CAPÍTULO 2.- ARQUITECTURAS COMERCIALES ABIERTAS Y CERRADAS.

2.1 INTRODUCCIÓN

Los avances y progresos en la tecnología de semiconductores, han reducido las diferencias en las velocidades de procesamiento de los microprocesadores con las velocidades de las memorias, lo que ha repercutido en nuevas tecnologías en el desarrollo de microprocesadores. Los microprocesadores **RISC** (reduced instruction set computer) están sustituyendo a los CISC (complex instruction set computer), pero existe el hecho que los microprocesadores CISC tienen un mercado de software muy difundido, aunque tampoco tendrán ya que establecer nuevas familias en comparación con el desarrollo de nuevos proyectos con tecnología RISC. A continuación se describirá brevemente la Arquitectura RISC.

Si lo que se busca es aumentar la velocidad del procesamiento, se descubrió en base a experimentos que con una determinada arquitectura de base, la ejecución de programas compilados directamente con microinstrucciones y residentes en memoria externa al circuito integrado resultaban ser más eficientes, gracias a que el tiempo de acceso de las memorias se fue decrementando conforme se mejoraba su tecnología de encapsulado.

Debido a que se tiene un conjunto de instrucciones simplificado, éstas se pueden implantar por hardware directamente en la CPU, lo cual elimina el microcódigo y la necesidad de decodificar instrucciones complejas.

Las principales características que presenta una arquitectura RISC son:

- Un conjunto limitado y simple de instrucciones. Se cuenta con un conjunto constituido por instrucciones capaces de ejecutarse en un ciclo de reloj.
- Instrucciones orientadas a los registros con acceso limitado a memoria. Un conjunto de tipo RISC ofrece pocas instrucciones básicas (*Load* y *Store*) que pueden ingresar datos en la memoria. El resto de ellas operan exclusivamente con registros
- Modos limitados de direccionamiento. Muchas computadoras de tipo RISC ofrecen sólo un modo para direccionar la memoria, generalmente un direccionamiento directo o indirecto de registros con un desplazamiento.
- Un gran banco de registros. Los procesadores de tipo RISC contienen muchos registros de manera que las variables y los resultados intermedios usados durante la ejecución del programa no requieran utilizar la memoria. Con ello se evitan muchas instrucciones del tipo *Load* y *Store*.
- Palabra de la instrucción con extensión y formatos fijos. Al hacer idénticos el tamaño y el formato de todas las instrucciones, es posible obtenerlas y decodificarlas por separado. No hay que esperar hasta conocer la extensión de una instrucción anterior a fin de obtener y decodificar la siguiente. Por tanto, esas dos acciones pueden llevarse a cabo en paralelo. En pocas palabras, la decodificación se simplifica.

2.2 SUN SPARC.

En 1987, Sun Microsystems anunció una arquitectura RISC abierta denominada SPARC (Scalable Processor ARChitecture, en español Arquitectura de Procesador Escalable), la cual sería la base de futuros productos de la empresa.

Alrededor de media docena de distribuidores de SPARC obtuvieron la licencia para fabricar pastillas SPARC usando diferentes tecnologías (CMOS, ECL, GaAs, Arreglos de compuertas, VLSI (Very Large Scale Integration), etc.). La intención fue alentar la competencia entre los distribuidores de dispositivos, con el fin de mejorar en el desempeño, reducir precios e intentar establecer la arquitectura SPARC como estándar en la industria.

La tecnología Sun, con respecto al SPARC, comenzó con una arquitectura de 32 bits, la cual es la que usan la mayoría de los procesadores fabricados actualmente, pero luego se expandió a una tecnología de 64 bits, lo cual significa el doble de tamaño de los registros y de bus de datos.

A continuación se dará una breve explicación de cada una de las partes que compone al CPU SPARC.

TABLA DE ESPECIFICACIONES DEL SUN ULTRA SPARCII

Core Frequency: 300 MHz
Board Frequency: 100 MHz
Clock Multiplier: 3.0
Data bus (ext.): 64 Bit
Address bus: 64 Bit
Transistor: 5, 400,000
Circuit Size: 0.35 µ
Voltage: 2.5 V
Introduced: 1997
Manufactured: week 50/1999
Made in: USA
L1 Cache: 16+16 KB
L2 Cache: 4 MB ext.
CPU Code: Blackbird
Package Type: Ceramic
Heat Spreader
LGA-787

Tabla 2.2 Especificaciones del Sun Ultra Sparc II.

SPARC es la primera arquitectura RISC que fue abierta y como tal las especificaciones de diseño están publis, así otros fabricantes de microprocesadores pueden desarrollar su propio diseño.

Una de las ideas innovadoras de esta arquitectura es la ventana de registros que permite hacer fácilmente compiladores de alto rendimiento y una significativa reducción de memoria en las instrucciones load/restore en relación con otras arquitecturas RISC. Las ventajas se aprecian sobre todo en programas grandes.

El cpu SPARC está compuesto de una unidad entera, UI (Integer Unit) que procesa la ejecución básica y una FPU (Floating-Point Unit) que ejecuta las operaciones y cálculos de reales. La IU y la FPU pueden o no estar integradas en el mismo chip.

La arquitectura SPARC se ha definido con mucho cuidado para permitir la implantación de procesamiento en serie muy avanzado. Entre otros aspectos, define retardos en carga y almacenamiento, bifurcaciones, llamadas y retornos. La implantación típica tiene un procesamiento en serie de cuatro etapas (como se muestra en la siguiente figura). Durante el primer ciclo se extrae de la memoria la palabra de la instrucción; en el segundo se decodifica; durante el tercero se ejecuta; por último en el cuarto ciclo se escribe el resultado otra vez en la memoria.

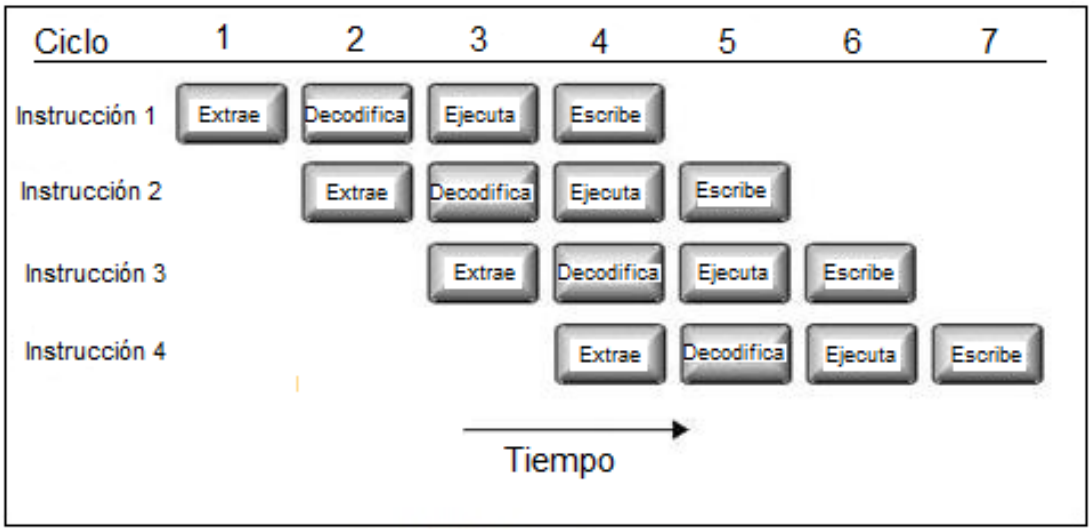


Figura 2.2 Arquitectura Sparc de procesamiento en serie.

2.2.1 PRINCIPALES CARACTERISTICAS.

Su característica distintiva a otras arquitecturas es que utiliza ventanas de registros, cuenta con 32 registros de enteros de 32 bits y16 registros de punto flotante de 64 bits (para el caso de doble precisión) que se pueden utilizar como 32 registros de 32 bits (para precisión simple).

Modos de direccionamiento:

- Inmediato, (constantes de 13 bits).
- Directo, (offset de 13 bits).

- Indirecto, (registro + offset de 13 bits o registro + registro).

Utiliza instrucciones retardadas (saltos, load y store).

Manejo de memoria:

- Espacio virtual de 4 Gigabytes.
- Unidad de manejo de memoria (MMU) que trabaja con páginas de tamaño configurable.

Otra característica importante de los procesadores SPARC es que son procesadores RISC (Computadora con reducido conjunto de instrucciones). Para que un procesador sea considerado RISC debe cumplir, entre otras cosas, que el tamaño de sus instrucciones no sea variable, y que en consecuencia estas se completen en un solo ciclo (entendiendo por ciclo la extracción de los operandos de un registro, colocarlos en el bus, ejecutarlos en la ALU, y guardar el resultado en un registro).

2.2.2 DESCRIPCIÓN DE SUS PARTES.

Componentes

Un procesador SPARC comprende una Unidad de Enteros (UE), una Unidad de Punto Flotante (UPF) y un Co-Procesador opcional, cada uno de ellos con sus propios registros. Ésta organización permite una máxima coordinación entre la ejecución de instrucción de enteros, de punto flotante y de co-procesador. Todos los registros, con la posible excepción de los del co-procesador, tienen una longitud de 32 bits.

El procesador puede estar en uno de dos modos: usuario o supervisor. En el modo supervisor el procesador puede ejecutar cualquier instrucción, incluyendo aquellas privilegiadas (sólo-supervisor). En el modo de usuario, un intento de ejecutar una instrucción privilegiada causaría una trap (señal) al software supervisor.

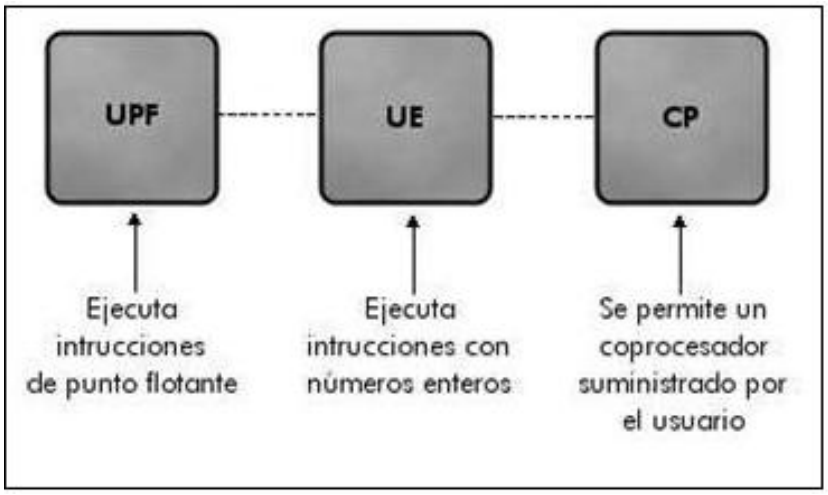


Figura 2.2-2 Diagrama de un procesador SPARC.

A continuación se describen cada una de las partes que conforman a un procesador SPARC:

Unidad de Enteros (UE)

La UE contiene los registros de propósito general y controla todas las operaciones del procesador. La UE ejecuta instrucciones aritméticas de enteros y computa direcciones de memoria para cargas y almacenamientos. También mantiene el contador de programa y controla la ejecución de instrucciones de la UPF y el CP. Una UE puede contener desde 40 hasta 520 registros r de propósito general de 32 bits cada uno. Esto corresponde a una agrupación de los registros en 8 registros r globales, más un stack circular de entre 2 y 32 sets de 26 registros cada uno, conocido como ventanas de registros.

Unidad de Punto Flotante (UPF)

La UPF tiene 32 registros de punto flotante de 32 bits cada uno. Para almacenar valores de doble precisión se utilizan 2 registros, y valores de cuádruple precisión ocupan un grupo de 4 registros adyacentes. En consecuencia, los registros de punto flotante pueden contener un máximo de 32 valores de simple precisión, 16 de doble precisión, u 8 de cuádruple precisión.

Las instrucciones de carga y almacenamiento en punto flotante son usadas para mover datos entre la UPF y la memoria. La dirección de memoria es calculada por la UE.

Co-Procesador (CP)

El Co-Procesador tiene su propio set de registros de normalmente 32 bits. Instrucciones de carga/almacenamiento del Co-Procesador son las que se usan para mover datos entre los registros del Co-Procesador y la memoria. Para cada instrucción de carga/almacenamiento de punto flotante, hay una instrucción de carga/almacenamiento del Co-Procesador análoga.

2.2.3 CATEGORÍAS DE INSTRUCCIONES.

La arquitectura SPARC tiene cerca de 50 instrucciones enteras, unas pocas más que el anterior diseño RISC, pero menos de la mitad del número de instrucciones enteras del 68000 de Motorola.

Las instrucciones de SPARC se pueden clasificar en cinco categorías:

- LOAD y STORE (La única manera de acceder a la memoria). Estas instrucciones usan dos registros o un registro y una constante para calcular la dirección de memoria a direccionar.
- Instrucciones Aritméticas/Lógicas/Shift. Ejecutan operaciones aritméticas, lógicas y operaciones de cambio. Estas instrucciones calculan el resultado si es una función de 2 operandos y guardan el resultado en un registro.
- Operaciones del Coprocesador. La IU extrae las operaciones de punto flotante desde las instrucciones del bus de datos y los coloca en la cola para la FPU. La FPU ejecuta los cálculos de punto flotante con un número fijo en unidad aritmética de punto flotante, (el número es dependiente de la aplicación). Las operaciones de punto flotante son ejecutadas concurrentemente con las instrucciones de la IU y con otras operaciones de punto flotante cuando es necesario. La arquitectura SPARC también especifica una interfaz para la conexión de un coprocesador adicional.
- Instrucciones de Control de Transferencia. Estas incluyen jumps, calls, traps y branches. El control de transferencia es retardado usualmente hasta después de la ejecución de la próxima instrucción, así el pipeline no es vaciado porque ocurre un control de tiempo. De este modo, los compiladores pueden ser optimizados por ramas retardadas.
- Instrucciones de control de registros Read/Write. Estas instrucciones se incluyen para leer y grabar el contenido de varios registros de control. Generalmente la fuente o destino está implícito en la instrucción.

De este modo existen instrucciones para cargar y almacenar cantidades de 8, 16, 32 y 64 bits, en los registros de 32 bits, usando en este último caso dos registros consecutivos.

Una diferencia de los procesadores CISC (Computadora con complejo conjunto de instrucciones) a los procesadores RISC es que una gran parte no poseen “stack”.

Dado que una de las características que se desea en un procesador es rapidez, debe tenerse en cuenta que las instrucciones que extraen sus operandos de los registros y almacenan los resultados también en ellos, se pueden manejar en un solo ciclo. Sin embargo, aquellas que cargan información desde la memoria o almacenan en ésta consumen demasiado tiempo. Es por ello que los procesadores RISC, incluyendo el SPARC, poseen una alta cantidad de registros internos de tal manera que las instrucciones ordinarias tienen operandos provenientes de los mismos**.**

2.2.4 VENTANAS DE REGISTROS.

Un rasgo único caracteriza al diseño SPARC, es la ventana de overlapping de registros. El procesador posee mucho más que 32 registros enteros, pero presenta a cada instante 32. Una analogía puede ser creada comparando la ventana de registros con una rueda rotativa. Alguna parte de la rueda siempre está en contacto con el suelo; así al girarla tomamos diferentes porciones de la rueda, (el efecto es similar para el overlap de la ventana de registros). El resultado de un registro se cambia a operando para la próxima operación, obviando la necesidad de una instrucción Load y Store extras.

Se acordó para la especificación de la arquitectura, poder tener 32 registros "visibles" divididos en grupos de 8.

De r0 a r7, Registros GLOBALES.

De r7 a r15, Registros SALIDA.

De r15 a r23, Registros LOCALES.

De r24 a r31, Registros ENTRADA.

Los registros globales son "vistos" por todas las ventanas, los locales son solo accesibles por la ventana actual y los registros de salida se solapan con los registros de entrada de la ventana siguiente (los registros de salida para una ventana deben ponerse como registros de entrada para la próxima, y deben estar en el mismo registro).

El puntero de ventana mantiene la pista de cual ventana es la actualmente activa. Existen instrucciones para "abrir" y "cerrar" ventanas, por ejemplo para una instrucción "call", la ventana de registros gira en sentido anti horario; para el retorno desde una instrucción "call", esta gira en sentido horario.

Una interrupción utiliza una ventana fresca, es decir, abre una ventana nueva. La cantidad de ventanas es un parámetro de la implementación, generalmente 7 u 8.

La alternativa más elaborada para circundar lentamente la ventana de registros es colocar los registros durante el tiempo de compilación. Para lenguajes como C, Pascal, etc., esta estrategia es difícil y consume mucho tiempo. Por lo tanto, el compilador es crucial para mejorar la productividad del programa.

"Recientes investigaciones sugieren que la ventana de registros, encontradas en los sistemas SPARC pero no en otras máquinas RISC comerciales, están en condiciones de proveer excelente rendimiento para lenguajes de desarrollo como Lisp y Smalltalk." (R. Blau, P.Foley, etc. 1984).

2.2.4.1 INSTRUCCIONES.

Se puede utilizar LOAD y STORE en cualquiera de los dos formatos 1a y 1b, en los que DEST sería el registro a cargar o almacenar, los 19 bits de orden inferior de la instrucción determinan cómo se realiza el cálculo de la dirección efectiva de memoria. Se proporcionan dos modos de direccionamiento (formatos 1a y 1b):

1- Dirección de memoria = ORG1 + ORG2

2- Dirección de memoria = ORG1 + constante de 13 bits con signo codificado en complemento a dos, perteneciente al rango de intervalo cerrado –4096, 4095.

Los 32 registros se pueden usar tanto como registros de origen o como de destino (ORG1 u ORG2)**.**

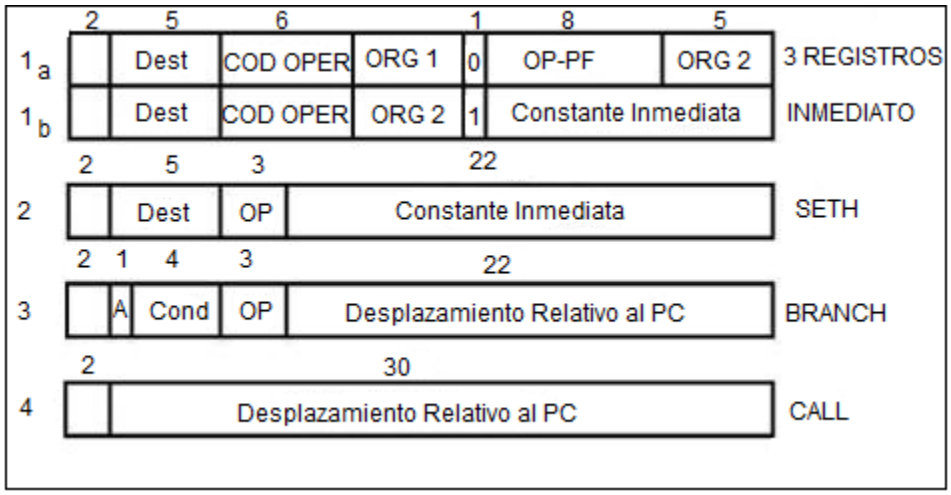


Figura 2.2-4 Formato de Instrucción SPARC.

Volviendo al problema sobre cómo colocar en cualquiera de ellos un valor de 32 bits si cualquier instrucción utiliza como máximo 32 bits, suponiendo que se desea colocar en el registro %4 el valor FFFF0000h. La instrucción debería utilizar algunos bits para codificar el 'mov.' y qué registro quiero utilizar con lo cual quedan menos de 32 bits para almacenar el CPU SPARC – 5 valores inmediatos FFFF0000h. La respuesta a la pregunta es claramente que no se puede realizar con una sola instrucción.

Dijimos que para realizar esta tarea el procesador SPARC provee una instrucción especial llamada SETHI que permite colocar un valor de hasta 22 bits en los 22 bits más significativos de algún registro. Y que utilizando la instrucción OR es posible colocar los 10 bits menos significativos de la constante en los 10 bits menos significativos del registro.

La forma de direccionar memoria utilizada por los procesadores SPARC es a través de registros y tiene la siguiente sintaxis: `#desp13` (registro) o (1) (registro) (registro) (2).

Aquí desp13 es un número de 13 bits con signo codificado en complemento a 2, es decir que pertenece al rango [-4096,4095] y registro es cualquiera de los 32 registros del procesador.

La semántica de la expresión (1), es decir a qué dirección representa, está dada por:

(Registro)+desp13 y la semántica de la expresión (2) es (registro1) + (registro2) (registro) es el valor almacenado en el registro**.**

2.2.4.2 TRAPS Y EXCEPCIONES.

El diseño SPARC soporta un set total de traps o interrupciones. Ellos son manejados por una tabla que soporta 128 interrupciones de hardware y 128 traps de software. Sin embargo las instrucciones de punto flotante pueden ejecutarse concurrentemente con las instrucciones enteras, los traps de punto flotante deben ser exactos porque la FPU provee (desde la tabla) las direcciones de las instrucciones que fracasan.

2.2.5 PROTECCIÓN DE MEMORIA.

Algunas instrucciones SPARC son privilegiadas y pueden ser ejecutadas únicamente mientras el procesador esta en modo supervisor. Estas instrucciones ejecutadas en modo protegido aseguran que los programas de usuario no sean accidentalmente alterados por el estado de la máquina con respecto a sus periféricos y viceversa. El diseño SPARC también proporciona protección de memoria, que es esencial para las operaciones multitarea.

El SPARC tiene muchas similitudes con el diseño de Berkeley, el RISC II. Semejante al RISC II, él usa una ventana de registros para reducir el número de instrucciones Load y Store.

2.2.6 SPARC SEGÚN SUN MICROSYSTEMS.

Hasta hace poco, las arquitecturas RISC tenían una pobre rendimiento con respecto a los cálculos de punto flotante. Por ejemplo, el IBM 801 implementaba las operaciones de punto flotante por software. Los proyectos de Berkeley, RISC I y RISC II, dejaban fuera a una VAX 11/780 en cálculos enteros pero NO en aritmética de punto flotante.

Esto también es cierto para el procesador de Stanford, el MIPS. Los sistemas SPARC, en cambio, son diseñados para un rendimiento óptimo en los cálculos de punto flotante y soportan precisión simple, doble y extendida en los operandos y en las operaciones como lo especifica la norma 754 del ANSI/IEEE del estándar sobre punto flotante."

El alto rendimiento en los cálculos de punto flotante resulta de la concurrencia de la IU y la FPU. La IU (Integer Unit) hace los "load" y "store" mientras la FPU (Floating Point Unit) ejecuta las operaciones y cálculos.

Los sistemas SPARC consiguen obtener velocidades elevadas como resultado del perfeccionamiento en las técnicas de fabricación de los chips.

El sistema SPARC entrega muy altos niveles de rendimiento. La flexibilidad de la arquitectura hace a los futuros sistemas capaces de obtener muchos mejores tiempos que el de la implementación inicial. Además, la arquitectura abierta hace esto posible por absorber los avances tecnológicos casi tan pronto como estos ocurren.

2.2.6.1 IMPLEMENTACIONES.

SPARC

Primera generación liberada en 1987.

Frecuencias de reloj de 16 a 50 Mhz.

Diseño escalar.

SUPER SPARC

Segunda generación liberada en 1992.

Frecuencias de reloj de 33 a 50 Mhz.

Diseño súper escalar.

ULTRA SPARC II

Lanzado a mediados de 1996.

Arquitectura súper escalar de 4 etapas y de 64 bits.

Cinco unidades de punto flotante.

Velocidades entre 250 y 300 Mhz.

Advanced Product Line (APL)

Lanzado a mediados de 2004.

Acuerdo comercial entre Sun Microsystems y Fujitsu

Arquitectura súper escalar compatible con en el diseño SPARC V9 de 64 bits.

Velocidades entre 1,35 y 2,7 GHz.

Utilizado por Sun Microsystems, Cray Research, Fujitsu / ICL y otros.

La industria de la electrónica se ha dado a la tarea de producir Chips Multiprocesadores (CMP) y Chips con una serie de multiprocesadores (CMT). CMP como su nombre lo indica es un grupo de procesadores integrados en el mismo chip. Más allá de los simples procesadores CMP, los chips multiproceso (CMT) dieron un paso más y soporte de hardware simultáneo a muchas líneas de actuación (o hilos) de ejecución por núcleo para tareas ejecutadas simultáneamente (SMT). Los beneficios de los procesadores CMT se manifiestan en una amplia variedad de aplicaciones, por ejemplo en el espacio comercial, las cagas de trabajo de un servidor caracterizado por altos niveles y los grandes conjuntos de trabajo.

Dado el crecimiento exponencial de transistores por chip en el tiempo, una regla general es que un diseño de la placa se convierte en un diseño de chips en menos de diez años. Por lo tanto la mayoría de los observadores de la industria espera que el multiproceso a nivel del chip se vuelva eventualmente en una tendencia de diseño dominante. Para el caso del procesador de un solo chip fue presentado en 1996 por el equipo de *Kunle Olukotun* en La Universidad de Stanford, para el cual pidió la integración de cuatro procesadores basados en un solo chip.

Sun anunció dos procesadores SPARC CMP: Géminis, un dualCore UltraSPARC II de derivados, y UltraSPARC IV que son los procesadores CMP de primera generación se derivan de los diseños con un solo procesador interior y los dos núcleos no comparten todos los demás recursos de caminos de datos.

2.2.6.2.- Características Relevantes

Ultra SPARC IV a 1 GHz, 1.2 GHz

Diseño Chip MultiThreading
– Dual thread
– Subsistema de memoria mejorado
– Mayores características RAS a Actualización
– Pin-compatible con UltraSPARC III
Tecnología 130 nm, próxima generación UIV+90 nm a Hasta 2x mayor throughput que UltraSPARC-III

UltraSPARC III a 1 GHz, 1.2 GHz

Proceso de fabricación
– 29 millones de transistores
– L1 64 KB 4-way data cache, asociativa por conjuntos
– L1 32 KB 4-way instruction cache, asociativa por conjuntos
– Transistores de puerta de 100 nm
– 0.13u CMOS 6 Layer metal, cobre
a Controlador de memoria dentro del chip
– 2.4 GB/SEC ancho de banda a memoria por proc.
Etiquetas de caché externa L2 dentro del proc

– Chequeo de coherencia ejecutando a la veloc.
– 9.6 GB/sec ancho de banda de coherencia

Sun® UltraSPARC IV Processor

Tipo de arquitectura:

- Chip multithreading (CMT) procesador con dos líneas o hilos (threads) por procesador
- Basado en dos UltraSPARC III pipelines
- Arquitectura de 64-Bit SPARC con Set de Instrucciones VIS
- 66 millones de transistores
- 4-vías superescalar
- 14-stage nonstalling pipeline

Tecnología de procesos:

- CMOS .13 micrones de procesos
- 1368-pin flip-chip cerámico Land Grid Array (LGA)

Interconexión:

- Sun Fireplane interconectado corriendo a 150 MHz.

Frecuencia:

- Frecuencia Clock: 1.05 - 1.2 GHz

Cache:

- L1 Cache (por pipeline): 64 KB 4-vías datos

32 KB 4-vías instrucciones
2 KB Write, 2 KB Prefetch

- L2 Cache: 16 MB externos (exclusivo acceso a 8 MB por pipeline)

Controlador On-chip y tags de direcciones

Escalabilidad:

- Escalabilidad de multiprocesamiento con soporte de arquitectura para más de 1000 procesadores/sistema.

Memoria:

- Memoria Máxima: Subsistema de 16GB de memoria por procesador.
- Controlador de Memoria: Controlador de memoria On-chip capaz de direccionar hasta 16 GB de memoria principal a 2.4 GB/s.

I/O:

- Interfaz de Bus: Interconexión Sun Fireplane.

Alimentación:

- Consumo de energía: Un máximo de 108 watts a 1.2 GHz, 1.35 volts.

Sistemas:

- Desarrollado en:

Servidores Sun FIRE High-End
Servidor Sun FIRE E25K
Servidor Sun FIRE E20K
Servidores Sun FIRE Midframe
Servidor Sun FIRE E2900

Servidor Sun FIRE E4900
Servidor Sun FIRE E6900
En 2006, Sn introdujo una forma de CMT de 32 procesadores SPARC, llamado UtraSPARC T1, el cual cuenta con ocho núcleos.

Figura 2.2-6 Descripción Técnica de la Arquitectura Sparc.

2.2.7 Arquitectura SPARC-V9.

SPARC-V9 supuso un cambio realmente significativo a la arquitectura SPARC desde su lanzamiento en 1987, poniendo a esta arquitectura en la cima del altamente competitivo mundo de los microprocesadores RISC. SPARC-V9 extiende el espacio de direcciones a 64 bits, añade nuevas instrucciones, e incorpora algunas mejoras realmente significativas.

Se han implementado procesadores SPARC usados en un amplio rango de computadores que abarca desde pequeñas consolas a supercomputadores. SPARC-V9 mantiene la compatibilidad binaria para las aplicaciones software desarrolladas para implementaciones anteriores de SPARC, incluyendo *microSPARC* y *SuperSPARC*.

SPARC-V9, como su predecesor SPARC-V8, es una especificación de microprocesador creada por *The SPARC Architecture Committee* de *SPARC International*, por lo que cualquiera puede realizar una implementación del microprocesador y obtener una licencia de *SPARC International*. *SPARC International* es un consorcio de fabricantes de computadores, que permite la asociación a cualquier compañía del mundo.

SPARC-V9 mejora la versión 8, proporcionando soporte explícito para:

- Direcciones y datos enteros de 64 bits.
- Prestaciones de sistema mejoradas.
- Compiladores optimizantes avanzados.

- Implementaciones superescalares.
- Sistemas operativos avanzados.
- Tolerante a fallos.
- Cambios de contexto y manejo de *traps* extremadamente rápido.
- Admite ordenación de bytes *big-endian* y *Little endian*.

SPARC-V9 soporta directamente direcciones virtuales de 64 bits y tamaños de datos enteros de hasta 64 bits, incorporando varias instrucciones que manejan de forma explícita valores de 64 bits. Por ejemplo, las instrucciones LDX y STX cargan y almacenan respectivamente valores de 64 bits.

CARACTERISTICAS MÁS RELEVANTES.

A pesar de estos cambios, los microprocesadores de 64-bits SPARC-V9 podrán ejecutar programas compilados para procesadores SPARC-V8 de 32 bits. Esto se logra asegurando que las antiguas instrucciones continúan generando el mismo resultado en los 32 bits de menor orden de los registros. No obstante, y para aprovechar mejor la extensión de dirección y las capacidades avanzadas de SPARC-V9, es recomendable recompilar los programas escritos para SPARC-V8.

Para aumentar las prestaciones, el juego de instrucciones ha sido mejorado introduciendo las siguientes características:

- Instrucciones de división y multiplicación enteras.
- Instrucciones de LOAD y STORE de cuádruples palabras en coma flotante.
- Predicción de saltos establecidos por *software*, lo que da al *hardware* una gran probabilidad de mantener el *pipeline* del procesador lleno.
- Saltos condicionales en función del contenido de un registro, lo que elimina la necesidad de ejecutar una instrucción de enteros que actualice dicho código de condición. De esta forma, se elimina un cuello de botella potencial y se crean mayores posibilidades de paralelismo.
- Instrucciones de *move* condicionales, que ayudan a minimizar saltos en el código de las aplicaciones.

SPARC-V9 contiene instrucciones específicas que permiten al *hardware* detectar solapamiento de punteros, ofreciendo al compilador una solución sencilla para este complejo problema. Pueden compararse dos punteros, y almacenar en un registro de enteros el resultado de la comparación. La instrucción FMOVRZ podría mover condicionalmente un registro de coma flotante basándose en el resultado de la comparación anterior.

Esta instrucción puede ser usada para corregir problemas de solapamiento, permitiendo adelantar las instrucciones LOAD tras los STORES. Todo esto supone una diferencia significativa en las prestaciones globales de los programas.

Se ha añadido un registro TICK que es incrementado una vez por cada ciclo de máquina. Este registro puede ser leído por una aplicación de usuario para realizar medidas simples y precisas de las prestaciones de un programa.

2.2.8 SISTEMAS OPERATIVOS AVANZADOS.

La interface con el sistema operativo ha sido completamente rediseñada en el SPARC-V8 para soportar mejor el desarrollo de nuevos sistemas operativos. Los registros privilegiados o de supervisor proporcionan una estructura única, lo que simplifica el acceso a información de control importante del procesador. Es por esto por lo que un cambio en el interface del sistema operativo no tendrá efecto en el *software* de la aplicación. Los programas a nivel de usuario no verían estos cambios, y por lo tanto mantienen la compatibilidad binaria sin tener que ser recompilados.

Con el objetivo de soportar un nuevo estilo de *microkernel*, SPARC-V9 proporciona niveles de *traps* anidados que permiten una estructuración del código más modular, también proporciona un soporte mejorado para cambios de contexto más rápidos que los de la arquitectura SPARC anterior. Esto permite una estructura de ventanas de registros más flexible que en los primeros SPARC, tal que el *kernel (núcleo)* puede proporcionar un banco de registros separados a cada proceso que se esté ejecutando. Como resultado, el procesador puede realizar un cambio de contexto sin prácticamente ninguna sobrecarga. Esta nueva implementación de las ventanas de registros también proporciona un mejor soporte para sistemas operativos orientados a objeto, ya que acelera la comunicación de procesos entre dominios diferentes.

Existe también un mecanismo que proporciona accesos a servidor eficientes a través de espacios de direcciones de clientes usando identificadores de espacio de direcciones de usuarios. La definición de un núcleo de espacio de direcciones permite al sistema operativo existir en un espacio de direcciones diferente que el del programa de usuario.

Las primeras implementaciones de SPARC soportaban multiproceso; este soporte ha sido ahora extendido a multiproceso a muy alta escala. Esta extensión incluye una nueva instrucción de memoria y un nuevo modelo de memoria llamado *Relax Memory Order (RMO)*. Con estas nuevas capacidades, los procesadores SPARC-V9 pueden planificar las operaciones de memoria para alcanzar mayores prestaciones al tiempo que mantienen la sincronización y el bloqueo de operaciones necesario para el multiproceso con memoria compartida.

Finalmente, se ha añadido soporte arquitectural para ayudar al sistema operativo a proporcionar ventanas de registros ``limpias´´ a sus procesos. Se ha de garantizar que una ventana limpia contenga inicialmente ceros y, durante su periodo de vida, sólo datos y direcciones generadas por el proceso. De esta forma se facilita la implementación de un sistema operativo seguro que pueda proporcionar un aislamiento completo entre sus procesos.

Cuando Sun Microsystems empezó a comercializar el chip multiproceso (CMT) UltraSPARC T1 sorprendió a la industria al anunciar que no sólo enviaría el procesador, sino también el código abierto del procesador por primera vez en la industria. En marzo de 2006 UltraSPARC había sido de código abierto para ser una distribución llamada OpenSPARC T1.

Para el siguiente año, Sun empezó a distribuir sus nuevos y avanzados procesadores UltraSPARC T2 con código abierto la mayor parte de ese diseño como OpenSPARC T2.

El código fuente para ambos procesos incluye no sólo a millones de líneas de hardware, lenguajes de descripción para estos microprocesadores, sino también las secuencias de comandos para compilar el código fuente.

2.2.9 ANTECEDENTES GENERALES DEL OPENSPARC.

OpenSPARC T1 es el primer chip multiprocesador que implementa totalmente la iniciativa de rendimiento de Sun. OpenSPARC T2 es la continuación del chip procesador multihilo (CMT) al procesador OpenSPARC T1.

Históricamente, los microprocesadores han sido diseñados para soportar cargas de trabajo y como resultado se han centrado en el funcionamiento de un solo hilo los más rápidamente posible. El rendimiento de un solo hilo en estos microprocesadores se obtiene por una combinación de tuberías muy profunda y por la ejecución de múltiples instrucciones en paralelo.

Para muchas cargas de trabajo comerciales, el núcleo del procesador físico estará libre o desocupado la mayor parte del tiempo de espera en la memoria, e incluso cuando se está ejecutando, y a menudo se podrá estar utilizando una pequeña fracción de su ancho, así en lugar de construir un procesador ILP grande y complejo que se encuentre inactivo la mayor parte del tiempo, se construirá un procesador con pequeños núcleos en el mismo chip. La combinación de múltiples núcleos en un procesador en un chip permite un rendimiento muy alto para muchos subprocesos en aplicaciones comerciales. Este enfoque se denomina paralelismo a nivel de hilos (TLP). La diferencia entre TLP e ILP se muestra en la siguiente figura. (Figura 2.1.9)

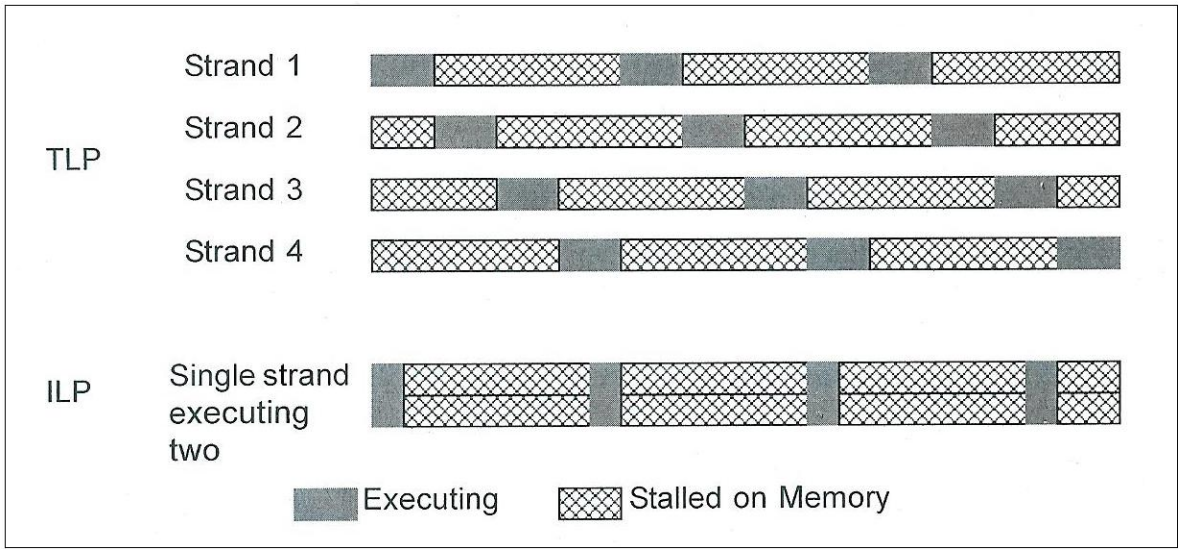


Figura 2.2-9 Diferencias entre TLP e ILP.

El tiempo parado de memoria de un hilo puede ser superpuesto con la ejecución de otros hilos en el núcleo del procesador con las mismas características físicas para ser ejecutados en paralelo.

2.2.9.1 USOS DEL OpenSPARC.

Usos Académicos

El uso académico más común de OpenSPARC hasta la fecha es como un completo ejemplo de arquitectura de proceso y / o ejecución. Puede ser utilizado en áreas tales como cursos de arquitectura de computadoras, diseño VLSI, el código del compilador, e ingeniería informática en general.

En cursos de laboratorio de universidades, OpenSPARC proporciona un diseño que puede ser utilizado como un buen punto de partida para proyectos asignados.

Usos Comerciales

OpenSPARC es un trampolín para el diseño de los procesadores comerciales. Por ejemplo, a partir de un completo y conocido diseño, y realizando una inspección adecuada el tiempo de salida al mercado para un procesador puede ser drásticamente recortado debido a que ya se cuenta con la infraestructura pero sobre todo a un prediseño altamente calificado. Desde el diseño derivado con un núcleo y todo el camino para desarrollar un diseño con ocho núcleos como el OpenSPARC T1 o T2.

Un diseño OpenSPARC puede ser sintetizado y se cargan en un FPGA, y se puede utilizar de muchas maneras:

- Una versión FPGA, el procesador puede ser usado para los prototipos de productos lo cual permite una rápida iteración del diseño.
- El FPGA se pude utilizar para proporcionar a un motor de simulación de alta velocidad un procesador de bajo desarrollo.
- En el caso extremo y al tiempo de las necesidades del mercado, donde los costos de producción por procesador no son críticos, dicho procesador podría ser incluso enviado en forma de FPGA y así podría también ser útil para la descarga de software.

Características del OpenSPARC T1.

Como anteriormente se mencionó, OpenSPARC T1 es un multiprocesador en un solo chip el cual contiene ocho núcleos de procesador SPARC. Cada núcleo de procesadores SPARC tiene soporte de hardware para cuatro procesadores virtuales (o "ramas"). Estos cuatro hilos se ejecutan simultáneamente con las instrucciones, así cuando existe un problema como un error de caché, es marcado como no disponible y las instrucciones de esa cadena no se emiten hasta que el problema no se haya resuelto, mientras las demás instrucciones restantes de los otros hilos continúan desarrollándose.

Los ocho núcleos físicos de SPARC están conectados a través de una barra transversal a un chip unificado con líneas de 64 bytes, así la caché L2 tiene cuatro maneras de proporcionar el ancho de banda suficiente para los ocho núcleos. En el mismo chip además se encuentran un controlador Bus y pines de entrada y salida accesibles a los núcleos, a continuación se ilustra un diagrama de bloques del chip OpenSPARC T1. (Figura 2.1.9-1).

Componentes del OpenSPARC T1.

- Núcleo físico SPARC.
- Unidad de punto flotante (FPU).
- Caché L2.
- Controlador DRAM.
- Entradas y salidas.
- Interface (JBI).
- Unidad de prueba de reloj (CTU).
- Fusible o tapón electrónico.

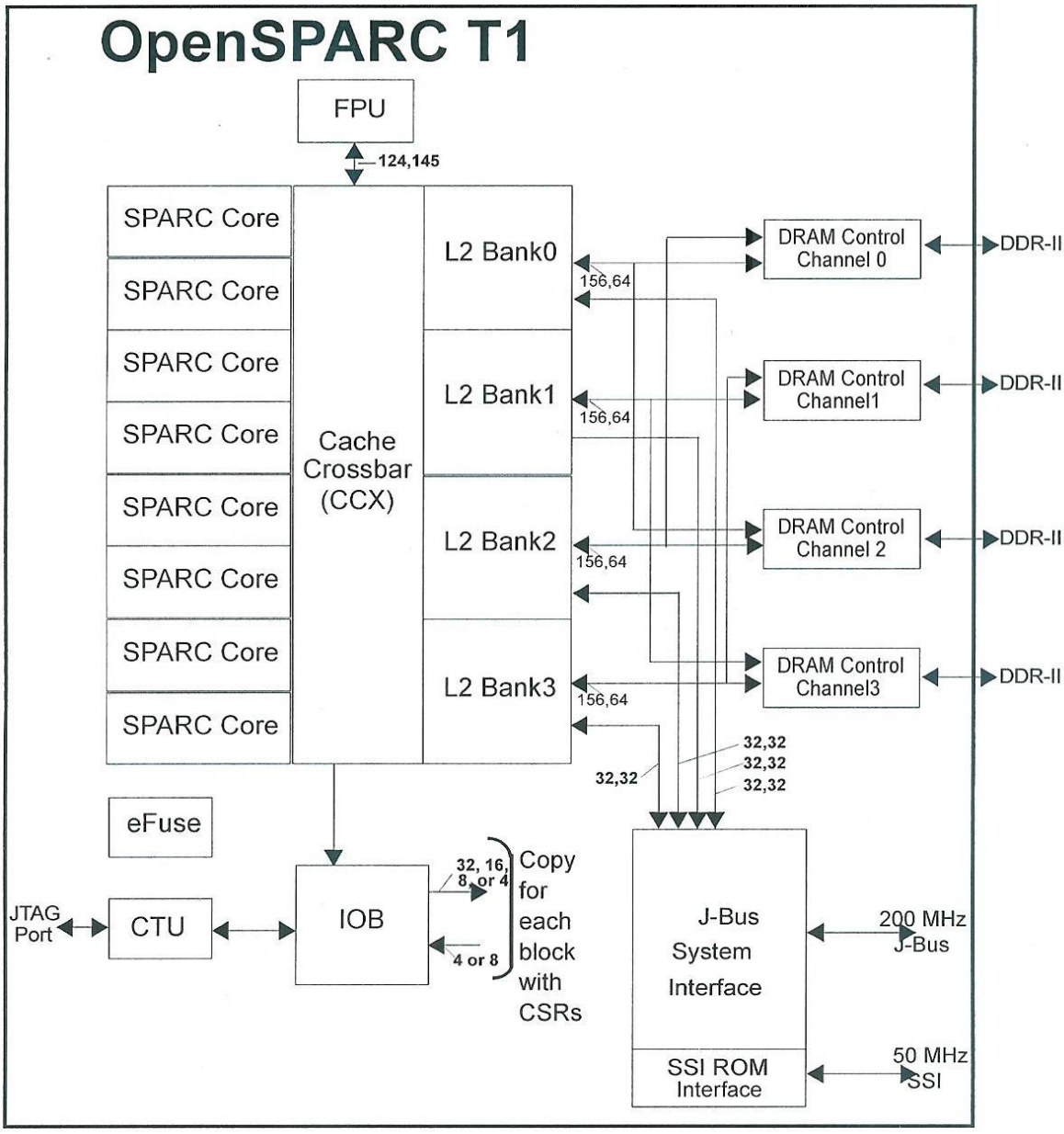


Figura 2.2.9-1 Diagrama de bloques del chip OpenSPARC T1. Cortesía OpenSPARC.

Núcleo físico: Cada núcleo del OpenSPARC T1 tiene soporte de hardware para sus cuatro ramas, este apoyo consiste en un archivo de registro completo (con ocho ventanas de registros) por ramal. Los cuatro ramales comparten instrucciones y cachés de datos.

La tubería principal se compone de seis etapas: Fetch, Switch, Decode, Run, Memory y Rewriter. Como se muestra en la Figura 2.1.9-1, la etapa interruptor contiene un registro de instrucción para cada ramal. Uno de los ramales es escogido como programador de cadena y la instrucción actual para esa rama se emite a la tubería. Si todo esto se hace bien, el hardware obtiene la siguiente instrucción para que la rama se actualice.

Así la instrucción programada es llevada hacia abajo del las demás etapas de la tubería, esto es similar a la ejecución de la instrucción en una máquina RISC.

Unidad de punto flotante: Una sola unidad de punto flotante es compartida por los ocho núcleos físicos del OpenSPARC, y es suficiente para la mayoría de las aplicaciones comerciales.

Caché L2: La caché L2 se deposita en cuatro direcciones con una selección basada en una dirección física.

Controlador DRAM: Se aloja con cada banco para interactuar juntos, así cada memoria debe tener la misma memoria instalada y habilitada.

Entradas y Salidas: El puente de entrada y salida realiza una decodificación en las entradas y salidas y los dirige al bloque interno necesario y apropiado para una interfaz externa (J-Bus o SSI).

Interfaz J-Bus (JBI): Es la interconexión entre OpenSPARC T1 y el subsistema de entrada y salida. Está a 200 MHz y recibe y responde solicitudes y transacciones.

Unidad de prueba de reloj (CTU): Contiene la generación de reloj, reset y circuitos JTAG.

Fusible o tapón electrónico: El fusible electrónico contiene información de configuración electrónicamente grabado desde su fabricación, incluidos el número de serie y la información de hilos disponibles.

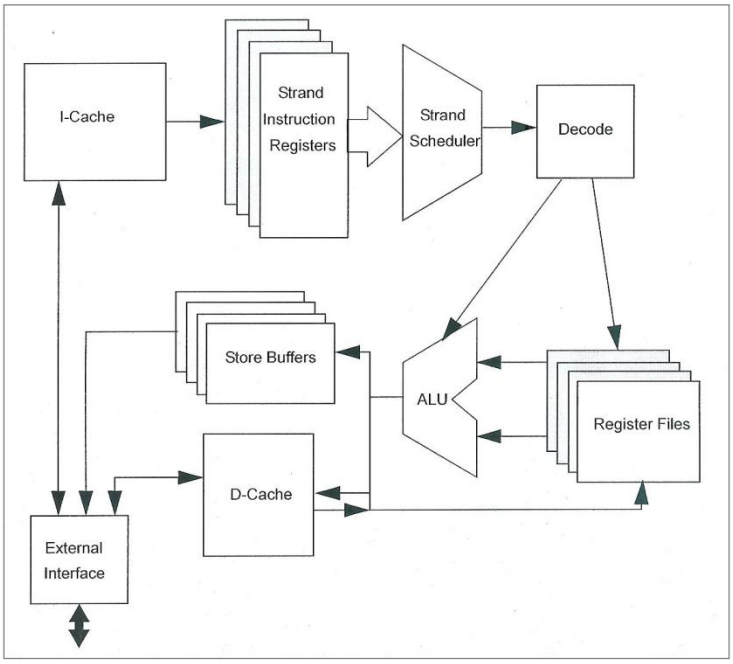


Figura 2.2.9-2 Diagrama de bloques del núcleo OpenSPARC. Cortesía OpenSPARC.

El código del OpenSPARC T1 y posteriores, contiene un compilador para definir y seleccionar entre un núcleo de cuatro hilos (el original) y un núcleo reducido de un hilo. Si se desea por ejemplo, un núcleo de dos hilos, el código puede ser modificado para apoyar esto. Todos los lugares donde el código tiene que cambiar se puede encontrar en el compilador `FPGA_SYN_1THREAD` en todos los archivos RTL y el código puede ser insertado en esos lugares para obtener una configuración de dos hilos.

Características del OpenSPARC T2

El procesador OpenSPARC T2 es un solo chip multiproceso (CMT). También cuenta con ocho núcleos de procesador SPARC. Cada núcleo SPARC tiene un soporte de hardware para ocho procesadores, dos tuberías de ejecución entera, una tubería de ejecución de punto flotante y una tubería de memoria. El punto flotante y tuberías de memoria son comunes a todas las ocho ramas o hilos.

La caché L2 se deposita en ocho maneras para proporcionar el ancho de banda suficiente para los ocho núcleos del OpenSPARC T2, y también se conecta al controlador DRAM. El diagrama de bloques del OpenSPARC T2 es mostrado a continuación. (Figura 2.1.9-3).

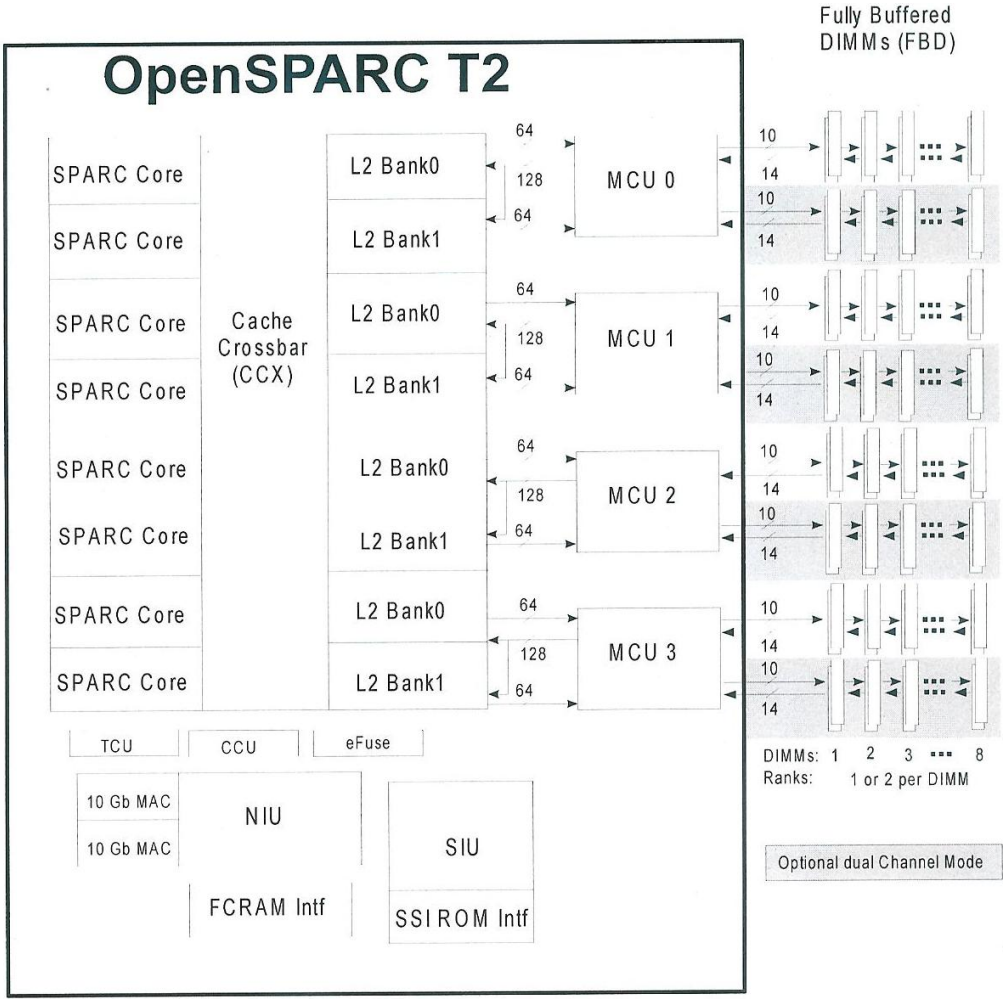


Figura 2.2.9-3 Diagrama de bloques del OpenSPARC T2. Cortesía de OpenSPARC.

Componentes del OpenSPARC T2.

- Núcleo físico.
- Caché L2.
- Unidad de memoria del controlador (MCU).
- Unidad no-caché (NCU).
- Sistema de interfaz (SIU).
- Interfaz SSI ROM (SSI).

Unidad de Memoria del controlador: OpenSPARC T2 tiene cuatro MCU, uno para cada rama de memoria con un par de bancos L2 para interactuar con la rama de DRAM. Las ramas son intercaladas sobre la base de bits con dirección física.

Unidad no caché: Realiza una decodificación de la dirección en la entrada / salida direccionables y los dirige al bloque apropiado. Por otra parte la NCU mantiene el estado de los registros de las interrupciones externas.
Sistema de interfaz: Se conecta a la DMU y a la caché L2. La SIU es el punto de acceso a la caché para el subsistema de red.
Interfaz SSI ROM: OpenSPARC tiene una interfaz de 50 Mb/s que se conecta a un ROM de arranque. Además el SSI admite accesos a través de él, apoyando de esta manera el control y los registros de estado o de otro tipo conectadas en él.

Cambiar el número de núcleos en el OpenSPARC T1 o T2 resulta ser sencillo. El código del OpenSPARC T1, se encuentra un archivo llamado `iop.v` el cual tiene un compilador para definir el número de núcleos del sistema. Estos se encuentran disponibles para apoyar ambientes de tamaño reducido. Por ejemplo, para un diseño con dos núcleos, el compilador deberá cumplir con las siguientes definiciones:

```
// create a chip with two cores
`define RTL_SPARC0
`define RTL_SPARC1
```

Una serie de plataformas de emulación en el mercado permiten una creación de un prototipo del diseño, en la siguiente figura se muestran algunos de ellos junto con sus datos de rendimiento relativo.

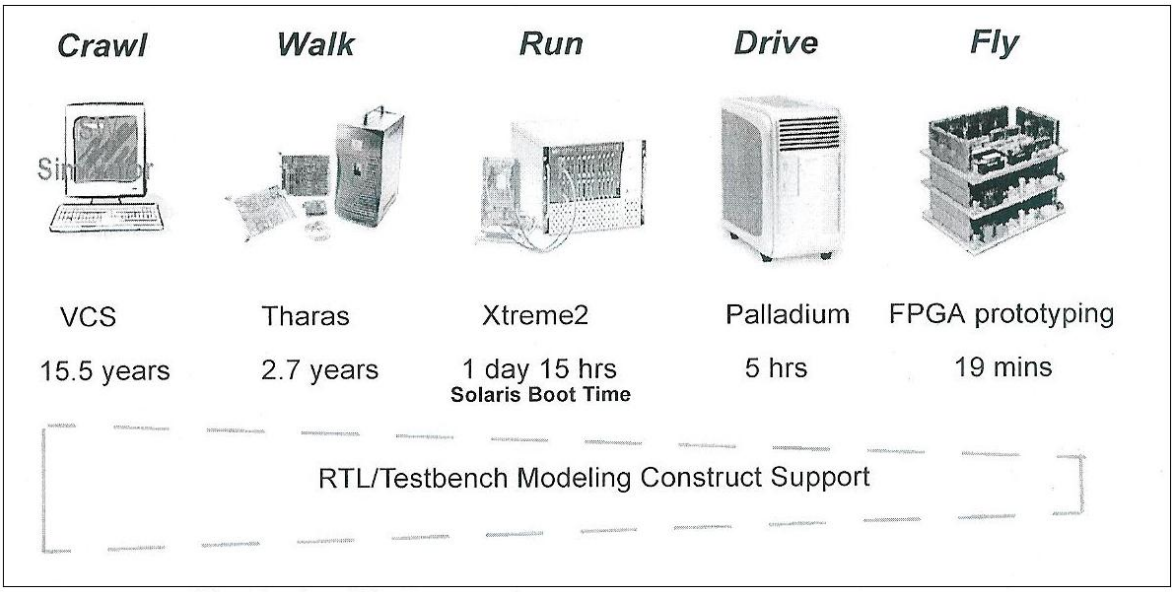


Figura 2.2.9-4 Plataformas de simulación. Cortesía OpenSPARC.

En la figura anterior se muestra el tiempo relativo necesario para arrancar un típico sistema operativo, como Solaris (suponiendo 8 mil millones de ciclos). Muchas

plataformas ilustradas en la figura son caras no pueden ser opción viable para todos los proyectos o prototipos.

2.2.10 TOLERANCIA A FALLAS.

La mayoría de las arquitecturas de procesadores existentes no proporcionan soporte explícito para fiabilidad y tolerancia a fallas. Aunque los diseñadores de sistemas pueden construir una máquina segura y tolerante a fallas, si se proporciona dicho soporte a nivel del procesador se simplifica el diseño y se obtiene un menor costo global del sistema

Para alcanzar este objetivo se ha añadido una instrucción de ``compara y cambia el contexto´´ que tiene unas propiedades de tolerancia a fallas bien conocidas. Así mismo también posee el beneficio añadido de proporcionar un modo eficiente de conseguir sincronización multiprocesador.

La incorporación de múltiples niveles de *traps* anidados permiten a los sistemas recobrarse limpiamente de varias clases de fallas, además de soportar más eficientemente diversos manejadores de interrupciones.

Finalmente, SPARC-V9 incluye un nuevo estado de procesador especial llamado *RED_state (Reset Error and Debug state)*. Su propio nombre define el comportamiento esperado cuando el sistema se enfrenta a errores graves, o durante el proceso de *reset* cuando está volviendo a dar servicio. Así su nivel robusto es una necesidad absoluta cuando construimos sistemas tolerantes a fallas.

2.3 Arquitectura Intel.

Introducción.

Después de varios años sin aportar algo nuevo al mundo de la micro-arquitectura, Intel da un gran paso hacia adelante con la presentación de NetBurst. AMD estaba ganando la partida comercial, con continuos aumentos en la frecuencia de sus procesadores. La barrera de los GHz fue alcanzada en primer lugar por Athlon, e Intel no tardó en responder.

De un gran paso, el Pentium 4 ha dejado atrás al resto de los procesadores CISC, y se puso a la altura de los mejores RISC incluso en el proceso en punto flotante para conseguir una gran mejoría, Intel tuvo que trabajar duro. 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 (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 través del presente punto se presentarán todas esas novedades así como el resto de componentes que conforman el núcleo funcional del procesador.

2.3.1 Visión Global

En la Figura 1 se muestra una visión global de la arquitectura del Pentium 4. Será dividida en cuatro grandes bloques.

El elemento principal del Front-End de esta nueva arquitectura es, 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. La ejecución de las micro-operaciones tiene lugar en diversas unidades funcionales que serán descritas más adelante.

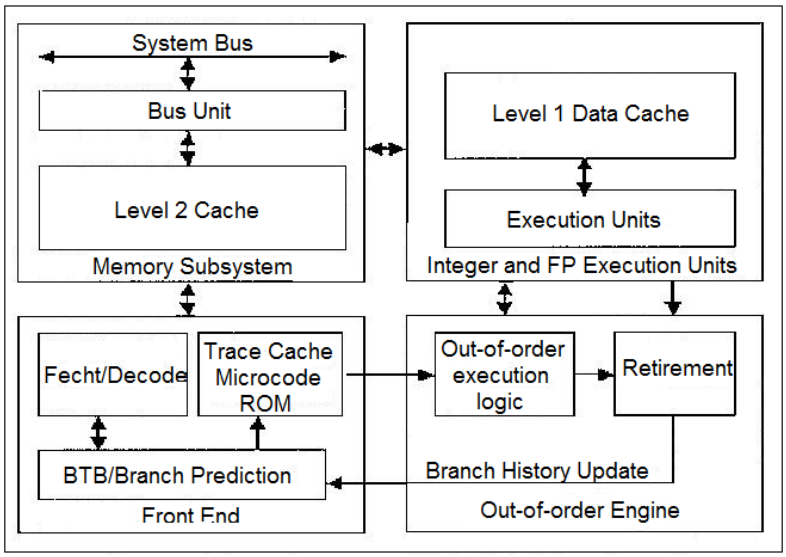


Figura 2.3.1 Arquitectura Global del Pentium 4. Cortesía The Microarchitecture of the Pentium 4 Processor.

Los datos para la ejecución de dichas operaciones se toman de la renovada jerarquía de memoria: una cache de primer nivel de 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.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP		TC Fetch		Drive	Alloc	Rename		Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	Br Ck	Drive

Figura 2.3.1-1 Etapas del Pipeline del Pentium 4.

En la Figura 2.3.1-1 son mostrados los distintos bloques básicos de ejecución que se distribuyen a lo largo de un pipeline profundo: 20 etapas. Intel ha querido aumentar la frecuencia de su procesador, y un modo de conseguirlo es alargar el pipe. Sin embargo, esta decisión conlleva numerosas contrapartidas; y 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 se detecta el error, es muy grande. 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 anterior, se explicarán brevemente cada una de las etapas del pipe

• 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 las 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, para determinar cuáles están listas, decidir cuáles se han de ejecutar. Para ello cuenta con distintos planificadores, en función de la naturaleza de la operación.

• Se llega el momento de lanzar a ejecutar operaciones. Hasta 6 pueden enviarse a las unidades funcionales en cada ciclo.

• 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 etapa 17 se produce la verdadera ejecución de las operaciones. Hay múltiples unidades funcionales con diferentes latencias. Tras ello, se generan las 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. Se puede comprobar que un error en la predicción supone un fuerte costo, pues tal situación no se conoce hasta las últimas etapas de ejecución.

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. Refiriéndonos al *Multithreading,* que consiste en lanzar varios 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.

Cabe señalar también la introducción de l44 instrucciones SSE2 (nueva extensión de las 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 son versiones en 128 bits anteriores.

2.3.2 Front-End

Como se indicó, 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.

2.3.3 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 microoperaciones 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.

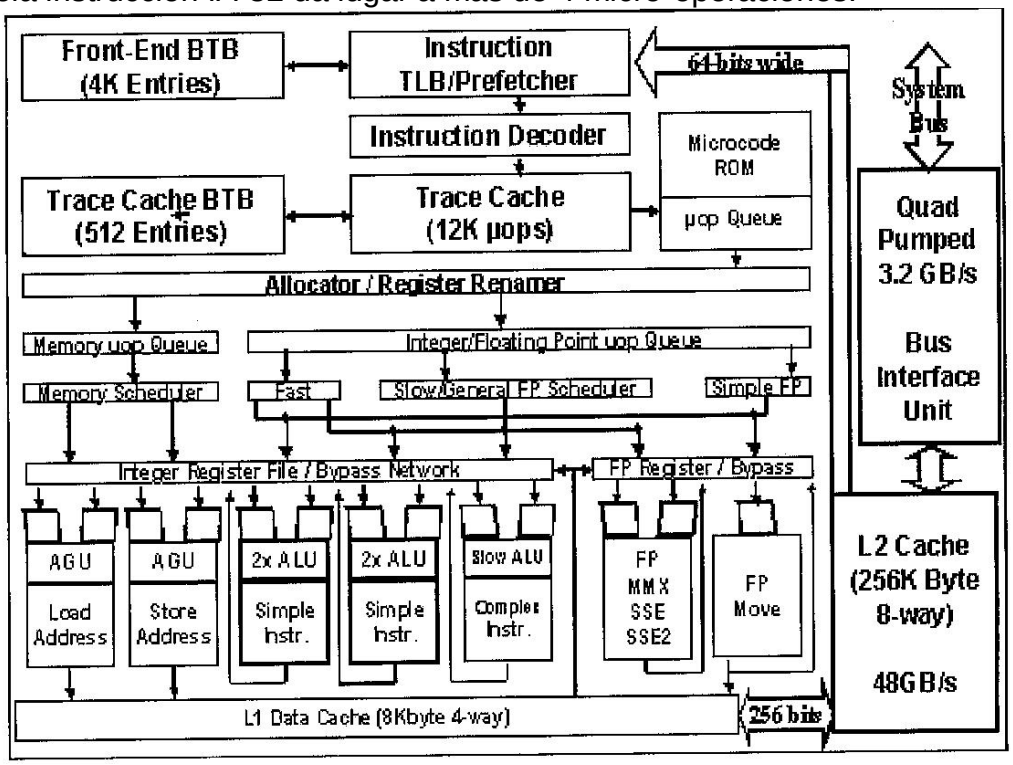


Figura 2.3.3 Microarquitectura del Pentium 4.

Para esta misión, todos los procesadores Intel más modernos disponen de un decodificador capaz de traducir una única instrucción IA-32 por ciclo1. 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.

ITLB

Las instrucciones IA-32 se almacenan 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 físicas 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.

2.3.4 Trace Cache.

Es uno de los puntos más interesantes de esta nueva arquitectura. Después de varios años acarreando la penalización que suponía traducir, durante el camino crítico, las instrucciones IA-32 a micro-operaciones, Intel encontró una solución que dio buenos resultados.

El concepto de Trace Cache fue presentado por primera vez en 1996. La idea básica es la de capturar el comportamiento dinámico de las secuencias de instrucciones, almacenando trazas de instrucciones en lugar de bloques contiguos (seguidos tras una ordenación estática. Ver Figura 2.3.4).

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.

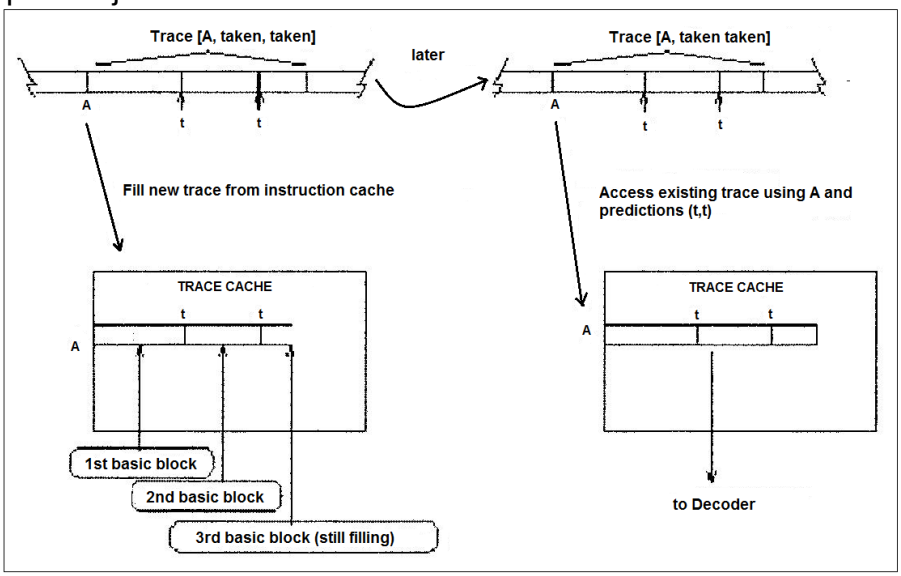


Figura 2.3.4 Ejemplo de funcionamiento de la Trace Cache.

En muchos esquemas, 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, olvidándose 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. Se desconoce 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 (en el supuesto orden de ejecución).
La Trace Cache incluye su propio predictor de saltos, más pequeño que el BTB del Front-End ya que 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.

2.3.5 Predictor de saltos.

Esta podría ser 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 se mostró anteriormente, la dirección efectiva del salto no se conoce hasta las dos últimas etapas del pipe, lo que supone una gran cantidad de trabajo sin necesidad por hacer.
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.

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.

Pila de direcciones de retorno.

Todos los retornos de una función son saltos tomados, pero estos pueden venir desde distintas direcciones, y no se puede 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.

2.3.6 Fase de ejecución.

La nueva arquitectura de Intel introduce 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.

Se puede 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*, 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.

Asignación de recursos.

En la primera fase de la vida de la 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).

Renombramiento de registros.
Intel con 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.

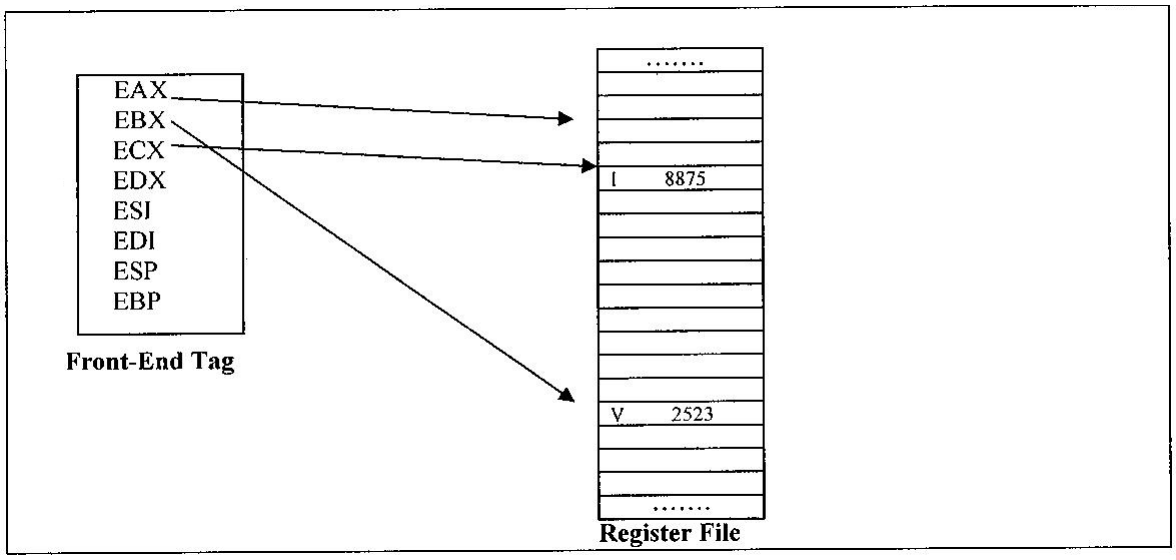


Figura 2.3.6 Situación tras renombrar la instrucción.

En la Figura 2.3.6 se observa 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 tomamos el dato (2523), sino el índice del registro; y se hará 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 (suponer 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. Tomará el índice del registro apuntado (asumiendo 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 se verá 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 2.3.6-1 se muestran las diferencias descritas en este párrafo.
Tras el renombramiento la microoperación 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.

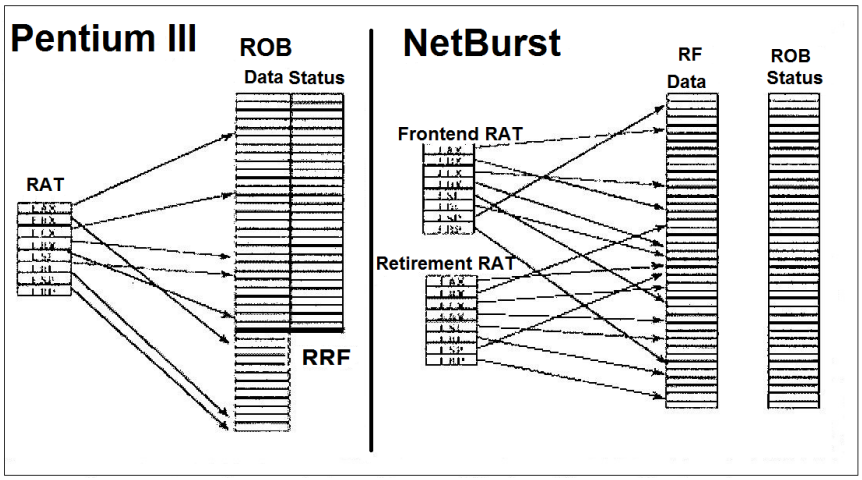


Figura 2.3.6-1 Renombramiento de registros en el Pentium III y en el Pentium 4.

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 microoperaciones 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 microoperación a ejecutar, y sólo dicen que está basado en la "edad" de las operaciones.
Existen varios planificadores en función del tipo de instrucción. Dichos planificadores deciden cuándo las microoperaciones 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 se presentan los aspectos fundamentales de cada una de las unidades funcionales, y se explicará el comportamiento de los "loads" y los "stores".

2.3.7 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.

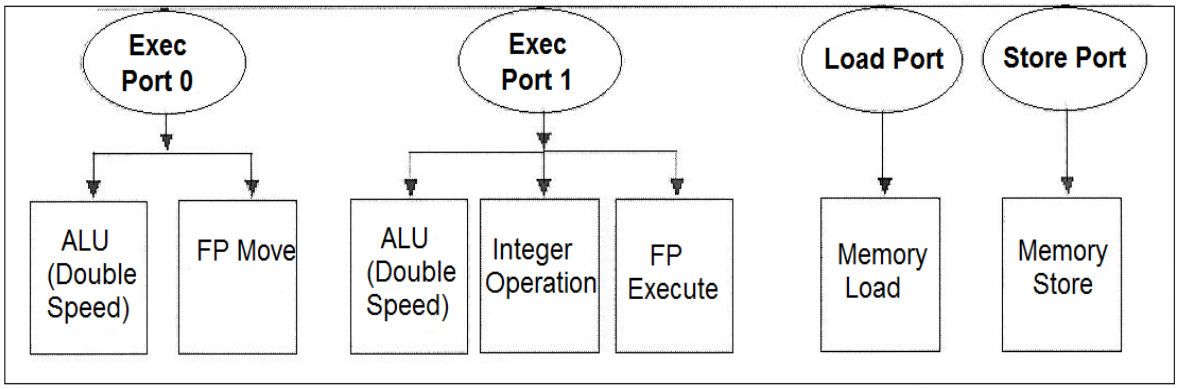


Figura 2.3.7 Puertos de ejecución y unidades funcionales.

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

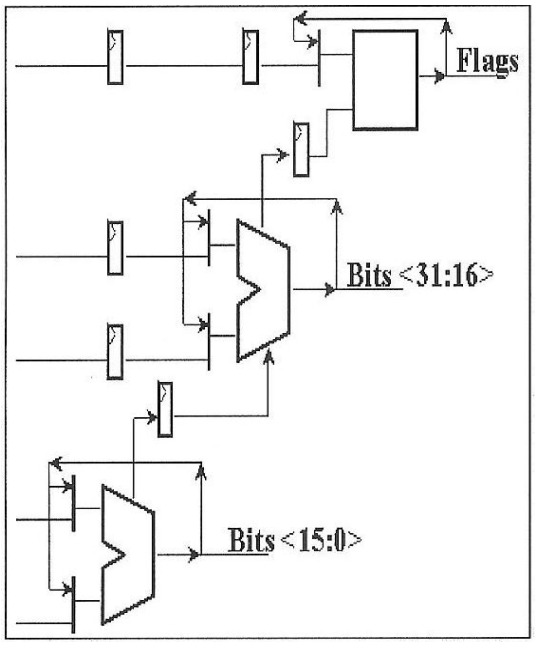


Figura 2.3.7-1 ALU de doble velocidad: staggered add.

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.

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, está 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.

Operaciones SIMD: MMX, SSE y SSE2.

Antes de terminar con la fase de ejecución, se presentan 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 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 en la Figura siguiente se detalla 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 precisión 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.

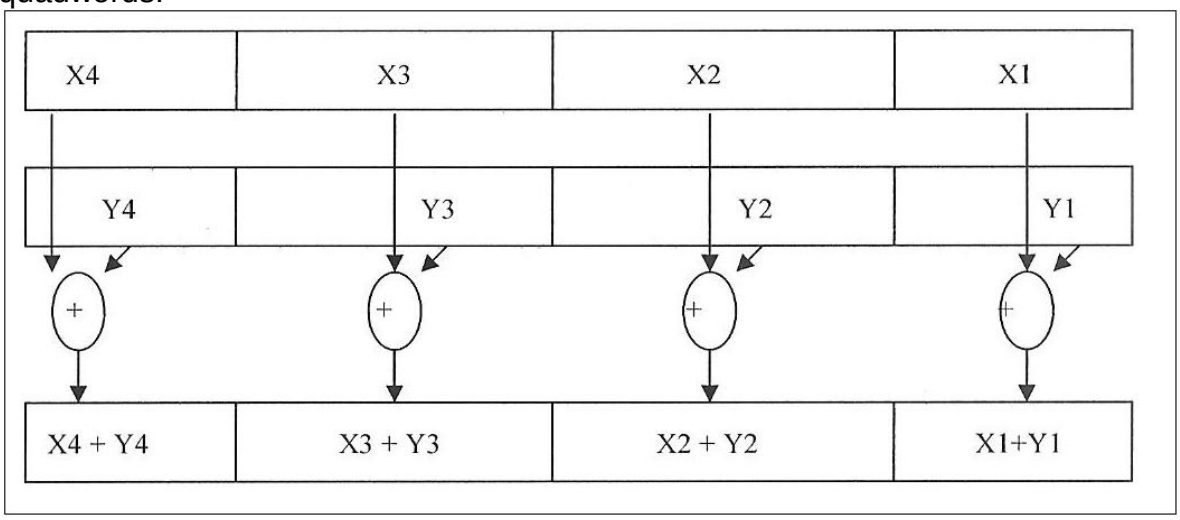


Figura 2.3.7-2 Suma SIMD.

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.

2.3.8 Finalización de instrucciones.

Nuestra ya casi olvidada micro-operación (EAX EBX + ECX) abandonó en su turno la cola de microoperaciones, 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 microoperación, 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-operaciones 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.

2.3.9 Acceso a memoria.

Hasta este momento, hemos pasado un poco por encima de las operaciones de lectura y escritura a memoria. Se ha 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 (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, se puede saber 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 2.3.3 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 microoperación 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.

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 se detallaron los 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	256KB	8	128	7/7	Write Back

Tabla 2.3 Parámetros de los distintos niveles de Cache.

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).

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 traducción4). Si se produce un fallo en la cache, ya se dispone de la dirección física para realizar el acceso al segundo nivel.

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.

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. 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.

Optimizaciones del sistema de memoria.
El Pentium 4 presenta un gran número de optimizaciones para el mejor funcionamiento de su sistema de memoria. 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)

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 contrarrestados por la merma que supondrá la espera por el control del bus.

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.

Ejecución especulativa de loads.

Debido a la profundidad del pipe, se hace necesario asumir que las lecturas en cache no fallarán y planificar las microoperaciones 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.

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 un 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.

2.3.10 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.

2.4 MOTOROLA

Familia 68000.

Esta familia de procesadores incluye varios procesadores que ofrecen diferentes niveles de desempeño. Todos los miembros de la familia tienen la misma arquitectura básica, pero los dispositivos más recientes tienen características adicionales que mejoran su rendimiento.

Registros y Direccionamiento.

El procesador 68000 es caracterizado por una longitud de palabra de 16 bits porque el chip del procesador tiene 16 pines para la conexión de memoria. Los datos son manipulados dentro del procesador en registros que contienen 32 bits. Los más avanzados modelos de esta familia son los procesadores 68030 y 68040, los cuales son encapsulados de 32 pines de datos. Por lo tanto pueden hacer frente a los datos tanto interna como externamente en cantidades de 32 bits.

La estructura de Registro 68000.

La estructura de Registro de la familia 68000 mostrada en la figura 1 tiene 8 registros de datos y 8 registros de direcciones cada uno de 32 bits de longitud. Los registros de datos sirven de uso general como acumuladores y como contadores.

Características del microprocesador Motorola 68000.

Es un microprocesador microprogramado con una estructura interna de 32 bits, aunque externamente (a efectos de comunicación), posee un bus de sólo 16 bits. Para disminuir el tamaño de la memoria de control, se ha implementado, a nivel de programación, una estructura de dos niveles, microprogramación y nano programación. Posee varias ayudas para la programación un modo de depuración y otro de ejecución paso a paso. Posee una arquitectura modular, por lo que se le pueden implementar nuevas prestaciones (de aquí sale el resto de la familia). Posee ocho registros de datos de 32 bits, direccionables como 8 ó 16, que se utilizan para obtener datos o como registros índice, siete registros de direcciones de 32 bits, para el cálculo de la dirección del dato, punteros a pila, registros base e índice. También posee un registro doble que apunta a la pila hardware del microprocesador.

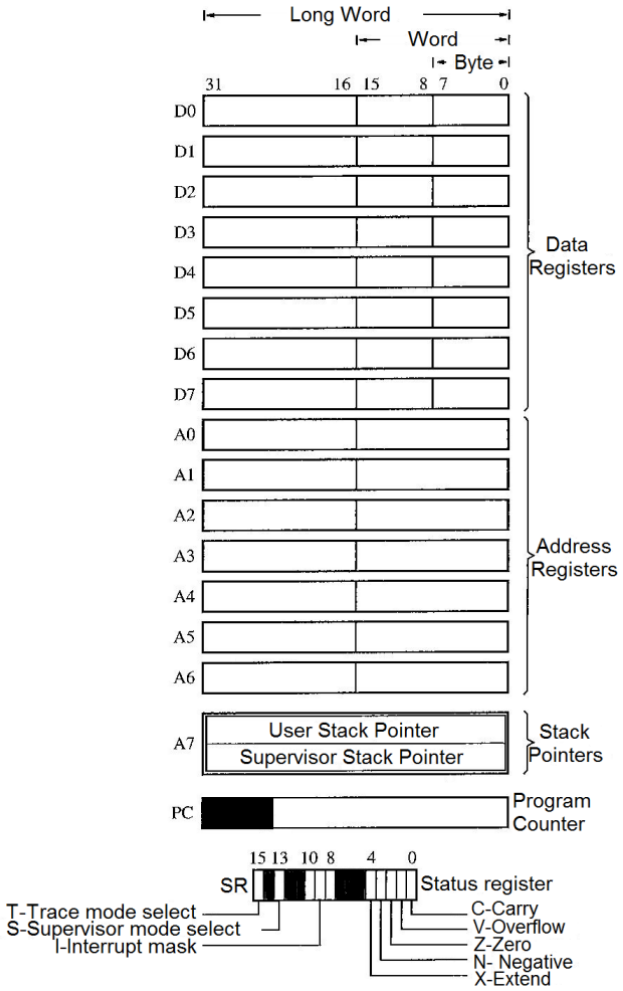
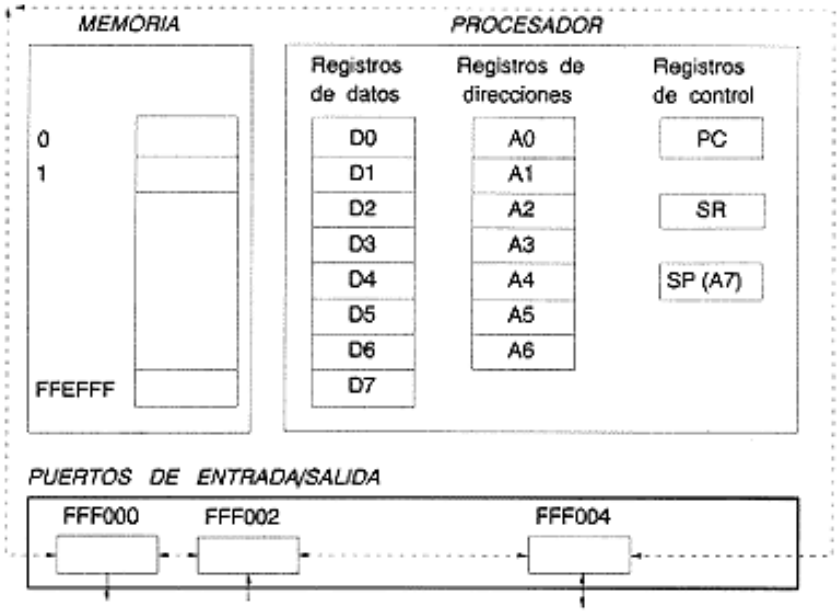


Figura 2.4-1 Estructura de Registros de la familia 68000. Cortesía Computer Organization.

El contador del programa tiene una longitud de 24 bits, pero en los últimos modelos, 68030 y 68040 tienen 32 bits. Posee un total de 61 instrucciones básicas; aunque parezcan pocas comparándolas con otros microprocesadores, no es así, puesto que entre otras razones, muchas de ellas son multifuncionales; además de poder ampliarse o modificarse el repertorio de instrucciones, actuando a nivel de microprograma o causando desviaciones al detectar los diferentes códigos de operación. Posee facilidades para la multiprogramación y existen dos modos de funcionamiento, normal y supervisor. Con las 24 líneas de direccionamiento de memoria, tiene una capacidad de direccionamiento de 16 Mb lineales sin segmentación, mayor en las versiones 68030 y 68040. Las interrupciones son vectorizadas, existiendo ocho niveles de prioridad.

A mediados de los años 70 Motorola comienza el diseño de un microprocesador de 16bit (proyecto que se conoció como MACSS: Motorola´s Advanced Computer System on Silicon) debiendo éste ser fácil de programar y capaz de aprovecharse del mercado existente de su antecesor de 8bit, el MC6800.



Así nace en 1979 el MC68000, un procesador de 16bit con registros de 32bit que intercambia data de E/S en formatos de 8, 16 y 32 bit y opera con un reloj a 8Mhz.

Primero conviene destacar las diferencias entre un procesador de 16bit con los de 8. Esto sobre todo porque el MC68000 viene a ser uno de los primeros procesadores de 16bit comerciales y tendrá que batirse con un mercado marcado por la tecnología en 8bit (como son los procesadores Z80, 6502, Intel 8086, MC6800).

En las tablas siguientes se muestra la codificación de un comando (op-code) para los procesadores MC6800 y MC68000. La mayor flexibilidad y también complejidad del caso 16bit se nota en la cantidad de registros accesibles, los diversos modos de direccionamiento y finalmente la capacidad de computar operandos de largo variable (8,16 y 32bit).

Op-Code MC6800

1	**0**	**1**	**0**	**1**	**1**	**0**	**1**
	Registro (2)	**Modo de direccionamiento (4)**		**Operación (12)**			

MC68000

1	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**
Operación (16)				**Registro dato (8)**			**a/de memoria (3)**	**Tamaño operando (3)**		**Dirección efectiva (8 registros 12 modalidades)**					

(Los números entre corchetes indican la cantidad de posibilidades validas de la "variable" respectiva)

El resultado de esto es una cantidad abismante de instrucciones (más de 10000) lo que hace de este procesador un claro representante de la tecnología CISC.

Registros 68000.

El MC68000 consta de 16 registros de propósito general. Primero cabe destacar los 8 registros de dirección (A0 - A7) de 32bit lo cual permite un manejo lineal de la memoria, de hecho se dispone de solo 24bit en forma cómoda, resultando accesibles 16MB directamente. Esto debía solucionar el problema que habían enfrentado los procesadores de 8bit con técnicas engorrosas como (paging/segmenting).

Para facilitar el intercambio, los 8 registros de datos (D0-D7) también se eligieron de 32 bits. Lo cual hace diferencia con sus antecesores por definir los registros con la misma funcionalidad. Además tiene dos punteros de pila (stack-pointers) de 16bit, uno para el usuario y otro de sistema (USP/SSP).

La instrucción actual señala el program counter (PC) que naturalmente es de 32 bits. Finalmente cabe mencionar el registro de flags de 16 bit para completar los registros internos del Mc68000.

Unidades Aritméticas 68000.

No debe olvidarse que se trata también de un chip de 16 bit, y se puede notar en el tamaño del bus de datos (16bit) y también por el hecho que la instrucción op-code se codifique en 16bit.

Sin embargo, al incluir a parte de el ALU principal dos unidades aritméticas exclusivamente dedicadas a calcular direcciones resulta un poder de cálculo de 48bit en paralelo, 16bit de datos y 32 para la dirección.

Destacable 68000.

Implementa un sistema de colas (prefetch queue) que adelanta la obtención de instrucciones para su más inmediato procesamiento por parte de la CPU, método conocido también como pipeline.

El MC68000 fue uno de los primeros procesadores desarrollados con la tecnología de microprogramas, facilitando su diseño gracias a la modularidad del método.

El 68000 ordena los bytes de datos partiendo con el byte menos significativo “leas significant byte first (LSB)”, lo cual resulta más natural porque el ordenamiento de un byte en sí es LSB.

Modos de direccionamiento 68000.

El 68000 es una máquina CISC y tiene 12 modos de direccionamiento. Estos pueden ser clasificados en 6 grupos.

1. Register Direct.
2. Address Register Indirect.
3. Absolute Data Register.
4. Program Counter Relative.
5. Inmediate Data.
6. Implied Addressing.

Procesador 68020.

68020 es un verdadero procesador de 32 bit y este es "object code" compatible con el 68000. Tiene muchos más registros. El PC es un verdadero registro de 32 bits y puede direccionar sobre 4GB de espacio de memoria. Hay nuevas instrucciones y nuevos modos de direccionamiento.

Procesador 68030.

68030 es un procesador de memoria virtual basada en 68020. Este 68030 tiene una unidad administradora de memoria sobre el chip, lo cual mejora la administración de memoria con datos compaginados. Hay 4 nuevas instrucciones para la parte MMU del procesador. Además tiene un cache de datos sobre el chip de 128 palabras de tamaño junto a la instrucción del cache.

Procesador 68040.

68040 es una mejora implementada al 68030. Tiene caches de instrucciones y datos más grandes. Además tiene una unidad de punto flotante cobre el chip.

2.4.1.- HARDWARE MOTOROLA 68000.

1.1 Memoria Familia 68000:

La memoria principal de esta computadora está formada por celdas de un byte (8bits), que constituyen la unidad básica de lectura o escritura, identificándose mediante una dirección.

Los procesos de lectura y escritura pueden realizarse con varias celdas consecutivas simultáneamente, debiendo indicar el procesador a la memoria principal dos parámetros, la dirección de la primera celda de memoria y la longitud de la información a la que se desea acceder. Siendo esta longitud de un byte, dos bytes (una palabra) o cuatro bytes (palabra larga).

El tamaño máximo de la memoria está determinado por el número de bits de los registros de direcciones que tiene el procesador, siendo en el caso del Motorola 68000 de 32 bits pero, debido a limitaciones en el montaje solo pueden utilizarse 24 como máximo, así que la máxima longitud que se puede usar de la memoria principal es de 2^{24} bytes, desde 0 hasta FFFFFF.

El procesador puede leer y escribir información de diferentes tamaños, existiendo una norma para almacenar las palabras (W) y las palabras largas (L), y se comienza con el byte más significativo.

Existen 7 registros de direcciones y son: A0, A1, A2, A3, A4, A5 y A6 y tales registros son de 32 bits, pero sólo se pueden utilizar 24 bits para direccionar.

1.2 Registros de Datos:

El Motorola 68000 consta de 8 registros de datos, que son D0, D1, D2, D3, D4, D5, D6 y D7. Cada uno consta de 32 bits. En muchas instrucciones existe la posibilidad de especificar el tamaño del dato, indicándose este mediante el sufijo S (B, W y L), que va añadido al nemotécnico de la instrucción. La forma en que se almacenan los datos en los registros, viene dada por su longitud, ya que como

esta es variable, irán ocupándolos de izquierda a derecha empezando por el bit menos significativo del registro.

1.3 Modos de Direccionamiento:
Existen cuatro modos de direccionamiento:

i) Direccionamiento inmediato: almacena el operando precedido del símbolo # en el registro indicado. Ejemplo: MOVE.L#$18,D6.

ii) Direccionamiento absoluto: almacena el operando que está en la dirección de memoria especificada en el registro de datos indicado. Ejemplo: ADD.W%0000000010000111110001,D2 suma la palabra que está en la dirección de memoria indicada a D2.

iii) Direccionamiento mediante registro: apunta a la dirección del registro donde está el dato. Ejemplo: MOVE.B D3,D4 copia el contenido del registro D3 (byte) a D4.

iv) Direccionamiento relativo a registro:

a) Direccionamiento mediante registro normal: se da la dirección del registro donde está la dirección del dato. El nombre del registro se escribe entre paréntesis.
Ejemplo: ADD.B (A0),D6 suma el contenido de la posición de memoria (byte) cuya dirección está en A0 al registro D6, guardando el resultado en este último.
b) Direccionamiento relativo a registro con posincremento: incrementa en una cantidad de memoria, según sea el tamaño del operando (1 para B, 2 para W y 4 para L), después de traer el contenido de la posición de memoria indicada por el registro de direcciones. Ejemplo: MOVE.W(A0)+,D0 copia en D0 el contenido de la posición de memoria direccionada por A0 y luego incrementa en 2 el contenido de A0.
c) Direccionamiento relativo a registro con predecremento: Decrementa en una cantidad de memoria, según sea el tamaño del operando, el registro de direcciones y trae despues el contenido de la posición de memoria cuya dirección es el nuevo valor de dicho registro. Ejemplo: MOVE.B -(A0),D0 decrementa en uno el contenido del registro A0 y luego copia en D0 el contenido de la nueva posición de memoria direccionada por A0.
d) Direccionamiento relativo a registro con desplazamiento: El contenido de la posición de memoria cuya dirección viene dada por la suma del valor del registro de direcciones y una cantidad fija denominada desplazamiento, pudiendo ser este positivo o negativo y su valor viene condicionado por el tamaño del operando. Ejemplo: MOVE.L N(A0),D1 copia en el registro D1 el contenido de la posición de memoria cuya dirección viene dada por la suma de N multiplicado por 4(L) al contenido de A0.

e) Direccionamiento relativo a registro con índice: Este modo de direccionamiento es la extensión natural del anterior, ya que permite usar desplazamientos variables, utilizando como desplazamiento el resultado de sumar un número fijo al contenido de un registro de datos denominado registro índice. Ejemplo: MOVE.B 4(A0,D1), D0 copia en el registro D0 el contenido de la posición de memoria cuya dirección es el resultado de sumar el número 4, el contenido del registro A0 y el contenido del registro D1. Este modo de direccionamiento no altera el registro de direcciones ni el registro índice.

f) Direccionamiento relativo al contador de programa con desplazamiento: Cuando es necesario hacer referencia a un operando relativo a la posición de la próxima instrucción que va a ser ejecutada. Ejemplo: MOVE.B 24(PC),D0 copia en el registro D0 el contenido de la posición de memoria cuya dirección es la suma de 24 y el valor del contador del programa.

g) Direccionamiento relativo al contador de programa con índice: Utiliza como desplazamiento el resultado de sumar un número fijo al contenido de un registro de datos. Ejemplo: MOVE.B 24(PC,D0),D1 copia en el registro D1 el contenido de la posición de memoria cuya dirección es el resultado de sumar el número 24, el contador de programa y el contenido del registro D0.

1.4 Formato de instrucciones:

Los formatos empleados para las instrucciones utilizan una o más palabras de 16 bits. La primera palabra especifica el código de la operación y en muchos casos la dirección de un operando. Las especificaciones para completar los operandos, cuando no es suficiente con una palabra van a continuación de la primera palabra.

Cada operando utilizará como máximo dos palabras de ampliación, equivalente a una palabra larga, después de la del código de operación, por lo que la instrucción más larga del Motorola 68000 ocupa 5 palabras (10 bytes), siendo 1 para el código de instrucción, 2 palabras de ampliación para el operando origen y 2 palabras de ampliación más para el operando destino.

La información que identifica la situación exacta del operando, denominada dirección efectiva, se codifica en los formatos de instrucción mediante dos campos, siendo uno el modo de direccionamiento (MD) y el otro el de registro. Cada campo tiene un tamaño de 3 bits y se incluyen en la primera palabra de instrucción. El campo MD identifica el modo de direccionamiento empleado y el campo CR indica el registro empleado para obtener la dirección del operando. A veces se utilizan las palabras de ampliación ya que la dirección efectiva requiere incluir más información sobre la situación del operando en la palabra de instrucción.

1.5 Instrucciones condicionales:

Al realizar operaciones matemáticas existe la posibilidad de desbordamiento, por lo que se realiza un test automáticamente, quedando el resultado almacenado en un registro de control, dedicado especialmente a tal efecto, denominado registro de código de condición (CCR), pudiéndose leer mediante la instrucción MOVE.

Este registro consta de 5 flags que se almacenan en los 5 bits menos significativos del registro de estado (SR). Estos 5 bits tienen las siguientes denominaciones, ordenadas desde el bit menos significativo:

- **C** indicador de acarreo o "carry" flag: indica el valor del bit de acarreo de la posición más significativa del resultado de una operación, poniéndose a 1 si existe desbordamiento.
- **V** indicador de desbordamiento o overflow flag: indica si en el resultado de una operación en complemento a 2 existe desbordamiento, poniéndose a 1.
- **Z** es el indicador de cero o "zero" flag, poniéndose a 1 cuando sea 0 el resultado de una operación aritmética o lógica.
- **N** es el indicador de número negativo o "negative" flag, poniéndose a 0 si es positivo y a 1 si es negativo el signo del resultado de una operación en complemento a 2.
- **X** es el indicador extendido o extended flag que funciona de la misma manera que C, pero únicamente con operaciones aritméticas o de desplazamiento.

1.6 Entrada/Salida:

Los computadores disponen de unos registros especiales, denominados puertos de E/S o I/O en inglés para poder comunicarse con el exterior. El Motorola 68000 consta de tres puertos de este tipo en las siguientes direcciones de memoria:

-FFF000 de E.

-FFF002 de S.

-FFF004 de E/S.

1.7 Gestión de Subrutinas:

Una subrutina es un conjunto de instrucciones que realizan una tarea concreta, que no puede ser ejecutada directamente sino que debe ser llamada por un programa principal. Esta subrutina denominada también subprograma, procedimiento o simplemente rutina se comienza a ejecutar cuando es llamada por el programa principal, desde la primera instrucción. Cuando se ha llegado a la última instrucción, se vuelve a ejecutar la siguiente instrucción del programa principal anterior a la invocación de la subrutina.

La gestión de las subrutinas se realiza mediante una estructura de almacenamiento de tipo de pila. El espacio en memoria que se reserva para la pila se define a partir de dos direcciones, la dirección de la posición inicial o fondo de la pila y la dirección de la posición máxima permitida o valor máximo que puede alcanzar la cima de la pila. El puntero de pila indica la posición de la cima de la pila, utilizando un registro de direcciones denominado SP (stack pointer) y que es el A7.

2.4.2.- INSTRUCCIONES:

2.1 Instrucciones de transferencias de datos.

El conjunto de estas instrucciones permite el movimiento de datos entre registros de la CPU, entre registros y memoria y entre posiciones de memoria. Estas son las siguientes:

MOVE: realiza la transferencia de un dato desde fuente a destino. Modifica el CCR de forma que refleja el signo del dato movido y el hecho de que sea cero o no. Los flags C y V se ponen a cero y el flag X no se modifica. También el manejo de la pila se lleva a cabo mediante esta instrucción, utilizando direccionamiento indirecto con el registro A7 que actúa como puntero de pila. Para llevar un dato a la pila el direccionamiento es indirecto con predecremento, mientras que para extraerlo se emplea el indirecto con postincremento.

MOVEA: (Move Adress) es una variante dedicada a la transferencia de direcciones. El tamaño del dato es de 16 ó 32 bits. No modifica el CCR.

MOVEQ: (Move Quick) tiene por finalidad la carga rápida de un registro de datos.

MOVEM: (Move Múltiple) carga una serie de posiciones consecutivas de memoria en varios registros a la vez. No afecta al CCR. Ejemplo: MOVEM.L $1000,D0-D2/A2-A4/D7 copia los contenidos de las posiciones $1000, $1002, $1004,...$100C en los registros D0, D1, D2, D7, A2,A3 y A4 respectivamente.

EXG y SWAP: la primera intercambia el contenido completo de dos registros de datos o de dirección, mientras que la segunda actúa sobre un único registro, siempre de datos, intercambiando sus palabras alta y baja. La realización de estas transferencias sin la existencia de estas instrucciones, requeriría la ejecución de tres instrucciones MOVE, y la utilización de otro registro o de la memoria como almacenamiento temporal.

LEA y PEA: (Load Effective Adress y Push Effective Adress) determinan la dirección efectiva del operando fuente. La primera almacena la dirección efectiva calculada en el registro de direcciones que se especifique como operando destino y la segunda lo lleva a la pila. Ninguna de las dos afecta al CCR.

LINK y UNLINK: facilitan las operaciones necesarias para el paso de parámetros a/de subrutinas a través de la pila, dando la ventaja de ser independiente de las áreas de memoria de datos de un programa, y de no necesitar el uso de etiquetas ni de direcciones concretas para acceder a dichos parámetros, dando la facilidad de la inclusión de las subrutinas en programas diferentes sin necesidad de hacer cambios en ellas.

La instrucción LINK reserva una zona de memoria en la pila, desplazando el puntero de pila (SP) hacia direcciones menores en el valor que se indique. La instrucción UNLINK restablece la pila en la situación que se encontraba antes

de la ejecución de LINK, cargando el puntero de pila con el contenido del registro de direcciones que se indica y a continuación extrae la palabra larga apuntada por SP y la lleva al registro de direcciones, quedando así restaurado.

2.2 Instrucciones aritméticas:

El Motorola 68000 dispone de instrucciones para las 4 operaciones aritméticas, sobre operandos binarios, y suma y resta sobre datos codificados en BCD. Además de cambio de signo para ambos tipo de datos, instrucciones de comparación, extensión de signo y actualización de los códigos de condición (CCR) según el valor de un dato.

ADD: realiza la suma de los operandos fuente y destino, quedando el resultado almacenado en destino. Modifica el CCR en función del resultado de la operación.

ADDA: (Add Adress) realiza la operación de la suma, siendo el destino un registro de direcciones en vez de datos.

ADDI: (Add Inmediate) realiza la operación suma mediante direccionamiento inmediato.

ADDQ: (Add Quick) realiza la operación resta mediante direccionamiento inmediato siendo el destino menor o igual que 8.

ADDX: (Add Extended) incluye en el resultado la suma del flag X, lo que facilita las operaciones con valores que superan la capacidad de los registros (precisión múltiple).

SUB: (Substract) realiza la diferencia entre fuente y destino, depositando el resultado en destino.

SUBA: (Substract Adress) realiza la operación de la resta, siendo el destino un registro de direcciones en vez de datos.

SUBI: (Substract Inmediate) realiza la operación resta mediante direccionamiento inmediato.

SUBQ: (Substract Quick) realiza la operación resta mediante direccionamiento inmediato siendo el destino menor o igual que 8.

SUBX: (Substract Extended) incluye en el resultado la resta del flag X, lo que facilita las operaciones con valores que superan la capacidad de los registros (precisión múltiple).

MULS y MULU: (Multiply Signed y Multiply Unsigned) realizan la multiplicación sobre datos de 16 bits, generando un resultado de 32 bits, siendo el destino un registro de datos. Los flags C y V siempre quedan a 0 y el flag X no se modifica, reflejando el resultado de la operación los flags Z y N.

DIVS y DIVU: (Signed y Unsigned) realizan la división de un dato de 32 bits (en destino) por otro de 16 bits (en fuente), generando dos resultados de 16 bits: el cociente y el resto, siendo obligatorio que el destino sea un registro de datos. El cociente se guarda en los 16 bits menos significativos y el resto en los 16 bits más significativos del registro destino. Ambas instrucciones ponen el flag C a 0, y el flag X no se modifica. Puede producirse desbordamiento si el divisor es pequeño, reflejándose en el flag V. Los flags N y Z son modificados reflejando el resultado de la operación.

CMP: (Compara) las instrucciones de comparación efectúan la substracción destino-fuente pero la diferencia no se lleva a ningún destino, limitándose a afectar a los flags N, Z, V y C, de forma que se puedan comprobar diversas relaciones entre los datos que se comparen con el fin de producir ramificaciones en el programa.

CMPA: (Compare Adress) utiliza como destino un registro de direcciones.

CMPI: (Compare Inmediate) utiliza direccionamiento inmediato para fuente. Para el operando destino puede utilizar todos los modos salvo direccionamiento directo a registro de direcciones, indirectos con predecremento o posincremento y los relativos a PC.

CMPM: (Compare Memory) compara dos datos en memoria con direccionamiento indirecto con posincremento facilitando así la comparación de bloques de memoria.

CLR: (Clear) carga un cero binario en el operando destino poniendo los flags N a 0, Z a 1, V a 0 y C a 0.

NEG: (Negate) esta instrucción devuelve el complemento a 2 del operando y esto supone el cambio aritmético de signo. Afecta a todos los flags del CCR.

NEGX: (Negate Extended) facilita el cambio de signo de valores de precisión múltiple.

EXT: (Extend) cuando se quiere ampliar la longitud de un dato con signo, por ejemplo de byte a palabra o de palabra a palabra larga, sin alterar su valor, el dato original debe mantenerse en el byte o palabra de menor peso, y el byte o palabra que se añade debe tener todos sus bits con el mismo valor que el bit de signo del dato original, recibiendo esta operación el nombre de extensión de signo.

2.3 Instrucciones lógicas:

El Motorola 68000 dispone de cuatro instrucciones que realizan funciones lógicas, que actúan bit a bit sobre datos de 8, 16, ó 32 bits y cuatro para manejar bits individuales sobre datos de 8 ó 32 bits.

AND: es la Y lógica. Admite todos los modos de direccionamiento para el operando fuente menos direccionamiento directo a registro de direcciones, mientras que el operando destino admite también todos menos los direccionamientos relativos a PC.

ANDI: realiza la misma función que AND, pero su direccionamiento es inmediato en el operando fuente y todos menos el direccionamiento directo a registro de direcciones y relativos a PC.

EOR: es el O exclusivo. Admite únicamente el direccionamiento directo a registro de datos para el operando fuente, mientras que el operando destino admite también todos menos los direccionamientos directo a registro de direcciones y los relativos a PC.

EORI: realiza la misma función que EOR, pero su direccionamiento es inmediato en el operando fuente y todos menos el direccionamiento directo a registro de direcciones y relativos a PC.

NOT: complementación lógica.

OR: es la O lógica. Admite todos los modos de direccionamiento para el operando fuente menos direccionamiento directo a registro de direcciones, mientras que el operando destino admite también todos menos los direccionamientos relativos a PC y directo a registro de direcciones.
ORI: realiza la misma función que OR, pero su direccionamiento es inmediato en el operando fuente y todos menos el direccionamiento directo a registro de direcciones y relativos a PC.
TST: (Test) comprueba un operando. Los flags V y C se ponen a 0.
SCC : comprueba los códigos de condición y pone a 1 el operando. Es de tamaño byte.

2.4 Instrucciones de desplazamiento y rotación:

Se caracterizan por desplazar o rotar el operando bit a bit a la derecha o a la izquierda. El operando destino, que es el afectado por el desplazamiento o por la rotación siempre será un registro de datos.
ASL: (Arithmetic Shift Left) desplaza a la izquierda los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag V se pone a 1 si el bit más significativo cambia en algún momento y C es el valor del último bit desplazado fuera del operando destino.
ASR: (Arithmetic Shift Right) desplaza a la derecha los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag V se pone a 0, y el C es el valor del último bit desplazado fuera del operando destino.
LSL: (Logical Shift Left) es el desplazamiento lógico a la izquierda. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.
LSR: (Logical Shift Right) es el desplazamiento lógico a la derecha. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.
ROL: (Rotate Left) rota a la izquierda los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.
ROR: (Rotate Right) rota a la derecha los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.
ROXL: (Rotate with Extended Left) rotación a la izquierda con extensión. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.
ROXR: (Rotate with Extended Right) rotación a la derecha con extensión. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.
SWAP: intercambia el contenido de los 16 bits más significativos con el de los 16 menos significativos, ya que los operandos son de tamaño palabra. Los flags V y C se ponen a 0.

2.5 Instrucciones de manipulación de bits:

El Motorola 68000 permite comprobar, poner a cero, poner a uno e invertir los bits individuales de un valor entero.

BTST: (Bit Test) sirve para comprobar el estado de un bit concreto de destino. Actualiza el flag Z en función del valor del bit indicado en la instrucción (Z=1 si el bit es cero). Admite el direccionamiento inmediato y el directo a registro de datos en el operando fuente y en el operando destino todos menos el directo a registro de direcciones.

BCLR: (Bit Clear) pone a 0 el bit indicado. Actualiza el flag Z en función del valor original del mismo, poniendo posteriormente a cero.

BSET: (Bit Set) igual que BCLR, pero poniendo el bit a 1.

BCHG: (Bit Change) en primer lugar actualiza el flag Z en función del valor del bit indicado y luego complementa el bit, es decir lo pone en el valor (0 ó 1) contrario al original.

2.6 Instrucciones de operación en código BCD:

De la misma manera que el Motorola 68000 opera con enteros, también permite programar la suma y la resta en código BCD.

ABCD: suma fuente al destino.

NBCD: niega el destino.

SBCD: resta fuente al destino.

2.7 Instrucciones de ramificación y salto:

Este microprocesador incorpora varios mecanismos para poder realizar instrucciones típicas de los lenguajes de alto nivel, como pueden ser los bucles o las instrucciones de condición, siendo el más elemental la instrucción de ramificación, que utiliza como operando una etiqueta, y que sirve para hacer que la próxima instrucción que se ejecute sea la que tenga dicha etiqueta. Cuando el procesador se encuentra con esta instrucción, sencillamente carga la etiqueta en el contador de programa, por lo que la etiqueta es la dirección de donde debe cargarse la próxima instrucción que vaya a ejecutarse.

Bcc: (Branch on condition code) utilizan un único argumento que indica la dirección de la siguiente instrucción en el caso de que se cumpla la condición indicada por cc.

DBcc: (Decrement and Branch on Condition Code) facilita la realización de bucles en un programa, y tiene dos argumentos siendo el primero un registro de datos y el segundo un desplazamiento de 16 bits respecto al PC. Su funcionamiento se puede detallar a continuación:

1.-Comprueba la condición cc, y si se cumple, pasa a ejecutar la instrucción siguiente (secuencia del programa) y en caso de estar realizando un bucle se sale del mismo.
2.-Decrementa el contenido del registro de datos.
3.-Si el contenido del registro es -1, se ejecuta la instrucción siguiente (salida del bucle)
4.-Modifica el PC con el desplazamiento indicado (salta al comienzo del bucle).

BRA: (Branch) utiliza direccionamiento relativo al PC, su único argumento es el desplazamiento de 8 ó 16 bits que se suma (con signo) al PC y se obtiene la dirección de la siguiente instrucción a ejecutar. Con desplazamiento corto (8 bits) los saltos máximos que se pueden realizar son -128 y +127. Con desplazamiento largo (16 bits) los saltos máximos son -32.768 y +32.767.

JMP: (Jump) utiliza direccionamiento absoluto y su argumento es la dirección de comienzo de la siguiente instrucción a ejecutar y puede estar situada en cualquier parte del mapa de memoria.
STOP: para la ejecución del programa de forma controlada y en el punto deseado.

2.8 Instrucciones de manejo de subrutinas:

BSR y JSR: (Branch to Subrutine y Jump to Subrutine) el operando asociado con estas instrucciones debe ser la dirección de memoria en la que se comienza la subrutina, con direccionamiento absoluto para JSR, y relativo a PC para BSR. Al ejecutarse cualquiera de las dos instrucciones, se almacena automáticamente en la pila el valor del contador de programa en el momento anterior al salto, cuando su contenido apunta a la dirección de comienzo de la instrucción siguiente a JSR o BSR.
La subrutina debe tener como última instrucción a ejecutar **RTS** (Return from Subrutine) o **RTR** (Return and Restore). Ambas recuperan de la pila el valor del contador del programa, lo que permite reanudar la ejecución del programa precisamente en el punto que había sido abandonado. La instrucción RTR también repone el CCR extrayendo una palabra de la pila inmediatamente antes de recuperar el PC. La subrutina debe llevar a la pila, justamente encima de las posiciones que contienen la posición de retorno, una palabra cuyos 5 bits menos significativos serán llevados al CCR al ejecutarse RTR, que puede ser una copia del CCR al entrar en la subrutina, o cualquier otro valor.