




Memória cache



28 de setembro de 2018

Aula por Alexandre Martins
<alemartf@gmail.com>



Objetivo

Estudar e simular o funcionamento das memórias cache.



Puzzly at a computer
Wikimedia Commons / Guillo

Roteiro

- Conceitos
 - Medidas de desempenho
 - Políticas de mapeamento
 - Políticas de substituição
 - Políticas de escrita
 - Parte prática
-

Conceitos

Conceitos

Revisão:

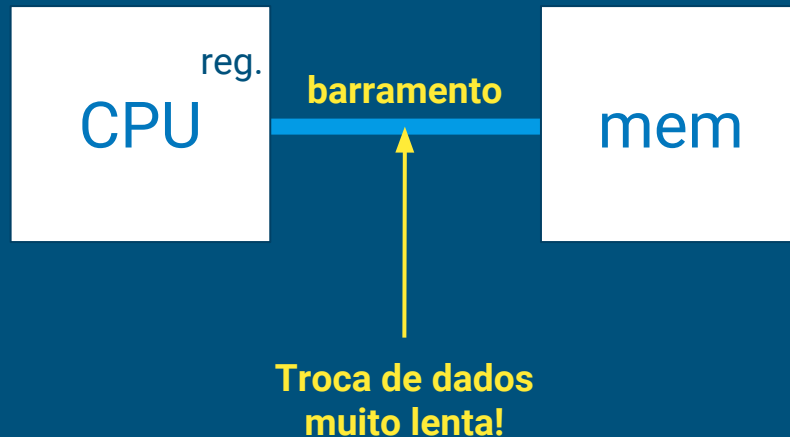
- Programas e dados ficam na memória
- CPU opera nos registradores
- CPU e memória se comunicam por meio do barramento local
- Memória é dividida em células
- Cada célula tem um endereço



Conceitos

Limitação:

- CPU funciona **muito** mais rápido que a memória RAM (*clock cycles* ~ 200x¹)
- Operações de leitura e escrita (memória) fazem com que o processador tenha que aguardar a memória (*wait states*)
- O sistema inteiro fica prejudicado em termos de desempenho



¹ Latency Numbers Every Programmer Should Know
<https://gist.github.com/jboner/2841832>

Conceitos

Princípio da localidade

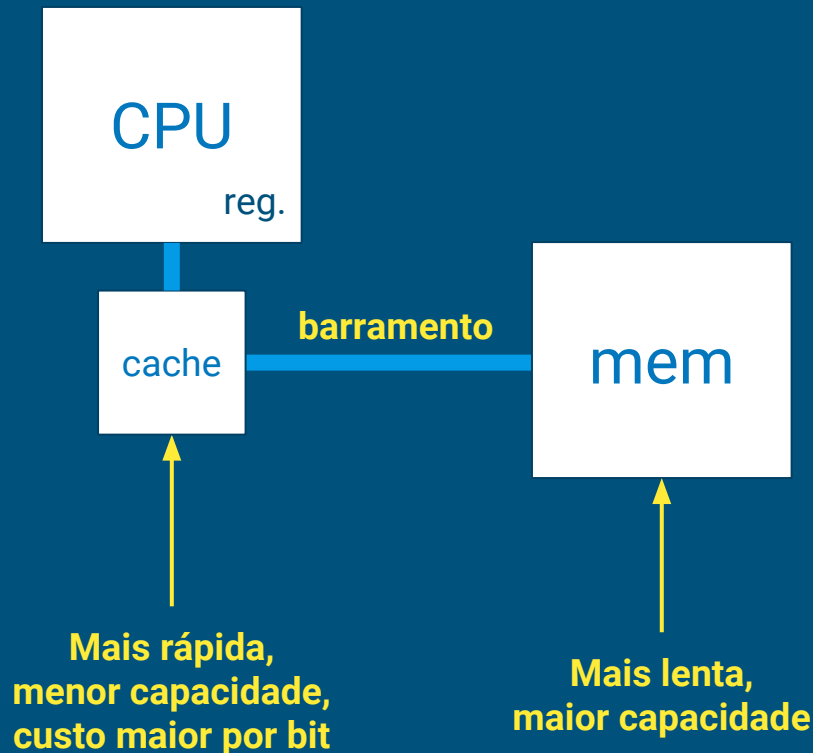
- Observa-se que os acessos à memória feitos pelos programas, em geral, têm:
- **Localidade temporal:** posições acessadas tendem a ser acessadas novamente num futuro próximo (ex: contadores, índices...)
- **Localidade espacial:** programas tendem a acessar posições vizinhas da memória (ex: instruções em sequência, vetores...)



Conceitos

Solução:

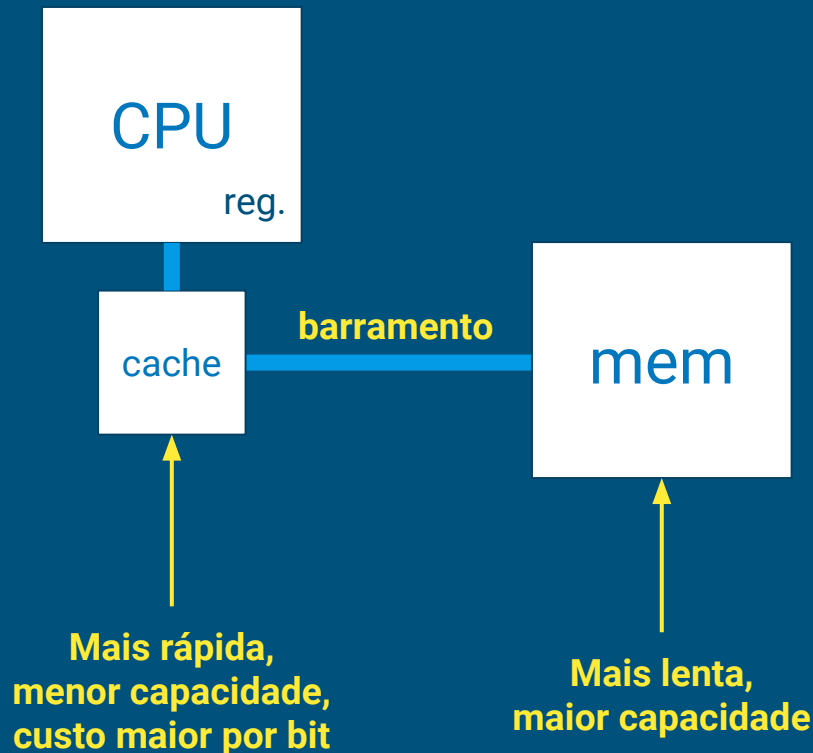
- Introduzir memória intermediária chamada **memória cache**
- Memória mais rápida e mais próxima ao processador, porém mais limitada
- Armazena cópia de parte do conteúdo da RAM, acelerando transferências
- O sistema inteiro melhora em termos de desempenho



Conceitos

Funcionamento do sistema:

- Quando a CPU precisa de uma palavra, ela primeiramente a procura na cache
- Se a palavra estiver na cache, ela é transferida em alta velocidade: **cache hit**
- Se a palavra não estiver na cache, ela é transferida da memória para a cache - e depois para o processador: **cache miss**



Conceitos

Hierarquia de memória:

- Registradores da CPU
- Memória cache (L1, L2 ...)
- Memória principal
- Memória secundária (disco ...)

A memória é organizada em vários níveis.

+ rápida
- capacidade

CPU
reg

cache
L1

cache
L2

Memória
principal

+ capacidade
- rápida

Conceitos

Hierarquia de memória:

Memória	Escala de tempo ¹
Cache L1	1
Cache L2	14
Memória RAM	200
Disco rígido (busca)	20 milhões

¹ Latency Numbers Every Programmer Should Know
<https://gist.github.com/jboner/2841832>

+ rápida
- capacidade

CPU
reg

cache
L1

cache
L2

Memória
principal

+ capacidade
- rápida

Conceitos

Hierarquia de memória:

Memória	Capacidade ¹	Escala
Cache L1	64 KB (32+32)	1
Cache L2	256 KB	4
Cache L3	8 MB	128
Memória RAM	32 GB	524 mil

¹ Intel Core i7-4770 (Haswell) - 4ª geração
<https://www.7-cpu.com/cpu/Haswell.html>

+ rápida
- capacidade

CPU
reg

cache
L1

cache
L2

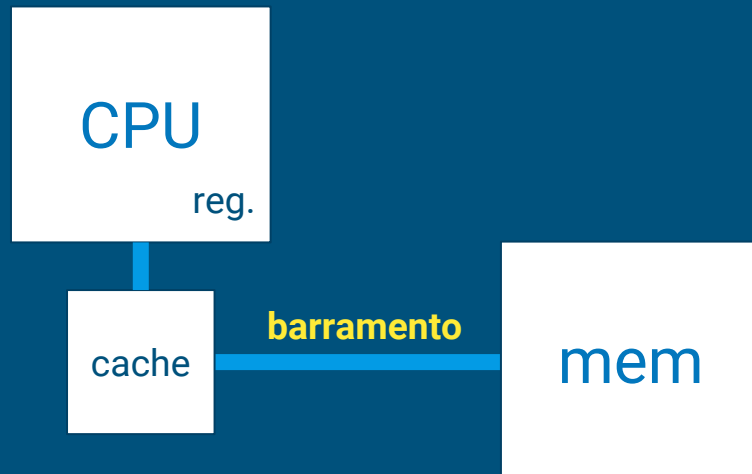
Memória
principal

+ capacidade
- rápida

Conceitos

Fatores que afetam o desempenho:

- **Tamanho da cache** ←
 - Quanto maior for a cache, maior será a possibilidade de se encontrar a palavra procurada (cache hit)
- Tempo de acesso das memórias
- Natureza dos programas (localidade)
- Políticas de mapeamento, etc.



Medidas de desempenho

Medidas de desempenho

Proporções:

- **Hit rate:** proporção de cache hits

$$hit_{rate} = \frac{\#hits}{\#acessos}$$

- **Miss rate:** proporção de cache misses

$$miss_{rate} = \frac{\#misses}{\#acessos}$$

Dois tipos de eventos:
cache hit ou cache miss.

$$hit_{rate} = 1 - miss_{rate}$$

$$\#acessos = \#hits + \#misses$$

Medidas de desempenho

Tempo:

- **Hit time:** tempo para transferir uma palavra da cache p/ o processador, quando houver cache hit.
 - Inclui tempo de busca da palavra na cache
- **Miss penalty:** tempo adicional para a transferência de uma palavra devido a um cache miss (penalidade média).
 - Deve-se buscar a palavra no nível seguinte
 - Dados serão copiados para a cache
 - Impacto importante na performance

Tempo médio de acesso à memória:

$$T = \text{hit time} + \text{miss penalty} * \text{miss rate}$$

Procura-se minimizar o miss rate



Se houver um cache miss, a CPU deve aguardar até que a palavra requisitada seja carregada da memória, ocasionando perda de desempenho (*memory stalls*)

Medidas de desempenho

Exemplo:

- Hit time = 1 ciclo
- Miss penalty = 80 ciclos
- Miss rate = 5%

Tempo médio de acesso à memória:

$$T = \text{hit time} + \text{miss penalty} * \text{miss rate}$$

$$T = 1 + 80 * 0,05$$

$$T = 1 + 4$$

$$T = 5$$

ciclos

Medidas de desempenho

O custo de um cache miss é **bem maior** que o de um cache hit.

Considere:

- Hit time = 1 ciclo
- Miss penalty = 100 ciclos
- arquitetura apenas c/ cache L1 e RAM

Prove: um hit rate de 97% custa o dobro de um hit rate de 99%

Hit rate = 1 - miss rate

Tempo médio de acesso à memória:

$T = \text{hit time} + \text{miss penalty} * \text{miss rate}$

$$T_{97} = 1 + 100 * (1 - 0,97) = 4 \text{ ciclos}$$

$$T_{99} = 1 + 100 * (1 - 0,99) = 2 \text{ ciclos}$$

$$\text{Logo, } T_{97} = 2 * T_{99}$$

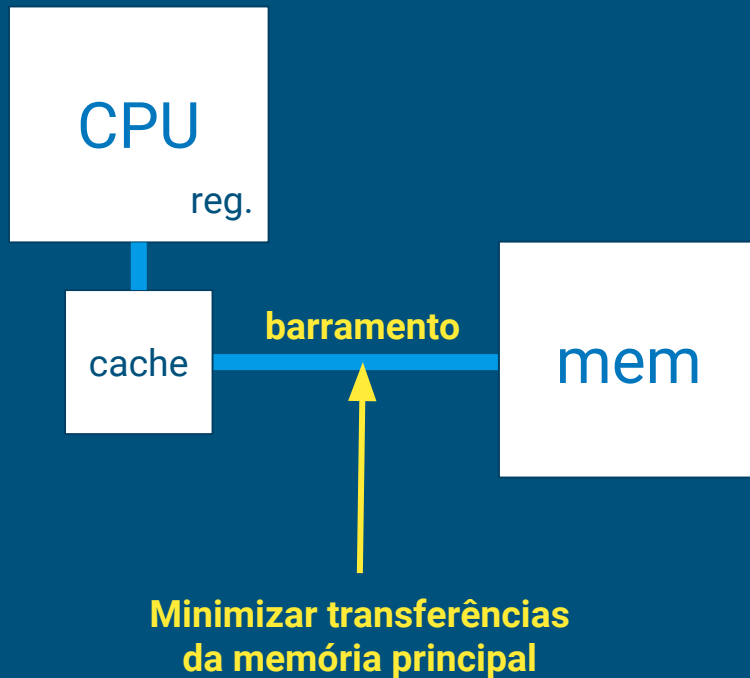
Apenas 2% a mais de miss rate,
e o tempo médio de acesso dobrou!

Medidas de desempenho

Como diminuir o miss rate?

- **Tamanho da cache** ←
 - Quanto maior for a cache, maior o hit rate (menor o miss rate)
- Explorar o princípio da localidade nos programas
- Política de mapeamento, etc.

Como o cache é menor que a memória, como escolher onde colocar os dados?



Políticas de mapeamento

Políticas de mapeamento

Revisão:

- Memória é dividida em células, tipicamente de 1 byte cada
- Cada célula tem um endereço (um número)

Memória

addr	byte
1568h	00000001
1569h	00011101
156ah	00001100
156bh	00010011

Políticas de mapeamento

Como mapear RAM p/ cache?

- Vamos dividir as memórias em **blocos**
- Cada bloco tem tamanho *block_size* ($= 2^b$)
 - Ex: 64 bytes por bloco (2^6)
- Se uma memória comporta *mem_size* bytes, em quantos blocos ela é dividida?

Memória

Byte 0
Byte 1
...
Byte 63

Bloco 0

Byte 64
Byte 65
...
Byte 127

Bloco 1

Byte 128
Byte 129
...
Byte 191

Bloco 2

...

Bloco 3

...

Bloco 4

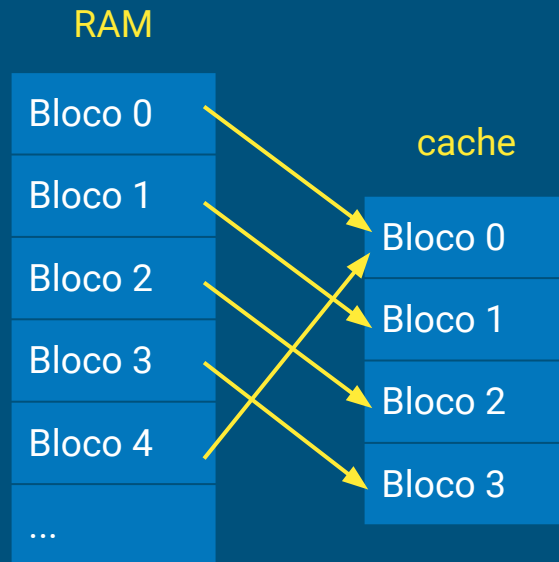
...

...

Políticas de mapeamento

Como mapear?

- Os dados são sempre transferidos em blocos, não em endereços individuais
 - Explora princípio da localidade
- A cache tem menos blocos (“linhas”) que a RAM
- Cada bloco tem um número
 - Numerados sequencialmente



Políticas de mapeamento

Como mapear?

- Se cada bloco tem 64 bytes, qual é o endereço do 1º byte do bloco 0?
addr = 0 (início da memória)
- E o endereço do 1º byte do bloco 100?
*addr = 100 * 64 = 6400*

Memória

Byte 0
Byte 1
...
Byte 63

Bloco 0

Byte 64
Byte 65
...
Byte 127

Bloco 1

Byte 128
Byte 129
...
Byte 191

Bloco 2

...

Bloco 3

...

Bloco 4

...

...

Políticas de mapeamento

Como mapear?

- E o endereço do 2º byte do bloco 100?

$$addr = 100 * 64 + 1 = 6401$$

offset

- Em qual bloco está o endereço 6401?

$$block_number = \lfloor 6401 / 64 \rfloor = 100$$

Memória

Byte 0
Byte 1
...
Byte 63

Bloco 0

Byte 64
Byte 65
...
Byte 127

Bloco 1

Byte 128
Byte 129
...
Byte 191

Bloco 2

...

Bloco 3

...

Bloco 4

...

...

Políticas de mapeamento

Como mapear?

- Qual é o offset do endereço 6401 no bloco 100?
offset = 6401 mod 100 = 1
- Qual é o offset do endereço 6400 no bloco 100?
offset = 6400 mod 100 = 0
 - Ou seja, 6400 é o 1º endereço do bloco

Memória

Byte 0
Byte 1
...
Byte 63

Bloco 0

Byte 64
Byte 65
...
Byte 127

Bloco 1

Byte 128
Byte 129
...
Byte 191

Bloco 2

...

Bloco 3

...

Bloco 4

...

...

Políticas de mapeamento

Como mapear?

- O endereço da palavra é dado pelo *block_number* e pelo *offset*:

$$\text{addr} = \text{block_number} * \text{block_size} + \text{offset}$$

↑
potência de 2 (2^b)

Memória

Byte 0
Byte 1
...
Byte 63

Bloco 0

Byte 64
Byte 65
...
Byte 127

Bloco 1

Byte 128
Byte 129
...
Byte 191

Bloco 2

...

Bloco 3

...

Bloco 4

...

...

Políticas de mapeamento

Como mapear?

- A partir de um endereço, extraímos o *block_number* e o *offset*:

$\text{block_number} = \lfloor \text{addr} / \text{block_size} \rfloor$

$\text{offset} = \text{addr} \bmod \text{block_size}$

Vale $0 \leq \text{offset} \leq (\text{block_size} - 1)$

Memória

Byte 0
Byte 1
...
Byte 63

Bloco 0

Byte 64
Byte 65
...
Byte 127

Bloco 1

Byte 128
Byte 129
...
Byte 191

Bloco 2

...

Bloco 3

...

Bloco 4

...

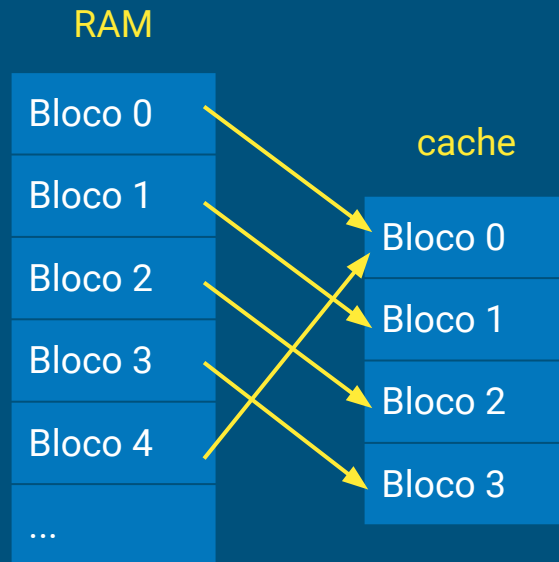
...

Políticas de mapeamento

Três políticas de mapeamento:

- Mapeamento direto
- Mapeamento associativo
- Mapeamento set-associativo

Objetivo: mapear bloco da RAM para bloco da cache (e vice-versa)

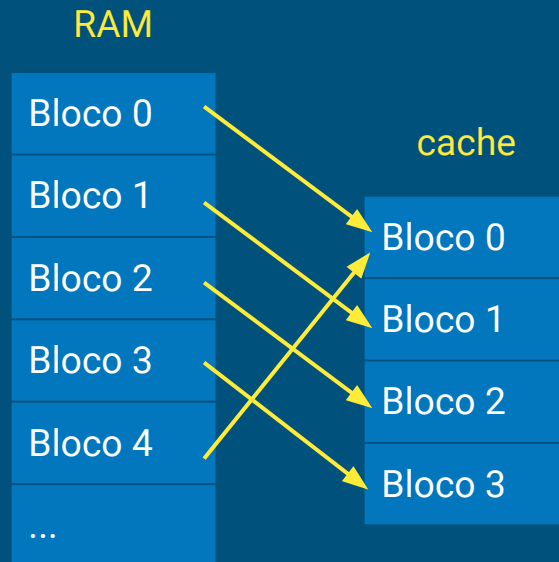


Políticas de mapeamento

Mapeamento direto

Suponha que a cache tenha 4 blocos:

- Bloco 0 da cache pode ser ocupado por
 - Blocos 0, 4, 8, 12... da RAM
- Bloco 1 da cache pode ser ocupado por
 - Blocos 1, 5, 9, 13... da RAM
- E assim por diante...



Políticas de mapeamento

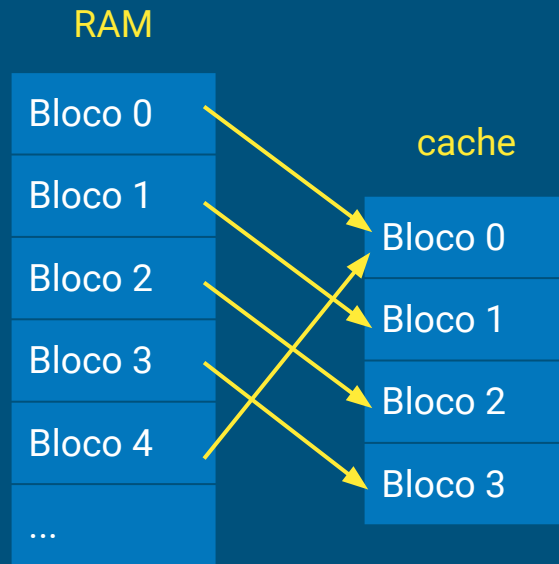
Mapeamento direto

Defina:

- `cache_size`: tamanho da cache
- `block_size`: tamanho do bloco
- `ram_block`: número do bloco na RAM
- `cache_block`: número do bloco na cache

Vale: $\text{cache_block} = \text{ram_block} \bmod n$

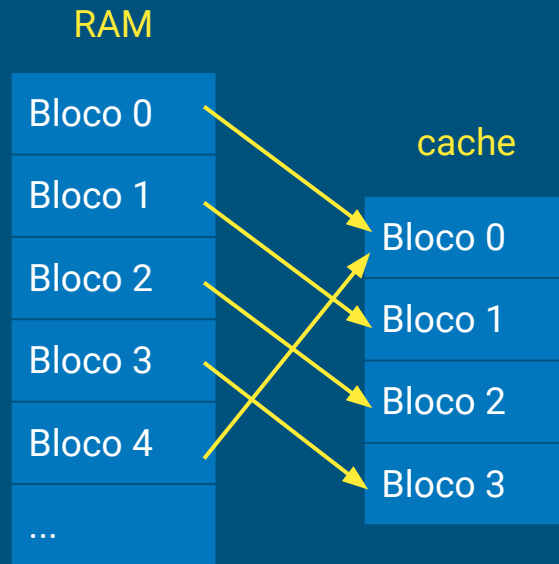
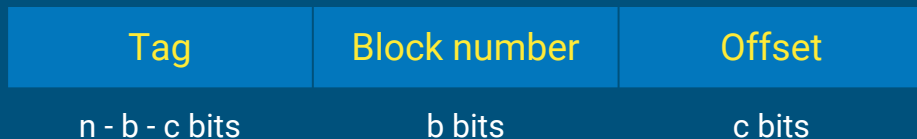
onde $n = \text{n}^\circ \text{ de blocos da cache } (\text{cache_size} / \text{block_size})$



Políticas de mapeamento

Como mapeamos os endereços individuais?

- Endereço de memória determina qual é o número de bloco e o offset
- Vamos tomar a notação binária do endereço de memória de n bits:

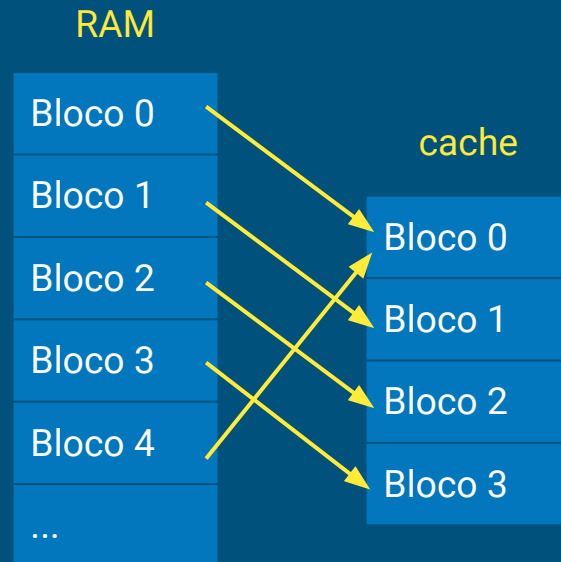
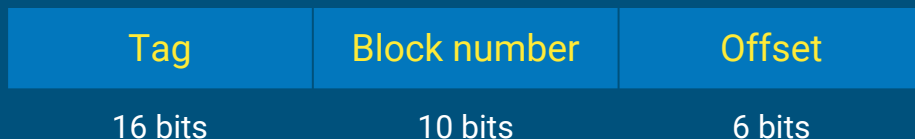


Políticas de mapeamento

Suponha:

- Endereços de 32 bits
- $\text{cache_size} = 64 \text{ KB} = 2^{16} \text{ bytes}$
- $\text{block_size} = 64 \text{ bytes} = 2^6 \text{ bytes}$ (offsets: 0, 1, ..., 63)
- $\text{num_blocks} = 1024 = 2^{10} \text{ blocos}$ (blocos: 0, 1, ..., 1023)

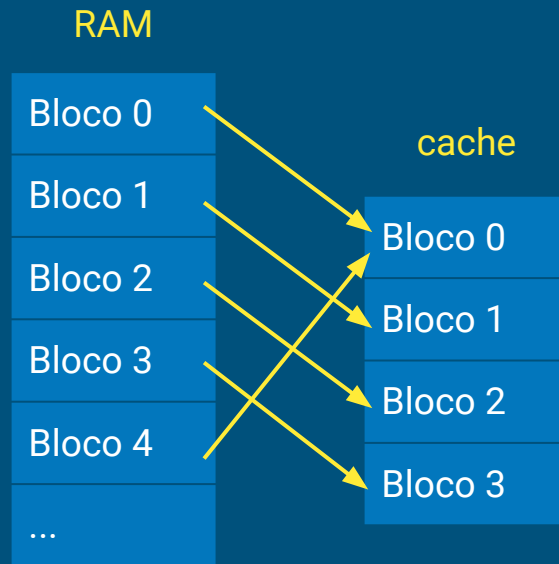
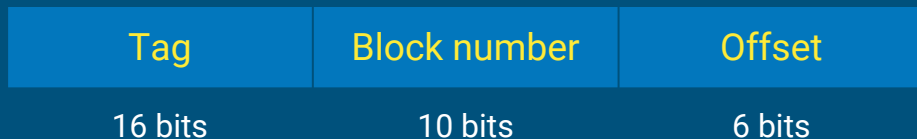
Endereço de memória:



Políticas de mapeamento

Conceitualmente:

- Block number e offset determinam a posição do endereço da RAM na cache
- O par (block number, offset) é conhecido como **índice** (bits menos significativos)

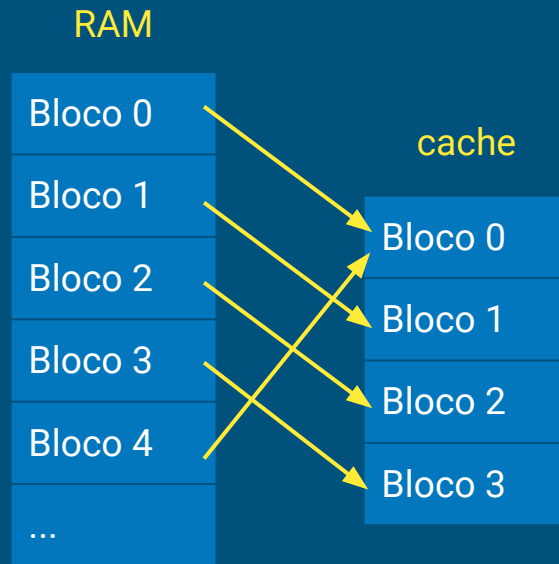


Políticas de mapeamento

Qual é a posição do endereço na cache?

- **índice** = bits menos significativos
- $\text{block_number} = \lfloor \text{índice} / \text{block_size} \rfloor$
 $\text{offset} = \text{índice} \bmod \text{block_size}$
- cache_size determina a qtd. de bits do índice

Tag	Block number	Offset
16 bits	10 bits	6 bits

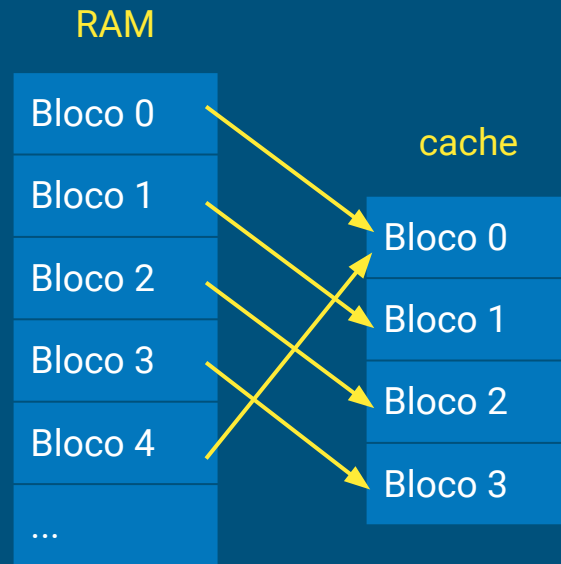


Políticas de mapeamento

Tratamento de colisões:

- Como a RAM tem mais capacidade que a cache, múltiplos blocos da RAM podem ser mapeados p/ o mesmo bloco da cache
- Para distinguir os endereços, usamos a **tag** (dato de controle)

Tag	Block number	Offset
16 bits	10 bits	6 bits



Políticas de mapeamento

Organização da memória cache (total de *num_blocks* blocos):



bit inicialmente vale 0 (por quê?)

Políticas de mapeamento

Cada bloco tem *block_size* endereços:

Posição no bloco

...

51

52

53

54

55

...

Dados

...

Dados (8 bits)

Dados (8 bits)

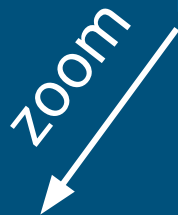
Dados (8 bits)

Dados (8 bits)

Dados (8 bits)

...

Dados do bloco



Políticas de mapeamento

Exemplo: quero buscar, na cache, o dado cujo endereço na RAM é addr.

1. Extraio a tag e o índice de addr
2. A partir do índice, encontro a localização de addr na cache
 - a. Comuto block_number e offset a partir do índice
3. Verifico se a tag do bloco bate com a tag do endereço
 - a. Se sim, e se o bit de validade do bloco for 1:
 - i. Temos um cache hit!
 - ii. Devolva o dado na posição offset do bloco p/ a CPU
 - b. Caso contrário, temos um cache miss.
 - i. Peça o dado para o próximo nível de memória
 1. Quando o bloco do dado chegar, defina seu bit de validade como 1
 2. Depois, devolva o dado à CPU (nota: até aqui CPU ficou esperando :)

Políticas de mapeamento

Resumindo:

- Na política de mapeamento direto, uma expressão matemática define qual é a localização, na cache, de um endereço de memória
- Um bloco de RAM é mapeado p/ um único bloco de cache
- Simples de implementar, porém apresenta pouca flexibilidade
 - Pode manter blocos vazios (não usa bem o espaço)
- Mais suscetível a cache misses

Políticas de mapeamento

Mapeamento associativo

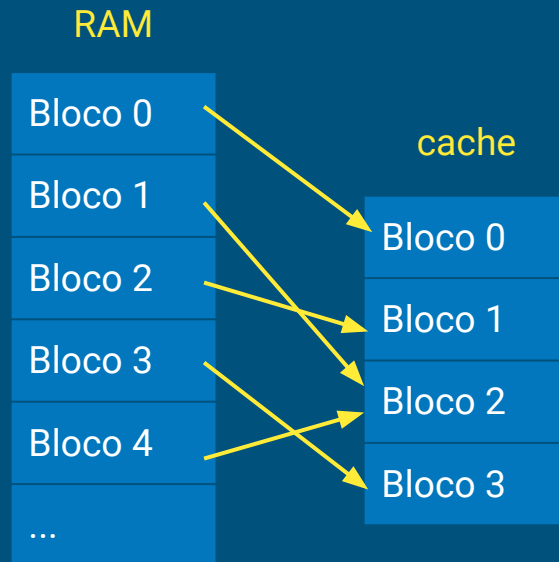
- Um bloco da memória pode ser transferido para qualquer bloco da cache
- Mais flexível que o mapeamento direto
 - Também conhecido como mapeamento completamente associativo
 - Usa melhor o espaço
- Cada bloco da cache armazena:
 - Tag: dado de controle
 - Dados do bloco
 - Bit de validade

Políticas de mapeamento

Suponha:

- Endereços de 32 bits
- $\text{cache_size} = 64 \text{ KB} = 2^{16} \text{ bytes}$
- $\text{block_size} = 64 \text{ bytes} = 2^6 \text{ bytes}$ (offsets: 0, 1, ..., 63)
- $\text{num_blocks} = 1024 = 2^{10} \text{ blocos}$

Endereço de memória:



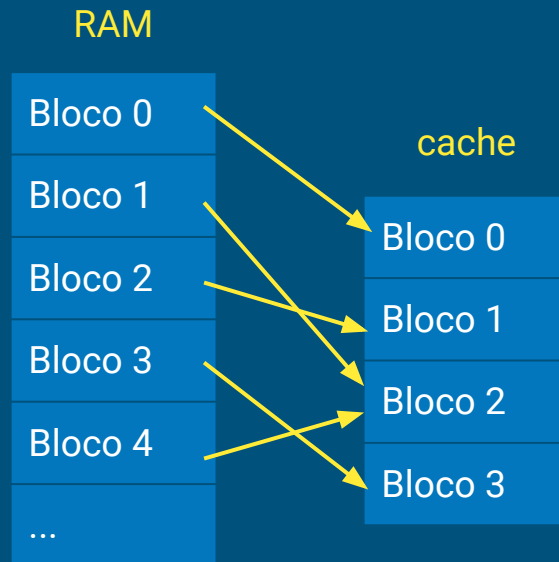
Políticas de mapeamento

Extrair **tag** e **offset** do endereço addr: (sup. block_size = 64)

- **tag** = $\lfloor \text{addr} / 64 \rfloor$
- **offset** = $\text{addr} \bmod 64$

Nota: block_size = 64 = 2^6 , potência de 2

offset é a posição do endereço no bloco da cache (0, 1, ..., 63)



Políticas de mapeamento

Mapeamento associativo

- Como encontrar o valor no endereço de memória addr? $\text{addr} = (\text{tag}, \text{offset})$
- Compare a tag do endereço com todas as tags da cache - processo de **lookup**
 - Se não encontrei a tag, temos cache miss
 - Se o bloco for inválido, temos cache miss
 - Se tag certa e bloco válido, então cache hit
 - Pegue o dado na posição offset do bloco
- Maior complexidade dos componentes
 - Comparação simultânea de muitas tags
 - Processo caro

Cache:

...

Tag	Dados do bloco	Válido?
...		

Políticas de mapeamento

Mapeamento set-associativo

- Em vez de comparar a tag do endereço requisitado com todas as tags da cache, comparo apenas com um subconjunto específico delas
- Cada bloco da memória pode ser colocado em um único conjunto de blocos da cache
 - ... mas dentro desse conjunto, o bloco pode ser colocado em qualquer lugar
- Solução intermediária (“*mistura*”)

Políticas de mapeamento

Mapeamento set-associativo

- Uma expressão matemática define qual é o conjunto de blocos a que um endereço pertence, mas não a qual bloco
 - Os blocos dentro de um conjunto são dispostos sequencialmente
- O bloco correto é encontrado por meio de um processo de **lookup** *dentro do conjunto*
- Para implementar, vamos estabelecer que cada conjunto tem k blocos ($k = 2^n$)

tag	Bloco 0	v	Set 0
tag	Bloco 1	v	
tag	Bloco 2	v	
tag	Bloco 3	v	
tag	Bloco 4	v	Set 1
tag	Bloco 5	v	
tag	Bloco 6	v	
tag	Bloco 7	v	
...

Políticas de mapeamento

Mapeamento set-associativo

Suponha:

- $\text{cache_size} = 64 \text{ KB} = 2^{16} \text{ bytes}$
- $\text{block_size} = 64 \text{ bytes} = 2^6 \text{ bytes}$
- $k = 4 \text{ blocos por conjunto} = 2^2 \text{ blocks/set}$

Qual é a quantidade de blocos?

$$\text{num_blocks} = \text{cache_size} / \text{block_size} = 2^{10}$$

Quantos conjuntos existem?

$$\text{num_sets} = \text{num_blocks} / k = 2^8$$

tag	Bloco 0	v	Set 0
tag	Bloco 1	v	
tag	Bloco 2	v	
tag	Bloco 3	v	
tag	Bloco 4	v	Set 1
tag	Bloco 5	v	
tag	Bloco 6	v	
tag	Bloco 7	v	
...

Políticas de mapeamento

Mapeamento set-associativo

Conhecido como mapeamento k-way associativo

Se $k = 4$, temos um **mapeamento 4-way associativo**

- São k blocos por conjunto
- **Lembrar:** k é sempre potência de 2

Se $k = 1$, temos o mapeamento direto

Se $k = \text{num_blocks}$, temos apenas 1 conjunto:
mapeamento (completamente) associativo

tag	Bloco 0	v	Set 0
tag	Bloco 1	v	
tag	Bloco 2	v	
tag	Bloco 3	v	
tag	Bloco 4	v	Set 1
tag	Bloco 5	v	
tag	Bloco 6	v	
tag	Bloco 7	v	
...

Políticas de mapeamento

Mapeamento set-associativo

Suponha:

- Mapeamento 4-way associativo
- $\text{cache_size} = 64 \text{ KB} = 2^{16} \text{ bytes}$
- $\text{block_size} = 64 \text{ bytes} = 2^6 \text{ bytes}$
- $\text{num_sets} = 256 = 2^8 \text{ conjuntos}$

Endereço de memória:

Tag	Set number	Offset
18 bits	8 bits	6 bits

tag	Bloco 0	v	Set 0
tag	Bloco 1	v	
tag	Bloco 2	v	
tag	Bloco 3	v	
tag	Bloco 4	v	Set 1
tag	Bloco 5	v	
tag	Bloco 6	v	
tag	Bloco 7	v	
...

Políticas de mapeamento

Mapeamento set-associativo

- Como encontrar o valor no endereço de memória $addr$? $addr = (tag, set\ number, offset)$
- Comparo a tag de $addr$ com as tags do conjunto set_number - processo de **lookup**
 - Se não encontrei a tag, temos cache miss
 - Se o bloco for inválido, temos cache miss
 - Se tag certa e bloco válido, então cache hit
 - Pegue o dado na posição $offset$ do bloco
- Lookup mais simples, mas ainda flexível

tag	Bloco 0	v	Set 0
tag	Bloco 1	v	
tag	Bloco 2	v	
tag	Bloco 3	v	
tag	Bloco 4	v	Set 1
tag	Bloco 5	v	
tag	Bloco 6	v	
tag	Bloco 7	v	
...

Políticas de mapeamento

- No mapeamento completamente associativo,
 - Um bloco pode ser colocado em qualquer lugar da cache
- No mapeamento k-way associativo,
 - Um bloco pode ser colocado em qualquer lugar de seu conjunto
- Como assim “qualquer lugar”? Onde colocar o bloco?
 - Se o bit de validade de um bloco for 0, pode-se usar esse bloco
 - E se o bit de validade de todos os blocos for 1?
 - Há a necessidade de **substituição** de um bloco
 - A **política de substituição** diz onde colocar o bloco

Políticas de substituição

Políticas de substituição

Políticas de substituição:

- Substituição aleatória
- Least Recently Used (LRU)
- Least Frequently Used (LFU)
- Outras
 - N-MRU (not most frequently used), etc.
 - Múltiplas políticas

Objetivo: minimizar os cache misses.

Políticas de substituição

Substituição aleatória

- Quando for necessário substituir um bloco,
 - Escolha qualquer um
- Simples de implementar
 - Não é necessário manter qualquer dado de controle sobre os blocos
 - Pode ser combinado a outras políticas de substituição
- Não leva em consideração a localidade temporal

Políticas de substituição

Least Recently Used (LRU)

- Quando for necessário substituir um bloco,
 - Escolha o bloco menos recentemente usado
- Várias implementações possíveis
 - Ex: manter um contador de “tempo ocioso” para cada bloco
 - Sempre que um bloco for acessado,
 - Incremente em 1 o “tempo ocioso” de todos os blocos
 - Reinicie o contador do bloco acessado
 - Ao substituir um bloco, escolha o “mais ocioso”

Políticas de substituição

Least Frequently Used (LFU)

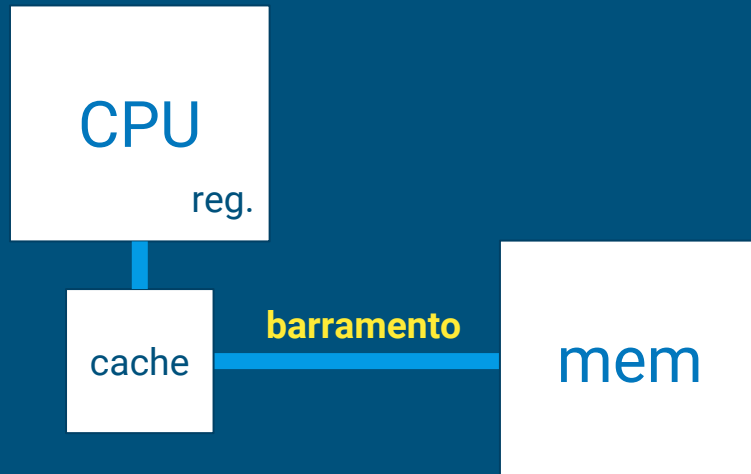
- Quando for necessário substituir um bloco,
 - Escolha o bloco menos frequentemente utilizado (i.e., “aquele que recebe menos atenção”)
- Exemplo de implementação:
 - Manter um contador de frequência para cada bloco
 - Sempre que um bloco for acessado (“receber atenção”),
 - Incremente em 1 seu contador de frequência
 - Ao substituir um bloco, escolha aquele cujo contador de frequência seja o menor
 - Se houver empate, pode substituir o “mais ocioso”
 - Fila de prioridades
 - Nota: poluição da cache

Políticas de escrita

Políticas de escrita

Como fazemos para escrever na memória?

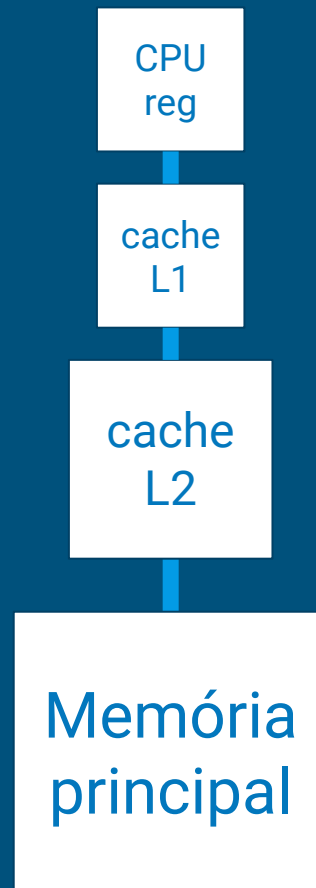
- Agora a CPU se comunica primeiramente com a cache
- Se o endereço no qual queremos escrever estiver na cache: **write hit**
- Se o endereço não estiver na cache, então: **write miss**



Políticas de escrita

Se houver **write hit**,

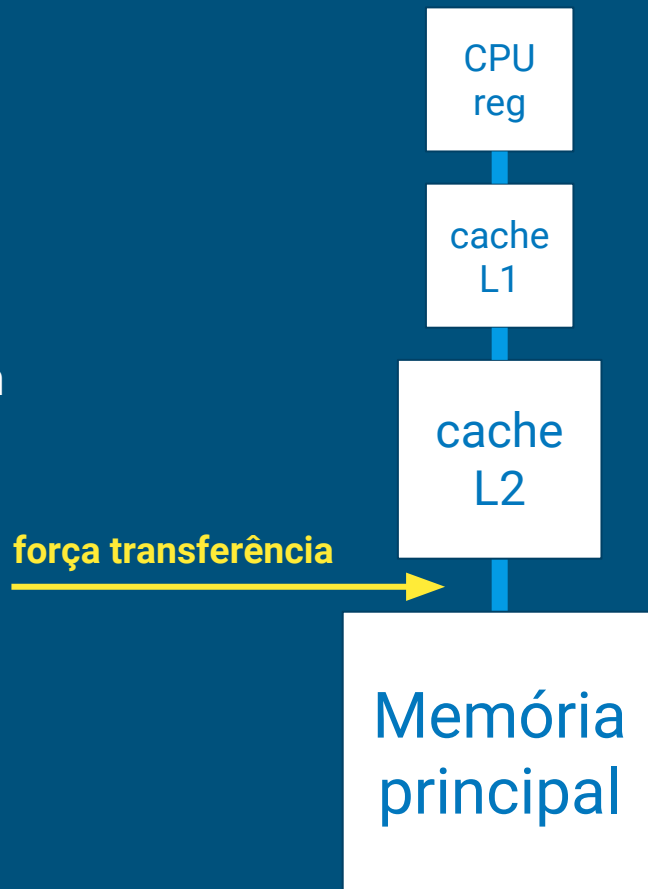
- Primeiramente, escreva o dado na cache
- Hierarquia de memória
 - Cache L1 é subconjunto da L2
 - Cache L2 é subconjunto da L3
 - Cache L3 é subconjunto da RAM
- Não pode haver inconsistência entre os dados!
 - Replicar os dados



Políticas de escrita

Política **write through**,

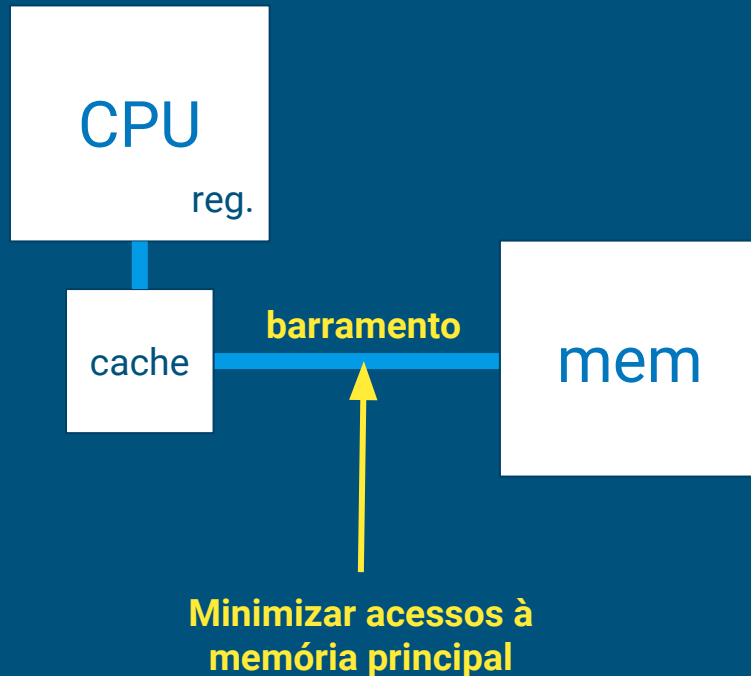
- Força a replicação de todas as escritas em todos os níveis da memória de uma só vez
- Resolve o problema de inconsistência
 - Fácil de implementar
- Desvantagens?
 - Tempos de acesso
 - Ocupa o barramento da memória principal



Políticas de escrita

Política **write back**,

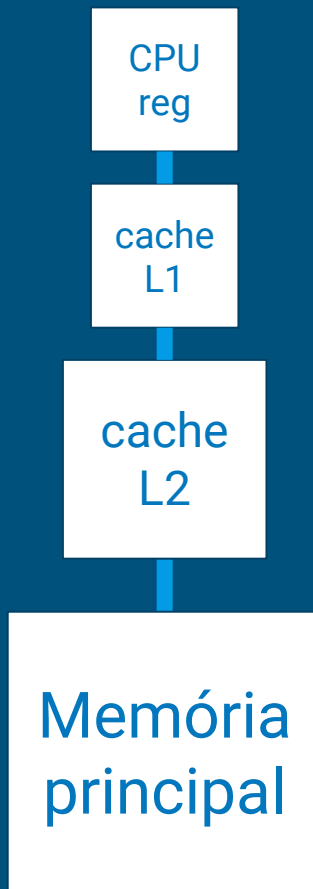
- Escreva o dado apenas na cache
 - Marque o bloco como “dirty” (1 bit)
- Quando um bloco marcado como “dirty” for substituído, replique-o nos outros níveis da memória
- Visa minimizar os acessos à RAM
 - Gera inconsistência temporária
 - Problema para dispositivos de E/S



Políticas de escrita

Se houver **write miss**, duas possibilidades:

- Política **write around** (“*write no-allocate*”)
 - Ignore o write miss
 - Simplesmente escreva o dado no nível seguinte (memória)
 - Bom quando não há acesso (R/W) ao dado logo em seguida
- Política **allocate on write**
 - Carregue o bloco correspondente ao dado na cache
 - Escreva o dado atualizado na cache
 - Bom se o dado for utilizado novamente logo em seguida

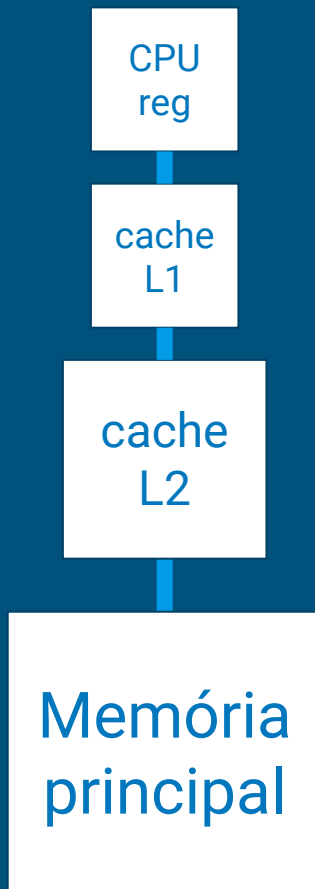


Políticas de escrita

Combinações típicas:

- Política **write through** + **write around**
 - Escreva diretamente
 - Replique assim que escrever
- Política **write back** + **allocate on write**
 - Escreva o dado na cache
 - Replique quando for substituir o bloco

Dúvidas?



Parte prática