Arquitectura NetBurst: Pentium 4

José Ignacio Gómez Pérez

1. Introducción

Tras varios años sin aportar nada nuevo al mundo de la micro-arquitectura, Intel da un gran paso hacia delante con la presentación de NetBurst. AMD estaba ganando la partida comercial, que tan bien había sabido jugar Intel hasta la fecha, con continuos aumentos en la frecuencia de sus procesadores. La barrera del GHz fue alcanzada en primer lugar por Athlon, si bien Intel no tardó en responder.

Pero parece que tenían preparada una mano ganadora. De un plumazo, el Pentium 4 ha dejado atrás al resto de los procesadores CISC, y se ha puesto a la altura de los mejores RISC incluso en el proceso en punto flotante (al menos, así lo dicen los resultados obtenidos por los SPEC2000. Tan solo el Power4, con multiprocessor-on-chip supera al nuevo procesador de Intel).

Para conseguir tamaña mejoría, los ingenieros de Intel han tenido que trabajar duro, pues el legado que sus antecesores en el puesto les dejaron, no es precisamente positivo. Un repertorio de instrucciones CISC, con tan solo 8 registros direccionables por el programador, un mayor número de accesos a memoria que en un procesador RISC, y una costosa traducción de instrucciones IA-32 a micro-operaciones (que serán ejecutadas por el núcleo RISC del Pentium 4) son los frutos de la siembra que Intel comenzó hace décadas.

Mantener la compatibilidad con las versiones anteriores, para no perder el privilegiado puesto que Intel consiguió en el mercado doméstico, supone afrontar todos esos problemas y partir en desventaja frente a otras alternativas.

A pesar de ello, los diseñadores de Intel han recogido diversas propuestas del ámbito académico, como la Trace Cache (quizás el punto estrella de la nueva arquitectura) o el Multithreading, y han intentado mitigar algunos de los problemas inherentes a toda arquitectura IA-32. A lo largo del presente trabajo, presentaremos todas esas novedades así como el resto de componentes que conforman el núcleo funcional del procesador.

La descripción del funcionamiento de un procesador es una tarea casi inacabable. Como nuestro tiempo es limitado, nos remitiremos a detallar los elementos constitutivos de la micro-arquitectura del procesador. Aspectos como el soporte para la memoria virtual (el Pentium 4 presenta la posibilidad de usar paginación, segmentación o ambas al mismo tiempo), optimizaciones de código, o protocolos de bus, no serán tratados a lo largo del trabajo.

Comencemos pues con una visión global del procesador, para continuar después con una análisis minucioso de cada uno de los componentes que éste alberga.

2. Visión Global

En la Figura 1 presentamos una visión global de la arquitectura del Pentium 4 [1]. Podemos dividirla, y así lo haremos con nuestra presentación, en cuatro grandes bloques.

Como ya hemos comentado, el elemento principal del Front-End de esta nueva arquitectura es, sin duda, la Trace Cache. Tanto ésta como el efectivo predictor de saltos y el decodificador de instrucciones IA-32, son los encargados de suministrar micro-operaciones al pipe.

Durante la etapa de ejecución, se produce el renombramiento de registros, la ejecución fuera de orden de las micro-operaciones y su posterior finalización ordenada. Veremos con detalle cómo se realizan estas operaciones, y las estructuras encargadas de llevarlas a cabo. La ejecución de las micro-operaciones tiene lugar en diversas unidades funcionales que describiremos posteriormente.

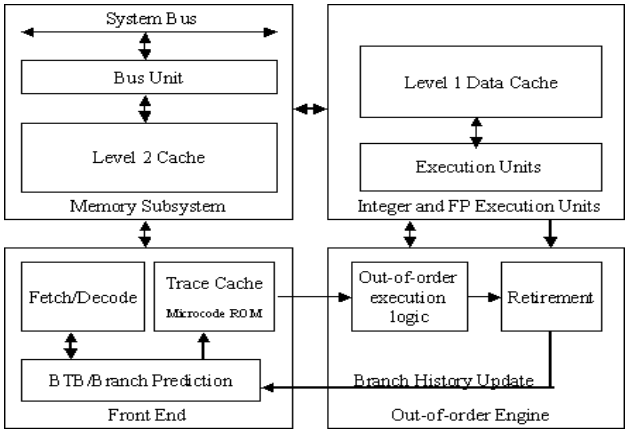


Figura 1 Arquitectura Global del Pentium 4

Los datos para la ejecución de dichas operaciones se toman de la renovada jerarquía de memoria: una cache de primer nivel de tan solo 8 Kb y muy baja latencia, junto con la tradicional Cache unificada de segundo nivel, de 256 Kb que trabaja al doble de velocidad que sus antecesoras.

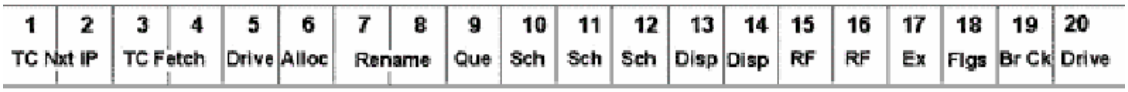


Figura 2 Etapas del Pipeline del Pentium 4

Como observamos en la **Figura 2**, los distintos bloques básicos de ejecución se distribuyen a lo largo de un pipeline realmente profundo: 20 etapas. Intel ha querido aumentar vertiginosamente la frecuencia de su procesador, y un modo de conseguirlo es alargar el pipe. Sin embargo, esta decisión conlleva numerosas contrapartidas; quizás la más importante sea la fuerte penalización que supone para dicho pipe un error en la predicción de un salto o un fallo de cache. El número de instrucciones incorrectamente ejecutadas hasta que nos percatamos del error, es enorme. Por ello, gran parte de los esfuerzos de los diseñadores se han centrado en conseguir una política de predicción de saltos efectiva, y una cache de primer nivel con muy baja tasa de fallos.

Otro problema añadido a la alta frecuencia es el escalón que ello supone de cara a la memoria. Por este motivo, la cache de primer nivel se ha visto reducida a la mitad (8 Kb frente a los 16Kb del Pentium Pro). De ese modo, y con un acceso a la memoria en forma de pipeline de tres etapas, se pretende asegurar un solo ciclo de latencia en caso de acierto. Al mismo tiempo, el bus del sistema se ha visto fuertemente mejorado: llega a un ancho de banda de 3,2Gb/s.

Siguiendo el análisis de la **Figura 2**, comentemos *grosso modo* cada una de las etapas del pipe. A lo largo del capítulo detallaremos cada una de estas explicaciones.

- Las 4 primeras etapas corresponden a la labor de la Trace cache. En las dos primeras, obtiene el puntero a la siguiente dirección. El BTB es el encargado de generar esta información. Durante las dos siguientes etapas, la Trace cache realiza el fetch de las micro-operaciones correspondientes.
- La quinta etapa es otra novedad: por primera vez se tienen en cuenta los tiempos de las conexiones. En este caso, corresponden al envío de las micro-operaciones al entorno de ejecución.
- En la sexta etapa, se procede a reservar la estructuras necesarias para la ejecución de las operaciones: entrada de la ROB, de la RAT….
- Se procede ahora al renombramiento de los registros que contendrán los operandos, y a su almacenamiento en las colas de micro-operaciones (las estaciones de reserva; etapas 8 y 9)

- Se reservan tres etapas (10, 11 y 12) para la planificación de las instrucciones: evaluación de dependencias, determinar cuáles están listas, decidir cuáles se han de ejecutar. Para ello cuenta con distintos planificadores, en función de la naturaleza de la operación.
- Llega el momento de lanzar a ejecutar operaciones. Hasta 6 pueden enviarse a las unidades funcionales en cada ciclo (aunque este número es, como veremos, muy optimista).
- Etapas 15 y 16: antes de entrar en la unidad funcional correspondiente, la micro-operación lee sus operandos del conjunto de registros (ya que los operandos no se almacenan en las estaciones de reserva).
- En la decimoséptima etapa se produce la verdadera ejecución de las operaciones. Hay múltiples unidades funcionales con diferentes latencias. Tras ello, se generan los flags correspondientes a la ejecución.
- En las dos últimas etapas se comprueba si la predicción del salto fue correcta, y se transmite dicha información al Front-End. Podemos comprobar que un error en la predicción supone un fuerte coste, pues tal situación no se conoce hasta las últimas etapas de ejecución.

Un poco a hurtadillas, Intel ha introducido en el Pentium 4 una técnica que posiblemente sea, junto al multiprocesador en un chip, la tendencia a seguir en los próximos años. Nos referimos al *Multithreading,* que consiste en lanzar varias threads de ejecución de un modo simultáneo para garantizar un flujo continuo de operaciones en el pipe. La documentación al respecto es más escasa y confusa, si cabe, que para el resto de los componentes del la nueva arquitectura. Intentaremos subrayar los aspectos más significativos del multithreading en las secciones finales del trabajo.

Para terminan con esta visión preliminar, señalaremos también la introducción de l44 instrucciones SSE2 (nueva extensión de las ya conocidas MMX) al repertorio IA-32. Se continúa así ampliando la gama de posibilidades en el cálculo SIMD (Single Instruction Multiple Data) que tan buen resultado está dando en las aplicaciones multimedia. La mayoría de las nuevas instrucciones no son más que versiones en 128 bits de las antiguas. En el apartado dedicado a la ejecución, dedicaremos un apartado a la filosofía SIMD, y a los resultados conseguidos con su incorporación al repertorio de instrucciones.

3. Front-End

Como ya indicamos , tres elementos fundamentales componen el Front-End del Pentium 4: el decodificador de instrucciones, el predictor de saltos y la trace cache. La Figura 3 muestra otro bloque de especial importancia: la TLB de instrucciones, que abastece al decodificador de instrucciones. A lo largo de esta sección intentaremos presentar con detalle cada una de las citadas estructuras, si bien la documentación que Intel ofrece sobre su último producto no es abundante, ni lo es la literatura a consultar.

3.1 Decodificador de instrucciones

Para mantener la compatibilidad con sus anteriores arquitecturas, y no arriesgarse a perder la privilegiada posición que ocupaba en el mercado doméstico, Intel ha ido arrastrando su repertorio de instrucciones IA-32. Es éste un repertorio CISC, de longitud de instrucción variable y, en muchos casos, de compleja ejecución. El tiempo ha ido dejando a un lado el control microprogramado en el ámbito de los superescalares, e Intel optó por dotar de un núcleo RISC a sus procesadores.

Pero esto suponía realizar una costosa traducción entre las antiguas instrucciones IA-32 y las micro-operaciones que el núcleo era capaz de ejecutar. En la mayor parte de las ocasiones, dicha traducción resulta ser 1:1, pero en algunos casos una sola instrucción IA-32 da lugar a más de 4 micro-operaciones.

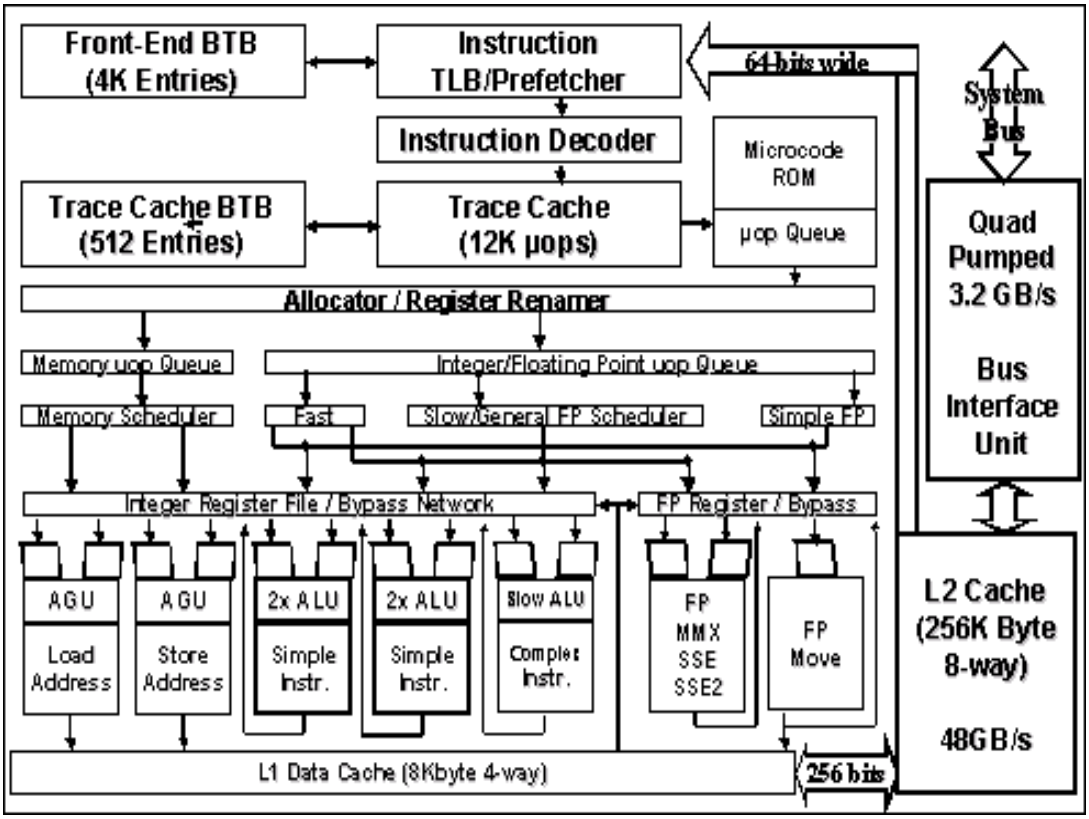


Figura 3 Microarquitectura del Pentium 4

Para esta misión, todos los procesadores Intel más modernos disponen de un decodificador capaz de traducir una única instrucción IA-32 por ciclo[1]. Dicho decodificador recibe 64-bits de la cache de segundo nivel, que va almacenando en un buffer hasta que reconoce una instrucción completa. Si dicha instrucción no supone más de cuatro micro-operaciones, realizará la traducción y seguirá recibiendo bits de la cache.

Para las instrucciones más complejas, como ciertas operaciones sobre cadenas, se dispone de una ROM que guarda las traducciones de dichas instrucciones. Estas micro-operaciones así obtenidas no pasan por la Trace-Cache, sino que las introduce en la misma cola de micro-operaciones utilizada por la Trace Cache para comunicarse con el corazón del procesador.

3.1.1 ITLB

Sin embargo, las instrucciones IA-32 han de almacenarse en la Cache de segundo nivel, o incluso en memoria principal. Para acceder a ellas, se debe traducir la dirección virtual utilizada por el procesador a la dirección fisicas que ocupan las instrucciones en memoria.

Este es el papel de la TLB de instrucciones, que además permite realizar comprobaciones de seguridad a nivel de página. La documentación del Pentium 4 no incluye el tamaño ni la organización de dicho TLB (creo que ya hemos indicado que la documentación al respecto del Pentium 4 es escasa y deficiente).

---

[1] Esta es la única información encontrada al respecto para el Pentium 4. Sin embargo, la arquitectura P6 disponía de tres decodificadores (dos de ellos sólo para instrucciones del tipo registro-registro), que supuestamente podían llegar a decodificar hasta 3 instrucciones, generando un máximo de 6 micro-operaciones. Ignoramos si el Pentium 4 ha simplificado dicha decodificación en pos de su Trace Cache.

Haciendo uso de la instrucción IA-32 CPUID[2] hemos llegado a la conclusión de que tanto el TLB de datos como el de instrucciones disponen de 64 entradas, en las que almacenan las traducciones más recientes.

3.2 Trace Cache

Llegamos a uno de los puntos más interesantes de esta nueva arquitectura. Tras años arrastrando la penalización que suponía traducir, durante el camino crítico, las instrucciones IA-32 a micro-operaciones, Intel ha encontrado una solución que parece dar buenos resultados.

El concepto de Trace Cache fue presentado por primera vez en 1996 [2]. La idea básica es la de capturar el comportamiento dinámico de las secuencias de instrucciones, almacenando trazas de instrucciones en lugar de bloques contiguos (contiguos tras una ordenación estática. Ver Figura 4).

La intención original de los autores del artículo, era dar una respuesta a la demanda de un flujo de instrucciones suficiente ante la cada vez mayor ventana de instrucciones. Efectivamente, la tradicional cache de instrucciones, junto con los mecanismos de predicción de saltos, eran incapaces de predecir más de una rama en un solo ciclo. Por ese motivo, tan solo un bloque básico podía ser suministrado al pipe; totalmente insuficiente si tenemos en cuenta que muchos procesadores pueden ejecutar hasta 6 instrucciones en cada ciclo (y que en un bloque básico encontramos, de media, 4 instrucciones que no sean un salto).

La misión de la Trace cache es encontrar esos bloques de instrucciones no contiguos en su disposición original, y almacenarlos uno tras otro para su posterior envío al pipe de ejecución. No vamos a entrar en mayor detalle en cuanto a la arquitectura interna de las Traces Caches. Sí diremos que, además de las trazas de instrucciones, contienen cierta información de control (como las predicciones realizadas para elaborar la traza), y el modo de continuar la traza al acabar la línea actual.

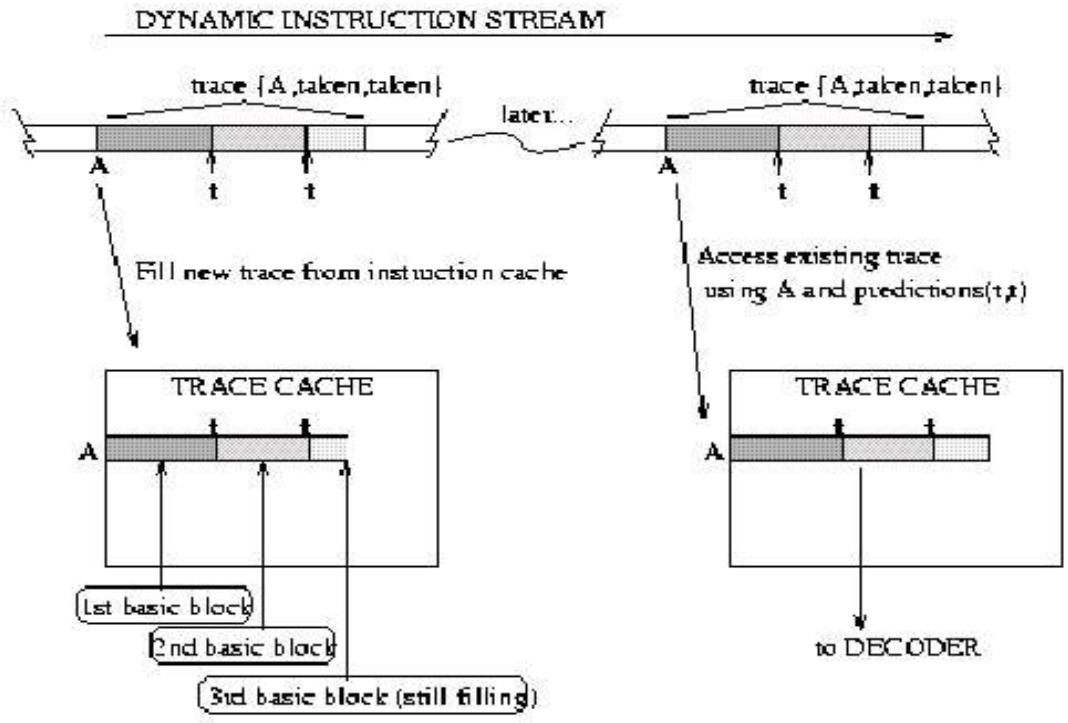


Figura 4 Ejemplo de funcionamiento de la Trace Cache

[2] Gracias Luis…

En muchos esquemas (como los propuestos en [3], la Trace Cache se usa en paralelo a la cache de instrucciones. Si la dirección accedida está en la Trace Cache, y las predicciones realizadas sobre esa dirección coinciden con las indicadas en la entrada correspondiente de la Trace Cache, su traza será la utilizada, olvidándonos del acceso a la cache de instrucciones.

En el caso del Pentium 4, sólo disponemos de una estructura, la propia Trace Cache. Si se produce un fallo en el acceso, se procederá a la búsqueda y decodificación de las instrucciones en la cache de segundo nivel. La Trace Cache dispone de capacidad suficiente para almacenar 12K micro-operaciones - que eran de 118 bits en la arquitectura P6. No hemos podido saber si este dato ha variado para NetBurst – organizadas en 8 conjuntos, con 6 micro-operaciones en cada línea. De estas 6 micro-operaciones, sólo 3 podrán ser suministradas en cada ciclo a la lógica de ejecución del procesador.

El uso de la Trace Cache por parte de Intel no responde sólo a la necesidad de proporcionar un mayor flujo de instrucciones al entorno de ejecución; es el medio de evitar la decodificación reiterada de instrucciones, ya que la Trace Cache almacena únicamente micro-operaciones. De este modo, en caso de fallo en la predicción de un salto, o de la reejecución de cierta parte del código, las instrucciones envueltas en la ejecución no deben ser decodificadas nuevamente, pues ya se encuentran en la Trace Cache (y además, en el supuesto orden de ejecución).

La Trace Cache incluye su propio predictor de saltos, más pequeño que el BTB del Front-End pues su único objetivo es predecir el comportamiento de las instrucciones de salto presentes en la Trace Cache en un momento concreto. Dispone de 512 entradas y de una pila de direcciones de retorno (para las llamadas a funciones y procedimientos) de 16 entradas. Veremos con más detalle cómo se realiza la predicción de los saltos en el siguiente apartado, cuando analicemos el predictor de saltos global.

3.3 Predictor de saltos

Ésta es quizás la fase crítica del Front-End de cualquier superescalar. Un mecanismo de prefetching de instrucciones accede a la cache de segundo nivel para llevar al decodificador las instrucciones IA-32 que se han predicho como las próximas en la ejecución.

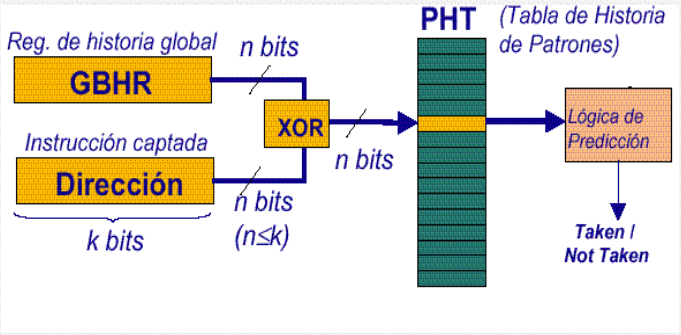
Para guiar dicho prefetching, el Pentium 4 consta de un complejo mecanismo de predicción de saltos, que combina la predicción estática con la dinámica. Las mejoras introducidas en los algoritmos de predicción han supuesto una reducción en la tasa de fallos cercana de 1/3 frente al predictor de la arquitectura P6.

Debido al gran número de etapas del pipeline, la correcta predicción de saltos es tremendamente importante en el Pentium 4. Como ya mostramos anteriormente, la dirección efectiva del salto no se conoce hasta las dos últimas etapas del pipe, lo que supone un gran cantidad de trabajo inútil a rehacer.

El sistema de predicción de la arquitectura NetBurst predice todos los saltos cercanos, pero no otro tipo de saltos tales como irets o interrupciones software.

3.3.1 Predicción dinámica

La predicción de saltos dinámica está basada en una tabla de historia de saltos y la correspondiente tabla de direcciones de los saltos. La arquitectura P6 [15] supuso el paso del predictor bimodal, siguiendo el algoritmo de Smith, al método presentado por Yeh y Patt [4], con cuatro bits de historia. De nuevo la documentación al respecto escasea, pero parece que el predictor del Pentium 4 está basado en dos niveles de historia, accediendo a la tabla de historia de saltos por compartición de índice (el método llamado *gshare*. Ver Figura 5) El tamaño del BTB ha pasado de 512 entradas en la arquitectura P6 a las 4K entradas de las que dispone en el Pentium 4, organizadas como una cache asociativa de 4 vías.



El registro de historia global se combina con la dirección de la instrucción de salto para obtener un índice con el que acceder a la PHT. Así obtendremos nuestro patrón de saltos, y con él la predicción del salto actual. Este tipo de predictor global presenta muy buenos resultados en los programas que operan sobre enteros, debido a la interrelación de los saltos.

Figura 5 Funcionamiento del método gshare

3.3.2 Predicción estática

Si no se encuentra ninguna entrada en el BTB correspondiente con el PC de la actual instrucción de saltos, el predictor estático realiza una predicción basada en la dirección del salto (conocida tras la decodificación).

Si el salto es hacia atrás (desplazamiento negativo; como en los bucles), el predictor considerará el salto como tomado. En caso contrario, el salto se considerará como no tomado. Conociendo este comportamiento, el código puede adaptarse para respetar al máximo esta política; esta adaptación no puede ser hecha automáticamente por un compilador (salvo tras una fase de profiling) pues implica el conocimiento de la semántica del código para reconocer las destinos más probables en cada salto.

3.3.3 Pila de direcciones de retorno

Todos los retornos de una función son saltos tomados, pero como éstos pueden venir desde distintas direcciones, no podemos utilizar un predicción estática. Para estos casos, el Pentium 4 dispone de una pila de direcciones de retorno (Return Stack) que predice las direcciones de vuelta para una serie de llamadas a funciones.

Incluso en caso de predecir correctamente la dirección y el destino de un salto, todos los saltos tomados merman en cierta medida el paralelismo inherente del código para los procesadores tradicionales. Esto es debido al ancho de banda desperdiciado en la decodificación que siguen al salto y que preceden al destino si estos no se encuentran al final y al principio de sus respectivas líneas de cache. En cierta medida esta inevitable merma se ve paliada por la Trace Cache, que, en el caso ideal, sólo trae una vez las instrucciones desde la cache.

Así mismo, el Pentium 4 ofrece la posibilidad de anotar, a nivel software, las instrucciones de salto. Esta información se añade como prefijo a las instrucciones, y es ignorada en los procesadores anteriores al Pentium 4. Estas marcas son utilizadas para ayudar al predictor de saltos en su labor, facilitando la generación de las trazas. Esta información tiene mayor prioridad que el predictor estático que por defecto utiliza el Pentium 4.

4. Fase de ejecución

La nueva arquitectura de Intel introduce, también en esta, fase ciertas novedades, algunas de ellas de cierta magnitud (como la renovada política de renombramiento, con la creación de un conjunto de registros independiente). Con todas estas modificaciones, Intel defiende la posibilidad, a simple vista irreal, de mantener 126 micro-operaciones, 48 loads y 24 stores al mismo tiempo en el pipe.

En dos pinceladas, podríamos definir la ejecución fuera de orden del Pentium 4 diciendo que dispone de dos tablas de renombramiento de 8 entradas direccionada con el índice del registro lógico con un Register

File de 128 registros independiente de la ROB de 126 entradas. El fetch de los operandos se realiza en el momento del *issue* (siguiendo la nomenclatura de D. Sima [5]), momento en el que las micro-operaciones salen de una de las dos colas de micro-operaciones (una para las operaciones de memoria, y otra para el resto) para dirigirse a uno de los cuatro puertos de ejecución que dan acceso a las 7 unidades funcionales disponibles.

A lo largo de esta sección, seguiremos el camino de ejecución de una micro-operación (EAX ← EBX + ECX) ,desde que sale del Front-End hasta que realiza la finalización. La presentación de cada una de las estructuras visitadas, se irá haciendo a lo largo del viaje. Dicho trayecto se debería realizar en paralelo con otras dos micro-operaciones, pero el comportamiento global es exactamente el mismo que el presentado aquí.

4.1 Asignación de recursos

En la primera fase de la vida de nuestra micro-operación, se le asignará espacio en las diversas estructuras de almacenamiento presentes en el interior del corazón del procesador. Si alguno de los recursos necesitados, como un registro, no estuviera disponible en el momento de la petición, esta fase quedará bloqueada hasta que se libere dicho recurso.

En primer lugar, la micro-operación se sitúa en una de las 126 entradas del Buffer de Reordenamiento (ROB), que se encarga de realizar el seguimiento de cada instrucción hasta su finalización, garantizando que ésta se produce en el orden correcto (esto es, la ejecución puede producirse en desorden, pero las instrucciones finalizadas permanecen en el ROB hasta que todas las instrucciones previas hayan terminado su ejecución).

La asignación de entradas en el ROB se realiza de modo secuencial. El ROB dispone de dos punteros: uno a la instrucción más antigua, y otro a la siguiente posición libre. De este modo, el ROB funciona como un buffer circular y nuestra micro-operación ocupará la primera posición libre. En la entrada del ROB, dispondremos de información sobre el status de la micro-operación (en espera de operandos, en ejecución ,ejecutada…), sobre el registro en que se debe escribir (tanto del renombrado como del registro lógico), y el PC de la instrucción (para las comprobaciones en las predicciones de saltos).

Así mismo, se le asignará un registro de entre los 128 disponibles para las instrucciones con enteros (hay otros 128 para las operaciones de punto flotante) para almacenar el resultado de la adición. En el Pentium 4 no existen, explícitamente, registros arquitectónicos (aunque sigan presentándose los mismos 8 registros lógicos al programador), así que cualquier registro disponible es válido. Si nuestra instrucción fuera un load o un store, se reservaría al mismo tiempo un entrada en el correspondiente buffer de loads/stores (48 entradas en el Load Buffer y 24 en el Store Buffer).

Finalmente, una entrada en una de las dos colas de micro-operaciones se destina a albergar la micro-operación, En nuestro caso concreto, entrará en la cola de micro-operaciones que no van a memoria (estas colas de micro-operaciones, se corresponden con el concepto de Estación de Reserva presentado en [6], y creado por R. Tomasulo [7]).

4.2 Renombramiento de registros

Para paliar el reducido número de registros lógicos, el Pentium 4 presenta un agresivo renombramiento de registros (Register File de 128 entradas), evitando así los conflictos por dependencias de datos. En un instante determinado, puede haber múltiples instancias de un mismo registro lógico (por ejemplo, EAX). Para llevar un control sobre las asignaciones, se dispone de dos tablas denominadas Register Alias Table (RAT).

La entrada correspondiente al registro EAX de la Front-End RAT quedará apuntando al registro que hayamos asignado a nuestra instrucción. De este modo, cualquier instrucción posterior que quiera leer de EAX sólo tendrá que consultar esa misma entrada de la RAT para saber de dónde debe coger su operando.

Al mismo tiempo, las entradas de los registros operandos se leen (EBX y ECX), obteniendo las entradas de los registros que albergarán los valores correctos en el momento preciso.

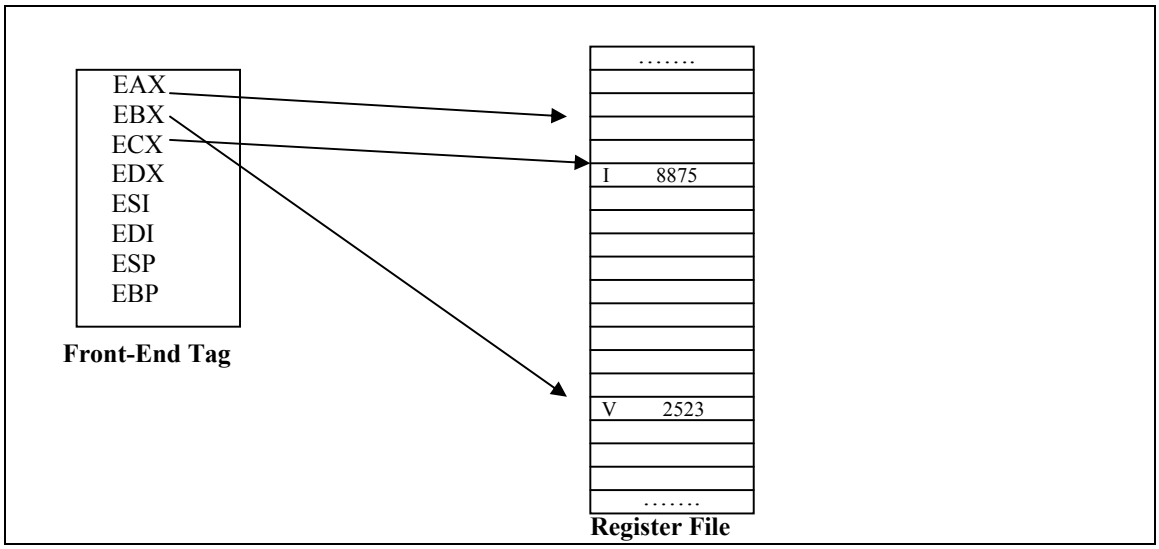


Figura 6 Situación tras renombrar la instrucción

En la Figura 6 observamos la situación de la Front-End RAT y de los registros tras el renombramiento. La entrada EAX apunta al registro que será destino de la instrucción, cuyo índice (podría ser el 3 en la figura) se pasa junto con el resto de la micro-operación, a la cola de instrucciones. El valor de EBX se encuentra en un registro válido (quiere decir que la instrucción que escribió en él antes de nuestra instrucción, ya terminó). A pesar de ello, no cogemos el dato (2523), sino el índice del registro; lo haremos en el momento de lanzar a ejecutar la suma. El valor del registro apuntado por ECX no es el correcto, y nos llevamos la referencia del registro (digamos que 7) para poder saber cuándo está preparado.

Si tras nuestra micro-operación llegara otra del estilo: EDX← EBX - EAX, se encontraría con que el dato de EBX es válido (y en el mismo registro que para nuestra micro-operación), y que el de EAX ya no es válido. Cogerá el índice del registro apuntado (hemos asumido que será 3), y se llevará la instrucciones a la cola.

Antes de pasar a la ejecución de nuestra instrucción, no está de más comentar que el proceso de renombramiento que acabamos de describir es muy diferente al de la anterior arquitectura P6. En el Pentium Pro y sucesores, sólo se disponía de una tabla RAT (después veremos el uso que en el Pentium 4 tiene la segunda de las tablas), y no existía un conjunto de registros independiente. En su lugar, los datos de las operaciones eran almacenados directamente en el ROB (de 40 entradas), actualizando en la fase de finalización los registros arquitectónicos (que aquí sí existían explícitamente). En la Figura 7 vemos las diferencias descritas en este párrafo.

Tras el renombramiento, nuestra micro-operación está a la espera de tener todos sus operandos preparados. Esta espera se produce en la cola de micro-operaciones. Sólo falta el dato del registro ECX (ahora renombrado a 7) para poder comenzar nuestra ejecución. Cuando una instrucción termina su ejecución, su resultado se comunica al resto por medio del bus común. Además de llevar el dato del resultado, el bus transmite el índice del registro donde se debe almacenar. De este modo, cuando nuestra instrucción vea en el bus que alguien va a escribir en el registro 7, sabrá que su dato ha sido producido, y que puede ser ejecutada.

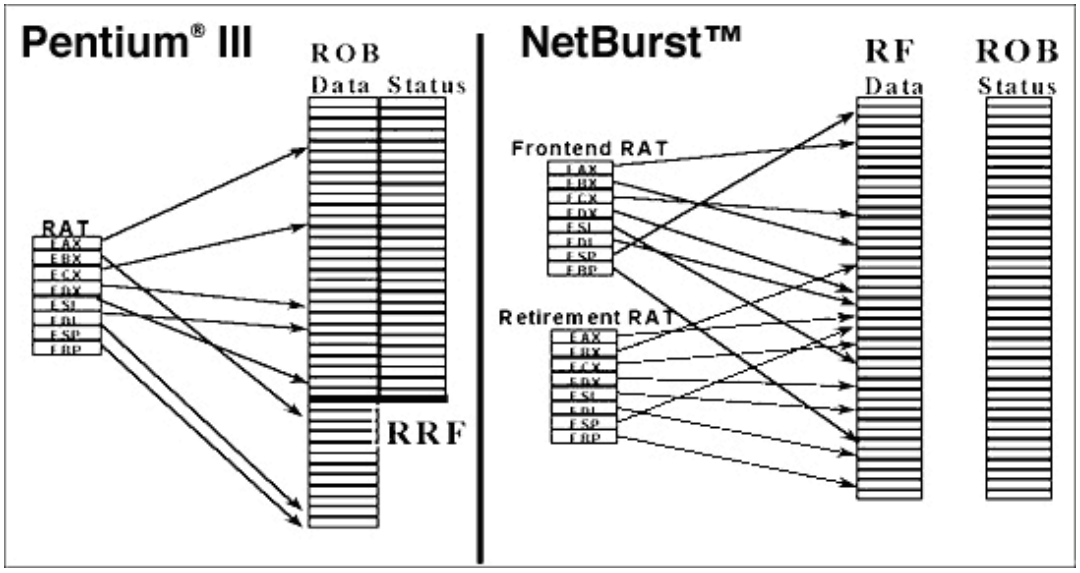


Figura 7 Renombramiento de registros en el Pentium III y en el Pentium 4

4.3 Planificación y lanzamiento a ejecución

En ese momento, todos los flags que indican la validez de los operandos estarán activados, y la entrada correspondiente del ROB indicará que la instrucción está lista para ejecutar. En este momento entran en acción los planificadores. Las micro-operaciones están en la cola en estricto orden de llegada (cola FIFO). Intel prefiere no aclarar el complejo conjunto de reglas seguido para determinar cuál es la siguiente micro-operación a ejecutar, y sólo dicen que está basado en la "edad" de las operaciones.

Existen varios planificadores en función del tipo de instrucción (asumimos que estamos en la cola de operaciones que no van a memoria). Dichos planificadores deciden cuándo las micro-operaciones del tipo adecuado están dispuestas para la ejecución, en función de la disponibilidad de sus operandos y de las unidades funcionales que necesitarán.

Estos planificadores están ligados a cuatro puertos diferentes, como se refleja en la Figura 8. Los dos primeros puertos pueden llegar a lanzar dos instrucciones en un solo ciclo. Múltiples planificadores comparten estos dos puertos, pero tan solo los planificadores de operaciones "fast ALU" pueden funcionar al doble de la frecuencia del procesador. Son los puertos los encargados de decidir qué tipo de operación debe ejecutarse en caso de que haya más de un planificador informando de la disponibilidad de instrucciones para la ejecución.

Con esta estructura, hasta un máximo de 6 micro-operaciones (4 fast Alu, un Load y un Store) pueden mandarse a ejecución simultáneamente. A continuación presentaremos los aspectos fundamentales de cada una de las unidades funcionales, y comentaremos el comportamiento de los loads y los stores.

4.4 Unidades funcionales: enteros y punto flotante

También en este punto se han realizado diversas mejoras respecto a las anteriores arquitecturas. Una de ellas es la disposición del conjunto de registros. Al haberse separado de la ROB, el nuevo emplazamiento se ha escogido acorde a la política de fetch de operandos. Ya hemos indicado que existen dos conjuntos de registros (uno para operaciones sobre enteros y otro para las de punto flotante/SSE) de 128 registros cada uno. Estos registros están situados entre el planificador y las unidades funcionales, de tal manera que una vez se ha decidido qué instrucción va a ejecutarse a continuación, ésta puede leer sus operandos del cercano banco de registros. Además, cada conjunto de registros cuenta con la lógica necesaria para realizar el bypass de los resultados recién calculados, evitando consultas innecesarias a los registros.

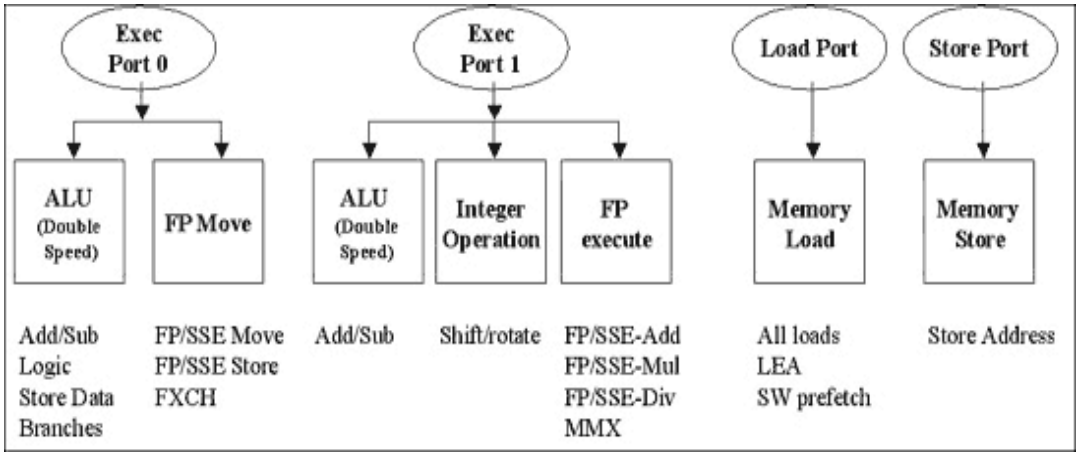


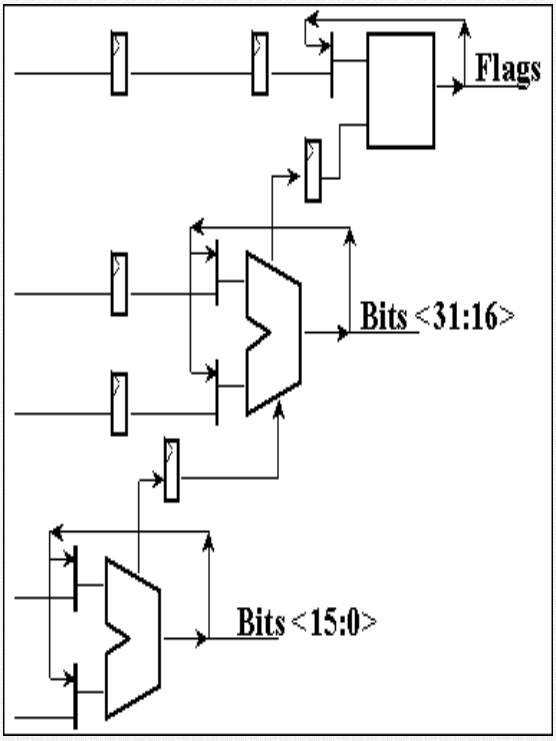
Figura 8 Puertos de ejecución y unidades funcionales

4.4.1 Operaciones sobre enteros

Como vemos en la Figura 8 disponemos de hasta tres unidades funcionales para operaciones con enteros (aunque sólo dos de ellas accesibles al mismo tiempo). Las mayores modificaciones se han introducido en las unidades etiquetadas "ALU (Double Speed)". Intel ha querido acelerar al máximo las operaciones más frecuentemente utilizadas. Para ello, ha reducido al máximo la carga de estas unidades, dejando únicamente el hardware esencial para la ejecución de dichas operaciones. Multiplicaciones, desplazamientos, generación de flags… son algunos ejemplos de operaciones que no se llevan a cabo en estas unidades.

En estas unidades, cada operación tarda un ciclo y medio en terminar su ejecución. Como están diseñadas con un pipe de tres etapas al doble de la velocidad del resto del procesador (ver

Figura 9), la latencia real de las instrucciones en caso de utilización continua de la unidad, se reduce a medio ciclo.



Cada operación se realiza en una secuencia de tres semi-ciclos de reloj. En el primer semi-ciclo, se calculan los 16 bits de menor peso; en ese momento, una segunda operación puede comenzar aprovechando el resultado de la primera.

En el siguiente semi-ciclo, se procesan los otros 16 bits, usando el carry generado por la fase anterior. Estos 16 bits de mayor peso estarán de este modo disponibles para la siguiente operación justo en el momento preciso.

En el último de los tres semi-ciclos se calculan los flags de la suma. Esto es lo que se conoce como "staggered add".

Figura 9 ALU de doble velocidad: staggered add

Para el resto de operaciones sobre enteros, se utiliza la unidad funcional del puerto 1. Las latencias de estas operaciones dependen de su naturaleza: 4 ciclos para desplazamientos, 14 para una multiplicación, 60 para una división…

4.4.2 Operaciones sobre punto flotante y SSE

Las dos unidades de punto flotante son las encargadas de ejecutar las instrucciones MMX, SSE, SSE2 y por supuesto las tradiciones operaciones de punto flotante. La mayoría de estas operaciones trabajan con operandos de 64 o 128 bits. Los 128 registros destinados a albergar operandos en punto flotante son, por tanto de 128 bits, como también lo son los puertos de las unidades funcionales destinadas a estas operaciones. De este modo, se puede comenzar una operación en cada ciclo de reloj.

En las primeras fases del diseño del Pentium 4 se disponía de dos unidades funcionales completas para punto flotante. Sin embargo, se comprobó que las ganancias en rendimiento no eran significativas, y se prefirió economizar especializando cada una de las unidades. El puerto de ejecución 0 está destinado a operaciones de movimiento entre registros de 128 bits y escrituras a memoria. El puerto 1 sí incluye una unidad funcional completa, y está dedicado a la ejecución de las operaciones habituales: sumas, multiplicaciones, divisiones y MMX.

A diferencia de la unidad funcional destinada a las operaciones con enteros, esta completa unidad funcional en punto flotante no está segmentada. Se ha preferido mantener la sencillez con una sola etapa, para conseguir una buena velocidad de ejecución con mucho menos coste. El Pentium 4 puede ejecutar una suma en punto flotante en un solo ciclo de reloj (y en medio ciclo si los operandos son de precisión simple), por lo que puede realizar una suma SSE de operandos de 128-bits en dos ciclos. También es capaz de realizar una multiplicación cada dos ciclos de reloj. Estas cifras significan un rendimiento máximo de 6 GFLOPS para operaciones en precisión simple y de 3 GFLOPS para doble precisión, a una frecuencia de 1.5 GHz.

4.4.3 Operaciones SIMD: MMX, SSE y SSE2

Antes de terminar con la fase de ejecución, presentaremos los conceptos de la ejecución SIMD y el conjunto de instrucciones diseñado por Intel al efecto. SIMD son las siglas de “Single Input Multiple Data” que explican por sí solas la naturaleza de esta técnica: procesamos en paralelo varios datos, aplicándoles la misma operación (para información más detallada sobre su implementación en el Pentium 4 ver [8,9]

En la Figura 10 mostramos un cálculo SIMD típico. Dos conjuntos de cuatro operandos son sumados en paralelo, almacenándose el resultado en una estructura similar a la de los operandos.

Las operaciones SIMD se introdujeron en el repertorio IA-32 con la tecnología MMX. Esta tecnología permitía que se realizasen operaciones sobre 64 bits empaquetados en bytes, palabras o dobles palabras. El Pentium III extendió el modelo con la introducción de las Streaming SIMD Extenseions (SSE). Los registros pasaron a ser de 128 bits, y se podían realizar operaciones sobre operandos que contenían cuatro elementos en punto-flotante de precisión simple.

El Pentium 4 ha aumentado aún más la funcionalidad de las operaciones SIMD con las llamadas SSE2. Estas operaciones son capaces de trabajar con elementos en punto flotante y doble precisón y con enteros empaquetados en 128 bits. Dispone de 144 nuevas instrucciones que pueden operar sobre dos datos empaquetados en doble precisión, o sobre enteros de 16 bytes, 8 palabras, 4 doble palabras y 2 quadwords.

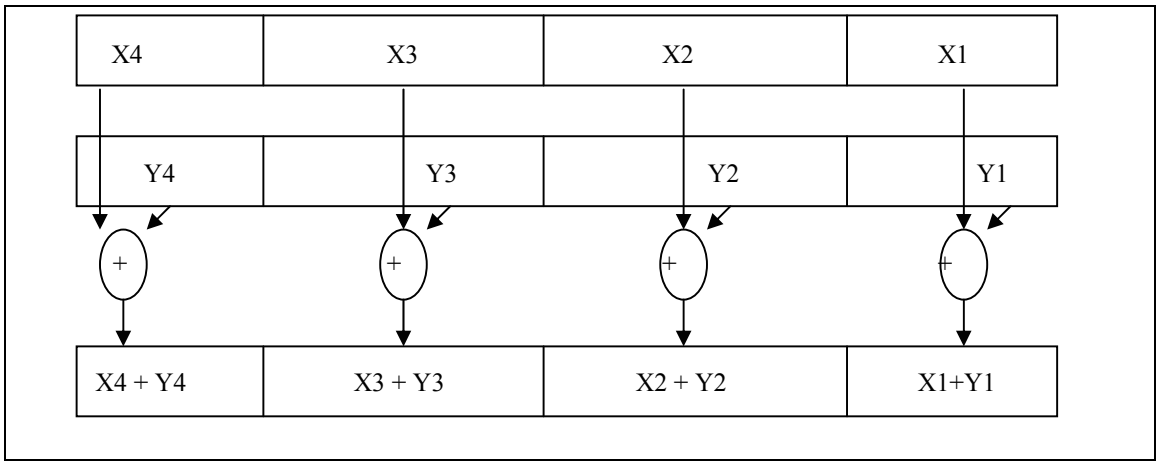


Figura 10 Suma SIMD

El repertorio SIMD mejora cuantiosamente el rendimiento en aplicaciones multimedia, como el procesamiento de gráficos 3D, reconocimiento del habla…. y cualquier otra aplicación que tenga un gran paralelismo inherente, que se traduce en patrones de acceso a memoria muy regulares, realizando las mismas operaciones sobre los datos accedidos.

4.5 Finalización de instrucciones

Nuestra ya casi olvidada micro-operación (EAX← EBX + ECX) abandonó en su turno la cola de micro-operaciones, y fue ejecutada por una de los dos ALU de doble velocidad. Su resultado, junto con el índice del registro en el que escribirá su resultado (recordemos, era el 3), se transmiten por el bus común a todas las entradas de la cola de micro-operaciones que estén interesadas (como nuestra siguiente micro-operación, que leía el contenido de EAX), y finalmente lleva dicha información al ROB.

Allí continúa presente la entrada que al principio del proceso reservamos para nuestra instrucción. Cuando recibe la noticia acerca de la ejecución de la suma, pone al día su status: ya está preparada para finalizar. Esto significa que aún tendrá que esperar a que todas las micro-operaciones que estaban delante de ella finalicen sus cálculos, y en ese momento, podrá ella escribir su resultado en los registros.

Cuando nuestra entrada se encuentre en la cabeza del ROB, será escogida junto con otras dos (si esto es posible. Quiere esto decir que el número máximo de micro-opraciones finalizadas por ciclo es de 3) para escribir sus resultados. Aquí entra en juego la segunda de las RAT: la “Retirement RAT”. Nuestra instrucción debía escribir en el registro arquitectónico EAX, y que éste fue renombrado al registro físico 3. En este momento, se hará apuntar la entrada correspondiente a EAX de la Retirement RAT al registro número 3; así mismo, se escribirá en dicho registro el resultado de nuestra suma.

¿Por qué mantener esta segunda tabla? Como ya no disponemos de registros arquitectónicos separados del resto, debe haber algún medio de indicar cuáles son en cada momento los registros que funcionan como tales. En caso de error en la predicción de un salto, hay que recuperar el estado de la máquina anterior al fallo en la predicción, y esto incluye el contenido de los registros. Con la segunda RAT garantizamos que nunca perderemos la información que ya ha modificado el contexto de la ejecución.

Otra función del ROB es ayudar a determinar si un salto fue correctamente predicho; cuando una operación de salto finaliza su ejecución (esto es, el cálculo de su dirección de destino), vuelve a actualizar

su entrada de la ROB. En ese momento, se comprueba que la verdadera dirección de destino (recién calculada) coincida con la que realmente se ha tomado. ¿Cómo? Es realmente sencillo si nos percatamos de que la siguiente entrada del ROB tendrá almacenado su PC, que es exactamente la dirección predicha para el salto. Si ambos coinciden, todo es correcto y se continúa normalmente. Si no coinciden, la predicción fue errónea; se informa de la situación al Front-End, y se vacía todo el pipe de ejecución: el ROB, las colas de micro-operaciones y los registros no apuntados por la Retirement RAT.

5. Acceso a memoria

Hasta este momento, hemos pasado un poco por encima de las operaciones de lectura y escritura a memoria. Sí hemos dicho que cuentan con una cola específica para ellas, y que dos unidades funcionales están destinadas al cálculo de las direcciones de acceso.

Un detalle importante es el hecho de que cada "store" es dividido en dos micro-operaciones: una que generará la dirección de escritura, y otra que calculará el dato a escribir. A partir de la división, ambas serán tratadas como micro-operaciones independientes (de hecho, una irá a la cola de instrucciones de memoria y la otra no), pero ambas escribirán en la misma entrada del store buffer (cada entrada de dicho buffer tendrá espacio para la dirección de escritura y para el dato a escribir). De ese modo, sabremos cuándo está preparada la escritura, y simplemente esperaremos a que llegue su turno (esto es, que todos las lecturas y escrituras previas se hayan producido).

Las instrucciones sobre memoria son bastante independientes respecto al resto de las operaciones. El momento en que se realice una escritura en memoria, sólo influye en las lecturas que se quieran realizar antes o después de la escritura. Las lecturas sí tienen un nexo de unión: escriben el dato leído en algún registro físico, por lo que deben respetar el orden lexicográfico del código original.

Esta independencia parcial queda plasmada con la introducción de una nueva estructura: el Buffer de ordenamiento de memoria (el MOB, que no aparece en la Figura 3 y debería situarse entre las dos AGU y la cache. Pocas veces se referencia en la documentación oficial de Intel, quien nunca ha facilitado su tamaño ni su estructura exacta). Del mismo modo que cualquier otra operación reserva una entrada en el ROB en las primeras fases de su recorrido, un store y un load deben reservar su entrada en el MOB (las lecturas también deben reservar su entrada en el ROB). Es sólo aventurar, pues nada de todo esto queda claro en la literatura existente, pero es seguro que la lectura a memoria se realizará antes de que la micro-operación llegue a la cabeza del ROB (de hecho se hará tan pronto como lo autorice el controlador del MOB). El dato leído se almacenará en el "Load buffer" correspondiente, y la actualización del registro destino esperará hasta obtener la autorización del ROB.

Las unidades de generación de direcciones (AGU) combinan todas las posibles direcciones el formato x86 (segmento, base, índice e inmediato) en un solo ciclo. La dirección resultante (aún una dirección virtual) se coloca en la entrada correspondiente del MOB, y se espera el momento de realizar el acceso a la cache. Cuando un store conoce tanto el dato que va a escribir como la dirección, está preparado para finalizar, que en su caso significa escribir en la cache. Un load está preparado para acceder a memoria tan pronto como conoce su dirección, pero no puede hacerlo hasta que todos las escrituras anteriores hayan finalizado (en teoría. Después veremos que la técnica denominada "Store forwarding" permite eliminar dichas esperas).

5.1 La jerarquía de memoria

La micro-arquitectura NetBurst soporta hasta tres niveles de memoria cache on-chip. Las implementaciones habituales del Pentium 4 sólo incorporan dos, dejándose el tercer nivel para servidores con grandes cargas de trabajo.

Estos dos niveles se distribuyen como sigue: una cache de datos y la trace cache (antigua cache de instrucciones) se sitúan próximas al núcleo de ejecución. Todos los demás niveles en la jerarquía serán unificados (comparten datos e instrucciones). En la Tabla 1 resumimos las parámetros principales de las

caches habituales en el Pentium 4. Todas ellas utilizan un algoritmo LRU (pseudo-LRU en realidad) para el reemplazamiento de bloques.

Nivel	Capacidad	Asociatividad (ways)	Longitud de Línea (bytes)	Latencia de acceso, Enteros/FP (ciclos)	Política de escritura
Primero(Datos)	8 KB	4	64	2/6	Write Through
Trace Cache	12K uops	8	N/A	N/A	N/A
Segundo	256 KB	8	128	7/7	Write Back

Tabla 1 Parámetros de los distintos niveles de Cache

De la Trace Cache ya hemos hablado con anterioridad (ver apartado 3.2), así que nos centraremos ahora en los aspectos fundamentales de la cache de datos del primer nivel, y la unificada del segundo.

5.1.1 Cache de datos (Nivel 1)

La latencia de las operaciones de lectura en memoria se convierte, para el Pentium 4, en un aspecto aún más determinante que para anteriores procesadores superescalares. Entre otros motivos, el reducido número de registros disponibles para el programador, provoca que los programas IA-32 contengan una gran cantidad de referencias a memoria, que mermarán el rendimiento global si no son tratadas con eficacia.

Intel ha optado por disponer una cache de datos de primer nivel de muy baja latencia, y por tanto, de tamaño reducido, complementada con una cache de segundo nivel mucho mayor y con un gran ancho de banda. Con estas especificaciones, se ha logrado una latencia de 2 ciclos en las lecturas de enteros y 6 ciclos para la lectura de un dato de punto flotante (o SSE).

Tanto esta cache de primer nivel como la unificada del segundo son no bloqueantes. Eso implica que, tras un primer fallo en el acceso, se pueden seguir realizando nuevas lecturas o escrituras; este detalle es fundamental si se tiene en cuenta la gran cantidad de lecturas y escrituras que puede haber en proceso simultáneamente en el Pentium 4. Aún así es necesario fijar un límite, que en el Pentium 4 queda establecido en 4 lecturas simultáneas (y que han fallado en su acceso a la cache)[3].

La traducción de la dirección virtual del dato a la dirección física con la que se direcciona la cache, se realiza a través del TLB de datos (también de 64 entradas). El acceso a dicha estructura se realiza en paralelo a la cache (aprovechando que los bits de menor peso de la dirección se mantendrán idénticos tras la traducción[4]). Si se produce un fallo en la cache, ya se dispone de la dirección física para realizar el acceso al segundo nivel.

5.1.2 Cache unificada de segundo nivel

En el segundo nivel de cache, el Pentium 4 almacena tanto instrucciones como datos. Como la Tabla 1 indica, el tamaño de línea es inusualmente grande: 128 bytes. En realidad, cada línea está divida en dos sectores que pueden accederse independientemente (pero un fallo en la cache implica la lectura de 128 bytes de memoria principal, para escribir ambos sectores).

---

[3] Nótese que no tiene sentido poner un máximo de fallos de escritura en el primer nivel de cache, pues ésta es write-through. Esto supone que apenas hay diferencia en el comportamiento de la cache cuando la escritura supone un fallo o un acierto.

[4] Para hacer posible este acceso paralelo, debe darse además la condición de que el número de bits destinados al desplazamiento dentro de la página coincida (o sea mayor) que el necesario para acceder al dato correspondiente de la cache. Tras la traducción se comprueba si el tag de la línea de cache es el correcto. En la documentación consultada no se menciona esta necesidad, ni se explica cómo se realiza el acceso en los casos en los que se usa segmentación y no sólo paginación.

El ancho de banda con el primer nivel de Cache se ha visto muy reforzado (ancho de 256 bits), para encarar las fuertes necesidades de las aplicaciones multimedia. El gran tamaño de línea tampoco es casual, pues junto con el mecanismo de prefetch que describiremos a continuación, ayudan a ocultar la gran latencia que suponen los accesos a memoria.

La cache está también segmentada, admitiendo una nueva operación cada dos ciclos de reloj, consiguiéndose un ancho de banda máximo de 48Gbytes por segundo, con el procesador de 1.5GHz.

5.1.3 Bus del sistema

Cuando se produce un fallo en el acceso al segundo nivel de cache, se inicia una petición a la memoria principal a través del bus del sistema. Este bus tiene un ancho de banda de 3.2 Gbytes/s, conseguidos gracias a que los 64 bits del bus se transfieren a una frecuencia efectiva de 400 MHz[10]. En realidad, la frecuencia de trabajo del bus es de 100 Mhz, pero el protocolo usado permite cuadruplicar el número de transferencias hasta un total de 400 millones por segundo.

A pesar del ancho de banda, tras un fallo de cache se emplean más de 12 ciclos (del procesador) en conseguir el bus, y entre 6 y 12 ciclos del bus en acceder a memoria (si no hay congestión). No hay ninguna cifra al respecto, pero creemos que, en media, una instrucción que acceda a memoria principal tendrá una latencia cercana a 100 ciclos del procesador.

5.2 Optimizaciones del sistema de memoria

El Pentium 4 presenta un gran número de optimizaciones para el mejor funcionamiento de su sistema de memoria [8,16]. Algunas de ellas implican recomendaciones de estructuración del código a la hora de programar. Omitiremos dichas recomendaciones para centrarnos únicamente en los aspectos hardware de las optimizaciones, a saber:

- Prefetching de datos, tanto software como hardware
- Reordenación de lecturas (Store forwarding) respecto a otras operaciones de memoria
- Uso del dato a escribir por un store por los loads dependientes de él (Store-to-Load forwarding)
- Ejecución especulativa de lecturas
- Combinación de escrituras
- Múltiples fallos de cache permitidos (ya comentado)

5.2.1 Prefetching

El Pentium 4 tiene dos mecanismos de prefetching: uno controlado por software y otro automático controlado por hardware.

Para el prefetch software, existen cuatro instrucciones IA-32 introducidas junto a las SSE, que permiten llevar una línea de cache al nivel deseado antes de que sea estrictamente necesario. El prefetching puede ocultar la latencia del acceso a memoria si nuestro código tiene un patrón de acceso regular que nos permita saber con cierta antelación qué datos vamos a necesitar.

El prefetch hardware es otra de las novedades del Pentium 4. Permite llevar líneas al segundo nivel de cache (desde memoria principal) siguiendo un patrón reconocido automáticamente por el procesador. En concreto, intenta estar siempre 256 bytes por delante de los datos actualmente en uso. El mecanismo de prefetch lleva la cuenta de los últimos fallos de cache, para intentar evitarlos en el futuro.

Tanto el prefetch software como el hardware suponen un aumento en el uso del ancho de banda, así que su uso debería ser limitado en aplicaciones que ya hacen un uso exhaustivo del ancho. Los posibles beneficios que el prefetch pueda ocasionar, se verán contrarreastados por la merma que supondrá la espera por el control del bus.

5.2.2 Store forwarding (y Store-to-load forwarding)

Aunque los nombre puedan engañar, Store Forwarding y Store-to-Load Forwarding son dos técnicas distintas, aunque ambas tienen que ver con la ejecución de las lecturas. El Store Forwarding permite que una operación de lectura que conoce su dirección de destino se ejecute antes que un store que estaba antes que ella en la cola de operaciones de memoria. Esta circunstancia sólo puede darse si la dirección de escritura también es conocida y no coincide con la del load. Teniendo en cuenta que las escrituras no se llevan a cabo hasta su finalización y el gran número de lecturas y escrituras simultáneas, la espera para realizar una lectura podría ser significativa. Esta técnica resuelve en gran medida este conflicto.

Si la dirección lineal de la lectura coincide con el de una escritura previa, no podremos realizar el Store Forwarding, pero no siempre será necesario esperar a que la escritura tenga lugar. Aquí entra en juego el Store-to-Load Forwarding, que consiste en que, una vez que el store conoce el dato a escribir, puede comunicar dicho dato a la instrucción de lectura, evitando así el acceso a memoria. Para que el dato pueda ser correctamente utilizado por la lectura deben cumplirse tres condiciones:

- Lógicamente, el dato enviado al load debe haber sido generado por un store anterior (anterior en orden lexicográfico) que ya ha sido ejecutado (esto es, se conoce su dato y su dirección).
- Los bytes del dato a leer deben ser un subconjunto de los bytes a escribir.
- Alineación: el store no puede sobre pasar el límite de una línea de cache, y la dirección linear del load debe ser la misma que la del store (para evitar problemas de aliasing).

5.2.3 Ejecución especulativa de loads. Replay

Debido a la profundidad del pipe, se hace necesario asumir que las lecturas en cache no fallarán y planificar las micro-operaciones en consecuencia. No hacerlo así, supondría una parada continua de todas las instrucciones dependientes de cualquier instrucción de lectura, pues la distancia (en ciclos) desde el planificador a la ejecución es mayor que la latencia del load en sí.

Para solucionar los casos en que la ejecución especulativa haya sido errónea, se pone en juego un mecanismo denominado *replay*, que únicamente volverá a ejecutar aquellas micro-operaciones dependientes del load que falló en cache.

5.2.4 Combinación de escrituras

El Pentium 4 dispone de 6 buffers de combinación de escrituras, cada uno de ellos del tamaño de una línea de cache (64 bytes). Cuando se produce una fallo de escritura en el primer nivel de cache, el dato a escribir se almacena en el buffer de combinación de escrituras. Todos los consiguientes fallos de escritura a la misma línea de cache también se almacenarán en dicho buffer.

Esto supone dos ventajas: ahorramos tráfico, pues sólo escribimos una vez en la cache de segundo nivel. Y, en segundo lugar, permite que la lectura de la línea a escribir sea menor: sólo necesitamos los bytes de la línea que no han sido ya escritos en el buffer. Los bytes de la línea leída se combinarán con los presentes en el buffer, y la línea puede ser escrita en la cache.

En [11] se presentan numerosas técnicas de optimización de código para hacer buen uso de todas las características del Pentium 4 aquí presentadas. Remitimos al lector interesado a dichas fuentes, pues las optimizaciones de código están fuera del propósito del presente trabajo.

6. Multithreading

A pesar de todas las técnicas superescalares que presenta el Pentium 4, el rendimiento obtenido no es del todo satisfactorio. El índice de micro-operaciones finalizadas por ciclo de reloj rara vez sobrepasa el 1, cuando el máximo se sitúa en 3 (el Pentium 4 puede lanzar hasta 6 micro-operaciones por ciclo, pero sólo puede retirar 3).

Esto significa que la influencia de los fallos en las predicciones de saltos, y las dependencias de datos son escollos difíciles de superar. Más aún, aplicando técnicas de predicción perfectas, se puede comprobar

que el rendimiento no mejoraría notablemente. En resumen, la dependencia de datos que impone la programación imperativa no puede solventarse ni aunque ampliemos la ventana de instrucciones a todo el código.

Por ello, están surgiendo nuevas propuestas que intentan aprovechar al máximo los recursos disponibles de modo continuado. Una de ellas, el multithreading [12], es la que incorpora el Pentium 4, aunque de modo limitado.

Intel no ha inventado el multithreading. Es una técnica que lleva años en discusión en foros académicos, pero que nunca había visto la luz hasta que el equipo de Alpha decidió incorporarlo en su no-nato procesador 21364. Con la compra de DEC por parte de Intel, todos aquellos proyectos pasaron a incorporarse a la pujante NetBurst.

En las primeras implementaciones del Pentium 4, el multithreading (limitado a 2 threads para este procesador) estaba en el chip, pero desactivado. Actualmente comienzan a salir unidades que hacen uso de esta potente posibilidad, pero aún no sabemos qué resultados está obteniendo.

En cuanto al multithreading en sí, podemos decir que pretende generar un mayor paralelismo ejecutando simultáneamente zonas del código supuestamente independientes (o incluso, códigos diferentes). Los recursos son compartidos por ambos threads, así que debe mantenerse un férreo control en la asignación, para distinguir a quién pertenece cada recurso en cada momento. El objetivo es tener siempre instrucciones que ejecutar, confiando en que, si uno de los threads ha quedado parado momentáneamente por alguna dependencia o fallo de predicción, el otro pueda hacer uso de las unidades funcionales y seguir avanzando su ejecución.

Los mecanismos de control que utiliza el Pentium 4, cómo decide qué es un thread y cómo obtenerlo y cómo gestiona la independencia de los dos threads que permite ejecutar en paralelo, lo desconocemos completamente (y parece que Intel no está por la labor de sacarnos de la ignorancia). Los resultados de los primeros procesadores en pleno funcionamiento dirán si la opción de Intel ha sido la adecuada.

7. Rendimiento

A pesar de la relativa juventud del procesador, Intel se ha encargado de publicar [13] numerosas comparativas del rendimiento ofrecido por su nueva arquitectura frente a sus antecesores y frente a sus más directos rivales. En casi todos los casos, el Pentium 4 sale claramente vencedor, y hay que admitir que , tras las últimas modificaciones tecnológicas, representa la mejor opción en el ámbito doméstico por su gran relación calidad/precio.

Presentamos a continuación (ver **¡Error! No se encuentra el origen de la referencia.**) la comparativa más divulgada, que representa las ganancias en rendimiento del nuevo Pentium 4 frente a su inmediato antecesor, el Pentium 3. La comparativa se realiza con algunos de los benchmarks más ampliamente utilizados: SPEC2000, Winstone y diversas aplicaciones multimedia.

Algunos de los resultados son bastante espectaculares, como la ganancia de un 75% en el caso de los benchmarks en punto flotante de los SPEC 2000. Los programas en punto flotante suelen presentar pocos saltos, y por tanto el Pentium 4 puede aprovechar su mayor ventana de ejecución (128 registros de renombramiento, frente a 40 entradas de la ROB en el Pentium III) para acelerar la ejecución.

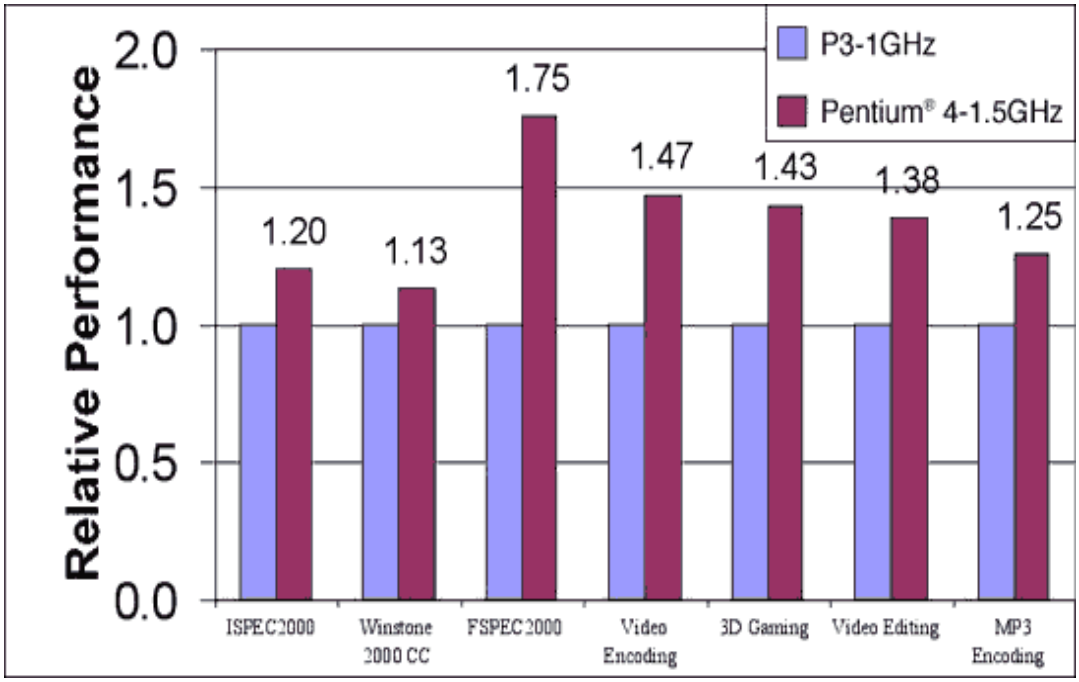


Figura 11Comparativa Pentium 4 - Pentium III

También parece notarse el efecto de las nuevas extensiones SSE2, como demuestran las sustanciales mejoras en "Video Encoding" y "3D Gaming". Las mejoras en los benchmarks con enteros (ISPEC2000) son menores, pero también significativas. Estos programas presentan gran cantidad de saltos de carácter difícil de predecir. Parece que la apuesta por la Trace Cache y las mejoras en los algoritmos de predicción de saltos han dado sus frutos.

Es importante hacer notar que a un aumento en la frecuencia del 50% (actualmente ya hay procesadores a 2.40GHz, así que el factor es aún mayor), no tiene que corresponder una mejora en el rendimiento de la misma magnitud. Esa gran frecuencia se consigue, como hemos visto, a costa de un pipeline más profundo, por lo que el trabajo que se hace en cada etapa es menor. Cabe pensar que la nueva arquitectura representa un reclamo comercial frente a las ofertas de sus competidores (AMD llegó antes que Intel a la barrera del GHz), conscientes que el dato que más vende en el mercado doméstico es la frecuencia.

Veamos a continuación (Figura 12 algunas comparativas entre las distintas versiones del Pentium 4, cada cual a mayor frecuencia. Como cabía esperar, las mejoras son constantes con cada aumento de frecuencia. Pero es interesante resaltar como el mayor aumento se produce por la introducción de una cache de segundo nivel de 512 KB al avanzar a la tecnología de 0,13u (en las gráficas indicado con una "A").

El benchmark elegido es de nuevo SPEC2000. Los resultados mostrados se limitan al conjunto de los programas enteros, pero los resultados obtenidos por los benchmarks en punto flotante son muy similares. Resaltamos de nuevo que los tres últimos modelos están diseñados en 0,13u, lo que les permite duplicar el tamaño de su cache de segundo nivel (el rendimiento mejora notablemente sólo con ese cambio, como se comprueba con el paso de 640 a 722 en el índice SPEC para dos versiones a 2 GHz).

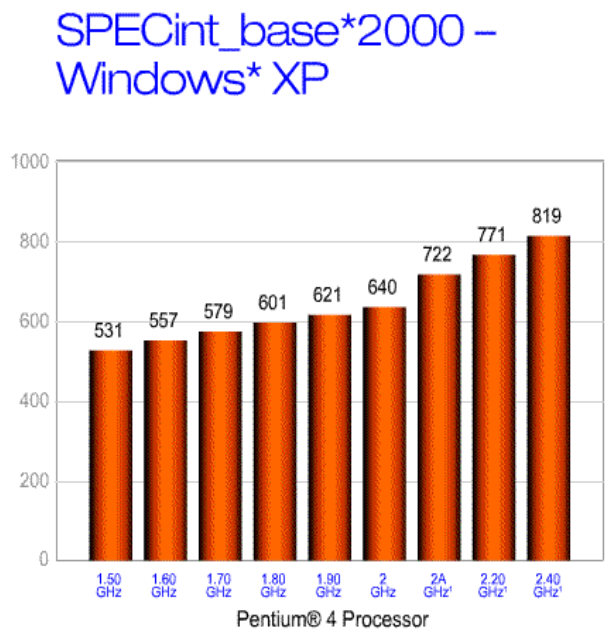


Figura 12 Comparativa entre diversos Pentium 4

No todos los resultados parecían positivos. De hecho, esta comparativa, aparecida en una conocida dirección de internet [14], era bastante desalentadora:

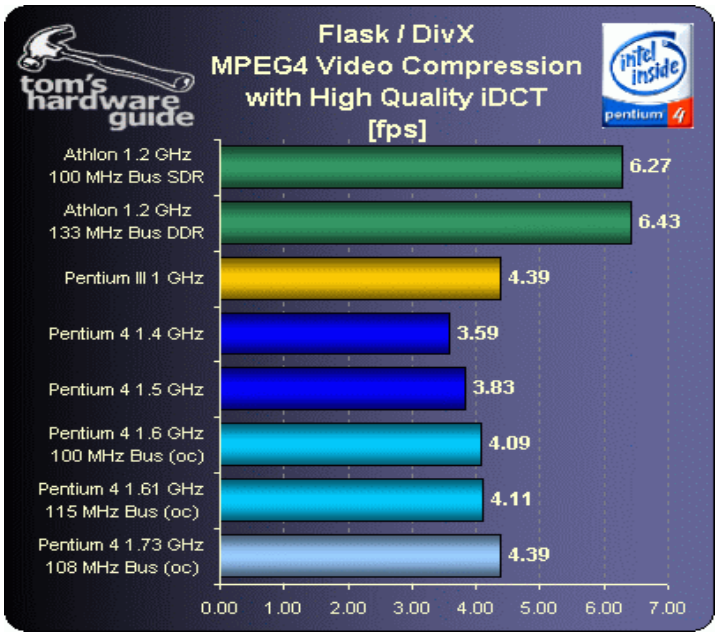


Figura 13 Resultados de diversos procesadores con MPEG4

Parecía incluso que el antiguo Pentium III se comportaba mejor que los nuevos Pentium 4 en un algoritmo tan extendido como MPEG4.

A la vista de los resultados, los ingenieros de Intel comenzaron a indagar en el código hasta dar con el motivo. Simples cambios en las opciones de compilación y un mejor uso de las nuevas extensiones SSE2 conducen a los resultados que presentamos en la Figura 14

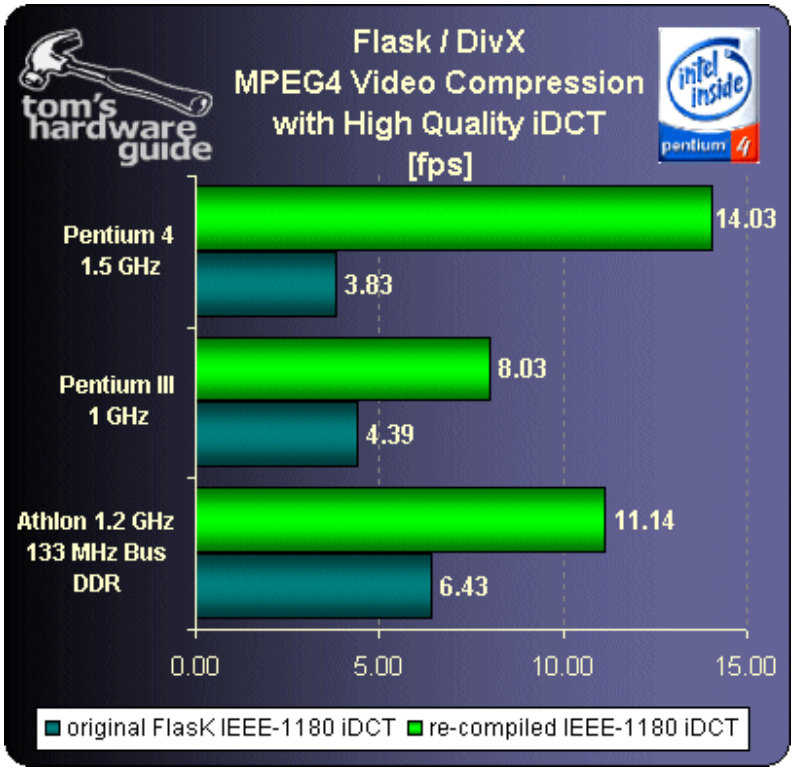


Figura 14 Resultados revisados para MPEG4

Es loable la diligencia con la que los diseñadores de Intel solventaron el percance. Muchas otras comparaciones, siempre positivas para Intel, se han realizado en estos pocos años de vida del procesador. De nuevo remitimos al lector interesado a alguna de las referencias indicadas [16, 17].

8. Conclusiones

Aún se podrían detallar mil aspectos más del funcionamiento del Pentium 4. El diseño e implementación de un superescalar presentan la suficiente complejidad como para poder comprenderlos en toda su magnitud de una sola pasada.

Sí podemos afirmar que Intel ha logrado un enorme avance (aunque las cifras de ventas no le hayan respaldado en los primeros meses) en cuanto a su arquitectura se refiere. Su diseño está pensado para continuar escalando potencia constantemente incluso sin la ayuda de un salto tecnológico.

En los modelos que sucedan al Pentium 4, habrá que ir limando los detalles que aún no terminan funcionar. La Trace Cache es una buena idea y una gran solución para muchos de los problemas de Intel, pero parece que el flujo de micro-operaciones que produce no es aún el deseado. Aún falta por comprobar el rendimiento del multithreading, y tras las primeras pruebas habrá que proceder a su ajuste.

Pero sin duda nos encontramos ante un salto importante para Intel, mayor del que a primera vista podría parecer al compararlo con la arquitectura P6. Es un nuevo punto de partida desde donde continuar buscando mayor rendimiento y, a ser posible, a bajo precio.

9. Bibliografía

[1] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, P. Roussel, The Microarchitecture of the Pentium 4 Processor, Intel Technology Journal, (Q1):13, Febero 2001

[2] Rotenburg, E., Bennett, S., Smith, J. E., Trace Cache: A Low Latency Approach to High Bandwitdh Instructio Fetching. Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture, pages 24-34, 1996, París, Francia.

[3] Patel, Zunaid. Trace Caches in the Context of other Cacche Enhancements, Julio 2000.

[4] Tse-Yu Patt, Yale Patt. Two-Level Adaptative Training Branch Prediction, 24th International Symposium on Microarchitecture, pp. 51-61, Nov. 1991.

[5] Dezso Sima, Terence Fountain, Péter Kacsuk, Advanced Computer Architectures, Addison-Wesley, Gran Bretaña, 1997.

[6] J. L. Hennessy, D. A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann Inc., San Mateo, CA, 1990

[7] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units", IBM Journal of Research and Development, vol 11, no. 1, pp. 34-53, Enero 1967

[8] IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture at https://developer.intel.com/design/pentium4/manuals/245470.htm.

[9] IA-32 Intel Architecture Software Developer's Manual Volume 2: Basic Architecture at https://developer.intel.com/design/pentium4/manuals/245470.htm.

[10] Peter N. Glaskowsky, Pentium 4 (Partially) Previewed, Micro Processor Online, Agosto, 2001

[11] Intel Pentium 4 Processor Optimization Reference Manual at http://developer.intel.com/design/pentium4/manuals/248966.htm.

[12] Gurindar S. Sohi, Amir Roth, Speculative Multithreaded Processors, IEEE Computer, 2001

[13] Intel Pentium 4 Processor and Intel 850 Performance Brief, Abril 2002

[14] http://www.tomshardware.com

[15] Linley Gwennap, Intel´s P6 Uses Decoupled Superscalar Design, Microprocessor Report, pages 9-15, Febrero 1996

[16] Desktop Performance and Optimization for Intel Pentium 4 Processor, Febrero 2001