

Memórias cache

Conceitos Básicos

João Canas Ferreira

Dezembro de 2017



Tópicos

1 Hierarquia de memória

2 Princípios de funcionamento de memórias cache

Operação de leitura

Outras operações

Contém figuras de “Computer Organization and Design”, D. Patterson & J. Hennessey, 3ª. ed., MKP

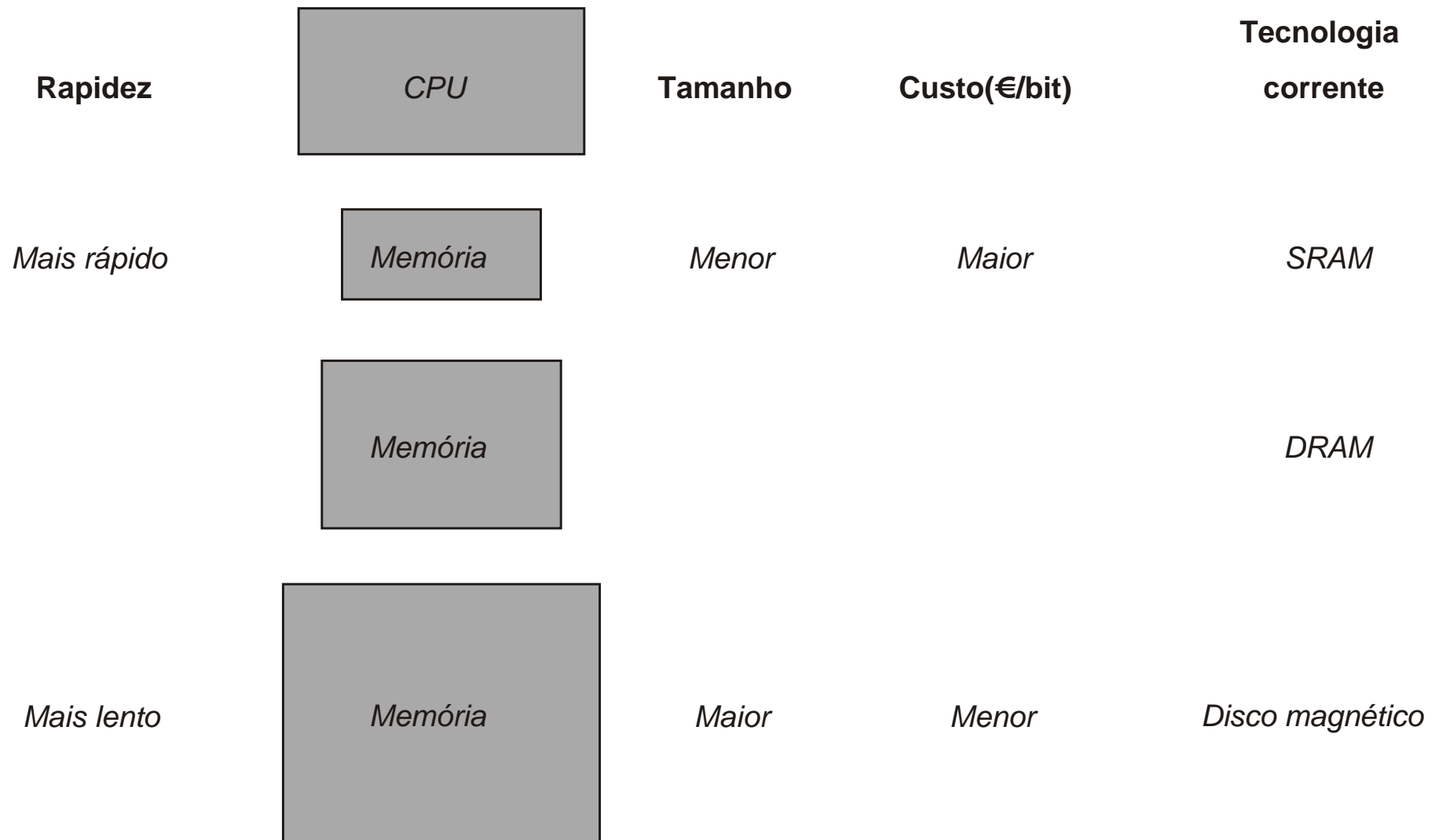
Observações gerais sobre sistemas de memória

- O custo dos diferentes tipos de memória é muito diferente:

Tecnologia	Tempo de acesso	Custo (\$/GiB em 2004)
SRAM	0.5–5 ns	\$4000–\$10000
DRAM	50–70 ns	\$100–\$200
disco magnético	5×10^6 – 2×10^7 ns	\$0.5–\$2

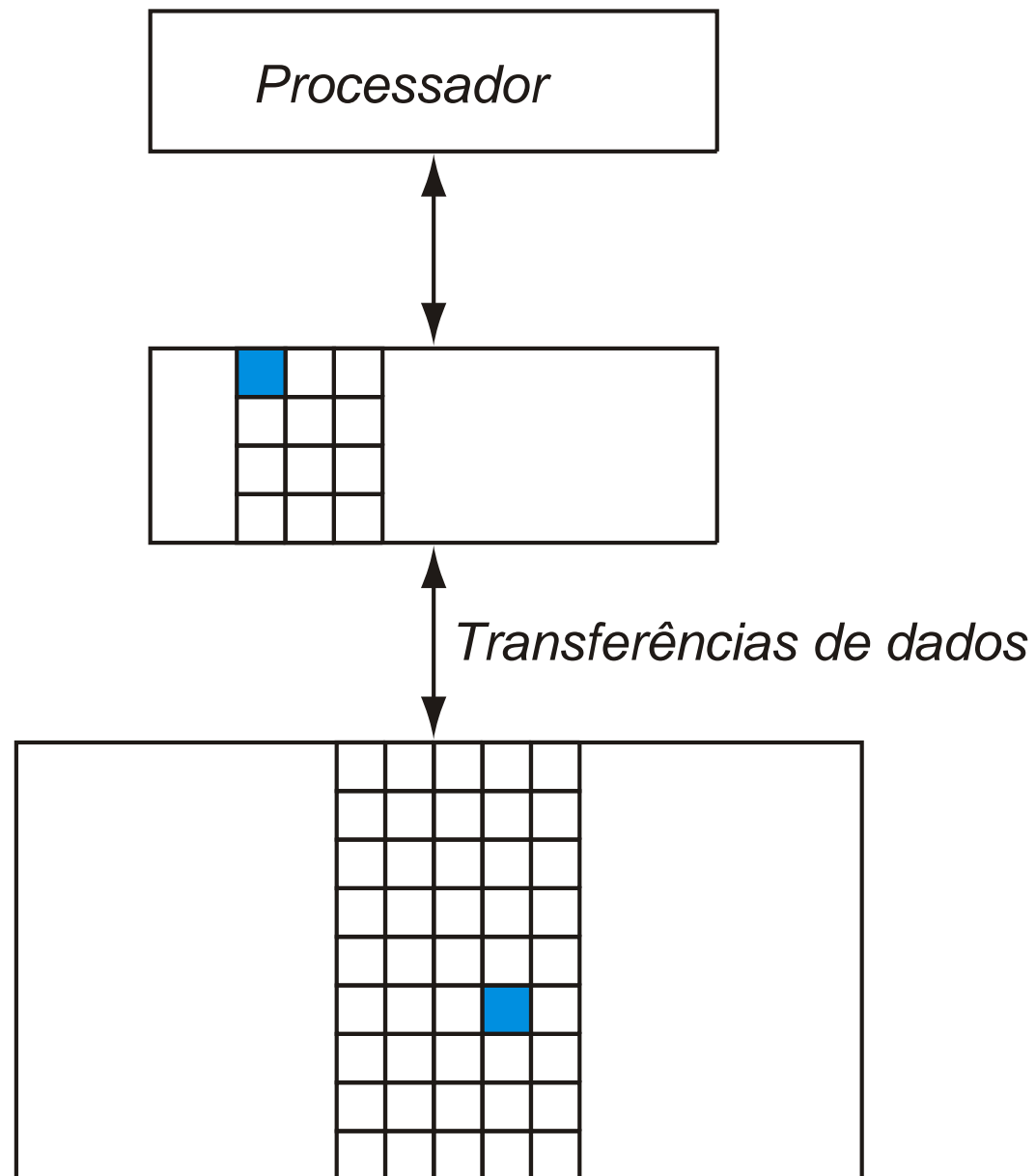
- Organizar o sistema de memória como uma hierarquia: memória rápida para satisfazer a maior parte dos acessos, memória mais lenta para armazenar os dados “menos usados”.
- Princípio de funcionamento (para cada nível):
 - Procurar a informação (dados ou instruções) num dado nível (inicialmente no nível 1, o que está mais próximo do CPU);
 - Caso a informação não exista aí, ir buscá-la ao nível seguinte e guardar uma cópia (pode vir a ser reusada brevemente).
- Se a maioria dos acessos for satisfeita pelo nível n , obtêm-se, em média, os tempos de acesso do nível n e a capacidade de armazenamento do nível seguinte $n + 1$.

Organização hierárquica de sistemas de memória



Fonte: [COD3]

Níveis da hierarquia de memória



Fonte: [COD3]

Hierarquia de memória: conceitos básicos

► Para uma acesso ao nível n da hierarquia:

- Uma hierarquia de memória pode ter vários níveis, mas a informação é transferida diretamente apenas entre níveis adjacentes: o nível n é mais rápido e tem menor capacidade que o nível $n + 1$.
- **Bloco:** quantidade mínima de informação que pode estar presente ou ausente de um determinado nível. A informação entre níveis é transferida em blocos.
- **Acerto (no nível n):** acesso ao nível n encontrou a informação pretendida (*hit*).
- **Falta:** acesso ao nível n **não** encontrou a informação (*miss*).
- **Taxa de acertos:** $(n^\circ \text{ de acertos}) / (n^\circ \text{ de acessos})$.
- **Taxa de faltas:** $(n^\circ \text{ de faltas}) / (n^\circ \text{ de acessos}) = 1 - (\text{taxa de acertos})$.
- **Tempo de acerto:** tempo de acesso ao nível n , incluindo a verificação da presença do item pretendido (*hit time*).
- **Penalidade de falta:** tempo necessário para copiar o item pretendido do nível $n + 1$ para o nível n , mais o tempo necessário para o nível n terminar o atendimento do acesso original (*miss penalty*).
- **Uma boa compreensão da hierarquia de memória é indispensável para a elaboração de programas de elevado desempenho.**

O princípio da proximidade

- A organização hierárquica de memória dá muito bons resultados na prática, porque as situações em que é aplicada se comportam de acordo com o princípio da proximidade.
- O princípio da proximidade (ou localidade) resume uma observação *empírica*:

Programas em execução acedem apenas a uma pequena parte do seu espaço de endereçamento durante um certo intervalo de tempo.

- **Proximidade temporal:** Se um item de informação é usado, tenderá a ser usado de novo em breve.

Exemplo: instruções no corpo de um ciclo, variáveis.

- **Proximidade espacial:** Quando um item de informação é usado, itens próximos tenderão também a ser usados em breve.

Exemplo: o conjunto de variáveis de uma subrotina, os elementos de uma sequência.

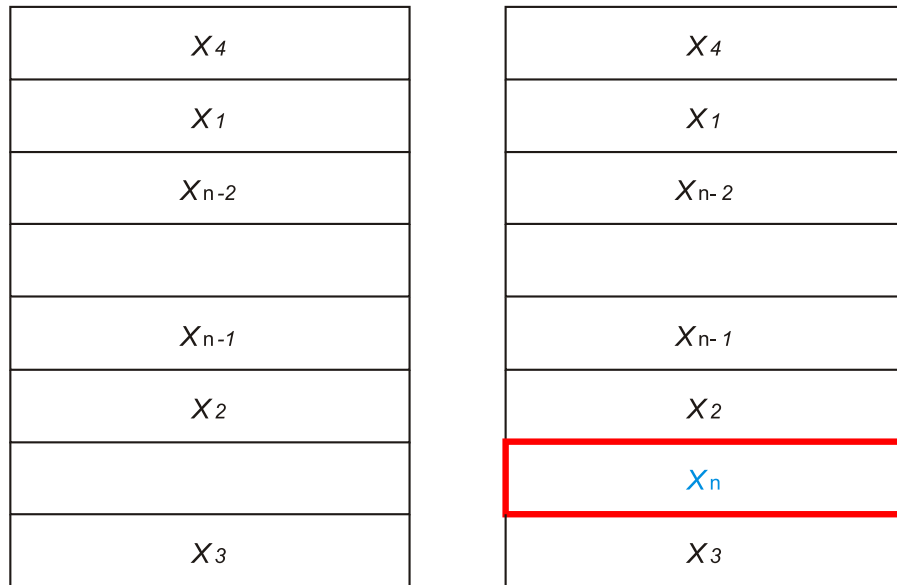
1 Hierarquia de memória

2 Princípios de funcionamento de memórias cache

Operação de leitura

Outras operações

Princípio básico de funcionamento (leitura)



a. **Antes** de referência a X_n

b. **Depois** de referência a X_n

- Se o acesso a memória *cache* não encontra item desejado, item é copiado de níveis inferiores da hierarquia.

Duas questões relacionadas:

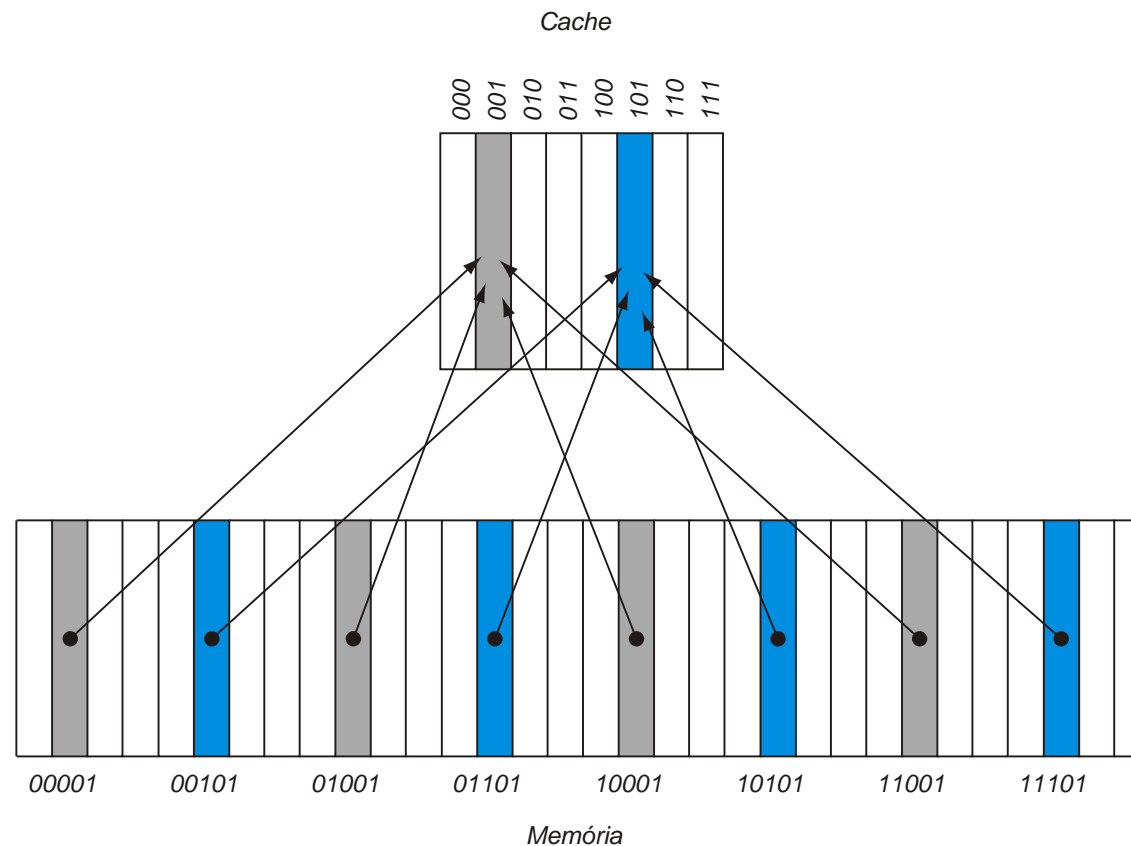
- 1 Como determinar a posição do item pretendido?
- 2 Como determinar a presença do item?

O mapeamento

endereço → **posição de cache** é sempre do tipo **muitos** → **1**.

Determinação da posição: mapeamento direto

- Índice = (endereço do bloco) mod (nº de blocos)
- Se (nº de blocos) = 2^b , a posição (o índice) é dada pelos b bits menos significativos do endereço.

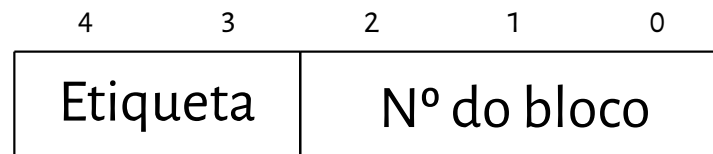


Fonte: [COD3]

Exemplo: Cache: 8 posições, memória principal: 32. Índice = Endereço mod 8

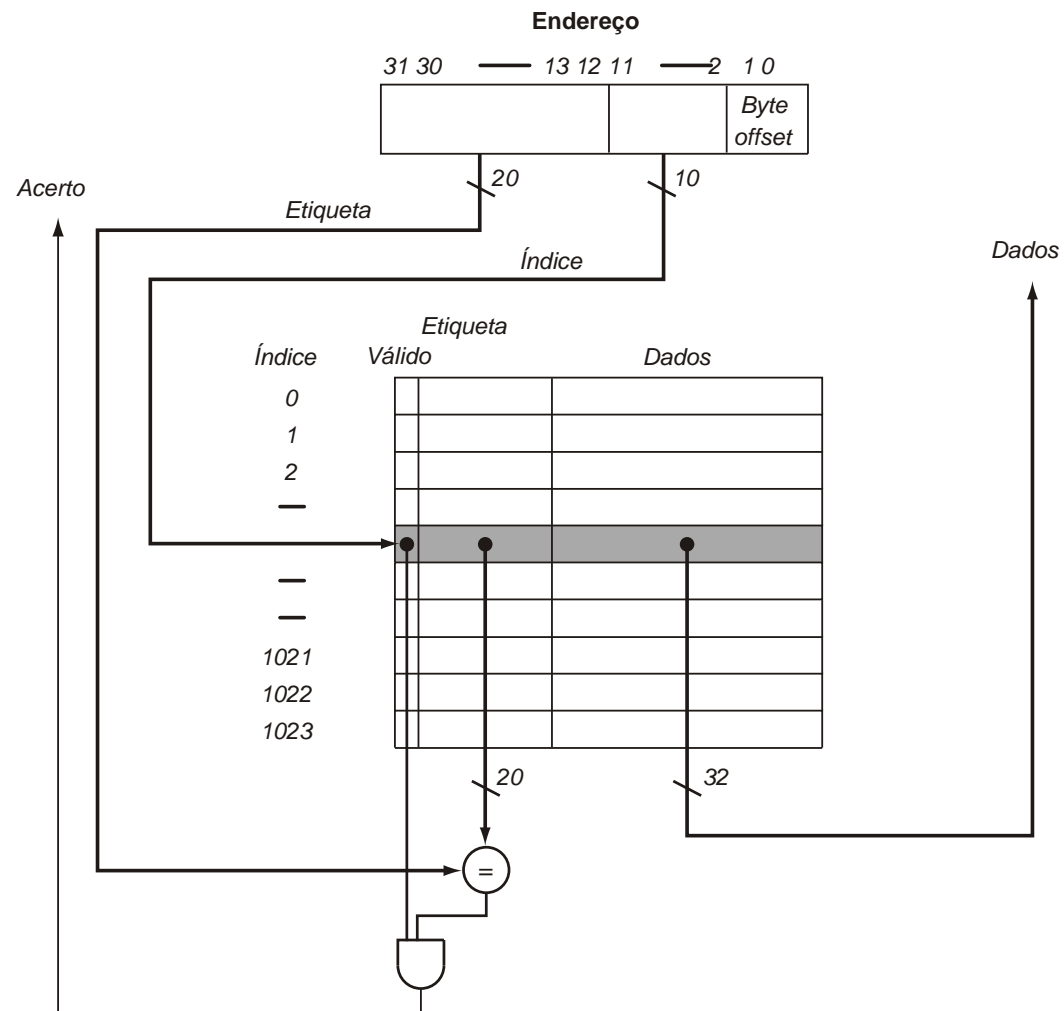
Determinação da presença: etiquetas

- **Questão:** Dada a posição de um item em cache, como determinar se é esse item que, de facto, lá se encontra?
- **Solução:** Associar a cada bloco, uma etiqueta (*tag*) única.
- A etiqueta pode ser constituída pelos bits mais significativos do endereço (bits não usados na determinação da posição).
- No exemplo anterior, a etiqueta teria 2 bits.



- Em certas situações, tanto o bloco como a etiqueta são inválidos. Exemplo: arranque do sistema.
- Para detetar esta situação, cada bloco tem associado um *bit de validade*. Se o seu valor for zero, então o bloco é inválido (a posição está “vazia”).

Exemplo: Cache para MIPS



Fonte: [COD3]

Como cada palavra (item) tem 4 bytes, os dois bits menos significativos não são usados.
(Todos os endereços são alinhados; têm os 2 bits menos significativos a 0).

Múltiplas palavras por bloco

- O armazenamento de uma palavra (item) por bloco *não aproveita a proximidade espacial*.
- Para isso devem ser usados blocos com L itens ($L > 1$).
- Para que seja fácil de identificar o item no interior do bloco, o número L deve ser uma potência de 2: $L = 2^w$.

Nesta situação, o item pode ser identificado por w bits do endereço. A parte do endereço que especifica a posição dentro do bloco designa-se por “deslocamento” (*offset*).

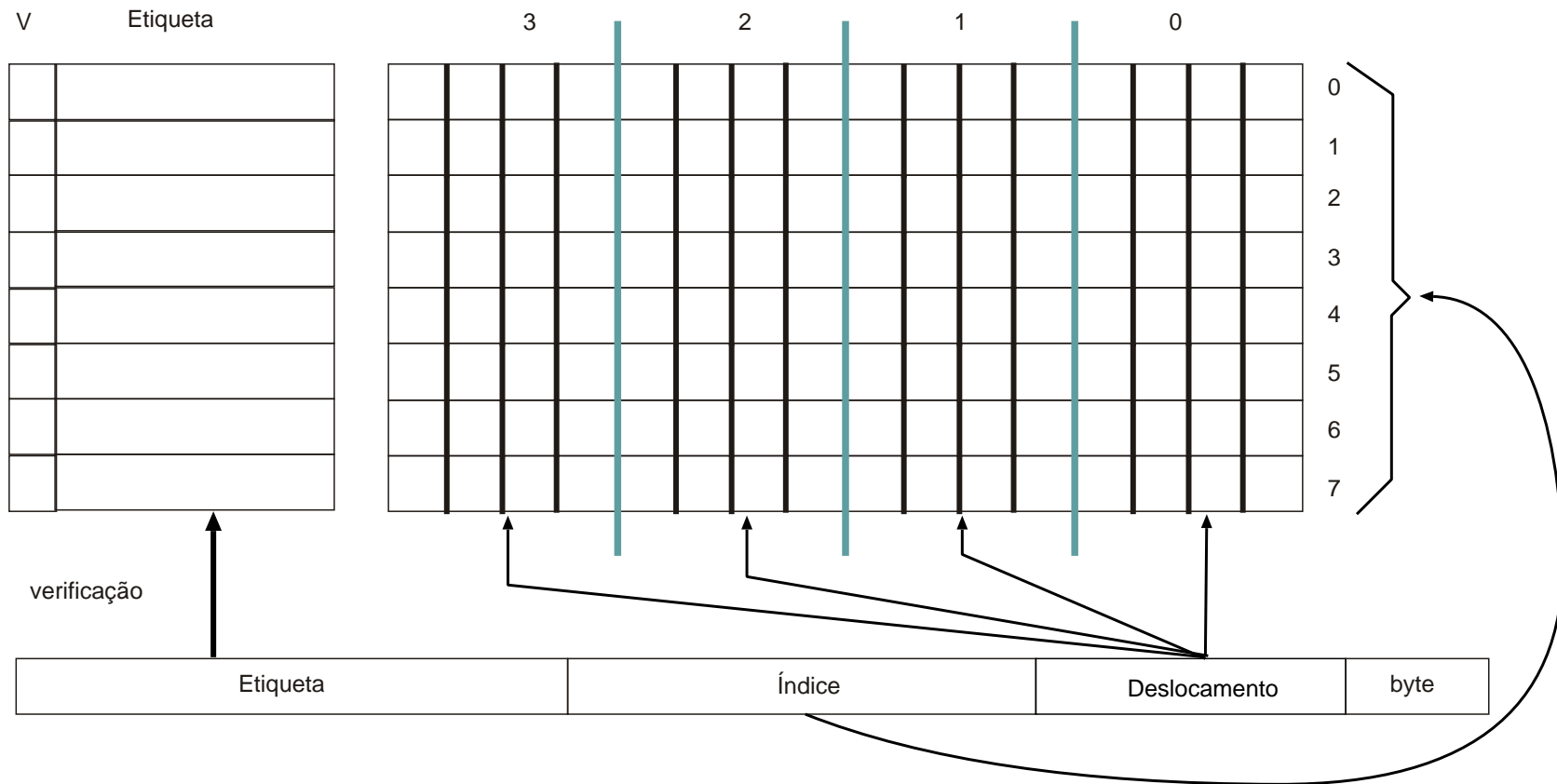
- Interpretação de um endereço de N bits:

Etiqueta ($N-b-w-x$)	Índice (b)	Deslocamento (w)	x
------------------------	----------------	----------------------	-----

- Quando os itens não têm apenas um byte (a unidade de endereçamento), existem alguns bits (menos significativos) do endereço que não são usados no acesso a memória (x).

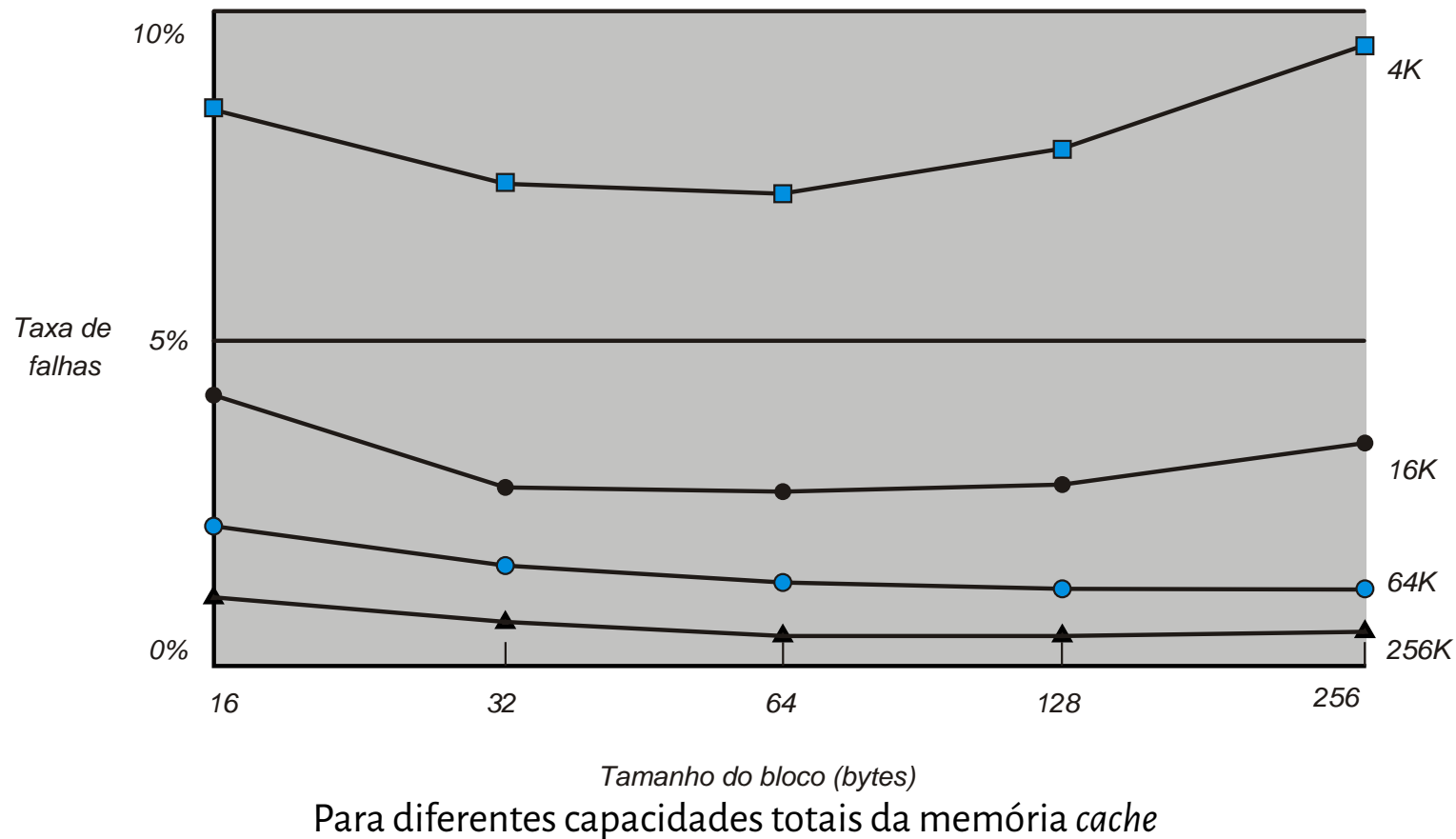
No caso do MIPS, os acessos são feitos por palavras de 4 bytes, pelo que os 2 bits menos significativos não são usados.

Estrutura de cache com múltiplas palavras por bloco



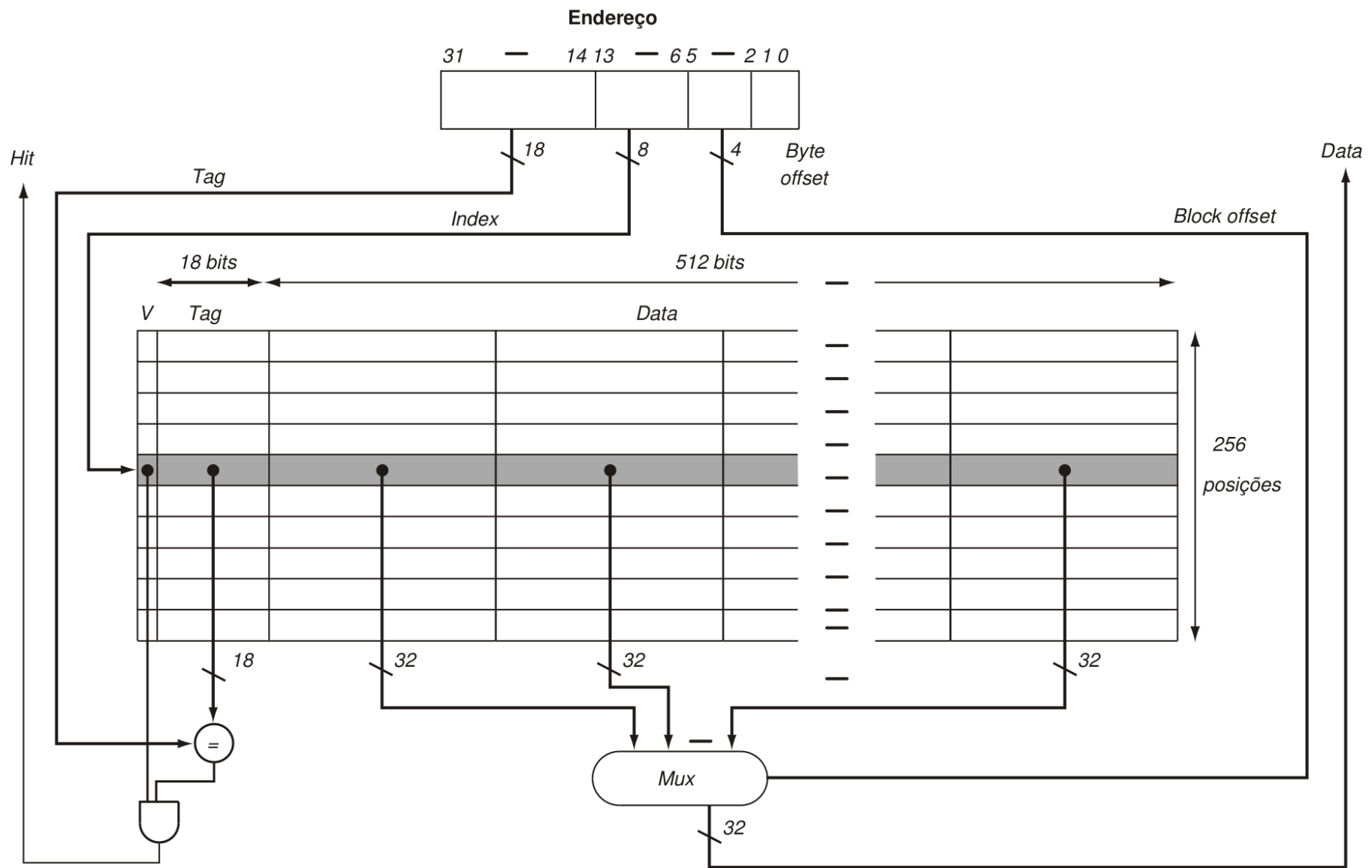
Cache com $b=3$, $w=2$.

Tamanho de bloco vs. taxa de faltas



- Para tamanhos de blocos crescentes, a taxa de faltas diminui.
- Mas para tamanhos muito grandes, a taxa de faltas aumenta outra vez!
- A penalidade de falha tende a aumentar com a dimensão do bloco, porque é preciso ir buscar mais dados ao nível seguinte.

Exemplo: Intrinsity FASTMath (1)



Fonte: [COD3]

Exemplo: Intrinsicity FASTMath (2)

- Capacidade: 16 KB, 256 blocos de 16 palavras.
- Etiqueta: 18 bits.
- Índice: 8 bits ($2^8 = 256$).
- Deslocamento (no bloco): 4 bits $2^4 = 16$.
- Exemplo de cálculo de posição:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Etiqueta (18)																		Índice (8)				Desl.				—					

Endereço = 0x00457544 (4552004_{10})

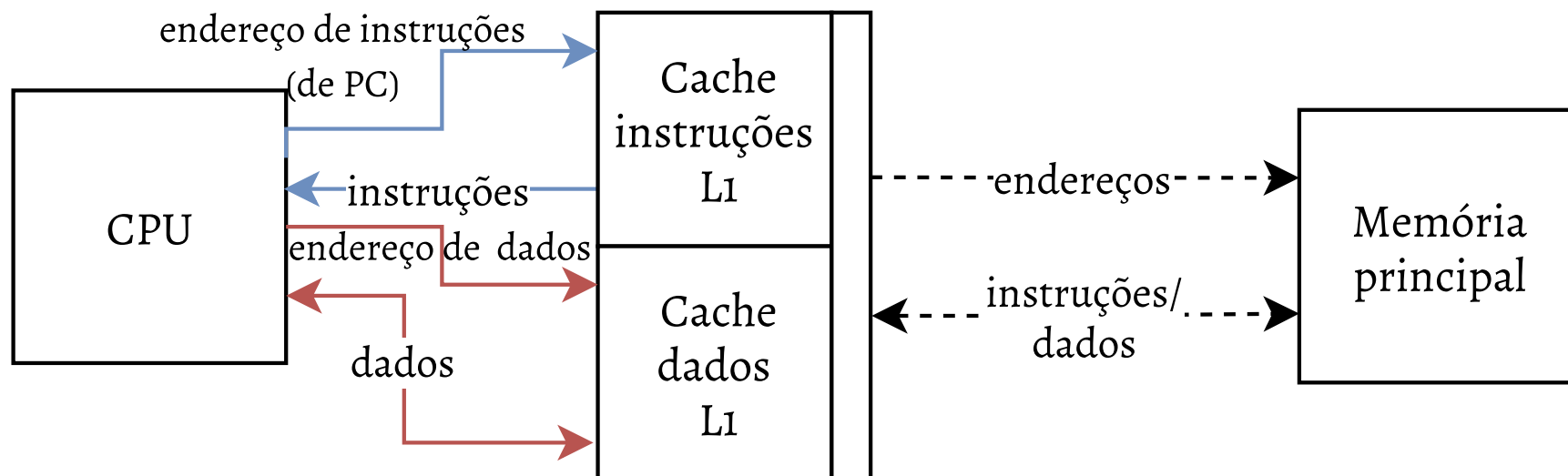
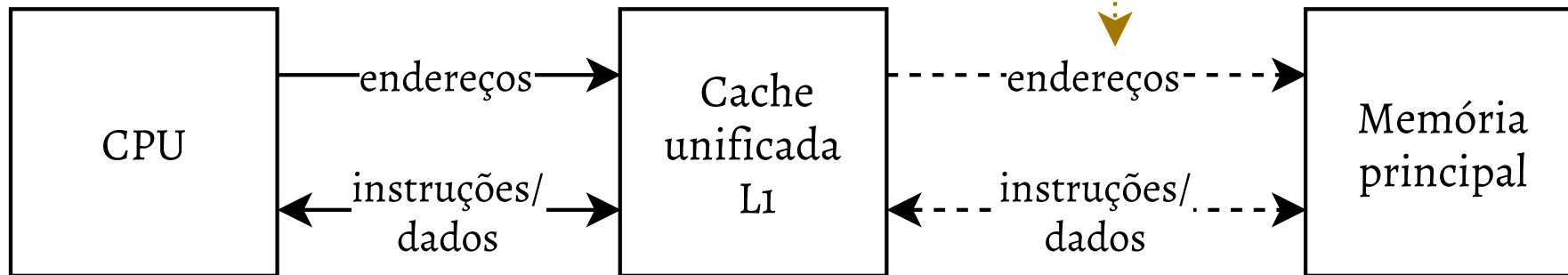
Etiqueta: 0000 0000 0100 0101 01 | 11 ... \rightarrow 0x00115

Bloco: ... 01 | 11 0101 01 | 00 0100 \rightarrow 0xD5 (213_{10})

Memórias cache separadas (1)

Memória cache unificada vs. separada (*split cache*)

estes acessos não se realizam se
dados/instruções estiverem em memória cache



Memórias cache separadas (2)

- Em muitos sistemas, existem memórias *cache* separadas para instruções (I-cache) e dados (D-cache).
- A I-cache é usada no ciclo IF e apenas permite acessos de leitura.
- A D-cache permite leituras e escritas; é usada no tratamento de instruções *load/store*.
- A utilização de memórias *cache* separadas (*split cache*), **permite aceder a instruções e dados no mesmo ciclo de relógio.**
- Para uma mesma capacidade total, memórias *cache* unificadas (U-cache) permitem obter menores taxas de faltas (mas a diferença é pequena).

Exemplo empírico de taxa de faltas com **I-Cache + D-Cache** vs. **U-Cache**:

- Caches separadas:
I-Cache(16 KiB): 0.4%, D-Cache (16 KiB): 11,4 %,
Média pesada: **3,24 %**
- Cache unificada de 32 KiB: **3,18 %**

1 Hierarquia de memória

2 Princípios de funcionamento de memórias cache

Operação de leitura

Outras operações

Tratamento de faltas de leitura: *read miss*

▣ Ao efetuar uma leitura da memória *cache*, esta pode detetar que o item pretendido não está armazenado. Então são executadas as seguintes operações:

- 1 O endereço é passado para o nível seguinte.
- 2 A memória cache espera que o nível seguinte responda com o bloco pretendido.
- 3 O novo bloco é colocado na cache (eventualmente usando uma posição já ocupada), a etiqueta é guardada e o bit de validade correspondente é colocado a 1.
- 4 O item pretendido é disponibilizado ao circuito que o solicitou.

▣ Uma falta de leitura pode acontecer durante a obtenção de uma instrução ou durante uma leitura de dados (*load*).

▣ Uma falta no acesso à memória *cache* leva a um protelamento longo (tipicamente mais de 100 ciclos).

O protelamento consiste simplesmente em suspender todas as operações do CPU.

Tratamento de operações de escrita: *write hit*

Só existem operações de escrita em memórias *cache* de dados ou unificadas.

Escrita com valor em *cache* para essa posição (*write hit*):

- Operações de escrita levantam a possibilidade de o valor existente na memória *cache* ser diferente do valor em memória principal.
- Cache *write-through*: a memória *cache* é atualizada ao mesmo tempo que a memória principal.
- Cache *write-back*: apenas o valor em memória *cache* é atualizado. A memória principal é atualizada apenas quando o bloco é substituído.
- A política de *write-back* reduz o número de acessos ao nível seguinte, mas é mais complexa de implementar.

Tratamento de operações de escrita: *write miss*

Escrita de valor em bloco está ausente da memória *cache* (*write miss*):

- As duas variantes principais são:
 - Política *allocate-on-miss*: o bloco necessário é obtido do nível inferior, após o que tudo se passa como no caso de um *write-hit*.
Pode envolver a substituição de um bloco já presente em memória *cache*.
 - Política *no-allocate*: não alterar o conteúdo da memória *cache* e alterar diretamente a memória principal.
Funciona como se a memória *cache* não existisse para esta classe de transações.
- O tratamento de *cache miss* é logicamente independente do tratamento de *cache hit*.

As combinações mais usadas são:

- *write-through + no-allocate*: ênfase na simplicidade.
- *write-back + allocate-on-miss*: ênfase na redução de tráfego para a memória principal.

Resumo das operações de memória cache

► Para as combinações referidas anteriormente
(operações comuns, operações apenas para *write-through*, operações apenas para *write-back*):

operação	acerto (<i>hit</i>)	falta (<i>miss</i>)
leitura	♦ devolver valor em cache ao CPU	♦ obter valor de memória principal ♦ enviar esse valor para CPU ♦ novo conteúdo para bloco de memória cache ^(a) ♦ colocar d=0
escrita	♦ alterar valor em cache ♦ atualizar memória principal ♦ colocar d=1	♦ atualizar memória principal ♦ novo conteúdo para bloco de memória cache ^(a) ♦ alterar valor de parte do bloco ♦ colocar d=1

^(a) Obter novo conteúdo para bloco de memória cache envolve:

- 1 “libertar” o bloco de memória cache:
se d=1 (bloco foi anteriormente alterado em cache) escrevê-lo para memória principal;
- 2 ler de memória principal o novo conteúdo.

Referências

COD4 D. A. Patterson & J. L. Hennessey, Computer Organization and Design, 4 ed.

COD3 D. A. Patterson & J. L. Hennessey, Computer Organization and Design, 3 ed.

Os tópicos tratados nesta apresentação são descritos nas seguintes secções de [COD4]:

- 5.1–5.2

Também são tratados nas seguintes secções de [COD3]:

- 7.1–5.2