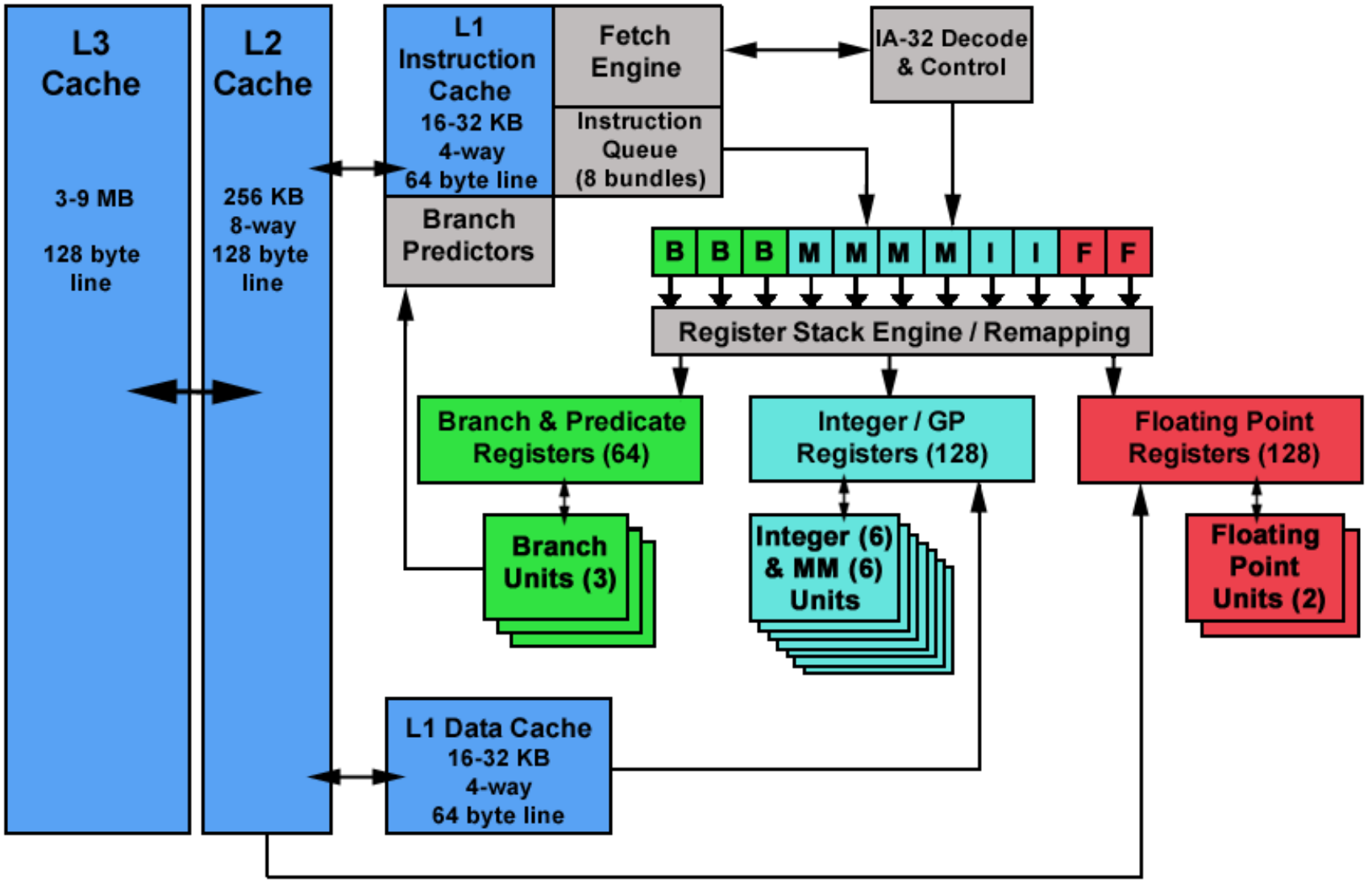
Itanium e Arquitetura IA64

Léo Willian Kölln
Renato Besen

Introdução

- Arquitetura de 64 bits
- Intel + HP
- Itanium, Itanium 2 e o futuro Itanium 3
- EPIC
- Suporte à IA32

IA64


L3 Cache
3-9 MB
128 byte line
L2 Cache
256 KB
8-way
128 byte line
L1 Instruction Cache
16-32 KB
4-way
64 byte line
Fetch Engine
Instruction Queue
(8 bundles)
Branch Predictors
IA-32 Decode & Control
B B B M M M M I I F F
Register Stack Engine / Remapping
Branch & Predicate Registers (64)
Integer / GP Registers (128)
Floating Point Registers (128)
Branch Units (3)
Integer (6) & MM (6) Units
Floating Point Units (2)
L1 Data Cache
16-32 KB
4-way
64 byte line

IA64

- EPIC
- Memória / Instruções
- Registradores de Estado, Pilha e 'Register Rotation'
- Predication
- Branches
- Comparações em Paralelo
- Interrupções

EPIC

- Explicity Paralel Instruction Computer
  - Instruction Level Paralelism
  - Complexidade do Hardware passa para o compilador
- Derivada do VLIW
- Bundles

Memória

- Endereçamento de 64 bits, com suporte a 32 bits
- Pode ser alinhada em 1, 2, 4, 8, 10 ou 16 bytes
- Campo 'be' do User Mask Register define little ou big endian

Instruções

- 6 tipos: A, I, M, F, B, L+X
- Empacotadas em 'bundles'
- Bundle dividido em 3 slots de 41 bits e um Template field, de 5 bits (128bits no total)
- Instruções extendidas ocupam 2 slots

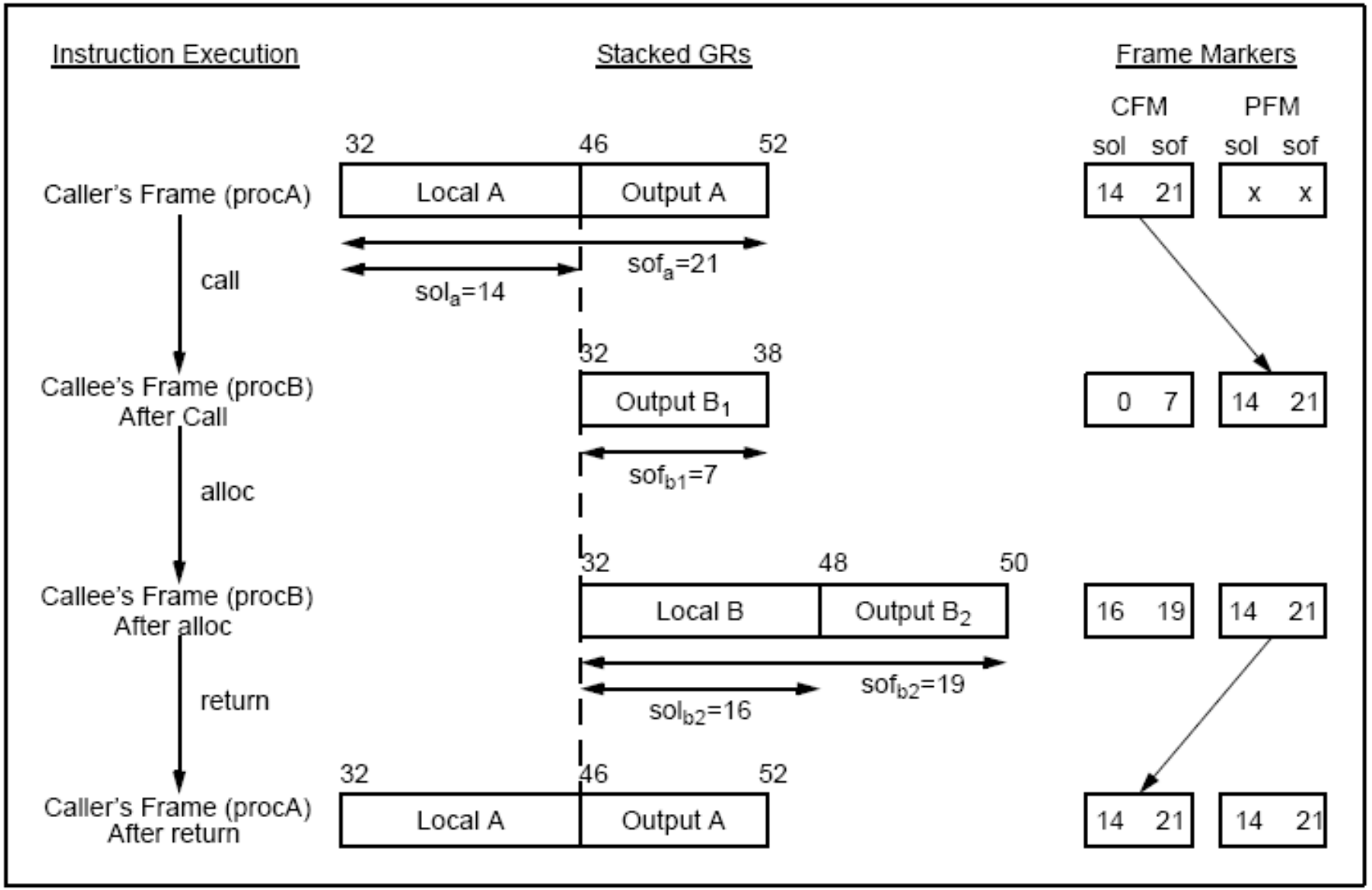
Registradores

- A IA64 possui um rico conjunto de registradores, podemos citar:
  - General Registers (GRs)
  - Floating-point Registers (FRs)
  - Predicate Registers (PRs)
  - Branch Registers (BRs)
  - Instruction Pointer (IP)
  - Current Frame Marker (CFM)
  - Previous Function State register (PFS)
  - Application Registers (ARs)
  - User Mask (UM)

Register Rotation e Register Stack

- Stack funciona alterando base para registrador inicial (Register Rotation)
  - br.call procB
    - CFM salvo no PSF
    - Tamanho local zerado, tamanho saída igual a saída de caller
  - Alocação de registradores (alloc)
  - br.ret
    - CMF restaurado a partir de PSF

Register Rotation e Register Stack



Predication

- Elimina branches condicionais
- Aumenta blocos básicos
- Dependências de Controle viram dependências de dados
- P0 - P63

Predication

```
// Antes
if (a > b) c = c * 5
else c = c / 3
// Depois
pT, pF = compare(a > b)
if (pT) c = c * 5
if (pF) c = c / 3
```

Branches

- Branches relativos ao IP
- 21 bits ou 60 bits
- Branch Hints

Branche Hints

- Estratégia
- Prefetch
- Deallocate

Multi-way Branchs

```
{ .bbb
(p1) br.cond target_1
(p2) br.cond target_2
(p3) br.call b1
}
```

4 possibilidades para a próxima instrução

Comparações em paralelo

if (a && b && c) {...}

Vira...

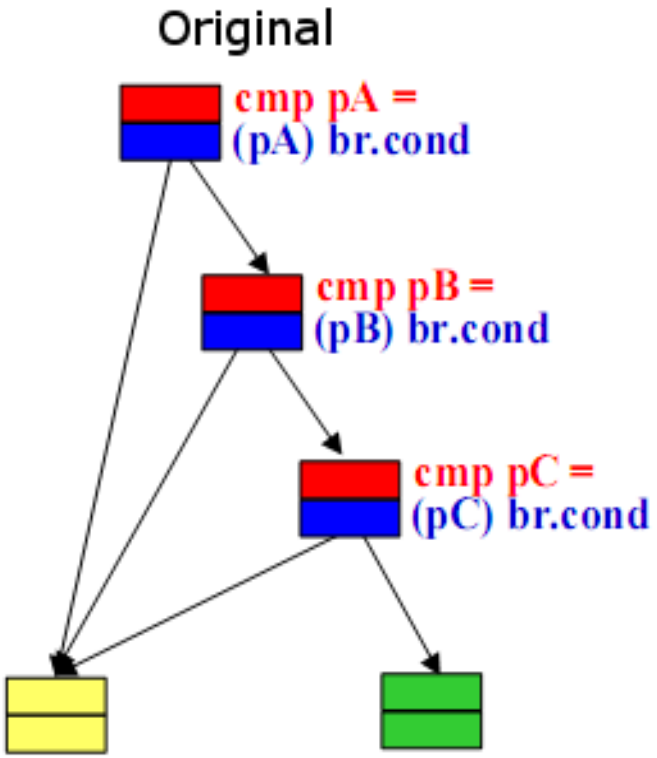
cmp.eq p1 = r0,r0 ;; // p1=1

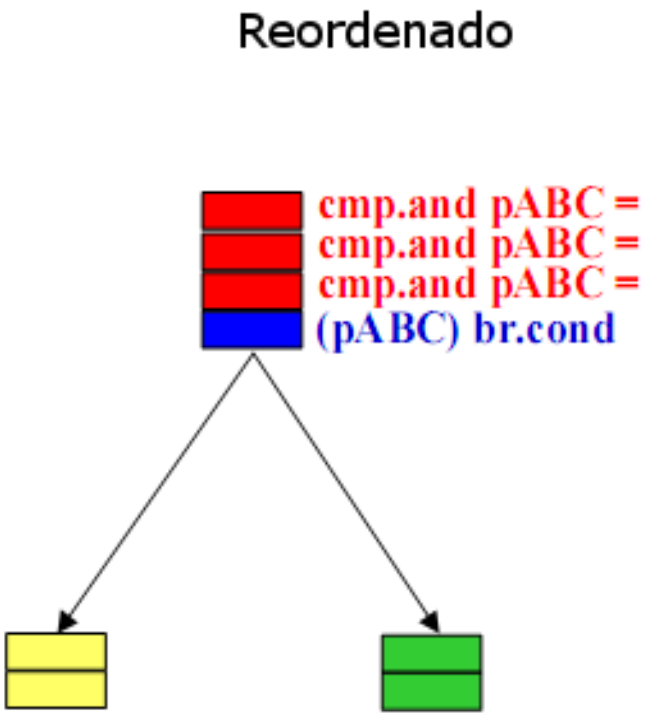
cmp.ne.and p1 = rA,0

cmp.ne.and p1 = rB,0

cmp.ne.and p1 = rC,0

Comparações em paralelo





Interrupções

- IVA-based
- PAL-based
- 4 grupos: aborts, interruptions, faults, traps
- Níveis de prioridade

Mercado

- Implementações
  - Itanium
    - Primeira implementação
    - Sem sucesso esperado
  - Itanium 2
    - Muitas melhorias, mais unidades funcionais
    - Melhor frequência e processo de fabricação
  - Itanium 2S
    - Aumento da cache, processo de fabricação
  - Itanium “3”
    - Adição SSE2, maior frequência, fabricação

Mercado

- Suporte de muitas empresas
  - Bull, Fujitsu, Fujitsu-Siemens Computers, Hitachi, HP, NEC, Silicon Graphics, Unisys, IBM, Dell
- Itanium Solutions Alliance
  - Promover arquitetura, acelerar porte de software
- Itanic
  - Apelido dado ao Itanium devido ao fracasso de sua primeira implementação.

IA64 x MIPS

- Finalidade: MIPS para sistemas embarcados, IA64 para servidores
- Paralelismo: MIPS no pipeline, IA64 no nível das instruções
- Registradores: MIPS tem 32 de uso geral, IA64 tem 128 de uso geral, 128 para float-point, 64 de predicados, ...
-

Códigos

- Multiplicação

```
(p0) mov r2=5 // Carrega imediato em inteiro
(p0) mov r3=2 // Carrega imediato em inteiro
;;
(p0) setf.sig f2=r2 // Move inteiro para flutuante
(p0) setf.sig f3=r3 // Move inteiro para flutuante
;;
(p0) xma.l    f8=f2,f3,f0
;;
(p0) getf.sig r8=f8
```

Códigos

- Salvamento de Contexto

```
procA:
...
(p0) br.call procB // Chama ProcB
...
procB:
(p0) st8.spill // Salva Static GRs
(p0) stf.spill // Salva FPs
... // EXECUCAO procB
(p0) stf.fill // Recupera FPs
(p0) st8.fill // Recupera Static GRs
(p0) br.ret // Volta para execução de procA
```