TEMA II

PROCESADORES SUPERESCALARES

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*.

2.1. Características de los procesadores superescalares

- Paralelismo
  - En un mismo instante de tiempo se ejecutan varias instrucciones en diferentes etapas
  - El paralelismo de máquina temporal
    - Máquina segmentada clásica (varias instrucciones en distintas etapas
  - Paralelismo espacial
    - Por replicación del hardware
    - Ancho o grado de la segmentación
      - El número de instrucciones que se pueden procesar en la misma etapa
- 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

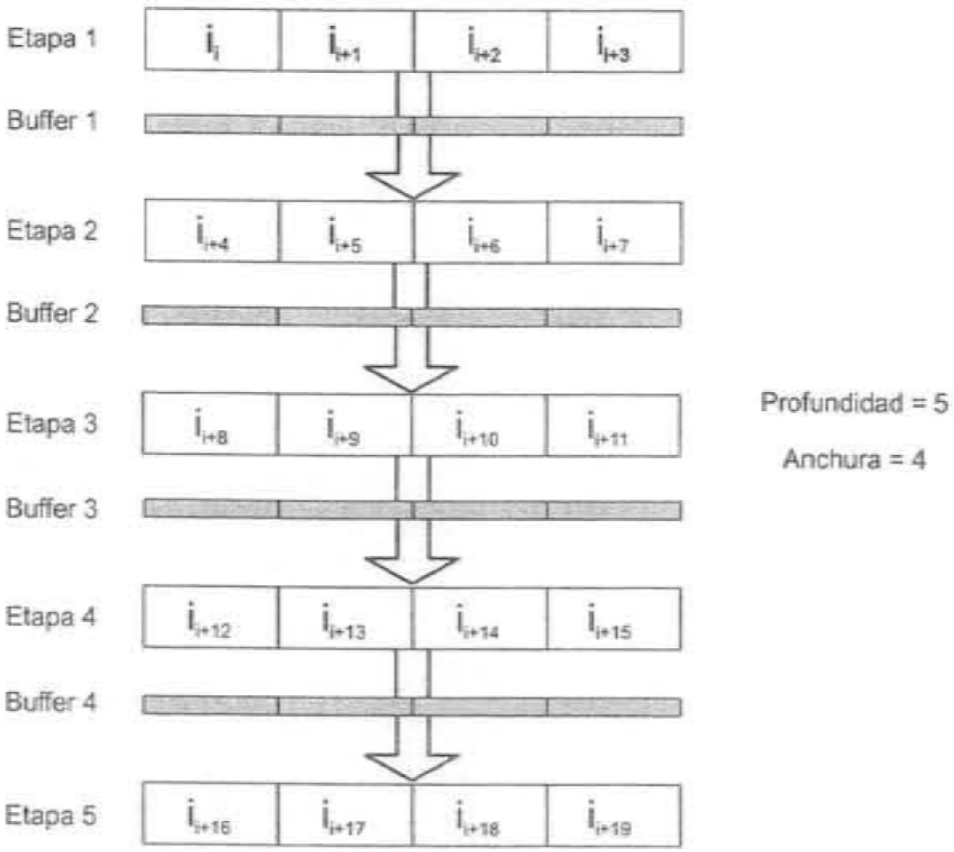
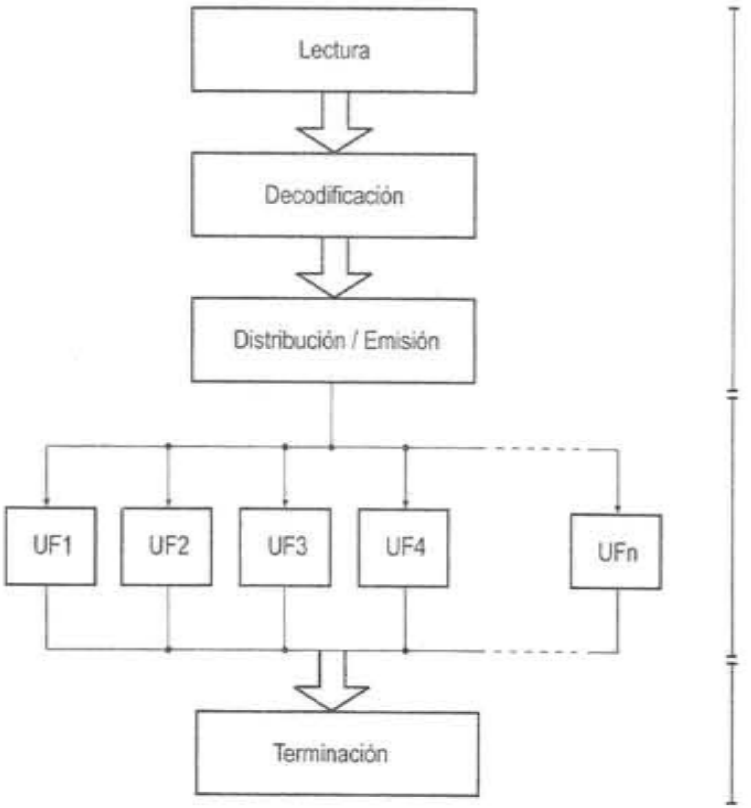


Figura 2.1: Esquema de una segmentación de profundidad 5 y anchura 4. En condiciones ideales pueden concluir su procesamiento de forma simultánea hasta 4 instrucciones por ciclo.

Características

- 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

- 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

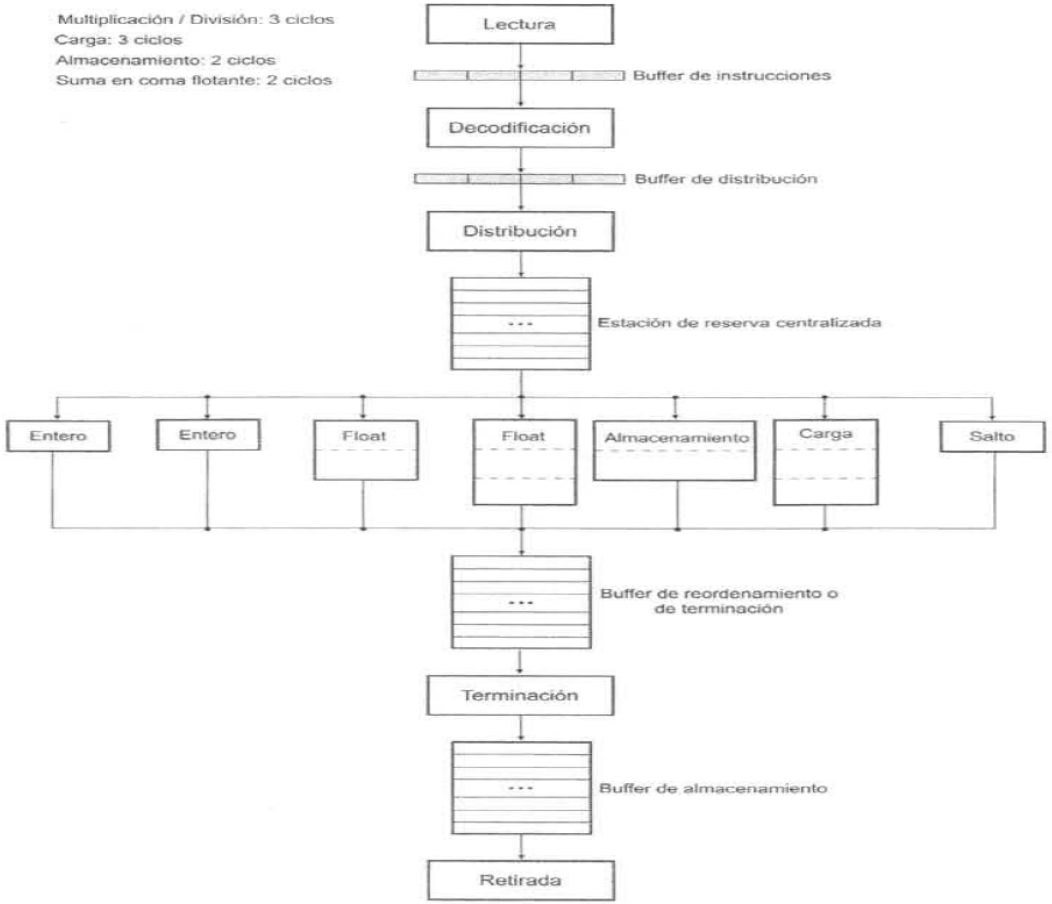


Figura 2.3: Ejemplo de segmentación superescalar genérica.

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.

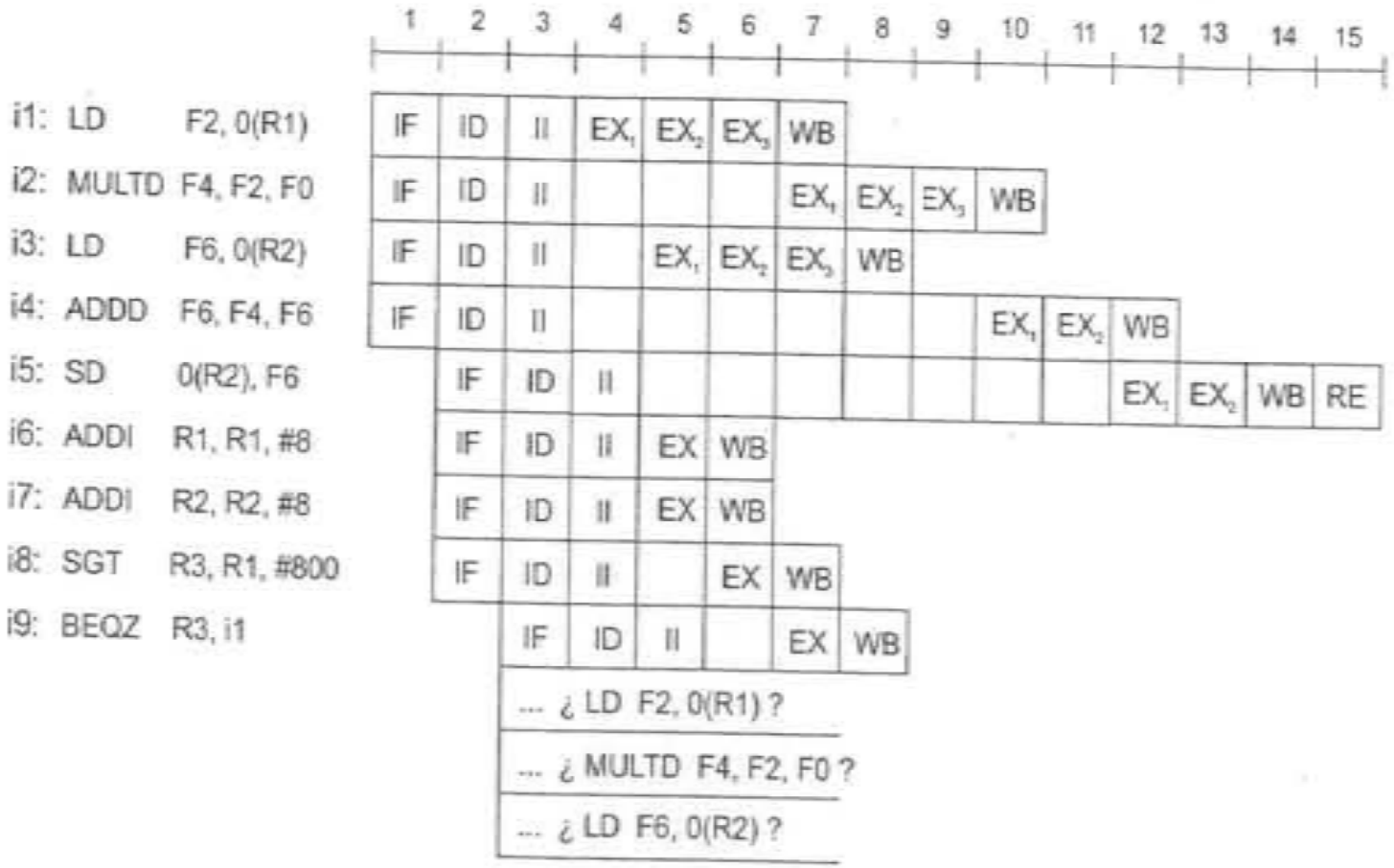


Figura 2.4: Secuencia de la ejecución del bucle DAXPY en la segmentación superescalar de 6 etapas.

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

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

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

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

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

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

- Que se adelanten resultados o se renombren los registros para evitar dependencias WAR y WAW y aumentar así el rendimiento de las unidades funcionales no garantiza la consistencia semántica del procesador y la memoria.
  - Para lograr la consistencia una vez deshecho el renombramiento hay que realizar la escritura ordenada de los registros arquitectónicos en la etapa de terminación y de las posiciones de memoria en la etapa de retirada.
- Tenemos que tener en cuenta que solo es posible terminar aquellas instrucciones que no sean el resultado de especular con una instrucción de salto.
- El buffer de reordenamiento o terminación se convierte en la pieza fundamental para conseguir esta consistencia del procesador, ya que es el sitio en donde se realiza el seguimiento de una instrucción desde que se distribuye hasta que se termina.

2.5 Lectura de instrucciones

- La diferencia entre un procesador superescalar y uno escalar en la etapa de lectura
  - 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.
    - Al conjunto de instrucciones que se extrae simultáneamente de la I-caché se le denomina grupo de lectura.
  - 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.

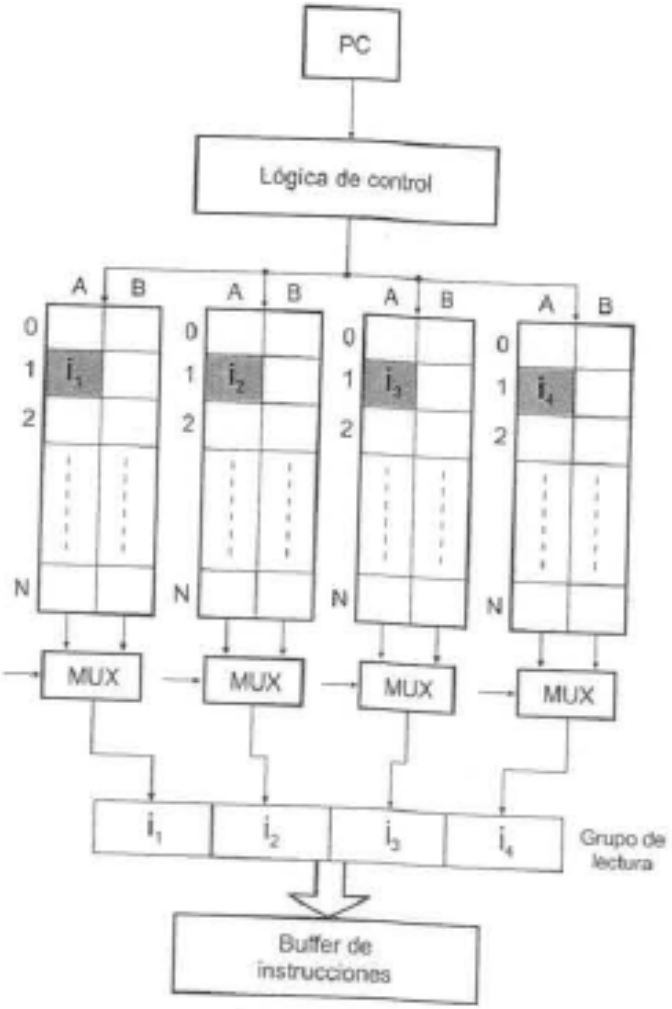


Figura 2.8: Organización de memoria caché en la que el tamaño del bloque físico corresponde al ancho de la etapa de decocificación.

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.
- Soluciones
  - **La red de alineamiento**
    - Consiste en reubicar las salidas de la caché mediante multiplexores que conducen las instrucciones leídas a su posición correcta dentro del grupo de lectura.
  - **La red de desplazamiento**
    - Recurre a registros de desplazamiento para mover las instrucciones.
  - **El prefetching o prelectura**
    - Es una técnica utilizada para minimizar el impacto de los fallos de lectura en la caché de instrucciones.

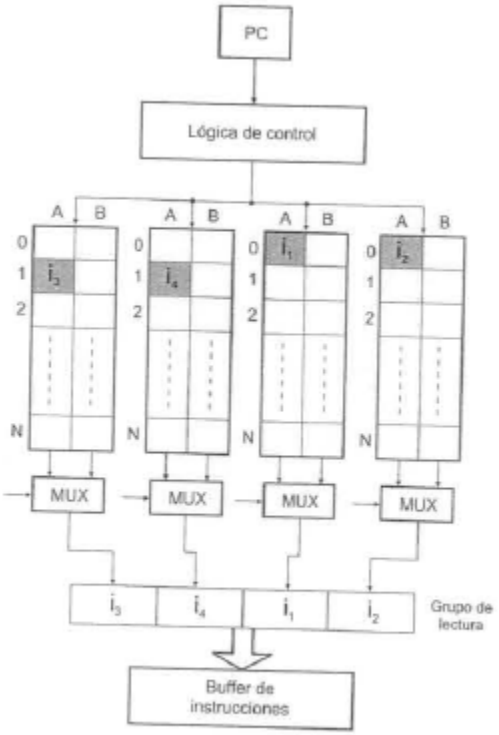


Figura 2.9: Grupo de lectura de cuatro instrucciones desalineado con respecto a la unidad de lectura, provocando la necesidad de dos accesos.

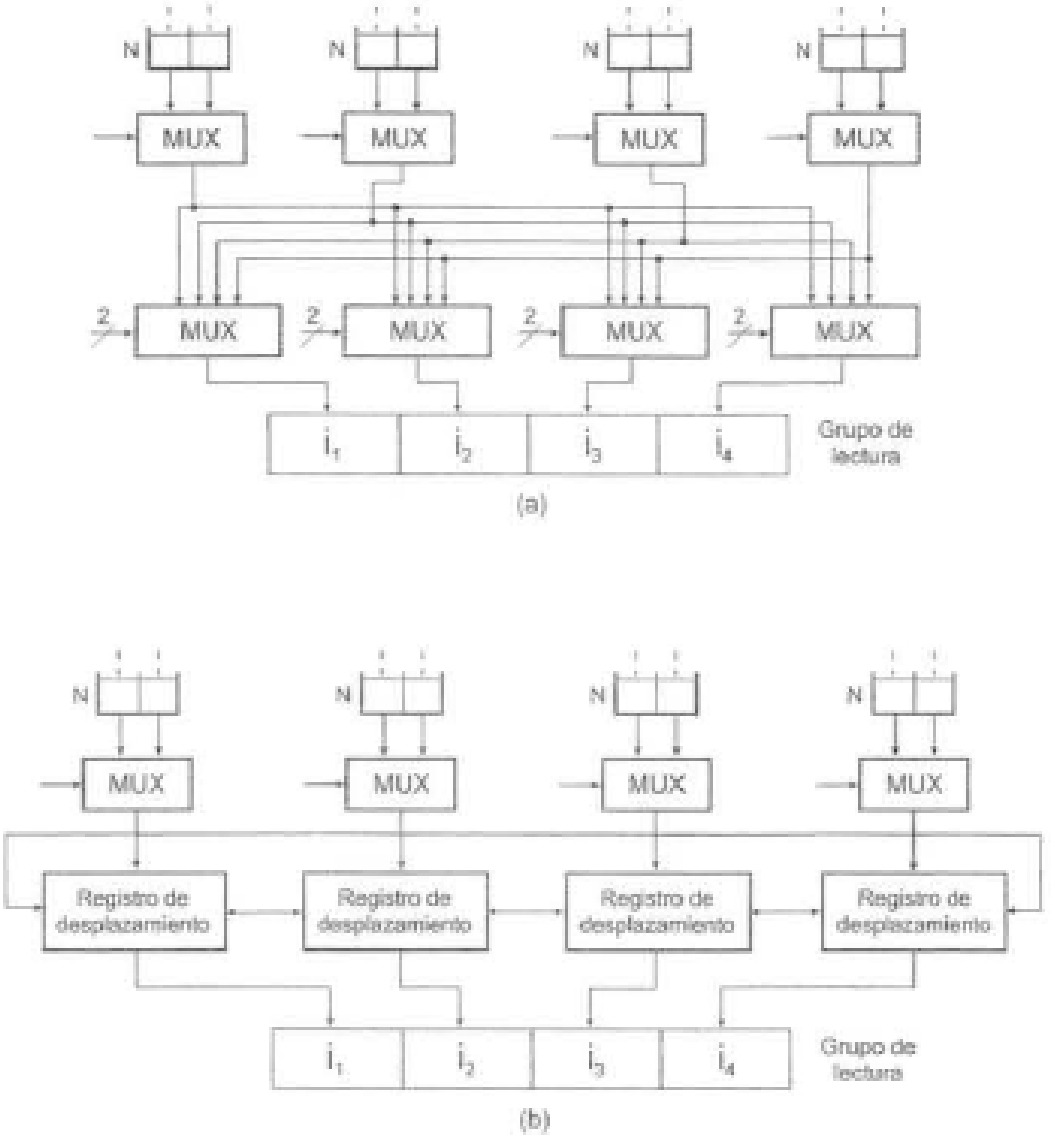


Figura 2.10: Red de alineamiento basada en multiplexores (a) y basada en registros de desplazamiento (b).

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.

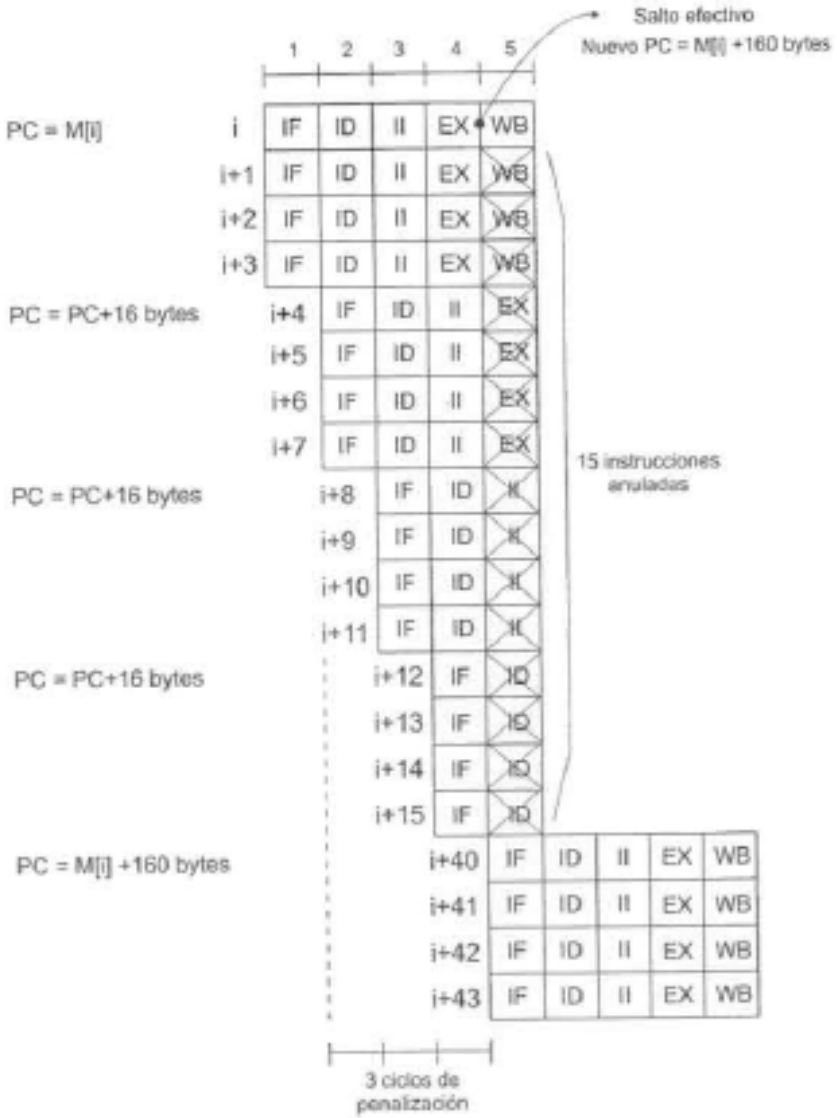


Figura 2.11: El nuevo valor del PC generado por la instrucción de salto en la etapa EX obliga a vaciar el cauce y proceder a la extracción del nuevo grupo de lectura.

2.5.3 Tratamiento de los saltos

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

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.
- Puede surgir el problema de los falsos positivos, la BTAC devuelve una dirección de destino pero en el grupo de lectura no hay realmente ningún salto, esto se descubre en la etapa ID.
  - Esto es lo que se conoce como saltos fantasmas o saltos en falso, estas instrucciones que el Predictor de destinos identifica inicialmente como saltos producen una penalización al tener que expulsar del cauce la instrucción de destino especulada.
  - Los falsos positivos son debidos a que el campo BIA de la BTAC no almacena una dirección completa, sólo una parte.
    - Esto provoca que a direcciones distintas se le asocie la misma BIA

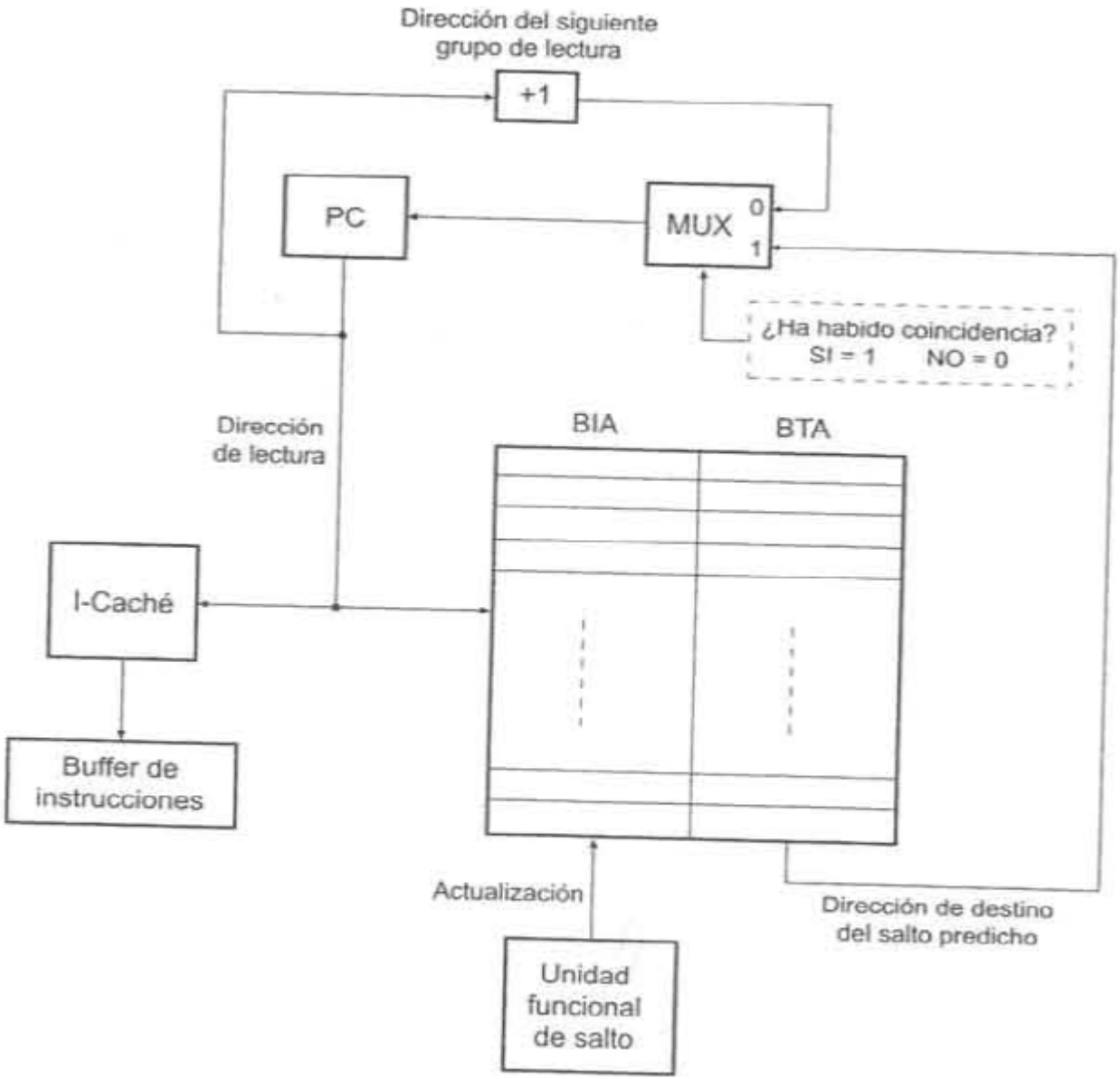


Figura 2.12: Esquema de una BTAC.

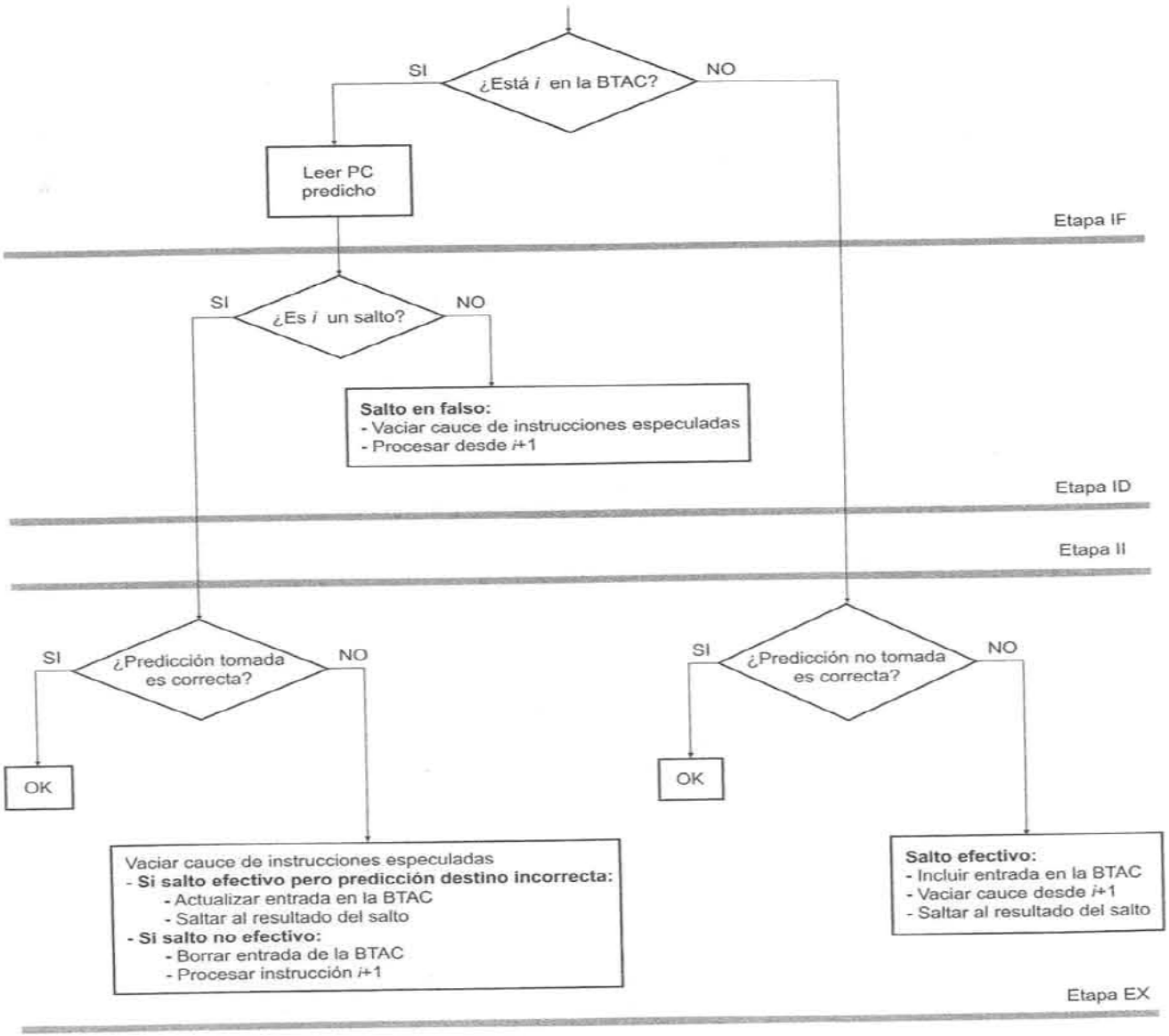


Figura 2.13: Diagrama con las posibles situaciones que se pueden presentar al realizar predicción mediante la BTAC.

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

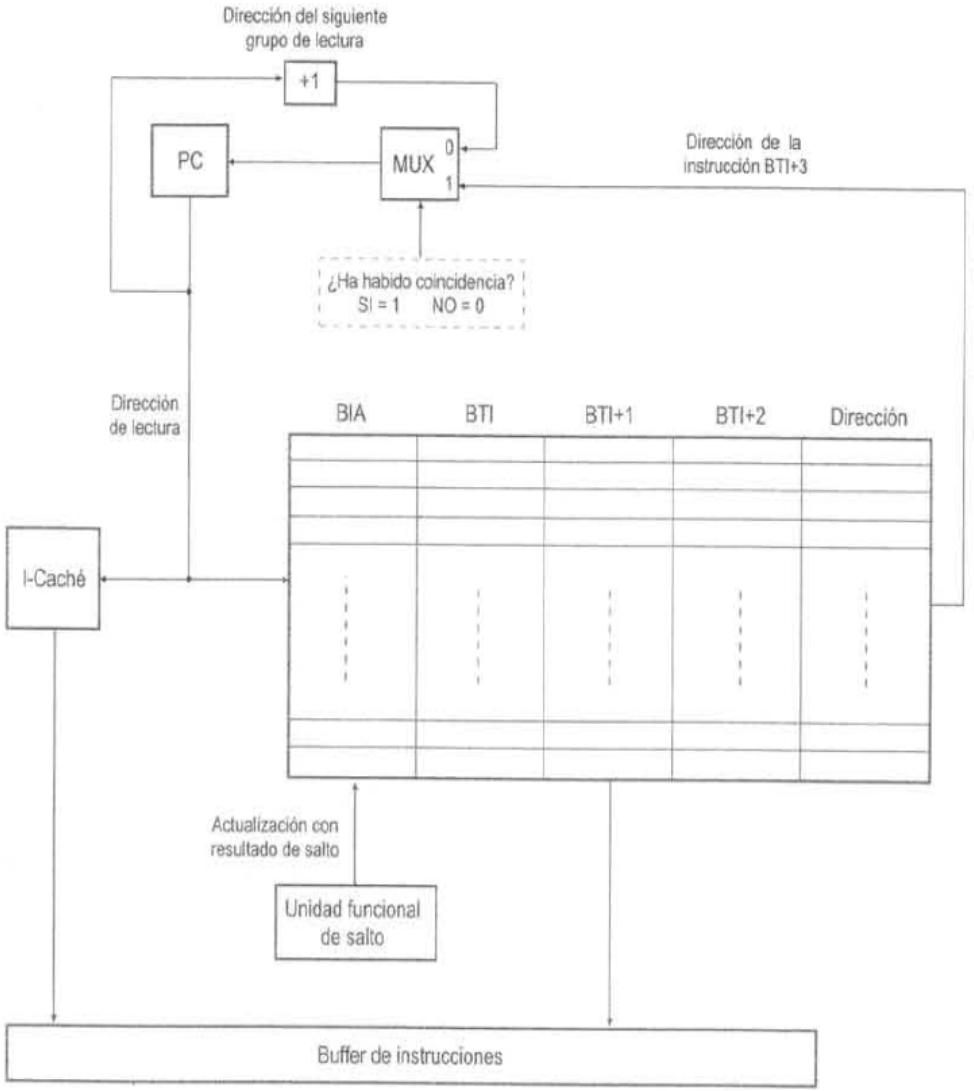


Figura 2.14: Esquema de una BTIB en la que se entregan al buffer de instrucciones paquetes de 3 instrucciones. Esto obliga a modificar el PC para que apunte a la instrucción que sigue a la última que forma el paquete, en este caso, el PC debe apuntar a la dirección de la instrucción BTI+3.

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).

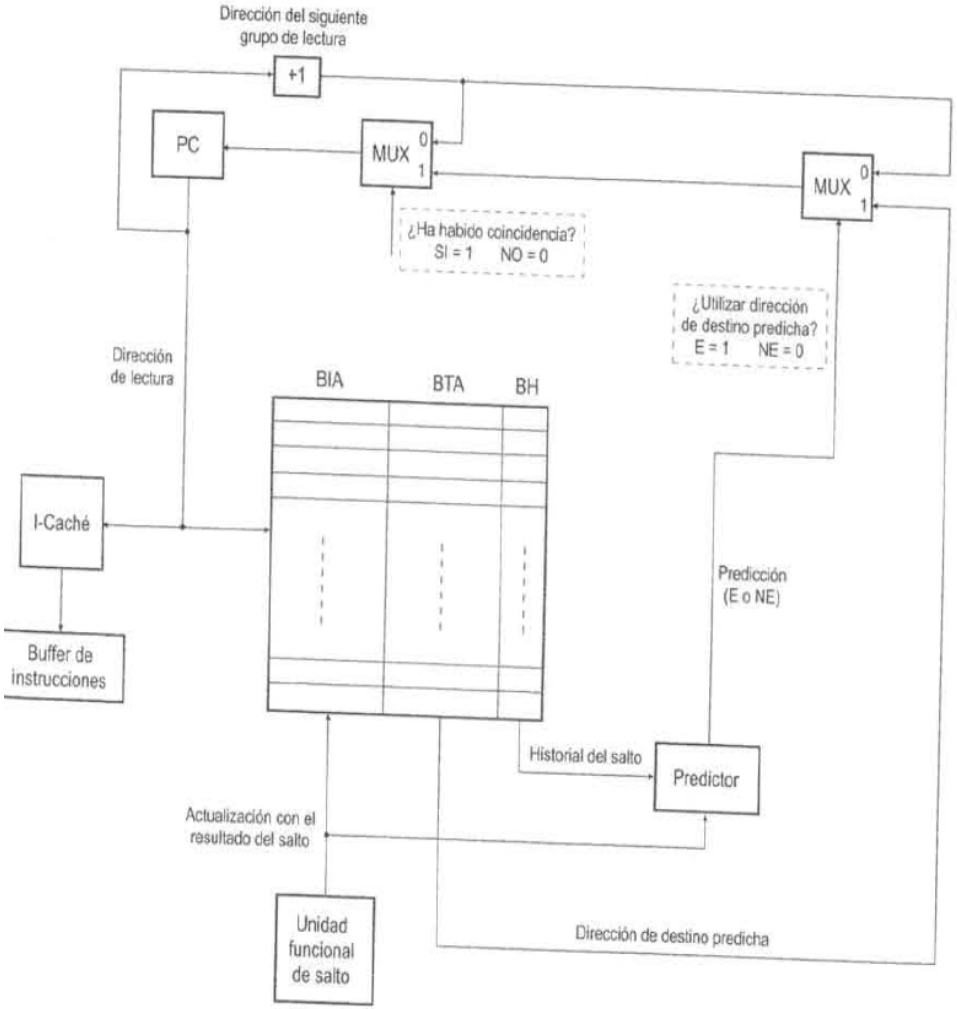
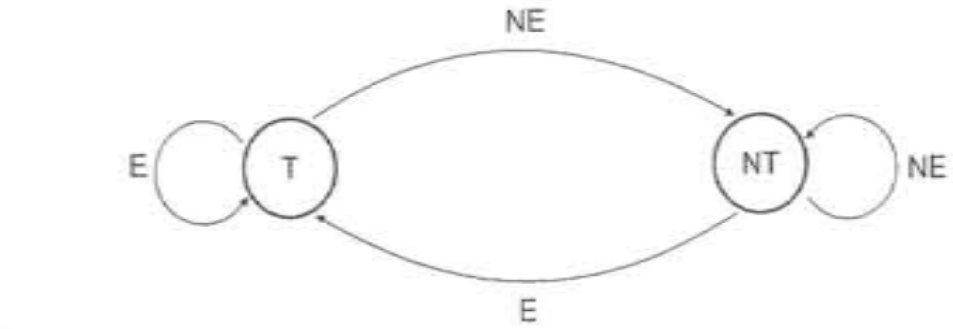


Figura 2.15: Esquema de una BTB con historial de salto.

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.



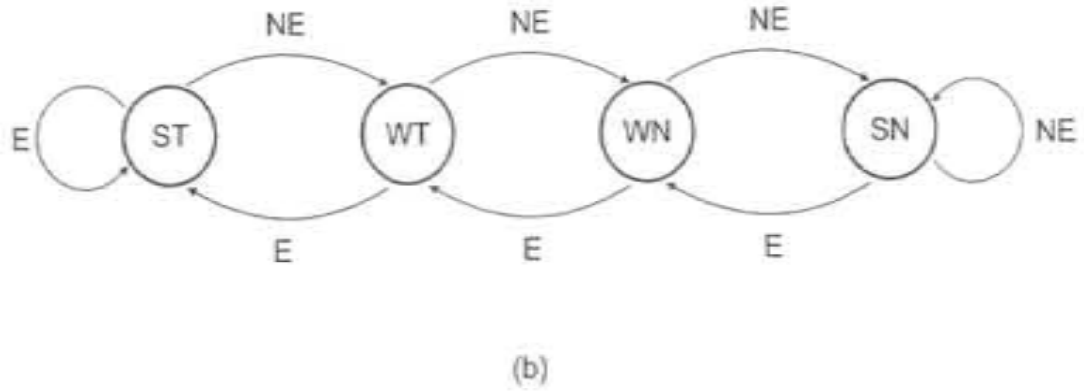


Figura 2.16: Predictores de Smith de 1 bit (a) y 2 bits de historial (b).

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
- 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.

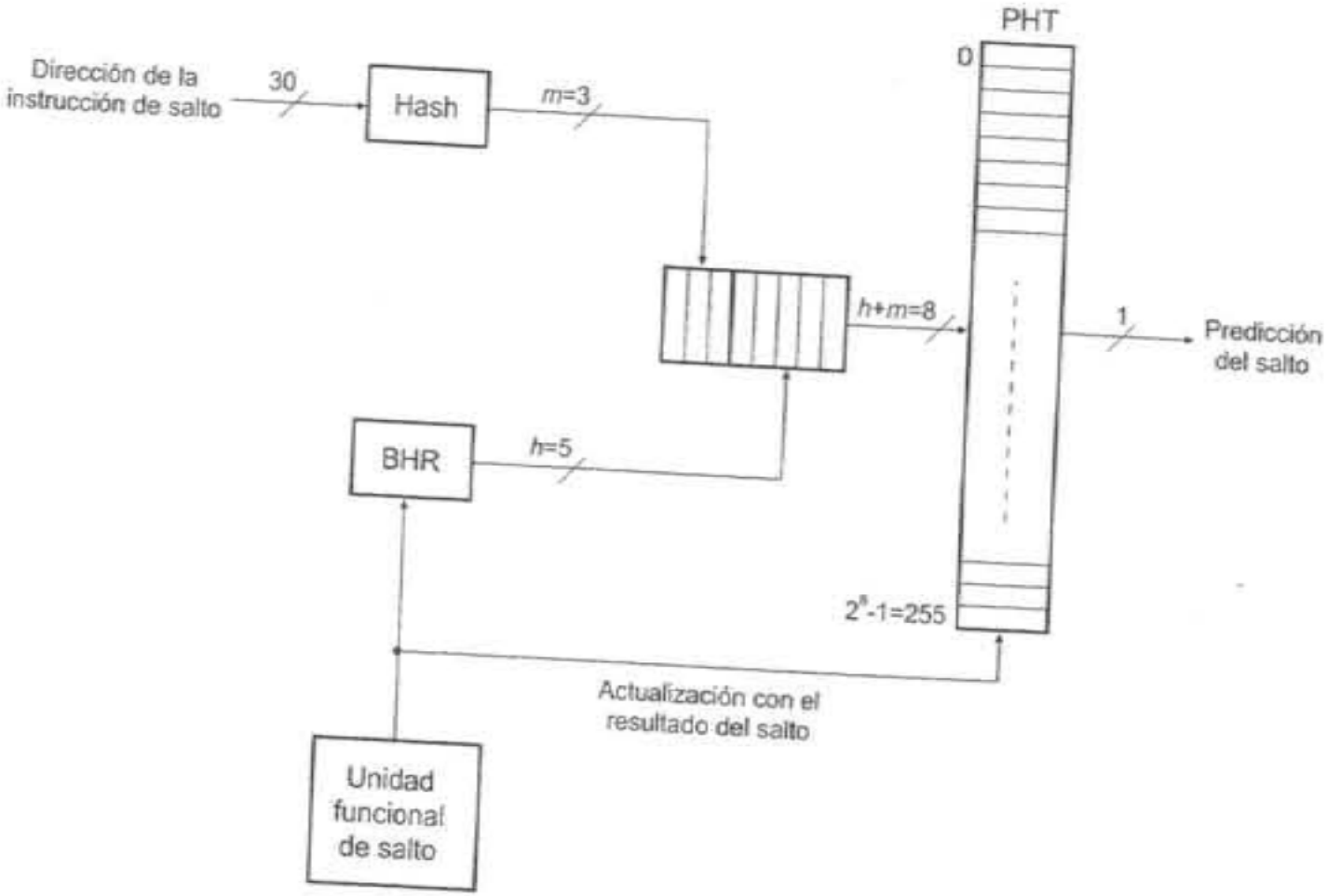


Figura 2.17: Esquema de un predictor de dos niveles basado en el historial global.

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.

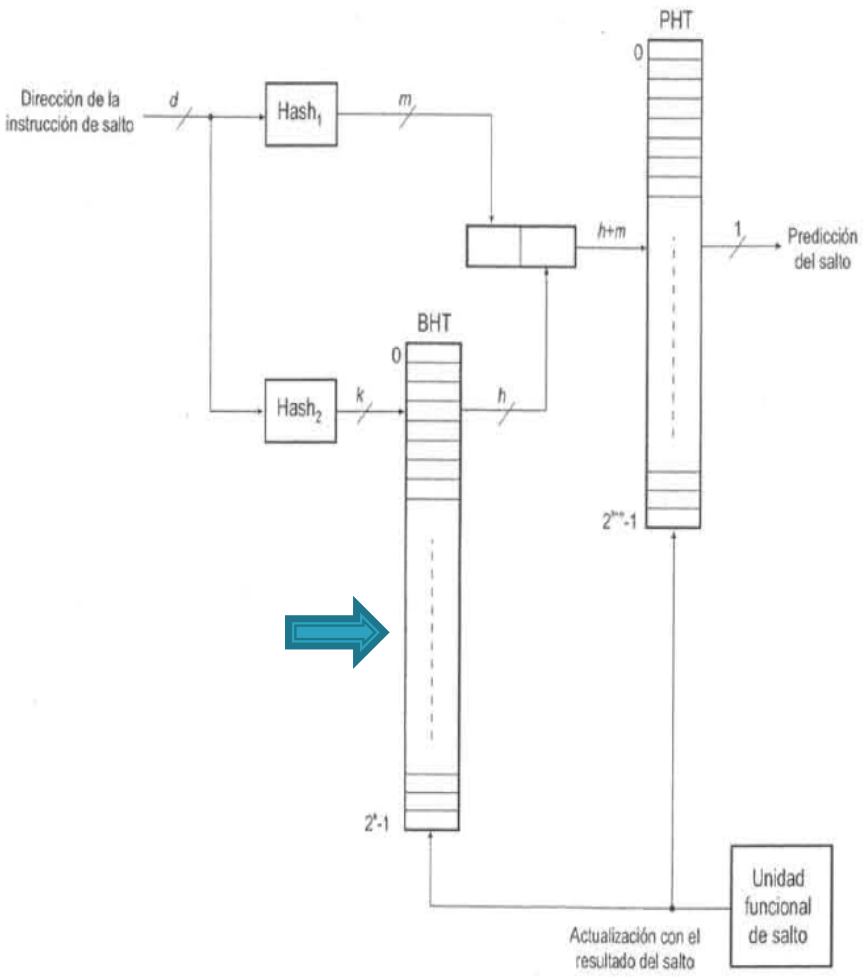


Figura 2.18: Esquema de un predictor de dos niveles basado en el historial local.

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.

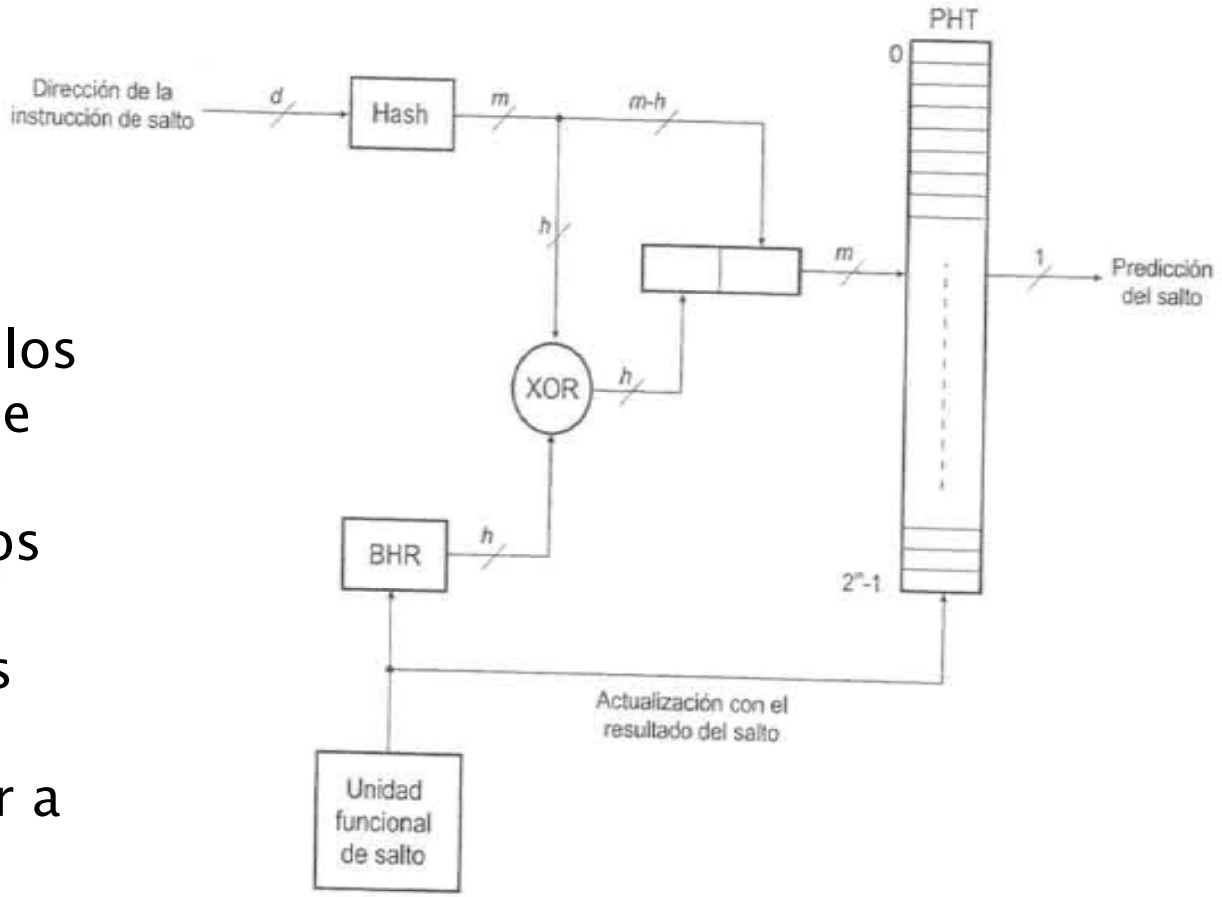


Figura 2.19: Esquema de un predictor *gshare*.

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

- La forma habitual para conocer y validar o anular las secuencias de instrucciones es etiquetarlas. Dos de esas etiquetas son:
  - La especulativa.
  - La de validez.
- 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.

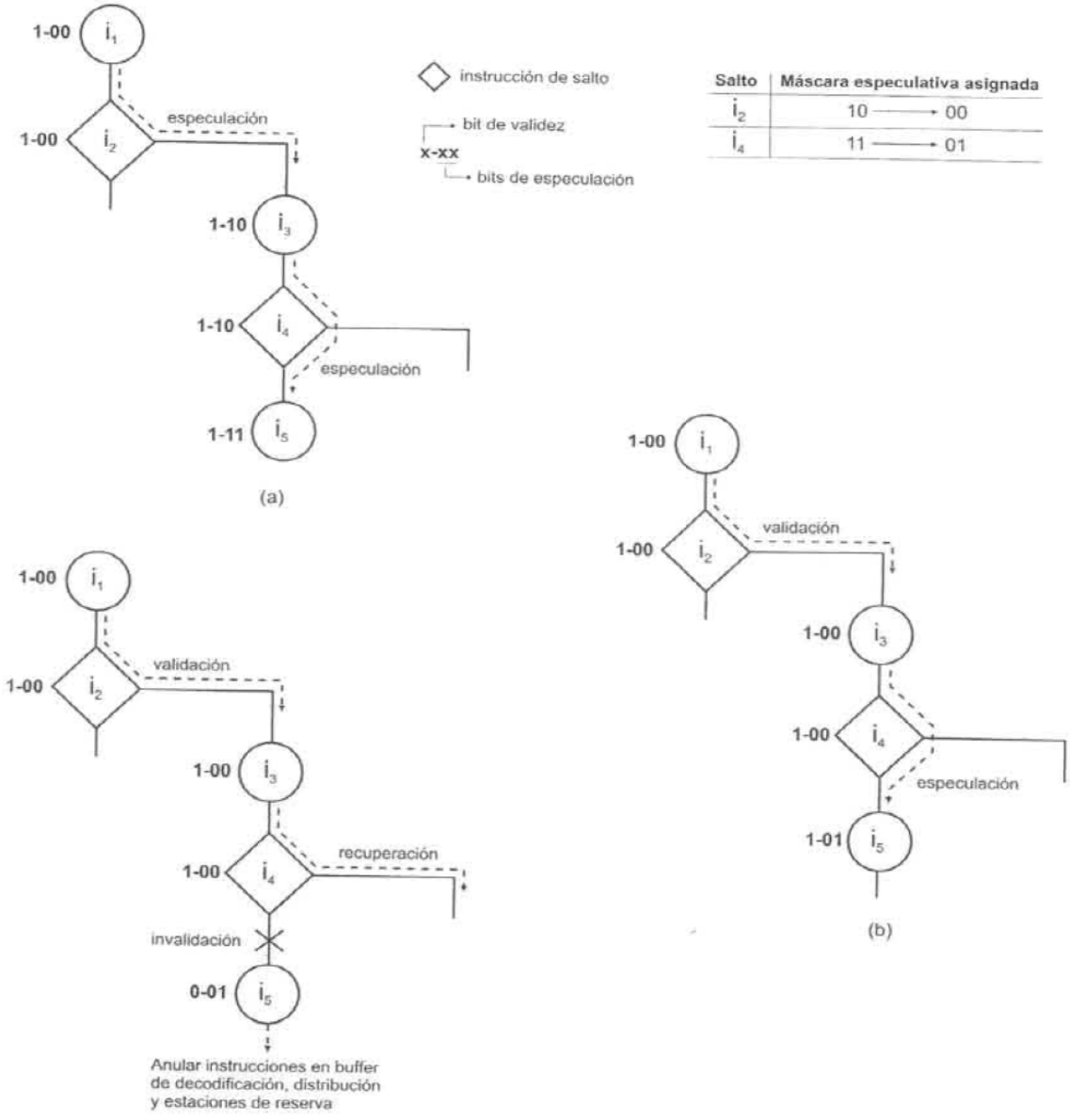


Figura 2.22: Secuencia de especulación, validación, invalidación y recuperación de dos rutas especuladas.

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.
    - Realizar una parte de la decodificación entes que la extracción de instrucciones de la I-caché, esto es la precodificación o decodificación previa.
    - 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.

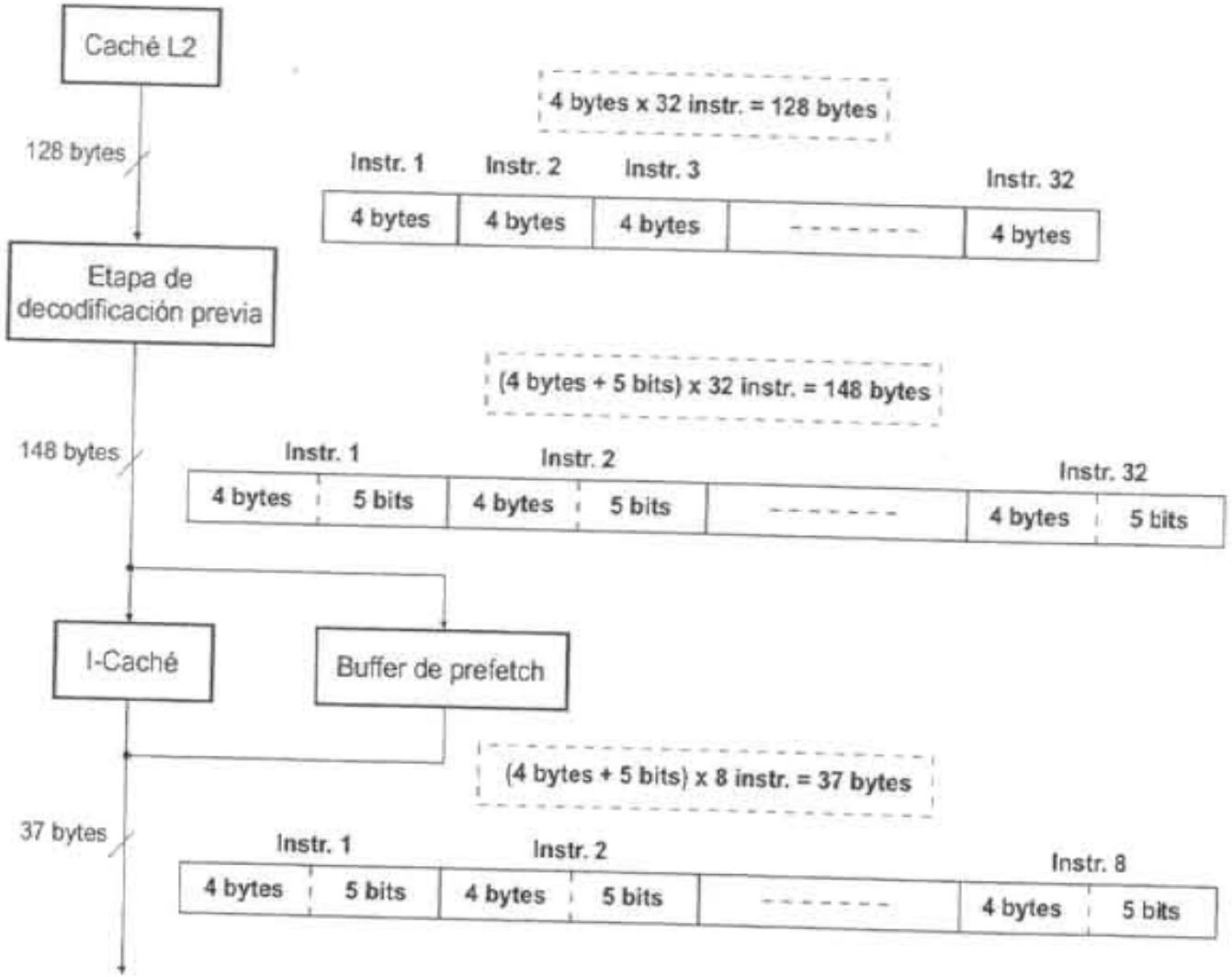


Figura 2.23: Decodificación previa en el PowerPC 970.

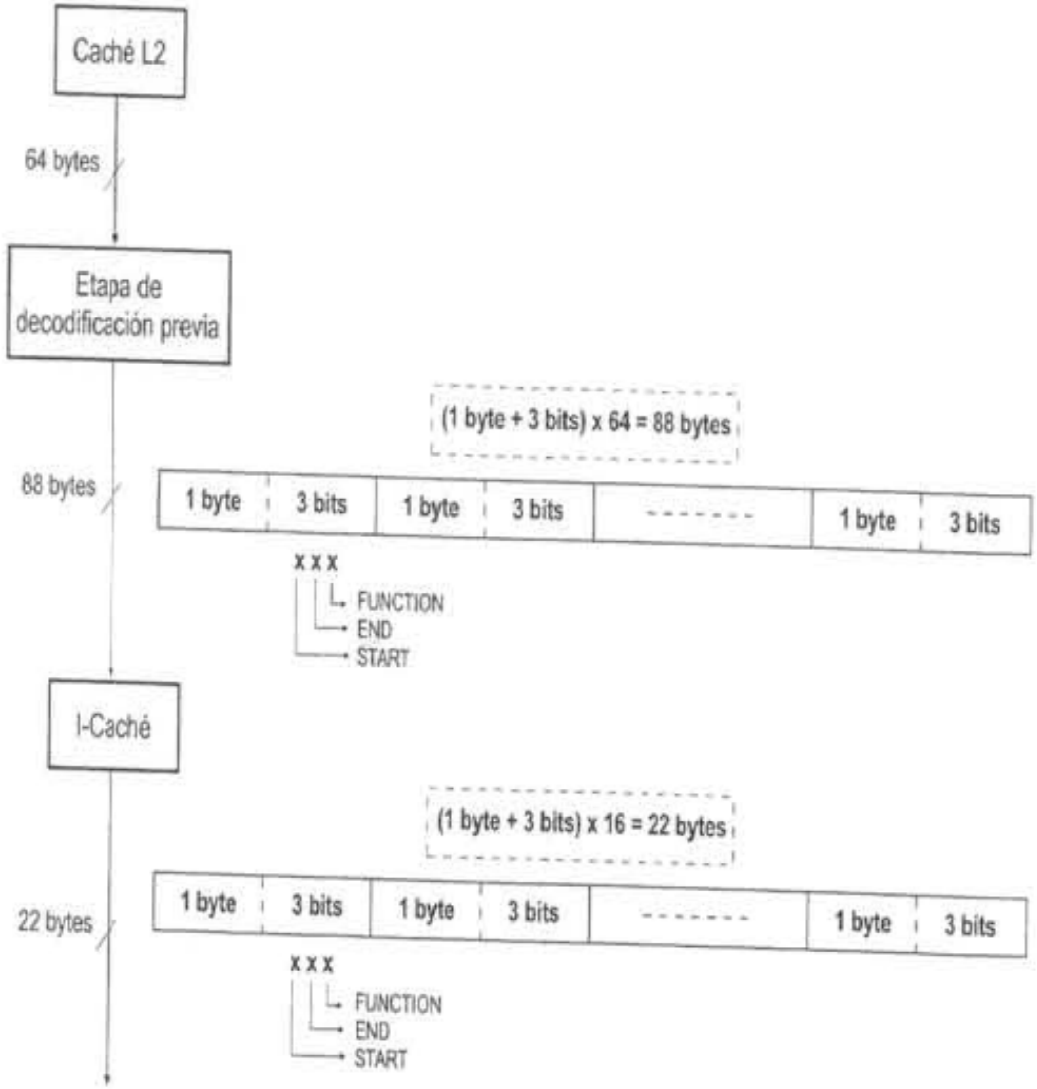


Figura 2.24: Decodificación previa en el AMD Opteron. Por simplificar, no se reflejan los 4 bits que se añaden cada 16 bytes.

2.6.2. Traducción de instrucciones

- Traducción de instrucciones complejas (CICS) en un conjunto de instrucciones mas básicas tipo RISC

Ejemplo de PowerPc

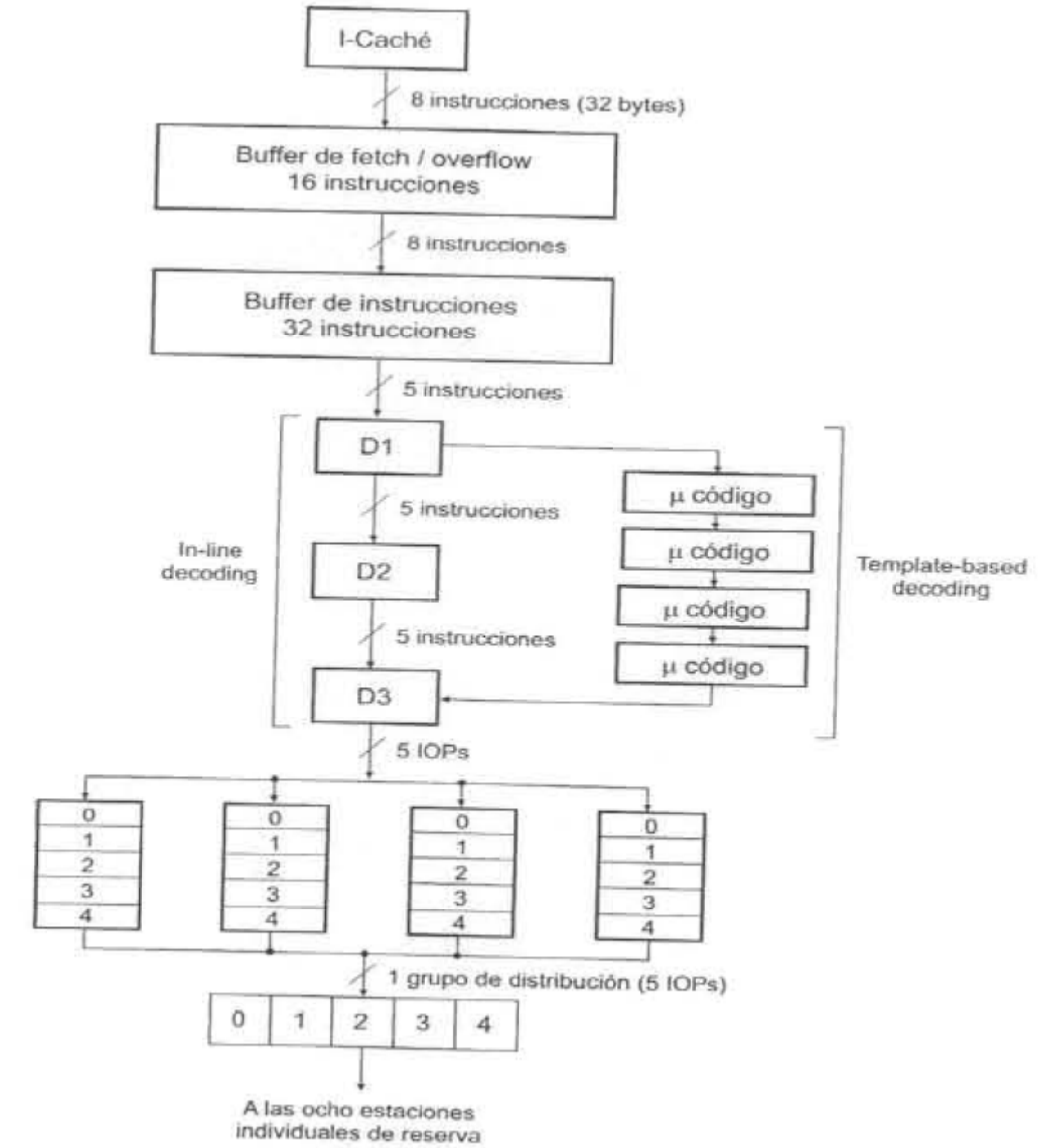


Figura 2.26: Etapa de decodificación del PowerPC 970.

Ejemplo de Intel core

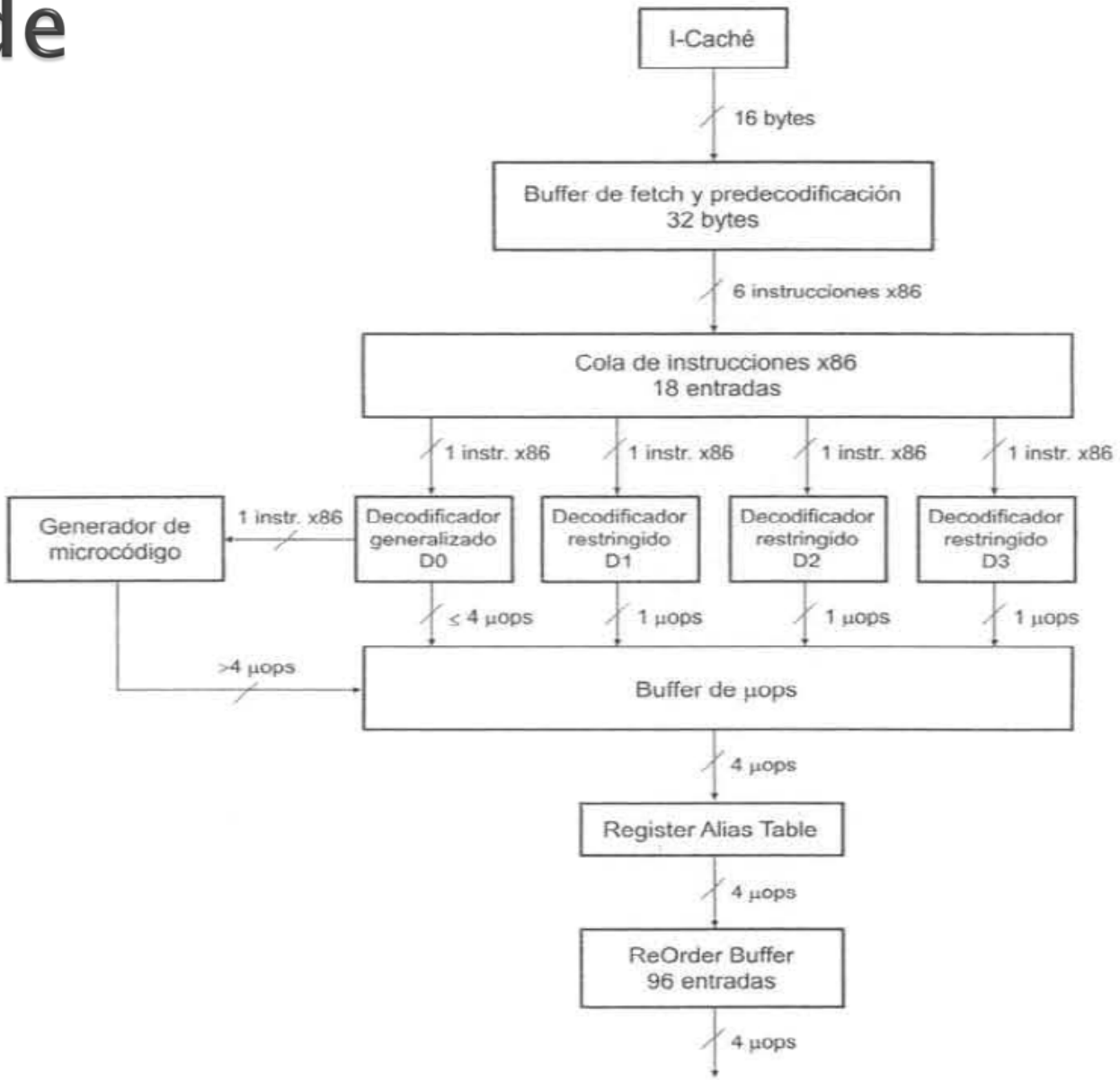


Figura 2.25: Decodificación en la arquitectura Intel Core Microarchitecture.

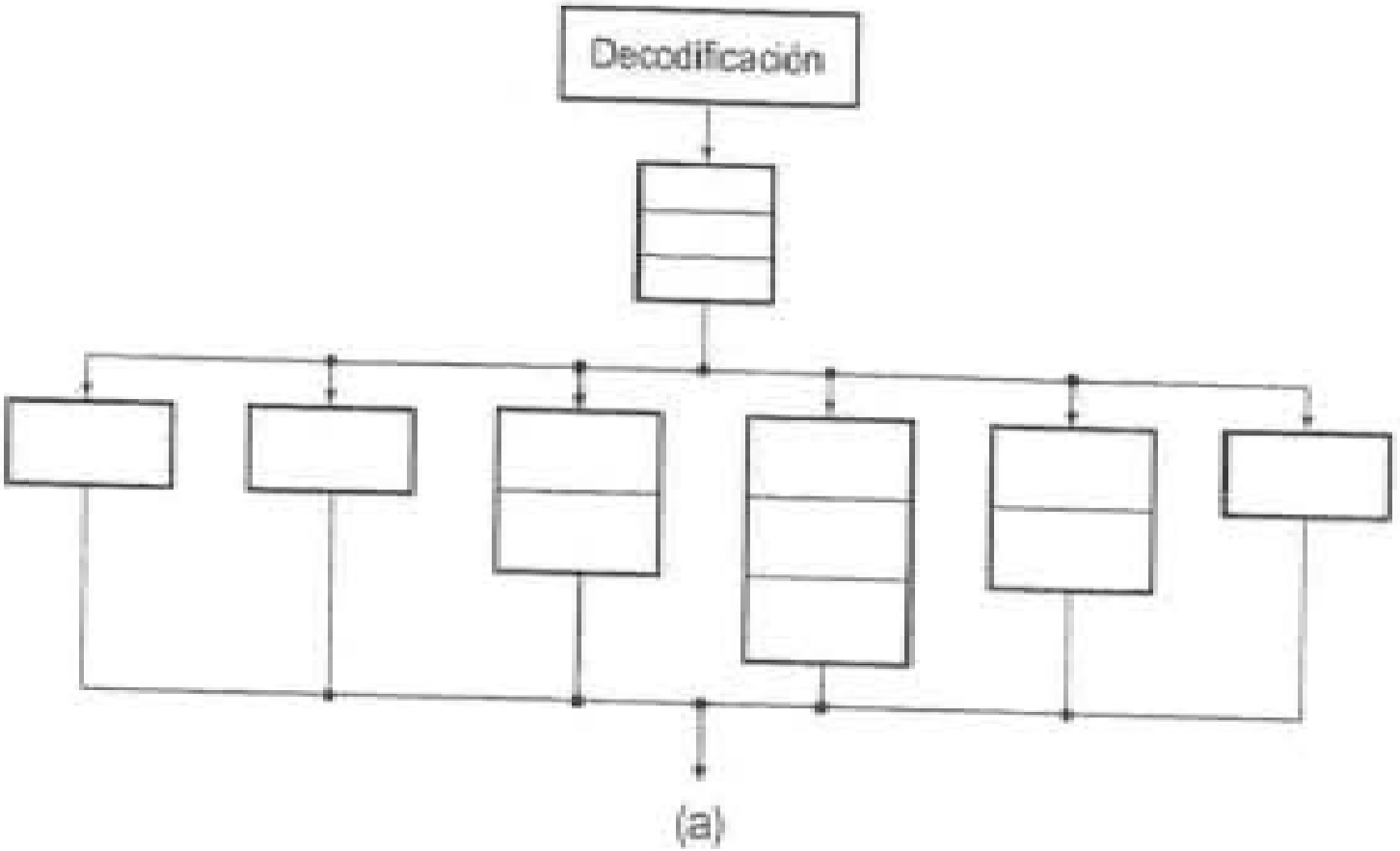
2.7 Distribución

- Se ocupa de repartir las instrucciones según su tipo entre las diferentes unidades funcionales para que se pueda proceder a su ejecución en paralelo
- 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
    - 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

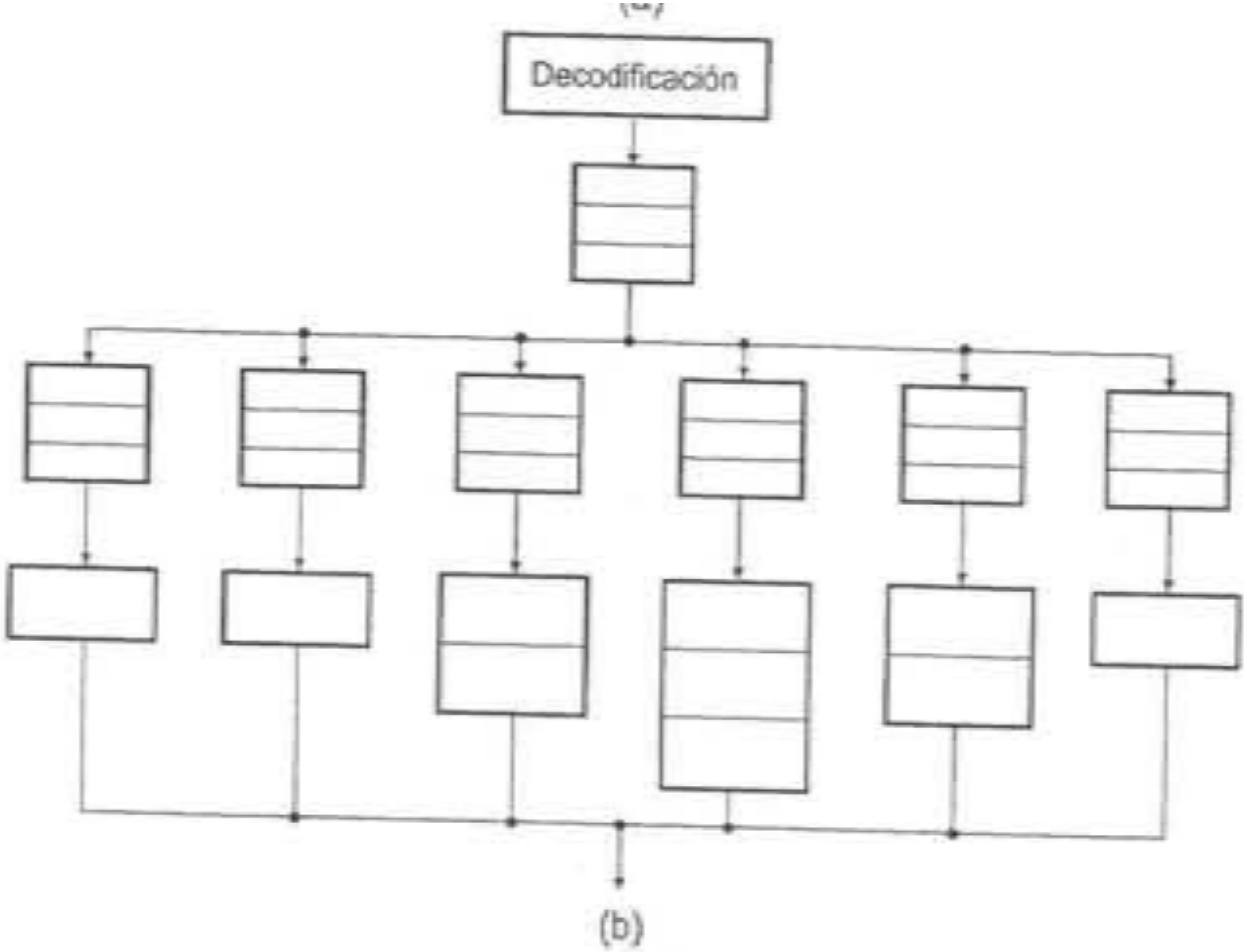
2.7.1. Organización de la ventana de instrucciones

- Estación de reserva centralizada
- Estación de reserva distribuida o individuales
- Estación de reserva en cluster o compartidas

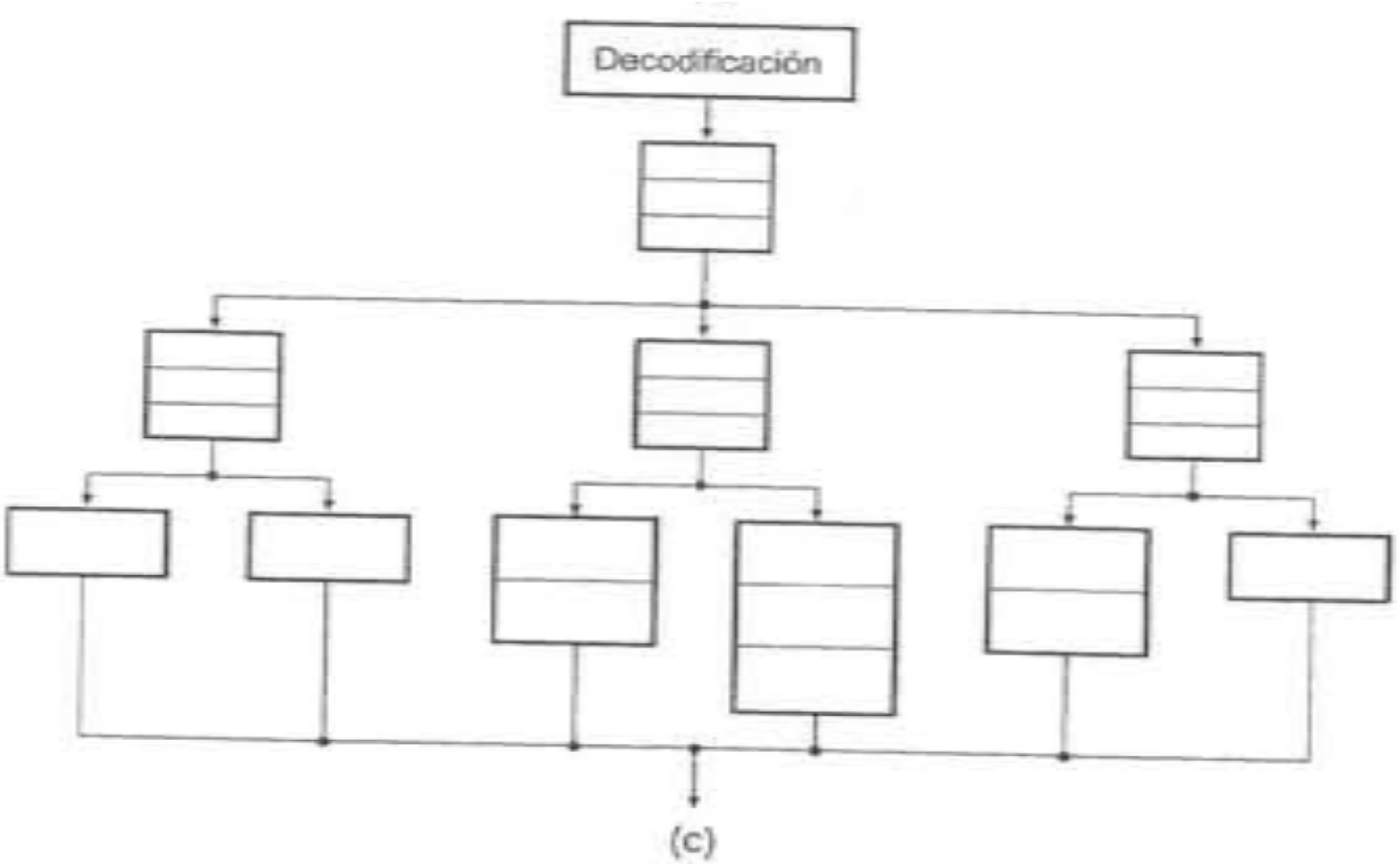
Centralizada



Distribuida



Compartida



2.7.2. Operativa de una estación de reserva individual

- 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

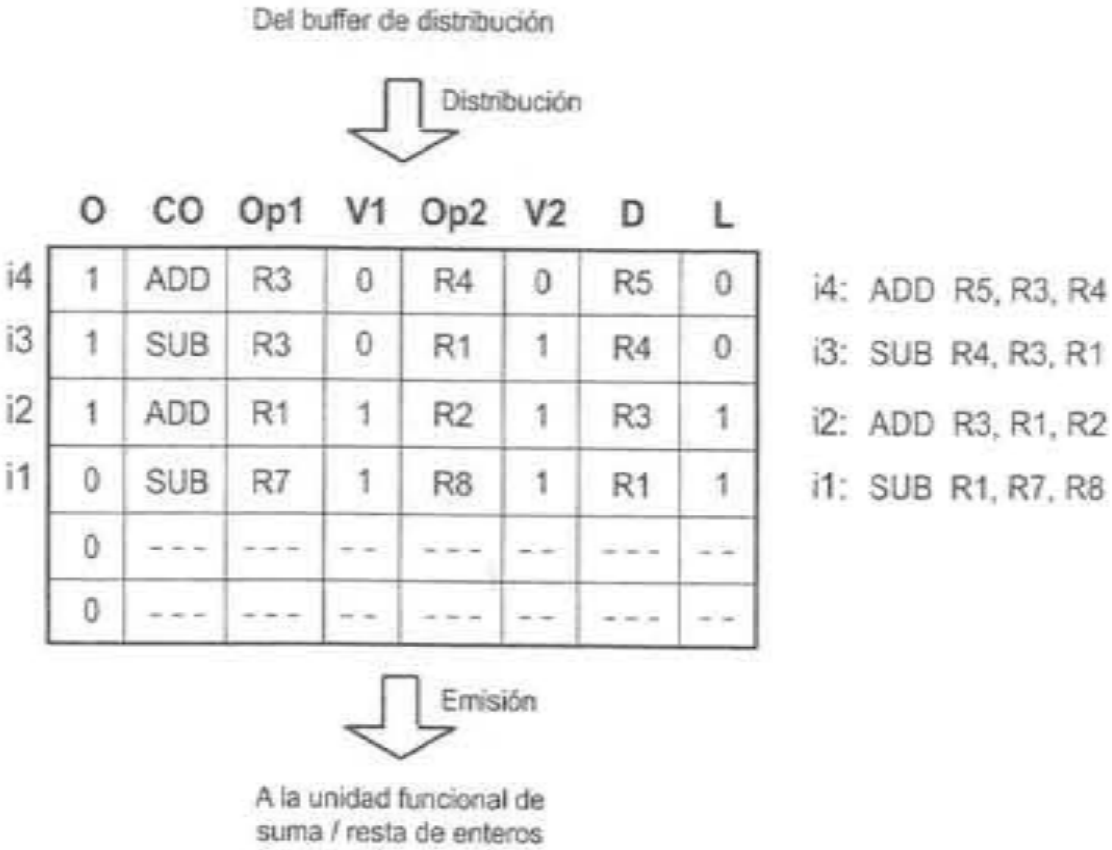


Figura 2.29: Estados de una instrucción en una estación de reserva individual.

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

	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

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

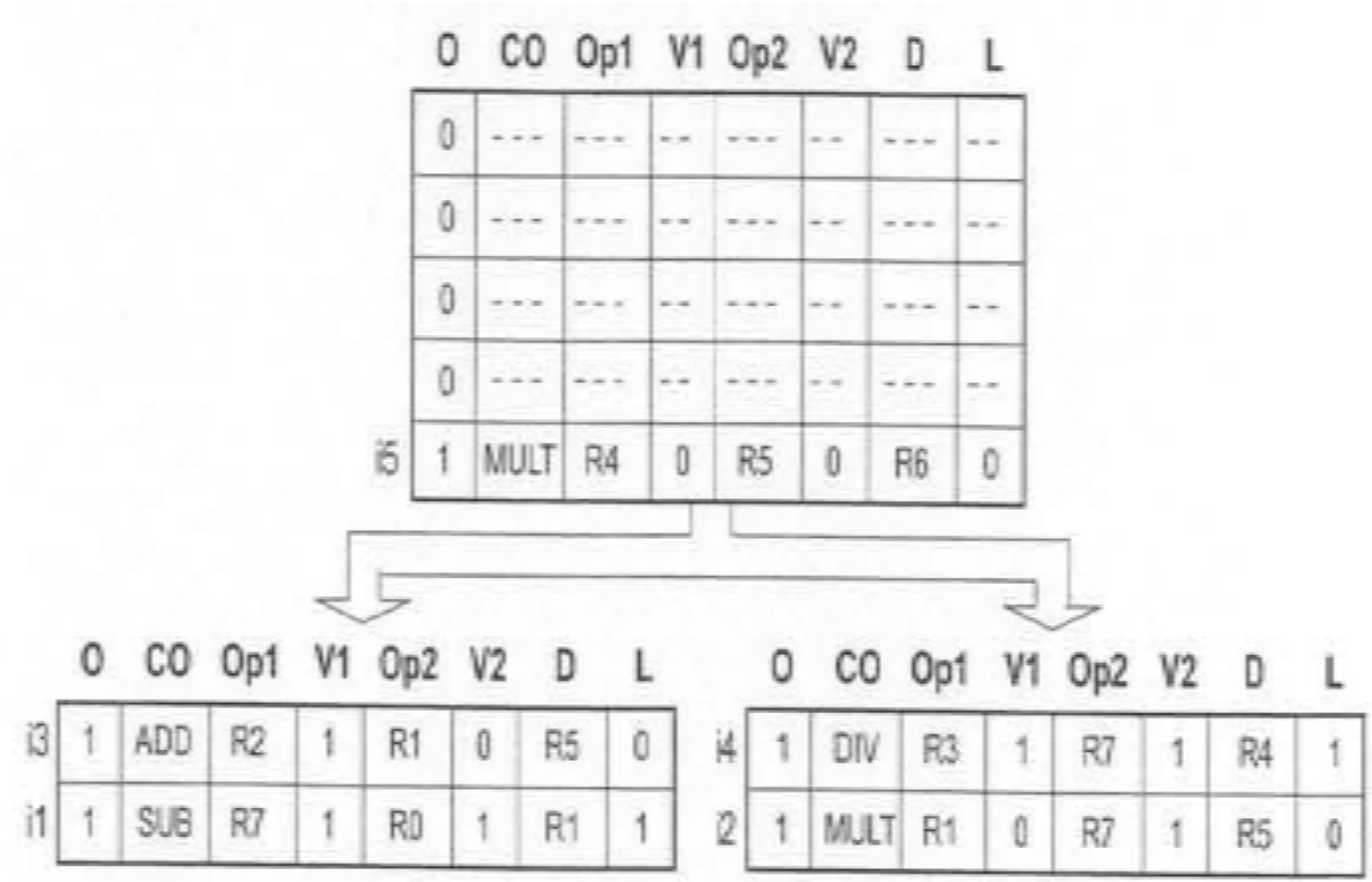


Figura 2.30: Ejemplo de emisión de instrucciones entre estaciones de reserva distribuidas (continúa).

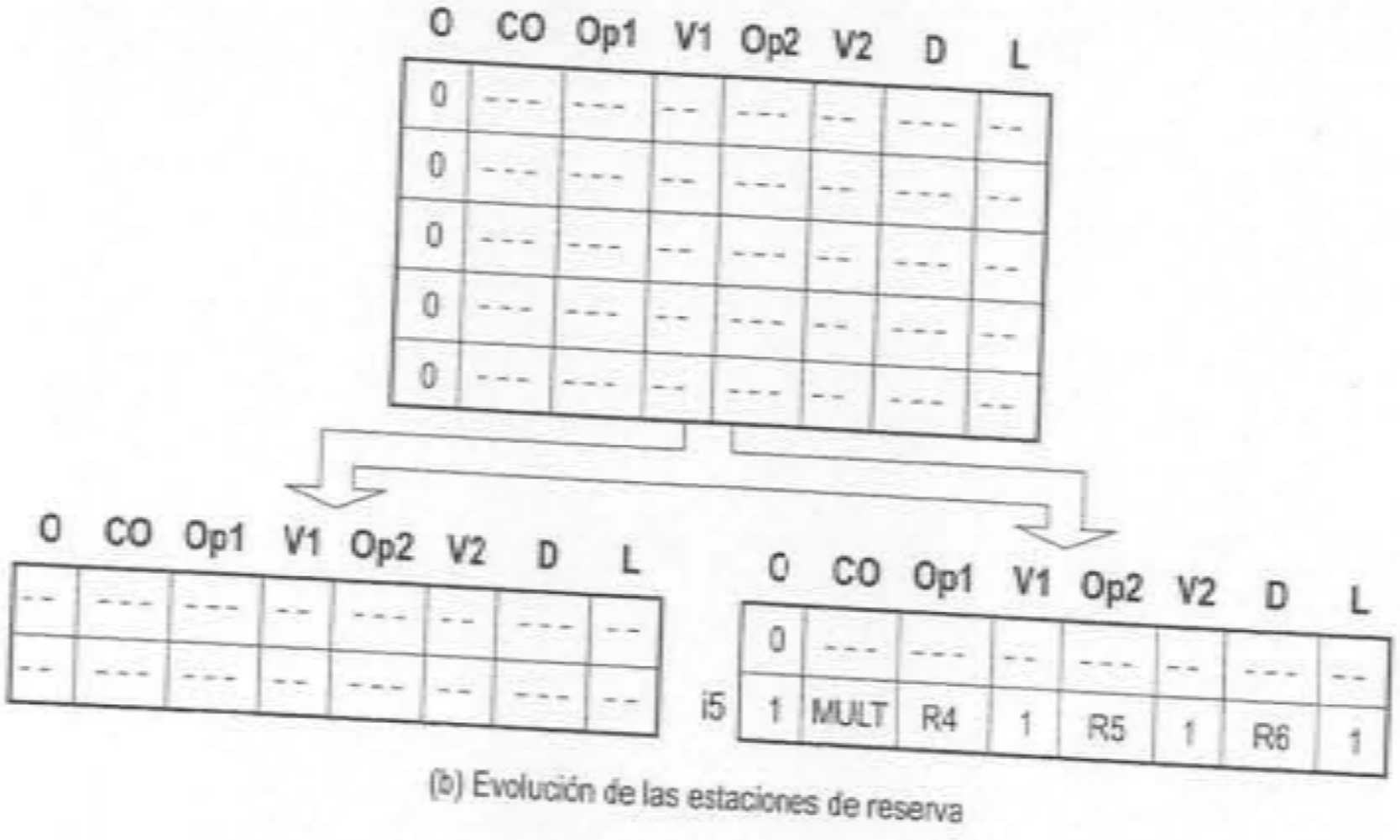
Ciclo i+2: Emisión de i1 e i4. Distribución de i5

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

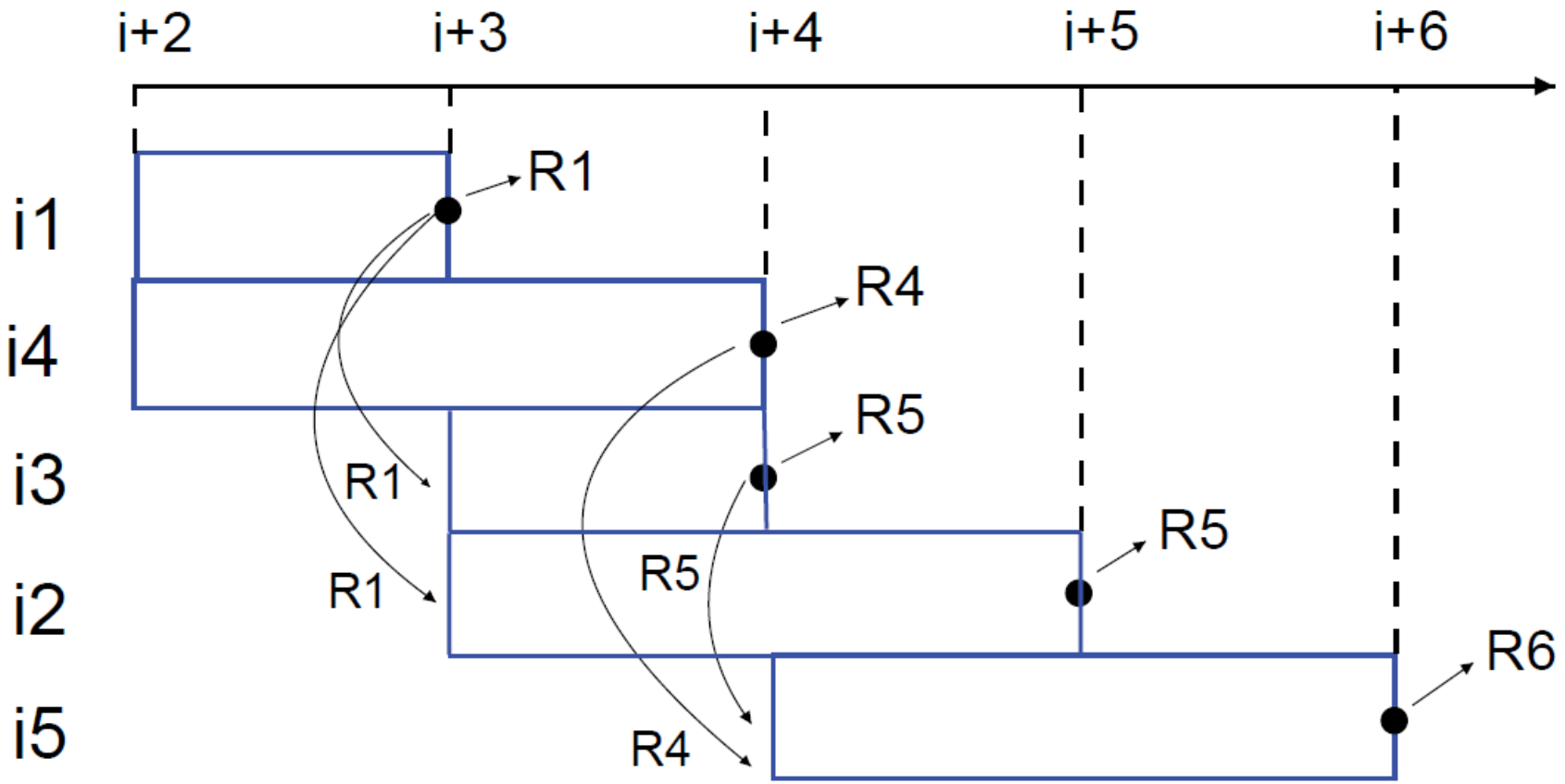
	O	CO	Op1	V1	Op2	V2	D	L
i3	1	ADD	R2	1	R1	0	R5	0

	O	CO	Op1	V1	Op2	V2	D	L
i5	1	MULT	R4	0	R5	0	R6	0
i2	1	MULT	R1	0	R7	1	R5	0

Ciclo i+3: Fin de ejecución de i1. Emisión de i2 e i3



(b) Evolución de las estaciones de reserva


i+2
i+3
i+4
i+5
i+6
i1
i4
i3
i2
i5
R1
R4
R5
R1
R5
R1
R5
R4
R6

2.7.2.1. Fase de distribución

- Envío de una instrucción desde el buffer de instruccion es a la estación de reserva individual

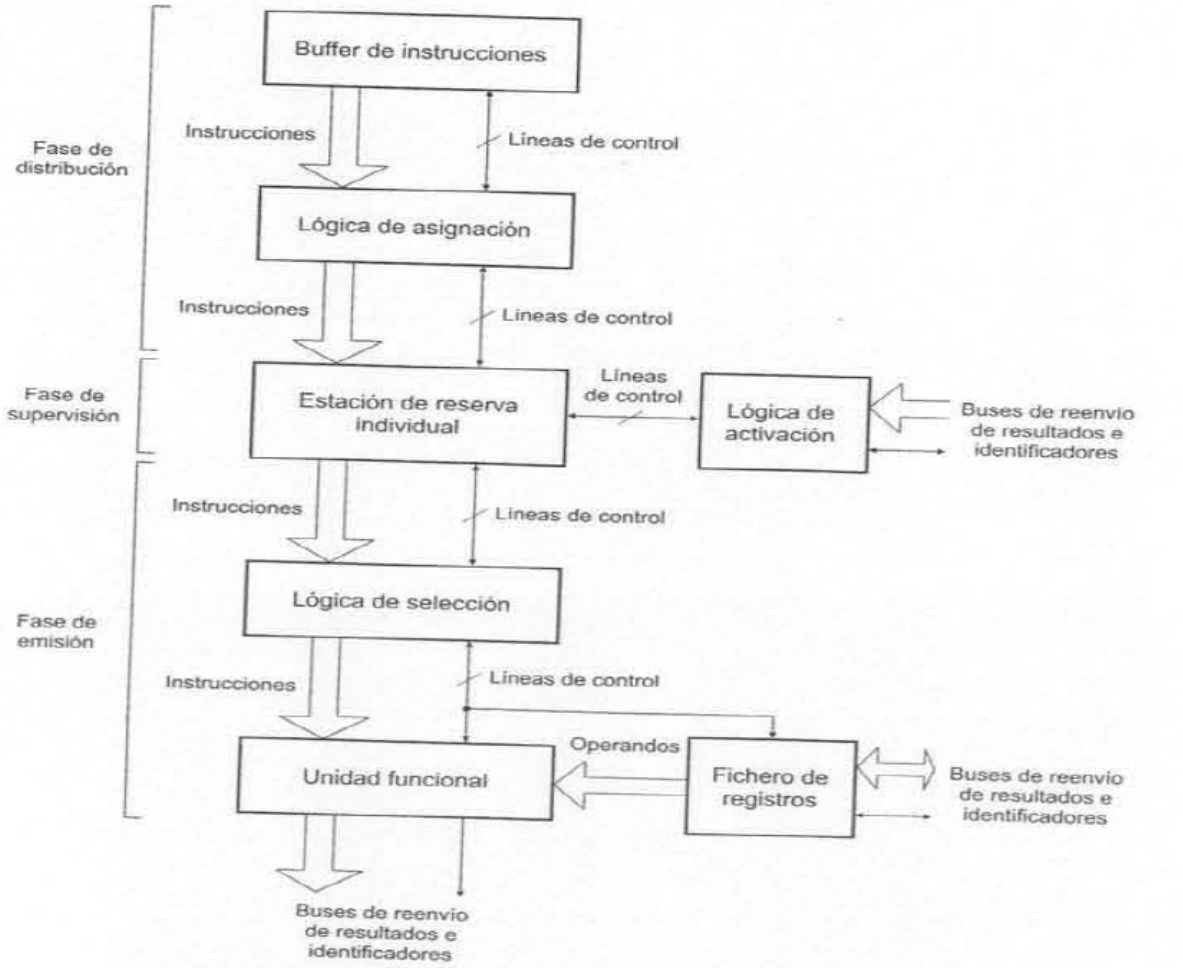


Figura 2.31: Fases de una instrucción en la etapa de distribución.

2.7.2.2. Fase de supervisión

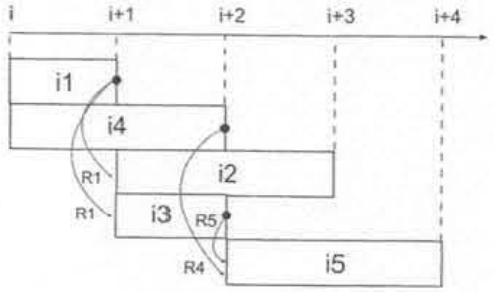
- La fase de supervisión concluye cuando tiene los dos operandos disponibles
- Revisa en CDB (Common Data Bus)
- El hardware que realiza la sipervisión de denomina lógica de activación

2.7.2.3. Fase de emisión

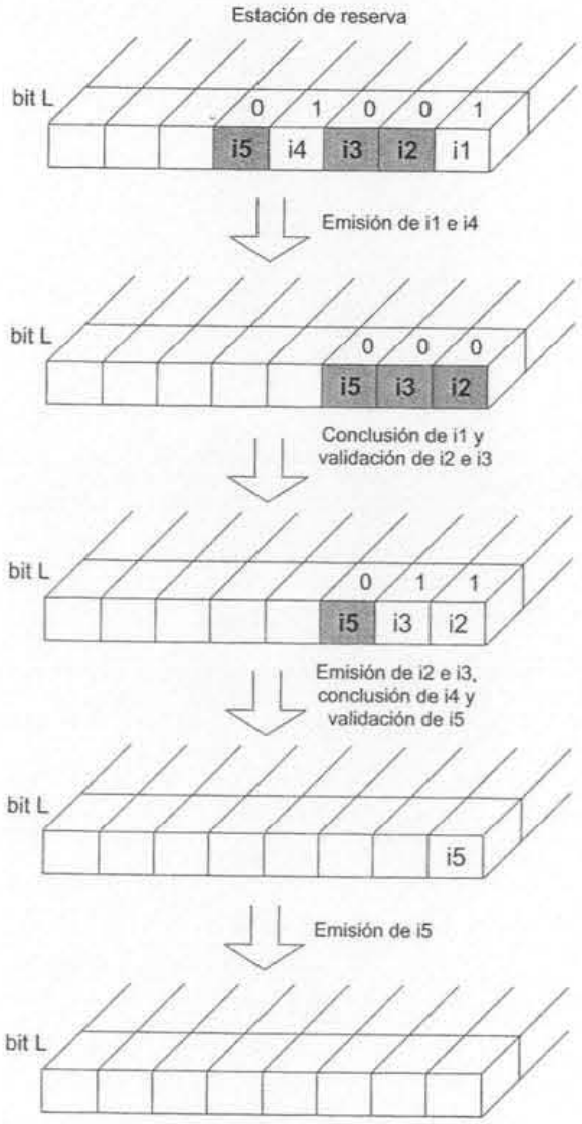
- La lógica de selección determina la instrucción que se puede emitir entre todas las disponibles
- Emisión sin bloqueo desordenada
- Política de planificación emisión la mas antigua
- Emisión alineada
  - La ventana de distribución no puede enviar nuevas instrucciones a la estación de reserva hasta que no esté completamente vacía
- Emisión no alineada
  - Puede emitir instrucciones siempre que queden entradas libres en las estaciones de reserva

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



(c) Secuencia temporal de ejecución y reenvío de operandos

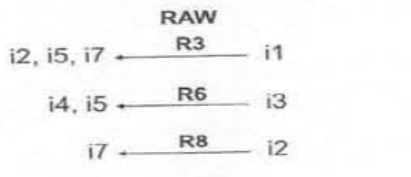


(b) Evolución de la estación de reserva

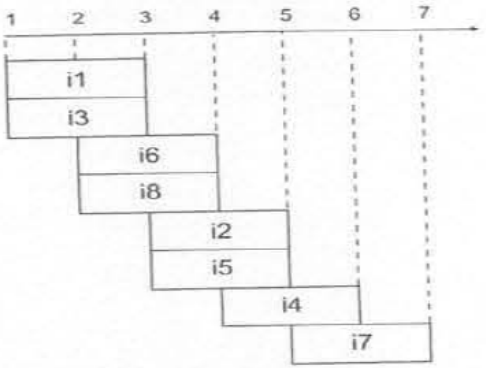
Figura 2.32: Ejemplo de emisión de una secuencia de cinco instrucciones a una estación de reserva individual que alimenta dos unidades funcionales.

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

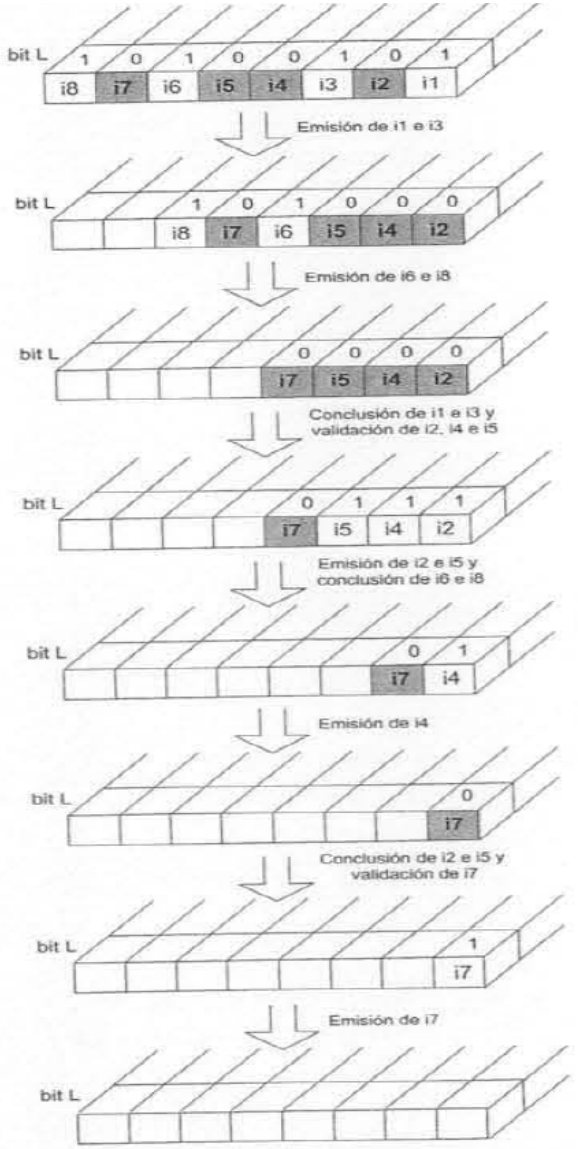
(a) Secuencia de instrucciones



(b) Dependencias



(c) Secuencia temporal de ejecución y reenvío de operandos

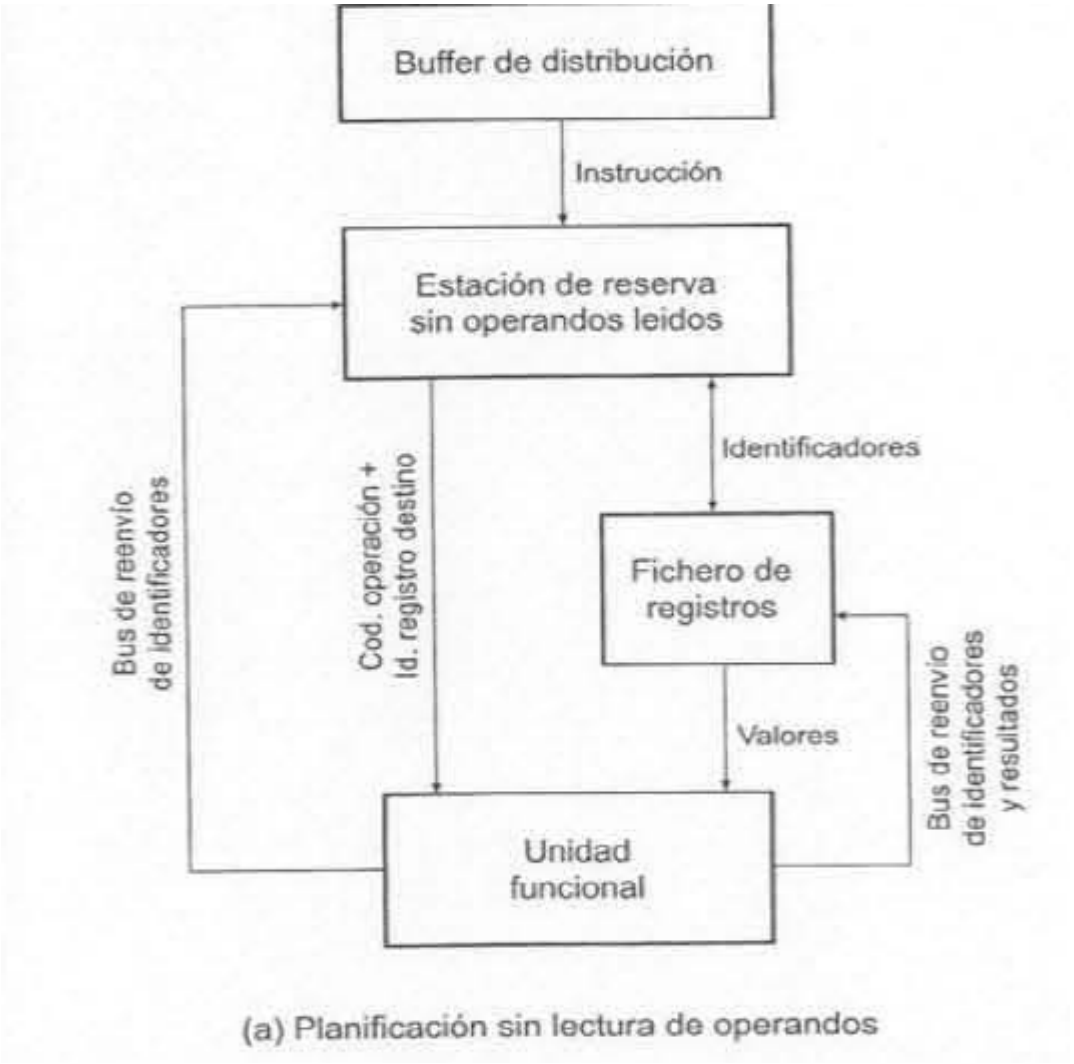


(d) Evolución de la estación de reserva

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

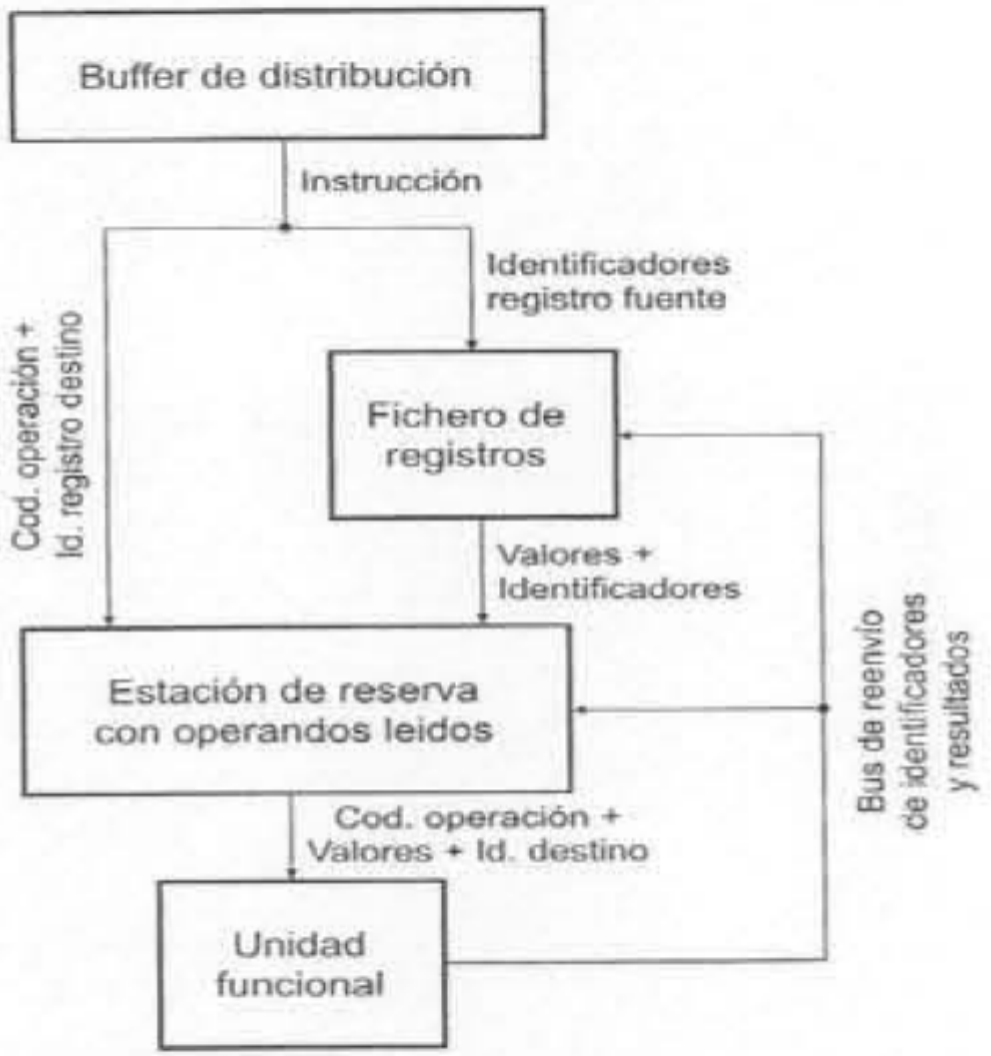
- Planificación sin lectura de operando
  - La lectura de operandos fuentes se hace en el momento de la emisión a las unidades funcionales



(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

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	---	---	--	---	--	---	--

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

igura 2.35: Ejemplo de aplicación de la planificación

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

Final ciclo i+2: Finalización de i1. Quedan disponibles 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	---	---	--	---	--	---	--
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

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

Final ciclo i+3: Finalización de i3e i4. Queda disponible 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	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

Ciclo i+4: Se emite 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	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	---	---	--	---	--	---	--

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	---	---	--	---	--	---	--

Ciclo i+5: Continúa el procesamiento 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	---	---	--	---	--	---	--

Final ciclo i+5: Finalización de if

Final ciclo i+5: Finalización 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	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	---	---	--	---	--	---	--

2.7.4. Renombramiento de registros

- 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
```

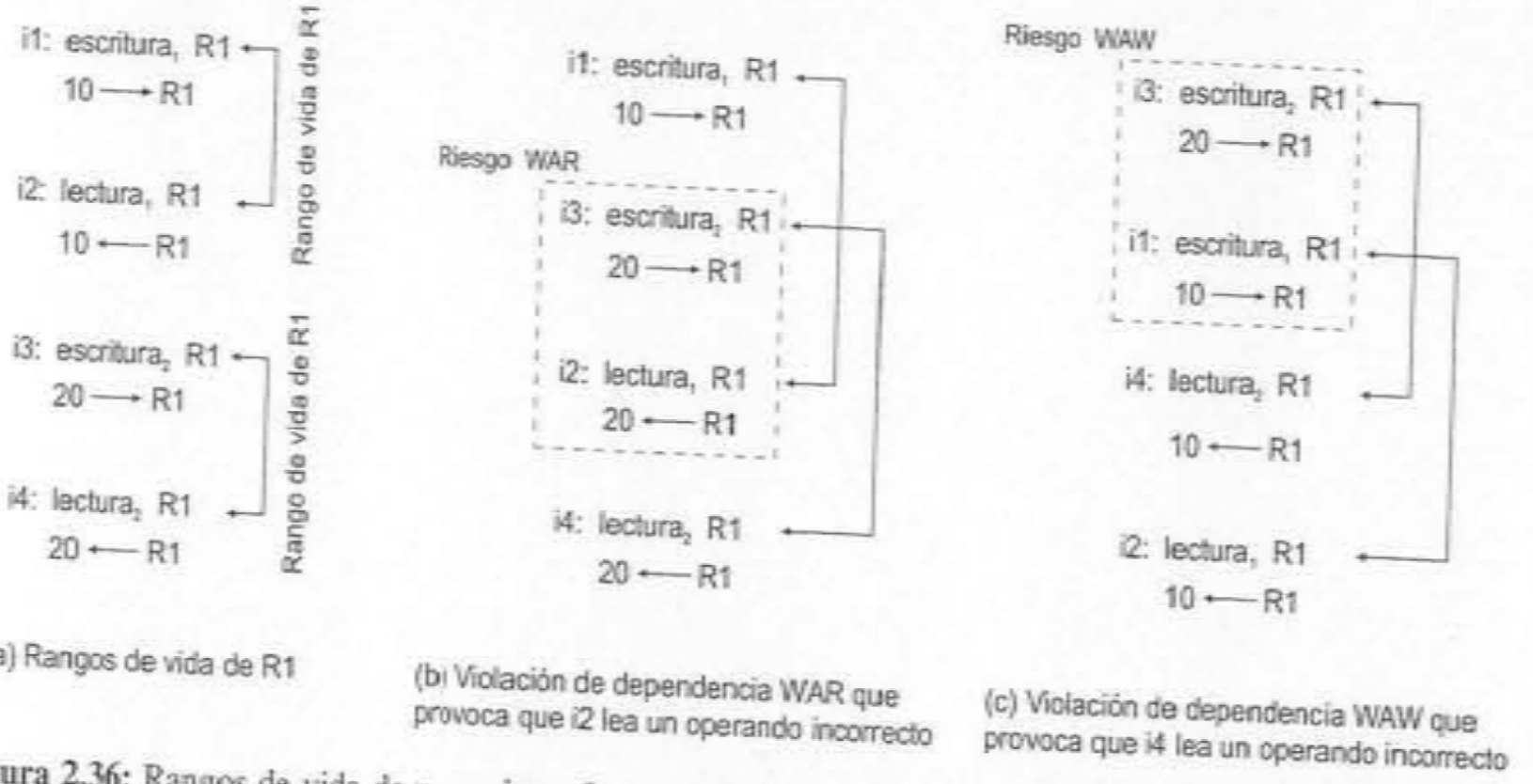


Figura 2.36: Rangos de vida de un registro. Los diferentes posibles solapamientos de dos rangos de vida de un mismo registro inducen diferentes tipos de riesgos.

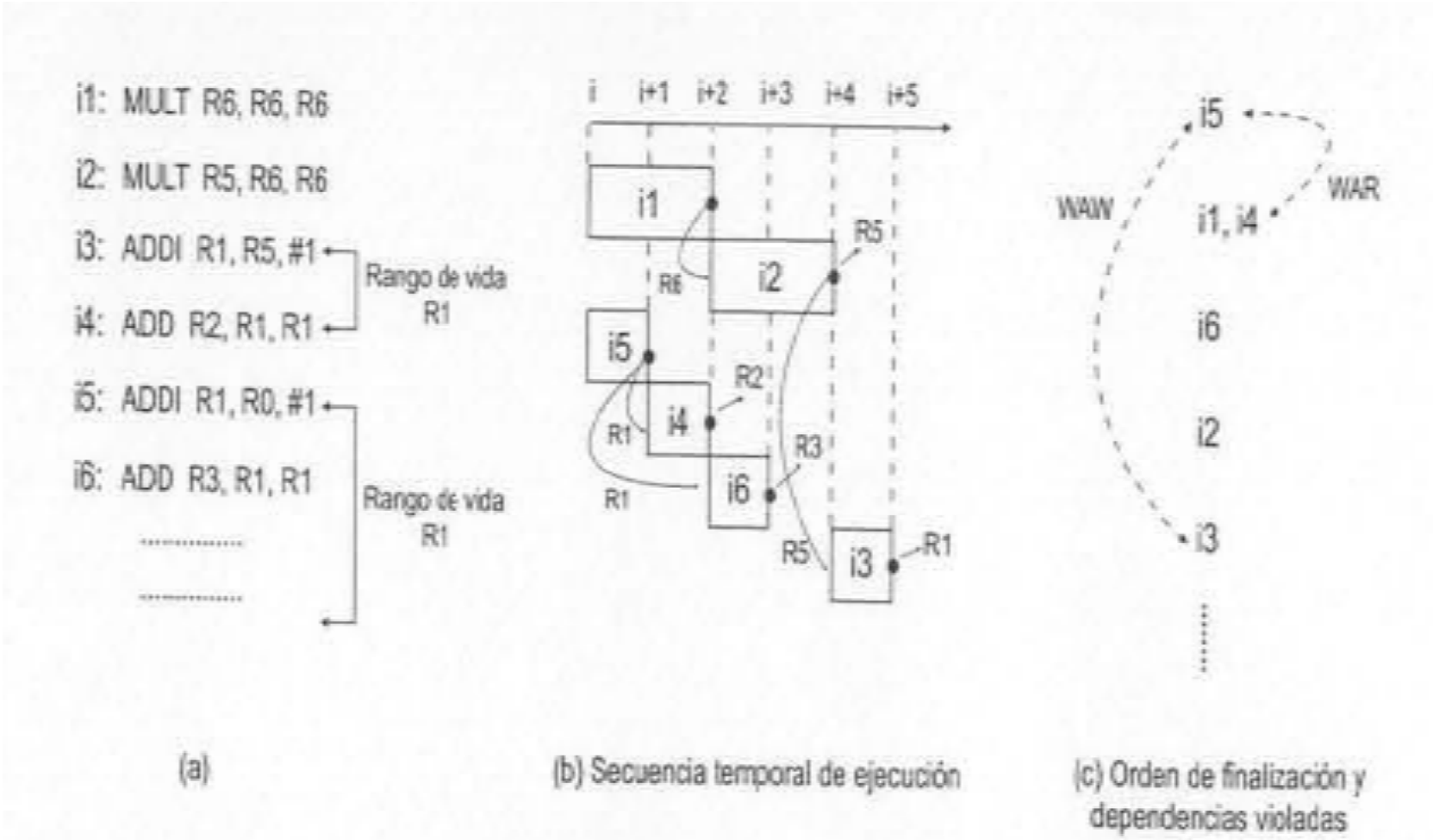
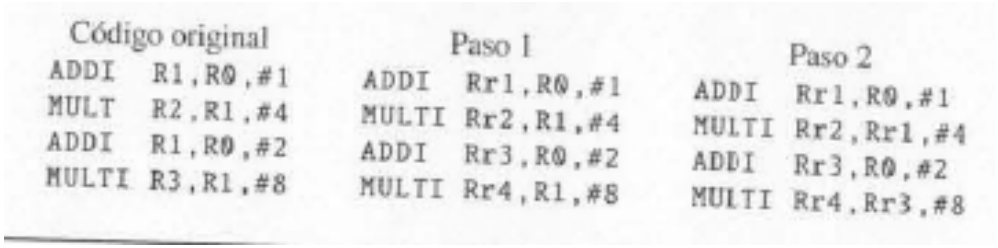


Figura 2.37: Violación de dependencias WAR y WAW como consecuencia de la ejecución fuera de orden.

Renombramiento dinámico de registros de la arquitectura mediante hardware

- Consiste en utilizar registros auxiliares invisibles al programador
  - Las instrucciones escriben los resultados en los registros de renombramientos para posteriormente escribirlos en los arquitectónicos
- 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



Código original

```
ADDI   R1,R0,#1
MULT   R2,R1,#4
ADDI   R1,R0,#2
MULTI  R3,R1,#8
```

Paso 1

```
ADDI   Rr1,R0,#1
MULTI  Rr2,R1,#4
ADDI   Rr3,R0,#2
MULTI  Rr4,R1,#8
```

Paso 2

```
ADDI   Rr1,R0,#1
MULTI  Rr2,Rr1,#4
ADDI   Rr3,R0,#2
MULTI  Rr4,Rr3,#8
```

- 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

- ARF
  - Ocupado: El registro ha sido renombrado
  - Indice: apunta a la entrada RFF
- RRF
  - Válido: indica que todavía no se ha realizado la escritura en RRF
  - Ocupado: si el registro está siendo utilizado por instrucciones pendientes de ejecución

ARF (16 entradas)

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

RRF (8 entradas)

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

2.7.4.2. Organización independiente del RRF con acceso asociativo

- 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

ARF (16 entradas)

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

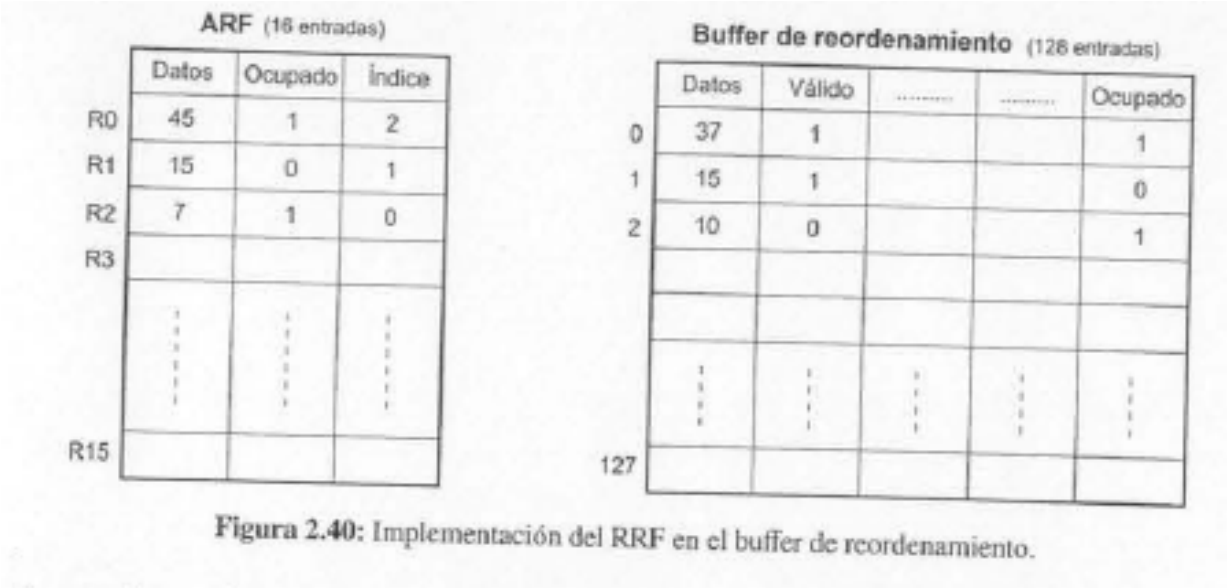
RRF (8 entradas)

	Destino	Datos	Válido	Último	Ocupado
Rr0	2	37	1	1	1
Rr1	1	15	1	1	0
Rr2	0	10	0	1	1
Rr3	3	35	1	0	1
Rr4	3	8	0	1	1
⋮	⋮	⋮	⋮	⋮	⋮
Rr7					

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

2.7.4.3. Organización del RRF como parte del buffer de reordenamiento

- Como parte del buffer de reordenamiento o terminacion
- Añadir dos entradas
  - Datos
  - Válido



ARF (16 entradas)

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

Buffer de reordenamiento (128 entradas)

	Datos	Válido	.....	.....	Ocupado
0	37	1			1
1	15	1			0
2	10	0			1
⋮	⋮	⋮	⋮	⋮	⋮
127					

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

2.7.5. Ejemplo de procesamiento de instrucciones con renombramiento

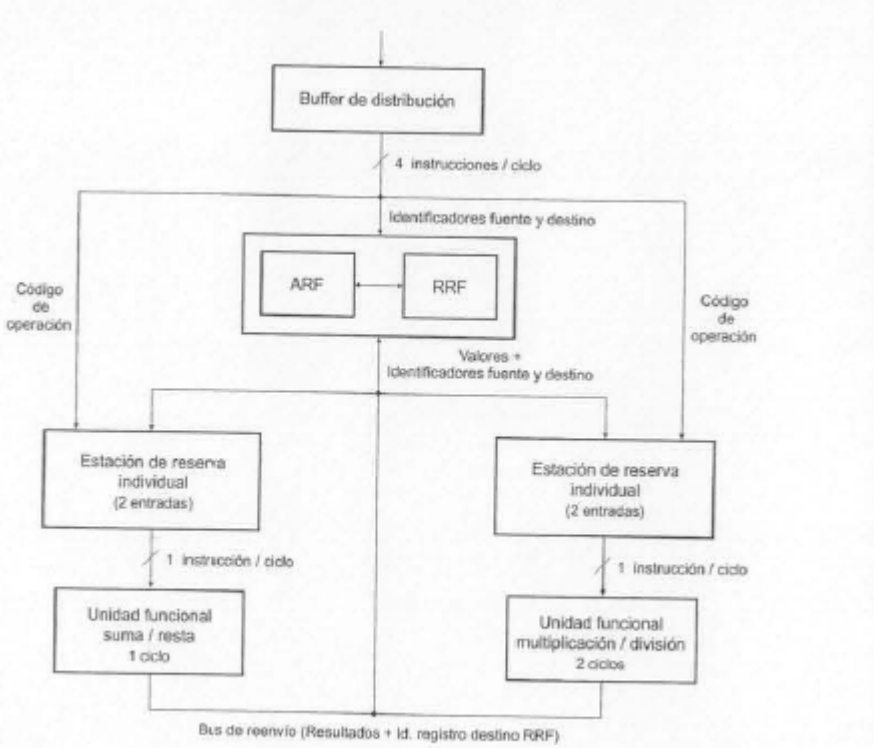


Figura 2.41: Núcleo de ejecución con planificación dinámica con lectura de operandos y RRF independiente con acceso indexado.

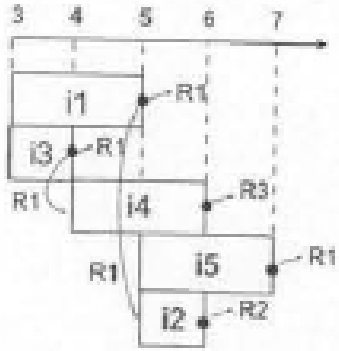
i1: MULT R1, R0, R3

i2: ADD R2, R1, R0

i3: SUB R1, R0, R4

i4: MULT R3, R1, R0

i5: MULT R1, R4, R0



(a) Secuencia de instrucciones

Ciclo 1: Recepción de las instrucciones del decodificador

	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

ARF

	Datos	Índice	Ocupado
R0	5		0
R1	10		0
R2	20		0
R3	30		0
R4	40		0

RRF

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

O	CO	Op1	v1	Op2	v2	D	L
0							
0							

O	CO	Op1	v1	Op2	v2	D	L
0							
0							

Ciclo 2: Distribución de i1, i2, i3, i4 y renombramiento de R1, R2 y R3

	O	CO	Op1	Op2	D
	0				
	0				
	0				
	0				
i5	0	MULT	R4	R0	R1

ARF

	Datos	Índice	Ocupado
R0	5		0
R1	10	Rr2	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

	O	CO	Op1	v1	Op2	v2	D	L
i3	1	SUB	5	1	40	1	Rr2	1
i2	1	ADD	Rr0	0	5	1	Rr1	0

	O	CO	Op1	v1	Op2	v2	D	L
i4	1	MULT	Rr2	0	5	1	Rr3	0
i1	1	MULT	5	1	30	1	Rr0	1

Figura 2.42: Evolución del procesamiento de instrucciones empleando RRF (continúa).

Ciclo 3 (inicio): Se emiten i1 e i3. Se distribuye i5 y se renombra R1 por tercera vez

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

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

Ciclo 4 (inicio): Se emite i4, i1 se encuentra en el 2º ciclo. Se libera Rr2 por terminación de i3

	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	0
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

Ciclo 4 (final): Finaliza i1. Se publica su resultado (150) y su destino (Rr0)

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	1
Rr1		0	1
Rr2	-35	1	0
Rr3		0	1
Rr4		0	1

	O	CO	Op1	v1	Op2	v2	D	L
	0							
i2	1	ADD	150	1	5	1	Rr1	1

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

Figura 2.42: [Continuación]

Ciclo 5 (inicio): Se emiten I2 e I5. I4 se encuentra en el 2° ciclo. Se libera Rr0 por terminación de I1

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							

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)

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

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

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

Figura 2.42: [Continuación] Evolución del procesamiento de instrucciones empleando RRF (continúa).

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

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

2.8. Terminación

- 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

Campos de una entrada del buffer de reordenamiento

- *Ocupada* (O): 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

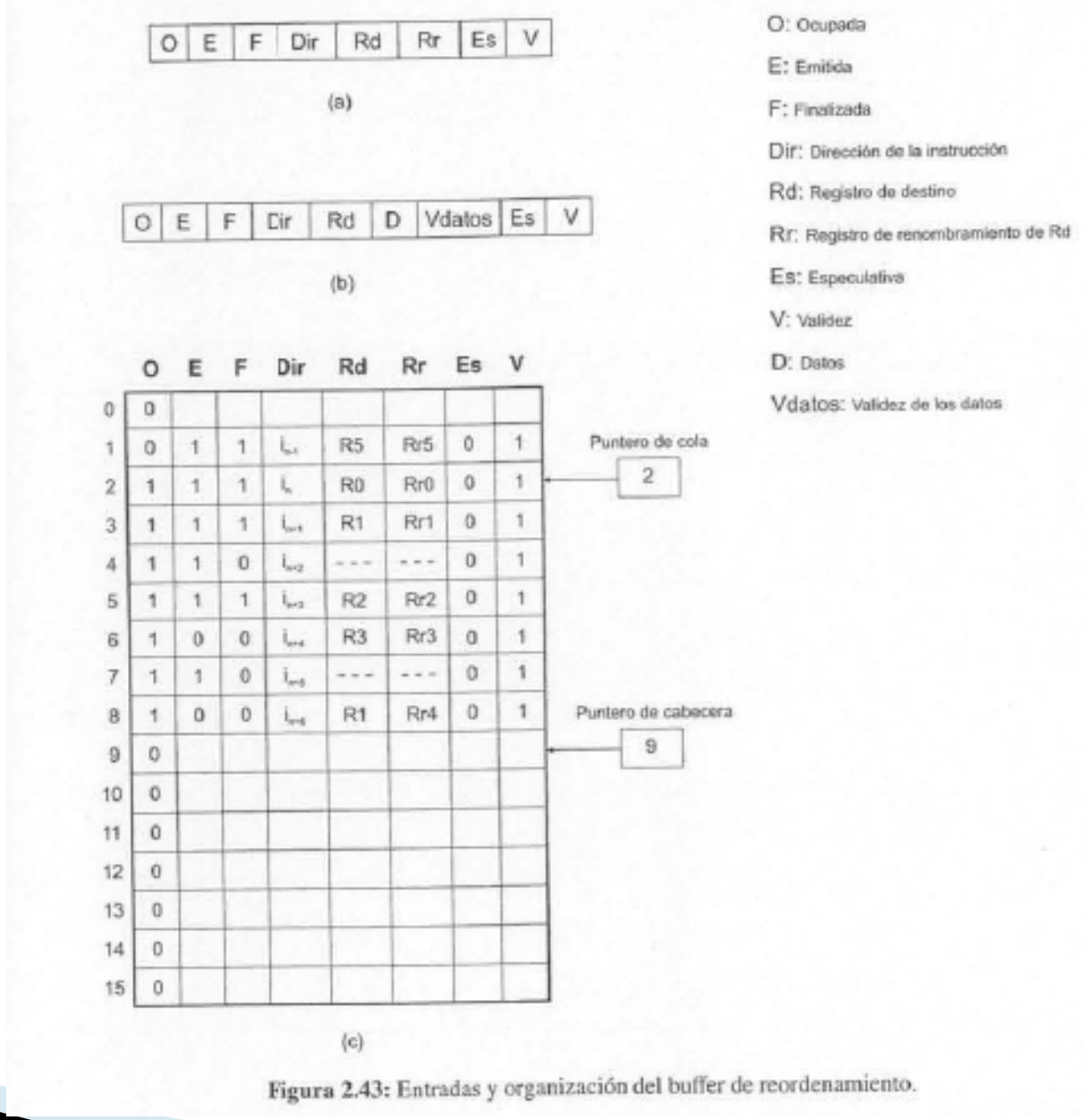


Figura 2.43: Entradas y organización del buffer de reordenamiento.

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

```
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 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

Figura 2.44: [Continuación] Evolución del buffer de terminación (continúa).

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

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

$$V_{pico} = \text{Máx. instrucciones terminadas por ciclo} * \text{Frecuencia procesador}$$

2.9. Retirada

- Es exclusiva de las instrucciones de almacenamiento
- Las instrucciones de carga/almacenamiento (seg. Superescalar) una vez emitidas, 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

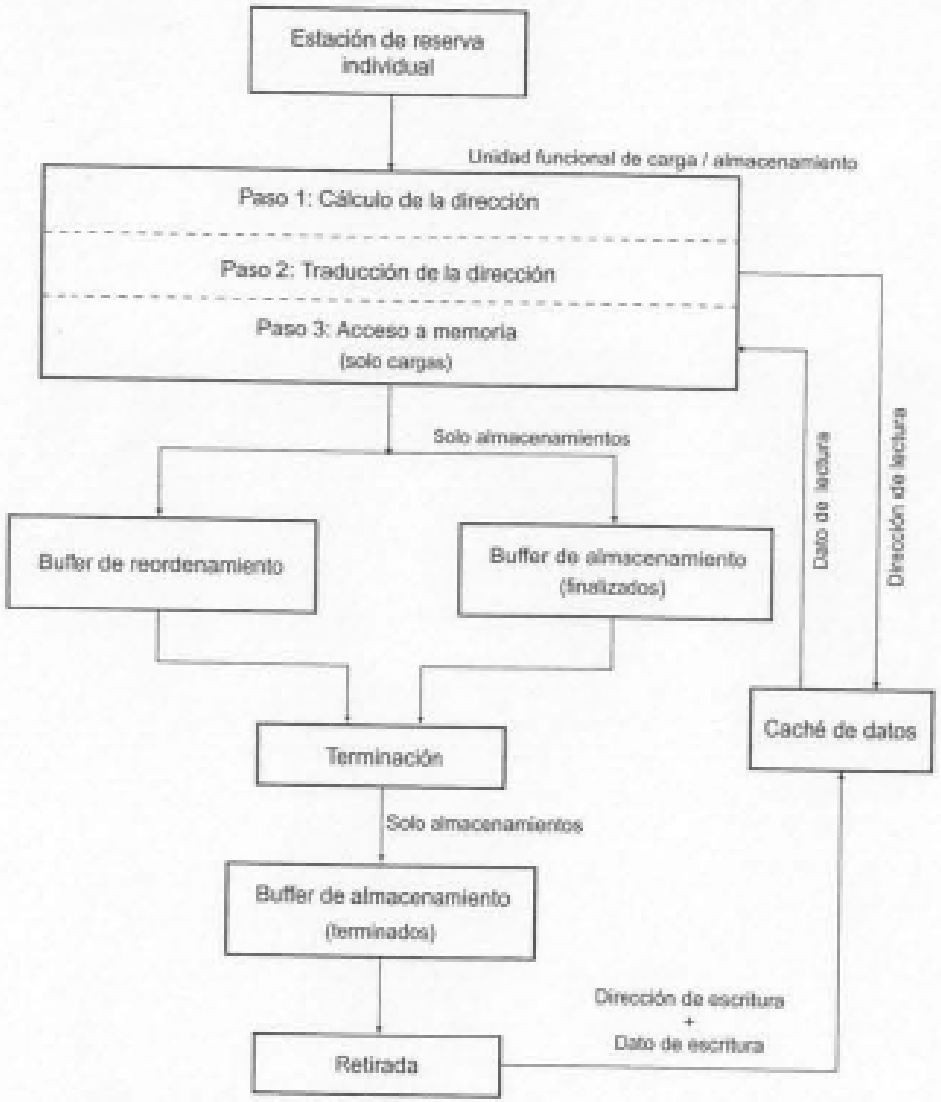


Figura 2.45: Organización de la etapa de ejecución y del *back-end* de la segmentación para el procesamiento diferenciado de las instrucciones de carga y almacenamiento.

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

- Terminación adelantada de Instrucciones de carga
  - Instrucciones de carga que escriben en un registro que constituyen un punto de arranque critico para un conjunto de instrucciones aritmetico logicas que dependen de este operando
    - Mejoramos si permitimos que estas instrucciones de carga terminen antes
- Reenvío de instrucciones de almacenamiento y de carga
  - Reenvío de datos entre una instrucción de almacenamiento hacia otra de carga que tenga operandos distintos y fuentes comunes
    - El dato que requiere la carga está en un registro del procesador y no hay que ir a la memoria para obtenerlo

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

- Reenvío de datos entre una instrucción de almacenamiento hacia otra de carga que tenga operandos distintos y fuentes comunes
  - 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

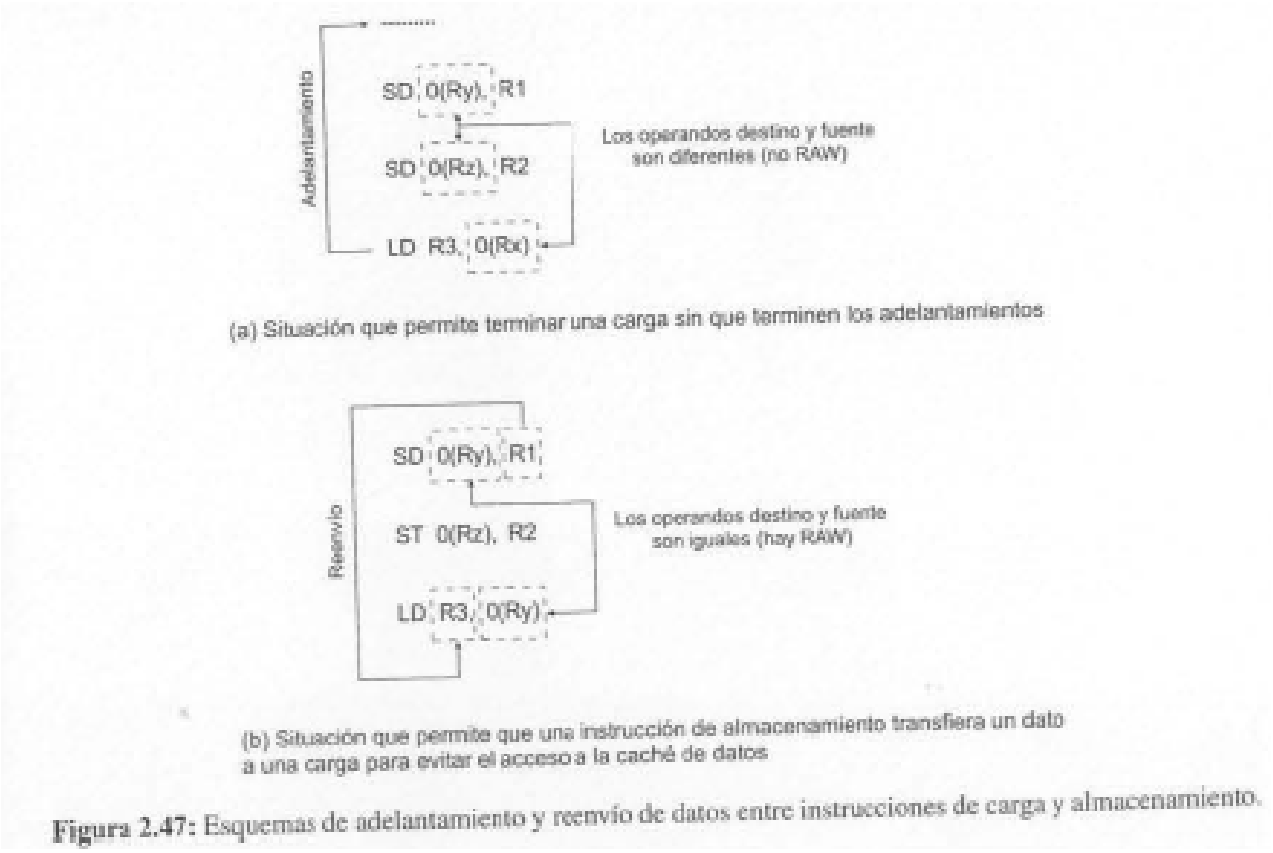


Figura 2.47: Esquemas de adelantamiento y reenvío de datos entre instrucciones de carga y almacenamiento.

Compara las direcciones de almacenamiento con las de carga
Si **hay** coincidencia la carga no necesita acceder a la cache
Si **no** hay coincidencia, tiene que acceder a la cache

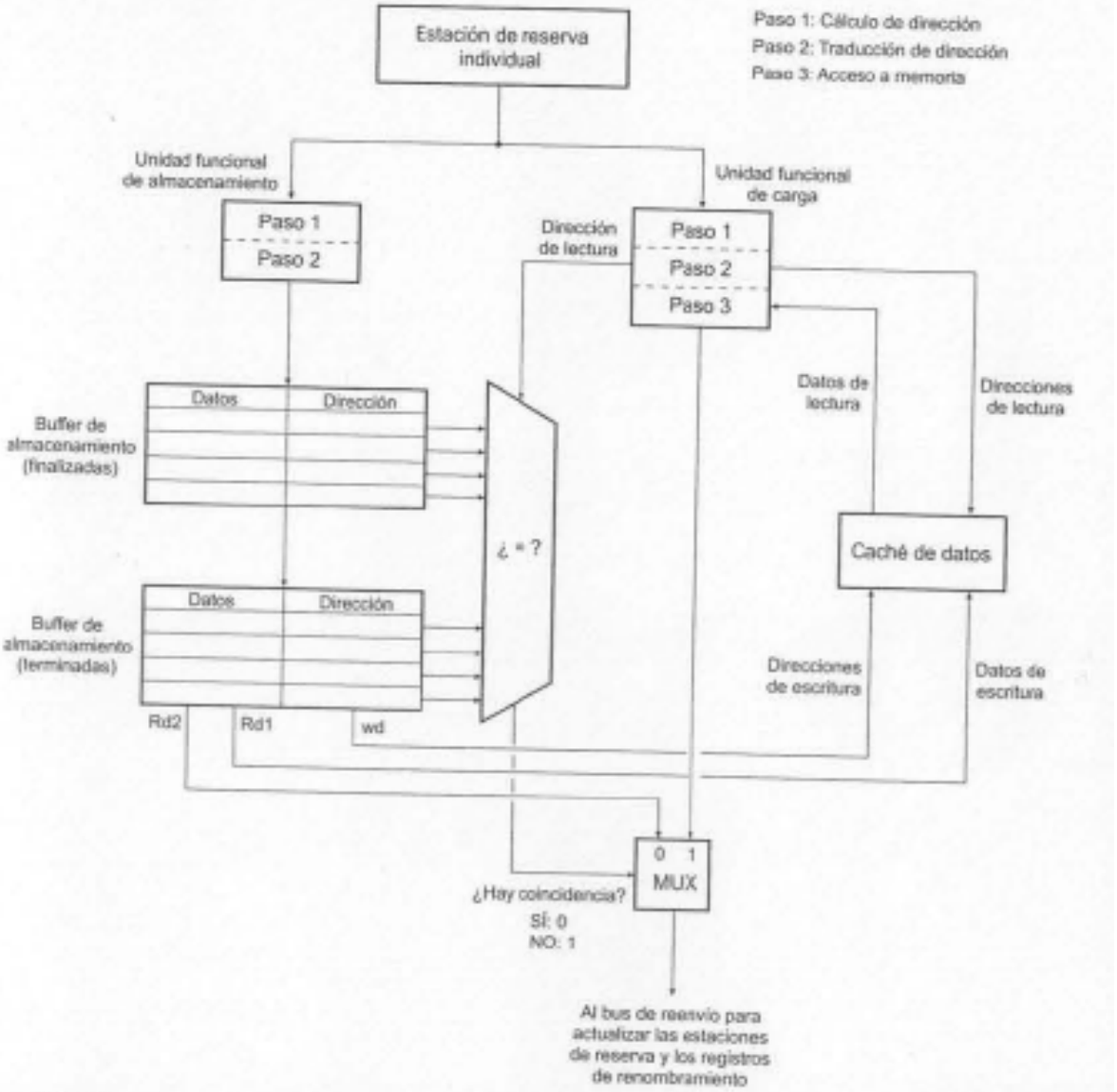


Figura 2.48: Esquema de reenvío de datos entre cargas y almacenamientos en el que se contempla la emisión ordenada de las cargas y los almacenamientos.

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.
- 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

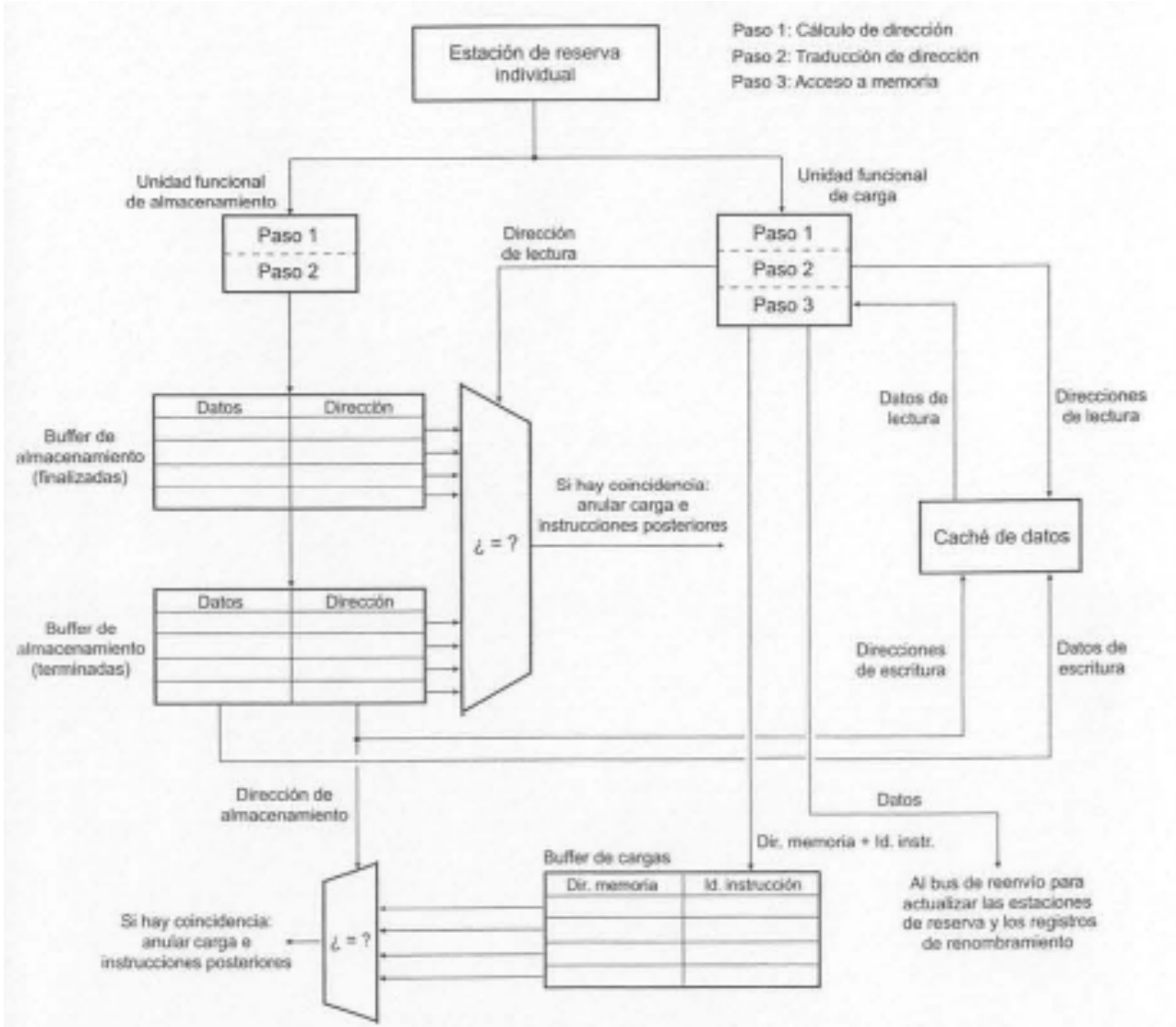


Figura 2.49: Esquema para realizar el adelantamiento de las cargas permitiendo la emisión fuera de orden de las cargas y los almacenamiento.

2.11. Tratamiento de interrupciones

- 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* producidas por eventos externos a la ejecución del programa (operación de E/S por un dispositivo externo, cambio de contexto, malfunción hardware, fallos de potencia, etc.) 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. Una vez que el sistema operativo haya realizado el tratamiento de la interrupción, se podrá recuperar el estado del procesador y continuar con la ejecución del programa a partir de la última instrucción que se terminó.

- *Excepciones, interrupciones de programa o traps* causadas, en general, en la etapa IF o por las instrucciones del programa en ejecución. Los ejemplos más conocidos son los errores numéricos (divisiones por cero, desbordamientos aritméticos), aunque no hay que olvidar los fallos de página en el sistema de memoria virtual tanto para instrucciones como datos. El tratamiento de las excepciones es similar al de las interrupciones externas aunque en algunos tipos de excepción no se considera la reanudación del programa como, por ejemplo, las divisiones por cero o algunos desbordamientos. Incluso en estas situaciones, en las que el programa no se reanuda, el estado del procesador se puede almacenar para tareas de depuración (*debugging*) que permitan conocer y analizar las causas del problema.

Técnicas de tratamiento de interrupciones(procesadoresque permitenejecución fuera de orden)

- Ignorar el problema y no garantizar la precisión de excepción. Muy utilizada en los años 60 y principios de los 70, y todavía empleada por algunos supercomputadores que no permiten ciertas interrupciones o si lo son se tratan vía hardware sin detener la ejecución del proceso.

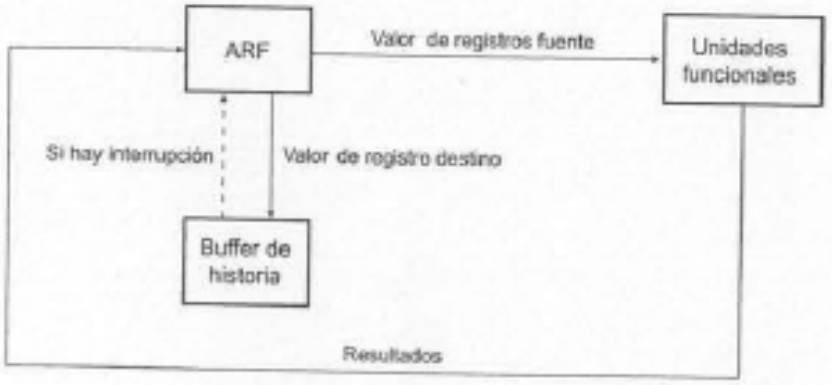
- Permitir que las interrupciones sean "algo" imprecisas pero guardando suficiente información para que la rutina de tratamiento software pueda recrear una secuencia precisa para recuperarse de la interrupción.

- Permitir que una instrucción sea emitida solo cuando se está seguro de que las instrucciones que la preceden terminarán sin causar ninguna interrupción.

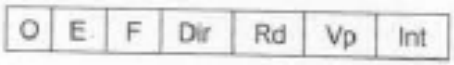
- Mantener los resultados de las operaciones en un buffer de almacenamiento temporal hasta que todas las operaciones previas hayan terminado correctamente y se puedan actualizar los registros arquitectónicos sin ningún tipo de riesgo. Entre estas técnicas se encuentran el buffer de historia (*history buffer*) y el fichero de futuro (*future file*). Esta forma de tratar las excepciones ha ido adaptándose a la evolución que han experimentado los procesadores superescalares hasta llegar al mecanismo actual basado en la combinación del renombramiento de registros, las estaciones de reserva y el buffer de terminación. A continuación, se describen con más detalle las técnicas pertenecientes a esta categoría.

Procesamiento de interrupciones basados en buffer de historia

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



(a) Procesamiento de interrupciones precisas basado en el buffer de historia



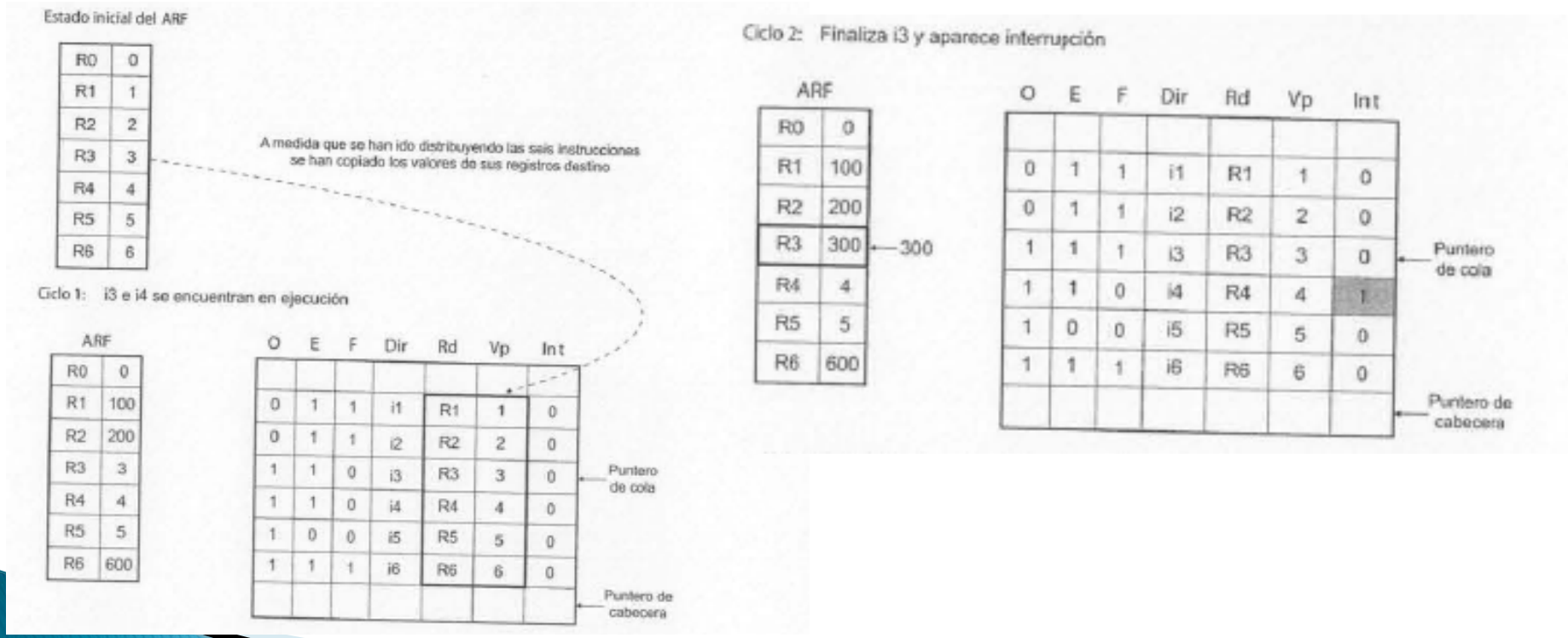
O: Ocupada
E: Emitida
F: Finalizada
Dir: Dirección de la instrucción
Rd: Registro de destino
Vp: Valor previo
Int: Interrupción

(b) Entrada del buffer de historia

Ejemplo de buffer de historia

```
i1: MULTI R1,R1,#100
i2: MULTI R2,R2,#100
i3: MULTI R3,R3,#100
i4: MULTI R4,R4,#100
i5: MULTI R5,R5,#100
i6: MULTI R6,R6,#100
```

La interrupción aparece en la instrucción i4, el ejemplo i1 , i2 terminadas, i3 , i4 en ejecución, i5 en espera de ser emitida e i6 finalizada



Ciclo 3: i3 termina e i4 finaliza

ARF

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

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	← Puntero de cola
1	0	0	i5	R5	5	0	
1	1	1	i6	R6	6	0	
							← Puntero de cabecera

Ciclo 4: i4 termina y se trata la interrupción

Ciclo 4: i4 termina y se trata la interrupción

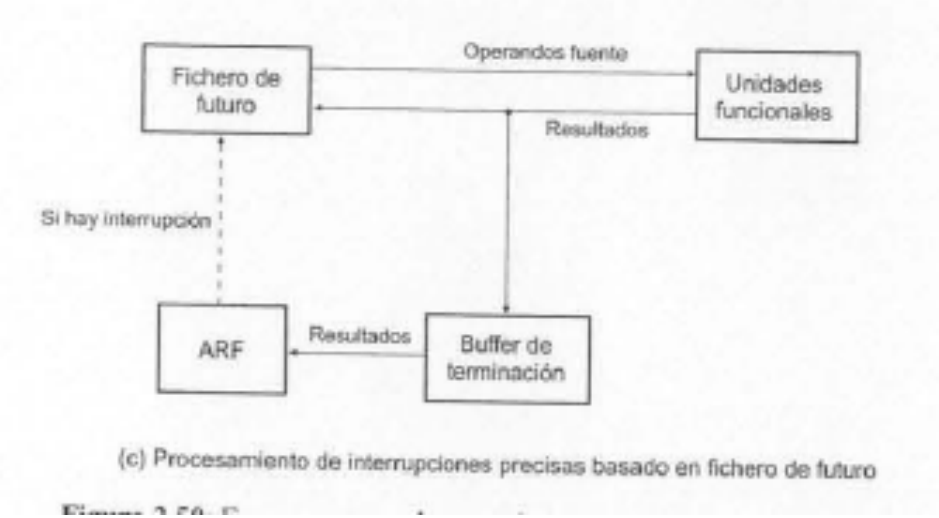
ARF

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

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

Ejemplo de fichero de futuro

Ciclo 1: i3 e i4 se encuentran en ejecución

FRF

R0	0
R1	100
R2	200
R3	3
R4	4
R5	5
R6	600

O	E	F	Dir	Rd	Va	V	Int
0	1	1	i1	R1	100	1	0
0	1	1	i2	R2	200	1	0
1	1	0	i3	R3	3	1	0
1	1	0	i4	R4	4	1	0
1	0	0	i5	R5	5	1	0
1	1	1	i6	R6	600	1	0

← Puntero de cola

← Puntero de cabecera

ARF

R0	0
R1	100
R2	200
R3	3
R4	4
R5	5
R6	6

← Se copian al ARF solo cuando se terminan

Ciclo 2: Finaliza i3 y aparece interrupción

FRF

R0	0
R1	100
R2	200
R3	300
R4	4
R5	5
R6	600

300

O	E	F	Dir	Rd	Va	V	Int
0	1	1	i1	R1	100	1	0
0	1	1	i2	R2	200	1	0
1	1	1	i3	R3	300	1	0
1	1	0	i4	R4	4	1	1
1	0	0	i5	R5	5	1	0
1	1	1	i6	R6	600	1	0

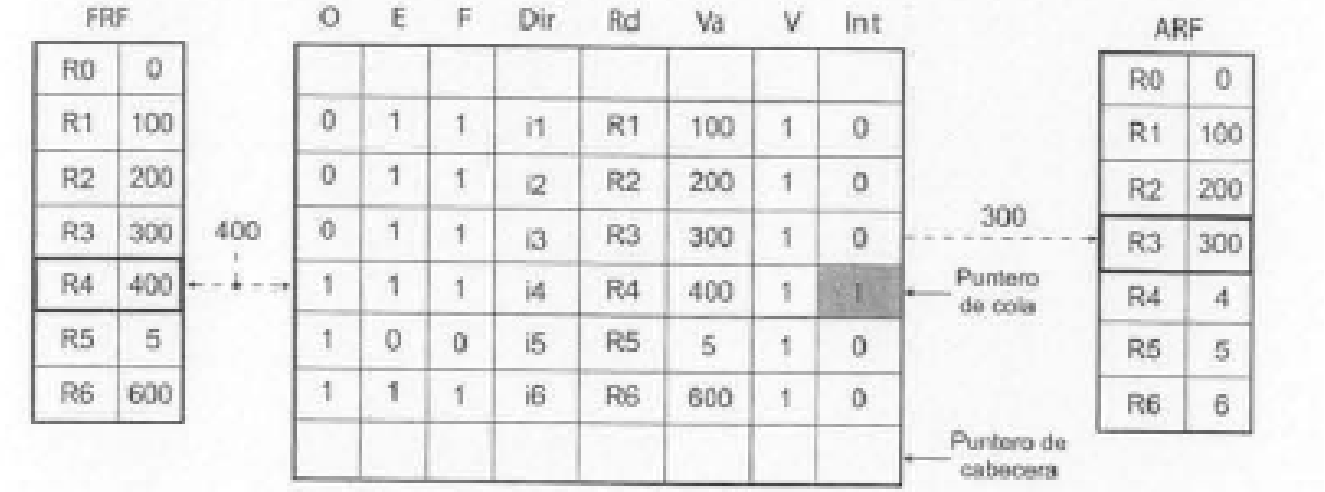
← Puntero de cola

← Puntero de cabecera

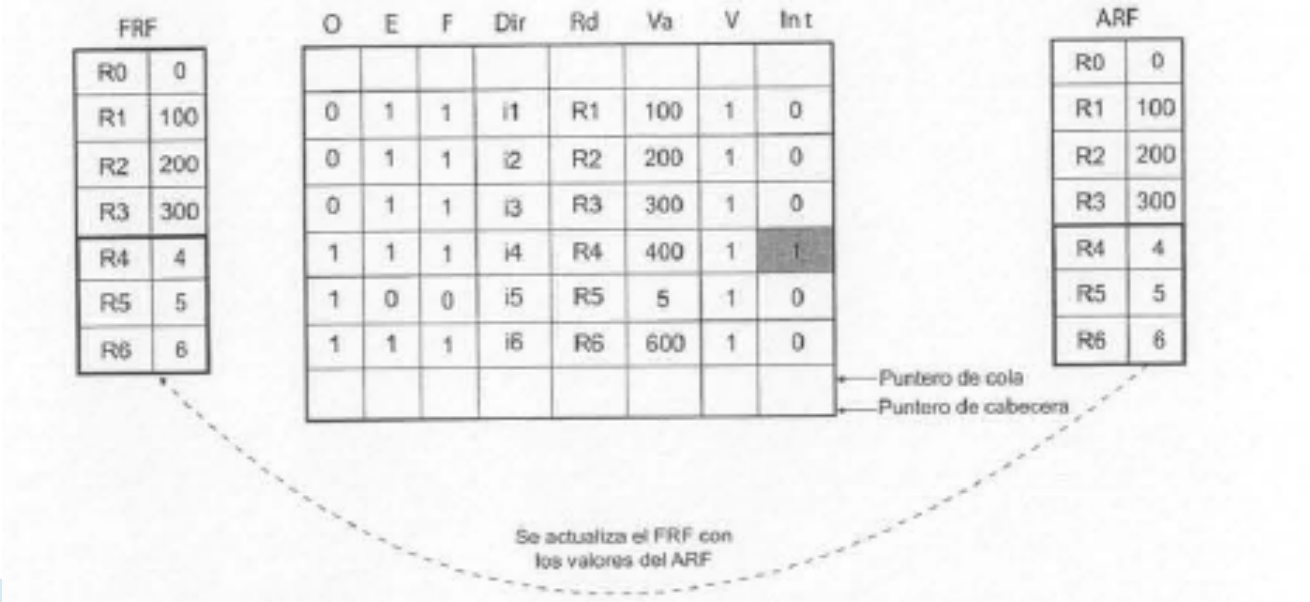
ARF

R0	0
R1	100
R2	200
R3	3
R4	4
R5	5
R6	6

Ciclo 3: i3 termina e i4 finaliza



Ciclo 4: i4 termina y se trata la interrupción



2.11.1. Excepciones precisas con buffer de reordenamiento

- Una columna de interrupción

```
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
```

O	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

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

2.12. Limitaciones de los procesadores superescalares

- Complejidad y costes de la fase de distribución de las instrucciones y el analisis de la dependencia
- El grado de paralelismo intrínseco existente en el flujo de instrucciones de un programa