# UNIVERSIDAD DE ZARAGOZA

**Tesis Doctoral**

Zaragoza, Junio de 2003

# Técnicas *Hardware* para Optimizar el Uso de los Registros en Procesadores Superescalares


Teresa Monreal Arnal

Directores:

Víctor Viñals Yúfera — Universidad de Zaragoza

Mateo Valero Cortés — Universitat Politècnica de Catalunya

Antonio González Colás — Universitat Politècnica de Catalunya



**DEPARTAMENTO**
**DE INFORMÁTICA E INGENIERÍA DE SISTEMAS**

# Técnicas *Hardware* para Optimizar el Uso de los Registros en Procesadores Superescalares


Memoria presentada por

**Teresa Monreal Arnal**


para obtener el título de Doctora Ingeniera en Informática


Departamento de Informática e Ingeniería de Sistemas


Zaragoza, Junio de 2003

Dirigida por:


Mateo Valero Cortés

Víctor Viñals Yúfera

Antonio González Colás

**UNIVERSIDAD DE ZARAGOZA**
**http://www.unizar.es**

**UNIVERSITAT POLITÈCNICA DE CATALUNYA**

A mis padres

## Agradecimientos

Debo empezar agradeciendo al Profesor José Luis Briz el interés que puso en mí desde el principio. Mi incorporación a la Universidad de Zaragoza no hubiese sido posible sin su empeño para que yo formase parte de la misma.

Los siguientes a los que debo agradecimientos son mis tres directores. Tengo que decir que he aprendido y disfrutado trabajando con los tres. Aprendí con Mateo cuando en mis estancias en el DAC me pasaba artículos y más artículos para leer y que luego discutíamos. He disfrutado y disfruto cada vez que lo escucho en cursos y charlas. Nunca dejará de asombrarme la facilidad innata con la que se comunica con la gente. Eso hizo que me resultase fácil integrarme con toda la gente del DAC. Mateo y yo somos familia y como tal es alguien con el que siempre he contado, puedo y sé que podré contar. Con Antonio he aprendido el interés que siempre pone en las cosas que hace. He disfrutado con ello y sobre todo con las discusiones que alguna que otra vez los dos hemos mantenido y que ahora entiendo que eran básicas para mi aprendizaje. Con Víctor he aprendido a entusiasmarme con el trabajo, a elaborarlo hasta el más mínimo detalle para acabar convirtiendo la solución más compleja en algo fácil de trasmitir para ser entendido. No me resultó costoso leer y discutir artículos, ni siquiera discutir decisiones, lo más costoso para mi siempre ha sido el llegar al detalle de las cosas. Víctor y yo nos hemos tenido que convertir en amigos y esto me ayuda a que trabajar con él vaya dejando de ser algo duro para mi y se esté convirtiendo en algo con lo que también disfruto.

A Mateo debo agradecerle además el que me presentase a Paqui Quintana. A Paqui debo darle las gracias por haberme aceptado desde el principio y aún sin conocerme, permitiéndome compartir piso con ella. Aunque nuestra relación comenzó de alguna forma siendo impuesta para ella, se convirtió en una bonita amistad.

Igualmente va mi agradecimiento a todo el personal del DAC y del CEPBA de la UPC. Desde el principio me he sentido integrada con ellos y siempre me ha resultado y me resulta fácil trabajar en su entorno.

Agradecimientos para todos los componentes del grupo de Arquitectura y Tecnología de Computadores del DIIS. No sólo somos un grupo (gaZ) sino que además hemos conseguido llegar a ser amigos. En especial, les doy las gracias a Pablo Ibañez y a Chus Alastruey. Sin ellos ni el formato ni la redacción de esta Tesis hubiesen quedado tan depurados como creo que están. Gracias a los dos. También quiero darles ánimos a Ana, Natalia, Chus, Kike y Luisma para que continúen con sus trabajos de investigación. Esto de la Tesis siempre se acaba un día u otro.

He dedicado esta Tesis a mis padres porque ellos son los que siempre están a mi lado apoyando todas las decisiones que tomo. Incluso mucho antes de que yo tenga las cosas claras, ellos ya tienen un buen consejo para darme. Lo que tengo que agradecerles a ellos es dificil de transmitir en unas cuantas líneas. Ellos hicieron que yo naciera, que tuviese un hermano que siempre ha cuidado de mi y que me criase en un pueblo como Alfamén. De ellos, de mi hermano, de toda mi familia, amigos y de la gente de mi pueblo he aprendido cosas que forman parte de mi vida y de las que siempre voy a estar agradecida. De mi familia he heredado además esa mezcla de inocencia, respeto, cariño y picardía que existe en todos ellos y de lo que me siento orgullosa.

Agradezco a mis cuatro chicas: Alida, Ariadna, Marta y Sonia las horas de distracción que me han permitido tener con ellas. Esos ratos han sido fundamentales para mi porque me han dejado romper la rutina semanal de trabajo. Para ellas pido que sigan manteniendo durante todas sus vidas parte de esa inocencia que ahora tienen y que este mundo no se complique demasiado para que les permita disfrutar siempre de libertad, al menos de la misma cantidad de la que yo he podido disfrutar hasta ahora.

J.M., no voy a acabar estos agradecimientos sin aprovechar para dejar constancia por escrito de que tú eres lo mejor que me ha pasado en la vida y por lo tanto esta Tesis también va por ti.

;-)

# INDICE

# Prólogo

## Resumen de la Tesis

*El Banco de Registros es considerado uno de los componentes más críticos que poseen los procesadores actuales. La tendencia actual de ejecutar más instrucciones en paralelo, hace que los procesadores se diseñen con un* hardware *que cada vez es más complejo. Para soportar la ejecución paralela de este número creciente de instrucciones, deben incorporarse a la microarquitectura elementos más complejos en tamaño y/o control. En particular, va a ser crítico el tiempo de acceso y el consumo de energía de un Banco de Registros que posea un número elevado de registros y de puertos. Esta Tesis trata de la optimización del uso que se hace de los registros con el objetivo de reducir la complejidad del Banco de Registros. Una mejor utilización de los registros, se traduce en dos direcciones: en un aumento del rendimiento (IPC) manteniendo el mismo tamaño del Banco de Registros o en una reducción del tamaño del Banco de Registros y por lo tanto de su tiempo de acceso, sin pérdida de rendimiento. Para ello, nos centramos primero en el estudio de la asignación y la liberación de los registros en un procesador convencional, detectando las ineficiencias de esta gestión convencional. En concreto, mostramos cómo los registros se asignan a las instrucciones mucho más pronto y se liberan mucho más tarde de lo que realmente es necesario. A continuación, presentamos un conjunto de técnicas* hardware *para mejorar la eficiencia en la asignación de los registros y que se basan en retrasar esa asignación hasta el final de la etapa de ejecución de las instrucciones. Después presentamos dos técnicas que permiten mejorar la eficiencia en la liberación de los registros y que se basan en adelantar esa liberación a la etapa de* commit *de las instrucciones que utilizan los registros por última vez. Finalmente, para la parte de la asignación y para la de liberación seleccionamos las dos mejores técnicas y las componemos juntas en una nueva. Como resultado se ha obtenido una nueva técnica que supera a cualquiera de las dos actuando por separado. Por ejemplo, consigue aumentar el rendimiento un 28% manteniendo el mismo Banco de registros o reducir en un 30% el tamaño del Banco de registros con una mínima pérdida de rendimiento.*

## Financiación

El trabajo desarrollado en esta Tesis se ha visto apoyado con las financiaciones y recursos computacionales que se listan a continuación.

Dos estancias de investigación realizadas en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad Politécnica de Cataluña (DAC-UPC). Las dos fueron financiadas a través del Programa Europa CAI CONSI + D de Estancias de Investigación.

Dos proyectos de I + D financiados en convocatorias públicas nacionales del Ministerio de Ciencia y Tecnología, Plan Nacional de I + D + I, Programa PRONTIC. Son los siguientes dos proyectos coordinados con el DAC-UPC:

> *Computación de Altas Prestaciones II. Ocultación de Latencia*. CICYT-TIC-1998-0511-C02-02.
>
> *Computación de Altas Prestaciones III. Jerarquía de Memoria de Altas Prestaciones.* CICYT-TIC-2001-0995-C02-02.

Toda la carga de experimentación que se ha realizado en esta Tesis ha sido soportada por los recursos computacionales del Centro Europeo de Paralelismo de Barcelona -CEPBA-.

## Estructura y Organización de esta Memoria

Esta Tesis se ha estructurado en seis Capítulos, dos Apéndices y el listado completo de las Referencias.

La primera parte de la Introducción tiene por objeto poner en contexto a un lector que no esté familiarizado con el tema. Presenta los conceptos básicos relacionados con el tipo de procesador que ha sido utilizado como base. A continuación presenta un conjunto de trabajos relacionados con el estudio y la optimización del banco de registros. Al final se listan las contribuciones de esta Tesis así como las publicaciones generadas.

El Capítulo 2 presenta en detalle el procesador de base que se va a utilizar a lo largo de esta Tesis. Se detallan aquí todas las estructuras que en este procesador (denominado *convencional*) se utilizan para implementar la técnica del *renombre de registros*. En este capítulo aparecen las definiciones relacionadas con el dimensionado del banco de registros y con los distintos estados en los que se pueden encontrar los registros en un momento dado. Finalmente, se presenta el concepto de *utilización de los registros* midiendo cuál es su valor para un procesador convencional. Este capítulo acaba evidenciando experimentalmente cómo un renombre de registros convencional posee ineficiencias claras tanto en la parte de asignación de los registros como en la parte de la liberación de los mismos.

En el Capítulo 3 se presentan las distintas estrategias que bajo el concepto de *registros virtuales-físicos* aumentan la eficiencia del renombre en la parte de la asignación de los registros. Se proponen implementaciones detalladas y realizables para todas las estrategias, demostrándose que el *hardware* propuesto no aumenta el tiempo de ciclo y que su rendimiento energético es razona-

ble. Las distintas estrategias se van comparando entre ellas y entre el mecanismo convencional, analizando las mejoras de rendimiento y sugiriendo puntos de diseño atractivos.

En el Capítulo 4 se presentan las distintas estrategias que bajo el concepto de *liberación anticipada de los registros* aumentan la eficiencia del renombre en la parte de la liberación de los registros. Se proponen dos implementaciones de complejidad creciente, ambas realizables y se demuestra que el *hardware* propuesto no aumenta el tiempo de ciclo y que posee un rendimiento energético razonable. Las dos estrategias se comparan entre sí y entre el mecanismo convencional, analizando las mejoras obtenidas de rendimiento y sugiriendo puntos de diseño atractivos.

El Capítulo 5 muestra una última propuesta que compone a las dos más agresivas vistas en los capítulos previos. Esta propuesta se denomina vp-LAER[1] y consigue tanto aumentar la eficiencia del renombre en la parte de la asignación de los registros como la eficiencia en la parte de la liberación de los registros. Es la estrategia más compleja de todas pero también es la que más beneficios permite obtener. De nuevo, se propone una implementación detallada y realizable de esta estrategia, se demuestra que el *hardware* propuesto no aumenta el tiempo de ciclo y que su rendimiento energético es favorable. La estrategia vp-LAER se compara con las presentadas en los capítulos anteriores y con el mecanismo convencional, analizando las mejoras de rendimiento obtenidas tanto en IPC como en IPS y sugiriendo en todos los casos puntos de diseño atractivos.

A continuación se presentan las Conclusiones y Líneas Abiertas que han generado esta Tesis.

Al final se han incluido todas las Referencias así como el Apéndice A y Apéndice B que completan los datos experimentales obtenidos en los Capítulos 2, 3, 4 y 5.

1. vp-LAER: del inglés *virtual-physical with Late Allocation and Early Release.*

CAPÍTULO 1

# Introducción


## Resumen

*El objetivo de esta Introducción es poner en contexto al lector de esta Tesis. Repasa los conceptos básicos asociados al tipo de procesador y a la estrategia del renombre de registros utilizados como base. Se presentan brevemente los conceptos de: procesado fuera de orden, interrupciones precisas, ejecución especulada de instrucciones, renombre de registros.*

*Un lector conocedor del tema puede perfectamente saltarse esta primera parte de la Introducción y pasar directamente al apartado que se denomina Estado del Arte. En este apartado se comentan trabajos relacionados con el estudio, diseño u optimización del banco de registros en procesadores con ejecución fuera de orden. Estos trabajos, o bien forman la base sobre la que construimos o bien plantean metas similares o complementarias a las nuestras. Finalmente, resumimos las Contribuciones de esta Tesis y las publicaciones generadas.*

Una de las últimas innovaciones en la arquitectura de los microprocesadores actuales ha sido la del procesado *fuera de orden*. El problema general que resuelven estos procesadores consiste en ejecutar un programa de forma no secuencial o paralela pero manteniendo la apariencia externa de una ejecución secuencial.

Si el procesador fuera de orden es además *superescalar*, busca y decodifica varias instrucciones a la vez. En adelante denominaremos *superescalar* a un procesador que es *superescalar y fuera de orden*. Con el objetivo de disponer de un flujo de instrucciones continuo, durante la etapa de búsqueda *predice* cual será el destino de las instrucciones de salto condicional. Este flujo de instrucciones se analiza luego para encontrar las *dependencias de datos* y las instrucciones se distribuyen hacia las distintas unidades funcionales. Una planificación dinámica de las instrucciones iniciará su ejecución en paralelo sólo teniendo en cuenta la disponibilidad de datos y recursos y no el orden original del programa.

El modelo de ejecución secuencial de un programa tiene asociado de forma natural el concepto de *estado preciso o arquitectural* de la máquina. Generalmente ese estado está formado por el contador del programa (*PC*), el contenido de los registros lógicos de la arquitectura y la memoria. A veces, la ejecución de un programa necesita ser interrumpida y luego restablecida, por ejemplo cuando ocurre un fallo de página o una interrupción *hardware*. Guardar el estado arquitectural de la máquina en el momento de la interrupción servirá para restablecer el programa tras el servicio de la interrupción. El restablecimiento puede consistir en reanudar el procesado de instrucciones desde la instrucción que interrumpió. Para mantener un modelo de ejecución secuencial (soportar *interrupciones precisas*), una vez que las instrucciones van acabando, sus resultados se reordenan de nuevo actualizando el estado del proceso según el orden original del programa. De esta forma, si ocurre alguna condición de interrupción podrá recuperarse el estado preciso de la máquina.

Un procesador superescalar altera el orden secuencial original y convierte el programa en una versión paralela de mayor rendimiento. Como las instrucciones del programa son las que se ejecutan en paralelo, se dice que el procesador superescalar está extrayendo del programa su paralelismo a nivel de instrucción (*ILP*).

Una de las técnicas principales para aumentar el ILP de un programa consiste en la eliminación de las *dependencias de control* mediante la predicción del camino tomado tras un salto (dirección destino del salto, comportamiento del salto condicional, etc.). A partir de este momento la ejecución de las instrucciones entrantes es *especulada*. Si más adelante se comprueba que la predicción del salto fue correcta, las instrucciones dejan de ser especuladas y su efecto sobre el estado del proceso debe ser el mismo que el de cualquier otra instrucción. Si por el contrario, se comprueba que la predicción del salto fue incorrecta, querrá decir que toda la ejecución especulada también fue incorrecta. Por lo tanto, el efecto producido por estas instrucciones debe ser neutralizado para evitar modificaciones incorrectas al estado real del proceso.

Las instrucciones se almacenan en la *ventana de lanzamiento* donde esperan a que se resuelvan sus dependencias de datos para poder iniciar la ejecución. Esas dependencias de datos ocurren porque distintas instrucciones necesitan leer o escribir en los mismos almacenes (registros o memoria). Se dice que existe un *riesgo* cuando dos instrucciones referencian al mismo almacén (riesgos *RAW*, *WAR* o *WAW*). A menos que se evite, existe la posibilidad de acceder al contenido

del almacén en un orden incorrecto. Existen dos tipos de dependencias de datos según la limitación en rendimiento del riesgo que suponen:

- Las *dependencias de datos verdaderas*. Provocan un riesgo RAW (es necesario leer *después* de escribir).

- Las *dependencias de datos falsas*. Provocan riesgos de tipo WAR (escribir *después* de leer) o riesgos de tipo WAW (escribir *después* de escribir).

Otra técnica para aumentar el ILP de un programa consiste en neutralizar las dependencias de datos falsas. Los procesadores superescalares actuales utilizan para ello el *renombre de operandos*. El renombre de operandos genera dinámicamente un nuevo programa sin dependencias de datos falsas. Esta técnica se aplica con facilidad a los operandos registro (*renombre de registros*) y de forma algo más limitada a los operandos de memoria (a veces bajo el término *desambiguación* de memoria o *store-load forwarding*). Al eliminar parte de las restricciones de precedencia entre las instrucciones, el renombre consigue aumentar el número de instrucciones de la ventana de lanzamiento que pueden ejecutarse en paralelo. En el siguiente apartado se explica más en detalle en que consiste el renombre de registros, objetivo de esta Tesis.

Después de eliminar las dependencias de control y las dependencias falsas, es el *hardware* de planificación de la ejecución en paralelo el que se encarga de lanzar a ejecutar instrucciones. Las únicas restricciones que el planificador tiene en cuenta ahora son las dependencias de datos verdaderas y la disponibilidad de unidades funcionales y caminos de datos. Por tanto las instrucciones pueden completar su ejecución en un orden distinto del que tenía el modelo de ejecución secuencial, es decir, pueden finalizar *fuera de orden*.

La ejecución fuera de orden junto con la especulación de instrucciones provoca que algunas instrucciones que ya han completado su ejecución no deberían ni siquiera haberse comenzado a ejecutar. En realidad debería haberse seguido el modelo que dicta la ejecución secuencial, por esta razón el *estado arquitectural* de la máquina no se puede actualizar de forma inmediata cuando las instrucciones completan su ejecución. Por ello, los resultados que genera toda instrucción se consideran temporales (o especulativos). Al final, cuando se comprueba que el modelo de ejecución secuencial sí que hubiera ejecutado la instrucción, los resultados temporales se convierten en permanentes actualizando el estado arquitectural. Este proceso de actualización se realiza en la *etapa de commit* de la instrucción y equivale a la recuperación del orden secuencial original del programa. La *estructura de reordenación* es la que guarda ese orden secuencial original de las instrucciones del programa (*Reorder* o *History Buffer* [SP85][SP88]).

La implementación de cualquiera de las anteriores técnicas implica un diseño cada vez más complejo del procesador. Iniciar varias instrucciones durante el mismo ciclo de reloj permite diseñar procesadores más rápidos aunque también de implementación más compleja [SS95]. Además, todo procesador que se diseñe con el objetivo de obtener una ejecución de alto rendimiento va a necesitar una serie de estructuras *hardware* cuya complejidad irá a su vez aumentando en la misma medida en que se desee soportar mayor rendimiento [PPE+97].

.

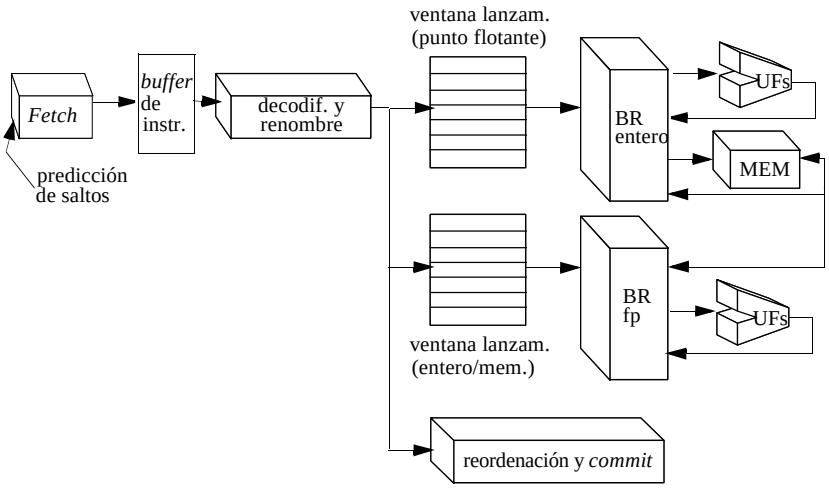


**Figura 1.1** Organización típica de un procesador superescalar.

La Figura 1.1 muestra las partes más importantes de la organización *hardware* de un procesador superescalar convencional. Estas partes de la arquitectura se corresponden con las etapas definidas en el procesador: búsqueda de instrucciones y predicción de saltos (*Fetch*), decodificación y análisis de dependencias, lanzamiento y ejecución de las instrucciones, análisis y ejecución de las operaciones de memoria y reordenación y *commit* de las instrucciones.

## 1.1 Renombre de Registros

En la fase de decodificación, las instrucciones se extraen del *buffer de instrucciones* para examinar las dependencias por registros. Esta fase incluye el renombre de registros, es decir la eliminación de las dependencias de datos falsas que se producen debido a la reutilización de los registros.

El *renombre de registros* es la técnica que se utiliza para eliminar los riesgos WAW y WAR y aumentar el paralelismo en la ejecución de los programas [SFK97]. En el código estático de un programa, las instrucciones almacenan sus resultados en los lugares *lógicos* de la arquitectura (registro destino o memoria). Pero durante la ejecución dinámica del programa se utilizan otros elementos de almacenamiento que ya ocupan lugares *físicos* y que pueden ser distintos de los lógicos. Las instrucciones utilizan esos lugares físicos para transmitirse sus resultados. Por tanto, es posible la existencia simultánea de varias versiones especulativas del mismo registro lógico.

La técnica del renombre *asigna* a todo registro lógico destino, un elemento de almacenamiento físico para guardar de forma temporal su valor (versión, instancia). A dicho elemento de almacenamiento se le denomina *registro de renombre*. Por ejemplo, para el siguiente fragmento de código:

```
ins1:       .. = r3 + ..

ins2:       r3 = .. * ..
```

el registro destino de la instrucción 2 (`r3`) se renombra al registro `p3` y la instrucción queda así:

```
ins2:       p3 = .. * ..
```

el resultado se escribirá en `p3` en lugar de en `r3`. Esta acción resuelve la antidependencia (riesgo WAR) que existía entre las instrucciones 1 y 2. Las posteriores referencias al registro lógico `r3` se redireccionarán al registro físico `p3` durante el tiempo que éste sea válido.

En los años 60, fueron el CDC 6600 [Tho64] y el IBM 360/91 [AST67] los primeros que incluyeron en su implementación técnicas de procesado paralelo de instrucciones. De ellos, el IBM 360/91 es el que incluye el concepto del renombre de registros en la forma en que lo describe Tomasulo en [Tom67] y que coincide con la que conocemos ahora. El diseño del renombre desarrollado en el 360/91 solo se incluyó en su unidad de punto flotante, pero era de una complejidad tal que hizo que no tuviese éxito en el mercado.

En los años 70, autores como Tjaden y Flynn [TF70] y Riseman y Foster [RF72] demuestran que para extraer un paralelismo razonable entre las intrucciones, es necesario invertir grandes cantidades de *hardware*. Más tarde Keller [Kel75] vuelve a introducir el renombre describiendo como implementarlo y extendiéndolo a todo tipo de instrucciones con registro destino, pero esta técnica no se utiliza hasta comienzos de los 90. Los primeros procesadores superescalares como HP-PA 7100 [DWY+92], Sun SuperSparc [BK92], DEC Alpha 21064 [McL93], MIPS R8K [Yan94] e Intel Pentium [AA93], inician la ejecución de instrucciones en orden y no utilizan renombre. Apareció luego de forma restringida (aplicado solo a determinados tipos de intrucciones) en los Power1 [Gro90], Power2 [BDH+94], PowerPC 601 [BAM+93] y en NextGen Nx586 [Gwe94a]. A comienzos de 1992 aparece tal y como lo conocemos ahora en los últimos modelos de IBM ES/9000 [Lip92] y luego en el PowerPC 603 [BAH+94]. En la actualidad, el renombre de registros se considera una característica estándar que aparece en muchos procesadores superescalares a excepción de la línea Sun UltraSparc [Gwe94b] y los procesadores de Alpha que preceden al DEC Alpha 21264 [Gwe96].

En este punto, es obligado mencionar a Itanium, la nueva línea de procesadores EPIC (*Explicitly Parallel Instruction Computing* [SR00]) desarrollada conjuntamente por Intel y HP, que al ser una evolución del concepto VLIW se caracteriza (por el momento, Itanium I [Qua00] e Itanium II [NCF+02]) por el lanzamiento en orden de múltiples instrucciones cuidadosamente planificadas por el compilador. Está todavía por ver si esta filosofía desbancará la ejecución fuera de orden y el renombre, o bien convivirán en el futuro como alternativas en competencia.

Una característica de las técnicas de renombre es el elemento de almacenamiento físico utilizado para almacenar las versiones consolidadas y especulativas [Sim00]. Las grandes opciones de diseño se diferencian por utilizar almacenes mas o menos distribuidos. En la opción mas distribuida, el almacén de renombre se divide en *estaciones de reserva* asociadas a las unidades funcionales [AST67]. Las estaciones de reserva contienen los operandos fuente (consolidados o

especulativos). A su vez los valores consolidados suelen estar centralizados en un banco de registros que se actualiza al consolidar cada instrucción. Ejemplos de esta opción es la familia Pentium Pro de Intel [Gwe95] (Pentium II [PII97] y Pentium III [KP99]).

En la opción centralizada el almacén de renombre puede contener únicamente versiones especulativas, o puede incorporar también las versiones consolidadas. En el primer caso, las versiones especulativas pueden organizarse en orden programa en un ROB (*Reorder Buffer*) con capacidad para alimentar a las unidades funcionales; ejemplos de esta opción son la familia HP-8000 [Hun95] y la familia K6 de AMD [SS98]. De nuevo, los valores consolidados suelen estar centralizados en un banco de registros que se actualiza en *commit*.

En el segundo caso, los dos conjuntos de registros (lógicos y de renombre) se juntan en un *banco de registros mezclado*. Ejemplos de este caso son IBM ES/9000 [Lip92], IBM Power1 [Gro90] y Power2 [BDH+94], MIPS R10K/12K [Yea96][Gwe97], Compaq Alpha 21264 [Gwe96] e Intel Pentium 4 [HSU+01]. Está opción de diseño parece tomar fuerza en procesadores recientes, porque elimina la necesidad de efectuar copia al consolidar. En esta Tesis nos proponemos mejorar la gestión de registros para este tipo de procesadores. Extender nuestras ideas a otras opciones de diseño es posible, pero queda fuera del ámbito que nos hemos fijado.

Para encontrar una discusión más completa de otras opciones de diseño, consultar [SFK97], [Omo99] o [HP03].

Un procesador superescalar con banco de registros mezclado, *asigna* un registros físico a cada registro lógico destino en decodificación y guarda dicha asignación en una *tabla de mapeo* (*MT*) [Kel75]. Esta actividad a veces se aísla de la decodificación y se concentra en una etapa separada que se denomina Renombre. La tabla de mapeo posee tantas entradas como número de registros lógicos tiene la arquitectura y guarda el identificador del registro físico asignado a cada lógico. Esta opción está implementada en procesadores superescalares actuales como MIPS R10K/.../18K, DEC Alpha 21264 e Intel Pentium 4 (aqui se llama *Frontend Register Alias Table* -*RAT*- [HSU+01]). En los procesadores citados el proceso de renombre se extiende tanto al banco de registros entero como al de coma flotante, duplicándose todos los recursos necesarios (tabla de mapeo, control de asignación/liberación, etc.).

Para *liberar* los registros de renombre cuando ya no se necesitan es necesario conocer con seguridad que ninguna instrucción en el procesador va a utilizar más el valor almacenado en el registro. La especulación y la necesidad de cumplir un modelo de interrupciones preciso, hace que sea especialmente complejo el detectar de forma segura cuando se realiza el último uso de un registro. Los procesadores superescalares actuales han optado por implementar lo anterior de la forma más sencilla que consiste en esperar a que otra instrucción que escribe al mismo registro lógico alcance su etapa de *commit*. En este momento es seguro que ninguna instrucción va a utilizar más el valor del anterior registro de renombre. Las instrucciones durante su ejecución permanecen en una determinada *estructura de reordenación* en espera de recuperar el orden secuencial original del programa. Una implementación para liberar registros consiste en guardar junto con la instrucción el identificador de su anterior registro físico en esa estructura de reordenación (*History Buffer indirecto*). Esta opción se utiliza en los procesadores MIPS R10K/12K [Yea96]/[Gwe97], DEC Alpha 21264 [Gwe96] e Intel Pentium 4 [HSU+01].

En la *etapa de commit* se modifica el estado arquitectural de la máquina y también se recupera el orden secuencial de las instrucciones. El conjunto de acciones que se llevan a cabo dependen de la técnica que se utilice para recuperar el estado preciso de la máquina. Existen distintas técnicas que se emplean para resolver este problema [SP85][SP88]. Para bancos de registros mezclados, una posibilidad de diseño consiste en utilizar *tablas de mapeo en orden* que sirven para restaurar la asignación original de los registros en orden de programa. El procesador Intel Pentium 4 implementa esta idea con el nombre de *Retirement Register Alias Table -RRAT-* [HSU+01].

El procesador de base que vamos a utilizar a lo largo de toda esta Tesis implementa el renombre sobre un banco de registros mezclado donde conviven registros con valores especulados y consolidados. La asignación de registros se realiza en decodificación y se controla mediante tablas de mapeo. Para su liberación suponemos la existencia de un *History Buffer* indirecto.

En el Capítulo 2 se detallan las estructuras de renombre utilizadas en el procesador de base, similar a las utilizadas en los procesadores MIPS R10K/18K.

## 1.2 Estado del Arte

La tendencia actual de explotar cada vez más ILP de los programas, provoca un incremento de la complejidad en el diseño de las microarquitecturas fuera de orden. Un *hardware* más complejo tiende a aumentar el *camino crítico* del procesador y por lo tanto a limitar su velocidad. Son varias las partes de la microarquitectura cuya complejidad aumenta con el incremento del ILP [Wal91]. En esta Tesis, nos centramos en la complejidad que representa la incorporación de un banco de registros mezclado a la microarquitectura.

Un banco de registros mezclado exige disponer de un número considerable de registros que puedan ser accedidos en un sólo ciclo y desde un número también considerable de puertos [FJC96][PJS97]. Lo anterior no debería comprometer el aumento de frecuencia que está contemplando la ejecución fuera de orden y debería consumir una energía razonable [ZK01]. De hecho, los procesadores *Simultaneous Multithreading* (*SMT*) sólo pueden aprovechar todo su potencial utilizando bancos de registros muy grandes para satisfacer las necesidades de almacenamiento procedentes de varios *threads* [TEL95]. De la misma forma, procesadores con grandes ventanas de lanzamiento también van a precisar de bancos de registros enormes [MSW+03].

Muchos son los trabajos que suponen que el tiempo de acceso al banco de registros puede afectar al tiempo de ciclo del procesador [BDA01b][CGV+00][FJC96][FCJ+97] y que el banco de registros es una de las estructuras que más energía consumen [ZK98]. Intel P4, por ejemplo, dedica dos ciclos a la lectura del banco de registros [HSU+01]. Veremos a continuación un extracto de cuáles han sido las distintas soluciones que se han propuesto para resolver el problema del acceso a este tipo de bancos de registros.

Una de las soluciones propuestas, consiste en segmentar el acceso al banco de registros [TEE+96]. Las mismas técnicas que se utilizan para segmentar memorias RAM pueden utilizarse para segmentar el banco de registros [NF94]. Sin embargo, segmentar el banco de registros no es trivial y puede limitar el rendimiento del procesador de forma importante. Un banco de registros que necesita de varios ciclos para su acceso lleva consigo el aumento de la penalización

en una mala predicción de salto y requiere añadir niveles extra de lógica de *bypass*. En definitiva, va a aumentar la longitud de algunos bucles claves en la microarquitectura. Por ejemplo, [BTM+02] ha estudiado la existencia de un par de bucles críticos en el camino de lanzamiento de instrucciones. Son bucles que implican a más de una etapa (*loose loops*): el bucle de resolución de un salto y el bucle de dependencias con memoria. La solución que proponen los autores consiste en mover el acceso al banco de registros fuera de ese camino, utilizando lo que denominan *DRA* (*Distributed Register Algorithm*).

Por otro lado, el tamaño del banco de registros (P registros) y su número de puertos (T puertos en total, los de lectura más los de escritura) determinan tanto el área que ocupa en el procesador como su tiempo de acceso y consumo de energía. Para los valores de P y T que nos interesan (P < 160 registros y T < 60 puertos), el área y el consumo son proporcionales a $P \cdot T^2$ y el tiempo de acceso es proporcional a $\sqrt{P} \cdot T$ [RDK+00].

Una forma más directa de reducir el tiempo de acceso al banco de registros consiste en reducir P, T o los dos. Las soluciones más comunes buscan que exista un equilibrio entre la disminución de *IPC* (instrucciones que hacen el *commit* por ciclo) y el incremento de *IPS* (instrucciones por segundo) obtenido.

Un primer grupo de soluciones actúa sobre la organización interna del banco de registros, sin cambiar su comunicación con las unidades funcionales (T no cambia) pero disminuyendo de alguna forma el valor de P. Algunos de estos ejemplos son el banco de registros *Minimally-Ported Banked* en [BDA01b] o la organización jerárquica del banco de registros en dos niveles, con sus contenidos gestionados bien por inclusión [CGV+00] o por exclusión [BDA01b].

Un segundo grupo de soluciones propone microarquitecturas *clusterizadas*, en las cuales el banco de registros monolítico se divide en bancos, cada uno de los cuales alimenta a un *cluster* de unidades funcionales. Este tipo de solución ha sido utilizado habitualmente para descentralizar no sólo el banco de registros, sino cualquier otra estructura crítica en el procesador. Un ejemplo es la arquitectura *Dependence-Based* en [PJS97], donde cada banco es una copia completa del banco de registros (igual número de registros P, menor número de puertos T). Lo mismo ocurre con los dos *clusters* que se han implementado en el procesador de Alpha 21264 [Kes99]. Otros ejemplos son la arquitectura *Multicluster* en [FCJ+97] donde a cada banco se le asigna un subconjunto de los registros del *ISA*, la arquitectura *Energy-Efficient Multicluster* [ZK01] en la que cada banco contiene un subconjunto de los registros físicos (en ambas se reducen tanto P como T) y todo tipo de heurísticas relacionadas con optimizar la asignación de instrucciones a los distintos *clusters* [BM00][CPG00].

En la propuesta del DRA [BTM+02], se unen las dos ideas de utilizar una jerarquía para el banco de registros y microarquitecturas descentralizadas. El algoritmo DRA consigue disminuir la latencia del camino de lanzamiento de instrucciones. Para ello, mueve fuera de ese camino al gran banco de registros que tiene asociado un tiempo de acceso elevado. DRA utiliza una lógica para el lanzamiento de las instrucciones muy parecida a la implementada en el Alpha 21264 [Kes99] y distribuye las instrucciones a los distintos *clusters* en la etapa de decodificación. El gran banco de registros mezclado se coloca ahora en el camino de decodificación de las instrucciones y un subconjunto de él se distribuye en pequeñas *caches* de registros que se colocan dentro de cada *cluster*.

En este trabajo, nosotros proponemos actuar en una dirección complementaria: actuar sobre el mecanismo que controla la *asignación* y *liberación* de los registros físicos, intentando reducir el número promedio de registros que se necesitan. Los procesadores actuales utilizan muchos más registros físicos que los estrictamente necesarios para almacenar los valores que están siendo leídos. Esto ocurre porque los registros se asignan demasiado pronto y se liberan demasiado tarde. A cada instrucción que posee registro destino, se le asigna un registro físico en su etapa de decodificación, mucho antes de que esté disponible el resultado que se va a almacenar en dicho registro. Por otro lado, un registro físico se libera mucho más tarde del momento en que la instrucción que lo lee por última vez alcanza su etapa de *commit*, exactamente cuando la primera instrucción que escribe sobre el mismo registro lógico alcanza la etapa de *commit*. Toda esta Tesis se dedica a reducir al máximo las pérdidas que provocan esos dos factores.

## 1.3 Contribuciones de esta Tesis

En esta Tesis se han desarrollado distintas técnicas *hardware* dedicadas a optimizar la utilización de los registros en procesadores superescalares que disponen de un banco de registros mezclado. En el Capítulo 2 se muestra cuál es el tipo de procesador de base que se va a utilizar y sobre el que se van a comparar todas las técnicas propuestas, demostrándose experimentalmente la poca eficiencia de los mecanismos convencionales de gestión de los registros físicos.

El primer conjunto de contribuciones se centra en la parte de la asignación y se concreta en el concepto *registros virtuales-físicos*. En el Capítulo 3 se presenta este concepto, las estructuras *hardware* que se proponen para implementarlo. Todo lo anterior dio origen a las publicaciones [GVG+97][MGV+99a][MGV+99b] y [MGV+00a].

La siguiente contribución se centró en la parte de la liberación de los registros y generó un par de estrategias de complejidad incremental que dieron origen a las publicaciones [MGV+00b][MVG+01] y [MVG+02]. En el Capítulo 4 se presenta una descripción completa del *hardware* propuesto para las estrategias de *liberación anticipada de registros*, así como del funcionamiento de las mismas.

La última de las contribuciones ha consistido en componer en un único mecanismo las dos estrategias más agresivas que se diseñaron para la parte de la asignación y liberación de los registros. Este mecanismo denominado vp-LAER[1] es el que se presenta en el Capítulo 5. Actualmente esta propuesta ha sido sometida a la revista *Transactions on Computers* del *IEEE*.

## 1.4 Referencias

[AA93] D. Alpert, and D. Avnon "Architecture of the Pentium Microprocessor," *IEEE Micro*, vol. 13, no. 3, pp. 11-21, May/June 1993.

[AST67] D.W. Anderson, F.J. Sparacio, and R.M. Tomasulo, "The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling," *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 8-24, January 1967.

1. vp-LAER: del inglés *virtual-physical with Late Allocation and Early Release.*

[BAH+94] B. Burgess, M. Alexander, Ying-wai Ho, S. Plummer, S. Mallick, D. Ogden, Sung-Ho Park, and J. Slaton "The Power PC 603 Microprocessor: A High Performance, Low Power, Superscalar RISC Microprocessor," *in Proceedings of the 39th IEEE Computer Society International Conference* (COMPCON 94), pp. 300-306, February 1994.

[BAM+93] M.C. Becker, M.S. Allen, C.R. Moore, J.S. Muhich, and D.P. Tuttle, "The PowerPC 601 Microprocessor," *IEEE Micro*, vol. 13, no. 5, pp. 54-68, October 1993.

[BDA01b] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi, "Reducing the Complexity of the Register File in Dynamic Superscalar Processors," *in Proceedings of the 34th International Symposium on Microarchitecture* (MICRO 01), pp. 237-249, December 2001.

[BDH+94] J. Barreh, S. Dhawan, T. Hicks, and D. Shippy "The POWER2 Processor," *in Proceedings of the 39th IEEE Computer Society International Conference* (COMPCON 94), pp. 389-398, February 1994.

[BK92] G. Blanck, and S. Kreuger, "The SuperSPARC Microprocessor," *in Proceedings of the 37th IEEE Computer Society International Conference* (COMPCON 92), pp. 136-141, February 1992.

[BM00] A. Baniasadi, and A. Moshovos, "Instruction Distribution Heuristics for Quad-Cluster, Dynamically Scheduled, Superscalar Processors," *in Proceedings of the 33rd International Symposium on Microarchitecture* (MICRO 00), pp. 337-347, December 2000.

[BTM+02] E. Borch, E. Tune, S. Manne, and J. Emer, "Loose Loops Sink Chips," in *Proceedings of the 8th International Symposium on High-Performance Computer Architecture* (HPCA 02), pp. 299-310, February 2002.

[CGV+00] J. Cruz, A. González, M. Valero, and N.P. Topham, "Multiple-Banked Register File Architectures," *in Proceedings of the 27th International Symposium on Computer Architecture* (ISCA 00), pp. 316-325, June 2000.

[CPG00] R. Canal, J.M. Parcerisa, and A. González, "Dynamic Cluster Assignment Mechanisms," *in Proceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 133-144, January 2000.

[DWY+92] E. DeLano, W. Walker, J. Yetter, and M. Forsyth, "A High Speed Superscalar PA-RISC Processor," *in Proceedings of the 37th IEEE Computer Society International Conference* (COMPCON 92), pp. 116-121, February 1992.

[FCJ+97] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic, "The Multicluster Architecture: Reducing Cycle Time Through Partitioning," *in Proceedings of the 30th International Symposium on Microarchitecture* (MICRO 97), pp. 149-159, December 1997.

[FJC96] K.I. Farkas, N.P. Jouppi, and P. Chow, "Register File Considerations in Dynamically Scheduled Processors," *in Proceedings of the 2nd International Symposium on High-Performance Computer Architecture* (HPCA 96), pp. 40-51, February 1996.

[Gro90] G.F. Grohoski, "Machine organization of the IBM RISC System/6000 processor," *IBM Journal Resources Development*, vol. 34, no. 1, pp. 37-58, January 1990.

[GVG+97] A. González, M. Valero, J. González, and T. Monreal, "Virtual Registers," *in Proceedings of the 4th International Conference on High Performance Computing* (HiPC 97), pp. 364-369, December 1997.

[Gwe94a] L. Gwennap, "NexGen enters market with 66-MHz Nx586," *Microprocessor Report*, vol. 8, no. 4, pp. 12-17, March 1994.

[Gwe94b] L. Gwennap, "UltraSparc unleashes SPARC performance," *Microprocessor Report*, vol. 8, no. 13, pp. 1-10, October 1994.

[Gwe95] L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," *Microprocessor Report*, vol. 9, no. 2, pp. 9-15, February 1995.

[Gwe96] L. Gwennap, "Digital 21264 Sets New Standard," *Microprocessor Report*, vol. 10, no. 14, pp. 11-16, October 1996.

[Gwe97] L. Gwennap, "MIPS R12000 to Hit 300 MHz," *Microprocessor Report, Micro Design Resources*, vol. 11, no. 13, pp. 1-4, October 1997.

[HP03] J.L. Hennessy, and D.A. Patterson, *Computer Architecture, A Quantitative Approach*, Morgan Kaufmann Publishers, San Francisco, Third Edition 2003.

[HSU+01] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The Microarchitecture of the Pentium 4 Processor," *Intel Technology Journal Q1*, February 2001.

[Hun95] D. Hunt, "Advanced Performance Features of the 64-bit PA-8000," *in Proceedings of the 40th IEEE Computer Society International Conference* (COMPCON 95), pp. 123-128, March 1995.

[Kel75] R.M. Keller, "Look-Ahead Processors," *ACM Computing Surveys*, vol. 7, no. 4, pp. 177-195, December 1975.

[Kes99] R.E. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24-36, March-April 1999.

[KP99] J. Keshava, and V. Pentkovski, "Pentium III Processor Implementation Tradeoffs," *Intel Technology Journal Q2*, 1999.

[Lip92] J.S. Liptay, "Design of the IBM Enterprise System/9000 high-end processor," *IBM Journal Resources Development*, vol. 36, no. 4, pp. 713-731, July 1992.

[McL93] E. McLellan, "The Alpha AXP Architecture and 21064 Processor," *IEEE Micro*, vol. 13, no. 3, pp. 36-47, May/June 1993.

[MGV+99a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, "Delaying Physical Register Allocation Through Virtual-Physical Registers," *in Proceedings of the 32nd International Symposium on Microarchitecture* (MICRO 99), pp. 186-192, November 1999.

[MGV+99b] T. Monreal, A. González, M. Valero y V. Viñals, *Registros Virtuales*, Report interno del DIIS, RR-99-08, 1999.

[MGV+00a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, "Dynamic Register Renaming Through Virtual-Physical Registers," *in The Journal of Instruction-Level Parallelism*, vol. 2, May 2000. http://www.jilp.org/vol2.

[MGV+00b] T. Monreal, A. González, V. Viñals y M. Valero, "Liberación Anticipada de Registros," *en XI Jornadas de Paralelismo*, pp. 21-27, Septiembre 2000.

[MSW+03] O. Mutlu, J. Stark, C. Wilkerson, and Y.N. Patt, "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture* (HPCA 03), pp. 129-140, February 2003.

[MVG+01] T. Monreal, V. Viñals, A. González, and M. Valero, *Early Register Release*, Report interno del DIIS, RR-01-01, pp. 1-21, 2001.

[MVG+02] T. Monreal, V. Viñals, A. González, and M. Valero, "Hardware Schemes for Early Register Release," *in Proceedings of the 31st International Conference on Parallel Processing* (ICPP 02), pp. 5-13, August 2002.

[NCF+02] S.D. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T.J. Sullivan, and T. Grutkowski, "The Implementation of the Itanium 2 Microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, November 2002.

[NF94] K.J. Nowka, and M.J. Flynn, *Wave Pipelining of High Performance CMOS Static RAM*, Technical Report 94/615, Computer Systems Laboratory, January 1994.

[Omo99] A.R. Omondi, *The Microarchitecture of Pipelined and Superscalar Computers*, Kluwer Academic Publishers, Boston, 1999.

[PII97] "Pentium II Processor Developer's Manual," *Intel Corporation 24350-2001*, October 1997.

[PJS97] A.S. Palacharla, N.P. Jouppi, and J.E. Smith, "Complexity-Effective Superscalar Processors," *in Proceedings of the 24th International Symposium on Computer Architecture* (ISCA 97), pp. 206-218, June 1997.

[PPE+97] Y. Patt, S.J. Patel, M. Evers, D.H. Friendly, and J. Stark, "One Billion Transistors, One Uniprocessor, One Chip," *IEEE Computer*, vol. 30, no. 9, pp. 51-57, September 1997.

[Qua00] N. Quach, "High Availability and Reliability in the Itanium Processor," *IEEE Micro*, vol. 20, no. 5, pp. 61-69, September-October 2000.

[RDK+00] S. Rixner, W.J. Dally, B. Khailany, P. Mattson, U.J. Kapasi, and J.D. Owens, "Register Organization for Media Processing," in *Proceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 375-386, January 2000.

[RF72] E.M. Riseman, and C.C. Foster, "Percolation of code to enhance parallel dispatching and execution," *IEEE Transactions on Computers*, vol. 21, no. 12, pp. 1411-1415, December 1972.

[SFK97] D. Sima, T. Fountain, and P. Kacsuk, *Advanced Computer Architectures, A Design Space Approach*, Addison Wesley Longman Inc., New York, 1997.

[Sim00] D. Sima, "The Design Space of Register Renaming Techniques," *IEEE Micro*, vol. 20, no. 5, pp. 70-83, September-October 2000.

[SP85] J.E. Smith, and A.R. Pleszkun, "Implementation of Precise Interrupts in Pipelined Processors," *in Proceedings of the 12th International Symposium on Computer Architecture* (ISCA 85), pp. 36-44, June 1985.

[SP88] J.E. Smith, and A.R. Pleszkun, "Implementing Precise Interrupts in Pipelined Processors," *IEEE Transactions on Computers*, vol. 37, no. 5, pp. 562-573, May 1988.

[SR00] M. Schlansker, and B. Rau, *An Architecture of Instruction Level Parallel Processors*, HP Laboratories Report HPL-1999-111, February 2000.

[SS98] B. Shriver, and B. Smith, "The Anatomy of a High-Performance Microprocessor," *IEEE CS Press*, 1998.

[SS95] J.E. Smith, and G.S. Sohi, "The Microarchitecture of Superscalar Processors," *in Proceedings of the IEEE*, vol. 83, no. 12, pp. 1609-1624, December 1995.

[TEE+96] D.M. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," *in Proceedings of the 25th International Symposium on Computer Architecture* (ISCA 96), pp. 191-202, May 1996.

[TEL95] D.M. Tullsen, S. Eggers, and H. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *in Proceedings of the 22nd International Symposium on Computer Architecture* (ISCA 95), pp. 392-403, June 1995.

[TF70] G.S. Tjaden, and M.J. Flynn, "Detection and Parallel Execution of Independent Instructions," *IEEE Transactions on Computers*, vol. C-19, pp. 889-895, October 1970.

[Tho64] J.E. Thornton, "Parallel Operation in the Control Data 6600," *in Proceedings AFIPS Fall Joint Computer Conference*, vol. 26, no. 2, pp. 33-40, 1964.

[Tom67] R.M. Tomasulo, "An Efficient algorithm for exploiting multiple arithmetic units," *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 25-33, January 1967.

[Wal91] D.W. Wall, "Limits of Instruction-Level Parallelism," *in Proceedings of the 4th International Conference on Architectural Support for Programing Languages and Operating Systems* (ASPLOS 91), pp. 176-188, April 1991.

[Yan94] P. Yan-Tek Hsu, "Designing the TFP Microprocessor," *in IEEE Micro*, vol. 14, no. 2, pp. 23-33, April 1994.

[Yea96] K.C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *in IEEE Micro*, vol. 16, no. 2, pp. 28-40, April 1996.

[ZK01] V.V. Zyuban, and P.M. Kogge, "Inherently Lower-Power High-Performance Superscalar Architectures," *IEEE Transactions on Computers*, vol. 50, no. 3, pp. 268-285, March 2001.

[ZK98] V.V. Zyuban, and P.M. Kogge, "The Energy Complexity of Register Files," *in Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 305-310, August 1998.

CAPÍTULO 2

# Renombre de Registros Convencional


## Resumen

*En este Capítulo se detallan las estructuras utilizadas para implementar la técnica del renombre de registros sobre un banco de registros mezclado. La misma implementación se utiliza en los procesadores MIPS R10K/12K, por eso se ha denominado técnica de renombre convencional. Se estudiará cómo se realiza la asignación y liberación de los registros con este mecanismo de renombre y cuáles son sus ineficiencias. Se introducen también toda una serie de definiciones que van a ir siendo utilizadas en el resto de los capítulos. Son definiciones relacionadas con el dimensionado que puede tener el banco de registros y con la situación de uso de los registros físicos desde que se asignan hasta que se liberan.*

En el código estático de los programas, las instrucciones almacenan sus resultados en los elementos de almacenamiento lógicos que se corresponden con registros o con lugares de la memoria principal. Durante la ejecución de los programas se utilizan otros elementos de almacenamiento físico (que pueden ser distintos de los lógicos) para aumentar el paralelismo a nivel de instrucción.

En la ejecución paralela de un programa pueden existir varios valores asociados al mismo registro lógico pero almacenados en distintos registros físicos. Cuando una instrucción genera un nuevo valor para su registro lógico destino, al lugar físico donde reside ese valor se le asigna un nombre que es conocido a nivel del *hardware*. A ese nombre se le denomina *registro de renombre* o *versión*. Esta tarea se lleva a cabo en la fase de decodificación de la instrucción y se denomina *renombre del registro.* A las instrucciones posteriores que usen dicho valor como entrada, se les debe proporcionar la versión correcta del registro. Las versiones de los registros se escriben una sola vez y se leen tantas veces como sea necesario. Todas las versiones que existen de un mismo registro lógico, excepto la más vieja, son especuladas y por lo tanto pueden acabar siendo invalidadas si ocurriese alguna excepción o una mala predicción de salto. Finalmente, toda versión se libera del registro lógico en la etapa de *commit* cuando deja de ser especulada.

## 2.1 Dimensionado del Banco de Registros

En todo este trabajo se va a considerar la opción de diseño en que las versiones de los registros se encuentran formando parte del banco de registros físico de la arquitectura. Este es el banco de registros mezclado que contiene tanto las versiones que ya representan el estado preciso de la máquina (versiones no especuladas) como las versiones que todavía son especuladas. A continuación se va a detallar cómo funciona el mecanismo de renombre convencional sobre este tipo de banco de registros y qué ineficiencias presenta tanto en la forma de asignar los registros como en la forma de liberarlos.

El banco de registros mezclado utilizado en un procesador superescalar debe contener un número elevado de registros (P) y de puertos (T) pero no debe comprometer al tiempo de ciclo del procesador y debe tener un consumo razonable. Una forma directa de reducir el tiempo de acceso a un banco de registros consiste en reducir el valor de P, T o de los dos. Las técnicas que se presentan en esta Tesis reducen el número de registros utilizados basándose en la modificación del mecanismo convencional de asignación y liberación de los registros físicos.

El tipo de dimensionado que caracteriza un determinado banco de registros puede ser *sobrado* o *ajustado*. La Figura 2.1 muestra estos dos tipos de dimensionado. N es el número de entradas que posee la estructura de reordenación del procesador y L es el número de registros lógicos que están definidos en su arquitectura. Un *banco de registros sobrado* es aquel para el que se cumple que $P \approx L + N$ (Figura 2.1.a). Un procesador con este tipo de estructura, nunca va a parar por quedarse sin registros y por lo tanto va a poder extraer el máximo ILP de los programas.

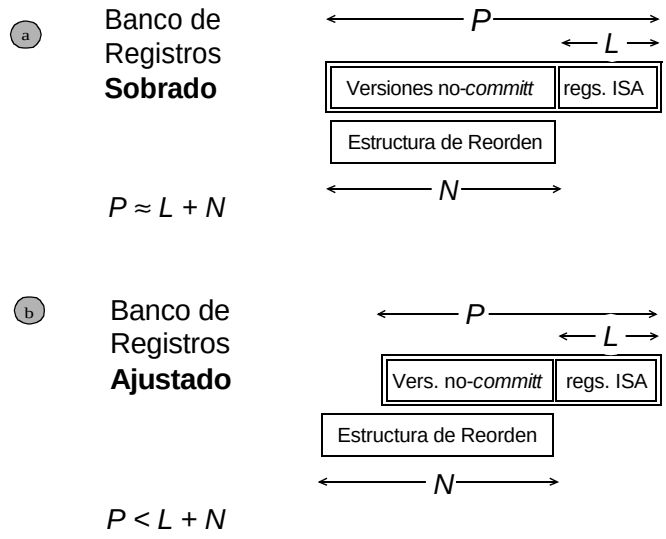


**Figura 2.1** Relación que existe entre P, L y N para un banco de registros sobrado o ajustado.

Por otro lado, en un *banco de registros ajustado* se cumple que P < L + N (Figura 2.1.b). En este caso, el procesador puede llegar a parar su fase de decodificación si aparece una secuencia de instrucciones que precisan de registro destino lo suficientemente grande para agotar todos los registros. En esta situación el IPC del programa puede disminuir de forma temporal.

Un diseño sobrado va a permitir que la ventana de lanzamiento se llene bajo cualquier condición, mientras que un diseño ajustado puede contribuir a reducir el tiempo de ciclo del procesador en el caso de que el banco de registros se encuentre en el camino crítico. Cualquiera de estos dos diseños es viable y se encuentra formando parte de la implementación de algunos procesadores superescalares actuales.

| | **MIPS R10K** | **MIPS R12K** | **ALPHA 21264** | **INTEL P4** |
|---|---|---|---|---|
| *P* = # de Reg. físicos del BR **Entero** | 64 | | 2 x 80 | 128[b] |
| *T* = # de puertos de Lectura y Escritura | 7L 3E | | 2 x (4L 6E)[a] | n.a. |
| *P* = # de Reg. físicos del BR **Fp** | 64 | | 72 | 128[c] |
| *T* = # de puertos de Lectura y Escritura | 5L 3E | | 6L 4E | n.a. |
| *N* = Tamaño de la Estruc. Reordenación | 32 | 48 | 80 | 126 µops |
| *Nombre de la Estructura de Reorden* | *Active List* | *Active List* | *In-Flight Window* | *Reorder Buffer* |

a. El Banco de registros entero del Alpha 21264 se ha replicado porque disponer de un único banco de registros con un número elevado de puertos, podía comprometer el rendimiento.
b. El acceso al banco de registros está completamente segmentado y utiliza dos ciclos.
c. FP(x87), MMX, SSE y SSE-II registros del ISA.

**Tabla 2.1** Ejemplo de procesadores superescalares con ejecución fuera de orden y banco de registros mezclados.

Como ejemplo, la Tabla 2.1 presenta cuatro procesadores que utilizan banco de registros mezclado en su implementación: MIPS R10K/12K [Yea96]/[Gwe97], DEC Alpha 21264 [Gwe96] e Intel P4 [HSU+01]. MIPS R10K soporta hasta N = 32 instrucciones en su estructura de reordenación (son instrucciones que todavía no han alcanzado la etapa de *commit*) la cual se denomina *Active List*. Como el ISA de MIPS posee L = 32 registros lógicos y P = 64 registros físicos, este procesador nunca va a parar por falta de registros físicos. Por tanto, MIPS R10K es uno de los procesadores que poseen un banco de registros sobrado ($P \approx L + N$ ). Por el contrario, en MIPS R12K y Alpha 21264, una secuencia de instrucciones que sea lo suficientemente grande -sin saltos ni *stores*-, puede hacer parar el proceso de renombre. Estos procesadores poseen un banco de registros ajustado ($P < L + N$ ). Con un banco de registros ajustado, una secuencia con menos de N instrucciones que escriben $P - L$ registros, consume todos los registros físicos, forzando al procesador a dejar de llenar la estructura de reordenación. El procesador Intel P4 posee bancos de registros grandes: 128 registros enteros + 128 registros fp, pero no podemos decir si esos bancos son sobrados o ajustados debido a la falta de detalles sobre como funciona en este procesador el mapeo de las instrucciones del ISA a μops.

## 2.2 Estructuras de renombre

Durante la fase de renombre, el procesador debe asignar los registros de renombre (versiones) a los registros lógicos destino de las instrucciones que lo precisen. Además, debe seguir la pista de las versiones usadas en cada momento y liberarlas cuando ya no se necesiten más. La fase de renombre posee tres actividades que deben ser cubiertas: asignación de los registros de renombre, seguimiento de las versiones que existen en cada momento y liberación de los registros de renombre.

Se va a suponer un mecanismo de renombre similar al utilizado en los procesadores de la Tabla 2.1. La Figura 2.2 muestra todos los componentes que se utilizan para la implementación: la tabla de mapeo (*MT, Map Table*), la lista de registros físicos disponibles o libres (*Free List*), la estructura de reordenación (*ROS, ReOrder Structure*) y la tabla de mapeo en orden (*IOMT, In Order Map Table*).

Con el mecanismo de renombre convencional, el procesador posee un banco de registros físico de tamaño mayor que el número de registros lógicos. La tabla de mapeo *MT* se utiliza para asociar un registro físico a un determinado registro lógico [Kel75]. La lista de registros libres (*Free List*) proporciona los identificadores para los registros físicos destino de las instrucciones (pd: versión actual del correspondiente registro lógico).

La estructura de reordenación *ROS* guarda en orden de programa información relativa a todas las instrucciones que todavía no han alcanzado la etapa de *commit*. Se va a suponer una implementación de esta estructura como un *buffer* circular cuyas entradas se asignan y desasignan de forma FIFO utilizando dos punteros (el *puntero a la cabeza* y el *puntero a la cola*). Cada una de sus entradas se puede usar así como identificador único de toda instrucción. El orden de las instrucciones va desde las *instrucciones más viejas* que residen en el *ROS* y que comienzan en el puntero situado a la cabeza hasta las *instrucciones más jóvenes* que finalizan en el puntero situado a la cola. Las instrucciones entran en orden de programa al *ROS* durante la etapa de decodificación, ocupando la entrada que indica el puntero de la cola y salen también en orden de

programa en la etapa de *commit*, cuando alcanzan el puntero de la cabeza (consultar la Figura 2.2). Con esta estructura, las instrucciones se retiran definitivamente del procesador sólo cuando se recupera el orden secuencial del programa. Esto ocurre en el momento en que la instrucción alcanza su etapa de *commit* y sólo cuando todas las instrucciones previas también han alcanzado esta etapa.

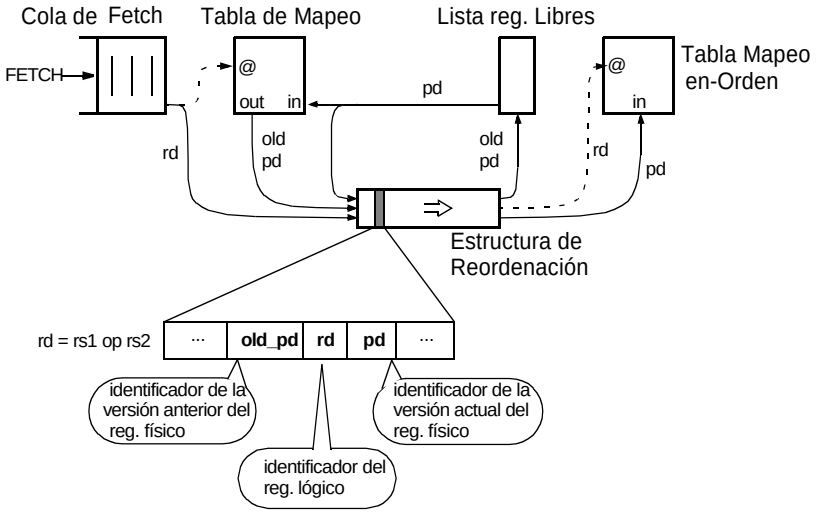


**Figura 2.2** Mecanismo convencional de asignación y liberación de los registros. Detalle de una entrada del *ROS*.

Por cada instrucción que se está decodificando se guardan tres identificadores en la entrada apuntada por el puntero a la cola del *ROS* (consultar la Figura 2.2):

- rd: identificador del registro lógico destino de la instrucción.
- pd: identificador del registro físico asignado como versión actual a rd.
- old_pd: identificador del registro físico que contiene la versión anterior del registro lógico. Este identificador se obtiene de la *MT*.

Al igual que en un *Reorder Buffer* (*ROB*), cada entrada del *ROS* almacena el identificador del resultado producido por la instrucción (pd). Pero, al igual que en un *History Buffer* (*HiB*) [SP88], también contiene el identificador de la versión anterior (old_pd). Esta es la razón por la que se ha adoptado el nombre de *ROS* como término general para nombrar a la estructura de reordenación.

La tabla *IOMT* guarda en el orden original del programa las asignaciones entre registros lógicos y registros físicos. Se va actualizando en la etapa de *commit* y sirve para restaurar el estado preciso de la máquina tras una interrupción.

### 2.2.1 Asignación de los registros

Las instrucciones se decodifican y renombran siguiendo el orden secuencial del programa. Cuando una instrucción se decodifica, a su registro lógico destino (rd) se le asigna un registro físico que se extrae de la *Free List* (pd). Es decir, a todo registro lógico destino de una instrucción se le asigna un nuevo registro físico que no está asignado a ningún otro registro lógico (una nueva versión). El registro físico seleccionado se elimina entonces de la *Free List*. En el caso de que no exista ningún registro físico disponible para asignar a la instrucción (la *Free List* está vacía), el procesador para de forma temporal la fase de decodificación.

La tabla de mapeo se consulta para conocer las asignaciones que existen en cada momento entre registros lógicos y registros físicos y se actualiza para reflejar las nuevas asignaciones. La *MT* posee tantas entradas como registros lógicos posee la arquitectura (normalmente 32). En la etapa de renombre, se indexa la *MT* con los identificadores de los registros *lógicos fuente* de la instrucción para encontrar los registros físicos que tienen asignados. Las instrucciones obtienen así la identificación de los lugares del banco de registros físico desde donde leer sus operandos fuente al comenzar su etapa de ejecución. La *MT* también se indexa con el registro *lógico destino* de la instrucción. El identificador leído se guarda en el campo old_pd que la instrucción tiene en el *ROS* (la versión anterior) y en la *MT* se almacena el identificador del nuevo registro físico asignado (pd: la nueva versión). Cada asignación en la *MT* se mantiene así hasta que el registro lógico asociado se renombra de nuevo. De esta forma la *MT* siempre refleja la última asignación de cada registro lógico permitiendo así hacer el seguimiento de qué versiones se están usando en cada momento.

Las instrucciones especuladas también irán actualizando la *MT* con versiones especuladas. En el momento en que se descubre una mala predicción de un salto, las versiones especuladas deben invalidarse y sustituirse por la versión asignada antes de la predicción del salto. La *MT* es lo suficientemente pequeña como para salvarla cada vez que se predice un salto y utilizarla para restaurar las versiones correctas tras una mala predicción. Un procesador capaz de gestionar varios saltos pendientes hace en orden de programa una copia de la *MT* en uso y la asocia a cada salto. Si más adelante, se descubre que el salto fue predicho de forma correcta, la correspondiente copia de la *MT* para ese salto se invalida. Si por el contrario, se descubre que el salto fue mal predicho, la *MT* en uso se sustituye por la correspondiente copia y se invalidan también todas las copias posteriores al salto resuelto. Todo este proceso permite al procesador recuperarse de saltos mal predichos en un sólo ciclo y se corresponde con un mecanismo de *checkpointing* parecido al que utiliza el MIPS R10K [Yea96].

### 2.2.2 Liberación de los registros

Los registros físicos asignados a los distintos registros lógicos deben ser liberados y volver a la *Free List*. Después de que el valor que almacena se lea por última vez, el registro físico puede quedar libre y ser utilizado por una instrucción posterior. Además, la liberación de los registros considerada en el mecanismo de renombre convencional permite una restauración del estado preciso de la máquina en caso de una interrupción, o lo que es lo mismo, soporta un mecanismo de interrupciones preciso. Para ello, un registro físico no se libera hasta que la instrucción que vuelve a renombrar al correspondiente registro lógico no alcanza la etapa de *commit*. En este

momento es seguro que la anterior versión del registro lógico ya no va a leerse más. Por tanto, en la etapa de *commit*, el registro físico identificado en el campo old_pd del *ROS* se devuelve a la *Free List*.

También en la etapa de *commit* los identificadores pd y rd son usados para actualizar la *IOMT*. De esta forma, la tabla *IOMT* guarda en el orden original del programa la asignación de registro lógico a registro físico. Tras una interrupción, el procesador podrá restaurar el estado preciso de la máquina directamente desde la tabla *IOMT*. Para ello la *MT* en uso se sustituye por la *IOMT*.

## 2.3 Ineficiencias del renombre

La técnica convencional del renombre de registros necesita disponer de un número de registros físicos elevado. Ese número de registros es además mayor que el estrictamente necesario para almacenar los valores generados por un programa. Esto es así debido a que los registros físicos se extraen de la *Free List* y se asignan a los registros lógicos demasiado pronto y también porque se devuelven a la *Free List* demasiado tarde.

Cada instrucción asigna un registro físico a su registro lógico destino mucho antes de que el valor a almacenar en el registro se produzca. También ese registro físico se libera mucho después de que el valor que almacena ya no sea utilizado por ninguna instrucción.

Ahora se va a ver con un poco más de detalle la actividad que realiza todo registro físico desde que es asignado en la fase de decodificación hasta que es liberado en la etapa de *commit*. Esto servirá para definir lo que entenderemos como *utilización* de los registros y facilitará la detección de las ineficiencias que posee el mecanismo de renombre convencional.

### 2.3.1 *Utilización* de los registros

Cada registro físico del banco de registros mezclado se encuentra Asignado o Libre. A su vez, un registro Asignado y según lo útil que es el valor que almacena, puede estar Vacío, Utilizado o Desocupado (consultar la Figura 2.3.a). Decimos que un registro físico está Vacío desde el momento en que es Asignado en la etapa de decodificación de una instrucción, hasta el momento en el que finalmente es utilizado para almacenar un valor (tras la fase ejecución de la instrucción). Un registro está Desocupado desde la etapa de *commit* de la instrucción que lo lee por última vez hasta la etapa de *commit* de la instrucción que produce su siguiente versión. Un registro Asignado decimos que está Utilizado cuando no está ni Vacío ni Desocupado.

La Figura 2.3.b muestra la ejecución de un fragmento de programa donde el registro físico p7 experimenta su paso por todos los estados. La instrucción V (*Version*), que escribe en el registro lógico r2, crea una nueva versión para su registro lógico destino r2 al renombrarlo al registro físico p7. Más adelante, la instrucción LU (*Last Use*, último uso de r2) lee por última vez el valor de r2. Finalmente, la instrucción NV (*Next Version*, siguiente versión de r2) reescribe r2.

En la Figura 2.3.b se observan dos ineficiencias que sufre el mecanismo convencional de renombre. El registro p7 empieza a estar Asignado en la etapa de decodificación de la instrucción V pero no almacena ningún valor hasta que dicha instrucción no completa su etapa de ejecución.

De esta forma, p7 se encuentra Asignado y Vacío durante todo ese tiempo. Después, p7 entra en el estado Utilizado en el cual estará disponible para cualquier instrucción que lo tenga como operando fuente. El registro p7 no se libera hasta que la instrucción NV llegue a su etapa de *commit*. p7 va a estar Asignado y Desocupado desde la etapa de *commit* de la instrucción LU hasta la etapa de *commit* de la instrucción NV.

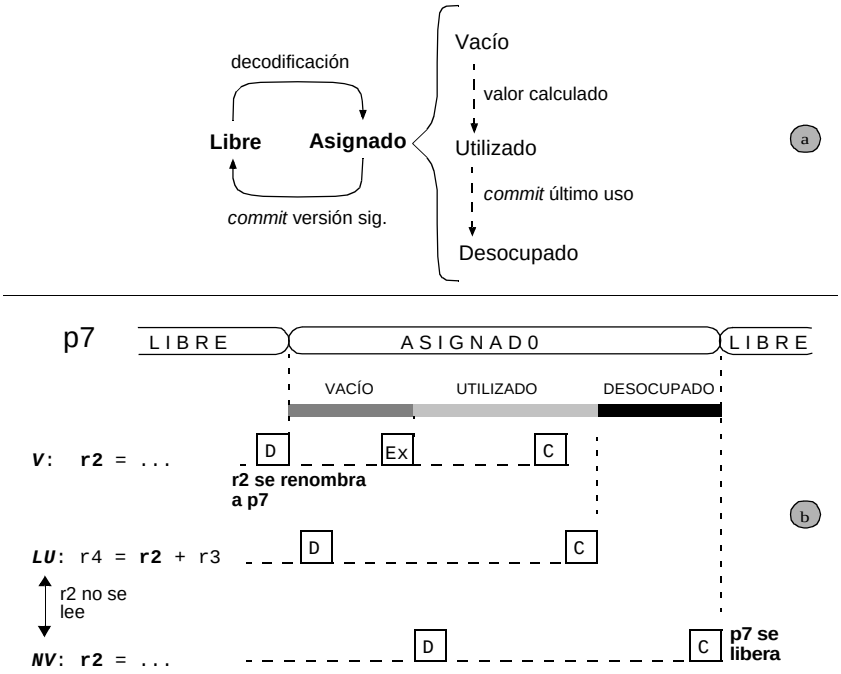


**Figura 2.3** Descomposición del estado Asignado **(a)** y ejemplo de la evolución de los estados por los que pasa el registro físico p7 utilizando una técnica de renombre convencional **(b)**. D = etapa de decodificación/renombre, Ex = ejecución, C = etapa de *commit*.

En resumen, el mecanismo de renombre convencional no es eficiente por dos razones:

1. Los registros físicos se asignan demasiado pronto. Los registros en estado Vacío no contienen información útil. En el Capítulo 3 se presentan los mecanismos propuestos para eliminar este estado Vacío de los registros. Dichos mecanismos se basan en la utilización de una técnica para retrasar la asignación de los registros físicos hasta la finalización de la etapa de ejecución de las instrucciones que escriben en ellos.
2. Los registros físicos se liberan demasiado tarde. Los registros en estado Desocupado no son utilizados nunca. En el Capítulo 4 se presentan los mecanismos propuestos para disminuir la permanencia de los registros en el estado Desocupado. Dichos mecanismos pasan a los registros al estado Libre lo antes posible manteniendo a su vez una recuperación precisa frente a las excepciones.

Cuando no dispone de registros físicos para asignar, el procesador debe parar la fase de decodificación, disminuyendo así el IPC efectivo. Se define como *IPC_loss* al número promedio de instrucciones que no pueden hacer el *commit* por ciclo debido a la no disponibilidad de registros físicos. Un procesador con el banco de registros más sobrado posible (P = 160, para L = 32 y N

= 128) convertiría este *IPC_loss* en trabajo efectivo, obteniendo un determinado *speed-up*. La Tabla 2.2 muestra los valores de *IPC_loss* obtenidos para un procesador superescalar de grado ocho, con un banco de registros mezclado y ajustado de 64 registros físicos (L = 32, P = $64_{ent}$+$64_{fp}$, N = 128) con respecto a la configuración sobrada (P=$160_{ent}$+$160_{fp}$). La última fila muestra el *speed-up* de la configuración sobrada con respecto a la ajustada. La anterior medida se ha realizado utilizando el mecanismo de renombre convencional sobre un subconjunto de programas de prueba de Spec95 -enteros y fp- [SPEC]. Para los programas de prueba de tipo entero se han considerado sólo los registros enteros y para los programas de prueba de tipo fp sólo los registros fp. El resto del entorno experimental en el que se ha realizado esta medida se muestra en el Capítulo 3.

| | ***enteros*** | | | | | ***fp*** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *comp* | *gcc* | *go* | *li* | *perl* | *mgri* | *tomc* | *appl* | *swim* | *hydr* |
| *IPC_loss* | *0.48* | *0.12* | *0.14* | *0.22* | *0.16* | *1.23* | *0.68* | *1.02* | *1.20* | *0.78* |
| *speed-up* | *1.18* | *1.06* | *1.06* | *1.07* | *1.06* | *1.60* | *1.43* | *1.40* | *1.67* | *1.39* |

**Tabla 2.2** *IPC_loss* y *speed-up* para un procesador de grado ocho con banco de registros de 64ent + 64fp y un *ROS* de 128 entradas.

En la tabla anterior se puede observar cómo al aumentar al máximo el número de registros, los programas de tipo entero podrían alcanzar un *speed-up* modesto aunque no nulo (1.08 en promedio -media armónica-), mientras que los programas de tipo fp se podrían beneficiar mucho (1.50 en promedio -media armónica-).

Se va a presentar ahora un experimento simple que pone en evidencia las ineficiencias del mecanismo de renombre convencional. Dicho experimento mide el número promedio de los registros Asignados que se encuentran en los estados de Vacío, Utilizado y Desocupado, así como la *utilización* de los registros. Definimos *utilización* como el porcentaje de registros Asignados que contienen un valor que va a ser leído en el futuro (número de registros Utilizados sobre el número de registros Asignados). La Figura 2.4 muestra los resultados de este experimento realizado utilizando el mismo entorno mencionado anteriormente.

El eje vertical de la Figura 2.4 muestra el número de registros que se encuentran Asignados (promedio temporal); está dividido en tres para mostrar por separado los estados Vacío, Utilizado y Desocupado. Para estos programas de prueba, del total de 64 registros físicos enteros, 53.5 se encuentran Asignados y 10.5 Libres (Ma -media aritmética-). De los Asignados, 11.9 están en estado Vacío, 21 en estado Utilizado y 20.6 en estado Desocupado. Lo anterior representa un porcentaje de registros en cada uno de los estados del 22%, 39% y 39% respectivamente. El mismo cálculo para los programas de prueba fp nos muestra que del total de 64 registros físicos fp, 60.2 se encuentran Asignados y 3.8 Libres. De los Asignados, 12.4 están en estado Vacío, 35.6 en estado Utilizado y 12.2 en estado Desocupado. Esto representa un porcentaje de registros en cada uno de los estados del 21%, 59% y 20% respectivamente.

En la Tabla A.1 del Apéndice A se pueden consultar los valores con los que se han construido las gráficas de la Figura 2.4.

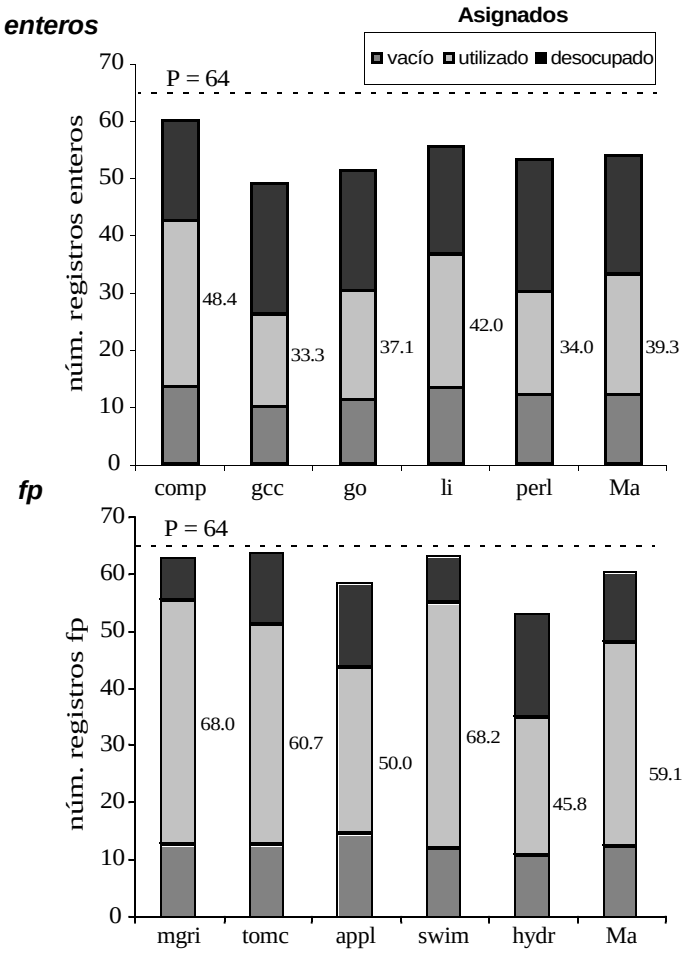


**Figura 2.4** Número de registros Asignados que se encuentran en los estados Vacío, Utilizado o Desocupado mediante un mecanismo de renombre convencional. 64ent+64fp registros.

La anterior medida nos permite observar que la presión sobre el banco de registros es relativamente baja para la mayoría de los programas enteros y además muchos de los registros Asignados están sin utilizar. A excepción de *compress* y *li*, todos los programas tienen más de 11 registros Libres, siendo 10.5 el valor promedio. Del total de 53.5 registros que están Asignados, 32.5 no se utilizan (Vacío + Desocupado: el 60.7% de los Asignados). Los programas enteros se caracterizan en general por una *utilización* de los registros muy baja (un 39.3% en promedio). Por otro lado, como indican en la Tabla 2.2 los modestos *speed-ups* que podrían alcanzarse en los programas de tipo entero, no es la falta de registros físicos la mayor limitación de estos programas. En este contexto, una mejora en el mecanismo de renombre permitirá principalmente ajustar el banco de registros entero sin perder IPC.

En los programas fp, a diferencia de lo que ocurre con los programas enteros, la presión sobre el banco de registros aumenta y es menor el número promedio de registros Asignados que están sin utilizar. Los programas *mgrid*, *tomcatv* y *swim* utilizan casi todos los registros físicos disponi-

bles y el número promedio de registros Libres es muy bajo: 3.8. De un total de 60.2 registros Asignados, 24.6 no se utilizan (Vacío + Desocupado: el 41% de los Asignados). Los programas fp se caracterizan en general por una *utilización* de los registros mucho mayor (59% en promedio). En este contexto, el principal efecto de la mejora del mecanismo de renombre resultará en reducir la presión ejercida sobre el banco de registros y en un incremento del IPC.

De esta forma, cualquier modificación en el mecanismo de renombre convencional que mejore la utilización de los registros va a ser beneficiosa para todo tipo de programas. Reducir el tamaño del banco de registros sin perder IPC será muy útil cuando el acceso al banco de registros se encuentre en el camino crítico del procesador. Su reducción puede traducirse en una reducción del tiempo de ciclo del procesador. Incrementar el IPC sin variar el tamaño del banco de registros, será útil en procesadores con bancos de registros pequeños porque se conseguirá sin degradar el tiempo de ciclo.

## 2.4 Conclusiones

El esquema de renombre de registros convencional no es eficiente porque asigna los registros físicos demasiado pronto y los libera demasiado tarde. Al mantener los registros físicos asignados durante un tiempo innecesario se impide que el procesador extraiga mayor paralelismo de los programas debido al agotamiento temprano de los registros. Un procesador podría beneficiarse de disponer de una cantidad de registros suficiente para todas las instrucciones que puede tener en vuelo -tamaño del ROS-.

La experimentación nos ha mostrado como un porcentaje importante de los registros asignados atraviesan por momentos durante los cuales no son utilizados, bien porque no almacenan ningún valor -estado vacío- o porque ya no hay instrucciones que los lean -estado desocupado-. Al mejorar el mecanismo de asignación y liberación se va a poder eliminar o disminuir la permanencia de los registros en el estado vacío o desocupado y se va a optimizar el uso que se hace de los registros.

## 2.5 Referencias


[Gwe96] L. Gwennap, "Digital 21264 Sets New Standard," *Microprocessor Report*, vol. 10, no. 14, pp. 11-16, October 1996.

[Gwe97] L. Gwennap, "MIPS R12000 to Hit 300 MHz," *Microprocessor Report, Micro Design Resources*, vol. 11, no. 13, pp. 1-4, October 1997.

[HSU+01] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The Microarchitecture of the Pentium 4 Processor," *Intel Technology Journal Q1*, February 2001.

[Kel75] R.M. Keller, "Look-Ahead Processors," *ACM Computing Surveys*, vol. 7, no. 4, pp. 177-195, December 1975.

[SP88] J.E. Smith, and A. R. Pleszkun, "Implementing Precise Interrupts in Pipelined Processors," *IEEE Transactions on Computers*, vol. 37, no. 5, pp. 562-573, May 1988.

[SPEC] The Standard Performance Evaluation Corporation, http://www.specbench.org.

[Yea96] K.C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *in IEEE Micro*, vol. 16, no. 2, pp. 28-40, April 1996.

CAPÍTULO 3

# Asignación Tardía de los Registros: la Técnica virtuales-físicos


## Resumen

*Este Capítulo presenta los mecanismos propuestos para mejorar el renombre convencional en la parte de la asignación. Se va a describir y evaluar una técnica de renombre de registros que permite retrasar la asignación de los registros físicos hasta el momento en que las instrucciones finalizan su ejecución. Como resultado, se mejora la utilización individual de cada registro físico y ello disminuye la "presión" sobre el banco de registros. En la evaluación de las diferentes variantes propuestas se muestran las posibilidades de la asignación tardía: bien aumentar el IPC en una situación de partida con banco de registros ajustado, bien reducir el tamaño del banco de registros sin perder IPC (para disminuir el retardo de acceso al banco y, potencialmente, el tiempo de ciclo del procesador).*

Para toda instrucción que produce un valor, el mecanismo de renombre convencional le asigna un registro físico durante un número de ciclos mayor que el necesario. Desde la etapa de decodificación de la instrucción hasta el final de su etapa de ejecución, el registro no contiene ningún valor (es el estado Vacío del registro definido en el Capítulo 2). La nueva técnica demora la asignación de registros, eliminando con ello los ciclos en los cuales los registros permanecen en el estado Vacío y reduciendo así la presión ejercida sobre el banco de registros. Notar que esa disminución puede ser muy útil en presencia de instrucciones de latencia larga y en partes del código de poco *ILP*. En estas circunstancias, algunas instrucciones permanecen durante mucho tiempo en la ventana de lanzamiento esperando sus operandos y utilizan de forma innecesaria un registro físico durante todo ese tiempo.

La fuente más común de instrucciones de latencia larga son las instrucciones de *load* que fallan en *cache*. Debido a que sigue aumentando la diferencia que existe entre la velocidad del procesador y de la memoria, también la latencia de fallo de un *load* medida en ciclos de procesador va a aumentar. Otra fuente de operaciones de latencia larga son las instrucciones complejas de aritmética en punto flotante, como son la división o la raíz cuadrada, aunque normalmente representan una fracción pequeña del total de instrucciones ejecutadas. En cualquier caso, e incluso para operaciones de latencia corta, reducir la presión que se ejerce sobre el banco de registros puede ser muy importante cuando el código incluye cadenas largas de instrucciones con dependencias. Además, también aumenta la cantidad de tiempo que las instrucciones permanecen en la ventana de lanzamiento antes de pasar a ejecutarse cuando dicha ventana crece, lo cual se espera que ocurra en los procesadores futuros.

El nuevo registro físico asignado al registro lógico destino de una instrucción sirve tanto para almacenar su valor resultado, como para seguir la pista de todas aquellas instrucciones dependientes de ésta. Sin embargo, hasta que se utiliza para almacenar un valor, el registro físico sólo actúa como *etiqueta* para identificar al último productor de cada registro lógico. Por esa razón, en la etapa de decodificación basta con asignar etiquetas (no almacenes) que se utilizarán para determinar desde dónde deben leerse los operandos. Estas etiquetas, que denominaremos *registros virtuales-físicos*, van a permitir retrasar la asignación de los registros físicos hasta la etapa de ejecución de las instrucciones.

Otro esquema para retrasar la asignación de los registros físicos fue propuesto por Wallace y Bagherzadeh en [WB96]. El objetivo de este trabajo era gestionar de forma escalable un banco de registros mezclado. Dicha escalabilidad la obtienen mediante una organización del banco de registros con múltiples bancos, cada uno con un único puerto de escritura. Al igual que con los registros *virtuales-físicos*, cada instrucción con registro destino se renombra en la etapa de decodificación utilizando un identificador único (denominado *itag*). Más adelante, las unidades funcionales solicitan un lugar físico para almacenar el resultado, lo que implica la asignación simultánea del puerto de alguno de los bancos y de un registro físico disponible de dicho banco. Esta asignación retardada la denominan DRR (*Dynamic Result Renaming*) y permite gestionar los múltiples bancos sin conflictos.

El Power3 también implementa lo que se denomina *renombre virtual* en [Son97]. Al igual que el PowerPC 620 [LTT95], posee dos bancos de registros: uno para los valores consolidados (el *banco de registros de la arquitectura*) y otro para valores no consolidados (*almacenes de re-*

*nombre*). En este procesador, existen 16 almacenes de renombre para datos de tipo entero y 24 para datos de tipo fp. Aunque permite hasta 32 instrucciones en vuelo, sólo deja lanzar a ejecutar las 16 (ó 24) instrucciones más viejas que poseen registro destino entero (o fp). Para ello, cada operando se identifica con un bit más de los necesarios para identificar al almacén de renombre. Este bit extra, denominado bit *virtual*, permite tener hasta dos instrucciones en vuelo que utilizan el mismo almacén de renombre para su resultado. La asignación más vieja se considera como la asignación que es *real*, mientras que la asignación más joven se considera como la asignación que es *virtual*. Ambas se distinguen por el valor de su bit *virtual*. Sólo son seleccionadas para lanzarse a ejecutar aquellas instrucciones que poseen todos sus operandos reales (fuentes y destino). Cuando una instrucción alcanza la etapa de *commit*, el almacén de renombre para su registro destino se cambia de virtual a real.

En apartados posteriores se verá cómo los esquemas de Wallace y Bagherzadeh y el del Power 3 representan dos puntos extremos en el espacio de diseño de los *registros virtuales-físicos*.

## 3.1 El concepto virtuales-físicos

La técnica de los registros *virtuales-físicos* se basa en añadir un nuevo tipo de registros a los dos que ya existen en un procesador convencional (los registros lógicos y los registros físicos). Los registros lógicos son los que se referencian por parte de las instrucciones del *ISA*. Cuando una instrucción se decodifica y como parte del proceso de renombre, se le asigna una nueva etiqueta a su registro lógico destino. Estas etiquetas no están asociadas a ningún almacén físico y son las que llamamos registros *virtuales-físicos* (registros *vp*). A este proceso ahora se le denomina *renombre virtual*. Más tarde, cuando la instrucción completa su etapa de ejecución, se le asigna un registro físico para almacenar su resultado. Para finalizar y cuando la instrucción alcanza la etapa de *commit*, se liberan tanto el registro físico como el registro vp asignados a la instrucción previa con el mismo registro lógico destino.

El concepto de los registros vp se presentó por primera vez en [GVG+97] y en [GGV98]. A continuación, se va a completar la descripción de los registros virtuales-físicos tal y como se hizo en los artículos citados. Para ello se van a presentar los distintos elementos *hardware* que son necesarios para utilizar este nuevo tipo de registros, así como su funcionamiento.

### 3.1.1 Esquema *hardware* y funcionalidad

En primer lugar, la técnica de los registros vp se puede aplicar tanto para registros de tipo enteros como para registros de coma flotante. Por eso, la implementación que se va a describir a continuación debe replicarse para los dos tipos de bancos de registros.

Las dos primeras tablas que se necesitan para la implementación de los registros vp se denominan *Tabla de Mapeo General* (*GMT*) y *Tabla de Mapeo Físico* (*PMT*) (ver la Figura 3.1.a). La primera de ellas (GMT) es una extensión de la MT utilizada en el mecanismo de renombre convencional. Al igual que la MT, la GMT se indexa por número de registro lógico y contiene los siguientes tres campos:

- VP: último registro virtual-físico asignado al registro lógico.

- P: último registro físico (si lo hay) asignado al registro lógico y a su correspondiente registro virtual-físico.

- V: bit que indica si el campo P contiene un valor válido, es decir, si ese registro lógico ya tiene asignado un registro físico.

La segunda de las tablas (PMT) posee una entrada por cada registro virtual-físico y contiene el último registro físico asignado a ese registro vp (ver la Figura 3.1.a). Se denomina *NVR* al número de registros vp de que dispone el procesador. Para garantizar que el procesador nunca va a parar por falta de registros virtuales-físicos, el parámetro NVR debe ser igual al número de registros lógicos (*NLR*) más el número de entradas en el ROS.

Finalmente, al igual que en el esquema de renombre convencional, existe un conjunto de registros físicos libres (la *Free List*) y otro de registros virtual-físicos también libres.

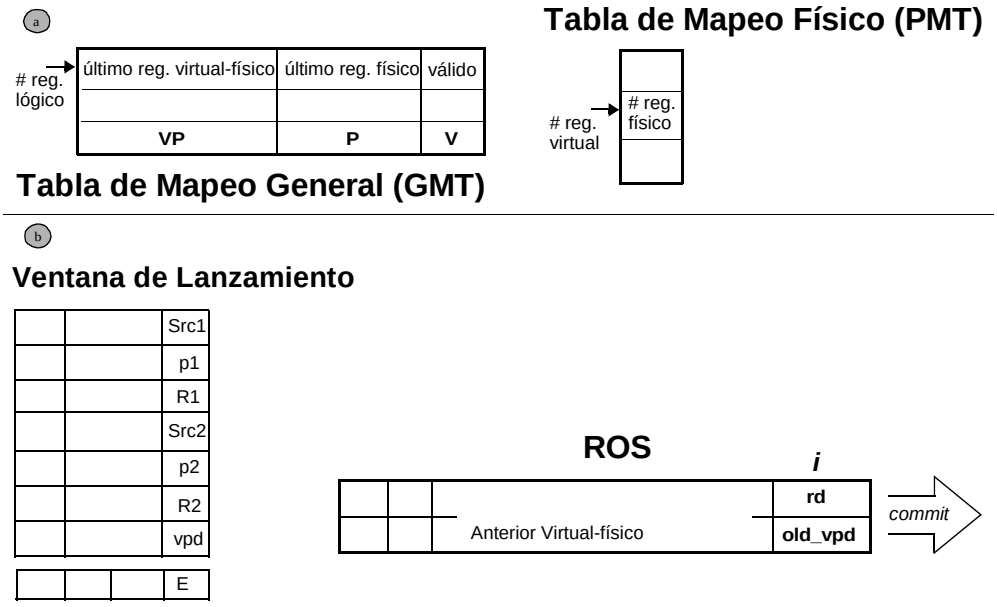


**Figura 3.1** Tablas que se necesitan para la técnica de registros virtuales-fisicos **(a)**. Ventana de lanzamiento y ROS **(b)**.

Cada vez que una instrucción se decodifica la tabla GMT se consulta para cada uno de sus registros fuentes. Si el bit V se encuentra a uno, el correspondiente registro lógico se renombra al registro físico especificado en el campo P; en caso contrario se renombra al registro virtual-físico del campo VP. Si la instrucción tiene registro lógico destino, éste se renombra a un registro virtual-físico de los que están libres. Su correspondiente entrada en la GMT se actualiza de la siguiente forma: el campo del registro VP se modifica para reflejar el nuevo mapeo y el campo V se pone a cero. El anterior valor almacenado en el campo VP se guarda junto con la instrucción en su correspondiente entrada del ROS (campo old_vpd en la Figura 3.1.b). Este valor servirá para restaurar el estado preciso del procesador en caso de que ocurra una predicción incorrecta de un salto o una interrupción. A continuación, la instrucción se coloca en la ventana de lanza-

miento, donde permanece hasta que comienza su etapa de ejecución y en el ROS, donde permanece hasta que alcanza la etapa de *commit*.

Cada entrada en el ROS posee los siguientes campos (consultar la Figura 3.1.b):

- rd: identificador del registro lógico destino.
- old_vpd: campo que identifica al registro virtual-físico asignado a la anterior instrucción que tenía el mismo registro lógico destino.

Cada entrada en la ventana de lanzamiento posee ahora los siguientes campos (consultar la Figura 3.1.b):

- Src1 y Src2: identificadores de los registros virtuales-físicos de los dos operandos fuente.
- p1 y p2: identificadores de los registros físicos de los dos operandos fuente. Estos no son campos añadidos por el mecanismo, forman parte de la propia ventana de lanzamiento del procesador [Yea96][Kes99].
- R1 y R2: bits de preparado de los operandos fuente. Cuando un operando se encuentra preparado, los campos Srci y pi contienen el identificador de un registro virtual-físico y el identificador de un registro físico respectivamente.
- vpd: registro virtual-físico que el renombre ha asignado al registro lógico destino. Se utiliza en la asignación de registro físico al final de la etapa de ejecución.
- E: bit que indica si la instrucción ya ha completado su ejecución o no.

Una instrucción puede lanzarse a ejecutar cuando los campos R de sus dos operandos se encuentran a uno. Esto también garantiza que los campos Srci y pi contienen los identificadores de los registros virtuales-físicos y físicos correctos. Cuando una instrucción comienza su etapa de ejecución, lee sus registros operandos desde el banco de registros físico utilizando los identificadores pi de su correspondiente entrada en la ventana, a no ser que el operando venga desde la salida de una unidad funcional. Para implementar el mecanismo vp y una variante suya que veremos en la sección 3.2 -vp-NRR- bastaría con utilizar Src1 y Src2 como almacenes de identificadores de registros virtuales-físicos o de registros físicos. Con esto, p1 y p2 no serían necesarios y un bit a uno en R1 ó R2 indicaría que el campo Sri contiene el idenficador del correspondiente registro físico. Sin embargo, todos estos campos en la IW sí serán necesarios en el mecanismo vp-DSY que se explica en la sección 3.3 y por esta razón han sido incluidos ya en este punto.

Una instrucción con registro destino pide un registro físico nuevo cuando finaliza su ejecución. En este momento se toma un registro físico p de la *Free List*. Para simplificar la explicación, se va a suponer que existen registros físicos libres para asignar. En caso contrario, surge el problema del *abrazo mortal* que se trata en la sección 3.1.3. A continuación, la tabla PMT se actualiza para reflejar la nueva asignación del registro físico p al registro virtual-físico destino de la instrucción (campo vpd en la ventana). Ese par de identificadores de registros (<vpd, p>) se emite a todas las entradas de la ventana de lanzamiento. Si existe coincidencia con algún campo Srci

cuyo bit R no tenga un valor de uno, el correspondiente campo pi se actualiza con el registro físico p y el bit R se pone a uno. El par <vpd, p> también se utiliza para actualizar la tabla GMT. La entrada correspondiente al registro lógico destino se consulta para comprobar si el campo VP coincide con el identificador del registro virtual-físico vpd. En tal caso, el identificador del registro físico p se copia en el campo P y el *flag* V se pone a uno. De esta forma, cualquier nueva instrucción que lea ese registro lógico, encontrará en la GMT el identificador correcto del registro físico que tiene asignado. Finalmente, se pone a uno el *flag* de ejecución E para la instrucción.

Cuando una instrucción alcanza la etapa de *commit*, se libera el registro virtual-físico asignado a la anterior instrucción con el mismo registro lógico destino. En este momento es seguro que dicha etiqueta no va a utilizarse más. Este registro virtual-físico está identificado en el campo old_vpd del ROS. También se libera el registro físico que tiene asociado. El identificador del registro físico se obtiene a través de la tabla PMT al indexarla con old_vpd.

Al igual que en el mecanismo de renombre convencional, en caso de que un salto se haya predicho mal o de que ocurra una excepción, se puede recuperar el estado preciso del procesador restaurando la copia correcta de la GMT o utilizando la IOMT respectivamente. Otra posibilidad consiste en deshacer todas las asignaciones de registros realizadas por las instrucciones que siguen a la instrucción que provocó la infracción. Esta recuperación del estado preciso se consigue eliminando del ROS aquellas entradas que van desde la más nueva hasta la que provocó la infracción. Para cada instrucción, el ROS almacena el registro lógico destino rd y el registro virtual-físico previo al cual fue asignado (old_vpd). Utilizando el identificador del registro lógico, se accede a la tabla GMT para obtener el actual registro virtual-físico. Si el *flag* V de esa entrada en la GMT se encuentra a uno, se obtiene también la actual asignación de registro físico. Los dos registros: el virtual-físico y el físico (si ya está asignado) vuelven a sus correspondientes listas de libres y el campo VP de la GMT se restaura con el campo old_vpd del ROS. Si además old_vpd ya está asignado a un registro físico este también se coloca el campo P de la GMT y el *flag* V se pone a uno (ese mapeo físico se obtiene de la tabla PMT); en caso contrario, V se pone a cero.

### 3.1.2 Otras alternativas del diseño

Una de las desventajas que posee la organización virtuales-físicos que se acaba de describir es en la necesidad de reejecutar las instrucciones que no encuentran registro físico libre al finalizar su ejecución.

Con el objetivo de eliminar tal desventaja, se investigaron soluciones alternativas basadas en asignar los registros físicos cuando las instrucciones se lanzan a ejecutar, en lugar de hacerlo cuando finalizan su ejecución. Este trabajo dio origen a la publicación del *Report* interno del DIIS [MGV+99b]. En estos esquemas, una instrucción con registro destino que esté preparada para ejecutar y que tenga disponible unidad funcional, solamente se lanzará si existe un registro físico disponible. Al comparar con el mecanismo de renombre convencional, esta nueva aproximación reduce la presión sobre el banco de registros, aunque en menor medida que un esquema basado en la asignación de registros al finalizar la ejecución.

Siguiendo en la línea de los registros virtuales-físicos, se evaluaron tanto la anterior estrategia (denominada vp-lanzar) como un par de estrategias híbridas más que combinan las dos anteriores. Las estrategias híbridas se basan en asignar registros a instrucciones de latencia larga cuando finalizan su ejecución. Para el resto de instrucciones, los registros se asignan cuando se lanzan a ejecutar. El primer diseño híbrido evaluado considera como instrucciones de latencia larga solo a *loads* (con posible fallo en memoria *cache*). El segundo diseño híbrido considera como instrucciones de latencia larga tanto a los *loads* como a las instrucciones de división. Con cualquier estrategia híbrida, el número de reejecuciones disminuye a cambio de una ligera disminución en el número de instrucciones que hacen el *commit* por ciclo (IPC). En la mayoría de los programas de prueba utilizados una estrategia híbrida supera en prestaciones a una estrategia que asigna los registros al lanzar a ejecutar y se acerca a las prestaciones que obtiene la estrategia original de virtuales-físicos.

### 3.1.3 El problema del *abrazo mortal*

Una organización de los registros virtuales-físicos puede diseñarse con cualquier número de registros lógicos, físicos y virtuales-físicos. Recordemos que se ha denominado NLR al número de registros lógicos de que dispone el procesador, NVR al número de registros virtuales-físicos y que *NPR* es el número de registros físicos. El NLR es una característica intrínseca del ISA y por lo tanto su valor permanece fijo para distintas implementaciones de la misma ISA. Por otro lado y debido a que los registros virtuales-físicos no son más que etiquetas, un procesador puede disponer de la cantidad de ellas que necesite. El máximo número de estas etiquetas coincide con el NLR más el máximo número de instrucciones en vuelo que es capaz de soportar el procesador. Sin embargo, el NPR sí que tiene un impacto muy importante en el coste *hardware* del procesador y posiblemente en su tiempo de ciclo. Para que el banco de registros físico mantenga un tiempo de acceso pequeño debe ser dimensionado con el menor tamaño que a su vez permita obtener un rendimiento que no quede muy por debajo del obtenido con un número ilimitado de registros. Ocurre así que NPR va a ser menor que NVR. De esta forma, no va a estar garantizado que toda instrucción encuentre un registro físico libre cuando acaba su etapa de ejecución.

Un esquema de renombre de registros convencional puede definirse como un esquema que asigna los registros físicos disponibles en el orden secuencial del programa y que fuerza al procesador a parar la etapa de decodificación de las instrucciones cuando ya no dispone de más registros físicos. Un esquema de renombre de registros virtuales-físicos añade al procesador la ventaja de que no necesita parar la etapa de decodificación aunque todos los registros físicos se encuentren en uso. Sin embargo, con el esquema de registros virtuales-físicos puede ocurrir que cuando una instrucción finaliza su ejecución encuentre que ya no hay registros físicos libres. La solución más sencilla que se puede adoptar al alcanzar esta situación consiste en mantener dicha instrucción en la ventana de lanzamiento para ejecutarla más tarde (hay que *reejecutar* la instrucción). Esta solución se basa en la suposición de que en un tiempo no muy lejano, algún registro físico quedará libre (alguna instrucción hará el *commit*). Pero también puede ocurrir que la instrucción que no encuentra registro al finalizar su ejecución sea la más vieja de todas las que residen en el ROS (la que va a la cabeza del ROS). En este caso, esta instrucción al no poder completar su ejecución tampoco podrá alcanzar su etapa de *commit*. Bajo estas circunstancias, ninguna instrucción podrá alcanzar su etapa de *commit* y por lo tanto tampoco ningún registro

físico se liberará. Es posible así que el procesador alcance una situación de bloqueo que se va a denominar *situación de abrazo mortal*.

Para resolver este problema hace falta asegurar que toda instrucción en el procesador va a poder alcanzar su etapa de *commit*. De esta forma, las instrucciones con registro destino podrán ir liberando los registros físicos. Con este objetivo, se han propuesto dos políticas de asignación de los registros físicos que van a ser detalladas a continuación.

## 3.2 Reserva de registros para las instrucciones más viejas: vp-NRR

La primera solución que se propuso para evitar el *abrazo mortal* dio origen a las publicaciones de [GVG+97], [GGV98] y [MGV+00a]. Esta solución consiste en modificar ligeramente la política original de gestión de los registros virtuales-físicos. Es suficiente con garantizar que un número dado de las instrucciones más viejas en el ROS que necesitan de registro destino tengan a su disposición registro físico para completar el renombre. A este parámetro se le denominó *Número de Registros Reservados* (*NRR*) y en el caso general puede ser diferente para registros de tipo entero ($NRR_{ent}$) o para registros fp ($NRR_{fp}$). Utilizando este parámetro, las NRR instrucciones más viejas que tienen registro destino y todas aquellas instrucciones intermedias que no tengan registro destino tienen garantizado el poder alcanzar su etapa de *commit*. Ya que cada instrucción que consume un registro libera otro cuando alcanza su etapa de *commit*, las siguientes NRR instrucciones más viejas que tienen registro destino y todas aquellas intermedias que no tengan registro destino vuelven a tener garantizado que van a llegar a su etapa de *commit*. Repitiendo el mismo razonamiento, se puede concluir que todas las instrucciones van a tener garantizado alcanzar su etapa de *commit* y por lo tanto que nunca va a ocurrir una situación de *abrazo mortal*.

A partir de este momento se va a llamar vp-NRR a esta nueva política de asignación de registros virtuales-físicos. A continuación se va a detallar su implementación.

### 3.2.1 Implementación de vp-NRR

La técnica vp-NRR añade dos punteros en el ROS a la implementación del mecanismo de los registros virtuales-físicos, un puntero para las instrucciones de tipo entero y otro para las de punto flotante. Estos punteros, denominados $PRR_{ent}$ y $PRR_{fp}$, van a identificar a las NRR instrucciones enteras o fp más viejas que tienen garantizado registro destino. Serán instrucciones cuya posición en el ROS se encuentre entre el puntero PRR y la cabeza del ROS. Existen además dos contadores que indican el número de esas instrucciones que poseen registro destino (entero o fp) y otros dos contadores que indican el número de aquéllas que ya tienen asignado registro físico. A todos esos contadores los denominamos $Reg_{ent}$, $Reg_{fp}$, $Used_{ent}$ y $Used_{fp}$ respectivamente.

La parte de la gestión de los punteros y contadores del mecanismo vp-NRR se va a aplicar sólo a instrucciones de tipo entero para simplificar su explicación.

Cada vez que una instrucción con registro destino de tipo entero alcanza la etapa de *commit*, $PRR_{ent}$ se mueve para apuntar a la siguiente instrucción con registro destino y de tipo entero. Si la nueva instrucción no tiene todavía asignado su registro físico, el puntero $Used_{ent}$ se decre-

menta; en caso contrario $Used_{ent}$ no se modifica. Si con el movimiento de $PRR_{ent}$ se alcanza la cabeza del ROS (recordar que se gestiona como un *buffer* circular) sin encontrar ninguna instrucción con registro destino de tipo entero, entonces tanto $Reg_{ent}$ como $Used_{ent}$ se decrementan.

Cuando se decodifica una nueva instrucción con registro destino, si $Reg_{ent}$ es menor que el parámetro $NRR_{ent}$ entonces $Reg_{ent}$ se incrementa y $PRR_{ent}$ se mueve para apuntar a esa instrucción. En caso contrario, no se hace nada con los punteros ni con los contadores.

Cuando una instrucción finaliza su ejecución se le asigna un nuevo registro físico en la forma que se ha descrito en la sección 3.1.1 si se cumple una de las dos condiciones siguientes:

- es una instrucción de las NRR que tienen reservado registro (está entre la cabeza del ROS y el puntero $PRR_{ent}$).
- hay más registros físicos disponibles que el valor de: $NRR_{ent}$ - $Used_{ent}$.

En caso contrario, la instrucción debe anularse y enviarse de nuevo a la ventana de lanzamiento.

El parámetro NRR puede tomar cualquier valor entre un mínimo de uno y un máximo igual al número de registros físicos menos el número de registros lógicos. Es difícil anticipar cuál es el mejor valor sin hacer ningún tipo de evaluación experimental. Cuando el procesador se queda sin registros físicos, la ejecución puede continuar pero utilizando sólo los NRR registros que se reservaron para las instrucciones más viejas. En tal caso, aunque existan instrucciones más jóvenes que ya han finalizado su ejecución y ya se les ha asignado un registro físico, no liberarán ningún registro hasta que ellas mismas y todas las instrucciones que las preceden alcancen la etapa de *commit*. El procesador pasa entonces a un modo de ejecución donde sólo pueden acabar su ejecución las NRR instrucciones más viejas. Este modo de ejecución está también limitado por la velocidad de *commit* de esas mismas NRR instrucciones, al ser ellas la única fuente proveedora de registros. Se puede decir que es un modo de ejecución secuencializado por el valor de NRR.

Elegir un valor bajo para NRR significa que el procesador posee de entrada más registros para asignar bajo demanda de ejecución, lo cual favorece una ejecución fuera de orden más agresiva lo que a su vez supone adelantar trabajo futuro. Sin embargo, cuando el procesador alcanza el modo de ejecución secuencializado por NRR, valores muy bajos de NRR serán negativos.

Elegir el valor máximo para NRR (igual a los registros de renombre), va a significar obtener un rendimiento al menos tan bueno como el que se obtiene con el esquema convencional de renombre. En esta situación y al igual que hace el mecanismo convencional, el mecanismo vp-NRR asigna siempre a las instrucciones más viejas todos los registros físicos disponibles. Sin embargo, este nuevo esquema de renombre posee importantes ventajas adicionales. En primer lugar, aún en el caso de que el procesador no disponga de registros físicos de un tipo dado (enteros o fp) va a poder continuar ejecutando instrucciones, mientras que en el esquema convencional el procesador hubiera parado. En segundo lugar, y para un tipo de instrucciones dado, con el esquema convencional el procesador no podría iniciar ninguna instrucción con registro destino que no fuera de las NRR instrucciones más viejas. Sin embargo, en una organización vp-NRR al procesador *sí* se le permite iniciar la búsqueda, decodificación y ejecución de nuevas instruccio-

nes. En realidad, estas dos ventajas son posibles gracias a la utilización de las etiquetas virtuales en la etapa de decodificación de las instrucciones y por lo tanto son efectivas para cualquier valor de NRR.

Resumiendo, con valores bajos de NRR se pueden ejecutar muchas más instrucciones de las ubicadas más allá del punto actual de *commit* y mucho más pronto de lo que se ejecutarían teniendo un valor alto de NRR. Sin embargo, cuando las instrucciones más viejas empiezan a ejecutarse sin disponer de registros físicos, el primer esquema posee un rendimiento muy bajo debido a que las instrucciones se ejecutan de forma secuencializada. De hecho, los dos trabajos referidos en la introducción [WB96] y [Son97] se pueden ver como representantes de los dos puntos extremos de los valores que toma NRR. El esquema propuesto en [WB96] es muy parecido al mecanismo vp-NRR con NRR igual a uno y el de [Son97] a vp-NRR con NRR igual al número de registros de renombre. Para el mecanismo vp-NRR, en el apartado de resultados se verá como la evaluación experimental muestra que el valor óptimo de NRR puede variar para los diferentes programas de prueba. Incluso para un mismo programa, puede ser beneficioso variar el valor de NRR en distintas secciones del código.

Es así crítico el disponer de un mecanismo que de forma dinámica pueda ajustar el valor del parámetro NRR. Esto último motivó la propuesta de la segunda política de asignación de los registros físicos.

## 3.3 Robo de registros a las instrucciones más jóvenes: vp-DSY

La anterior técnica vp-NRR no es la única solución que evita el problema del *abrazo mortal*. La siguiente solución que se propuso dio origen a las publicaciones de [MGV+99a] y [MGV+00a]. Encontrar la política de asignación de registros óptima parece un problema no resoluble incluso disponiendo de un conocimiento perfecto de todas las referencias futuras que se harán de los registros. Por eso hay que utilizar una serie de heurísticas que intenten aproximarse al máximo a un esquema óptimo. La solución propuesta se basa en las siguientes dos reglas:

1. Los registros deben asignarse a aquellas instrucciones que van a utilizarlos más pronto. De este modo, se minimizará el número medio de registros que no están siendo utilizados.
2. Dadas dos instrucciones cualesquiera, si la ejecución de una de ellas debe retrasarse por falta de registros, la candidata más apropiada es la instrucción más joven de las dos. Retrasar la ejecución de la instrucción más vieja podría retrasar a su vez su etapa de *commit*, lo cual supondría retrasar también la etapa de *commit* de la instrucción más joven.

Estos dos criterios se van a cumplir en el siguiente esquema. Siempre que existan registros físicos disponibles, a cada instrucción se le va a asignar uno en el último ciclo de su etapa de ejecución. Con esto se cumple el primero de los criterios ya que los registros siempre se asignarán a aquellas instrucciones que primero finalizan su ejecución. Si una instrucción alcanza el último ciclo de su etapa de ejecución y no hay ningún registro físico disponible, se busca si existe alguna instrucción más joven que ya tenga asignado registro físico. Si se encuentra alguna, basándonos en el segundo de los criterios, va a ser mejor parar la instrucción más joven que parar la instrucción más vieja. Como se sugiere en [JRB+98] esto se puede conseguir robando a la instrucción más joven el registro que tiene asignado para asignárselo a la instrucción más vieja. En

el caso de que exista más de una instrucción más joven con registro físico asignado, se elige a la más joven de todas. A esta nueva política de asignación de registros se le va a llamar vp-DSY, siglas que vienen del inglés *on-Demand allocation with Stealing from Younger*. Resumiendo, esta política de asignación de registros virtuales-físicos consiste en asignar los registros bajo demanda y posiblemente robarlos a instrucciones más jóvenes. A continuación se va a detallar su implementación.

### 3.3.1 Implementación de vp-DSY

La técnica vp-DSY añade a la implementación del mecanismo de los registros virtuales-físicos la necesidad de identificar si existe alguna instrucción entre las más jóvenes que ya tenga asignado registro físico. Para hacer esto basta con buscar en la ventana de lanzamiento aquella entrada más joven que tenga el bit E a uno, indicando que la instrucción ya ha completado su ejecución. Si se encuentran varias, se selecciona la más joven de todas. Se va a llamar *i1* a la instrucción que pide un registro e *i2* a la instrucción a la que se roba el registro (*S* en el código de la Figura 3.2). Sean vpd1 y vpd2 los registros virtuales-físicos destino de las instrucciones *i1* e *i2* respectivamente y p2 el registro físico destino de la instrucción *i2*. La Figura 3.2.a muestra el contenido de las tablas GMT, PMT y de la IW tras la ejecución de *i2*. Más tarde, la instrucción *i1* le roba el registro físico a la instrucción *i2*. El identificador del registro virtual-físico que tiene asignado la instrucción *i2* se obtiene de la ventana de lanzamiento en su campo vpd -vpd2-y el identificador del registro físico -p2- se obtiene de la tabla PMT. La tabla PMT y la tabla GMT se actualizan para reflejar que ahora vpd1 está asignado a p2 y que vpd2 ya no está asignado a p2 (consultar la Figura 3.2.b).

La instrucción *i2* a la que se roba el registro debe reejecutarse de nuevo. Una solución simple consiste en mantener la instrucción en la ventana de lanzamiento hasta que alcance su etapa de *commit* y con un bit que indica si ha sido lanzada a ejecutar o no. Al marcar la instrucción *i2* como no lanzada (E = cero en la IW en Figura 3.2.b), la lógica de lanzamiento de las instrucciones la volverá a seleccionar para lanzarla de nuevo.

Como esa instrucción *i2* ya fue lanzada a ejecutar, los consumidores del registro vpd2 marcaron alguno de sus operandos como preparado (*i3* en la Figura 3.2.a). Sin embargo, en el momento en que el registro físico se roba, dicho operando deja de estar preparado y debe volver al estado de no preparado (Figura 3.2.b). Para ello se pueden utilizar los mismos *buses* que se usan en la ventana para despertar a las instrucciones de la forma que se describe a continuación.

En un procesador convencional, el identificador del registro físico destino de una instrucción se emite a todas las entradas de la ventana de lanzamiento cuando la instrucción finaliza su ejecución[1]. Cada entrada verifica si alguno de los identificadores de sus operandos fuente coincide con el emitido y se marcan como preparados los casos donde exista coincidencia. Si se utilizan registros virtuales-físicos, cada entrada en la ventana identifica además a todos los operandos fuente con el identificador que se guarda en los campos Srci. Cuando una instrucción finaliza, tanto el identificador del registro virtual-físico como el identificador del registro físico son emi-

1. En realidad, esto se hace algunos ciclos antes para poder solapar la latencia de la etapa de lanzamiento y la lógica de lectura con los últimos ciclos de la ejecución.

tidos hacia la ventana de lanzamiento. Cada entrada compara los identificadores de todos sus operandos fuente con el virtual-físico emitido y en caso de que exista coincidencia, el identificador físico emitido se copia en el campo correspondiente y se pone a uno su bit de preparado. En el caso de que el registro físico destino de una instrucción se robe, su correspondiente identificador virtual-físico (vpd2 en el ejemplo) se emite a la ventana para marcar como no preparado a todo operando fuente donde exista coincidencia (Figura 3.2.b).

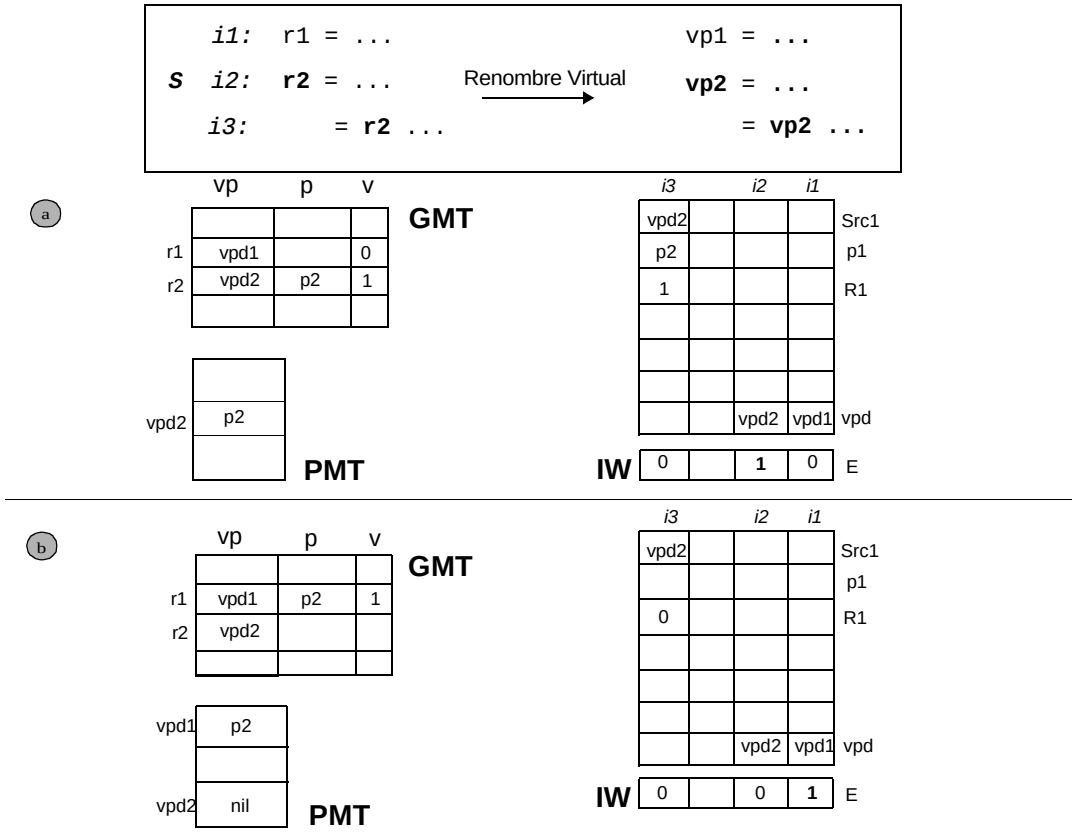


**Figura 3.2** Ejemplo del esquema de renombre vp-DSY.

Notar que algunas instrucciones que tienen vpd2 como operando fuente pueden haber sido lanzadas a ejecutar y más adelante este mismo operando se marca como no preparado. Estas instrucciones leyeron un valor correcto para su operando fuente y por lo tanto los resultados que generan son correctos. Cuando una instrucción de éstas acaba su ejecución, si hay registros físicos disponibles, podrá almacenar en uno de ellos su resultado y las instrucciones dependientes de ésta también podrán ser lanzadas a ejecutar. Sin embargo, cuando la instrucción *i2* se ejecuta de nuevo se le asigna un nuevo registro físico para su registro destino, (sea p3). En ese momento, los identificadores de vpd2 y p3 se han de emitir a la ventana de lanzamiento y aquellas instrucciones que consumen vpd2 copiarán el nuevo mapeo físico y se volverán a lanzar. Lo mismo ocurre para las instrucciones consumidoras de estas instrucciones que han sido lanzadas de nuevo: volverán a ser lanzadas sin importar si ya fueron lanzadas anteriormente.

### 3.3.2 Beneficios implícitos del esquema vp-DSY

Como se acaba de describir, el esquema de renombre vp-DSY puede generar la ejecución múltiple de algunas instrucciones, aunque las reejecuciones de una misma instrucción vayan a producir el mismo resultado. En este apartado se va a ver cómo estas ejecuciones múltiples tienen un efecto implícito muy positivo tanto en el mecanismo de especulación como en la memoria *cache* de datos.

Excepto la última ejecución, el resto de ocasiones en que se ejecuta una misma instrucción puede verse como una *ejecución anticipada* que no hubiese ocurrido si el procesador hubiera asignado los registros físicos desde las instrucciones más viejas hacia las instrucciones más jóvenes. Recordar que el registro físico de una instrucción *i* se roba sólo cuando en la ventana de lanzamiento hay instrucciones más viejas que precisan de registro físico. Un mecanismo de renombre convencional hubiera asignado todos los registros físicos a las instrucciones más viejas y la instrucción *i* ni siquiera habría sido buscada. Además, con los registros virtuales-físicos esta instrucción *i* va a disponer de registro físico si finaliza su ejecución antes que alguna instrucción más vieja que ella. Sin embargo, cuando una instrucción más vieja finaliza su ejecución y no encuentra registro físico disponible, roba el registro a la instrucción *i* provocando su posterior reejecución.

La ejecución anticipada de cualquier instrucción tiene beneficios importantes que son más acusados para instrucciones de salto e instrucciones *load*:

- La ejecución anticipada de una instrucción de salto valida su predicción con anticipación. Para el caso de que se haya realizado una predicción incorrecta, las acciones que se deben tomar para recuperarse del salto (neutralizar las instrucciones del camino incorrecto y continuar buscando las instrucciones del camino correcto) van a comenzar también con anticipación. En el esquema de renombre convencional, esa validación del salto ocurre mucho más tarde debido a que la instrucción se retrasa al no existir registros disponibles.

- La ejecución anticipada de una instrucción *load* que falla en *cache* lanza la búsqueda de los datos con anticipación. Cuando la instrucción se reejecuta esos datos pueden encontrarse ya en la memoria *cache* y por lo tanto convertirse en un acierto. En otras palabras, la ejecución anticipada de instrucciones de memoria se comporta como un esquema de *prebúsqueda* de datos y por lo tanto oculta la latencia de memoria de algunos fallos de *cache*.

En el apartado de resultados se analizará más en detalle el efecto de adelantar la ejecución de las instrucciones. A continuación, se presenta el entorno experimental utilizado para el la obtención de todos los resultados.

## 3.4 Entorno experimental

Toda la experimentación de esta Tesis se ha realizado utilizando un simulador ciclo a ciclo de un procesador superescalar. Para ello se ha tomado como base la herramienta de simulación SimpleScalar v3.0 [BA97]. Se ha extraído de esta herramienta su simulador fuera de orden (*sim-outorder*) y se ha modificado para incluirle un banco de registros físico mezclado (entero y fp). Los resultados de todas las instrucciones se guardan ahora en ese banco de registros en lugar de

guardarlos temporalmente en la *RUU* [Soh90] para moverlos al banco de registros en la etapa de *commit*. Esta implementación se corresponde con la utilizada en algunos procesadores actuales como el MIPS R10K [Yea96] y el Alpha 21264 [Kes99].

Al simulador inicial se le ha añadido el mecanismo de renombre de registros convencional que se detalla en el Capítulo 2. Este simulador representa al procesador de base que se ha utilizado y por esa razón se le ha denominado *simulador del procesador convencional*. Posteriormente el simulador convencional se ha ido extendiendo para incluir los distintos mecanismos de renombre propuestos en esta Tesis. En concreto, se ha desarrollado un simulador para cada uno de los mecanismos propuestos manteniendo los mismos parámetros de la arquitectura.

| **Parámetro** | **Valor** |
|---|---|
| anchura del *Fetch* | 8 instrucciones (hasta 2 saltos tomados) |
| *cache* ins. L1 | 32 KB, 2-*way set-assoc*, líneas de 32 *Bytes*, 1 ciclo de tiempo de acierto |
| Predicción de saltos | predictor Gshare de 18 bits con actualizaciones especuladas, hasta 20 saltos pdtes. |
| tamaño vent. lanzam. (*N*) | 128 entradas |
| Unidades Func. (*latencia*) | 8 simple ent (*1*), 4 mult ent (*7*), 4 *load*/*store*, 6 simple fp (*4*), 4 mult fp (*4*), 4 div fp (*16*) |
| Cola de *Load*/*Store* | 64 entradas con *store-load forwarding* |
| mecanismo de lanzam. | fuera de orden. Los *loads* se ejecutan cuando son conocidas las @'s de todos los *stores* previos |
| Registros Físicos (*P*) | 40-160 ent / 40-160 fp (32 ent / 32 fp lógicos) |
| *cache* datos L1 | 32 KB, 2-*way set-assoc*, líneas de 64 *Bytes*, 1 ciclo de tiempo de acierto |
| *cache* unificada L2 | 1 MB, 2-*way set-assoc*, líneas de 64 *Bytes*, 12 ciclos de tiempo de acierto |
| Memoria Principal | tamaño ilimitado, 50 ciclos de tiempo de acceso |
| anchura del *Commit* | 8 instrucciones |

**Tabla 3.1** Parámetros de la microarquitectura simulada.

El simulador convencional puede hacer la búsqueda de hasta ocho instrucciones consecutivas cada ciclo. La predicción de saltos se implementa utilizando un predictor *Gshare* con 18 bits de historia que se actualiza de forma especulada en la etapa de búsqueda, un *BTB* con 1024 conjuntos y asociatividad 4 y un *RAS* de 64 entradas. Se utilizan un ROS y una ventana de lanzamiento con 128 entradas cada una. Las instrucciones de memoria (*load*/*store*) se encuentran separadas en una *LSQ* (*Load Store Queue*) de 64 entradas. Existe un banco de registros físico para datos de tipo entero y otro para los de tipo fp. El número de registros físicos tanto entero como fp, se ha variado desde un mínimo de 40 registros físicos hasta un máximo de 160. En la Tabla 3.1 se muestran los principales parámetros de la microarquitectura simulada.

Se han utilizado diez programas de prueba del conjunto Spec95 [SPEC]: cinco programas de tipo entero (*compress*, *gcc*, *go*, *li* y *perl*) y otros cinco programas de tipo fp (*mgrid*, *tomcatv*, *applu*, *swim* e *hydro2d*). Estos diez programas fueron elegidos como representativos del conjunto Spec95 y se han ejecutado utilizando como entrada los ficheros *ref* con algunas modificaciones (consultar la Tabla 3.2). Todos han sido ejecutados completos excepto el programa *tomcatv* para el cual se han saltado las primeras instrucciones que se encargan de leer un fichero de entrada enorme. Todos los programas fueron compilados con los compiladores de *Compaq*/*Alpha*

*Fortran* y *C* de una DEC AlphaStation 600 5/266 con procesadores Alpha DEC 21164. En el proceso de compilación se usó el máximo nivel de optimización (*-O5* para *Fortran* y *-O4 -migrate* para *C*). En la Tabla 3.2 se listan los programas, las entradas utilizadas (incluye las modificaciones hechas a las entradas *ref*) y el número de instrucciones que hacen el *commit*.

| | Aplicación | Entradas | Instr. *commit.* (M) |
|---|---|---|---|
| **enteros** | *compress* | 40000 e 2231 | 170 |
| | *gcc* | genrecog.i | 145 |
| | *go* | 9 9 | 146 |
| | *li* | 7 queens | 243 |
| | *perl* | scrabbl.in | 47 |
| **fp** | *mgrid* | test (sustituyendo las dos primeras líneas por 5 y 18) | 169 |
| | *tomcatv* | test | 191 |
| | *applu* | train (cambiando dt=1.5e-03 y nx=ny=nz=13) | 398 |
| | *swim* | train | 431 |
| | *hydro2d* | test (sustituyendo ISTEP=1) | 472 |

**Tabla 3.2** Programas utilizados.

Los dos esquemas *hardware* que se acaban de presentar para el renombre y que utilizan el concepto de los registros virtuales-físicos (vp-NRR y vp-DSY) se han evaluado utilizando este entorno experimental. A continuación se van a ver los resultados más representativos obtenidos para cada uno de los mecanismos propuestos. Se verá que la estrategia vp-DSY es la que mejores resultados proporciona. En el siguiente apartado se hará un análisis del coste de implementación de las estructuras que son clave para soportar la estrategia vp-DSY.

## 3.5 Resultados: vp-NRR

En anteriores apartados se ha visto que para el mecanismo vp-NRR el valor óptimo del parámetro NRR puede variar dependiendo del programa que se ejecuta. Incluso para un mismo programa, variar el valor del parámetro NRR a través de diferentes secciones de la ejecución puede proporcionar beneficios significativos. De forma experimental, se ha observado cómo el rendimiento del procesador es muy sensible al valor que toma el parámetro NRR. De todas las configuraciones de tamaños de bancos de registros medidas, se ha seleccionado la de $64_{ent}+64_{fp}$ registros físicos ($32_{ent}+32_{fp}$ lógicos y $32_{ent}+32_{fp}$ de renombre) que es el tamaño implementado en los procesadores MIPS R10K/12K [Yea96][Gwe97]. Para este banco de registros se ha medido el IPC que se obtiene con el mecanismo de renombre convencional y con vp-NRR donde el parámetro NRR varía desde un valor mínimo de uno hasta el valor máximo de 32. En promedio, para todos los programas de prueba medidos (enteros y fp), se ha observado como vp-NRR siempre proporciona el mayor rendimiento para el valor máximo de NRR (32) y deja de ser efectivo para valores de NRR por debajo de 8 para programas enteros y de 16 para fp. Las úni-

cas excepciones observadas las presentan el programa *li*, con un valor óptimo de NRR de 16 y el programa *mgrid* para el que cualquier valor de NRR obtiene un rendimiento peor.

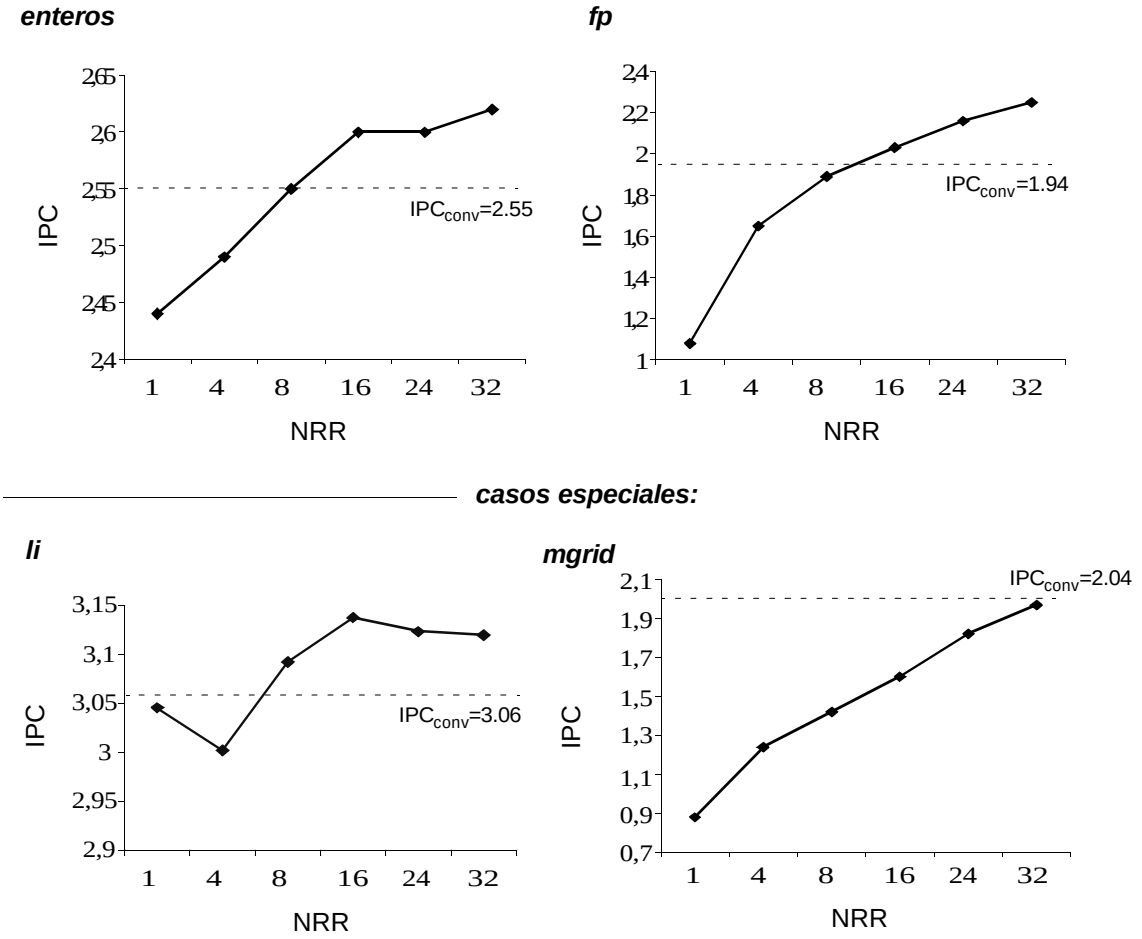


**Figura 3.3** Efecto al variar el parámetro NRR para los programas de prueba y 64ent+64fp registros. Casos especiales de *li* y *mgrid*.

La Figura 3.3 muestra la media armónica del IPC obtenido para los programas enteros y fp, y para los casos especiales de *li* y *mgrid*.

Por otro lado, una de las desventajas que posee vp-NRR está en la necesidad de repetir la etapa de ejecución de las instrucciones cuando no encuentran registro físico libre al finalizar su ejecución. De forma experimental, se ha observado cómo el promedio del número total de instrucciones reejecutadas disminuye al aumentar el valor del parámetro NRR. La Tabla 3.3 muestra el porcentaje de reducción del número total de instrucciones reejecutadas (enteras y fp) que se obtiene al comparar entre sí vp-NRR con distintos valores de NRR. Excepto algún caso particular, se observa que aumentar el número de registros reservados para asignar a las instrucciones cuando finalizan su ejecución es sustancialmente beneficioso para reducir el número de reejecuciones.

| | programas enteros | | | | |
|---|---|---|---|---|---|
| | NRR = 1 *vs.* NRR = 4 | NRR = 4 *vs.* NRR = 8 | NRR = 8 *vs.* NRR = 16 | NRR = 16 *vs.* NRR = 24 | NRR = 24 *vs.* NRR = 32 |
| reducción | *13%* | *20%* | *9%* | *-5%* | *37%* |

| | programas fp | | | | |
|---|---|---|---|---|---|
| | NRR = 1 *vs.* NRR = 4 | NRR = 4 *vs.* NRR = 8 | NRR = 8 *vs.* NRR = 16 | NRR = 16 *vs.* NRR = 24 | NRR = 24 *vs.* NRR = 32 |
| reducción | *43%* | *19%* | *11%* | *9%* | *10%* |

**Tabla 3.3** Porcentaje de reducción en el número total de reejecuciones obtenido con una estrategia vp-NRR al variar el valor del parámetro NRR. 64ent+64fp registros.

Estos dos resultados (aumento en rendimiento y disminución en el número de reejecuciones) justifican la utilización del valor máximo de NRR al comparar la estrategia vp-NRR con las siguientes soluciones de asignación que también utilizan los registros virtuales-físicos.

Con el objetivo de eliminar el factor de reejecución que sufre el mecanismo de registros virtuales-físicos, se investigaron soluciones alternativas a vp-NRR. Estas estrategias se basan en asignar los registros físicos cuando las instrucciones se lanzan a ejecutar y no cuando finalizan su ejecución. Se denominó vp-lanzar a la estrategia consistente en asignar registro en la etapa de lanzamiento. También se diseñaron dos estrategias híbridas: vp-*load* (asignación de registro al salir de la unidad funcional para *loads* y asignación en la etapa de lanzamiento para el resto) y vp-*load&div* (asignación de registro al salir de la unidad funcional para *loads* y divisiones y asignación en la etapa de lanzamiento para el resto). Las tres estrategias se diseñaron utilizando la solución NRR para el problema del *abrazo mortal* y se compararon con el mecanismo original vp-NRR (asignación de registro al salir de la unidad funcional para todas las instrucciones con registro destino). Todas ellas se evaluaron con el máximo valor para NRR y se compararon con la estrategia vp-NRR original también con valor máximo para NRR.

Después de experimentar con varias configuraciones de tamaño de banco de registros, se observó que el rendimiento obtenido por cualquiera de las nuevas estrategias siempre supera al obtenido por el mecanismo convencional, pero también que todas quedan por debajo del rendimiento que da vp-NRR. La Figura 3.4 muestra este resultado para la configuración de $64_{ent}$+$64_{fp}$ registros físicos. Las gráficas representan la media armónica de IPC para los dos conjuntos de programas de prueba y también el caso especial de *mgrid*. Una estrategia de virtuales-físicos que asigna siempre los registros al salir de la unidad funcional (vp-NRR original), lleva al límite el retraso en la asignación de registro físico para todo tipo de instrucción, lo cual

permite disponer de más registros durante más tiempo y por lo tanto aumentar el rendimiento. En el otro extremo está la estrategia vp-lanzar que es la menos agresiva en el retraso de la asignación de registros y por tanto obtiene un rendimiento menor. Es de esperar también que cualquier estrategia híbrida obtenga rendimientos intermedios entre esos dos extremos. En todo el conjunto de programas de prueba no se ha observado mejora por parte de las dos estrategias híbridas. En promedio, para los programas enteros los resultados de la estrategia vp-lanzar son peores que los de las estrategias híbridas y éstos a su vez no alcanzan a los de vp-NRR original. En los programas fp se obtienen diferencias mayores de rendimiento, ya que predominan instrucciones de latencia larga y por lo tanto es de esperar que la mejor estrategia sea retrasar la asignación de registros hasta la salida de la unidad funcional. En la Figura 3.4 se observa como al programa *mgrid* le afecta de forma negativa cualquier asignación de registros efectuada al finalizar su ejecución las instrucciones. Para *mgrid* de todas las estrategias que hacen uso de los registros virtuales-físicos, sólo vp-lanzar es efectiva.

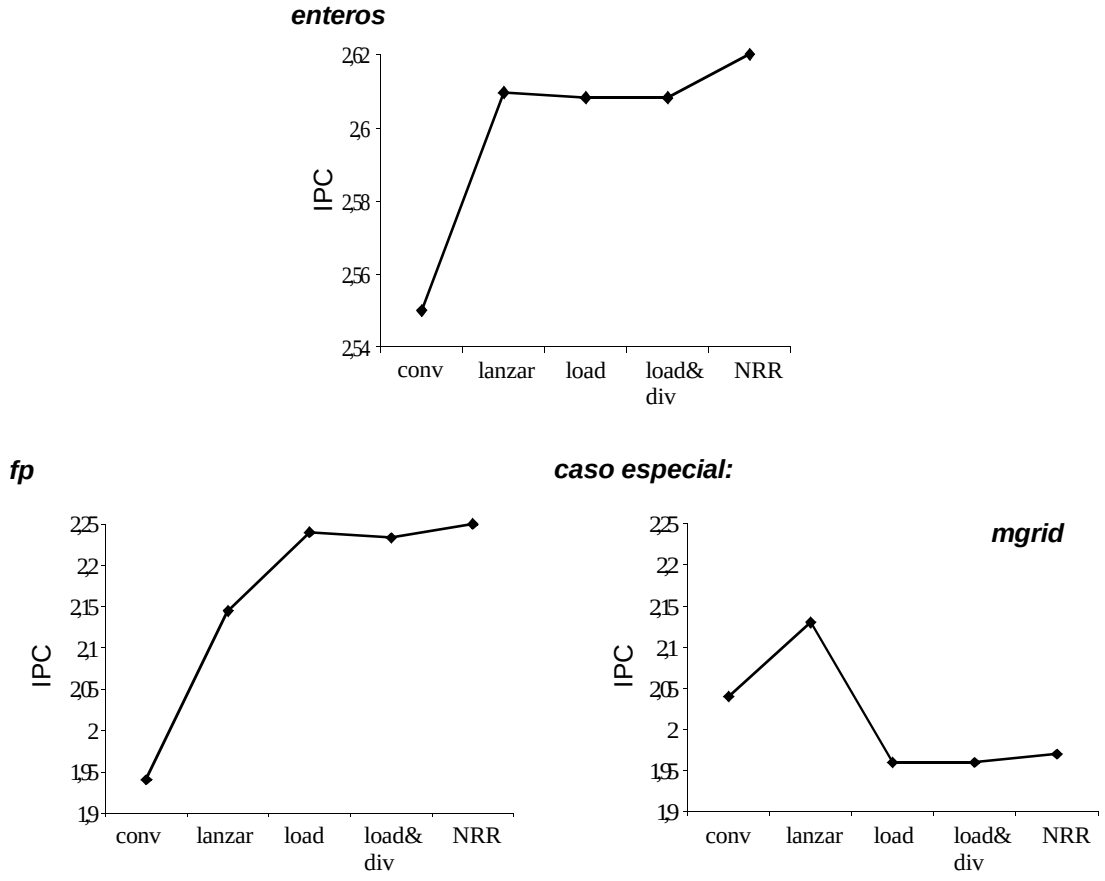


**Figura 3.4** Efecto en el rendimiento de las estrategias vp alternativas. 64int+64fp registros. Caso especial de *mgrid*.

Cuando la disponibilidad de registros empieza a ser crítica, una estrategia como vp-NRR debe reejecutar instrucciones que al salir de las unidades funcionales no encuentran registro físico disponible. Estas reejecuciones pueden ser muy graves cuando las instrucciones pertenecen al camino crítico. La estrategia vp-lanzar no produce reejecuciones, pero la etapa de lanzamiento

puede parar más veces por falta de registros físicos. De forma experimental se ha vuelto a observar como cualquiera de las estrategias híbridas provoca menos reejecuciones que vp-NRR y hace parar menos veces la etapa de lanzamiento que vp-lanzar. Por ejemplo, para $64_{ent}$+$64_{fp}$ registros físicos, con cualquier estrategia híbrida el promedio del número total de reejecuciones se reduce en un 74% y en un 37% para programas enteros y fp respectivamente. El número total de ocasiones que hay que parar la etapa de lanzamiento se reduce en promedio en un 23% y 84% (enteros y fp).

Uno de los objetivos prioritarios en esta Tesis era buscar estrategias que proporcionen el mayor rendimiento posible para los programas de prueba elegidos como representativos. Por esta razón, de las anteriores estrategias diseñadas sobre el mecanismo de registros virtuales-físicos, se optó por utilizar vp-NRR con NRR máximo para compararla con el siguiente mecanismo (vp-DSY) aún a expensas del problema de las reejecuciones.

## 3.6 Resultados: vp-NRR *versus* vp-DSY

El mecanismo vp-NRR no es la única solución que evita el problema del *abrazo mortal* presente con los registros virtuales-físicos. Hay que intentar acercarse lo máximo posible a una política de asignación óptima y utilizar para ello nuevas heurísticas de asignación. Con estas ideas en mente, se desarrolló el siguiente mecanismo de asignación vp-DSY.

En la Figura 3.5 se han comparado los tres esquemas de renombre: el renombre convencional (conv), vp-NRR y vp-DSY. La figura muestra el número promedio de instrucciones que hacen el *commit* por ciclo (IPC) para cada uno de los programas de prueba y la media armónica (Mh) para los programas de tipo entero y fp. Se supone de nuevo la configuración de $64_{ent}$+$64_{fp}$ registros. Una organización con registros virtuales-físicos y estrategias NRR o DSY supera de forma significativa a la organización convencional. Con NRR se obtienen *speed-ups* promedios de un 3% y de un 16% para códigos enteros y fp respectivamente. Para DSY esos *speed-ups* son de un 5% y de un 24% respectivamente. Se puede observar como los beneficios que reportan los registros virtuales-físicos son mucho más importantes para los programas fp que para los programas enteros. Este resultado era de esperar ya que los programas fp en general ejercen mucha más presión sobre el banco de registros (Capítulo 2).

La diferencia entre las políticas de asignación de registros NRR y DSY también es notable. Vp-DSY puede ser tan agresiva como vp-NRR, pero es capaz de detectar cuando las instrucciones más viejas están progresando de forma lenta debido a la no disponibilidad de registros físicos y reacciona robando registros a instrucciones más jóvenes para asignarlos a esas instrucciones más viejas. En el caso particular de la Figura 3.5, vp-DSY proporciona un *speed-up* promedio de un 2% y un 7% respectivamente, para los códigos enteros y fp.

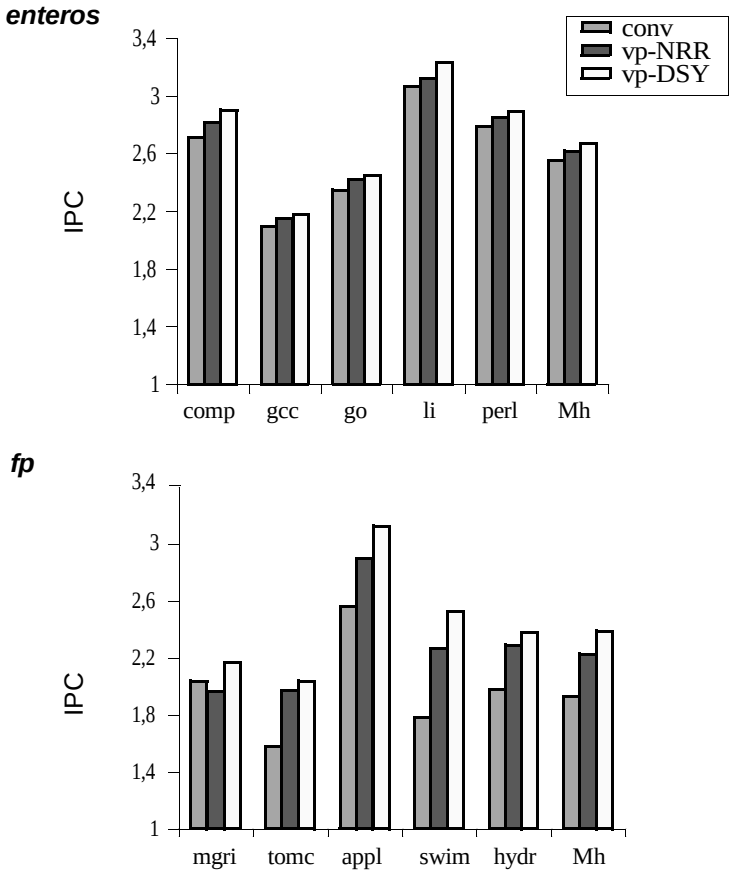


**Figura 3.5** Rendimiento de los dos mecanismos de renombre virtuales-físicos (vp-NRR, vp-DSY) *versus* el esquema convencional para un banco de 64ent+64fp registros.

El análisis del efecto de variar el número de registros en el rendimiento del procesador puede ser más interesante que la experimentación sólo con un tamaño del banco de registros en particular. En general todo diseñador de procesadores debería estar interesado en encontrar la mejor relación entre el número de registros y el rendimiento. La Figura 3.6 muestra como el rendimiento del procesador (media armónica del IPC de todos los programas de cada tipo) varía en función del número de registros físicos para la organizaciones convencional y de registros virtuales-físicos. Para esta última de nuevo se han comparado los dos esquemas que eliminan el problema del *abrazo mortal*: vp-NRR y vp-DSY. Los registros virtuales-físicos con la estrategia DSY funcionan siempre mejor que los registros virtuales-físicos con la estrategia NRR, la cual es a su vez mejor que el esquema de renombre convencional. Notar también que la diferencia entre los tres esquemas vuelve a ser mucho más importante para códigos fp y que se acentúa si disminuye el número de registros físicos.

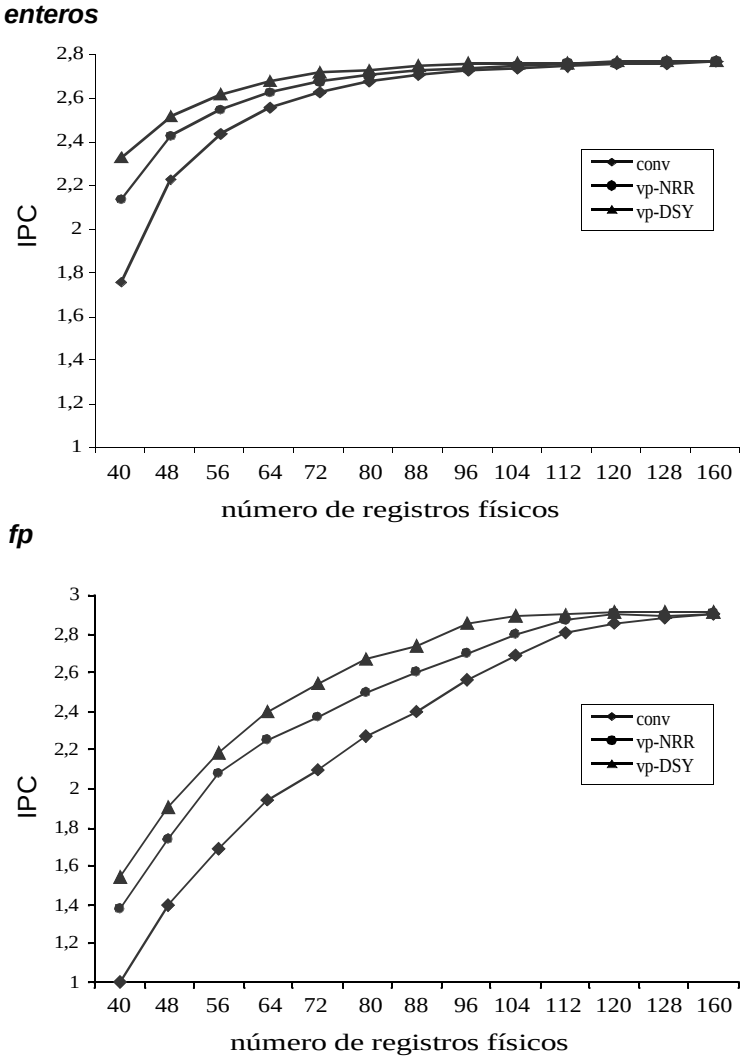


**Figura 3.6** Rendimiento en función del tamaño del banco de registros para la organización de registros convencional (conv) y el esquema de renombre virtuales-físicos con las dos estrategias que evitan el *abrazo mortal* (vp-NRR, vp-DSY).

Respecto al problema de las reejecuciones, comparando los dos esquemas se observó como éstas aparecen en menor grado en vp-DSY que en vp-NRR. Instrucciones muy jóvenes que finalizan su ejecución son siempre reejecutadas con vp-NRR aunque existan registros libres. Sin embargo, vp-DSY cuando hay disponibilidad de registros, permite avanzar a esas mismas instrucciones sin reejecutarlas. Sólo si más tarde, la disponibilidad de los registros empieza a ser crítica, DSY reacciona robando el registro a esas instrucciones para reejecutarlas entonces. Se ha medido el número de reejecuciones del total de instrucciones que alcanzan la etapa de *commit*. De vp-NRR a vp-DSY y con $64_{ent}+64_{fp}$ registros físicos, ese número se reduce un 34% para los programas enteros y un 6% para fp. Con $96_{ent}+96_{fp}$ registros esa reducción llega a ser de hasta un 41% y un 45% en programas enteros y fp respectivamente.

## 3.7 Resultados: vp-DSY

En la Figura 3.6 se observa cómo es el comportamiento que el mecanismo vp-DSY posee bajo diferentes condiciones de presión sobre el banco de registros. Respecto del mecanismo convencional, para los programas enteros se obtienen *speed-ups* que disminuyen a medida que el banco de registros se hace más sobrado: desde un 33% hasta un 2% en el rango de 40 y 88 registros. Para programas fp los rendimientos obtenidos son buenos para casi todo el rango de registros y superan a los de los programas enteros. Los *speed-ups* varían ahora desde un 54% con 40 registros hasta un 2% con 120 registros (con 88 registros es de un 14%). En las Figuras B.1 a B.3 del Apéndice B se encuentran las mismas medidas de la Figura 3.6 por separado para cada uno de los programas de prueba.

De la Figura 3.6 se puede observar también como los registros virtuales-físicos permiten una reducción del número de registros. El dimensionado ideal del banco de registros físicos del procesador sería aquel que con el menor número de registros proporcionara un rendimiento cercano al obtenido con un tamaño máximo del banco de registros (es decir, con tantos registros como el número de entradas del ROS más el número de registros lógicos). Por ejemplo, si podemos permitirnos un 10% de degradación en el IPC respecto de su máximo valor, necesitaríamos 61 registros enteros y 101 registros fp utilizando el esquema de renombre convencional, mientras que utilizando los registros virtuales-físicos y la estrategia DSY sería suficiente con 45 y 77 registros -enteros y fp-. Lo anterior supone una reducción en el tamaño del banco de registros de un 26% y un 24% respectivamente. Esto se traduce directamente en una reducción en el tiempo de acceso al banco de registros y en su área ya que ambos valores se ven afectados de forma significativa por el número de registros [FJC96].

En el apartado anterior se ha expuesto que las reejecuciones siguen existiendo en vp-DSY, siendo éstas un factor crítico del rendimiento. Estas reejecuciones son abundantes en configuraciones con bancos de registros muy ajustados y van disminuyendo al usar configuraciones más sobradas. Para la configuración de $48_{ent}+48_{fp}$ registros, del total de instrucciones que hacen el *commit*, el porcentaje de ellas que se reejecutan porque se les roba su registro físico destino es de un 19.7% para programas enteros y de un 47.7% para códigos fp. Para $64_{ent}+64_{fp}$ registros esos mismos valores bajan hasta un 6.6% y un 31.2% respectivamente. Con $96_{ent}+96_{fp}$ registros, las reejecuciones representan sólo el 1% y el 8%. La diferencia que se observa entre programas enteros y fp de nuevo es debida al diferente comportamiento de estas aplicaciones. Los programas fp poseen mayor ILP y también un porcentaje bajo de fallos al predecir los saltos, lo cual permite mantener llena la ventana de lanzamiento casi todo el tiempo. Con ello, un esquema vp-DSY puede asignar registros a instrucciones muy jóvenes, incrementando así la cantidad de ILP a extraer y también el rendimiento. Todo lo anterior a expensas de continuar realizando un número importante de reejecuciones. Por otro lado, las aplicaciones de tipo entero experimentan un porcentaje elevado de fallos en la predicción de los saltos, lo cual implica a su vez que la ventana de lanzamiento no puede llenarse completamente y entonces, el esquema vp-DSY no es capaz de extraer de este tipo de programas mucho ILP, y por tanto no consigue aumentar mucho el rendimiento.

En cualquier caso, el objetivo de extraer el mayor rendimiento posible de los programas motivó que se eligiera al mecanismo vp-DSY como representativo entre todos aquellos que utilizaban

los registros virtuales-físicos. A continuación, se van a examinar otros resultados interesantes que fueron evaluados para vp-DSY. Estos resultados también se reproducirán para las distintas estrategias que se irán viendo en los siguientes Capítulos.

### 3.7.1 *Utilización* de los registros

Se ha medido el número promedio de los registros Asignados que se encuentran en los distintos estados (Vacío, Utilizado y Desocupado), así como la *utilización* de los registros, de la misma forma que se hizo en el Capítulo 2 para el renombre convencional. Recordar que en ese Capítulo se definió *utilización* como el porcentaje del número de registros Utilizados sobre el número de registros Asignados.

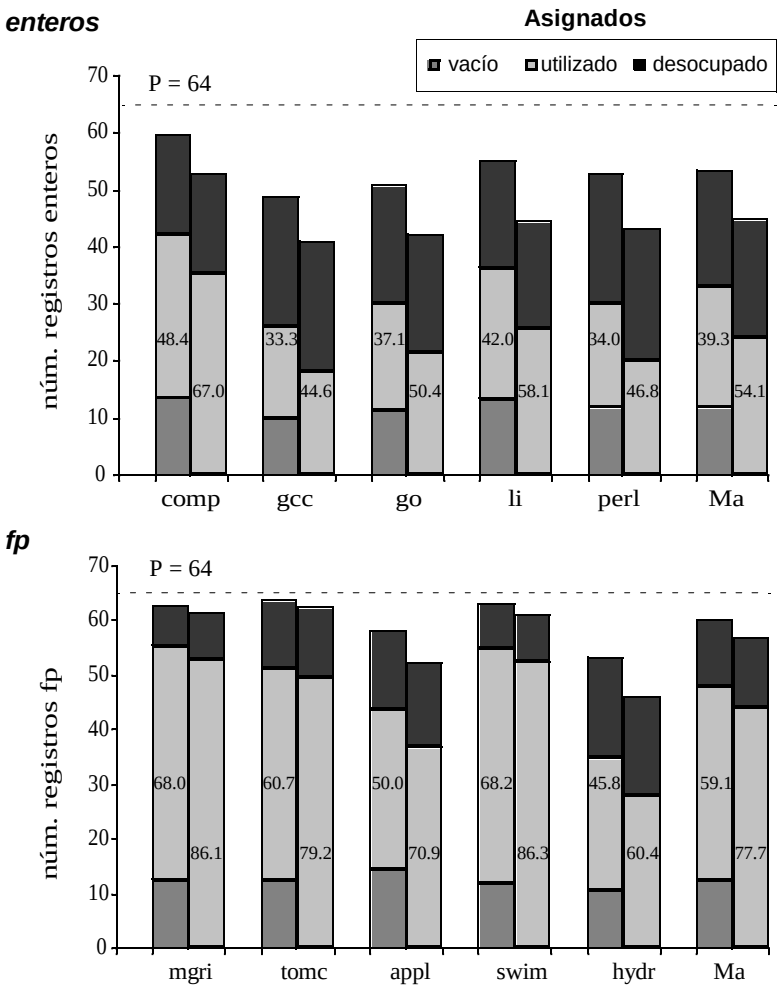


**Figura 3.7** Número de registros Asignados que se encuentran en los estados Vacío, Utilizado o Desocupado. Se comparan el mecanismo de renombre convencional y vp-DSY. 64ent+64fp registros.

La Figura 3.7 compara los resultados del anterior experimento para $64_{ent}+64_{fp}$ registros físicos y para la estrategia vp-DSY frente a la convencional. La columna de la izquierda vuelve a repro-

ducir los datos que se presentaron en la Figura 1.4 (mecanismo convencional) y se comparan con los obtenidos por el mecanismo vp-DSY (columna de la derecha). Los valores de utilización se han incluido dentro de las correspondientes columnas para el mecanismo convencional y para vp-DSY. En la Tabla A.2 del Apéndice A se pueden consultar los valores con los que reproducir las columnas de vp-DSY que aparecen en la Figura 3.7.

Se puede comprobar como al retrasar la asignación de registro físico hasta el final de la ejecución en vp-DSY ha desaparecido el estado Vacío. Esto hace que para todo tipo de programas se observe una considerable reducción en la presión ejercida sobre el banco de registros. En concreto, se produce un incremento promedio en el número de registros que se encuentran Libres. Para todo tipo de programas el número de registros Libres aumenta alrededor de un 2% respecto del mecanismo convencional. Por otro lado, en promedio, la utilización de los registros que en los programas enteros era baja (39.3%) aumenta con vp-DSY hasta un 54.1%. Para fp la utilización pasa de un 59.1% a un 77.1%. Sin embargo, para los programas fp se ha observado que el número de registros en estado Desocupado aumenta, en promedio un 4%. Este estado Desocupado formaba parte de la segunda ineficiencia del renombre convencional. En el siguiente Capítulo se van a mostrar las soluciones propuestas para reducir este valor.

### 3.7.2 Adelanto en la ejecución de las instrucciones

Como se ha visto en la sección 3.3.2 la ejecución anticipada de las instrucciones tiene beneficios que pueden ser importantes. En este apartado se van a presentar estadísticas relacionadas con este efecto para el mecanismo vp-DSY.

Se considera que una instrucción se *anticipa* cuando se ejecuta en una situación en la que el esquema de renombre convencional no la hubiese ejecutado. Una instrucción se ejecuta de forma anticipada cuando en el momento en que se lanza a ejecutar, el procesador tiene en la ventana de ejecución un número de instrucciones más viejas, que habrían consumido ya todos los registros físicos en el esquema convencional. En esta situación, el esquema convencional no habría podido ejecutar dicha instrucción.

Suponiendo un banco de registros con $64_{ent}+64_{fp}$ registros físicos, se evalúa por un lado el porcentaje de instrucciones anticipadas y por otro, el número de ciclos que se anticipan. Para ello se ha seguido el siguiente proceso:

- Si una instrucción encuentra en la ventana de lanzamiento más de 32 instrucciones que son más viejas que ella y que además requieren de registro físico, se considera que esta instrucción es anticipada. Se registra el ciclo en el que ocurre esto (el *ciclo de anticipación*).

- En el momento en el que en la ventana de lanzamiento hay menos de 32 instrucciones que requieren de registro físico más viejas que una instrucción anticipada (esta situación siempre ocurre antes de que la instrucción alcance su etapa de *commit*) se considera que este es el instante en que dicha instrucción se hubiera ejecutado en el esquema de renombre convencional. También se registra el ciclo en el que ocurre esto (el *ciclo convencional*).

- Cuando una instrucción alcanza la etapa de *commit*, el número de ciclos que se ha anticipado se calcula como la diferencia entre el *ciclo convencional* y el *ciclo de anticipación*.

La Figura 3.8.a muestra el porcentaje de *loads* que fallan en memoria *cache*, desglosados en anticipados y no anticipados. Se puede observar que existe un grado razonable de *loads* que fallan y que son anticipados (un 20% en promedio para los programas de prueba enteros y un 86% para fp). Un caso de especial interés lo representa el programa *tomcatv*, para el cual además se obtiene el mayor *speed-up*. *Tomcatv* anticipa el 99% de los *loads* que fallan en memoria *cache*. La Figura 3.8.b muestra el porcentaje de saltos condicionales que son mal predichos -anticipados y no anticipados-. En promedio, el 6% de los saltos condicionales mal predichos son anticipados para los programas enteros y el 74% para los fp. De nuevo *tomcatv* es el que más se beneficia de la anticipación de instrucciones ya que se anticipan el 97% de los saltos condicionales mal predichos.

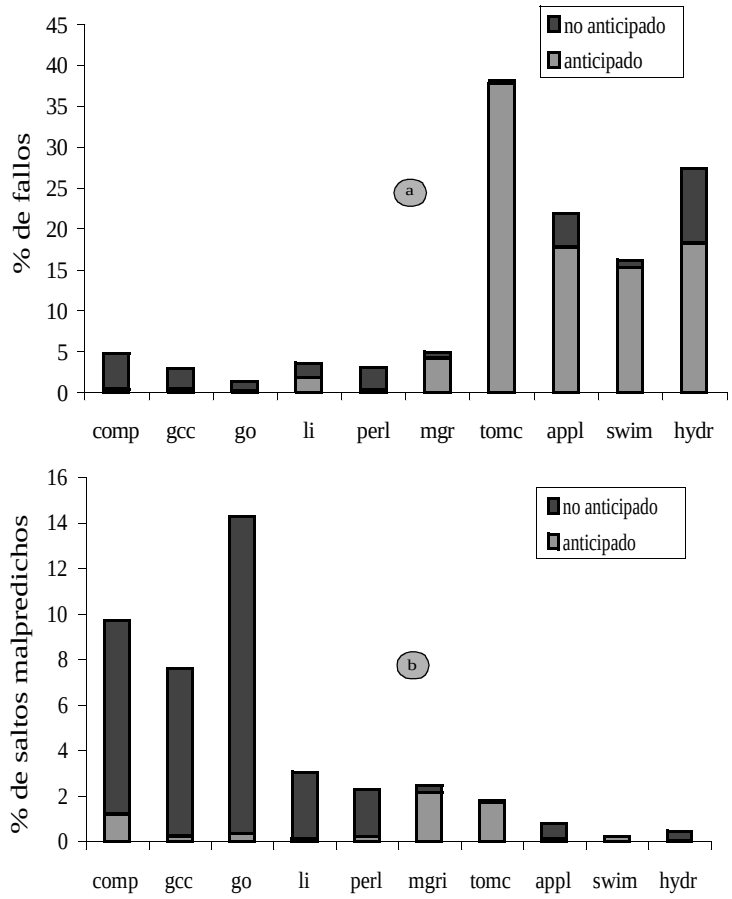


**Figura 3.8** Porcentaje de *loads* que fallan en *cache* -anticipados y no anticipados- **(a)** y porcentaje de saltos condicionales que son mal predichos -anticipados y no anticipados- **(b)**. Mecanismo vp-DSY.

Otra estadística interesante es el número promedio de ciclos que se anticipa una instrucción. Las instrucciones de *load* se anticipan un promedio de 9.4 ciclos para los programas enteros y 26.9 ciclos para fp. Para los saltos condicionales esos mismos cálculos dan 4.8 y 28 ciclos respectivamente. Para las instrucciones de división, multiplicación y simples se obtienen respectivamente 2.6, 3 y 6.1 ciclos para los programas enteros y 13.3, 13.9 y 17.1 ciclos para los fp.

El mecanismo de renombre de registros virtuales-físicos facilita la entrada de algunas instrucciones en la ventana de lanzamiento y su ejecución se realiza antes de lo que lo harían con el mecanismo de renombre convencional. Como han confirmado las estadísticas anteriores, este beneficio es mucho más importante para programas fp que para enteros. Los programas fp poseen un porcentaje mayor de instrucciones anticipadas y en promedio estas instrucciones se anticipan también un número de ciclos mayor que en los programas enteros. Esto se explica de nuevo por el hecho de que los programas enteros poseen mayor número de saltos que son mal predichos lo cual limita el número de instrucciones que pueden encontrarse en vuelo en el procesador en un momento dado. Para un mismo número de registros físicos en el procesador, el mecanismo vp-DSY permite tener más instrucciones en vuelo que el esquema convencional. Sin embargo, los saltos mal predichos impiden que el procesador pueda sacar ventaja de esta característica en todos los casos.

## 3.8 Implementación

En este apartado se va a analizar el impacto que supone implementar el mecanismo de los registros virtuales-físicos. El análisis se va a centrar en el coste de almacenamiento, tiempo de ciclo y consumo de energía que tendrían aquellas partes del *hardware* propuesto que se consideran más críticas. Para ello se ha utilizado el modelo de tiempo de acceso y potencia de Rixner et al. para una tecnología de 0.18 μm [RDK+00].

### 3.8.1 Almacenamiento

Debido a que los registros virtuales-físicos no están asociados a ningún almacén, no ocupan espacio y por lo tanto, su número posee un impacto pequeño en el coste *hardware* de la técnica. Sin embargo, el coste de implementar la tabla PMT sí que puede ser elevado debido al gran número de entradas que posee. Esta tabla posee NVR filas de $\lceil \log_2(NPR) \rceil$ bits cada una (recordar: NVR = número de registros virtuales-físicos; NPR = número de registros físicos). Una solución para reducir el tamaño de la tabla PMT consiste en implementarla utilizando una memoria CAM (memoria direccionada por contenido) con un número de entradas igual al número de registros físicos que es mucho menor que el número de registros virtuales-físicos. Por ejemplo, esta es la solución que se utiliza en el procesador Alpha DEC 21264 [Gwe96] para implementar la tabla de mapeo MT. Con ello, la tabla PMT se puede implementar con NPR filas de $\lceil \log_2(NVR) \rceil$ bits cada una. Por su parte, la tabla GMT posee NLR -número de registros lógicos-filas con un número de bits igual a $\lceil \log_2(NVR) \rceil + \lceil \log_2(NPR) \rceil + 1$ cada una. Para este mismo procesador que posee un ROS de 80 entradas, 80 registros enteros y 72 registros fp, el almacenamiento que habría que añadir para soportar las estructuras de los registros virtuales-físicos sería de tan sólo 197 Bytes.

### 3.8.2 Tiempo de Ciclo

Un mecanismo de registros virtuales-físicos no implica añadir ningún retardo al camino crítico del procesador cuando lo comparamos con el mecanismo convencional, a excepción de la etapa de *commit*, la cual puede retrasarse en un ciclo debido a la necesidad de acceder a la tabla PMT. En cuanto a la asignación de los registros físicos, ésta puede hacerse durante el último ciclo de

la etapa de ejecución para que esté disponible al comienzo de la etapa de escritura. El acceso a la tabla GMT es equivalente al que se realiza en el mecanismo convencional, sin embargo, debido a su incremento de tamaño frente a la MT, la tabla GMT junto con la PMT se van a utilizar como estructuras clave del análisis. Se considera la gestión vp-DSY al ser la que más accesos genera a las tablas.

En la Figura 3.9.a se presentan los valores para el tiempo de acceso de las tablas GMT, PMT y del banco de registros que se han obtenido utilizando el modelo de Rixner et al.. Se muestran los valores para las tablas GMT y PMT de tipo fp al ser éstas las que mayor número de puertos poseen. El tamaño del banco de registros se ha variado desde 40 hasta 160 registros físicos. Según los parámetros utilizados en las evaluaciones, el número de puertos para cualquier banco de registros entero es de 48 y para cualquier banco de registros fp es de 54. Para la tabla GMT y debido a que cada acceso a ella no siempre se realiza a sus tres campos -VP, P, V- la hemos supuesto dividida en tres partes: GMT.VP, GMT.P y GMT.V. Cada parte dispone de 32 entradas, la anchura de GMT.VP es de 8 bits; la de GMT.P varía según el tamaño del banco de registros: 6 bits para bancos de registros entre 40 y 64 registros, 7 bits entre 72 y 128 registros y 8 bits entre 136 y 160 registros; la anchura de GMT.V es de 1 bit. El total de puertos para GMT.VP es de 68: 60 de lectura y 8 de escritura; para GMT.P es de 34: 16 de lectura y 18 de escritura; para GMT.V es de 60: 16 de lectura y 44 de escritura. La Figura 3.9.a presenta los valores calculados para GMT.VP de tipo fp al ser ésta la que mayor tiempo de acceso necesita. Por su parte, la PMT posee 160 entradas cada una de 6,7 u 8 bits (según el tamaño del banco de registros). El total de puertos de cualquier PMT es de 44: 26 de lectura y 18 de escritura. En la Figura 3.9.a se puede observar como el tiempo de acceso a las tablas GMT o PMT siempre es inferior al de cualquier tamaño de banco de registros.

### 3.8.3 Consumo de Energía

En la Figura 3.9.b se pueden ver también los valores calculados con el modelo de Rixner et al. [RDK+00] para el consumo de energía de la tablas $GMT_{fp}$ y $PMT_{fp}$. Para todas las configuraciones, el consumo de energía siempre es menor que el del banco de registros más reducido (40 registros).

El efecto que pueden tener las reejecuciones sobre el consumo de energía sí que puede llegar a ser importante. Aunque no se ha considerado en este análisis, el consumo de recursos de las instrucciones que se reejecutan y su latencia pueden reducirse de forma significativa mediante algún mecanismo de reutilización [SS97]. Las instrucciones que deben reejecutarse pueden guardar sus resultados en un *buffer* de reutilización. Más adelante cuando haya registros físicos disponibles, estos resultados se copian directamente desde los *buffers* hacia los registros físicos. Utilizando este tipo de mecanismos, la reejecución de instrucciones no tiene por qué provocar ningún tipo de contención en las unidades funcionales ni consumos elevados de energía.

En el apartado de resultados, se ha visto cómo el mecanismo vp-DSY puede utilizarse como un medio de reducir el número de registros necesarios para obtener un determinado nivel de rendimiento. Por ejemplo, se vio que permitiendo una degradación del 10% de IPC con un mecanismo convencional de renombre se necesita un banco de registros con 61 registros enteros + 101 registros fp, mientras que con vp-DSY basta con disponer de un banco registros con 45 registros

enteros + 77 registros fp. Al comparar el consumo de energía de estas dos alternativas, se observa lo siguiente:

$E_{convencional}(BR_{61ent} + BR_{101fp}) = 1689.4$ pJ + 3433.7 pJ = 5123.1 pJ,

y con vp-DSY:

$E_{vp\text{-}DSY}(BR_{45ent} + BR_{77fp} + GMT_{ent} + GMT_{fp} + PMT_{ent} + PMT_{fp}) =$

1258.8 pJ + 2636 pJ + 301.3 pJ + 345.9 pJ + 315.8 pJ + 434.9 pJ = 5292.7 pJ.

Ocurre que la energía consumida es algo mayor -169.6 pJ- para el caso de vp-DSY. La cantidad de energía que se reduce al reducir el tamaño del banco de registros es casi comparable con la energía que consumen las estructuras que permiten esa reducción.

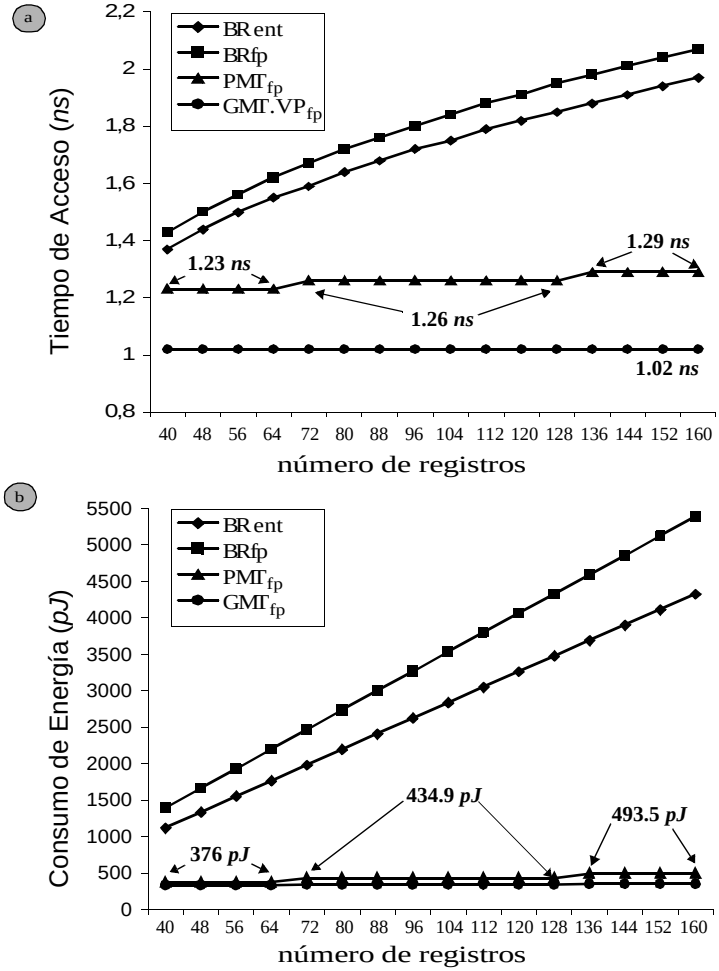


**Figura 3.9** Tiempo de acceso y consumo de energía para las tablas GMT, PMT y el banco de registros. Se considera la gestión vp-DSY.

## 3.9 Conclusiones

Los esquemas de renombre de registros convencionales asignan los registros de forma conservadora, incrementando innecesariamente la necesidad de registros en los procesadores con ejecución fuera de orden. La asignación tardía de registros se presenta como otra posible herramienta de diseño que hace que el banco de registros sea un componente menos crítico en el procesador y que además posee una utilidad doble:

- Permite incrementar el IPC mientras se mantiene el mismo tamaño del banco de registros. Esta utilidad sólo tiene sentido para sistemas con bancos de registros ajustados. El IPC mejora sin degradar el tiempo de ciclo.

- Permite reducir el tamaño del banco de registros para ajustar su tiempo de acceso al tiempo de ciclo del procesador, sin perder IPC. Esta utilidad tiene sentido tanto para sistemas con bancos de registros sobrados como ajustados, pero que se encuentran en el camino crítico del procesador.

Lo anterior es cierto para todas las estrategias propuestas y por lo tanto las Conclusiones de los siguientes Capítulos van a seguir un desarrollo similar al de este. Los resultados obtenidos para cada propuesta en particular, muestran en que grado mejora cada una de ellas al mecanismo convencional.

En este Capítulo se ha presentado un nuevo esquema de renombre que permite retrasar la asignación de los registros físicos. Para ello, se ha definido primero el concepto de los registros virtuales-físicos. La idea es asignar registro físico cuando la instrucción alcanza el final de su etapa de ejecución, en lugar de hacerlo en la etapa de decodificación como hacen los procesadores con renombre convencional. En este contexto, se han propuesto varias técnicas para evitar el problema del *abrazo mortal* presente en el mecanismo de registros virtuales-físicos. Todas ellas asignan los registros físicos bajo demanda y difieren en la forma en que el abrazo mortal es resuelto. Las primeras previenen la aparición de abrazo mortal al reservar un número determinado de registros que garantizan que las instrucciones más viejas que necesitan de registro destino tienen uno disponible (vp-NRR y las alternativas: vp-lanzar, vp-*load*, vp-*load&div*). La siguiente técnica y cuando los registros físicos se agotan, reacciona robando registros a instrucciones más jóvenes (vp-DSY). La comparación realizada entre todas ellas ha demostrado que el esquema vp-DSY es el más efectivo. Vp-DSY consigue avanzar la ejecución de las instrucciones de la forma más rápida y además este avance nunca penaliza a las instrucciones más viejas.

La asignación tardía de registros utilizando vp-DSY muestra mejoras que para códigos fp llegan a ser de hasta un 54% y de un 33% en códigos de tipo entero. Se ha observado también que vp-DSY aumenta de forma significativa la utilización de los registros: de un 39.3% a un 54.1% y de un 59.1% a un 77.1% para programas enteros y fp respectivamente. El porcentaje de registros asignados con contenido vacío es nulo. Además, se puede conseguir una reducción en el número de registros enteros y fp de hasta un 26% y un 24% respectivamente, a la vez que se mantiene el mismo promedio de IPC obtenido por el esquema convencional. Esta reducción en el número de registros se traduce directamente en un menor tiempo de acceso al banco de registros, lo que probablemente va a ser uno de los mayores problemas para los procesadores futuros.

También se ha visto como este esquema permite que los saltos condicionales se resuelvan con anticipación y que las instrucciones de memoria (*load/store*) puedan hacer *prefetch* de sus datos. Se ha cuantificado que el porcentaje de *loads* que fallan en memoria y que son anticipados es de un 20% y un 86% para programas de tipo entero y fp respectivamente. En códigos enteros, el 6% de los saltos condicionales mal predichos son resueltos con anticipación.

## 3.10 Referencias


[BA97] D. Burger, and T.M. Austin, *The SimpleScalar Tool Set v2.0*, Technical report 97-1342, University of Wisconsin-Madison, Computer Science Department, June 1997.

[FJC96] K.I. Farkas, N.P. Jouppi, and P. Chow, "Register File Considerations in Dynamically Scheduled Processors," *in Proceedings of the 2nd International Symposium on High-Performance Computer Architecture* (HPCA 96), pp. 40-51, February 1996.

[GGV98] A. González, J. González, and M. Valero, "Virtual-Physical Registers," *in Proceedings of the 4th International Symposium on High-Performance Computer Architecture* (HPCA 98), pp. 175-184, January/ February 1998.

[GVG+97] A. González, M. Valero, J. González, and T. Monreal, "Virtual Registers," *in Proceedings of the 4th International Conference on High Performance Computing* (HiPC 97), pp. 364-369, December 1997.

[Gwe96] L. Gwennap, "Digital 21264 Sets New Standard," *Microprocessor Report*, vol. 10, no. 14, pp. 11-16, October 1996.

[Gwe97] L. Gwennap, "MIPS R12000 to Hit 300 MHz," *Microprocessor Report, Micro Design Resources*, vol. 11, no. 13, pp. 1-4, October 1997.

[JRB+98] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, and A. Yoaz, "A Novel Renaming Scheme to Exploit Value Temporal Locality Through Physical Register Reuse and Unification," *in Proceedings of the 31st International Symposium on Microarchitecture* (MICRO 98), pp. 216-225, November 1998.

[Kes99] R.E. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24-36, March-April 1999.

[LTT95] D. Levitan, T. Thomas, and P. Tu, "The PowerPC 620 Microprocessor: A High-Performance Superscalar RISC Microprocessor," *in Proceedings of the 40th IEEE Computer Society International Conference* (COMPCON 95), pp. 285-291, March 1995.

[MGV+99a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, "Delaying Physical Register Allocation Through Virtual-Physical Registers," *in Proceedings of the 32nd International Symposium on Microarchitecture* (MICRO 99), pp. 186-192, November 1999.

[MGV+99b] T. Monreal, A. González, M. Valero y V. Viñals, *Registros Virtuales*, Report interno del DIIS, RR-99-08, 1999.

[MGV+00a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, "Dynamic Register Renaming Through Virtual-Physical Registers," *in The Journal of Instruction-Level Parallelism*, vol. 2, May 2000. http://www.jilp.org/vol2.

[RDK+00] S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U.J. Kapasi, and J. D. Owens, "Register Organization for Media Processing," *in Proceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 375-386, January 2000.

[Soh90] G.S. Sohi, "Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers," *in IEEE Transactions on Computers*, vol. 39, no. 3, pp. 349-359, March 1990.

[Son97] P. Song, "IBM's Power3 to Replace P2SC," *Microprocessor Report*, vol. 11, no. 15, pp. 23-27, November 1997.

[SPEC] The Standard Performance Evaluation Corporation, http://www.specbench.org.

[SS97] A. Sodani, and G.S. Sohi, "Dynamic Instruction Reuse," *in Proceedings of the 24th International Symposium on Computer Architecture* (ISCA 97), pp. 194-205, June 1997.

[WB96] S. Wallace, and N. Bagherzadeh, "A Scalable Register File Architecture for Dynamically Scheduled Processors," *in Proceedings of the 5th International Conference on Parallel Architectures and Compilation Techniques* (PACT 96), pp. 179-184, October 1996.

[Yea96] K.C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *in IEEE Micro*, vol. 16, no. 2, pp. 28-40, April 1996.

CAPÍTULO 4

# Liberación Anticipada de Registros: dos Esquemas *Hardware*


## Resumen

*En este Capítulo se describen y evalúan dos mecanismos* hardware *propuestos para disminuir la ineficiencia del mecanismo de renombre convencional en la liberación de los registros. Estos mecanismos liberan los registros del procesador de forma anticipada. Tienen en cuenta la especulación frente a saltos y permiten recuperación precisa de excepciones. Además, las estructuras que se añaden están fuera del camino crítico del procesador. Los dos mecanismos podrán servir a alguno de estos dos objetivos:*

- *Aumento del IPC manteniendo el mismo número de registros. Será el caso de bancos de registros cuyo diseño es lo suficientemente ajustado para no afectar al tiempo de ciclo del procesador.*
- *Reducción del tamaño del banco de registros y por lo tanto de su tiempo de acceso sin pérdida de rendimiento. Será el caso de bancos de registros que se encuentran dentro del camino crítico del procesador. Su reducción puede permitir a su vez una reducción en el tiempo de ciclo del procesador.*

Para soportar un mecanismo de recuperación frente a excepciones que sea preciso, el mecanismo de liberación convencional mantiene a todo registro físico Asignado durante un número de ciclos que en su mayor parte no son necesarios. El mecanismo convencional espera a la etapa de *commit* de la instrucción que crea una nueva *versión* del registro para liberarlo. Sin embargo, desde que la instrucción que utiliza el registro por última vez alcanza la etapa de *commit* hasta que llega a esa misma etapa la instrucción que lo redefine, el contenido del registro no es utilizado (es el estado Desocupado del registro definido en el Capítulo 2).

Las nuevas técnicas de liberación que se presentan en este Capítulo van a disminuir el número de ciclos durante los cuales los registros están Desocupados, reduciendo así la presión que se ejerce sobre el banco de registros. El *hardware* presentado va a permitir detectar con anticipación y de forma segura cuáles son las instrucciones que hacen el último uso de un determinado registro. Utilizar esta información permitirá adelantar la liberación de los registros físicos a la etapa de *commit* de la instrucción que lo ha utilizado por última vez.

En el mismo contexto de liberación anticipada de los registros, Farkas et. al comparan en [FJC96] un mecanismo impreciso de liberación anticipada de registros frente al mecanismo convencional. Las condiciones que proponen para liberar los registros son similares a las que aquí se utilizan en uno de los mecanismos de liberación. Sin embargo, su mecanismo es impreciso debido a que realizan la liberación cuando las instrucciones que hacen el último uso del registro finalizan su ejecución, en lugar de esperar que alcancen su etapa de *commit*. Los autores están más interesados en obtener fronteras de rendimiento y por lo tanto no proponen ningún tipo de implementación para su propuesta.

Otra propuesta para liberar los registros de forma anticipada fue sugerida por Moudgill et al. en [MPV93]. En este trabajo, los autores sugieren una liberación agresiva de los registros físicos que se realiza tan pronto como las instrucciones que hacen el último uso del registro finalizan su ejecución fuera de orden. La identificación que hacen de un último uso se basa en una serie de contadores que registran el número de lecturas pendientes que posee cada registro físico. Esta propuesta inicial no soporta un mecanismo preciso de recuperación frente a excepciones ya que los anteriores contadores no se restauran de forma correcta cuando las instrucciones son eliminadas a causa de una excepción. Posteriormente en ese artículo, los autores presentan una solución simple para soportar excepciones precisas, la cual se basa en retrasar la liberación de los registros conectándola con la etapa de *commit* de la instrucción que genera la siguiente versión del registro. Esta es la propuesta que en esta Tesis se ha considerado como liberación convencional de los registros y que ha servido de base en todas las evaluaciones.

El mismo mecanismo impreciso de liberación anticipada de [MPV93] se utiliza en los trabajos de Balasubramonian et al. en [BDA01a] y en [BDA01b]. En [BDA01b] se propone una organización del banco de registros con dos niveles para reducir el tamaño del banco de registros. El primer nivel (L1) del banco de registros contiene sólo aquellos valores que proporcionan valores activos a las unidades de cálculo, mientras que el segundo nivel (L2) contiene aquellos valores que esperan a que se cumpla un mecanismo preciso de excepciones para ser liberados. Los registros se asignan en la etapa de decodificación desde L1 y se mueven hacia L2 utilizando el mecanismo de liberación anticipada impreciso de [MPV93]. Los valores se mantienen en L2 hasta asegurar corrección en la ejecución del programa, ya que podrían necesitarse tras una mala

predicción de un salto o tras una excepción. Por otro lado, en [BDA01a] todos los registros disponibles se asignan de forma dinámica entre dos *threads*, denominados *thread principal* y *thread futuro*. Utilizando de nuevo el mecanismo impreciso de liberación anticipada, el *thread futuro* es capaz de progresar en la ejecución del programa a base de ir examinando una ventana de lanzamiento mucho mayor y ejecutando de forma anticipada instrucciones de las que están preparadas. El *thread futuro* es generado de forma dinámica por el *hardware* en el momento en que el *thread principal* debe parar por quedarse sin registros físicos disponibles. Este *thread futuro* consta de un contador del programa (PC) y del estado de los registros, sirviendo sólo al propósito de calentar el banco de registros, las caches de instrucciones y datos y de resolver con anticipación los saltos mal predichos. Más adelante, cuando el *thread principal* deja de estar parado, reejecutará estas instrucciones para asegurar el *commit* en orden y la corrección del programa.

Existen también otros trabajos ortogonales a los que aquí se presentan que proponen utilizar el compilador para detectar las instrucciones que hacen el último uso de los registros y liberarlos de forma anticipada [LPE+99][MRF97]. El compilador puede identificar registros que almacenan valores que ya no son utilizados (valores que están *muertos*) y a continuación informar de ese hecho al *hardware*. Para hacer esto se requiere un cambio en el conjunto de instrucciones de la arquitectura (ISA), bien en la línea de definir bits extra para cada instrucción o de añadir nuevas instrucciones mediante las cuales el compilador puede indicar que se liberen registros. Sin embargo, el compilador posee un conocimiento limitado del flujo dinámico del programa y por lo tanto la liberación de los registros se hará de forma conservadora. Por el contrario, las soluciones *hardware* poseen el potencial de identificar de forma dinámica los últimos usos, liberando así más registros de forma anticipada.

## 4.1 Mecanismo Básico

La idea básica de la *liberación anticipada* que se propone aquí consiste en asociar la liberación de un registro físico a la etapa de *commit* de la instrucción que lo utiliza por última vez en orden de programa. Se va a utilizar la notación presentada en el Capítulo 2, donde se llama V a la instrucción que define un registro, LU a la instrucción que hace el último uso del registro y NV a la primera instrucción que reescribe el mismo registro. También en el Capítulo 2, se vio como el renombre de registros convencional fuerza que un registro físico se encuentre Desocupado desde la etapa de *commit* de la instrucción LU hasta la etapa de *commit* de la instrucción NV. La *liberación anticipada* va a disminuir la cantidad de tiempo que los registros se encuentran en el estado Desocupado.

En los dos códigos de la Figura 4.1 pueden verse dos ejemplos en los que el registro lógico r2 se renombra al registro físico p7. La Figura 4.1.a muestra el caso más habitual donde la instrucción que hace el último uso de r2 es una lectura y la Figura 4.1.b muestra el caso menos habitual donde la instrucción que hace el último uso de r2 es una escritura. En la Figura 4.1.b la instrucción etiquetada como V, LU calcula un resultado que nunca va a ser utilizado, creando una nueva versión para el registro r2 y utilizándola por última vez.

Para realizar una *liberación anticipada* se propone que en la etapa de decodificación de la instrucción NV -con registro físico p7 asignado a su registro lógico destino r2- se lleven a cabo las siguientes dos acciones:

- *Identificar* a la instrucción que utiliza p7 por última vez -LU-.
- *Planificar la liberación anticipada* de p7 para la etapa de *commit* de LU. En el caso de que la instrucción LU haya alcanzado su etapa de *commit* cuando se decodifica su par NV, el registro físico correspondiente puede liberarse de forma inmediata.

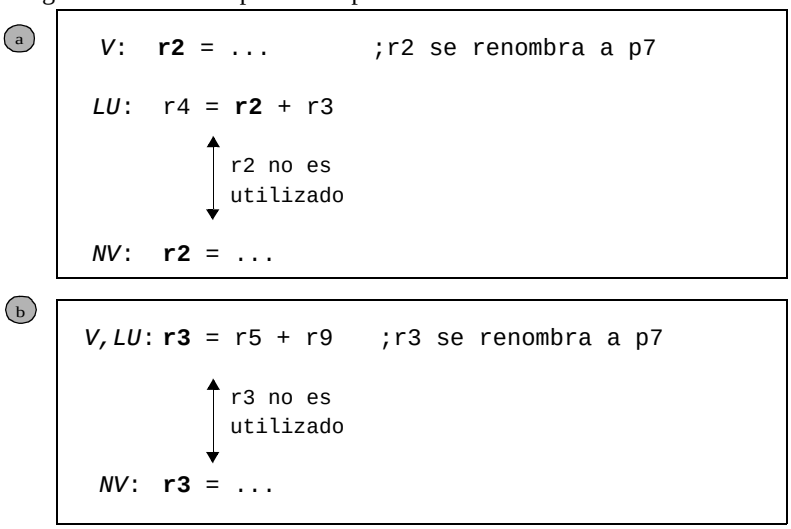


**Figura 4.1** Dos ejemplos del potencial de la liberación anticipada de registros. En los dos códigos, el registro físico p7 puede liberarse cuando la instrucción LU alcanza la etapa de *commit* en lugar de esperar a que lo haga la instrucción NV.

Si existen una o varias instrucciones de salto entre las instrucciones LU y NV, al igual que ocurre con la recuperación frente a saltos mal predichos, se ha de utilizar un *hardware* específico de recuperación capaz de deshacer los efectos que haya producido el camino mal predicho. Si una instrucción NV que era especulada tiene que desaparecer, es necesario deshacer cualquier liberación que dicha instrucción haya planificado. Llegados a este punto, hay que distinguir dos casos:

1. Entre las instrucciones LU y NV no existen saltos pendientes de ser confirmados. Este caso ocurre cuando LU y NV pertenecen al mismo *bloque básico*. También ocurre cuando aún estando LU y NV en diferentes bloques básicos, al decodificar NV todos los saltos que hay entre ellas ya han sido ejecutados y sus condiciones y direcciones destino verificadas.
2. Al decodificar la instrucción NV existen todavía saltos pendientes de verificar entre ella y su par previo LU.

El concepto de la liberación anticipada y precisa de registros que se presenta en esta Tesis dio origen a las publicaciones en [MGV+00b], [MVG+01] y [MVG+02]. A continuación, se va a completar la descripción de los mecanismos propuestos tal y como se hizo en los artículos citados. Para ello se irán presentando de forma incremental los distintos elementos *hardware* nece-

sarios para liberar los registros de forma anticipada, así como su funcionamiento. En este mismo apartado se va a introducir una posible implementación de un mecanismo básico de liberación que cubre sólo el primero de los casos anteriores. En el siguiente apartado, se verá como se extiende este mecanismo para cubrir también el segundo caso.

El mecanismo básico sólo planificará liberaciones anticipadas si en el momento de decodificar la instrucción NV se identifican de forma *segura* el par de instrucciones LU-NV. Esta situación *segura* se corresponde con el primero de los casos citados arriba. Para el resto, la planificación sigue una política de liberación de registros como la convencional.

Para implementar el mecanismo básico, cualquier instrucción debe tener posibilidad de liberar en la etapa de *commit* cualquiera de sus registros físicos fuentes (Figura 4.1.a) y también su registro físico destino (Figura 4.1.b). Además, para el caso de que un registro se libere de forma anticipada, debe ser posible desconectar las acciones que el mecanismo convencional realiza para liberar la versión anterior del registro (old_pd).

### 4.1.1 Recursos *Hardware*

La Figura 4.2 muestra una implementación que se basa en añadir campos al ROS y a la tabla de mapeo (MT). El *ROS extendido* posee ahora los siguientes campos:

- r1, r2, rd: identificadores de los registros lógicos.
- p1, p2, pd: identificadores de los registros físicos.
- old_pd: identificador del registro físico asignado como versión anterior al registro lógico destino.
- rel_old: bit de liberación de la versión anterior. Si se encuentra a cero en la etapa de *commit*, el registro old_pd no será liberado.
- rel1, rel2, reld: bits de liberación anticipada para p1, p2 y pd respectivamente. Cuando se encuentran a uno en la etapa de *commit*, fuerzan a la liberación del correspondiente registro físico.

En la etapa de decodificación/renombre, el bit de liberación de la versión anterior se inicializa a uno y todos los bits de liberación anticipada se inicializan a cero.

La extensión de la MT se va a describir como una estructura separada que se denomina *Tabla de Ultimos Usos* (*LUsT*). La LUsT identifica las instrucciones que utilizan por última vez un registro determinado. Cada entrada posee los siguientes campos:

- ROSid: para cada registro lógico, este campo guarda el identificador de la instrucción que lo utiliza por última vez, es decir, la instrucción LU.
- Kind: el tipo de uso que se hace del registro: fuente1, fuente2 o destino.
- C: bit que informa si la instrucción LU ha alcanzado ya la etapa de *commit* (C = uno) o no (C = cero).

Una vez conocida a través de esta tabla la identificación de la instrucción LU (número de entrada que dicha instrucción posee en el ROS), puede planificarse una liberación anticipada poniendo a uno el correspondiente bit de liberación anticipada. De la misma forma que ocurre para MT, se supone que en cada predicción de salto se hace una copia de la tabla LUsT, y que se recuperará en caso de mala predicción del salto [HP87].

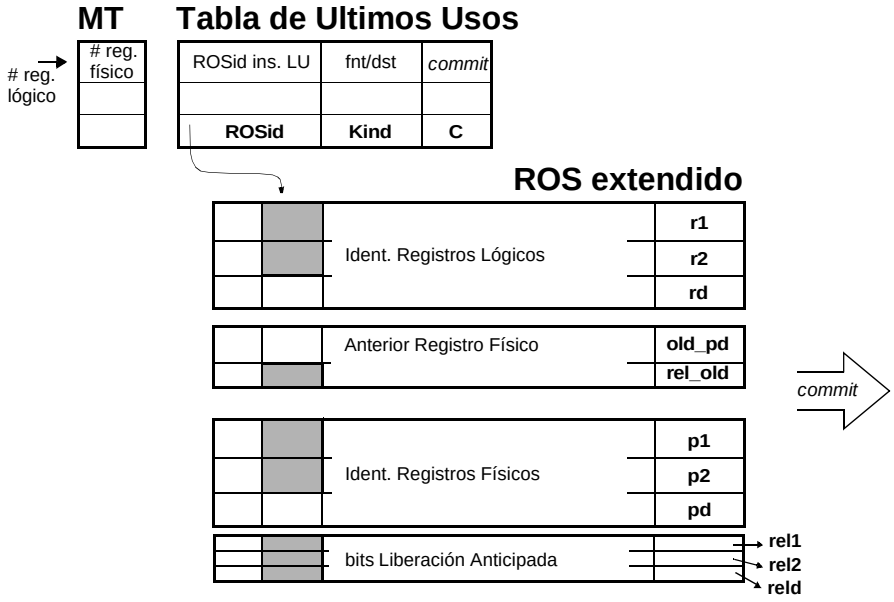


**Figura 4.2** Mecanismo básico. Las áreas sombreadas destacan los campos añadidos para extender el ROS.

### 4.1.2 Control

Los distintos pasos de control se realizan en las etapas de renombre y *commit*. Se van a describir a continuación aunque en orden inverso.

#### *COMMIT*: ACTUALIZACIÓN DEL BIT C Y LIBERACIÓN DE REGISTROS.

Cuando una instrucción *i* alcanza la etapa de *commit*, los identificadores de todos sus registros lógicos están disponibles en la cabeza del ROS. La tabla LUsT se indexa entonces con estos identificadores para leer los campos ROSid y compararlos con el identificador *i*. En aquellos casos donde existe coincidencia, se pone a uno el bit C[1]. De esta forma, queda registrado que una potencial instrucción LU ha alcanzado la etapa de *commit*. Notar que esta acción realizada en el bit C debe extenderse a todas las copias de la tabla LUsT para obtener una recuperación correcta en caso de mala predicción.

1. Otra posibilidad sería acceder de forma asociativa a todos los campos ROSid con el identificador y poner a uno todos aquellos bits C donde exista coincidencia. Con esta opción, los campos r1 y r2 en el ROS no son necesarios.

En paralelo, todos los bits de liberación anticipada que la instrucción tenga a uno provocarán la liberación de los correspondientes registros físicos. Pueden devolverse a la lista de registros libres un máximo de cuatro identificadores: p1, p2, pd (por liberación anticipada) y old_pd (por liberación convencional).

*RENOMBRE*.

Los dos pasos que se describen ahora se realizan en la etapa de renombre. Para su explicación se va a utilizar el código sencillo de la Figura 4.3. Se va a suponer el caso de instrucciones en las que intervienen tres registros (fuente1, fuente2 y destino). Recordar que estamos en el caso de que no existen saltos pendientes de verificarse entre las instrucciones NV y LU.

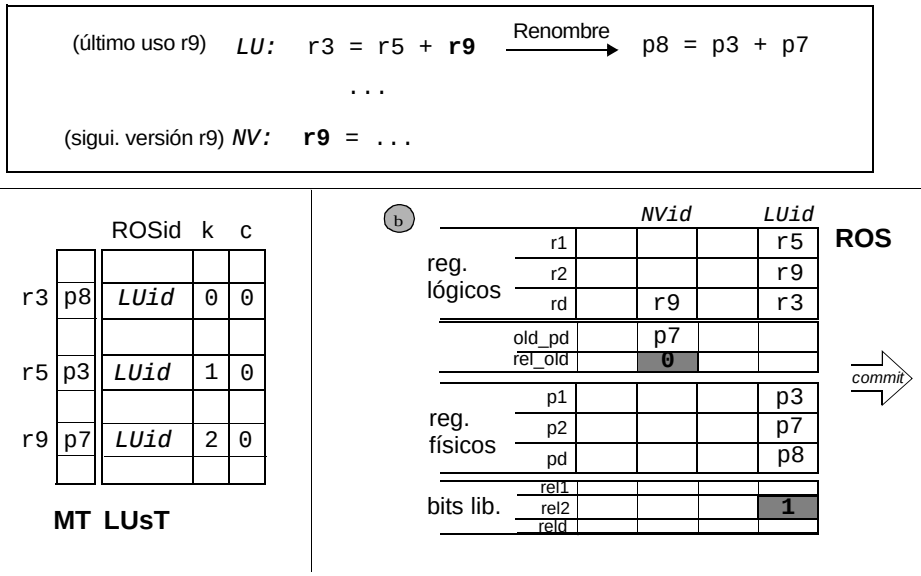


**Figura 4.3** Tablas MT y LUsT después de la etapa de renombre de la instrucción LU **(a)**. Decodificación de la instrucción NV: se consulta la tabla LUsT, rel_old se pone a cero para desactivar la liberación de p7 por parte de NV y se planifica para la instrucción LU una liberación anticipada de p7 **(b).**

ACTUALIZACIÓN DE LA TABLA LUsT.

Para anotar los usos de los registros de forma correcta por cada instrucción que se renombra deben actualizarse tres entradas en la tabla LUsT. La posición de la instrucción en el ROS se coloca en el campo ROSid, el tipo de uso que se hace del registro se coloca en el campo Kind (fuente1, fuente2 o destino) y el bit C se pone a cero. De esta forma, la identidad de la instrucción que utiliza un determinado registro lógico por última vez se guarda en orden de programa. La Figura 4.3.a muestra un ejemplo de este paso.

PLANIFICACIÓN DE LIBERACIÓN O REUTILIZACIÓN DE REGISTRO.

Por cada instrucción que posea registro destino -una instrucción NV-, este paso planifica una liberación anticipada para el registro físico de la versión anterior o su reutilización.

Para hacer esto, el registro lógico destino de NV se utiliza para indexar la tabla LUsT. Sea LU la instrucción localizada y LUid su dirección en el ROS. A continuación, se va a actuar de la siguiente forma:

- Si la instrucción LU todavía no ha alcanzado la etapa del *commit* (C es igual a cero), en su entrada en el ROS se pone a uno el correspondiente bit de liberación anticipada: relx[LUid] = uno (x es 1, 2 ó d). También se pone a cero el bit de liberación de la versión anterior para la instrucción NV: rel_old[NVid] = cero. La Figura 4.3.b ilustra este paso.

- Si la instrucción LU ya ha alcanzado la etapa de *commit* (C es igual a uno), el registro físico que está anotado en la MT puede liberarse de forma inmediata. Como en el caso anterior, el bit de liberación rel_old de la instrucción NV también se pone a cero. De hecho, se puede *reutilizar* para el renombre de NV el mismo registro físico manteniendo intacto el mapeo y sin necesidad de asignar un nuevo registro físico.

La Tabla 4.1 expresa de forma algorítmica las condiciones de liberación de los registros físicos:

```
Liberación al Commit de LU:

    if ( ROS[LUid].rlxx == 1 ) then
       liberar(reg. físico #ROS[LUid].Pxx)

Liberación al Decodificar NV:

    if ( LUsT[rd].c == 1 ) then
       liberar(reg. físico #MT[rd])    /* reutilización */

/* esta actúa como la liberación convencional: */

Liberación al Commit de NV:

    if ( ROS[NVid].rel_old == 1 ) then
       liberar(reg. físico #ROS[NVid].old_pd)
```

**Tabla 4.1** Condiciones de liberación en el mecanismo básico.

## 4.2 Mecanismo Extendido

El mecanismo básico que se acaba de describir es sencillo, pero las posibilidades que tiene de liberar registros de forma anticipada son limitadas. Cada vez que una instrucción que redefine un registro (NV) se decodifica, si todavía existe algún salto previo a ella pendiente de resolverse (NV es especulada) se pierde la oportunidad de liberar el registro de forma anticipada. Se va a ver ahora un mecanismo extendido que tiene en cuenta este problema y que por lo tanto soporta liberaciones que se van a llamar *especuladas*. La idea que hay detrás de este mecanismo extendido es muy similar a las copias que se hacen de la MT para recuperarse de forma correcta después de un salto mal predicho. Para aquellas instrucciones NV que no son especuladas se van a aplicar las mismas reglas ya explicadas en el mecanismo básico. Además, la implementación

que se propone va a eliminar las acciones que el mecanismo convencional realizaba para liberar registros en la etapa de *commit* de la instrucción NV (old_pd). Con esto, desaparece el coste que supone almacenar los campos old_pd y rel_old en el ROS.

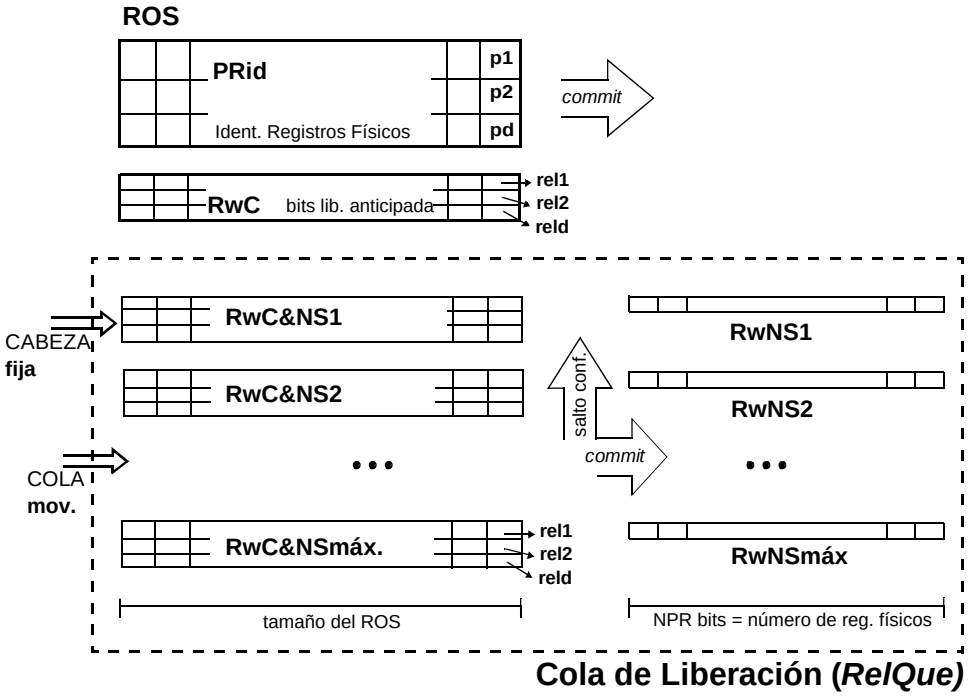


**Figura 4.4** Mecanismo extendido.

### 4.2.1 Recursos *Hardware*

La Figura 4.4 muestra los componentes que soportan el mecanismo extendido. Del mecanismo básico se mantienen los identificadores de los registros físicos y el vector de bits de liberación anticipada al *commit* -incluidos en el ROS extendido-. Por separado forman las dos estructuras que se llaman respectivamente *PRid* (*Physical Register identifiers*) y *RwC* (*Release when Commit*).

Se va a denominar *Cola de Liberación* (*Rel*ease*Que*ue) a la estructura clave del diseño extendido. En la dimensión vertical la cola actúa como una FIFO con tantos niveles ocupados como la cantidad de saltos que existen pendientes de confirmarse. De esta forma, la RelQue debe dimensionarse verticalmente con tantos niveles como el número de saltos pendientes que soporta el procesador.

Un determinado nivel de la RelQue almacena planificaciones de *liberaciones especuladas*. Se dice que una liberación es especulada siempre que la instrucción NV que la origina sea especulada. El nivel número *n* almacena aquellas planificaciones que dependen de la validación de los *n* saltos pendientes más viejos. Cada nivel en la RelQue contiene a su vez dos tipos de estructuras: una matriz de tres x ROS bits que se denomina *RwC&NSn* (RwC&NS1, RwC&NS2,...) y

un vector de NPR bits que se denomina *RwNSn* (RwNS1, RwNS2,...). Se describe a continuación la utilidad de cada una de estas estructuras:

- RwC&NSn (*Release when Commit and Non Speculative*), almacena las liberaciones especuladas que deben sincronizarse con la etapa de *commit* de instrucciones LU. La identidad del registro físico que debe liberarse está codificada en la estructura PRid. De la misma forma que lo hace el ROS, hace falta que en la dimensión horizontal todas las estructuras RwC&NSn soporten una operación de movimiento de izquierda a derecha.

- RwNSn (*Release when non Speculative*), almacena las liberaciones especuladas que provienen de instrucciones LU que ya han alcanzado la etapa de *commit*. El almacenamiento se hace de forma codificada (un registro físico = un bit).

En el siguiente apartado se va a ver cómo funcionan las distintas operaciones que se realizan sobre la RelQue.

### 4.2.2 Control

La idea básica del funcionamiento está en añadir a la RelQue un nuevo nivel cada vez que se decodifica un salto. Las instrucciones NV que se decodifiquen después de dicho salto seguirán consultando la tabla LUsT para identificar a las instrucciones LU y planificar liberaciones. Pero ahora la planificación de dichas liberaciones se va a hacer en la estructura RelQue, en RwC&NSn si la instrucción LU todavía no ha alcanzado la etapa del *commit* o en RwNSn en el caso contrario. Más adelante, a la vez que las predicciones de los saltos se van confirmando, las liberaciones especuladas planificadas tanto en RwC&NSn como en RwNSn se mueven hacia RwC (es el movimiento *salto conf.* de la Figura 4.4). En un salto mal predicho, se eliminan de la RelQue tanto la entrada correspondiente a dicho salto como todas aquellas más jóvenes. De esta forma, se anulan todas las liberaciones especuladas que hayan sido planificadas en el camino mal predicho.

Se va a ver ahora en detalle los distintos pasos de control necesarios para gestionar la RelQue y para liberar los registros. En la Tabla 4.2 se puede encontrar una explicación más formal de estos mismos pasos.

Se va a suponer que los saltos pueden verificarse fuera de orden y para que se entienda mejor, se supone también que la cabeza de la FIFO RelQue ocupa un lugar fijo. Su último nivel ocupado es el marcado en la Figura 4.4 con el puntero COLA.

#### PASO 1. DECODIFICACIÓN DE UNA INSTRUCCIÓN DE SALTO

Se añade a la RelQue un nuevo nivel con todos los bits puestos a cero. Para ello, basta con incrementar el puntero COLA.

### PASO 2. DECODIFICACIÓN DE UNA INSTRUCCIÓN NV ESPECULADA

En el nivel de la RelQue apuntado por COLA se marca la liberación especulada de un determinado registro físico p. Para una instrucción NV que posee *n* saltos previos a ella pendientes de confirmarse, hay que distinguir dos casos:

- si la correspondiente instrucción LU todavía no ha alcanzado la etapa de *commit*, la liberación especulada del registro físico p se marca en la matriz de bits RwC&NSn (RwC&NSn[LUid]rel$\underline{x}$ = 1, donde $\underline{x}$ = 1, 2 ó d).

- si la correspondiente instrucción LU ya ha alcanzado la etapa de *commit*, la liberación especulada del registro físico p se marca en el vector de bits RwNSn (RwNSn[p] = 1).

### PASO 3. SALTO MAL PREDICHO

La predicción del salto número *n* fue equivocada. Se eliminan de la RelQue todos los niveles desde *n* hasta COLA. Además, COLA se desplaza para apuntar al nivel *n-1*.

### PASO 4. SALTO CONFIRMADO

La predicción del salto pendiente número *n* es correcta. Todos los niveles localizados entre las entradas *n* y COLA de la RelQue se mueven hacia RwC (consultar el ejemplo de la Figura 4.5.a). Al mismo tiempo, se hace la operación *OR* entre las entradas *n* y *n-1*. Un poco más adelante, se verá cómo el caso de *n* = 1 exige algo más de trabajo.

### PASO 5. EFECTOS DE LA ETAPA DE *COMMIT* EN LA RELQUE

Ya se ha dicho que la RelQue necesita un movimiento FIFO de izquierda a derecha en las estructuras RwC&NSn. De esta forma, todos los bits que hay en las matrices de bits RwC&NSn de cada nivel se desplazan hacia la derecha tantas posiciones como el número de instrucciones que hacen el *commit*.

Una instrucción LU puede alcanzar la etapa de *commit* antes de que su instrucción par NV deje de ser especulada. En este caso, debido a que la predicción del salto todavía puede confirmarse, todas las planificaciones asociadas a las instrucciones que están haciendo el *commit* se mueven desde la cabeza de *commit* de RwC&NSn hacia RwNSn (es la operación de *Marcado* de la Figura 4.5.b). Este movimiento requiere decodificar los identificadores de los registros físicos guardados en la cabeza del ROS.

### PASO 6. LIBERACIÓN DE REGISTROS FÍSICOS

Como resultado de todos los pasos previos, los registros podrán liberarse en la etapa de *commit* de las instrucciones LU o cada vez que se confirme el salto más viejo. En el primer caso, para todas las instrucciones que hacen el *commit*, se liberan los registros planificados en las entradas de RwC (operación de *Liberación al Commit* de la Figura 4.5.b). En el segundo caso, cuando se confirma el salto más viejo, se liberan los registros planificados en RwNS1 (operación de *Libe-*

*ración al Confirmar Salto* de la Figura 4.5.c). Este es el trabajo que hay que añadir para el caso de $n = 1$ y que se había avanzado al describir el PASO 4.

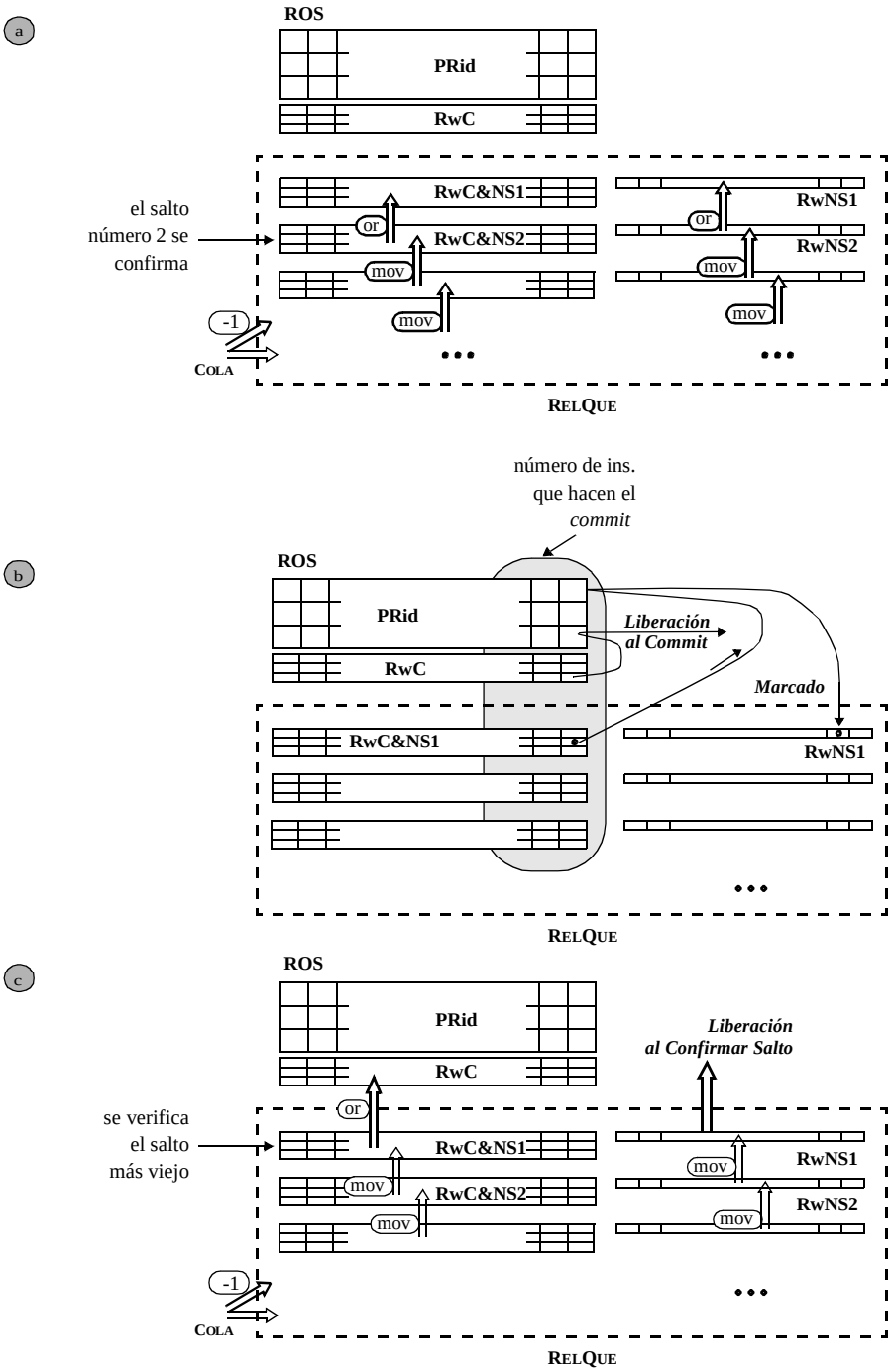


**Figura 4.5** Se confirma el segundo salto más viejo **(a)**, operaciones de *Marcado* en RwNSn y de *Liberación al Commit* **(b)**, la confirmación del salto más viejo activa la operación de *Liberación al Confirmar Salto* **(c)**.

Las reglas que han de seguirse para el caso de decodificar una instrucción NV que no sea especulada coinciden con las que se aplican en el mecanismo básico:

- si la correspondiente instrucción LU todavía no ha alcanzado la etapa de *commit*, la liberación se planifica en RwC.
- si la correspondiente instrucción LU ya ha alcanzado la etapa de *commit*, la liberación puede realizarse de forma inmediata. Este es un tercer caso donde el mecanismo extendido también libera registros (*Liberación al Decodificar NV* que no es especulada y LU ya ha hecho el *commit*). Como se dijo en el mecanismo básico, el mismo registro físico se puede *reutilizar* para el renombre de NV sin necesidad de asignar uno nuevo.

Para finalizar, la Tabla 4.2 resume todos los pasos anteriores utilizando notación algorítmica:

| | |
|---|---|
| Paso 1 | COLA++;<br>$RwC\&NS_{COLA}$[1:ROStam].rlxx = 0 y $RwNS_{COLA}$[1:#reg.fís.] = 0 |
| Paso 2 | case LU ya *commit*: $RwNS_{COLA}$[Tabla de Mapeo[rd]] = 1<br>case LU en entrada *i* del ROS: $RwC\&NS_{COLA}$[*i*].rlxx = 1 |
| Paso 3 | COLA = *n* - 1 |
| Paso 4 | RelQue[*n* - 1] = RelQue[*n* - 1] *or* RelQue[*n*];<br>for (j = *n* + 1) to COLA do RelQue[j - 1] = RelQue[j] ;<br>donde RelQue[j]= <$RwC\&NS_j$,$RwNS_j$><br>COLA--; |
| Paso 5 | /* la instrucción *i* está haciendo *commit*: */<br>if ( $RwC\&NS_j$[*i*].rlxx == 1 ) then $RwNS_j$[PRid[*i*].Pxx] = 1 |
| Paso 6 | /* la instrucción *LU* está haciendo *commit*: */<br>*Liberación al Commit*:<br>if ( RwC[*LUid*].rlxx == 1 ) then<br>liberar(reg. físico #PRid[*LUid*].Pxx)<br>/* el salto más viejo se está confirmando: */<br>*Liberación al Confirmar Salto*:<br>if ( $RwNS_1$[*i*] == 1 ) then<br>liberar(reg. físico #*i*)<br>/* la instrucción *NV* no es especulada: */<br>*Liberación al Decodificar NV*:<br>if ( LUsT[rd].c == 1 ) then<br>liberar(reg. físico #MT[rd]) /* reutilización */ |

**Tabla 4.2** Algoritmo detallado de la gestión de la RelQue y de la liberación de los registros.

## 4.3 Implicaciones del Modelo de Excepciones

Entre la instrucción que hace el último uso de un registro lógico r (LU) y su redefinición más cercana (NV) puede existir una tercera instrucción que provoque la generación de una excepción. En este caso, pueden aparecer problemas si el registro físico asignado a la versión actual de r ya ha sido liberado por LU tras una planificación de liberación anticipada.

La función de tratamiento de la excepción guarda el PC de la instrucción que provoca la excepción para reejecutarla más adelante. De la misma forma, si la excepción precisa de un cambio de contexto también se guardan todos los registros lógicos en el *Bloque de Control de Procesos*. Con esto, la función puede guardar para r un valor que puede ser distinto del último valor almacenado en ese registro lógico. Más adelante, cuando el contexto se restaure, el mismo valor incorrecto será de nuevo guardado en r. Sin embargo, el valor asociado a r realmente no importa ya que el *hardware* de liberación anticipada sólo libera una versión de un registro r cuando está garantizado que la siguiente operación sobre r será una escritura.

Por otro lado, Smith y Pleszkun definieron en [SP85] lo siguiente:

*"Una excepción es precisa si el estado del proceso guardado coincide con el modelo secuencial de ejecución del programa, donde una instrucción finaliza su ejecución antes de que comience la ejecución de la siguiente instrucción"*

Si la anterior definición se sigue como condición habitual de ser preciso, se podría concluir diciendo que el mecanismo presentado para liberar anticipadamente registros no cumple la condición habitual de ser preciso. Sin embargo, esta definición de excepciones precisas es suficiente pero no necesaria para garantizar una recuperación frente a excepciones correcta. La optimización que aquí se ha propuesto es segura en el sentido de que está garantizado que esos valores incorrectos de los registros no van a ser utilizados más por parte del programa. Optimizaciones en esta misma línea, aunque en *software* ya han sido propuestas por otros autores [LPE+99].

A continuación se van a ver los resultados más representativos obtenidos por los mecanismos propuestos. En el siguiente apartado se hará un análisis del coste de implementación de las estructuras clave para soportar la liberación anticipada.

## 4.4 Resultados

Los dos esquemas *hardware* para la liberación anticipada de registros presentados se evaluaron utilizando el simulador fuera de orden incluido en la herramienta de simulación SimpleScalar v3.0 [BA97]. Como se dijo en el Capítulo 3, este simulador se modificó para incluir las distintas técnicas de renombre propuestas. En este caso, a partir del *simulador convencional* se desarrollaron los simuladores para la liberación anticipada utilizando el mecanismo *básico* y el mecanismo *extendido*. La información relativa a los parámetros de la arquitectura utilizados y a los programas de prueba y sus entradas se puede encontrar en el apartado *Entorno Experimental* del Capítulo 3.

Para un diseño dado, una liberación anticipada de registros podrá utilizarse con dos objetivos:

- Incrementar el rendimiento (IPC) mientras se mantiene fijo el tamaño del banco de registros. Este diseño tiene sentido para procesadores donde el banco de registros es pequeño y permite mejorar el IPC sin afectar al tiempo de ciclo.

- Reducir el tamaño del banco de registros sin perder rendimiento. Este diseño tiene sentido cuando el banco de registros se encuentra en el camino crítico del procesador y su reducción implica una reducción en el tiempo de ciclo.

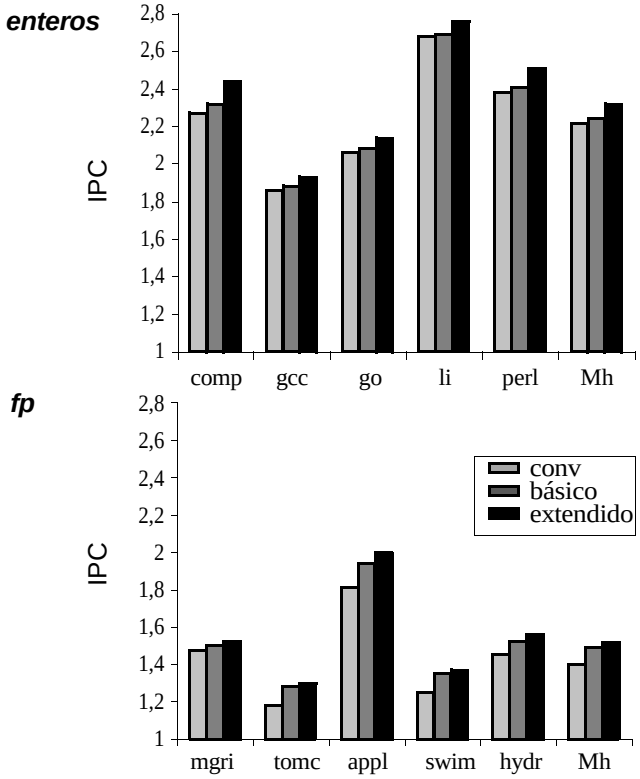


**Figura 4.6** IPC para la liberación anticipada. Mecanismos *básico* y *extendido versus* el mecanismo convencional (*conv*). El banco de registros es de 48+48 registros.

Se verá primero el impacto en el rendimiento de la liberación anticipada para un procesador con un banco de registros muy ajustado de $48_{ent}+48_{fp}$ registros. La Figura 4.6 muestra el número promedio de instrucciones que hacen el *commit* por ciclo (IPC) para cada programa de prueba. También se puede observar el valor de la media armónica (Mh) tanto para enteros como para fp. Se han comparado los tres mecanismos de liberación: el convencional (*conv*), el mecanismo *básico* y el *extendido*. Se puede observar como las ganancias en el rendimiento con liberación anticipada son mucho mas importantes para códigos fp que para programas enteros. El mecanismo básico proporciona un *speed-up* promedio sobre el mecanismo convencional de un 6% para programas fp, mientras que para los programas enteros dicho *speed-up* es prácticamente nulo. Sin embargo, el mecanismo extendido alcanza un *speed-up* promedio sobre el mecanismo convencional de un 8% y un 5% respectivamente, para programas fp y para códigos enteros. Estos resultados eran esperados ya que, en general los programas fp ejercen mucha más presión sobre el banco de registros y los programas enteros poseen mucha más cantidad de instrucciones de sal-

to. Recordar que los saltos limitan la efectividad de la liberación anticipada, retrasando sus efectos hasta realizar la verificación de la predicción del salto.

A continuación, se ve cómo se comporta la liberación anticipada bajo diferentes condiciones de presión sobre el banco de registros. Se considera para ello todo el rango de tamaños del banco de registros (desde $40_{ent}+40_{fp}$ registros hasta $160_{ent}+160_{fp}$ registros) y se calcula el IPC para todas las configuraciones y todos los mecanismos. La Figura 4.7 muestra este IPC para las tres políticas de liberación. Un primer punto importante que se observa es que las mejoras de la liberación anticipada disminuyen conforme aumenta el número de registros, además de la baja presión que se ejerce sobre el banco de registros para los programas de tipo entero. Para este tipo de programas, un banco de registros sobrado (con más de 72 registros) no tiene sentido, con o sin liberación anticipada. Excluyendo los diseños sobrados, la liberación anticipada siempre obtiene ventaja en el rendimiento, siendo la diferencia más importante para programas fp. En las Figuras B.4 a B.6 del Apéndice B se encuentra la misma medida de esta Figura 4.7 pero por separado para cada uno de los programas de prueba.

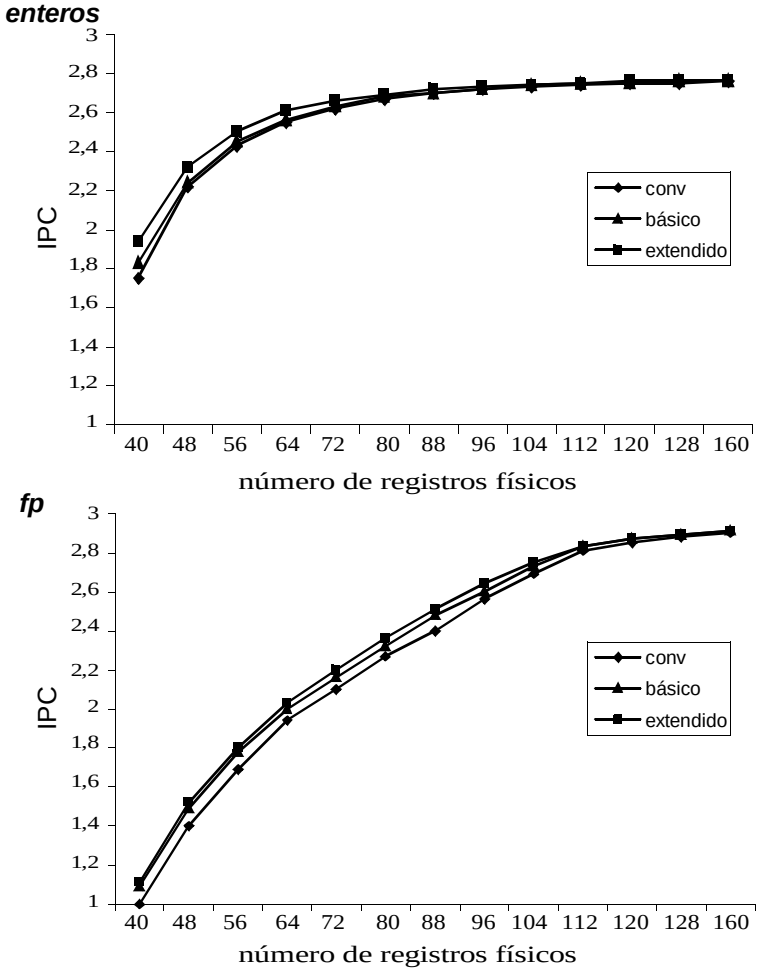


**Figura 4.7** Media armónica de IPC *versus* el número de registros físicos para los mecanismos *convencional*, *básico* y *extendido*.

La mejora del rendimiento con el mecanismo básico es sólo visible para bancos de registros ajustados. Con bancos de registros físicos de tamaño $64_{ent}+64_{fp}$, éste mecanismo alcanza en programas fp un *speed-up* promedio sobre el mecanismo convencional de un 3%. En programas enteros el *speed-up* promedio casi no es apreciable. Pero para bancos de registros muy ajustados la liberación anticipada obtiene beneficio para los dos tipos de programas. Para una configuración del banco de registros de tamaño $40_{ent}+40_{fp}$, se obtiene un *speed-up* del 5% para códigos enteros y del 9% para códigos fp.

Comparando los dos mecanismos de liberación (*básico* y *extendido*), se ve que el mecanismo extendido se comporta especialmente bien para programas de tipo entero, mientras que para códigos fp ambos mecanismos se comportan de forma similar. Los códigos fp obtienen importantes ganancias con tamaños del banco de registros entre 40 y 104 registros. En este rango, el mecanismo extendido obtiene un *speed-up* que va decrementando de forma suave desde un 10% hasta un 2% (con 64 registros es de un 5%). Este mismo decremento va desde un 9% hasta un 1% con el mecanismo básico. Sin embargo, programas con una presión en el banco de registros muy alta pueden experimentar mejoras mucho mayores con el mecanismo extendido:

*hydro2d* consigue un 12% de *speed-up* con 40+40 registros

*tomcatv* consigue respectivamente un 16%, 12% y 8% para 40+40, 56+56 y 88+88 registros.

Los códigos de tipo entero también se benefician de la liberación anticipada pero sólo para bancos de registros muy ajustados (entre 40 y 64 registros). Dentro de este rango, el mecanismo extendido obtiene un *speed-up* que va decrementando desde un 11% hasta un 2% (desde un 5% hasta un 0% con el mecanismo básico). En el caso particular de *compress* se obtiene hasta un 13% con un banco de 40 registros.

Finalmente, se puede utilizar la liberación anticipada como un medio para reducir los requerimientos de registros físicos en el procesador para un determinado nivel de rendimiento. Para ello, la Tabla 4.3 muestra varias configuraciones de registros que obtienen el mismo IPC. Esta tabla también muestra la reducción que se obtiene en el almacenamiento.

| **códigos fp** | | | **códigos ent.** | | |
|---|---|---|---|---|---|
| convencional | ***extendido*** | **reducción %** | convencional | ***extendido*** | **reducción %** |
| 69 | 64 | 7.2% | 64 | 56 | 12.5% |
| 79 | 72 | 9% | 72 | 64 | 11.1% |

**Tabla 4.3** Tamaños del banco de registros que obtienen el mismo IPC.

Para programas de tipo entero, podrá obtenerse un IPC de por ejemplo 2.5 con 64 registros y un mecanismo de liberación convencional o bien con sólo 56 registros y un mecanismo de liberación anticipada extendido. Lo anterior representa una reducción del 12.5% en el tamaño del banco de registros. De la misma forma y para programas de tipo fp, se puede mantener un IPC de 2.2 y reducir a la vez el tamaño del banco de registros en un 9%.

A continuación, se van a examinar otros resultados interesantes obtenidos por el mecanismo extendido. Son los dos tipos de resultados que se van reproduciendo a lo largo de toda esta Tesis para las estrategias óptimas.

### 4.4.1 *Utilización* de los registros

De la misma forma que se hizo en el Capítulo 2 para el renombre convencional, se ha medido el número promedio de los registros Asignados que se encuentran en los distintos estados (Vacío, Utilizado y Desocupado), asi como la *utilización* de los registros. Recordar que en ese Capítulo se definió *utilización* como el porcentaje del número de registros Utilizados sobre el número de registros Asignados.

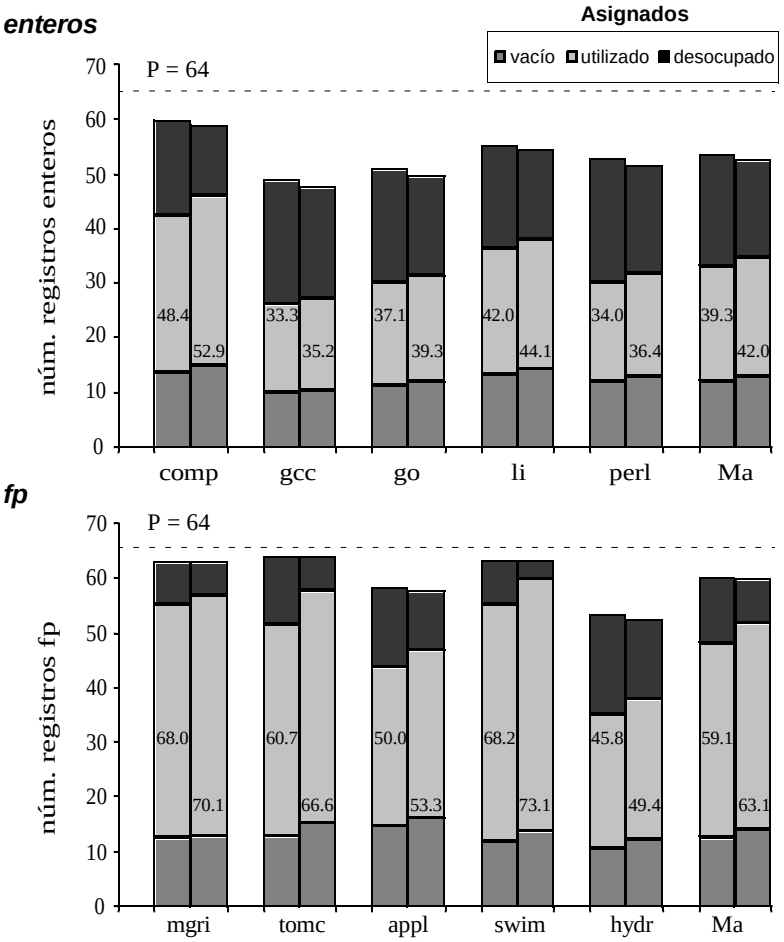


**Figura 4.8** Número de registros Asignados que se encuentran en los estados Vacío, Utilizado o Desocupado. Se comparan el mecanismo de liberación convencional y el de liberación anticipada extendido. 64ent+64fp registros.

La Figura 4.8 compara los resultados del experimento anterior para $64_{ent}$+$64_{fp}$ registros físicos y empleando la liberación anticipada frente a la convencional. La columna de la izquierda vuelve a reproducir los datos que se presentaron en la Figura 1.4 (mecanismo convencional) y se comparan con los obtenidos por el mecanismo extendido de liberación anticipada (columna de

la derecha). Los valores de utilización se han puesto dentro de las correspondientes columnas. En la Tabla A.3 del Apéndice A se pueden consultar los valores con los que reproducir las columnas de la liberación anticipada que aparecen en la Figura 4.8.

Una liberación de registros anticipada mantiene una presión sobre el banco de registros muy parecida a la del mecanismo convencional. En general, sólo se observa un incremento promedio en el número de registros que se encuentran Libres de un 1% para todo tipo de programas. Sin embargo, se puede observar como al liberar los registros físicos de forma anticipada, en el mecanismo extendido disminuye considerablemente el número de registros que se encuentran en estado Desocupado. Este decremento en promedio es de un 15% para los programas enteros y de un 33% para los programas fp. En algunos casos particulares se observa como ese decremento llega a ser de hasta un 26% (*compress*) o hasta un 50% y 59% (*tomcatv* y *swim*). Por otro lado y también en promedio, la utilización de los registros que en los programas enteros era baja (39.3%) aumenta con la liberación anticipada hasta un 42%. Para fp la utilización pasa de un 59.1% a un 63.1%. Finalmente, para los dos tipos de programas se ha observado como efecto negativo que el número de registros en estado Vacío aumenta de forma ligera. En promedio este aumento es de un 1%.

### 4.4.2 Adelanto en la ejecución de las instrucciones

La liberación anticipada de registros permite disponer de algunos registros más para asignar a las instrucciones. De esta forma aparecerán instrucciones que van a poderse ejecutar con anterioridad al momento en que se hubiesen ejecutado con el esquema de liberación convencional. Esta es la definición que se dio en el Capítulo 3 para considerar que una instrucción se *anticipa*.

Siguiendo las mismas reglas que se presentaron en el Capítulo 3 en el apartado que lleva el mismo nombre que éste, se ha evaluado tanto el porcentaje de instrucciones anticipadas como el número de ciclos que estas instrucciones se anticipan bajo el mecanismo de liberación anticipada. También se ha supuesto un banco de registros con $64_{ent}+64_{fp}$ registros físicos.

La Figura 4.9.a muestra el porcentaje de *loads* que fallan en memoria *cache*, desglosados en anticipados y no anticipados. En promedio, para programas fp se anticipa el 46% de los *loads* que fallan en memoria *cache*, mientras que para los programas enteros este número es del 3%. Dos casos interesantes son *swim* y *tomcatv* que anticipan el 62.6% y el 59.2% de los *loads* que fallan en memoria *cache* respectivamente. La Figura 4.9.b muestra el porcentaje de saltos condicionales que son mal predichos, desglosados en anticipados y no anticipados. Se observa que en promedio, para programas enteros se anticipa el 2% de los saltos condicionales mal predichos y para los fp el 19%. De nuevo *tomcatv* es uno de los que más se beneficia de la anticipación de instrucciones ya que anticipa el 57.4% de sus saltos condicionales mal predichos.

Otra estadística interesante es el número promedio de ciclos que una instrucción es anticipada. Se ha obtenido que las instrucciones de *load* son anticipadas un promedio de 7.2 ciclos para los programas enteros y 16.2 ciclos para fp. Para los saltos condicionales esos mismos cálculos dan 3.8 y 13.2 ciclos respectivamente. Para las instrucciones de división, multiplicación y simples los resultados son respectivamente 2, 3.4 y 4 para los programas enteros y 9.3, 0.1 y 4.4 para los fp.

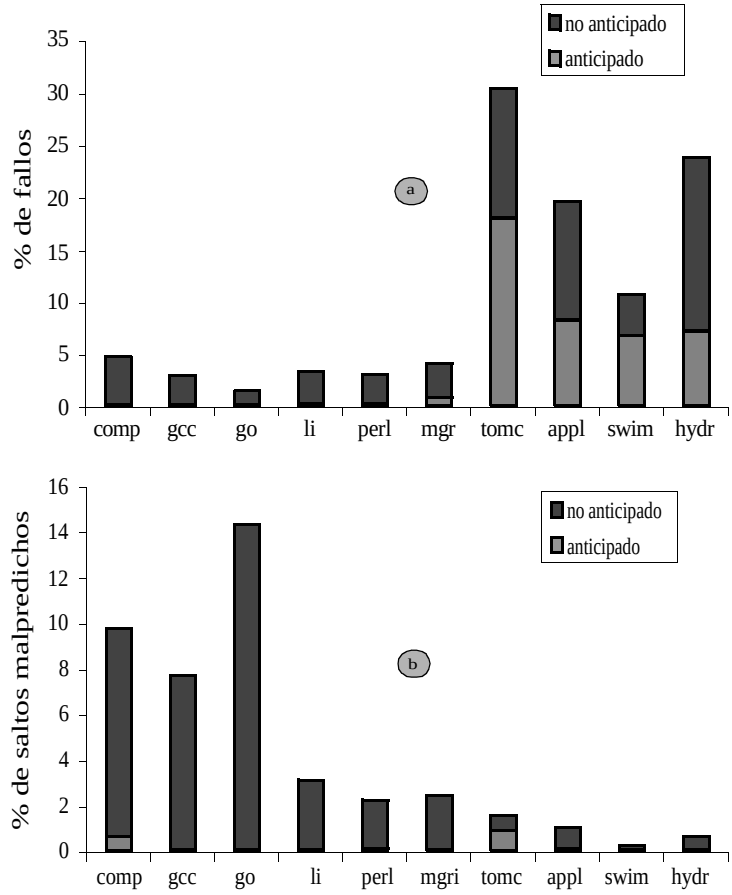


**Figura 4.9** Porcentaje de *loads* que fallan en *cache* -anticipados y no anticipados- **(a)** y porcentaje de saltos condicionales que son mal predichos -anticipados y no anticipados- **(b)**. Mecanismo extendido de liberación anticipada.

Disponer de más registros físicos permite que algunas instrucciones puedan entrar en la ventana de lanzamiento y se ejecuten antes de lo que lo harían con el mecanismo convencional. Como han confirmado las estadísticas anteriores, de nuevo este beneficio es más importante para programas fp que para enteros. Se ha visto que para un mismo número de registros físicos en el procesador, la liberación anticipada permite tener más instrucciones en vuelo que el esquema convencional. Sin embargo, especialmente en los programas enteros, los saltos mal predichos impiden que el procesador pueda extraer todo el potencial de este mecanismo.

## 4.5 Implementación

A continuación se va a analizar el impacto que supone la implementación del mecanismo extendido. Es un análisis sobre el coste de almacenamiento, tiempo de ciclo y consumo de energía que tendrían aquellas partes del *hardware* propuesto que se han considerado más críticas. El análisis de la influencia sobre el tiempo de ciclo y el consumo de energía se ha realizado utili-

zando el modelo de tiempo de acceso y potencia de Rixner et al. para una tecnología de 0.18 μm. [RDK+00].

### 4.5.1 Almacenamiento

El coste de almacenamiento del mecanismo extendido se ha calculado utilizando algunos de los parámetros del procesador Alpha 21264 [Gwe96]. Este procesador puede gestionar hasta un máximo de veinte saltos pendientes de confirmarse, tiene un ROS de 80 entradas y un banco de registros físico con 80 registros enteros y 72 registros fp. Con estos parámetros todas las estructuras que necesita el mecanismo extendido (RelQue y ROS en la Figura 4.4) ocupan alrededor de 1.22 KBytes. Las dos tablas LUsT (entera y fp) añaden alrededor de 128 Bytes. Esta cantidad es mucho menor que el coste de almacenamiento que ocupan las tablas necesarias para la predicción de los saltos en este procesador (alrededor de 4.4 KBytes) [Gwe96]. Por lo tanto, se considera que es un coste de almacenamiento relativamente pequeño y que su implementación es viable en el contexto de microprocesadores de alto rendimiento.

### 4.5.2 Tiempo de Ciclo

Para analizar el coste relativo al tiempo de ciclo se ha considerado la tabla LUsT como la estructura clave del mecanismo. Esta tabla, para un procesador superescalar de grado ocho, puede implementarse como una SRAM con gran cantidad de puertos de lectura y escritura. La hemos supuesto dividida en tres partes: LUsT.ROSid, LUsT.Kind y LUsT.c. Para los parámetros que se han utilizado en las evaluaciones (consultar en el Capítulo 3 el apartado de *Entorno Experimental*), cada parte dispone de 32 entradas de anchura 7, 2 y 1 bit respectivamente. El total de puertos para LUsT.ROSid es de 56 -32 de lectura y 24 de escritura-; para LUsT.Kind es de 32 -8 de lectura y 24 de escritura-; para LUsT.c es de 56 -8 de lectura y 48 de escritura-. La Figura 4.10.a presenta los valores calculados para LUsT.ROSid al ser ésta la que mayor tiempo de acceso necesita. En esta figura también se muestran esos valores para los bancos de registros entero y fp considerados en el diseño. El tamaño del banco de registros varía desde 40 hasta 160 registros físicos. El número de puertos para cualquier banco de registros entero es de 48 y para cualquier banco de registros fp es de 54. En la Figura 4.10.a se puede observar como el tiempo de acceso de la tabla LUsT está por debajo de cualquier tamaño de banco de registros. Se encuentra un 31% por debajo del banco de registros entero de menor tamaño (40 registros físicos).

### 4.5.3 Consumo de Energía

Para evaluar el consumo de energía se ha considerado de nuevo la tabla LUsT como la estructura clave. Respecto de la RelQue, el número total de bits con un valor de uno en toda esta estructura, se encuentra limitado por el tamaño del ROS. Para ser más exactos, hay tantos bits con valor de uno como la cantidad de instrucciones con registro destino que aún no han alcanzado la etapa de *commit*. Por lo tanto, si implementamos la RelQue como un verdadero registro de desplazamiento de dos dimensiones, la mayoría del tiempo los valores que han de desplazarse son

ceros. Esta es la razón por la que de forma segura se supone nula la contribución que la RelQue hace al consumo de energía del mecanismo extendido.

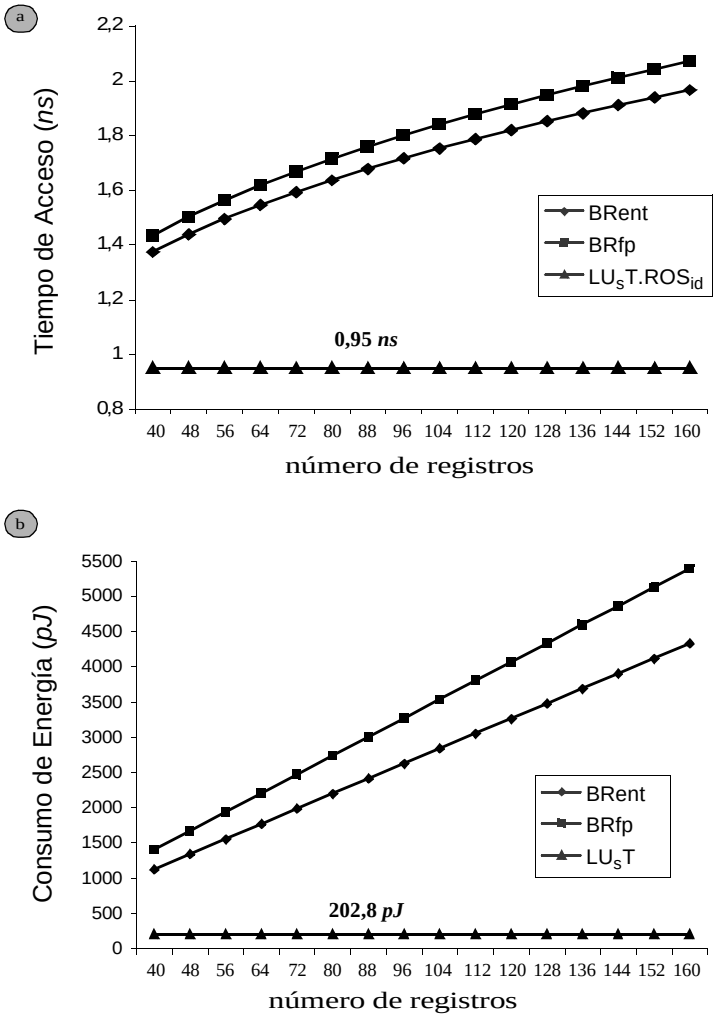


**Figura 4.10** Tiempo de acceso y consumo de energía para la tabla LUsT y el banco de registros. En el eje de ordenadas se varía el número de registros.

En la Figura 4.10.b se puede ver también el consumo de energía de la tabla LUsT. De nuevo se observa como la tabla LUsT sólo consume el 18% de la energía consumida por el menor de los bancos de registros (40 registros).

En el apartado de resultados, se ha visto cómo el mecanismo de liberación anticipada de registros puede utilizarse como un medio de reducir el número de registros necesarios para obtener un determinado nivel de rendimiento. Por ejemplo, se vio que se obtiene el mismo valor de IPC con un mecanismo convencional de renombre y banco de registros con 64 registros enteros + 79 registros fp que con un mecanismo con liberación anticipada y un banco registros con 56 registros enteros + 72 registros fp. Al comparar el consumo de energía de estas dos alternativas, se observa lo siguiente:

$E_{convencional}(BR_{64ent} + BR_{79fp})$ = 1769.9 pJ + 2702.6 pJ = 4472.5 pJ,

y con liberación anticipada:

$E_{liberación\ anticipada}(BR_{56ent} + BR_{72fp} + 2x\ LUsT)$ =

1555 pJ + 2469.4 pJ + 2x 202.8 pJ = 4430 pJ.

La energía consumida es ligeramente menor -42.5pJ- para el caso de la liberación anticipada. La cantidad de energía que se ahorra al reducir el tamaño del banco de registros es comparable con la energía que consumen las estructuras que permiten esa reducción.

## 4.6 Conclusiones

Los esquemas de renombre de registros convencionales liberan los registros de forma conservadora, incrementando innecesariamente la necesidad de registros en los procesadores con ejecución fuera de orden. La liberación anticipada de registros se presenta como otra posible herramienta de diseño que hace que el banco de registros sea un componente menos crítico en el procesador y que además posee una utilidad doble:

- Permite incrementar el IPC mientras se mantiene el mismo tamaño del banco de registros. Esta utilidad sólo tiene sentido para sistemas con bancos de registros ajustados. El IPC mejora sin degradar el tiempo de ciclo.

- Permite reducir el tamaño del banco de registros para ajustar su tiempo de acceso al tiempo de ciclo del procesador, sin perder IPC. Esta utilidad tiene sentido tanto para sistemas con bancos de registros sobrados como ajustados, pero que se encuentran en el camino crítico del procesador.

En este Capítulo se han presentado dos mecanismos dedicados a liberar los registros con anticipación y que suponen un aumento incremental tanto en complejidad como en rendimiento obtenido. Tales mecanismos soportan tanto ejecución especulada como recuperación precisa frente a excepciones. Se ha visto que el *hardware* añadido para la implementación de la liberación anticipada se encuentra fuera de cualquier camino crítico del procesador y que es perfectamente viable en coste de almacenamiento.

La evaluación realizada ha mostrado mejoras prometedoras especialmente para códigos numéricos y para un amplio rango de tamaños de bancos de registros ajustados. En promedio, varían desde un 10% hasta un 2% en la medida en que el tamaño del banco de registros incrementa hasta alcanzar el tamaño sobrado. En algunos programas se llegan a alcanzar mejoras de hasta un 16% para bancos de registros ajustados. En códigos de tipo entero, la liberación anticipada de registros sólo resulta efectiva para bancos de registros muy ajustados donde se pueden alcanzar mejoras de hasta un 11%.

La utilización de los registros aumenta con la liberación anticipada de un 39.3% a un 42% y de un 59.1% a un 63.1% para programas enteros y fp respectivamente. El porcentaje de registros asignados cuyo contenido no se lee -desocupados- disminuye un 15% en programas enteros y un

33% en programas fp.

De forma alternativa, la liberación anticipada de registros se puede utilizar para ajustar el tamaño del banco de registros mientras se sigue manteniendo el mismo IPC. En lo que se refiere a microarquitecturas típicas, se ha visto que el tamaño del banco de registros puede reducirse en un 12.5% y un 9% para códigos enteros y fp respectivamente, sin reducción de IPC. Esto es un punto importante a considerar ya que una reducción en el tamaño del banco de registros se traduce directamente en un tiempo de acceso menor.

Para finalizar, también se ha observado cómo el impacto relativo de los dos mecanismos (*básico* y *extendido*) depende mucho del tipo de código al que se aplica. Esto sugiere que un esquema híbrido que utilizara la implementación básica para los registros fp y la implementación extendida para los registros de tipo entero podría dar como resultado un nuevo mecanismo con un muy buen equilibrio entre coste y rendimiento.

## 4.7 Referencias


[BA97] D. Burger, and T.M. Austin, *The SimpleScalar Tool Set v2.0*, Technical report 97-1342, University of Wisconsin-Madison, Computer Science Department, June 1997.

[BDA01a] R. Balasubramonian, S. Dwarkadas, and D.H. Albonesi, "Dynamically Allocating Processor Resources between Nearby and Distant ILP," *in Proceedings of the 28th International Symposium on Computer Architecture* (ISCA 01), pp. 26-37, June 2001.

[BDA01b] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi, "Reducing the Complexity of the Register File in Dynamic Superscalar Processors," *in Proceedings of the 34th International Symposium on Microarchitecture* (MICRO 01), pp. 237-249, December 2001.

[FJC96] K.I. Farkas, N.P. Jouppi, and P. Chow, "Register File Considerations in Dynamically Scheduled Processors," *in Proceedings of the 2nd International Symposium on High-Performance Computer Architecture* (HPCA 96), pp. 40-51, February 1996.

[Gwe96] L. Gwennap, "Digital 21264 Sets New Standard," *Microprocessor Report*, vol. 10, no. 14, pp. 11-16, October 1996.

[HP87] W.W. Hwu, and Y.N. Patt, "Checkpoint Repair for Out-of-Order Execution Machines," *in Proceedings of the 14th International Symposium on Computer Architecture* (ISCA 87), pp. 18-26, June 1987.

[LPE+99] J.L. Lo, S. S. Parekh, S.J. Eggers, H. M. Levy, and D.M. Tullsen, "Software-Directed Register Deallocation for Simultaneous Multithreaded Processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 9, pp. 922-933, September 1999.

[MGV+00b] T. Monreal, A. González, V. Viñals y M. Valero, "Liberación Anticipada de Registros," *en XI Jornadas de Paralelismo*, pp. 21-27, Septiembre 2000.

[MPV93] M. Moudgill, K. Pingali, and S. Vassiliadis, "Register Renaming and Dynamic Speculation: an Alternative Approach," *in Proceedings of the 26th International Symposium on Microarchitecture* (MICRO 93), pp. 202-213, November 1993.

[MRF97] M.M. Martin, A. Roth, and C.N. Fischer, "Exploiting Dead Value Information," *in Proceedings of the 30th International Symposium on Microarchitecture* (MICRO 97), pp. 125-135, December 1997.

[MVG+01] T. Monreal, V. Viñals, A. González, and M. Valero, *Early Register Release*, Report interno del DIIS, RR-01-01, pp. 1-21, 2001.

[MVG+02] T. Monreal, V. Viñals, A. González, and M. Valero, "Hardware Schemes for Early Register Release," *in Proceedings of the 31st International Conference on Parallel Processing* (ICPP 02), pp. 5-13, August 2002.

[RDK+00] S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U.J. Kapasi, and J. D. Owens, "Register Organization for Media Processing," *in Proceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 375-386, January 2000.

[SP85] J.E. Smith, and A. R. Pleszkun, "Implementation of Precise Interrupts in Pipelined Processors," *in Proceedings of the 12th International Symposium on Computer Architecture* (ISCA 85), pp. 36-44, June 1985.

C A P Í T U L O 5

# Asignación Tardía y Liberación Anticipada: el Esquema vp-LAER


## Resumen

*En los Capítulos 3 y 4 se han presentado y evaluado distintas técnicas* hardware *orientadas a atacar las dos ineficiencias más importantes que tiene el mecanismo de renombre de registros convencional y que se presentaron en el Capítulo 2. De todas estas técnicas se ha seleccionado la más eficiente para cada una de las ineficiencias y se han combinado en un sólo mecanismo. En este Capítulo se describe y evalúa este mecanismo combinado que se ha denominado vp-LAER. En vp-LAER además de retrasar la asignación de los registros físicos a las instrucciones, también se van a anticipar su liberación. En concreto, los registros físicos se van a asignar al final de la etapa de ejecución y se van a liberar cuando el procesador sepa con seguridad que ya no se van a utilizar más.*

*Vp-LAER es un mecanismo de renombre factible de ser implementado que mejora las prestaciones de cualquiera de los dos esquemas por separado. Vp-LAER va a incrementar significativamente la utilización de los registros, lo cual se traduce en la ejecución anticipada de instrucciones que con un esquema de renombre convencional se habrían parado debido a la falta de registros físicos. Con vp-LAER se va a poder obtener un* speed-up *importante para un determinado tamaño del banco de registros o se va a poder reducir el tamaño del banco de registros para un determinado nivel de rendimiento. Lo anterior es particularmente cierto para códigos fp para los cuáles la presión que se ejerce sobre el banco de registros es especialmente alta.*

## 5.1 El esquema de renombre vp-LAER

Siguiendo en la línea de toda esta Tesis de intentar reducir el número promedio de registros requeridos en la ejecución de un determinado programa, se va a seguir actuando sobre el mecanismo que controla tanto la asignación de los registros físicos como su liberación. Para ello se ha diseñado y evaluado la política de renombre denominada vp-LAER, que de forma simultánea realiza una asignación tardía de los registros (*Late Allocation*) basada en el mecanismo vp-DSY descrito en el Capítulo 3 y una liberación anticipada de los registros (*Early Release*) basada en el concepto de *Last-Use Next-Version* descrito en el Capítulo 4. Actualmente la propuesta vp-LAER se encuentra en proceso de revisión de la revista *Transactions on Computers* del *IEEE*.

Se comenzará recordando por separado los conceptos que se utilizan para la asignación y la liberación de los registros en vp-LAER para pasar después a presentar la arquitectura conjunta. Finalmente se verán más detalles de este mecanismo de renombre así como varios ejemplos de su utilización.

### 5.1.1 Asignación de los registros en vp-LAER

El mecanismo de renombre convencional asigna los registros físicos en la etapa de decodificación y renombre. Sin embargo, las instrucciones no necesitan realmente esos registros hasta que los resultados estén calculados. La razón de realizar esta asignación temprana radica en que los registros físicos se utilizan no sólo para almacenar valores sino también para seguir la pista de las dependencias entre las instrucciones. Este último uso de los registros físicos no necesita asignar registro físico en la etapa de decodificación. La asignación de un elemento de almacenamiento puede retrasarse si el procesador utiliza para seguir la pista de las dependencias un mecanismo aparte. En el Capítulo 3 de esta Tesis se describió un mecanismo de este tipo que se denominó registros virtuales-físicos (vp) en [GVG+97][GGV98][MGV+99a].

Los registros vp son etiquetas y por lo tanto no requieren de ningún tipo de almacenamiento físico. El número de registros vp que se necesitan con el objetivo de no parar el procesador debido a su agotamiento es igual al número máximo permitido de instrucciones en vuelo (número de entradas en la estructura de reordenación ROS) más el número de registros lógicos (enteros más fp). Cuando una instrucción se decodifica, a su registro lógico destino se le asigna un registro vp que se obtiene de una lista de *registros vp* que se encuentran *libres*. Más adelante, cuando la instrucción se encuentra en el último ciclo de su etapa de ejecución, al registro vp se le asigna un registro físico que se obtiene de una lista de *registros físicos* que se encuentran *libres*. Al alcanzarse la etapa de liberación de los registros, tanto el registro vp como el registro físico son devueltos a sus respectivas listas de registros libres. En los trabajos previos que utilizan la técnica de los registros virtuales-físicos (descritos en el Capítulo 3), la liberación actúa siguiendo el mecanismo convencional, pero la técnica vp-LAER va a utilizar el mecanismo de liberación anticipada más agresivo que se describió en el Capítulo 4 de esta Tesis y que se resume en la siguiente subsección.

Cuando una instrucción alcanza el último ciclo de su etapa de ejecución y no existe ningún registro físico libre para asignar, la instrucción tiene que esperar a volver a ser reejecutada más adelante con la esperanza de que algún registro físico sea liberado durante ese tiempo. Sin em-

bargo, si esa instrucción es la más vieja de todas las que están activas en el procesador, ninguna instrucción podrá alcanzar su etapa de *commit* y por lo tanto, ningún registro físico podrá liberarse. Para evitar esta situación de *abrazo mortal*, una política de asignación de registros factible consiste en garantizar a un número dado de las instrucciones más viejas en la estructura de reordenación (ROS) la disponibilidad de registros físicos cuando finalicen su ejecución. A este parámetro de la implementación se le denominó el *Número de Registros Reservados* (NRR). En pocas palabras, se puede decir que este esquema de asignación reserva NRR registros para asignarlos a las instrucciones más viejas que hay en el ROS y que poseen registro destino, mientras que el resto de registros físicos se siguen asignando bajo demanda (es decir, se asignan a las instrucciones que antes alcanzan su último ciclo de ejecución). En el Capítulo 3 ya se han dado todos los detalles sobre este esquema (vp-NRR) y su rendimiento.

El mecanismo de asignación vp-NRR no es la única solución que puede garantizar el evitar la situación de *abrazo mortal* que sufre la técnica de los registros virtuales-físicos. Encontrar el esquema de asignación de registros que sea el óptimo parece ser un problema irresoluble incluso aún teniendo un conocimiento perfecto de las referencias futuras de los registros. Sin embargo, se encontró una heurística implementable y que a la vez mejora en prestaciones al mecanismo vp-NRR y se acerca al esquema óptimo. Esta heurística también se ha descrito en el Capítulo 3 y se denominó *on-Demand with Stealing from Younger* (vp-DSY en [MGV+99a] y en [MGV+00a]). Vp-DSY sigue las siguientes dos reglas para asignar los registros:

- los registros se asignan a las instrucciones que más pronto pueden utilizarlos para almacenar un determinado valor. De esta forma, se minimiza el número promedio de registros que están siendo inutilizados.

- dadas dos instrucciones cualesquiera, si la ejecución de una de ellas debe retrasarse debido a la falta de registros físicos, la candidata más apropiada para ello es la instrucción más joven. Retrasar la ejecución de la instrucción más vieja podría a su vez retrasar su etapa de *commit*, lo cual se traduciría en retrasar también la etapa de *commit* de la instrucción más joven.

El esquema de renombre vp-LAER utiliza la técnica de asignación de registros vp-DSY. Con ello, a cada instrucción que ha de escribir en un registro se le asigna registro físico (si existen libres) en el último ciclo de su etapa de ejecución. Esto cumple el primero de los criterios anteriores ya que los registros se asignan a aquellas instrucciones que primero alcanzan su etapa de ejecución. Cuando una instrucción alcanza el último ciclo de su ejecución y no existe ningún registro físico disponible, se procede a comprobar si existe alguna instrucción más joven que ya dispone de registro físico asignado. Si se da el caso, va a ser mejor parar a la instrucción más joven en lugar de parar a la instrucción más vieja, cumpliendo así el segundo de los criterios. Lo anterior puede llevarse a cabo robando el registro asignado a la instrucción más joven y asignándolo a la más vieja. Si existe más de una instrucción joven con registro asignado, se elige a la más joven de todas.

La instrucción a la que se roba el registro debe reejecutarse de nuevo. Como ya se vio en el apartado de resultados del Capítulo 3, el factor de reejecución para la técnica vp-DSY es menor que el que posee la técnica vp-NRR. Vp-LAER reduce incluso más este efecto negativo debido a la utilización de la técnica de liberación anticipada, la cual proporciona un número extra de registros.

### 5.1.2 Liberación de los registros en vp-LAER

El renombre de registros convencional fuerza a que un registro físico se encuentre Desocupado desde la etapa de *commit* de la instrucción que lo utiliza por última vez (su instrucción *Last-Use*) hasta la etapa de *commit* de la primera instrucción que define su siguiente versión (su instrucción *Next-Version*). Como ya se dijo en el Capítulo 2 de esta Tesis, esta es una implementación sencilla que soporta tanto especulación frente a instrucciones de salto como un mecanismo de recuperación precisa frente a excepciones. Sin embargo, el procesador puede llegar a una situación donde debe parar ya que no existen registros físicos disponibles para asignar aunque una cantidad considerable de los registros estén en estado Desocupado. Por el contrario, vp-LAER es capaz de liberar los registros Desocupados mucho antes, permitiendo así la ejecución de instrucciones que de otra forma estarían paradas debido a la falta de registros. Utilizando vp-LAER los registros físicos podrán liberarse cuando instrucciones LU alcancen su etapa de *commit*.

La idea consiste en pasar completamente la responsabilidad de la liberación desde la instrucción NV hacia la instrucción LU. Esto se corresponde con la política de liberación anticipada que se ha descrito en el Capítulo 4 de esta Tesis y que denominó *Early Release* en [MVG+02]. Para cada registro se necesita identificar y marcar su instrucción LU. La identificación de toda instrucción LU y su posterior marcado puede hacerse de forma simple cuando se está decodificando su correspondiente instrucción par NV. Instrucciones LU que hayan sido marcadas por sus pares NV liberarán los registros cuando alcancen su etapa de *commit*. Disminuye así la necesidad de mantener dichos registros Desocupados hasta que alcance la etapa de *commit* la correspondiente instrucción NV. En [MVG+02] se denominó a este marcado como *planificación de una liberación anticipada* (de un registro para la etapa de *commit* de su instrucción LU). Puede ocurrir que la instrucción LU ya haya alcanzado su etapa de *commit* cuando se está decodificando su instrucción par NV, en este caso no necesita planificarse ninguna liberación ya que dicha liberación puede hacerse de forma inmediata.

Bajo especulación frente a instrucciones de salto, es necesario un *hardware* específico para deshacer los efectos causados por la ejecución de las instrucciones del camino mal predicho. De la misma forma, si una instrucción especulada NV debe neutralizarse, es necesario deshacer cualquier planificación de liberación anticipada que haya generado esta instrucción. La implementación del *hardware* de recuperación de caminos mal predichos en vp-LAER se hace distinguiendo los siguientes dos casos:

- *La instrucción NV que se está decodificando* ***no es especulada***. NV puede pertenecer o no al mismo bloque básico en el que se encuentra su correspondiente instrucción LU. En cualquier caso, todos los saltos intermedios, si existen, entre la última instrucción que ha hecho el *commit* en el procesador (la instrucción más vieja) y la instrucción NV han verificado tanto sus condiciones como sus direcciones de destino. Cualquier *planificación de liberación anticipada* que haga NV se dirá que es una ***liberación segura***.

- *La instrucción NV que se está decodificando* ***es especulada***. En este caso, todavía existen saltos previos a NV que se encuentran pendientes de ser verificados. Además, cualquier planificación de liberación anticipada que haga la instrucción NV mientras es especulada, debe considerarse como condicional y por lo tanto sujeta a ser eliminada. Cualquier *planificación de liberación anticipada* que haga NV se dirá que es una ***liberación especulada***.

Todos estos conceptos sobre la liberación se pueden aplicar tanto a los registros virtuales-físicos como a los registros físicos. En realidad, lo que se va a hacer en vp-LAER es liberar siempre un par de registros: un registro virtual-físico y el correspondiente registro físico al cual se encuentra asignado.

## 5.2 Recursos *Hardware*

La Figura 5.1 presenta el esquema global de todo el proceso de renombre y liberación considerado en vp-LAER. La asignación de registros de vp-LAER se basa en la utilización de la Tabla de Mapeo General (GMT) además de una pequeña modificación en la ventana de lanzamiento (IW). La liberación de los registros de vp-LAER se basa en la utilización de la Tabla de Ultimos Usos (LUsT), una pequeña modificación de la estructura de reordenación (ROS) y una Cola de Liberación (*Release Queue*). Todo lo anterior se completa con dos listas de registros libres (una *Free List* para registros virtuales-físicos y otra *Free List* para registros físicos) y con una Tabla de Mapeo Físico (PMT).

A continuación se van a describir de forma separada todos los componentes. Las dos subsecciones siguientes describirán la interacción entre estos componentes y mediante un ejemplo se verá cuál es el procedimiento que se sigue tanto para asignar los registros como para liberarlos.

### TABLA DE MAPEO GENERAL (GMT)

La **GMT** almacena los registros virtuales-físicos y físicos asignados a los registros lógicos. Se indexa con el identificador de registro lógico y posee los siguientes tres campos:

- vp: registro virtual-físico asignado actualmente al registro lógico.
- p: registro físico al que los registros lógicos y virtual-físico están asignados, si los hay.
- v: bit que indica si el campo p contiene un valor válido, es decir si el registro lógico ya tiene asignado un registro físico.

Cada instrucción lee los campos vp de todos sus registros fuente y escribe los campos vp y p de su registro destino. La escritura en el campo vp se realiza en la etapa de decodificación y la escritura en el campo p se realiza al final de la etapa de ejecución.

### TABLA DE ULTIMOS USOS (LUST)

La **LUsT** identifica a la instrucción que utiliza por última vez un determinado registro lógico. Se indexa con el identificador de registro lógico y posee los siguientes tres campos:

- ROSid: guarda el identificador de la instrucción LU (aquella que utiliza el registro lógico por última vez).
- k: indica el tipo de uso del registro por parte de la instrucción LU (1 = fuente 1, 2 = fuente 2, 0 = destino).

- c: indica si la instrucción LU está todavía en el ROS (c es igual a 0) o por el contrario ya ha alcanzado su etapa de *commit* (c es igual a 1).

En la etapa de decodificación, se registran los usos de todos los registros lógicos de cada instrucción, escribiendo el ROSid de la instrucción que se está decodificando y activando también k con su valor correcto (deben realizarse hasta tres escrituras para cada instrucción de tres registros). Con esto, cada instrucción que ha de escribir en un registro (NV) encuentra el ROSid de su correspondiente instrucción par LU leyendo la entrada de LUsT correspondiente a su registro lógico destino.

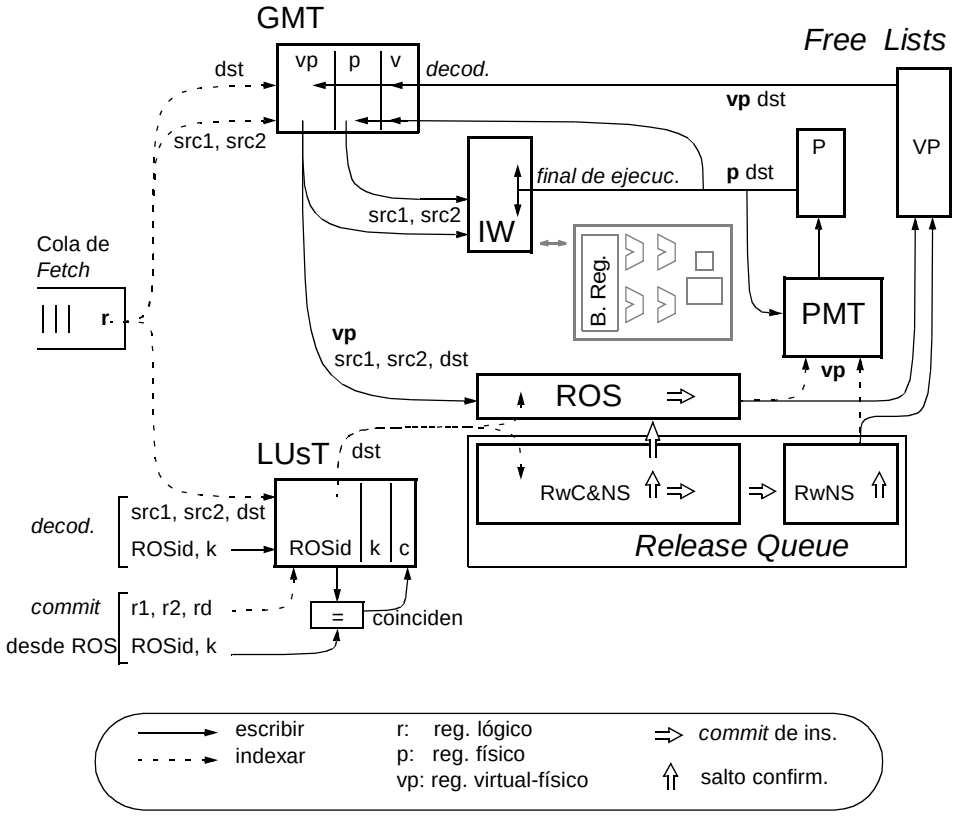


**Figura 5.1** Proceso de renombre en vp-LAER.

Para poder soportar todos los accesos necesarios a ella, la tabla LUsT precisa de un *ancho de banda* considerable, convirtiéndose en una estructura con un número de puertos elevado. Sin embargo y debido a su pequeño tamaño, su tiempo de acceso se encuentra muy por debajo del más pequeño de un banco de registros físicos razonable. Por tanto, la tabla LUsT queda fuera de cualquier camino crítico del procesador [MVG+02].

De la misma forma que ocurre con la tabla GMT, se supone que tras cada predicción de salto se realiza una copia de la tabla LUsT. De esta forma, el mecanismo de recuperación puede recuperar una copia correcta de la tabla tras una mala predicción de un salto [HP87].

### FREE LISTS Y TABLA DE MAPEO FÍSICO (PMT)

Existen dos conjuntos de registros que se encuentran libres, uno de registros virtuales-físicos y otro de registros físicos. Se van a llamar respectivamente las listas *Free VP* y *Free P*. Estas listas se leen para obtener nuevos identificadores para un registro destino y se escriben en el momento de la liberación del registro.

La tabla PMT refleja la asignación de registro físico que posee cada registro virtual-físico. Sus entradas se actualizan al final de la etapa de ejecución y se indexa con el número del registro virtual-físico. En el proceso de liberación, la tabla PMT se utilizará para alimentar a la lista *Free P* con el registro físico que está asignado a un determinado registro virtual-físico.

### VENTANA DE LANZAMIENTO DE INSTRUCCIONES (IW)

Para poder soportar el mecanismo vp-LAER, cada entrada en la **IW** posee los siguientes campos:

- vp_fuente 1, 2: identificadores de los registros virtuales-físicos asignados a los operandos fuente.
- p_fuente 1, 2: identificadores de los registros físicos asignados a los operandos fuente.
- *ready* 1, 2: son los bits de preparado de los operandos fuente. Cuando el operando fuente uno o fuente dos se encuentra preparado, los campos p_fuente 1 ó p_fuente 2 contienen los identificadores de los registros físicos que actualmente están asignados a los registros virtuales-físicos que hay en vp_fuente 1 ó en vp_fuente 2.
- vpd: registro virtual-físico destino. Se utiliza para la asignación de registro físico al final de la etapa de ejecución de la instrucción.
- pd: identificador del registro físico destino. Si la instrucción se selecciona como víctima para ser robada, este campo proporciona el correspondiente identificador físico sin necesidad de acceder a la tabla PMT.
- exec: *flag* de ejecución. Se pone a uno cuando la instrucción se ha ejecutado y su resultado se ha escrito en el registro físico.

Una instrucción puede lanzarse a ejecutar cuando sus dos operandos fuente se encuentran preparados. Cuando una instrucción se lanza a ejecutar, lee sus operandos fuente del banco de registros físicos (o desde la *red de bypass*) utilizando los identificadores p_fuente 1, 2.

### ESTRUCTURA DE REORDENACIÓN (ROS) Y COLA DE LIBERACIÓN (RELQUE)

Como es habitual, la estructura de reordenación ROS permite que las instrucciones alcancen en orden su etapa de *commit*. Además, también permite activar la liberación de los registros virtuales-físicos. Más especificamente, la planificación de liberaciones que son *seguras* se almacenan directamente en el ROS, mientras que aquellas planificaciones de liberaciones que son *especuladas* se almacenan de forma temporal en la RelQue y evolucionan dentro de ella.

Una entrada en el **ROS** posee los siguientes campos (consultar la Figura 5.2):

- r_1, r_2, r_d: identificadores de los registros lógicos (fuente 1, fuente 2 y destino).

- vp_1, vp_2, vp_d: identificadores de los registros virtuales-físicos (fuente 1, fuente 2 y destino). Reflejan el *renombre virtual* de toda instrucción tras su decodificación.

- rel1, rel2, reld: bits de liberación en la etapa de *commit*. Estos bits representan las planificaciones de liberaciones que son *seguras*. Cuando se encuentran con el valor de uno, la etapa de *commit* de la instrucción fuerza por un lado la liberación de los registros virtuales-físicos identificados en los campos vp_1, vp_2 ó vp_d respectivamente y por otro lado la liberación de sus correspondientes pares de registros físicos. Esos registros físicos se obtienen de la PMT indexándola con los identificadores de los registros virtuales-físicos. Se ha denominado **RwC** (*Release-when-Commit*) al conjunto de todos estos bits, los cuales se añaden a todas las entradas de instrucciones del ROS (consultar la Figura 5.2).

La estructura **RelQue** guarda todas las planificaciones de liberaciones que son *especuladas*. Esta estructura posee tantos niveles horizontales como número de saltos pendientes de ser confirmados soporta el procesador (desde 1 hasta *max* en la Figura 5.2). En cuanto a los saltos se refiere, los distintos niveles de la RelQue se gestionan de forma FIFO. Cada vez que se decodifica un salto, se añade un nuevo nivel al final de la RelQue. Los niveles más cercanos a la cabeza de la RelQue son los asociados a los saltos más viejos. Por sencillez, todos los dibujos suponen que la cabeza de la FIFO se encuentra en una posición fija y que el último nivel que está ocupado es aquel identificado con el puntero COLA (se corresponde con el último salto decodificado). Cada vez que se confirma un salto, las planificaciones de liberaciones *especuladas* evolucionan hacia la cabeza de la RelQue (es la flecha *salto conf.* de la Figura 5.2).

Cada nivel *n* está formado a su vez por dos estructuras que poseen un número distinto de entradas, **RwC&NS*n*** y **RwNS*n***:

- RwC&NS*n* (*Release when Commit and Non Speculative*): esta estructura guarda las planificaciones de liberación *especuladas* que provienen de instrucciones NV que son especuladas y que se decodifican teniendo por encima de ellas *n* saltos pendientes de ser verificados. Cualquier instrucción que tenga alguno de estos bits a uno se trata de una instrucción LU que se encuentra activa en el procesador. Posee tantas entradas como tiene el ROS y cada entrada posee tres bits. Cada RwC&NS*n* soporta desplazamientos de izquierda a derecha para sincronizarse con la función de *commit* del ROS (es la flecha *commit* de la Figura 5.2).

- RwNS*n* (*Release when Non Speculative*): es una matriz de bits que guarda las liberaciones *especuladas* planificadas desde una instrucción NV que posee frente a ella *n* saltos que todavía están pendientes de ser verificados pero que cuya instrucción par LU ya ha alcanzado la etapa de *commit*. Posee un bit por cada registro virtual-físico.

En la siguiente subsección 5.2.2 se detalla a través de un ejemplo cómo funcionan las distintas operaciones que se realizan en la RelQue.

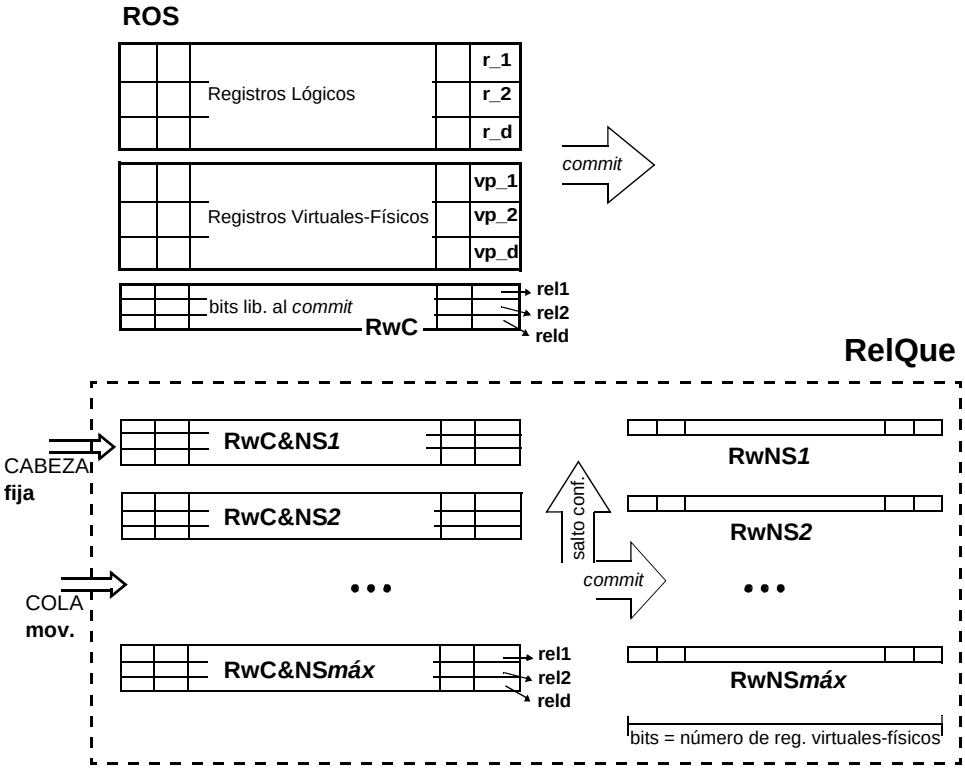


**Figura 5.2** Estructuras de reordenación (ROS) y Cola de Liberación (RelQue). Las instrucciones entran por la izquierda, se colocan en el ROS y salen por la derecha. La validación de un salto desplaza hacia arriba las entradas de la RelQue.

### 5.2.1 Ejemplo de Asignación de los registros

A continuación se va a ver cómo funciona la asignación de los registros en el mecanismo vp-LAER utilizando el código simple que aparece en la Figura 5.3. Esta figura muestra tres momentos en la ejecución de tres instrucciones (i1, i2 e i3) a través del contenido de GMT, IW y PMT. La Figura 5.3 muestra primero la decodificación de las tres instrucciones y el final de la etapa de ejecución de la instrucción i2 (Figura 5.3.a) y después el robo de registro a la instrucción i2 por parte de la instrucción i1 (Figura 5.3.b). La Figura 5.3.a muestra así dos movimientos, el primero de ellos indica la situación después de decodificar las instrucciones i1, i2 e i3 y el segundo (marcado con flechas) indica la situación después de que la instrucción i2 finalice su ejecución y se le asigne el registro físico p5. La Figura 5.3.b muestra la situación después de que la instrucción i1 se ejecute y robe el registro p5, que estaba asignado a la instrucción i2. En todo momento se está suponiendo que la instrucción i3 no ha podido lanzarse a ejecutar.

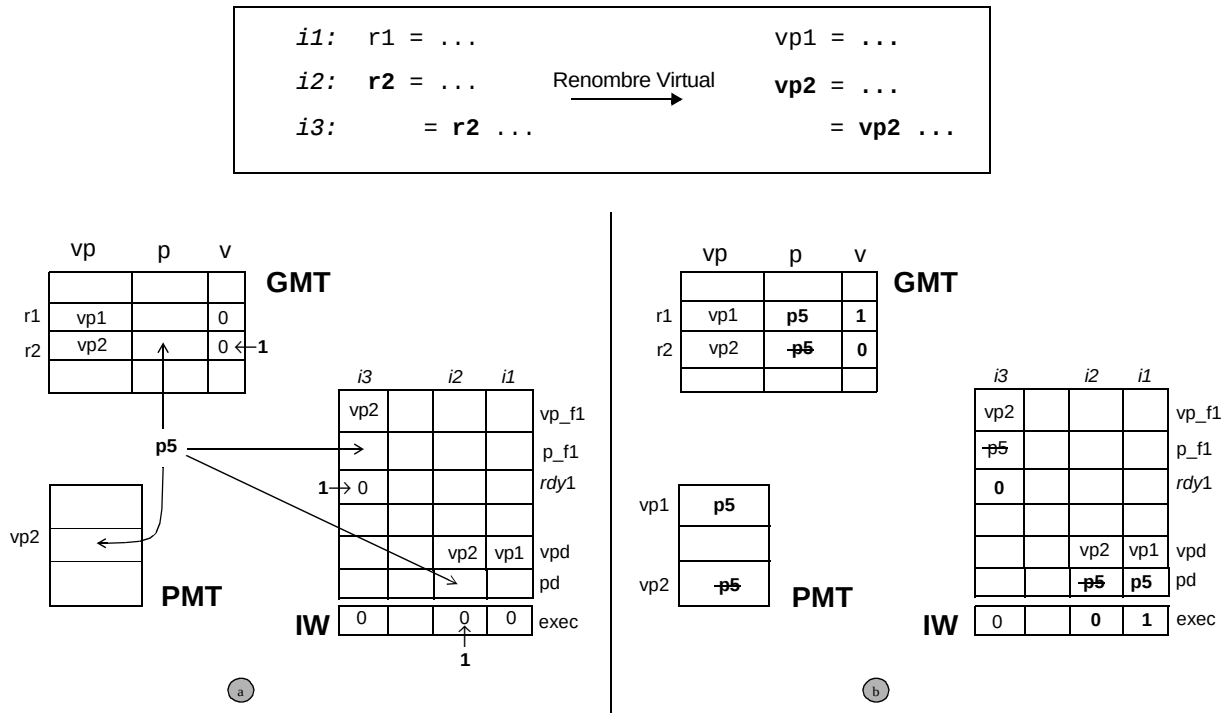


**Figura 5.3** Ejemplo de la asignación vp-DSY en el renombre de registros vp-LAER. Primero se decodifican las instrucciones i1, i2 e i3; segundo, las flechas muestran los cambios que ocurren después que finaliza la instrucción i2 y se le asigne p5 **(a)**. Antes de que la instrucción i3 se lance a ejecutar, la instrucción i1 roba el registro p5 a la instrucción i2 **(b)**.

Se describe ahora paso a paso las actividades que se realizan en cada momento en las tablas GMT, PMT y el contenido que va teniendo la IW para cada una de las instrucciones del ejemplo.

- **Las instrucciones i1, i2 e i3 se decodifican,** Figura 5.3.a. Para las instrucciones i1 e i2 los identificadores de los registros virtuales-físicos vp1 y vp2 se obtienen de la lista Free VP, y a continuación se escriben en la tabla GMT poniendo a cero el bit v. Además, las dos instrucciones se instalan en la IW rellenando sus correspondientes campos del registro destino vpd con vp1 y vp2. Para la instrucción i3, su registro lógico fuente 1 -r2- se renombra a través de la GMT. El registro virtual-físico obtenido -vp2- se escribe en la entrada de i3 en la IW y se marca como no preparado (bit *ready* 1 = cero). Todas las instrucciones se marcan como no ejecutadas (bit exec = cero en la IW).

- **La instrucción i2 finaliza su ejecución y se le asigna el registro p5,** flechas en la Figura 5.3.a. Cuando la instrucción i2 finaliza su ejecución, o algunos ciclos antes, se pide a la lista Free P el identificador de un registro físico, que en el ejemplo resulta ser p5. La entrada de i2 en la IW se actualiza de la siguiente forma: p5 se escribe en el campo pd y la instrucción se marca como ejecutada (bit exec = uno).

Después y para reflejar la nueva asignación en la PMT, con el registro p5 se actualizan tanto la GMT como la IW. Para ello, se indexa PMT con vp2 y esa entrada se escribe con p5. Para actualizar GMT, se busca de forma asociativa vp2 en todos los campos vp y si existe alguna coincidencia, se reemplaza esa entrada con el registro p5 poniendo a uno el bit v. Al mismo tiempo los campos de los operandos fuentes virtuales-físicos de toda instrucción en la IW (vp_fuente1,2) se acceden de forma asociativa y siempre que aparezca vp2, se escribe p5 en el campo del operando fuente físico correspondiente (en el ejemplo, en p_fuente1 de la instrucción i3) y se marca como preparado (bit *ready* 1 = uno).

- **La instrucción i1 se ejecuta y roba p5 a i2,** Figura 5.3.b. Supongamos que la instrucción i1 finaliza ahora su ejecución y no queda ningún registro físico disponible para asignar a su registro virtual-físico destino -en el ejemplo vp1-. Entonces, se identifica a una instrucción más joven que ya tenga registro físico asignado. Esto puede realizarse inspeccionando la ventana de lanzamiento, buscando una entrada más joven que tenga el bit de ejecución activo. Si se encuentran varias, se elige a la más joven de todas ellas y se leen sus campos vpd y pd -en el ejemplo, es la instrucción i2 y el par <vp2, p5>-. Todas las estructuras deben reflejar que ahora vp1 está asignado a p5 y que vp2 ya no está asignado a p5. Para hacer esto y actuando de forma parecida a como se ha descrito en el paso anterior, se invierten las asignaciones de las tablas GMT y PMT, p5 se escribe ahora en el campo pd de la instrucción i1 marcándola como ejecutada (consultar la Figura 5.3.b).

  Como la instrucción robada i2 debe volver a ejecutarse, su bit de ejecución se pone de nuevo a cero en la IW, permitiendo de esta forma que la lógica de lanzamiento pueda elegir de nuevo a i2. Además, desde que i2 fue ejecutada, los consumidores de vp2 (la instrucción i3 en el ejemplo) tienen marcado este operando como preparado. Sin embargo, dejan de estar preparados desde el momento en que el registro físico ha sido robado. En ese momento, hay que poner a cero los bits de preparado y esto puede hacerse utilizando los buses que sirven para despertar las instrucciones en la IW (en el ejemplo, bit *ready* 1 = cero).

### 5.2.2 Descripción de Liberación de los registros

Si no existen saltos que estén pendientes de verificarse, ninguna instrucción que se decodifique será especulada y las acciones de planificación de liberación que han de realizarse en este caso son simples y no es necesario utilizar la estructura RelQue. Se va a explicar primero cómo funciona este caso simple. A continuación, se explicará como funciona el caso más complejo donde al menos la instrucción siguiente versión (NV) es especulada.

#### LAS INSTRUCCIONES QUE SE DECODIFICAN NO SON ESPECULADAS

El soporte básico para llevar a cabo la liberación comienza en la etapa de renombre, donde todos los registros virtuales-físicos se escriben en la entrada que toda instrucción tiene asignada en el ROS. Esta es la actividad denominada como *renombre virtual*. Se va a considerar como ejemplo de soporte el código simple de la Figura 5.4.a donde sólo se muestran dos instrucciones: la que lee por última vez el registro r2 (LU) y la que produce su siguiente versión (NV). En el ejemplo, antes de decodificar la instrucción LU, suponemos que vp2 y p2 son los registros virtual-físico y

físico que están asignados a r2 (consultar los contenidos de GMT en la Figura 5.4.a). Se va a ver cuáles son las acciones a realizar para liberar ese par de registros.

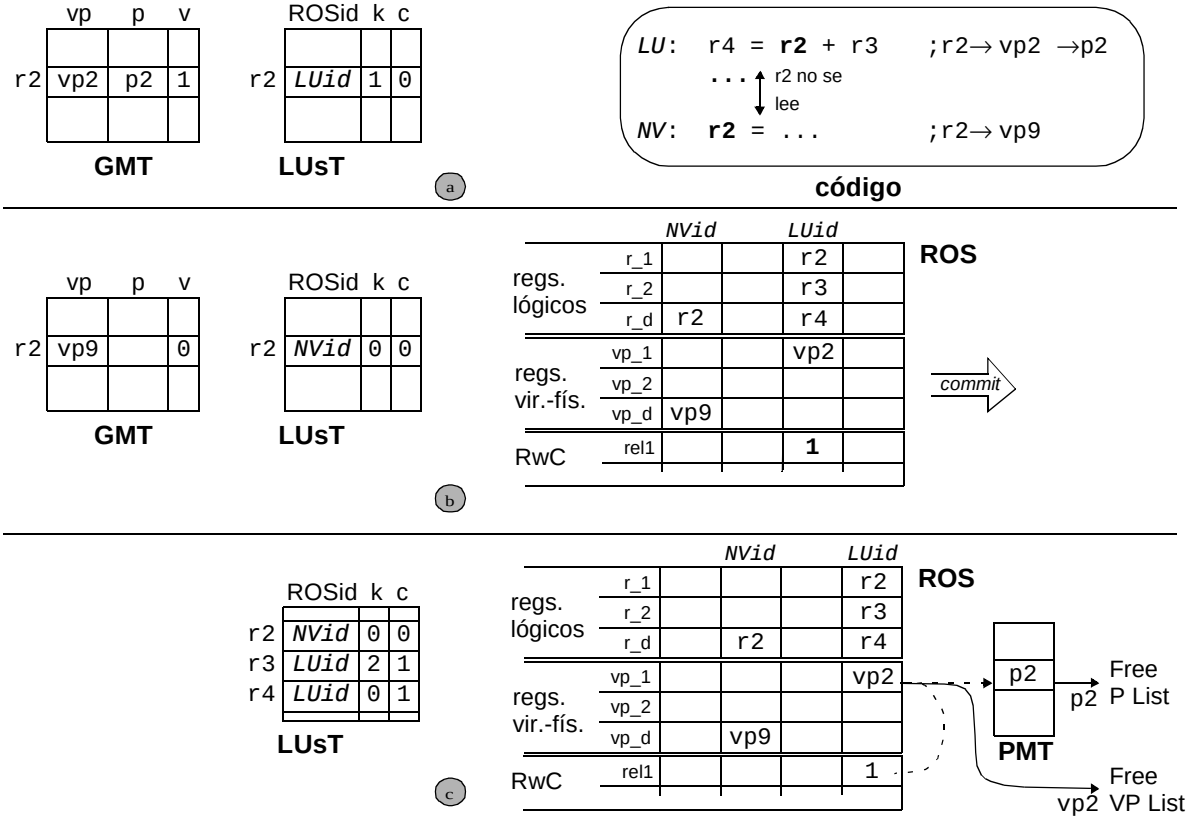


**Figura 5.4** Ejemplo de liberación de registros en vp-LAER. Código ejemplo con NV no especulada y estado en que quedan GMT/LUsT tras decodificar LU **(a)**. Decodificación de NV y planificación de liberación anticipada para vp2 en la entrada LUid del ROS **(b)**. LUsT tras el *commit* de LU y liberación del par <vp2, p2> en la etapa de *commit* de LU **(c)**.

Cada instrucción debe consultar y/o actualizar las tablas LUsT y RwC. Para hacer más sencilla la explicación, esas acciones se van a explicar en tres pasos: A, B que se llevan a cabo en la decodificación y el paso C en la etapa de *commit*. En el primero de esos pasos (*A*) es donde se registran las instrucciones que hacen el último uso de los registros. En este paso se actualiza la LUsT. En el segundo paso (*B*), una instrucción NV planifica la liberación de la versión anterior de su registro destino para la instrucción que hizo su último uso (LU que se obtiene consultando la LUsT): bien a través de una *planificación de liberación anticipada* (*B1*) o bien a través de una *liberación inmediata* (*B2*). En el primer caso aún no es seguro liberar la versión anterior porque la instrucción LU todavía se encuentra en el ROS -aquí se actualiza RwC-. En el segundo caso ya es seguro liberar la versión anterior porque ya no es necesaria al haber desaparecido del procesador su instrucción LU (LU ya ha alcanzado la etapa de *commit*). Para finalizar, el tercer paso (*C*) se lleva a cabo cuando las instrucciones alcanzan el *commit*. Se actualizan los bits de *commit* en la LUsT (*C1*) y las instrucciones LU liberan registros cuando tienen planificaciones de liberación en RwC (*C2*).

- *(A) Decodificación de los operandos fuente*

  Este paso, para el código de la Figura 5.4.a, actúa escribiendo en una de las entradas de la LUsT para registrar la lectura que en orden de programa hace la instrucción LU del registro fuente 1 -r2-. En la Figura 5.4.a se muestra el estado en que queda la entrada del registro r2 en la LUsT después de decodificar la instrucción LU (el campo ROSid se rellena con la dirección que ocupa la instrucción LU en el ROS, el campo k muestra el número del operando fuente de que se trata y el bit de *commit* c se pone a cero).

- *(B) Decodificación del operando destino*

  Para toda instrucción NV, antes de realizar el renombre se utiliza su registro lógico destino para indexar las tablas GMT y LUsT y obtener cuatro campos: <p, ROSid, k, c>, donde c posee el valor de uno o de cero según si la correspondiente instrucción LU ha alcanzado la etapa de *commit* o no. En el ejemplo de la Figura 5.4.a NV leería <p2, LUid, 1, 0>. Ahora y dependiendo del valor de c, pueden tener lugar dos alternativas tanto de renombre como de acciones de liberación:

  *(B1) Planificación de una liberación anticipada.* La instrucción LU identificada todavía se encuentra activa en el ROS (LUid con bit c = cero) y por lo tanto la versión del registro que utiliza no puede perderse hasta que alcance su etapa de *commit*. En este caso, la acción a realizar consiste en planificar una liberación *segura* para liberar un registro virtual-físico cuando la instrucción LU alcance su etapa de *commit.* Esto se hace poniendo a uno en la matriz RwC el bit rel*k* para aquella entrada ROSid donde se encuentra la instrucción LU. La Figura 5.4.b muestra el estado en el que queda el ROS después de llevar a cabo esta acción. Se ha puesto a uno el bit rel*1* en la entrada LUid, lo que significa que en la etapa de *commit* de LU se liberará vp2 (su campo vp_*1*). En paralelo, se renombra el registro destino de NV con vp9 y con él se actúa en la LUsT de forma similar a como se ha hecho para los registros fuentes en el paso anterior (*A*). Las tablas GMT y LUsT quedan ahora para r2 como se muestra en la Figura 5.4.b.

  *(B2) Liberación del registro físico y reutilización del registro virtual-físico.* La instrucción LU identificada (ROSid) ya ha abandonado el ROS (bit c = uno) y por lo tanto la versión del registro que utilizaba por última vez ya no es necesaria. En este caso, la acción a realizar se desdobla en dos acciones: por un lado, se libera de forma inmediata el registro físico indicado en p, devolviéndolo a la lista Free P y por otro lado, se reutiliza el registro virtual-físico de la versión anterior para la instrucción NV (en el ejemplo, se mantendría vp2 en la GMT en lugar de recoger vp9 de la lista Free VP). Después la única acción que se debe realizar en la tabla GMT consiste en poner a cero el bit v. Si este fuera el caso en el ejemplo, la entrada r2 de GMT quedaría con <vp2, -, 0> y p2 debería añadirse a la lista Free P.

- *(C) Etapa de* commit *de la instrucción*

  El último de los pasos se realiza cuando las instrucciones alcanzan la etapa de *commit*, actualizando, si es necesario, los bits de *commit* en la LUsT (*C1*) y liberando los registros planificados en RwC (*C2*):

  *(C1) Actualización de los bits de* commit *en la LUsT.* Para cada registro lógico referenciado en cada instrucción en el ROS, se indexa la LUsT para leer los campos ROSid y comparar-

los con el identificador de la instrucción que está haciendo el *commit*. En toda coincidencia que se encuentre, el bit c se pone a uno. Notar que las acciones realizadas en los bits c deben extenderse a todas las copias de la tabla LUsT para poder llevar a cabo una recuperación correcta tras un salto mal predicho. La Figura 5.4.c muestra la situación en que queda la LUsT después de que la instrucción LU haga el *commit* y suponiendo que entre las instrucciones LU y NV no se ha hecho ningún uso ni de r3 ni de r4.

*(C2) Liberación de los registros.* Esta acción tiene lugar cuando se hacen visibles al *commit* los bits que tienen el valor de uno en RwC. En el ejemplo, cuando la instrucción LU alcanza su etapa de *commit*, el bit rel1 puesto a uno fuerza la liberación tanto del registro vp2 como del registro físico p2 que tiene asociado, el cual se obtiene a través de la PMT como muestra la parte derecha de la Figura 5.4.c.

### LAS INSTRUCCIONES QUE SE DECODIFICAN SON ESPECULADAS

Se va a explicar ahora el caso en el que las instrucciones NV son especuladas. Para cada salto predicho, se coloca en una pila de recuperación una copia de las tablas GMT y LUsT para poder restaurar el estado correcto tras una mala predicción. Ahora, las planificaciones de liberación que se deban hacer son *especuladas* y por lo tanto deben colocarse en la RelQue, donde irán evolucionando a la vez que las predicciones de los saltos se vayan confirmando o descartando. Por ejemplo, una instrucción NV especulada que tenga *n* saltos previos a ella pendientes de ser verificados colocará sus planificaciones de liberación (*especuladas*) en el nivel *n* de la RelQue (aquel que está marcado con el puntero COLA). El trabajo que hay que hacer para consultar y actualizar la LUsT es el mismo que se ha explicado antes para instrucciones no especuladas, pero ahora y en relación al estado de *commit* de la instrucción LU identificada (bit c), son posibles las siguientes dos acciones de planificación:

- *La instrucción LU todavía se encuentra activa en el ROS* (c = cero). La planificación de liberación *especulada* se realiza en la matriz de bits RwC&NS*n* que posee la entrada LUid del ROS. Para convertir esta planificación de liberación en una liberación efectiva, se necesitan cumplir dos condiciones: que se confirmen las predicciones que se han hecho para los *n* saltos pendientes y que la instrucción LU alcance la etapa de *commit*.

- *La instrucción LU ya ha alcanzado la etapa de* commit (c = uno). La planificación de liberación *especulada* se realiza en la matriz de bits RwNS*n* en la entrada del registro virtual-físico que se ha de liberar. Para convertir esta planificación de liberación en una liberación efectiva, sólo se necesita cumplir una condición: que se confirmen las predicciones que se han hecho para los *n* saltos pendientes.

Una instrucción LU puede alcanzar su etapa de *commit* antes de que su instrucción par NV deje de ser especulada. En este caso, no pueden perderse las planificaciones *especuladas* asociadas a las instrucciones que están haciendo el *commit* y por lo tanto deben moverse desde RwC&NS hacia RwNS para quedar allí esperando a que se verifiquen los saltos. El anterior movimiento

requiere decodificar los identificadores de los registros virtuales-físicos para activar los bits correctos en RwNS (consultar la operación *Marcado* en la Figura 5.5.a).

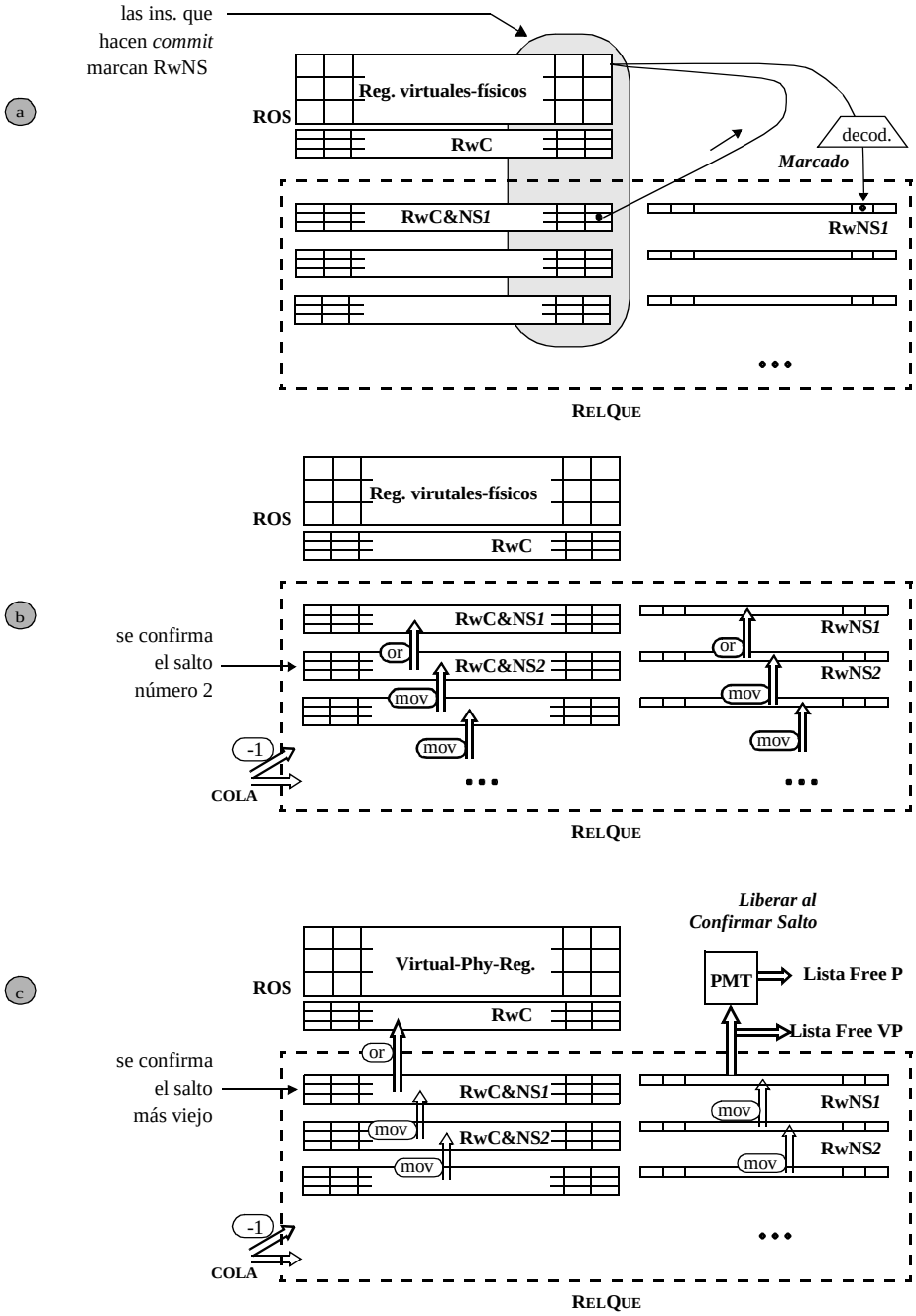


**Figura 5.5** Ejemplos de varias operaciones soportadas por la RelQue. Marca de un bit en RwNS1 para registrar una liberación especulada que proviene de una instrucción LU que está haciendo el *commit* **(a)**; se confirma el segundo salto más viejo **(b)**; la confirmación del salto más viejo genera la liberación al confirmar el salto **(c)**.

Cuando el salto pendiente número *n* se confirma, se hace la operación ***or*** de todo el nivel *n* de la RelQue con el nivel *n-1*, colocando el resultado en el nivel *n-1*. Al mismo tiempo, todas las planificaciones *especuladas* más jóvenes (aquellas que se encuentran desde el nivel COLA hasta el nivel *n+1*) se mueven (operación ***mov***) un nivel hacia la cabeza del ROS. Consultar la Figura 5.5.b para ver un ejemplo de esta operación en el caso de *n* = dos.

La confirmación del salto más viejo (caso *n* = uno) posee algo de trabajo añadido. Este es el caso que representa la operación *Liberar al Confirmar Salto* de la Figura 5.5.c. En este momento se hacen efectivas todas las planificaciones de liberación de registros que se encuentran en RwNS*1*. Los identificadores de los registros virtuales-físicos se envían a la lista Free VP y la PMT se indexa para obtener los identificadores de los registros físicos que se envían a la lista Free P. Se puede suponer que la comunicación entre RwNS*1* y PMT está correctamente diseñada para poder evitar el decodificador de direcciones interno a la PMT.

Por otro lado, si la predicción del salto número *n* es incorrecta, se eliminan de la RelQue todos sus niveles desde *n* hasta COLA, haciendo así desaparecer todas las planificaciones de liberación *especuladas* que procedían del camino mal predicho. Además, COLA se desplaza para apuntar ahora al nivel *n-1*. Notar que vp-LAER permite que los saltos se verifiquen fuera de orden.

Resumiendo, los registros se liberan bajo tres posibles situaciones: las dos que se han descrito en los pasos *B2* (liberación inmediata) y *C2* (liberación al *commit* de LU -desde RwC-) y cuando se confirma el salto más viejo (*Liberar al Confirmar Salto* -desde RwNS1-). En todos los casos, la liberación de un registro se realiza antes que bajo el esquema de renombre convencional. Para evitar que el acceso a la PMT retrase la liberación de los registros, se va a suponer un diseño de esta tabla donde toda liberación se traduzca en la generación de un vector de bits de registros físicos a liberar. De esta forma las liberaciones de registros físicos se pueden convertir en una operación OR de ese vector de bits y la Free List P.

A continuación se van a ver algunos de los resultados más representativos obtenidos para el mecanismo vp-LAER. En el siguiente apartado se hará un análisis del coste de implementación de las estructuras que son clave para soportar todo el mecanismo.

## 5.3 Resultados

En este apartado se van a analizar aquí los beneficios que aportaría al rendimiento de un procesador la sustitución del mecanismo de renombre convencional por el mecanismo vp-LAER. Dichos beneficios se van a comparar también con las dos componentes de vp-LAER pero actuando por separado (vp-DSY -en el Capítulo 3- y la liberación anticipada -en el Capítulo 4-).

La Figura 5.6 muestra el número promedio de instrucciones que alcanzan la etapa de *commit* por ciclo (IPC) para los cuatro esquemas de renombre: convencional (conv), liberación anticipada, vp-DSY y vp-LAER. Todos ellos con una configuración de $64_{ent}+64_{fp}$ registros. El último grupo de columnas muestra el valor de la media armónica (Mh) tanto para enteros como para fp. A primera vista se observa como los programas de tipo fp se benefician más de cualquier forma de

mejora en el renombre de registros. Lo anterior se debe a la alta presión que ejercen sobre el banco de registros fp que ha sido elegido para este experimento.

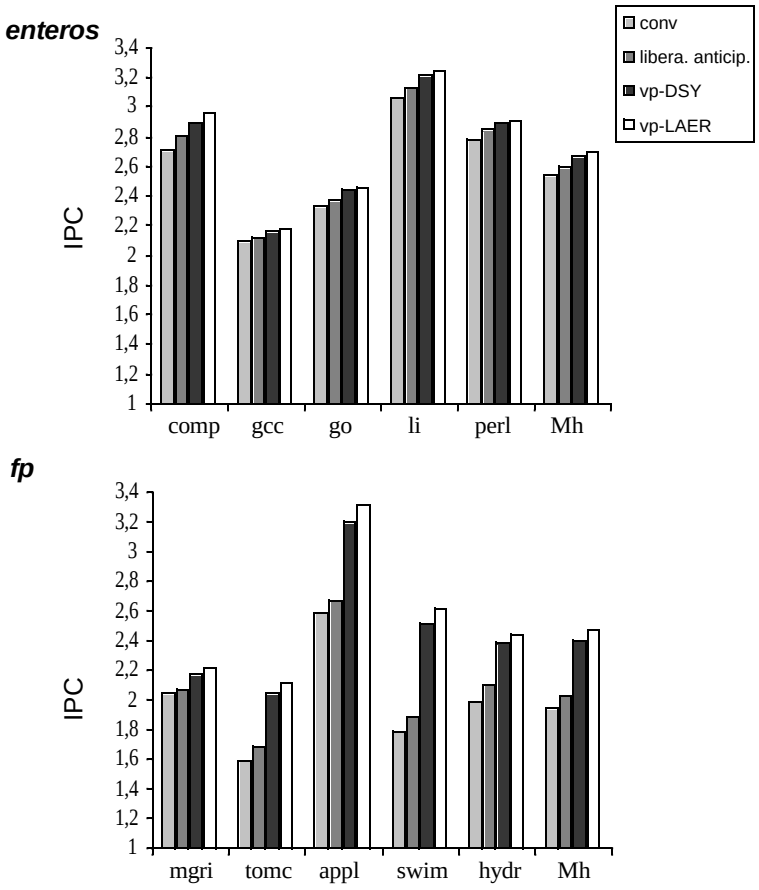


**Figura 5.6** IPC obtenido con los esquemas alternativos de renombre para un banco de registros de 64ent+64fp. El grupo de barras puesto a la derecha de cada gráfico muestra la media armónica del IPC.

Haciendo un análisis separado de las dos componentes de vp-LAER, se ve que para esta configuración, sólo con la liberación anticipada se obtiene un *speed-up* promedio del 2% y del 5% para programas enteros y fp respectivamente. Vp-DSY actuando en solitario consigue hasta un 5% y un 24% respectivamente. Para acabar, vp-LAER consigue un 6% y un 28%, mejorando a ambos componentes actuando en solitario y presentando un comportamiento casi aditivo. Las dos componentes en vp-LAER aunque actúan de forma ortogonal, presentan momentos de cooperación que resultan beneficiosos. El mayor número de registros libres que proporciona la liberación anticipada permite aumentar el rendimiento que se obtendría sólo con vp-DSY.

Para ver cómo se comporta vp-LAER bajo diferentes condiciones de presión sobre el banco de registros, se ha considerado todo el rango de tamaños del banco de registros (desde $40_{ent}$+$40_{fp}$ registros hasta $160_{ent}$+$160_{fp}$) y se ha calculado el IPC para todas las configuraciones. La Figura 5.7 muestra éste IPC para las todas las políticas. En las Figuras B.7 a B.9 del Apéndice B se

puede encontrar esta misma medida por separado para cada uno de los programas de prueba. El intervalo entre 128 y 160 registros no se representa por la saturación que existe en IPC.

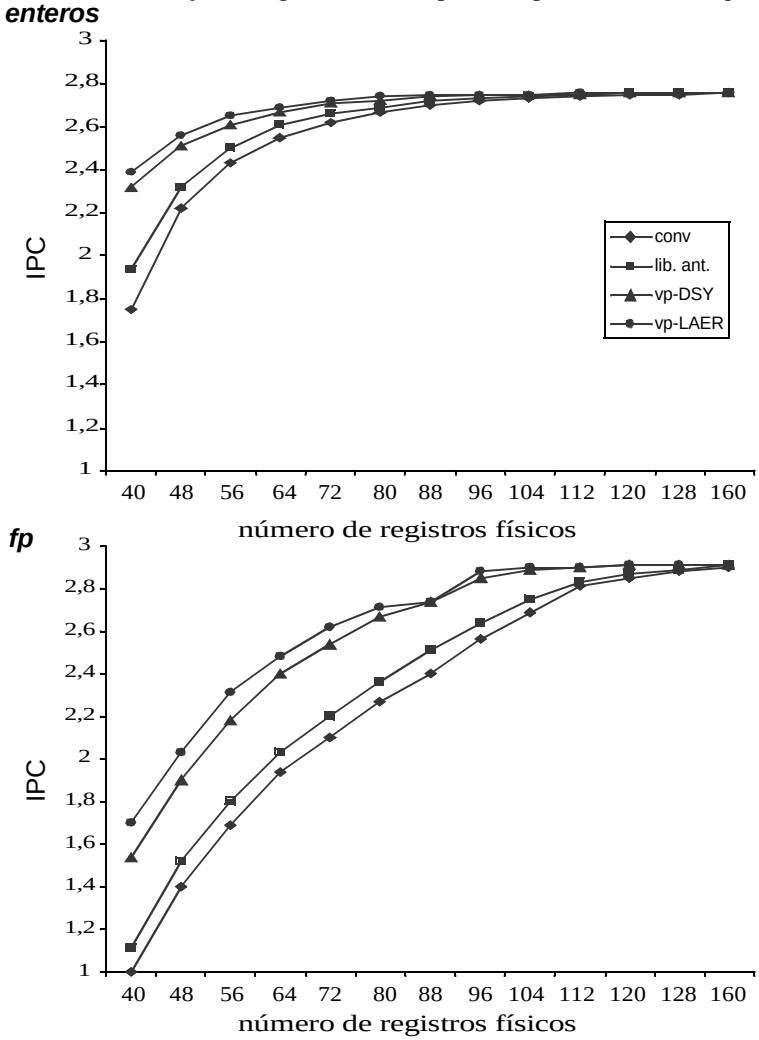


**Figura 5.7** Media armónica en IPC *versus* el número de registros físicos. Mecanismos convencional, liberación anticipada, vp-DSY y vp-LAER.

Se puede observar cómo en el IPC obtenido con vp-LAER domina sobre todo el peso de vp-DSY, mientras que es más modesta la contribución de la liberación anticipada. Se acaba de decir que la mejora del rendimiento que se obtiene con vp-LAER es muy cercana a la suma de la mejora de rendimientos que se obtienen con sus dos componentes actuando por separado. Sin embargo, para bancos de registros muy ajustados (entre 40 y 56 registros) y para códigos fp, la Figura 5.7 muestra como con vp-LAER puede llegarse a estar incluso por encima de dicha suma.

La liberación anticipada, consigue reducir de forma significativa la sobrecarga que representaban las reejecuciones en vp-DSY. Con vp-DSY se retrasa la asignación de los registros físicos hasta la finalización de la etapa de ejecución y si en este momento no existe ningún registro físi-

co libre, se roba el que tiene asignado aquella instrucción más joven. Más adelante, la instrucción a la que se le roba el registro debe reejecutarse (Capítulo 3). Sin embargo, el añadido de la liberación anticipada ayuda a reducir ese número de reejecuciones, lo cual está directamente relacionado con el consumo de energía. La Figura 5.8 muestra el porcentaje de reducción en el número total de reejecuciones obtenido al utilizar vp-LAER. Por ejemplo, con un banco de $64_{ent}+64_{fp}$ registros, el número de reejecuciones disminuye un 20% y un 14% para programas de tipo entero y programas fp respectivamente. Esta reducción aumenta a la vez que el banco de registros se hace más sobrado, donde un número pequeño de registros adicionales es suficiente para hacer desaparecer la reejecución y finalmente cae hasta cero cuando el banco de registros es tan sobrado que no existe necesidad de ninguna reejecución.

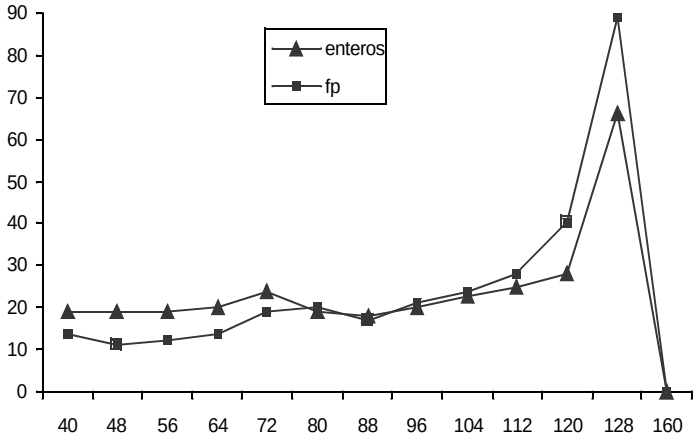


**Figura 5.8** Porcentaje de reducción en el número de reejecuciones. Vp-DSY *versus* vp-LAER.

Continuando con el estudio del rendimiento obtenido por vp-LAER, se van a considerar ahora dos situaciones distintas. Primero se va a suponer el caso en que el banco de registros se encuentre fuera de cualquier camino crítico del procesador, de tal forma que no exista relación entre en tiempo de ciclo del procesador y el tamaño del banco de registros. Después se considerará la situación opuesta, donde se supondrá que el tiempo de ciclo está estrechamente relacionado con el tiempo de acceso al banco de registros.

### 5.3.1 IPC *versus* el tamaño del banco de registros

Para el caso de que el tiempo de acceso al banco de registros sea menor que el tiempo de ciclo del procesador, puede establecerse como buena medida del rendimiento el IPC que se obtiene. En esta situación, todo diseñador de procesadores podría incorporar vp-LAER, bien como un mecanismo que provee IPC sin tener que cambiar el tamaño del banco de registros o bien como un mecanismo capaz de ajustar el tamaño del banco de registros a una configuración que no provoca pérdida en el rendimiento.

### VP-LAER COMO PROVEEDOR DE IPC SIN CAMBIAR EL TAMAÑO DEL BANCO DE REGISTROS

En la Figura 5.7 se ha mostrado cómo varía el IPC en función del número de registros físicos. Se puede ver cómo la separación de IPC entre el renombre convencional y vp-LAER va aumentando a la vez que el banco de registros se hace más ajustado (su tamaño va disminuyendo). En otras palabras, vp-LAER hace que el rendimiento del procesador sea menos dependiente del tamaño del banco de registros. En códigos enteros, son interesantes los resultados obtenidos para bancos de registros con tan pocos registros como 40, 48, 56 ó 64 ya que es posible alcanzar con ellos un IPC cercano al que se obtiene con un banco de registros sobrado. Los *speed-ups* conseguidos en esos puntos son respectivamente de un 37%, 15%, 9% y 6%. En códigos fp, es muy elevado el *speed-up* conseguido en el rango de los tamaños más pequeños del banco de registros: desde un 70% hasta un 25% para 40 y 72 registros. Por otro lado, un banco de 96 registros (*speed-up* del 12%) se comporta tan bien como un banco de registros sobrado.

### VP-LAER COMO AJUSTADOR DEL TAMAÑO DEL BANCO DE REGISTROS SIN PÉRDIDA DE RENDIMIENTO

Vp-LAER puede utilizarse para mantener un IPC dado mientras se reduce el número de registros, el cual está se relaciona linealmente con el área que ocupa el banco de registros dentro del chip [FJC96]. La Tabla 5.1 muestra dos ejemplos de configuraciones sin y con vp-LAER que alcanzan respectivamente un 90% y un 85% del máximo IPC promedio convencional. Las reducciones que se obtienen en área se encuentran entre un 25% y un 30%, siendo ligeramente mayores para códigos fp.

| | **códigos enteros** | | | **códigos fp** | | |
|---|---|---|---|---|---|---|
| %IPC max | conv | vp-LAER | **reducc. %** | conv | vp-LAER | **reducc. %** |
| 90% | 59 r | 44 r | 25.4% | 100 r | 72 r | 28% |
| 85% | 55 r | 40 r | 27.3% | 92 r | 64 r | 30.4% |

**Tabla 5.1** Tamaños del banco de registros sin y con vp-LAER al permitir dos pequeñas degradaciones del IPC.

## 5.3.2 *IPS versus* el tamaño del banco de registros

Procesadores muy segmentados pueden tener que elegir entre utilizar un banco de registros que necesita varios ciclos de acceso y que posiblemente limita el IPC obtenido o utilizar un banco de registros que sólo necesita un ciclo de acceso, pero posiblemente limita el tiempo de ciclo del procesador [BTM+02][FJC96][TEE+96][TEL95]. La Figura 5.9 muestra el rendimiento del procesador expresado en *BIPS* (*Billions of Instructions Per Second*), suponiendo que el tiempo de ciclo del procesador es igual a tiempo de acceso del banco de registros más lento, en nuestro caso es el banco de registros fp debido a su mayor número de puertos. Los resultados se han calculado utilizando el modelo de tiempos de Rixner et al. para una tecnología de 0.18μ

[RDK+00]. En las Figuras B.10 a B.12 del Apéndice B se puede encontrar esta misma medida pero por separado y para cada uno de los programas de prueba.

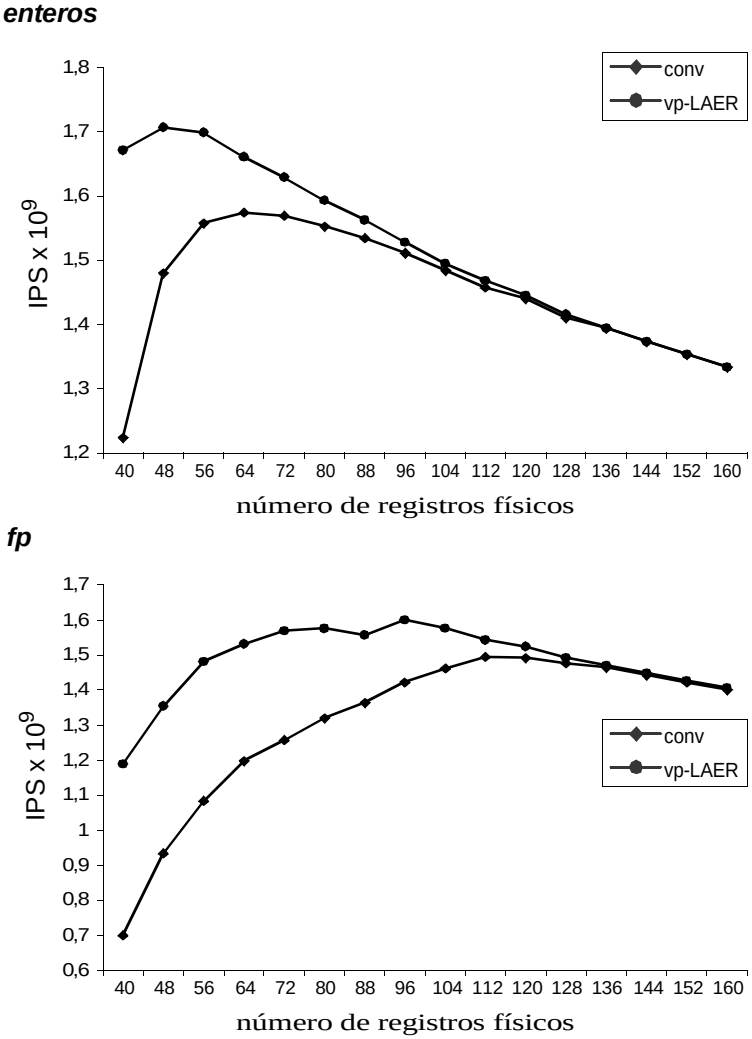


**Figura 5.9** Media armónica de BIPS *versus* el número de registros físicos para los renombres convencional y vp-LAER.

Con el banco de registros situado en un camino crítico, el trabajo del diseñador consiste en encontrar el tamaño del banco de registros más ajustado que dé a la vez el rendimiento más alto. Utilizando vp-LAER se pueden marcar puntos de diseño donde se requieren de forma simultánea menos registros y donde se alcanzan más BIPS. En códigos enteros, la mejor opción de diseño reduciría el número de registros de 72 a 48, mientras que los BIPS aumentarían en un 8.8%. En códigos de tipo fp, el mejor diseño reduciría el banco de registros de 120 registros hasta los dos puntos atractivos de 96 y 80 registros, con un incremento de BIPS de un 7.2% y un 5.6% respectivamente.

Si lo que se quiere es optimizar el diseño de un procesador que ejecuta tanto aplicaciones de tipo entero como de tipo fp, se puede considerar el promedio de BIPS para una carga de trabajo mez-

clada. Haciendo esto, el mejor punto de diseño obtenido, se mueve ahora desde 120 hasta 72 registros con un incremento de BIPS del 9.1%.

Para finalizar con los resultados, al igual que se ha hecho en los Capítulos 3 y 4, se verán a continuación los resultados relacionados con la *utilización* de los registros y el adelanto en la ejecución de instrucciones.

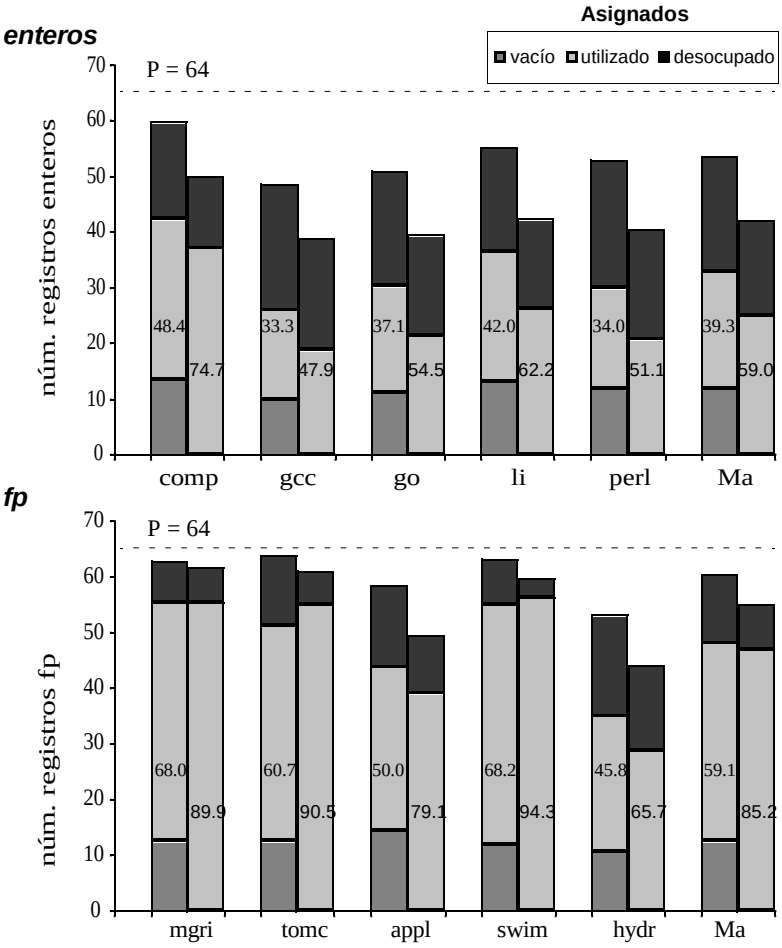


**Figura 5.10** Número de registros Asignados que se encuentran en los estados Vacío, Utilizado o Desocupado. Mecanismo de liberación convencional y vp-LAER. 64ent+64fp registros.

### 5.3.3 *Utilización* de los registros

En este apartado se revisa como mejora la *utilización* cuando se aplica vp-LAER a un banco de registros de 64 registros enteros y 64 registros fp. Recordar que este está siendo el punto de diseño representativo al hablar del concepto de *utilización* de los registros. La Figura 5.10 muestra el número promedio de registros Asignados que se encuentran en estado Vacío, Preparado o Desocupado. Se comparan los valores obtenidos por el mecanismo de renombre convencional y por vp-LAER. La figura muestra también el valor que tiene la *utilización* para ambos mecanismos. En la Tabla A.4 del Apéndice A se pueden consultar los valores con los que reproducir las columnas de vp-LAER que aparecen en la Figura 5.10.

Como era de esperar, con vp-LAER no hay registros en el estado Vacío y el número de registros en estado Desocupado ha disminuido. Se han incrementado el número promedio de registros en estado Utilizado y Libres. En concreto, el número de registros en estado Utilizado se ha visto incrementado en un 19% y un 32% respectivamente para los programas de tipo entero y de tipo fp. En algunos programas como *compress* o *tomcatv* ese valor llega a ser de hasta un 29% y 42% respectivamente. El número de registros en estado Desocupado se ha visto decrementado en un 16% y un 33%. De nuevo en los casos particulares de *compress*, *swim* o *tomcatv*, estos valores llegan al 28%, 59% y 53% respectivamente. Por consiguiente, con vp-LAER se consigue mejorar la utilización de los registros de forma importante a base de eliminar el estado Vacío y de conseguir que haya menos registros Desocupados. En concreto, la *utilización* ha aumentado desde un 39.3% hasta un 59% en los códigos enteros y desde un 59.1% hasta un 85.2% en los códigos fp, mientras que el número promedio de registros Libres ha aumentado más del doble en los dos tipos de códigos.

Vp-LAER elimina completamente el efecto negativo de aumentar el estado Desocupado que poseía el mecanismo vp-DSY actuando en solitario y mantiene el mismo decremento de ese estado que se obtenía con la liberación anticipada. Respecto del estado Vacío, también vp-LAER elimina el efecto negativo de aumentarlo que poseía el mecanismo de liberación anticipada actuando en solitario ya que ahora este estado ha desaparecido por completo.

### 5.3.4 Adelanto en la ejecución de las instrucciones

Como se presentó en los Capítulos 3 y 4, las instrucciones son *anticipadas* con vp-DSY y también con la liberación anticipada. A continuación se va a ver cómo también existe anticipación en las instrucciones al utilizar el mecanismo combinado vp-LAER. De nuevo con las mismas reglas que se presentaron en el Capítulo 3 en el apartado que lleva el mismo nombre que éste, se han evaluado tanto el porcentaje de instrucciones anticipadas como el número de ciclos que estas instrucciones se anticipan bajo el mecanismo vp-LAER. Se ha supuesto también un banco de registros con $64_{ent}+64_{fp}$ registros físicos.

La Figura 5.11.a muestra el porcentaje de *loads* que fallan en memoria *cache*, desglosados en antipados y no anticipados. En promedio y para programas fp el porcentaje de *loads* que se anticipan es del 87% mientras que para los programas enteros es del 22%. Tres casos interesantes son *tomcatv*, *swim* y *li* donde se anticipan respectivamente el 99.5%, el 93.1% y el 50% de los *loads* que fallan en memoria *cache*. La Figura 5.11.b muestra el porcentaje de saltos condicionales que son mal predichos -anticipados y no anticipados-. En promedio, para programas enteros el 7% de los saltos condicionales mal predichos son anticipados y el 72% para los fp. Ahora los que más se benefician de la anticipación de instrucciones son *tomcatv*, *mgrid* y *compress* ya que anticipan el 96.7%, 86.4% y 15.8% de los saltos condicionales mal predichos.

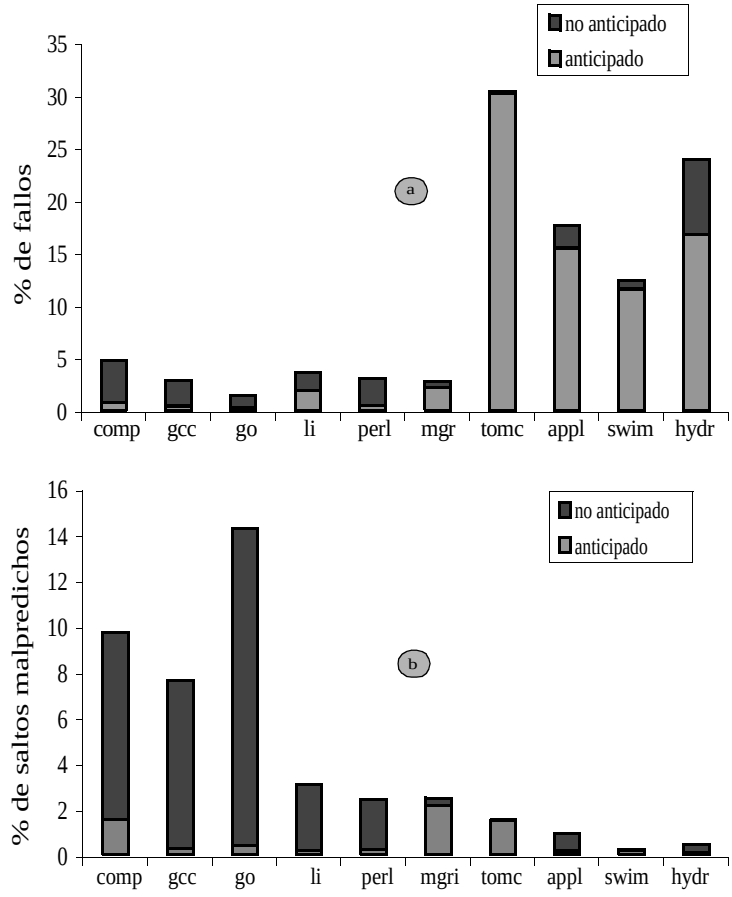


**Figura 5.11** Porcentaje de *loads* que fallan en *cache* -anticipados y no anticipados- **(a)** y porcentaje de saltos condicionales que son mal predichos -anticipados y no anticipados- **(b)**. Mecanismo vp-LAER.

Otra estadística interesante la representaba el número promedio de ciclos que una instrucción es anticipada. Se ha obtenido que con vp-LAER las instrucciones de *load* son anticipadas un promedio de 7 ciclos para los programas enteros y 21.3 ciclos para fp. Para los saltos condicionales esos mismos cálculos dan 5.3 y 31.1 respectivamente. Para las instrucciones de división, multiplicación y simples esos cálculos dan respectivamente 2.5, 3.5 y 6.3 para los programas enteros y 14.4, 13.6 y 17.3 para los fp.

El renombre de registros virtuales-físicos y la liberación anticipada de vp-LAER permiten la entrada en la ventana de lanzamiento de más instrucciones que el mecanismo de renombre convencional y que además se ejecuten antes. De nuevo, este beneficio sigue siendo más importante para programas fp que para enteros. Los programas fp poseen un porcentaje mayor de instrucciones anticipadas y en promedio se anticipan también un número de ciclos mayor que en los programas enteros. Para un mismo número de registros físicos en el procesador, el mecanismo vp-LAER permite tener un número mayor de instrucciones en vuelo y de registros libres de lo que se permitiría con un esquema convencional. Sin embargo, el hecho de que los programas

enteros posean mayor número de saltos que son mal predichos limita tanto el número de instrucciones que pueden encontrarse en vuelo como el aprovechamiento de las mejoras que aporta la liberación anticipada.

| % | códigos fp | | | códigos enteros | | |
|---|---|---|---|---|---|---|
| | vp-DSY | liber. anticip. | **vp-LAER** | vp-DSY | liber. anticip. | **vp-LAER** |
| **loads anticipados** | 86% | 46% | 87% | 20% | 3% | 22% |
| **saltos anticipados** | 74% | 19% | 72% | 6% | 2% | 7% |

**Tabla 5.2** Porcentaje de *loads* y saltos que se anticipan. Vp-DSY, liberación anticipada y vp-LAER.

| CICLOS | códigos fp | | | códigos enteros | | |
|---|---|---|---|---|---|---|
| | vp-DSY | liber. anticip. | **vp-LAER** | vp-DSY | liber. anticip. | **vp-LAER** |
| **loads** | 26.9 | 16.2 | 21.3 | 9.4 | 7.2 | 7 |
| **saltos** | 28 | 13.2 | 31.1 | 4.8 | 3.8 | 5.3 |
| **división** | 13.3 | 9.3 | 14.4 | 2.6 | 2 | 2.5 |
| **multiplicación** | 13.9 | 0.1 | 13.6 | 3 | 3.4 | 3.5 |
| **simples** | 17.1 | 4.4 | 17.3 | 6.1 | 4 | 6.3 |

**Tabla 5.3** Número de ciclos que son anticipados todo tipo de instrucciones. Vp-DSY, liberación anticipada y vp-LAER.

En las tablas 4.2 y 4.3 se ha comparado el grado de anticipación obtenido con vp-LAER con los que se obtuvieron con los mecanismos vp-DSY y liberación anticipada actuando por separado. En general se observa que el beneficio de la anticipación en vp-LAER está más cercano al mejor obtenido con cualquiera de sus componentes actuando por separado. Sólo en el caso del número de ciclos que son anticipadas las instrucciones de *loads* (consultar la Tabla 5.3) éste es mayor en vp-DSY que en vp-LAER para todo tipo de programas. Las reejecuciones habilitan la anticipación de las instrucciones de *load* y se ha visto un poco más arriba como el número de reejecuciones disminuye hasta un 20% con el mecanismo vp-LAER.

## 5.4 Implementación

De la misma forma que se hizo en los Capítulos 3 y 4, se va a analizar el impacto que supondría la implementación del mecanismo vp-LAER. En concreto, se va a cuantificar el coste de almacenamiento, tiempo de ciclo y consumo de energía que tendrían aquellas partes del *hardware* propuesto consideradas más críticas. El análisis de la influencia sobre el tiempo de ciclo y el consumo de energía se ha realizado utilizando el modelo de tiempo de acceso y potencia de Rixner et al. para una tecnología de 0.18 μm. [RDK+00].

### 5.4.1 Almacenamiento

De nuevo y para cuantificar el coste de almacenamiento del mecanismo vp-LAER se han utilizado algunos de los parámetros del procesador Alpha 21264 [Gwe96]. Este procesador es capaz de gestionar hasta un máximo de veinte saltos pendientes de confirmarse, tiene un ROS de 80 entradas y un banco de registros físico con 80 registros enteros y 72 registros fp. Con estos parámetros las estructuras que necesita la parte de la liberación anticipada (RelQue y ROS en la Figura 5.2) ocupan ahora alrededor de 1.4 KBytes. Las dos tablas LUsT (entera y fp) añaden una ocupación de alrededor de 128 Bytes. Respecto de la parte de los registros virtuales-físicos, el total de coste de almacenamiento que habría que añadir sería de 197 Bytes: GMT y PMT implementada como una memoria CAM con un número de entradas igual al número de registros físicos. Todo lo anterior hace que en total el mecanismo vp-LAER precise de 1.72 KBytes de almacenamiento, valor que sigue quedando muy por debajo de lo que ocupan las tablas utilizadas en el Alpha 21264 para la predicción de los saltos -alrededor de 4.4 KBytes[Gwe96]-. Por tanto, la implementación de vp-LAER es viable en el contexto de microprocesadores de alto rendimiento.

### 5.4.2 Tiempo de Ciclo

Para analizar el coste relativo al tiempo de ciclo se han utilizado los mismos parámetros con los que se hicieron todas las medidas y que están indicados en el Capítulo 3 -apartado de *Entorno Experimental*-. El mecanismo vp-LAER no va a añadir ningún retardo al camino crítico del procesador al compararlo con el mecanismo convencional. Sólo la necesidad de acceder a la tabla PMT para liberar los registros podría suponer un retraso de un ciclo. Supondremos a la PMT diseñada como se dijo en la subsección 5.2.2 y por esa razón no la vamos a considerar en este análisis.

Se van a utilizar las tablas GMT y LUsT como estructuras clave del análisis. Al igual que se hizo en los capítulos previos, las dos tablas se consideran partidas siendo $GMT.VP_{fp}$ y $LUsT.ROS_{id}$ aquellas partes que mayor tiempo de acceso poseen. Para los parámetros utilizados en las evaluaciones, $GMT.VP_{fp}$ dispone de 32 entradas de tamaño 8 bits y un total de puertos de 68: 60 de lectura y 8 de escritura. $LUsT.ROS_{id}$ dispone también de 32 entradas de 7 bits y un total de 56 puertos: 32 de lectura y 24 de escritura. El número de puertos para cualquier banco de registros entero es de 48 y para cualquier banco de registros fp es de 54. La Figura 5.12.a muestra como el tiempo de acceso a cualquiera de las anteriores tablas es menor que el de cualquier banco de registros. Para el banco de registros entero de menor tamaño (40 registros físicos) el acceso más costoso -GMT- es un 26% menor.

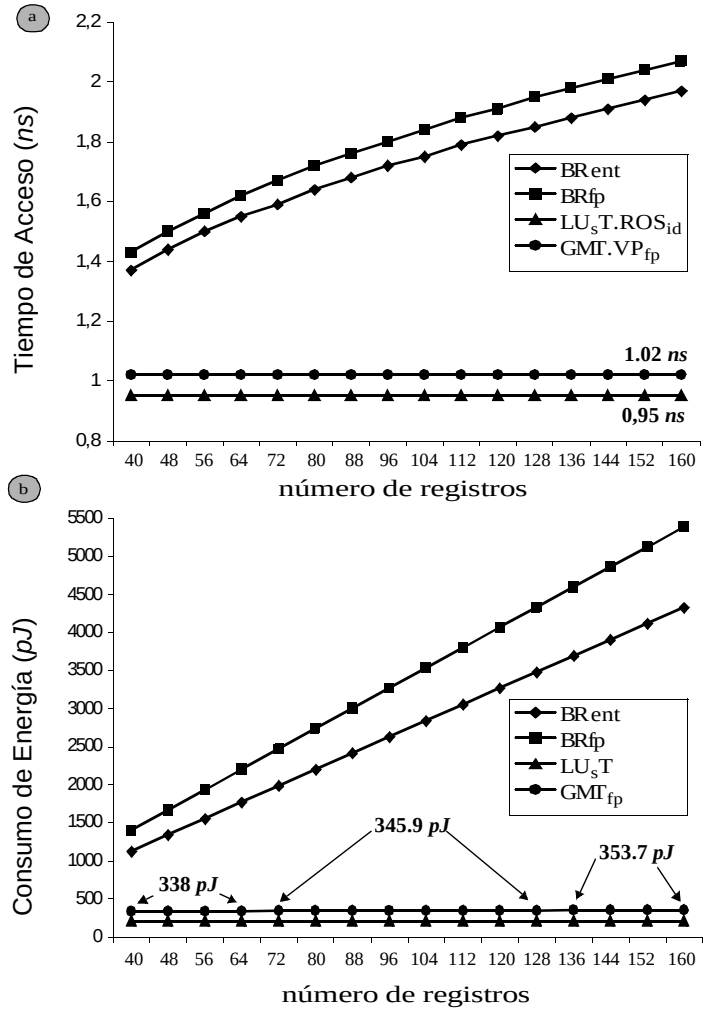


**Figura 5.12** Tiempo de acceso y consumo de energía para las tablas GMT, LUsT y el banco de registros. Mecanismo vp-LAER.

### 5.4.3 Consumo de Energía

En el estudio del consumo de energía se han considerado como estructuras clave a las tablas GMT, LUsT. Se va a volver a suponer de nuevo una contribución nula al consumo de energía por parte de la RelQue. Una implementación de este conjunto de bits como un verdadero registro de desplazamiento de dos dimensiones, en el que la mayoría del tiempo los valores que han de desplazarse son ceros no supondría ningún incremento de energía al mecanismo. Por otra parte, el consumo de energía que pueden tener las instrucciones que han de reejecutarse se podría resolver utilizando algún mecanismo de *reutilización* [SS97] aunque esto no forma parte de este análisis. También se ha visto en el apartado de resultados que este aporte de energía sería menor en vp-LAER que en vp-DSY al producir el primero menor número de reejecuciones.

Además también se vio como las reejecuciones siguen teniendo el efecto positivo de adelantar la ejecución de las instrucciones.

En la Figura 5.12.b se pueden ver los valores calculados con el modelo de Rixner et al. [RDK+00] para el consumo de energía de la tablas GMT y LUsT. De nuevo, para todas las configuraciones, el consumo de la energía queda por debajo de lo que consume el menor de los bancos de registros. La GMT sólo consume el 31% de la energía que consume el menor de los bancos de registros (el de 40 registros).

En el apartado de resultados se ha visto como el mecanismo vp-LAER puede utilizarse como un medio de reducir el número de registros necesarios para obtener un determinado nivel de rendimiento. Por ejemplo, se vio que permitiendo una degradación del 15% en IPC con un mecanismo convencional de renombre se necesita un banco de registros con 55 registros enteros + 92 registros fp, mientras que con un mecanismo vp-LAER basta con disponer de un banco registros con 40 registros enteros + 64 registros fp. Al comparar el consumo de energía de estas dos alternativas, se observa lo siguiente:

$E_{convencional}(BR_{55ent} + BR_{92fp}) = 1528.1\ pJ + 3134.9\ pJ = 4663\ pJ,$

y con vp-LAER:

$E_{vp\text{-}LAER}(BR_{40ent} + BR_{64fp} + 2 \times LUsT + GMT_{ent} + GMT_{fp}) =$

$1123.7\ pJ + 2202.6\ pJ + 2x\ 202.8\ pJ + 301.3\ pJ + 338\ pJ = 4371\ pJ.$

La energía consumida es menor con el mecanismo vp-LAER que con el convencional -diferencia de 291.8 pJ-. En el Capítulo 3 se vio que la reducción en el tamaño del banco de registros obtenida por el mecanismo vp-DSY no era suficiente para compararla con la energía que consumían sus estructuras añadidas -169.6 pJ más de consumo-. Sin embargo y al unir vp-DSY con la liberación anticipada, el mecanismo vp-LAER resultante obtiene una reducción en el tamaño del banco de registros que es comparable con la energía que consumen las estructuras que permiten esa reducción -291.8 pJ menos de consumo-.

## 5.5 Conclusiones

En este Capítulo se presenta vp-LAER, un mecanismo de renombre alternativo para procesadores con ejecución fuera de orden en los que tanto las versiones de los registros especuladas como las no especuladas están mezcladas dentro de un banco de registros centralizado. Vp-LAER incrementa la utilización del banco de registros a base de incrementar el número promedio de registros físicos asignados que poseen lecturas pendientes. Incrementar la utilización significa mejorar el rendimiento en aplicaciones que poseen un ILP relativo alto y que son capaces de ejercer fuerte presión sobre el banco de registros. Esta situación ocurre de forma típica en aplicaciones de tipo fp (en un amplio rango de tamaños de bancos de registros) y en aplicaciones de tipo entero que trabajan con bancos de registros ajustados.

Vp-LAER optimiza la utilización del banco de registros usando dos mecanismos complementarios. Por un lado, la asignación de registros físicos se retarda lo máximo posible, justo hasta el

final de la etapa de ejecución. Los registros virtuales-físicos permiten realizar esta asignación tardía a base de separar en el proceso de renombre la parte del seguimiento de dependencias entre las instrucciones de la parte de la gestión de los lugares donde se almacenan los valores de las instrucciones. Por otro lado, los registros se liberan tan pronto como exista garantía de que no se van a utilizar más, mucho antes que con el mecanismo convencional y manteniendo la capacidad de realizar una recuperación precisa frente a excepciones.

Vp-LAER es una alternativa más compleja que la del renombre convencional, pero esta complejidad se encuentra fuera de los caminos críticos del procesador, lo cual permite su uso en todo el rango de diseño de bancos de registros. Vp-LAER se puede ver como un mecanismo para ajustar el tamaño del banco de registros de un procesador equipado con un mecanismo de renombre convencional. Además, se ha visto como el coste de almacenamiento dedicado a vp-LAER y el consumo de energía que genera es viable en el contexto de microprocesadores de alto rendimiento.

Diseños que posean una velocidad del reloj moderada y en los cuales el tiempo de acceso al banco de registros sea menor que el tiempo de ciclo del procesador pueden utilizar vp-LAER en dos direcciones: bien para incrementar el IPC mientras se mantiene el mismo tamaño del banco de registros, o bien para reducir el tamaño del banco de registros mientras se mantiene el mismo IPC. Por ejemplo, un banco de registros ajustado de 64 registros de tipo fp y con vp-LAER ofrece un 28% más de IPC. También, se puede sustituir un banco de registros de 55 registros de tipo entero por uno de 40 registros sin pérdida de IPC si se reemplaza el renombre convencional por vp-LAER.

En el otro extremo, el tiempo de ciclo del procesador podría estar determinado por el tiempo de acceso del banco de registros. En esta situación, el trabajo del diseñador consiste en encontrar el tamaño más pequeño que obtiene el mayor promedio de IPS. Con una carga de trabajo combinada (entero más fp) comparándolo con el renombre convencional, vp-LAER incrementa en un 9.1% el IPS mientras reduce en un 40% el tamaño del banco de registros para una tecnología de 0.18μ.

## 5.6 Referencias


[BTM+02] E. Borch, E. Tune, S. Manne, and J. Emer, "Loose Loops Sink Chips," *in Proceedings of the 8th International Symposium on High-Performance Computer Architecture* (HPCA 02), pp. 299-310, February 2002.

[FJC96] K.I. Farkas, N.P. Jouppi, and P. Chow, "Register File Considerations in Dynamically Scheduled Processors," *in Proceedings of the 2nd International Symposium on High-Performance Computer Architecture* (HPCA 96), pp. 40-51, February 1996.

[GVG+97] A. González, M. Valero, J. González, and T. Monreal, "Virtual Registers," *in Proceedings of the 4th International Conference on High Performance Computing* (HiPC 97), pp. 364-369, December 1997.

[GGV98] A. González, J. González, and M. Valero, "Virtual-Physical Registers," *in Proceedings of the 4th International Symposium on High-Performance Computer Architecture* (HPCA 98), pp. 175-184, January/ February 1998.

[Gwe96] L. Gwennap, "Digital 21264 Sets New Standard," *Microprocessor Report*, vol. 10, no. 14, pp. 11-16, October 1996.

[HP87] W.W. Hwu, and Y.N. Patt, "Checkpoint Repair for Out-of-Order Execution Machines," *in Proceedings of the 14th International Symposium on Computer Architecture* (ISCA 87), pp. 18-26, June 1987.

[MGV+99a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, "Delaying Physical Register Allocation Through Virtual-Physical Registers," *in Proceedings of the 32nd International Symposium on Microarchitecture* (MICRO 99), pp. 186-192, November 1999.

[MGV+00a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, "Dynamic Register Renaming Through Virtual-Physical Registers," *in The Journal of Instruction-Level Parallelism*, vol. 2, May 2000. http://www.jilp.org/vol2.

[MVG+02] T. Monreal, V. Viñals, A. González, and M. Valero, "Hardware Schemes for Early Register Release," *in Proceedings of the International Conference on Parallel Processing* (ICPP 02), pp. 5-13, August 2002.

[RDK+00] S. Rixner, W.J. Dally, B. Khailany, P. Mattson, U.J. Kapasi, and J.D. Owens, "Register Organization for Media Processing," *in Proceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 375-386, January 2000.

[TEE+96] D.M. Tullsen, S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo, and R.L. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," *in Proceedings of the 25th International Symposium on Computer Architecture* (ISCA 96), pp. 191-202, May 1996.

[TEL95] D.M. Tullsen, S.J. Eggers, and H.M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *in Proceedings of the 22nd International Symposium on Computer Architecture* (ISCA 95), pp. 392-403, June 1995.

CAPÍTULO 6

# Conclusiones y líneas abiertas

En esta Tesis se han presentado y evaluado distintas estrategias que actúan sobre el mecanismo de renombre de registros de un procesador convencional con ejecución fuera de orden y con un banco de registros centralizado. Todas las estrategias tienen como objetivo incrementar la utilización de los registros, o lo que es lo mismo, aumentar el número promedio de los registros que están asignados y que poseen lecturas pendientes. Aumentar la utilización de los registros es beneficioso para aplicaciones que poseen mucho paralelismo a nivel de instrucción y que por lo tanto son capaces de ejercer fuerte presión sobre el banco de registros. Las aplicaciones de tipo entero con bancos de registros ajustados y las aplicaciones de tipo fp con cualquier tamaño del banco de registros son las que más se benefician.

En el Capítulo 2, hemos visto como el esquema de renombre de registros convencional asigna y libera los registros de forma conservadora, incrementando innecesariamente la necesidad de registros. Todas las estrategias que hemos propuesto actúan en las partes de la asignación y la liberación de los registros físicos mejorando en mayor o menor grado al mecanismo convencional. Todas han sido presentadas como posibles herramientas de diseño que hacen que el banco de registros sea un componente menos crítico en el procesador y se ha visto que poseen una utilidad doble:

- Permitir aumentar el rendimiento mientras se mantiene el mismo tamaño para el banco de registros. Esta utilidad tiene sentido para sistemas con bancos de registros ajustados donde el IPC va a mejorar sin degradar el tiempo de ciclo.

- Permitir reducir el tamaño del banco de registros para ajustar su tiempo de acceso al tiempo de ciclo del procesador sin perder rendimiento. Esta utilidad tiene sentido tanto para sistemas con bancos de registros sobrados como para ajustados que se encuentran en el camino crítico del procesador.

Respecto a la asignación de los registros se ha presentado un nuevo esquema de renombre que retarda la asignación de los registros físicos y se denomina registros virtuales-físicos. Este es un nuevo tipo de registros que permite que los registros físicos se asignen a las instrucciones al finalizar su etapa de ejecución, en lugar de hacerlo en la etapa de decodificación como el renombre convencional. En el Capítulo 3 se presentan las distintas técnicas que hemos propuesto para evitar el problema del *abrazo mortal* presente en el mecanismo de registros virtuales-físicos. Se han comparado todas ellas y se ha demostrado que la técnica más efectiva es aquella que asigna los registros físicos bajo demanda y en ausencia de registros resuelve la asignación robándolos a instrucciones más jóvenes. Esta técnica se ha denominado vp-DSY.

La asignación tardía de registros utilizando la técnica vp-DSY muestra que se pueden alcanzar *speed-ups* para códigos fp que varían desde un 54% hasta un 2% en la medida en que el tamaño del banco de registros incrementa hasta alcanzar un tamaño sobrado. En códigos de tipo entero, se obtienen *speed-ups* un poco más moderados pero que pueden llegar a ser de hasta un 33% para bancos de registros muy ajustados.

Aplicando la técnica vp-DSY, se ha observado que la utilización de los registros aumenta de forma significativa. Por ejemplo, se ha evaluado que se puede conseguir una reducción en el número de registros enteros y fp de hasta un 26% y un 24% respectivamente, a la vez que se mantiene el mismo promedio de IPC que obtiene el esquema convencional. Esta reducción en el número de registros se traduce directamente en un menor tiempo de acceso al banco de registros, el cual es muy probable que sea uno de los grandes problemas para los procesadores futuros.

También se ha visto como resultado de la técnica el que permite que las instrucciones de memoria puedan hacer *prefetch* de sus datos y que los saltos condicionales se resuelvan con anticipación. Se ha cuantificado que el porcentaje de *loads* que fallan en memoria y que son anticipados es de un 20% y un 86% para programas de tipo entero y fp respectivamente. Por su parte, en códigos enteros el 6% de los saltos condicionales mal predichos son resueltos con anticipación.

En cuanto a la liberación de los registros, se han presentado un par de técnicas de complejidad incremental que permiten liberar antes que el mecanismo convencional. La liberación anticipada actúa cuando la instrucción que utiliza un registro por última vez alcanza su etapa de *commit*, en lugar de hacerlo en la etapa de *commit* de la instrucción que lo redefine. En el Capítulo 4 se presentan los dos mecanismos diseñados para la liberación anticipada y que se han denominado *básico* y *extendido* respectivamente. Ambos mecanismos actúan en presencia de ejecución especulada y permiten una recuperación precisa frente a excepciones. Se ha visto también que el *hardware* añadido para la implementación de la liberación anticipada se encuentra fuera del camino crítico del procesador y que es perfectamente viable en coste de almacenamiento. Se han comparado los dos mecanismos y se ha visto que sus rendimientos dependen mucho del tipo de código que se ejecuta. Se ha observado que un esquema híbrido que utiliza el mecanismo *básico* para los registros de tipo fp y el mecanismo *extendido* para los registros de tipo entero consigue un buen equilibrio entre coste y rendimiento.

La experimentación realizada ha mostrado *speed-ups* prometedores especialmente para códigos fp y para un amplio rango de tamaños de bancos de registros ajustados. En promedio, estos *speed-ups* para el mecanismo *extendido* varían desde un 10% hasta un 2% en la medida en que el tamaño del banco de registros incrementa hasta alcanzar el tamaño sobrado. En algunos pro-

gramas se llegan a alcanzar *speed-ups* de hasta un 16% para bancos de registros ajustados. En códigos de tipo entero, la liberación anticipada de registros sólo resulta efectiva para bancos de registros muy ajustados donde se pueden alcanzar *speed-ups* de hasta un 11%.

De forma alternativa, se ha visto que en lo que se refiere a microarquitecturas típicas y utilizando la liberación anticipada, el tamaño del banco de registros puede reducirse en un 12.5% y un 9% para códigos enteros y fp respectivamente sin reducción de IPC. Punto este importante a considerar ya que una reducción en el tamaño del banco de registros se traduce directamente en un tiempo de acceso menor.

Respecto de la cantidad de *prefetch* de sus datos que las instrucciones de memoria pueden hacer y de la resolución anticipada de los saltos condicionales, se ha cuantificado el porcentaje de *loads* que fallan en memoria y que son anticipados utilizando la liberación anticipada: un 3% y un 46% para programas de tipo entero y fp respectivamente. Ahora el 2% de los saltos condicionales mal predichos son resueltos con anticipación en los programas de tipo entero.

Finalmente, la técnica vp-DSY de asignación tardía de los registros se ha combinado con el mecanismo *extendido* de liberación anticipada en un único mecanismo. Esta unión ha dado origen a una nueva técnica *hardware* que se ha denominado vp-LAER, en la que los registros físicos se asignan a las instrucciones tan tarde como sea posible -justo al final de la etapa de ejecución- y se liberan tan pronto como exista garantía de que no se van a utilizar más, antes que con el mecanismo convencional. Todo lo anterior manteniendo la capacidad de realizar una recuperación precisa frente a excepciones.

Vp-LAER ha resultado ser una alternativa de renombre más compleja que la convencional, pero esta complejidad se encuentra fuera de los caminos críticos del procesador, lo cual permite su uso en todo el rango de diseño de bancos de registros. Vp-LAER puede seguir viéndose como un mecanismo que sirve para reducir el tamaño del banco de registros de un procesador equipado con un mecanismo de renombre convencional. Diseños que posean una velocidad del reloj moderada y en los cuales el tiempo de acceso al banco de registros se encuentre dentro del tiempo de ciclo del procesador, pueden utilizar vp-LAER en dos direcciones: bien para incrementar el IPC mientras se mantiene el mismo tamaño del banco de registros, o bien para reducir el tamaño del banco de registros mientras se mantiene el mismo IPC. Se ha visto además que tanto el coste de almacenamiento dedicado a vp-LAER como el consumo de energía que se genera son reducidos y que por tanto la implementación de vp-LAER es viable en el contexto de microprocesadores de alto rendimiento.

Los resultados obtenidos por vp-LAER superan a los de sus componentes actuando por separado -vp-DSY, *extendido*-. Se ha observado que se pueden alcanzar ahora *speed-ups* aditivos y que para códigos fp varían desde un 70% hasta un 12% en la medida en que el tamaño del banco de registros se incrementa hasta alcanzar el tamaño sobrado. En códigos de tipo entero, se obtienen *speed-ups* que pueden llegar a ser de hasta un 37% para bancos de registros muy ajustados.

Respecto de la reducción en el tamaño del banco de registros con una mínima pérdida de rendimiento, se ha observado que se puede reducir el número de registros enteros y fp hasta un 27% y un 30% respectivamente. Por otro lado, para el caso de que el tiempo de ciclo del procesador esté determinado por el tiempo de acceso del banco de registros, se ha observado que vp-LAER

con una carga de trabajo combinada -entero más fp- incrementa en un 9.1% el IPS mientras reduce en un 40% el tamaño del banco de registros (tecnología de 0.18μ).

El porcentaje de *loads* que fallan en memoria y que son anticipados es ahora de un 22% y un 87% para programas de tipo entero y fp respectivamente. En programas enteros, el 7% de los saltos condicionales mal predichos son resueltos de forma anticipada.

## Líneas Abiertas

Se van a citar a continuación las principales líneas abiertas que han aparecido derivadas del trabajo presentado en esta Tesis.

En primer lugar, se plantea la modificación del entorno experimental sobre el que se ha desarrollado toda la evaluación de esta Tesis. Sería interesante modelar un procesador con unos valores de parámetros que se ajusten más a los de las microarquitecturas actuales. En la misma línea, se pretende también extender la evaluación a códigos de programas que sean más adecuados a las necesidades actuales de procesamiento. Trabajar en un nuevo entorno con un tipo de códigos diferente introducirá sin duda nuevos compromisos y variantes de diseño para cualquiera de las técnicas que se han presentado.

En segundo lugar, respecto a la asignación tardía de los registros, se ha observado que el problema de las reejecuciones en el mecanismo DSY representa una limitación muy grave. Van a ser motivo de estudio nuevas estrategias que eliminen la necesidad de reejecutar las instrucciones y que a la vez permitan seguir obteniendo buenos rendimientos.

En tercer lugar, en lo que a la liberación anticipada de los registros respecta, la experimentación ha mostrado que todavía se puede ganar algo más actuando en esta línea. Va a ser así interesante pensar en nuevas técnicas que de forma más agresiva intenten liberar los registros anticipadamente y que a la vez sigan manteniendo una recuperación precisa frente a excepciones.

Finalmente de forma más general, se ha planteado aplicar las técnicas desarrolladas en esta Tesis al ámbito de procesadores *multithreading*. Una de las mayores limitaciones que posee este tipo de procesadores es la necesidad de un banco de registros enorme. Técnicas como las que aquí se han presentado, cuyo objetivo es el de optimizar al máximo la utilización de los registros, permitirán a este tipo de procesadores superar cualquier limitación relacionada con el tamaño del banco de registros.

# Apéndice A: Utilización de los registros

Número de registros que se encuentran Libres y Asignados en un procesador con renombre de registros convencional (L = 32, P = 64ent + 64fp, N = 128):

| | | **Asignados** | | |
|---|---|---|---|---|
| | **Libres** | Vacío | Utilizado | Desocupado |
| *compress* | *4.3* | *13.4* | *28.9* | *17.4* |
| *gcc* | *15.2* | *9.8* | *16.3* | *22.7* |
| *go* | *13* | *11.2* | *18.9* | *20.9* |
| *li* | *8.8* | *13.2* | *23.2* | *18.8* |
| *perl* | *11.1* | *11.9* | *18* | *23* |
| Promedio | 10.5 | 11.9 | 21 | 20.6 |

| | | **Asignados** | | |
|---|---|---|---|---|
| | **Libres** | Vacío | Utilizado | Desocupado |
| *mgrid* | *1.2* | *12.5* | *42.7* | *7.6* |
| *tomcatv* | *0.3* | *12.6* | *38.7* | *12.4* |
| *applu* | *5.7* | *14.5* | *29.1* | *14.7* |
| *swim* | *0.8* | *11.9* | *43.1* | *8.2* |
| *hydro2d* | *10.8* | *10.5* | *24.4* | *18.3* |
| Promedio | 3.8 | 12.4 | 35.6 | 12.2 |

**Tabla A.1** Valores que reproducen las gráficas de la Figura 2.4 del Capítulo 2.

Número de registros que se encuentran Libres y Asignados en un procesador con el mecanismo vp-DSY (L = 32, P = 64ent + 64fp, N = 128):

| | **Asignados** | | |
|---|---|---|---|
| | **Libres** | Utilizado | Desocupado |
| *compress* | *11.2* | *35.4* | *17.4* |
| *gcc* | *23* | *18.3* | *22.7* |
| *go* | *21.8* | *21.3* | *20.9* |
| *li* | *19.4* | *25.9* | *18.7* |
| *perl* | *20.7* | *20.3* | *23* |
| Promedio | 19.2 | 24.2 | 20.6 |

| | **Asignados** | | |
|---|---|---|---|
| | **Libres** | Utilizado | Desocupado |
| *mgrid* | *2.4* | *53* | *8.6* |
| *tomcatv* | *1.5* | *49.5* | *13* |
| *applu* | *11.9* | *36.9* | *15.2* |
| *swim* | *2.9* | *52.7* | *8.4* |
| *hydro2d* | *18* | *27.8* | *18.2* |
| Promedio | 7.3 | 44 | 12.7 |

**Tabla A.2** Valores que reproducen las columnas de la derecha de las gráficas de la Figura 3.7 del Capítulo 3.

Número de registros que se encuentran Libres y Asignados en un procesador con el mecanismo extendido de liberación anticipada de registros (L = 32, P = 64ent + 64fp, N = 128):

| | **Asignados** | | | |
|---|---|---|---|---|
| | **Libres** | Vacío | Utilizado | Desocupado |
| *compress* | *5.2* | *14.8* | *31.1* | *12.9* |
| *gcc* | *16.5* | *10.4* | *16.7* | *20.4* |
| *go* | *14.5* | *11.9* | *19.4* | *18.2* |
| *li* | *9.6* | *14.1* | *24* | *16.3* |
| *perl* | *12.4* | *12.9* | *18.8* | *19.9* |
| Promedio | 11.6 | 12.8 | 22 | 17.6 |

| | **Asignados** | | | |
|---|---|---|---|---|
| | **Libres** | Vacío | Utilizado | Desocupado |
| *mgrid* | *1.2* | *12.6* | *44* | *6.2* |
| *tomcatv* | *0.3* | *15.1* | *42.4* | *6.2* |
| *applu* | *6.4* | *16.1* | *30.7* | *10.8* |
| *swim* | *0.8* | *13.6* | *46.2* | *3.4* |
| *hydro2d* | *11.7* | *12* | *25.8* | *14.5* |
| Promedio | 4.1 | 13.9 | 37.8 | 8.2 |

**Tabla A.3** Valores que reproducen las columnas de la derecha de las gráficas de la Figura 4.8 del Capítulo 4.

Número de registros que se encuentran Libres y Asignados en un procesador con el mecanismo vp-LAER (L = 32, P = 64ent + 64fp, N = 128):

| | **Asignados** | | |
|---|---|---|---|
| | **Libres** | Utilizado | Desocupado |
| *compress* | *14.2* | *37.2* | *12.6* |
| *gcc* | *25.2* | *18.6* | *20.2* |
| *go* | *24.5* | *21.5* | *18* |
| *li* | *21.6* | *26.4* | *16* |
| *perl* | *23.6* | *20.6* | *19.8* |
| Promedio | 21.8 | 24.9 | 17.3 |

| | **Asignados** | | |
|---|---|---|---|
| | **Libres** | Utilizado | Desocupado |
| *mgrid* | *2.5* | *55.3* | *6.2* |
| *tomcatv* | *3.1* | *55.1* | *5.8* |
| *applu* | *14.5* | *39.2* | *10.3* |
| *swim* | *4.4* | *56.2* | *3.4* |
| *hydro2d* | *20* | *28.9* | *15.1* |
| Promedio | 8.9 | 46.9 | 8.2 |

**Tabla A.4** Valores que reproducen las columnas de la derecha de las gráficas de la Figura 5.10 del Capítulo 5.

# Apéndice B: Rendimiento por programa

Media armónica de IPC en función del tamaño del banco de registros y para las tres configuraciones de asignación: convencional, vp-NRR y vp-DSY:

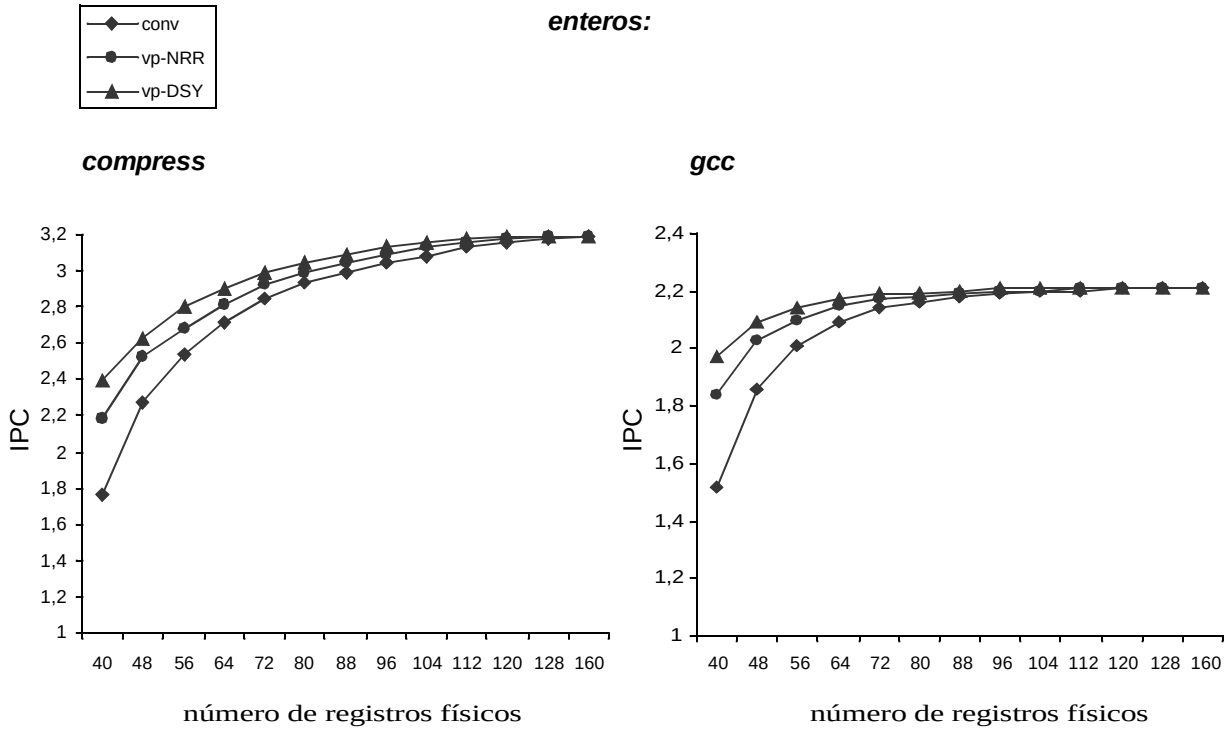


**Figura B.1** Estas gráficas reproducen por programa el mismo resultado de la Figura 3.6 del Capítulo 3.

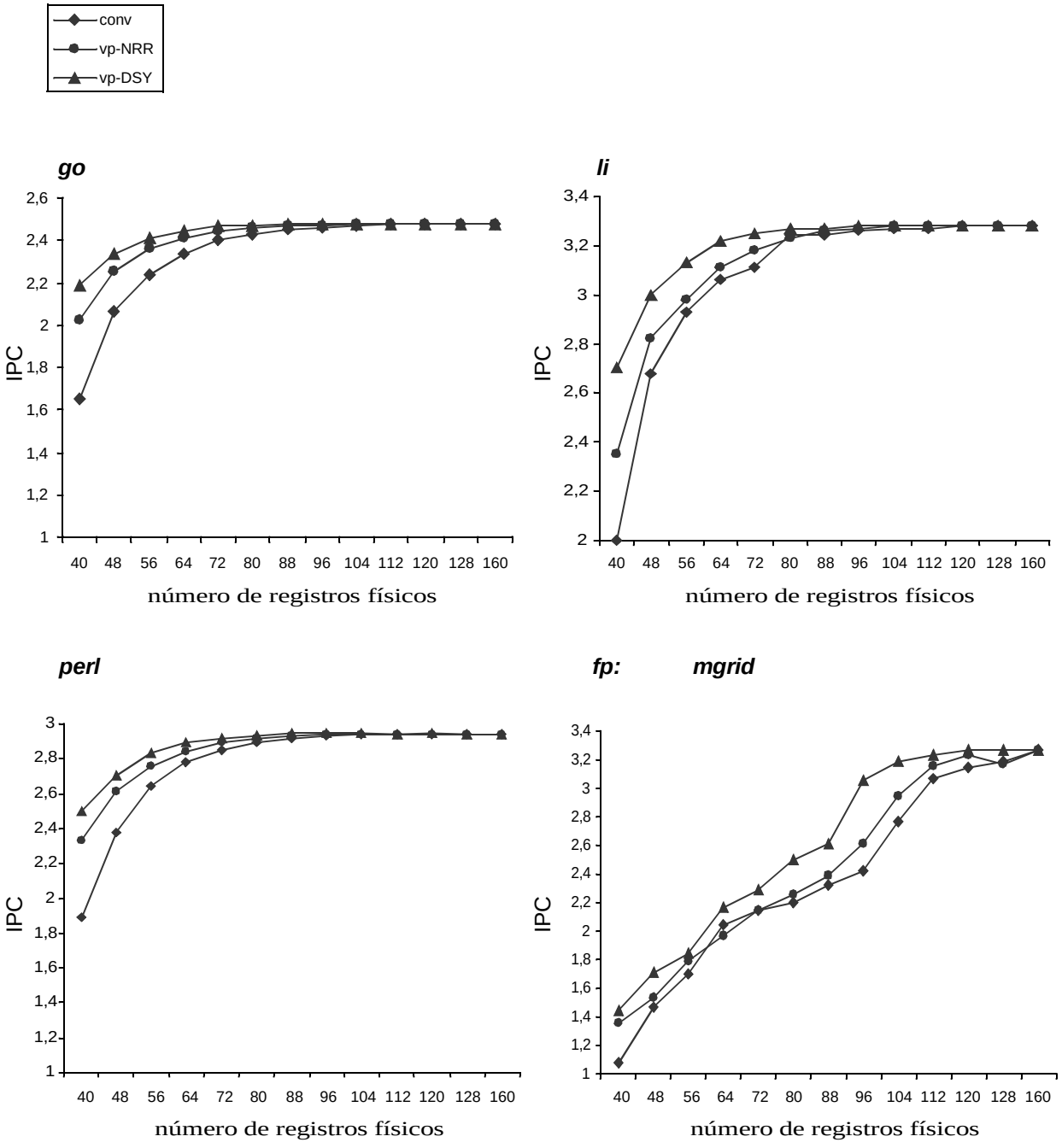


**Figura B.2** Continuación de la Figura B.1.

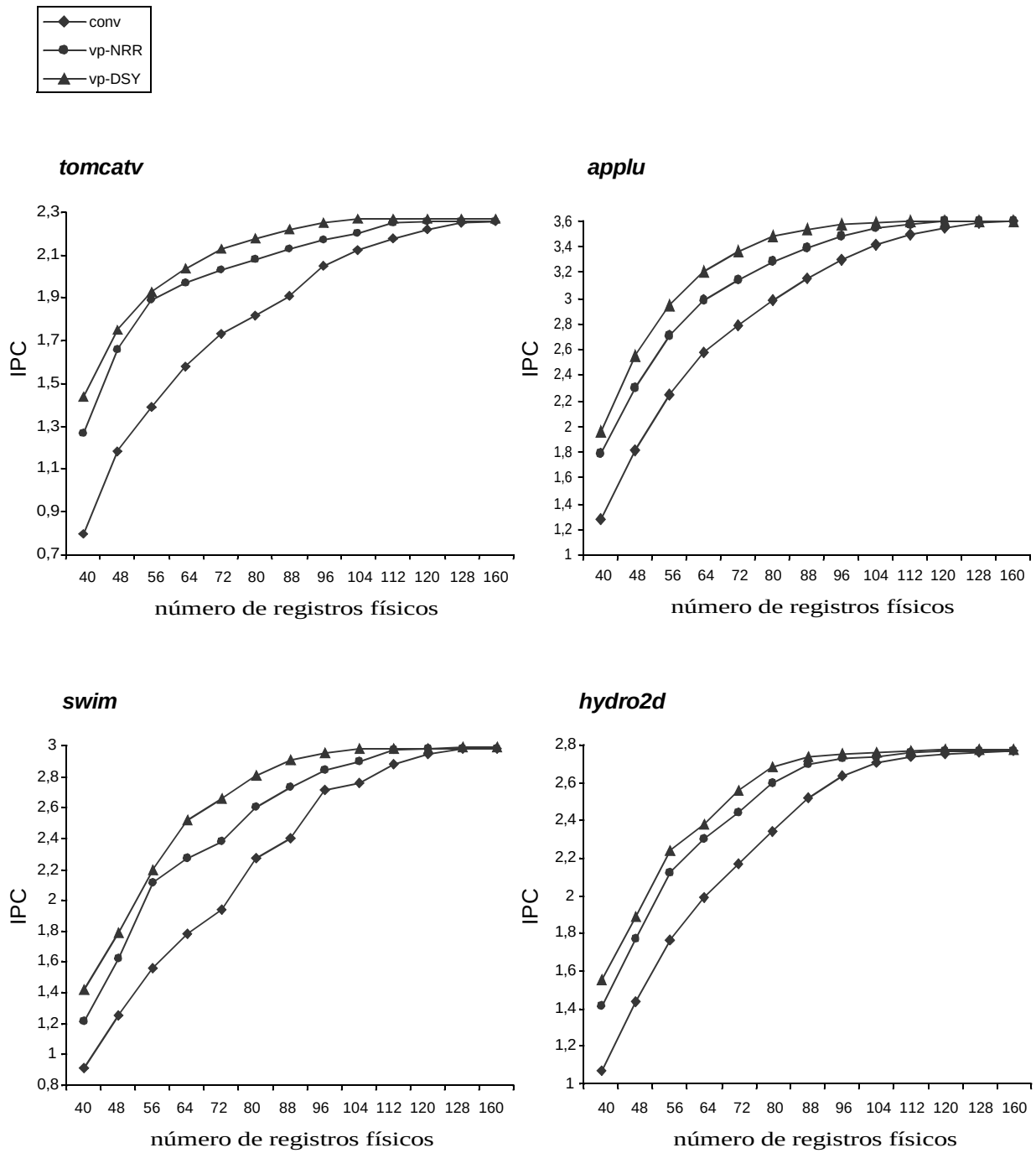


**Figura B.3** Continuación de la Figura B.1.

Media armónica de IPC en función del tamaño del banco de registros y para las tres configuraciones de liberación: convencional, liberación anticipada -básico y extendido-:

***enteros:***

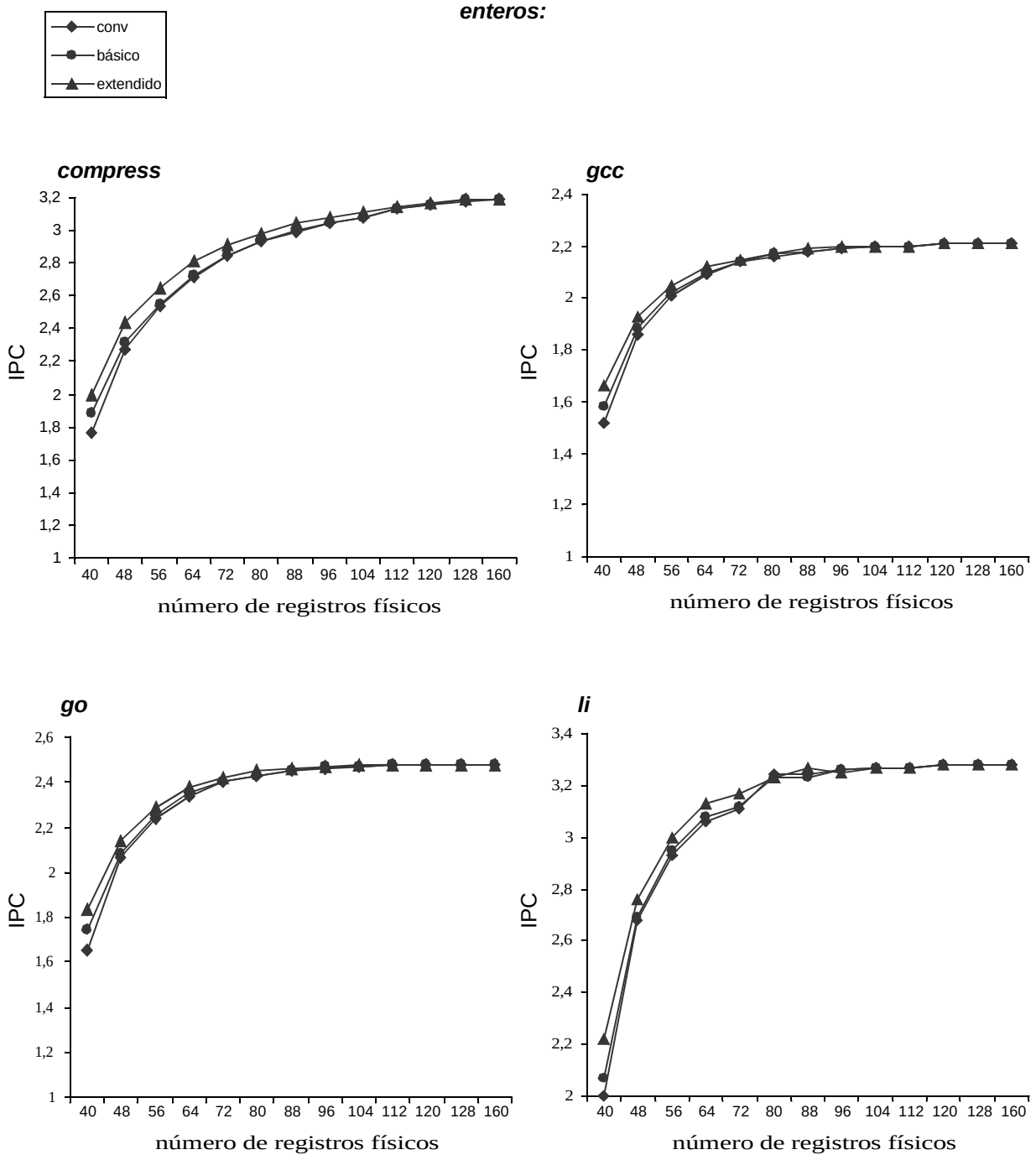


**Figura B.4** Estas gráficas reproducen por programa el mismo resultado de la Figura 4.7 del Capítulo 4.

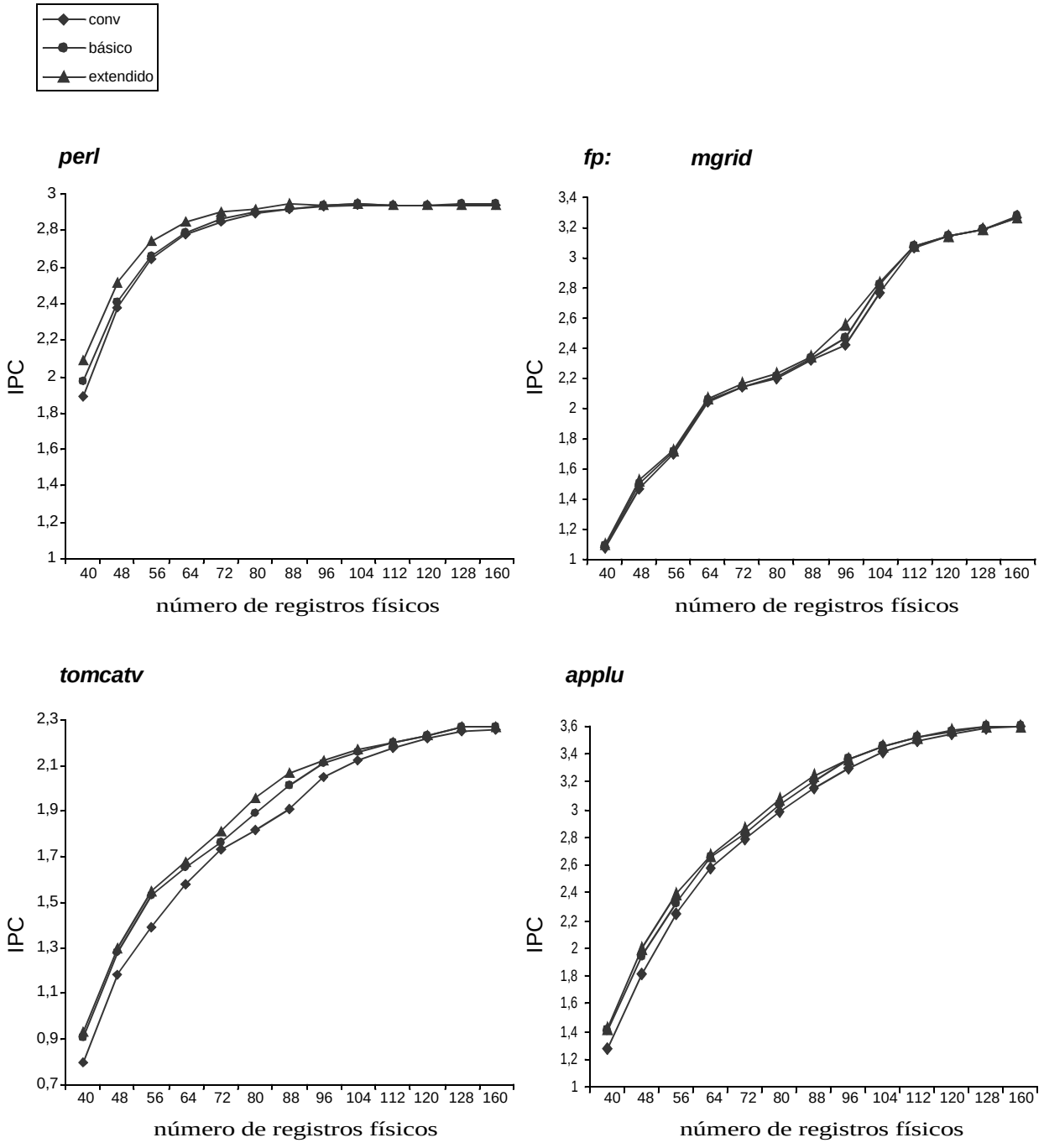


**Figura B.5** Continuación de la Figura B.4.

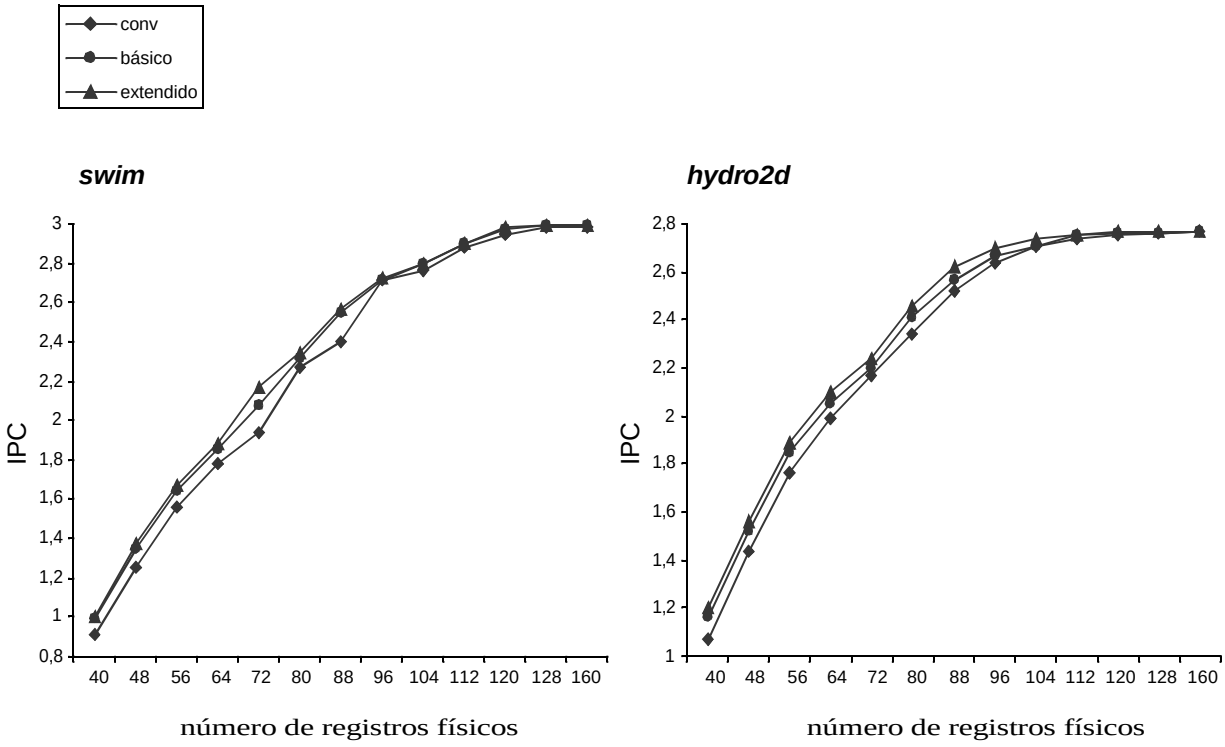


**Figura B.6** Continuación de la Figura B.4.

Media armónica de IPC en función del tamaño del banco de registros y para las cuatro configuraciones de renombre: convencional, liberación anticipada, vp-DSY y vp-LAER:

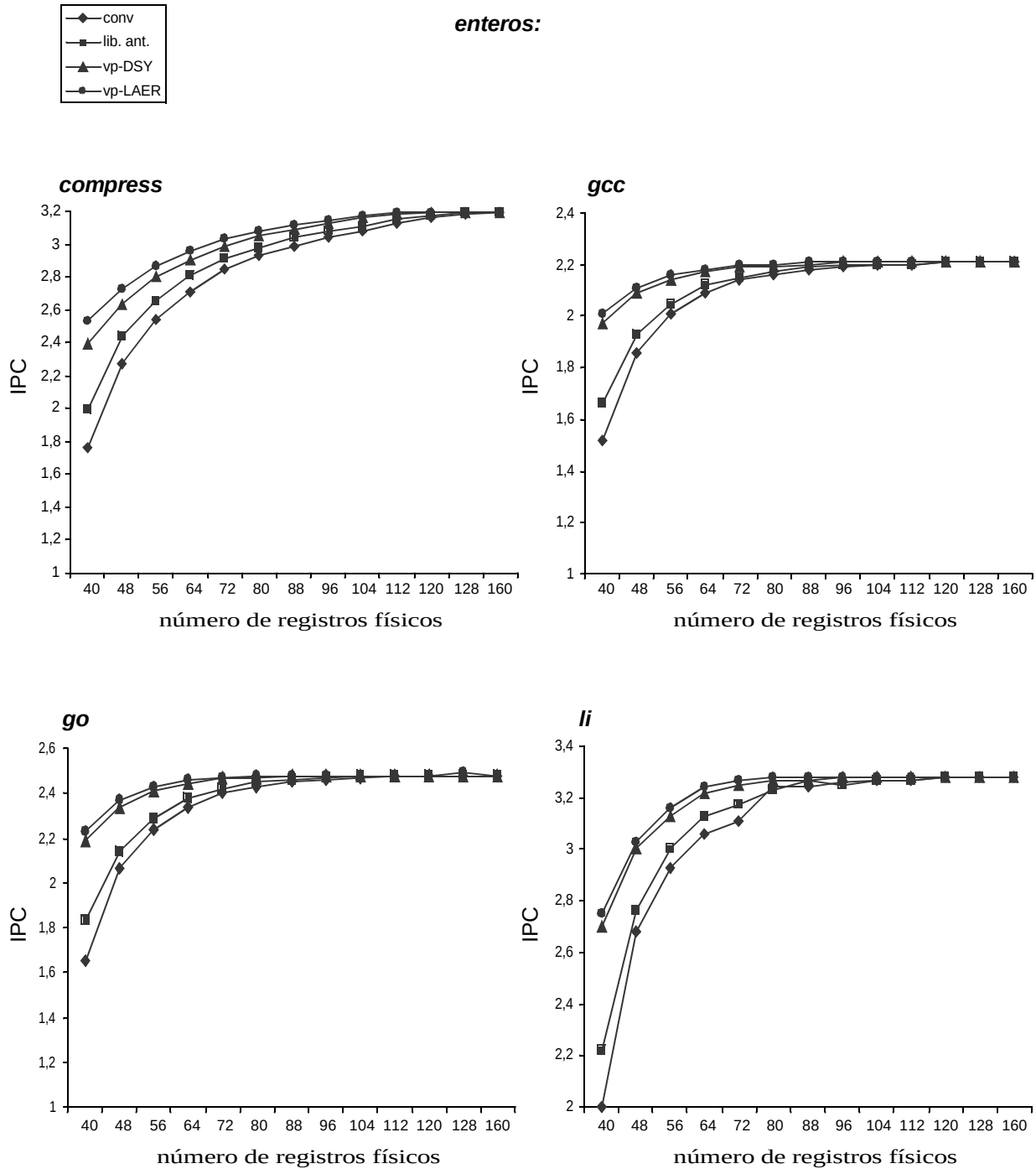


**Figura B.7** Estas gráficas reproducen por programa el mismo resultado de la Figura 5.7 del Capítulo 5.

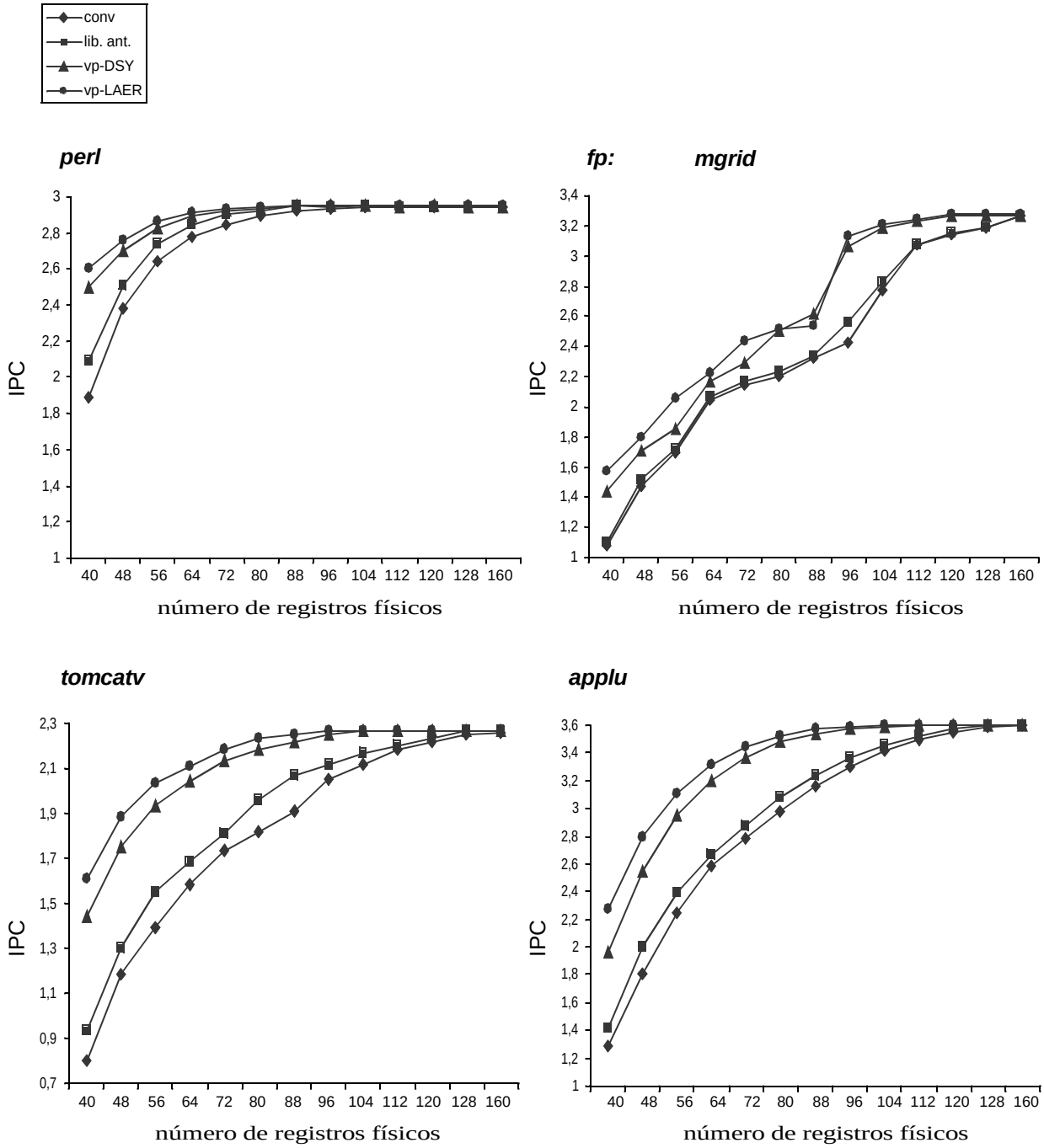


**Figura B.8** Continuación de la Figura B.7.

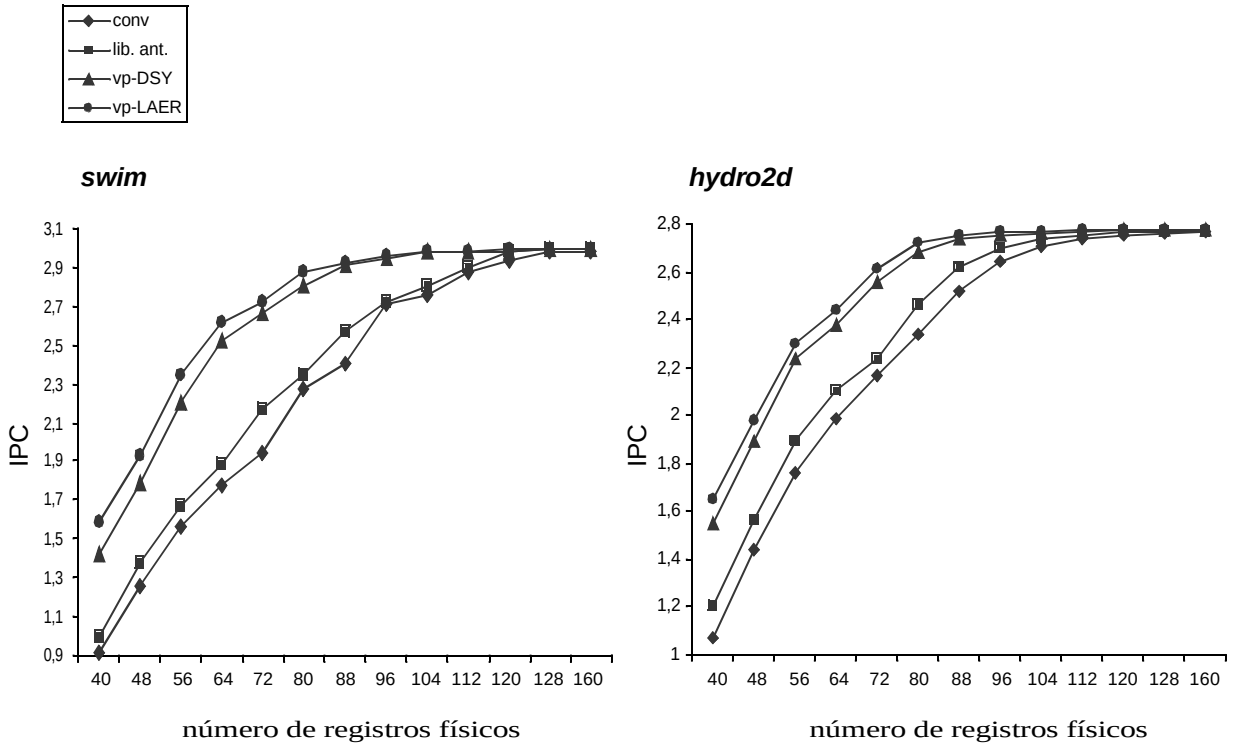


**Figura B.9** Continuación de la Figura B.7.

Media armónica de BIPS *versus* el número de registros físicos para los renombres convencional y vp-LAER:

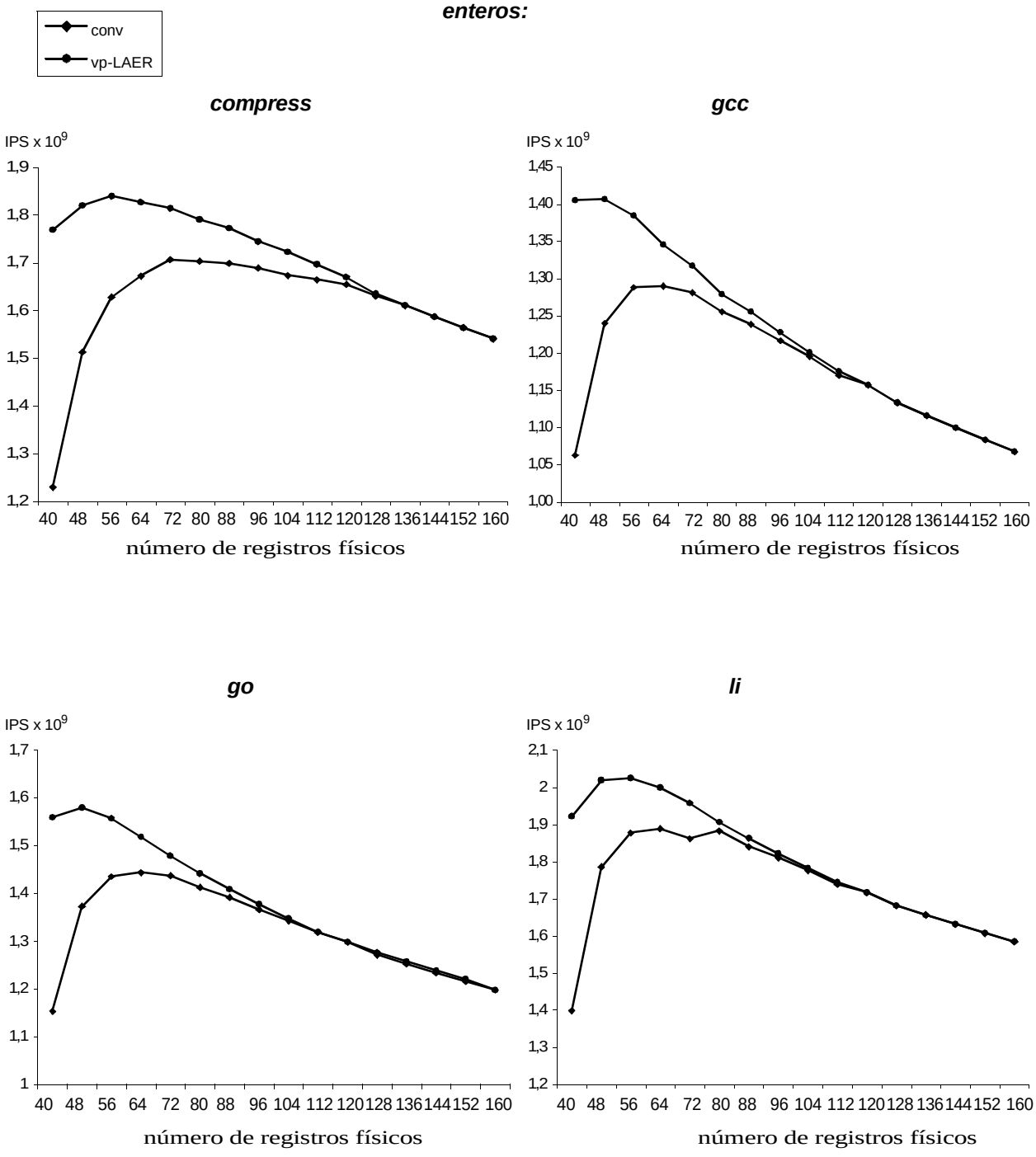


**Figura B.10** Estas gráficas reproducen por programa el mismo resultado de la Figura 5.9 del Capítulo 5.

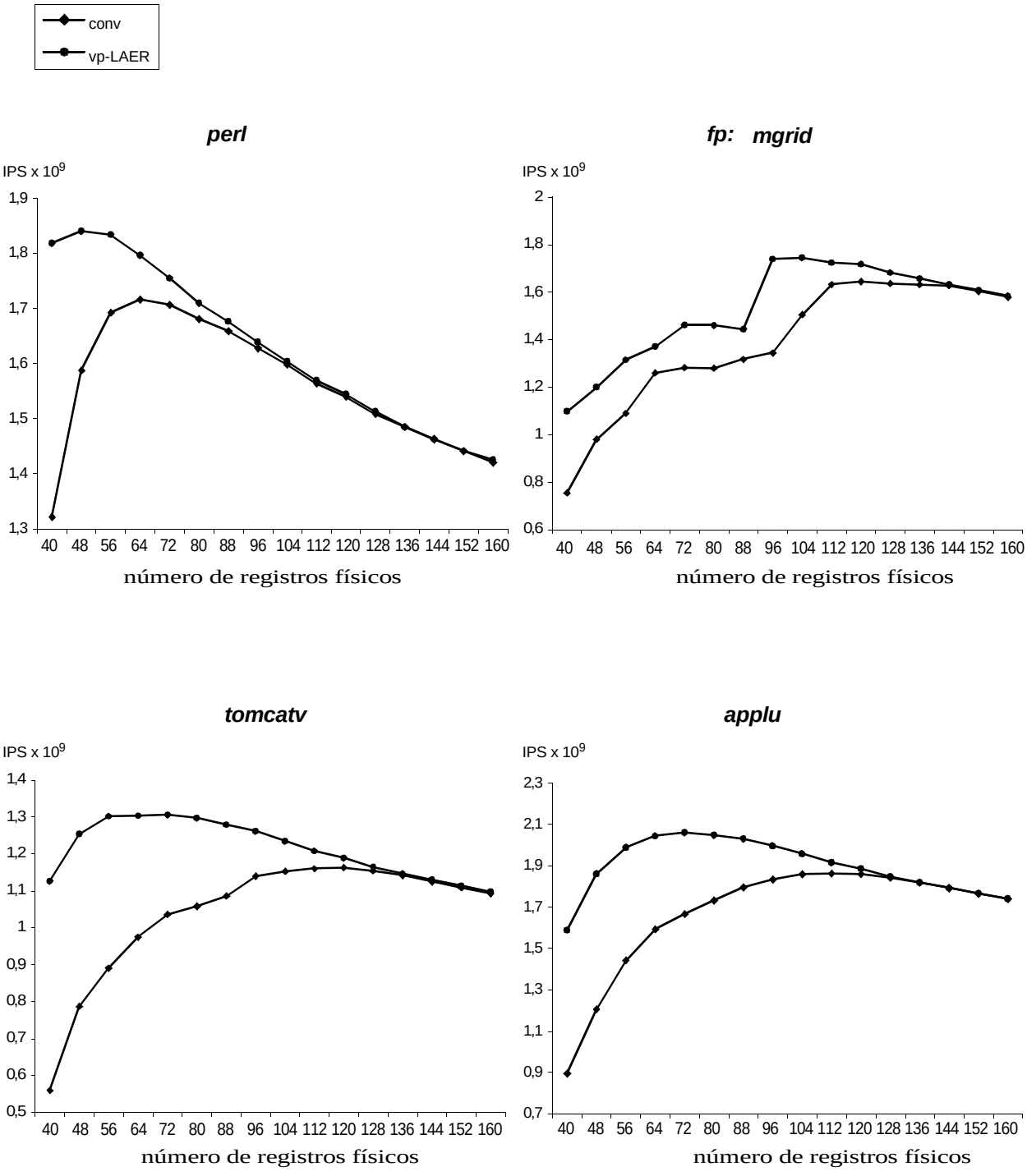


**Figura B.11** Continuación de la Figura B.10.

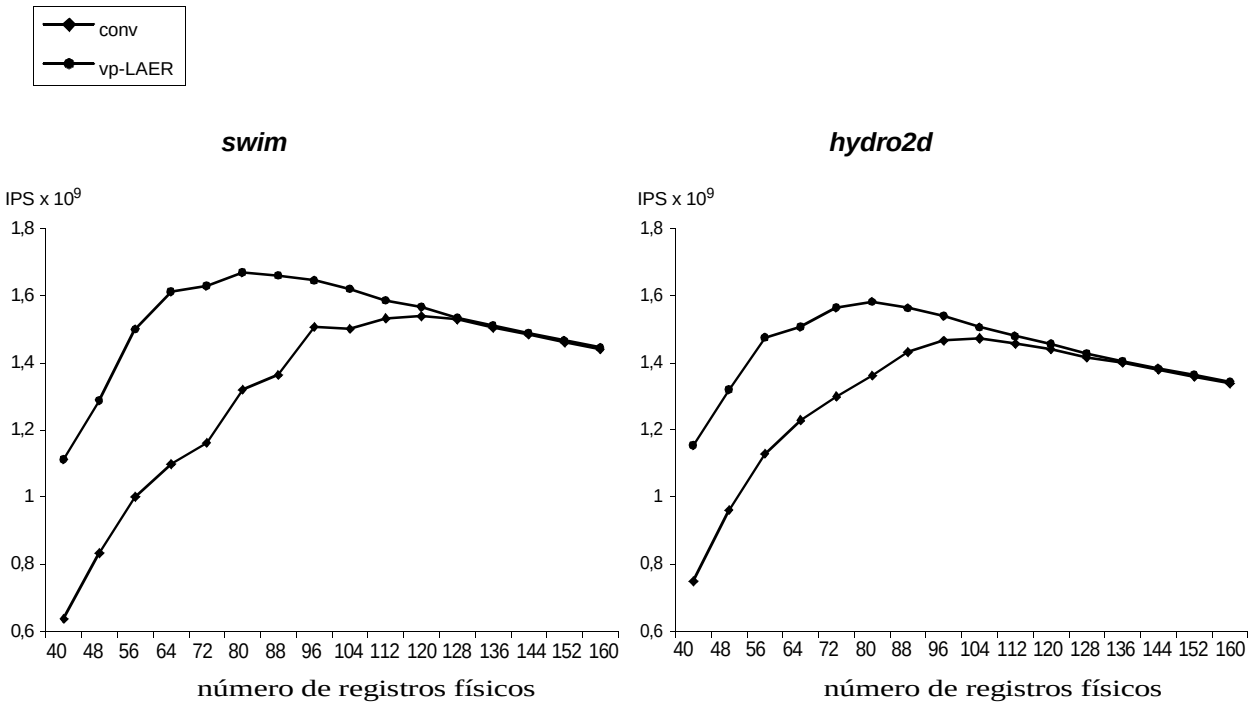


**Figura B.12** Continuación de la Figura B.10.

# Listado completo de Referencias


[AA93] D. Alpert, and D. Avnon "Architecture of the Pentium Microprocessor," *IEEE Micro*, vol. 13, no. 3, pp. 11-21, May/June 1993.

[AST67] D.W. Anderson, F.J. Sparacio, and R.M. Tomasulo, "The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling," *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 8-24, January 1967.

[BA97] D. Burger, and T.M. Austin, *The SimpleScalar Tool Set v2.0*, Technical report 97-1342, University of Wisconsin-Madison, Computer Science Department, June 1997.

[BAH+94] B. Burgess, M. Alexander, Ying-wai Ho, S. Plummer, S. Mallick, D. Ogden, Sung-Ho Park, and J. Slaton "The Power PC 603 Microprocessor: A High Performance, Low Power, Superscalar RISC Microprocessor," *in Proceedings of the 39th IEEE Computer Society International Conference* (COMPCON 94), pp. 300-306, February 1994.

[BAM+93] M.C. Becker, M.S. Allen, C.R. Moore, J.S. Muhich, and D.P. Tuttle, "The PowerPC 601 Microprocessor," *IEEE Micro*, vol. 13, no. 5, pp. 54-68, October 1993.

[BDA01a] R. Balasubramonian, S. Dwarkadas, and D.H. Albonesi, "Dynamically Allocating Processor Resources between Nearby and Distant ILP," *in Proceedings of the 28th International Symposium on Computer Architecture* (ISCA 01), pp. 26-37, June 2001.

[BDA01b] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi, "Reducing the Complexity of the Register File in Dynamic Superscalar Processors," *in Proceedings of the 34th International Symposium on Microarchitecture* (MICRO 01), pp. 237-249, December 2001.

[BDH+94] J. Barreh, S. Dhawan, T. Hicks, and D. Shippy "The POWER2 Processor," *in Proceedings of the 39th IEEE Computer Society International Conference* (COMPCON 94), pp. 389-398, February 1994.

[BK92] G. Blanck, and S. Kreuger, "The SuperSPARC Microprocessor," *in Proceedings of the 37th IEEE Computer Society International Conference* (COMPCON 92), pp. 136-141, February 1992.

[BM00] A. Baniasadi, and A. Moshovos, "Instruction Distribution Heuristics for Quad-Cluster, Dynamically Scheduled, Superscalar Processors," *in Proceedings of the 33rd International Symposium on Microarchitecture* (MICRO 00), pp. 337-347, December 2000.

[BTM+02] E. Borch, E. Tune, S. Manne, and J. Emer, "Loose Loops Sink Chips," *in Proceedings of the 8th International Symposium on High-Performance Computer Architecture* (HPCA 02), pp. 299-310, February 2002.

[CGV+00] J. Cruz, A. González, M. Valero, and N.P. Topham, "Multiple-Banked Register File Architectures," *in Proceedings of the 27th International Symposium on Computer Architecture* (ISCA 00), pp. 316-325, June 2000.

[CPG00] R. Canal, J.M. Parcerisa, and A. González, "Dynamic Cluster Assignment Mechanisms," *in Proceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 133-144, January 2000.

[DWY+92] E. DeLano, W. Walker, J. Yetter, and M. Forsyth, "A High Speed Superscalar PA-RISC Processor," *in Proceedings of the 37th IEEE Computer Society International Conference* (COMPCON 92), pp. 116-121, February 1992.

[FCJ+97] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic, "The Multicluster Architecture: Reducing Cycle Time Through Partitioning," *in Proceedings of the 30th International Symposium on Microarchitecture* (MICRO 97), pp. 149-159, December 1997.

[FJC96] K.I. Farkas, N.P. Jouppi, and P. Chow, "Register File Considerations in Dynamically Scheduled Processors," *in Proceedings of the 2nd International Symposium on High-Performance Computer Architecture* (HPCA 96), pp. 40-51, February 1996.

[GGV98] A. González, J. González, and M. Valero, "Virtual-Physical Registers," *in Proceedings of the 4th International Symposium on High-Performance Computer Architecture* (HPCA 98), pp. 175-184, January/ February 1998.

[Gro90] G.F. Grohoski, "Machine organization of the IBM RISC System/6000 processor," *IBM Journal Resources Development*, vol. 34, no. 1, pp. 37-58, January 1990.

[GVG+97] A. González, M. Valero, J. González, and T. Monreal, "Virtual Registers," *in Proceedings of the 4th International Conference on High Performance Computing* (HiPC 97), pp. 364-369, December 1997.

[Gwe94a] L. Gwennap, "NexGen enters market with 66-MHz Nx586," *Microprocessor Report*, vol. 8, no. 4, pp. 12-17, March 1994.

[Gwe94b] L. Gwennap, "UltraSparc unleashes SPARC performance," *Microprocessor Report*, vol. 8, no. 13, pp. 1-10, October 1994.

[Gwe96] L. Gwennap, "Digital 21264 Sets New Standard," *Microprocessor Report*, vol. 10, no. 14, pp. 11-16, October 1996.

[Gwe95] L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," *Microprocessor Report*, vol. 9, no. 2, pp. 9-15, February 1995.

[Gwe97] L. Gwennap, "MIPS R12000 to Hit 300 MHz," *Microprocessor Report, Micro Design Resources*, vol. 11, no. 13, pp. 1-4, October 1997.

[HP03] J.L. Hennessy, and D.A. Patterson, *Computer Architecture, A Quantitative Approach*, Morgan Kaufmann Publishers, San Francisco, Third Edition 2003.

[HP87] W.W. Hwu, and Y.N. Patt, "Checkpoint Repair for Out-of-Order Execution Machines," *in Proceedings of the 14th International Symposium on Computer Architecture* (ISCA 87), pp. 18-26, June 1987.

[HSU+01] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The Microarchitecture of the Pentium 4 Processor," *Intel Technology Journal Q1*, February 2001.

[Hun95] D. Hunt, "Advanced Performance Features of the 64-bit PA-8000," *in Proceedings of the 40th IEEE Computer Society International Conference* (COMPCON 95), pp. 123-128, March 1995.

[JRB+98] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, and A. Yoaz, "A Novel Renaming Scheme to Exploit Value Temporal Locality Through Physical Register Reuse and Unification," *in Proceedings of the 31st International Symposium on Microarchitecture* (MICRO 98), pp. 216-225, November 1998.

[Kel75] R.M. Keller, "Look-Ahead Processors," *ACM Computing Surveys*, vol. 7, no. 4, pp. 177-195, December 1975.

[Kes99] R.E. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24-36, March/April 1999.

[KP99] J. Keshava, and V. Pentkovski, "Pentium III Processor Implementation Tradeoffs," *Intel Technology Journal Q2*, 1999.

[Lip92] J.S. Liptay, "Design of the IBM Enterprise System/9000 high-end processor," *IBM Journal Resources Development*, vol. 36, no. 4, pp. 713-731, July 1992.

[LPE+99] J.L. Lo, S. S. Parekh, S.J. Eggers, H. M. Levy, and D.M. Tullsen, "Software-Directed Register Deallocation for Simultaneous Multithreaded Processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 9, pp. 922-933, September 1999.

[LTT95] D. Levitan, T. Thomas, and P. Tu, "The PowerPC 620 Microprocessor: A High-Performance Superscalar RISC Microprocessor," *in Proceedings of the 40th IEEE Computer Society International Conference* (COMPCON 95), pp. 285-291, March 1995.

[McL93] E. McLellan, "The Alpha AXP Architecture and 21064 Processor," *IEEE Micro*, vol. 13, no. 3, pp. 36-47, May/June 1993.

[MGV+99a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, "Delaying Physical Register Allocation Through Virtual-Physical Registers," *in Proceedings of the 32nd International Symposium on Microarchitecture* (MICRO 99), pp. 186-192, November 1999.

[MGV+99b] T. Monreal, A. González, M. Valero y V. Viñals, *Registros Virtuales*, Report interno del DIIS, RR-99-08, 1999.

[MGV+00a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, "Dynamic Register Renaming Through Virtual-Physical Registers," *in The Journal of Instruction-Level Parallelism*, vol. 2, May 2000. http://www.jilp.org/vol2.

[MGV+00b] T. Monreal, A. González, V. Viñals y M. Valero, "Liberación Anticipada de Registros," *en XI Jornadas de Paralelismo*, pp. 21-27, Septiembre 2000.

[MPV93] M. Moudgill, K. Pingali, and S. Vassiliadis, "Register Renaming and Dynamic Speculation: an Alternative Approach," *in Proceedings of the 26th International Symposium on Microarchitecture* (MICRO 93), pp. 202-213, November 1993.

[MRF97] M.M. Martin, A. Roth, and C.N. Fischer, "Exploiting Dead Value Information," *in Proceedings of the 30th International Symposium on Microarchitecture* (MICRO 97), pp. 125-135, December 1997.

[MSW+03] O. Mutlu, J. Stark, C. Wilkerson, and Y.N. Patt, "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture* (HPCA 03), pp. 129-140, February 2003.

[MVG+01] T. Monreal, V. Viñals, A. González, and M. Valero, *Early Register Release*, Report interno del DIIS, RR-01-01, pp. 1-21, 2001.

[MVG+02] T. Monreal, V. Viñals, A. González, and M. Valero, "Hardware Schemes for Early Register Release," *in Proceedings of the 31st International Conference on Parallel Processing* (ICPP 02), pp. 5-13, August 2002.

[NCF+02] S.D. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T.J. Sullivan, and T. Grutkowski, "The Implementation of the Itanium 2 Microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, November 2002.

[NF94] K.J. Nowka, and M.J. Flynn, *Wave Pipelining of High Performance CMOS Static RAM*, Technical Report 94/615, Computer Systems Laboratory, January 1994.

[Omo99] A.R. Omondi, *The Microarchitecture of Pipelined and Superscalar Computers*, Kluwer Academic Publishers, Boston, 1999.

[PII97] "Pentium II Processor Developer's Manual," *Intel Corporation 24350-2001,* October 1997.

[PJS97] A.S. Palacharla, N.P. Jouppi, and J.E. Smith, "Complexity-Effective Superscalar Processors," *in Proceedings of the 24th International Symposium on Computer Architecture* (ISCA 97), pp. 206-218, June 1997.

[PPE+97] Y. Patt, S.J. Patel, M. Evers, D.H. Friendly, and J. Stark, "One Billion Transistors, One Uniprocessor, One Chip," *IEEE Computer*, vol. 30, no. 9, pp. 51-57, September 1997.

[Qua00] N. Quach, "High Availability and Reliability in the Itanium Processor," *IEEE Micro*, vol. 20, no. 5, pp. 61-69, September-October 2000.

[RDK+00] S. Rixner, W.J. Dally, B. Khailany, P. Mattson, U.J. Kapasi, and J.D. Owens, "Register Organization for Media Processing," in *Proceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 375-386, January 2000.

[RF72] E.M. Riseman, and C.C. Foster, "Percolation of code to enhance parallel dispatching and execution," *IEEE Transactions on Computers*, vol. 21, no. 12, pp. 1411-1415, December 1972.

[SFK97] D. Sima, T. Fountain, and P. Kacsuk, *Advanced Computer Architectures, A Design Space Approach*, Addison Wesley Longman Inc., New York, 1997.

[Sim00] D. Sima, "The Design Space of Register Renaming Techniques," *IEEE Micro*, vol. 20, no. 5, pp. 70-83, September-October 2000.

[SP85] J.E. Smith, and A.R. Pleszkun, "Implementation of Precise Interrupts in Pipelined Processors," *in Proceedings of the 12th International Symposium on Computer Architecture* (ISCA 85), pp. 36-44, June 1985.

[SP88] J.E. Smith, and A.R. Pleszkun, "Implementing Precise Interrupts in Pipelined Processors," *IEEE Transactions on Computers*, vol. 37, no. 5, pp. 562-573, May 1988.

[SR00] M. Schlansker, and B. Rau, *An Architecture of Instruction Level Parallel Processors*, HP Laboratories Report HPL-1999-111, February 2000.

[SS98] B. Shriver, and B. Smith, "The Anatomy of a High-Performance Microprocessor," *IEEE CS Press*, 1998.

[SS95] J.E. Smith, and G.S. Sohi, "The Microarchitecture of Superscalar Processors," *in Proceedings of the IEEE*, vol. 83, no. 12, pp. 1609-1624, December 1995.

[SS97] A. Sodani, and G.S. Sohi, "Dynamic Instruction Reuse," *in Proceedings of the 24th International Symposium on Computer Architecture* (ISCA 97), pp. 194-205, June 1997.

[Soh90] G.S. Sohi, "Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers," *in IEEE Transactions on Computers*, vol. 39, no. 3, pp. 349-359, March 1990.

[Son97] P. Song, "IBM's Power3 to Replace P2SC," *Microprocessor Report*, vol. 11, no. 15, pp. 23-27, November 1997.

[SPEC] The Standard Performance Evaluation Corporation, http://www.specbench.org.

[TEL95] D.M. Tullsen, S. Eggers, and H. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *in Proceedings of the 22nd International Symposium on Computer Architecture* (ISCA 95), pp. 392-403, June 1995.

[TEE+96] D.M. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," *in Proceedings of the 25th International Symposium on Computer Architecture* (ISCA 96), pp. 191-202, May 1996.

[TF70] G.S. Tjaden, and M.J. Flynn, "Detection and Parallel Execution of Independent Instructions," *IEEE Transactions on Computers*, vol. C-19, pp. 889-895, October 1970.

[Tho64] J.E. Thornton, "Parallel Operation in the Control Data 6600," *in Proceedings AFIPS Fall Joint Computer Conference*, vol. 26, no. 2, pp. 33-40, 1964.

[Tom67] R.M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 25-33, January 1967.

[Wal91] D.W. Wall, "Limits of Instruction-Level Parallelism," *in Proceedings of the 4th International Conference on Architectural Support for Programing Languages and Operating Systems* (ASPLOS 91), pp. 176-188, April 1991.

[WB96] S. Wallace, and N. Bagherzadeh, "A Scalable Register File Architecture for Dynamically Scheduled Processors," *in Proceedings of the 5th International Conference on Parallel Architectures and Compilation Techniques* (PACT 96), pp. 179-184, October 1996.

[Yan94] P. Yan-Tek Hsu, "Designing the TFP Microprocessor," *in IEEE Micro*, vol. 14, no. 2, pp. 23-33, April 1994.

[Yea96] K.C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *in IEEE Micro*, vol. 16, no. 2, pp. 28-40, April 1996.

[ZK01] V.V. Zyuban, and P.M. Kogge, "Inherently Lower-Power High-Performance Superscalar Architectures," *IEEE Transactions on Computers*, vol. 50, no. 3, pp. 268-285, March 2001.

[ZK98] V.V. Zyuban, and P.M. Kogge, "The Energy Complexity of Register Files," *in Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 305-310, August 1998.