

Hierarquia de Memórias e Cache

Yuri Kaszubowski Lopes

UDESC

Precisamos de Memória

- Precisamos de memória para armazenar as instruções dos nossos programas e os dados
- Todos desejamos uma quantidade infinita de memória
- Por questões físicas e orçamentárias não podemos ter uma memória muito grande

Hierarquia de Memória

- Além de grande, desejamos uma memória rápida
 - ▶ Pelo menos tão rápida quanto a CPU consegue solicitar informações
 - ▶ Manter pipeline cheio

Hierarquia de Memória

- Além de grande, desejamos uma memória rápida
 - ▶ Pelo menos tão rápida quanto a CPU consegue solicitar informações
 - ▶ Manter pipeline cheio
 - ▶ Mas se não conseguimos “pagar” nem por uma memória grande, uma memória grande e rápida se torna um sonho quase fora de alcance

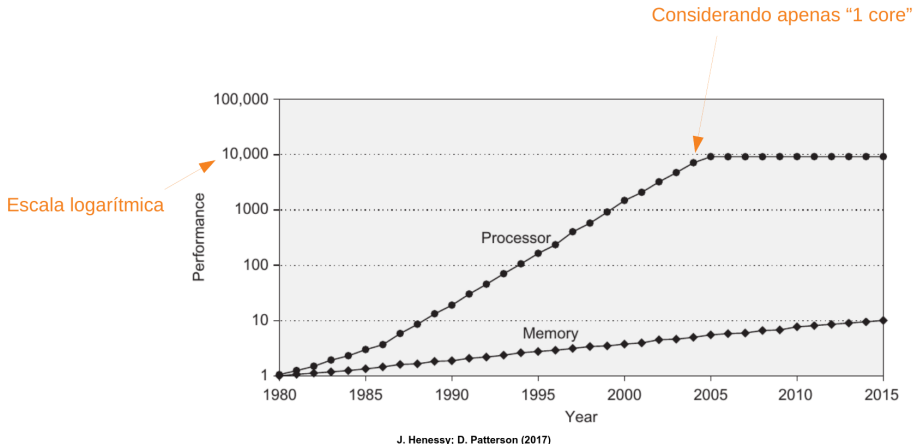
O Quão Rápida a Memória Precisa Ser?

- 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 $12,8 \times 10^9$ 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 $12,8 \times 10^9$ 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 34,1 GiB/s

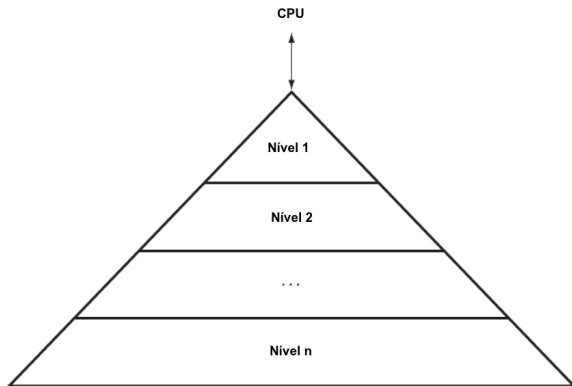
Diferença de desempenho



- Diferença entre a velocidade em que o processador consegue processar instruções versus a velocidade em que a memória consegue suprir essas instruções.

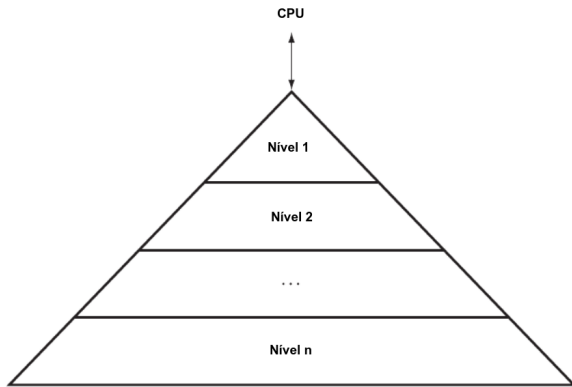
Hierarquia de Memórias

- Para amenizar esses problemas, criamos uma hierarquia de memórias, com n níveis
- A quantidade de níveis vai depender da arquitetura da máquina



Hierarquia de Memórias

- O que aumenta nos níveis mais próximos da CPU?
- O que aumenta nos níveis mais distantes da CPU?



Hierarquia de Memórias

- Tomando como base o seu computador x86-64, você consegue especificar quais memórias estão em quais níveis?
 - ▶ Caches
 - ▶ Memória Principal (RAM)
 - ▶ Memória Secundária (persistente): HD ou SSD (ou ambos)

Uma questão de custos

Memory technology	Typical access time	\$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	\$500–\$1000
DRAM semiconductor memory	50–70 ns	\$10–\$20
Flash semiconductor memory	5,000–50,000 ns	\$0.75–\$1.00
Magnetic disk	5,000,000–20,000,000 ns	\$0.05–\$0.10

Patterson, Hennessy 5^a ed (2014 – Edição Americana)

Compare

Memory technology	Typical access time	\$ per GiB in 2020
SRAM semiconductor memory	0.5–2.5 ns	\$500–\$1000
DRAM semiconductor memory	50–70 ns	\$3–\$6
Flash semiconductor memory	5,000–50,000 ns	\$0.06–\$0.12
Magnetic disk	5,000,000–20,000,000 ns	\$0.01–\$0.02

Patterson, Hennessy 6^a ed (2021)

Memory technology	Typical access time	\$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	\$500–\$1000
DRAM semiconductor memory	50–70 ns	\$10–\$20
Flash semiconductor memory	5,000–50,000 ns	\$0.75–\$1.00
Magnetic disk	5,000,000–20,000,000 ns	\$0.05–\$0.10

Patterson, Hennessy 5^a ed (2014 – Edição Americana)

Memory technology	Typical access time	\$ per GB in 2008
SRAM	0.5–2.5 ns	\$2000–\$5000
DRAM	50–70 ns	\$20–\$75
Magnetic disk	5,000,000–20,000,000 ns	\$0.20–\$2

Patterson, Hennessy 4^a ed (2011 – Edição Americana) – Traduzido em 2014

Tecnologias envolvidas

- Considerando a hierarquia comumente encontrada nos x86-64:
 - ▶ Caches: SRAM (static random access memory)
 - ★ Construídas utilizando flip-flops
 - ▶ Memória Principal: DRAM (dynamic random access memory)
 - ★ Memórias capacitivas
 - ★ Capacitores carregados/descarregados indicam o estado do bit
 - ▶ Memória Secundária (armazenamento permanente): SSD ou HD
 - ★ Memórias não voláteis 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

A Memória Principal

- A memória principal fica entre os níveis da cache e armazenamento permanente
 - ▶ Cache: desejamos mais velocidade
 - ▶ Armazenamento permanente: desejamos mais espaço
 - ★ Utilizar o armazenamento permanente como se fosse a memória principal implica na utilização do conceito de paginação atualmente
 - ★ Tema para a disciplina de Sistemas Operacionais
 - ★ Envolve muito do hardware
 - ★ Uso de interrupções e de unidades de tradução de endereços (MMU)

Memória Cache

- Vamos começar a pensar em uma memória cache básica
- Enquanto nossos computadores hoje têm comumente 16GB de memória principal (DRAM), eles possuem cerca de 12MB de cache (SRAM)
- É comum encontrarmos processadores para servidores com 64MB de cache, mas dificilmente passamos disso
 - ▶ A diferença entre a capacidade ainda é gigantesca

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)
 - ▶ Para dados o programador deve transferir da memória secundária para a principal
 - ★ O S.O. ajuda muito (é simples para o programador)
 - ★ Entre Memória Principal e cache é problema do Hardware e S.O.

Memória Cache

- Como podemos mapear a memória principal para dentro de nossa cache de maneira automática?
- Que regras seguir? Qual dado é importante?

Memória Cache

- Como podemos mapear a memória principal para dentro de nossa cache de maneira automática?
- Que regras seguir? Qual dado é importante?
- Se analisarmos nossos programas, vamos ver que temos dois tipos de **localidade**
 - ▶ **Localidade temporal**
 - ▶ **Localidade espacial**

Localidade temporal e espacial

- **Localidade temporal:**

- ▶ Se um item da memória é acessado, é muito provável que ele seja acessado novamente num futuro próximo
- ▶ E.g., 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. E.g.:

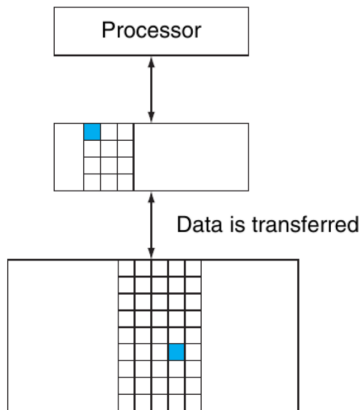
- ▶ Nossos programas são (quase) sequencias, então quando a instrução i é carregada, é provável que $i+1$, $i+2$, ... também sejam úteis
- ▶ Quando operamos no elemento k de um vetor, o elementos $k+1$, $k+2$, ... provavelmente também serão úteis

Memória Cache

- Nossa memória cache é muito menor que a memória principal
- Vamos tirar vantagem das localidades espacial e temporal
- Manter na cache os dados utilizados mais recentemente, e seus vizinhos
 - ▶ Objetivo: diminuir acesso à memória principal (mais lenta que a cache)

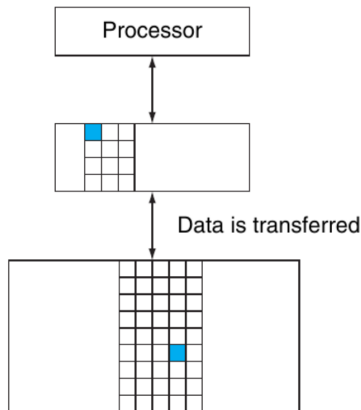
Conceitos de Mapeamento

- Vamos dividir nossas memórias (principal e cache) em blocos
 - ▶ Os blocos possuem o mesmo tamanho
 - ▶ Obviamente, nossa memória cache suporta menos blocos que a principal



Requisitando dados

- Quando o processador requisita um dado:
 - ▶ 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)
 - ★ Vamos gastar tempo para carregar o bloco



Medidas de Performance

- **hit rate**

- ▶ Também chamado de hit ratio
- ▶ Fração dos acessos a memória nos quais tivemos um hit
- ▶ $\text{hitRate} = \text{hits} / \text{totalAcessos}$

- **miss rate**

- ▶ Também chamado de miss ratio
- ▶ Fração dos acessos a memória nos quais tivemos um miss
- ▶ $\text{MissRate} = 1 - \text{hitRate}$

- Ration: Comparação entre dois valores (normalmente ambos são de **mesma** unidade)

- Rate: Ration entre valores de **diferente** unidades

Medidas de Performance

- **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

- Nesse contexto, consideraremos cache a memória que está no nível mais alto
 - ▶ Em nosso processador MIPS, poderíamos simplesmente substituir nossas 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
 - ▶ E.g., muitos HDs possuem uma pequena cache interna, onde os dados mais requisitados são salvos

Cache diretamente mapeada

- Vamos considerar que:
 - ▶ O processador sempre requisita uma palavra
 - ★ 32 bits no MIPS de nossas aulas
 - ▶ Cada bloco da cache também armazena uma palavra
- Quanto o processador requisita uma palavra X_n que não está na cache:
 - ▶ Temos um miss
 - ▶ A palavra então é carregada para a cache

Cache diretamente mapeada

Cache

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

Antes do miss

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_n
X_3

Depois do miss

Cache diretamente mapeada

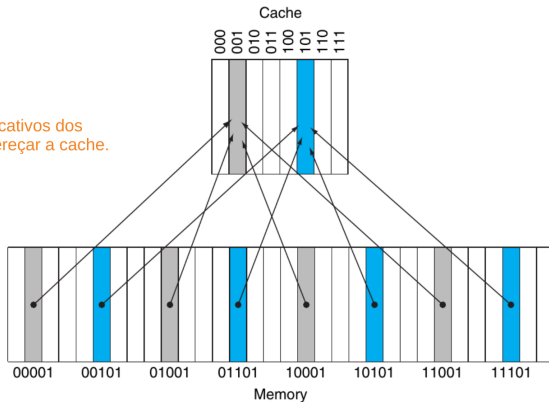
- Como encontrar um item nessa cache?
- Como saber se um item está na cache ou não?

Cache diretamente mapeada

- Em uma cache diretamente mapeada:
 - ▶ Cada endereço de memória é mapeado para exatamente uma posição na cache
 - ★ Mas cada posição da cache pode ser responsável por um subconjunto de vários endereços da memória principal
 - ▶ Construimos 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

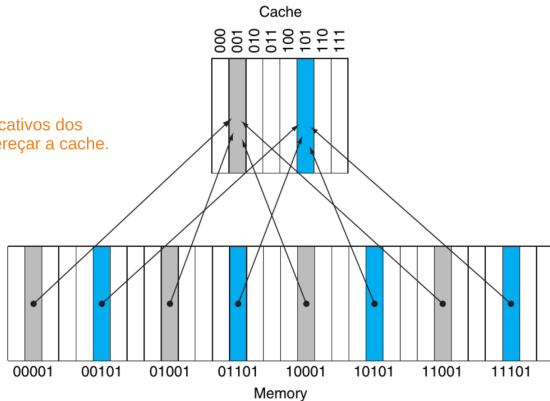
Cache de $2^3=8$ entradas.
Utilizamos os 3 bits menos significativos dos
endereços de memória para endereçar a cache.



Exemplo

Cache de $2^3=8$ entradas.

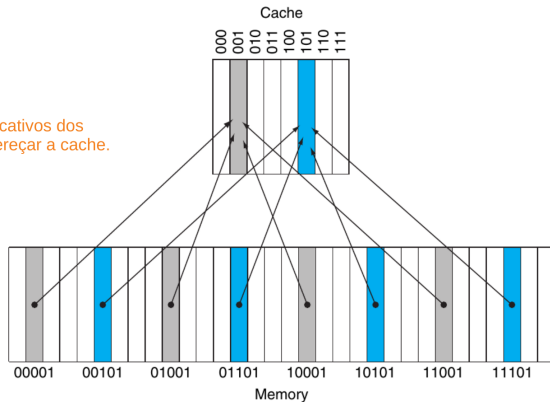
Utilizamos os 3 bits menos significativos dos endereços de memória para endereçar a cache.



- 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 nesse endereço

Dados da cache

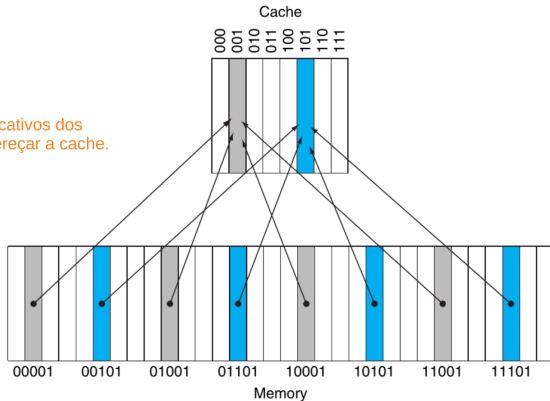
Cache de $2^3=8$ entradas.
Utilizamos os 3 bits menos significativos dos
endereços de memória para endereçar a cache.



- Cada endereço da cache contém uma palavra
 - ▶ A palavra será utilizada pelo processador
 - ▶ O endereço da cache, e a palavra contida nela são suficientes para sabermos se determinada palavra da memória está na cache?

Dados da cache

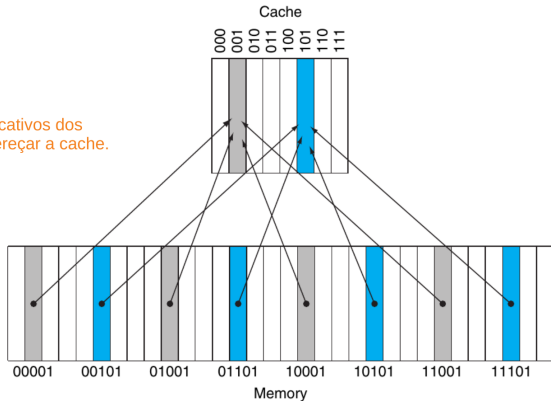
Cache de $2^3=8$ entradas.
Utilizamos os 3 bits menos significativos dos
endereços de memória para endereçar a cache.



- Considere mais uma vez que o endereço 00101_2 é solicitado
 - ▶ O processador busca o dado no endereço 101_2 da cache
 - ▶ Qual palavra da memória está nesse endereço?

Dados da cache

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



- Considere mais uma vez que o endereço 00101_2 é solicitado
 - ▶ O processador busca o dado no endereço 101_2 da cache
 - ▶ Qual palavra da memória está nesse endereço?
 - ★ $00101_2, 01101_2, 10101_2, 11101_2$

Tag

- Além da palavra, cada endereço da cache deve conter o tag da palavra
 - ▶ O tag são os bits mais altos da palavra, que não foram utilizados para mapear a cache

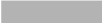

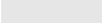

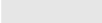
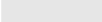
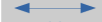
Exemplo

Palavras de 32 bits

Cache de 8 palavras (3 bits para endereço)

Memória de 256 palavras (8 bits para endereço)

O dado no endereço 00101101_2 está na cache

Cache	
	Tag Palavra
000	 Dado x
001	
010	
011	
100	
101	00101 Dado9
110	
111	

5 bits 32 bits

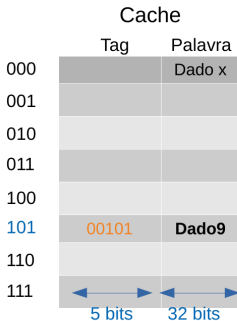
Memória Principal

	Palavra
...	...
00100101	Dado1
00100110	Dado2
00100111	Dado3
00101000	Dado4
00101001	Dado5
00101010	Dado6
00101011	Dado7
00101100	Dado8
00101101	Dado9
00101110	Dado10
...	...

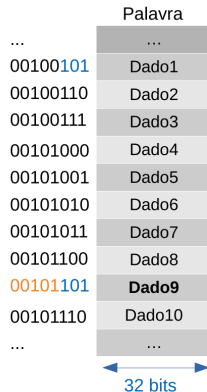
32 bits

Exemplo

Agora sabemos que é o dado do endereço 00101101 que está na cache, e não o do endereço 00100101, por exemplo.



Memória Principal



Validade da cache

- Imagine que acabamos de ligar o computador
- A cache é inicializada com lixo
 - ▶ Palavras e tags aleatórias
- Qual o problema? Como resolver?

Validade da cache

- Imagine que acabamos de ligar o computador
- A cache é inicializada com lixo
 - ▶ Palavras e tags aleatórias
- Qual o problema? Como resolver?
 - ▶ Solução: Bit de validade

Bit de validade

- Adicionamos ainda um **bit de validade** para cada entrada da cache
 - ▶ Se o bit é 0, devemos desconsiderar a cache
- Exemplo de cache de 8 palavras:

	valido	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 alguma 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**

	valido	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

- Caso ocorra um miss:
 - ▶ A palavra é solicitada do nível de memória inferior, e é carregada para o endereço de memória correto
 - ★ Caso já haja algo nesse endereço da cache, o dado é substituído
 - ★ O campo tag é atualizado
 - ★ O bit de validade é setado
 - ▶ Finalmente a palavra é enviada ao processador

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.

Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores: Interface Hardware/Software**. 5a Edição. Elsevier Brasil, 2017.
- J. Henessy; D. Patterson. **Arquitetura de computadores: Uma abordagem quantitativa**. 6a Edição. 2017.
- STALLINGS, William. **Arquitetura e organização de computadores**. 8. ed. São Paulo: Pearson Education do Brasil, 2010.

Hierarquia de Memórias e Cache

Yuri Kaszubowski Lopes

UDESC