

# Infraestrutura de Hardware

## Explorando a Hierarquia de Memória



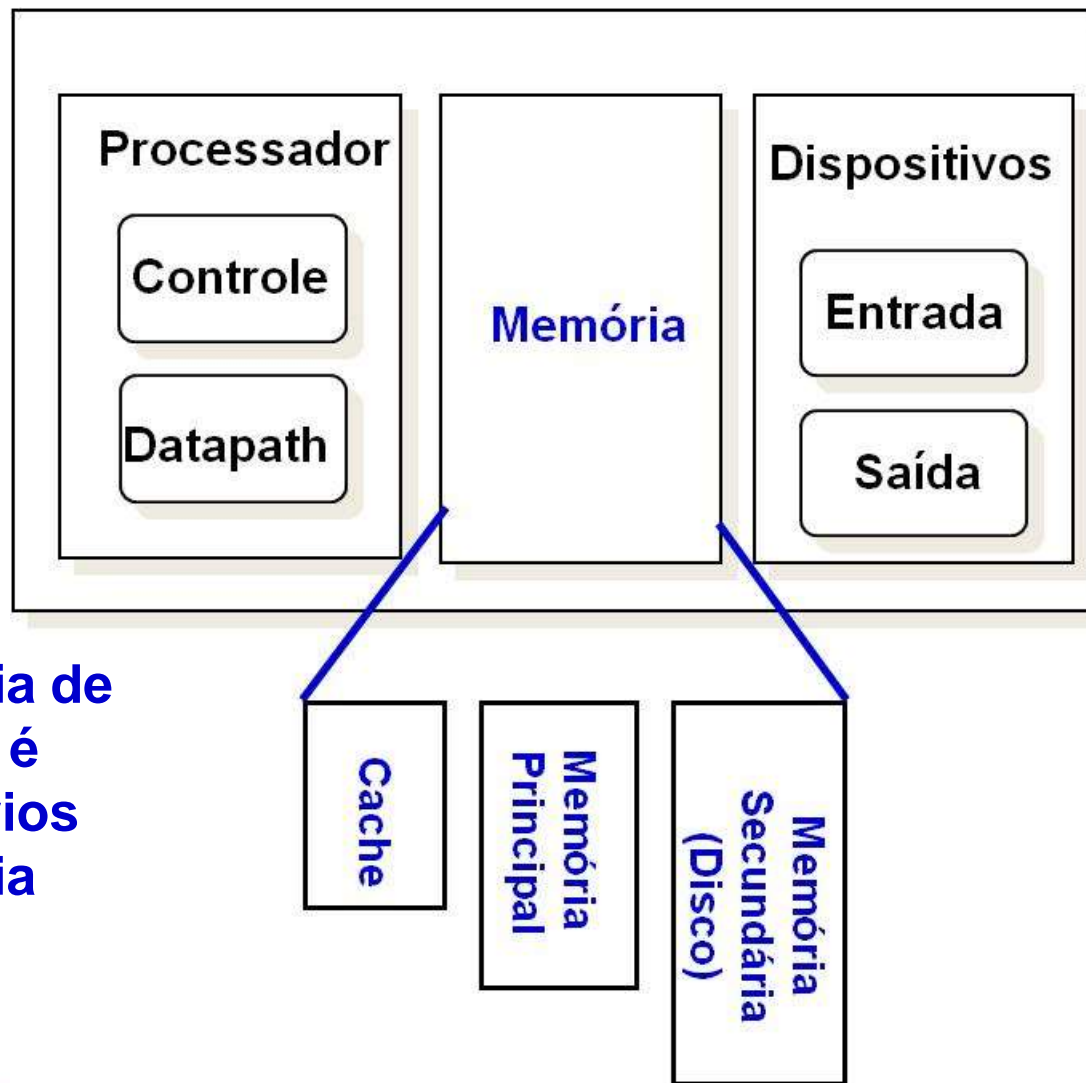
UNIVERSIDADE  
FEDERAL  
DE PERNAMBUCO

[CIn.ufpe.br](http://CIn.ufpe.br)

# Perguntas que Devem ser Respondidas ao Final do Curso

- Como um programa escrito em uma linguagem de alto nível é entendido e executado pelo HW?
- Qual é a interface entre SW e HW e como o SW instrui o HW a executar o que foi planejado?
- O que determina o desempenho de um programa e como ele pode ser melhorado?
- **Que técnicas um projetista de HW pode utilizar para melhorar o desempenho?**

# Componentes de um Computador



**Sistema de memória de um computador é composto por vários tipos de memória**

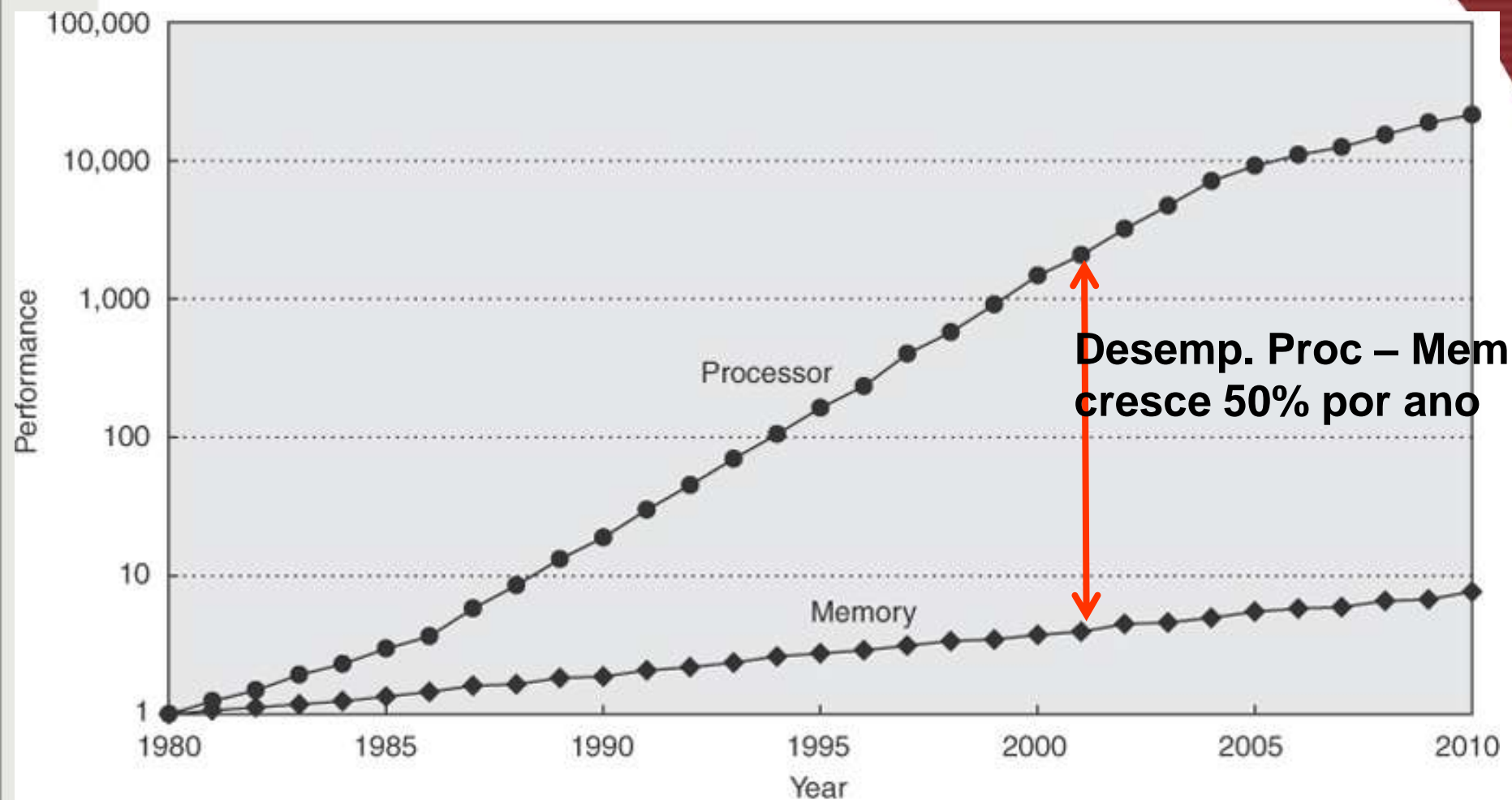
# Importância da Memória no Desempenho

- Uma forma de aumentar o desempenho de um computador é melhorar o desempenho da CPU

Algumas técnicas vistas:

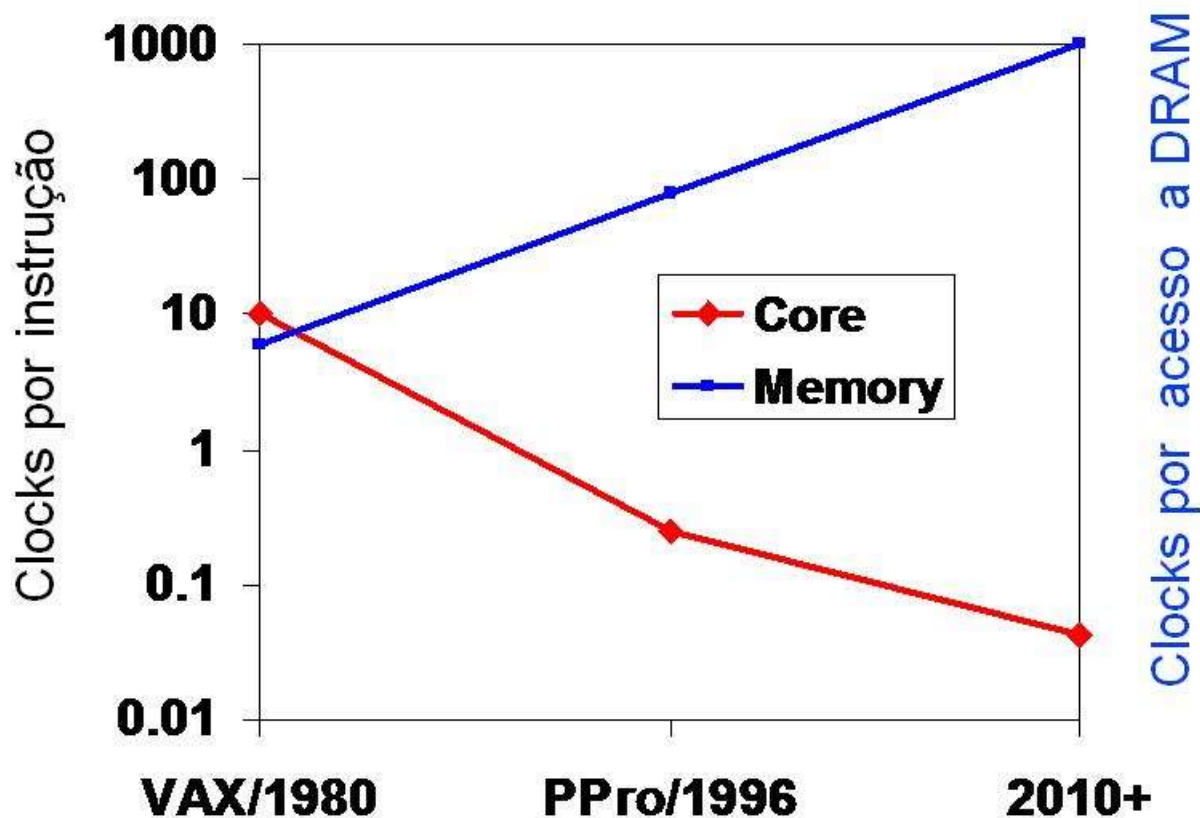
- Pipeline, Superescalar, Multicore
- Comumente, programas contêm muitas instruções que acessam a memória
- Portanto, tempo de acesso a memória deve também ser otimizado
- Tempo de acesso a memória representa, na maioria das vezes, o gargalo da execução de um programa

# Processador x Memória: Desempenho



# “Memory Wall”

- Desempenho do sistema limitado pela memória
- Bom projeto de sistema de memória é fundamental para melhorar desempenho



# Requisitos Mínimos do Windows 7



Search this website

bin

[Home](#) [Discover Windows](#) [Products](#) [Shop](#) [Downloads](#) [Help & How-to](#)

[Windows 7 home](#) [What is Windows 7](#) [Compare](#) [Features](#) [Videos](#)

## Windows 7 system requirements

**If you want to run Windows 7 on your PC, here's what it takes:**

- 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64) processor
- 1 gigabyte (GB) RAM (32-bit) or 2 GB RAM (64-bit)
- 16 GB available hard disk space (32-bit) or 20 GB (64-bit)
- DirectX 9 graphics device with WDDM 1.0 or higher driver

**Requisitos do  
sistema de memória**

# Tecnologias de Memória

Tecnologia	Ano	Tempo de acesso	U\$/Gbyte
Static RAM (SRAM)	2012	0,5 – 2,5ns	500 a 1000
Dynamic RAM (DRAM)	2012	50 - 70ns	10 - 20
Disco	2012	5-20 ms ou 5.000.000 a 20.000.000 ns	0,05 a 0.1

- Memória ideal: velocidade de SRAM , tamanho e custo de disco!



# RAM Dinâmica vs. Estática

## ■ DRAM (Dynamic Random Access Memory)

Maior tempo de acesso (50 – 70 ns)

Perda de informação após algum tempo

- Necessidade de refreshing (~8ms)

Grande capacidade de integração (baixo custo por bit)

## ■ SRAM (Static Random Access Memory)

Pequeno tempo de acesso (0,5 – 2,5 ns)

Não existe necessidade de refreshing

Alto custo por bit (baixa integração)

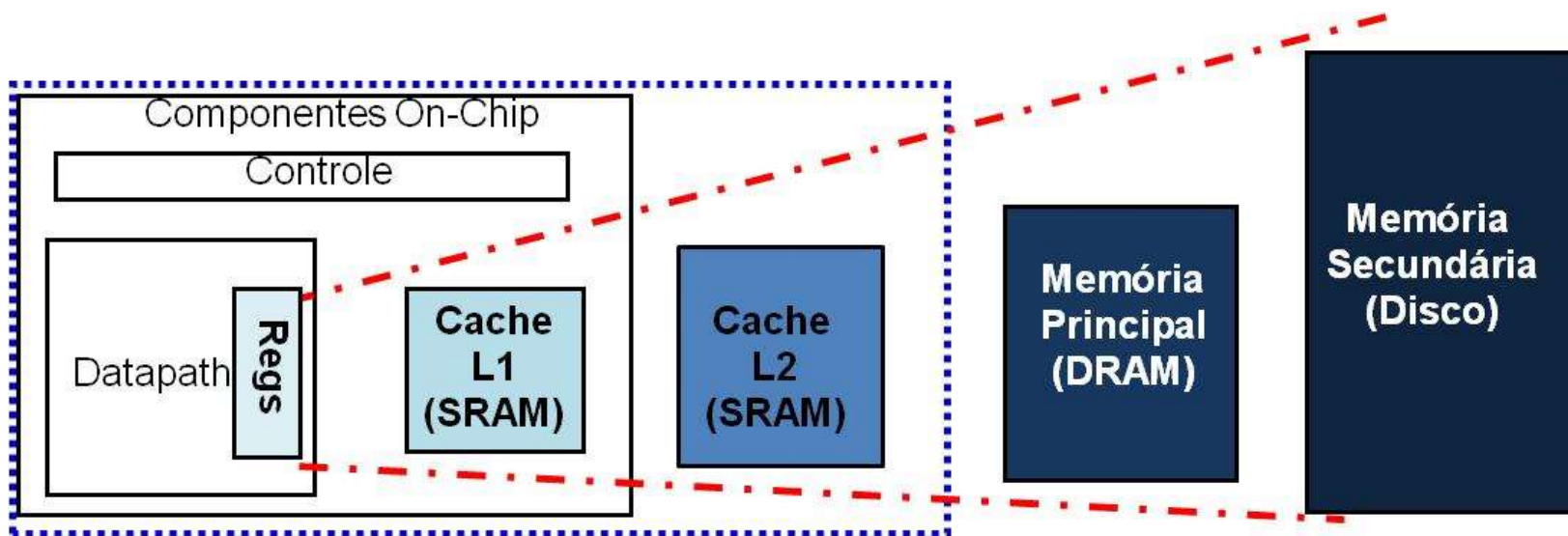
# Como Minimizar Tempo de Acesso a Memória?

- Programador deseja ter memória maior e mais veloz
- Existem vários tipos de memória construídos com tecnologia diferente
  - SRAM, DRAM, Magnético, etc
- Diferença de velocidade e de custo
  - Fato: Mais velozes → mais caras
    - Maiores são mais lentas, menores são mais rápidas
- Deve-se criar ilusão de oferecer memória maior, mais barata e mais rápida (na maioria das vezes)

**Hierarquia de Memória!**

# Hierarquia de Memória

- Aproveitar **princípio da localidade** para oferecer o máximo de memória de baixo custo com uma velocidade de acesso oferecida pelo tipo de memória mais rápida



<b>Tempo (%ciclos):</b>	½'s	1's	10's	100's	10000's
<b>Tamanho(bytes):</b>	100's	10K's	M's	G's	T's
<b>Custo:</b>	mais caro			mais barato	

# O Que é Princípio da Localidade?

## ■ **Localidade Temporal**

Programa tende a referenciar as instruções e dados referenciados recentemente

- **Mantenha itens mais recentemente referenciados junto ao processador**
- Ex: instruções de um loop

## ■ **Localidade Espacial**

Programa tende a referenciar as instruções e dados que tenham endereços próximos das últimas referências

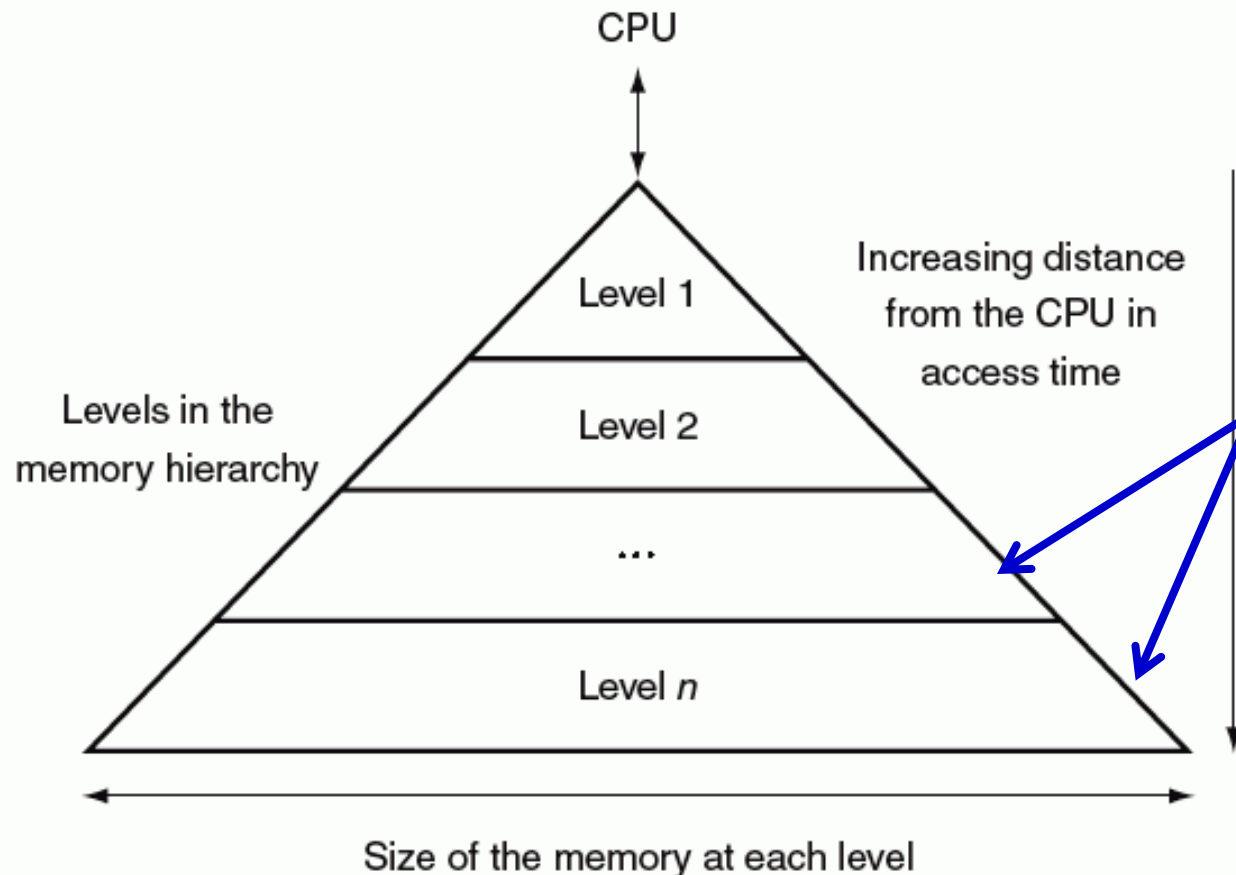
- **Mova blocos de dados de palavras contíguas para junto do processador**
- Ex: dados de um array

# Aproveitando o Princípio da Localidade

- Armazene tudo em disco  
Ilusão de que a memória é muito grande
- Copie itens acessados **recentemente** e itens em endereços **próximos** do disco para uma memória DRAM que é menor  
Memória principal
- Copie itens acessados **mais recentemente** e itens em endereços **mais próximos** da DRAM para uma SRAM menor  
Memória cache on-chip  
Ilusão de que a memória é muito rápida

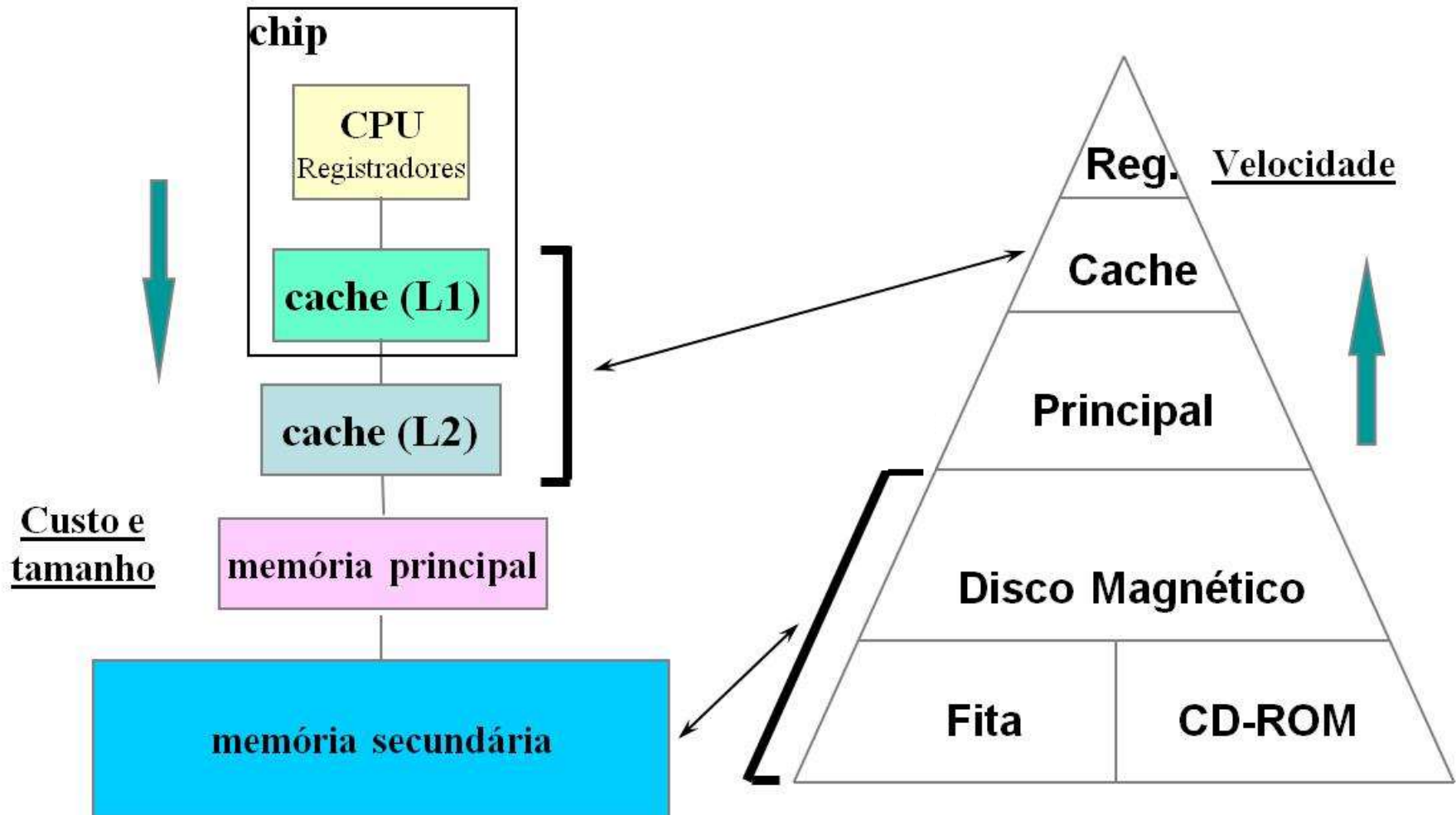
# Idéia Geral de Hierarquia de Memória

- Níveis mais altos de memória são mais rápidas
- Níveis mais baixos são maiores



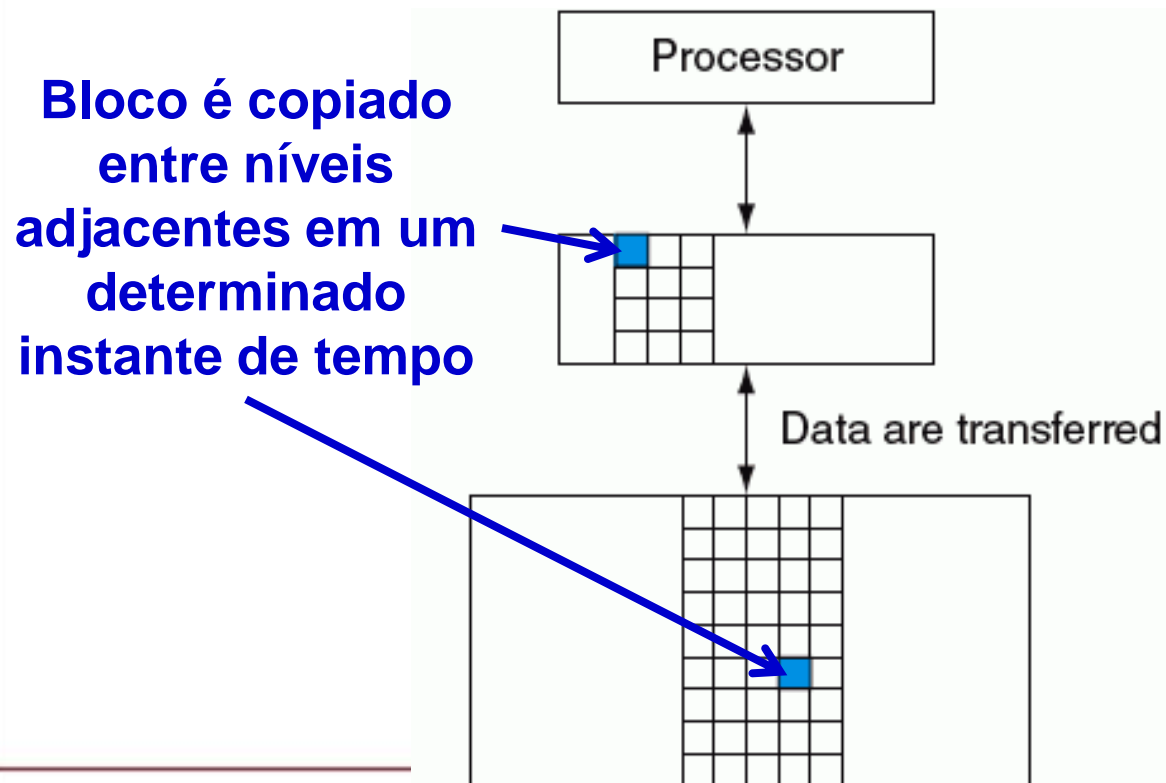
**Cada nível mais alto possui um subconjunto de dados/instruções do nível mais abaixo**

# Sistema Hierárquico de Memória



# Hierarquia de Memória: Terminologias

- **Bloco** (linha) é a unidade mínima de informação que pode estar presente ou não em uma cache  
Geralmente é a unidade que é copiada entre um nível e outro





# Hierarquia de Memória: Terminologias

- **Hit** (acerto) ocorre quando os dados requisitados pela CPU estão em algum bloco do nível de memória desejado
  - Hit time** – Tempo para acessar dado no nível desejado
  - Hit rate** – Hits/acessos
- **Miss** (falta) ocorre quando os dados requisitado pela CPU não estão em algum bloco do nível de memória desejado
  - Miss rate** – Misses/acessos ou  $(1 - \text{hit rate})$
- **Miss penalty** é o tempo que leva para buscar o bloco de um nível mais abaixo para o nível mais acima e enviá-lo para a CPU
  - Tempo de acesso do bloco no nível mais abaixo + tempo de transmissão do bloco para nível mais acima + tempo de escrita do bloco no nível mais acima + tempo de transmissão para a CPU

## Exercício

- Quais afirmações são geralmente verdadeiras?

Caches se aproveitam da localidade temporal.

**Verdadeiro**

Em uma instrução de leitura, o valor retornado depende dos blocos que estão na cache.

**Falso**

Maior parte do custo de uma hierarquia de memória é da memória de nível mais alto.

**Falso**

Maior parte da capacidade de armazenamento em uma hierarquia de memória vem do nível mais baixo.

**Verdadeiro**

# Cache

- Memória SRAM menor que memória principal (DRAM) localizada perto da CPU

Acesso rápido

- Tecnologia mais rápida
  - Nível de memória mais perto da CPU, não utiliza barramento comum aos outros dispositivos
- Utiliza-se do princípio da localidade para armazenar dados acessados mais recentemente e de endereços mais próximos

# Exemplo de Cache Simples

- Suponha cache onde bloco é igual a uma palavra
- Dados os acessos as palavras  $X_1, \dots, X_{n-1}$ , o acesso a  $X_n$  geraria uma falta (miss), depois o elemento seria buscado e colocado na cache

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

Cache antes do acesso a  $X_n$

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

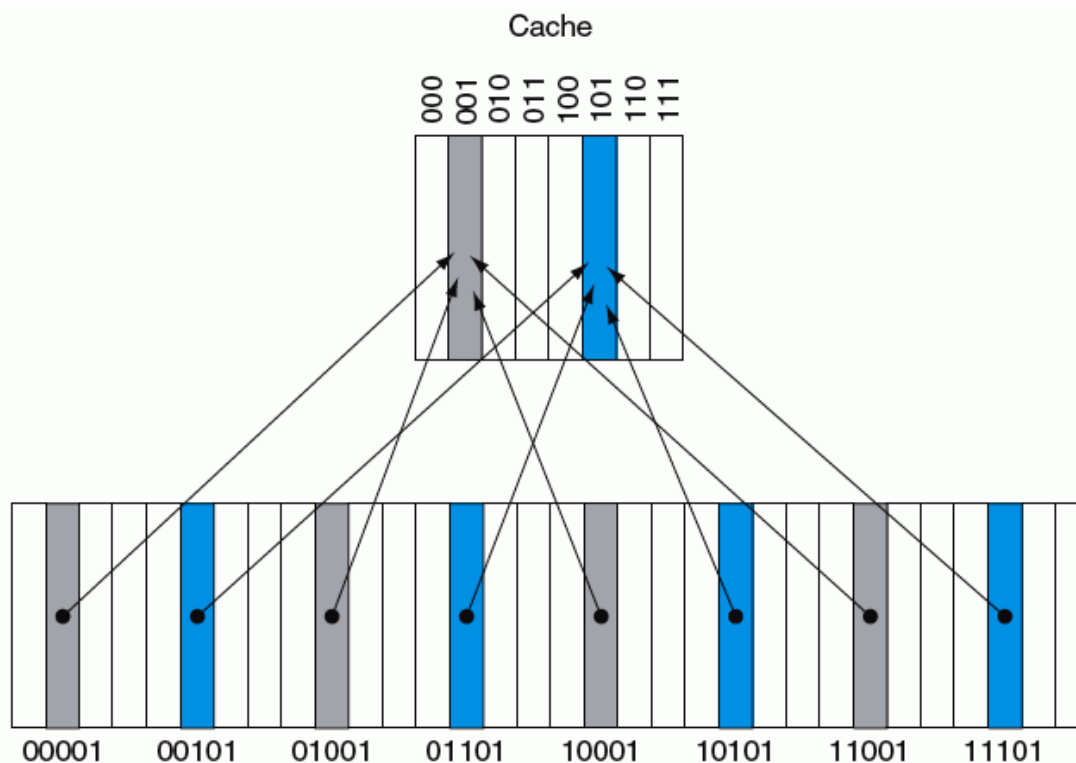
Cache depois do acesso a  $X_n$

# Acessando Dados da Cache

- Pergunta 1: Como saber se um dado está na cache?
- Pergunta 2: Se dado estiver, como localizamos ele na cache?

# Mapeamento Direto

- Forma mais simples de determinar a localização de um bloco na cache é pelo endereço do bloco na memória
- Cache com mapeamento direto faz com que cada bloco da memória seja sempre mapeado para o mesmo bloco da cache



# Localizando Blocos com Mapeamento Direto

- Localização do bloco de memória na cache (resposta da pergunta 2):

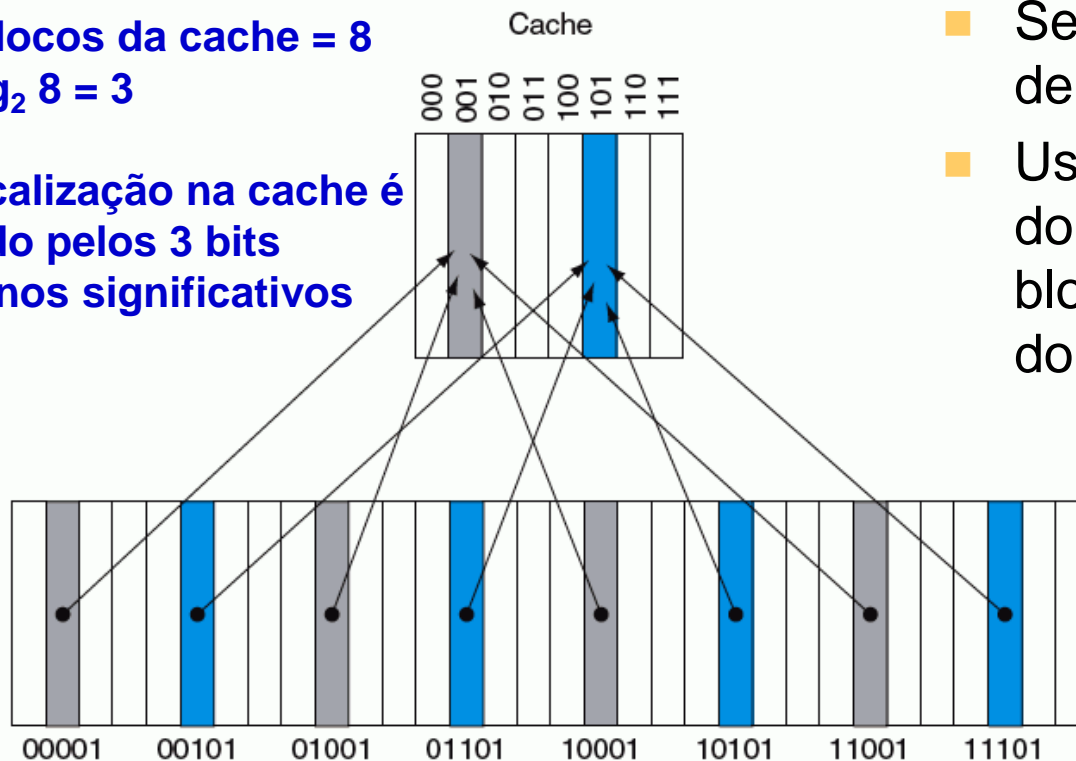
**(endereço do bloco) modulo (#blocos na cache)**

- Muitos blocos de memória compartilham o mesmo bloco da cache em instantes diferentes

# blocos da cache = 8

$\log_2 8 = 3$

Localização na cache é  
dado pelos 3 bits  
menos significativos



- Se # blocos da cache, potência de 2

- Usa-se bits menos significativos do endereço da memória do bloco para determinar o número do bloco da cache

$\log_2 \# \text{blocos}$  bits menos significativos

# Verificando se Bloco Está na Cache

- Cache armazena não só os dados de um bloco de memória, mas também parte do endereço deste bloco

## Tag

- Uma **tag** é formada pelos bits mais significativos do endereço do bloco

Todos os bits do endereço do bloco, menos os usados para identificar um bloco da cache

- Uso de tags permite saber se bloco de memória está ou não na cache

Resposta para a pergunta 1

- Cache ainda armazena um bit de validade (valid bit) para saber se o conteúdo do bloco ainda é válido

Para os casos onde computador é iniciado e ou quando a cache ainda possuir algum lixo por causa do fim de execução de programa



# Endereçando a Cache

- Composição de um endereço de memória

Tag

Índice do bloco

Posição de byte dentro do bloco

} **Endereço do bloco**

- Exemplo:

Memória: endereço de 32 bits, cada bloco é uma palavra, acesso por palavra(32 bits), endereçamento por byte

Cache: capacidade para armazenar 64 palavras

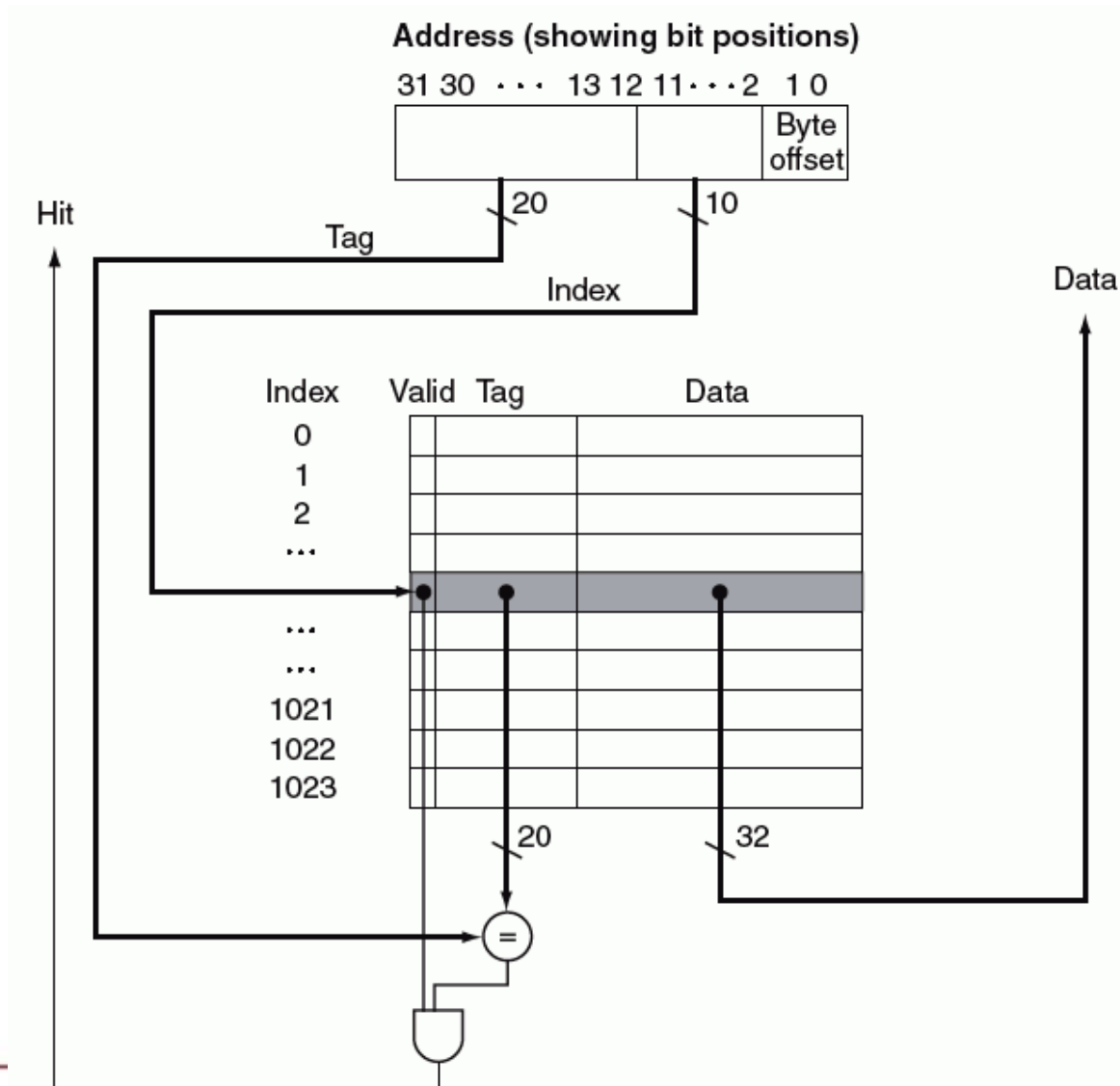


Tag (24)

Índice(6)

Posição do  
byte (2)

# Mapeando Endereços de Memória na Cache



# Tamanho do Bloco da Cache

- Cada bloco de uma cache pode conter mais de uma palavra

Aproveita a localidade espacial

- **Exemplo:**

Memória: endereço de 32 bits, acesso por palavra(32 bits), endereçamento por byte

Cache: capacidade para armazenar 64 blocos de **4 palavras cada** (tamanho bloco:16 Bytes)



## Exemplo: Mapeando Endereço em Bloco da Cache

- 64 blocos, 16 bytes/bloco

Endereço do byte 1200 é mapeado para que bloco da cache?

- Endereço do bloco na memória =  $\lfloor 1200/16 \rfloor = 75$
- Número do bloco da cache =  $75 \text{ modulo } 64 = 11$

# Exemplo de Utilização de Cache com Mapeamento Direto

- 8 blocos, 1 palavra/bloco, mapeamento direto

## Estado Inicial

<u>Índice</u>	<u>V</u>	<u>Tag</u>	<u>Dado</u>
<u>000</u>	<u>N</u>		
<u>001</u>	<u>N</u>		
<u>010</u>	<u>N</u>		
<u>011</u>	<u>N</u>		
<u>100</u>	<u>N</u>		
<u>101</u>	<u>N</u>		
<u>110</u>	<u>N</u>		
<u>111</u>	<u>N</u>		

# Exemplo de Utilização de Cache com Mapeamento Direto

End. decimal	End. Binário	Hit/miss	Bloco da cache
22	10 110	Miss	110

Apenas essa parte é armazenada na cache

Índice	V	Tag	Dado
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
<b>110</b>	<b>Y</b>	<b>10</b>	<b>Mem[10110]</b>
111	N		

# Exemplo de Utilização de Cache com Mapeamento Direto

End. decimal	End. Binário	Hit/miss	Bloco da cache
26	11 010	Miss	010

Índice	V	Tag	Dado
000	N		
001	N		
<b>010</b>	<b>Y</b>	<b>11</b>	<b>Mem[11010]</b>
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

# Exemplo de Utilização de Cache com Mapeamento Direto

End. decimal	End. binário	Hit/miss	Bloco da cache
22	10 110	Hit	110
26	11 010	Hit	010

Índice	V	Tag	Dado
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



# Exemplo de Utilização de Cache com Mapeamento Direto

End. decimal	End. binário	Hit/miss	Bloco da cache
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Índice	V	Tag	Dado
<b>000</b>	<b>Y</b>	<b>10</b>	<b>Mem[10000]</b>
001	N		
010	Y	11	Mem[11010]
<b>011</b>	<b>Y</b>	<b>00</b>	<b>Mem[00011]</b>
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

# Exemplo de Utilização de Cache com Mapeamento Direto

End. decimal	End. binário	Hit/miss	Bloco da Cache
18	10 010	Miss	010

Índice	V	Tag	Dado
000	Y	10	Mem[10000]
001	N		
<b>010</b>	<b>Y</b>	<b>10</b>	<b>Mem[10010]</b>
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

**Bloco de memória  
sobrescreve o antigo**



# Faltas (Misses) na Cache

- Ocorrendo uma falta na cache

Congela o pipeline

Busca bloco na memória

- Com ajuda de controlador de HW que acessa memória e preenche cache
- Se o bloco da cache estiver preenchido com outro bloco, a informação é sobre-escrita

Caso a falta seja referente a uma instrução

- Reinicia a busca da instrução

Caso a falta seja referente a um dado

- Completa o acesso ao dado

- Desempenho é penalizado

# Tipos de Acesso à Cache

- Leitura

Simples de implementar

- Escrita

Lento e complicado

Dado e tag são atualizados na cache

Inconsistencia entre memória principal e cache!!

- se um bloco da cache foi alterado pela CPU, não pode ser descartado da cache sem garantir que foi copiado para a memória principal

**Como resolver?**

# Políticas de Escrita e Consistência

## ■ Caches do tipo Write through

Cache e memória são atualizadas simultaneamente

Cache e memória sempre consistentes

Penaliza desempenho, CPU deve esperar acesso a memória

## ■ Caches do tipo Write back

Memória principal somente é atualizada quando bloco é substituído da cache

Cache e memória momentaneamente inconsistentes

Usa dirty bit para marcar linhas alteradas na cache

Reduz quantidade de acessos a memória

# Analizando as Diferentes Políticas de Escrita

<i><b>Write through</b></i>	<i><b>Write back</b></i>
<i><b>facilidade de implementação</b></i>	<i><b>redução de acessos à memória</b></i>
<i><b>consistência da memória principal</b></i>	

- Dois métodos requerem que a CPU espere a escrita
- **Solução: Write buffers**
  - Bloco é escrito no buffer e CPU pode continuar execução