O Processador Itanium 2

Jurandy Gomes de Almeida Junior
Instituto de Computação
Universidade Estadual de Campinas
ra041470@ic.unicamp.br

RESUMO
A Intel e a Hewlett-Packard, atendendo as necessidades do mercado, desenvolveram o processador Itanium 2. Esse processador baseia-se na arquitetura de 64 bits IA-64, que é capaz de alcançar altos níveis de paralelismo através da tecnologia EPIC. Esse trabalho descreve as diferentes tecnologias e características de implementação adotadas nessa arquitetura, oferecendo uma visão geral do funcionamento desse processador.

PALAVRAS-CHAVE
Arquitetura de Computadores, Processadores, Itanium 2

1. INTRODUÇÃO
Em 8 de Julho de 2002, a fim de suprir as necessidades do mercado de servidores de alto desempenho, a Intel e a Hewlett-Packard anunciaram o lançamento do processador Itanium 2, a segunda implementação da linha Itanium. Embora similar ao seu antecessor [7], esse processador apresenta avanços significativos em desempenho. Esses avanços resultam de diversas melhorias na freqüência, no *pipeline* e na *cache*. Dessa forma, sua arquitetura permite ao processador atender com eficácia uma grande variedade de necessidades computacionais.

2. VISÃO GERAL
Muitas das características do processador Itanium 2 (Figura 1) são frutos das oportunidades associadas a sua arquitetura, denominada de IA-64 [4]. Além de definir operações e registradores de 64 bits, essa arquitetura implementa esquemas flexíveis de gerenciamento de memória e diversas ferramentas que os compiladores podem utilizar para aumentar o desempenho. Ela permite a execução paralela das instruções sem recorrer a complexos projetos de reordenação de *pipeline*, apenas através da indicação explícita de quais intruções podem ser executadas em paralelo sem causar conflitos nos dados. Para isso, três instruções são agrupadas estaticamente em pacotes de 16 bytes. Múltiplos pacotes de instruções podem ser executados em paralelo, ou paradas explícitas podem interromper a execução paralela para evitar inconsistências. Cada pacote codifica quais os recursos serão exigidos durante a sua execução: inteiro (I), memória (M), ponto flutuante (F) e desvios (B). Assim, as operações de memória, de ponto flutuante e de desvios que podem executar em paralelo constituem um pacote do tipo MFB. Os desenvolvedores do processador Itanium 2 examinaram as vantagens desse paralelismo explícito para projetar um *pipeline* de seis intruções . A simplicidade desse sistema permitiu ao grupo de projetistas focar recursos no desempenho

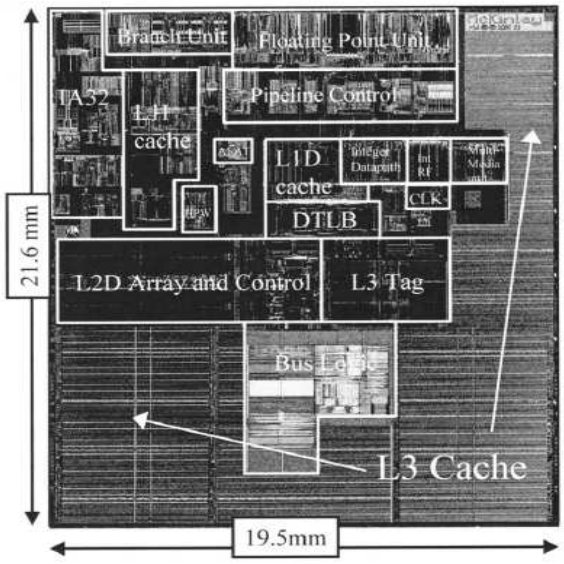

Figura 1: O processador Itanium 2

do subsistema de memória e explorar as diversas oportunidades de desempenho associadas a sua arquitetura.

3. REGISTRADORES
A arquitetura do Itanium 2, de acordo com a Intel [2], possui um amplo conjunto de registradores:

- 128 registradores de uso geral de 64 bits (r0-r127), utilizados para operações sobre inteiros e operações multimídia. Esses registradores possuem um bit adicional, o NaT (*Not a Thing*) que indica a validade do valor armazenado. A execução de instruções especulativas pode fazer com que o NaT seja setado. O registrador r0 só pode ser lido e sempre retorna o valor zero.

- 128 registradores de ponto flutuante de 82 bits (f0-f127), utilizados para cálculos de ponto flutuante. Os dois primeiros, f0 e f1, só podem ser lidos e sempre contém os valores +0.0 e +1.0, respectivamente.

- 64 registradores de predicado de 1 bit (p0-p63), que determinam se as instruções em que são especificados devem ser executadas. O registrador p0 só pode ser lido e sempre retorna 1 (*true*).

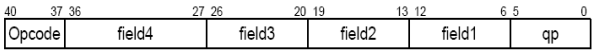

Figura 2: Formato das Instruções

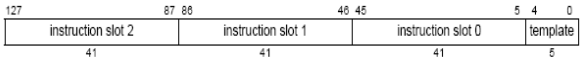

Figura 3: Pacote de Instruções

- 8 registradores de *branch* de 64 bits (b0-b7), utilizados para especificar endereços de destino de desvios indiretos.
- 128 registradores de aplicação de 64 bits (ar0-ar127). Apesar de nem todos existirem fisicamente, todos os 128 identificadores estão reservados para uso futuro.
- O *Instruction Pointer* (IP), um registrador de 64 bits que contém o endereço do pacote de instruções que está sendo executado.

4. INSTRUÇÕES

A arquitetura do Itanium 2, segundo a Intel [2], é composta por instruções de 41 bits (Figura 2). O campo **qp** (*qualifying predicate*) determina se a instrução deve ser executada. Os campos **field1** a **field4** permitem especificar até quatro operandos. O campo **Opcode** indica o *major opcode*.

A Intel [2] dividiu essas instruções em seis grandes classes:

- **A:** operações executadas na ALU.
- **I:** operações com inteiros.
- **M:** instruções de *load*/*store*.
- **B:** instruções de desvio de execução.
- **F:** operações com ponto flutuante.
- **LX:** instruções maiores que 41 bits.

Dessa forma, as unidades de execução dessa arquitetura, de acordo com a Intel [2], estão organizadas em quatro grupos, diferenciados pela classe de instrução que podem executar, como mostrado na Tabela 1:

Tabela 1: Unidades de Execução

Tipo	Unidade
A	I-unit ou M-unit
I	I-unit
M	M-unit
B	B-unit
F	F-unit
LX	I-unit ou B-unit

5. EPIC

O processador Itanium 2 lê da memória conjuntos de 3 instruções, dentro de uma palavra longa de 128 bits do tipo *little endian*, alinhados em endereços múltiplos de 16 bytes (Figura 3). Essa técnica, denominada *Explicitly Parallel Instruction Computing* (EPIC) [6], consegue combinar as vantagens das técnicas RISC (*Reduced Instruction Set Computer*) e VLIW (*Very Long Instruction Word*), permitindo a execução paralela das instruções sem a necessidade da reordenação do *pipeline*, alcançando altos níveis de IPC (*Instructions Per Cycle*).

O compilador, segundo a Intel [2], deve acrescentar na palavra de 128 bits, alguns bits de controle que indiquem ao processador quais instruções poderão ser executadas em paralelo, eliminando a necessidade de analisar o código, em tempo de execução, à procura de possíveis paralelismos. O paralelismo é determinado pelo compilador e indicado explicitamente na própria palavra de instruções.

Dessa forma, o compilador deve suportar uma técnica chamada *predication* (predição), através da qual cada desvio encontrado no código causa a execução em paralelo de cada possível alvo desse desvio, até que o resultado válido seja determinado, descartando, então, os resultados inválidos, aumentando o grau de paralelismo.

Igualmente cabe ao compilador procurar instruções de *load* pelo código e executar instruções de acesso à memória e de checagem de forma especulativa. Assim, o acesso à memória é feito antes que o programa realmente necessite dos dados correspondentes. A instrução de checagem verifica, então, a validade dos dados antes que eles sejam utilizados. Isso permite, além de aumentar o paralelismo, minimizar os efeitos dos períodos de latência inerentes ao sistema de memória.

Deixar essas tarefas para o compilador ao invés de utilizar um componente específico do processador garante diversas vantagens. Primeiro, o compilador disponibiliza de um tempo maior para examinar o código, o que não ocorre em um processador, devido à rapidez com que necessita executar as instruções. Portanto, os resultados obtidos através do compilador podem ser mais eficazes. Além disso, os circuitos de predição são complexos e, dessa forma, a predição em tempo de compilação reduz a complexidade de execução das instruções.

Entretanto, o comportamento de um programa em tempo de execução nem sempre é óbvio em relação ao código obtido, podendo variar consideravelmente dependendo dos dados reais que estão sendo processados. Assim, o fluxo de execução suposto pelo compilador pode não ser o mesmo seguido pelo processador devido à decisões tomadas com base nos dados obtidos em tempo de execução. Isso significa que o compilador pode cometer um erro de predição com mais freqüência que o processador. Por isso, essa tecnologia confia no desempenho dos compiladores, diminuindo a complexidade do *hardware* ao custo do aumento da complexidade dos *softwares*.

6. PREDIÇÃO

Nas maioria das arquiteturas, as declarações do tipo IF-THEN-ELSE são convertidas em pequenos blocos de código organizados seqüêncialmente e separados por instruções de desvio. Quando o processador encontra um desvio, ele tenta

adivinhar qual dos blocos será o alvo resultante. Se ele errar, haverá uma grande penalidade decorrente da necessidade de esvaziar o *pipeline* e iniciar a execução do bloco correto. Claramente este tipo de abordagem dificulta a obtenção de maiores níveis de paralelismo.

Na arquitetura do Itanium 2, o compilador deve ser capaz de analisar cada desvio e determinar se ele é candidato ou não a uma predição. Se o compilador determinar que a predição pode ser utilizada nesse desvio, ele irá rotular cada um dos caminhos com um identificador de predicado.

As instruções apresentam um campo de 6 bits (qp) para indicar o rótulo do predicado. Assim, a cada instante, 64 predicados diferentes podem ser utilizados, sendo que todas as instruções do mesmo ramo de um desvio devem apresentar o mesmo predicado.

Dessa forma, o compilador pode verificar quais instruções poderão ser executadas em paralelo, agrupando-as em pacotes de 128 bits. Esse pacote deve conter bits de controle que indiquem ao processador quais instruções poderão ser executadas independentemente e quais instruções são dependentes entre si.

As instruções em cada pacote não necessitam estar em sua ordem original e podem representar caminhos inteiramente independentes dos desvios. Além disso, instruções dependentes podem ser intercaladas com instruções independentes. Em tempo de execução, o processador pode despachar as instruções independentes para as unidades funcionais disponíveis e agendar a execução das instruções dependentes.

Ao encontrar um predicado, o processador não tenta adivinhar qual dos caminhos deve ser tomado. Todos os caminhos possíveis do desvio são executados. Desta forma, não existem desvios em nível de execução. No decorrer do fluxo de execução, o processador será capaz de determinar qual dos caminhos do desvio deve ser seguido.

Nesse ponto, instruções referentes a ambos os caminhos podem ter sido executadas. Entretanto, o processador ainda não armazenou os resultados obtidos. Esse processo é executado após seu registrador de predicado ser verificado. Se esse registrador estiver setado, os resultados serão armazenados, caso contrário, serão descartados.

7. ESPECULAÇÃO

O Itanium 2 pode, além de realizar operações de busca na memória antes que os dados sejam necessários, postergar a ocorrência de uma exceção se o resultado obtido nesse processo for incorreto. A idéia é separar a busca dos dados na memória do seu uso efetivo pelo processador.

Inicialmente, o compilador procura por instruções no programa que possam necessitar de dados na memória, inserindo, sempre que possível, uma instrução de busca especulativa em um ponto anterior do código e uma instrução de verificação imediatamente antes da instrução que irá utilizar esses dados.

Em tempo de execução, ao encontrar uma instrução especulativa, o processador tenta buscar dados na memória. Esses dados podem ser invalidados caso constituam um bloco de código seguinte a uma instrução de desvio. Nesse caso, a maioria dos processadores lançaria uma exceção invalidando os dados carregados.

O processador Itanium 2 posterga o lançamento da exceção até que uma instrução de verificação seja executada. Nesse momento o processador pode analisar o desvio que causou essas inconsistências. Se esse desvio foi tomado, os dados devem ser descartados e a exceção deve ser lançada, caso contrário, os dados serão validados.

Uma forma adicional de especulação é a especulação de endereços, que permite agendar, em uma determinada posição de memória, a ocorrência de um *load* antes da ocorrência de um *store*. Todos *loads* ocorridos são mantidos em uma tabela chamada *Advanced Load Address Table* (ALAT).

Se um *store* entra em conflito com um *load* dessa tabela, sua entrada correspondente é invalidada. Se um *load* não apresenta uma entrada correspondente nessa tabela, os dados são carregados diretamente da memória, adicionando uma nova entrada nessa tabela. Caso contrário, os dados são carregados a partir dos valores armazenados em sua entrada correspondente.

Outra técnica implementada é a *Loop Unrolling and Rotation*, que multiplica o código de um laço de repetição, evitando um grande número de desvios. Este processo, denominado de *unrolling*, é suportado por diversas arquiteturas RISC convencionais.

A arquitetura do Itanium 2 parece diferenciar-se no uso de uma técnica chamada *Register Rotation*, que permite, sob o controle do compilador, rotacionar os registradores, incrementando seus valores em cada passo da rotação.

Dessa forma, mesmo que o compilador realize um *unrolling*, os registradores continuam a ser incrementados. Se o número de repetições desejado for atingido, dois contadores dedicados, denominados *loop count* e *epilog count*, retornam os predicados em zero, finalizando esse laço.

8. UNIDADES FUNCIONAIS

O processador Itanium 2 baseia-se em uma arquitetura de 64 bits (Figura 4), denominada IA-64 [4]. Seu funcionamento pode ser dividido em cinco grupos básicos:

- **Processamento da Instrução:** contém a lógica para o *prefetch* e o *fetch* da instrução, predição de desvios, desmembramento de *buffers* (*decoupling buffers*) e a pilha de registradores (*register stack*).
- **Execução:** contém a lógica para as unidades de inteiros, ponto flutuante, multimídia, desvios, além dos bancos de registradores.
- **Controle:** trata de exceções e do *pipeline*.
- **Subsistema de Memória:** contém a *cache* de instruções L1 (L1I), a *cache* de dados L1 (L1D), os *caches* unificados L2 e L3, o *Programmable Interrupt Controller* (PIC), a *Advanced Load Address Table* (ALAT) e os barramentos.

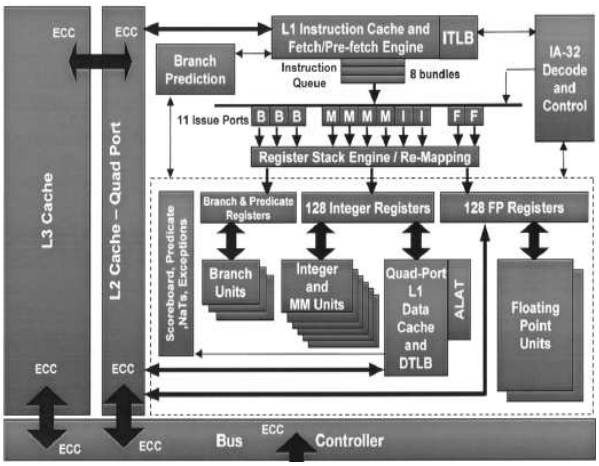

Figura 4: A arquitetura do Itanium 2

- **Execução IA-32:** contém o *hardware* para tratar instruções IA-32.

8.1 PROCESSAMENTO DA INSTRUÇÃO

A primeira etapa do *datapath* do processador Itanium 2 consiste no processamento da instrução a ser executada. Atuando em conjunto com uma sofisticada predição de desvios e *hardware* de correção, o Itanium 2 especulativamente carrega (*fetch*) as instruções da *cache* de instruções L1 (L1I) em um *buffer* [5]. Esse *buffer* produz um *front end* para que, de forma especulativa, o carregamento prossiga, diminuindo a latência da *cache* de instruções e da predição de desvios.

Uma hierarquia de estruturas de predição de desvios visa garantir predições com alta precisão e poucas falhas. O *hardware* de predição de desvios é ajudado pelas diretrizes dos *branch hints* fornecidos pelo compilador. As diretrizes fornecem o endereço do alvo do desvio, *hints* estáticos nos direcionamentos dos desvios, bem como a indicação de quando utilizar a predição dinâmica. Tais diretrizes são programadas dentro das estruturas de predição de desvios e usadas em conjunto com os esquemas de predição dinâmicas.

O conjunto de 4 *caches* de instruções de 16KB preenche todo o *pipeline* e pode prover 32B de código (dois pacotes de instruções ou seis instruções) a cada ciclo de *clock* [5]. Isso é suportado em um monociclo por uma TLB (*Translation Lookaside Buffer*) de instruções com 64 entradas que é totalmente associativa. O código carregado é inserido no *buffer*, que pode suportar oito pacotes de instruções. As instruções lidas desse *buffer* são enviadas para a lógica de lançamento e renomeação de instruções, com base na disponibilidade dos recursos de execução.

8.2 EXECUÇÃO

O Itanium 2, segundo a Intel [1], possui as seguintes unidades de execução:

- quatro unidades para inteiros
- quatro unidades para multimídia
- quatro unidades para ponto flutuante
- três unidades para desvio
- duas unidades para *load/store*

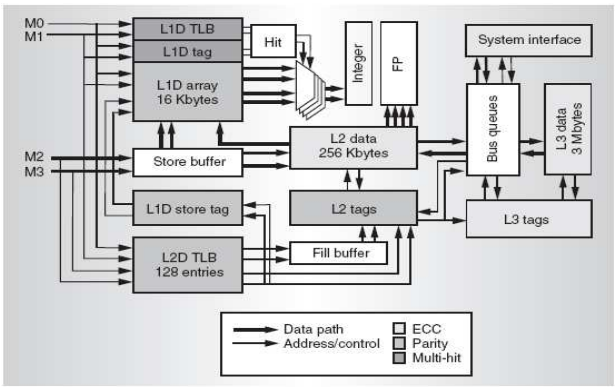

Figura 5: O Subsitema de Memória

8.3 CONTROLE

O grupo de controle do Itanium 2 foi desenvolvido para o tratamento de exceções e controle do *pipeline*. O tratamento de exceções implementa a priorização de exceções.

O controle do *pipeline* usa uma tabela para detectar dependências de registradores fonte e um suporte especial para especulação de dados e de controle.

Na especulação de controle o *hadware* gerencia a criação e propagação de *tokens* de exceções (NaT). Na especulação de dados o processador fornece uma tabela de endereços de *loads* antecipados (ALAT).

8.4 MEMÓRIA

O subsistema de memória do Itanium 2 (Figura 5), segundo a Intel [2], é composto por um *cache* de instruções L1 (L1I), um *cache* de dados L1 (L1D), um *cache* unificado L2, um *cache* unificado L3, um *Programmable Interrupt Controller* (PIC), uma *Advanced Load Address Table* (ALAT) e pela lógica do barramento do sistema e do barramento *backside*.

O *cache* de instrução L1 possui tamanho de 16KB e é organizado de forma *4-way set-associative* possuindo linhas de tamanho de 64B. O L1I preenche todo o *pipeline* e pode entregar uma linha de 64B contendo dois pacotes de instruções (seis instruções) em cada ciclo de *clock*.

O *cache* de dados L1 possui duas portas e tem 16 KB de tamanho, sendo organizado de forma *4-way set-associative* com linhas de 64B e latência de 1 ciclo por carga. Ele pode suportar dois *stores* e *loads* concorrentes e somente fornece dados para a unidade de inteiros.

O cache unificado L2 possui tamanho de 256KB, tem duas pseudo-portas, suporta acesso concorrente e é organizado de forma *8-way set-associative* com linhas de 128B. Uma característica importante do cache L2 é que ele utiza a política de alocação de escrita (*write-allocate*) para realizar as reescritas (*write back*).

O *cache* unificado L3 pode ter tamanhos de 1.5 a 9MB sendo organizado de forma *12-way set-associative* com linhas de 128B. Ele é acessado somente via barramento *backside* de 128 bits.

A estrutura *Advanced Load Address Table* (ALAT) é utilizada para prover especulação de dados a arquitetura. A ALAT mantém informações sobre *loads* especulativos de dados feitos pelo processador e *stores* que utilizem os mesmos endereços que esses *loads*. A estrutura física da ALAT possui 32 entradas e é totalmente associativo.

O Itanium 2 é composto por quatro unidades *Translation Lookaside Buffer* (TLB): os TLBs de dados de primeiro e segundo nível (L1 DTLB e L2 DTLB, respectivamente), e os TLBs de instruções de primeiro e segundo nível (L1 ITLB e L2 ITLB, respectivamente). A ocorrência de um *miss* em qualquer um desses quatro TLBs é servida pela tabela de paginação do *hardware*, que suporta o formato da *Virtual Hash Page Table* (VHPT) de 8B e 32B.

O TLB de dados (DTLB) possui uma hierarquia de dois níveis, L1 DTLB e L2 DTLB, que são não-inclusivos. O L1 DTLB possui 32 entradas, é totalmente associativo e mantém cópias em *cache* do L2 DTLB. Já o L2 DTLB possui 128 entradas, é totalmente associativo e mantém todos os tamanhos de página e entradas do tipo *Translation Register* (TR) de dados e *Translation Cache* (TC) de dados.

Analogamente, o TLB de instrução (ITLB) também possui uma hierarquia de dois níveis, L1 ITLB e L2 ITLB, que são não-inclusivos. O L1 ITLB possui 32 entradas, é totalmente associativo e mantém cópias em *cache* do L2 ITLB. O L2 ITLB possui 128 entradas, é totalmente associativo e mantém todos os tamanhos de página e entradas TR e TC de instruções.

8.5 EXECUÇÃO IA-32

O processador Itanium 2, de acordo com a Intel [2], suporta binários com instruções IA-32 via *hardware*. A unidade de IA-32 (Figura 6) foi desenvolvida para fazer o uso de registradores, *caches* e recursos de execução da máquina EPIC. Para prover alto desempenho, a unidade de execução IA-32 sincroniza as instruções dinamicamente.

Entretanto, uma vez que esse processador foi desenvolvido para obter desempenho em intruções EPIC, a execução de um código da modalidade IA-32 nos processadores da linha Itanium apresenta uma severa queda de desempenho em relação à um mesmo código da modalidade IA-64 e em relação a sua execução nos processadores da linha Pentium.

9. PIPELINE

Na arquitetura do Itanium 2, a execução de uma instrução é realizada em várias etapas, correspondentes aos diversos estágios do *pipeline*. A Tabela 2 descreve os oito estágios nos quais a Intel [1] organizou o *pipeline* dessa arquitetura.

Esses estágios estão agrupados em duas fases maiores (Figura 7): *front end*, composto pelos estágios IPG e ROT; e *back end*, composto pelos estágios EXP, REN, REG, EXE, DET e WRB.

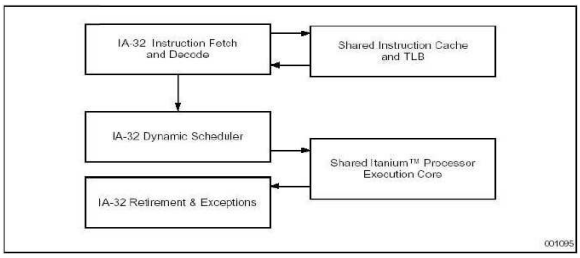

Figura 6: A Unidade IA-32

Tabela 2: Tarefas realizadas em cada estágio

Mnemônico	Descrição das Atividades
IPG	Geração do apontador de instrução.
ROT	Rotaciona as instruções do grupo corrente de forma que o pacote 0 contenha a primeira instrução que deve ser executada.
EXP	Com base no campo *template* do pacote, distribui até 6 instruções através de 11 portas para as unidades de execução.
REN	Renomeia, ou seja, remapeia os registradores das instruções para registradores físicos (feito pelo *Register Stack Engine*) e decodifica as instruções.
REG	Leitura dos registradores.
EXE	Executa as operações.
DET	Detecta exceções; abandona resultados de execução se a condição do predicado da instrução for falso; trata previsões erradas de desvio.
WRB	Efetiva a gravação dos resultados nos registradores.

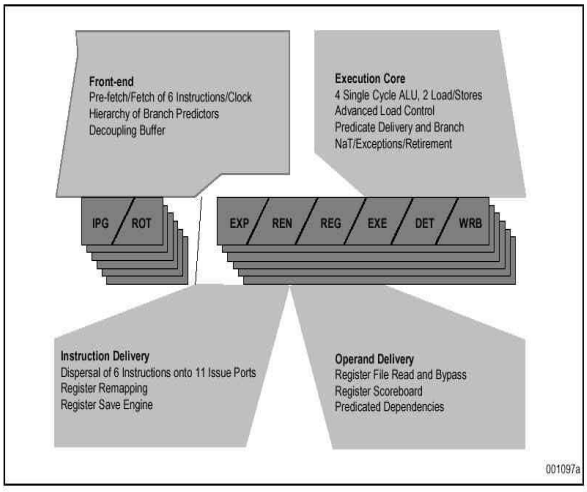

Figura 7: Estágios do *Pipeline*

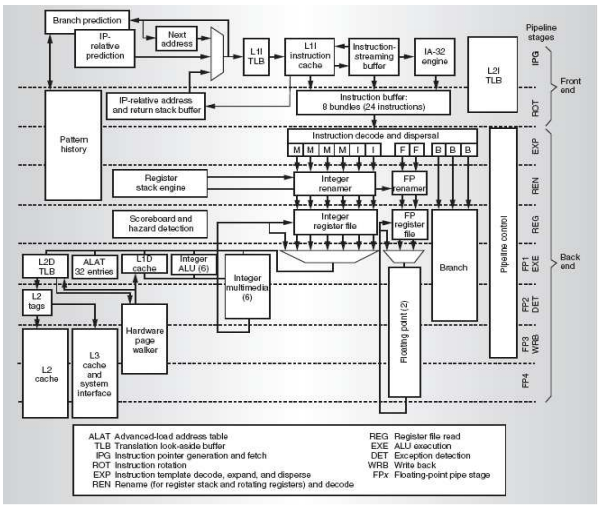

Figura 8: O *pipeline* do Itanium 2

Esse conjunto de estágios acomoda a execução de todos os tipos de instruções do Itanium 2. Circuitos especializados (Figura 8) tornam possível a execução em paralelo das unidades que manipulam os tipos de instrução: A, I, M, B e F.

O *pipeline* de ponto flutuante é constituído de dois estágios adicionais devido a maior complexidade das operações de multiplicação e adição, nas quais se baseiam muitas instruções de ponto flutuante. Dessa forma, quatro estágios de execução de operações ponto flutuante são substituídos pelos estágios EXE e DET nas operações inteiras.

Embora o Itanium 2 trate cada instrução na ordem dada pelo pacote de instruções, uma fila de espera (*buffering*) permite que as instruções se completem fora de ordem. Isso previne que instruções de inteiros tenham que esperar pelas instruções de ponto flutuante, que demoram mais tempo para executar.

Para o processador Itanium 2, uma falha na predição de um desvio, quando o *pipeline* está normalizado, dispende seis ciclos. Dessa forma, qualquer instrução que se inicie baseada em uma pressuposição errada de desvio deve ser impedida de afetar o estado da máquina.

10. EVOLUÇÃO

Diferentes versões dos processadores da linha Itanium foram desenvolvidos desde o lançamento dessa arquitetura:

- **Itanium:** O primeiro modelo da série foi apresentado em Junho de 2001. Desenvolvido com a tecnologia de 180nm, o chip funcionava com freqüências de 766 MHz e 800 MHz, com a possibilidade de escolha de 2 ou 4 MB de cache L3.
- **McKinley:** Foi a primeira versão do Itanium 2, apresentado em Julho de 2002. Implementado em um processo de 180nm, além das freqüências de 900 MHz e 1 GHz e das caches de 1.5 MB e 3 MB, esse modelo possuia suporte de hardware para as instruções de desvios longos.
- **Madison:** Foi apresentado em Junho de 2003 através de 3 implementações: 1.3 GHz com 3 MB de cache, 1.4 GHz com 4 MB de cache e 1.5 GHz com 6 MB de cache. Desenvolvido em um processo de 130nm, esse processador possui 374mm^2 de dimensão. Em Setembro de 2003, foram disponibilizadas as versões de 1.4 GHz com 1.5 MB de cache. Modelos de 1.6 GHz com 6 MB de cache e de 1.4 GHz com 3 MB de cache foram lançadas em Abril de 2004. Em Novembro de 2004, foi apresentada uma versão de 1.6 GHz com 9 MB de cache.
- **Hondo:** Foi anunciado pela Hewlett-Packard em Fevereiro de 2003 e lançado no início de 2004, sendo caracterizado por um módulo dual, conhecido como mx2 (dois processadores em um único módulo). Esse processador é composto por dois núcleos Madison de freqüências de 1.1 GHz com 4 MB de cache L3 associados através de 32 MB de cache L4.
- **Deerfield:** Lançado em Setembro de 2003, possui freqüência de 1 GHz e 1.5 MB de cache. Foi o primeiro processador da linha Itanium de baixa voltagem, utilizando 62 watts.
- **Fanwood:** Apresentado em Novembro de 2004, incluem modelos com 1.6 GHz e 3 MB de cache L3 e 200 MHz ou 266 MHz de *Front Side Bus* (FSB).

11. MODELOS ATUAIS

Os modelos atuais disponíveis para o Itanium 2, segundo a Intel [3], estão divididos em 3 categorias: Multi Processor (MP)/Madison, Dual Processor (DP)/Hondo e Low Voltage (LV)/Fanwood. Suas principais características podem ser descritas na Tabela 3.

Tabela 3: Descrição do Processador Itanium 2

Característica	MP	DP	LV
Cache L1	32KB	32KB	32KB
Cache L2	256KB	256KB	256KB
Cache L3	3, 4, 6 e 9MB	1.5 e 3MB	1.5 e 3MB
Clock (GHz)	1.30 a 1.60	1.40 a 1.60	1.00 a 1.30
Chipset	Intel E8870	Intel E8870	Intel E8870
Barramento	400 MHz	400 / 533 MHz	400 MHz
Vazão E/S	6.4 GB/sec	6.4 GB/sec	6.4 GB/sec
Consumo	130W	99W	62W
Plataforma	SR870BN4	SR870BH2	SR870BH2

Cada modelo possui determinadas características que permitem explorar um conjunto específico de aplicações. MP é indicada para o multiprocessamento de aplicações voltadas para banco de dados, aplicações inteligentes, *Enterprise Resource Planning* (ERP) e *High Performance Computing* (HPC). Já a implementação DP busca explorar uma computação mais técnica, como HPC para clusters, servidor de rede e gerenciamento de sistemas de segurança. No LV a preocupação maior está no baixo consumo e no custo do servidor, possibilitando a utilização de um sistema dual.

12. CONCLUSÕES

A arquitetura do Itanium 2 fornece novos paradigmas de integração, paralelismo, latência e eficiência, baseados no projeto e no controle de um rico conjunto de metodologias, que

permite ao processador Itanium 2 ajustar-se aos mais variados ambientes de trabalho. A equipe de projeto aproveitou as oportunidades de desempenho disponíveis na arquitetura IA-64 para produzir um processador de alto desempenho capaz de implementar poderosos e versáteis sistemas computacionais.

13. REFERÊNCIAS

[1] I. Corporation. *Intel(R) Itanium(TM) 2 Processor Reference Manual*. Intel Corporation, Junho 2002.

[2] I. Corporation. *Intel(R) Itanium(TM) Architecture Software Developer's Manual*. Volume 1–3. Intel Corporation, Outubro 2002.

[3] I. Corporation. Intel itanium 2 processor for demanding enterprise and technical applications. Technical report, Intel Corporation, 2005.

[4] J. Huck, D. Morris, J. Ross, A. Knies, H. Mulder, and R. Zahir. Introducing the ia-64 architecture. *IEEE Micro*, 20(5):12–23, 2000.

[5] C. McNairy and D. Soltis. Itanium 2 processor microarchitecture. *IEEE Micro*, 23(2):44–55, 2003.

[6] M. S. Schlansker and B. R. Rau. Epic: Explicitly parallel instruction computing. *Computer*, 33(2):37–45, 2000.

[7] H. Sharangpani and K. Arora. Itanium processor microarchitecture. *IEEE Micro*, 20(5):24–43, 2000.