dir	Rd	Vp	Int
0	1	1	i1	R1	1	0
0	1	1	i2	R2	2	0
0	1	1	i3	R3	3	0
1	1	1	i4	R4	4	1
1	0	0	i5	R5	5	0
1	1	1	i6	R6	6	0

Puntero de cola

Puntero de cabecera

Ciclo 4: i4 termina y se trata la interrupción

ARF

R0	0
R1	100
R2	200
R3	300
R4	4
R5	5
R6	6

O	E	F	Dir	Rd	Vp	Int
0	1	1	i1	R1	1	0
0	1	1	i2	R2	2	0
0	1	1	i3	R3	3	0
0	1	1	i4	R4	4	1
0	0	0	i5	R5	5	0
0	1	1	i6	R6	6	0

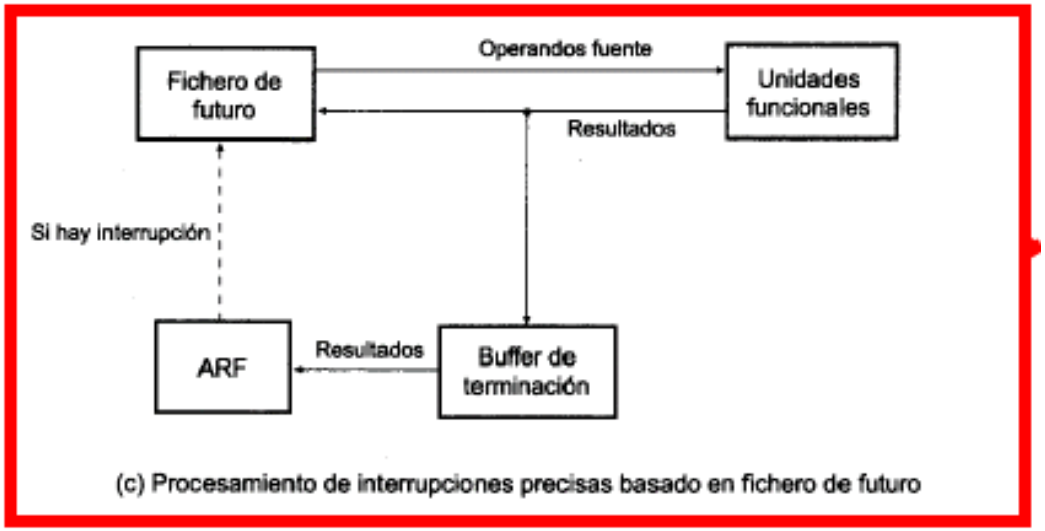
Puntero de cola

Puntero de cabecera

Anula resultado de i4 y restablece los valores de R4, R5 y R6 (Resultados de las instrucciones i4, i5 e i6.

Se restaura el ARF con la información del buffer de historia

Procesamiento de interrupciones basado en fichero de futuro

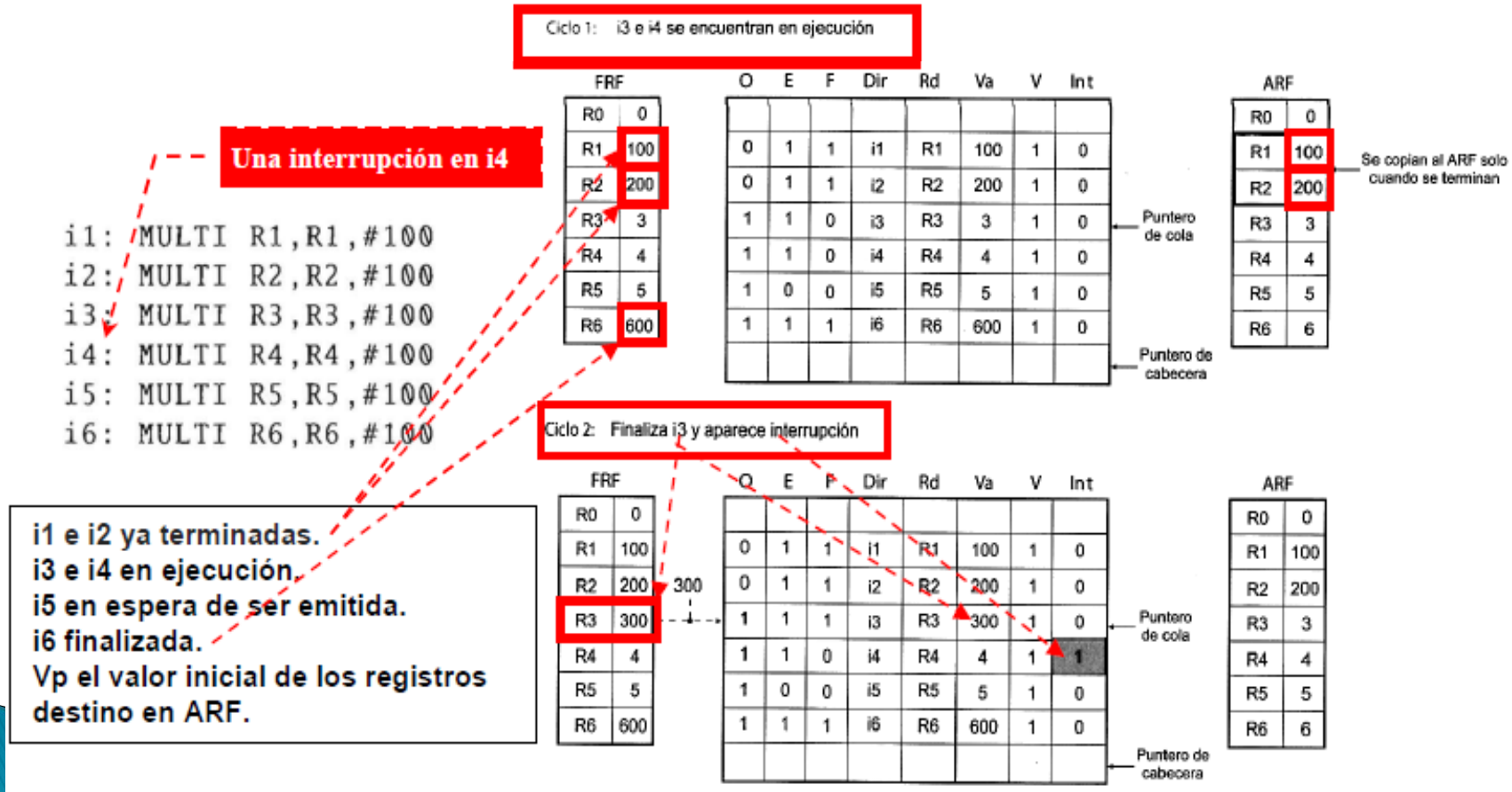


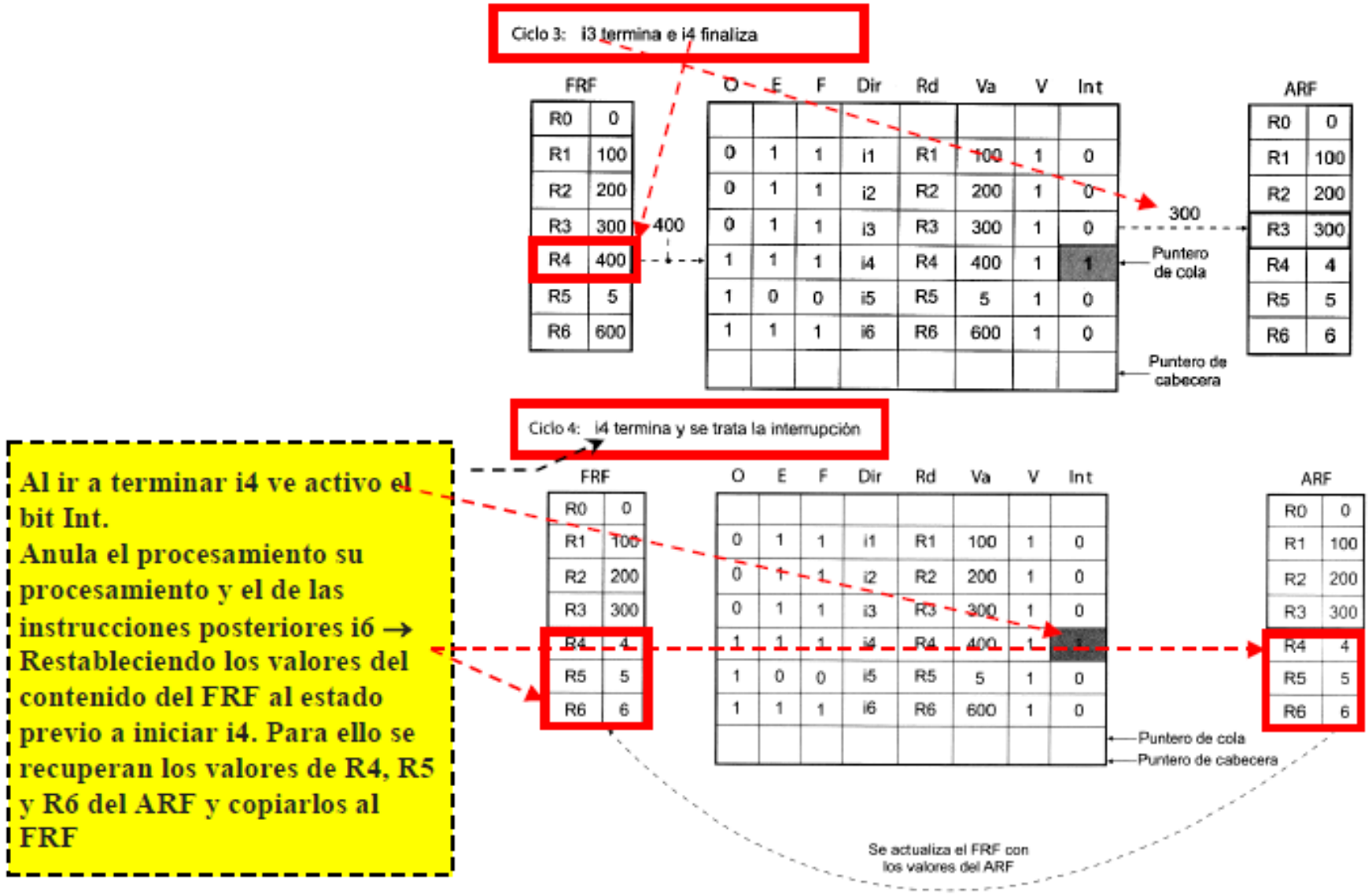
(c) Procesamiento de interrupciones precisas basado en fichero de futuro

- Combina un fichero de registros de futuro 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 futuro
  - En el buffer de terminación.
- Cuando las instrucciones terminan, el ARF se actualiza con la información del buffer de terminación.
- Si una instrucción es interrumpida, el fichero de futuro recupera los valores originales de los registros destino de la instrucción interrumpida y de las posteriores desde el ARF.

Ejemplo de fichero de futuro

Al finalizar una instrucción su resultado se guarda en FRF y en el Buffer de Reordenamiento (Va) para una vez terminada copiarlo en el ARF




Ciclo 3: i3 termina e i4 finaliza
FRF
R0 0
R1 100
R2 200
R3 300
R4 400
R5 5
R6 600
400
O E F Dir Rd Va V Int
0 1 1 i1 R1 100 1 0
0 1 1 i2 R2 200 1 0
0 1 1 i3 R3 300 1 0
1 1 1 i4 R4 400 1 1
1 0 0 i5 R5 5 1 0
1 1 1 i6 R6 600 1 0
300
Puntero de cola
Puntero de cabecera
ARF
R0 0
R1 100
R2 200
R3 300
R4 4
R5 5
R6 6
Ciclo 4: i4 termina y se trata la interrupción
Al ir a terminar i4 ve activo el bit Int.
Anula el procesamiento su procesamiento y el de las instrucciones posteriores i6 →
Restableciendo los valores del contenido del FRF al estado previo a iniciar i4. Para ello se recuperan los valores de R4, R5 y R6 del ARF y copiarlos al FRF
Se actualiza el FRF con los valores del ARF

2.11.1. Excepciones precisas con buffer de reordenamiento

- El elemento clave que permite mantener la consistencia del procesador es el buffer de reordenamiento o de terminación.
- Para conocer si una instrucción ha sufrido una interrupción durante su etapa de ejecución, las entradas del buffer de reordenamiento, el campo Int. = 1 para indicar que una instrucción ha sido interrumpida
- Tratamiento de la excepción:
  - 1. Expulsar del cauce a i4 y a todas las instrucciones posteriores a ella.
  - 2. Almacenar el estado del procesador, es decir, guardar el estado de los registros arquitectónicos y del contador de programa.
  - 3. Tratar la interrupción según del tipo que sea.
- 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.

Situación inicial:

- **i1 e i2 terminado**
- **i3 en ejecución**
- **i4 finalizado pero recibió una interrupción durante su etapa de ejecución.**
- **i5 e i6 en espera de ser emitidas.**

```
i1: LD    R1, 100(R0)    // R1 renombrado como Rr1
i2: ADD   R3, R2, R1     // R3 renombrado como Rr3
i3: DIV   R4, R4, R3     // R4 renombrado como Rr4
i4: MULT  R6, R5, R5     // R6 renombrado como Rr5
i5: SD    100(R1), R4    // No hay registro destino
i6: ADD   R4, R4, R3     // R4 renombrado como Rr6
```

	Q	E	F	Dir	Rd	Rr	Es	V	Int	
	0	1	1	i1	R1	Rr1	0	1	0	
	0	1	1	i2	R3	Rr3	0	1	0	
	1	1	0	i3	R4	Rr4	0	1	0	Puntero de cola
	1	1	1	i4	R6	Rr5	0	1	1	
	1	0	0	i5	.....	.....	0	1	0	Se anula su procesamiento
	1	0	1	i6	R4	Rr6	0	1	0	
										Puntero de cabecera

Cuando puntero de cola llegue a la entrada de i4 y compruebe que se puede terminar pero que Int=1 → se inicia el proceso de tratamiento de la excepción

Figura 2.53: Estado del buffer de reordenamiento al producirse una interrupción.

2.12. 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 y el análisis de las dependencias existentes entre ellas.
  - El grado de paralelismo intrínseco existente en el flujo de instrucciones de un programa
  - En un procesador superescalar con recursos hardware limitados y un ancho de emisión de k instrucciones:
  - La complejidad de la lógica de distribución/emisión de instrucciones es de orden n^k (n= nº inst. del repertorio)
  - El retardo es de orden $k^2 \log n$
  - Líneas de trabajo en el diseño de nuevos 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.

resumen

- Para finalizar el capítulo, se plantea un resumen de los pasos que se dan en cada etapa considerando
- Lectura.
  - Se extrae el grupo de lectura de la [-caché.
  - Si es un salto, se especula el resultado y la dirección de destino y se cambia el contador de programa según la especulación.
- Decodificación.
  - Se decodifican las instrucciones en paralelo.
  - Se envían al buffer de distribución.
- Distribución.
  - Si hay espacio en las estaciones de trabajo, en el buffer de terminación y en el RRF se distribuye la instrucción.
  - Se leen los operandos (identificadores o valores) desde el ARF o el RRF.
  - Se introduce la instrucción en el buffer de reordenamiento.
  - Se renombra el registro destino de la instrucción.
  - Se introduce la instrucción en la estación de reserva que corresponda.
  - Las estaciones de reserva examinan los buses de reenvío en busca de coincidencias de identificadores de registro destino.
  - Si todo los operandos fuente se encuentran disponibles, se emite la instrucción a la unidad funcional. Por lo general, se libera la entrada que ocupaba en la estación de reserva.
- Ejecución.
  - Se cambia el estado de la instrucción en el buffer de reordenamiento.
  - Si hay una interrupción, se marca el campo correspondiente en el buffer de terminación.
  - Si es una instrucción de salto se comprueba que la especulación coincida con el resultado del salto. Si es así, se validan las instrucciones especuladas; en caso contrario, se invalidan.
  - Cuando finaliza la ejecución, se publica el resultado de la instrucción y el identificador del registro destino en los buses de reenvío para permitir las actualizaciones del RRF, de las entradas de las estaciones de reserva y del buffer de reordenamiento. Si es un almacenamiento, se introduce en el buffer de almacenamiento y se marca como finalizado .

Resumen

- Terminación.
  - Se cambia el estado de la instrucción en el buffer de reordenamiento.
  - Si no es una instrucción interrumpida, se libera su entrada del buffer de reordenamiento y se da por terminada arquitectónicamente.
  - Se actualiza el ARF desde el RRF y se libera el registro de renombramiento.
  - Si es un almacenamiento, se marca en el buffer de almacenamiento como terminado.
  - Si es una instrucción interrumpida, se eliminan ella y todas las siguientes del buffer sin actualizar el ARF, estén o no finalizadas. También se eliminan los almacenamientos del buffer de almacenamiento y las cargas del buffer de cargas especuladas (en caso de que exista) quesean posteriores a la instrucción interrumpida.
  - Si es una instrucción especulada, queda en espera de terminación.
  - Si es una instrucción invalidada, se termina sin actualizar el ARF .
- Retirada (solo almacenamientos).
  - Esperar a que el bus de acceso a memoria esté libre y realizar la escritura.
  - La instrucción se retira del buffer de almacenamiento.