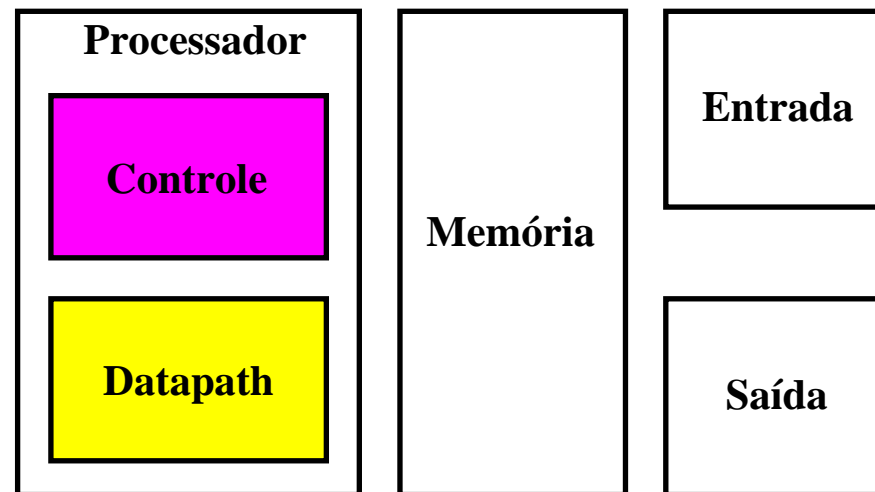


Pipelining avançado – Parte 2



Exemplo de emissão estática: IA-64

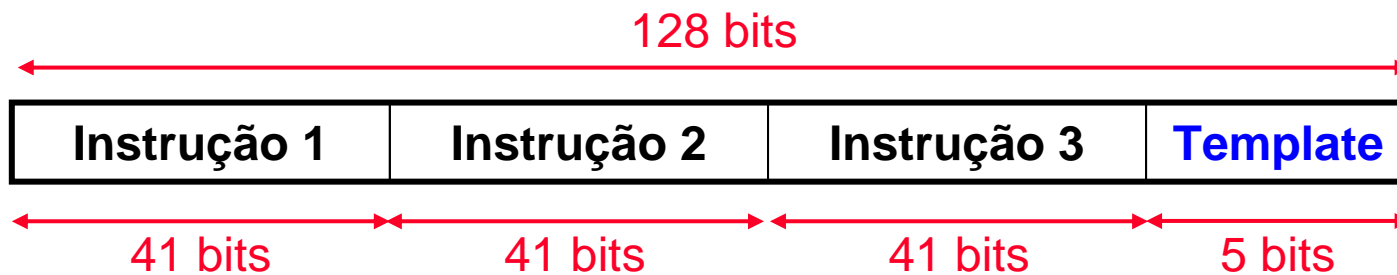
- Máquina load/store
- ISA com estilo RISC
- Suporte para exploração **explícita** de ILP
- Papel do compilador
 - Identifica e aproveita o paralelismo
 - Codifica o paralelismo no formato de instrução
- EPIC
 - “Explicit Parallel Instruction Computer”

Exemplo de emissão estática: IA-64

- **Diferenças entre MIPS e IA-64**
 - **IA-64 tem mais registradores**
 - » 128 para inteiros; 128 para ponto flutuante
 - » 8 registradores especiais para desvios
 - » 64 registradores de 1 bit para predicados
 - **IA-64 empacota instruções em “bundles”**
 - » Formato fixo
 - » Codificação explícita de dependências

Exemplo de emissão estática: IA-64

- “Bundle”



Exemplo de emissão estática: IA-64

- “Bundle”

Instrução 1	Instrução 2	Instrução 3	Template
-------------	-------------	-------------	----------

```
{    .mii
    ld4    r1 = [r5]
    sub    r3 = r7, r8
    add    r2 = r5, r6
}
```

Exemplo de emissão estática: IA-64

- “Bundle”

Instrução 1	Instrução 2	Instrução 3	Template
-------------	-------------	-------------	----------

```
{  
    .mii  
    ld4    r1 = [r5]      // mapeia para M0  
    sub    r3 = r7, r8    // mapeia para I0  
    add    r2 = r5, r6    // mapeia para I1  
}
```

Compilador determina em que
UF cada instrução executa



Exemplo de emissão estática: IA-64

- “Bundle”

ld4 r1 = [r5]	sub r3 = r7, r8	add r2 = r5, r6	00000
---------------	-----------------	-----------------	-------

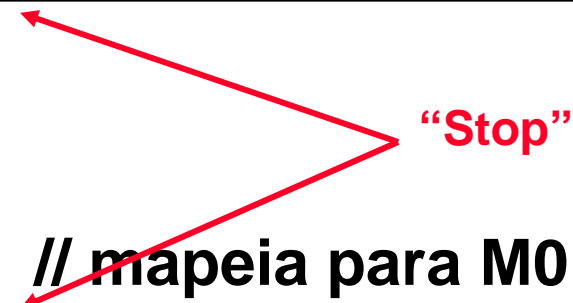
```
{  
    .mii  
    ld4    r1 = [r5]    // mapeia para M0  
    sub    r3 = r7, r8  // mapeia para I0  
    add    r2 = r5, r6  // mapeia para I1  
}
```

Exemplo de emissão estática: IA-64

- “Bundle”

ld4 r1 = [r5]	sub r3 = r7, r8	add r2 = r5, r6	00010
---------------	-----------------	-----------------	-------

{
 .mii
 ld4 r1 = [r5] // mapeia para M0
 sub r3 = r7, r8;; // mapeia para I0
 add r2 = r5, r6 // mapeia para I1
}

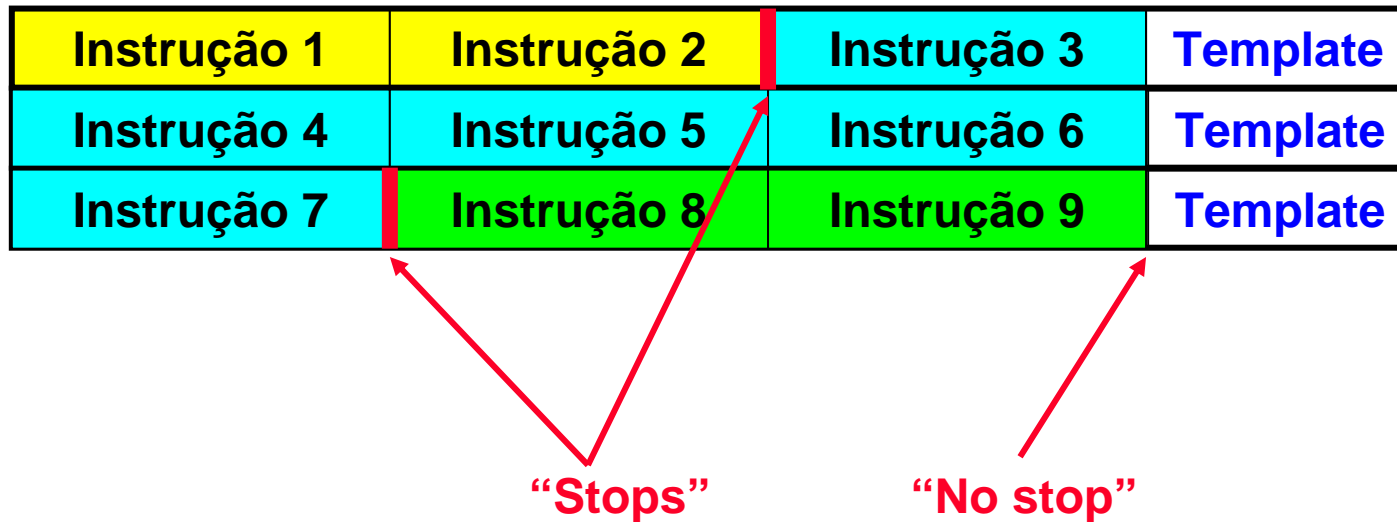


Exemplo de emissão estática: IA-64

Instrução 1	Instrução 2	Instrução 3	Template
Instrução 4	Instrução 5	Instrução 6	Template
Instrução 7	Instrução 8	Instrução 9	Template

Exemplo de emissão estática: IA-64

- “Instruction group”



Exemplo de emissão estática: IA-64

- “Instruction group”

Instrução 1	Instrução 2	Instrução 3	00010
Instrução 4	Instrução 5	Instrução 6	00000
Instrução 7	Instrução 8	Instrução 9	01010

A posição de um stop é codificada como parte do template
(onde também é codificado o mapeamento para UF)

O paralelismo entre as instruções é codificado explicitamente!
(EPIC)

Exemplo de emissão estática: IA-64

- Suporte para **predicação**
- Conceito
 - Execução condicional de uma instrução
 - Dependendo de um predicado
- Conseqüência
 - Eliminação de desvios condicionais
 - » Redução de hazards de controle
- Exemplo: **if (p) {S1} else {S2}**
 - (p) S1
 - (~p) S2

IA-64: exemplo de predicação

- Exemplo: código fonte

```
x = A[i];  
if (y > 0) y = k + w;  
k = x - z;
```

- Exemplo: código assembly

```
{  
    .mii  
    ld4    r20=[r22]  
(pr1)    add    r2 = r3, r6 ;;  
    sub    r3 = r20, r8  
}
```

ILP: Especulação

- **Permite compilador ou HW “adivinhar”**
 - **Propriedades de uma instrução**
 - » **Resultado de um desvio condicional**
 - » **Se um load depende de um store**
- **Desvios**
 - **Executar o desvio**
 - **Antes de ser conhecido o resultado do teste**
- **Load**
 - **Executar o load**
 - **Antes de saber se o store tem mesmo endereço**

Especulação: desvio

if (A==0) A = B; else A = A + 4;

Hipótese: $A \rightarrow 0(\$s3)$; $B \rightarrow 0(\$s2)$

Previsão: desvio tomado

gera
hazard

```
lw    $s1, 0($s3)
bne   $s1, $zero, L1
lw    $s1, 0($s2)
j     L2
```

L1: addi \$s1, \$s1, 4

L2: sw \$s1, 0(\$s3)

Solução: armazenar o
resultado provisoriamente

em um buffer

**Novo valor de \$s1 não
pode ser escrito no banco
de registradores**

```
lw    $s1, 0($s3)
addi   $s1, $s1, 4
bne   $s1, $zero, L3
lw    $s1, 0($s2)
L3:   sw    $s1, 0($s3)
```

**Só atualizar o banco de
registradores se hipótese
resultar verdadeira**

Especulação: load

```
sw    $s1, 0($s3)
...
lw    $s2, 4($s4)
```



- Pode-se executar o lw antes do sw ?
 - Se $0 + \$s3 = 4 + \$s4$, são dependentes
 - » Não se pode reordená-las
- Pode-se fazê-lo desde que:
 - O valor lido pelo lw não seja escrito em \$s2
 - Seja armazenado em buffer
 - Até que os endereços efetivos sejam conhecidos

Emissão múltipla dinâmica

- **Superscalar**
- **Instruções emitidas em ordem**
 - CPU decide número de instruções emitidas
 - » Nenhuma, uma ou mais
- **Compilador favorece**
 - Reordenando instruções
 - Para aumentar chances de emissão múltipla

Emissão múltipla dinâmica

- **Migração entre implementações**
 - Código herdado (“legacy code”)
- **VLIW requer recompilação**
 - Para garantir execução correta ou
 - Para assegurar desempenho aceitável
- **E o superscalar?**
- **Superscalar garante execução correta**
 - Independentemente do código gerado
 - » Detecção de hazards e pausas
- **Superscalar garante execução eficiente**
 - Mesmo sem recompilação
 - » Escalonamento dinâmico

Pipeline: escalonamento dinâmico

- **Motivação**

`lw $t0, 20($s2)`

`addu $t1, $t0, $t2`

`sub $s4, $s4, $t3`

`slti $t5, $s4, 20`

Pode se revelar muito lenta
em tempo de execução

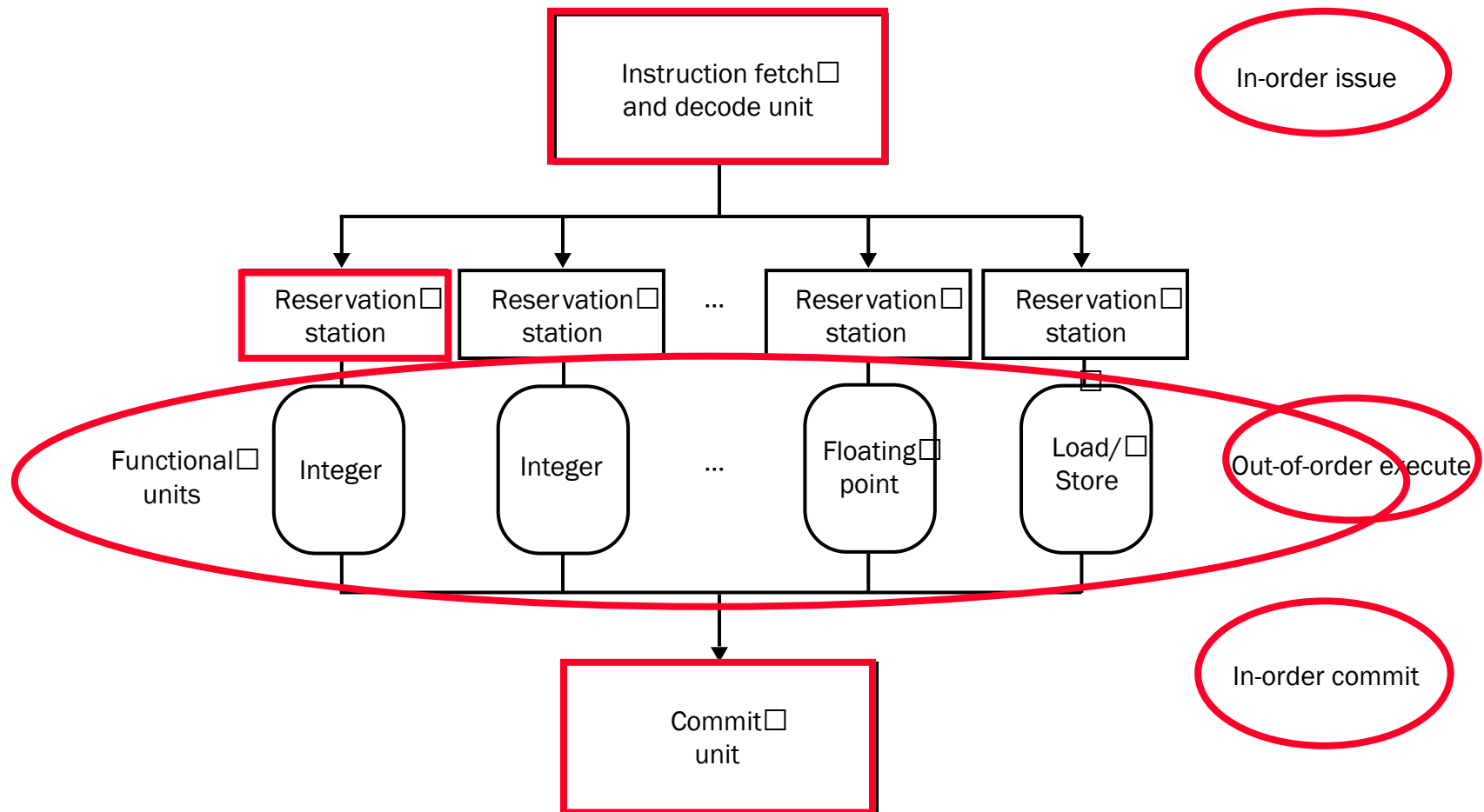
Não depende de `lw` nem
de `addu`, mas não pode
iniciar execução antes
delas.

A instrução `sub` poderia completar antes das duas primeiras

Isto requer **execução fora de ordem** (“out-of-order” execution)

Pipeline com escalonamento dinâmico

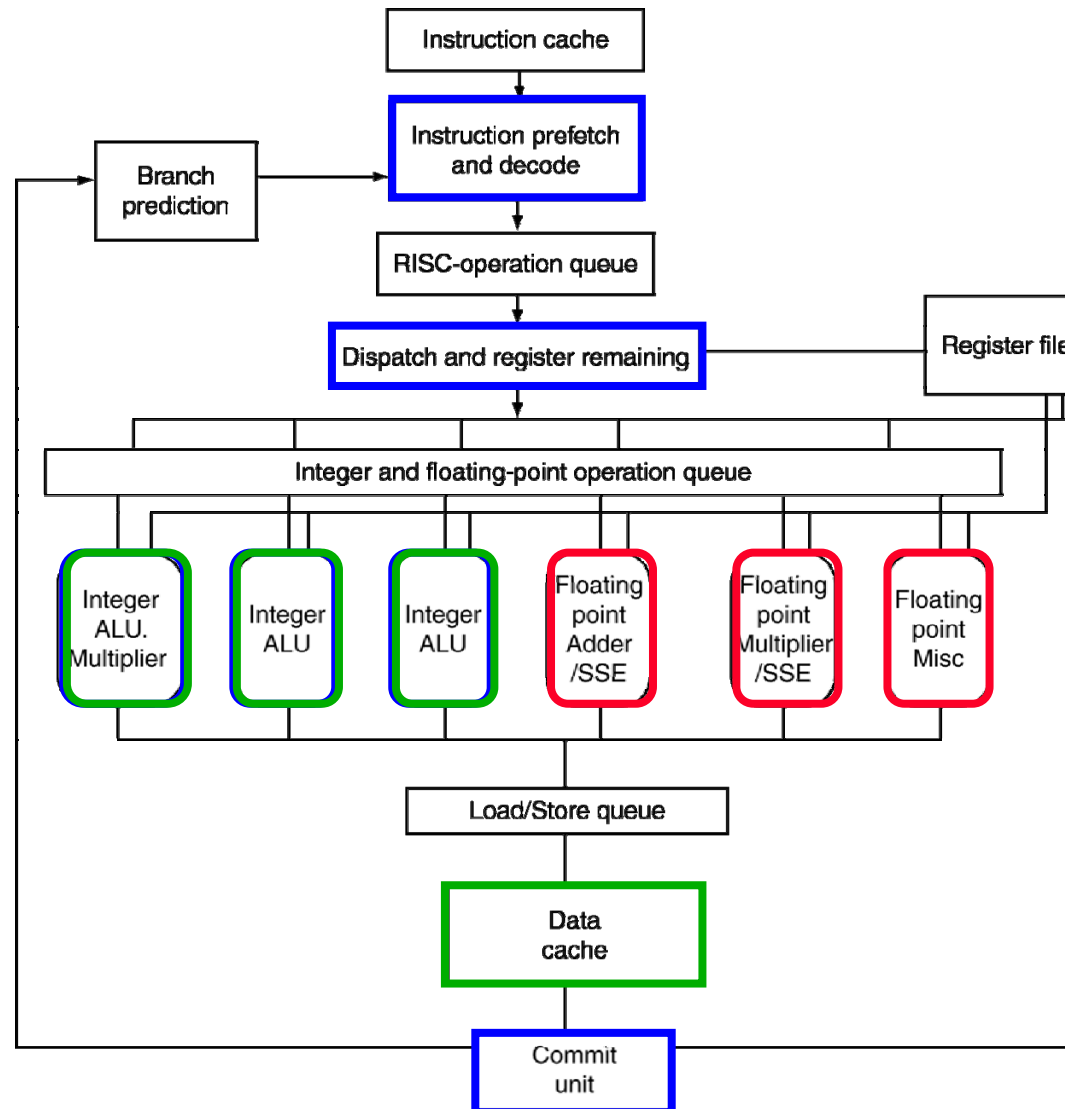
Pipeline: escalonamento dinâmico



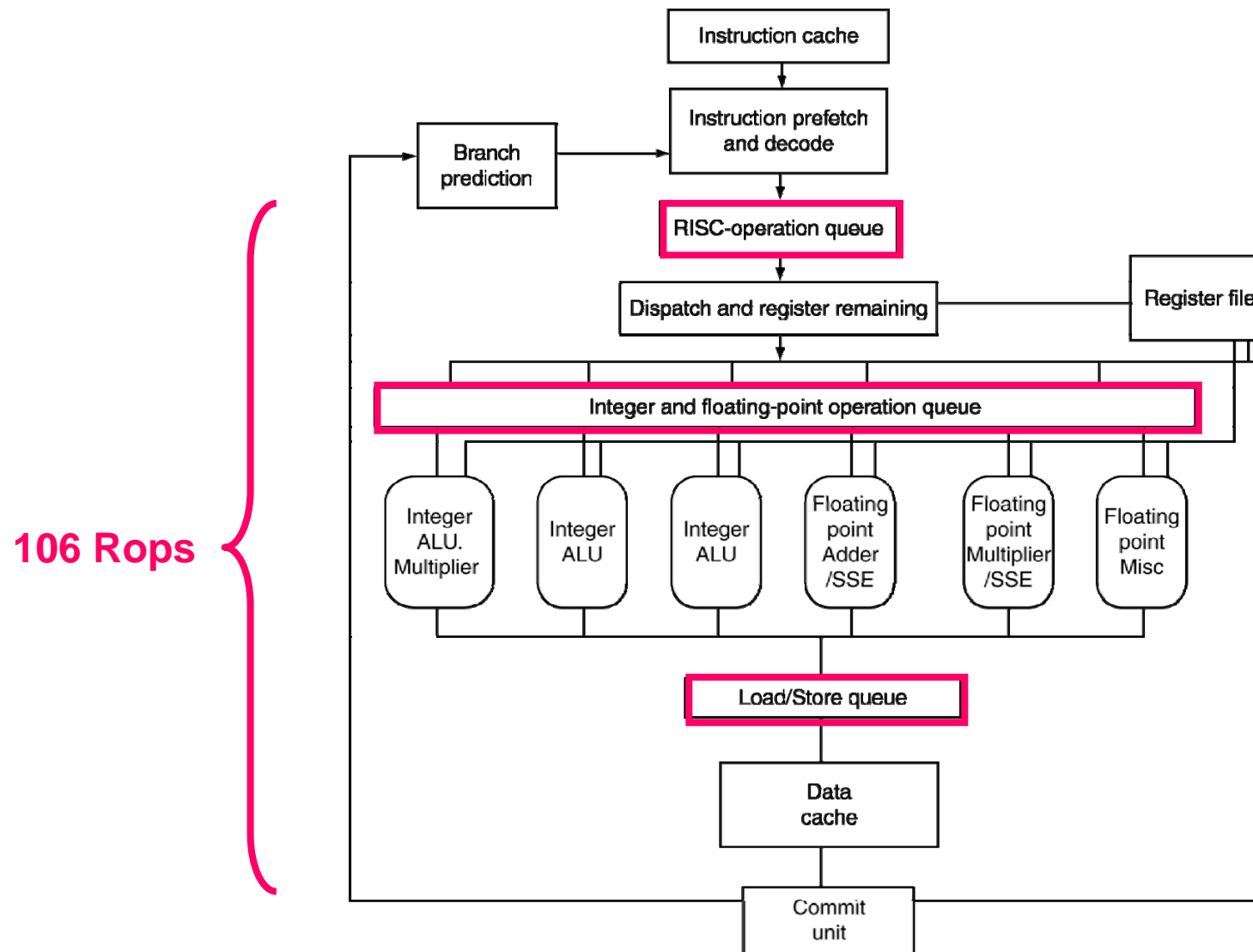
Exemplo: AMD Opteron X4 (Barcelona)

- Reformata instruções x86 (CISC)
 - Instruções tornam-se *RISC-operations* (AMD)
 - » *Micro-operations* (Intel)
- Superscalar
 - Emissão múltipla dinâmica → minimizar CPI
 - » 3 *RISC-operations* por ciclo
 - Pipeline profundo → maximizar f
 - » 12 estágios (int), 17 estágios (ponto flutuante)
- Pipeline com escalonamento **dinâmico**
 - Executa as micro-operações
 - » Execução **especulativa**
 - » Execução fora de ordem

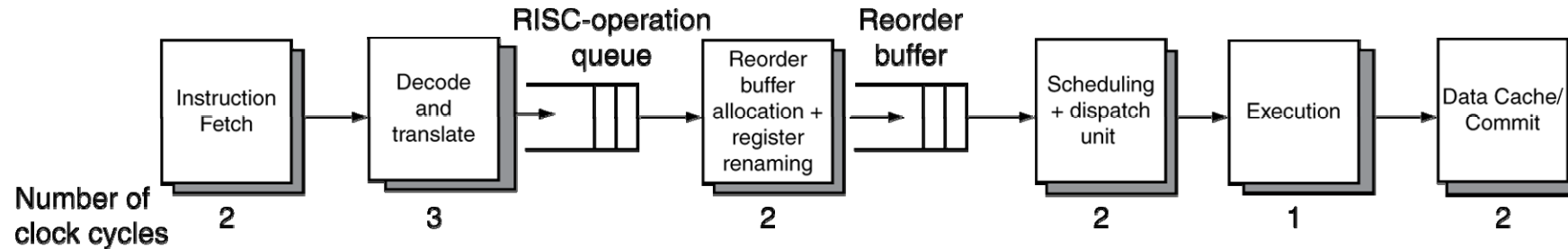
AMD Opteron X4: micro-arquitetura



AMD Opteron X4: micro-arquitetura



AMD Opteron X4: pipeline (inteiro)



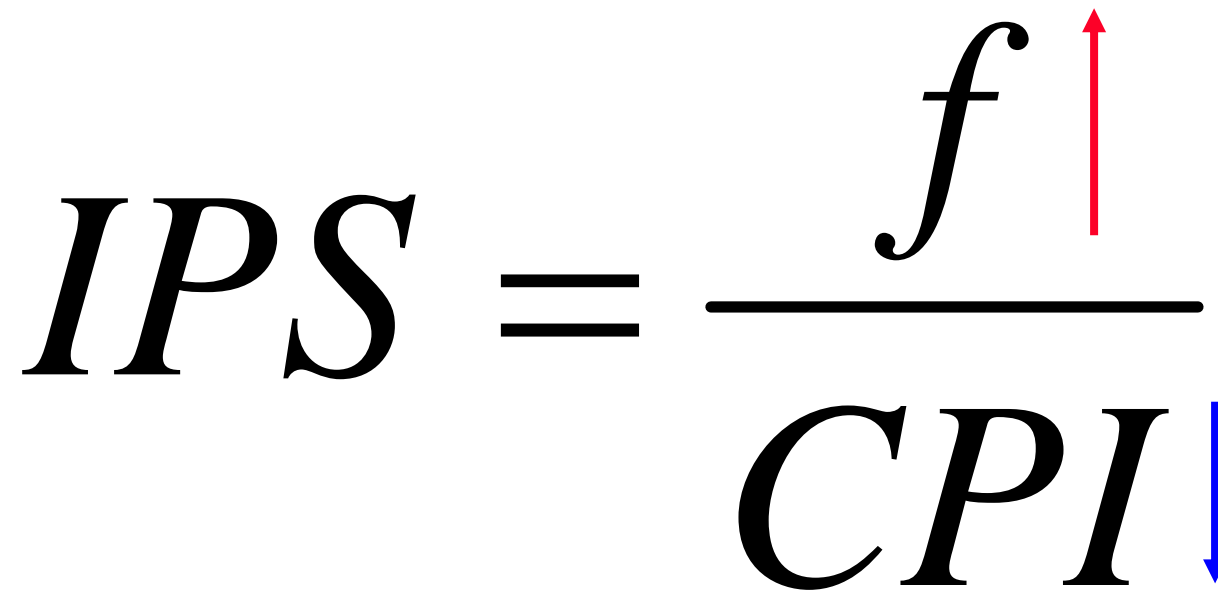
- **Idéia-chave:** minimizar latência entre instruções para diminuir impacto das dependências de dados.
- **Gargalos de desempenho:**
 - Instruções x86 complexas não mapeiam para poucas Rops
 - Desvios de difícil previsão
 - » Má-previsão: pausas ou recomeços (especulação)
 - Longas dependências
 - » Instruções demoradas ou falta na cache
 - Atrasos no acesso à memória
 - » Penalidade de falta na cache

Discussão e conclusão

- **O porquê do escalonamento dinâmico ?**
- **Por que não deixar isso para o compilador?**
- **Algumas pausas não são previsíveis**
 - Exemplo: faltas na cache
- **HW faz previsão dinâmica de desvios**
 - Para especulação dinâmica
 - Não faz sentido sem escalonamento dinâmico
- **Migração entre implementações**
 - Código velho roda eficientemente na máquina nova

Desempenho: idéias-chave até 2004

Miniaturização e superpipeling

$$IPS = \frac{f}{CPI}$$


Emissão múltipla, escalonamento (dinâmico e estático),
especulação, previsão de desvios (estática e dinâmica),
memórias cache

Pipelines, cores e potência

Microprocessor	Year	Clock Rate (MHz)	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power (W)
Intel 486	1989	25	5	1	No	1	5
Intel Pentium	1993	66	5	2	No	1	10
Intel Pentium Pro	1997	200	10	3	Yes	1	29
Intel Pentium Willamette	2001	2000	22	3	Yes	1	75
Intel Pentium Prescott	2004	3600	31	3	Yes	1	103
Intel Core	2006	2660	14	4	Yes	2	75
Sun UltraSPARC III	2003	1950	14	4	No	1	90
Sun UltraSPARC T1 (Niagara)	2005	1200	6	1	No	8	70

10 vezes em 4 anos

1,8 vezes nos 3 anos seguintes

**Limite compatível com
dissipação
economicamente viável**

**Profundidade do
pipeline correlata
com a frequência**

**Potência correlata
com frequência**

Pipelines, cores e potência

Microprocessor	Year	Clock Rate (MHz)	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power (W)
Intel 486	1989	25	5	1	No	1	5
Intel Pentium	1993	66	5	2	No	1	10
Intel Pentium Pro	1997	200	10	3	Yes	1	29
Intel Pentium Willamette	2001	2000	22	3	Yes	1	75
Intel Pentium Prescott	2004	3600	31	3	Yes	1	103
Intel Core	2006	2930	14	4	Yes	2	75
Sun UltraSPARC III	2003	1950	14	4	No	1	90
Sun UltraSPARC T1 (Niagara)	2005	1200	6	1	No	8	70

Emissão múltipla ou simples?

Escalonamento dinâmico ou não?

Volta ao pipeline com emissão simples e poucos estágios

Transição para multicore

Redução no número de estágios

Mesma potência com 4 vezes mais cores

Pipelines, cores e potência

Microprocessor	Year	Clock Rate (MHz)	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power (W)
Intel 486	1989	25	5	1	No	1	5
Intel Pentium	1993	66	5	2	No	1	10
Intel Pentium Pro	1997	200	10	3	Yes	1	29
Intel Pentium Willamette	2001	2000	22	3	Yes	1	75
Intel Pentium Prescott	2004	3600	31	3	Yes	1	103
Intel Core	2006	2930	14	4	Yes	2	75
Sun UltraSPARC III	2003	1950	14	4	No	1	90
Sun UltraSPARC T1 (Niagara)	2005	1200	6	1	No	8	70

Conclusão 1: as técnicas de **ILP** continuarão minimizando a ociosidade de um dado core, enquanto técnicas de **TLP** buscarão compensar a menor frequência (e o menor número de instruções emitidas por ciclo, quando for o caso)

Pipelines, cores e potência

Microprocessor	Year	Clock Rate (MHz)	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power (W)
Intel 486	1989	25	5	1	No	1	5
Intel Pentium	1993	66	5	2	No	1	10
Intel Pentium Pro	1997	200	10	3	Yes	1	29
Intel Pentium Willamette	2001	2000	22	3	Yes	1	75
Intel Pentium Prescott	2004	3600	31	3	Yes	1	103
Intel Core	2006	2930	14	4	Yes	2	75
Sun UltraSPARC III	2003	1950	14	4	No	1	90
Sun UltraSPARC T1 (Niagara)	2005	1200	6	1	No	8	70

Conclusão 2: o limite à dissipação de potência tornou crucial a **Programação Concorrente, mas há ainda muitas dificuldades para sincronização, consistência de memória e depuração de software paralelo**