"Todos os problemas na Ciência da Computação podem ser resolvidos adicionando um nível extra de indireção, exceto o problema de se ter muitos níveis de indireção" (David J. Wheeler).

Hierarquia de Memórias

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Precisamos de Memória

Precisamos de memória para armazenar as instruções de programas e os dados.

Todos desejamos uma quantidade infinita de memória.

Por questões físicas e orçamentárias não podemos ter uma memória muito grande.

Precisamos de Memória

Além de grande, desejamos uma memória rápida.

Pelo menos tão rápida quanto a CPU consegue solicitar informações.

Precisamos de Memória

Além de grande, desejamos uma memória rápida.

Pelo menos tão rápida quanto a CPU consegue solicitar informações.

Mas se não conseguimos “pagar” nem por uma memória grande, uma memória **grande** e **rápida** se torna impraticável.

O Quão Rápida a Memória Precisa Ser?

Segundo Henessy e Patterson (2017).

Um processador Intel i7 7600 - 4 núcleos @4,1GHz em seu pico de operação:

- Faz duas referências de 64 bits a memória por core a cada ciclo de clock.
- Pode também demandar até 12,8 *10^9 instruções de 128 bits por segundo.

O Quão Rápida a Memória Precisa Ser?

Segundo Henessy e Patterson (2017).

Um processador Intel i7 7600 - 4 núcleos @4,1GHz em seu pico de operação:

Faz duas referências de 64 bits a memória por core a cada ciclo de clock.

Pode também demandar até 12,8 *10^9 instruções de 128 bits por segundo.

A taxa de transferência entre a memória principal e o processador deveria ser de **409,6 GiB/s** para suprir o processador.

O Quão Rápida a Memória Precisa Ser?

Segundo Henessy e Patterson (2017).

Um processador Intel i7 7600 - 4 núcleos @4,1GHz em seu pico de operação:

- Faz duas referências de 64 bits a memória por core a cada ciclo de clock.
- Pode também demandar até 12,8 *10^9 instruções de 128 bits por segundo.

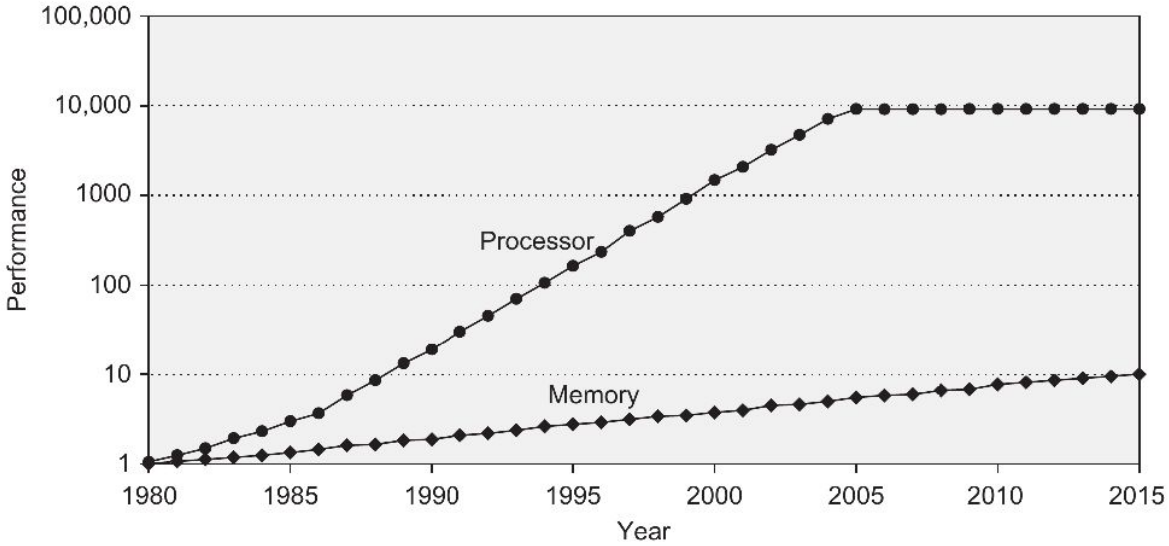
A taxa de transferência entre a memória principal e o processador deveria ser de **409,6 GiB/s** para suprir o processador.

As memórias DRAM comumente utilizadas em conjunto com esse processador, configuradas em “dual chanel” conseguem suprir apenas **34,1GiB/s**.

Diferença de desempenho

Considerando apenas “1 core”

Escala logarítmica!

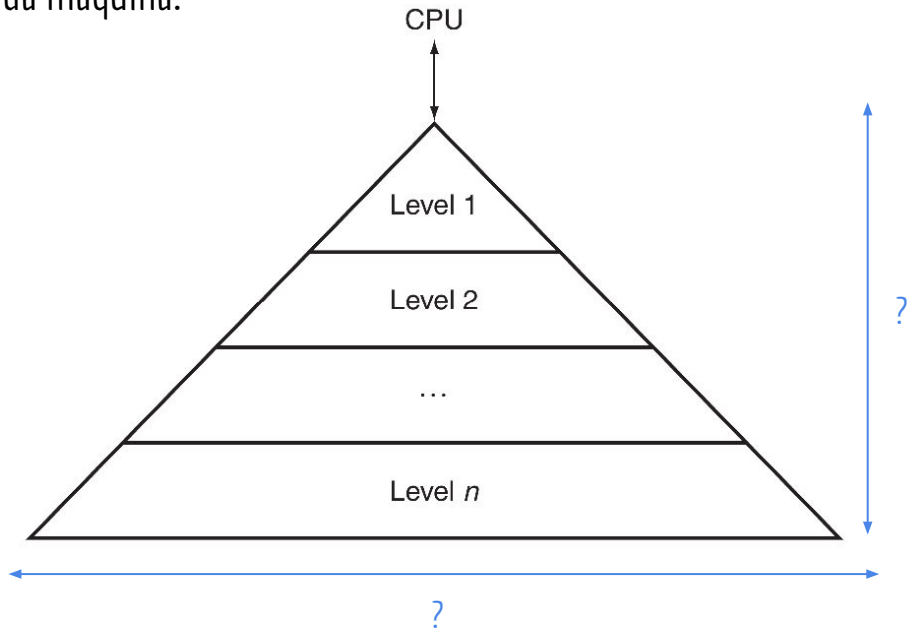


Diferença entre a velocidade em que o processador consegue processar instruções versus a velocidade em que a memória consegue suprir essas instruções. (Henessy, Patterson; 2019).

Hierarquia de Memórias

Para amenizar esses problemas, é usada uma hierarquia de memórias, com *n* níveis.

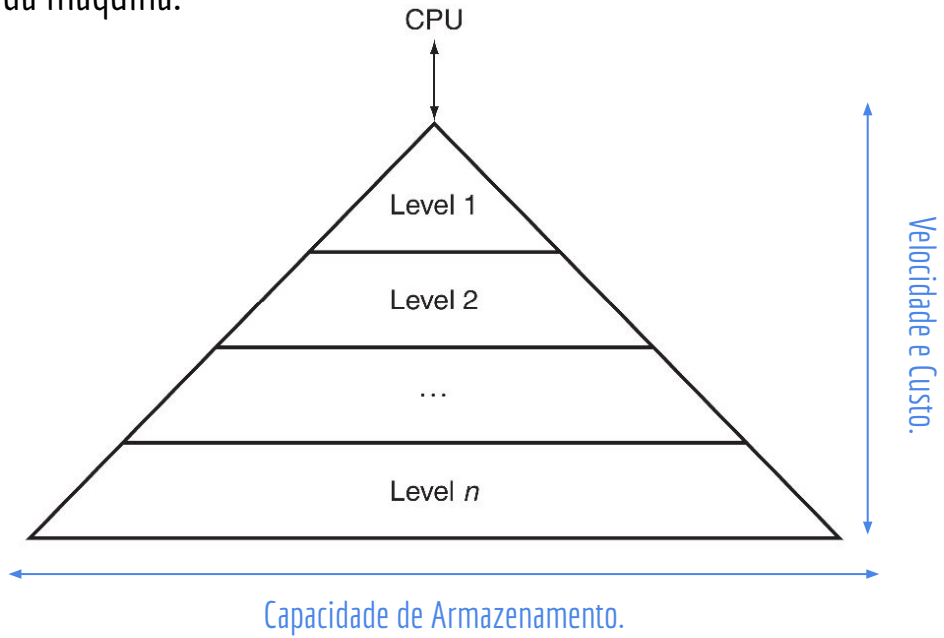
A quantidade de níveis vai depender da arquitetura da máquina.



Hierarquia de Memórias

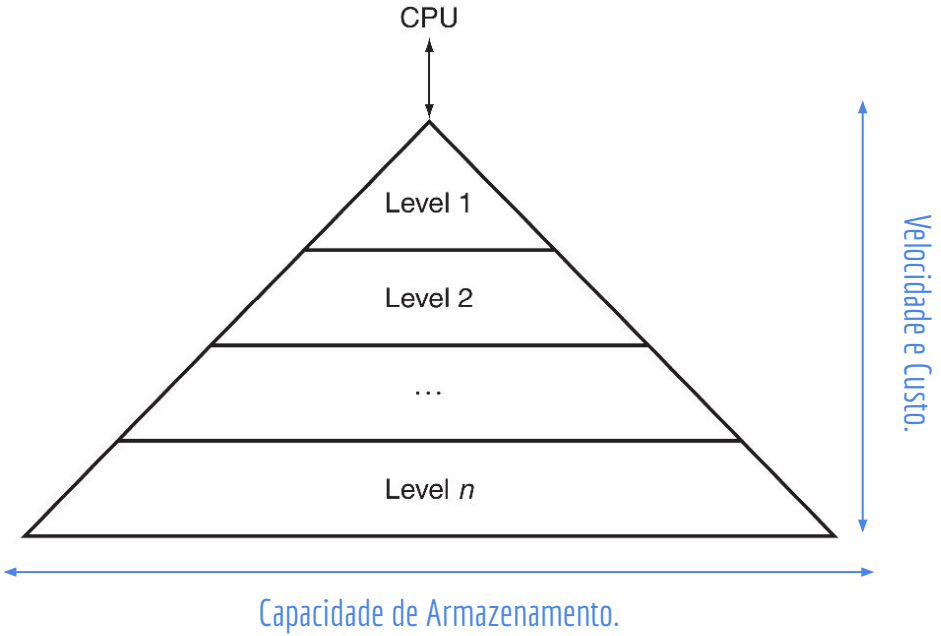
Para amenizar esses problemas, é usada uma hierarquia de memórias, com *n* níveis.

A quantidade de níveis vai depender da arquitetura da máquina.



Hierarquia de Memórias

Tomando como base o seu computador x86-64, você consegue especificar quais memórias estão em quais níveis?



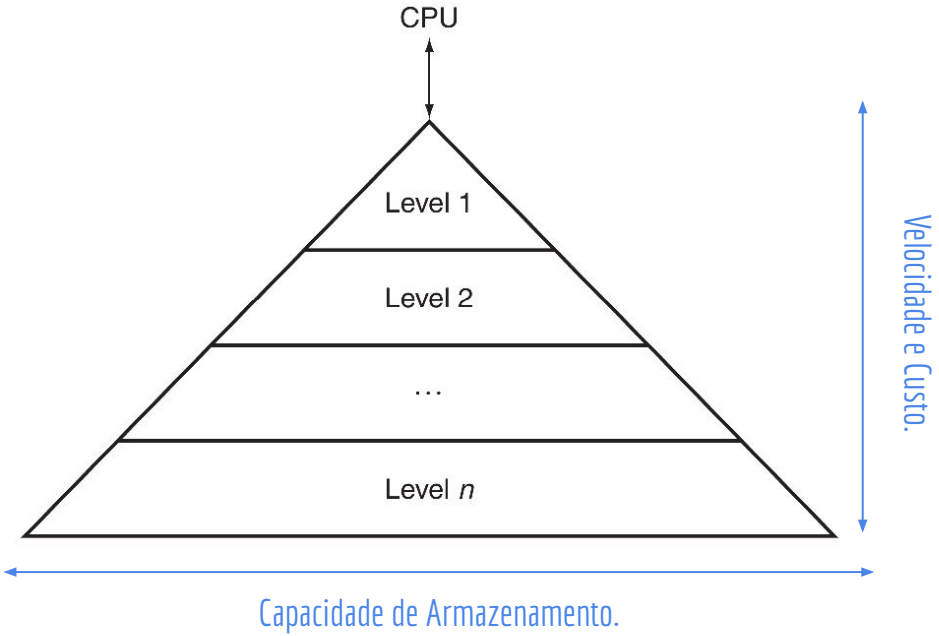
Hierarquia de Memórias

Tomando como base o seu computador x86-64, você consegue especificar quais memórias estão em quais níveis?

Caches.

Memória Principal (DRAM).

HD ou SSD (ou ambos).



Acesso dos níveis

Importante

Na maioria das implementações, temos uma hierarquia real

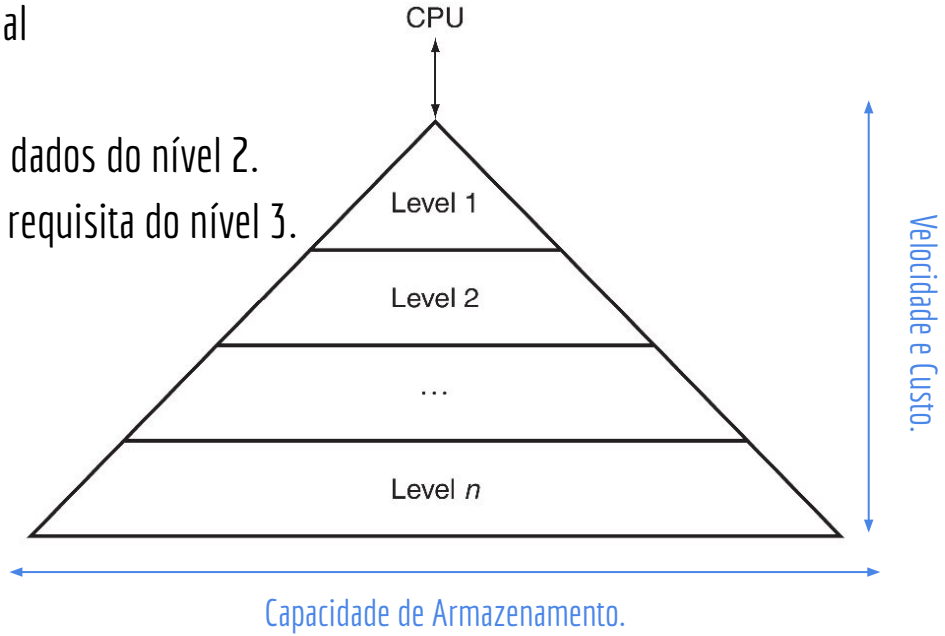
A CPU acessa os dados apenas no nível 1.

O nível 1 serve dados para a CPU, e requisita dados do nível 2.

O nível 2 serve dados para o nível 1, e requisita do nível 3.

...

Não pulamos níveis em uma hierarquia real.



Uma questão de custos

Tecnologia	Tempo de acesso típico
SRAM	0.5 – 2.5 ns
DRAM	50 – 70 ns
Flash	5.000 – 50.000 ns
Disco Magnético	5.000.000 – 20.000.000 ns

Patterson, Hennessy. Computer Organization and Design RISC-V Edition. 2020 (Ed. Americana).

Uma questão de custos

Tecnologia	Tempo de acesso típico	Dólares por GiB em 2020
SRAM	0.5 – 2.5 ns	$500 – $1000
DRAM	50 – 70 ns	$3 – $6
Flash	5.000 – 50.000 ns	$0,03 – $0,12
Disco Magnético	5.000.000 – 20.000.000 ns	$0,01 – $0,02

Patterson, Hennessy. Computer Organization and Design RISC-V Edition. 2020 (Ed. Americana).

Compare

Tecnologia	Tempo de acesso típico	Dólares por GiB em 2020
SRAM	0.5 – 2.5 ns	$500 – $1000
DRAM	50 – 70 ns	$3 – $6
Flash	5.000 – 50.000 ns	$0,03 – $0,12
Disco Magnético	5.000.000 – 20.000.000 ns	$0,01 – $0,02

Patterson, Hennessy. Computer Organization and Design RISC-V Edition. 2020 (Ed. Americana).

Tecnologia	Tempo de acesso típico	Dólares por GiB em 2012
SRAM	0.5 – 2.5 ns	$500 – $1000
DRAM	50 – 70 ns	$10 – $20
Flash	5.000 – 50.000 ns	$0,75 – $1,00
Disco Magnético	5.000.000 – 20.000.000 ns	$0,05 – $0,1

Patterson, Henessy 5a ed (2014 – Edição Americana).

Compare

Tecnologia	Tempo de acesso típico	Dólares por GiB em 2020
SRAM	0.5 – 2.5 ns	$500 – $1000
DRAM	50 – 70 ns	$3 – $6
Flash	5.000 – 50.000 ns	$0,03 – $0,12
Disco Magnético	5.000.000 – 20.000.000 ns	$0,01 – $0,02

Patterson, Hennessy. Computer Organization and Design RISC-V Edition. 2020 (Ed. Americana).

Tecnologia	Tempo de acesso típico	Dólares por GiB em 2012
SRAM	0.5 – 2.5 ns	$500 – $1000
DRAM	50 – 70 ns	$10 – $20
Flash	5.000 – 50.000 ns	$0,75 – $1,00
Disco Magnético	5.000.000 – 20.000.000 ns	$0,05 – $0,1

Patterson, Henessy 5a ed (2014 – Edição Americana).

Tecnologia	Tempo de acesso típico	Dólares por GiB em 2008
SRAM	0.5 – 2.5 ns	$2000 – $5000
DRAM	50 – 70 ns	$20 – $75
Disco Magnético	5.000.000 – 20.000.000 ns	$0,2 – $2

Patterson, Henessy 4a ed (2011 – Edição Americana).

Tecnologias envolvidas

Considerando a hierarquia comumente encontrada nos x86-64.

Caches: SRAM (static random access memory).

Construídas utilizando transistores com retroalimentação (de forma similar à flip-flops).

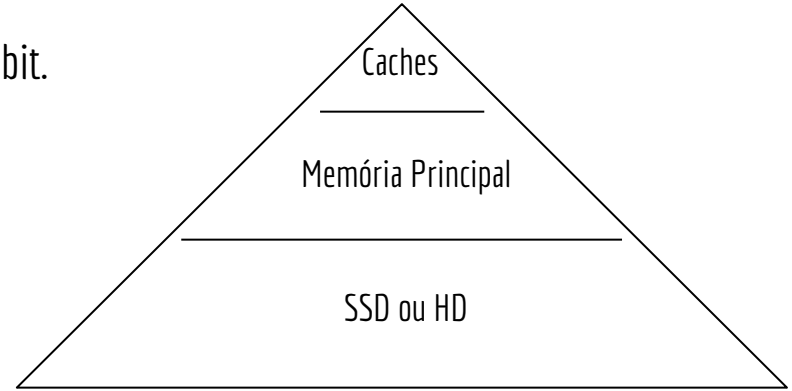
Memória Principal: DRAM (dynamic random access memory).

Memórias capacitivas.

Capacitores carregados/descarregados indicam o estado do bit.

SSD ou HD.

Flashes NAND ou NOR, ou discos magnéticos.

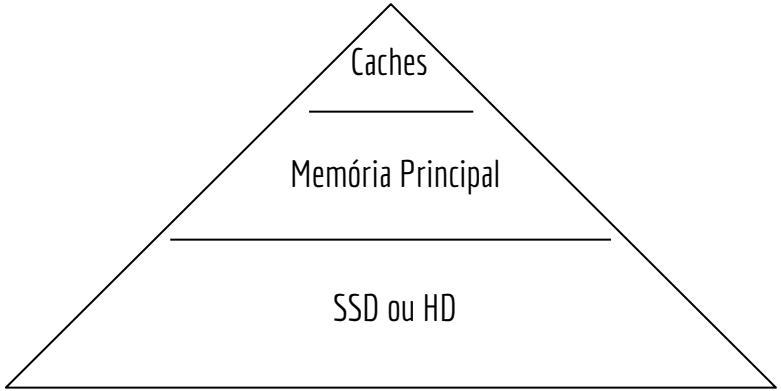


A Memória Principal

A memória principal fica entre os níveis da cache e armazenamento permanente.

Cache -› desejamos mais velocidade.

Armazenamento permanente -› desejamos mais espaço.



Armazenamento permanente

Armazenamento permanente.

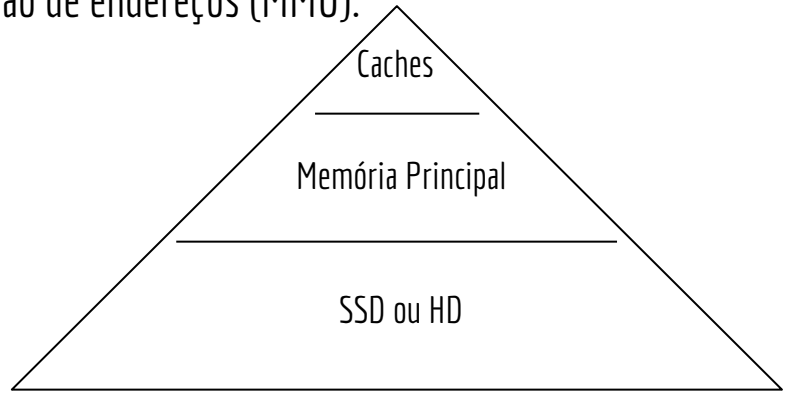
Utilizar o armazenamento permanente como se fosse a memória principal implica na utilização do conceito de paginação.

Tema para a disciplina de Sistemas Operacionais.

Envolve muito do hardware.

Exemplo: uso de interrupções e de unidades de tradução de endereços (MMU).

Não se aventure em S.O. antes de aprender A.O.C.



Veremos o básico sobre paginação no futuro.

Memória Cache

Começando com uma memória cache básica.

Enquanto os computadores atuais têm comumente 16GiB de memória principal (DRAM), eles possuem cerca de 40MiB de cache (SRAM).

É comum encontrarmos processadores para servidores com quase 100MiB de cache, mas dificilmente passamos disso.

A diferença entre a capacidade ainda é gigantesca.

Transparência

Desejamos transparência

O programador sempre vai assumir que o programa está na memória principal, e não vai programar explicitamente para os demais níveis.

Se o programa realmente está na memória principal, ou está na cache, ou no disco é problema do:

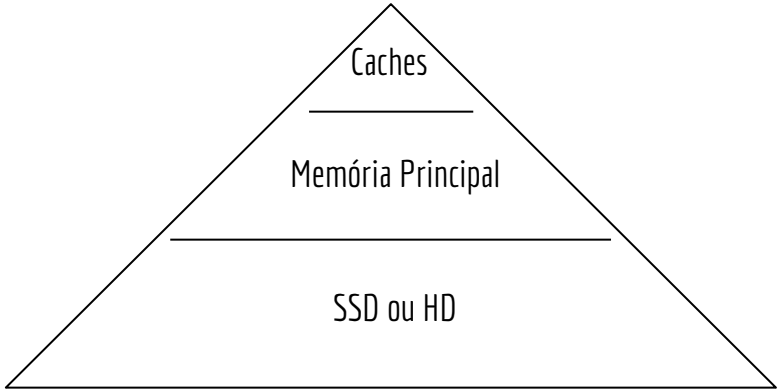
- Hardware no caso da memória cache.
- Hardware + Software (S.O.) no caso do Disco/SSD (paginação).

Memória Cache

Como podemos mapear a memória principal para dentro da cache de maneira automática?

Que regras seguir? Qual dado é importante?

Ideias?



Localidade temporal e espacial

Se analisarmos vários programas, veremos que temos dois tipos de localidade:

Localidade temporal.

Localidade espacial.

Localidade Temporal

Se um item da memória é acessado, é muito provável que ele seja acessado novamente num futuro próximo.

Exemplo:

Quando armazenamos um item na pilha para fazer uma chamada de função, vamos ler ele novamente quando a função terminar.

Localidade Espacial

Se acessamos um item da memória, é muito provável que seus vizinhos também sejam acessados.

Exemplos:

Nossos programas são (quase) sequencias, então quando a instrução *i* é carregada, é provável que *i+1*, *i+2*, ... também sejam úteis.

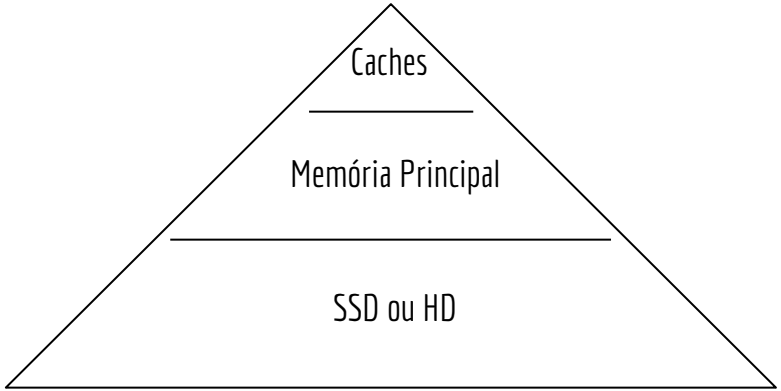
Quando operamos no elemento *k* de um vetor, o elementos *k+1*, *k+2*, ... provavelmente também serão úteis.

Memória Cache

A memória cache é muito menor que a memória principal.

Necessário tirar vantagem das localidades espacial e temporal.

Manter na cache os **dados utilizados mais recentemente, e seus vizinhos**.

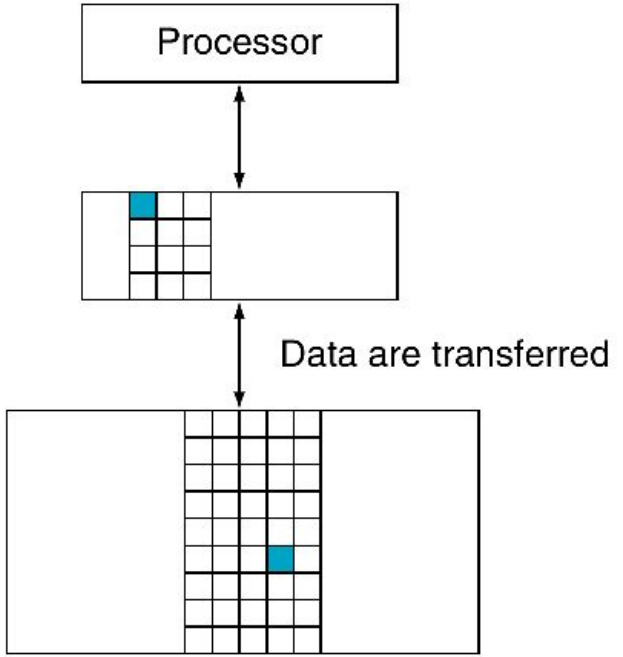


Mapeamento

Dividir as memórias (principal e cache) em blocos.

Blocos de mesmo tamanho.

Obviamente, a memória cache suporta menos blocos que a principal.



Requisitando dados

Quando o processador requisita um dado e:

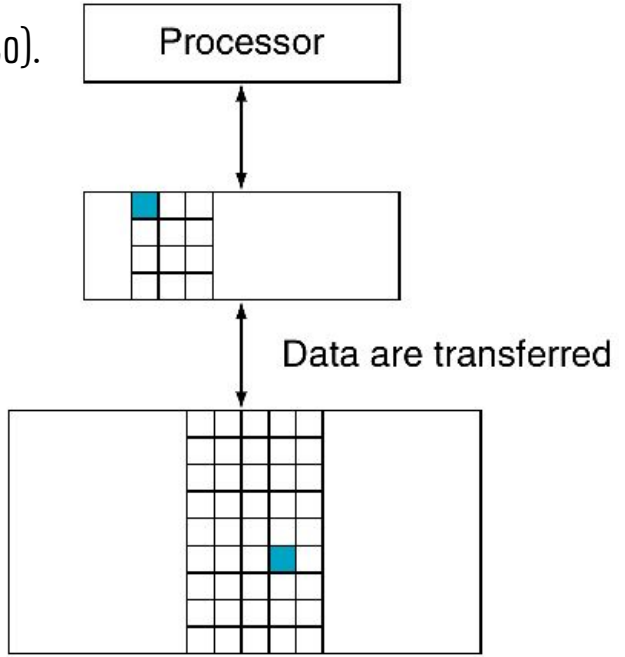
O dado está presente em um bloco no nível de memória mais alto (cache nesse caso).

Temos um **hit** (acerto).

Se o dado não está presente.

Temos um **miss** (erro).

Necessário tempo para carregar o bloco.



Métricas de Desempenho

hit rate

Também chamado de hit ratio.

Fração dos acessos a memória nos quais tivemos um hit.

hitRate = hits/totalAcessos

miss rate

Também chamado de miss ratio.

Fração dos acessos a memória nos quais tivemos um miss.

MissRate = 1 - hitRate

Métricas de Desempenho

Hit time

Tempo de hit.

Tempo necessário para acessar um bloco de memória no nível alto, considerando ainda o custo para verificar se temos ou não um hit.

Miss penalty

Penalidade de miss.

Tempo necessário para carregar/substituir um bloco do nível mais alto por um do nível logo abaixo.

Cache

Consideraremos como cache a memória que está no nível mais alto.

No processador MIPS32, poderíamos simplesmente substituir as memórias de dados e instruções por caches.

No entanto, em um âmbito mais geral, cache também pode se referir a qualquer sistema de armazenamento que se beneficia da localidade de acesso.

Ex.: Muitos HDs possuem uma pequena cache interna, onde os dados mais requisitados são salvos.

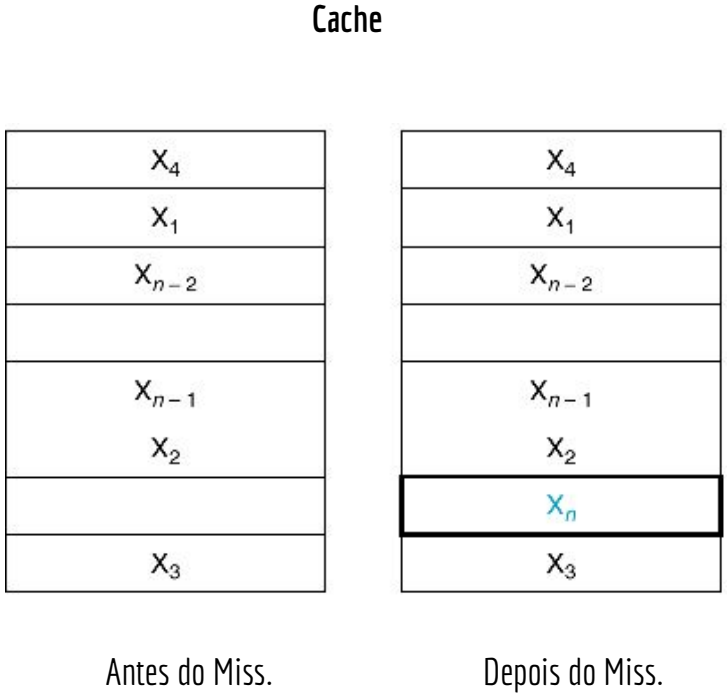
Cache diretamente mapeada

Considere que:

- O processador sempre requisita **uma palavra**.
- Cada bloco da cache armazena exatamente **uma palavra**.

Quanto o processador requisita uma palavra X_n que **não** está na cache.

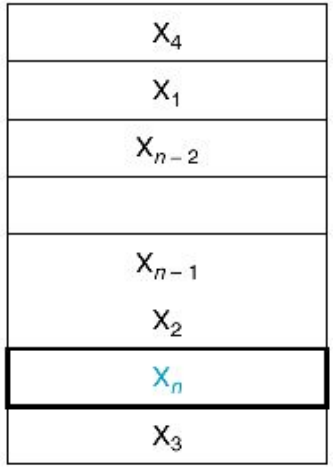
- Ocorreu um miss.
- A palavra então é carregada para a cache.



Cache diretamente mapeada

Como encontrar um item nessa cache?

Como saber se um item está na cache ou não?



Cache diretamente mapeada

Em uma cache **diretamente mapeada**.

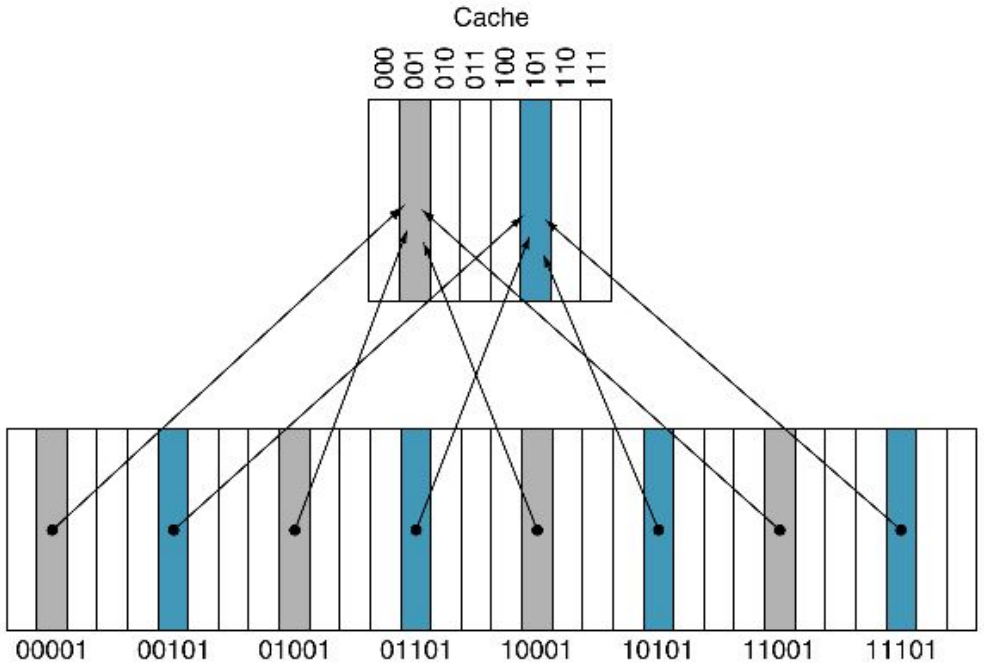
Cada endereço de memória é mapeado para exatamente uma posição na cache.

Construímos nossa cache para suportar uma potência de 2 de palavras.

- O mapeamento se torna trivial.
- Para uma cache de 2^n palavras, utilizamos os *n* bits mais baixos dos endereços para mapear a cache.

Exemplo

Cache de 2^3=8 entradas.
Utilizamos os 3 bits menos significativos dos endereços de memória para endereçar a cache.



Exemplo

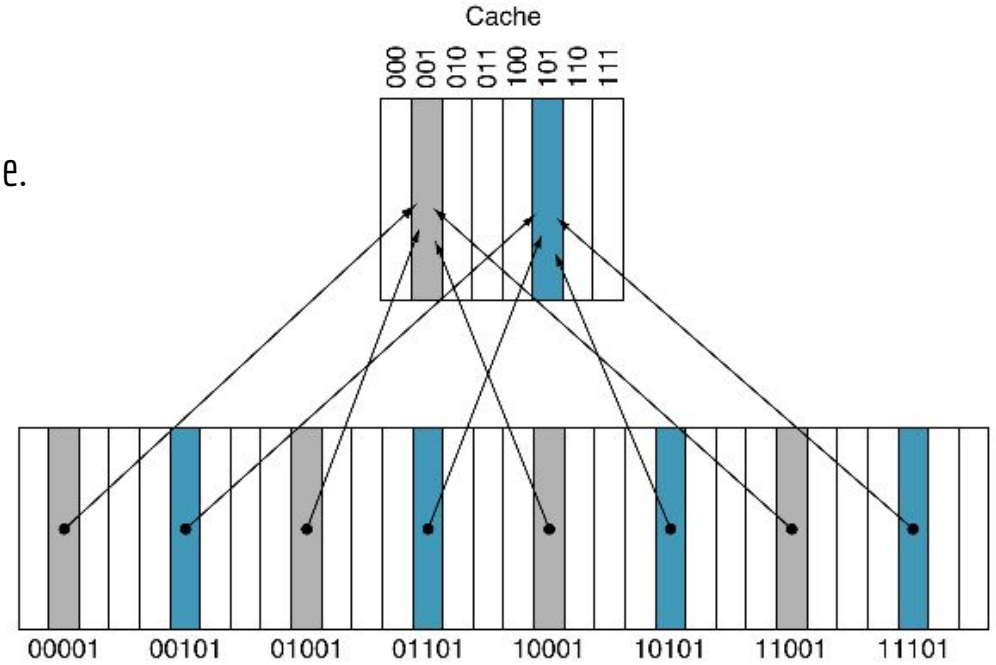
Se o processador solicitar o endereço 00101_2.

Sabendo que a cache possui 2^3 entradas.

O processador busca o dado no endereço 101_2 da cache.

A palavra pode ou não estar neste endereço.

O endereço da cache, e a palavra contida nela são suficientes para saber se a palavra está na cache?



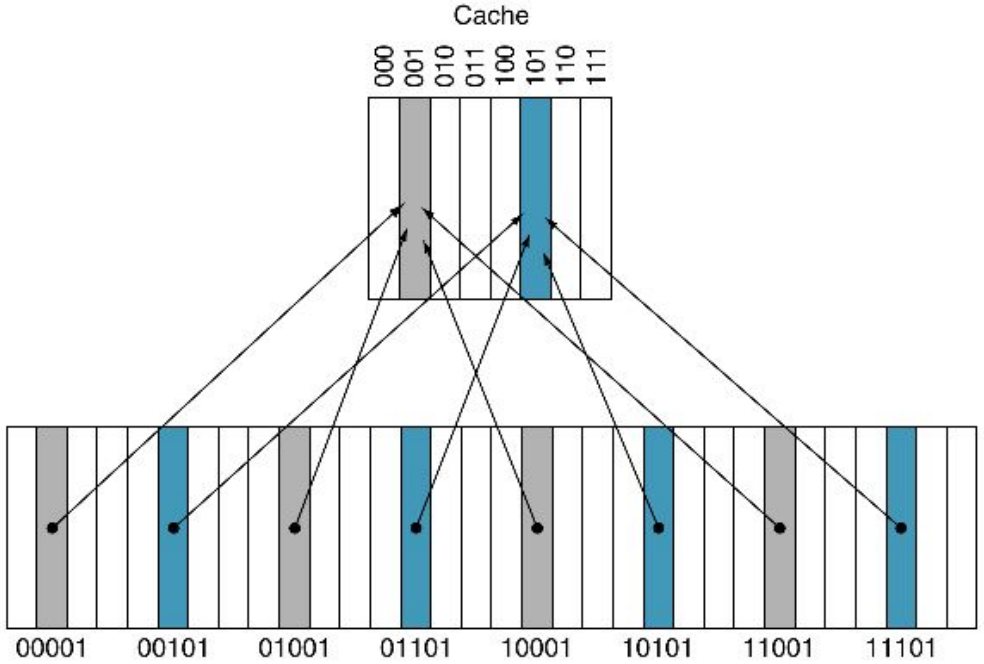
Exemplo

O processador busca o dado no endereço 101_2 da cache.

Qual palavra da memória está nesse endereço?

00101_2
01101_2
10101_2
11101_2
...

Soluções?



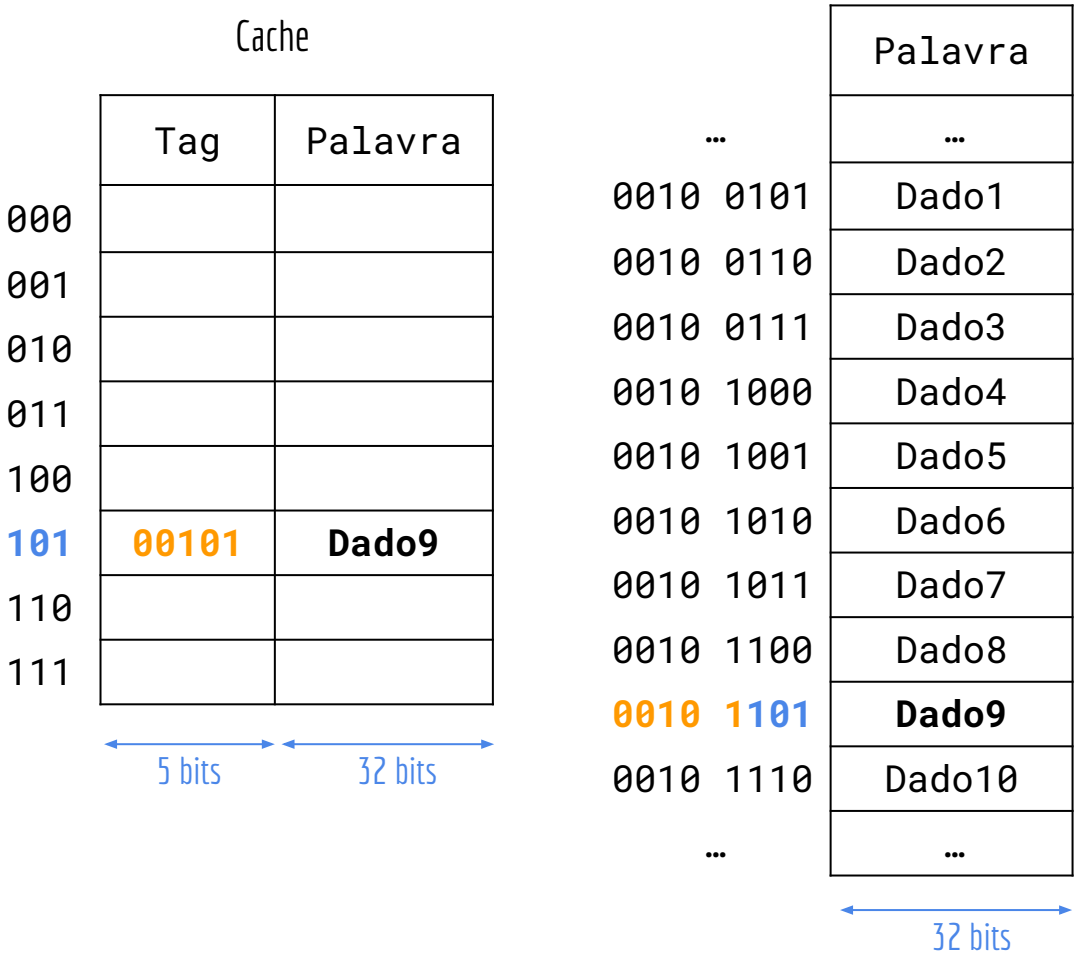
Tag

Além da palavra, cada endereço da cache deve conter o **tag** da palavra.

Bits mais altos da palavra, que não foram utilizados para mapear a cache.

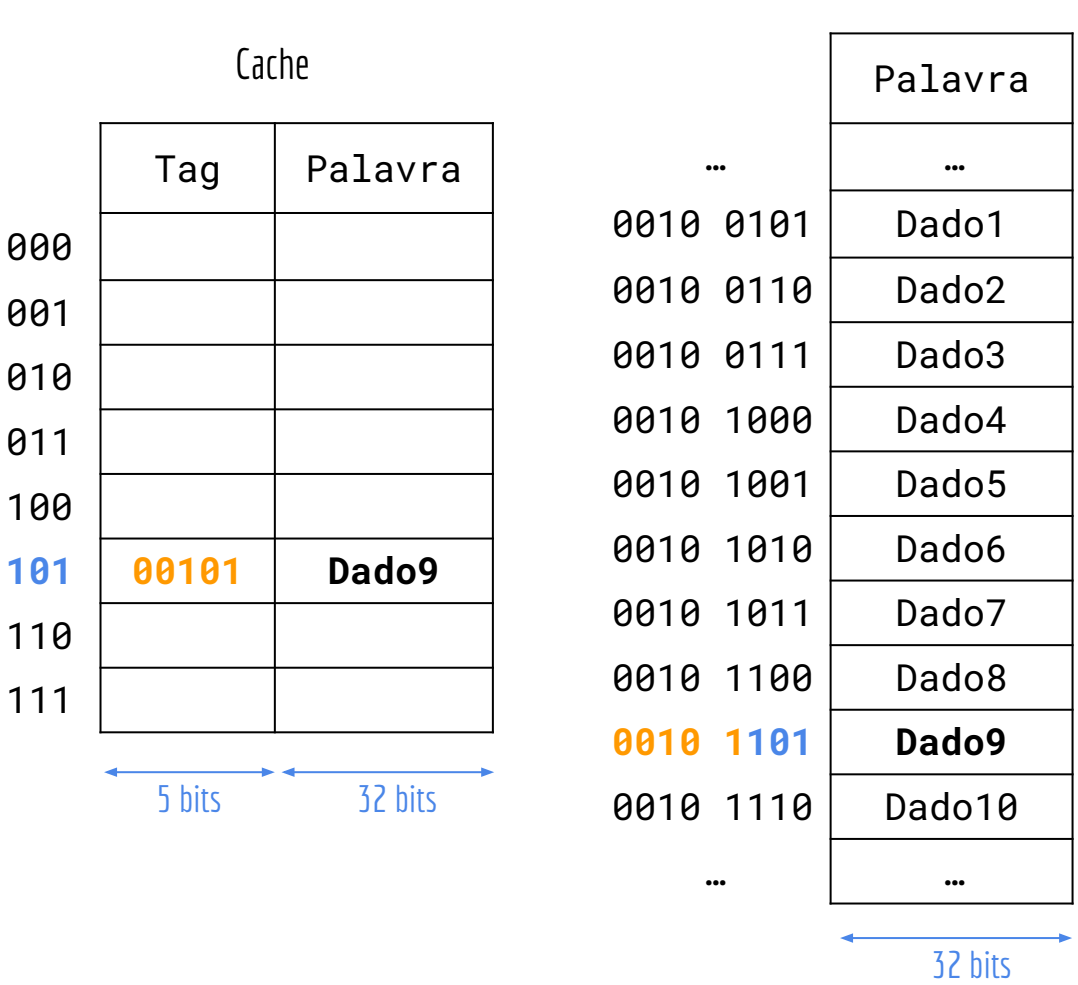
Exemplo

Palavras de 32 bits.
Cache de 8 palavras (3 bits para endereço).
Memória de 256 palavras (8 bits para endereço).
O dado no endereço 0010 1101_2 está na cache.



Exemplo

Agora sabemos que é o dado do endereço $0010\ 1101_2$ que está na cache, e não o do endereço $0010\ 0101_2$, por exemplo.



Validade da cache

Acabamos de ligar a máquina.

- A cache é inicializada com lixo.
- Palavras e tags aleatórias.

Qual o problema? Como resolver?

Bit de validade

Bit de validade para cada entrada da cache.

Se o bit é 0, desconsiderar a cache.

Exemplo de cache de 8 palavras.

	val?	Tag	Palavra
000	1	00101	Dado X
001	0	lixo	lixo
010	0	lixo	lixo
011	1	11111	Dado Y
100	1	00000	Dado Z
101	1	00101	Dado W
110	0	lixo	lixo
111	1	10101	Dado K

Miss ou Hit?

Quando o processador precisa ler uma informação no endereço *X*.

Utiliza os bits mais baixos de *X* para encontrar a entrada na cache.

Os demais bits de *X* são comparados com o tag.

O bit de validade é comparado.

Se tudo isso bater, temos um **hit**.

Caso contrário, temos um **miss**.

	val?	Tag	Palavra
000	1	00101	DadoX
001	0	lixo	lixo
010	0	lixo	lixo
011	1	11111	Dado Y
100	1	00000	Dado Z
101	1	00101	Dado W
110	0	lixo	lixo
111	1	10101	Dado K

Miss

Em caso de miss.

A palavra é solicitada do nível de memória inferior.

Carregada para o endereço de memória correto da cache.

Caso já haja algo nesse endereço da cache, o dado é substituído.

O campo tag é atualizado.

O bit de validade é *setado*.

A palavra é enviada ao processador.

	val?	Tag	Palavra
000	1	00101	DadoX
001	0	lixo	lixo
010	0	lixo	lixo
011	1	11111	Dado Y
100	1	00000	Dado Z
101	1	00101	Dado W
110	0	lixo	lixo
111	1	10101	Dado K

Falácias

“Para programar você não precisa conhecer os detalhes da hierarquia de memória.”

Realmente você sempre pode considerar que o programa está na memória principal, e que ela é um vetor com endereços físicos fixos.

Tudo vai funcionar.

Mas para aplicações que **exigem o mínimo de desempenho**, saber como os diferentes níveis de memória operam é **fundamental**.

Exercícios

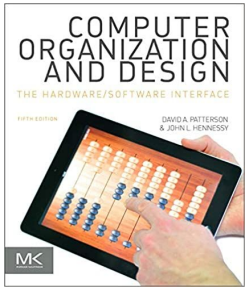
1. Considere uma cache de 8 entradas inicialmente vazia, e que os seguintes endereços são solicitados em ordem: 10110_2, 11010_2, 10110_2, 11010_2, 10000_2, 00011_2, 10000_2, 10010_2, 10000_2.

   Qual o estado final da cache depois de solicitar todos os endereços? Para simplificar, considere que o dado no endereço X é DadoX.

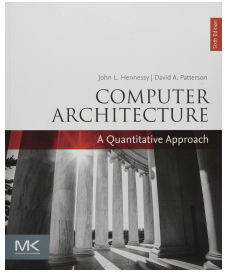
2. Considere uma memória principal com endereços que ocupam 16 bits, e uma memória cache com 256 entradas. Considere que cada endereço de memória principal/cache suporta uma palavra de 32 bits. Qual é o tamanho da memória cache considerando apenas as palavras da memória que ela suporta? Qual é o tamanho total da memória cache? Qual a percentagem de dados na cache que formam os dados de controle (e.g. bit de validade) quando comparados ao percentual de dados de palavras da cache?

Referências

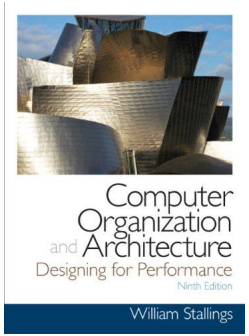
Patterson, Hennessy . Arquitetura e Organização de Computadores: A interface hardware/software. 2014.



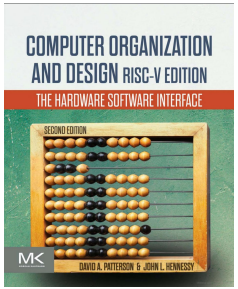
Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 2019.



Stallings, W. Organização de Arquitetura de Computadores. 10a Ed. 2016.



Patterson, Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2020.



Licença