Arquiteturas que Exploram Paralismos: VLIW e Superscalar

Introdução

- VLIW (Very Long Instruction Word):
  - Compilador “empacota” um número fixo de operações em uma instrução VLIW
  - As operações dentro de uma instrução VLIW são emitidas e executadas em paralelo
    - Exemplo: TMS320C6201 (DSP)

- Superscalar
  - Processadores que exploram ILP através de escalonamento dinâmico e técnicas de hardware
  - A ênfase da exploração do paralelismo está no hardware e não no software
    - Exemplo: PII, PIII, diversos microprocessadores de propósito geral

VLIW

- Processadores VLIW adotam uma instrução longa que contém, em geral, um número fixo de operações
- Essas operações são buscas, decodificadas, emitidas e executadas simultaneamente
- Todas as operações colocadas dentro de uma instrução VLIW devem ser independentes (não tem dependência de dados)
- Características gerais:
  - Palavras de instruções longas (até 1K poer instrução),
  - Cada operação utiliza uma quantidade (conhecida) de ciclos de *clock* para executar
  - Unidade de controle “emite” uma instrução VLIW a cada ciclo
  - O hardware é formado por múltiplas Unidades Funcionais que são conectadas por meio de um banco de registradores global

VLIW e Superscalar

- Instruções são escalonadas estaticamente pelo compilador

- Emissão de instruções é “menos complicado” que em um processador superscalar
  - Desvantagem: Processadores VLIW não conseguem “reagir” diante de eventos dinâmico (Ex: misses na cache), com a mesma flexibilidade que os processadores superscalares

- Preencher instruções VLIW com nops é, frequentemente, necessário.
  - Impacto: aumento do tamanho do código!
  - Observação: Processadores VLIW mais recentes usam formatos de código mais densos (codificação de instruções) para remover os nops

VLIW e Superscalar

- Solução Superscalar no processador RISC
  - Baseada na semântica de execução seqüêncial
  - Papel do compilador é limitado
  - Hardware superscalar identifica e explora o paralelismo

- EPIC (evolução de VLIW e superscalar)
  - Baseada na semântica de execução paralela
  - ISA EPIC suporta paralelização estática
  - Compilador tem “responsabilidade” na identificação do paralelismo
  - Colaboração entre compilador e hardware

Fusão de Técnicas VLIW e Superscalares

- Superscalar: Necessitam de melhorias para paralelização estática
  - Escalonamento estático
  - Suporte limitado para execução predicada

- VLIW: Necessitam de melhorias para paralelização dinâmica
  - Caches introduzem mudanças na latência das instruções
  - Compatibilidade: tamanho da emissão (tamanho da instrução) e latência podem mudar com um novo hardware

- EPIC: exibe características derivadas de VLIW e Superscalar
  - Interlocks & hardware para execução foram de ordem são compatíveis com EPIC
  - Processadores EPIC podem fazer uso de tradução dinâmica para paralelizar em software

Desafios para Processadores EPIC

- Desenvolver arquiteturas aplicáveis para computação de propósito geral
  - Encontrar substancial paralelismo em programas que são, inerentemente, “difíceis de paralelizar”
  - Fornecer compatibilidade através de gerações de harware
  - Suporte para aplicações emergentes (Ex: multimidia)
- Compilador deve encontrar e “criar” suficiente ILP
- Combinar as melhores características de VLIW & Superscalar
- Escalar a arquitetura para implementações (processo de fabricação) de chips microprocessados modernos

Processador EPIC Itanium

- Projeto conjunto da HP (Hewlett-Packard) e Intel (Junho de 1994)

- Esse projeto resultou em Explicitly Parallel Instruction Computing (EPIC) padrão de projeto:
  - Especificar ILP explicitamente em código de máquina, ou seja,paralelismo está codificado diretamente nas instruções
  - Conjunto de instruções com suporte a predicação
  - Conjunto de instruções escalável (isto é, habilidade para escalar para máquinas com quantidades de hardware diferente)
  - Muitos registradores
  - Execução especulativa das instruções de *load* e *store*

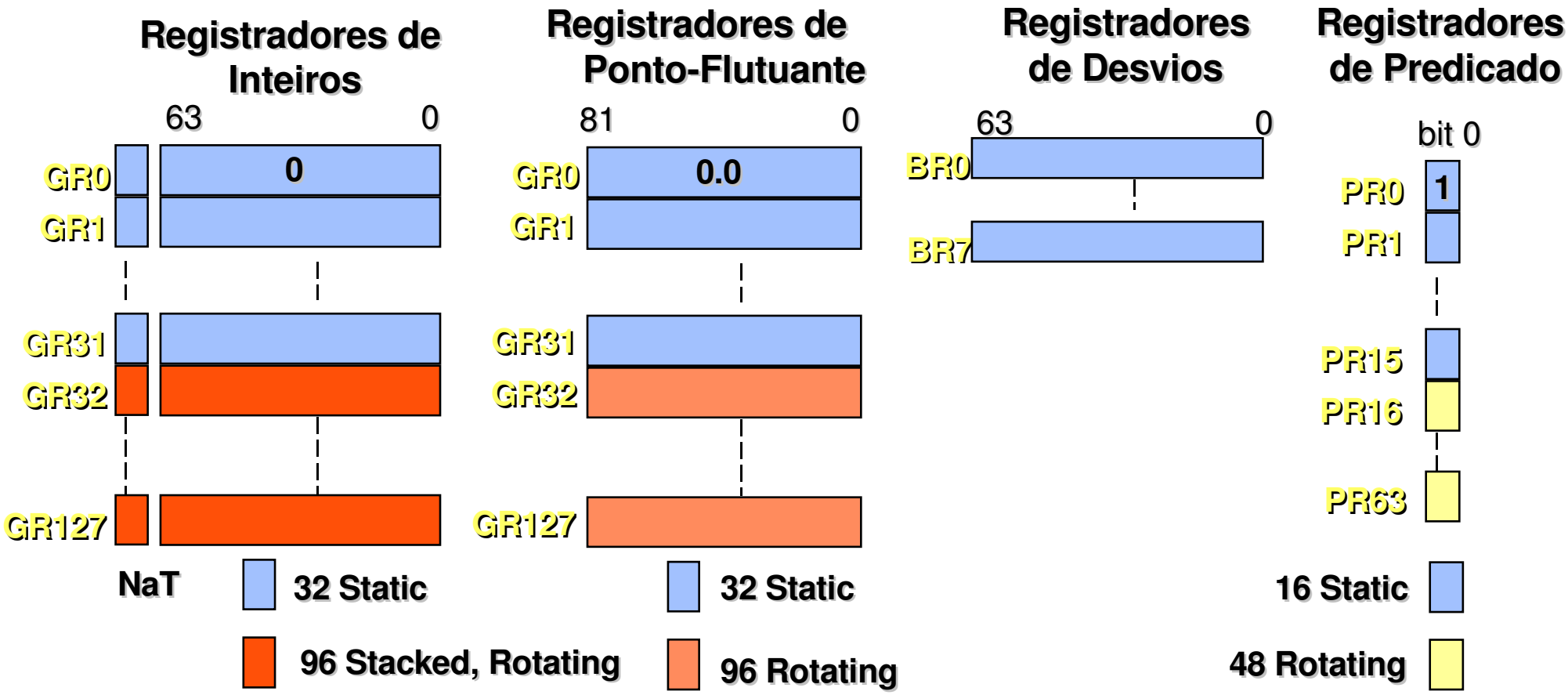
Arquitetura IA-64

- Paralelismo explícito e uso de *templates*
- Suporte para predicação e especulação
- Arquitetura com suporte para ponto-flutuante e multímidia
- Recursos disponíveis para aplicações
  - Grande conjunto de registradores visíveis para o programador

ISA Intel IA-64

- Modelo de Registradores da arquitetura Intel 64-bits (IA-64) :
  - 128 registradores de propósito geral, cada com com 64-bits (GR0-GR127) para armazenar valores de computações de inteiro e aplicações multímidia
    - Cada registrador possui um bit adicional NaT (*Not a Thing*) para indicar se o valor armazenado é válido
  - 128 registradores de ponto-flutuante de 82-bits (FR0-FR127)
    - registradores f0 e f1 são apenas para leitura com valores +0.0 e +1.0
  - 64 registradores de predicado de 1-bit (P0-PR63)
    - O registrador p0 é somente para leitura com valor 1 (true)
  - 8 registradores de desvio de 64-bits (BR0-BR7) para armazenar o alvo de desvios indiretos

Bancos de Registradores do Intel IA-64



Conjunto de Instruções IA-64

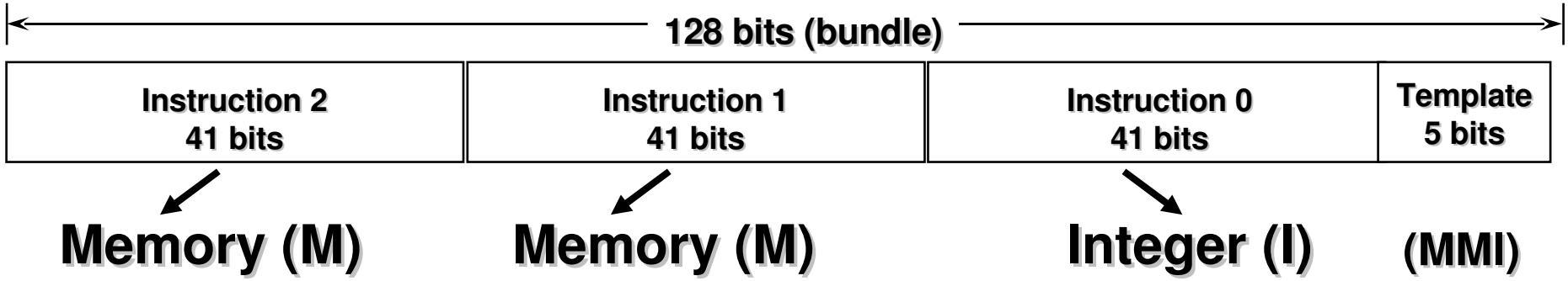
- Operações IA-64 possuem 41-bits e o formato é o seguinte:
  - op-code
  - Campo de predicado (6 bits)
  - 2 campos para operandos (7 bits cada)
  - 1 campo para registrador de destino (7 bits)
  - Campos especiais
- 6 tipos de operações
  - A: ULA - inteiros ==> I-unit or M-unit
  - I: Não-ULA - inteiros ==> I-unit
  - M: Memória ==> M-unit
  - B: Desvio ==> B-unit
  - F: Ponto-Flutuante ==> F-unit
  - L: Imediato longo ==> I-unit
- Operações IA-64 são organizadas pelo compilador em instruções (*bundles*)

Instruções IA-64

- Um *bundle* possui128bits e contém 3 operações de 41-bits e um campo *template* de 5-bit que armazena informações sobre o agrupamento das operações
- IA-64 não insere nops para preenchimento de instruções
- O *template* indica:
  - Primeiros 4 bits: tipos das operações
  - Último bit (stop bit): se a instrução (*bundle*) pode ser executada em paralelo com a próxima instrução

- O compilador pode colocar operações dependentes e independentes em um mesmo *bundle* pois o *template* mantém informação se essas operações podem executar em paralelo ou não

Instruções IA-64



- **Template IA-64 indica:**
  - **Tipo da operação**
    - MFI, MMI, MII, MLI, MIB, MMF, MFB, MMB, MBB, BBB
  - **Relacionamento Intra-bundle**
    - M / MI or MI / I
  - **Relacionamento Inter-bundle**

M=Memory
F=Floating-point
I=Integer
L=Long Immediate
B=Branch

Escalabilidade IA-64

- Uma instrução contendo 3 operações corresponde a um conjunto de três UFs

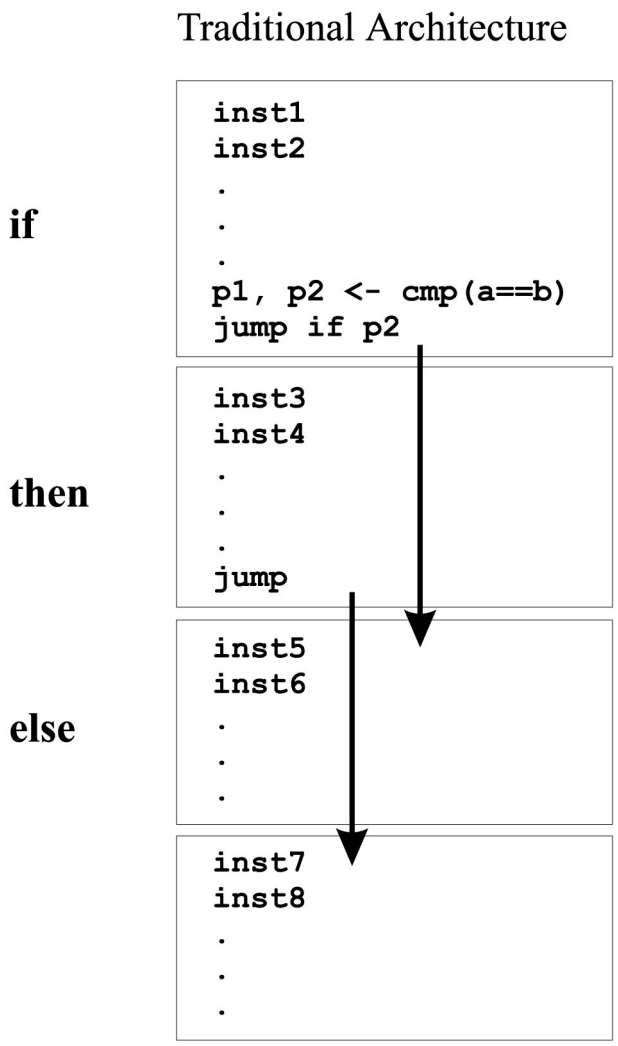
- Se um processador IA-64 possui *n conjuntos de 3 UFs* cada um deles usando informações do template, seria possível encadear instruções de forma a criar uma instrução maior (*bundle*) de tamanho: n*tamanho_instrução_original

- Esta é a maneira utilizada para fornecer escalabilidade IA-64 para qualquer número de UFs

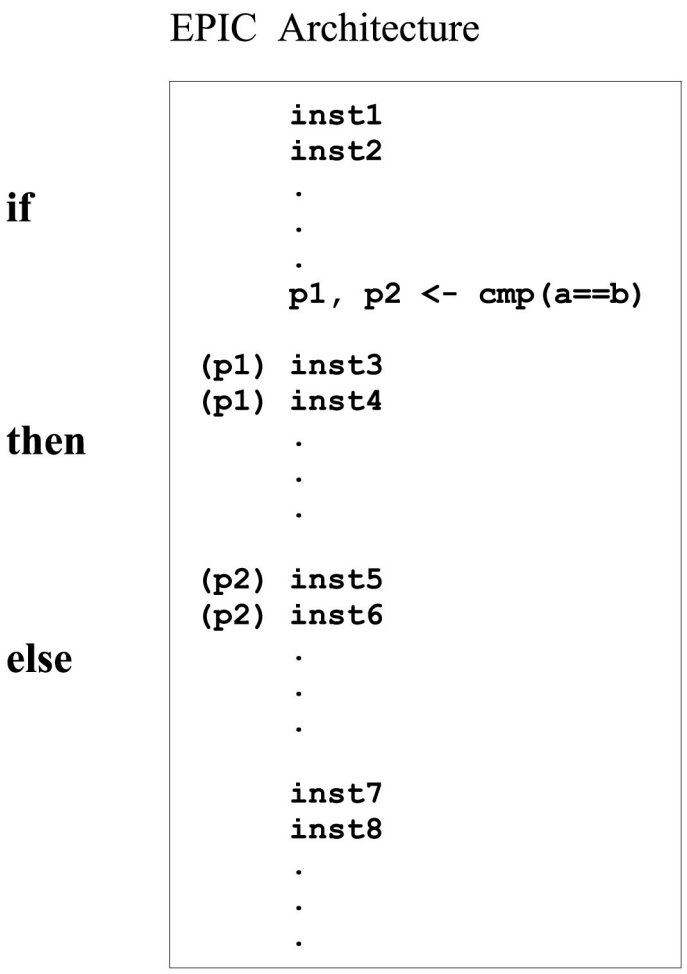
Predicação em IA-64

- Predição de desvios: Paga-se uma forte penalidade, em termos de ciclos perdidos, se a predição está errada
- Compiladores IA-64 usam predicação para remover penalidades causadas pelos “erros” na previsão de desvios
- Quando o compilador encontrar uma declaração de desvio, ele (o compilador) marca todas as instruções em cada caminho do desvio com um identificador predicado
- IA-64 define um campo de 6bits (endereço do registrador de predicado) em cada *bundle*
- Instruções que estão em um mesmo “lado” do desvio, irão compartilhar o mesmo predicado

Declarações If-then-else



(a)



(b)

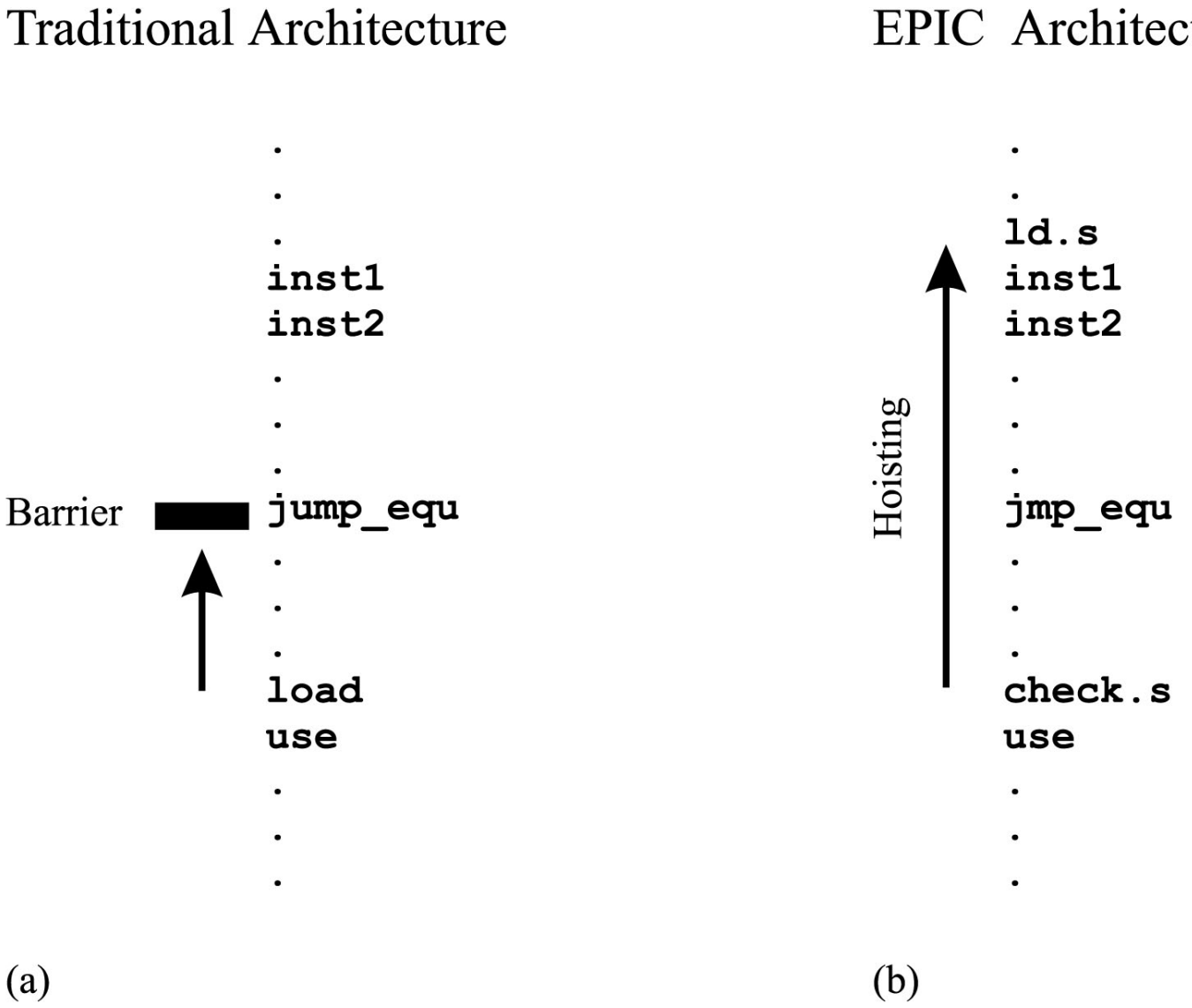
Predicação em IA-64

- Em tempo de execução a UC verifica o template e emite operações independentes em paralelo para as UFs
- Desvio predicado: o processador executa o código dos “dois lados” do desvio
  - Nenhum dos resultados é definitivamente armazenado até ter a “certeza” de qual foi o “lado” tomado do desvio
- Para isso, o processador verifica os registradores de predicado:
  - Se o reg. de predicado contém 1,
    - ===> a operação está no caminho válido
  - Se o reg. de predicado contém 0,
    - ===> a operação é inválida, o processador descarta o resultado

Técnica de Especulação

- Consiste em carregar dados da memória antes que o programa necessite deles
  - Objetivo é minimizar o impacto da latência de memória
- Carregamento (*load*) especulativo é uma combinação de otimizações em tempo de compilação e tempo de execução
  - ==> especulação controlada pelo compilador
- O compilador verificar instruções que necessitarão de dados da memória e, sempre que possível, move um *load* para um ponto inicial no fluxo de instruções a frente (antes) da instrução que necessitará dos dados
- Em processadores superscalares:
  - Um *load* é movido até a 1a. Instrução de desvio que representa uma barreira
- Carregamento especulativo combinado com predicação permite mais flexibilidade para que o compilador reordene as instruções e para que possa deslocar *loads* antes de desvios

Execução Especulativa



Carregamento (*loading*) Especulativo

Instrução de *load* especulativo **ld.s**
Instrução para verificação (*check*) especulativo **chk.s**

- O Compilador:
  - Insere a instrução de verificação *(check)* especulativa imediatamente antes da instrução que utilizará os dados da memória
  - Reorganiza as instruções (já com a posição alterada) de modo que o processador possa emiti-las em paralelo
- Em tempo de execução:
  - O processador encontra ld.s primeiro e tenta recuperar os dados da memória
  - ld.s procede a busca dos dados na memória e detecção de exceções (ex: verifica a validade do endereço)
  - Se uma exceção é detectada, ld.s não “reporta” essa exceção
  - Ao invés disso, ld.s apenas “marca” o registrador alvo (através de um bit de token)

Carregamento (*loading*) Especulativo - “especulação de dados”

- Mecanismo de especulação também pode ser usado para mover um *load* antes de um *store*

- Mesmo se ainda não é conhecido se as referências do *load* e o *store* são sobrepostas

Checagem e Carregamento Especulativo

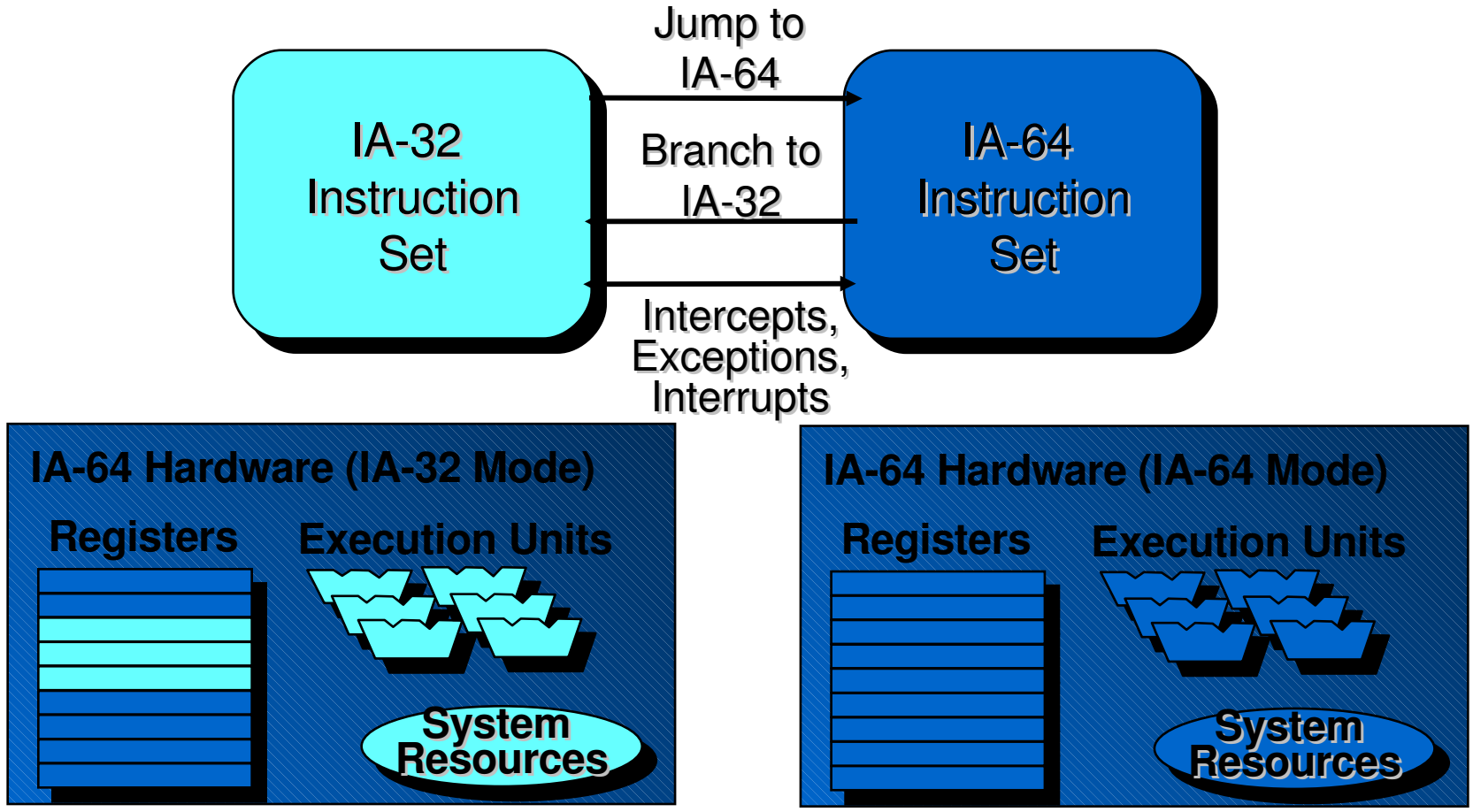
- Distribuição de exceções é de responsabilidade da instrução chk.s
  - chk.s invoca o sistema operacional se o registrador alvo é marcado (se o bit de token é 1)

- Dependendo da saída do desvio, a instrução chk.s pode ser (ou não) executada

  ==> De forma que pode acontecer de uma exceção detectada por ld.s é nunca detectada

- Carregamento especulativo com ld.s/chk.s parece com as declarações TRY/CATCH em linguagens de programação de alto nível (e.x: Java)

Compatibilidade Binária com Instruções IA-32



- **Instruções IA-32 são suportadas por meio de recursos de hardware compartilhados**
- **Desempenho similar a processadores IA-32**

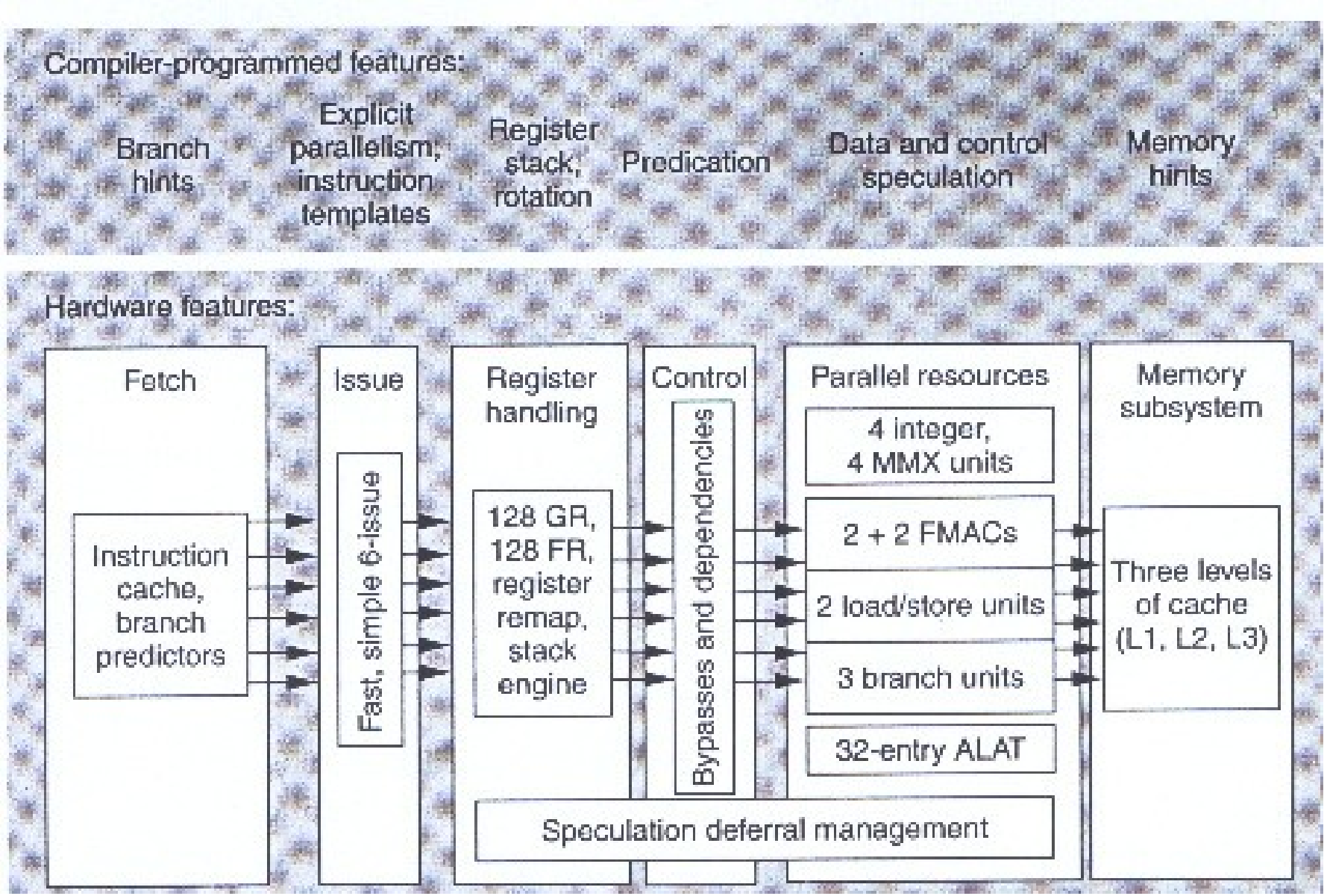
Compatibilidade binária com PA-RISC

- **Transparência:**
  - **Tradutor dinâmico de código objeto em HP-UX converte, automaticamente, código PA-RISC para código nativo IA-64**
  - **Código traduzido é preservado para reuso posterior**
- **Desempenho:**
  - **Tradução "toma", em média, 1-2% do tempo de execução Execução do código nativo toma 98-99%**

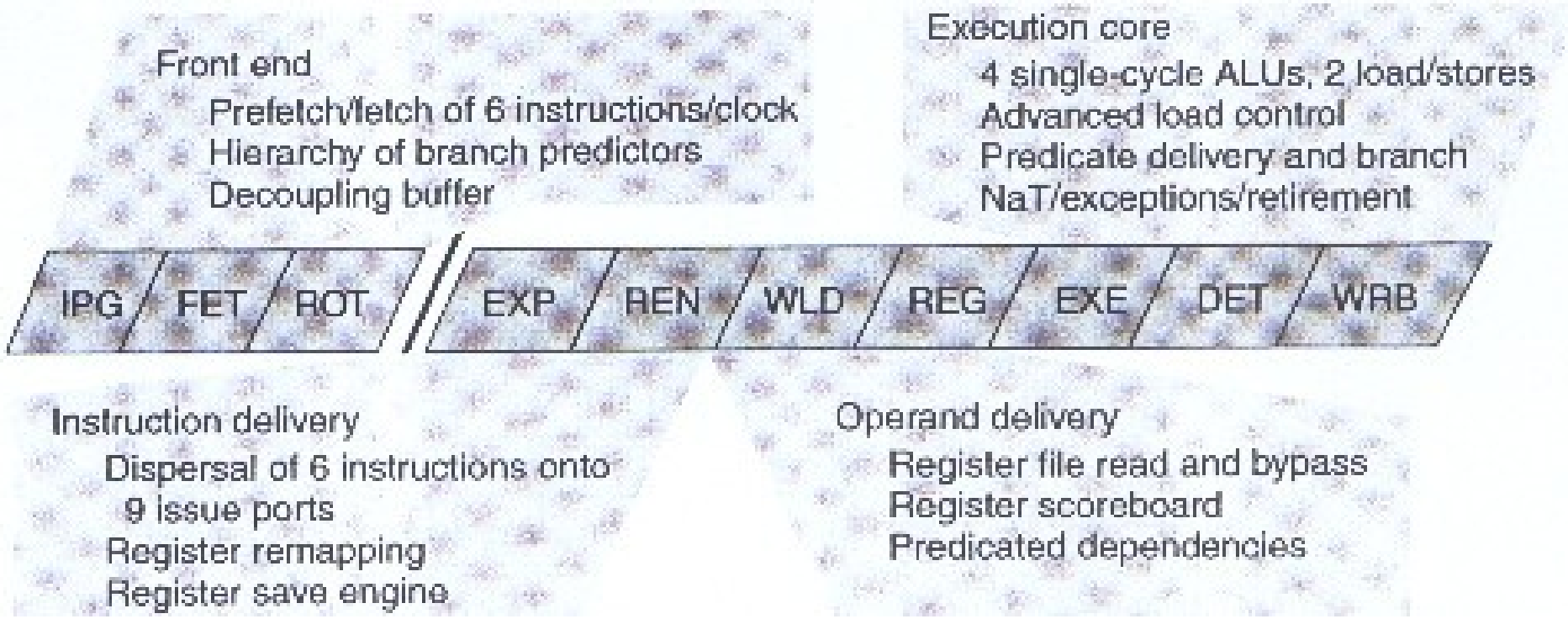
Itanium

- Processador de 64bits
- Compatibilidade (binária) com código Intel IA-32
- EPIC (*explicitly parallel instruction computing*) é aplicado.
- Pipeline Largo (3 instruções EPIC)
- 10 estágios de pipeline
- UFs: 4 inteiros, 4 multimidia, 2 load/store, 3 desvios, 2 Ponto-Flutuante estendido, 2 Ponto-Flutuante de precisão simples
- Predição de desvios multiníveis além da predicação
- Caches de Dados e Instruções de 16 KB 4-way set-associative (L1)
- Cache L2 de 96 KB 6-way set-associative
- Cache L3 de 4 MB
- 800 MHz, processo de fabricação de 0.18 micron

Visão Conceitual do Itanium



Pipeline do Processador Itanium



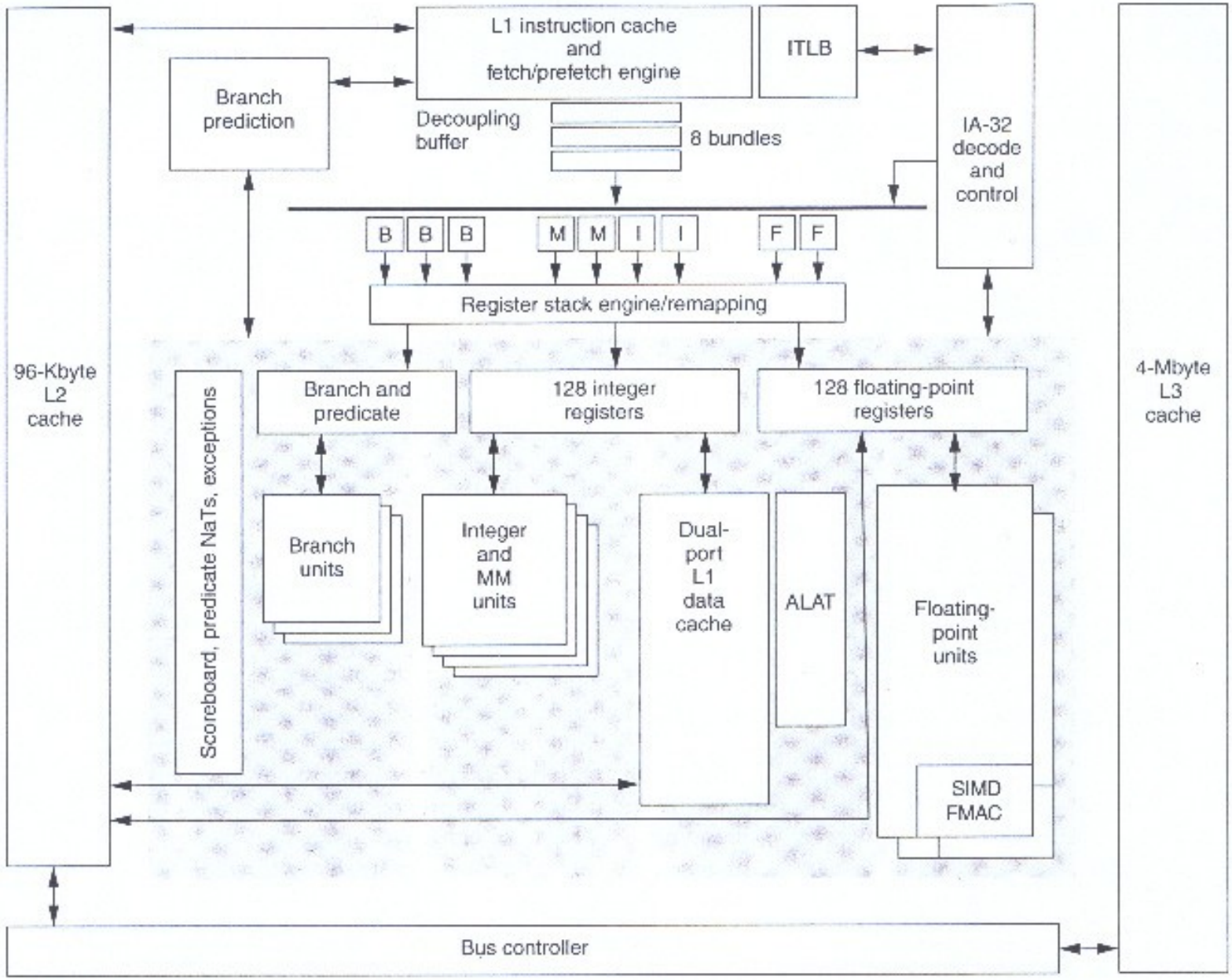
ROT: instruction rotation

pipelined access of the large register file:

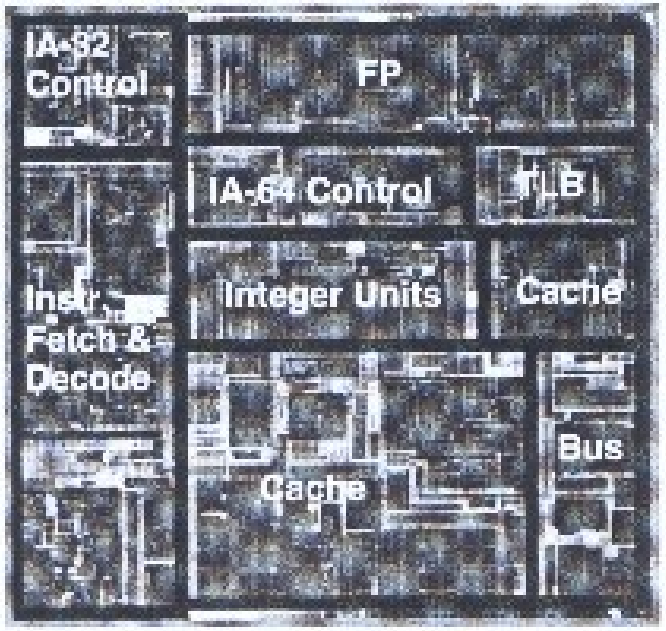
WDL: word line decode:

REG: register read

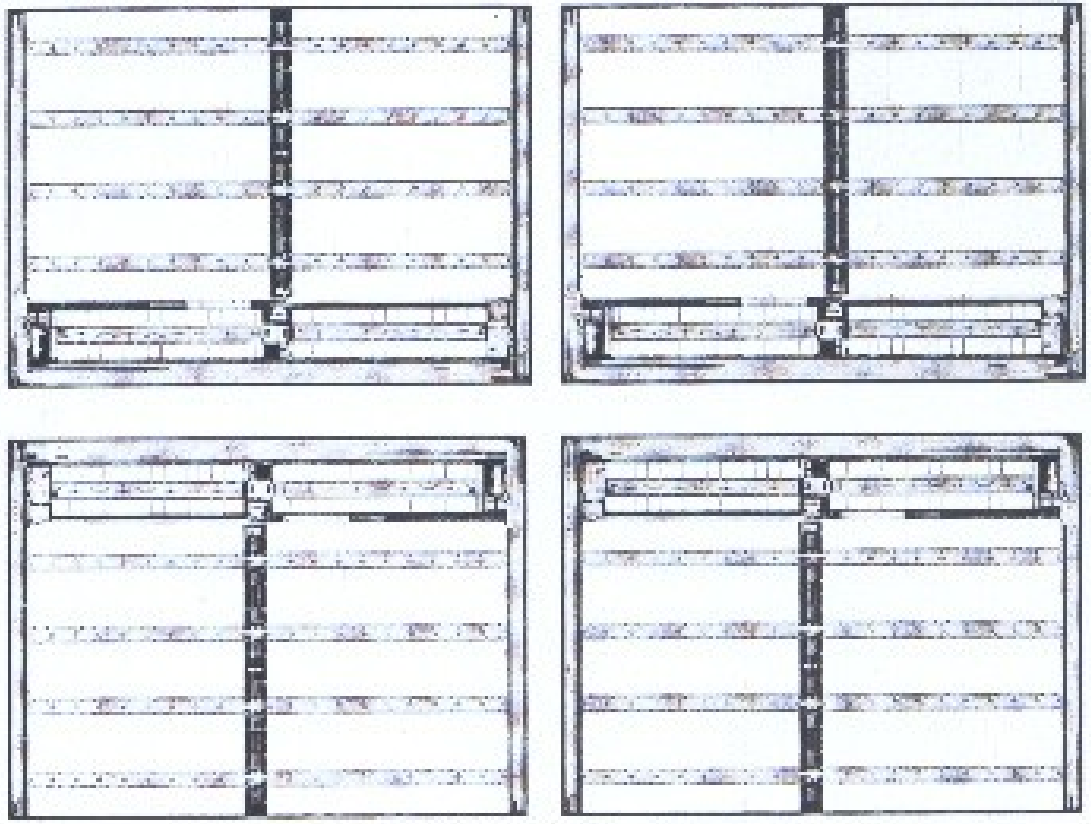
DET: exception detection (~retire stage)


L1 instruction cache and fetch/prefetch engine
ITLB
Branch prediction
Decoupling buffer
8 bundles
IA-32 decode and control
B B B M M I I F F
Register stack engine/remapping
96-Kbyte L2 cache
Scoreboard, predicate NaTs, exceptions
Branch and predicate
128 integer registers
128 floating-point registers
4-Mbyte L3 cache
Branch units
Integer and MM units
Dual-port L1 data cache
ALAT
Floating-point units
SIMD FMAC
Bus controller

Itanium Die



Core processor die



4 x 1Mbyte L3 cache