Introducción a la arquitectura IA-64 y al procesador Itanium 2

Tomás Balderas Contreras
balderas@ccc.inaoep.mx

Instituto Nacional de Astrofísica, Óptica y Electrónica
Coordinación de Ciencias de la Computación
Arquitectura de Computadoras

24 de noviembre, 2002

Contenido

Resumen

En 1994, Hewlett-Packard e Intel iniciaron un proyecto de investigación que tuvo como objetivo diseñar una nueva arquitectura de conjunto de instrucciones (ISA) para microprocesadores de 64 bits. El objetivo fue definir la arquitectura sucesor de los conjuntos de instrucciones IA-32 de Intel y PA-RISC de Hewlett-Packard. El resultado de este esfuerzo conjunto es la arquitectura IA-64 y sus primeras implantaciones, los procesadores Itanium e Itanium 2. Los principios fundamentales que dirigieron el diseño de IA-64 fueron establecidos a partir 1989, durante las investigaciones realizadas por el personal de Hewlett-Packard Laboratories; estos principios definen un nuevo enfoque en el diseño de arquitecturas para microprocesadores de propósito general: EPIC (*Explicitly Parallel Instruction Computing*).

Palabras Clave: EPIC, IA-64, Itanium, Compiladores, Procesadores Superescalares, VLIW.

1 EPIC

En la actualidad el mejor medio para alcanzar niveles más altos de rendimiento en un procesador, sin efectuar modificaciones en los programas que ejecuta, es el Paralelismo a Nivel de Instrucción (*Instruction Level Parallelism*, ILP), que consiste en ejecutar varias instrucciones simultáneamente. El ILP se lleva a cabo combinando técnicas de optimización de código en los compiladores y de diseño de hardware (e.g. ejecución fuera de orden, procesamiento superescalar, etc.). Estas técnicas son invisibles al programador e incrementan la complejidad del procesador. Los procesadores que alcanzan altos niveles de ILP se clasifican en dos grupos ([7]):

Procesadores superescalares: Las instrucciones ejecutadas no tienen conocimiento de la organización del hardware y, por lo tanto, no proporcionan al procesador información explícita sobre el paralelismo existente. El procesador decide en forma dinámica el orden de ejecución de las instrucciones y la asignación de operaciones a unidades funcionales; este dinamismo permite tomar en cuenta factores que solo pueden ser determinados en tiempo de ejecución. Sin embargo, la complejidad de este tipo de procesadores es considerable.

Procesadores VLIW: Las instrucciones proporcionan información explícita sobre el paralelismo que hay entre ellas. El compilador tiene pleno conocimiento del procesador e identifica el paralelismo en el programa y comunica al hardware las operaciones que son independientes unas de otras y las unidades funcionales donde deben ser procesadas, por lo tanto el procesador no realiza verificaciones adicionales. Sin embargo, puede ocurrir que no haya una compatibilidad adecuada del código entre los miembros de una familia de procesadores. Son incapaces de determinar de forma precisa los factores dinámicos que afectan la ejecución.

1.1 El enfoque EPIC

EPIC es una clase de arquitecturas que se suscriben a una filosofía común. EPIC es una evolución de VLIW que adopta principios superescalares para alcanzar altos niveles de ILP sin incrementar la complejidad del hardware a niveles inaceptables ([5, 6]).

Las características más importantes de este esquema son las siguientes:

1. El compilador tiene la responsabilidad de diseñar el plan de ejecución para cada programa. El comportamiento del procesador en tiempo de ejecución debe ser predecible y controlable desde el punto de vista del compilador. Además, una arquitectura EPIC debe proporcionar los recursos que permitan al compilador realizar un reordenamiento de código de manera correcta.

2. Existen varias ambigüedades en un programa, producto de condiciones que solo pueden ser determinadas en tiempo de ejecución, que provocan que haya varios casos a considerar en el flujo de ejecución del programa; siendo imposible para el compilador optimizar todos los casos. Por lo tanto el compilador optimiza el caso más probable y la arquitectura proporciona los medios para garantizar los resultados correctos en cualquiera de los otros casos.

3. El conjunto de instrucciones debe ser lo suficientemente rico para expresar las decisiones del compilador sobre las instrucciones que deben ser ejecutadas simultáneamente y los recursos que les son asignados. Además, una arquitectura EPIC debe proporcionar los medios que permitan controlar, mediante instrucciones, componentes que normalmente son gestionados por la microarquitectura u organización, por ejemplo la memoria caché.

2 La arquitectura IA-64

Las características más importantes de la arquitectura IA-64 son las siguientes ([3, 4]):

- Un extenso archivo de registros
- Ejecución condicional de instrucciones
- Especulación de control y de datos
- Optimización de llamados a procedimientos a través de una pila en el archivo de registros
- Predicción dinámica de saltos para minimizar el costo de las transferencias de control
- Varias unidades de ejecución y puertos a memoria
- Modos *big-endian* y *little-endian* de almacenamiento de datos en memoria (*bi-endian*)
- Memoria direccionable en unidades de 1, 2, 4, 8, 10 y 16 bytes
- Instrucciones de 41 bits de longitud
- Las instrucciones están agrupadas en paquetes (*bundles*) de 16 bytes de longitud. Cada paquete contiene tres instrucciones
- Compatibilidad con IA-32 en forma directa y con PA-RISC mediante traducción binaria
- Instrucciones paralelas para procesamiento multimedia compatibles con MAX-2 y MMX

2.1 Registros

La figura 1 muestra el conjunto de registros definidos por IA-64 que son visibles a los programas de aplicación. El objetivo de explotar e incrementar el paralelismo a nivel de instrucciones motiva la asignación de muchos recursos de almacenamiento temporal. A continuación se proporciona una descripción de los registros:

Registros de propósito general (gr_0–gr_{127}): Almacenan los datos empleados por todas las operaciones enteras. Son visibles por todas las instrucciones, tanto las de nivel de usuario como las privilegiadas. Para cada registro existe un bit **NaT** (Not a Thing) que cuando tiene un valor de uno indica que alguna operación especulativa de carga hacia dicho registro ha generado una excepción que debe ser atendida. El valor del registro gr_0 es invariablemente cero.

Los registros gr_{32}–gr_{127} forman la *pila de registros*. Cada procedimiento de un programa solicita que le sea asignado un cierto número de estos registros, los cuales conforman el *marco de pila* (*stack frame*) del procedimiento. Este grupo de registros está conformado por un área de *registros locales* y otra de *registros de salida*.

Registros de punto flotante (fr_0–fr_{127}): Son usados por todas las operaciones de punto flotante. Se encuentran visibles en todos los niveles de privilegio (modo usuario y supervisor). Los valores de fr_0 y fr_1 son siempre +0.0 y +1.0, respectivamente, y cualquier intento de escribir algún valor a estos dos registros genera una excepción.

Predicados (pr_0–pr_{63}): Son registros de un bit de longitud y son visibles por todas las instrucciones. Son empleados por las operaciones de comparación para almacenar su resultado, por las instrucciones de transferencia de control condicional y como modificador de otro tipo de instrucciones, en cuyo caso el valor del predicado determina si la operación debe

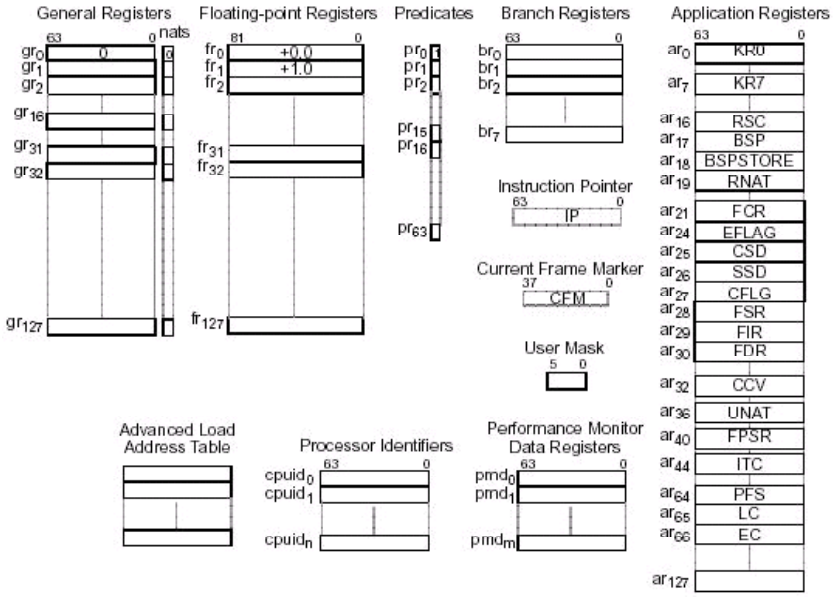

Figura 1: Los registros definidos por la arquitectura IA-64.

modificar el estado del procesador (1) o no (0). El registro pr_0 siempre almacena el valor uno.

Registros de saltos (br_0–br_7): Son visibles en todos los niveles de privilegio y, por lo tanto, por todas las instrucciones. Almacenan la dirección destino para las instrucciones de transferencia de control que emplean el modo de direccionamiento indirecto.

Apuntador de instrucciones (IP): Este registro almacena la dirección en memoria del paquete que contiene la instrucción en ejecución. Es posible leer directamente el valor del registro, se incrementa conforme se ejecutan las instrucciones y puede ser modificado por una instrucción de transferencia de control. Los cuatro bits menos significativos de la dirección almacenada son siempre cero, pues cada paquete es de 16 bytes de longitud y sus direcciones en memoria son múltiplos de 16.

Registros de aplicación (ar_0–ar_{127}): Estos registros almacenan datos de propósito específico o registros de control para ciertas funciones del procesador. Por ejemplo, el registro ar_{40} almacena el registro de estado de punto flotante (FPSR).

Marcador del marco actual (CFM): Cada marco de pila tiene asociado un marcador. Este registro de 38 bits es el marcador para el marco de pila activo y contiene información sobre su estado. Entre otras cosas, este estado incluye el número de registros que conforman el marco de pila y el número de registros de salida dentro del marco de pila.

2.2 Tipos y formato de instrucciones

La arquitectura IA-64 define cinco tipos de instrucciones, las cuales pueden ser ejecutadas por cierto tipo de unidades funcionales existentes en el procesador. La tabla 1 muestra más información al respecto.

Todas las instrucciones son de 41 bits de longitud y se agrupan en paquetes de tres instrucciones. Cada paquete tiene 128 bits de longitud y está compuesto de cuatro campos, tres de ellos almacenan instrucciones y, por lo tanto, cada uno es de 41 bits de longitud; el cuarto campo es de 5 bits de longitud y almacena un valor plantilla (*template*) que indica la forma en que se lleva a cabo el mapeo entre las instrucciones en el paquete y las unidades de ejecución. La figura 2 muestra el formato de los paquetes y la figura 3 ilustra los 32 mapeos posibles definidos por IA-64 para cada valor del campo plantilla. El procesador realiza un proceso denominado *dispersión* (*dispersal*) para asignar instrucciones en los paquetes a las unidades funcionales del microprocesador.

Los paquetes se encuentran almacenados en memoria en orden *little-endian*, es decir, la dirección del paquete contiene el byte menos significativo y al incrementar la dirección se pueden leer los bytes más significativos del paquete. El campo plantilla se encuentra en el byte menos significativo del paquete.

Existen muchos formatos para codificar las instrucciones y sus parámetros.

2.3 Ejecución condicional (*predication*)

En IA-64 cada instrucción puede estar etiquetada con el valor de un registro predicado (pr_0–pr_{63}), el cual de-

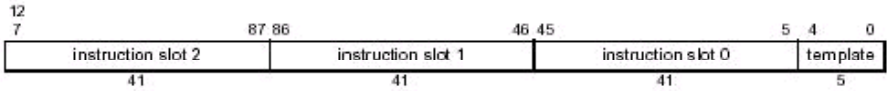

Figura 2: Formato de un paquete de instrucciones.

Tipo de instrucción	Descripción	Unidad de ejecución
A	Instrucción entera aritmética o lógica	I-unit o M-unit
I	Instrucción entera no aritmética o lógica	I-unit
M	Transferencia a memoria	M-unit
F	Instrucción de punto flotante	F-unit
B	Transferencia de control	B-unit
L+X	Extendida	I-unit/B-unit

Tabla 1: Relación entre los tipos de instrucciones y los tipos de unidades de ejecución.

pende del resultado de una instrucción de comparación. Cuando el valor del registro predicado es 1 la instrucción es ejecutada de forma normal y puede modificar el estado del CPU; sin embargo, si el valor del predicado es cero la instrucción no actualiza el estado del procesador o puede no ejecutarse del todo, comportándose como una instrucción `nop`. El objetivo de esta técnica es eliminar los saltos y reducir sus costos en el rendimiento, además de simplificar la labor de optimización del compilador ([7, 1]).

La figura 4a muestra un fragmento de código de una estructura *if then else*. En 4b observamos que un compilador para una arquitectura típica genera un código formado de cuatro bloques básicos[1], con los saltos para ejecutar la parte *else* si la condición de la comparación no se cumple y para saltar desde el bloque básico de la parte *then* al bloque que incrementa la variable `i`. En la figura 4c podemos observar que un compilador para EPIC elimina los saltos y etiqueta las instrucciones del bloque básico de la parte *then* con el registro p_1; mientras que las instrucciones del bloque básico de la parte *else* son etiquetadas con el registro p_2. La instrucción de comparación asigna valores a los registros predicados de tal forma que p_2 es siempre el complemento de p_1. De esta forma es posible despachar simultáneamente todas las instrucciones y garantizar que el programa se comporta de la manera correcta, debido a que algunas de estas instrucciones serán realmente ejecutadas y las otras no, dependiendo del valor de los bits de predicado.

2.4 Especulación

La técnica de especulación es resultado de la combinación de optimizaciones realizadas por el compilador y características del hardware. Consiste en mover y ejecutar una instrucción antes de su posición original en el flujo del programa, sin afectar la semántica del mismo. El objetivo es esconder la latencia de la instrucción y explotar el ILP entre dicha instrucción y otras.

2.4.1 Especulación de control

Cuando el compilador optimiza una secuencia de código de tal forma que reubica una instrucción de carga `ld` hacia alguna posición anterior al bloque básico que la contenía originalmente, la instrucción se convierte en una carga especulativa `ld.s`. La figura 5 muestra la forma en que la instrucción `ld` (5a) se transforma en una instrucción `ld.s` (5b) después de reemplazarla por la instrucción `check.s` y reubicarla varias instruccciones antes de la instrucción de salto `jmp`. En tiempo de ejecución la instrucción `ld.s` realiza la lectura a memoria y verifica la ocurrencia de alguna excepción (debido a que la dirección es inválida, a que ha ocurrido un fallo de página de memoria virtual, etc.), en este caso `ld.s` no desapacha la excpeción sino que pone a uno el bit **NaT** del registro destino. La instrucción `check.s` (verificación especulativa) se encarga de despachar la excepción e invocar al manejador apropiado del sistema operativo si el bit **NaT** es uno. Además la ejecución de la instrucción `check.s` depende de las instrucciones de comparación y de transferencia de control, de tal forma que el programa no recibe la excepción si la instrucción `check.s` no se ejecuta.

2.4.2 Especulación de datos

Si en el programa se tiene una instrucción de almacenamiento seguida de una instrucción de carga se puede determinar si las direcciones sobre las que operan ambas instrucciones se traslapan o no, en cuyo caso la operación de carga puede ejecutarse antes que la de almacenamiento. En casos como este la instrucción de carga

[1] Un *bloque básico* es una secuencia de instrucciones entre dos instrucciones de transferencia de control.

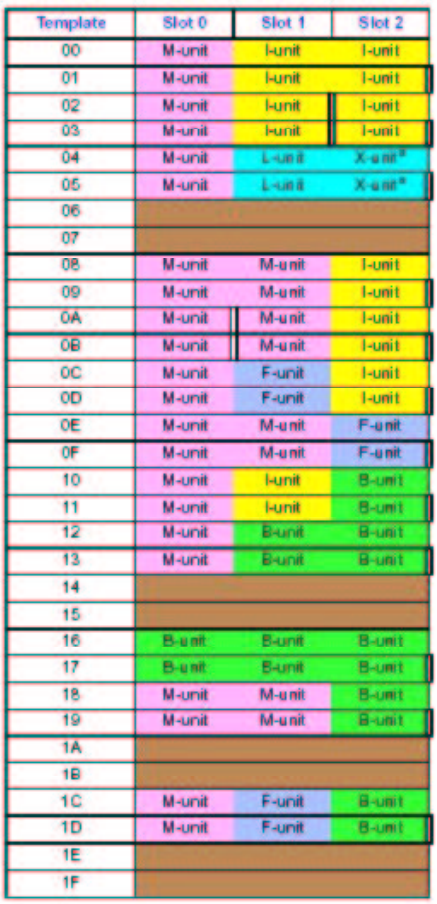

Figura 3: Mapeo entre instrucciones en paquetes y unidades de ejecución.

sería especulativa por datos con respecto a la instrucción de almacenamiento (*advanced load*).

2.5 Pila de registros

Cada procedimiento tiene acceso a la parte estática (gr_0–gr_{31}) del archivo de registros de propósitpo general y a una porción local o marco de pila en la parte de la pila de registros (gr_{32}–gr_{127}), cuyo tamaño puede variar entre cero y 96 registros.

La figura 6 ilustra el estado de la pila de registros previo a la llamada a un procedimiento, durante la ejecución del procedimiento y al regreso del procedimiento. La siguiente secuencia describe el funcionamiento de la pila en detalle:

1. Al principio el marco de pila del procedimiento A tiene asignados los 21 primeros registros de la pila (gr_{32}–gr_{52}), a los cuales tiene acceso mediante las direcciones de registro 32–52. El marco está dividido en la región de registros locales (gr_{32}–gr_{45}) y la región de registros de salida (gr_{46}–gr_{52}). Además el registro CFM almacena la longitud del área de registros locales (sol_a=14) y la longitud total del marco (sof_a=21).

2. Después del llamado al procedimiento B mediante la instrucción `br.call` el contenido del registro CFM se copia al campo PFM del registro ar_{64} (PFS — Previous Function State). El marco de pila del procedimiento B en este momento consiste únicamente de una región de registros de salida que se solapa con la región de registros de salida del marco del procedimiento A. Sin embargo, el registro gr_{46} de A es renombrado y visto como el registro gr_{32} por B. Todo lo anterior coincide con los nuevos valores almacenados en CFM, sol_{b1}=0 y sof_{b1}=7

3. El procedimiento B solicita una extensión de su marco de pila mediante la instrucción `alloc`. Esta instrucción expande el marco de 7 a 19 registros (gr_{32}–gr_{50}), define una nueva región de registros locales de 16 registros (gr_{32}–gr_{47}) y redefine la longitud del área de registros de salida a tres registros. `alloc` almacena la nueva longitud del marco de pila y de la región de registros locales (sof_{b2}=19 y sol_{b2}=16) en el registro CFM.

4. Finalmente, el procedimiento B termina, ejecuta la

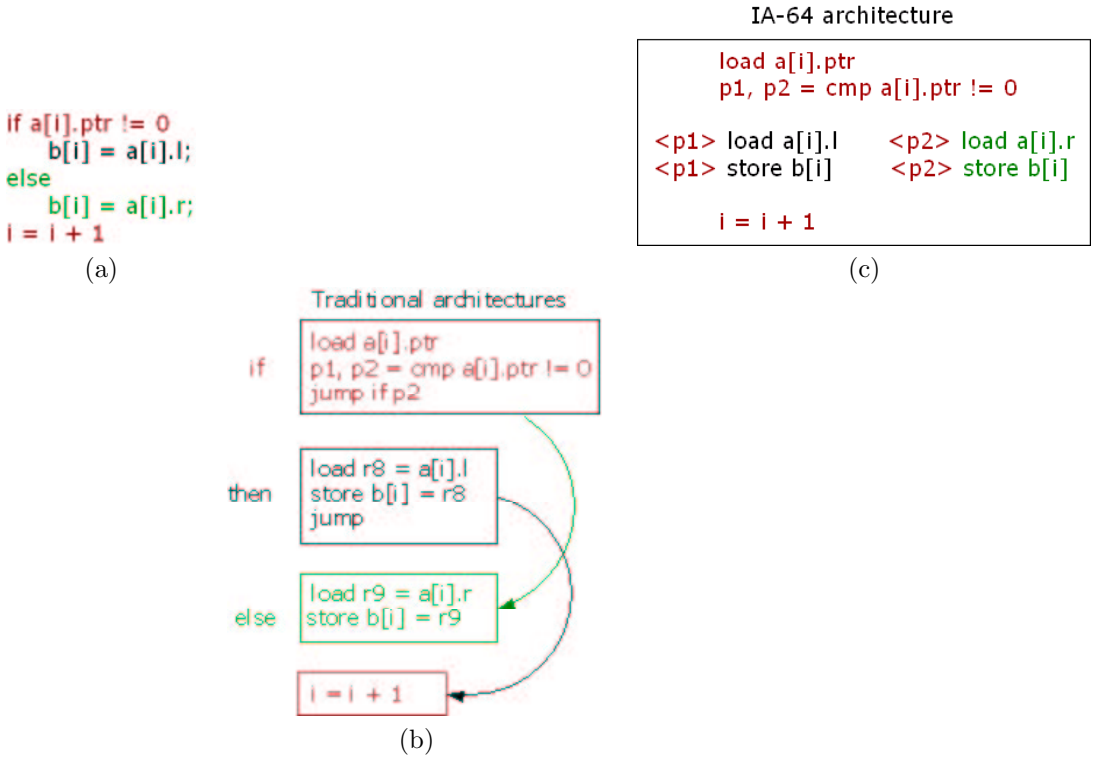

Figura 4: Uso de la técnica de ejecución condicional de instrucciones.

instrucción `br.ret` y regresa el control al procedimiento A. Después de ejecutar la instrucción el valor original del registro CFM se restaura desde el registro PFM, donde estuvo almacenado durante la ejecución del procedimiento B. La pila de registros vuelve al estado en el que se encontraba antes del llamado al procedimiento B.

El procedimiento A puede pasar parámetros al procedimiento B escribiéndolos en sus registros de salida, después del llamado el procedimiento B puede leerlos desde sus registros locales. B puede regresar algún valor escribiéndolo en el área local de su marco, parte de la cual se solapa con la región de salida de A. Este esquema permite que el paso de parámetros entre procedimientos se lleve a cabo enteramente dentro del archivo de registros, evitando los costosos accesos a memoria para guardar y restaurar los valores de los registros.

3 El procesador Itanium 2

Itanium 2 es la segunda implantación de la arquitectura IA-64. Algunas características de este procesador son las siguientes ([2]):

- Disponible en frecuencias de 1.0 GHz y 900 MHz
- 128 registros de propósito general y 128 registros de punto flotante con soporte para rotación de registros
- Motor de pila de registros (RSE —Register Stack Engine) para gestión efectiva de los recursos del procesador
- Soporte para especulación y ejecución condicional de instrucciones
- Jerarquía de tres niveles de memoria caché:
  - Memoria caché de primer nivel de datos (L1D) e instrucciones (L1I) de 16 KB
  - Memoria caché unificada de segundo nivel (L2) de 256 KB
  - Memoria caché unificada de tercer nivel (L3) de 3 MB ó 1.5 MB
- Bus de datos de 128 bits
- Direcciones virtuales de 64 bits y bus de direcciones físico de 50 bits
- Es capaz de despachar hasta 6 instrucciones (2 paquetes) por ciclo de reloj
- Pipeline de 8 etapas
- Múltiples unidades funcionales y puertos de expedición
- Soporte para programas de aplicación de IA-32

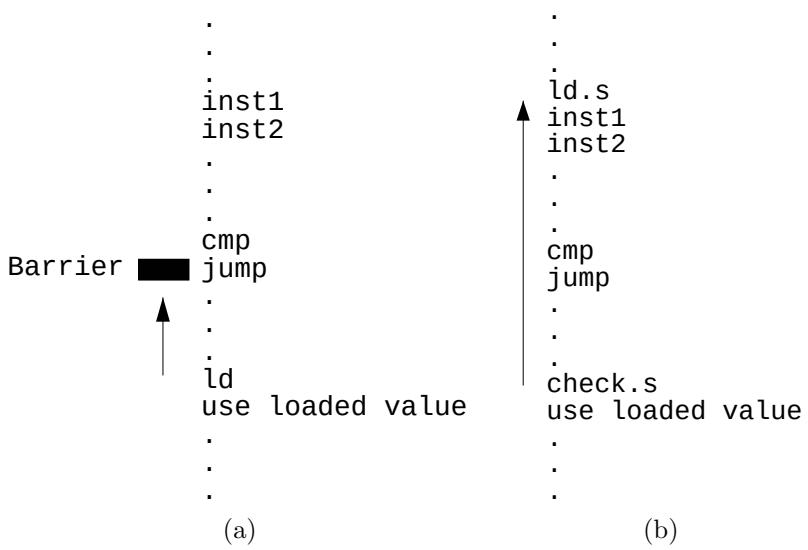

Figura 5: Uso de la técnica de especulación de control.

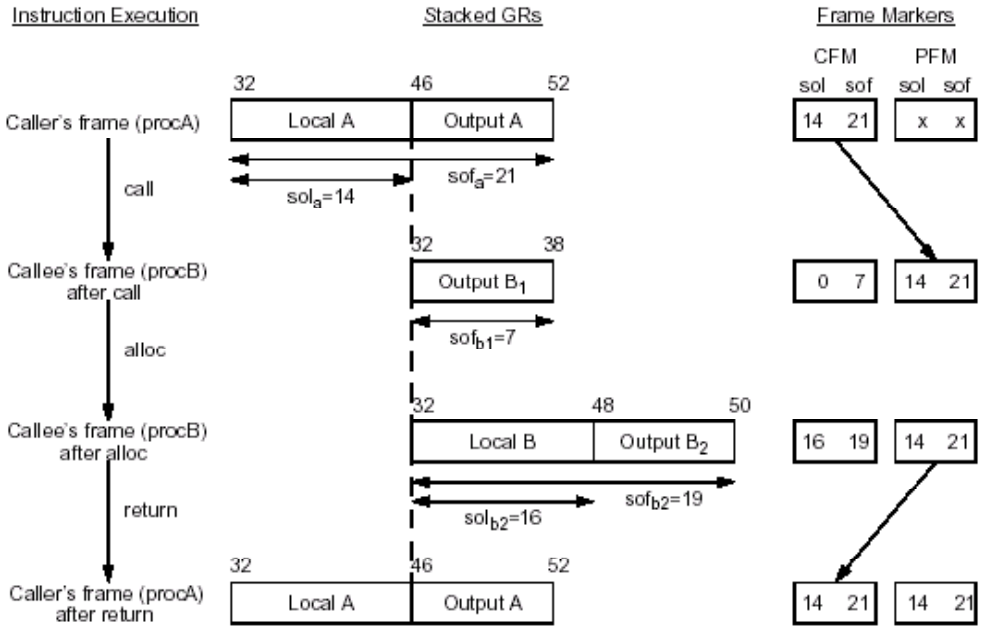

Figura 6: Funcionamiento de la pila de registros durante llamados a procedimientos.

- Procesador bi-endian, es decir, soporta tanto el modo de almacenamiento little-endian y el modo de almacenamiento big-endian

3.1 Unidades funcionales

El procesador Itanium 2 puede recolectar hasta 2 paquetes por ciclo de reloj y despachar hasta 6 instrucciones por ciclo. El procesador cuenta con las siguientes unidades funcionales, de las cuales solo algunas son utilizadas en cada ciclo de reloj:

- 6 unidades aritmético lógicas de propósito general (ALU0–ALU5). Llevan a cabo operaciones aritméticas, comparaciones, varias instrucciones multimedia, etc. Sin embargo, el procesador sólo puede despachar cuatro instrucciones por ciclo a estas unidades.
- 2 unidades enteras (I0–I1).
- 1 unidad de corrimiento (ISHIFT). Utilizada para llevar a cabo corrimientos y otras operaciones especiales.
- 1 unidad de caché de datos (DCU). Contiene cuatro puertos a memoria, generalmente dos puertos son empleados para operaciones de carga y los otros dos para operaciones de almacenamiento. El procesador puede dispersar un máximo de cuatro instrucciones a esta unidad por ciclo.
- 6 unidades de procesamiento multimedia (PALU0–PALU5). Llevan a cabo operaciones multimedia. El procesador puede despachar hasta 6 instrucciones por ciclo a estas unidades.
- 2 unidades paralelas de corrimiento (PSMU0–PSMU1).
- 1 unidad de multiplicación paralela (PMUL).
- 1 unidad de conteo de población (POPCNT).
- 2 unidades para suma y multiplicación de punto flotante (FMAC).
- 2 unidades para operaciones adicionales de punto flotante (FMISC). Llevan a cabo comparaciones de punto flotante, mezclas, etc.
- 3 unidades de saltos.

Todas las unidades están completamente segmentadas (*pipelined*) y en condiciones favorables cada una puede aceptar una instrucción por ciclo. El proceso de dispersión se lleva a cabo a través de 11 puertos de expedición en el procesador. Hay tres puertos para dispersar instrucciones de transferencia de control (B0, B1, B2) y ocho puertos para instrucciones de otro tipo (M0, M1, M2, M3, I0, I1, F0 y F1). Cada instrucción es asignada a un subconjunto de puertos en base a su tipo y después es mapeada a un puerto particular del subconjunto en base a la posición que ocupa dentro del grupo de instrucciones a dispersar. La figura 7 muestra un diagrama a bloques del procesador donde es posible ubicar los puertos de expedición, los componentes de la jerarquía de memoria caché, las unidades funcionales, etc.

3.2 Pipeline

El pipeline central del procesador está dividido en dos partes. La primera parte (Front-End) comprende las primeras dos fases y se encarga de recolectar dos paquetes por ciclo, es decir, seis instrucciones, de la memoria caché de instrucciones (L1I) y de depositarlas en el *búffer de instrucciones*. La segunda parte del pipeline (Back-End) está desacoplada de la primera, de tal manera que ambas son completamente independientes. En esta parte se lleva a cabo el proceso de dispersión, renombramiento de registros, detección de excepciones y modificación del estado del procesador.

El búffer de instrucciones es una cola que almacena ocho paquetes, es decir, 24 instrucciones. Este búffer permite a las dos partes principales del pipeline operar en completa independencia una de otra. La figura 8 muestra un esquema del pipeline y la siguiente es una breve descripción de cada etapa:

IPG (Instruction Pointer Generation). Recolecta paquetes de la memoria caché de instrucciones a partir de la dirección del apuntador de instrucciones.

ROT (Rotate). Rota el búffer de instrucciones para alinear los paquetes que serán procesados por la siguiente etapa del pipeline.

EXP (Expand). Esta etapa decodifica el campo plantilla (template) de cada paquete y dispersa hasta seis instrucciones a las unidades funcionales.

REN (Rename). Realiza la decodificación de instrucciones y el proceso de mapeo entre las direcciones de los registros empleadas por las instrucciones y las direcciones físicas en la pila de registros.

REG (Register Read). Realiza la transferencia de operandos a las unidades funcionales. Existen varios multiplexores para seleccionar los operandos de las instrucciones de entre los valores de los registros, valores provenientes de etapas posteriores (DET, WRB y EXE) (forwarding) y valores provenientes de instrucciones de carga.

EXE (Execution). Etapa de ejecución.

DET (Detection). Es la última etapa donde se realiza la detección de excepciones, para el final de esta fase todas las potenciales excepciones deben ser conocidas para evitar la modificación del estado del procesador.

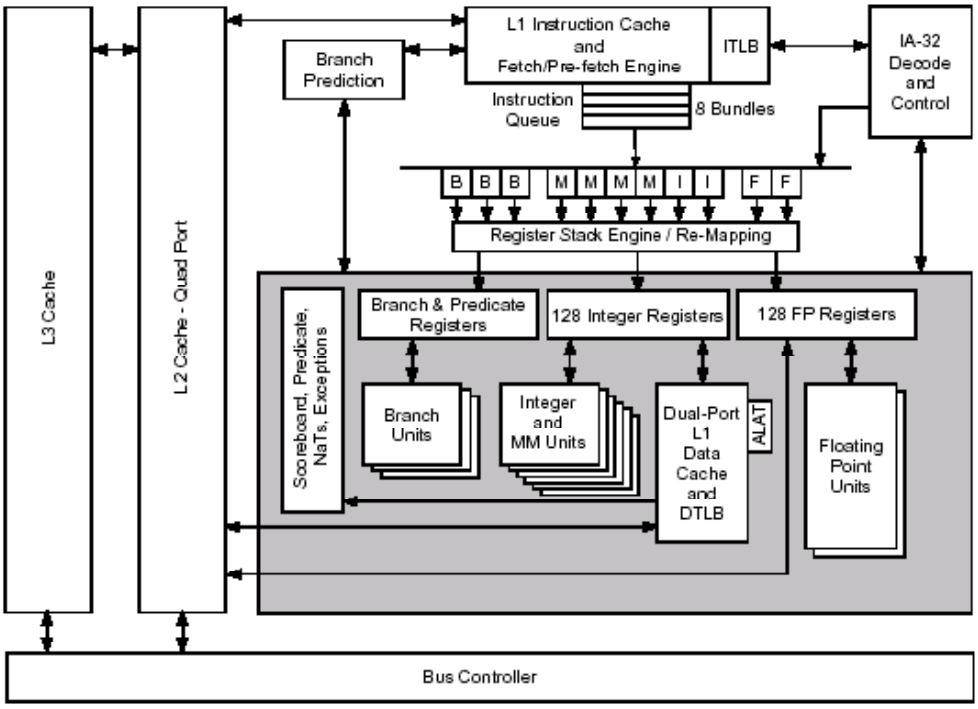

Figura 7: Diagrama a bloques del procesador Itanium 2.

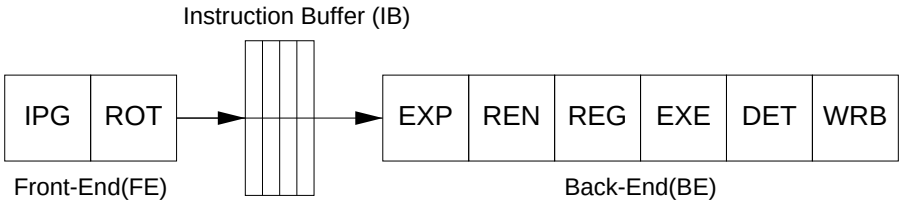

Figura 8: Pipeline del procesador Itanium 2.

WRB (Write Back). Escribe el resultado de las operaciones a los archivos de registros. Una vez concluida esta etapa se puede garantizar que la instrucción ha modificado el estado del procesador.

4 Conclusiones

Este reporte presentó una introducción al enfoque EPIC, una alternativa para el diseño de arquitecturas de conjuntos de instrucciones que exploten el paralelismo a nivel de instrucciones (ILP) mediante una combinación de técnicas de optimización en los compiladores y técnicas de implantación de arquitecturas. EPIC es una evolcución del esquema VLIW que considera factores dinámicos, conocidos únicamente en tiempo de ejecución, que afectan el rendimiento e implanta los métodos para resolverlos, los cuales son comunes en procesadores superescalares.

Se presentó una descripción de las características de la arquitectura IA-64 tales como el extenso conjunto de registros, la forma de codificar instrucciones en paquetes, las técnicas de especulación, ejecución condicional e implantación de la pila de registros. IA-64 pretende ser una opción para el cómputo de muy alto rendimiento que puede ser utilizada en una amplia gama de aplicaciones.

Finalmente se presentaron algunas características importantes sobre el procesador Itanium 2, la segunda implantación de la arquitectura IA-64. Sólo resta esperar y observar la forma en que estas nuevas propuestas afectan el desarrollo y evolución del cómputo de alto rendimiento. Esperemos que en el futuro los sistemas basados en IA-64, ya sea de cómputo personal, multiprocesadores o dispositivos empotrados, estén al alcance de todos nosotros y que el interés por los sistemas de hardware y software para cómputo de alto rendimiento (procesadores vectoriales, supercómputo, compiladores, sistemas operativos, etc.) se difunda entre la comunidad industrial y académica.

Referencias

[1] Dulong, C. "The IA-64 Architecture at Work." *IEEE Computer* jul. 1998: 24–32.

[2] Intel Corp. *Intel Itanium 2 Processor Reference Manual*. Documento: 251110-001. 2002.

[3] Intel Corp. *Intel Itanium Architecture Software Developer's Manual. Vol. 1: Application Architecture*. Documento: 245317-003. 2001.

[4] Intel Corp. *Intel Itanium Architecture Software Developer's Manual. Vol. 3: Instruction Set Reference*. Documento: 245319-003. 2001.

[5] Schlansker, M. S. y Rau, B. R. *EPIC: An Architecture for Instruction Set Level Processors*. Hewlett-Packard Laboratories. Reporte Técnico: HPL-1999-111. feb. 2000.

[6] Schlansker, M. S. y Rau, B. R. "EPIC: Explicitly Parallel Instruction Computing." *IEEE Computer* feb. 2000: 37–45.

[7] Šilc, J., B. Robič y T. Ungerer. *Processor Architecture*. Berlín: Springer-Verlag, 1999.