

Universidade Federal do Rio de Janeiro
Pós-Graduação em Informática

**Microarquiteturas de Alto
Desempenho**

**Arquiteturas
VLIW**

Gabriel P. Silva

Introdução

- **A arquitetura Very Long Instruction Word (VLIW) tenta alcançar maiores níveis de paralelismo de instrução pela execução de instruções longas compostas por múltiplas operações.**
- **As palavras de instruções longas consistem de várias operações aritméticas, lógicas e de controle, onde cada uma delas poderia ser uma operação individual em um processador RISC comum.**
- **O processador VLIW executa o conjunto de operações concorrentemente, alcançando assim um alto grau de paralelismo no nível de instrução.**
- **É responsabilidade do compilador escalonar as operações de modo a utilizar o melhor possível as unidades funcionais disponíveis no processador.**
- **Um dos maiores obstáculos à evolução das arquiteturas VLIW tem sido a falta de compatibilidade binária com as arquiteturas convencionais.**

Histórico

- **Mesmo antes do advento das primeiras máquinas VLIW, havia diversos processadores e dispositivos computacionais que utilizavam uma instrução longa para controlar o funcionamento de diversas unidade funcionais em paralelo.**
- **Contudo, essas máquinas eram normalmente programadas manualmente e o código utilizado para essas máquinas não podia ser generalizado para outras arquiteturas, porque os compiladores daquela época só exploravam o paralelismo dentro dos limites dos blocos básicos.**
- **Joseph A. Fisher, um pioneiro da VLIW, desenvolveu uma técnica global de compactação de microcódigo, chamada de “trace scheduling”, que poderia ser utilizada em compiladores para gerar código para arquiteturas do tipo VLIW a partir de código seqüencial.**
- **Suas descobertas levaram ao desenvolvimento do processador ELI-512 e ao compilador “trace scheduling” Bulldog.**

Histórico

- **Duas companhias foram fundadas em 1984 para construir computadores com tecnologia VLIW: Multiflow e Cydrome.**
- **A Multiflow foi iniciada por Fisher e seus colegas da Universidade de Yale.**
- **A Cydrome foi fundada por Bob Rau, que foi um outro pioneiro da VLIW e seus colegas.**
- **Em 1987 a Cydrome lançou o seu primeiro processador comercial, o Cydra 5, com uma palavra de 256 bits e incluía suporte em hardware para a técnica de software pipeline.**
- **No mesmo ano a Multiflow lançou a Trace/200, com uma palavra de 256 bits, para um despacho de até 7 operações por ciclo.**
- **As primeiras máquinas VLIW foram um fracasso comercial, o que levou ao fechamento da Cydrome em 1998 e da Multiflow em 1990.**

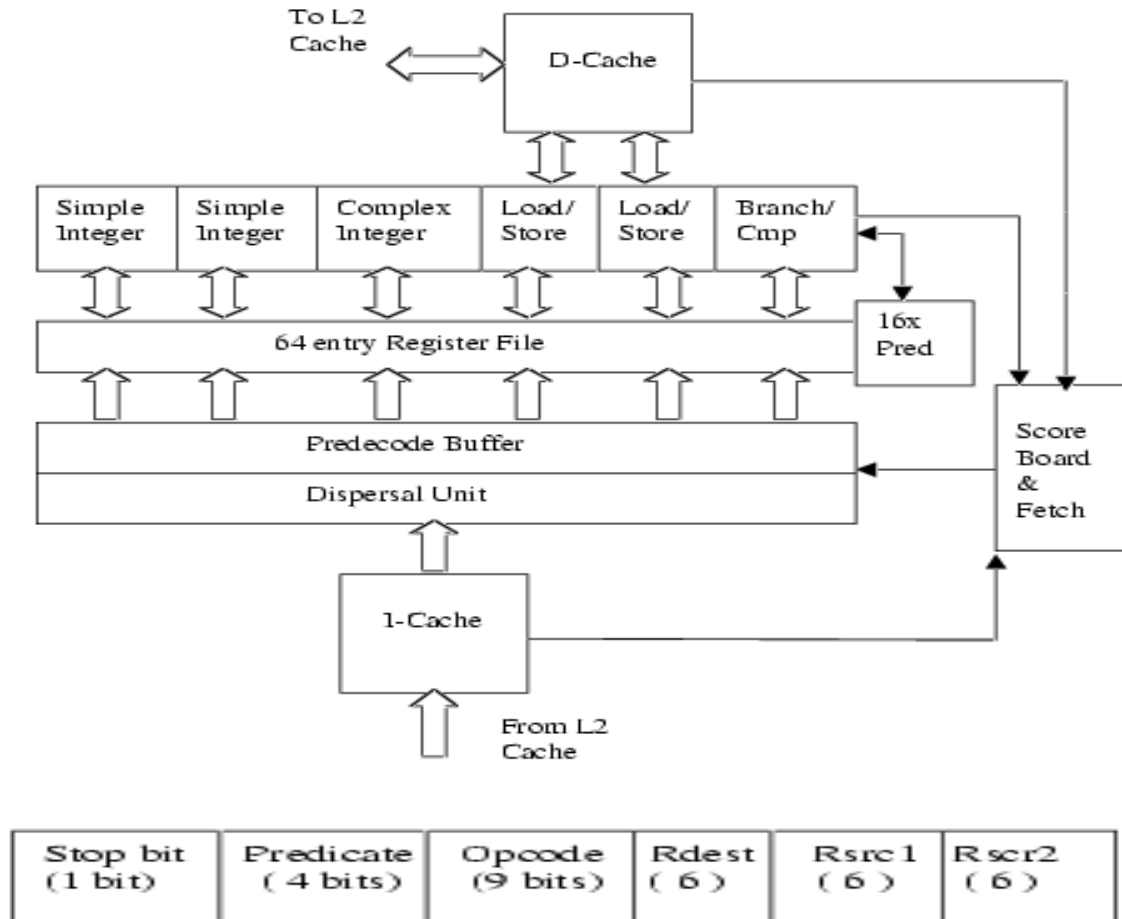
Histórico

- **Comerciais:**
 - **Processador IA-64 ou Itanium da Intel**
 - **Processador Crusoe da Transmeta**
 - **Processador Trimedia da Philips**
 - **Processador TMS320C62x DSPs da Texas Instruments**
- **Experimentais:**
 - **Processador Playdoh dos Laboratórios HP**
 - **Processador Tinker da North Carolina State University**
 - **Processador de imagens Imagine em desenvolvimento na Universidade de Stanford.**

Características

- Cada instrução longa é formada por um conjunto de operações que podem ser executadas em paralelo.
- As instruções longas são montadas através de técnicas de escalonamento por “software” aplicadas em tempo de compilação ou através da compactação do código gerado por um compilador convencional.
- Grau de concorrência das operações situa-se na prática entre 2 e 4.
- Código objeto não é compatível com o de um processador com arquitetura convencional.
- Arquiteturas organizadas com múltiplas unidades funcionais e banco de registradores com múltiplas portas de leitura
- As arquiteturas VLIW apresentam menor complexidade do que as arquiteturas superescalares baseadas em escalonamento dinâmico (por “hardware”) de instruções.

Exemplo de Arquitetura



Arquitetura do Processador VLIW Defoe

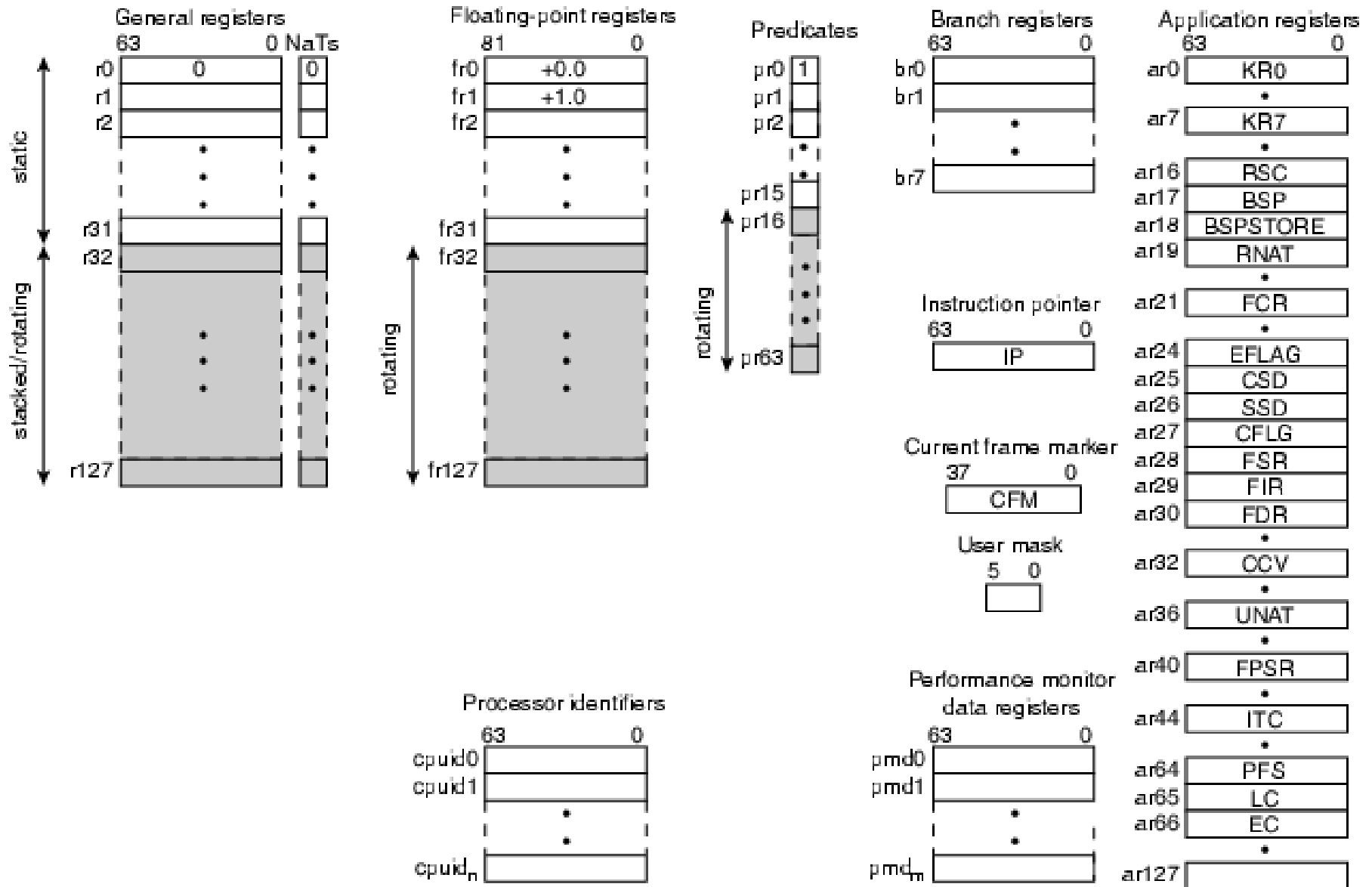
Itanium IA-64

Intel

Itanium IA-64

- O processador Itanium é a primeira implementação da arquitetura IA-64 da Intel.
- A IA-64 é uma ISA no estilo VLIW denominado EPIC (Explicitly Parallel Instruction Computing) pela Intel e HP.
- Na sua primeira implementação, este processador executava a 800 MHz, possuindo um pipeline com 10 estágios.
- É uma arquitetura VLIW de 64 bits, capaz de executar até 6 operações/ciclo, possuindo 4 unid. inteiras, 4 multimedia, 2 de load/store, 2 de ponto flutuante com precisão estendida e 2 de ponto flutuante com precisão simples.
- Cada palavra VLIW consiste de um ou mais pacotes de 128 bits. Cada pacote de 128 bits consiste de 3 operações e um template. O template serve para indicar quais instruções podem ser executadas em paralelo.
- Duas instruções são buscadas a cada ciclo, totalizando até 6 operações buscadas, despachadas e executadas por ciclo.

Conjunto de Registradores do IA-64



Registradores do IA-64 (1)

- **Registradores de Uso Geral**
 - 128 registradores de 64 bits de uso geral.
 - r0-r31 estáticos
 - r32-r127 podem ser usados como registradores de rotação para o “software pipeline” ou para “register stack”.
 - Referências são virtuais
 - O hardware pode renomear automaticamente.
- **Registradores de Ponto Flutuante**
 - 128 registradores de 82 bits de ponto flutuante
 - Valores IEEE 754 armazenados em formato duplo estendido.
 - Registradores fr0-fr31 estáticos, fr32-fr127 podem ser rotacionados para o “software pipeline”

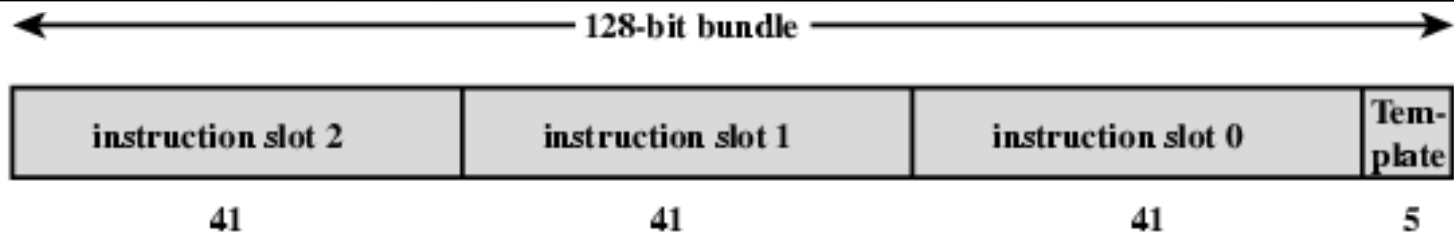
Registradores do IA-64 (2)

- **Registradores de Predicado**
 - 64 registradores de 1 bit usados como predicados.
 - pr0 é sempre 1 para permitir instruções sem predicado.
 - pr1-pr15 são estáticos, pr16-pr63 podem ser rotacionados.
- **Registradores de Desvio**
 - 8 registradores de 64 bits
- **Apontador de Instruções**
 - Guarda o endereço da instrução atualmente em execução.
- **Current frame marker (CFM)**
 - Informação de “status” relativa à pilha de registradores de uso geral atual.
 - Informação de rotação para fr and pr
- **Máscara de Usuário**
- **Registradores de Aplicação**
 - Registradores de Aplicação Específica

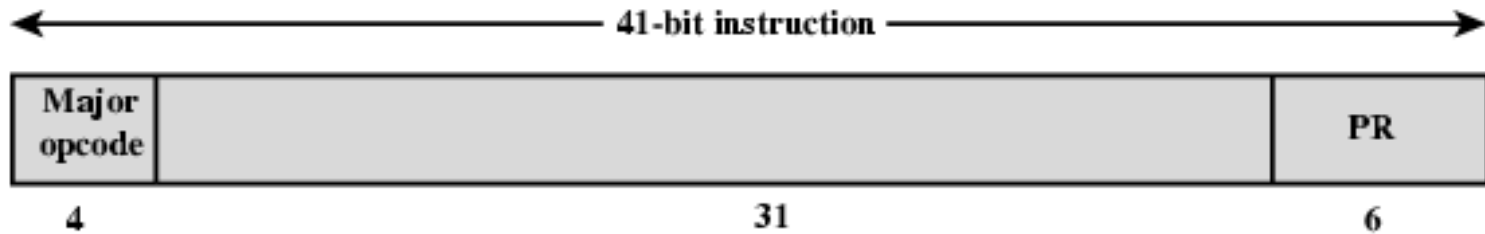
Formato das Instruções

- **Pacotes de 128 bits**
 - **Contém 3 instruções (operações) mais um template**
 - **Pode-se buscar um ou mais pacotes por vez.**
 - **O template contém informação sobre quais instruções podem ser executadas em paralelo.**
 - **Não estão limitadas a um único pacote.**
 - **e.g. um fluxo de 8 instruções pode ser executado em paralelo**
 - **Compilador reordena as instruções para formar pacotes contíguos.**
 - **Pode-se misturar instruções dependentes e independentes no mesmo pacote.**
 - **Instrução tem 41 bits de largura.**
 - **Endereça mais registradores que o RISC**
 - **Registradores de execução predicada.**

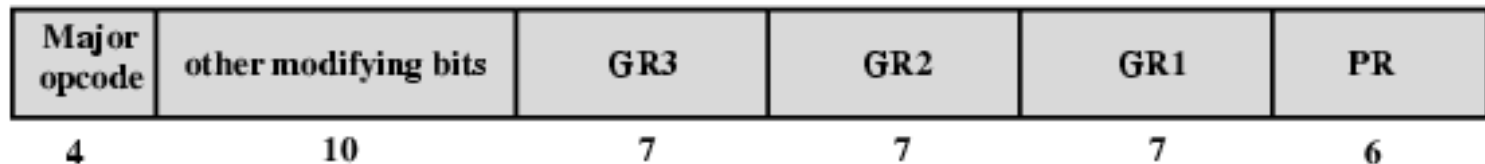
Formato das Instruções



(a) IA-64 bundle



(b) General IA-64 instruction format



(c) Typical IA-64 instruction format

PR = Predicate register

GR = General or floating-point register

Formato da Linguagem de Montagem

- `[qp] mnemonic [.comp] dest = srcs //`
- **qp** – registrador de predicado
 - 1: o resultado é aproveitado
 - 0: o resultado é descartado
- **mnemônico** – nome da instrução
- **comp** – um ou mais complementos da instrução usado para qualificar o mnemônico.
- **dest** – um ou mais operandos destino
- **srcs** – um ou mais operandos fonte
- **//** - comentário
- **Grupos de instruções e paradas** indicada pelo **template**:
 - Sequência sem RAW ou WAW.
 - Não á necessidade de verificação de dependências de dados em tempo de execução pelo mecanismo de despacho do pipeline do Itanium.

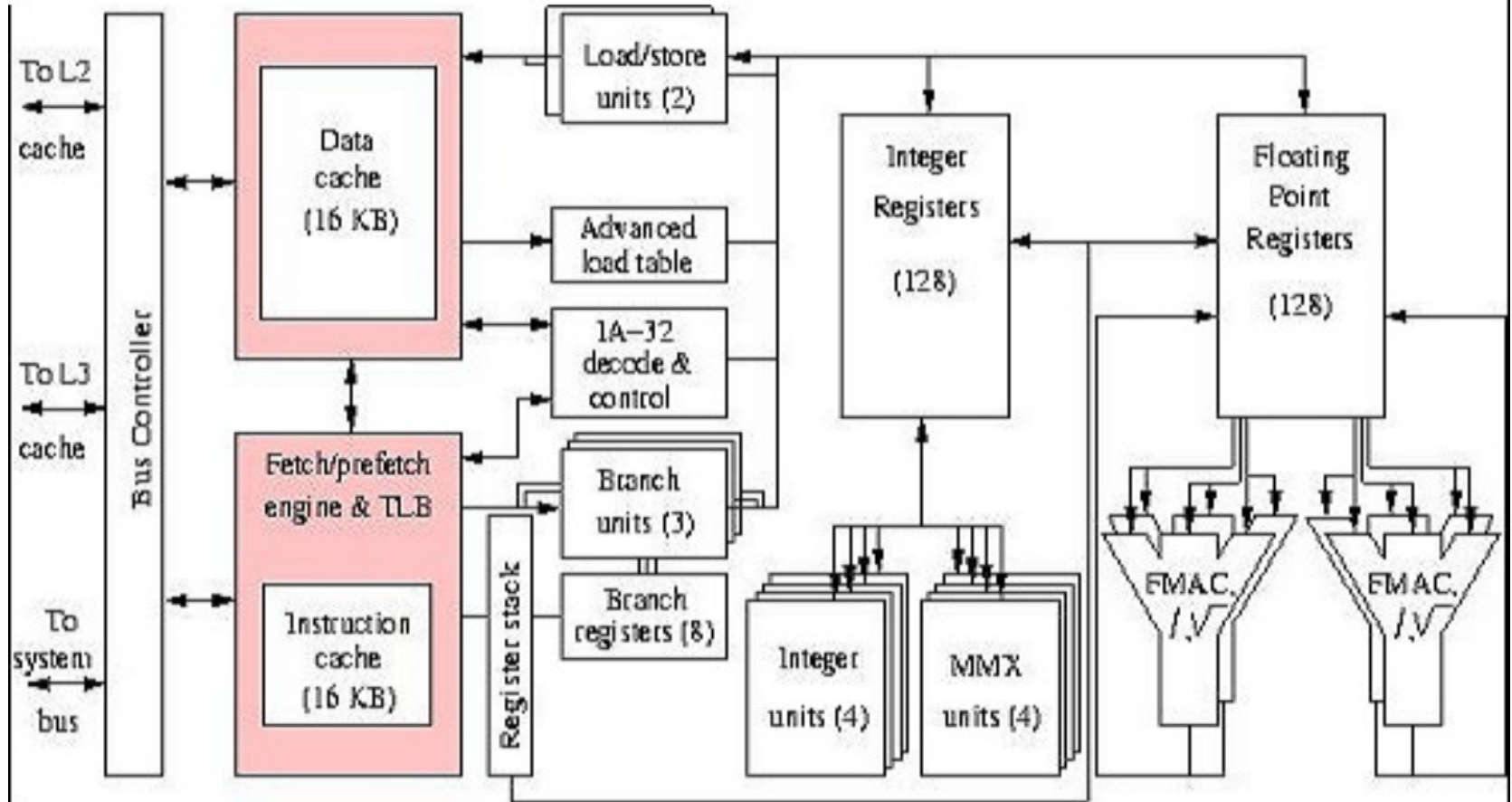
Exemplos em Linguagem de Montagem

ld8 r1 = [r5] ;; //first group

add r3 = r1, r4 //second group

- **Segunda instrução depende do valor de r1**
 - **Modificado pela primeira instrução**
 - **Não podem estar no mesmo grupo para execução paralela**

Itanium IA-64



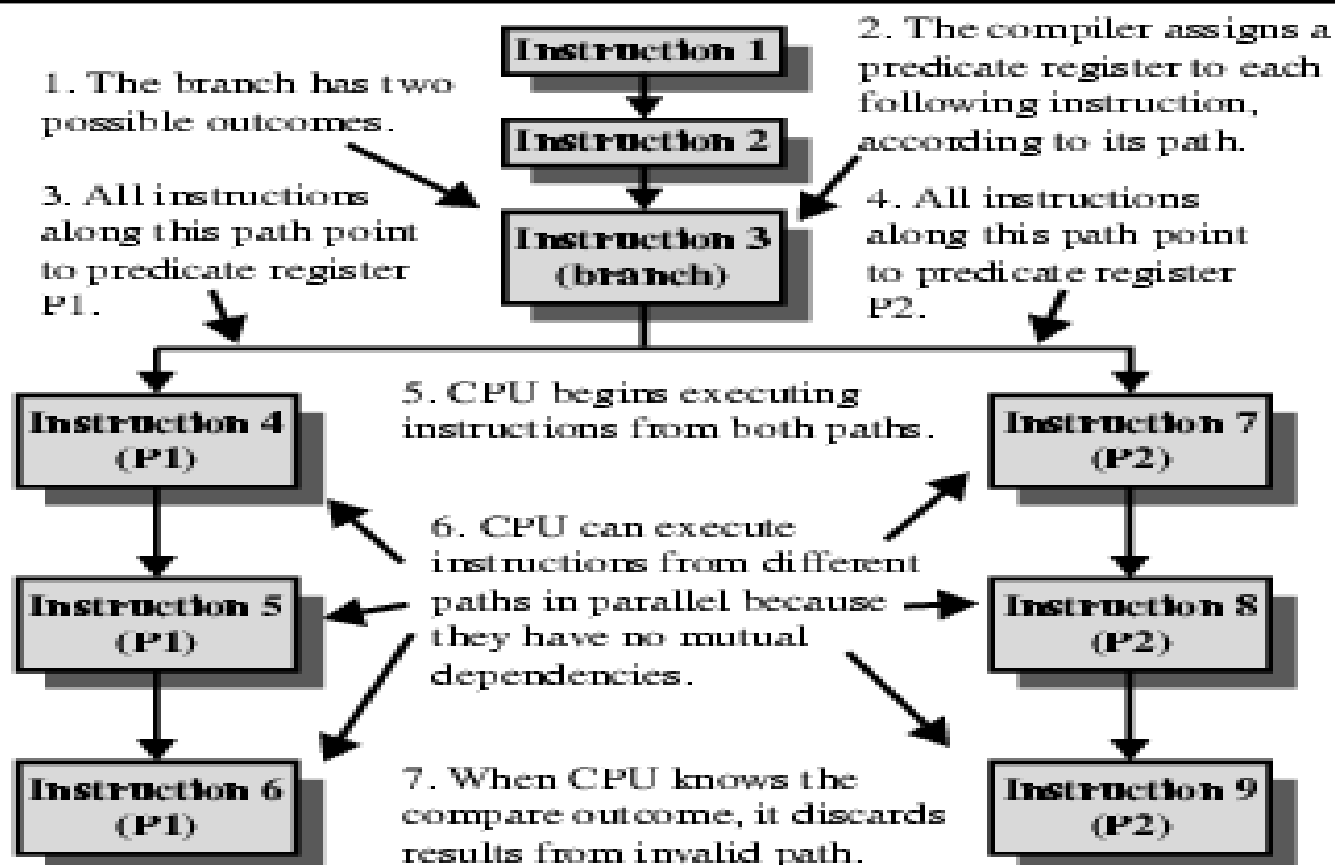
Itanium IA-64

- O IA-64 suporta tanto a predição dinâmica como a estática para os desvios.
- O IA-64 também inclui instruções de SIMD para o processamento multimídia. Instruções multimídia similares às MMX e SSE da arquitetura x86 enxergam os registradores de uso geral como dois operandos de 32 bits, quatro de 16 bits ou oito de 8 bits e os opera em paralelo.
- Para compatibilidade com a família Pentium, uma unidade de controle e decodificação especial para instruções do IA-32 está presente no Itanium.

Itanium IA-64

- Na tentativa de aumentar o desempenho, o processador IA-64 inclui diversas facilidades que não são encontradas em arquiteturas VLIW tradicionais, tornando-se assim o processador VLIW mais complexo já projetado.
- Isso é um paradoxo, já que a arquitetura VLIW tem como objetivo simplificar o hardware transferindo complexidade para o compilador.

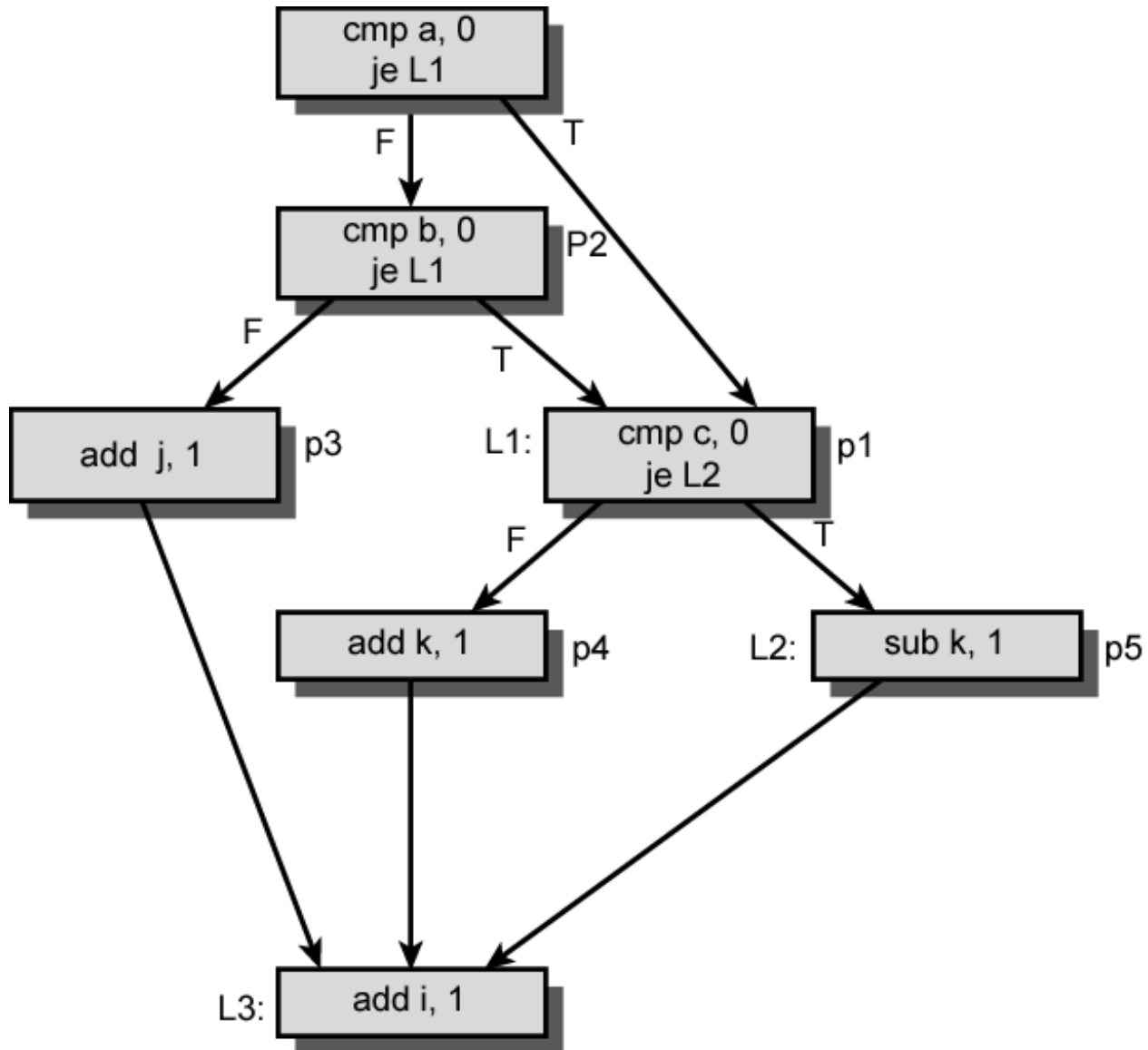
Predicação



The compiler might rearrange instructions in this order, pairing instructions 4 and 7, 5 and 8, and 6 and 9 for parallel execution.



Exemplo de Predicação

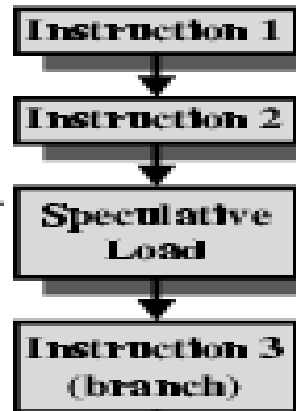


Load Especulativo

- O IA-64 suporta especulação de controle e dados controlada pelo software.
- Para realizar especulação de controle, o compilador move os “loads” para antes do desvio que o controla. O “load” é então marcado como especulativo. O processador não sinaliza exceções em um “load” especulativo.
- Se o desvio de controle for tomado posteriormente, o compilador utiliza uma operação especial check.s para verificar se a exceção ocorreu, desviando então para uma rotina de exceção, quando for o caso.

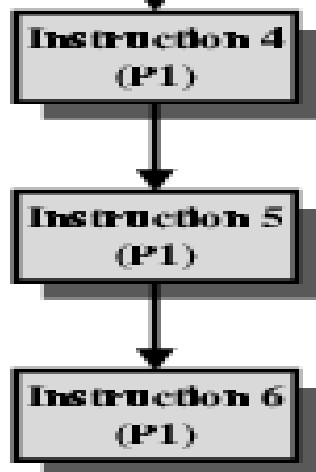
Load Especulativo

1. The compiler scans the source code and sees an upcoming load (instruction 8). It removes the load, inserts a speculative load here and a speculative check immediately before the operation that will use the data (instruction 9).

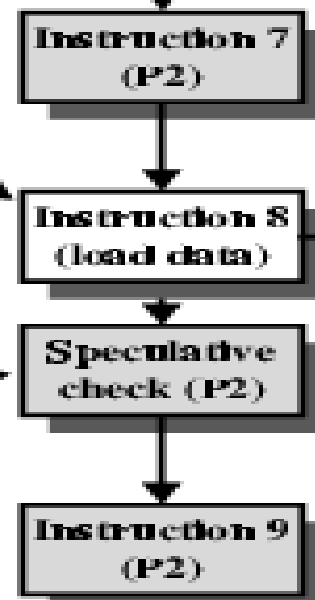


2. At run time, this instruction loads the data from memory before it is needed. If the load would trigger an exception, the CPU postpones reporting the exception.

5. In effect, IA-64 has hoisted the load above the branch.



3. The compiler replaced this load with the speculative load above, so instruction 8 does not actually appear in the program.



4. This instruction checks the validity of the data. If it is OK, the CPU does not report an exception.

Load Avançado

- Para o suporte à especulação de dados, o processador utiliza um tipo especial de “load” chamado de “load avançado”. Se o compilador não conseguir verificar com certeza se pode passar um load na frente de um store, então o “load avançado” é utilizado.
- O processador usa uma estrutura especial chamada ALAT para verificar se o store realizado posteriormente escreveu na mesma posição lida pelo “load avançado”.
- Mais tarde, na posição original do “load avançado”, o compilador usa uma operação especial de verificação para saber se o store invalidou o “load avançado”.
- Se for o caso, a operação de verificação transfere o controle para uma rotina especial de recuperação dos dados.

Software Pipelining

- O compilador pode escalonar os loops segundo a técnica de “software pipeline”. Tradicionalmente, isso requer que o laço seja desenrolado e que os registradores de iterações sucessivas sejam renomeados.
- A arquitetura IA-64 reduz a sobrecarga para esse tipo de técnica eliminando a necessidade de renomeação pela rotação dos registradores para frente.
- Depois de uma rotação, o valor que estava no registrador X será encontrado no registrador $X+1$. Quando usado em conjunto com a predicação, isso permite a expressão natural do “software pipeline”, assim como no seu similar por hardware.

Software Pipelining

```
L1:   ld4 r4=[r5],4 ;;           //ciclo 0 load posinc de 4
      add r7=r4,r9 ;;          //ciclo 2
      st4 [r6]=r7,4 ;;        //ciclo 3 store posinc de 4
      br.cloop L1 ;;          //ciclo 3
```

- Adiciona constante a um vetor e armazena em outro.
- Não há oportunidade para paralelismo no nível de instrução.
- Instrução na iteração i são todas executadas antes de começar a iteração $i+1$.
- Se não houver conflito de endereço entre loads e stores, instruções poderiam ser movidas do laço $i+1$ para o laço i .

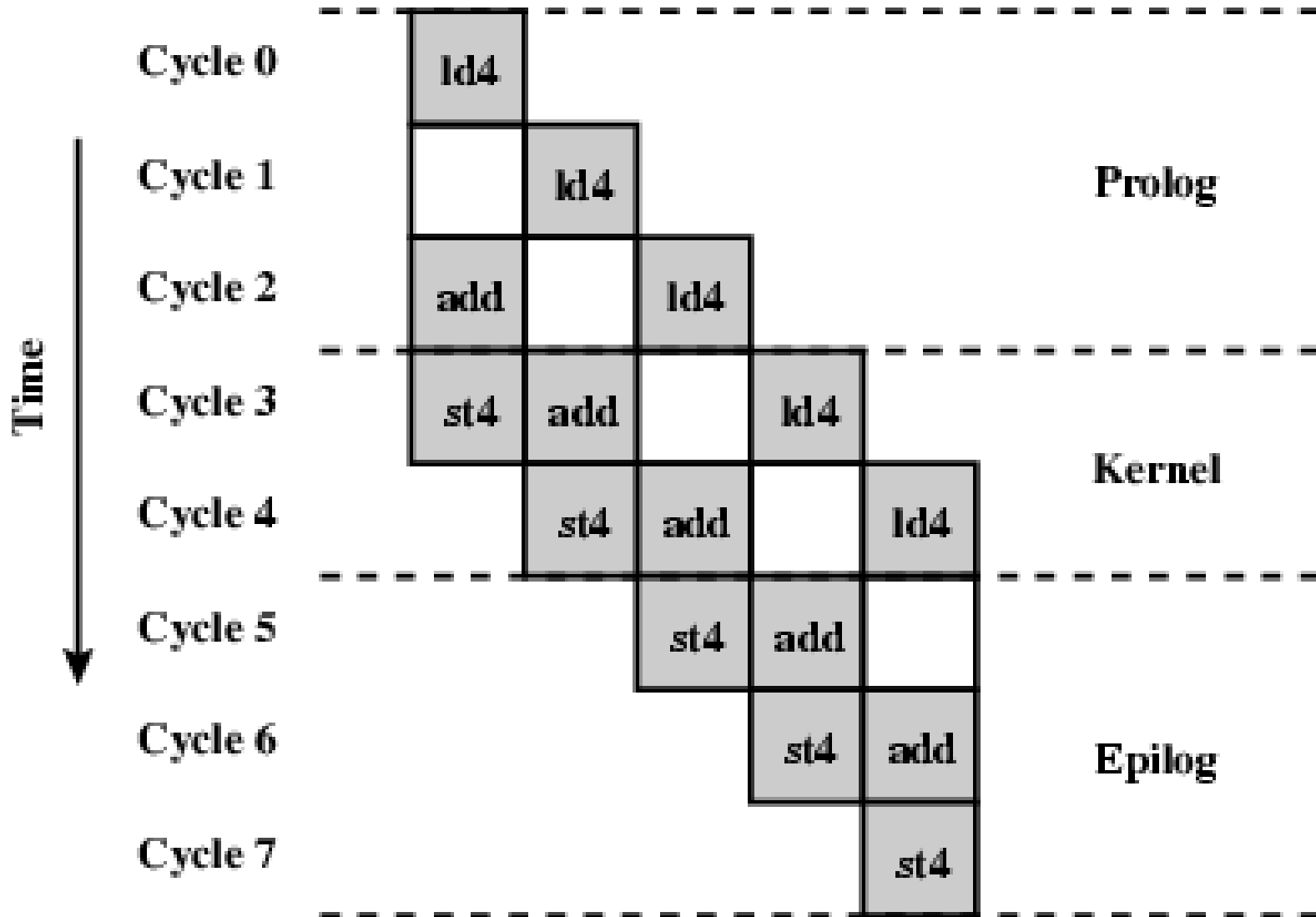
Laço Desenrolado

```
ld4 r32=[r5],4;; //ciclo 0
ld4 r33=[r5],4;; //ciclo 1
ld4 r34=[r5],4 //ciclo 2
add r36=r32,r9;; //ciclo 2
ld4 r35=[r5],4 //ciclo 3
add r37=r33,r9 //ciclo 3
st4 [r6]=r36,4;; //ciclo 3
ld4 r36=[r5],4 //ciclo 4
add r38=r34,r9 //ciclo 4
st4 [r6]=r37,4;; //ciclo 4
add r39=r35,r9 //ciclo 5
st4 [r6]=r38,4;; //ciclo 5
add r40=r36,r9 //ciclo 6
st4 [r6]=r39,4;; //ciclo 6
st4 [r6]=r40,4;; //ciclo 7
```

Detalhe do Desenrolamento do Laço

- **Completa 5 iterações em 7 ciclos**
 - **Comparado com 20 ciclos no código original.**
- **Assume duas portas de memória**
 - **Load e store podem ser feitos em paralelo.**

Diagrama do Software Pipeline



Suporte para o Software Pipelining

- **Renomeação automática de registradores**
 - Uma área de tamanho fixo dos registradores de predicado (p16-p32) e de ponto flutuante (fr32-fr127) pode ser rotacionada.
 - Uma área de tamanho programável dos registradores de uso geral (max r32-r127) pode ser rotacionada.
 - Laço usando r32 na primeira iteração automaticamente usa o r33 na segunda.
- **Predicação**
 - Cada instrução no laço é predicada pela rotação do registrador de predicado.
 - Determina se o pipeline está no prólogo, núcleo ou epílogo.
 - Instruções especiais para terminar o laço.
 - Instruções de desvio que fazem o registrador rotacionar e o contador de desvio decrementar.

Itanium IA-64

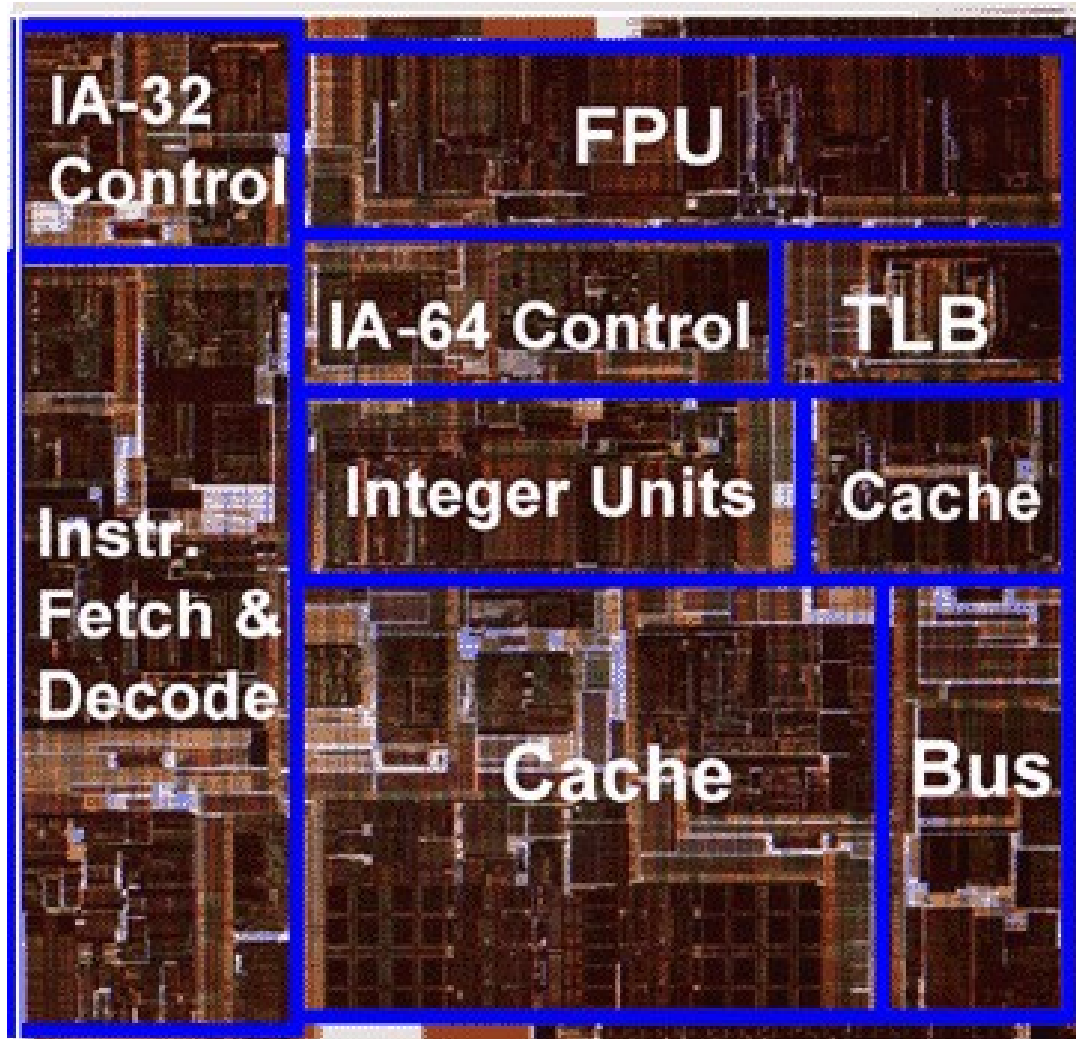
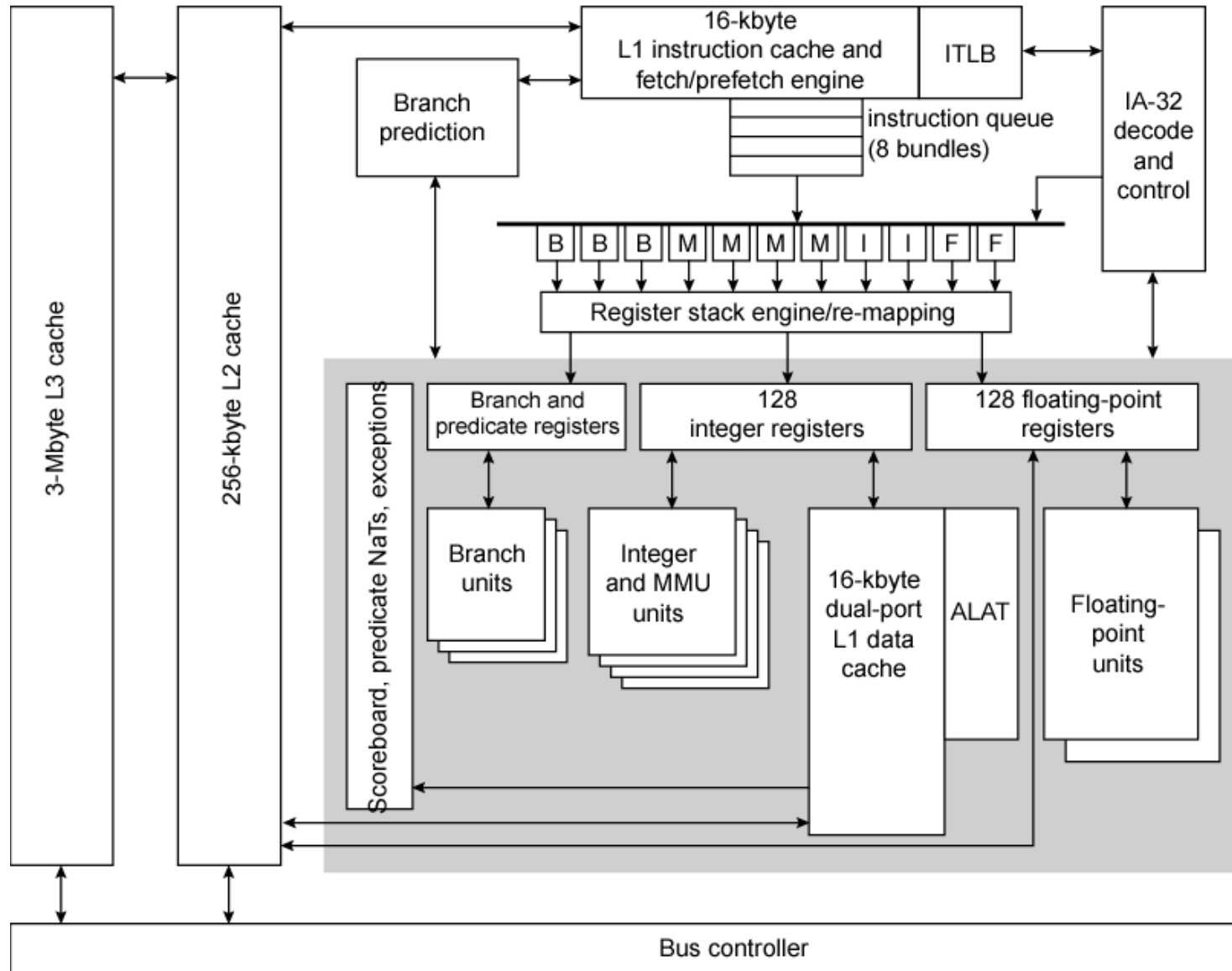


Diagrama do Processador Itanium 2



Itanium 2 (1)

- **Pipeline de 8 estágios**
 - Exceto para as instruções de ponto flutuante
- **Estágios do Pipeline;**
 - 1) Geração do Ponteiro de Instruções (IPG)**
 - Delivers and instruction pointer to L1I cache
 - 2) Rotação das Instruções (ROT)**
 - Busca as instruções e as posiciona.
 - Pacote 0 contém a primeira instrução a ser executada.
 - 3) Decodificação do template das instruções, expansão e dispersão (EXP)**
 - Decodifica o template das instruções
 - Dispersa até 6 instruções através de 11 portas juntamente com o código de operação para as unidades funcionais.
 - 4) Renomeação e Decodificação (REN)**
 - Renomeia registradores
 - Decodifica instruções
 - 5) Leitura do banco de registradores (REG)**
 - Fornece os operandos para as unidades funcionais

Itanium 2 (2)

1) ALU execution (EXE)

- Executa operações

2) Último estágio para detecção de exceções (DET)

- Detecta exceções ;
- Abandona resultados se o predicado da instrução não for verdadeiro;
- Redireciona desvios preditos incorretamente.

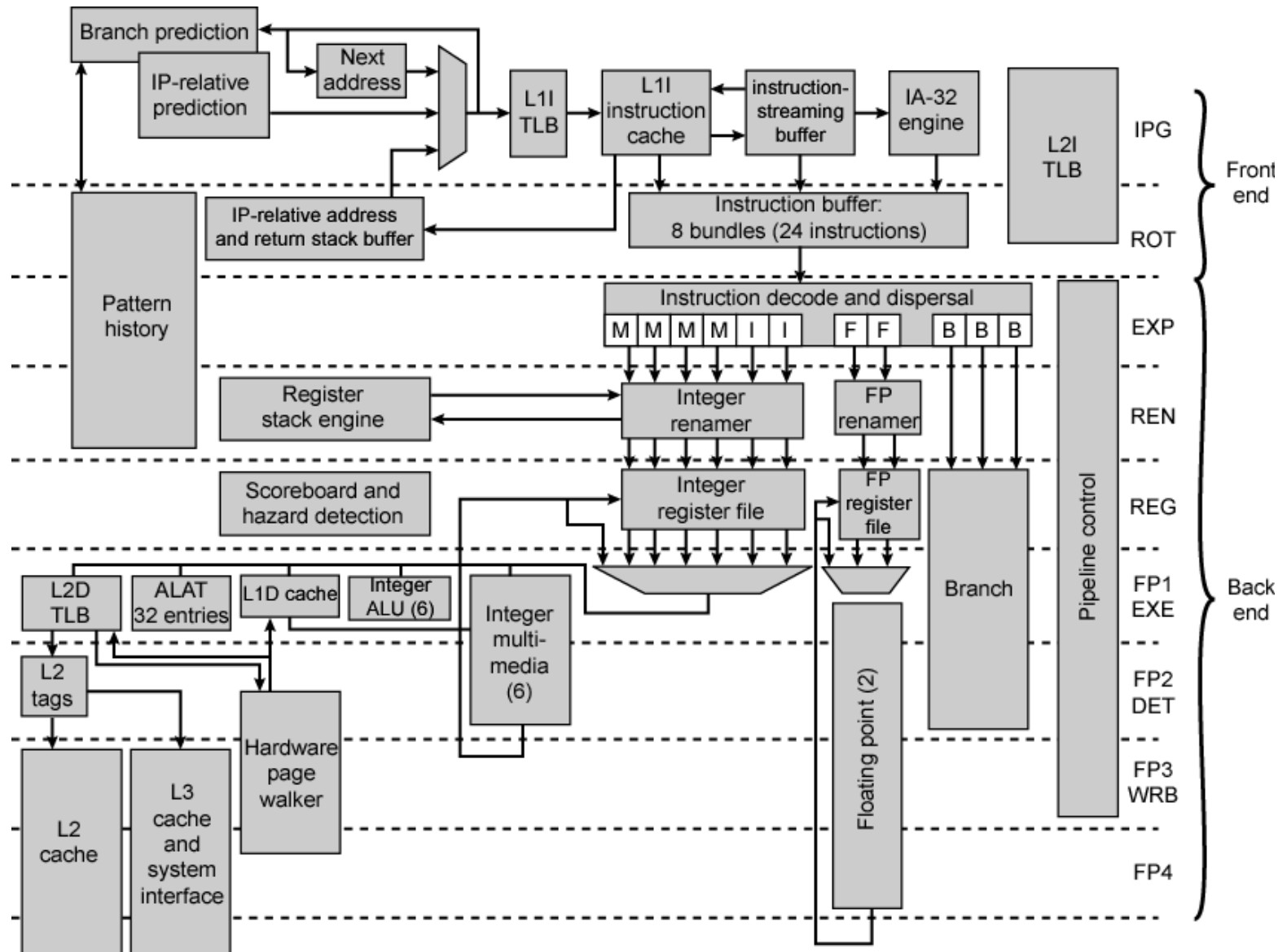
3) Write back (WRB)

- Escreve os resultados de volta para o banco de registradores.

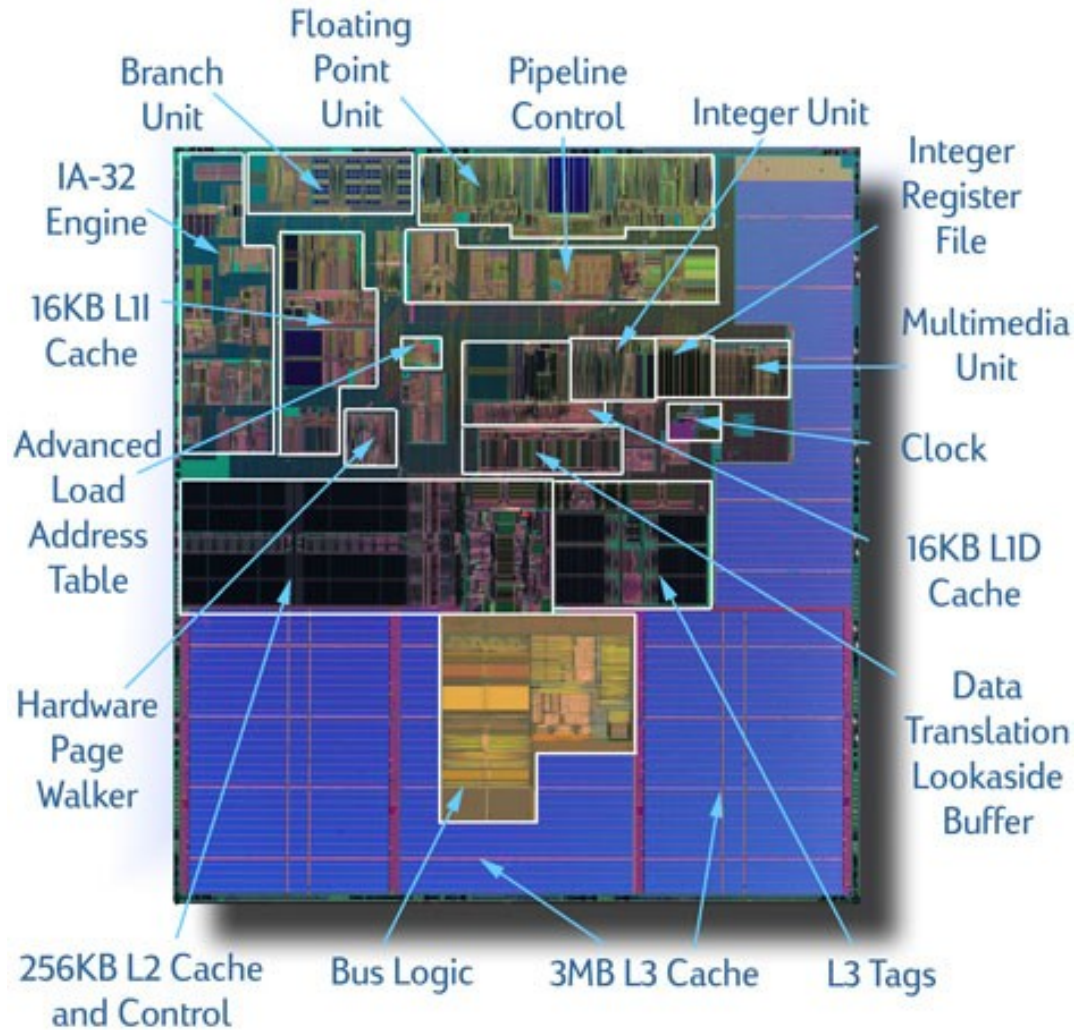
- Instruções de Ponto Flutuante

- Os primeiros cinco estágios são os mesmos;
- Seguidos por quatro estágios de ponto flutuante;
- Seguidos por estágio de “write-back”.

Pipeline do Processador Itanium 2



Itanium 2



McKinley microprocessor

Crusoe

Transmeta

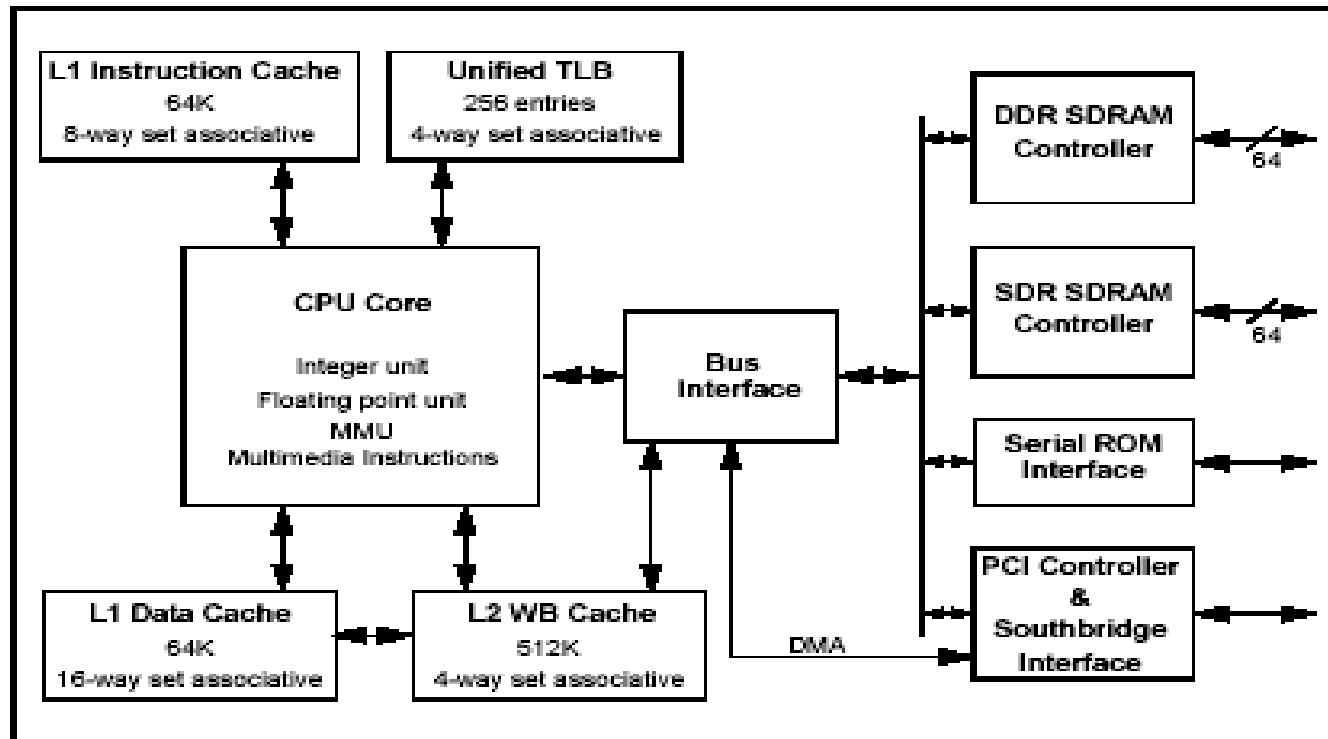
Transmeta Crusoe

- A arquitetura Crusoe representa um ponto muito interessante na história dos processadores VLIW. Tradicionalmente os processadores VLIW foram projetados com o objetivo de maximizar desempenho.
- Os projetistas do Crusoe desenvolveram uma arquitetura com baixo consumo de energia, voltada para aplicações móveis e que fosse capaz de emular a arquitetura de outros processadores, em particular a arquitetura ISA do 80x86 e a máquina virtual Java.
- Possui 64 registradores de uso geral e suporta despacho estritamente em ordem .
- O Crusoe tem duas unidades funcionais para inteiros, uma de ponto flutuante, uma unidade memória (load/store) e uma unidade de desvio.

Crusoe

FIGURE 1

Crusoe Processor Block Diagram - Model TM5800



Crusoe

- **O Crusoe inclui uma cache de instruções L1 associativa de 64KB com associatividade 8 e de associatividade 16 para dados, além da cache L2 unificada de 512KB.**
- **Pipeline para inteiros de 7 estágios e de 10 estágios para ponto flutuante.**
- **Controlador de memória DDR SDRAM com 100-133 MHz, 2.5V**
- **Controlador de memória SDR SDRAM com 100-133 MHz, 3,3V**
- **Controlador PCI (PCI 2.1 compatível) com 33 MHz, 3.3V**
- **Consumo médio de 0.4-1.0 W executando a 367-800MHz, 0.9-1.3V executando aplicações multimedia.**

Crusoe

- A palavra longa no Crusoe tem ou 64 ou 128 bits de largura. Uma palavra de instrução de 128 bits é chamada de molécula pela Transmeta e codifica 4 operações chamadas de átomos.
- O formato da molécula determina como as operações são roteadas para as unidades funcionais.
- O Crusoe, ao contrário do IA-64, usa códigos de condição que são idênticos aos utilizados na arquitetura X86 para facilitar a simulação.

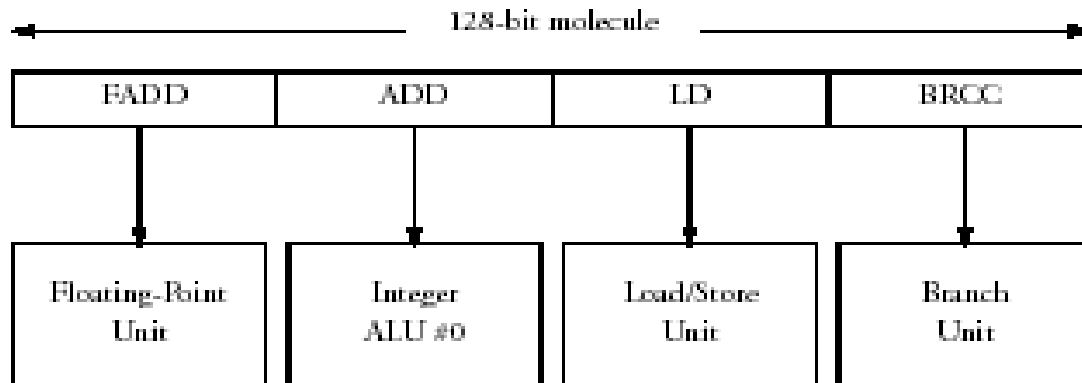


Figure 1. A molecule can contain up to four atoms, which are executed in parallel.

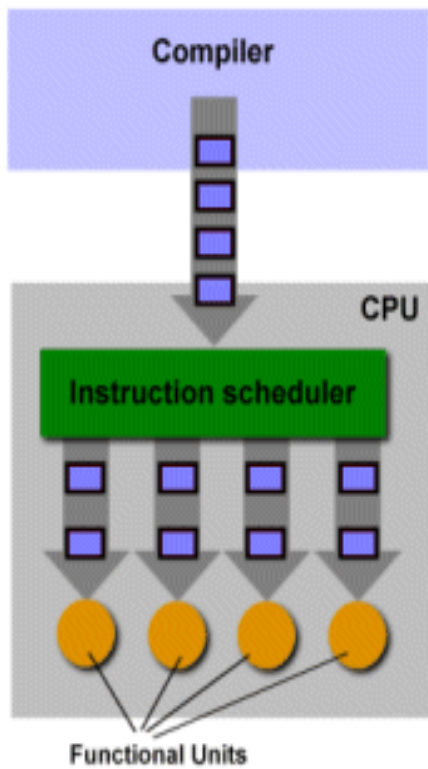
Crusoe

- Todos os *átomos* dentro de uma *molécula* são executados em paralelo, e o formato da molécula diretamente determina como átomos são roteados para as unidades funcionais; isso simplifica bastante o hardware do despacho e decodificação.
- A arquitetura contém 64 registradores de números inteiros, numerados de %r0 a %r63.
- Por convenção, alguns destes registradores são utilizados para manter estados x86, enquanto outros contém estados internos ao sistema, ou podem ser usados como registradores temporários, por exemplo, para renomear os registradores por software.

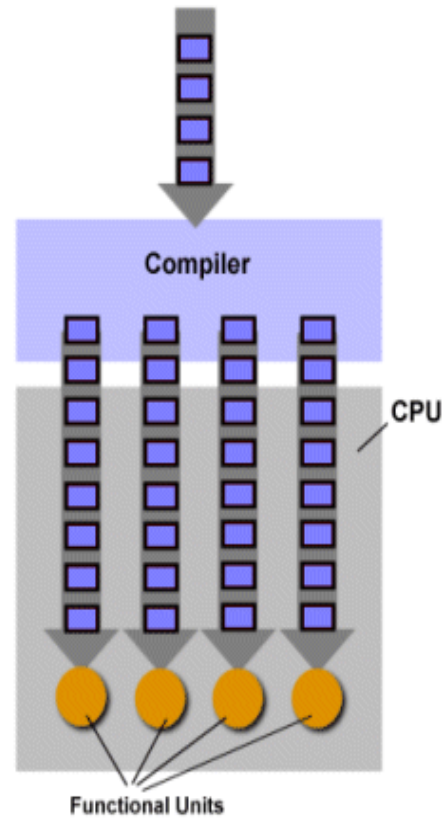
Crusoe – Code Morphing

- **Programas binários do x86 são emulados com ajuda de um tradutor binário chamado “code morphing”, projetado para traduzir dinamicamente as instruções x86 em VLIW . Isso torna a compatibilidade de código binário um problema ultrapassado.**
- **O “code morphing” é um programa que reside em um Flash ROM e é a primeira aplicação a iniciar quando o Crusoe é ligado.**
- **Apenas o código nativo do “code morphing” necessita ser modificado em caso de atualização da arquitetura x86.**
- **Como uma forma de otimizar o desempenho e o consumo, o hardware e o software mantêm em conjunto uma cache de código traduzido.**
- **As traduções são instrumentadas para coletar frequência de execução e histórico de desvio, como forma de auxiliar uma tradução mais eficiente do código pelo “code morphing”.**

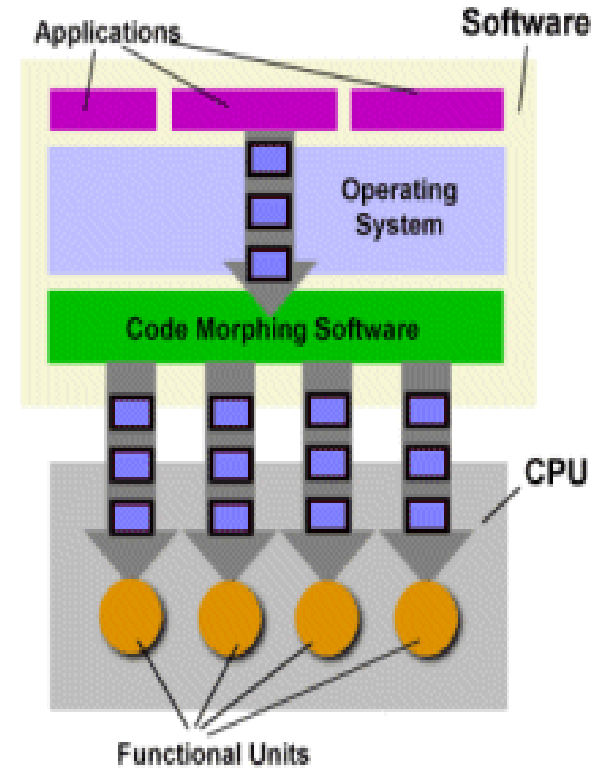
Decoding and Scheduling



**Dynamic Superscalar
Instruction Scheduling**



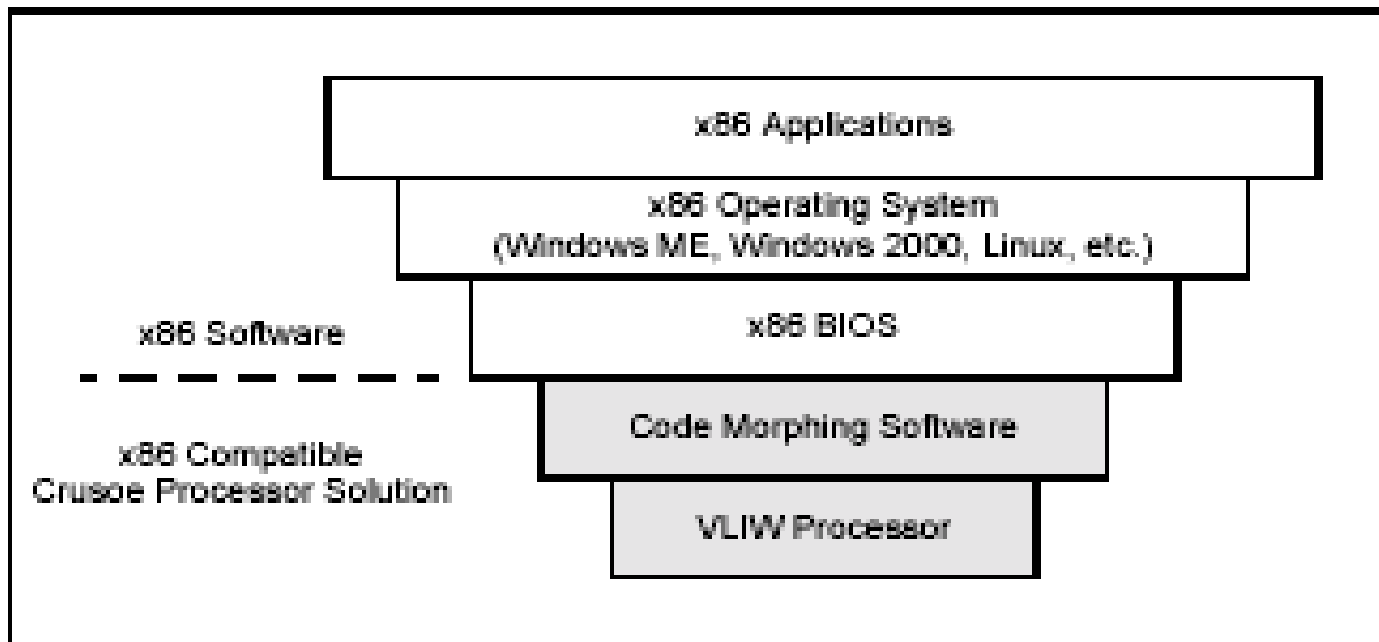
VLIW Instruction Scheduling



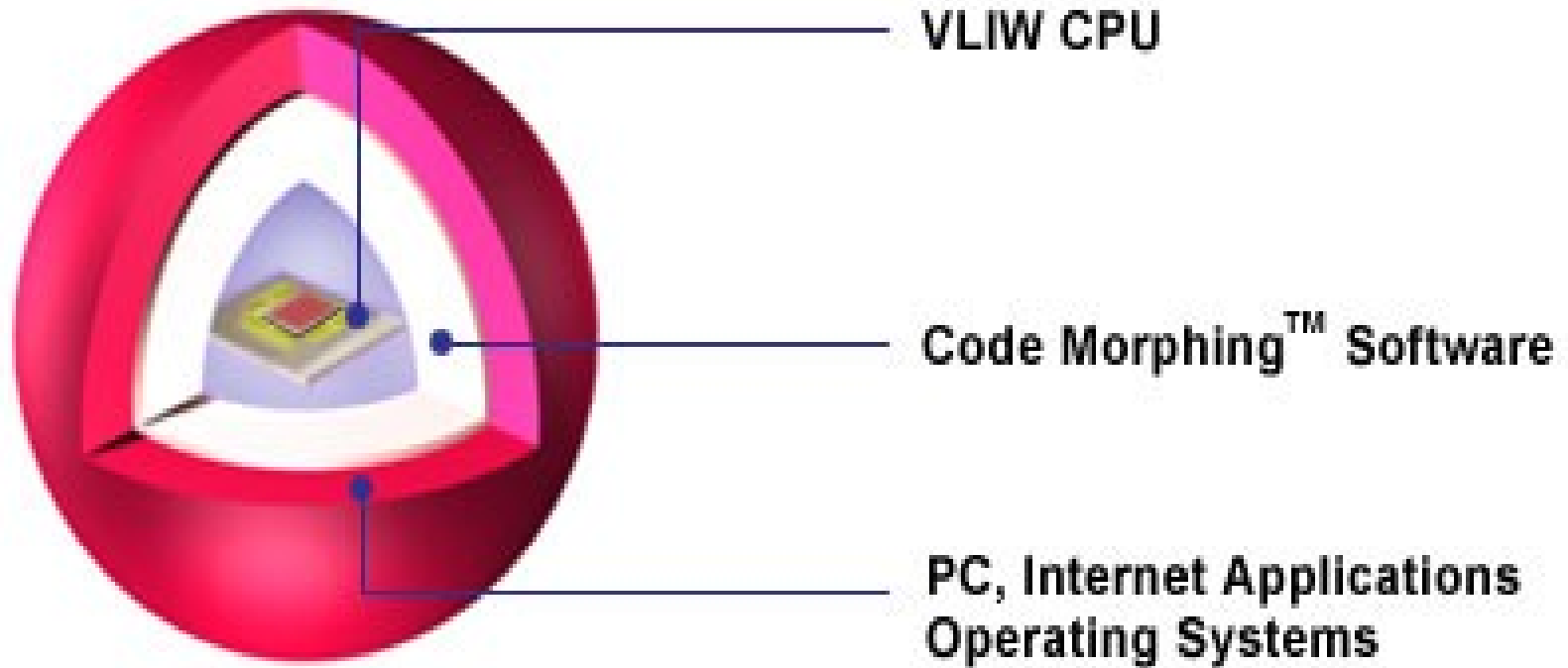
**Crusoe's VLIW
Instruction Scheduling**

Crusoe - Code Morphing

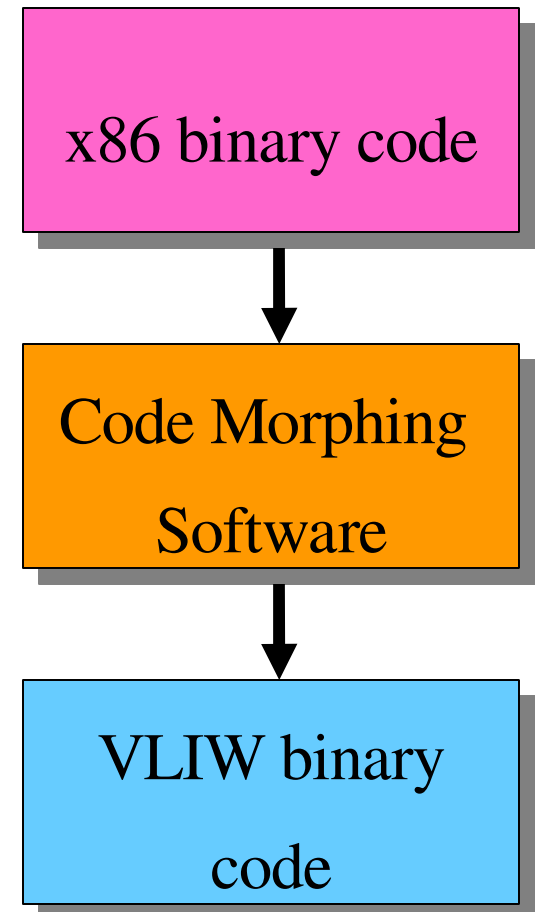
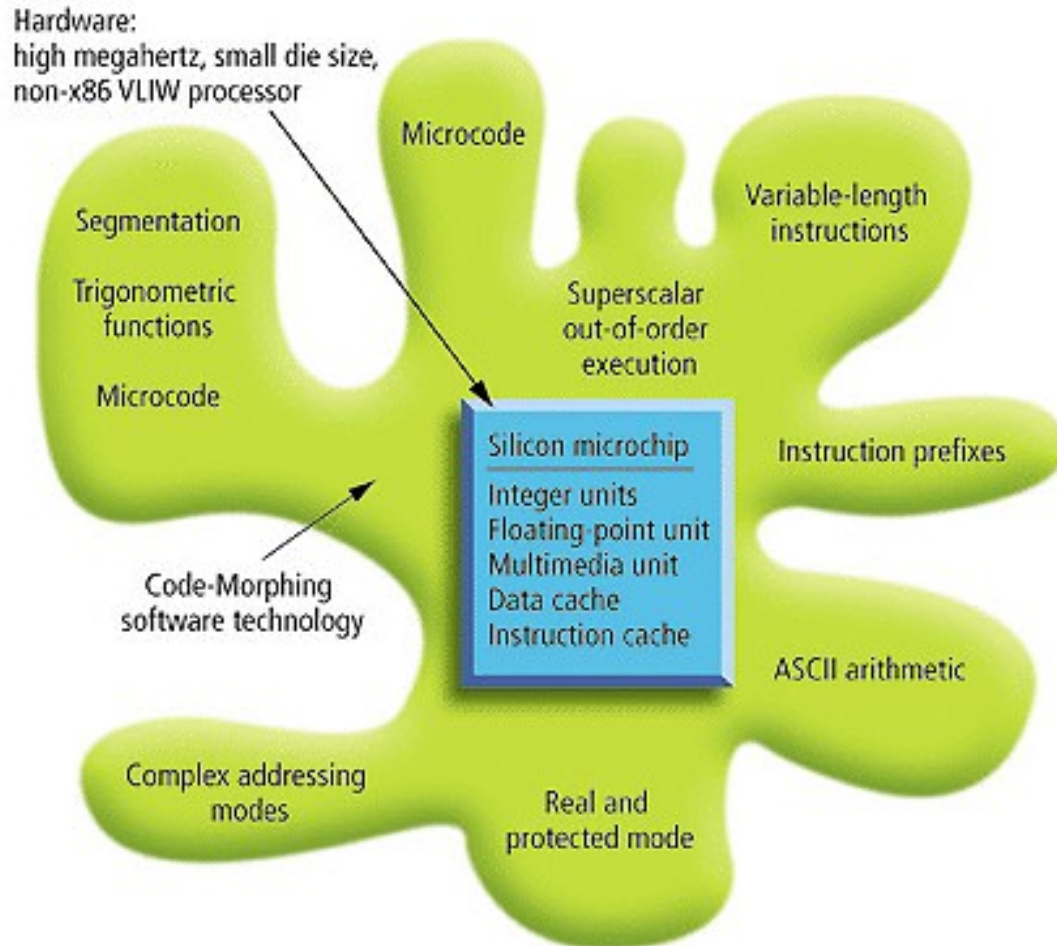
Crusoe Processor Software Hierarchy



Crusoe - Code Morphing



Code Morphing

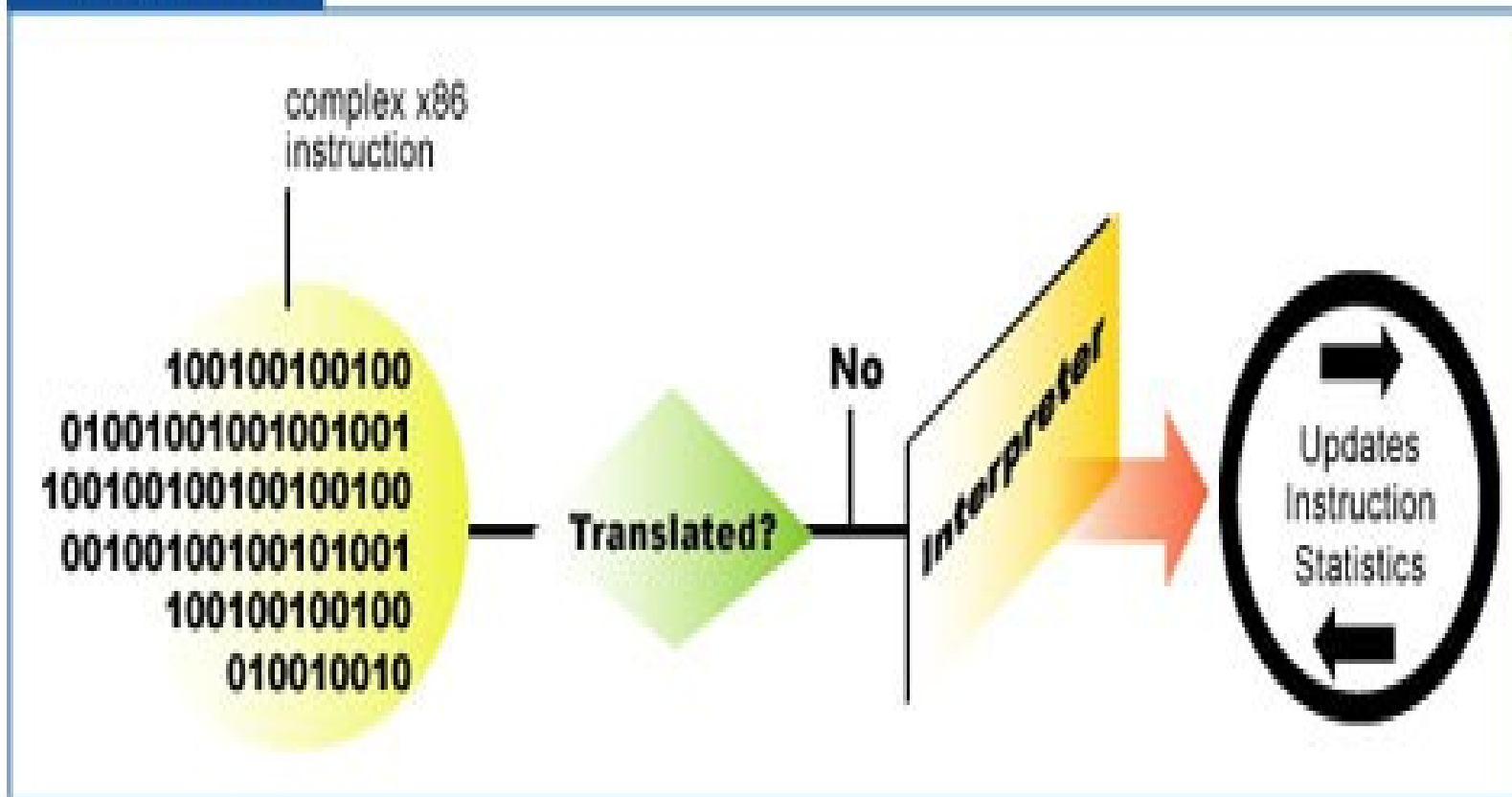


Crusoe - Code Morphing

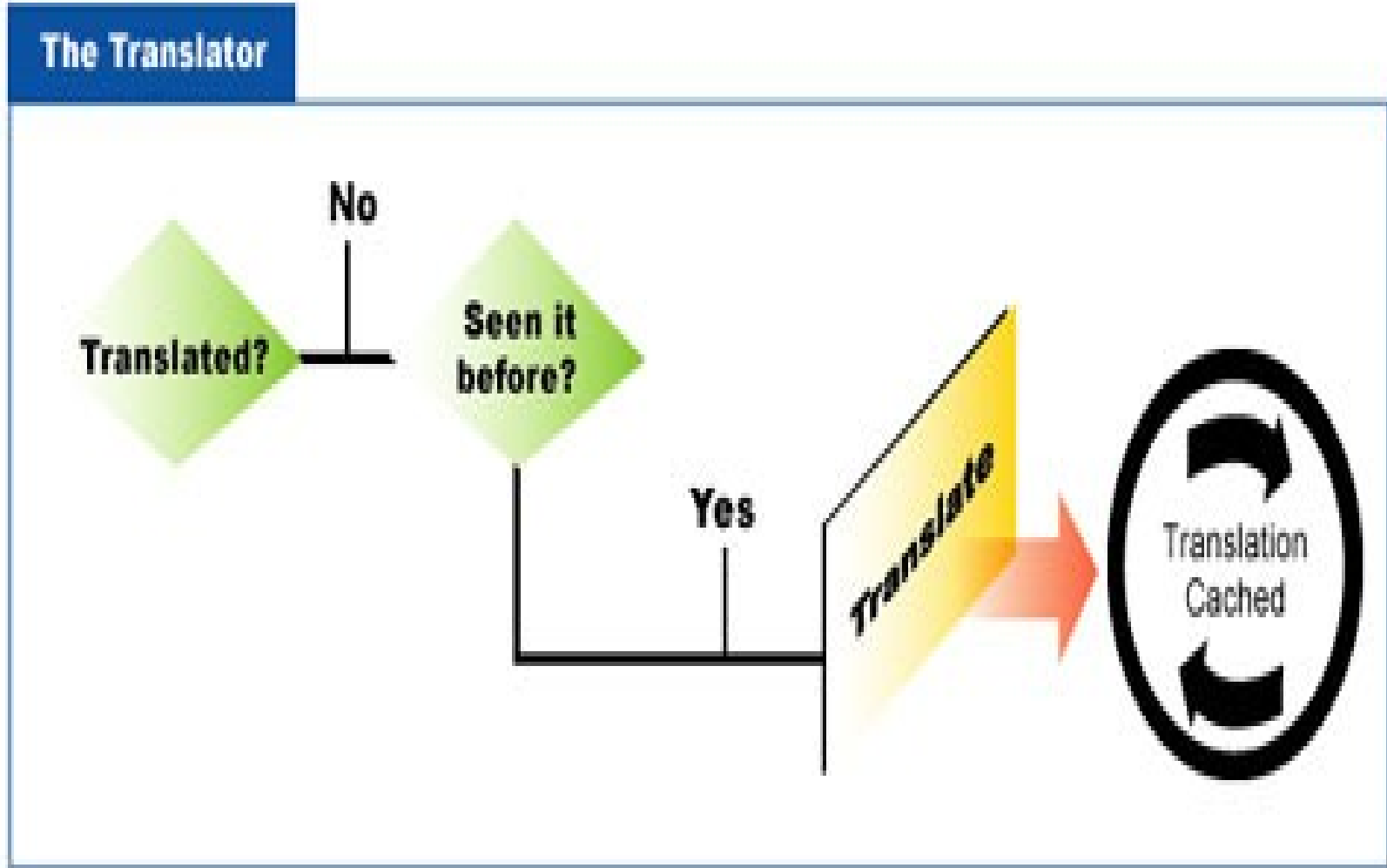
- O software “code morphing” consiste de dois módulos principais que trabalham em conjunto para implementar as funções de um processador x86: Interpretador e Tradutor.
- Interpretador - Interpreta instruções x86, como vários processadores tradicionais. Também filtra códigos executados com pouca frequência para que não seja necessária a otimização que é feita ao reunir informações de estatísticas de execução das instruções mais utilizadas.
- Tradutor – Com base nas instruções x86 mais frequentemente utilizadas, o software “code morphing” invoca o módulo **tradutor** que recompila as instruções x86 em instruções VLIW otimizadas chamada “traduções”.
- As “traduções” nativas reduzem o número de instruções executadas e resultam numa melhor performance.

Crusoe - Code Morphing

Interpreting



Crusoe - Code Morphing



Crusoe - Code Morphing

- O tradutor contém código cujo único propósito é coletar informações tais como frequência de execução de blocos ou histórico de desvios.
- Estes dados podem ser usados para decidir quando e o que otimizar e traduzir. Por exemplo, se um dado desvio é normalmente tomado, o sistema pode então otimizar a favor do caminho mais frequentemente tomado.
- Além disso, o tradutor pode decidir especular o melhor caminho baseado no histórico de desvios.

Crusoe - Code Morphing

```
A. addl %eax, (%esp)      // load data from stack, add to %eax
B. addl %ebx, (%esp)      // ditto, for %ebx
C. movl %esi, (%ebp)      // load %esi from memory
D. subl %ecx, 5           // subtract 5 from %ecx register
```

- **As instruções x86 acima são traduzidas para código VLIW em duas “moléculas”:**

- 1. `ld %r30, [%esp]; sub.c %ecx, %ecx, 5`**
- 2. `ld %esi, [%ebp]; add %eax, %eax, %r30; add %ebx, %ebx, %r30`**

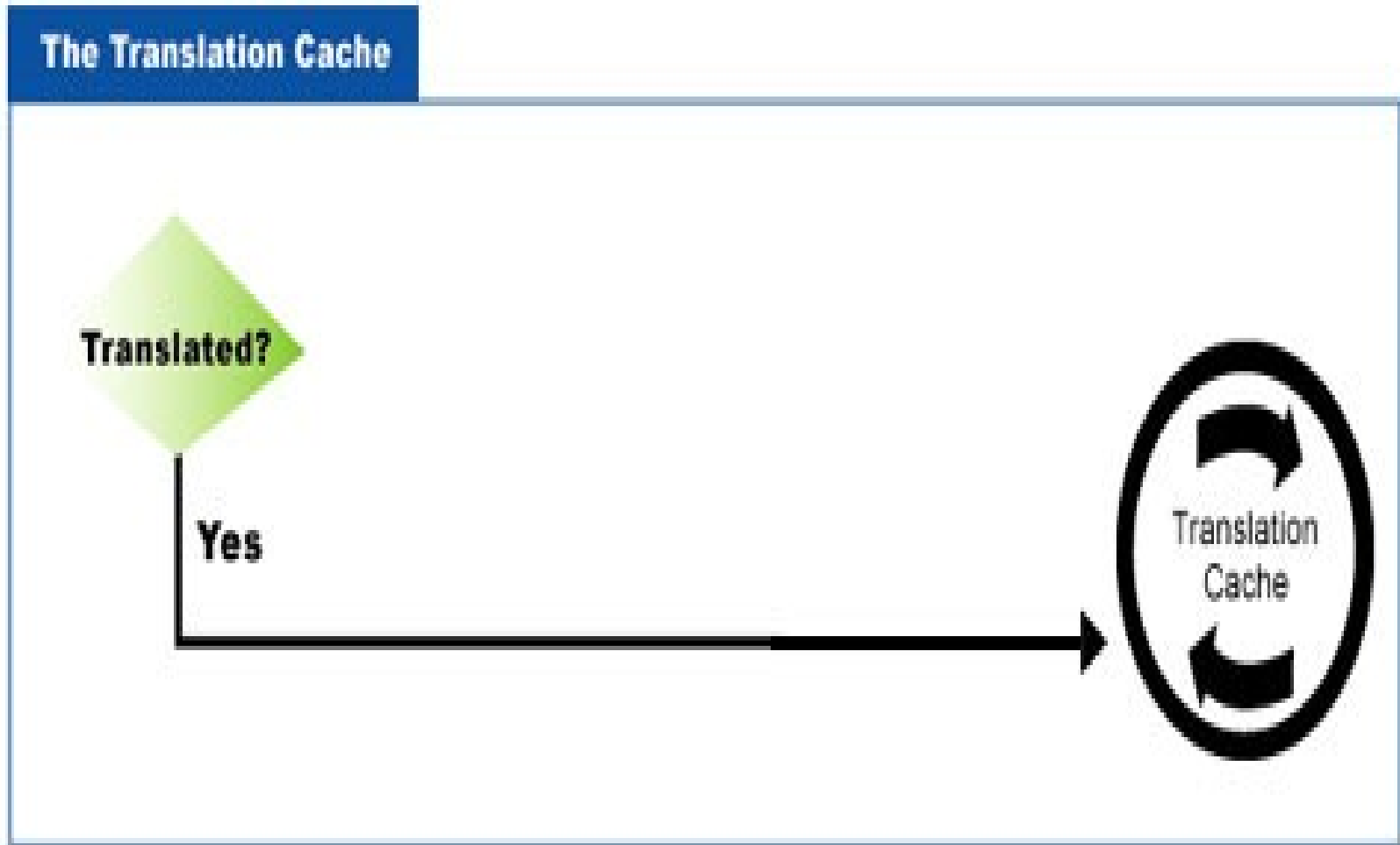
Crusoe - Code Morphing

- Quando o tradutor move uma operação de *load* para a frente de uma operação de *store*, ele converte o *load* em um *load-and-protect* (no qual além de carregar os dados, também registra o endereço e tamanho dos dados armazenado) e o *store* em um *store-under-alias-mask* (o qual verifica as regiões protegidas).
- No caso da operação de *store* sobrescrever os dados lidos anteriormente, o processador dispara uma exceção e o sistema pode tomar uma ação corretiva.
- Usando este mecanismo, é sempre seguro reordenar operações de *load e store*.

Cache de Tradução

- **Cache de Tradução - permite ao software “code morphing” reutilizar as traduções e eliminar as redundâncias.**
- **Ao encontrar seqüências de instruções x86 traduzidas previamente, o “code morphing” pula o processo de tradução e executa diretamente o conteúdo da cache de traduções.**
- **A cache de traduções explora o alto grau de repetição tipicamente encontrados em programas do mundo real.**
- **O custo inicial da tradução é amortizado pelas várias execuções repetidas.**

Cache de Tradução



Crusoe - Arquitetura

- Para modelar corretamente um modelo de exceção precisa, a parte do banco de registradores que contém o estado arquitetural do x86 é duplicada.
- Se ocorrer uma exceção enquanto uma unidade de código traduzida estiver sendo executada, o “code morphing” usa essa cópia para recriar o estado de exceção precisa.
- O processador Crusoe manipula operações x86 de *store* mantendo os dados armazenados em um “*gate store buffer*”, do qual eles são somente liberados para o sistema de memória no momento em que a instrução termina sem exceções.
- Existe suporte de hardware similar ao do IA-64 para especulação de dados e detecção de “alias”.

Crusoe - Arquitetura

Processadores x86 Convencionais	Processador Crusoe com Code Morphing
Traduz uma instrução simples por vez	Traduz um grupo inteiro de instruções por vez.
Traduz uma instrução x86 toda vez que é encontrada.	Traduz as instruções apenas uma vez, salvando o resultado da tradução em uma cache para reuso.
Muito complexo, lógica de transistores complexa, maior consumo.	Muito da complexidade do processador é implementada por software. Menor lógica nos transistores, menor consumo.

Conjunto de Registradores

■ O processador tem 64 registradores de uso geral com a seguinte semântica:

- ◆ **%r63 (%zero)** sempre retorna zero se for lido como operando fonte;
- ◆ **%r62 (%sink)** é um destino descartável (e.g., para comparações); não é lido nunca.
- ◆ **%r59 (%from)** endereço de retorno salvo
- ◆ **%r58 (%link)** endereço de retorno
- ◆ **%r47 (%sp)** é o ponteiro de pilha atual
- ◆ **%r0 (%eax)** para o estado atual da máquina x86
- ◆ **%r1 (%ecx)** para o estado atual da máquina x86
- ◆ **%r2 (%edx)** para o estado atual da máquina x86
- ◆ **%r3 (%ebx)** para o estado atual da máquina x86

Conjunto de Registradores

- ❑ **The 48 of these GPRs are backed by shadowed GPRs: whenever a bundle has its commit bit set, the Commit stage latches the current values of the GPRs into the 'known good' shadow GPRs.**
- ❑ **The processor also includes 32 80-bit floating point registers and 16 FP shadow registers.**
- ❑ **There are also a wide variety of special purpose registers (SPRs), including the condition codes, profiling registers, power control settings and so on.**

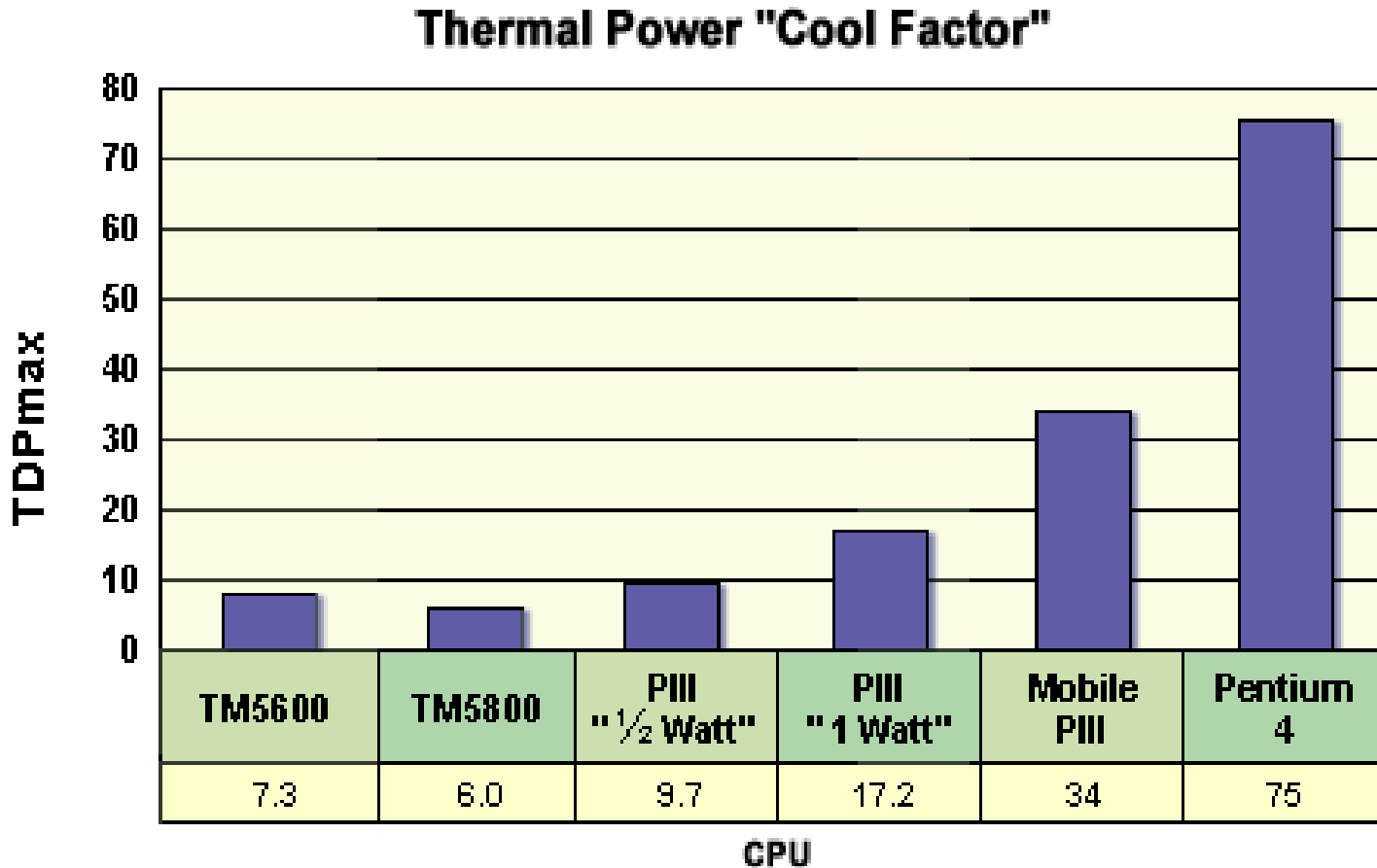
Crusoe - Consumo

- O processador Crusoe foi projetado especificamente para diminuir o consumo de energia, usando o software Code Morphing no lugar da lógica de transistores, gerando bem menos calor que os processadores convencionais.
- Utiliza a tecnologia de gerenciamento de consumo “LongRun”.
- Esta tecnologia fornece ao “code morphing” a habilidade de ajustar a voltagem e frequência de clock “on the fly”, ou seja, dependendo da demanda colocada no processador Crusoe pelo “software”.
- Há também a vantagem de efetuar “upgrade” na lógica do microprocessador sem alterar o “hardware”. Desta forma melhorias são liberadas a um baixo custo e consumo simplesmente liberando uma nova versão do software.

Crusoe - Consumo

- **Como o LongRun funciona? O LongRun opera configurando o processador para rodar a um número de frequências e voltagens diferentes. O algoritmo do “LongRun” no software “code morphing” monitora o processador Crusoe e dinamicamente troca entre estes pontos conforme as condições de mudança em tempo de execução.**
- **O tempo ocioso é monitorado e o LongRun acha um ponto de frequência/voltagem que minimiza o tempo ocioso para o trabalho atual.**

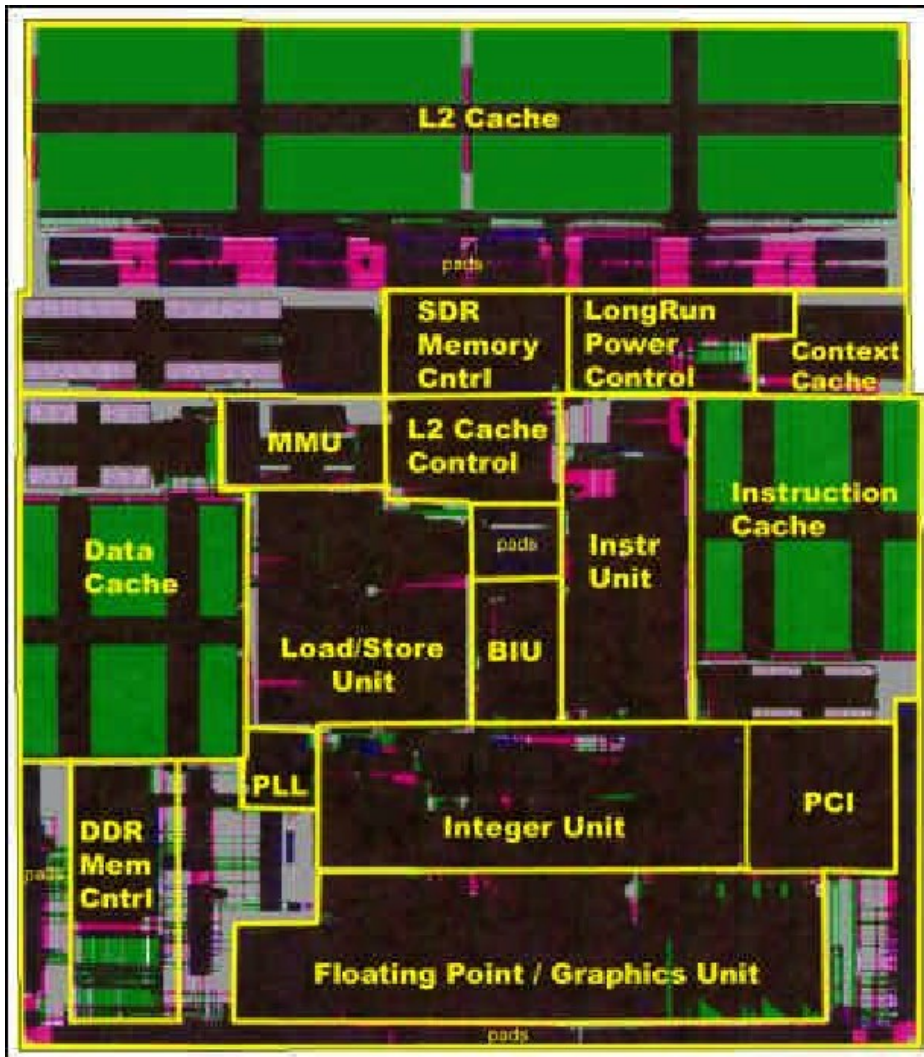
Crusoe - Consumo



Especificações

Crusoe Chips	TM5400	TM5500	TM5600
Frequency Range	500-700MHz	667-800MHz	500-700MHz
L1 Cache	128KB	128KB	128KB
L2 Cache	256KB	256KB	512KB
Main Memory	DDRAM-SDRAM (100 to 133MHz)	DDRAM-SDRAM (100 to 133MHz)	DDRAM-SDRAM (100 to 133MHz)
Upgrade Memory	SDRAM (66 to 133MHz)	SDRAM (66 to 133MHz)	SDRAM (66 to 133MHz)
North Bridge	Integrated	Integrated	Integrated
Package	474 BGA	474 BGA	474 BGA
Sample	Now	Now	Now
Production	Now	Now	Now

TM5400



	TM5400
Frequency Range	500-700 MHz
L1 Cache	128K
L2Cache	256K
Main Memory	DDR-SDRAM
Upgrade memory	SDRAM
North Bridge	Integrated
Package	474 BGA
Fab Partner	IBM
Process Technology	.18u
Die Size	73mm
Sample	Now
Production	Mid 2000

Referências

2. Site da Transmeta – <http://www.transmeta.com>
3. Alexander Klaiber, “The Technology Behind Crusoe Processors”, White Paper, Transmeta Corporation, 2000