UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"REVISIÓN DE ARQUITECTURAS COMPUTACIONALES PARA LA ASIGNATURA DE ARQUITECTURA DE COMPUTADORAS"

TESIS
QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELÉCTRICO-ELECTRÓNICO

PRESENTA:
RUBÉN LÓPEZ LENA VILLASANA

DIRECTOR DE TESIS:
M EN I JORGE VALERIANO ASSEM

CIUDAD UNIVERSITARIA 2010

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

"REVISIÓN DE ARQUITECTURAS COMPUTACIONALES PARA LA ASIGNATURA DE ARQUITECTURA DE COMPUTADORAS"

RUBÉN LÓPEZ LENA VILLASANA

Tesis presentada a la Comisión integrada por los profesores:

Ing. Laura Sandoval Montaño.

M. en I. Norma Elva Chávez Rodríguez.

M. en I. Rubén Anaya García.

Ing. Vicente Flores Olvera.

MÉXICO D.F. Octubre 2010

(A mis Padres, hermanos y amigos, que me apoyaron en la elaboración de este trabajo.)

AGRADECIMIENTOS

A mis padres por sus palabras, sus consejos, su amor, pero sobre todo por darme la vida y por dejarme la mejor herencia, “mi profesión”.

A mis hermanos, profesores y amigos por brindarme su apoyo y confiar en mí durante mis estudios y realización de la tesis.

Al M. en I. Jorge Valeriano Assem por sustentar, apoyar y guiar este proyecto de tesis.

Gracias también a todos los profesores del jurado por sus valiosos comentarios.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

Pág

ÍNDICE DE TABLAS

Pág

INTRODUCCIÓN

Este texto se ocupará de la arquitectura y organización de las computadoras a través del estudio del concepto de los niveles en la arquitectura de computadoras, la idea básica es que existen muchas capas en los que la computadora puede ser considerada, es decir, desde el nivel más alto donde el usuario ejecuta los programas, hasta el nivel más bajo, que consiste en transistores y cables. Entre el nivel más alto y el más bajo hay un gran número de niveles intermedios. Con el fin de extraer los niveles que nos puedan ayudar a manejar la complejidad de las computadoras, estos niveles son aquellos que nos muestran la organización y arquitectura de computadoras digitales en los que nos enfocaremos a los principales procesadores que se encuentran en el mercado actual para someterlos a revisión con ideas innovadoras ya que en algunos de ellos se encuentra libre su arquitectura.

El trabajo de tesis se realiza por la constante actualización de la arquitectura y organización de los dispositivos que componen los procesadores del mercado actual y que ésta arquitectura todavía no se toma en cuenta para el estudio de la asignatura: Arquitectura de Computadoras impartida en la carrera de Ingeniería en Computación.

El concepto de arquitectura en el entorno informático proporciona una descripción de la construcción y distribución física de los componentes de la computadora.

CAPÍTULO 1. MARCO TEÓRICO

1.1 Estructura general de una computadora.

La computadora digital es un ordenador numérico, automático, secuencial y universal.

Es numérico porque toda la información que se puede encontrar dentro de la computadora está codificada por un conjunto ordenado de ceros y unos. Esta codificación es de tal naturaleza que un conjunto de ceros y unos, pueden estar representando una letra o un número, por esto decimos que la información dentro de una computadora puede ser alfanumérica.

Se dice que es automática pues puede operar sin la intervención del operador al pasar de una operación a otra en la resolución de un determinado problema.

Es secuencial porque debe seguir una serie ordenada de pasos para la resolución de cada problema.

Y finalmente decimos que es universal porque está capacitada para resolver muchos problemas que se plantee, dependiendo de cómo se haya programado a la máquina.

La computadora es un sistema compuesto de cinco elementos diferenciados: 1.-CPU (unidad central de Procesamiento), 2.-Dispositivos de entrada, 3.-Dispositivos de almacenamiento, 4.-Dispositivos de salida y una 5.-Red de comunicaciones, denominada bus, que enlaza todos los elementos del sistema y conecta a éste con el mundo exterior. A continuación se muestra un diagrama de bloques de una computadora (figura 1.1).

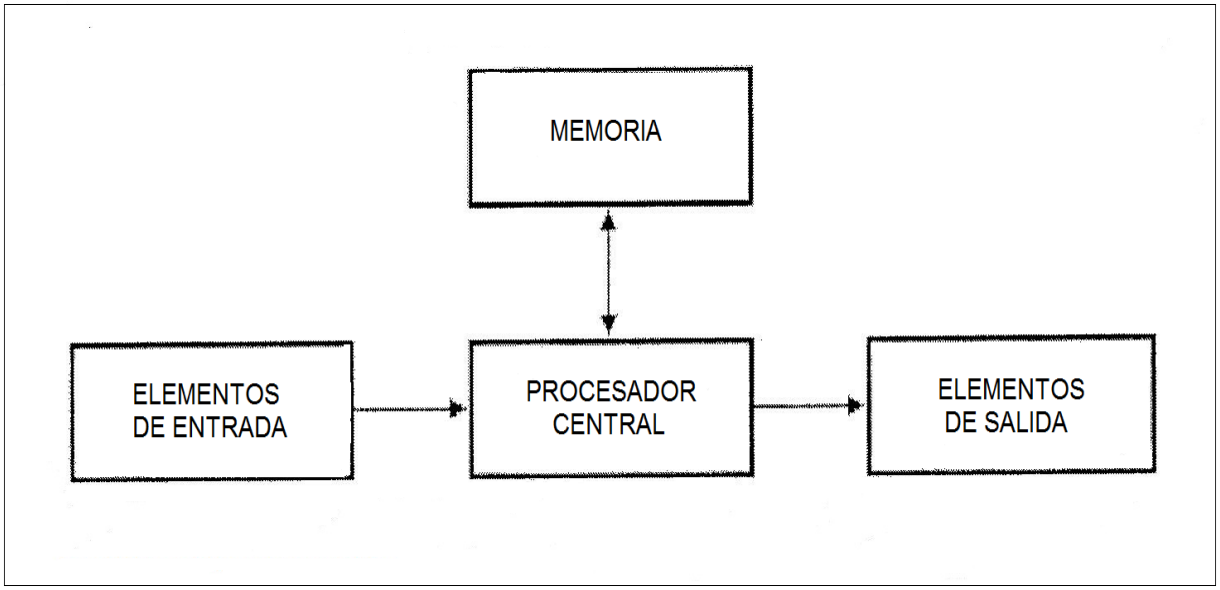


Figura 1.1 Estructura general de una computadora.

1.1.1 –CPU (Unidad Central de Proceso):

Interpreta y lleva a cabo las instrucciones de los programas, efectúa manipulaciones aritméticas y lógicas con los datos y se comunica con las demás partes del sistema. Una CPU es una colección compleja de circuitos electrónicos. Cuando se incorporan todos estos circuitos en un chip de silicio, a este chip se le denomina microprocesador. La CPU, chips y componentes electrónicos se ubican en un tablero de circuitos o tarjeta madre.

La mayoría de los chips del CPU y de los microprocesadores están compuestos de 4 secciones funcionales:

- Una unidad aritmética/lógica que proporciona al chip su capacidad de cálculo.
- Unos registros que son áreas de almacenamiento temporal que contienen datos, realizan seguimiento de instrucciones y conservan la ubicación y los resultados de las operaciones.
- Una sección de control que temporiza y regula las operaciones de la totalidad del sistema informático, lee las configuraciones de datos en un registro designado y las convierte en una actividad e indica en qué orden utilizará la CPU las operaciones individuales y el tiempo que consumirá cada operación.
- Bus interno, red de líneas de comunicación que conecta los elementos internos del procesador y envía también información a los conectores externos que enlazan al procesador con los demás elementos del sistema informático.
- Hay 3 tipos de bus en la CPU: bus de control, bus de dirección y bus de datos.

1.1.2-Dispositivos de entrada:

Son todos aquellos elementos que permiten la interacción del usuario con la unidad de procesamiento central y la memoria.

En esta se encuentran:

- Teclado.
- Mouse o Ratón.
- Escáner o digitalizador de imágenes.
- Micrófonos.

El Teclado:

Es un dispositivo periférico de entrada, que convierte la acción mecánica de pulsar una serie de pulsos eléctricos codificados que permiten identificarla. Las teclas que lo constituyen sirven para entrar caracteres alfanuméricos y comandos a una computadora es similar al de las máquinas de escribir.

Mouse y Joysticks:

Son dispositivos que convierten el movimiento físico en señales eléctricas binarias que permitan reconstruir su trayectoria con el fin de que la misma sea repetida en el monitor.

Escáner o digitalizador de imágenes:

Están concebidos para interpretar caracteres, combinación de caracteres, dibujos gráficos escritos a mano o en maquinas o impresoras y traducirlos al lenguaje que la computadora entiende.

Micrófonos:

Módulos de reconocimiento de voz que convierten la palabra hablada en señales digitales comprensibles para el ordenador.

1.1.3 Dispositivos de Almacenamiento:

En esta se encuentran:

- Disco Duro.
- DVD.
- Cintas magnéticas.

Disco Duro:

Este está compuesto por varios platos, es decir, varios discos de material magnético montados sobre un eje central en el cual giran. Para leer y escribir datos en estos platos se usan las cabezas de lectura / escritura que mediante un proceso electromagnético codifican / decodifican la información que han de leer o escribir. La cabeza de lectura / escritura en un disco duro está muy cerca de la superficie, de forma que casi da vuelta sobre ella, sobre el colchón de aire formado por su propio movimiento. Debido a esto, están cerrados herméticamente, porque cualquier partícula de polvo puede dañarlos.

Este se divide en unos círculos concéntricos cilíndricos (coincidentes con las pistas de los disquetes), que empiezan en la parte exterior del disco (primer cilindro) y terminan en la parte interior (ultimo). Asimismo, estos cilindros se dividen en sectores, cuyo número está determinado por el tipo de disco y su formato, siendo todos ellos de un tamaño fijo en cualquier disco. Cilindros como sectores se identifican con una serie de números que se les asigna, empezando por el 1, pues el numero 0 de cada cilindro se reservan para propósitos de identificación más que para almacenamientos de datos. Estos escritos / leídos en el disco deben ajustarse al tamaño fijado del almacenamiento de los sectores. Habitualmente, los sistemas de discos duros contienen más de una unidad en su interior, por lo que el número de caras puede ser más de dos. Estas se identifican con un número, siendo el 0 para la primera. En general su organización es igual a los disquetes. La capacidad del disco resulta de multiplicar el número de caras por el de pistas por cara y por el de sectores por pista, al total por el número de bytes por sector.

Disco de Video Digital:

Disco de vídeo digital (DVD), un dispositivo de almacenamiento masivo de datos cuyo aspecto es idéntico al de un disco compacto, aunque contiene hasta 15 veces más información y puede transmitirla a la computadora unas 20 veces más rápido que un CD-ROOM. El DVD, denominado también disco de Súper Densidad

(SD) tiene una capacidad de 8,5 gigabytes de datos o cuatro horas de vídeo en una sola cara. En la actualidad, están desarrollándose discos del estilo del DVD regrabables y de doble cara.

Cintas Magnéticas:

Utilizados por los grandes sistemas informáticos.

1.1.4-Dispositivos de Salida:

Estos dispositivos permiten al usuario ver los resultados de los cálculos o de las manipulaciones de datos de la computadora. El dispositivo de salida más común es el monitor, pantalla en la que se ve la información suministrada por el ordenador. Según el tipo se clasifican en VGA Monocromáticos, VGA Color, SVGA Color y UVGA Color.

La resolución se define como el número de puntos que puede representar el monitor por pantalla, en horizontal x vertical. Así, un monitor cuya resolución máxima sea de 1024x768 puntos puede representar hasta 768 líneas horizontales de 1024 puntos cada una, probablemente además de otras resoluciones inferiores, como 640x480 u 800x600. Cuan mayor sea la resolución de un monitor, mejor será la calidad de la imagen en pantalla, y mayor será la calidad (y por consiguiente el precio) del monitor.

Otro de los dispositivos de salida comunes es la impresora es la que permite obtener en un soporte de papel una copia visualizada, perdurable y transportable de la información procesada por un computador.

Las impresoras permiten tener un registro en papel de las operaciones realizadas.

Por último se puede hacer mención a el módem, el cual enlaza dos ordenadores transformando las señales digitales en analógicas para que los datos puedan transmitirse a través de las telecomunicaciones.

1.1.5-Red de Comunicaciones:

Un sistema computacional es un sistema complejo que puede llegar a estar constituido por millones de componentes electrónicos elementales. Esta naturaleza multinivel de los sistemas complejos es esencial para comprender tanto su descripción como su diseño. En cada nivel se analiza su estructura y su función en el sentido siguiente:

Estructura: La forma en que se interrelacionan las componentes

Función: La operación de cada componente individual como parte de la estructura.

Por su particular importancia se considera la estructura de interconexión tipo bus. El bus representa básicamente una serie de cables mediante los cuales pueden cargarse datos en la memoria y desde allí transportarse a la CPU. Por así decirlo es la autopista de los datos dentro del PC ya que comunica todos los componentes del ordenador con el microprocesador. El bus se controla y maneja desde la CPU.

1.1.5 Modelo Von Neumann.

Computadoras digitales convencionales tienen una forma común que es atribuida a John von Neumann (1903-1957). El modelo von Neumann consiste de cinco componentes como los ilustrados en la figura 1.1.5. La unidad de entrada provee instrucciones y datos al sistema, los cuales están almacenados en la unidad de memoria. Las instrucciones y datos son procesados por la Unidad Aritmética y Lógica (ALU) bajo la dirección de la Unidad de Control. Los resultados son enviados a la Unidad de Salida. La ALU y la unidad de control son frecuentemente llamadas en conjunto: Unidad Central de Proceso (CPU). Computadoras más comerciales pueden ser divididas en estas cinco unidades básicas.

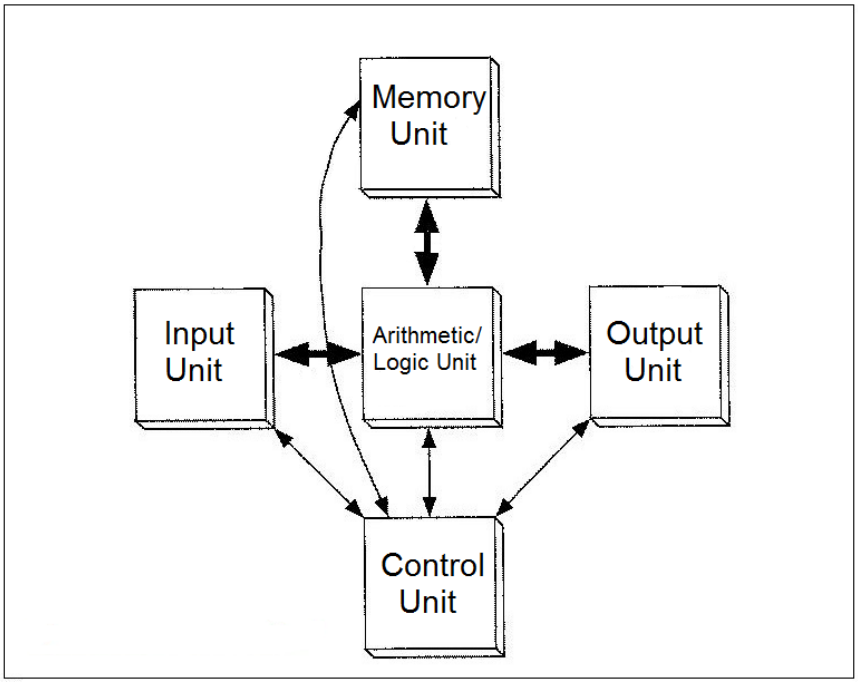


Figura 1.1.5 Modelo Von Neumann.

La ejecución de un programa almacenado es el aspecto más importante del modelo von Neumann. Un programa es almacenado en la memoria de la computadora a través y con los datos a ser procesado. Aunque demos por hecho lo anterior, para el almacenamiento del programa de la computadora, los programas son cargados en un medio externo, como paneles o tarjetas perforadas. El programa puede ser manipulado como si fueran sólo datos, esto brinda un aumento a los compiladores y sistemas de operación y hace posible la gran versatilidad de las computadoras modernas como las conocemos actualmente.

1.2 SECUENCIADORES Y CARTAS ASM.

1.2.1 MAQUINAS DE ESTADOS.

El modelo de máquina de estados contiene los elementos necesarios para describir la conducta de un sistema en términos de entradas, salidas y tiempo.

En el siguiente diagrama se representa un modelo general de una máquina de estados (figura 1.2.1).

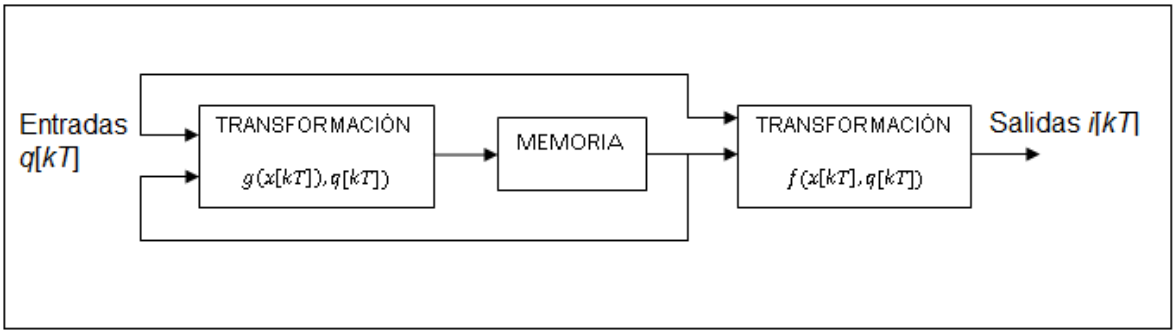


Figura 1.2-1 Modelo general de una máquina de estados.

x[kT], representa el estado en tiempo kT.

x[(k+1)T]=g(x[kT], q[kT]), representa el siguiente estado.

q[kT], representa las entradas en el tiempo kT.

i[kT]=f(x[kT], q[kT]), representa las salidas en el tiempo kT.

T es el periodo de duración de cada estado y k es un contador entero.

1.2.2 REPRESENTACIÓN DE ESTADOS.

El estado de una maquina de estados es la memoria de la historia pasada, suficiente para determinar las condiciones futuras se muestra la representación del estado.

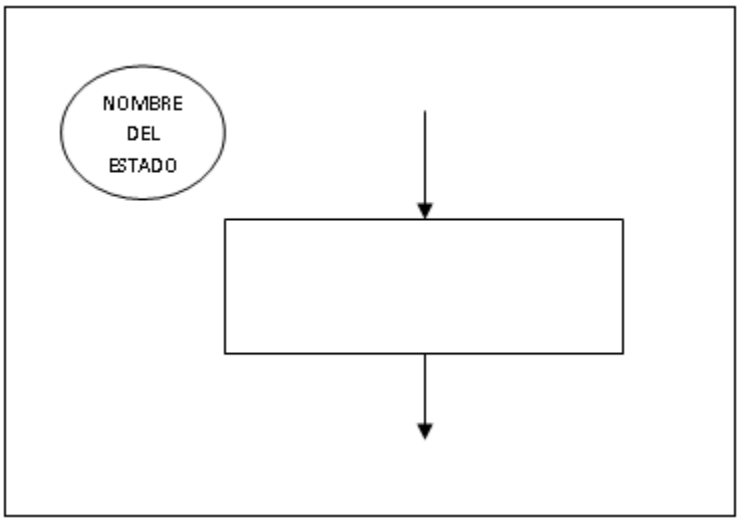


Figura 1.2-2 Representación de un estado.

Un estado se representa con un rectángulo y con su nombre simbólico en el externo superior, encerrado en un círculo.

1.2.3 REPRESENTACIÓN DE DECISIONES.

Las decisiones son usadas para seleccionar el camino que el algoritmo de la máquina de estados debe tomar de acuerdo a la variable o variables de entrada evaluadas. Las decisiones se representan a través de un rombo con el nombre de la variable a probar o una función que evalúe varias variables.

1.2.4 REPRESENTACIÓN DE SALIDAS.

Salidas no condicionales. Sirven para indicar la activación de una variable de salida. Para representarlas, se escriben dentro de un rectángulo de estado, los nombres de las variables de salida que se activan en ese estado. Las salidas no condicionales no dependen de las condiciones de entrada, solo dependen del estado actual.
Salidas condicionales. Estas salidas se presentan solamente cuando ciertas condiciones de entrada existen. Se representan con un óvalo y los nombres de las salidas dentro de él.

1.2.5 CARTAS ASM.

Las cartas AMS son una descripción gráfica del desarrollo de instrucciones en microoperaciones, también se dice que es un grafo orientado y cerrado cuyos nodos son bloques ASM, un bloque ASM es similar a un estado en un circuito secuencial síncrono; todas las acciones asociadas a un bloque ASM tienen lugar en el mismo ciclo de reloj. Un bloque ASM puede estar formado por tres tipos de elementos (gráficos):

- Una caja de estado.
- Cajas de decisión.
- Cajas de acción condicional.

Cajas de estado:
-Existe una y sólo una en cada bloque ASM.
-Especifica las acciones incondicionales del bloque ASM (son aquellas que se activan siempre que se ejecuta el bloque)
-Las acciones son transferidas entre registros= activación de señales de control.
A continuación se presenta un diagrama de Caja de Estado (figura 1.2.5)

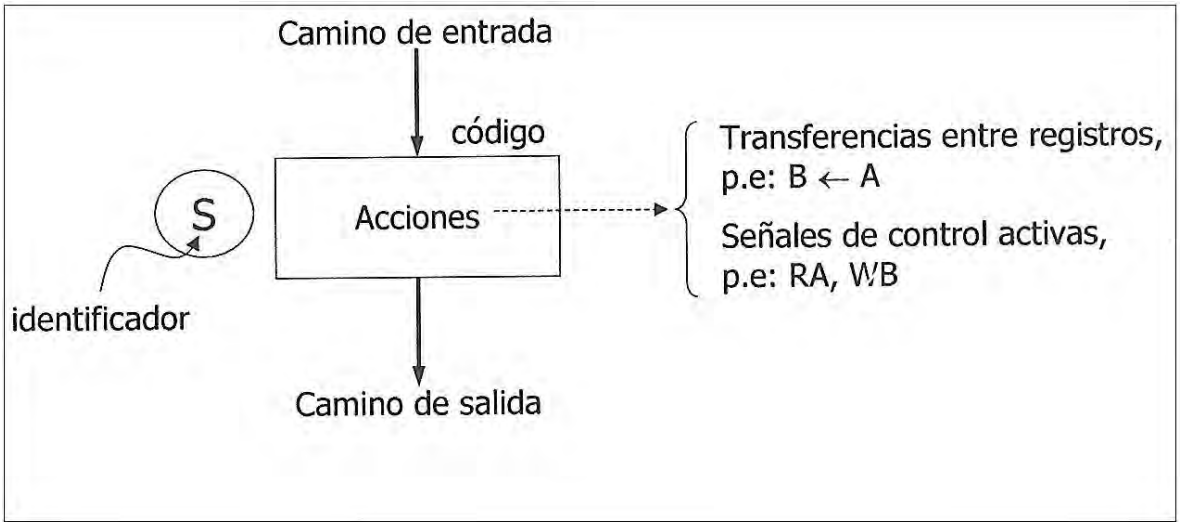


Figura 1.2-5 Cajas de estado.

Cajas de decisión:

-Puede haber varias en un mismo bloque, o ninguna.

-Permiten especificar bifurcaciones dentro del bloque ASM.

-Están caracterizadas por una condición sobre: las entradas de control externas del sistema, y las salidas de control de la unidad de datos. Diagrama de caja de decisión se muestra en la siguiente figura (figura 1.5)

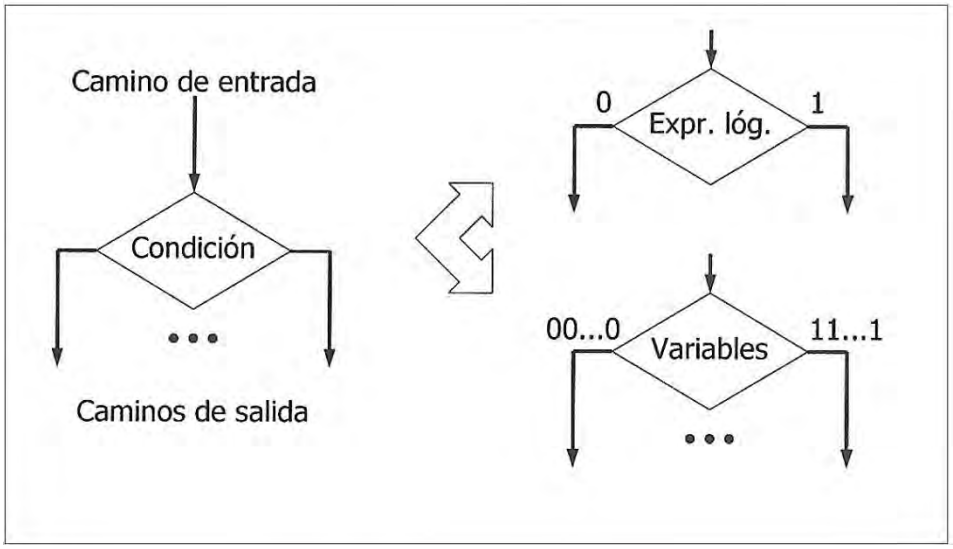


Figura 1.2-6 Caja de decisión.

Cajas de acción condicional:

-Puede haber varias en un mismo bloque.

-Especifica las acciones condicionales del bloque ASM (aquellas que se activan siempre que se ejecuta el bloque y además se cumpla una condición)

-Van siempre asociadas a las cajas de decisión.

-Las acciones son transferidas entre registros= activación de señales de control.

Representación de cajas de acción condicional en la figura 1.6

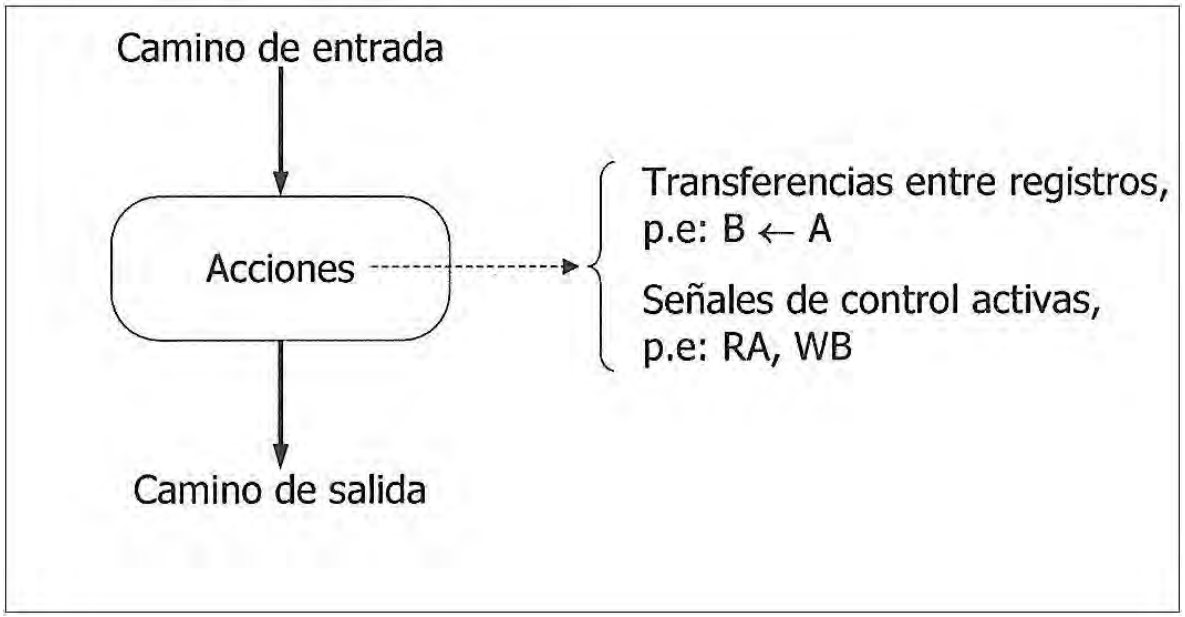


Figura 1.2-7 Caja de acción condicional.

SECUENCIADORES

El Secuenciador Básico.

En el diseño de módulos de control de una computadora es necesario contar con máquinas de estado capaces de ejecutar algoritmos complejos. Para ello se deben hacer modificaciones y agregar componentes a la variante del direccionamiento implícito para crear máquinas de estado que efectúen cartas ASM con llamadas a subrutinas, como las estructuras DO, WHILE, iteraciones tipo FOR, etc. Los dispositivos que son capaces de realizar este tipo de operaciones se llaman secuenciadores.

En la siguiente figura (figura 1.8) se muestra el diagrama de bloques de un secuenciador básico. En él se puede observar la dirección del estado siguiente, dada por el bus y puede proveer de dos lugares posibles.

1.- El registro contador, llamado de micro-programa (µPC), contiene la dirección del estado presente más uno. Es decir, la dirección que se encuentra en la salida del multiplexor es incrementada en una unidad y posteriormente cargada en este registro en el siguiente ciclo de reloj.

2.- En la entrada D se introduce una dirección de salto. Esta dirección puede provenir de tres diferentes lugares: del campo de liga, del registro de transformación o del registro de interrupciones.

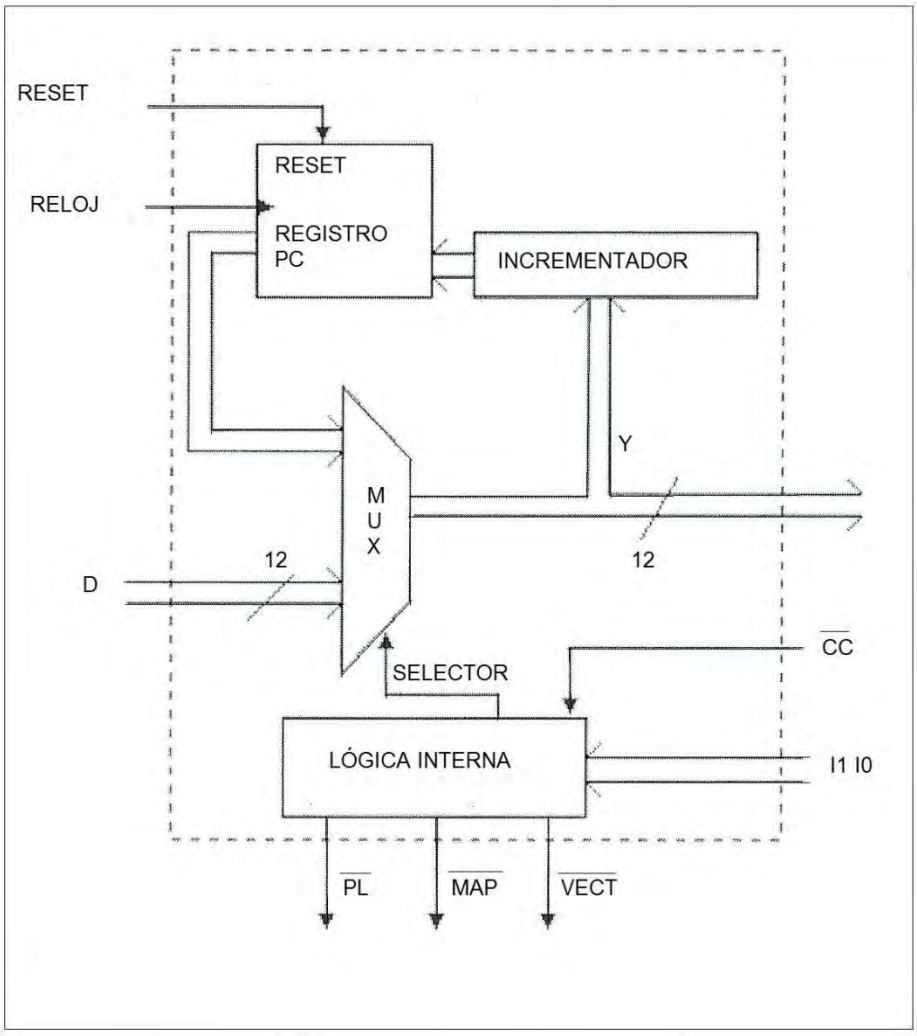


Figura 1.2-8 Diagrama de bloques de un secuenciador básico.

Un secuenciador está formado por una lógica interna que es la que se encarga de general señales que controlan al multiplexor. Dependiendo de la instrucción emitida por las líneas I_1 e I_0 y de la línea $\overline{CC}$, la lógica del secuenciador tiene la capacidad de seleccionar entre la salida del registro μPC o la entrada D. La salida direcciona una memoria que contiene el estado siguiente del algoritmo de la máquina de estados.

Las líneas $\overline{PL}$, $\overline{MAP}$ y $\overline{VECT}$, son generadas por la lógica interna, las cuales seleccionan unos registros en que las salidas están conectadas a la entrada D del secuenciador. Así la dirección de salto podrá venir de tres diferentes lugares. Al diseñar la unidad central de procesos (CPU) se utilizará esta característica.

La figura 1.2-9 muestra las señales de entrada y salida del secuenciador.

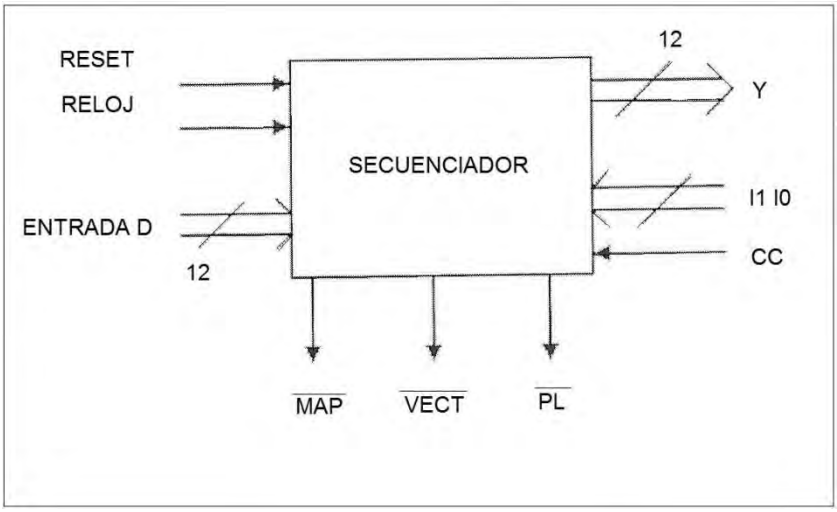


Figura 1.2-9 El secuenciador básico.

Instrucciones para el Secuenciador.

Continua (C)

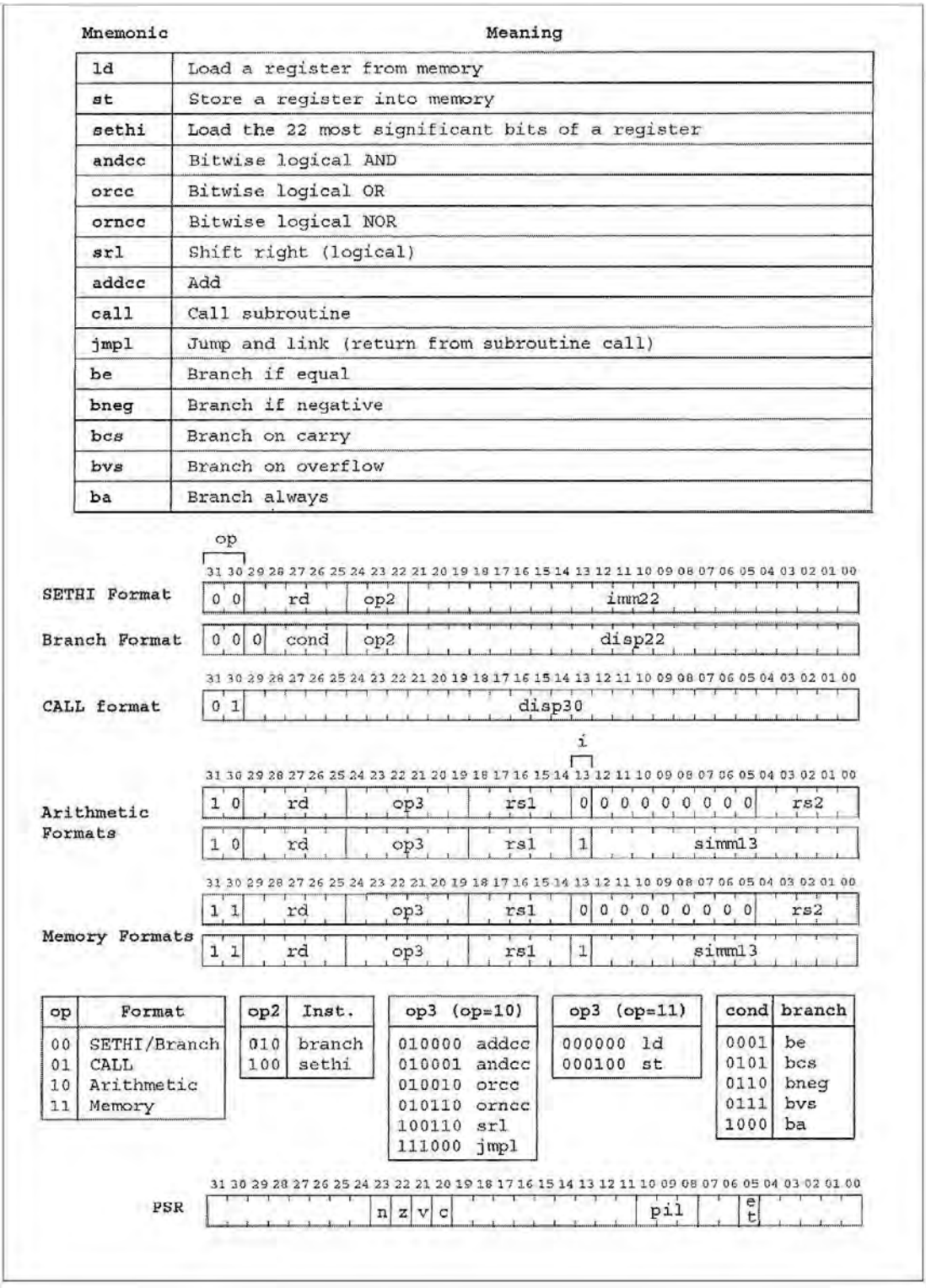
En esta instrucción, la dirección del estado siguiente la proporciona el registro μPC.

Salto condicional (SCO)

Se realiza una revisión al valor de la línea $\overline{CC}$, cuando es igual a uno, la dirección del estado siguiente es proporcionada por el registro μPC; si es igual a cero, la dirección del estado siguiente, contenida en el registro seleccionado por $\overline{PL}$ se ingresa por la entrada D.

1.3 DATA PATH.

En esta sección se va a considerar el datapath y sus señales de control asociadas. Las instrucciones del conjunto que concentraremos y el formato de instrucciones para el conjunto ARC se mostrarán en la siguiente figura en la que se visualizan quince instrucciones que son agrupadas en cuatro formas. El registro del estado del procesador también es mostrado en la parte inferior de la figura.



Mnemonic	Meaning
ld	Load a register from memory
st	Store a register into memory
sethi	Load the 22 most significant bits of a register
andcc	Bitwise logical AND
orcc	Bitwise logical OR
orncc	Bitwise logical NOR
srl	Shift right (logical)
addcc	Add
call	Call subroutine
jmpl	Jump and link (return from subroutine call)
be	Branch if equal
bneg	Branch if negative
bcs	Branch on carry
bvs	Branch on overflow
ba	Branch always

op	Format
00	SETHI/Branch
01	CALL
10	Arithmetic
11	Memory

op2	Inst.
010	branch
100	sethi

op3 (op=10)
010000 addcc
010001 andcc
010010 orcc
010110 orncc
100110 srl
111000 jmpl

op3 (op=11)
000000 ld
000100 st

cond	branch
0001	be
0101	bcs
0110	bneg
0111	bvs
1000	ba

Figura 1.3 Conjunto de instrucciones para ARC, cortesía de Computer Architecture and Organization.

1.3.1 Visión general de datapath.

Un datapath es una colección de unidades funcionales, tales como unidades de aritmética lógica o multiplicadores que realizan el desarrollo de operaciones. La mayoría de las unidades centrales de proceso consistirá en un camino de datos y una unidad de control, con una gran medida de la unidad de control dedicado a la regulación de la interacción entre el camino de datos y la memoria principal.

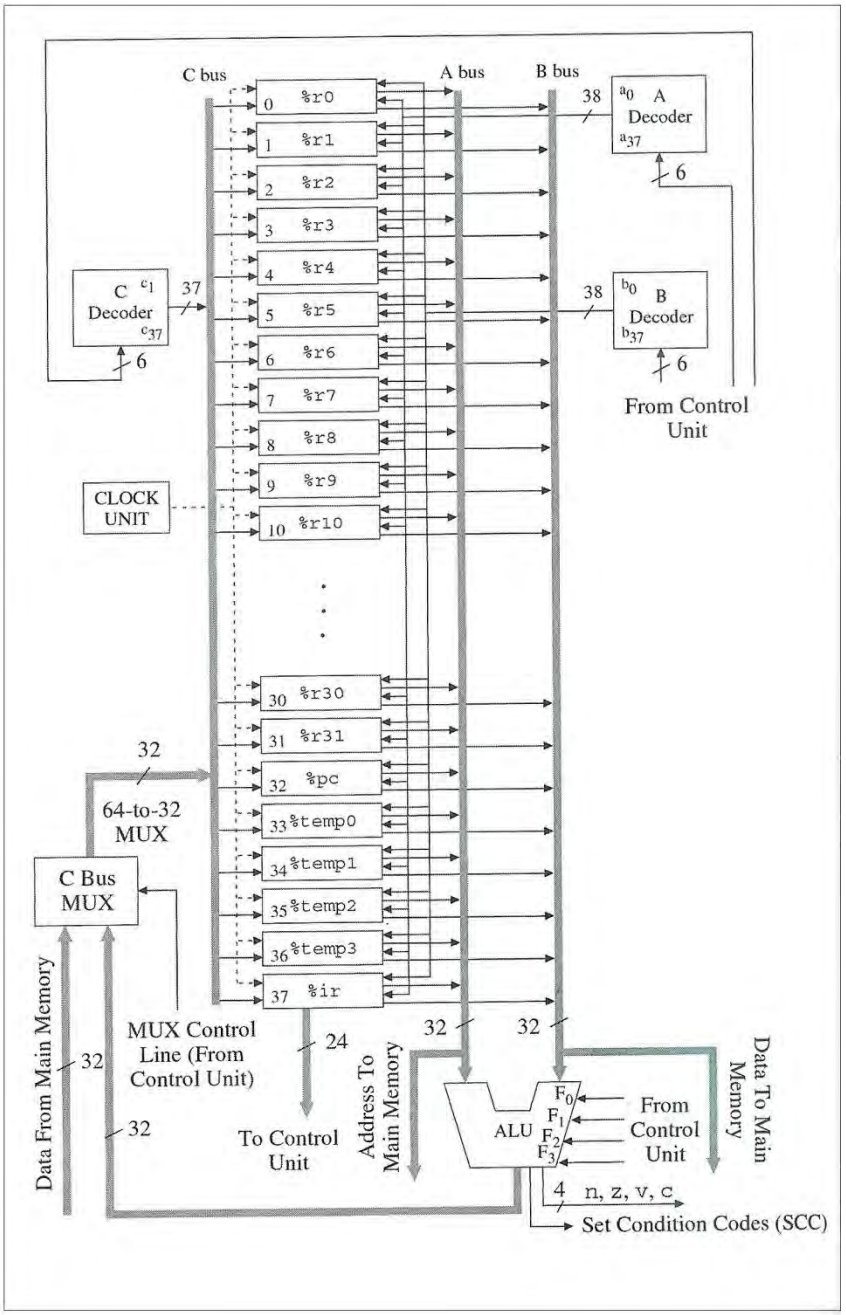


Figura 1.3-1 Datapath de ARC, Cortesía de Computer Architecture and Organization.

1.4 UNIDAD DE CONTROL.

La unidad de control de la computadora tiene como funcionalidad la decodificación de las instrucciones en lenguaje ensamblador y de ejecutar las micro-operaciones que sean necesarias (representadas a través de cartas ASM) para llevarlas a cabo. Para cada instrucción en ensamblador, se activarán secuencialmente una serie de señales que indican a los diferentes componentes de la arquitectura la operación que deben realizar. El secuenciador es la parte fundamental de la unidad de control como se muestra en la figura 1.4

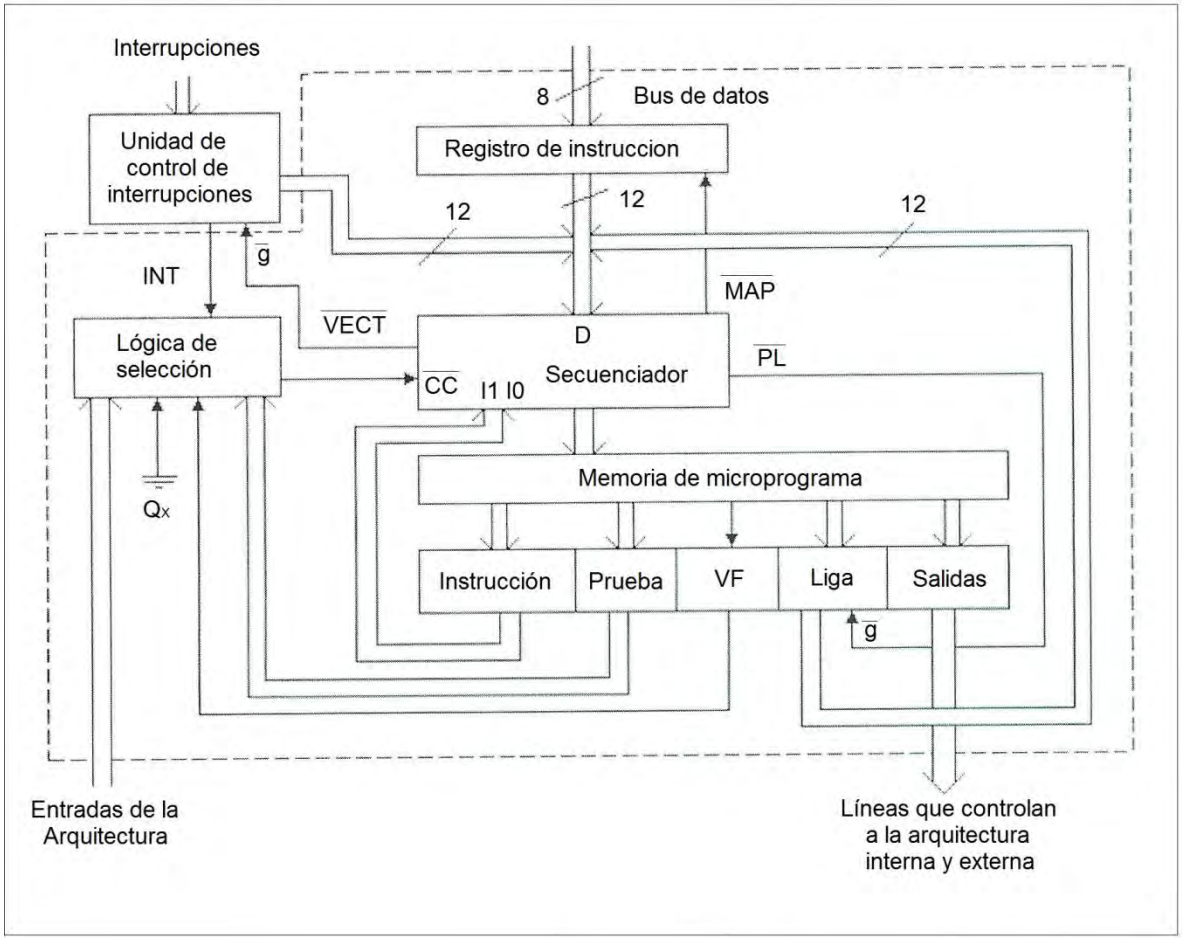


Figura 1.4 Unidad de control de la computadora.

Si el procesador es el núcleo del sistema de computación, la unidad de control lo es del procesador. Tiene tres funciones principales.

- Leer o interpretar instrucciones del programa.
- Dirigir la operación de los componentes internos del procesador.
- Controlar el flujo de programas y datos hacia y desde la RAM.

La unidad de control dirige otros componentes del procesador para realizar las operaciones necesarias y ejecutar la instrucción.

Por otra parte, las entradas de la arquitectura indican el estado en el que se encuentra tanto la arquitectura interna como la externa, y sirven para que el secuenciador pueda tomar decisiones de acuerdo a ciertos criterios. Las entradas son seleccionadas en el bloque de lógica de selección, a través del campo de prueba. La línea de INT también se conecta a la lógica de selección para revisar si existe alguna petición de interrupción.

Existe otra alternativa para diseñar la unidad de control, y es utilizando los lenguajes de descripción de hardware como Verilog HDL, VHDL ó AHDL. Con uno de estos lenguajes y el código de la instrucción que se desea ejecutar, es posible describir los pasos para ejecutar dicha instrucción.

1.5 PIPELINE.

Para lograr paralelismo sin repetir componentes se utiliza un método denominado "línea de ensamblaje" (pipeline). En este caso la computadora se encuentra dividida en varias etapas, en la cual cada etapa realiza operaciones sobre instrucciones diferentes.

La arquitectura en pipeline (basada en filtros) consiste en ir transformando un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior.

Esta arquitectura es muy común en el desarrollo de programas para el intérprete de comandos, ya que se pueden concatenar comandos fácilmente con tuberías (pipe).

También es una arquitectura muy natural en el paradigma de programación funcional, ya que equivale a la composición de funciones matemáticas.

El flujo de instrucciones a través de una tubería sigue los pasos tomados cuando la instrucción es ejecutada.

La primera etapa trae la instrucción, la segunda, decodifica la instrucción, la tercera, trae los operandos de la instrucción y por último la cuarta, ejecuta la instrucción (Figura 1.5).

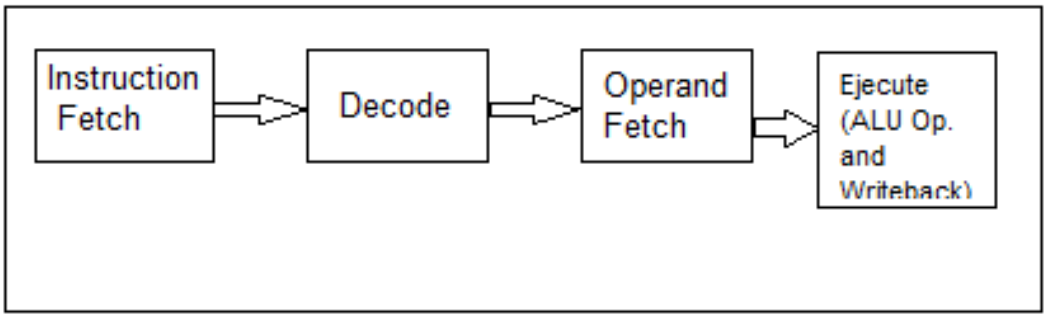


Figura 1.5 Etapas de una instrucción por tubería.

Para cada diseño se debe tener un enfoque de la conducción o tubería desde una perspectiva, dependiendo sobre el particular diseño, objetivos y el set de

instrucciones. Por ejemplo la implementación original SPARC tuvo sólo cuatro tuberías, mientas que algunas tuberías de puntos flotantes pueden tener una docena de etapas.

Cada una de las unidades de ejecución realiza una operación diferente en busca de cerrar el ciclo. Después la instrucción busca que la unidad termine su tarea para que sea llevada a la unidad decodificadora. Hasta este punto la unidad de instrucción buscada puede empezar a buscar la siguiente instrucción, la cual se superpone con la decodificación de la instrucción previa. Cuando la unidad de instrucción y de decodificación completan sus tareas, llevan las otras tareas a la siguiente unidad (el operando es la siguiente unidad a codificar). El flujo de control continúa hasta que todas las unidades están llenadas.

1.5.1 Mantenimiento de Pipeline.

Se debe tomar en cuenta un punto importante: aunque tiene varios pasos para ejecutar una instrucción en este modelo, en promedio, una instrucción puede ser ejecutada por ciclo siempre y cuando la tubería (pipeline) se mantenga llena. La tubería no se queda llena a menos que seamos cuidadosos en cuanto a cómo están ordenadas las instrucciones. En la siguiente tabla (Tabla 1.5.1) se puede visualizar que aproximadamente una de cada cuatro instrucciones es una rama.

Statement	Average Percent of Time
Assignment	47
If	23
Call	15
Loop	6
Goto	3
Other	*1*

Tabla 1.5-1 Frecuencia de aparición de tipos de instrucciones para una variedad de idiomas.

Los porcentajes no suman cien debido al redondeo.

Si bien es cierto, no podemos buscar la instrucción que siga a la rama hasta que ésta haya completado su ejecución. En consecuencia tan pronto como la tubería se llena, la rama es encontrada y luego la tubería tiene que limpiarse llenándola con no operaciones (NOPs). Estas NOPs están a veces referidas como “burbujas”. Una situación similar se presenta con las instrucciones: LOAD y STORE. Ellas generalmente requieren un ciclo de reloj adicional en el cual accesan a la

memoria, que tiene el efecto de ampliar la fase de ejecución de un ciclo a dos ciclos, los ciclos de espera son llenados con NOPs.

En la siguiente tabla (Tabla 1.5.2) se ilustra el comportamiento de la tubería durante la referencia de memoria y también durante la rama del conjunto de instrucción ARC. La instrucción *addcc* entra a la tubería en primer ciclo, para el segundo ciclo la instrucción *ld* que se refiere a la memoria, entra a la tubería y *addcc* se mueve a la etapa de decodificación. La tubería continúa llenándose con las instrucciones *srl* y *subcc* en los ciclos tres y cuatro respectivamente. En el cuarto ciclo la instrucción *addcc* es ejecutada y abandona la tubería. Para el quinto ciclo la instrucción *ld* alcanza el nivel de ejecución, pero no termina la ejecución porque un ciclo adicional es requerido para referencia de memoria. La instrucción *ld* termina de ejecutarse durante el sexto ciclo.

	1	2	3	4	5	6	7	8
Instruction Fetch	adcc	ld	srl	subcc	be	nop	nop	nop
Decode		addcc	ld	srl	subcc	be	nop	nop
Operand Fetche			addcc	ld	srl	subcc	be	nop
Execute				addcc	ld	srl	subcc	be
Memory Reference						ld		

Tabla 1.5-2 Comportamiento de la tubería durante una referencia de memoria.

1.5.2 Riesgos en Pipeline.

Los riesgos en tuberías son los menos problemáticos en el mantenimiento de éstas cuando están llenas. Un riesgo por ejemplo es la situación cuando un resultado es generado por una instrucción no disponible y dicha instrucción es requerida por una instrucción siguiente en la tubería. El potencial de daños en tuberías existe cuando las instrucciones están realizándose, es decir, en algún momento de la ejecución.

1.6 PARALELISMO.

El paralelismo es una forma de computación en que muchos cálculos se llevan a cabo al mismo tiempo, que operan en el principio de que los problemas grandes, se pueden dividir en otros más pequeños, que se resuelven al mismo tiempo ("en paralelo"). Hay varias formas diferentes de paralelismo: a nivel de bits , nivel de instrucción, los datos y paralelismo de tareas. Máquinas paralelas pueden ser clasificadas según el nivel en que el hardware es compatible con el paralelismo en procesadores de computadoras.

Un método para la mejora del rendimiento de un procesador es aumentar la frecuencia de reloj. Esto funciona hasta un punto en que nuestro límite de energía se vuelve incontrolable (mayor frecuencia de reloj consume más energía) para ello se requiere una adecuada ventilación a través de un disipador de calor y un ventilador.

Una alternativa para aumentar la frecuencia de reloj será incrementando el número de procesadores, y se descomponen y distribuyen en un programa único para el procesador. Este enfoque es conocido como procesamiento en paralelo en el cual un número de procesadores trabajan colectivamente, en paralelo para una tarea común.

1.6.1 Midiendo el desempeño.

Una arquitectura paralela se puede caracterizar en términos de tres parámetros: (1) el número de elementos de proceso, (2) la red de interconexión entre los elementos de proceso, y (3) la organización de la memoria. Como ejemplo en las cuatro etapas de una instrucción por pipeline que se muestra en la figura 1.5 existen cuatro elementos de proceso. La interconexión de la red entre los elementos de proceso es un anillo, cabe mencionar que la memoria RAM es una corriente externa al pipeline.

La caracterización de la arquitectura de un procesador en paralelo es una tarea relativamente fácil, pero la medición del rendimiento no es tan sencillo. Aunque podemos medir fácilmente el aumento de la velocidad con un acercamiento al pipeline, la aceleración global depende de los datos: no todos los programas y datos conjuntan el grupo de una arquitectura pipeline.

Las medidas comunes de rendimiento son el tiempo de paralelismo, la aceleración, la eficiencia y el rendimiento. El tiempo de paralelismo es simplemente el tiempo necesario para ejecutar un programa en un procesador en paralelo. El aumento a la velocidad (aceleración) es la relación del tiempo en que el programa se ejecuta en un procesador secuencial (es decir, no paralelo), al tiempo que ese mismo programa se ejecuta en un procesador paralelo. De una forma sencilla representaremos la aceleración (S, en el contexto de procesamiento paralelo) como:

$$S = \frac{T_{secuencial}}{T_{paralelo}}$$

Desde un algoritmo secuencial y una versión paralela del mismo algoritmo, puede ser programado muy diferente para cada máquina, para ello requerimos calificar $T_{secuencial}$ y $T_{paralelo}$ para que se apliquen a la mejor implementación para cada máquina.

Por ejemplo, si queremos un aumento de la velocidad cercana a cien, no será suficiente distribuir cien procesadores en el programa a ejecutar. El problema es que no todos los cálculos son fáciles de descomponer en una forma en que se aprovechen al máximo. Incluso si hay un pequeño número de operaciones secuenciales en un programa paralelo, entonces la velocidad puede ser limitada considerablemente. Esto se resume en la ley de Amdahl en que la aceleración se

expresa en términos del número de procesadores *p* y la fracción *f* de las operaciones que deben realizarse en forma secuencial:

$$S = \frac{1}{f + \frac{1-f}{p}}$$

Por ejemplo, si *f*= 10% de las operaciones deben realizarse en forma secuencial, así, la aceleración puede ser superior a diez, independientemente de cuántos procesadores se utilizan.

$$S = \frac{1}{0.1 + \frac{0.9}{10}} \cong 5.3 \qquad S = \frac{1}{0.1 + \frac{0.9}{\infty}} = 10$$

p=10 procesadores *p*= ∞ procesadores

Esto nos lleva a medir la eficiencia. Eficiencia la definimos como la relación entre el aumento de la velocidad con el número de procesadores que se utilizan. Para un aumento de velocidad de 5.3 usando 10 procesadores, la eficiencia es:

$$\frac{5.3}{10} = 0.53 \text{ ó } 53\%$$

Si doblamos el número de procesadores a 20, entonces el aumento de velocidad se elevará a 6.9 pero la eficiencia será reducida a 34%. Por lo tanto al paralelizar un algoritmo podemos mejorar el rendimiento teniendo como límite la cantidad de operaciones secuenciales.

Así la eficiencia se reduce drásticamente si el aumento de velocidad se acerca a su límite, por lo que no tiene sentido utilizar simplemente más procesadores con la esperanza de obtener una ganancia.

El rendimiento es una medida de cuánto se logra con el tiempo, y es de especial interés para las entradas y salidas de la tubería. Para el caso de una tubería de cuatro etapas que esté llena en el que cada etapa de la tubería completa su tarea en 1ns, el tiempo medio para acompletar una operación es de 1 ns a pesar de que se necesitan 4 ns para ejecutar cualquier operación. El rendimiento general para esta situación es entonces:

$$1.0 \frac{operacion}{ns} = 10^9 operaciones\ por\ segundo$$

1.6.2 LA TAXONOMÍA DE FLYNN.

La arquitectura de computadoras puede ser clasificada en términos del flujo de datos e instrucciones, usando la taxonomía desarrollada por M. J. Flynn (Flynn 1972). Un procesador secuencial convencional encaja en la categoría del flujo de instrucciones y secuencia de datos (SISD) como se ilustra en la figura 1.6-b en que una sola instrucción se ejecuta al momento en un procesador SISD, aunque la tubería puede permitir que varias instrucciones estén en diferentes fases de ejecución en un momento dado.

En un procesador con flujo de instrucciones, flujo múltiple de datos SIMD (single instruction stream), varios procesadores idénticos realizan la misma secuencia de operaciones en diferentes conjuntos de datos, como los ilustrados en la figura 1.6-

b. El sistema SIMD aunque puede ser usado como una sala de llenado, todas las piezas de clasificación diferentes se encuentran en el mismo conjunto de contenedores.

En un flujo de instrucciones múltiples y datos múltiples (MIMD), el procesador realiza diferentes operaciones en distintos conjuntos de datos, pero todas están coordinadas para ejecutarse en un programa paralelo, como se muestra en la figura 1.6-c

El flujo de instrucciones múltiples y datos único (MISD), una secuencia de datos única es operada por varias unidades funcionales, como se ilustra en la figura 1.6-d. El flujo de datos suele ser un vector de varias secuencias relacionadas. Esta configuración es conocida como arreglo o matriz sistólica.

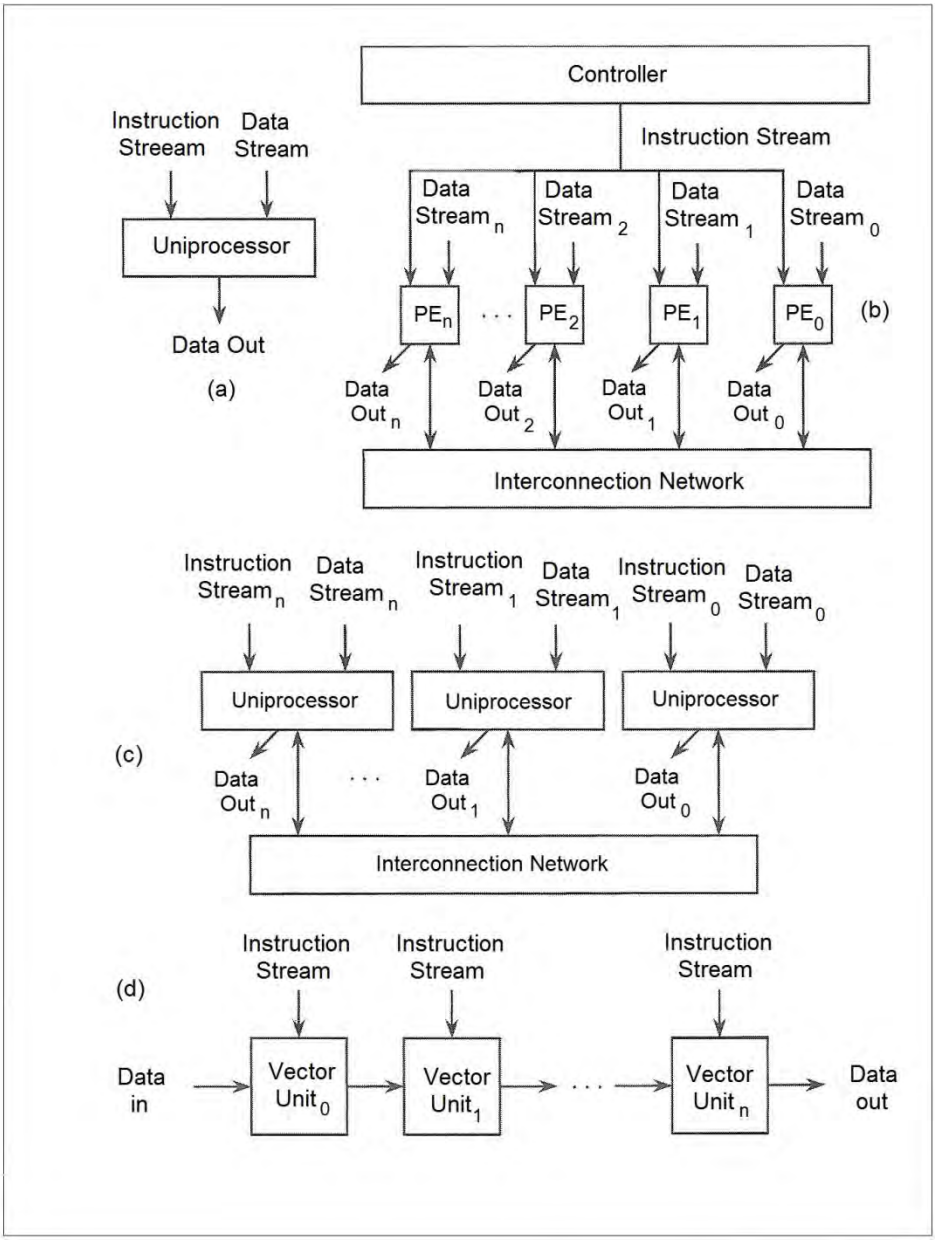


Figura 1.6-1 Clasificación de Arquitecturas de acuerdo a la taxonomía de Fynn (a) SISD; (b) SIMD; (c) MIMD; (d) MISD.

1.6.3 Redes de interconexión.

Cuando la computadora es distribuida por encima de sus elementos de proceso (PEs), ellos necesitan comunicarse con cada uno de ellos a través de una red de interconexión. Existe una variedad de topologías para redes de interconexión, cada una con sus propias características en términos de la complejidad de sus puntos de cruce, diámetro y bloqueo.

Una de las redes más poderosas es la *"crossbar",* en la cual cada elemento de proceso está directamente conectado con los demás elementos de proceso. A continuación se explicarán algunas topologías más representativas para la configuración de redes.

En el otro extremo tenemos la topología *"bus"* la cual es ilustrada en la figura 1.6.1-b, con ella se comparte una parte del ancho de banda entre los elementos de proceso.

En la topología *"ring"* tenemos *N* puntos de cruce para *N* elementos de proceso como se muestran en la figura 1.6.1-c. Cada punto de cruce está contenido dentro de los elementos de proceso.

Para la topología *"mesh"* se tienen *N* puntos de cruce para *N* elementos de proceso pero el diámetro es $2\sqrt{N,}$ como se muestra en la Figura 1.6.1-d. Todos los elementos de proceso pueden estar comunicados simultáneamente en solo $3\sqrt{N}$ pasos.

En la topología *"star"* tenemos un eje central a través del cual todos los elementos de proceso están comunicados (Figura 1.6.1-e).

"tree" es la topología en que tenemos *N* puntos de cruce para *N* elementos de proceso y el diámetro es $2\log_2 N - 1$ (Figura 1.6.1-f). Es efectiva para aplicaciones en las cuales hay una gran cantidad de datos distribuidos.

En la topología *"perfect shuffle"* también tenemos *N* puntos de cruce para *N* elementos de proceso (Figura 1.6.1-g). El diámetro es $\log_2 N$ y pasa a través de la red a conectar cada uno de los elementos de proceso.

Finalmente *"hypercube"* tiene *N* puntos de cruce para *N* elementos de proceso, con diámetro de $\log_2 N - 1$ (Figura 1.6.1-h). El menor número de puntos de cruce con respecto a la topología *perfect shuffle* y está balanceado porque tiene una complejidad mayor en los elementos de proceso.

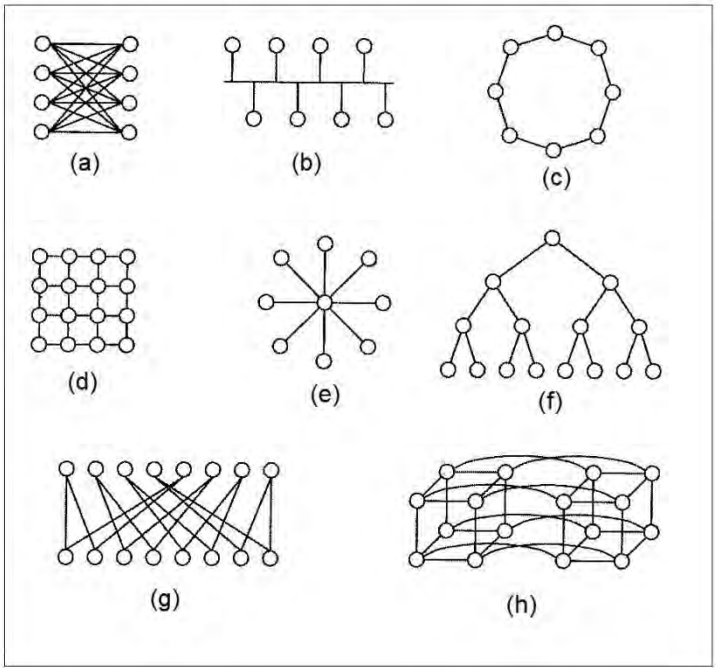


Figura 1.6-2 Topología de redes: (a) crossbar; (b) bus; (c) ring; (d) mesh; (e) star; (f) tree; (g) perfect shuffle; (h) hypercube. Cortesía de Computer Arqchitecture and Organization.

1.6.4 Asignación de Algoritmo en una arquitectura paralela.

El proceso de asignación de un algoritmo en una arquitectura paralela comienza con un análisis de dependencia en que la dependencia de datos entre las operaciones en un programa son identificadas. Consideramos el código C mostrado en la Figura 1.6-3. En un procesador SISD, las cuatro declaraciones numeradas requieren cuatro tiempos para ser completadas, como las ilustradas en la Figura 1.6-4

Número de operaciones

```
func(x, y)   /* Compute (x² + y²) × y² */
int x, y;
{
int temp0, temp1, temp2, temp3;

0  temp0 = x * x;
1  temp1 = y * y;
2  temp2 = temp0 + temp1;
3  temp3 = temp1 * temp2;

return(temp3);
}
```

Figura 1.6-3 Función C en computadoras $(x^2 + y^2)\times y^2$. Cortesía de Computer Arqchitecture and Organization.

El gráfico de dependencias mostrado en la Figura anterior (Figura 1.6-3) expone un paralelismo natural en el control de secuencias. También este gráfico fue creado para la asignación de cada operación en el programa original a un nodo en el gráfico, y luego un trazo se dirige desde cada nodo que produce un resultado al nodo que lo necesita.

La secuencia de control requiere cuatro pasos para completarse, pero el gráfico de dependencias muestra que el programa puede ser completado en solo tres pasos, desde operaciones 0 y 1 no dependen de cada uno y pueden ser ejecutados simultáneamente.

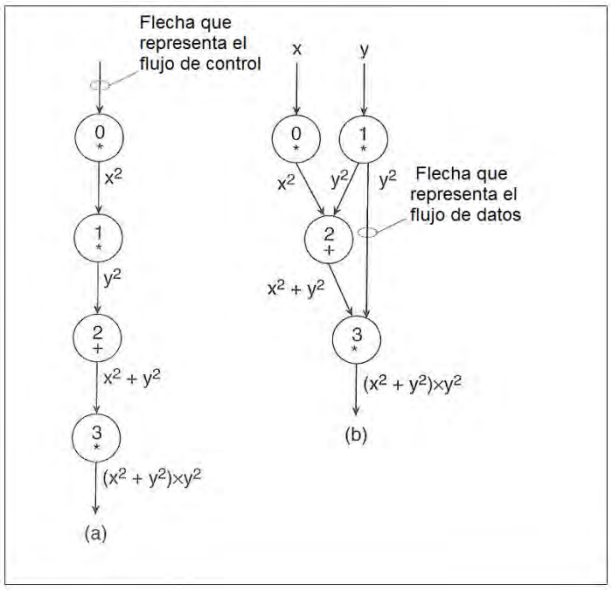


Figura 1.6-4 (a) Control de Secuencia para el programa C; (b) Dependencia gráfica para el programa C. Cortesía de Computer Arqchitecture and Organization.

1.7 CPLD´S, FPGA´S Y VHDL.

1.7.1 CPLD´S.

Un CPLD (Complex Programmable Logic Device) extiende el concepto de un PLD, pero con un mayor nivel de integración, ya que permite implementar sistemas con un mejor desempeño porque utilizan menor espacio, mejoran la confiabilidad en el circuito, y reducen costos. Un CPLD se forma a través de múltiples bloques lógicos, cada uno similar a un PLD. Los bloques lógicos se comunican entre sí utilizando una matriz programable de interconexiones lo cual hace más eficiente el uso del silicio, conduciendo a un mejor desempeño y un costo reducido. A continuación se explican brevemente las principales características de la arquitectura de un CPLD (Figura 1.70).

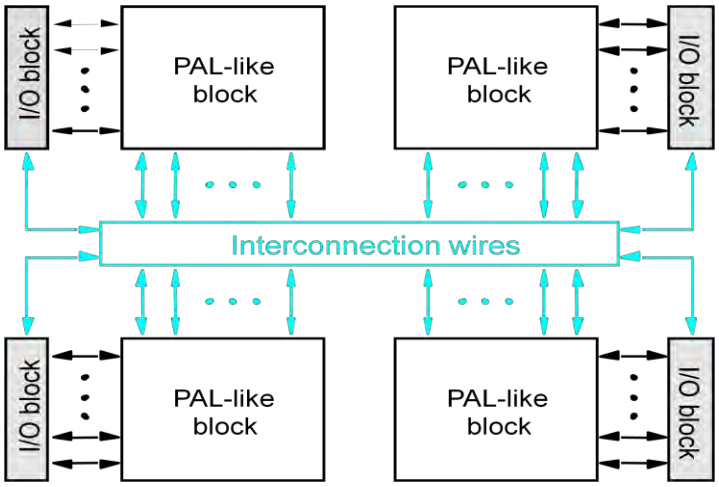


Figura 1.7.1-1 Estructura de un CPLD.

- Alto rendimiento, los dispositivos de lógica programable basados en EEPROM (PLD´s) están basados en la segunda generación de arquitectura MAX.
- Pueden trabajar desde los 2.5 hasta 5 [V].
- El MAX7000S cuenta con 128 o más macrocélulas.
- Retraso lógico de 5ns con una frecuencia máxima de 175.4 MHz

Cuentan con un ahorro de energía programable con una reducción de más del 50% en cada macrocélula, también cuentan con 44 a 208 pines disponibles en el chip. Tiene un soporte de diseño de software y automático en un lugar y ruta proporcionada por Altera en el desarrollo para PC basado en Windows y Sun (Tabla 1.7).

Características	**EPM7032S**	**EPM7064S**	**EPM7128S**	**EPM7160S**	**EPM7192S**	**EPM7256S**
Compuertas Utilizables	600	1250	2500	3200	3750	5000
Macrocelulas	32	64	128	160	192	156
Matriz Lógica de bloques	2	4	8	10	12	16
Pines Máximos I/O	36	68	100	104	124	164
$t_{PD}(ns)$	5	5	6	6	7.5	7.5
$t_{SU}(ns)$	2.9	2.9	3.4	3.4	4.1	3.9
$t_{FSU}(ns)$	2.5	2.5	2.5	2.5	3	3
$t_{CO1}(ns)$	3.2	3.2	4	3.9	4.7	4.7
$f_{CNT}(MHz)$	175.4	175.4	147	149.3	125	90.9

Tabla 1.7.1 Características de la Familia MAX 7000S. Cortesía Altera Corporation.

La familia de alta densidad MAX 7000 tienen PLD´s de alto rendimiento que se basan en la segunda generación de la arquitectura Altera MAX. Fabricados con tecnología avanzada CMOS y basados en EEPROM, la familia MAX 7000 tiene entre 600 y 5000 compuertas utilizables. Su retraso entre pines es de sólo 5ns con velocidades de hasta 151,5 MHz. (Ver tabla 1.7.2).

Dispositivo	Grado de Velocidad									
	-5	-6	-7	-10P	-10	-12P	-12	-15	-15T	-20
EPM7032	√	√	√		√		√	√	√	
EPM7032S		√	√		√					
EPM7064		√	√		√		√	√		
EMP7064S	√	√	√		√					
EPM7096			√		√		√	√		
EPM7128E			√	√	√		√	√		√
EPM7128S		√	√		√			√		
EPM7160E				√	√		√	√		√
EPM7160S		√	√		√			√		

Tabla 1.7.2 Grado de Velocidades de la Familia MAX 7000S. Cortesía Altera Corporation.

Existen versiones mejoradas en que los dispositivos tienen reloj mundial adicional, pueden permitir controles, aumento de interconexión de los recursos, registros de entrada rápida, este tipo de dispositivos soporta TTL al 100%.

La familia de dispositivos MAX 7000 utilizan células CMOS EEPROM para implementar la lógica de funciones. El usuario puede configurar este tipo de familias ya que tiene capacidad en su arquitectura para una variedad de funciones independientes de lógica combinatoria y secuencial. Los dispositivos pueden ser programados para las iteraciones rápidas y eficaces en el desarrollo de diseño y ciclos de depuración, lo mejor es que se puede programar y borrarse hasta 100 veces.

El MAX 7000 proporciona velocidad programable / potencia optimizada. Partes críticas de la velocidad de un diseño puede funcionar como alta velocidad / potencia, mientras que las porciones reducidas trabajan como velocidad / potencia baja. También disponen de una opción que reduce la velocidad de subida de los búferes de salida, reduciendo al mínimo el ruido transitorio cuando la velocidad no es crítica. Los controladores de salida de toda la familia MAX 7000 (excepto dispositivos de 44 pines) se pueden establecer entre 3.3 [V] o 5[V] lo que permite el uso de estos dispositivos en sistemas mixtos de tensión.

La familia MAX 7000 es compatible con los sistemas de Altera, que son paquetes integrados que ofrecen esquemas, textos, incluyendo VHDL, Verilog y el lenguaje de descripción de hardware de Altera (AHDL).

Funciones Principales.

- Bloques lógicos
- Macroceldas
- Expansión o distribución de términos de producto (compatible y paralelos)
- Matriz de interconexión programable
- Entrada y Salida de bloques de control

Un bloque lógico es similar a un PLD, cada bloque está compuesto por un arreglo de compuertas AND y OR en forma de suma de productos. El tamaño del bloque está definido por la capacidad del CPLD y así mismo depende del tamaño de la función booleana que pueda ser implementada dentro del bloque. Los bloques lógicos generalmente cuentan con 4 a 20 macroceldas.

Las macroceldas tienen registros, control de polaridad y buffers para salidas en alta impedancia. Por lo general un CPLD tiene macroceldas de *entrada/salida, macroceldas de entrada* y *macroceldas internas* u *ocultas*.

Para la expansión o distribución de productos tenemos diferencias en cuanto a las matrices de productos, para ello dependemos del CPLD y del fabricante. El tamaño de las sumas es el factor más importante para la implementación de funciones booleanas. Aunque la mayoría de las funciones lógicas se pueden implementar con los cinco términos de productos disponibles en cada macrocélula, la función lógica más compleja requiere condiciones adicionales. Otra macrocélula puede ser usada para suministrar los recursos necesarios de la lógica, así mismo la arquitectura MAX 7000 también permite compartir en términos de expansión de producto (“expansores”) que proveen productos adicionales directamente a cualquier macrocélula en la misma matriz de bloques lógicos. Para ello tenemos la expansión compartida y expansión paralela, en la primera se tienen 16 expansores que pueden ser vistos como un conjunto de términos cada uno con salidas invertidas que se alimentan de nuevo en la matriz lógica y los expansores puden alimentar múltiples macrocélulas, para la expansión paralela, tenemos términos sin usar que se pueden asignar a una macrocélula vecina para aplicar la lógica de funciones complejas, y se tiene permitido hasta veinte términos de productos para alimentar directamente la macrocélula, el diagrama se muestra en la Figura 1.71.

La matriz de interconexiones programables (PIA) se encarga de enlazar los pines de entrada/salida a las entradas del bloque lógico, o las salidas del bloque lógico a las entradas de otro bloque lógico o incluso también a las entradas del mismo. Las configuraciones usadas para esta matriz son: la interconexión mediante arreglo o interconexión mediante multiplexores, de las cuales sólo usan una de ellas. En la interconexión mediante arreglo es una matriz de filas y columnas que contiene una celda programable de conexión en cada intersección y puede ser activada para conectar o desconectar la correspondiente fila y columna. Para la interconexión

mediante multiplexores se tiene un multiplexor por cada entrada al bloque lógico. Las vías de interconexión programables son conectadas a las entradas de un número de multiplexores por cada bloque lógico.

La arquitectura MAX 7000 incluye cuatro entradas (Figura 1.7.1-2) dedicadas que pueden ser utilizadas como insumos de uso general o como de alta velocidad, con control mundial de señales (reloj, clear y dos salidas disponibles).

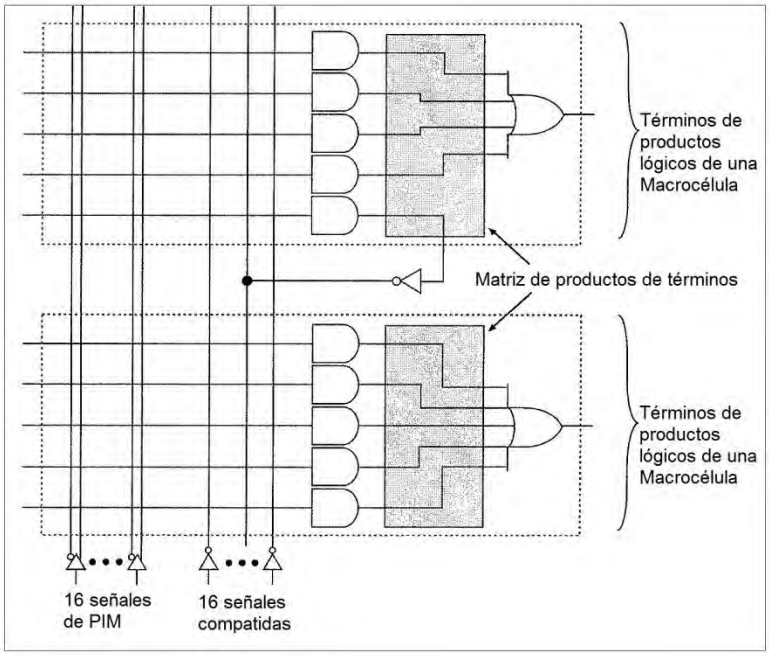


Figura 1.7.1-2 Expansores compartidos. Cortesía Altera Corporation.

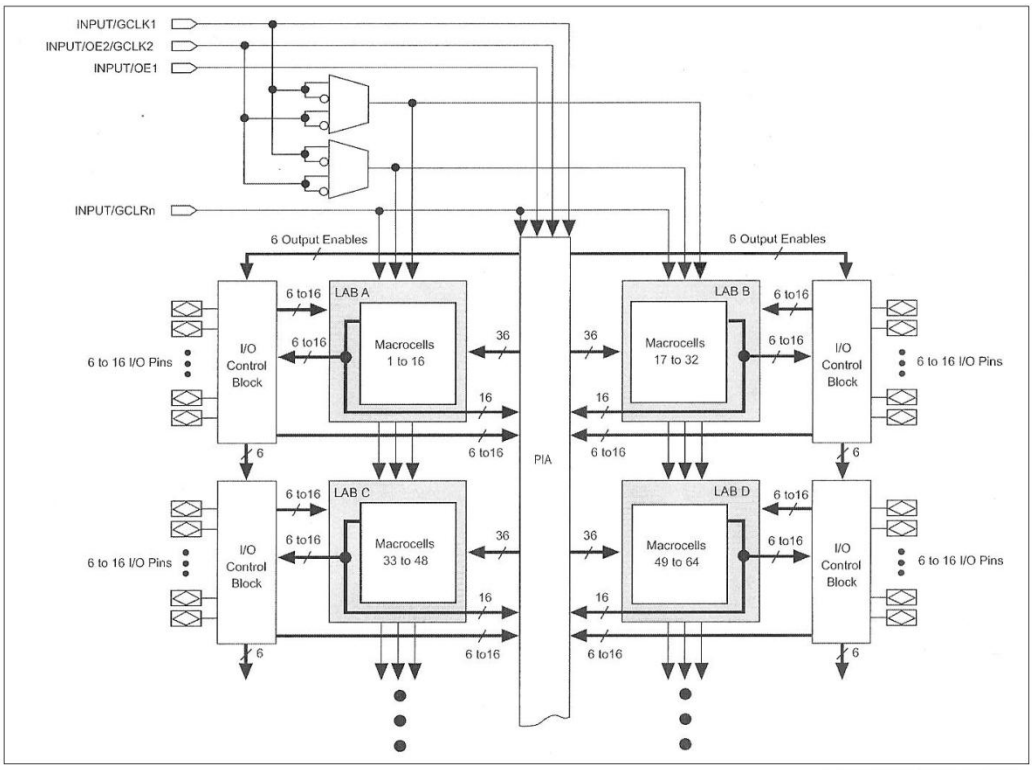


Figura 1.7.1-3 Arquitectura de la familia MAX7000. Cortesía Altera Corporation.

Tiempo de Programación.

El tiempo necesario para poner en práctica cada una de las etapas se menciona a continuación:

- Un tiempo de pulso para borrar, programar, o leer las células EEPROM.
- Un cambio de tiempo basado en la frecuencia de reloj de prueba (TCK) y el número de ciclos de TCK para cambiar las instrucciones, dirección y datos del dispositivo.

Al combinar el pulso y los tiempos de cambio para cada uno de las etapas de los programas, el tiempo puede derivar en función de la frecuencia TCK. Porque diferentes dispositivos con capacidad de ISP tienen un número diferente de células EEPROM, y la duración total de variables fijas son únicas para un solo dispositivo.

El tiempo necesario para programar el dispositivo MAX 7000 se puede calcular a través de la siguiente expresión:

$$t_{PROG} = t_{PPULSE} + \frac{C_{cycle}{}_{PTCK}}{f_{TCK}}$$

Donde:

t_{PROG} = tiempo de programación.

t_{PPULSE} =suma del tiempo fijado para borrar, programar, y verificar las células EEPROM.

$C_{cycle_{PTCK}}$=número de ciclos TCK para programar el dispositivo.

f_{TCK} =frecuencia TCK.

Los tiempos ISP para una verificación independiente de un dispositivo MAX 7000 se pueden calcular de la siguiente manera:

$$t_{VER} = t_{VPULSE} + \frac{C_{cycle_{VTCK}}}{f_{TCK}}$$

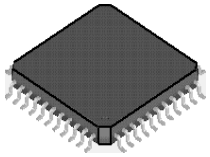
Donde:

t_{VER}= tiempo de verificación.

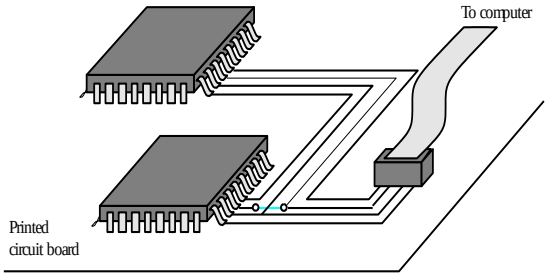
t_{VPULSE}= suma del tiempo fijado para verificar las células EEPROM.

$C_{cycle_{VTCK}}$=número de ciclos TCK para verificar el dispositivo.

Los dispositivos MAX 7000 pueden ser programados en PC basadas en Windows con el programador de tarjetas lógico de Altera, una unidad de programación Maestra (MPU), y el adaptador del dispositivo adecuado. El MPU realiza una prueba de continuidad para asegurar un contacto eléctrico adecuado entre el adaptador y el dispositivo. Ver Figura 1.7.1-3



(a) CPLD in a Quad Flat Pack (QFP) package



(b) JTAG programming

Figura 1.7.1-4 Programación y empaquetado de un CPLD.

A continuación se muestran las condiciones de operación del MAX 7000. Tabla 1.7.3

Símbolo	Parámetro	Condiciones	MIN	MAX	UNIDAD
V_{CC}	Supply voltaje	With respect to ground	-2.0	7.0	V

V_I	DC input voltaje		-2.0	7.0	V
I_{OUT}	DC output current, per pin		-25	25	Ma
T_{STG}	Storage temperature	No bias	-65	150	°C
T_{AMB}	Ambient temperature	Under bias	-65	135	°C
T_J	Junction temperature	Ceramic packages, under bias		150	°C
		PQFP and RQFP packages, under bias		135	°C

Tabla 1.7.3 Valores máximos absolutos del dispositivo MAX 7000.

Consumo de potencia.

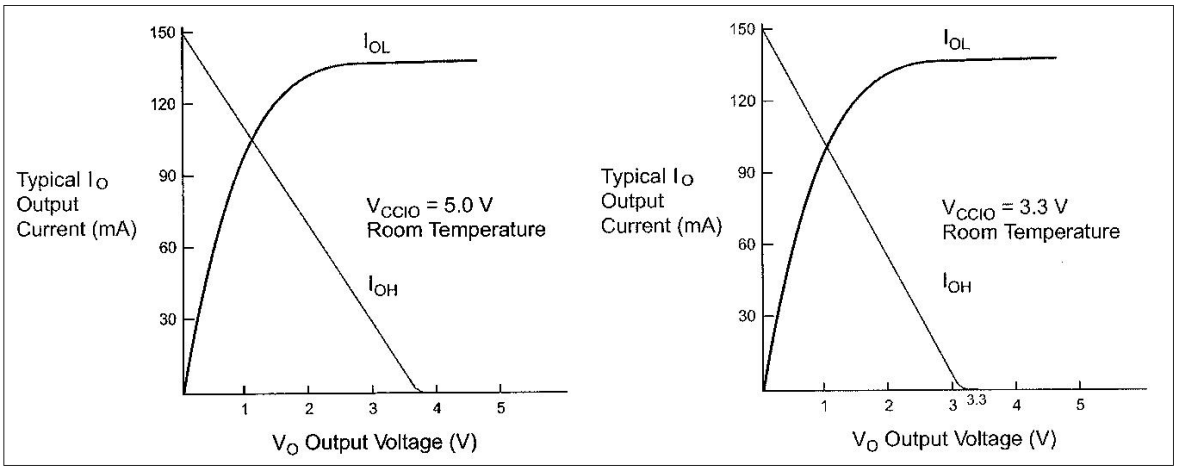
La fuente de energía (P) contra la frecuencia (F_{MAX} en MHz) para dispositivos MAX 7000 se calcula con la siguiente ecuación:

$$P = P_{INT} + P_{IO} = I_{CCINT} \times P_{IO}$$

El valor P_{IO} depende de las características de carga de salida y una frecuencia de conmutación.

El valor I_{CCINT} que depende de la frecuencia de conmutación y la lógica de aplicación.

A continuación se ilustran las características típicas de salida de los dispositivos MAX 7000. Gráfica 1



Gráfica 1. Características típicas de salida de los dispositivos MAX 7000. Cortesía Altera Corporation.

1.7.2 FPGA´S.

Un FPGA (*Field programable Gate Array*) es un dispositivo resultado de la evolución de los CPLD´s, que como ya se mencionó anteriormente, es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser programada. Ambos dispositivos contienen un gran

número de elementos lógicos programables. Haciendo una analogía de la densidad de elementos lógicos programables en compuertas lógicas equivalentes (número de compuertas NAND equivalentes que se pueden programar en un dispositivo) se puede decir que en el CPLD tenemos el orden de decenas de miles de compuertas lógicas, mientas que en un FPGA estaríamos en el orden de cientos de miles o hasta millones de compuertas lógicas programables como se muestra en la Tabla 1.8

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB=Four Slices)			Distributed RAM Bits (K=1024)	Block RAM Bits (K=1024)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLB´s						
XC3S50	50K	1,728	16	12	192	12K	72K	4	2	124	56
XC3S200	200K	4,320	24	20	480	30K	216K	12	4	173	76
XC3S400	400K	9,064	32	28	896	56K	288K	16	4	264	116
XC3S1000	1M	17,280	48	40	1,920	120K	432K	24	4	391	175
XC3S1500	1.5M	29,952	64	52	3,328	208K	576K	32	4	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	4	565	270
XC3S4000	4M	62,208	96	72	6,912	432K	1,728K	96	4	712	312
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	4	784	344

Tabla 1.8 Características del Modelo Spartan-3 FPGA. Cortesía XILINX.

Los FPGA´s están diseñados específicamente para satisfacer las necesidades de alto volumen, de costo razonable para aplicaciones en el consumo de electrónica.

La familia Spartan-3 se basa en el éxito de su familia anterior Spartan-IIE al aumentar la cantidad de los recursos de lógica, la capacidad de la memoria interna RAM, el número total de E/S, y el nivel global de rendimiento, así como mejorar las funciones del reloj.

La lógica de elementos flexibles de la arquitectura FLEX incorpora todas las características necesarias para poner en práctica megafunciones en el arreglo de compuertas con un máximo de 250,000 compuertas. La familia FLEX 10K proporciona densidad, velocidad y características para integrar sistemas completos, incluyendo múltiples buses de 32 bits en un solo dispositivos.

Son dispositivos reconfigurables que permiten la inspección al 100% previa a su envío.

Cada dispositivo FLEX 10K contiene una matriz incrustada para aplicar la memoria y la lógica de funciones especializadas, y una gran lógica para aplicar lógica general.

Debido al bajo costo, los FPGA´s son ideales para una amplia gama de aplicaciones de productos electrónicos, como el acceso de banda ancha, redes domésticas, proyecciones y equipos de televisión digital. También son una alternativa superior para enmascarar ASIC´s programados.

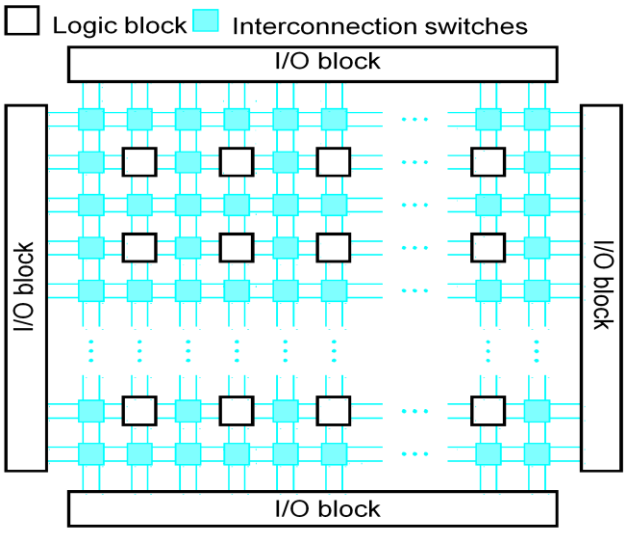


Figura 1.7.2-1 Estructura general de un FPGA.

La arquitectura de la familia Spartan-3 consta de cinco fundamentales elementos programables:

- Contienen bloques lógicos configurables (CBLs) basados en RAM para aplicar la lógica y almacenamiento de elementos que pueden ser utilizados como flip-flops o cerraduras.
- Bloques de E/S (IOB´s) para controlar el flujo de datos entre los pines I/O y la lógica interna del dispositivo. Cada IOB apoya el flujo de datos bidireccional más un tercer estado de operación.
- Bloque de memoria RAM proporciona almacenamiento de datos en forma de 18Kbit.
- Multiplicador de bloques para aceptar dos números binarios de 18 bits como insumos y poder calcular el producto.
- Un bloque administrador de reloj digital (DCM) que proporciona auto calibración, y soluciones completamente digitales para distribuir, retrasar, multiplicar, dividir y cambiar el reloj de fase en señales. Ver Figura 1.8.2

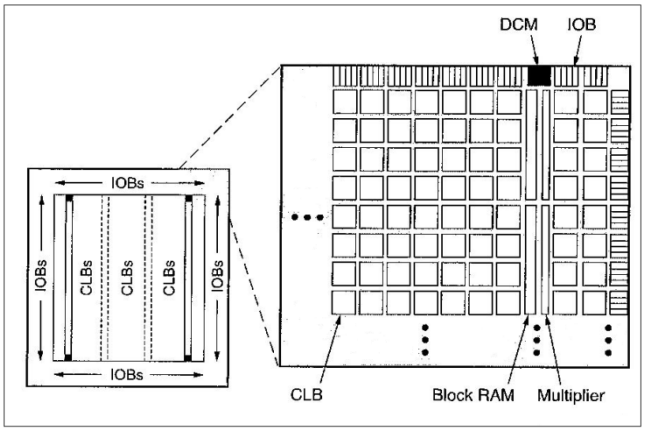


Figura 1.7.2-2 Arquitectura de la Familia Spartan-3. Cortesía Xilinx.

Antes de encender el FPGA, los datos de configuración se almacenan externamente en una PROM o algún otro medio dentro o fuera de la tarjeta.

Dentro de la familia Spartan-3 todos los dispositivos son compatibles pin a pin por encapsulado.

El EAB (Embedded Array Block) es un bloque flexible de RAM con registros sobre los puertos de entrada y salida, y se utiliza para poner en práctica infinidad de arreglo de compuertas.

El LE (Logic Element) es la unidad más pequeña de la lógica en la arquitectura de estos dispositivos, tiene un tamaño compacto que proporciona la utilización de la lógica eficiente, cada uno cuenta con cuatro entradas, que es un generador de funciones que puede calcular rápidamente cualquier función de cuatro variables. Además cada LE contiene un flip-flop programable, una cadena de transporte, además de una cadena en cascada. Ver Figura 1.8.3

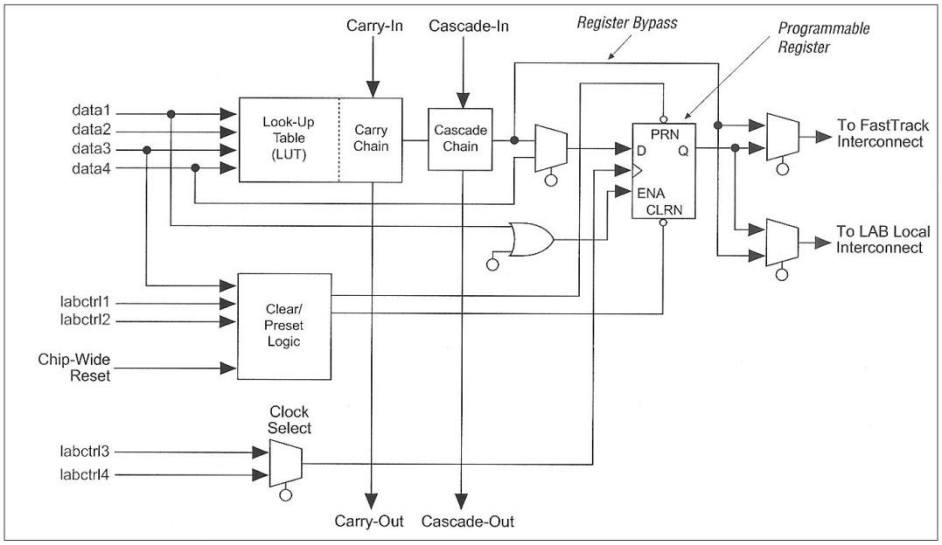


Figura 1.7.2-3 Elemento Lógico (LE). Cortesía Altera Corporation.

La cadena realiza un rápido traspaso de funciones entre ellas. Esta característica permite a la familia FLEX 10K implementar contadores de alta velocidad y comparadores de tamaño de manera eficiente. Ver Figura 1.8.4

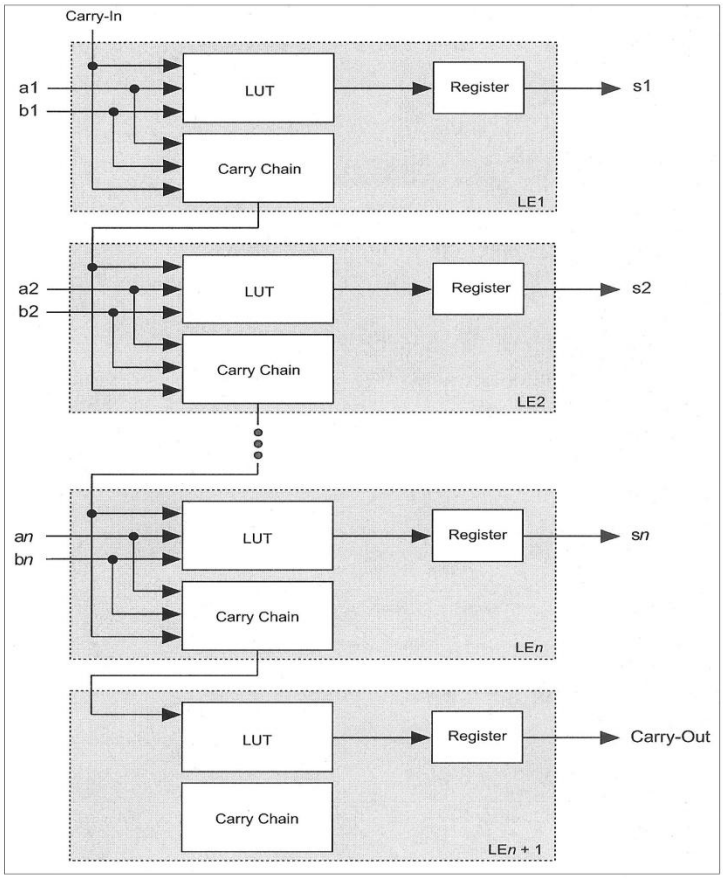


Figura 1.7.2-4 Cadena de Cascadas. Cortesía Altera Corporation.

Con la cadena en cascada se aplican funciones que tienen un amplio abanico de entrada se utilizan para calcular las porciones de la función en paralelo. Pueden utilizar un AND lógico o un OR lógico para conectar las salidas adyacentes. Figura 1.7.2-5

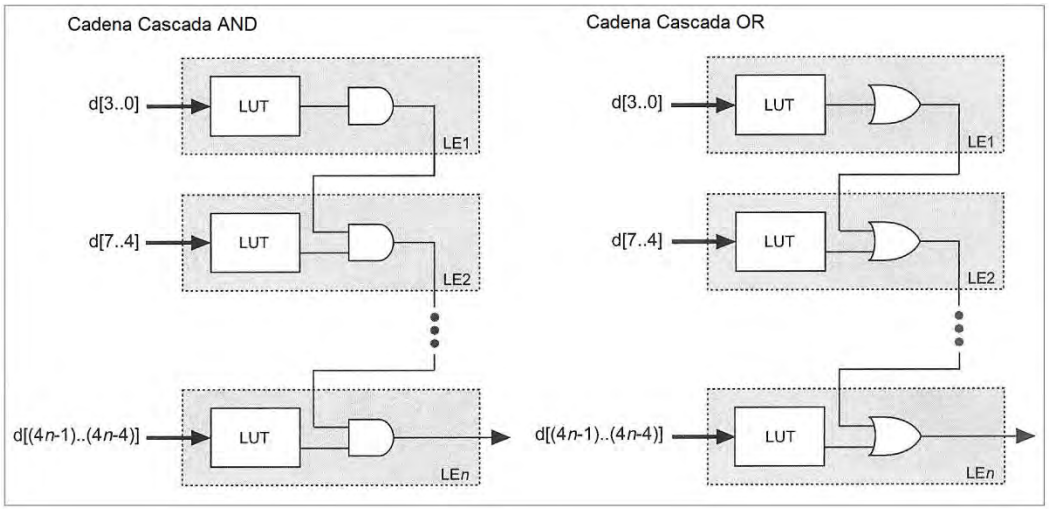


Figura 1.7.2-5 Cadena Cascada AND y OR. Cortesía Altera Corporation.

Los elementos lógicos (LE) pueden operar en los siguientes cuatro modos:

- Modo normal (Normal mode).
- Modo de aritmética (Arithmetic mode).
- Modo contador ascendente (Up-down counter mode).
- Modo contador de limpieza (Clearable counter mode).

Ver Figura 1.8.6

El modo normal es adecuado para aplicaciones de la lógica general y una amplia decodificación de las funciones que pueden tomarle ventaja a una cadena en cascada. En modo normal tenemos cuatro entradas de datos.

En modo aritmético o de cálculo, tenemos tres entradas ideales para la aplicación de complementos, acumuladores y comparadores. Primero se usa el traslado de la señal y dos entradas de datos desde el nivel local de la interconexión de LAB (Logic Array Blocks) para generar una salida.

El contador ascendente (arriba/abajo) ofrece un contador disponible, reloj, y control síncrono arriba/abajo, los cuales controlan las señales generadas por las entradas de datos de la interconexión LAB.

Para apoyar diseños de alta velocidad el fabricante Altera ofrece un reloj que contiene un bucle de enganche de fase que se utiliza para aumentar la velocidad de diseño y reducir el uso de recursos, para ello se hace uso de una sincronización que reduce el retraso del reloj del dispositivo.

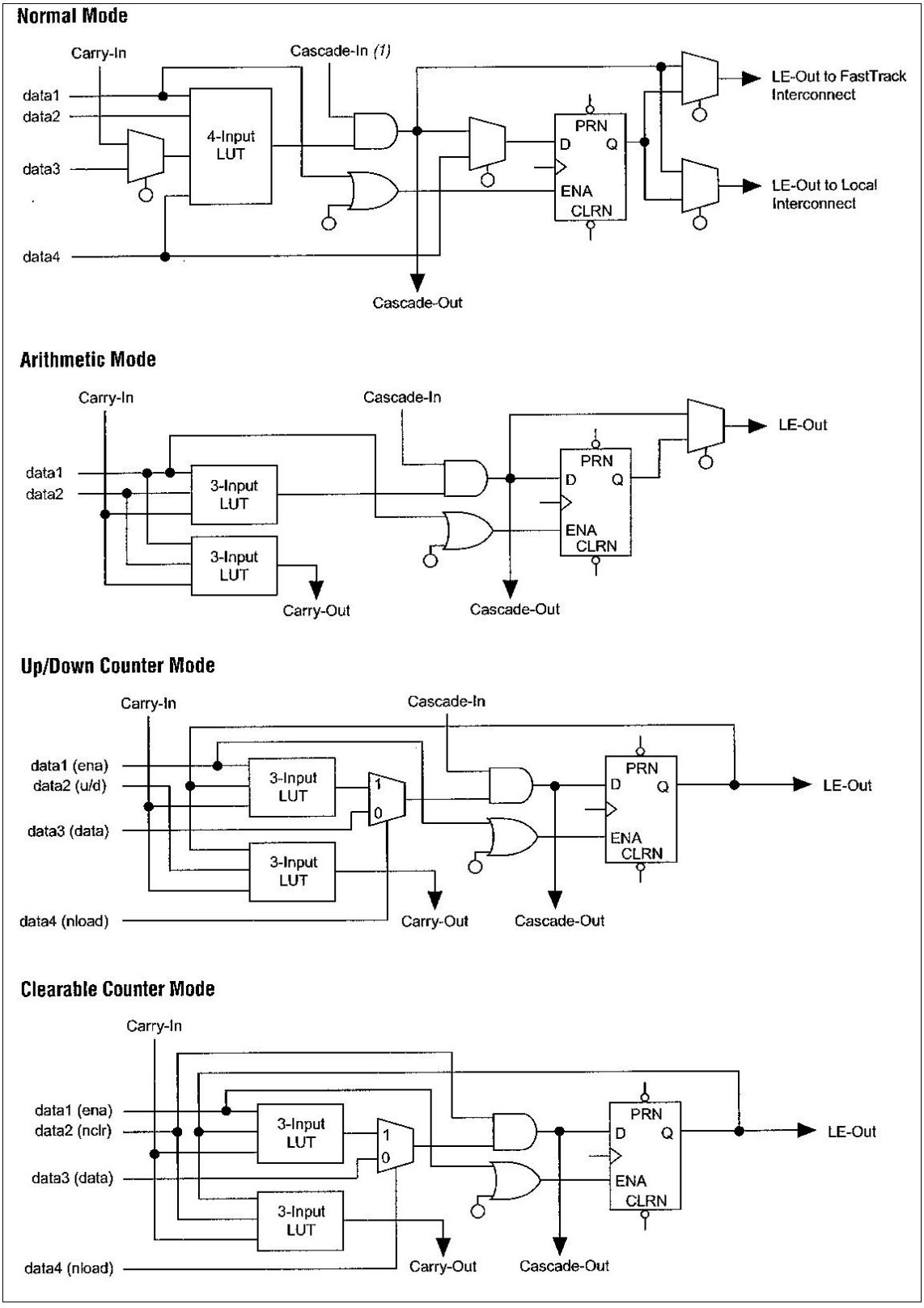


Figura 1.7.2-6 Modos de Operación de Elementos Lógicos (LE). Cortesía Altera Corporation.

A continuación se en la tabla 1.9 se muestran los valores absolutos del dispositivo FLEX 10K.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply Voltage	With respect to ground	-2.0	7.0	V
V_I	DC input Voltage		-2.0	7.0	V
I_{OUT}	DC output Voltage		-25	25	mA
T_{STG}	Storage Temperature	No bias	-65	150	°C
T_{AMB}	Ambient Temperature	Under bias	-65	135	°C
T_J	Junction Temperature	Ceramic packages, under bias		150	°C
		PQFP, TQPF, RQFP packages under bias		135	

Tabla 1.9 Valores absolutos de la Familia FLEX 10K. Cortesía Altera Corporation.

Consumo de potencia.

La fuente de alimentación P de la familia de dispositivos FLEX 10K se calcula a través de la siguiente expresión:

$$P = P_{INT} + P_{IO} = (I_{CCSTANDBY} + I_{CCACTIVE}) \times V_{CC} + P_{IO}$$

El valor de la corriente $I_{CCACTIVE}$ es calculado con la siguiente ecuación:

$$I_{CCACTIVE} = K \times f_{MAX} \times N \times tog_{LC} \times \frac{\mu A}{MHz \times LE}$$

Donde :

f_{MAX} = Frecuencia máxima de operación en MHz.

N = Número total de células lógicas usadas en el dispositivo.

tog_{LC} = Promedio en porciento de células lógicas alternadas en cada reloj.

K = Constante del dispositivo.

A continuación se ilustran las gráficas de salida de los dispositivos FLEX 10K, figura 1.8.7

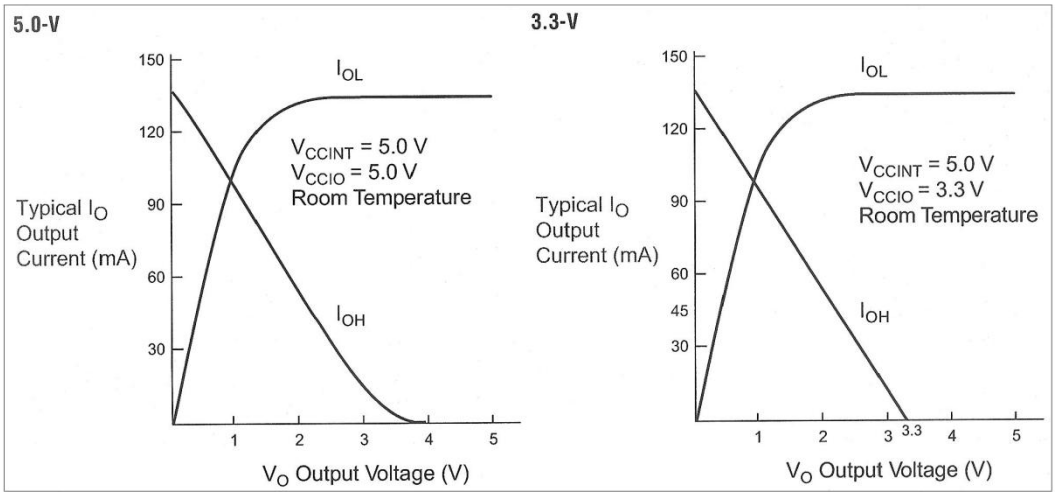


Figura 1.7.2-7 Características de salida de los dispositivos FLEX 10K. Cortesía Altera Corporation.

1.7.3 Lenguaje VHDL.

VHDL proviene de VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. El cual es un lenguaje de descripción y modelado diseñado para describir (en una forma que los humanos y las máquinas puedan leer y entender) la funcionalidad y la organización de sistemas hardware digitales, placas de circuitos, y componentes.
El lenguaje VHDL fue desarrollado como un lenguaje para el modelado y simulación lógica dirigida por eventos de sistemas digitales, y actualmente se utiliza también para la síntesis automática de circuitos. El VHDL fue desarrollado de forma muy parecida al ADA debido a que el ADA fue también propuesto como un lenguaje puro pero que tuviera estructuras y elementos sintácticos que permitieran la programación de cualquier sistema hardware sin limitación de la arquitectura. El ADA tenía una orientación hacia sistemas en tiempo real y al hardware en general, por lo que se lo escogió como modelo para desarrollar el VHDL.
VHDL es un lenguaje con una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento hardware. VHDL permite el modelado preciso en distintos estilos del comportamiento de un sistema digital conocido y el desarrollo de modelos de simulación.
Uno de los objetivos del lenguaje VHDL es el modelado. Modelado es el desarrollo de un modelo para simulación de un circuito o sistema previamente implementado cuyo comportamiento, por tanto, se conoce. El objetivo del modelado es la simulación.
Otro de los usos de este lenguaje es la síntesis automática de circuitos. En el proceso de síntesis, se parte de una especificación de entrada con un determinado nivel de abstracción, y se llega a una implementación más detallada, menos abstracta. Por tanto, la síntesis es una tarea vertical entre niveles de abstracción, del nivel más alto en la jerarquía de diseño se va hacia el más bajo nivel de la jerarquía.
Aunque puede ser usado de forma general para describir cualquier circuito se usa principalmente para programar PLD (*Programable Logic Device* - Dispositivo Lógico Programable), FPGA (*Field Programmable Gate Array*), ASIC y similares.
Algunas ventajas del uso de VHDL para la descripción hardware son:

1. VHDL permite diseñar, modelar, y comprobar un sistema desde un alto nivel de abstracción bajando hasta el nivel de definición estructural de compuertas.
2. Circuitos descritos utilizando VHDL, siguiendo unas guías para síntesis, pueden ser utilizados por herramientas de síntesis para crear implementaciones de diseños a nivel de compuertas.

3. Al estar basado en un estándar (IEEE Std 1076-1987) los ingenieros de toda la industria de diseño pueden usar este lenguaje para minimizar errores de comunicación y problemas de compatibilidad.
4. VHDL permite diseño Top-Down, esto es, permite describir (modelar) el comportamiento de los bloques de alto nivel, analizándolos (simulación), y refinar la funcionalidad de alto nivel requerida antes de llegar a niveles más bajos de abstracción de la implementación del diseño.
5. Modularidad: VHDL permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades más pequeñas.

VHDL describe Estructura y Comportamiento.

Existen dos formas de describir un circuito. Por un lado se puede describir un circuito indicando los diferentes componentes que lo forman y su interconexión, de esta manera tenemos especificado un circuito y sabemos cómo funciona; esta es la forma habitual en que se han venido describiendo circuitos y las herramientas utilizadas para ello han sido los esquemas y las descripciones netlist.
La segunda forma consiste en describir un circuito indicando lo que hace o como funciona, es decir, describiendo su comportamiento. Naturalmente esta forma de describir un circuito es mucho mejor para un diseñador puesto que lo que realmente lo que interesa es el funcionamiento del circuito más que sus componentes. Por otro lado, al encontrarse lejos de lo que un circuito es realmente puede plantear algunos problemas a la hora de realizar un circuito a partir de la descripción de su comportamiento.
El VHDL es interesante puesto que va a permitir los dos tipos de descripciones:

Estructura: VHDL puede ser usado como un lenguaje de Netlist normal y corriente donde se especifican por un lado los componentes del sistema y por otro sus interconexiones.
Comportamiento: VHDL también se puede utilizar para la descripción funcional de un circuito. Esto es lo que lo distingue de un lenguaje de Netlist. Sin necesidad de conocer la estructura interna de un circuito es posible describirlo explicando su funcionalidad. Esto es especialmente útil en simulación ya que permite simular un sistema sin conocer su estructura interna, pero este tipo de descripción se está volviendo cada día más importante porque las actuales herramientas de síntesis permiten la creación automática de circuitos a partir de una descripción de su funcionamiento.

Estructura de programa.

VHDL fue diseñado en base a los principios de la programación estructurada. La idea es definir la interfaz de un modulo de hardware mientras deja invisible sus detalles internos. La entidad (ENTITY) en VHDL es simplemente la declaración de las entradas y salidas de un modulo mientras que la arquitectura (ARCHITECTURE) es la descripción detallada de la estructura interna del modulo o de su comportamiento. A continuación se describe el concepto anterior. Muchos

diseñadores conciben la Entity como una funda de la arquitectura dejando invisible los detalles de lo que hay dentro (architecture). Esto forma la base de un sistema de diseño jerárquico, la arquitectura de la entidad de más nivel (top level) puede usar otras entidades dejando invisible los detalles de la arquitectura de la identidad de menos nivel. En la descripción, las entidades B, E y F no utilizan a otras entidades. Mientras que la entidad A utiliza a todas las demás. A la pareja entidad-arquitectura se le llama modelo. En un fichero texto VHDL la entidad y la arquitectura se escriben separadas, por ejemplo a continuación se muestra un programa muy simple en VHDL de una compuerta de 2 entradas. Como en otros programas VHDL ignora los espacios y saltos de líneas. Los comentarios se escribieron con 2 guiones (--) y termina al final de la línea. En la figura siguiente se muestra la estructura de un modelo en VHDL. SINTASIS PARA LA DECLARACION DE LA ENTIDAD VHDL define muchos caracteres especiales llamados “palabras reservadas”. Aunque las palabras reservadas no son sensibles a las mayúsculas o minúsculas, el ejemplo que sigue las utilizaremos en mayúsculas para identificarlas.

```
ENTITY Nombre_entidad IS
PORT ( Nombre de señal: modo tipo de señal;
                . . .
       Nombre de señal:  modo    tipo de señal ) ;
END nombre_entidad ;
```

Ejemplo básico de descripción VHDL.

Ejemplo 1.7 Describir en VHDL un circuito que multiplexe dos líneas (a y b) de un bit, a una sola línea (salida) también de un bit; la señal `selec` sirve para indicar que a la salida se tiene la línea a (selec='0') o b (selec='1').
En la figura 1.7.3 se muestra el circuito implementado con puertas lógicas que realiza la función de multiplexación.
Lo que se va a realizar a continuación es la descripción comportamental del circuito de la figura 1.7.3 y luego se realizará la descripción estructural para ver las diferencias. Más adelante se verá que hay dos tipos de descripción comportamental, pero de momento, el presente ejemplo únicamente pretende introducir el lenguaje VHDL y su estructura.

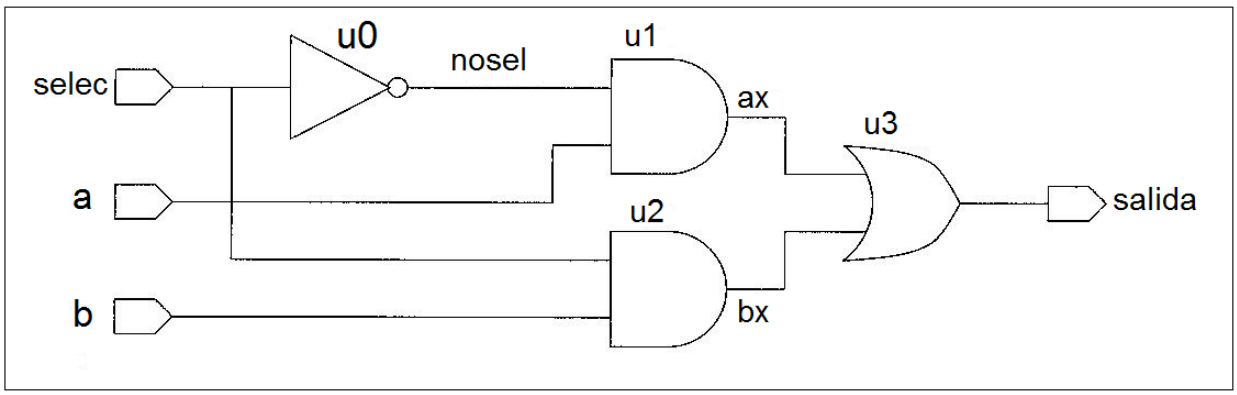


Figura 1.7.3 Esquema del ejemplo básico en VHDL.

La sintaxis del VHDL no es sensible a mayúsculas o minúsculas por lo que se puede escribir como se prefiera. A lo largo de las explicaciones se intentará poner siempre las palabras claves del lenguaje en mayúsculas para distinguirlas de las variables y otros elementos.
En primer lugar, sea el tipo de descripción que sea, hay que definir el símbolo o entidad del circuito. Lo primero es definir las entradas y salidas del circuito, es decir, la caja negra que lo define. Se le llama entidad porque en la sintaxis de VHDL esta parte se declara con la palabra clave ENTITY. Esta definición de entidad, que suele ser la primera parte de toda descripción VHDL, se expone a continuación:

```
-- Los comentarios empiezan por dos guiones
ENTITY mux IS
PORT ( a: IN bit;
b: IN bit;
selec: IN bit;
salida: OUT bit);
END mux;
```

Esta porción del lenguaje indica que la entidad mux (que es el nombre que se le ha dado al circuito) tiene tres entradas de tipo bit, y una salida también del tipo bit. El tipo bit simplemente indica una línea que puede tomar los valores '0' y '1'.

La entidad de un circuito es única, sin embargo, se mostró que un mismo símbolo, en este caso entidad, podía tener varias vistas o en el caso de arquitecturas VHDL. Cada bloque de arquitectura, que es donde se describe el circuito, puede ser una representación diferente del mismo circuito. Por ejemplo, puede haber una descripción estructural y otra de comportamiento o funcional, ambas son descripciones diferentes, pero ambas descripciones corresponden al mismo circuito, símbolo, o entidad. Veamos entonces la descripción de comportamiento:

```
ARCHITECTURE comportamental OF mux IS
BEGIN
PROCESS(a,b,selec)
BEGIN
IF (selec='0') THEN
salida<=a;
ELSE
salida<=b;
END IF;
END PROCESS;
END comportamental;
```

Elementos sintácticos del VHDL.

El lenguaje VHDL es verdaderamente un lenguaje, por lo que tiene sus elementos sintácticos, sus tipos de datos, y sus estructuras como cualquier otro tipo de lenguaje.
El hecho de que sirva para la descripción hardware lo hace un poco diferente de un lenguaje convencional. Una de estas diferencias es probablemente la posibilidad de ejecutar instrucciones a la vez de forma concurrente.
Algunos de estos elementos sintácticos se muestran a continuación:
Comentarios: Cualquier línea que empieza por dos guiones "`--`" es un comentario.
Identificadores: Son cualquier cosa que sirve para identificar variables, señales, nombres de rutina, etc. Puede ser cualquier nombre compuesto por letras incluyendo el símbolo de subrayado "`_`". Las mayúsculas y minúsculas son consideradas iguales, así que `JOSE` y `jose` representan el mismo elemento. No puede haber ningún identificador que coincida con alguna de las palabras clave del VHDL.
Números: Cualquier número se considera que se encuentra en base 10. Se admite la notación científica convencional para números en coma flotante. Es posible poner números en otras bases utilizando el símbolo del sostenido "`#`". Ejemplo: `2#11000100# y 16#C4#` representan el entero `196`.
Caracteres: Es cualquier letra o carácter entre comillas simples: '`1`','`3`','`t`'.
Cadenas: Son un conjunto de caracteres englobados por comillas dobles: "`Esto es una cadena`".
Cadenas de bits: Los tipos `bit` y `bit_vector` son en realidad de tipo carácter y matriz de caracteres respectivamente. En VHDL se tiene una forma elegante de definir números con estos tipos y es mediante la cadena de bits. Dependiendo de la base en que se especifique el número se puede poner un prefijo B (binario), O (octal), o X (hexadecimal). Ejemplo: `B"11101001", O"126", X"FE".`

Operadores y expresiones.

Las expresiones en VHDL son prácticamente iguales a como pudieran ser en otros lenguajes de programación o descripción, por lo que se expondrán brevemente los existentes en VHDL y su utilización.

Operadores varios.

& (Concatenación) Concatena matrices de manera que la dimensión de la matriz resultante es la suma de las dimensiones de las matrices sobre las que opera:
punto<=x&y mete en la matriz punto la matriz x en las primeras posiciones, y la matriz y en las _ultimas.
Operadores aritméticos.

****** (exponencial) Sirve para elevar un número a una potencia: 4**2 es 42. El operador de la izquierda puede ser entero o real, pero el de la derecha sólo puede ser entero.
ABS() (valor absoluto) Como su propio nombre indica esta función devuelve el valor absoluto de su argumento que puede ser de cualquier tipo numérico.
***** (Multiplicación) Sirve para multiplicar dos números de cualquier tipo (los tipos `bit` o `bit_vector` no son numéricos).
/ (División) También funciona con cualquier dato de tipo numérico.
MOD (módulo) Calcula en módulo de dos números. Exactamente se define el módulo como la operación que cumple: a=b*N+(a MOD b) donde N es un entero. Los operandos sólo pueden ser enteros. El resultado toma el signo de b.
REM (resto) Calcula el resto de la división entera y se define como el operador que cumple: a=(a/b)*b+(a REM b), siendo la división entera. Los operandos sólo pueden ser enteros. El resultado toma el signo de a.
+ (suma y signo positivo) Este operador sirve para indicar suma, si va entre dos operandos, o signo, si va al principio de una expresión. La precedencia es diferente en cada caso. Opera sobre valores numéricos de cualquier tipo.
- (resta y signo negativo) Cuando va entre dos operandos se realiza la operación de sustracción, y si va delante de una expresión le cambia el signo. Los operandos pueden ser numéricos de cualquier tipo.

Operadores de desplazamiento.

SLL, SRL (desplazamiento lógico a izquierda y a derecha). Desplaza un vector un número de bits a izquierda (`SLL`) o derecha (`SRL`) rellenando con ceros los huecos libres. Se utiliza en notación infija de manera que la señal a la izquierda del operador es el vector que se quiere desplazar y el de la derecha es un valor que indica el número de bits a desplazar. Por ejemplo `dato SLL 2` desplaza a izquierda el vector dato, es decir, lo multiplica por 4.
SLA, SRA (desplazamiento aritmético a izquierda y derecha).
ROL, ROR (rotación a izquierda y a derecha). Es como el de desplazamiento pero los huecos son ocupados por los bits que van quedando fuera.

Operadores Relacionales.

Devuelven siempre un valor de tipo booleano (`TRUE` o `FALSE`). Los tipos con los que pueden operar dependen de la operación:
`=, /=` (igualdad). El primero devuelve `TRUE` si los operandos son iguales y `FALSE` en caso contrario. El segundo indica desigualdad, así que funciona justo al revés.
Los operandos pueden ser de cualquier tipo con la condición de que sean ambos del mismo tipo.
`<,<=,>,>=` (menor mayor). Tienen el significado habitual. La diferencia con los anteriores es que los tipos de datos que pueden manejar son siempre de tipo escalar o matrices de una sola dimensión de tipos discretos.

Operadores lógicos.
Son `NOT`, `AND`, `NAND`, `OR`, `NOR` y `XOR`. El funcionamiento es el habitual para este tipo de operadores. Actúan sobre los tipos `bit`, `bit vector` y `boolean`. En el caso de realizarse estas operaciones sobre un vector, la operación se realiza bit a bit, incluyendo la operación `NOT`.

Enteros: Son datos cuyo contenido es un valor numérico entero. La forma es que se definen estos datos es mediante la palabra clave RANGE, es decir, no se dice que un dato es de tipo entero, sino que se dice que un dato está comprendido en cierto intervalo especificando los límites del intervalo con valores enteros.
Ejemplos:

```
TYPE byte IS RANGE 0 TO 255;
TYPE index IS RANGE 7 DOWNTO 1;
TYPE integer IS -2147483647 TO 2147483647; -- Predefinido en
el lenguaje
```

Este último tipo viene ya predefinido en el lenguaje aunque no es muy conveniente su utilización, especialmente pensando en la posterior síntesis del circuito.
Físicos: Como su propio nombre indica se trata de datos que se corresponden con magnitudes físicas, es decir, tienen un valor y unas unidades.
Ejemplo:

```
TYPE longitud IS RANGE 0 TO 1.0e9
  UNITS
     um;
     mm=1000 um;
     m=1000 mm;
     in=25.4 mm;
  END UNITS;
```

Hay un tipo físico predefinido en VHDL que es `time`. Este tipo se utiliza para indicar retrasos y tiene todos los submúltiplos, desde `fs` (femtosegundos), hasta `hr` (horas). Cualquier dato físico se escribe siempre con su valor seguido de la unidad: 10 mm, 1 in, 23 ns.

Reales: Conocidos también como coma flotante, son los tipos que definen un número real. Al igual que los enteros se definen mediante la palabra clave RANGE, con la diferencia de que los límites son números reales.
Ejemplos:

```
TYPE nivel IS RANGE 0.0 TO 5.0;
TYPE real IS RANGE -1e38 TO 1e38; -- Predefinido en el
lenguaje
```

Enumerados: Son datos que pueden tomar cualquier valor especificado en un conjunto finito o lista. Este conjunto se indica mediante una lista encerrada entre paréntesis de elementos separados por comas.
Ejemplos:

```
TYPE nivel_logico IS (nose,alto,bajo,Z);
TYPE bit IS ('0','1'); -- Predefinido en el lenguaje
```

Hay varios tipos enumerados que se encuentran predefinidos en VHDL. Estos tipos son: `severity_level, boolean, bit` y `character.`

Subtipos de datos.

VHDL permite la definición de subtipos que son restricciones o subconjuntos de tipos existentes. Hay dos tipos. El primero son subtipos obtenidos a partir de la restricción de un tipo escalar a un rango. Ejemplos:

```
SUBTYPE raro IS integer RANGE 4 TO 7;
SUBTYPE digitos IS character RANGE '0' TO '9';
SUBTYPE natural IS integer RANGE 0 TO entero_mas_alto; --
Predefinido en VHDL
SUBTYPE positive IS integer RANGE 1 TO entero_mas_alto; --
Predefinido en VHDL
```

El segundo tipo de subtipos son aquellos que restringen el rango de una matriz:

```
SUBTYPE id IS string(1 TO 20);
SUBTYPE word IS bit_vector(31 DOWNTO 0);
```

La ventaja de utilizar un subtipo es que las mismas operaciones que servían para el tipo sirven igual de bien para el subtipo. Esto tiene especial importancia por ejemplo cuando se describe un circuito para ser sintetizado, ya que si utilizamos `integer` sin más, esto se interpretará como un bus de 32 líneas (puede cambiar dependiendo de la plataforma) y lo más probable es que en realidad necesitemos muchas menos. Otro caso se da cuando tenemos una lista de cosas y les queremos asignar un entero a cada una, dependiendo de las operaciones que queramos hacer puede resultar más conveniente definirse un subtipo a partir de `integer` que crear un tipo enumerado.

Atributos.

Los elementos en VHDL, como señales, variables, etc, pueden tener información adicional llamada atributos. Estos atributos están asociados a estos elementos del lenguaje y se manejan en VHDL mediante la comilla simple “ ' “. Por ejemplo, `t'LEFT` indica el atributo `'LEFT` de `t` que debe ser un tipo escalar (este atributo indica el límite izquierdo del rango).

Hay algunos de estos atributos que están predefinidos en el lenguaje y a continuación se muestran los más interesantes. Suponiendo que `t` es un tipo escalar tenemos los siguientes atributos:

`t'LEFT` Límite izquierdo del tipo `t`.

`t'RIGHT` Límite derecho del tipo `t` .

`t'LOW` Límite inferior del tipo `t`.

`t'HIGH` Límite superior del tipo `t`.

Para tipos `t`, `x` miembro de este tipo, y `N` un entero, se pueden utilizar los siguientes atributos:

`t'POS(x)` Posición de x dentro del tipo `t`.

`t'VAL(N)` Elemento `N` del tipo `t`.

`t'LEFTOF(x)` Elemento que está a la izquierda de `x` en `t`.
`t'RIGHTOF(x)` Elemento que está a la derecha de `x` en `t`.
`t'PRED(x)` Elemento que está delante de `x` en `t`.
`t'SUCC(x)` Elemento que está detrás de `x` en `t`.

Para a siendo un tipo u elemento de tipo matriz, y `N` un entero de 1 a al número de dimensiones de la matriz, se pueden usar los siguientes atributos:
`a'LEFT(N)` Límite izquierdo del rango de dimensión `N` de `a`.
`a'RIGHT(N)` Límite derecho del rango de dimensión `N` de `a`.
`a'LOW(N)` Límite inferior del rango de dimensión `N` de `a`.
`a'HIGH(N)` Límite superior del rango de dimensión `N` de `a`.
`a'RANGE(N)` Rango del índice de dimensión `N` de `a`.
`a'LENGTH(N)` Longitud del índice de dimensión `N` de `a`.
Suponiendo que `s` es una señal, se pueden utilizar los siguientes atributos (se han escogido los más interesantes):
s'EVENT Indica si se ha producido un cambio en la señal.
s'STABLE(t) Indica si la señal estuvo estable durante el último periodo `t`.
El atributo `'EVENT` es especialmente útil en la definición de circuitos secuenciales para detectar el flanco de subida o bajada de la señal de reloj. Es por esto que es probablemente el atributo más utilizado en VHDL.

Declaración de constantes, variables y señales.

Un elemento en VHDL contiene un valor de un tipo especificado. Para ello existen tres tipos de elementos en VHDL, están las variables, las señales y las constantes. Las variables y constantes son similares a las que encontramos en cualquier lenguaje. A diferencia de las señales, que son elementos cuyo significado es muy diferente y es consecuencia directa de que aunque VHDL es un lenguaje muy parecido a los convencionales, no deja en ningún momento de ser un lenguaje de descripción hardware por lo que se encuentran algunas diferencias.

Constantes.

Una constante es un elemento que se inicializa a un determinado valor y no puede ser cambiado una vez inicializado, conservando para siempre su valor. Ejemplos:

```
CONSTANT e: real :=2.71828;
CONSTANT retraso: time :=10ns;
CONSTANT max_size: natural;
```

En la última sentencia la constante `max_size` no tiene ningún valor asociado. Esto solamente se permite cuando el valor sea declarado en algún otro sitio.

Variables.

Una variable es similar a una constante con la diferencia de que su valor puede ser alterado en cualquier instante. A las variables también se les puede asignar un valor inicial. Ejemplo:

```
VARIABLE contador: natural :=0;
VARIABLE aux: bit_vector(31 DOWNTO 0);
```

Señales.

Las señales son declaradas de la misma forma que las constantes y variables sólo que las señales pueden ser de varios tipos, que son normal, register y bus. Por defecto son de tipo normal. Al igual que en variables y constantes, a las señales se les puede dar un valor inicial si es requerido. Ejemplos:

```
SIGNAL selec: bit := "0";
SIGNAL datos: bit_vector(7 DOWNTO 0)BUS :=B"00000000";
```

Constantes señales y variables se procesan diferente. Las variables, por ejemplo, sólo tienen sentido dentro de un proceso (PROCES) o un subprograma, esto es, que solo tienen un sentido en entornos de programación donde las sentencias son ejecutadas en serie, por lo tanto las variables sólo se declaran en los procesos o subprogramas. Las señales pueden ser declaradas únicamente en las arquitecturas, paquetes (PACKAGE), o en lo bloques concurrentes (BLOCK). Las constantes pueden ser generalmente declaradas en los mismos sitios que las variables y señales.

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.

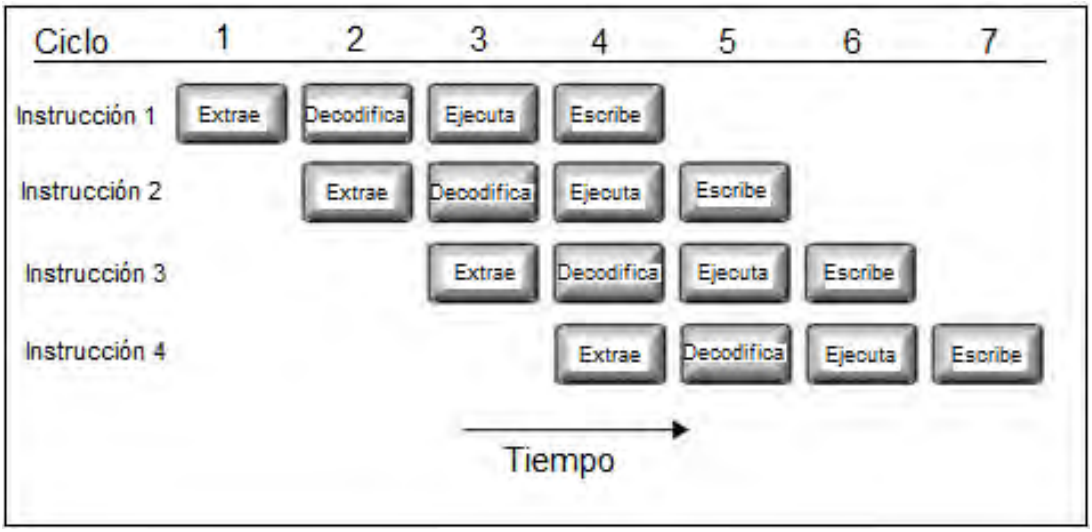


Figura 2.2 Arquitectura Sparc de procesamiento en serie.

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.

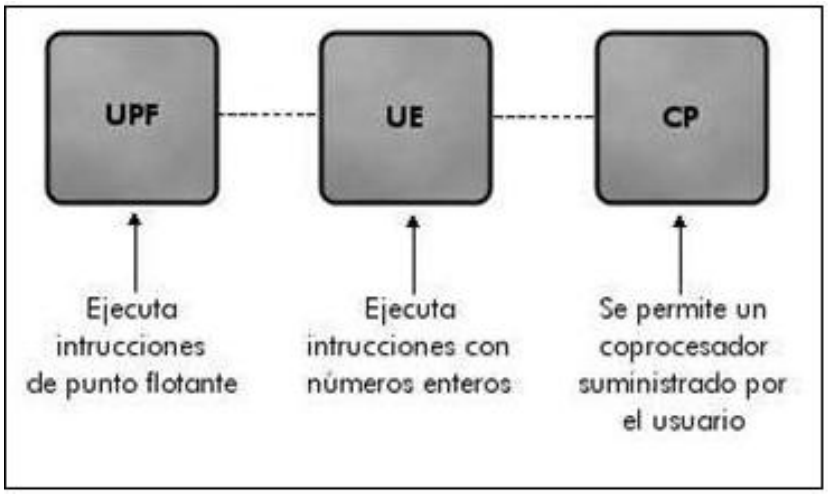


Figura 2.2-2 Diagrama de un procesador SPARC.

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).

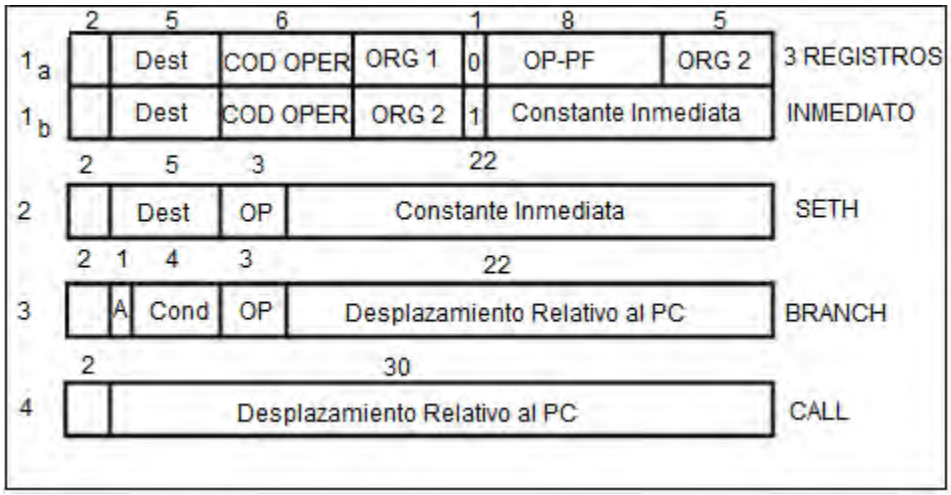


Figura 2.2-4 Formato de Instrucción SPARC.

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)

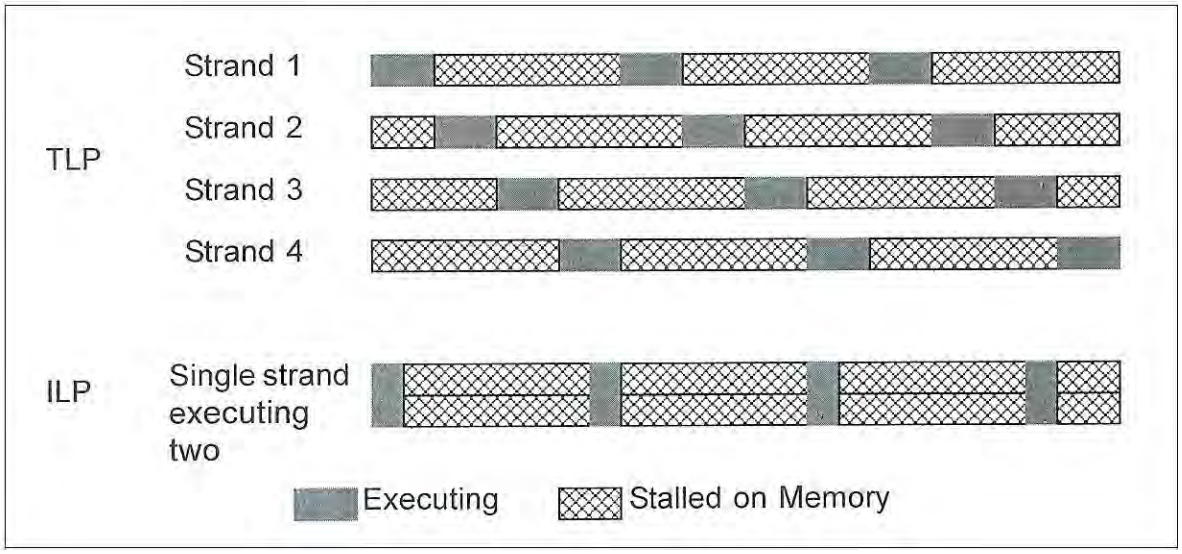


Figura 2.2-9 Diferencias entre TLP e ILP.

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.

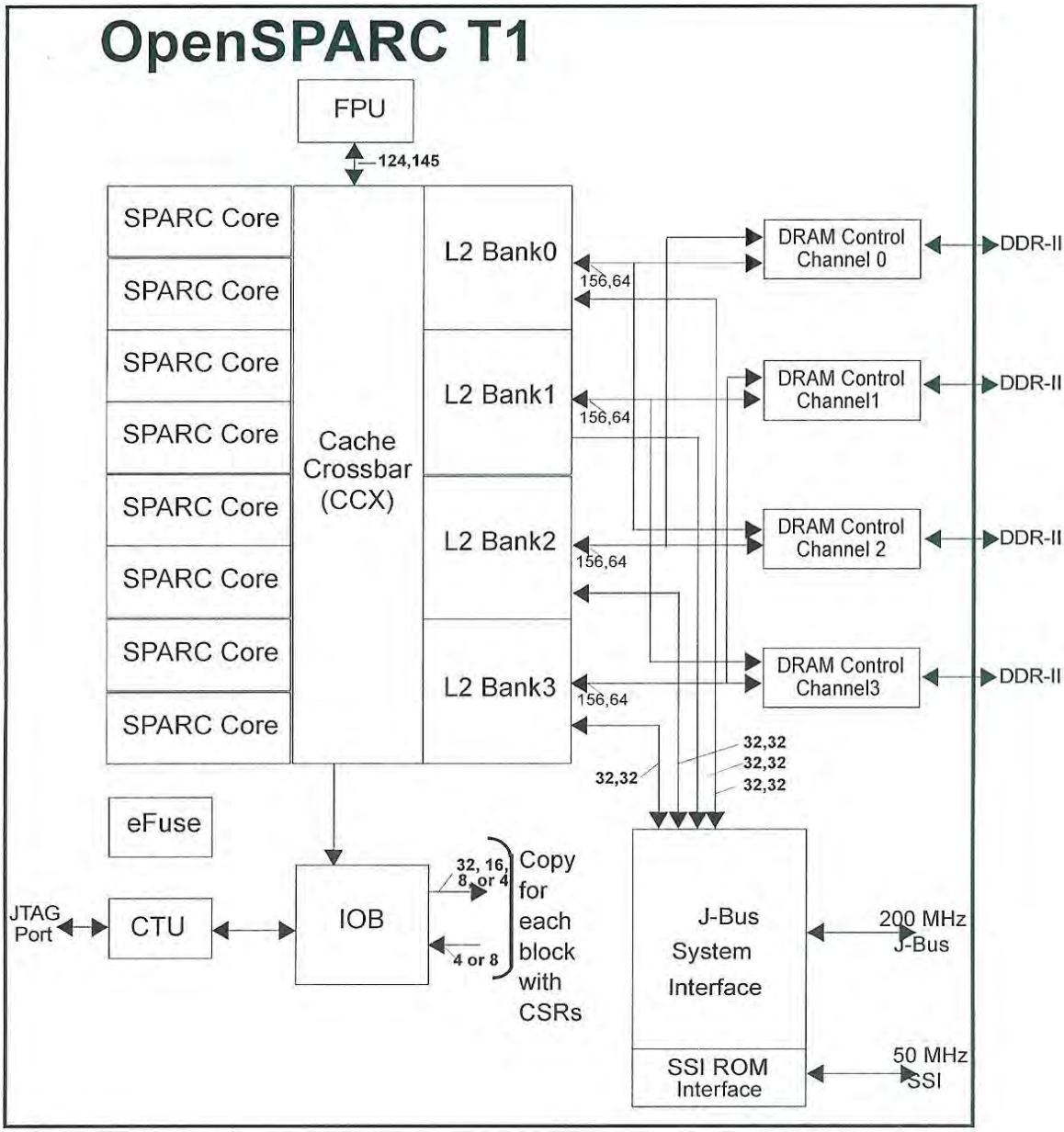


Figura 2.2.9-1 Diagrama de bloques del chip OpenSPARC T1. Cortesía OpenSPARC.

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.

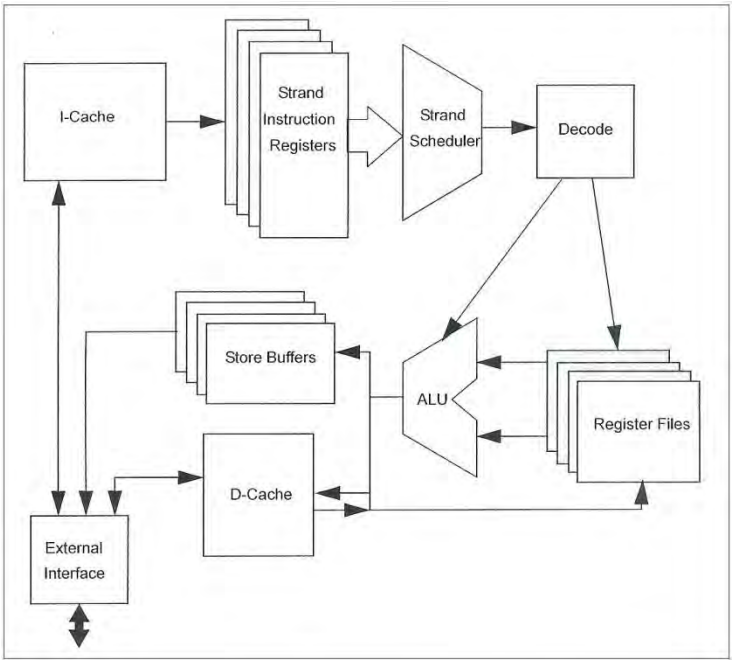


Figura 2.2.9-2 Diagrama de bloques del núcleo OpenSPARC. Cortesía OpenSPARC.

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).

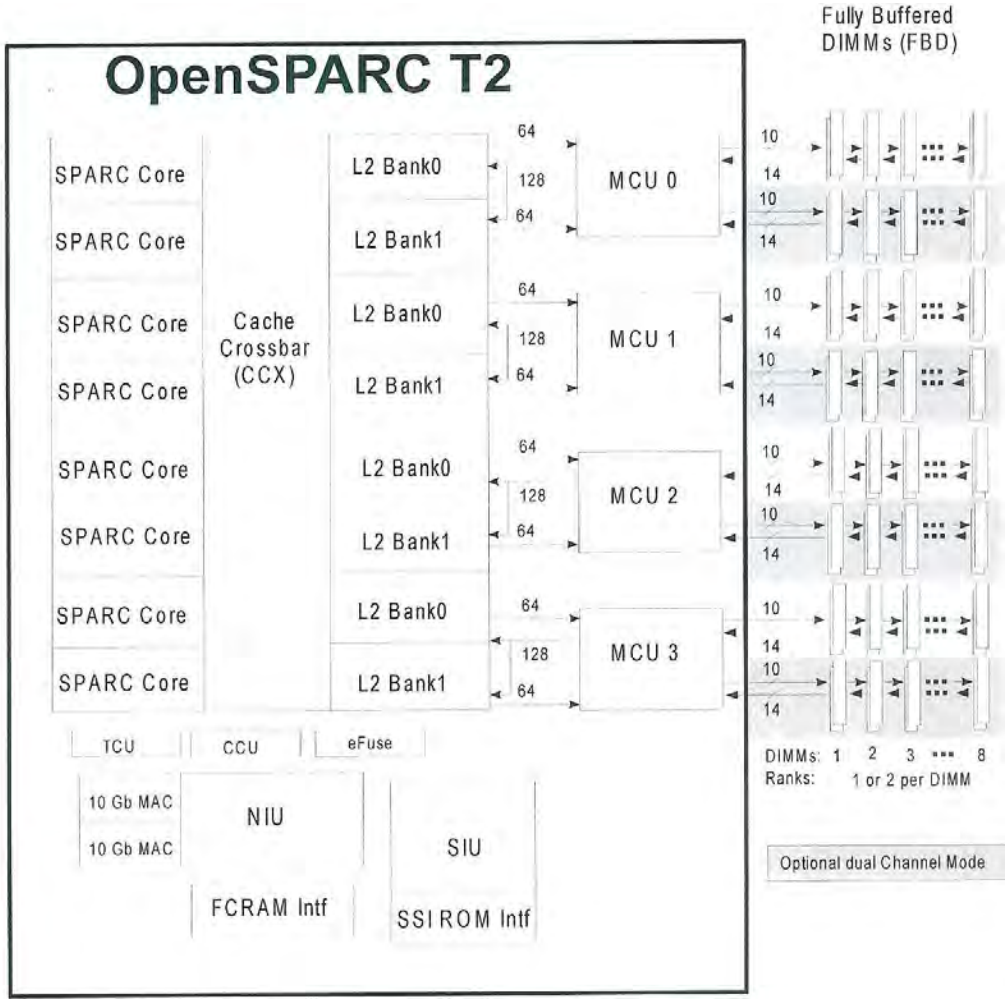


Figura 2.2.9-3 Diagrama de bloques del OpenSPARC T2. Cortesía de OpenSPARC.

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.

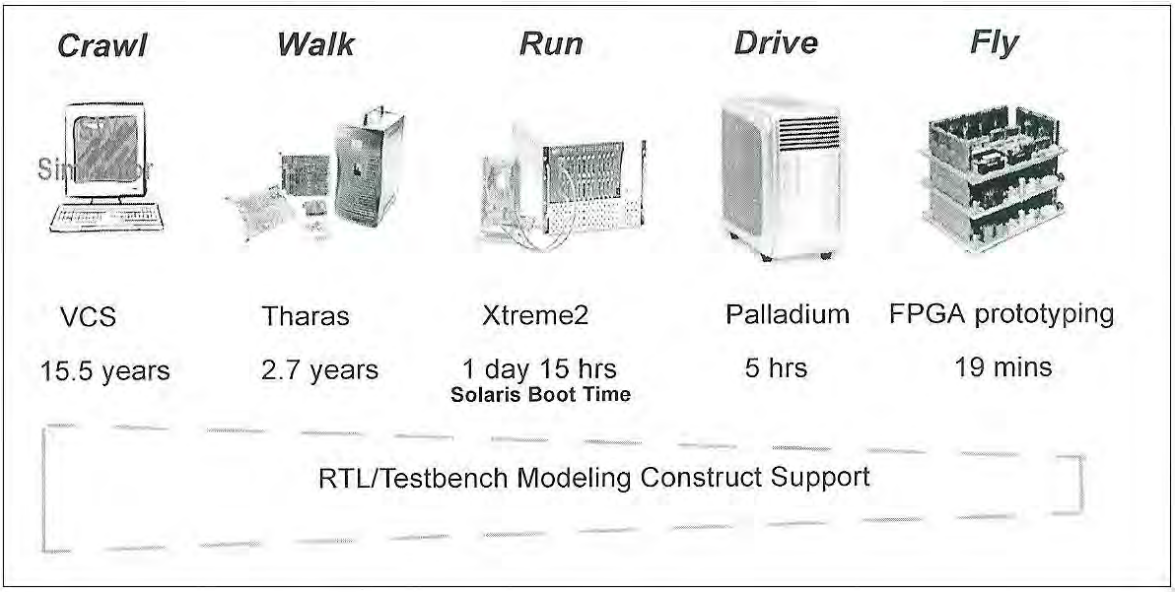


Figura 2.2.9-4 Plataformas de simulación. Cortesía OpenSPARC.

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.

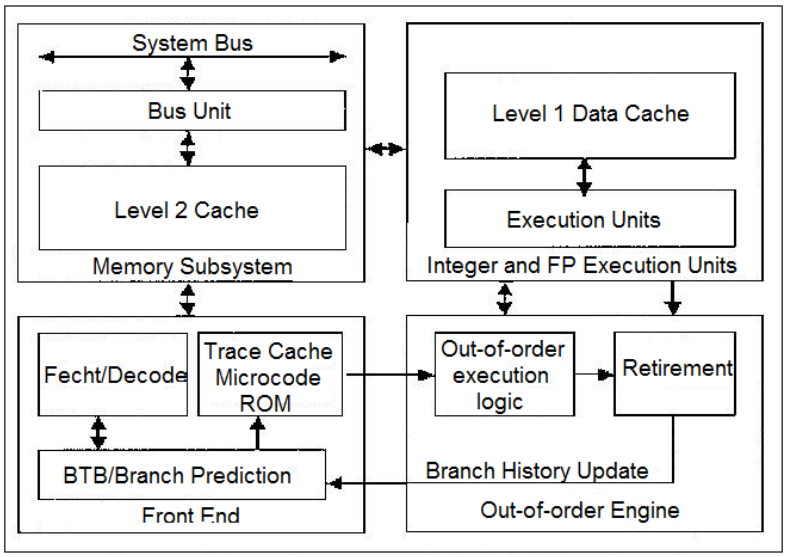


Figura 2.3.1 Arquitectura Global del Pentium 4. Cortesía The Microarchitecture of the Pentium 4 Processor.

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.

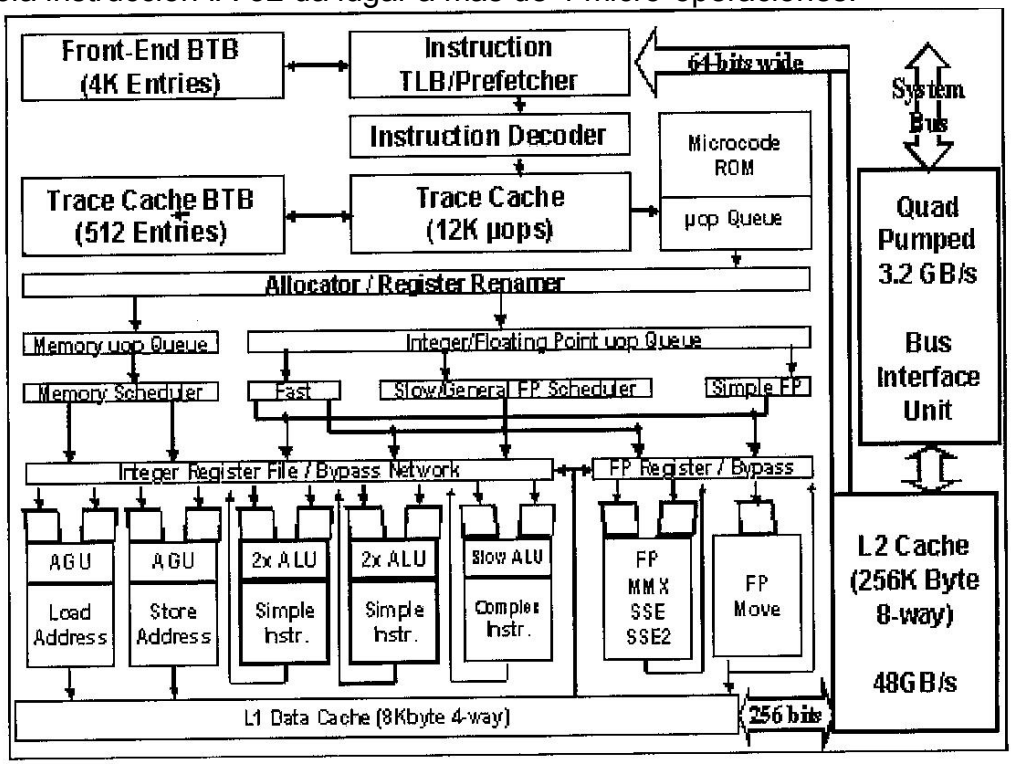


Figura 2.3.3 Microarquitectura del Pentium 4.

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.

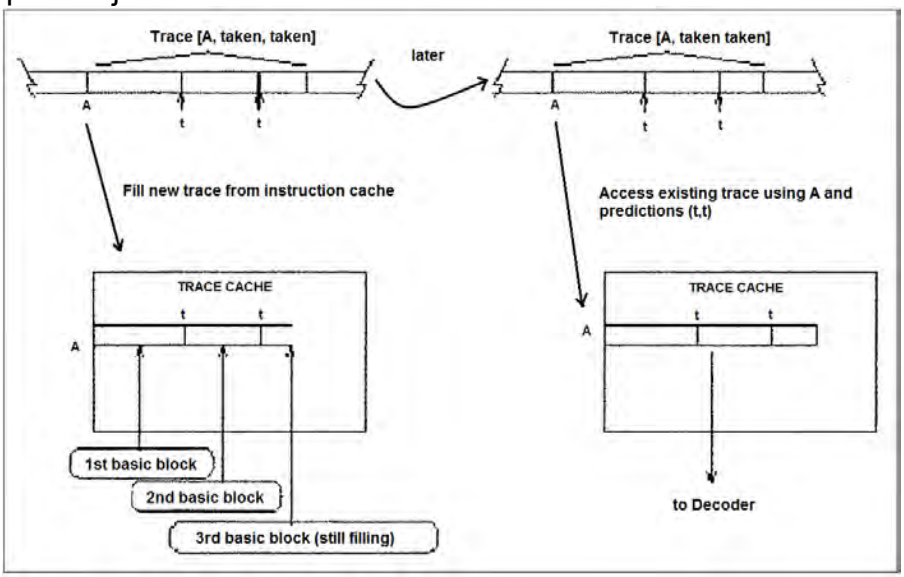


Figura 2.3.4 Ejemplo de funcionamiento de la Trace Cache.

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.

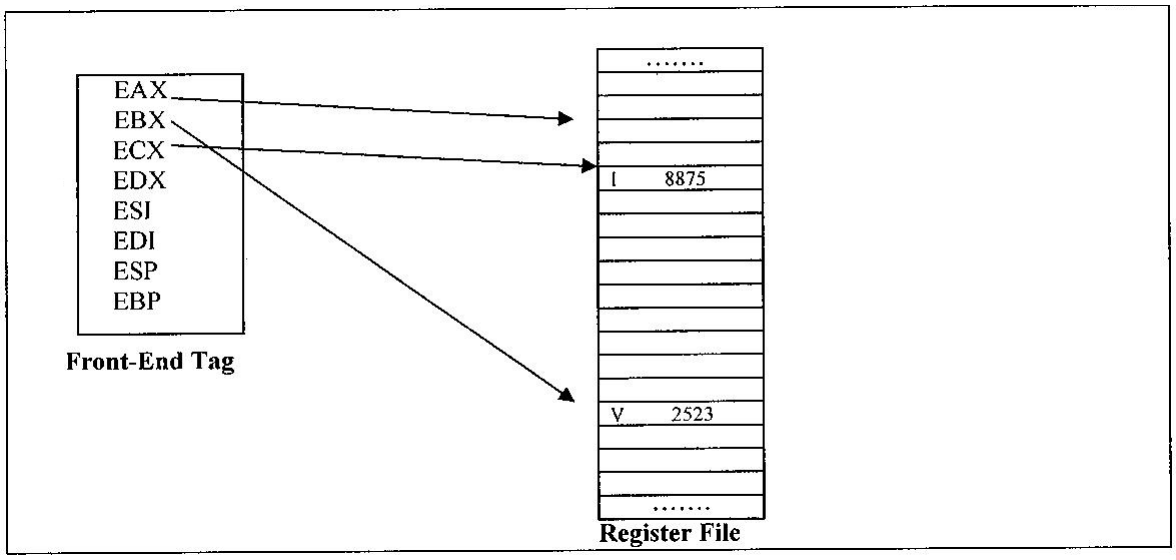


Figura 2.3.6 Situación tras renombrar la instrucción.

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.

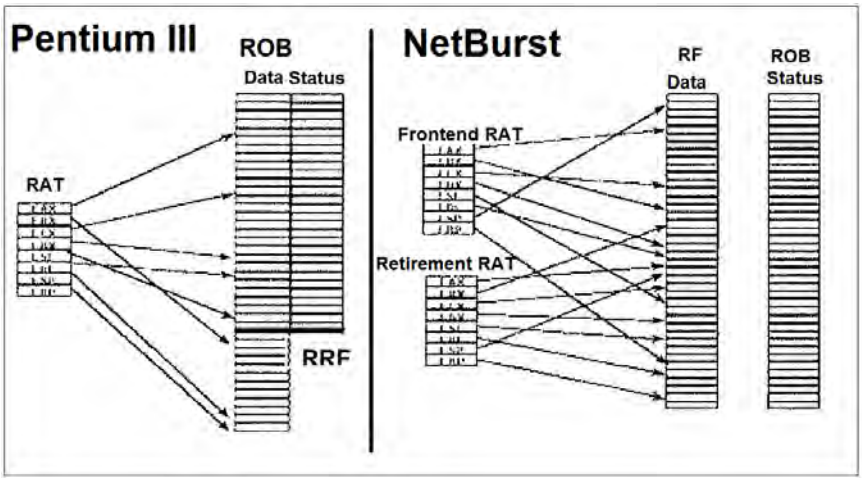


Figura 2.3.6-1 Renombramiento de registros en el Pentium III y en el Pentium 4.

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.

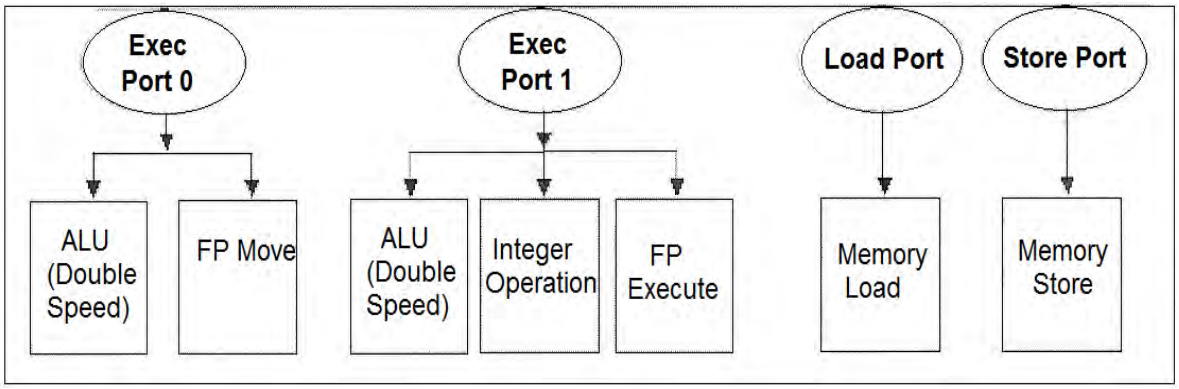


Figura 2.3.7 Puertos de ejecución y unidades funcionales.

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.

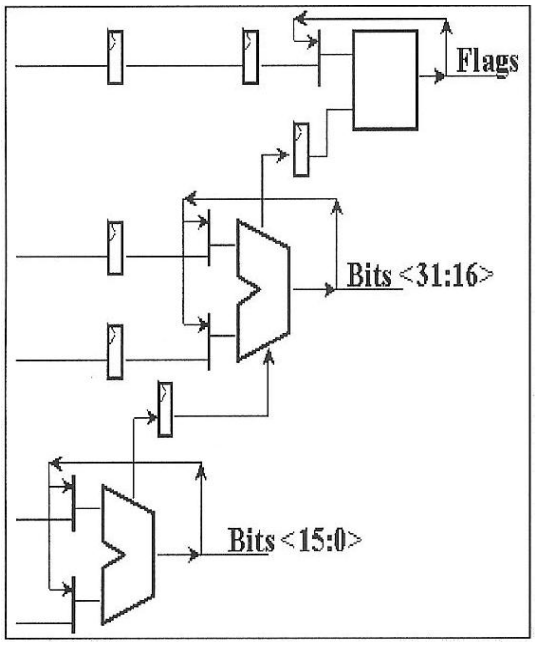


Figura 2.3.7-1 ALU de doble velocidad: staggered add.

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.

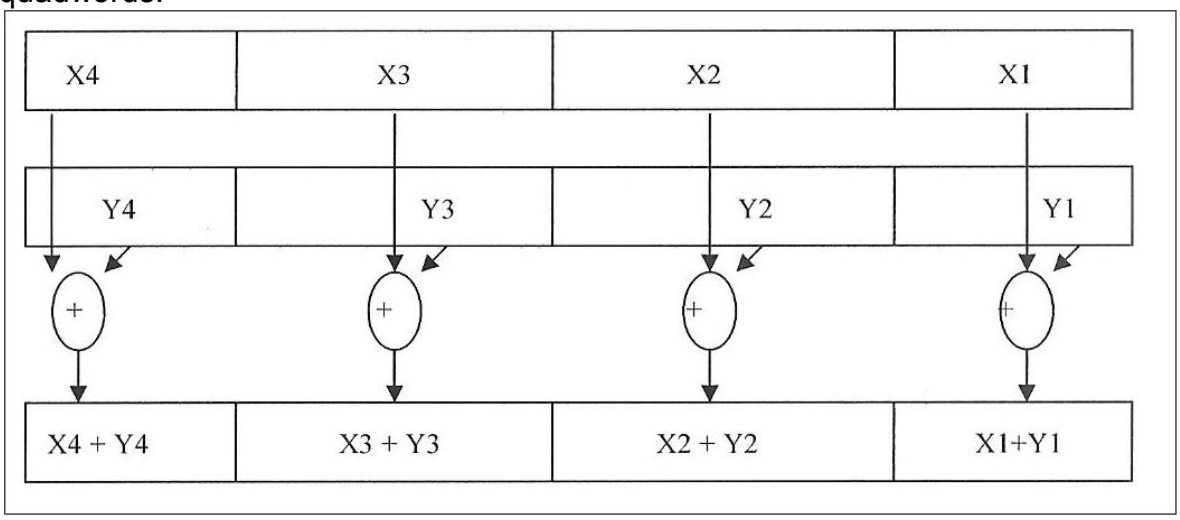


Figura 2.3.7-2 Suma SIMD.

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.

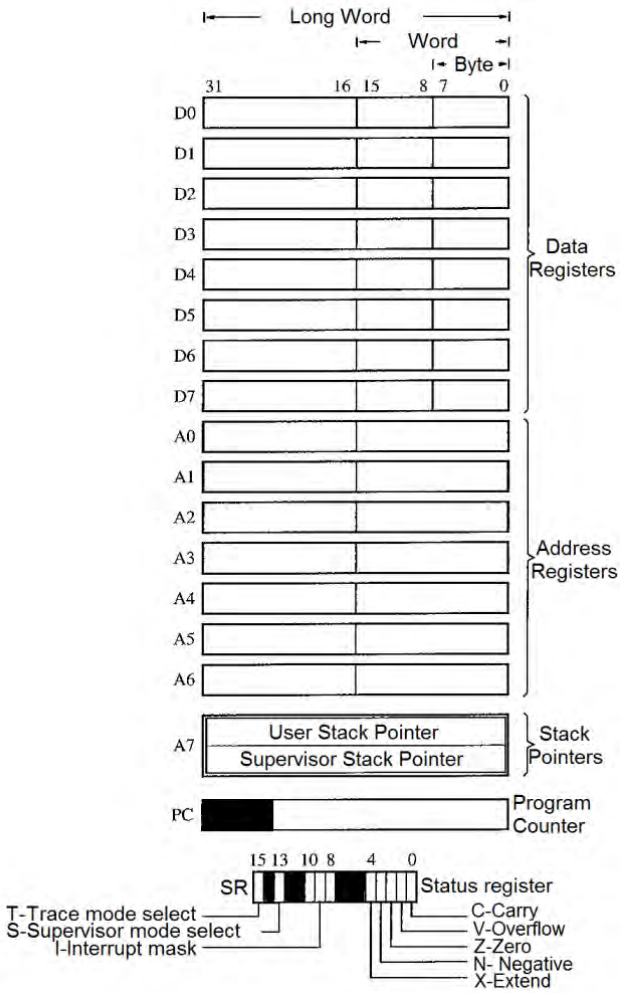
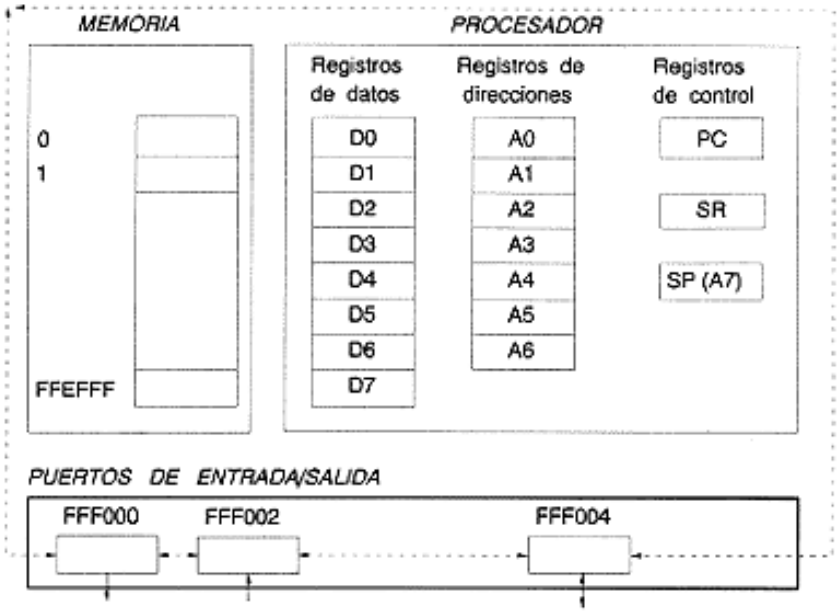


Figura 2.4-1 Estructura de Registros de la familia 68000. Cortesía Computer Organization.

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%00000000100011111 0001,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.

CAPÍTULO 3.- PROPUESTA DE ARQUITECTURA ABIERTA PARA LA ASIGNATURA DE ARQUITECTURA DE COMPUTADORAS.

3.1 INTRODUCCIÓN.

El procesador OpenSPARC T1 es el primer chip multiprocesador (CMT) que implementa totalmente el rendimiento de cualquier procesador actualmente. Es un gran procesador integrado en que se aplica la arquitectura SPARC de 64 bits. Este procesador está diseñado para aplicaciones comerciales tales como servidores de aplicación y servidores de bases de datos.

OpenSPARC T1 contiene ocho núcleos de procesador SPARC los cuales tienen soporte de hardware para cuatro subprocesos. Cada núcleo SPARC tiene una instrucción caché, una caché de datos, una instrucción totalmente asociativa y conversión de buffers. Los ocho núcleos de SPARC están conectados a través de una barra transversal a un chip de caché L2.

Los cuatro chips de memoria de acceso aleatorio dinámico (DRAM) controlan directamente la interfaz de datos. Además existe un chip controlador J-Bus que proporciona una interconexión entre las entradas y salidas del subsistema del procesador OpenSPARC T1.

Las características del procesador OpenSPARC T1 incluyen:

- 8 núcleos SPARC V9, con cuatro hilos por núcleo, por un total de 32 hilos.
- 132 Gbytes/seg de barra transversal de interconexión para la comunicación entre el chip.
- 16 Kbytes de caché de instrucción primaria por núcleo.
- 8 Kbytes de memoria caché de datos primarios por núcleo.
- 3 Mbytes de memoria caché secundaria.
- Sistema de interfaz serie (SSI) para el arranque PROM.
- Interfaz J-Bus para entrada y salida.

La figura 3.1-1 muestra un diagrama de bloques del procesador OpenSPARC T1 que ilustra las diversas interfaces y componentes integrados del chip.

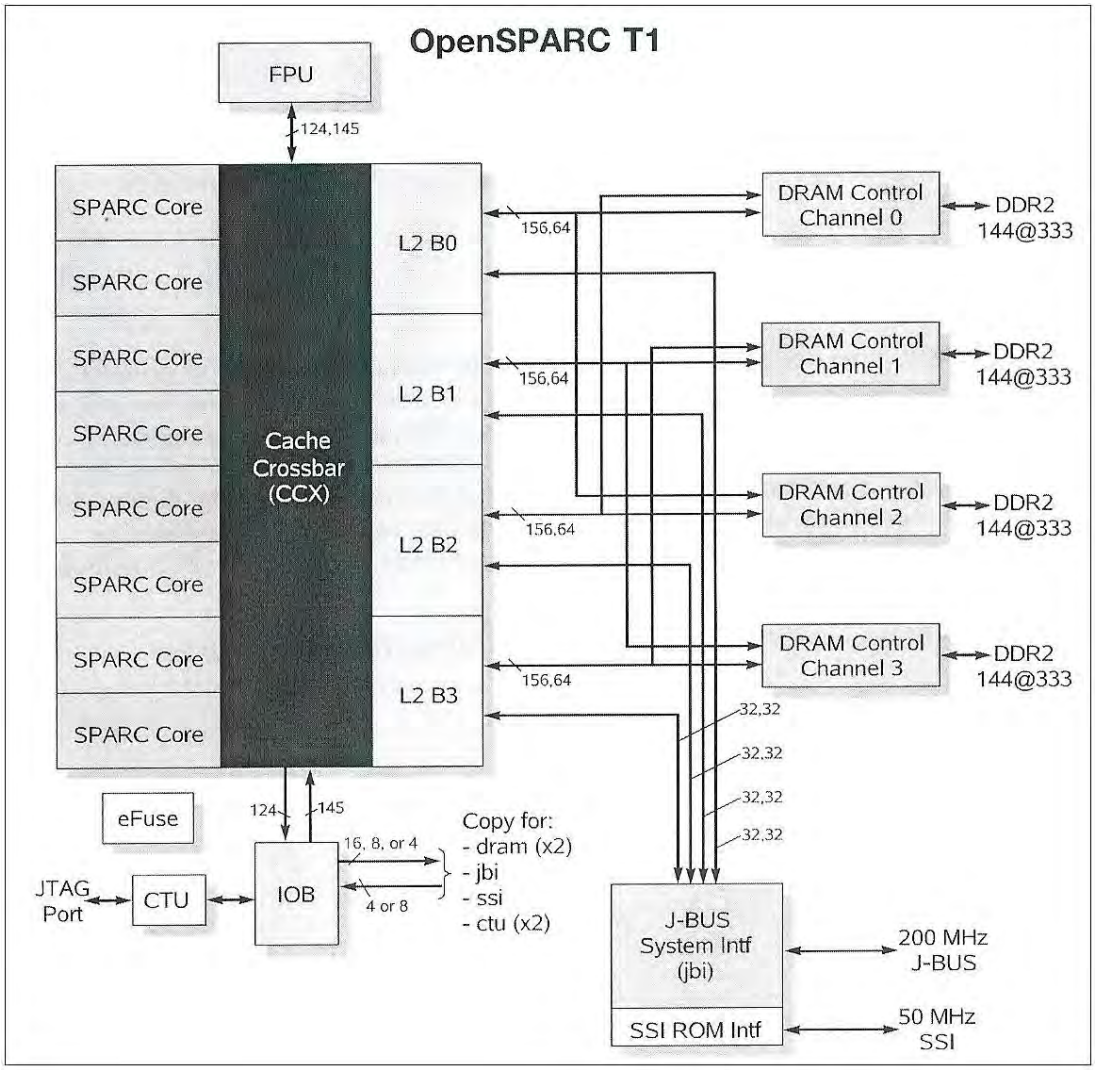


Figura 3.1-1 Diagrama de Bloques del Procesador OpenSPARC T1. Cortesía OpenSPARC.

Núcleo SPARC

Cada núcleo tiene soporte de hardware SPARC para cuatro hilos o subprocesos. Este apoyo consiste en un archivo de registro completo (con ocho ventanas de registros) por hilo, con la mayor parte de la dirección con identificadores de espacio, los registros auxiliares de estado y los registros privilegiados replicados por cada hilo. Los cuatro hilos comparten la instrucción, los caché de datos y los TLB. Cada caché de instrucciones es de 16Kbytes con un tamaño de línea de 32 bytes y sus datos se escriben a través de 8Kbytes.

Cada núcleo SPARC tiene una cuestión concreta, con seis etapas de tubería o pipeline. Estas seis etapas son:

1. Fetch.
2. Thread selection.

3. Decode.
4. Execute.
5. Memory.
6. Write Back.

La siguiente figura muestra la tubería del núcleo SPARC utilizada en el procesador OpenSPARC T1.

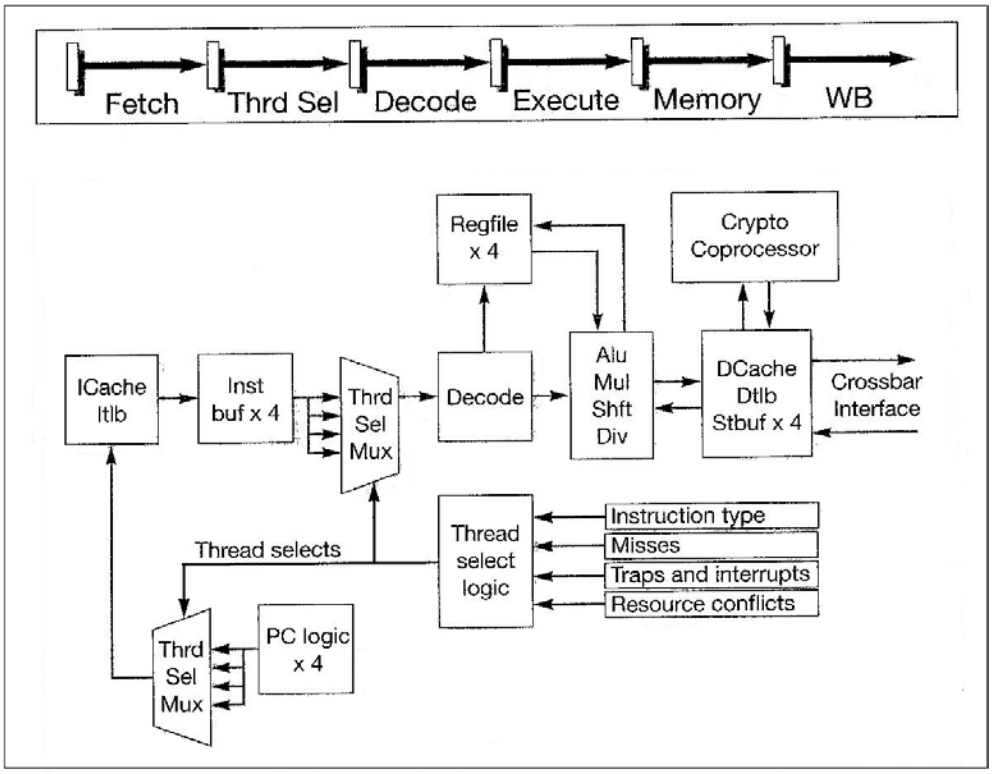


Figura 3.1-2 Tubería del Núcleo SPARC (Pipeline). Cortesía OpenSPARC T1.

Cada núcleo SPARC tiene las siguientes unidades:

1. Unidad de Instrucción de captura (IFU-Instruction Fetch Unit). Que incluye las etapas de tuberías de buscar, seleccionar, decodificar.
2. Unidad de ejecución (EXU-Execute Unit). Incluye la ejecución de la etapa de la tubería.
3. Unidad de carga y almacenamiento (LSU-Load/Store Unit). Incluye memoria de reescritura de etapas y un caché de datos complejos.
4. Unidad de trampa lógica (TLU-Trap Logic Unit). Incluye una trampa lógica y contadores de programa.
5. Unidad de procesamiento de flujo (SPU-Stream Processing Unit). Se utiliza para las funciones de la aritmética modular.
6. Unidad de manejo o gestión de memoria (MMU-Memory Management Unit).

7. Unidad de interfaz de punto flotante (FFU-Floating point Frontend Unit).

Barra Transversal Caché.

Los ocho núcleos de SPARC, los cuatro bancos de memoria caché L2, el puente de entrada y salida, y la FPU, todos ellos conectados a la barra transversal (CrossBar). La siguiente figura 3.1-3 muestra el diagrama de bloques de la barra transversal (CCX).

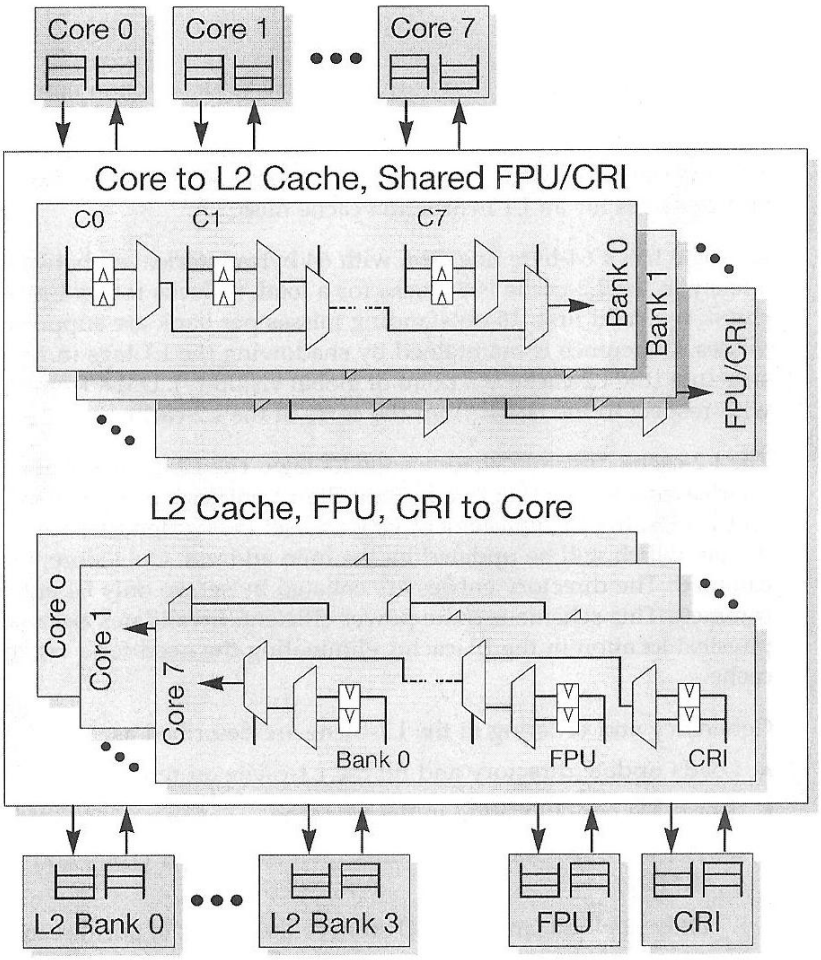


Figura 3.1-3 Diagrama de Bloques de la Barra Transversal CCX. Cortesía OpenSPARC T1.

Interfaz J-Bus.

La interfaz J-Bus (JBI) es la interconexión entre el procesador OpenSPARC T1 y el subsistema de entrada y salida. Tiene una velocidad de 200 MHz, 128 bits de ancho, direcciones multiplexadas, un bus de datos, y es utilizado sobre todo para el tráfico del acceso directo a memoria (DMA), además de una entrada y salida programables (PIO) utilizadas para su control.

La interfaz J-Bus es el bloque funcional de conexión de la recepción, así como responder a las solicitudes de DMA, realiza expediciones a los bancos L2, así como la expedición de las transacciones PIO en nombre de los hilos del procesador y el reenvío de respuestas.

Unidad de Carga y Almacenamiento (Load Store Unit-LSU).

Esta unidad procesa la memoria y hace referencia a los códigos de operación, tales como los diversos tipos de cargas, almacenamiento y descargas. Las interfaces de la LSU junto con su núcleo SPARC y sus unidades funcionales actúan como la puerta de enlace entre el núcleo y el CCX. A través de el CCX los datos de rutas de transferencia pueden ser establecidas con el subsistema de memoria y las entradas y salidas del subsistema (la transferencia de datos es realizada con paquetes).

La arquitectura de hilos de la LSU puede procesar cuatro cargas, cuatro almacenamientos, una búsqueda, operación de flujo, interrupción y un paquete de reenvío.

LSU Pipeline.

Existen cuatro etapas en la tubería LSU. La siguiente figura muestra sus diferentes etapas:

E	M	W	W2
Cache TLB setup	Cache/Tag TLB read	stb lookup traps bypass	pcx rcq gcn. and writeback

Figura 3.1-4 Tubería de la LSU. Cortesía OpenSPARC T1.

Flujo de Datos.

La LSU incluye un caché de 8 Kbytes la cual es una parte de caché de nivel 1 que comparten cuatro subprocesos. Se tiene un almacenamiento de buffer por hilo (STB).

Las cargas perdidas se mantienen en la LSM (Load miss queue) las cuales se comparten con otros subcódigos tales como los atómicos y los de búsqueda.

Los paquetes entrantes del CCX son ordenados y puestos en cola para la distribución de otras unidades a través del DFQ (Data Fill Queue). La DTLB es totalmente asociativa y responde hacia la traducción de direcciones. Así mismo las operaciones ASI son serializadas por la LSU y también secuenciadas por la cola

ASI hacia las unidades destino del chip. En la siguiente figura se puede visualizar el concepto de flujo de datos de la LSU (Figura 3.1-5).

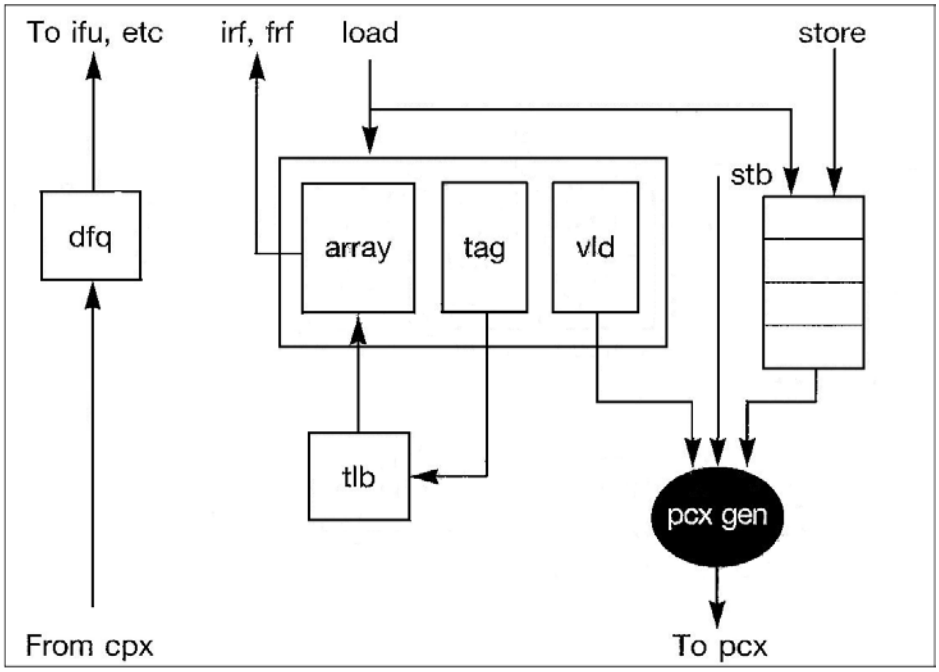


Figura 3.1-5 Concepto de Flujo de Datos de la LSU. Cortesía OpenSPARC T1.

Level 1 Data Cache (D-Cache).

Los 8Kbytes del nivel 1 D-cache tiene cuatro formas de conjunto asociativo y el tamaño de la línea es de 16 bytes. El D-Cache tiene un solo puerto de lectura y escritura para el arreglo de datos y etiquetas. El arreglo V-bit (Valid bit) está constituido por un doble puerto de lectura y un puerto para la escritura (1R/1W). El arreglo de bit válido se encarga de mantener el estado de la línea de cache válida o inválida.

Una carga cacheable carga o pierde la asignación de línea y ejecutará la escritura por almacenamiento. El almacenamiento no hace asignaciones y almacenamientos locales pueden actualizar la memoria caché L1 según lo determinado por la cache L2. Si se considera que no está presente en la cache L1, el almacenamiento local hará que las líneas sean detectadas como invalidas, y puede llegar a soportar simultáneamente hasta cuatro líneas de datos invalidas.

Data Translation Lookaside Buffer (DTLB).

El DTLB se encuentra en la TLB y tiene cuatro hilos (threads) los cuales son compartidos en la DTLB. Tiene un puerto CAM y otro puerto de lectura y escritura (1RW). La DTLB soporta las siguientes operaciones de 32bits de traducción de direcciones:

- VA->PA [virtual address to physical address].

- VA=PA [dirección de derivación para las operaciones de modo de supervisión].
- RA->PA [Real address to physical address].

La etiqueta y datos de TTE (Translation Table Entries) están protegidos y si hay errores serán imposibles de corregir. TTE accesa a los errores de paridad para cargar instrucciones con lo cual causará precisar la etiqueta.

Store Buffer.

La estructura física del Store buffer (STB) consiste en una regulación de almacenamiento CAM y un almacenamiento de un arreglo de datos (STBDATA). Cada hilo está asignado con ocho entradas fijas en una estructura de datos compartida. La SCM tiene un puerto CAM y un puerto RW, y la STBDATA tiene un puerto de lectura (1R) y otro de escritura (1W).

Todos los almacenes residen en el buffer de almacenamiento hasta que sean llevados a un almacenamiento total (TSO). El ciclo de vida de un almacenamiento TSO cumple con las siguientes etapas:

1. Validación.
2. Perpetración (expedido a cache L2).
3. Reconocimiento (cache L2 envía la respuesta).
4. Invalidación o Actualización.

El buffer de almacenamiento implementa parcial y totalmente la revisión de la lectura y después la escritura (RAW). Todos los datos RAW son devueltos a la lista de archivos por la tubería. Los éxitos parciales RAW forzarán la carga para acceder a la cache L2 mientras son enlazados con el almacenamiento y será entregado a CCX.

Data Fill Queue (DFQ).

Un núcleo SPARC se comunica con la memoria de entrada y salida mediante paquetes. Los paquetes destinados a un núcleo SPARC se ponen en cola en el DFQ. El DFQ mantiene una obligación predefinida de ordenar los requerimientos de todos los paquetes entrantes, y su objetivo principal es entregar los paquetes que incluyan la instrucción de búsqueda de unidad (IFU), LSU, la unidad de procesamiento de flujo (SPU).

3.2 UNIDAD DE EJECUCIÓN (EXU).

La unidad de ejecución contiene las siguientes subunidades:

- Unidad Aritmética y Lógica (ALU).
- Shifter (SHFT) o palanca de cambios.
- Integer Multuplier (IMUL) o multiplicador de Entero.

- Integer Divider (IDIV) o divisor de Entero.

A continuación se presenta un diagrama de nivel superior de la unidad de ejecución:

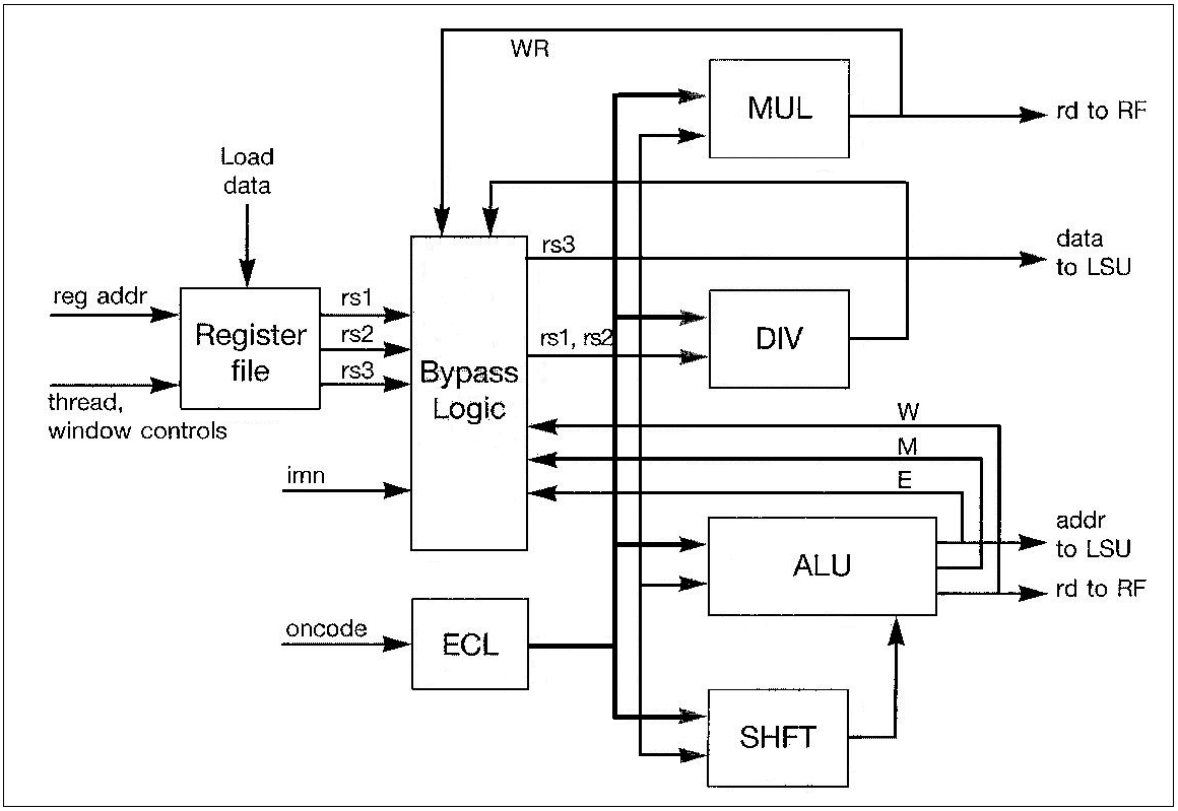


Figura 3.2-1 Diagrama de la Unidad de Ejecución. Cortesía OpenSPARC T1.

La unidad aritmética y lógica (ALU) está formada por una cadena de operaciones lógicas tales como: ADD, SUB, AND, NAND, OR, NOR, XOR, XNOR y NOT. La ALU es también usada en cálculos de direcciones virtuales. La Figura 2-17 ilustra el diagrama de bloques de la ALU.

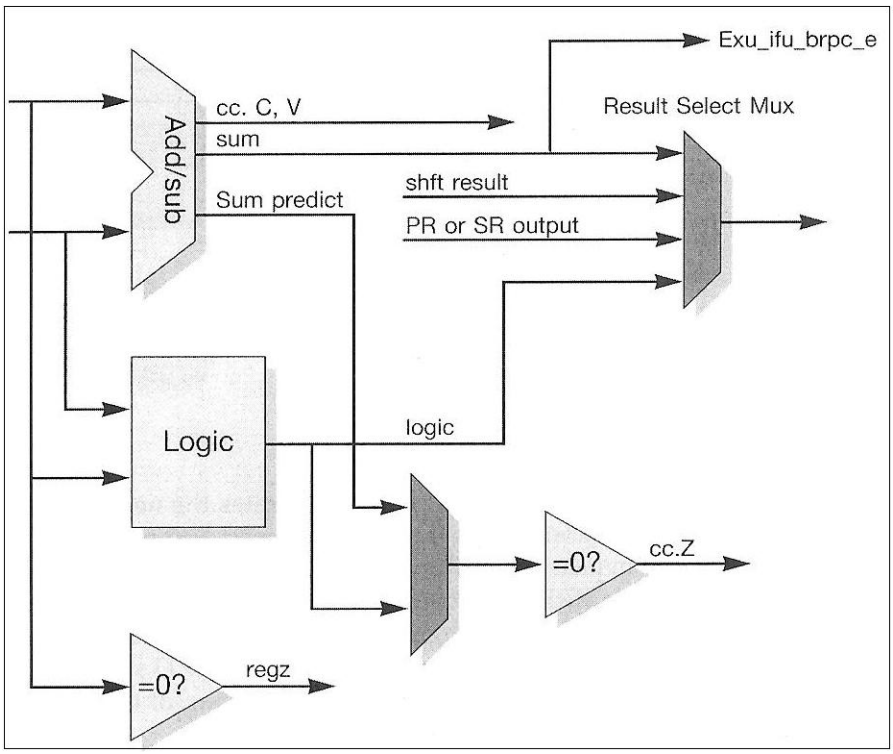


Figura 3.2-2 Diagrama de Bloques de la ALU. Cortesía OpenSPARC T1.

La subunidad Shifter (SHFT) aplica de 0 a 63 bits por cambios o turnos. En la figura siguiente se muestra el diagrama de bloques del SHFT. Figura 3.2-3

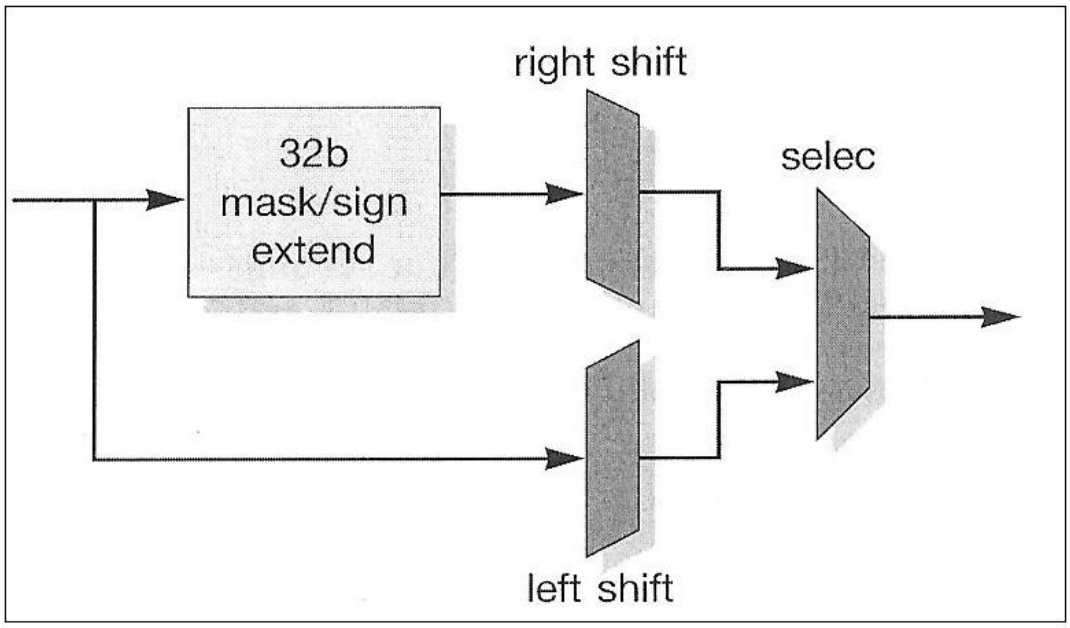


Figura 3.2-3 Diagrama de Bloques de "Shifter". Cortesía OpenSPARC T1.

3.3 UNIDAD DE INTERFAZ PUNTO FLOTANTE (FLOATING POINT FRONTENED UNIT-FFU).

La unidad de interfaz de punto flotante (FFU) es la responsable de la ordenación de las operaciones de punto flotante (OPS FP) hacia la unidad de punto flotante (FPU) a través de la LSU, así como la ejecución de operaciones sencillas e instrucciones. La FFU también mantiene el registro del estado del punto flotante (FSR) y los registros de estado gráficos (GSR). Sólo puede haber una instrucción pendiente en la FFU a la vez.

La FFU está compuesta por cuatro bloques:

- Los archivos de registro de punto flotante (FFU_FRF).
- El bloque de control (FFU_CTL).
- El bloque del Data-Path (FFU_DP).
- El bloque de ejecución VIS (FFU_VIS).

La figura 3.3-1 muestra el diagrama de bloques de la FFU ilustrando sus cuatro subbloques:

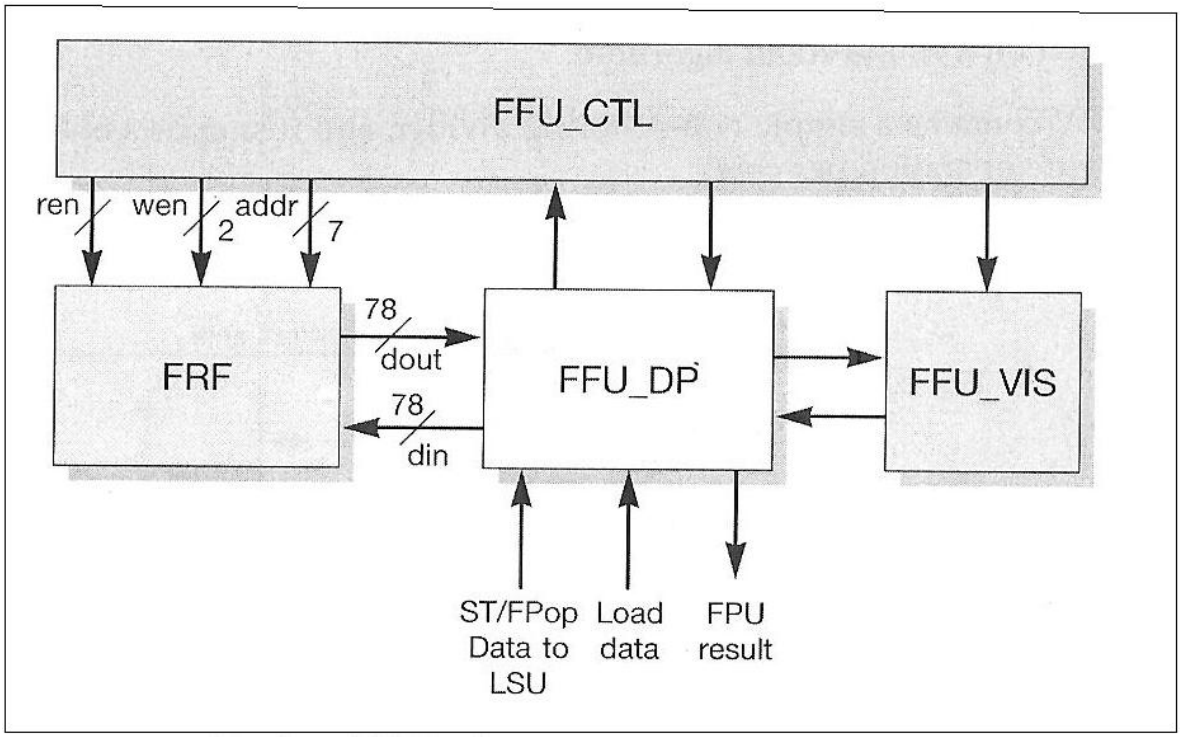


Figura 3.3-1 Diagrama de Bloques de la FFU. Cortesía OpenSPARC T1.

Archivos de registro de Punto Flotante.

El archivo de registro de punto flotante (FRF) dispone de 128 entradas de 64Bits de datos, además de 14Bits de ECC. El puerto de escritura está habilitado para la

mitad de los datos y se manejan Bits para la longitud de palabra, puede ser inferior o superior.

Bloque de Control (FFU_CTL).

El bloque de control FFU implementa el control lógico de la FFU y genera que el multiplexor seleccione señales adecuadas del Data-Path. El control de la FFU también decodifica el fp_opcode y contiene el estado de máquina para la tubería FFU. También genera las trampas de FP y elimina señales, así como la señalización de la LSU cuando los datos están listos para ser procesados.

Bloque del Data-Path (FFU_DP).

Este bloque se compone de multiplexores y datos que han sido leídos o que están a punto de escribirse en el FRF. El FFU_DP opera los datos para el STF, hace implementaciones como FMOV, FABS, etc. Realiza revisiones al ECC para la lectura de datos desde la FRF y genera que en la ECC los datos estén escritos en la FRF.

Bloque de Ejecución VIS (FFU_VIS).

En este bloque se realizan implementaciones a un subconjunto de instrucciones gráficas VIS, incluyendo la adición y sustracción así como operaciones lógicas. Todas las operaciones se ejecutan en un solo ciclo, y las entradas y salidas de datos están conectadas a la FFU_DP.

3.4 UNIDAD MULTIPLICADORA (MUL).

La unidad multiplicadora SPARC (MUL) realiza la multiplicación de dos entradas de 64-Bits. La MUL es compartida entre la EXU y la SPU y tiene un bloque de control y ruta de acceso a datos por categorías. A continuación se ilustra cómo está el multiplicador conectado a otros bloques funcionales:

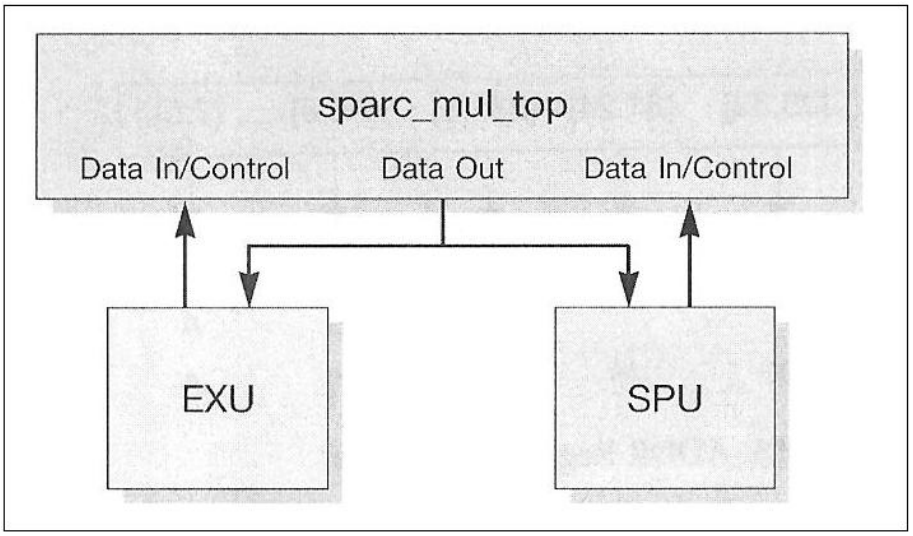


Figura 3.4-1 Diagrama de Bloques de Multiplexor (MUL). Cortesía OpenSPARC T1.

3.5 UNIDAD DE PROCESAMIENTO DE FLUJO (SPU).

Cada núcleo SPARC está equipado con una unidad de procesamiento de flujo (SPU) la cual realiza el soporte de operaciones asimétricas de hasta un tamaño de clave de 2048-Bits.

La SPU comparte el entero multiplicador con la unidad de ejecución (EXU) para las operaciones aritméticas (MA). Mientras la SPU se comparte entre todos los hilos del núcleo SPARC, sólo un hilo puede utilizar la SPU a la vez. La operación SPU es configurada por un almacenamiento de un hilo al control de registro y luego regresa al procesamiento normal, así, la SPU iniciará la carga de flujo o el almacenamiento de operaciones al nivel caché L2 y calcular operaciones para el entero multiplicador. Una vez que la operación se puso en marcha puede operar paralelamente a la ejecución de instrucciones del núcleo SPARC.

Registros ASI de la SPU.

Todos los identificadores alternativos de espacio (ASI) se registran en el SPU y tienen una longitud de 8 bytes. El acceso a todos los registros ASI de la SPU cuentan con supervisión, y sólo se puede acceder en modo supervisión. En la siguiente lista se destacan los registros de la ASI:

1. El registro de la dirección física de aritmética modular (MPA).

Este registro lleva la dirección física utilizada para acceder a la memoria principal.

2. El registro de la dirección de memoria (MA_ADDR).

Este registro lleva la dirección de memoria para la compensación de varios operandos y el tamaño del exponente.

En la siguiente figura se muestra la disposición de los campos de bits.

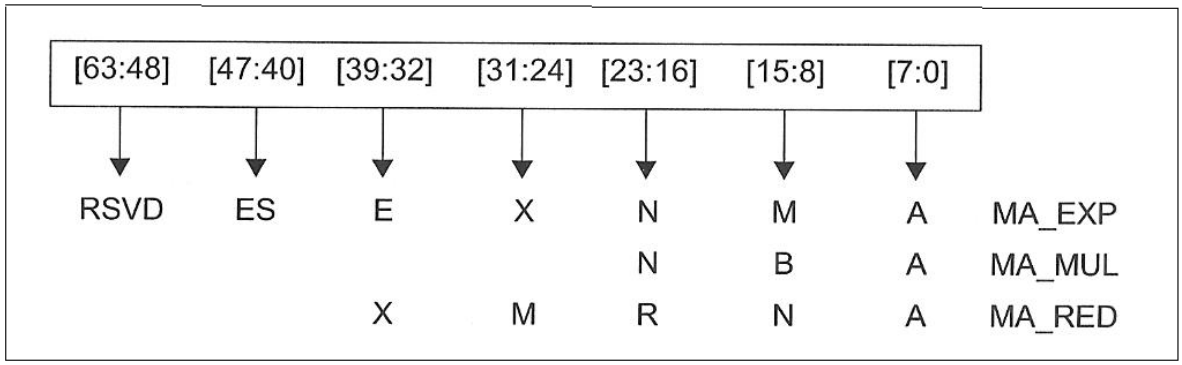


Figura 3.5-1 Registro de Campos de Bits MA_ADDR. Cortesía OpenSPARC T1.

Flujo de Datos de las Operaciones Aritméticas Modulares.

Se ilustran a continuación el flujo de datos de las operaciones aritméticas:

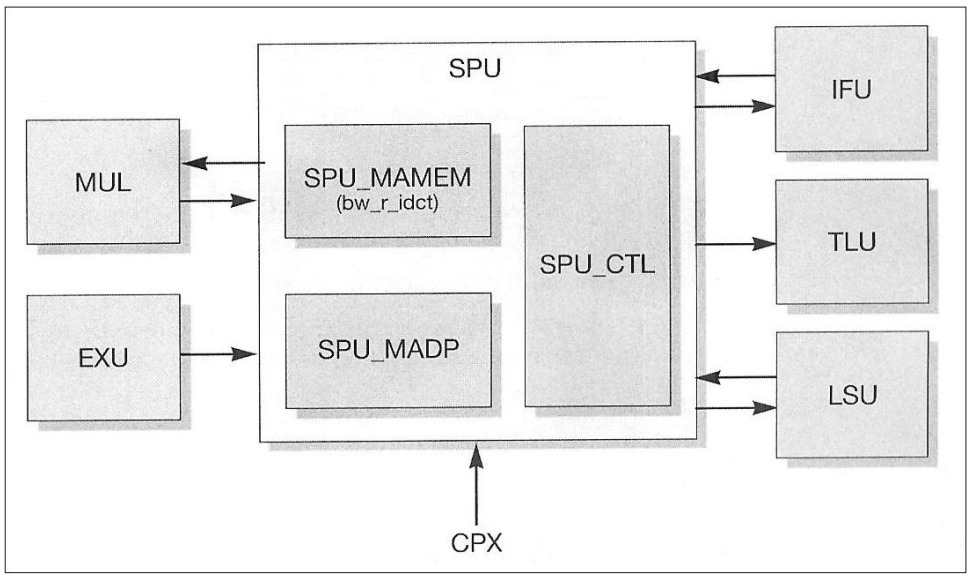


Figura 3.5-2 Diagrama de Bloques del Flujo de Datos de las Operaciones Modulares. Cortesía OpenSPARC T1.

Memoria Modular Aritmética (MA Memory).

Con un total de 1280 bytes de memoria local y un solo puerto de lectura y escritura (1RW) que es utilizado para el suministro de operandos de la Aritmética modular. La memoria MA alberga 5 operandos de 32 palabras cada uno, que admitirá un tamaño máximo de 2048 bits y está protegida con 2 bits por cada palabra de 64 bits.

La memoria MA requiere una inicialización de software antes del comienzo de las operaciones realizadas en ella. Tres operaciones MA_LD están obligadas a iniciar

las 160 palabras de la memoria ya que el campo de longitud MA_CTL permite un máximo de 64 palabras para ser cargadas en la memoria MA.

Los accesos escritos en la memoria MA pueden estar en los límites de 16 bytes o de 8 bytes. Para los accesos de lectura en la memoria MA, se debe estar en los 8 bytes.

Operaciones Modulares Aritméticas (MA ops).

Todos los registros de aritmética modular se deben iniciar antes de lanzar un sistema de operación modular aritmética. Las operaciones de la aritmética modular (MA ops) comienzan con una *"stxa"* hacia el registro de la MA_CTL si el buffer almacenado para ese hilo se encuentra vacío. De lo contrario los hilos esperarán hasta que el buffer esté vacío antes de ser enviado el stx_ack a la LSU. Una operación MA que esté en proceso, puede ser abortada por otro hilo por medio de *"stx"* hacia el registro MA_CTL.

La figura 3.5-3 muestra las operaciones MA usando un diagrama de estado de transición.

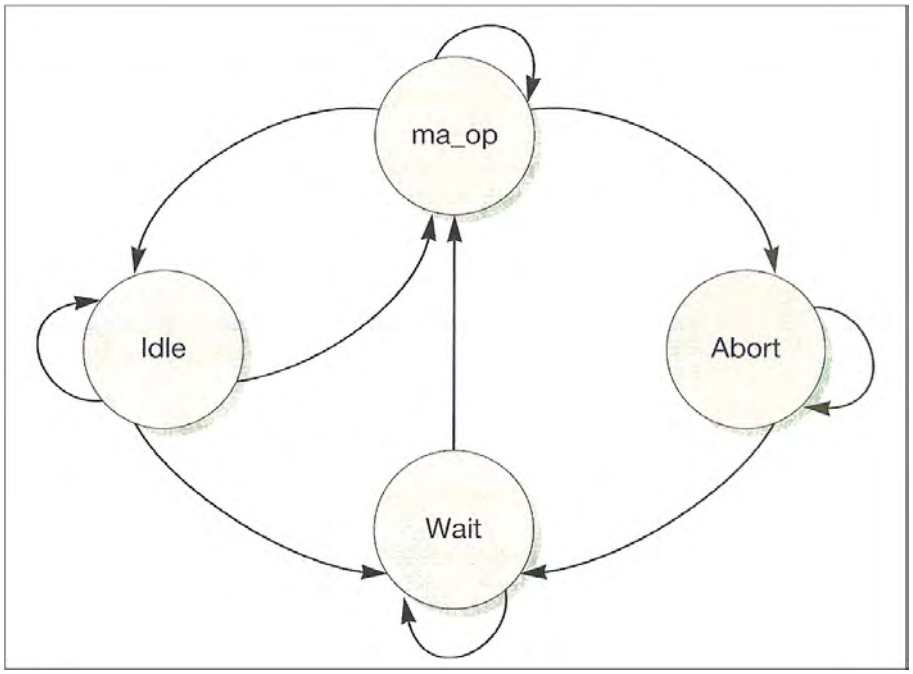


Figura 3.5-3 Diagrama de Estado de Transición de Operaciones MA. Cortesía OpenSPARC T1.

El estado de transición está mostrado por las siguientes ecuaciones:

```
tr2_maop_frm_idle = cur_idle & stxa_2ctlreq & ~wait_4stb_empty & ~wait_4trapack_set;
tr2_abort_frm_maop = cur_maop & stxa_2ctlreg;
```

```
tr2_wait_frm_abort = cur_abort & ma_op_complete;
tr2_maop_frm_wait = cur_wait & ~(stxa_2ctlreg | wait_4stb_empty | wait_4trapack_set);
tr2_idl_frm_maop = cur_maop & ~stxa_2cltreg & ma_op_complete;
tr2_wait_frm_idle = cur_idle & stxa_2ctlreg & (wait_4stb_empty | wait_4trapack_set);
```

La operación MA_ST se inicia con un *"stxa"* hacia el código de registro MA_CTL que es igual a MA_ST y su longitud de campo especifica el número de palabras a enviar al nivel cache L2. La SPU envía al procesador una memoria interfaz cache (PCX) solicitar la LSU, así mismo espera una confirmación de la LSU antes de enviar otra petición. Cuando es necesario guarda los reconocimientos que son devueltos desde la cache L2 a la interfaz del procesador (PCX), y se irá a la LSU con el fin de invalidar el nivel cache L1. Posteriormente el LSU hará llegar al SPU un acuse de recibo, y la SPU realizará un decremento a un contador local y espera a que todos los almacenamientos sean enviados para ser reconocidos y la transición al estado realizado.

3.6 UNIDAD DE MANEJO DE LA MEMORIA (MMU).

La MMU mantiene el contenido de la instrucción del buffer de transacción y del buffer de datos. El buffer de transacción (ITLB) reside en la unidad de búsqueda de instrucción (IFU) y el buffer de transacción de datos (DTLB) reside en la unidad de carga y almacenamiento (LSU). A continuación se ilustra la relación entre la MMU y el TLB.

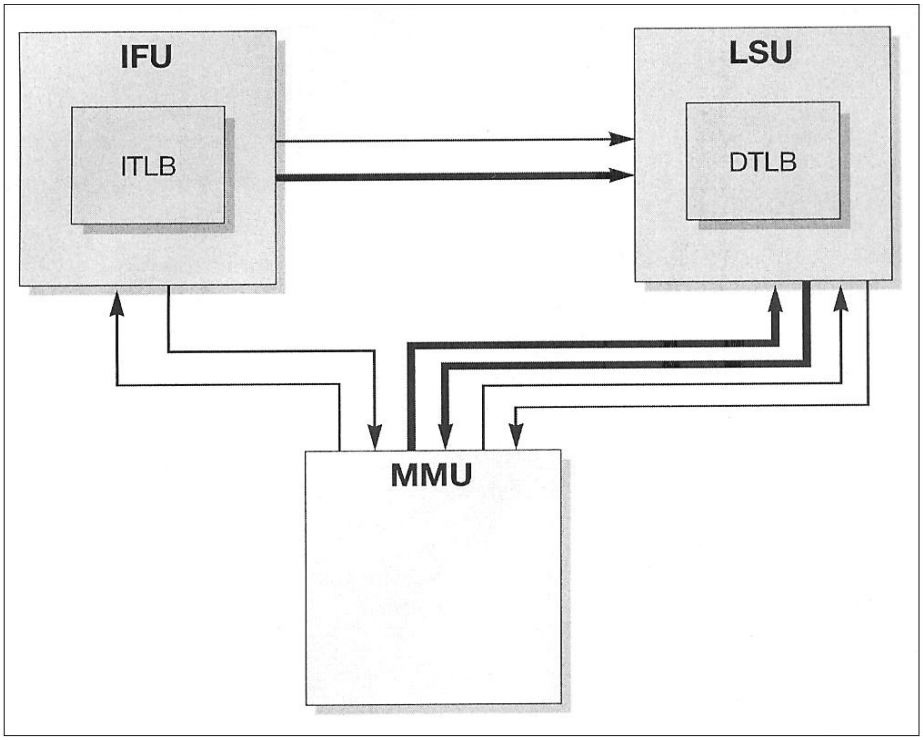


Figura 3.6-1 Relación entre MMU y los TLB´s. Cortesía OpenSPARC T1.

Estructura del Buffer de Traducción.

El buffer de traducción (TLB) consiste en un contenido de memoria direccionable (CAM) y una memoria direccionable aleatoria (RAM). CAM tiene un puerto de comparación y un puerto de lectura y escritura (1C1RW), y la RAM tiene sólo un puerto de lectura y escritura (1RW). El TLB soporta los siguientes eventos:

1. CAM.
2. Lectura.
3. Escritura.
4. Bypass o derivación.
5. Demap.
6. Reset suave.
7. Reset duro.

CAM está compuesto por el siguiente campo de bits, un identificador de partición ID (PID), un identificador de contexto ID (CTXT) y una dirección virtual (VA). La dirección virtual se subdivide en el campo del tamaño de página. El campo CTXT también tiene su propia habilitación para su flexibilidad al estar implementada. La porción de los campos CAM son para efectos de comparación. En cuanto a la RAM, es constituida por los siguientes bits; la dirección física (AF) y atributos. La porción de los campos RAM están únicamente para propósitos de lectura, donde la lectura puede ser causada por un software de lectura o una lectura basada en CAM.

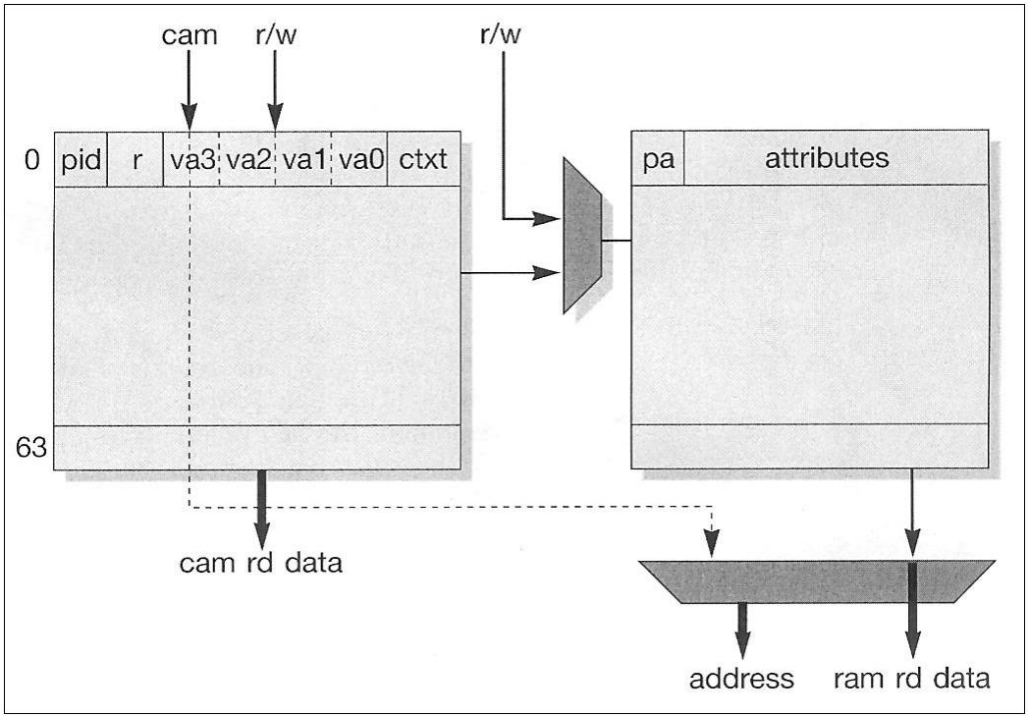


Figura 3.6-2 Estructura del TLB. Cortesía OpenSPARC T1.

Especificaciones de Acceso de Escritura al TLB.

Un stxa hacia la entrada de datos o acceso de datos hace que una operación de escritura sea asíncrona al flujo de la tubería (pipe). Las peticiones de escritura son originadas por la entrada cuatro del FIFO en la LSU. La LSU pasará la petición de escritura hacia la MMU la cual la reenviará a la ITLB (Instruction Translation Lookaside Buffer) o a la DTLB.

Especificaciones de Acceso de Lectura al TLB.

El TLB lee las operaciones siguiendo un protocolo tal y como lo hace en la escritura de operaciones. El acceso de datos ASI hará la lectura en la parte RAM, entonces el TLB leerá datos y serán regresados a través de un direccionamiento en la LSU. Si no existe error de paridad, la LSU reenviará los datos.

3.7 TRAP LOGIC UNIT (TLU).

Contiene seis niveles de trampa, y una trampa puede estar en uno de los siguientes cuatro modos:

- Modo Reset-error-debug (RED).
- Modo hipervisor (HV).
- Modo supervisor (SV).
- Modo usuario.

Las trampas harán en la tubería del núcleo SPARC una limpieza y se producirá un Switch de hilo para que sea producido hasta que el vector de trampa haya sido resuelto.

Interrupciones de software se entregan a cada uno de los núcleos virtuales utilizando el nivel de interrupción a través del registro SOFTINT_REG. Interrupciones de entrada y salida son llevadas a cada núcleo virtual usando el vector de interrupción (interrupt_vector). Se pueden tener en cola hasta 64 interrupciones pendientes por hilo o subproceso, uno por cada vector de interrupción. Los vectores de interrupción tienen prioridad, así cada fuente de interrupción de entrada/salida tiene un número de cableado de interrupción que se utiliza como vector de interrupción por el puente de bloqueo de entrada/salida.

El TLU se encuentra en una posición lógicamente central para recoger todas las trampas y las interrupciones para después ser reenviadas. La siguiente figura ilustra el papel de la TLU respecto a los demás bloques en un núcleo SPARC.

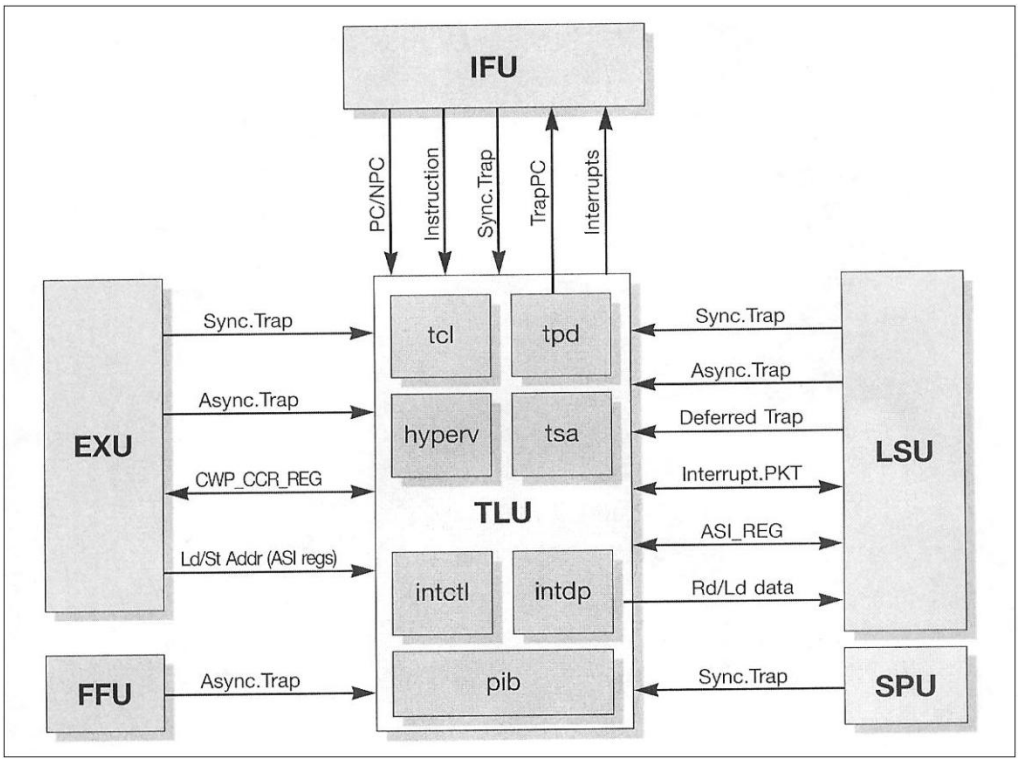


Figura 3.7-1 Diagrama de la Función TLU respecto a los demás Bloques en un Núcleo SPARC. Cortesía OpenSPARC T1.

La siguiente lista destaca las funciones de la TLU:

- Recoge las trampas de todas las unidades del núcleo SPARC.
- Detecta algunos tipos de trampas internas de la TLU.
- Resuelve la trampa con prioridad y genera el vector trampa.
- Envía el lavado de tubería hacia otras unidades SPARC mediante un conjunto de trampas LSU.
- Mantiene registros de estado del procesador.
- Administra la trampa de pila.
- Restaura el estado del procesador de la trampa de la pila y retira instrucciones.
- Implementa un hilo (thread) interno para la entrega.
- Recibe y procesa todos los tipos de interrupciones.
- Mantiene y compara las marcas y los registros relacionados SOFTINT.
- Genera un temporizador de interrupciones y las interrupciones de software (interrupt_level-n type).
- Mantiene los contadores de rendimiento de instrumentos (PIC).

Arquitectura de los Registros de la TLU.

A continuación se mencionan los registros que mantienen la TLU:

1. Estado del procesador y registros de control.
- Registro PSTATE (processor state).
- Registro TL (Trap level).
- Registro GL (global register window level).
- Registro PIL (Processor interrupt level).
- Registro TBA (Trap base address).
- Registro HPSTATE (Hypervisor processor state).
- Registro HTBA (Hypervisor trap base address).
- Registro HINTP (Hypervisor interrupt pending).
- Registro HSTICK_CMPR_REG (Hypervisor system tick compare).

2. Trampa de pila.
- Registro TPC (trap PC).
- Registro TNPC (Trap next PC).
- Registro TTYPE (Trap type).
- Registro TSTATE (Trap state).
- Registro HTSTATE (Hypervisor trap state).

3. Registros auxiliaries de estado.
- Registro TICK_REG (tick).
- Registro STICK_REG (system tick).
- Registro TICK_CMPR_REG (Tick compare).
- Registro STICK_CMPR_REG (system tick compare).
- Registro SOFTINT_CMPR_REG (software interrupt).
- Registro SET_SOFTINT (set software interrupt register).
- Registro CLEAR_SOFTINT (clear software interrupt register).
- Registro PERF_CONTROL_REG (performance control register).
- Registro PERF_COUNTER (performance counter.

4. Mapa de registros ASI.
- Registros Scratch-pad (ocho de ellos).
- Registros de CPU y dispositivos.
- Punteros de cabeza y cola.
- Registro de interrupción de CPU.
- Registro de interrupción.
- Registro del vector entrante.
- Registro de expedición de interrupciones (para llamadas cruzadas).

Tipos de Trampa (TRAPS).

Las trampas pueden ser generadas por el código de usuario, código supervisor o código hipervisor. Una trampa es entregada a diferentes niveles del controlador de trampa para ser procesada, el nivel supervisor (SV), también conocido como el

nivel privilegiado, o el nivel hipervisor (HV). La manera en que las trampas se generan puede ayudar a categorizar en síncrona (para la tubería del núcleo SPARC) o asíncrona.

Existen tres categorías definidas de las trampas: precisa, diferida e interrumpida. En los párrafos siguientes se describen brevemente la naturaleza de cada una de ellas.

1. Trampa precisa.

Una trampa precisa es introducida por una instrucción particular y ocurre antes de que cualquier estado de programa visible haya cambiado por la instrucción. Cuando una trampa precisa ocurre, se debe cumplir las siguientes condiciones:

- Haber guardado puntos en TCP a la instrucción que indujo la trampa y guardarlos en la NTPC para la instrucción que será ejecutada a continuación.
- Todas las instrucciones emitidas antes de que induzcan la trampa deben tener completada su ejecución.
- Las instrucciones emitidas después de que la indujo la trampa deben seguir siendo ejecutadas.

2. Trampa Diferida.

Una trampa diferida es inducida por una instrucción particular. Sin embargo la trampa puede ocurrir después de que estado de programa visible haya cambiado la ejecución de cualquiera de las otras trampas o más instrucciones. Si una instrucción induce una trampa en diferido, y la trampa ocurre simultáneamente, la trampa diferida no podrá aplazarse más allá de la trampa precisa.

3. Trampa Interrumpida.

La trampa interrumpida o perturbada se causa por una condición (por ejemplo, una interrupción), en lugar de ser causada por una instrucción. Cuando una trampa interrumpida ha sido servida, la ejecución del programa se reanuda donde se quedó anteriormente. La interrupción de las trampas es controlada por una combinación del nivel de interrupción del procesador (PIL) y la habilitación de la interrupción (IE). La condición de ruptura o de interrupción se ignora cuando las interrupciones están deshabilitadas (PSTATE.IE=0) o la condición es inferior a la especificada en el PIL.

La siguiente tabla ilustra los tipos de trampas del procesador OpenSPARC T1.

Trap Type	Deferred	Disrupting	Precise
Asynchronous	*None*	None	Spill traps, FPU traps, DTLB parity error on loads, SPU-MA Memory error return on load to SYNC reg
Synchronous	DTLB parity error on stores (precise to SW)	Interrupts and some error traps	All other traps

Tabla 3.7 Tipos de trampas (Traps) en el OpenSPARC T1. Cortesía OpenSPARC T1.

Flujo de Trampas.

Una trampa asíncrona se asocia normalmente con las instrucciones de longitud de latencia y de guarda/restaura, por lo que la aparición de una trampa no es sincrónico con la operación de tubería del núcleo SPARC. Todas esas trampas son precisas en el procesador OpenSPARC T1.

La interrupción de las trampas está asociada con ciertas condiciones particulares. El TLU las recoge y las reenvía a la IFU, la cual las hace bajar por la tubería como interrupciones en lugar de enviar las instrucciones desde abajo.

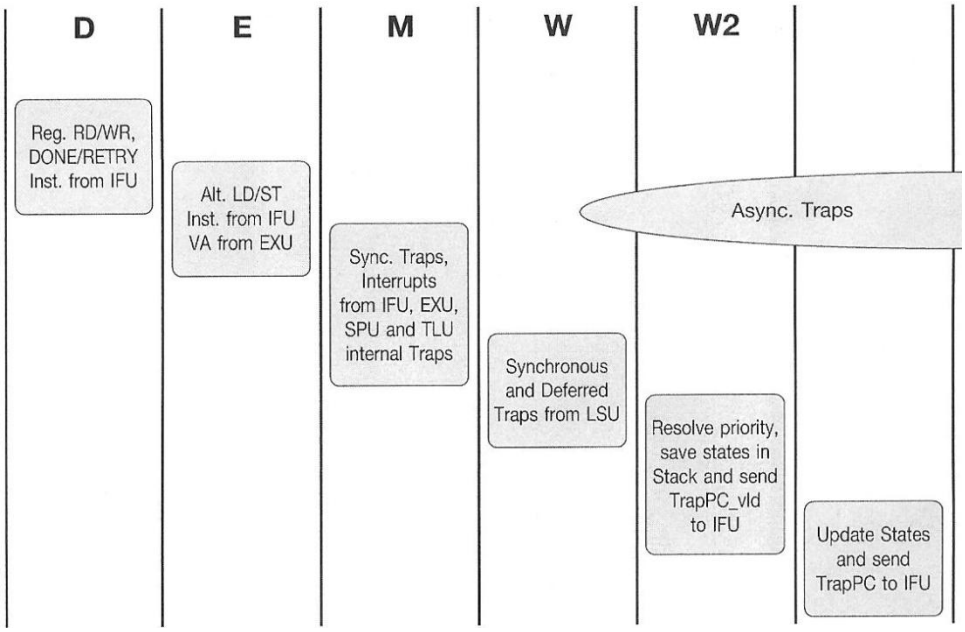


Figura 3.7-2 Secuencia del Flujo de Trampas. Cortesía OpenSPARC T1.

Todas las trampas de la IFU, EXU, SPU, LSU y de la TLU serán ordenadas por tipo para resolver la prioridad de cada una y para determinar el tipo de trampa y

tipo de vector. Después de que estos se resuelven, la dirección base de la trampa (TBA) será seleccionada para viajar por la tubería para su ejecución.

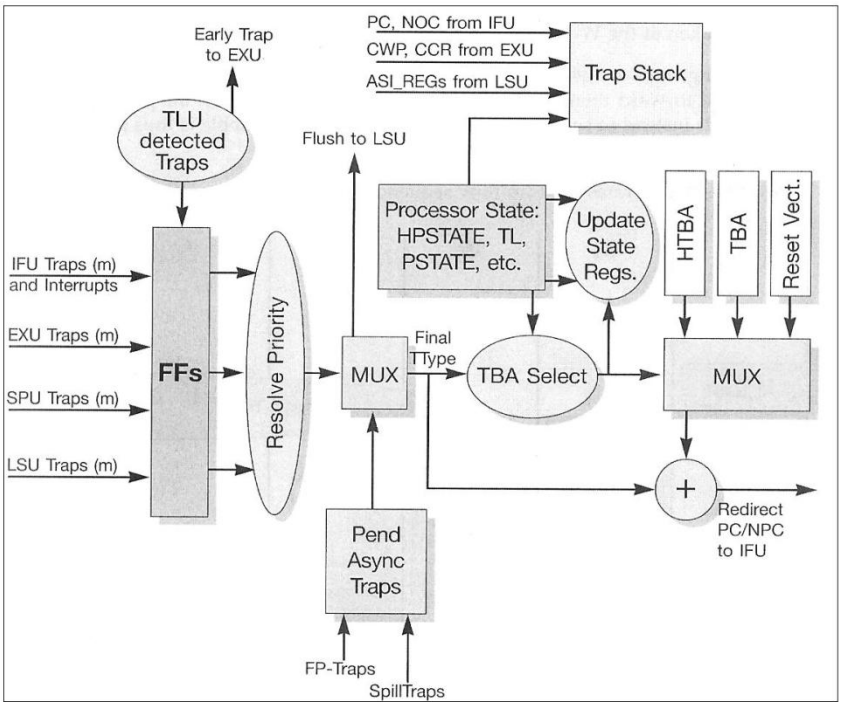


Figura 3.7-3 Diagrama del Flujo de trampas con respecto al bloque de hardware. Cortesía OpenSPARC T1.

Construcción del Programa de Contador de Trampas.

Se describirá el algoritmo para la construcción del programa de contador de Trampas (TPC):

- Supervisor trap (SV trap)
  Redirect PC <= {TBA[47:15], (TL>0), TTYPE[8:0], 5'b00000}
- Hypervisor trap (HV trap)
  Redirect PC <= {TBA[47:14], TTYPE[8:0], 5'b00000}
- Traps in non-split mode
  Redirect PC <= {TBA[47:15], (TL>0), TTYPE[8:0], 5'b00000}
- Reset trap
  Redirect PC <= {RSTVAddr[47:8], (TL>0), RST_TYPE[2:0], 5'b00000}
- RSTVAddr = 0xFFFFFFFFF0000000
- Done instruction
  Redirect PC <= TNPC[TL]
- Retry instruction
  Redirect PC <= TPC[TL]
  Redirect NPC <= TNPC[TL]

Interrupciones.

Las interrupciones de software se entregan a cada núcleo virtual mediante el "interrupt_level_n" y trampas (0x41-0x4f) a través del registro SOFTINT_REG. Llamadas cruzadas (cross-call) de CPU y Entradas/Salidas interrupciones se entregan a cada núcleo virtual usando la trampa "interrupt_vector" (0x60).

La trampa "interrupt_vector" para interrupciones de software tiene un registro de 64 bits ASI_SWVR_INTR_RECEIVE.

Dispositivos de entrada y salida, y llamadas cruzadas de CPU contienen un identificador de 6 bits, el cual determina qué vector de interrupción (nivel) en el registro ASI_SWVR_INTR_RECEIVE la interrupción será destinada.

Cada cadena de registro ASI_SWVR_INTR_RECEIVE puede tener en cola hasta 64 interrupciones pendientes, uno para cada vector de interrupción. Vectores de interrupción tienen prioridad con el vector 63 siendo la máxima prioridad y el vector 0 será la prioridad más baja.

Cada fuente de interrupción de Entrada/Salida tiene gran cableado de interrupciones que se utiliza como índice de una tabla de información del vector de interrupción (INT_MAN) en el puente de la unidad de Entrada/Salida. Generalmente a cada fuente de interrupción de Entrada/Salida se le asignará un único objetivo y su nivel de vector.

Flujo de Interrupción.

La siguiente figura ilustra el flujo de interrupciones y el vector de interrupciones:

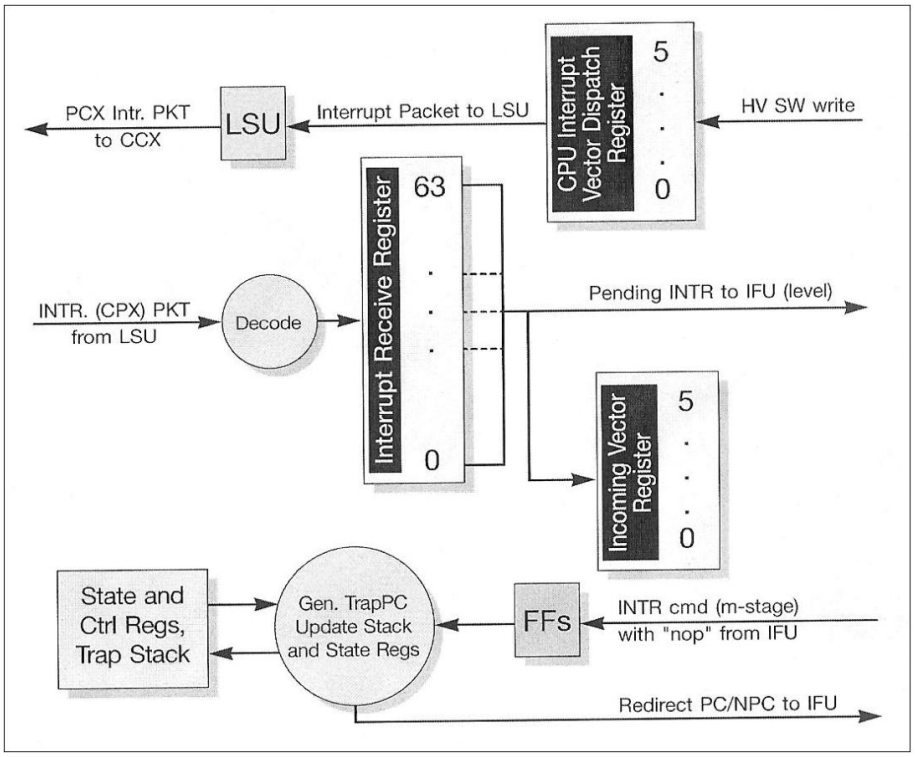


Figura 3.7-4 Flujo de Hardware y Vectores de Interrupción. Cortesía OpenSPARC T1.

Comportamiento de Interrupciones y enmascaramiento de Interrupciones.

En la siguiente lista se describe el comportamiento y enmascaramiento de interrupciones.

1. Interrupciones del Hipervisor no pueden ser enmascaradas por el supervisor ni tampoco por el usuario, sólo pueden enmascararse por el Hipervisor mediante el bit PSTATE.IE. Tales interrupciones incluyen interrupciones de hardware HINTP.
2. Interrupciones normales inter-core o inter-thread como las llamadas cruzadas (cross-calls) pueden ser mandadas por un software de escritura al registro CPU INT_VEC_DIS_REG.
3. Interrupciones especiales como “reset”, “idle”, o “resume” sólo pueden ser enviadas por software a través del puente Entrada/Salida (IOB) por escrito al registro INT_VEC_DIS_REG.
4. Hipervisor siempre suspenderá interrupciones del Supervisor.
5. Algunas interrupciones de supervisor como Mondo-Qs sólo pueden ser enmascaradas por el bit PSTATE.IE.
6. Interrupciones tipo “Interrupt_level_n” sólo podrán ser enmascaradas por el PIL y el bit PSTATE.IE en el nivel supervisor o usuario.

Modos de Transición de Trampas.

La figura 3.7-5 muestra el modo de transición desde los diferentes niveles de trampas:

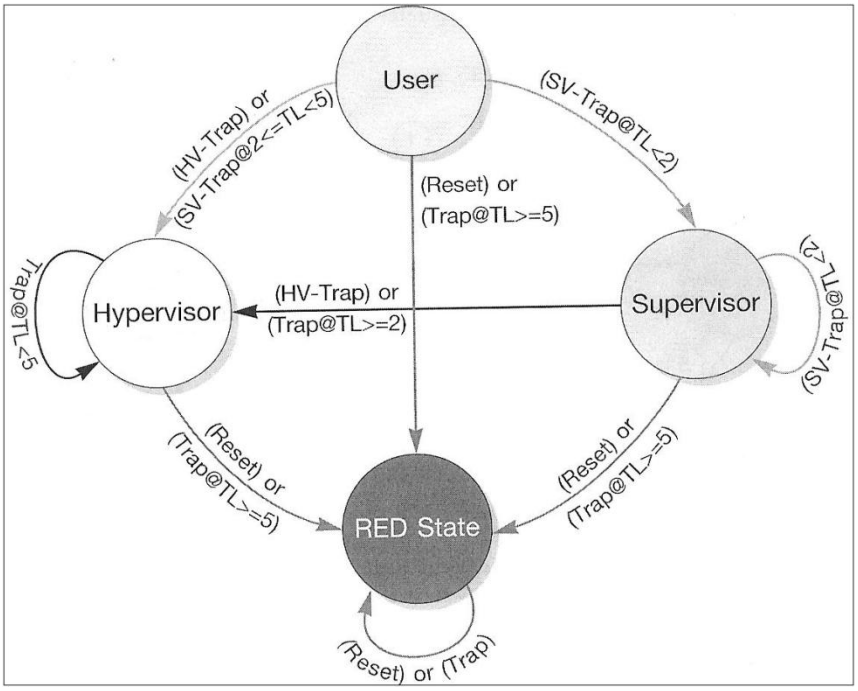


Figura 3.7-5 Modos de Transición de Trampas (Traps). Cortesía OpenSPARC T1.

3.8 CPU CACHE CROSSBAR.

Descripción Funcional.

La barra de cruce (crossbar-CCX) se encarga de manejar la comunicación desde los ocho núcleos, los cuatro bancos cache L2, el puente de Entrada/Salida y la Unidad de Punto Flotante (FPU). Todas estas unidades funcionales se comunican por el envío de paquetes y la CCX se encarga de administrar esa entrega de paquetes.

Cada núcleo SPARC de la CPU puede enviar un paquete a cualquiera de: los bancos cache L2, el puente de Entrada/Salida o a la FPU. Pero también pueden ser enviados en la dirección inversa, es decir, cualquiera de estas unidades pueden enviar paquetes a los ocho núcleos de la CPU. La siguiente figura muestra que cada uno de los ocho núcleos SPARC puede comunicarse con cada uno de los bancos cache L2, el puente Entrada/Salida y la FPU.

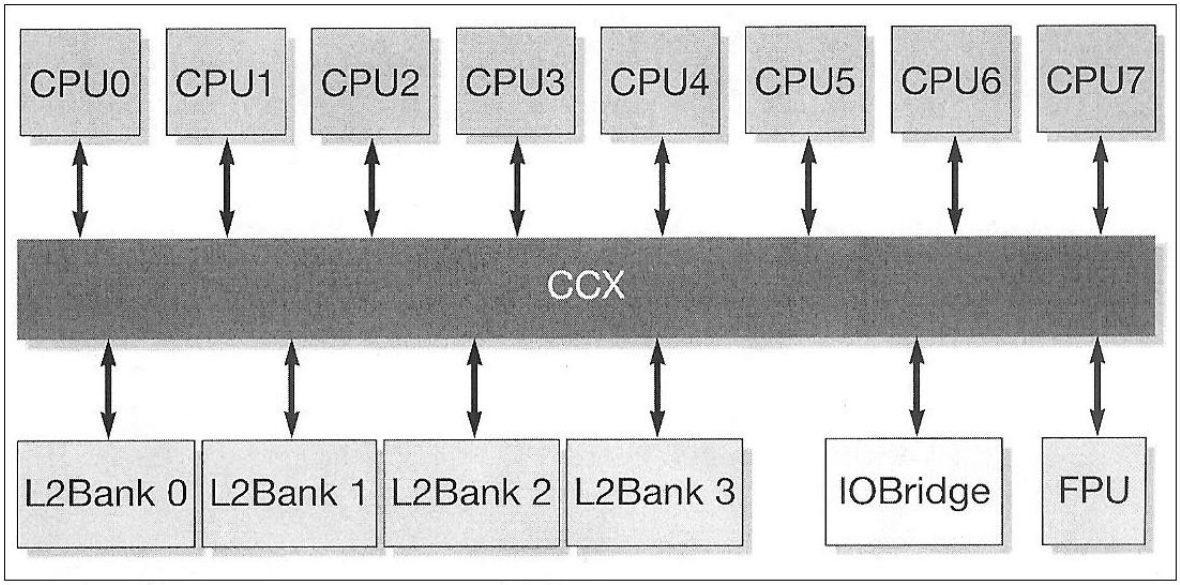


Figura 3.8-1 Interfaz del CCX. Cortesía OpenSPARC T1.

Entrega de Paquetes CCX.

La CCX consiste en dos bloques principales “processor cache crossbar” (PCX) y la “cache processor crossbar” (CPX). El bloque PCX se encarga de la comunicación de cualquiera de las ocho fuentes de CPU a cualquiera de: cuatro bancos cache L2, puente de Entrada/Salida o FPU. El CPX maneja la comunicación de los bancos cache L2, puente Entrada/Salida o FPU a cualquiera de los ocho destinos de la CPU. A continuación se ilustra las conexiones PCX y CPX.

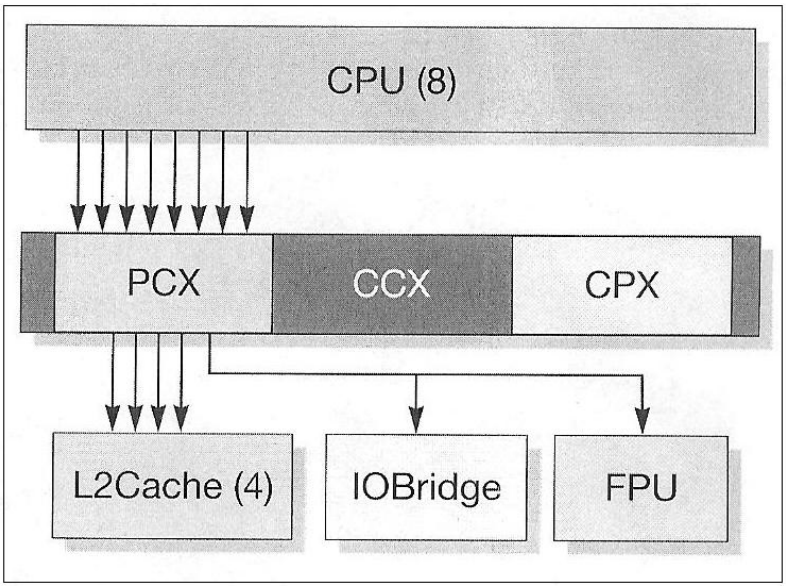


Figura 3.8-2 Interfaz del PCX. Cortesía OpenSPARC T1.

En un ciclo sólo un paquete puede ser entregado a un destino en particular. El CCX maneja dos tipos de solicitud comunicación. El primer tipo de solicitud contiene un paquete y será entregado en un ciclo. En cambio el segundo tipo de solicitud contiene dos paquetes y esos dos paquetes son entregados en dos ciclos.

El número total de ciclos requeridos de un paquete para viajar a través de la fuente hacia su destino puede ser mayor al número de ciclos de entrega del paquete.

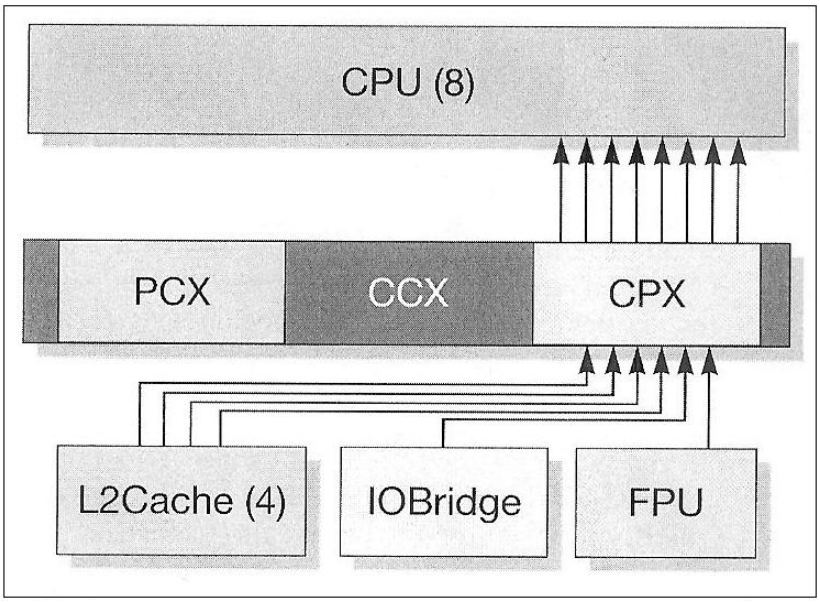


Figura 3.8-3 Interfaz del CPX. Cortesía OpenSPARC T1.

Procesamiento de Transacciones de PCX.

LOAD-CARGA

Una transacción LOAD transfiere datos desde L2 o I/O hacia el núcleo. Acceso cacheable del L2 tendrá un tamaño de 16 bits. Por lo tanto el tamaño en el paquete PCX deberá ser ignorado. 16 Bits de datos de carga se devuelven en el bus CPX.

PREFETCH-PREBUSQUEDA

Podrá ser emitida a L2. Desde la perspectiva L2 la prebúsqueda es una simple carga que no es cacheable en la L1. La L2 se hará valer en el bit PFL en el regreso del paquete CPX así el núcleo no sabrá actualizar el registro de archivos como sucedería con un “load”.

STORE-ALMACENAMIENTO

Solicitudes de Almacenamiento ocasionan que los datos sean actualizados en L2 o I/O. El núcleo SPARC enviará 64B de datos e indicará el tamaño de almacenamiento. Para almacenamientos menores a 64B los datos serán duplicados como se muestra en la siguiente tabla:

Operation	Data Fill
Stb 0x14	*0x14141414_14141414*
Sth 0x0102	0x01020102_01020102
Stw 0x01020304	0x01020304_01020304
Stx 0x01020304_05060708	0x01020304_05060708

Tabla 3.8 Campo de datos. Cortesía OpenSPARC T1.

Descripción del Bloque Funcional PCX.

El PCX contiene cinco módulos idénticos de mediadores o jueces (arbiter), uno para cada destino. El “arbiter” almacena los paquetes de las fuentes de un destino particular. El PCX después juzga y entrega los paquetes hacia su destino final. La siguiente figura muestra un diagrama de bloques de la mediación.

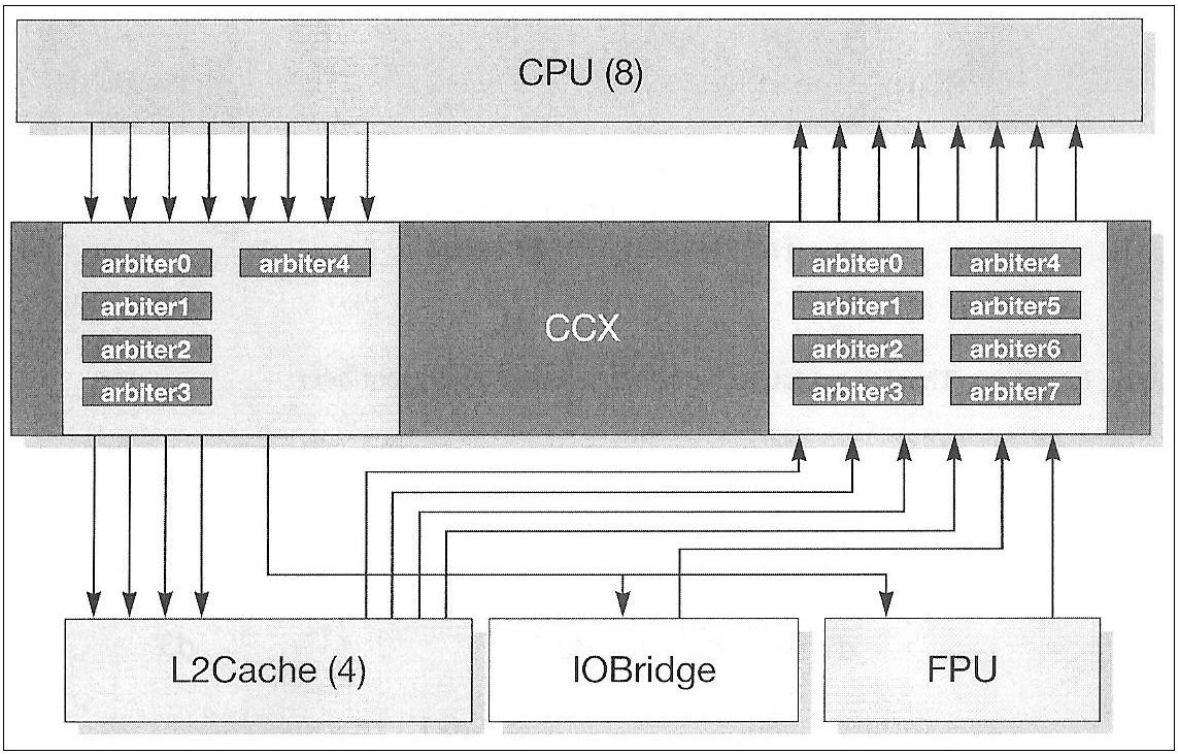


Figura 3.8-4 Bloque Interno de PCX y CPX. Cortesía OpenSPARC T1.

Flujo de Datos del Mediador PCX.

Si bien el flujo de datos es similar en el interior de los demás jueces (arbiter) se hará la descripción de sólo uno de ellos (ARB0). Se cuenta con un bus con un ancho de 124 bit por cada núcleo SPARC que se extiende a los cinco jueces (un bus por cada juez).

El ARB0 puede recibir paquetes de cualquiera de los ocho núcleos. Por lo tanto ARB0 contiene ocho colas, y cada cola es una entrada FIFO y cada entrada puede contener un paquete. Un paquete tiene un ancho de 124 bit y contiene la dirección, los datos y los bits de control. ARB0 entrega paquetes al Banco “Bank0” con un ancho de 124 bit. En la figura abajo descrita, se ilustra el flujo de datos:

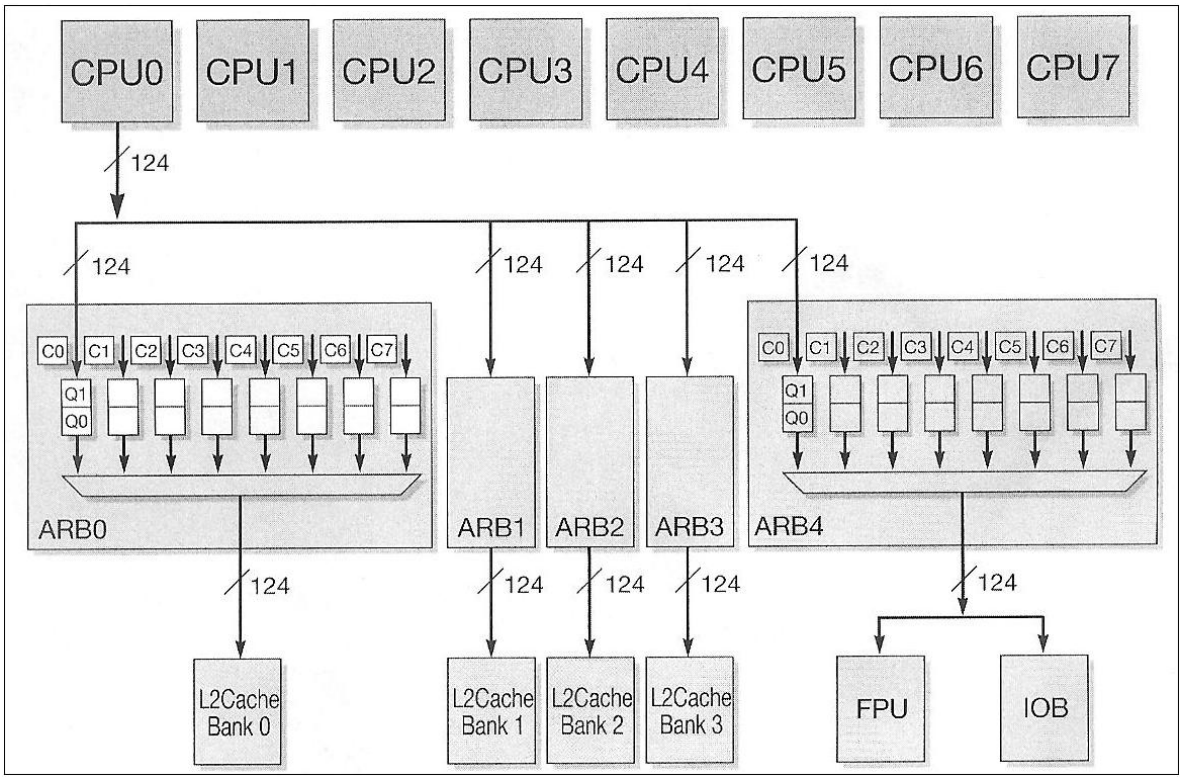


Figura 3.8-5 Flujo de Datos dentro de un "arbiter" en el PCX. Cortesía OpenSPARC T1.

ARB1, ARB2 y ARB3 reciben paquetes de la L2 cache Bank1, Bank2 y Bank3 respectivamente. ARB4 recibe paquetes tanto de la FPU y del puente I/O.

Flujo de Control del Mediador PCX.

El ARB0 gestiona paquetes en el orden en que los recibe. Por lo tanto un paquete recibido en el ciclo 4 se enviará antes de un paquete recibido en el quinto ciclo. Cuando múltiples fuentes administran un paquete en el mismo ciclo, ARB0 sigue una política "roun-robin" para arbitrar los paquetes de las fuentes.

Un bus de 5 bit por cada CPU y el bit correspondiente al destino será alto mientras todos los demás bits sean bajos.

El esquema de arbitraje del mediador se implementa mediante un tablero como se ilustra a continuación:

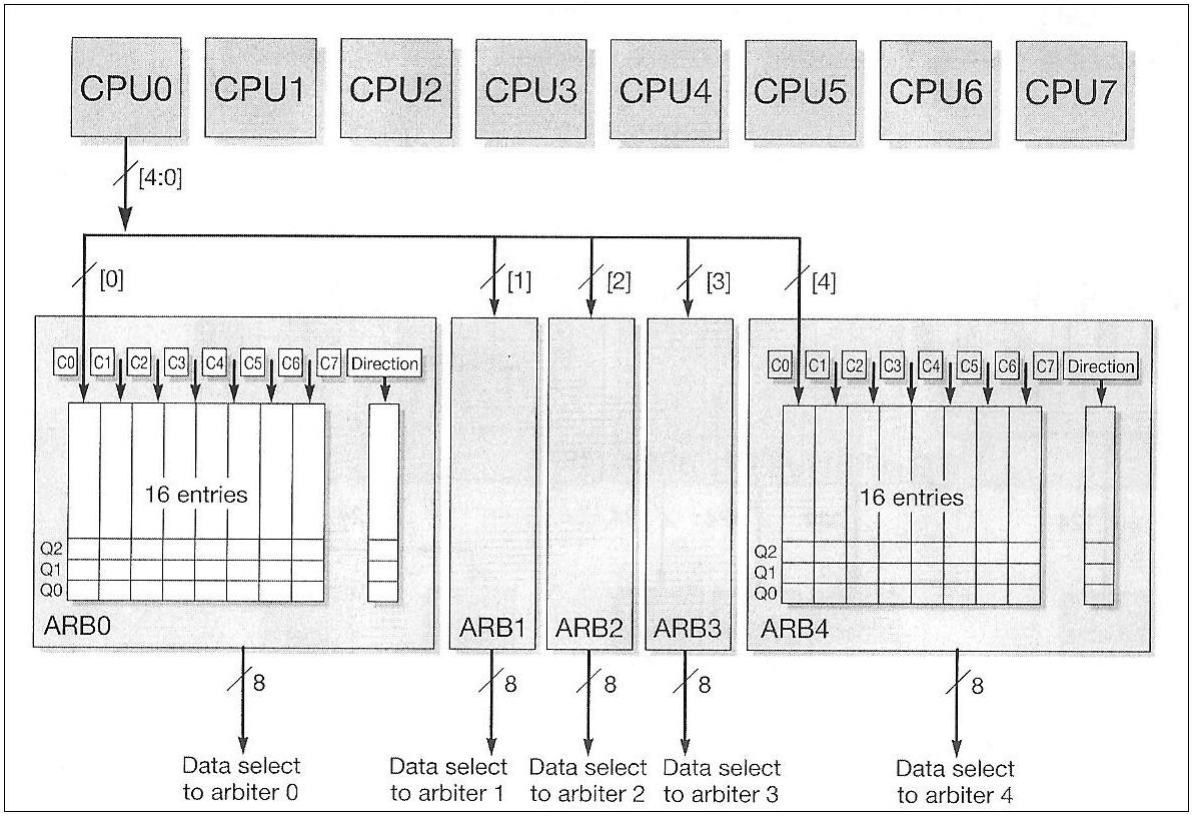


Figura 3.8-6 Flujo de Control del "arbiter" en el PCX. Cortesía OpenSPARC T1.

El tablero consta de ocho FIFO´s. Cada FIFO tiene 16 entradas y cada entrada tiene un solo bit válido recibido desde su correspondiente CPU. Cada entrada válida del FIFO representará un paquete válido de una fuente para el Bank0 cache L2. Desde cada fuente se podrá enviar como máximo dos entradas para el Bank0 cache L2, pero no puede haber dos bits válidos por cada FIFO. Por lo tanto el tablero puede tener un máximo de 16 bits válidos. El máximo caso se representa cuando el Bank0 cache L2 no es capaz de procesar nuevas entradas.

3.9 CACHE L2.

Visión General.

El cache L2 del procesador OpenSPARC T1 tiene un tamaño de 3 Mbytes y se compone de cuatro bancos simétricos que se intercalan en una frontera de 64 bytes. Cada banco opera independientemente uno de otro. Cada banco es formado por un conjunto de 12 vías asociativas con un tamaño de 768 Kbytes. El tamaño de bloque (línea) es de 64 bytes y cada banco cache L2 tiene 1024 conjuntos.

El caché L2 acepta solicitudes desde el PCX del núcleo SPARC y le responde a la CPX. La cache L2 también es responsable de mantener la coherencia en el chip en todos los chaches L1 y de mantener una copia de las etiquetas L1 en un directorio.

Descripción Funcional del Banco Cache L2.

El caché L2 está organizado en cuatro bancos idénticos, en el cual, cada banco cuenta con su propia interfaz, con la J-bus, el controlador DRAM y el larguero o barra cruzada del CPU (CCX).

Cada banco caché L2 realiza interfaz con ocho núcleos SPARC a través del larguero PCX. El PCX realiza la administración de las peticiones o solicitudes de la caché L2 como: carga, almacenamiento, accesos, etc, hacia los ocho núcleos apropiados. El PCX también acepta el regreso de datos, invalidación de paquetes, almacenamiento de paquetes de cada banco caché L2 y los reenvía al CPU apropiado.

Cada banco caché L2 consiste en los siguientes tres sub-bloques:

- sctag (secondary cache tag): el cual contiene un arreglo de etiquetas y de VUAD, un directorio cache L2 y un controlador caché.
- scbuf: contiene un buffer de escritura trasera (WBB), buffer de llenado (FB) y un buffer DMA.
- scdata: contiene el arreglo scdata.

Los cuatro estados de bits de “sctag” están organizados en un arreglo de puertos en el VUAD L2, y esos cuatro estados son: Valid (V), Used (U), Allocated (A) y Dirty (D). Este arreglo de bits cuenta con 2 puertos de lectura y dos de escritura.

La palabra “scdata” implica un arreglo de bancos en una estructura de puerto SRAM. Cada caché L2 tiene un tamaño de 768 Kbytes con su línea lógica de 64 bytes.

Los estados de tubería (pipeline) de la Caché L2 son ocho (C1 a C8).

3.10 PUENTE DE ENTRADA/SALIDA (I/O BRIDGE).

El puente de Entrada/Salida (IOB) es la interfaz entre la barra de cruce caché CPU (CCX) y los demás bloques en el procesador OpenSPARC T1. Las principales funciones se presentan a continuación:

- Decodificación de direcciones I/O.
    - Mapas o códigos del destino propio interno o externo.
    - Genera el registro de control y estado de acceso de agrupaciones.
    - Acceso programado de entrada/salida al J-Bus externo.
- Interrupciones:

  - Se encarga de recolectar las interrupciones de agrupaciones y del J-Bus.
  - Reenvía interrupciones al núcleo e hilos correctos.
- Interfaz entre la Lectura/Escritura de la SSI.
- Provee pruebas de acceso a los puertos (TAP) a la SCR´s, memoria, caché L2 y CPU ASI´s.

Interfaces del IOB.

Las principales interfaces del puente de Entrada/Salida son:

- Crossbar (CCX).
- Bus de conexión universal (UCB).
- Puertos de depuración.
- Controlador del Fusible electrónico (EFC).

La siguiente figura muestra las interfaces del IOB del resto de los bloques.

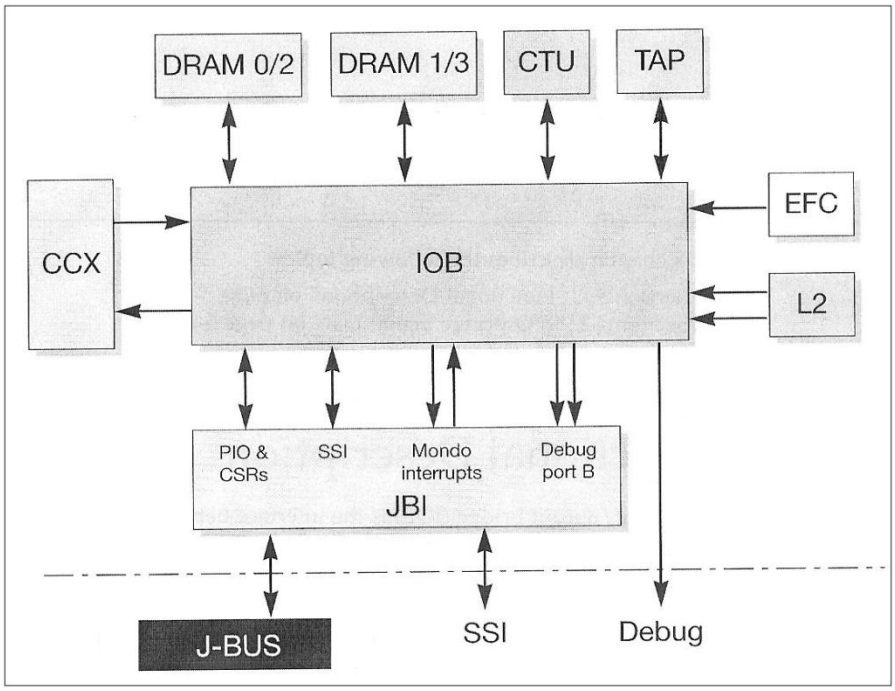


Figura 3.10-1 Interfaz IOB. Cortesía OpenSPARC T1.

Diagrama de Bloques del IOB.

En la siguiente figura se ilustra el diagrama de bloques interno del IOB. Las solicitudes PCX del CPU son procesadas por un bloque llamado “c2i” (CPU to I/O) y genera solicitudes UCB hacia varios bloques. Las solicitudes UCB desde varios bloques son procesadas por otro bloque llamado “i2c” (I/O to CPU) los cuales generarán paquetes CPX. Los registros de control/estado (CSRs) son controlados

por el bloque CSR. La depuración de bloques toma datos de la caché L2 y los manda a depurar al puerto A (un puerto externo).

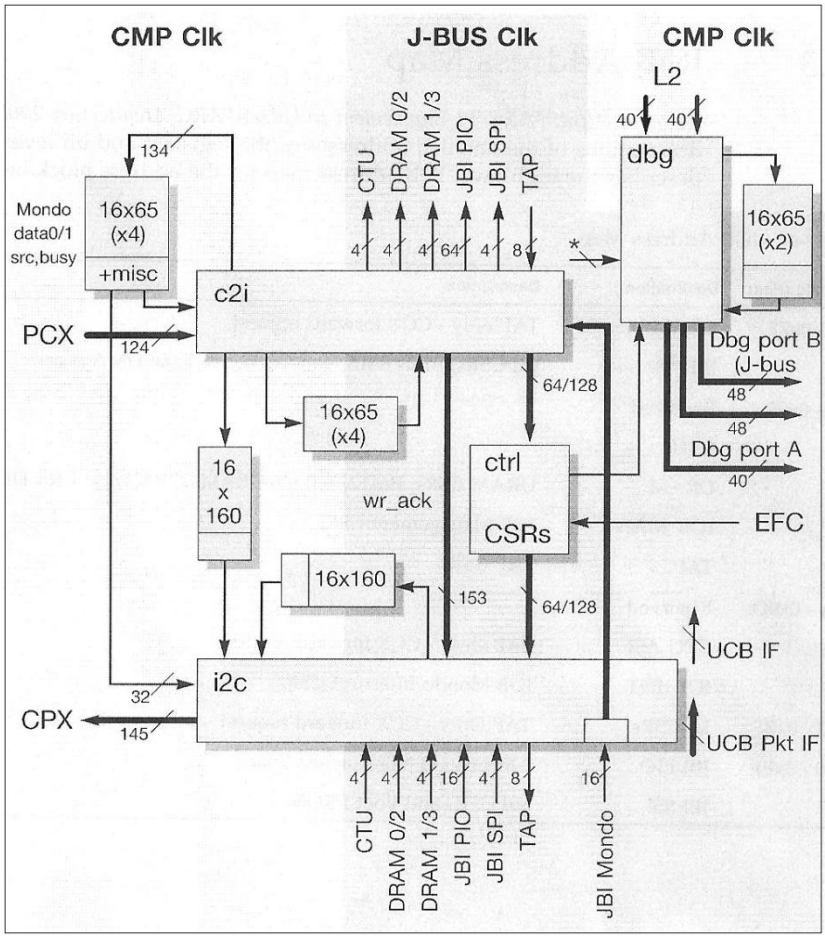


Figura 3.10-2 Diagrama de Bloques IOB. Cortesía OpenSPARC T1.

3.10.1 INTERFAZ J-BUS.

Descripción Funcional.

El bloque JBI en el procesador OpenSPARC T1 realiza interfaz con los siguientes bloques:

- Caché L2- para leer y escribir datos hacia caché L2.
- IOB- para entradas/salidas programadas (PIO), interrupciones y depuración de puertos.

Muchos de los sub-bloques JBI usan el reloj J-bus y otras partes del reloj del núcleo.

A continuación se ilustra el diagrama de bloques de JBI.

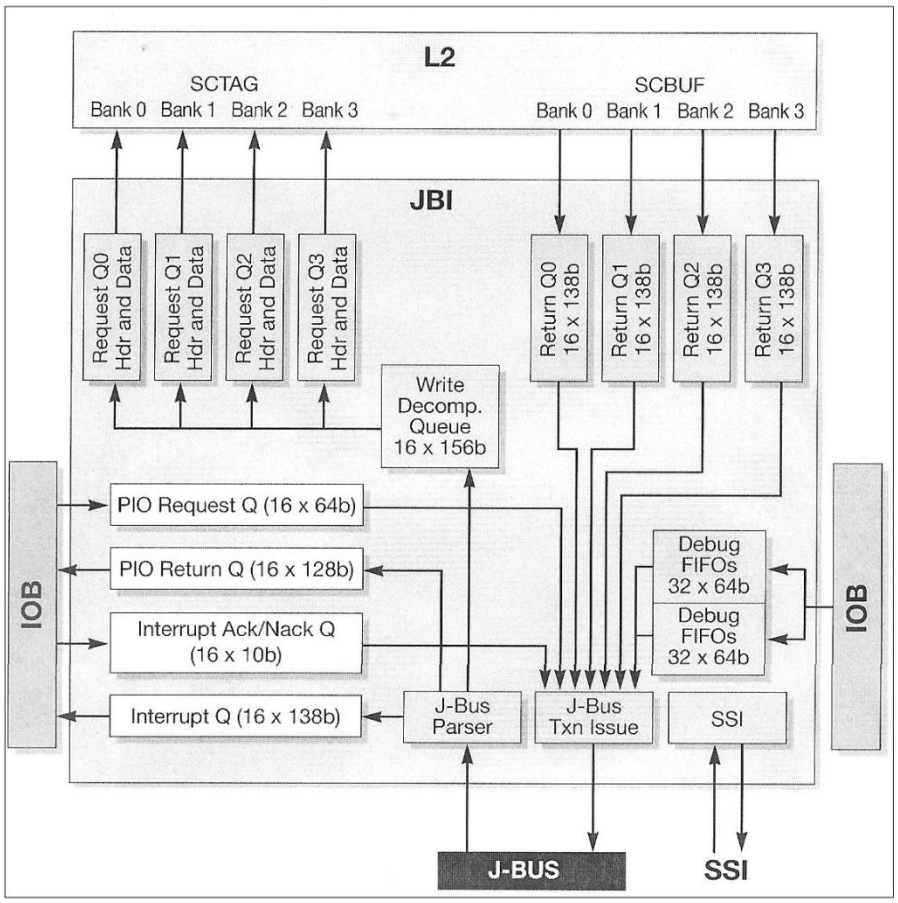


Figura 3.10-3 Diagrama de Bloques de JBI. Cortesía OpenSPARC T1.

3.11 UNIDAD DE PUNTO FLOTANTE (FPU).

Descripción Funcional.

La unidad de punto flotante del OpenSPARC T1 tiene las siguientes características:

- Implementa el conjunto de instrucciones de punto flotante del SPARC V9.
- La FPU no apoya el conjunto de instrucción visual (VIS).
- La FPU es una fuente compartida del procesador OpenSPARC T1. Cada uno de sus ocho núcleos puede tener un máximo de una instrucción FPU.
- El archivo de registro de punto flotante (FRF) y estado de registro de punto flotante (FSR) no están situados físicamente dentro de la FPU.
- Incluye tres tuberías independientes de ejecución:
    - Floating Point Adder (FPA)- Realiza suma, resta, comparación y conversión.
    - Floating Point Multiplier (FPM)- Realiza multiplicación.
    - Floating Point Divider (FPD)- Realiza División.
- Una instrucción por ciclo puede ser emitida.

- Una instrucción por ciclo puede ser completada y liberada de la FPU.

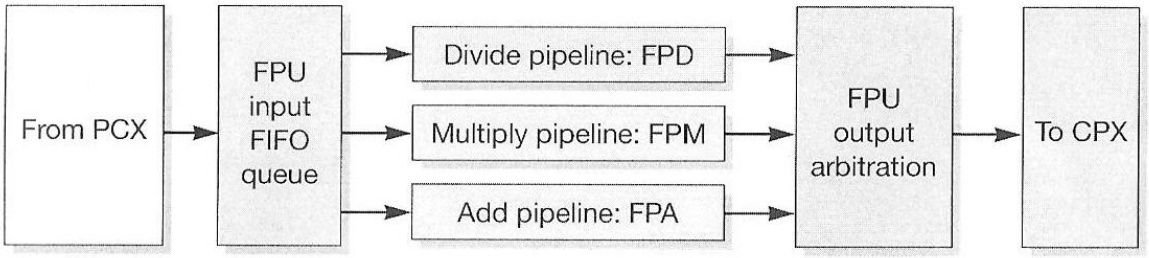


Figura 3.11-1 Diagrama de Bloques de la Función de la FPU. Cortesía OpenSPARC T1.

CONTROLADOR DRAM.

Descripción Funcional.

El controlador DDR-II DRAM del OpenSPARC T1 tiene las siguientes características:

- Contiene cuatro controladores DRAM independientes, y cada controlador está conectado a un Banco Caché L2 y a un canal de memoria DDR-II
- Tiene un máximo espacio de dirección de 37 bits para un tamaño de memoria máximo de 128 Gbytes.
- Las líneas caché de 64 bytes se intercalan a través de cuatro canales.
- Su rango de operación es de 125 MHz hasta 200 MHz con una velocidad de datos de 250 hasta 400 MT/sec.
- Su pico de ancho de banda es de 23 Gbyte/sec a 200 MHz.
- El controlador DRAM opera en dos modos: modo de cuatro canales y modo de dos canales, el cual es programable.

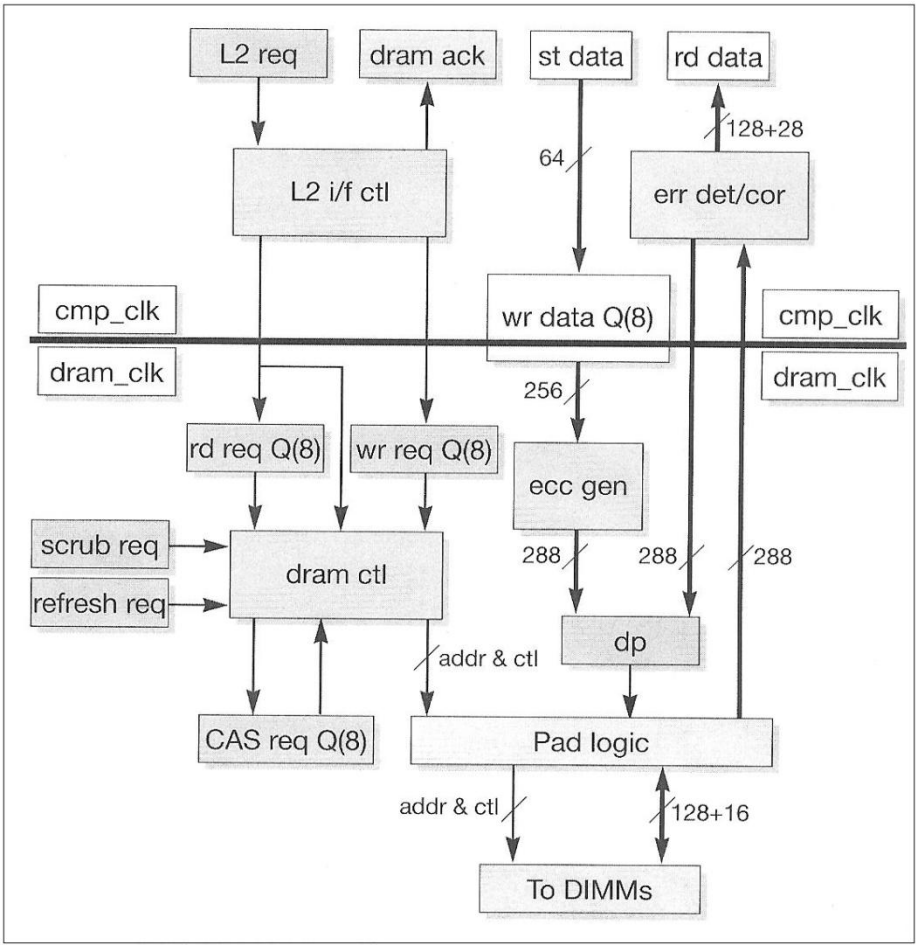


Figura 3.11-2 Diagrama de Bloques del Controlador DDR-II DRAM. Cortesía OpenSPARC T1.

Prioridad de Arbitraje.

Las solicitudes leídas cuentan con mayor prioridad sobre las peticiones de escritura, pero existe un contador que habilita la escritura. El controlador DRAM nunca verá solicitudes leídas seguidas de solicitudes escritas y el orden de prioridad será como se describe a continuación:

1. Actualizar solicitudes.
2. En espera de la columna de dirección de estrobos (CAS) y solicitudes (round-robin).
3. Solicitudes de fila de dirección de estrobos (RAS).
4. Espera de solicitudes escritas RAS que coinciden con direcciones.
5. Solicitudes de lectura o escritura RAS.

Diagramas de estado del Controlador DRAM.

La siguiente figura presenta el nivel superior del diagrama de estado del controlador DRAM. Un software deberá iniciar el controlador.

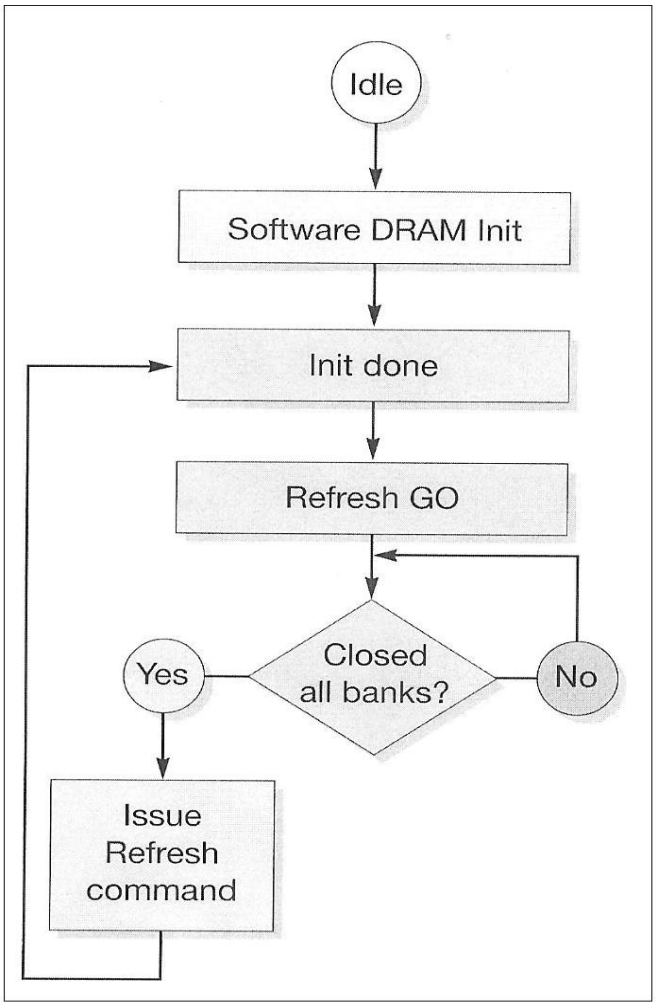


Figura 3.11-3 Diagrama de nivel superior del Controlador DDR-II DRAM. Cortesía OpenSPARC T1.

CONTROL DE ERRORES.

El procesador OpenSPARC T1 detecta, registra y reporta toda serie de errores al software. A continuación se describirán brevemente los tipos de errores y cómo se detectan, registran y reportan dichos errores.

Existen tres tipos de errores en el procesador OpenSPARC T1:

1. Errores Corregibles (CE).

Son corregidos por el hardware y él mismo puede generar trampas e interrumpir para dar un seguimiento de la frecuencia de errores.

2. Errores Incorregibles (UE).

Estos tipos de errores no pueden ser corregidos por el hardware y deberán ser corregidos por el software.

3. Errores Fatales (FE).

Los errores fatales pueden crear potencialmente daños y se tendrá que realizar un restablecimiento.

CAPÍTULO 4.- RESULTADOS Y CONCLUSIONES.

Este documento describió la revisión de arquitecturas computacionales como una herramienta eficaz para estudiar la asignatura “Arquitectura de Computadoras” en el que se tiene un gran número de ideas concisas para que el alumno que cursa dicha asignatura realice su propio diseño. Además, cuenta con un estudio general de las arquitecturas más comerciales líderes del mercado actual: Intel, Motorola y OpenSPARC; siendo ésta última en este texto la más importante y adecuada para su estudio y diseño ya que cuenta con código abierto para que el diseñador (alumno) pueda comprender su organización y arquitectura para realizar su propio diseño. El presente trabajo es una excelente alternativa para llevar a buen término los proyectos de desarrollo limitados por la carencia de infraestructura por parte del alumno para su aprendizaje en el diseño de microprocesadores. Con la finalidad de exponer ideas introductorias involucradas en el diseño de microprocesadores se describió con un buen nivel de detalle las características de la arquitectura OpenSPARC.

Para que el presente documento funcione como una fuente de información y de fundamentar la documentación de diseño, se presentaron las características más importantes y generales de otras arquitecturas: Intel y Motorola; para que el lector obtenga una breve información acerca de varios tipos de arquitecturas usados en los principales procesadores del mercado actual y sea capaz de diferenciarlos con el diseño OpenSPARC a través de la organización y diseño de cada una de ellas.

Se ha probado en base a sus características mencionadas en esta tesis, que la arquitectura OpenSPARC cuenta con la mejor organización y diseño respecto a los demás fabricantes ya que su arquitectura está mejor diseñada y distribuida, un ejemplo específico es su “pipeline” o tubería, no obstante de esta arquitectura se han basado algunos fabricantes para diseñar sus procesadores.

Sun Microsystems fabricante de la Arquitectura SPARC nos presentó el camino futuro de procesadores al liberar las especificaciones de diseño, y su estrategia de este tipo de procesadores ha sido desde el inicio, avanzar y lograr mejoras en el desempeño para hacer eficaz el desempeño para la línea de procesadores.

Finalmente este trabajo de tesis plantea la propuesta de Arquitecturas Computacionales a través de un estudio sólido de Arquitectura de Computadoras a nivel organización de computadoras digitales comparando los microprocesadores del mercado actual desde un estudio breve de la organización y tipos de computadoras, hasta un estudio más profundo cuyo conjunto de instrucciones permite poder ejecutar muchos tipos de algoritmos lo que lo hace aplicable para los cursos: “Arquitectura de Computadoras”, “Microprocesadores”,

entre otros a nivel licenciatura. Con esta revisión, se sientan las bases para el estudio de arquitecturas más complejas y/o sucesoras a las empleadas en la actualidad. Cabe mencionar que la arquitectura OpenSPARC es una propuesta, y cada usuario debe decidirse a favor o en contra de determinada arquitectura de procesador en función de la aplicación concreta que quiera realizar. Nunca será decisiva únicamente la capacidad de procesamiento del microprocesador, y sí la capacidad real que puede alcanzar el sistema en su conjunto, por otra parte, los costos también deberán ser evaluados por el usuario.

BIBLIOGRAFÍA.

Miles Murdocca; Vincent Heuring, *"COMPUTER ARCHITECTURE AND ORGANIZATION, An Integrated Approach"*, Ed Wiley, United States of America 2007.

John Paul Shen; Mikko H. Lipasti, "MODERN PROCESSSOR DESIGN, Fundamentals of Superscalar Processors", Ed Mc Graw Hill, 2005.

Carl Hamacher; Zvonko Vranesic; Safwat Zaky, "COMPUTER ORGANIZATION", International Edition, Fifth Edition, 2002.

David L. Weaver, "OPENSPARC INTERNALS BOOK", Sun MicroSystems, Inc. First Edition, 2002.

Savage Jesús, Vázquez Gabriel, "DISEÑO DE MICROPROCESADORES", Facultad de Ingeniería, México, 2004.

Pardo Fernando, "VHDL, LENGUAJE PARA DESCRIPCIÓN Y MODELADO DE CIRCUITOS", Universidad de Valencia, 1997.

"Arquitectura de una Computadora", Universidad Tecnológica Nacional, facultad Regional Rosario, Departamento Ingeniería Eléctrica.

ALTERA CORPORATION, Flex 10K Embedded Programmable Logic Device Family Data Sheet.

ALTERA CORPORATION, Max 7000 Programmable Logic Device Family Data Sheet.

XILINX, Spartan-3 FPGA Family Complete Data Sheet, 2006.

OpenSPARC™ T1 Microarchitecture Specification, Sun Microsystems, August 2006.

www.sun.com

http://www.infor.uva.es/~fernando/asignaturas/microp/pentium4.pdf

http://microprocesadoresymicrocontroladores.blogspot.com/2008_04_01_archive.html

Capítulo 2

SOBRE SPARC

http://www.cpu-collection.de/?l0=i&i=1999&s=big&tb=1&n=0&sd=1(own website)

http://www.oracle.com/us/sun/index.html

http://www.dcc.uchile.cl/~rbaeza/cursos/proyarq/mroco/page3.htm

http://www.alegsa.com.ar/Dic/sparc.php

http://es.sun.com/products/pdf/sparc-t5240.pdf

http://www.worldlingo.com/ma/enwiki/es/SPARC#External_links

http://www.apuntesdeinformatica.netai.net/foro/contenidos/simm/Unidades.Didacticas/Unidad.Didactica.2/2.Arquitectura/Componentes.HW.de.un.PC/Fabricantes.HW.para.PC.pdf

http://redalyc.uaemex.mx/pdf/730/73012215014.pdf

http://www.azc.uam.mx/publicaciones/enlinea2/num1/1-2.htm

www.OpenSPARC.net

SOBRE INTEL

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

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

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

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

http://www.tomshardware.com

SOBRE MOTOROLA

Hamacher, Vranesic, Zaky, “Computer Organization”, International Edition, Mc Graw Hill, fifth edition 2002.

http://cedicyt.usach.cl/microcomputadores/2004/grupo3/Jose%20Morey/MICROCOMPUTADORES/PRIMERA.htm