

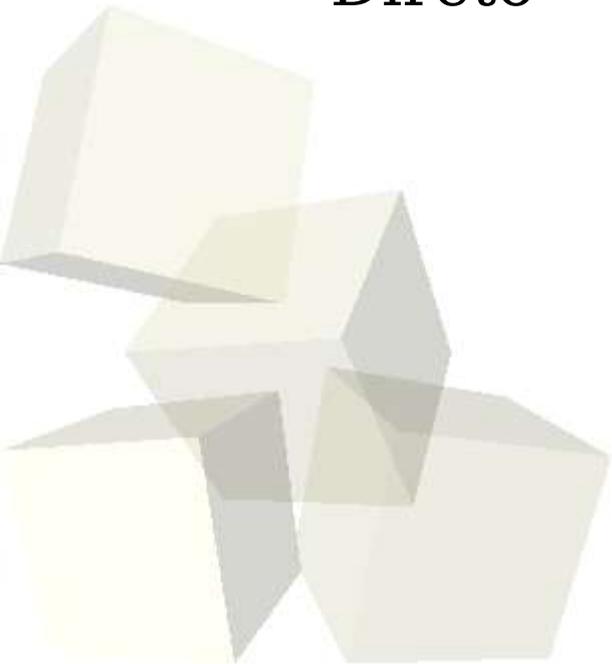
Hierarquia de Memória

Prof. Msc. Eduardo Luzeiro Feitosa
efeitosa@dcc.ufam.edu.br



Conteúdo

- Introdução
 - ◆ Exemplo da biblioteca
 - ◆ Princípio da localidade
- Hierarquia de memória
- Cache
 - ◆ Definições (terminologia)
 - ◆ Mapeamento
 - Direto





Introdução

- Nos últimos anos vem se investindo no aumento da velocidade dos processadores
- Porém, a velocidade de processamento de um sistema não é determinada somente pelo seu processador
- Não adianta ter o processador mais rápido do mundo se a alimentação de informações não consegue acompanhar o mesmo ritmo
- Como tanto o fornecimento dos dados como seu armazenamento após o processamento são efetuados na memória, a velocidade média de acesso a memória é importante no cálculo de velocidade de um sistema.



Introdução

- Além da velocidade, o tamanho da memória também é importante.
- O ideal seria:
 - ♦ Memória de tamanho ilimitado;
 - ♦ Memória com um tempo de acesso muito rápido.
- Entretanto, esse são objetivos contraditórios:
 - ♦ Por problemas tecnológicos, quanto maior a memória mais lenta será o seu tempo de acesso
- Solução:
 - ♦ Criar uma ilusão para o processador de forma que a memória pareça ilimitada e muito rápida.



Exemplo da Biblioteca

- Um estudante recebe a tarefa de fazer um trabalho sobre Redes de Computadores
- Ele vai a biblioteca, senta em uma baia e inicia sua pesquisa.
 - Supondo que ele leva um (1) minuto para procurar um livro nas estantes e leva um (1) minuto para ir buscar o livro.
- O estudante gastará 10 minutos para procurar em 5 livros.
- É claro que se ele esquecer algum livro ou tiver que buscar um livro novamente, esse tempo crescerá.



Exemplo da Biblioteca

- Agora, supondo que ao chegar na biblioteca, o estudante encontre uma mesa vazia (e com espaço suficiente)
- Então, ao invés de gastar 10 minutos, o estudante agora gastará 6 minutos
- Entretanto, existem situações complicadoras:
 - Todos os livros requeridos pelo estudante podem não caber na mesa;
 - Ao sair para tomar um café, um colega chega e pega a mesa.



Princípio da localidade

- É nesse contexto que se insere o princípio da localidade.
- Ele estabelece que os programas acessam uma parte relativamente pequena do seu espaço de endereçamento em um instante qualquer, assim como o estudante acessa uma parcela pequena de livros da biblioteca em um dado instante.
- Existem dois tipos de localidade:
 - ◆ Localidade temporal (localidade no tempo)
 - ◆ Localidade espacial (localidade no espaço)



Princípio da localidade

■ Localidade Temporal

- ♦ Se um item é referenciado, ele tende a ser referenciado novamente dentro de um espaço de tempo curto. Se o estudante tiver trazido o livro recentemente para sua mesa, é provável que o faça em breve novamente.

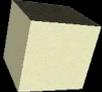
■ Localidade Espacial

- ♦ Se um item é referenciado, itens cujos endereços sejam próximos dele tendem a ser logo referenciados.

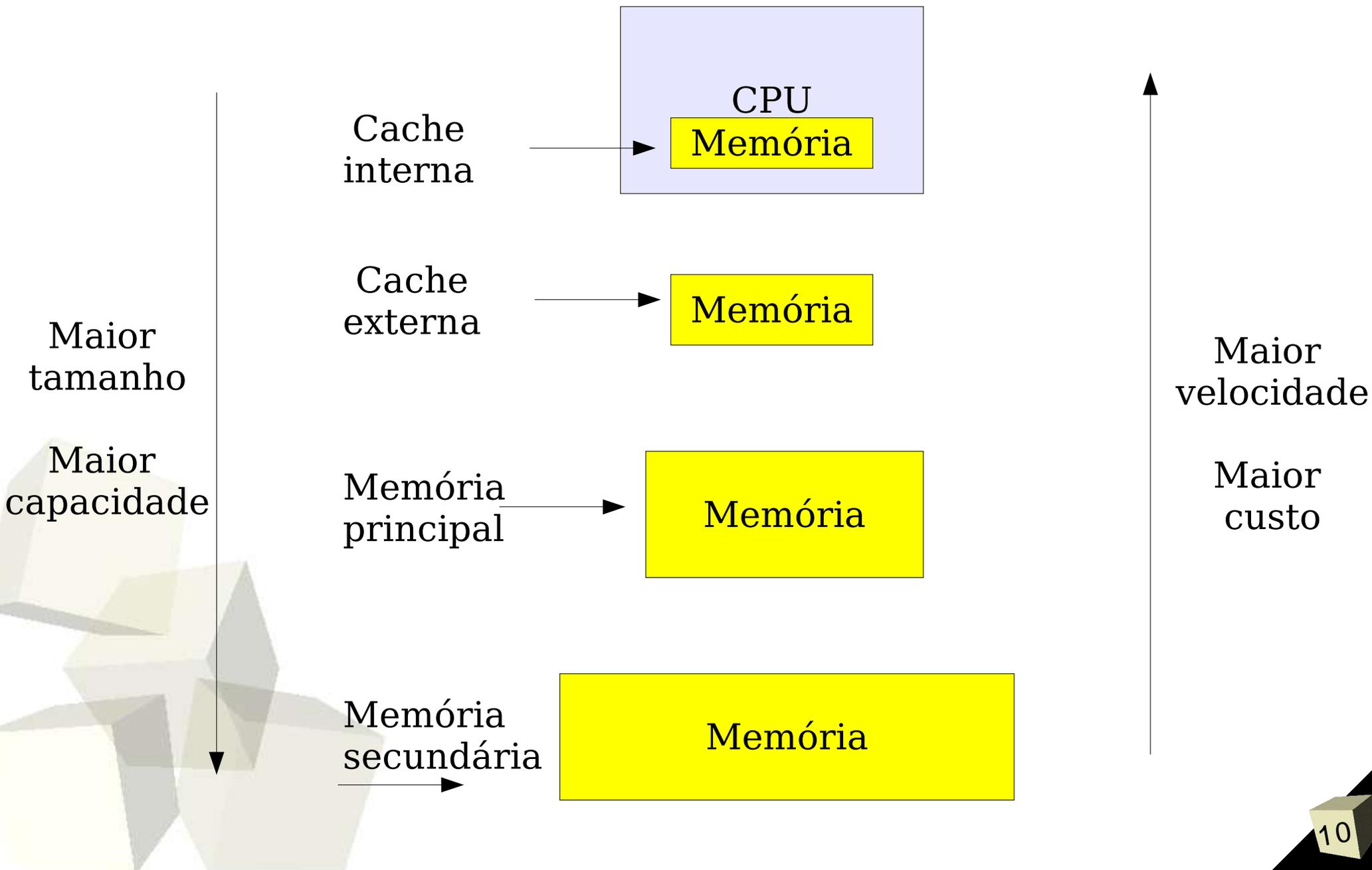


Hierarquia de Memória

- Ilusão de uma memória ilimitada e rápida obtida através da utilização de diversos níveis de acesso
- A hierarquia de memória explora o princípio da localidade
 - ◆ Localidade de memória é o princípio que diz que os próximos acessos ao espaço de endereçamento tendem a ser próximos
- A hierarquia é formada por:
 - ◆ Registradores;
 - ◆ Cache;
 - ◆ Memória principal;
 - ◆ Disco rígido;
 - ◆ CDROM, flexíveis, etc.

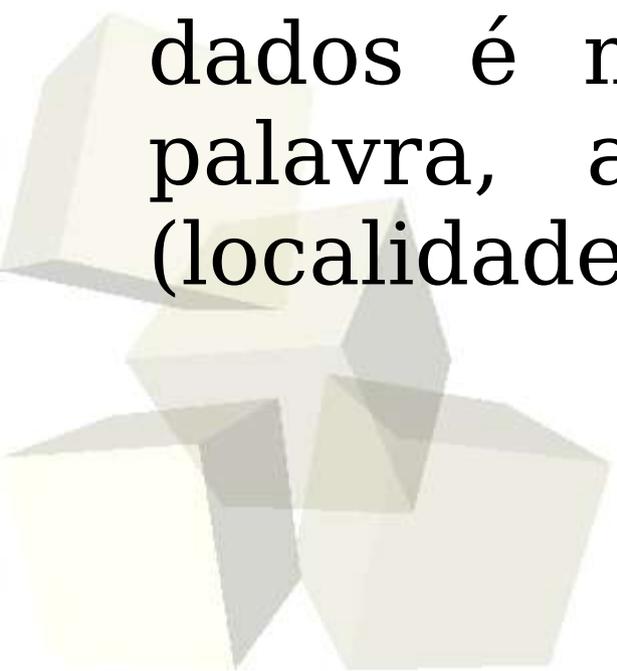


Hierarquia de Memória





Hierarquia de Memória

- A idéia de memória secundária já é aplicado a décadas.
 - Os dados são transferidos para níveis mais altos a medida que são usados
 - A transferência entre níveis é feita com grupos de palavra (bloco, página) pois o custo relativo de transferir um grupo de dados é menor do que para uma única palavra, além de já antecipar acessos (localidade espacial)
- 

Hierarquia de Memória

Tipo	Tempo de acesso	Tamanho	Custo (por MB)
Registradores	Ciclos de CPU	32-64 bits	---
Cache interna L1	Ciclos de CPU	32-64 Kbytes	---
Cache externa L2	8-35 ns	512 Kb - 2 Mb	50 Us\$
Memória Principal	40-120 ns	64 Mb - 1 Gb	1 Us\$
Memória secundária	5 ms	6 Gb - 128 Gb	0,02 Us\$



Hierarquia de Memória

- Vale lembrar que para movimentar dados entre os níveis são necessários mecanismos baseados em políticas
 - ♦ Ex: é preciso mover dados de um nível superior que já está cheio. Alguém deve ser retirado? Quem?
 - ♦ Uma decisão errada pode afetar todo o desempenho do sistema



Definições de Cache

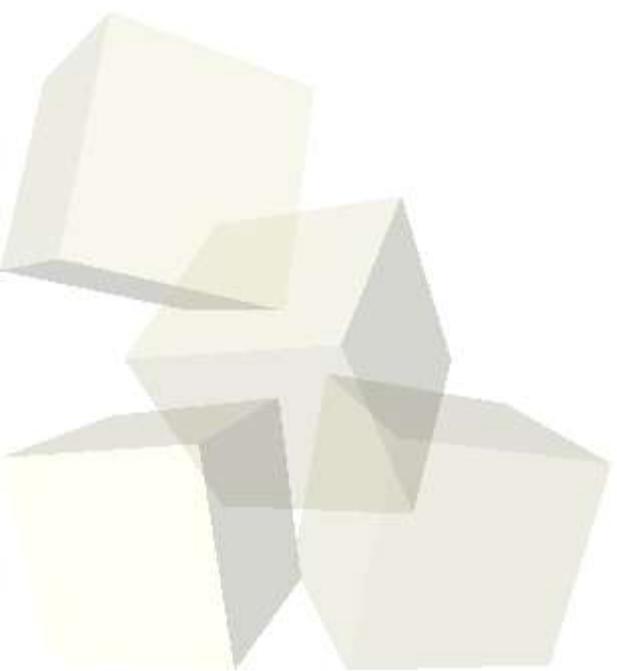
- Hit - dado encontrado no nível procurado.
- Miss - dado não encontrado no nível procurado.
- Hit-rate (ratio) - percentual de hits no nível.
- Miss-rate (ratio) – percentual de misses no nível. É complementar ao hit-rate.
- Hit-time – tempo de acesso ao nível incluindo tempo de ver se é hit ou miss.
- Miss-penalty – tempo médio gasto para que o dado não encontrado no nível desejado seja transferido dos níveis mais baixos.



Definições de Cache

- Calcule o tempo médio (tme) efetivo de acesso a uma memória cache considerando
 - ♦ Hit-ratio = 80%
 - ♦ Hit-time = 2 μ s
 - ♦ Miss-penalty = 10 μ s

- $T_{me} = \text{hit-time} + (1 - \text{hit-rate}) * \text{miss-penalty}$





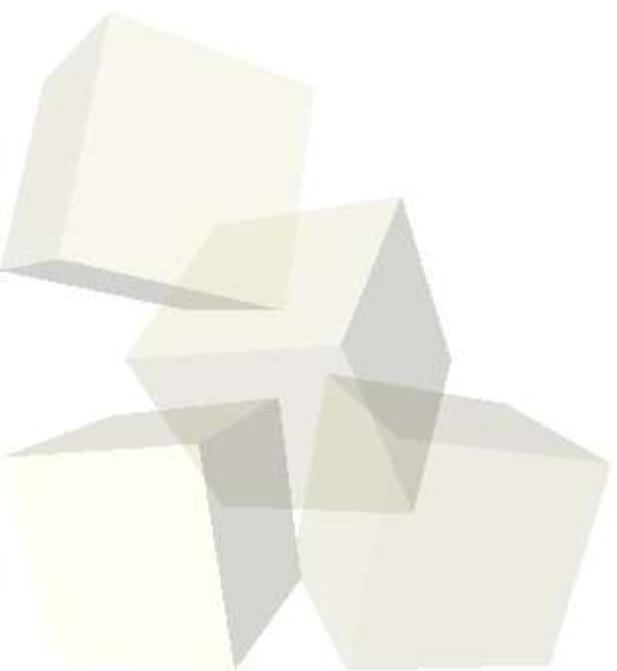
Memória Cache

- Cache significa lugar seguro para esconder (guardar) coisas
- Como a cache só pode ter parte dos dados por causa de seu tamanho, tem-se dois problemas:
 - ♦ Como identificar se o dado procurado está na cache;
 - ♦ Se ele estiver na cache, como acessá-lo de forma rápida.
- A solução é fazer o mapeamento de endereços



Mapeamento em Cache

- O termo mapeamento é usado para indicar o relacionamento dos dados do nível inferior com as posições da memória cache
- Existem três (3) tipos de mapeamento
 - ◆ Direto
 - ◆ Associativo
 - ◆ Conjuntivo associativo



Mapeamento em Cache

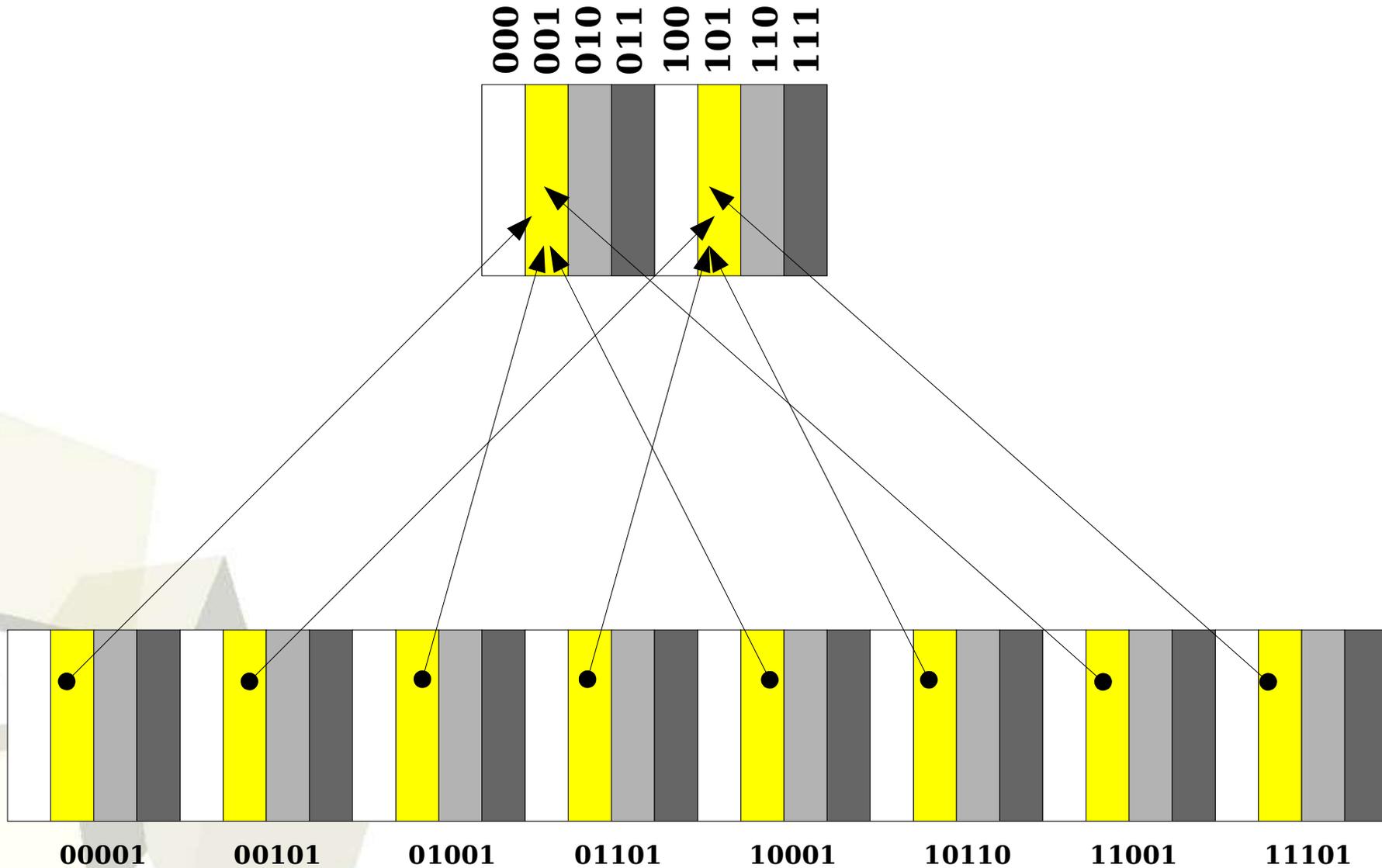
■ Mapeamento Direto

- É a forma mais simples de mapeamento;
- Cada bloco na memória principal é mapeado em uma linha da cache
- Este mapeamento é dado diretamente através de uma operação no endereço que se está procurando \Rightarrow **(Endereço do bloco) módulo (Número de blocos da cache)**
- Exemplo: Para uma cache de 8 posições e uma memória de 32 endereços teríamos palavras de 4 bits.
- Cada posição da cache pode ter 4 posições da memória
- Os 3 bits menos significativos são usados para indexar os blocos ($\log_2(8)$)



Mapeamento em Cache

■ Mapeamento Direto





Mapeamento em Cache

■ Mapeamento Direto

- ♦ Agora é preciso resolver um problema
- ♦ Como saber se o dado armazenado na cache corresponde ao solicitado? Em outras palavras, como saber se determinada palavra requisitada está ou não na cache?
- ♦ A resposta é de utilização de rótulos.
- ♦ Os rótulos só precisa conter a parte superior do endereço, correspondente aos bits que não estão sendo usados como índice na cache.
- ♦ No exemplo anterior, os rótulos tem 2 bits, já que dos 5 bits do endereço, 3 são usados para identificar os blocos.



Mapeamento em Cache

■ Mapeamento Direto

- ♦ Para realmente conseguirmos ter acesso a cache, ainda falta uma etapa
- ♦ Ainda é preciso saber reconhecer se um bloco da cache possui informação válida
 - Por exemplo, quando o processador inicia, a cache está vazia e os rótulos não tem nada
- ♦ Por isso é preciso saber quais rótulos precisam ser ignorados por não possuire informação.
- ♦ A solução mais simples é usar um bit de validade para indicar se uma entrada contém ou não um endereço válido.
- ♦ Se o bit for zero, não há necessidade comparar os rótulos.



Acesso a Cache

■ Estado inicial da cache

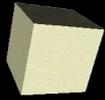
Índice	Bit validade	Rótulo	Informação
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		



Acesso a Cache

■ Solicitações do processador

Endereço da referência em decimal	Endereço da referência em binário	Falta ou acerto na cache	Bloco da cache
22	10110		$(10110 \text{ mod } 8) = 110$
26	11010		$(11010 \text{ mod } 8) = 010$
22	10110		$(10110 \text{ mod } 8) = 110$
26	11010		$(11010 \text{ mod } 8) = 010$
16	10000		$(10000 \text{ mod } 8) = 000$
3	00011		$(00011 \text{ mod } 8) = 011$
16	10000		$(10000 \text{ mod } 8) = 000$
18	10010		$(10010 \text{ mod } 8) = 010$



Acesso a Cache

- Estado após o tratamento da falta pela referência ao endereço (10110_2)

Índice	Bit validade	Rótulo	Informação
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	S	10	Memória (10100)
111	N		



Acesso a Cache

■ Solicitações do processador

Endereço da referência em decimal	Endereço da referência em binário	Falta ou acerto na cache	Bloco da cache
22	10110	Falta	$(10110 \text{ mod } 8) = 110$
26	11010		$(11010 \text{ mod } 8) = 010$
22	10110		$(10110 \text{ mod } 8) = 110$
26	11010		$(11010 \text{ mod } 8) = 010$
16	10000		$(10000 \text{ mod } 8) = 000$
3	00011		$(00011 \text{ mod } 8) = 011$
16	10000		$(10000 \text{ mod } 8) = 000$
18	10010		$(10010 \text{ mod } 8) = 010$



Acesso a Cache

- Estado após o tratamento da falta pela referência ao endereço (11010_2)

Índice	Bit validade	Rótulo	Informação
000	N		
001	N		
010	S	11	Memória (11010)
011	N		
100	N		
101	N		
110	S	10	Memória (10100)
111	N		

Acesso a Cache

■ Solicitações do processador

Endereço da referência em decimal	Endereço da referência em binário	Falta ou acerto na cache	Bloco da cache
22	10110	Falta	$(10110 \text{ mod } 8) = 110$
26	11010	Falta	$(11010 \text{ mod } 8) = 010$
22	10110	Acerto	$(10110 \text{ mod } 8) = 110$
26	11010	Acerto	$(11010 \text{ mod } 8) = 010$
16	10000		$(10000 \text{ mod } 8) = 000$
3	00011		$(00011 \text{ mod } 8) = 011$
16	10000		$(10000 \text{ mod } 8) = 000$
18	10010		$(10010 \text{ mod } 8) = 010$



Acesso a Cache

- Estado após o tratamento da falta pela referência ao endereço (10000_2)

Índice	Bit validade	Rótulo	Informação
000	S	10	Memória (10000)
001	N		
010	S	11	Memória (11010)
011	N		
100	N		
101	N		
110	S	10	Memória (10100)
111	N		

Acesso a Cache

■ Solicitações do processador

Endereço da referência em decimal	Endereço da referência em binário	Falta ou acerto na cache	Bloco da cache
22	10110	Falta	$(10110 \text{ mod } 8) = 110$
26	11010	Falta	$(11010 \text{ mod } 8) = 010$
22	10110	Acerto	$(10110 \text{ mod } 8) = 110$
26	11010	Acerto	$(11010 \text{ mod } 8) = 010$
16	10000	Falta	$(10000 \text{ mod } 8) = 000$
3	00011		$(00011 \text{ mod } 8) = 011$
16	10000		$(10000 \text{ mod } 8) = 000$
18	10010		$(10010 \text{ mod } 8) = 010$



Acesso a Cache

- Estado após o tratamento da falta pela referência ao endereço (00011_2)

Índice	Bit validade	Rótulo	Informação
000	S	10	Memória (10000)
001	N		
010	S	11	Memória (11010)
011	S	00	Memória (00011)
100	N		
101	N		
110	S	10	Memória (10100)
111	N		

Acesso a Cache

■ Solicitações do processador

Endereço da referência em decimal	Endereço da referência em binário	Falta ou acerto na cache	Bloco da cache
22	10110	Falta	$(10110 \text{ mod } 8) = 110$
26	11010	Falta	$(11010 \text{ mod } 8) = 010$
22	10110	Acerto	$(10110 \text{ mod } 8) = 110$
26	11010	Acerto	$(11010 \text{ mod } 8) = 010$
16	10000	Falta	$(10000 \text{ mod } 8) = 000$
3	00011	Falta	$(00011 \text{ mod } 8) = 011$
16	10000	Acerto	$(10000 \text{ mod } 8) = 000$
18	10010		$(10010 \text{ mod } 8) = 010$



Acesso a Cache

- Estado após o tratamento da falta pela referência ao endereço (10010_2)

Índice	Bit validade	Rótulo	Informação
000	S	10	Memória (10000)
001	N		
010	S	10	Memória (10010)
011	S	00	Memória (00011)
100	N		
101	N		
110	S	10	Memória (10100)
111	N		

Acesso a Cache

■ Solicitações do processador

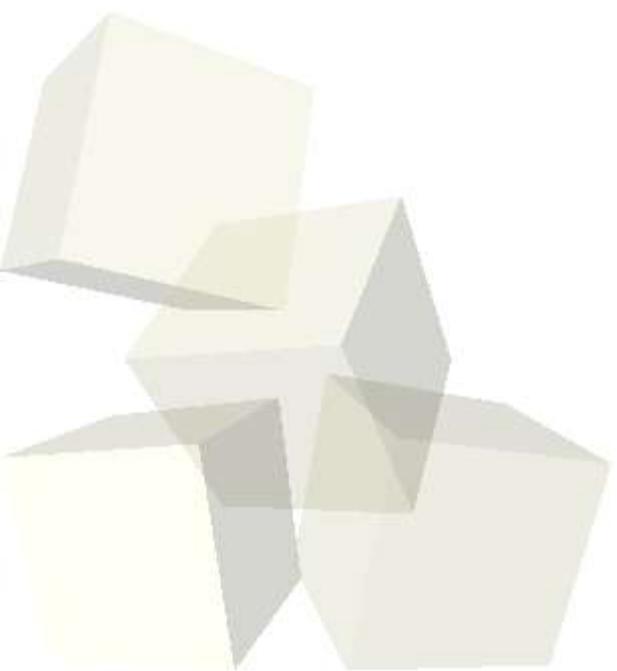
Endereço da referência em decimal	Endereço da referência em binário	Falta ou acerto na cache	Bloco da cache
22	10110	Falta	$(10110 \text{ mod } 8) = 110$
26	11010	Falta	$(11010 \text{ mod } 8) = 010$
22	10110	Acerto	$(10110 \text{ mod } 8) = 110$
26	11010	Acerto	$(11010 \text{ mod } 8) = 010$
16	10000	Falta	$(10000 \text{ mod } 8) = 000$
3	00011	Falta	$(00011 \text{ mod } 8) = 011$
16	10000	Acerto	$(10000 \text{ mod } 8) = 000$
18	10010	Falta	$(10010 \text{ mod } 8) = 010$



Trabalho de Casa

■ Resumo

- ◆ Citar as vantagens e desvantagens do Mapeamento Direto





Mapeamento em Cache

■ Mapeamento Associativo

- ♦ Caracteriza-se por um bloco da memória principal poder ser colocado em qualquer posição da cache, ou seja, um bloco de memória pode ser associado a qualquer entrada da cache.
- ♦ Isso produz 100% de aproveitamento da cache
- ♦ Conseqüentemente é preciso:
 - Pesquisar todas as entradas da cache para encontrar um determinado bloco, uma vez que tal bloco pode estar em qualquer lugar da cache.
 - Política de substituição, quando se tem falta (miss), a cache está cheia e é preciso tirar alguém.



Mapeamento em Cache

■ Mapeamento Associativo

- ♦ A solução para tornar a pesquisa rápida, é fazê-la em paralelo com um comparador associado (hardware) a cada uma das entradas da cache.
- ♦ Tais comparadores aumentam o custo de hardware, o que torna o mapeamento associativo ideal somente para pequenas caches, com capacidade para armazenar um pequeno número de blocos.



Mapeamento em Cache

■ Mapeamento Associativo

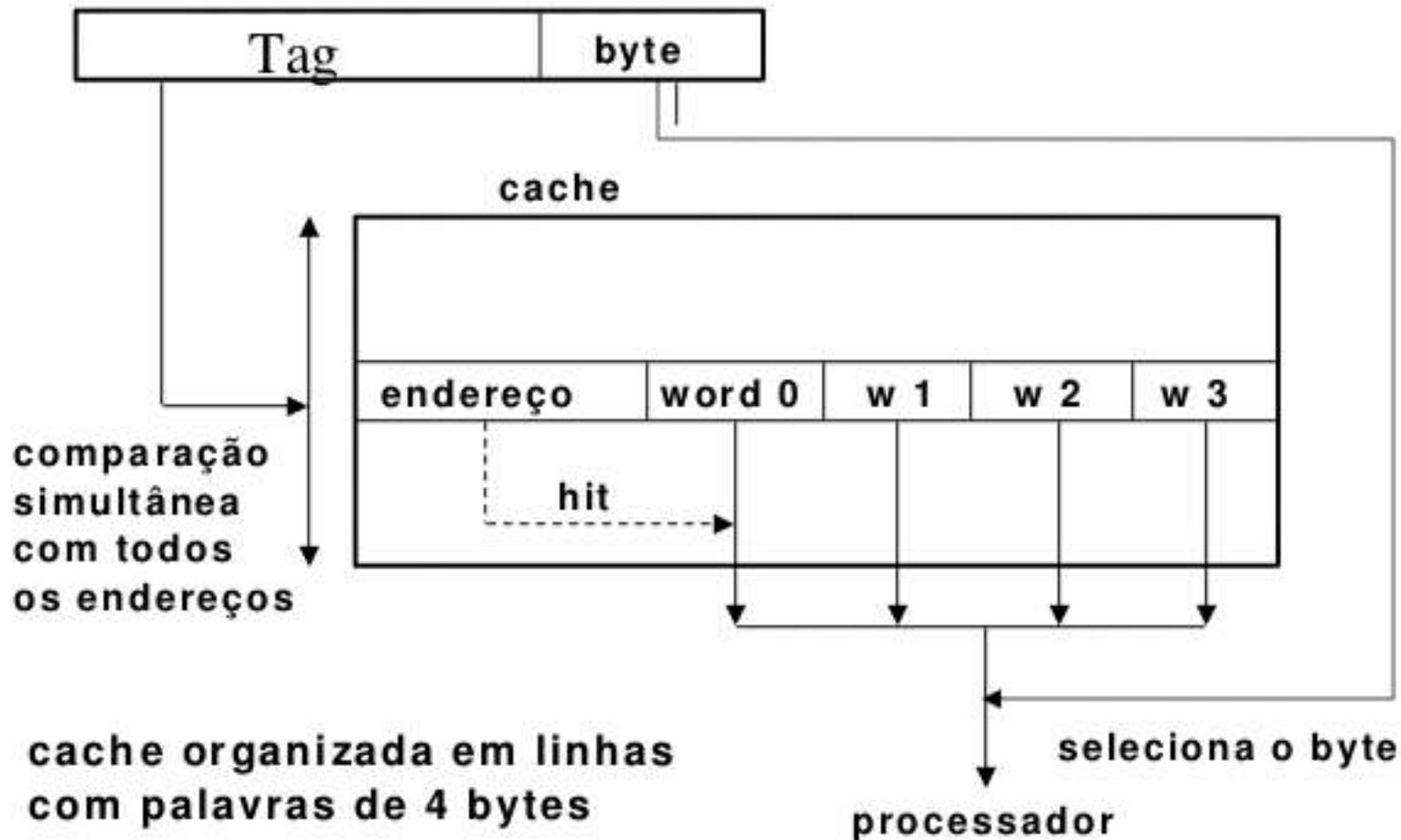
- ♦ Já a solução para substituição pode ser:
 - Randômica: escolher aleatoriamente uma posição a ser substituída
 - LFU (Least Frequent Used): a posição da cache que foi usada menos vezes será substituída. É preciso incrementar um contador a cada acesso e comparação para escolha
 - LRU (Least Recent Used): a posição da cache que foi usada a mais tempo será substituída. É preciso incrementar um contador a cada acesso e comparação para escolha.



Mapeamento em Cache

■ Mapeamento Associativo

- ♦ O endereço é dividido em um rótulo (tag) que identifica a linha e o número do identificador do byte





Mapeamento em Cache

- Passos para acesso usando Mapeamento Associativo
 1. Alimentar a memória associativa com o tag procurado.
 2. Se o tag procurado não está na cache acontece miss. Ir para 4.
 3. Se acontece hit, acessar a memória cache com o índice fornecido e efetuar leitura. FIM.
 4. Se não existir posição livre na cache, escolher um endereço para substituir (LRU).
 5. Buscar o endereço procurado no nível mais baixo e colocar a posição livre (ou escolhida) da cache cadastrando essa posição e a tag na memória e efetuar leitura. FIM.



Mapeamento em Cache

■ Mapeamento Associativo

◆ Vantagens

- Melhor aproveitamento das posições da cache, pois depois de cheia se tem 100% de aproveitamento
- Dados de controle não ficam na cache

◆ Desvantagens

- Memória associativa tem alto custo e tamanho limitado
- Limita o número de linhas da cache
- Necessita de política de substituição, que custa tempo e ainda pode-se escolher mal.



Mapeamento em Cache

- Mapeamento associativo conjuntivo
 - ♦ Meio termo entre mapeamento direto e associativo.
 - ♦ Uma cache associativa conjuntiva é dividida em S conjuntos (set) de N blocos
 - Se $S = 1$, mapeamento associativo;
 - Se $S = N$, mapeamento direto
 - ♦ Um endereço da memória principal pode ser mapeado para qualquer endereço no conjunto (endereço mod S) da cache
 - ♦ É preciso fazer procura dentro do conjunto
 - ♦ Também faz uso de política para substituição



Mapeamento em Cache

■ Passos para acesso usando Mapeamento Associativo conjuntivo

1. Calcular o módulo do endereço procurado pelo número de conjuntos S da cache.
2. Alimentar a memória associativa deste conjunto com o tag procurado.
3. Se o tag procurado não está na cache acontece miss. Ir para 5.
4. Se acontece hit, acessar a memória cache com o índice fornecido e efetuar leitura. FIM.
5. Se não existir posição livre na cache, escolher um endereço para substituir (LRU).
6. Buscar o endereço procurado no nível mais baixo e colocar a posição livre (ou escolhida) da cache cadastrando essa posição e a tag na memória e efetuar leitura. FIM.



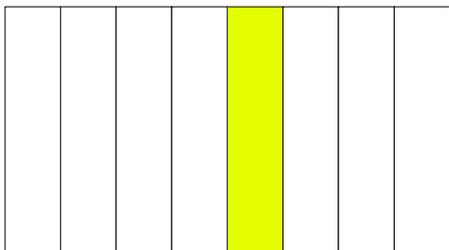
Mapeamento em Cache

- Mapeamento Associativo conjuntivo
 - ♦ Vantagens
 - Aumenta o tamanho da cache mantendo o tamanho da memória associativa
 - Bastante flexível
 - Usa a totalidade da área da cache para dados
 - ♦ Desvantagens
 - Tem alto custo e tamanho limitado
 - Necessita de política de substituição
 - Tempo acesso maior devido ao cálculo
 - Dependendo da geração de endereços não se aproveita a totalidade das posições da cache



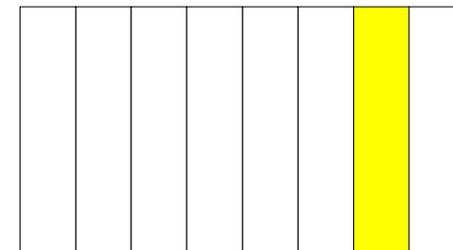
Mapeamento em Cache

bloco 0 1 2 3 4 5 6 7

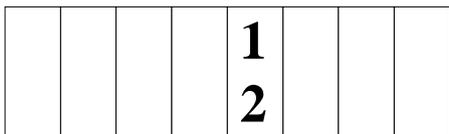


Direto

Associativa



Rótulo



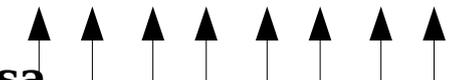
Rótulo



Pesquisa



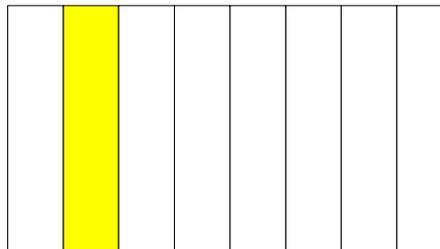
Pesquisa



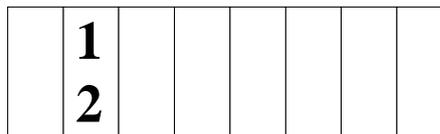
Associativa por Conjunto

No. do Conjunto

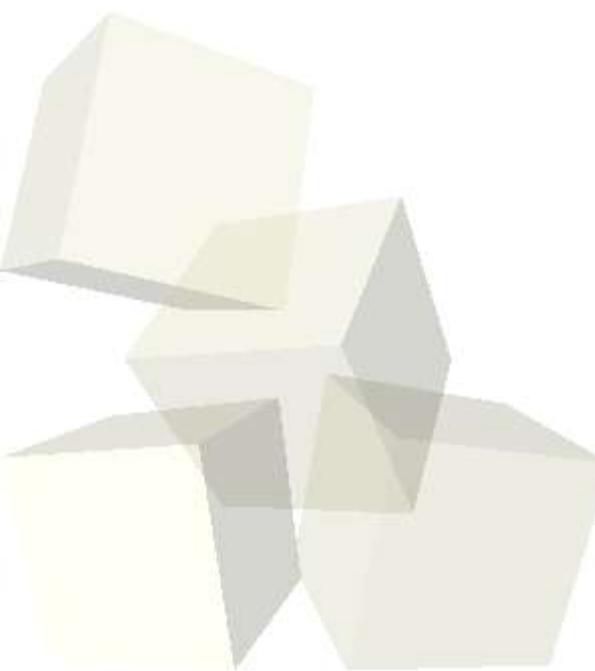
0 1 2 3



Rótulo



Pesquisa



Associatividade nas Caches

- Três (3) caches, cada qual com 4 blocos de uma palavra. A primeira é totalmente associativa, a segunda é associativa conjuntiva com 2 posições e a última é direta.
- Encontrar o número de faltas (miss) em cada uma delas, considerando a seguinte sequência de endereços de blocos: 0, 8, 0, 6, 8.

Associatividade nas Caches

■ Mapeado diretamente

Endereço do bloco	Bloco na cache
0	$(0 \bmod 4) = 0$
6	$(6 \bmod 4) = 2$
8	$(8 \bmod 4) = 0$

Endereço do bloco de memória acessado	Falta ou acerto	Conteúdo do bloco da cache após a referência			
		0	1	2	3
0	falta	Memória[0]			
8	falta	Memória[8]			
0	falta	Memória[0]			
6	falta	Memória[0]		Memória[6]	
8	falta	Memória[8]		Memória[6]	

Associatividade nas Caches

■ Mapeamento associativo conjuntivo

Endereço do bloco	Conjunto na cache
0	$(0 \bmod 2) = 0$
6	$(6 \bmod 2) = 0$
8	$(8 \bmod 2) = 0$

Endereço do bloco de memória acessado	Falta ou acerto	Conteúdo do bloco da cache após a referência			
		Conjunto 0	Conjunto 1	Conjunto 2	Conjunto 3
0	falta	Memória[0]			
8	falta	Memória[0]	Memória[8]		
0	acerto	Memória[0]	Memória[8]		
6	falta	Memória[0]	Memória[6]		
8	falta	Memória[8]	Memória[6]		

Associatividade nas Caches

- Mapeamento totalmente associativo

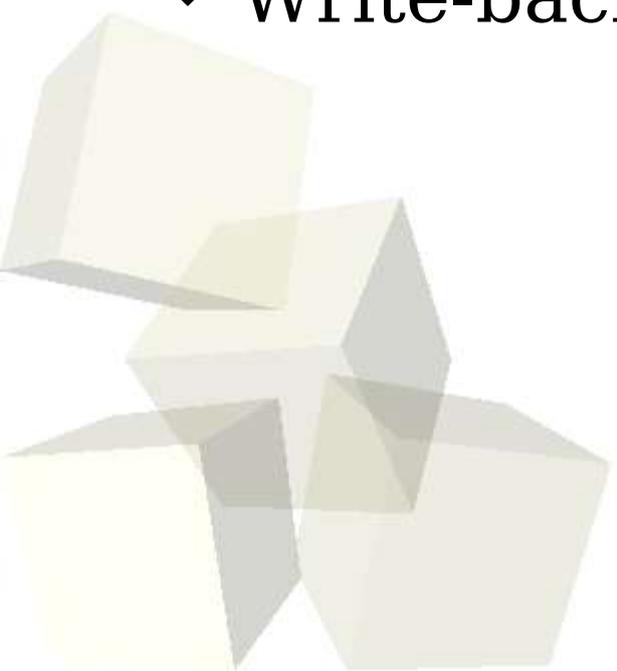
Endereço do bloco de memória acessado	Falta ou acerto	Conteúdo do bloco da cache após a referência			
		Conjunto 0	Conjunto 1	Conjunto 2	Conjunto 3
0	falta	Memória[0]			
8	falta	Memória[0]	Memória[8]		
0	acerto	Memória[0]	Memória[8]		
6	falta	Memória[0]	Memória[8]	Memória[6]	
8	acerto	Memória[0]	Memória[8]	Memória[6]	

Integridade de dados na cache

- Problema: ocorreu uma falta (miss) e o endereço desejado foi buscado no nível inferior da hierarquia de memória
- Mas a cache está cheia e não há lugar para escrever este dado
- Um algoritmo de substituição é acionado e uma posição é escolhida, só que estes dados foram alterados e não podem ser simplesmente descartados.
- Este problema ocorre porque uma escrita foi efetuada apenas no nível da cache e as cópias deste dado nos outros níveis não estão atualizadas.

Integridade de dados na cache

- Perguntas:
 - ♦ Como saber que os dados foram alterados?
 - ♦ Como salvar essas alterações?
 - ♦ Em que momento salvar as informações?
- Existem duas técnicas para manter a integridade dos dados
 - ♦ Write-through
 - ♦ Write-back



Integridade de dados na cache

■ Write-Through

- ♦ É a técnica mais antiga
- ♦ Escreve-se as alterações em todos os níveis
 - Quando (sempre)
 - Quanto (somente a palavra alterada)
- ♦ Estatisticamente, somente de 5% a 34% dos acessos a memória são escritas
- ♦ Vantagens
 - Dados sempre atuais
 - Escreve somente o necessário
- ♦ Desvantagens
 - Escreve-se muitas vezes
 - Uso maior do barramento

Integridade de dados na cache

■ Write-Back

- ♦ É a técnica mais recente
- ♦ Escreve-se as alterações somente quando é preciso ser substituída
- ♦ A estratégia mais simples é escrever mesmo que a linha não tenha sido modificada
- ♦ A estratégia alternativa é só escrever de volta quando a linha foi modificada
- ♦ Como sei que foi alterado
 - Dirty-bit (bit de sujeira) é setado
- ♦ Vantagens
 - Escreve menos vezes
 - Usa menos o barramento
- ♦ Desvantagens
 - Escreve mais dados de cada vez
 - Aumenta o tempo de substituição

Integridade de dados na cache

■ Passos de leitura da cache

1. Verificar se foi hit, se não foi vai para 3.
2. Procura o bloco desejado (tag ou direto), ler e repassar ao processador. Ir para 8.
3. Requisito nível mais baixo.
4. Receber bloco, procurar onde colocar e se cache cheia vai para 5. Se houver posição livre, escrever bloco, ler a palavra desejada no bloco, repassar dado ao processador e ir para 8.
5. Procurar bloco para substituir.
6. Se Write-back, salvar o bloco a ser substituído no nível mais baixo.
7. Substituir bloco, ler palavra desejada e repassar dado ao processador.
8. Pronto.

Integridade de dados na cache

■ Passos de escrita na cache

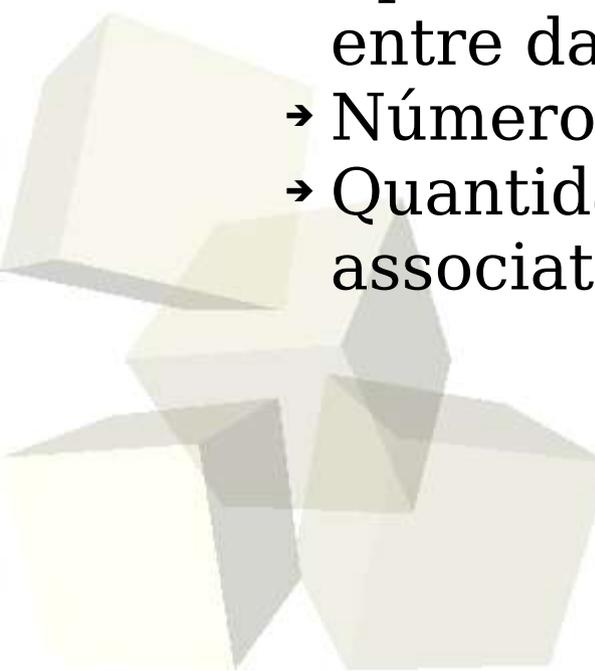
1. Verificar se foi hit, se não foi vai para 3.
2. Procura o bloco desejado (tag ou direto) e escrever. Se Write-through escrever palavra também nos níveis mais baixos. Se Write-back setar dirty-bit. Ir para 8.
3. Requisitar nível mais baixo.
4. Receber bloco, procurar onde colocar e se cache cheia vai para 5. Se houver posição livre, escrever bloco e efetuar escrita da palavra. Se Write-through escrever palavra nos níveis mais baixos. Se Write-back seta dirty-bit e ir para 8.
5. Procurar bloco para substituir.
6. Se Write-back e dirty-bit ligado, salvar o bloco a ser substituído no nível mais baixo.
7. Substituir bloco e escrever palavra. Se Write-through escrever palavra nos níveis mais baixos. Se Write-back seta dirty-bit
8. Pronto.



Exercícios

■ Dimensionamento de memória cache

1. A área de memória disponível para implementação de uma cache L2 é 512 Kbytes. Considerando que a memória a ser endereçada possui 64 Mbytes (2^{26}) e a cache deve trabalhar com blocos de 16 palavras de 32 bits. Calcule para as três técnicas (direta, totalmente associativa e associativa conjuntiva com 4 conjuntos):

- Divisão de bits de endereço
 - Aproveitamento efetivo da área da cache (relação entre dados e controle)
 - Número de linhas
 - Quantidade e tamanho em Kbytes das memórias associativas (quando necessário)
- 



Exercícios

■ Cache com mapeamento associativo conjuntivo com 4 conjuntos

- ♦ Divisão de endereços
 - Endereço de 26 bits (64 Mbytes)
 - 20 bits (tag)
 - 2 bits (conjunto)
 - 4 bits (palavra)
- ♦ Aproveitamento da cache
 - 100% afinal somente dados ficam na cache
- ♦ Número de linhas
 - Tamanho da linha?
 - Cada linha tem bloco de 16 palavras de 32 bits, ou seja, $16 * 32 = 512$ bits / 8 = 64 bytes
 - Quantas linhas cabem na cache?
 - Cache tem 512 kbytes, então $512 * 1024 = 524288$ bytes / 64 bytes = 8192 linhas.



Exercícios

■ Cache com mapeamento associativo conjuntivo com 4 conjuntos

- ♦ Tamanho das memórias associativas
 - Quantas?
 - Uma para cada conjunto, ou seja, **4**
 - Tamanho de cada uma?
 - Cada linha tem tag de 20 bits
 - O número de linhas é igual ao número de linhas da cache que ela endereça. Então, como a cache tem 8192 linhas e são 4 memórias, cada memória associativa endereça $8192 / 4 = \mathbf{2048 \text{ linhas}}$
 - Uma MA tem $2048 \text{ (linhas)} * 20 \text{ (tag)} = 40960 \text{ bits} / 8 = 5120 \text{ bytes} / 1024 = \mathbf{5 \text{ kbytes}}$.

Exercícios

■ Cache com mapeamento totalmente associativo

- ♦ Divisão de endereços
 - Endereço de 26 bits (64 Mbytes)
 - 22 bits (tag) e 4 bits (palavra)
- ♦ Aproveitamento da cache
 - 100% afinal somente dados ficam na cache
- ♦ Número de linhas
 - Tamanho da linha?
 - Cada linha tem bloco de 16 palavras de 32 bits, ou seja, $16 * 32 = 512$ bits / 8 = 64 bytes
 - Quantas linhas cabem na cache?
 - Cache tem 512 kbytes, então $512 * 1024 = 524288$ bytes / 64 bytes = 8192 linhas.



Exercícios

■ Cache com mapeamento totalmente associativo

- ♦ Tamanho das memórias associativas
 - Quantas?
 - Uma única memória associativa
 - Tamanho?
 - Cada linha tem tag de 22 bits
 - O número de linhas é igual ao número de linhas da cache que ela endereça. Como a cache tem 8192 linhas, então ela endereça **8192 linhas**
 - Uma MA tem $8192 \text{ (linhas)} * 22 \text{ (tag)} = 180224 \text{ bits} / 8 = 22528 \text{ bytes} / 1024 = \underline{22 \text{ kbytes}}$.





Exercícios

■ Cache com mapeamento direto

- ♦ Número de linhas
 - Tamanho da linha?
 - Conteúdo de cada linha
 - . 1 bit de validade, ? bit de tag e 512 (bloco)
 - Cada linha tem um bit de validade, os bits de tag e bloco de 16 palavras de 32 bits. O problema é o tamanho da tag pois como ele depende do número de linhas da cache, que é o que se está calculando. A solução é experimentar.
 - . Com 12 bits para linha, pode-se endereçar 4096 linhas (poucos bits)
 - . Com 14 bits para linha, pode-se endereçar 16384 linhas (muitas linhas)
 - . Com 13 bits para linha, pode-se endereçar 8192 linhas (ok)



Exercícios

■ Cache com mapeamento direto

- ♦ Divisão de endereços
 - Endereço de 26 bits (64 Mbytes)
 - 9 bits (tag), 13 bits (linha) e 4 bits (palavra)
- ♦ Aproveitamento da cache
 - Dados em cada linha: um bloco de 16 palavras de 32 bits = 512 bits
 - Tamanho total da linha = 1 (validade) + 9 (tag) + 512 = 522 bits
 - Percentual de aproveitamento:
 - 522 -> 100%
 - 512 -> X%
 - Aproveitamento efetivo de 98,08%
- ♦ Memória associativa não usada



Exercícios

■ Dimensionamento de memória cache

1. A área de memória disponível para implementação de uma cache L2 é 256 Kbytes. Considerando que a memória a ser endereçada possui 256 Mbytes (2^{28}) e a cache deve trabalhar com blocos de 8 palavras de 16 bits. Calcule para as três técnicas (direta, totalmente associativa e associativa conjuntiva com 2, 4 e 16 conjuntos):

- Divisão de bits de endereço
- Aproveitamento efetivo da área da cache (relação entre dados e controle)
- Número de linhas
- Quantidade e tamanho em Kbytes das memórias associativas (quando necessário)



FIM

