

## Aula 11: Previsão Dinâmica de Branches, Superescalar, VLIW, e *Software Pipelining*

## Revisão do Algoritmo de Tomasulo

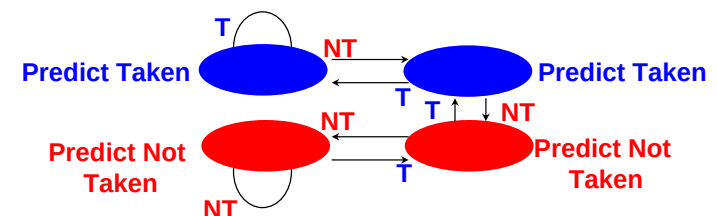
- Registradores não são gargalo
- Evita hazards de WAR, WAW e hazards no Scoreboard
- Não está limitado a blocos básicos (desde que tenhamos previsão de branches)
- Permite *loop unrolling* em HW
- Contribuições que existem até hoje
  - Escalonamento dinâmico
  - *Register renaming*
  - Separação entre Loads e Stores

## Previsão Dinâmica de Branches

- Performance =  $f(\text{precisão}, \text{custo de previsão errada})$
- **Branch History Table** é a mais simples
  - LSBs do endereço dado pelo PC endereça tabela de valores de 1 bi
  - Indica se branch naquela entrada da tabela foi tomado ou não
- Problema: em um loop, BHT de 1 bit causará duas previsões erradas:
  - Final do loop, quando o branch não é tomado
  - Primeira vez da iteração seguinte, quando o branch da última vez não foi tomado

## Previsão Dinâmica de Branches

- Solução: Utilização de 2-bits, onde mudança de previsão ocorre somente se previsão errada ocorre duas vezes: (Figura 4.13, p. 264)

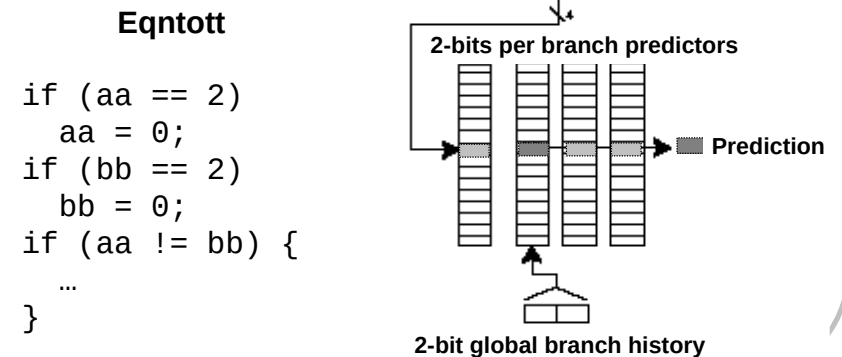


## Precisão do BHT

- Previsão errada ocorre em dois casos:
  - Previsão errada para esse branch
  - Leu da tabela previsão para branch errado
- Tabela com 4096 entradas
  - Programas variam de 1% de erro (nasa7, tomcatv) para 18% (eqntott), com spice em 9% e gcc em 12%
- Tabela com 4096 entradas é tão boa quanto uma com um número infinito de entradas, mas requer muito hardware
- Contador com 2 bits é tão bom quanto um contador com um número infinito de bits

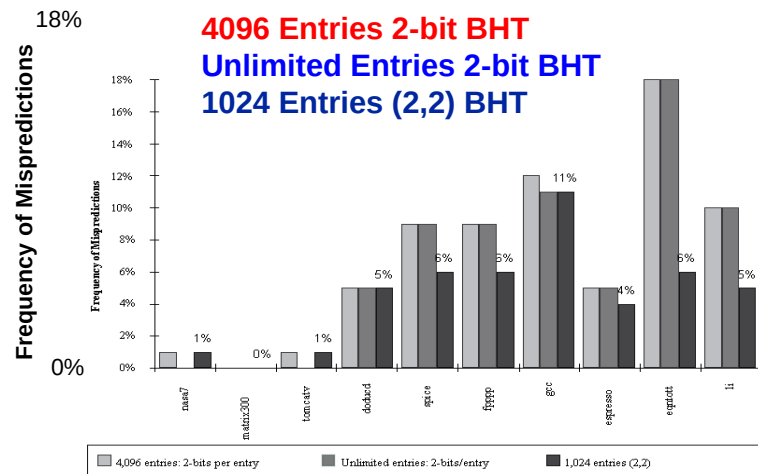
## Branches Correlacionados

Idéia: tomado/não tomado dos branches executados recentemente está relacionado com o comportamento do branch (tão relacionado quanto o histórico deste branch)



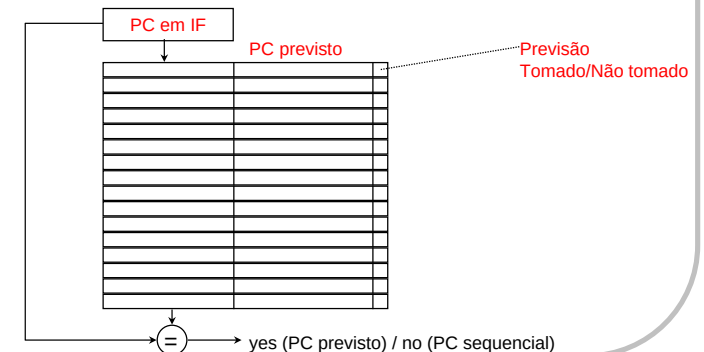
## Precisão dos Esquemas Diferentes

(Figura 4.21, p. 272)



## DLX Precisa do Endereço ao Mesmo Tempo que Previsão

- Branch Target Buffer (BTB):** Endereço do índice de branch busca previsão E endereço (se branch for tomado) ao mesmo tempo
  - Nota: Precisamos checar agora se o branch é realmente aquele na tabela, porque não podemos usar o endereço errado. Portanto, precisamos checar o endereço (**Figura 4.22, p. 273**)



## Conseguindo CPI < 1: Disparando Múltiplas Instruções/Ciclo

- Duas Variações
- Superescalar: variando o número de instruções/ciclo (1 a 8), escalonadas pelo compilador ou por hardware (Tomasulo)
  - IBM PowerPC, Sun SuperSparc, DEC Alpha, HP 7100
- *Very Long Instruction Words (VLIW)*: número de instruções fixas (16) escalonadas pelo compilador
  - HP/Intel para 1998?

## Conseguindo CPI < 1: Disparando Múltiplas Instruções/Ciclo

- DLX superescalar: 2 instruções, 1 FP & 1 inteira, load ou store
  - Busca de 64-bits/clock; Int à esquerda, FP à direita
  - Segunda instrução dispara somente se primeira instrução é disparada
  - Mais portos para registradores FP para fazer load/store FP

| <i>Tipo</i> | <i>Estágios</i> |    |    |     |     |     |    |
|-------------|-----------------|----|----|-----|-----|-----|----|
| instr. Int  | IF              | ID | EX | MEM | WB  |     |    |
| instr. FP   | IF              | ID | EX | MEM | WB  |     |    |
| instr. Int  |                 | IF | ID | EX  | MEM | WB  |    |
| instr. FP   |                 | IF | ID | EX  | MEM | WB  |    |
| instr. Int  |                 |    | IF | ID  | EX  | MEM | WB |
| instr. FP   |                 |    | IF | ID  | EX  | MEM | WB |

- 1 ciclo de load delay expande para **3 instruções** em SS
  - instrução à direita não pode usá-la, nem podem instrs. no próx. slot

## Unrolled Loop Minimiza Stalls para Superescalar

|         |      |                         |                     |
|---------|------|-------------------------|---------------------|
| 1 Loop: | LD   | F0, 0(R1)               | LD to ADDD: 1 Ciclo |
| 2       | LD   | F6, -8(R1)              | ADDD to SD: 2 Ciclo |
| 3       | LD   | F10, -16(R1)            |                     |
| 4       | LD   | F14, -24(R1)            |                     |
| 5       | ADDD | F4, F0, F2              |                     |
| 6       | ADDD | F8, F6, F2              |                     |
| 7       | ADDD | F12, F10, F2            |                     |
| 8       | ADDD | F16, F14, F2            |                     |
| 9       | SD   | 0(R1), F4               |                     |
| 10      | SD   | -8(R1), F8              |                     |
| 11      | SD   | -16(R1), F12            |                     |
| 12      | SUBI | R1, R1, #32             |                     |
| 13      | BNEZ | R1, LOOP                |                     |
| 14      | SD   | 8(R1), F16 ; 8-32 = -24 |                     |

14 ciclos, ou 3.5 ciclos por iteração

## Loop Unrolling em Superescalar

|       | Instr. Inteira | Instr. FP       | Ciclos |
|-------|----------------|-----------------|--------|
| Loop: | LD F0,0(R1)    |                 | 1      |
|       | LD F6,-8(R1)   |                 | 2      |
|       | LD F10,-16(R1) | ADDD F4,F0,F2   | 3      |
|       | LD F14,-24(R1) | ADDD F8,F6,F2   | 4      |
|       | LD F18,-32(R1) | ADDD F12,F10,F2 | 5      |
|       | SD 0(R1),F4    | ADDD F16,F14,F2 | 6      |
|       | SD -8(R1),F8   | ADDD F20,F18,F2 | 7      |
|       | SD -16(R1),F12 |                 | 8      |
|       | SD -24(R1),F16 |                 | 9      |
|       | SUBI R1,R1,#40 |                 | 10     |
|       | BNEZ R1,LOOP   |                 | 11     |
|       | SD -32(R1),F20 |                 | 12     |

- **Unrolled 5 vezes para evitar atrasos (+1 devido ao SS)**
  - 12 ciclos, ou 2.4 clocks por iteração

## Escalonamento Dinâmico em Superescalares

- Dependências para o disparo
- Código compilado para versão escalar executa inadequadamente em SS
  - Precisamos manter compatibilidade
- Idéia simples: separar controle de Tomasulo para *reservation stations* das unidades Inteira e de FP

## Escalonamento Dinâmico em Superescalares

- Como disparar 2 instruções e manter disparo em ordem para Tomasulo?
  - Issue executa 2X mais rápido que o resto da máquina, portanto, issue é mantido em ordem
  - Somente loads FP podem causar dependência entre issue de unidade inteira e FP:
    - » Troca *reservation station* de load por fila; operandos tem que ser lidos na ordem em que são buscados
    - » Verifique endereços de Loads com endereços de Store na fila de Store para evitar violação de RAW
    - » Verifique endereços de Store na fila de Loads e Stores para evitar WAR e WAW

## Performance de Escalonamento Dinâmico em SS

| Iteração no. | Instrs.       | Issues | Executa ciclo | Escreve |
|--------------|---------------|--------|---------------|---------|
| 1            | LD F0,0(R1)   | 1      | 2             | 4       |
| 1            | ADDD F4,F0,F2 | 1      | 5             | 8       |
| 1            | SD 0(R1),F4   | 2      | 9             |         |
| 1            | SUBI R1,R1,#8 | 3      | 4             | 5       |
| 1            | BNEZ R1,LOOP  | 4      | 5             |         |
| 2            | LD F0,0(R1)   | 5      | 6             | 8       |
| 2            | ADDD F4,F0,F2 | 5      | 9             | 12      |
| 2            | SD 0(R1),F4   | 6      | 13            |         |
| 2            | SUBI R1,R1,#8 | 7      | 8             | 9       |
| 2            | BNEZ R1,LOOP  | 8      | 9             |         |

- 4 por iteração

Branches, Decrementos ainda levam 1 ciclo

## Limites de Processadores Superescalares

- Enquanto quebra de unidades Inteiras/FP é simples em HW, só conseguimos CPI de 0,5 para programas com:
  - 50% de operações Inteiras de FP
  - Nenhum hazard
- Se mais instruções disparam ao mesmo tempo, é mais difícil fazer decodificação e issue
  - Até para 2-scalar => exame de 2 opcodes, 6 registradores, & decidir se 1 ou 2 instruções podem ser disparadas
- VLIW: decodificação vs. tamanho da instrução
  - Há espaço no código da instr. para diversas FUs
  - Operações definidas pelo compilador para executar na mesma palavra podem executar em paralelo
  - Ex., 2 operações inteiras, 2 operações FP, 2 refs. memória, 1 branch
    - » 16 a 24 bits por campo => 112 bits a 168 bits de tamanho
  - Precisa escalonar código através de branches para ser efetivo

## Loop Unrolling em VLIW

| Memória 1      | Memória 2      | FP 1            | FP 2            | Int/<br>branch | Clock |
|----------------|----------------|-----------------|-----------------|----------------|-------|
| LD F0,0(R1)    | LD F6,-8(R1)   |                 |                 |                | 1     |
| LD F10,-16(R1) | LD F14,-24(R1) |                 |                 |                | 2     |
| LD F18,-32(R1) | LD F22,-40(R1) | ADDD F4,F0,F2   | ADDD F8,F6,F2   |                | 3     |
| LD F26,-48(R1) |                | ADDD F12,F10,F2 | ADDD F16,F14,F2 |                | 4     |
|                |                | ADDD F20,F18,F2 | ADDD F24,F22,F2 |                | 5     |
| SD 0(R1),F4    | SD -8(R1),F8   | ADDD F28,F26,F2 |                 |                | 6     |
| SD -16(R1),F12 | SD -24(R1),F16 |                 |                 |                | 7     |
| SD -32(R1),F20 | SD -40(R1),F24 |                 |                 | SUBI R1,R1,#48 | 8     |
| SD -0(R1),F28  |                |                 |                 | BNEZ R1,LOOP   | 9     |

*Unrolled 7 vezes para evitar atrasos*

- 7 executadas em 9 ciclos, ou 1.3 ciclos por iteração
- Precisa de mais registradores em VLIW

## Limitações para Máquinas *Multi-Issue*

- Limitações inerentes de ILP
  - 1 branch em 5 instruções => como manter um 5-way VLIW ocupado?
  - Latência das unidades => muitas operações precisam ser escalonadas
- Dificuldades na construção do HW
  - Duplicação das FUs para conseguir execução em paralelo
  - Aumento do número de portos no banco de registradores (VLIW do exemplo precisa de 6 portos de leitura e 3 de escrita para inteiros, e 6 de leitura e 4 de escrita para registradores de FP)
  - Aumento do número de portos para memória
  - Decodificação de SS afeta frequência da máquina, e profundidade do pipeline

## Limitações para Máquinas *Multi-Issue*

- Limitações específicas para SS ou VLIW
  - Lógica de decodificação e issue em SS
  - Tamanho do código para VLIW: desenrolamento de loops + campos não utilizados em VLIW
  - Travamento de VLIW => 1 hazard => todas as instruções param
  - VLIW compatibilidade de código => quase nula

## Processadores Existentes no Mercado

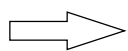
|               | MIPS R10K | Ultra SPARC | Alpha 21164 | Intel P6 | HPPA 8K | PowerPC 620 |
|---------------|-----------|-------------|-------------|----------|---------|-------------|
| Clock         | 200 Mhz   | 167 Mhz     | 300 Mhz     | 133 Mhz  | 200Mhz  | 133 Mhz     |
| Taxa de Issue | 4         | 4           | 4           | 3        | 4       | 4           |
| out-of-order  | 32        | 0           | 6 lds       | 40       | -       | 16          |
| Fus           | 6         | 4           | 4           | 7        | 7       | 6           |

## Extraindo Mais Paralelismo

- Execução condicional de instruções

```
if (a == 0) s=t
```

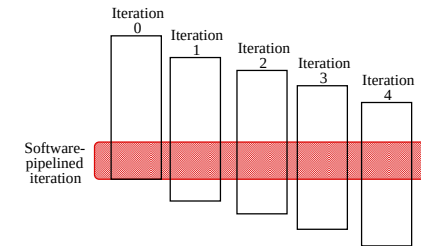
```
beqz r1,L          cmovz r2,r3,r1
mov r2,r3
L:  ...
```



- Mas ainda leva tempo se instrução não for executada
- Popular nas máquinas modernas
  - MIPS, Alpha, PowerPC e SPARC possuem move condicional
  - HPPA permite instr. R-R cancelar condicionalmente a instrução seguinte

## Software Pipelining

- Observação: se iterações de loops são independentes, então podemos ganhar ILP executando instruções de diferentes iterações de cada vez
- Software pipelining*: reorganiza loops de forma que cada iteração executada é realizada por instruções escolhidas das iterações diferentes do loop original (isto é, Tomasulo em SW)



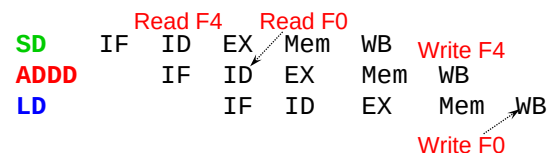
## Exemplo de SW Pipelining

Antes: Unrolled 3 vezes

```
1 LD  F0,0(R1)
2 ADDD F4,F0,F2
3 SD  0(R1),F4
4 LD  F6,-8(R1)
5 ADDD F8,F6,F2
6 SD  -8(R1),F8
7 LD  F10,-16(R1)
8 ADDD F12,F10,F2
9 SD  -16(R1),F12
10 SUBI R1,R1,#24
11 BNEZ R1,LOOP
```

Depois: Software Pipelined

```
LD  F0,0(R1)
ADDD F4,F0,F2
LD  F0,-8(R1)
1 SD  0(R1),F4; Stores M[i]
2 ADDD F4,F0,F2; Adds to M[i-1]
3 LD  F0,-16(R1); loads M[i-2]
4 SUBI R1,R1,#8
5 BNEZ R1,LOOP
SD  0(R1),F4
ADDD F4,F0,F2
SD  -8(R1),F4
```



## Resumo

- Previsão de Branches
  - Branch History Table: 2 bits para precisão alta de branches
  - Correlação: branches executados recentemente estão relacionados com próximo branch
  - Branch Target Buffer: inclui endereço junto com previsão
- Superescalar e VLIW
  - CPI < 1
  - Disparo dinâmico vs. estático
  - Mais instruções disparam ao mesmo tempo, maior penalidade de hazards
- SW Pipelining
  - Loop Unrolling simbólico para conseguir o máximo do pipeline com o mínimo de expansão do código