

#### COMPUTER ORGANIZATION AND DESIGN



The Hardware/Software Interface

# Capítulo 5

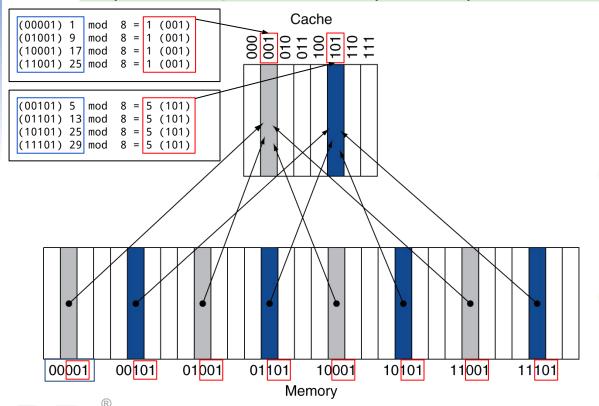
#### Grande e rápida: Explorando a Hierarquia de Memória

## Princípios Básicos de cache

- Começaremos a ver uma cache muito simples
  - Cada requisição do processador é uma palavra e os blocos também consistem em uma única palavra
  - Vamos nos concentrar em explicar como uma cache trata das leituras
    - As leituras não precisam mudar o conteúdo da cache



- Localização determinada pelo endereço
- Mapeado direto: apenas uma escolha
  - (Endereço de bloco) mod (Número Blocos na cache)



- Número de Blocos é uma potência de 2
- Usa bits de endereço de ordem inferior

# Tags e Bit Validade

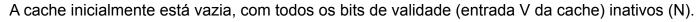
- Como sabemos qual bloco específico está armazenado em um local de cache?
  - Armazene o endereço do bloco, bem como os dados
  - Na verdade, só precisa dos bits de alta ordem
  - Chamado de tag
- E se não houver dados em um local?
  - Bit validade: 1 = presente, 0 = ausente
  - Inicialmente 0



- 8 blocos, 1 palavra/bloco, mapeamento direto
- Estado inicial

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		





End. Decimal	End. Binário	Hit/miss	Bloco Cache
22	10 110	Miss	110

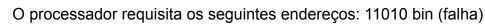
O campo tag conterá apenas a parte superior do endereço. O endereço completo de uma palavra contida no bloco de cache i com o campo tag j para essa cache é j × 8 + i ou, de forma equivalente, a concatenação do campo tag j e o campo índice i. Por exemplo, o índice 010 bin possui tag 10 bin e corresponde ao endereço 10110 bin .

Indice	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



End. Decimal	End. Binário	Hit/miss	Bloco Cache
26	11 010	Miss	010

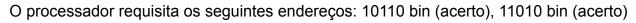
Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Υ	10	Mem[10110]
111	N		



End. Decimal	End. Binário	Hit/miss	Bloco Cache
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Υ	11	Mem[11010]
011	N		
100	N		
101	N		
110	Υ	10	Mem[10110]
111	N		





End. Decimal	End. Binário	Hit/miss	Bloco Cache
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Υ	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Υ	10	Mem[10110]
111	N		



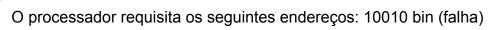


Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Quando o endereço 10010 bin (18) é referenciado, a entrada para o endereço 11010 bin (26) precisa ser substituída, e uma referência a 11010 bin causará uma falha subsequente.

Index	V	Tag	Data
000	Υ	10	Mem[10000]
001	N		
010	Υ	11	Mem[11010]
011	Υ	00	Mem[00011]
100	N		
101	N		
110	Υ	10	Mem[10110]
111	N		





Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Quando o endereço 10010 bin (18) é referenciado, a entrada para o endereço 11010 bin (26) precisa ser substituída, e uma referência a 11010 bin causará uma falha subsequente.

Index	V	Tag	Data
000	Υ	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Υ	10	Mem[10110]
111	N		

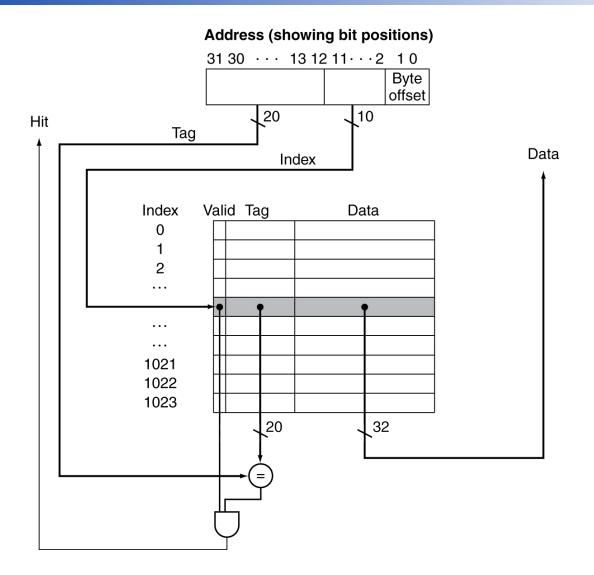




Sequência de nove referências da memória a uma cache vazia de oito blocos

Endereço decimal da referência	Endereço binário da referência	Acerto ou falha na cache	Bloco de cache atribuito
22	10110bin	falha	(10110bin mod 8) = 110bin
26	11010bin	falha	(11010bin mod 8) = 010bin
22	10110bin	acerto	(10110bin mod 8) = 110bin
26	11010bin	acerto	(11010bin mod 8) = 010bin
16	10000bin	falha	(10000bin mod 8) = 000bin
3	00011bin	falha	(00011bin mod 8) = 011bin
16	10000bin	acerto	(10000bin mod 8) = 000bin
18	10010bin	falha	(10010bin mod 8) = 010bin
16	10000bin	acerto	(10000bin mod 8) = 000bin



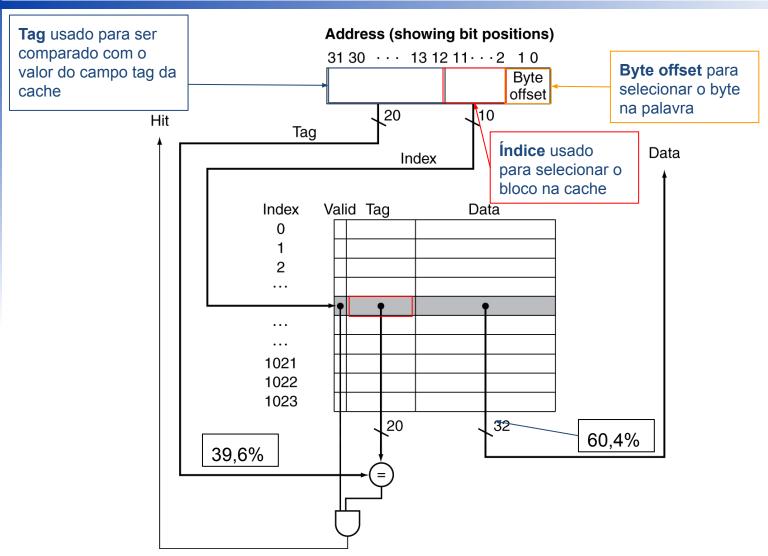




- Como um endereço referenciado é dividido em:
  - Um campo tag
    - Usado para ser comparado com o valor do campo tag da cache
  - Um campo índice
    - Usado para selecionar o bloco na cache

31		12	11	2	1 0
	Tag		Índice		Offset de byte
	20 bits	_	10 bits		2 bits







- Na arquitetura MIPS
  - As palavras são alinhadas como múltiplos de 4 bytes
  - Os dois bits menos significativos de cada endereço especificam um byte dentro de uma palavra (offset de byte)
    - São ignorados ao selecionar uma palavra no bloco

31		12	11	2	1 0
	Tag		Índice		Offset de byte
	20 bits	_	10 bits		2 bits

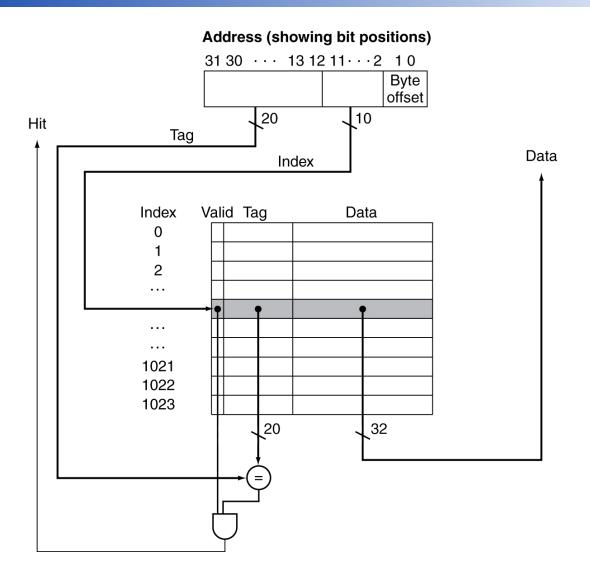


#### Na arquitetura MIPS

- 1 k posições precisa de 10 bits para endereçar
   (2<sup>10</sup>=1024) -> índice = 10
- Tag = total de bits (bits de índice + bits offset byte) Tag = 32 (10 + 2) = 20 bits

31	12 11	2	1 0
Tag	Índice		Offset de byte
20 bits	s 10 bits		2 bits

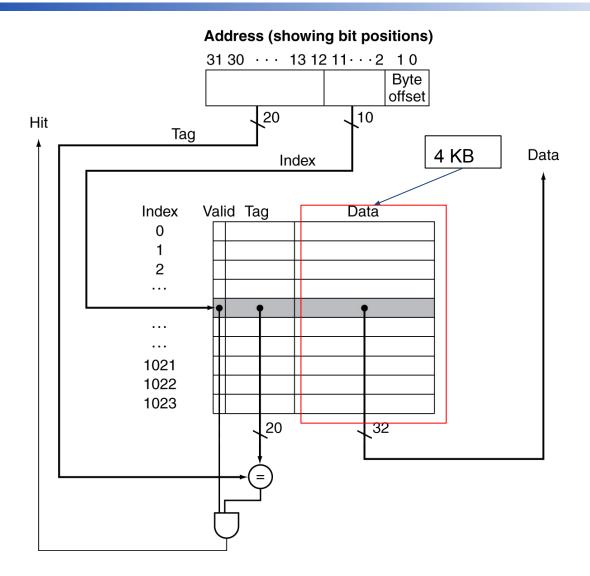






- Qual a capacidade total de dados úteis dessa cache?
  - Cache com 1024 blocos (linhas)
  - Tamanho do bloco 1 palavra (32 bits)
  - Total = Número blocos x (bits de cada bloco)
  - Total = 1024 x 32 = 32.768 bits = 32 Kb = 4 KB

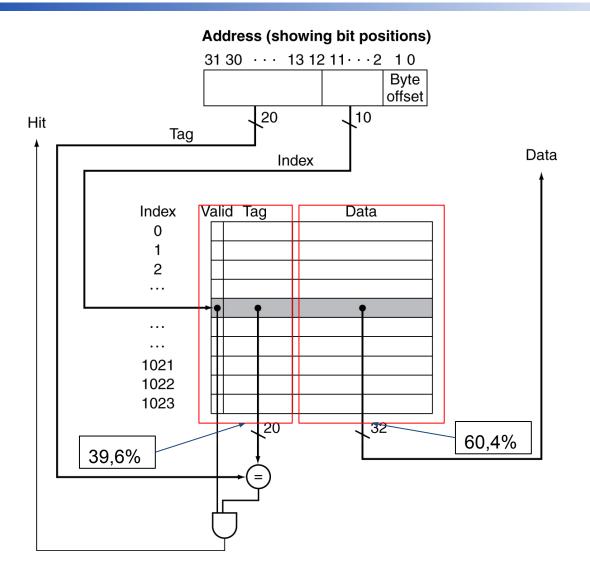






- Qual a quantidade total de bits necessária para implementar essa cache?
  - Cache com 1024 blocos (linhas)
  - Cada bloco tem: bit validade + tag + dados
  - Total = Número blocos x (bits de cada bloco)
  - Total =  $1024 \times (1+20+32) = 1024 \times 53 \text{ bits} = 54.272$ bits = 53 Kbits
    - Do total de 53 Kbits para implementar, a parte de dados útil corresponde 32 Kbits (4 KB) 60,4%, o restante 39,6% corresponde ao controle







#### **Exemplo 1: Tamanho de Bloco Maior**

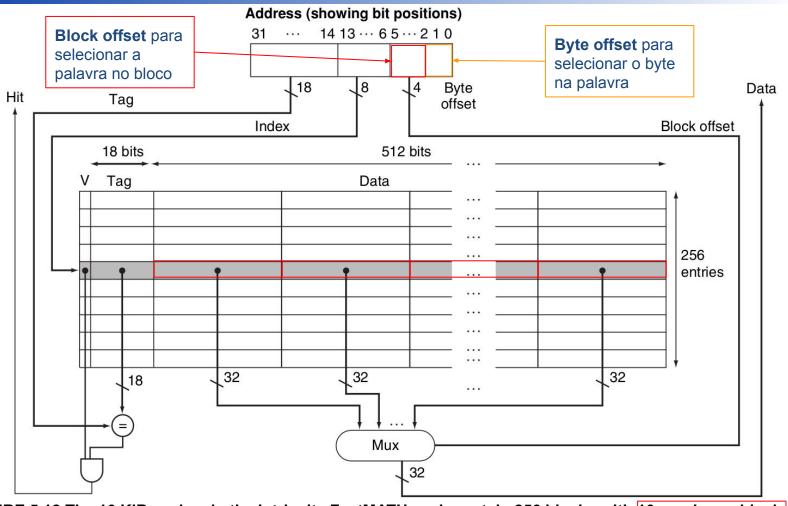


FIGURE 5.12 The 16 KiB caches in the Intrinsity FastMATH each contain 256 blocks with 16 words per block. Copyright © 2013 Elsevier Inc. All rights reserved



#### Exemplo 2: Tamanho de Bloco Maior

- Cache com 16 KB de dados e blocos de 4 palavras
  - Cache com 16 KB de dados e blocos de 4 palavras



#### **Exemplo 2: Tamanho de Bloco Maior**

- 64 blocos, 16 bytes/bloco
  - 16 bytes/bloco = 4 palavras/bloco
  - Para qual número de bloco o endereço em bytes 1200 é mapeado?

```
Endereço bloco = [1200/16] = 75
```

Número de bloco = 75 modulo 64 = 11

Número de bloco = (Endereço do bloco) módulo (Número de blocos da cache)

Endereço bloco = [Endereço em bytes/Bytes por bloco]



#### **Exemplo 3: Tamanho de Bloco Maior**

- Exercício 5.2.2
- Cache de mapeamento direto com blocos de duas palavras e um tamanho total de oito blocos
- 8 blocos, 8 bytes/bloco (2 palavras/bloco)
  - 8 bytes/bloco = 2 palavras/bloco
  - Para qual número de bloco o endereço em bytes 190 é mapeado?

```
Endereço bloco = \lfloor 190/8 \rfloor = 23
```

Número de bloco = 23 modulo 8 = 7



#### Taxa de Falhas versus Tamanho do Bloco

- Blocos maiores devem reduzir a taxa de erros
  - Devido à localização espacial
- Mas em uma cache de tamanho fixo
  - Blocos maiores ⇒ menos blocos
    - Mais competição ⇒ aumento da taxa de erros
  - Blocos maiores ⇒ poluição de cache¹
- Maior penalidade de falha
  - Pode suplantar o benefício da taxa de falha reduzida
  - O reinício precoce e a palavra requisitada primeiro podem ajudar

<sup>1 -</sup> Poluição de cache é quando dados que seriam reutilizados pelo processador são substituídos da cache por dados que não serão úteis.



#### Falhas de Cache

- No acerto do cache, a CPU continua normalmente
- Na falha de cache
  - Para o pipeline da CPU
  - Buscar bloco do próximo nível da hierarquia
  - Falha na cache de instrução
    - Reiniciar busca de instrução
  - Falha na cache de dados
    - Acesso completo aos dados



# Write-Through

- No acerto de escrita de dados, poderia apenas atualizar o bloco na cache
  - Mas então a cache e a memória seriam inconsistentes
- Write through: também atualiza a memória
- Mas escritas levarão mais tempo
  - Ex., se CPI sem falhas = 1, 10% das instruções sejam stores, gastam 100 ciclos extras em cada escrita
    - CPI efetivo = 1 + 100 x 10% = 11
- Solução: buffer de escrita
  - Armazena os dados enquanto estão esperando para serem escritos na memória
  - CPU continua imediatamente
    - Só para na gravação se o buffer de escrita já estiver cheio

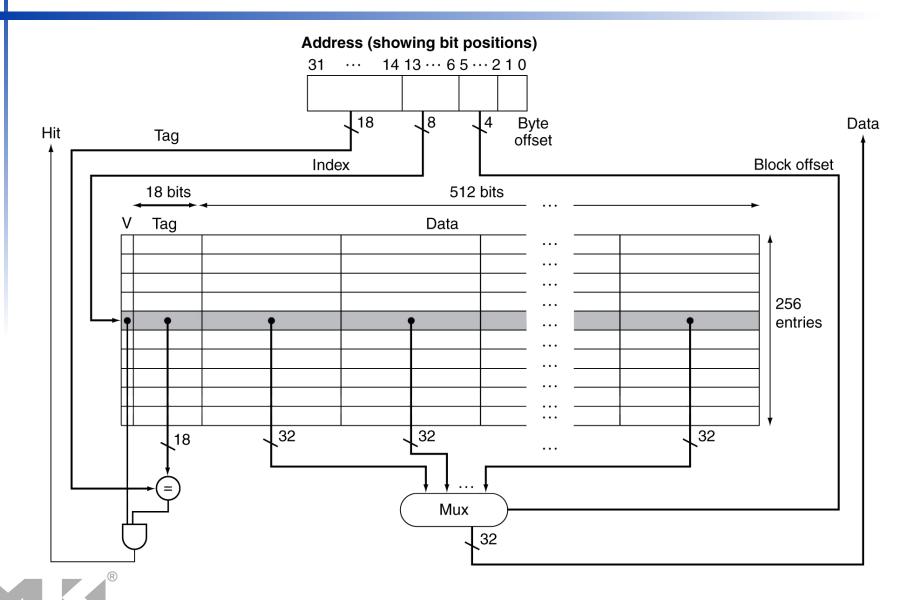


#### Write-Back

- Alternativa: no acerto de escrita de dados, basta atualizar o bloco no cache
  - Acompanhar se cada bloco é modificado
- Quando um bloco modificado é substituído
  - Escreva de volta na memória
  - Pode usar um buffer de escrita onde o bloco modificado é movido para permitir que o bloco de substituição seja lido primeiro



# **Exemplo: Intrinsity FastMATH**



#### **Caches Associativos**

- Cache totalmente associativo
  - Permitir que um determinado bloco entre em qualquer entrada de cache
  - Requer que todas as entradas sejam pesquisadas de uma vez
  - Comparador por entrada (caro)

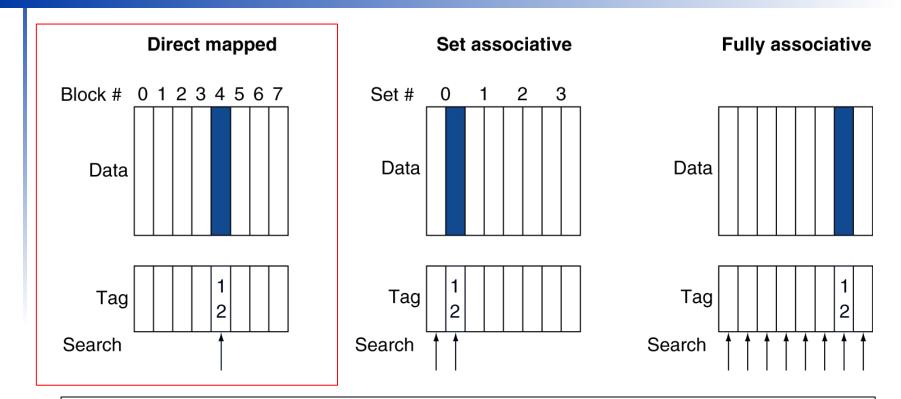


#### **Caches Associativos**

- Cache associativa por conjunto (n-way)
  - Cada conjunto contém n entradas
  - O número do bloco determina qual conjunto
    - (Número bloco) módulo (Número de blocos na cache)
  - Pesquisar todas as entradas em um determinado conjunto de uma vez
  - n comparadores (menos caro)



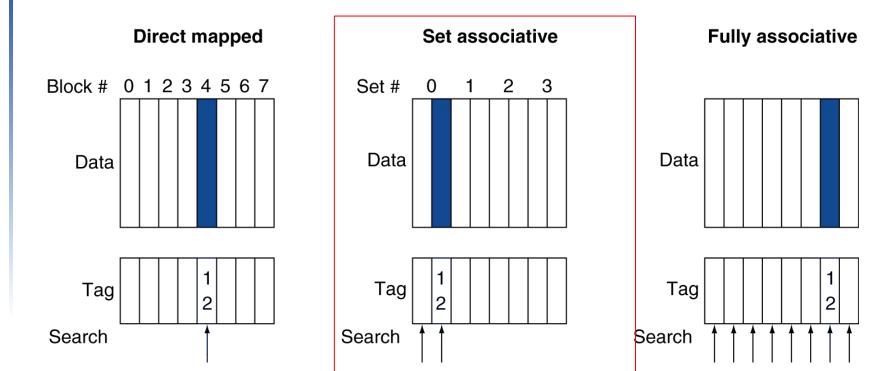
## **Exemplo Caches Associativos**



No posicionamento diretamente mapeado, há apenas um bloco de cache em que o bloco de memória 12 pode ser encontrado, e esse bloco é dado por (12 módulo 8) = 4.



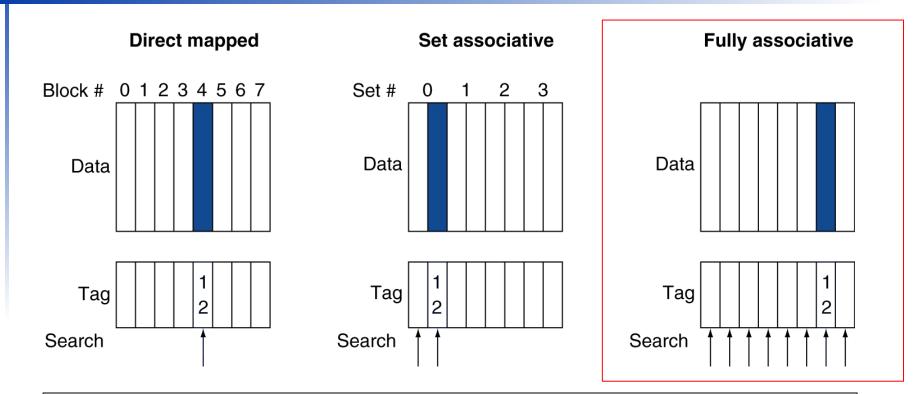
## **Exemplo Caches Associativos**



Em uma cache associativa por conjunto de duas vias, haveria quatro conjuntos e o bloco de memória 12 precisa estar no conjunto (12 mod 4) = 0; o bloco de memória pode estar em qualquer elemento do conjunto.



## **Exemplo Caches Associativos**



Em um posicionamento totalmente associativo, o bloco de memória para o endereço de bloco 12 pode aparecer em qualquer um dos oito blocos de cache.



# Espectro da Associatividade

## Para um cache com 8 entradas

O tamanho total da cache em blocos é igual ao número de conjuntos multiplicado pela associatividade

Para uma cache de tamanho fixo, aumentar a associatividade diminui o número de conjuntos enquanto aumenta o número de elementos por conjunto

Com oito blocos, uma cache associativa por conjunto de oito vias é igual a uma cache totalmente associativa

## One-way set associative (direct mapped)

3lock	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

#### Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

#### Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

#### Eight-way set associative (fully associative)

Tag	Data														



- Compare caches de 4 blocos
  - Mapeamento direto, conjunto de duas vias associativo, totalmente associativo
  - Sequência de acesso do bloco: 0, 8, 0, 6, 8
- Mapemaneto direto

Block	Cache	Hit/miss	Cache content after access			
address	ress index		0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	



### Mapeamento direto

Endereço do bloco	Bloco da cache
0	(0 módulo 4) = 0
6	(6 módulo 4) = 2
8	(8 módulo 4) = 0

Endereço	Ídice	Acerto ou	Conteúdo dos blocos da cache após a referência				
do bloco	Cache	Falha	0	1	2	3	
0	0	falha	Mem[0]				
8	0	falha	Mem[8]				
0	0	falha	Mem[0]				
6	2	falha	Mem[0]		Mem[6]		
8	0	falha	Mem[8]		Mem[6]		



## Associativa por dois conjuntos (2-way)

Endereço do bloco	Bloco da cache
0	(0 módulo 2) = 0
6	(6 módulo 2) = 0
8	(8 módulo 2) = 0

Endereço	Índice	Acerto	Conteúdo	dos blocos da	cache após a referência
do bloco	cache	ou falha	Conjunto 0		Conjunto 1
0	0	falha	Mem[0]		
8	0	falha	Mem[0]	Mem[8]	
0	0	acerto	Mem[0]	Mem[8]	
6	0	falha	Mem[0]	Mem[6]	
8 **R	0	falha	Mem[8]	Mem[6]	

### Totalmente associativa

Endereço do bloco	Acerto ou falha	Conteúdo dos blocos da cache após a referência				
0	falha	Mem[0]				
8	falha	Mem[0]	Mem[8]			
0	acerto	Mem[0]	Mem[8]			
6	falha	Mem[0]	Mem[8]	Mem[6]		
8	acerto	Mem[0]	Mem[8]	Mem[6]		



## **Quanta Associatividade**

- O aumento da associatividade diminui a taxa de erros
  - Mas com retornos decrescentes
- Simulação de um sistema com 64KB D-cache, blocos de 16 palavras, <u>SPEC2000</u>

1-way: 10.3%

2-way: 8.6%

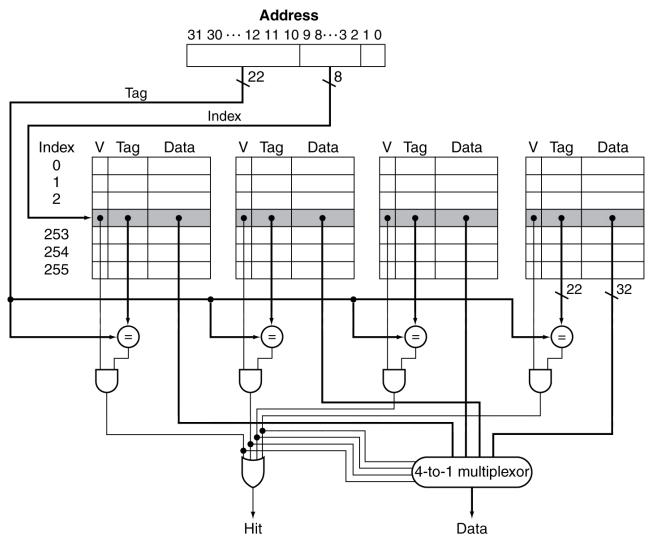
4-way: 8.3%

8-way: 8.1%

Passar da associatividade de uma via para duas vias diminui a taxa de falhas em aproximadamente 15%, mas há pouca melhoria adicional em passar para uma associatividade mais alta.



# Cache Associativa por Conjunto





# Política de Substituição

- Mapeado direto: sem escolha
- Associativo por conjunto
  - Prefira uma entrada inválida, se houver uma
  - Caso contrário, escolha entre as entradas do conjunto
- LRU (Least-recently used usado menos recentemente)
  - Escolha aquele que não é usado há mais tempo
    - Simples para 2 vias, gerenciável para 4 vias, muito difícil além disso

### Aleatório

 Oferece aproximadamente o mesmo desempenho que LRU para alta associatividade



## Projeto da Hierarquida de Memória

#### **The BIG Picture**

- Princípios gerais e comuns que aplicam a todos os níveis da hierarquia de memória
  - Conforme a discussão para cache
- Em cada nível da hierarquia
  - localização do bloco
  - encontrar um bloco
  - substituição em caso de falha
  - política de escrita



# Localização do Bloco

- Determinado pela associatividade
  - Mapeamento direto (associatividade 1-way)
    - Uma escolha
  - Associativa por n conjuntos (associatividade n-way)
    - N escolhas dentro de um conjunto
  - Totalmente associativa
    - Qualquer lugar
- Associatividade mais alta reduz a taxa de falhas (miss rate)
  - Aumenta complexidade, custo e tempo de acesso



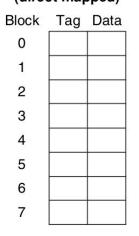
# Localização do Bloco

Nome do esquema	Número de conjuntos	Blocos por conjuntos
Mapeamento direto	Número de blocos na cache	1
Associativo por conjunto (n-way)	Número de blocos na cache/associatividade	Associatividade (normalmente 2 a 16)
Totalmente Associativa	1	Número de blocos na cache



# Localização do Bloco

### One-way set associative (direct mapped)



#### Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

#### Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0			×					
1								

#### Eight-way set associative (fully associative)

Tag	Data														

FIGURE 5.15 An eight-block cache confi gured as direct mapped, two-way set associative, four-way set associative, and fully associative. Copyright © 2009 Elsevier Inc. All rights reserved



## **Encontrando um Bloco**

Associatividade	Método de localização	Comparações de Tag		
Mapeamento direto	Indexação	1		
Associativo por conjunto (n-way)	Indexação do conjunto, pesquisa entre os elementos	Grau de associatividade		
Totalmente Associativa	Pesquisa de todas as entradas da cache	Tamanho da cache		
	Tabela de consulta separada	0		

### Caches em hardware

Reduzir comparações para reduzir custo

### Memória virtual

 Tabela de lookup completas tornam associatividade total possível



## Substituindo um Bloco

- Escolha de entrada que será substituída em caso de falha
  - Least recently used (LRU)
    - Complexa e custosa para hardware com alta associatividade
  - Aleatório
    - Resultados próximos a LRU, mais fácil de implementar
- Memória vitual
  - Aproximação de LRU com suporte de hardware



## Política de Escrita

- Write-through
  - Atualiza os vários níveis
  - Simplifica substituição, buffer de escrita ajuda
- Write-back
  - Atualiza somente nível mais próximo
  - Atualiza próximo nível quando bloco for substituído
  - Precisa manter mais informação de estado
- Memória virtual
  - Somente write-back é viável dado o tempo de escrita em disco



## Referências

- Seções 5.1 a 5.5 Organização e Projeto de Computadores - A Interface Hardware/Software, David A. Patterson & John L. Hennessy, Campus, 4 edição, 2013.
- Seção 5.1 a 5.5 Computer organization and design: the hardware/software interface/David A. Patterson, John L. Hennessy, Elsevier, 5th ed, 2013.

