“Arquitetura e Organização de Computadores I – Aula_09 – Hierarquia de Memória”

Prof. Dr. Emerson Carlos Pedrino

DC/UFSCar

São Carlos

Sumário

- Estudo do sistema hierárquico de memória
- Memória *cache*
- Memória virtual

Memória SRAM

- SRAM (*Static Random Access Memory*): suas células são essencialmente *flip-flops* que permanecem em um dado estado indefinidamente, desde que a alimentação do circuito não seja interrompida.
- Um circuito de memória estática SRAM é mais rápido que os circuitos de memórias dinâmicas DRAM (*Dynamic Random Access Memory*).
- Possuem maior consumo de energia que as DRAMs.

Memória DRAM

- DRAM (*Dynamic Random Access Memory*): *bits* armazenados como cargas em capacitores.
- Esses capacitores se descarregam com o tempo, logo, o conteúdo desse tipo de memória deve ser reavivado (*refreshed*) em intervalos de tempo de alguns milissegundos.
- Esse circuito é mais lento (5 a 10 x) que o circuito de memória SRAM. Porém, um circuito DRAM é menor e, portanto, de custo, por *bit*, menor que um circuito SRAM.

Resumo

- SRAM: tempo de acesso de 2 a 25 ns e custo por *Mbyte* em torno de R$ 100,00.
- DRAM: tempo de acesso de 60 a 120 ns e custo em torno de R$ 1,00 por *Mbyte*.
- Disco magnético: tempo de acesso de 10 a 20 milhões de nanossegundos e custo de alguns centavos de reais por *Mbyte*.

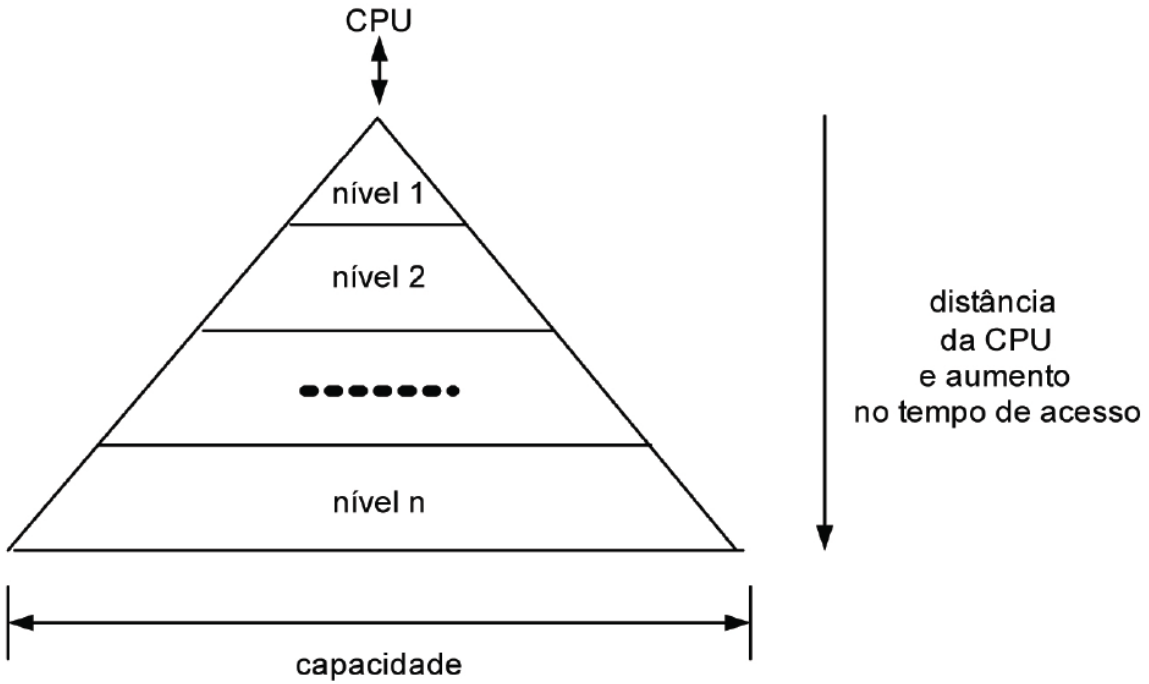
Resumo

- É mais difícil projetar um sistema com DRAMs do que com SRAMs.
- A maior capacidade e o menor consumo fazem com que as DRAMs sejam escolhidas em sistemas onde as considerações de projeto mais importantes são: tamanho, custo e consumo reduzidos.
- Para aplicações onde a velocidade e a complexidade são mais críticas do que o custo, o espaço e as considerações de consumo, as SRAMs ainda são melhores.

Observações

- Processadores: precisam de memórias de grande capacidade e rápidas.
- Quesito capacidade: atendido pelos discos magnéticos.
- Quesito rapidez no acesso: atendido pelas memórias SRAM.
- Para conciliar o uso dos tipos de memória anteriores, é proposta a construção de um sistema de hierarquia de memória, onde várias tecnologias de memória convivem para servir o processador.

Sistema hierárquico de memória



Obs.: no diagrama, cada nível representa um tipo de memória.

Sistema hierárquico de memória

- Memória: referenciada pela CPU, em busca de um conteúdo.
- Conteúdo: presente por inteiro na memória de nível *n*.
- Níveis superiores: estão presentes cópias de partes do conteúdo.

Sistema hierárquico de memória

- Referência à memória: começa sempre pelo nível superior, mais próximo da CPU.
- Se a memória de nível 1, por exemplo, consiste no conteúdo referenciado, a referência termina.
- Caso um nível superior não contenha o dado referenciado, o nível imediatamente inferior é consultado.

Sistema hierárquico de memória

- O processo anterior termina quando a consulta for satisfeita por um dos níveis da hierarquia.
- Em seguida, uma cópia de uma porção da memória referenciada (um bloco) é transferida para todos os níveis superiores.
- Um bloco, normalmente, contém mais do que uma palavra referenciada pela CPU, e o seu tamanho aumenta à medida que o nível hierárquico da memória se torna inferior.

Sistema hierárquico de memória

- Como as memórias de nível superior têm capacidade menor, mantêm apenas algumas cópias de blocos.
- Assim, quando existe a transferência de bloco e não há espaço num nível superior, um bloco desse nível deve ser substituído pelo novo bloco.

Sistema hierárquico de memória

- Sistema de hierarquia de memória eficiente -> CPU faz a maioria de suas referências nas memórias de nível superior.
- Princípio de localidade: a CPU faz referências aos mesmos blocos por duas razões, a primeira é a localidade temporal, que é a propriedade da CPU fazer referências repetidas à mesma localidade, e a segunda é a localidade espacial, onde as referências subsequentes da CPU são os endereços próximos ou mesmo subsequentes da memória.

Localidade Espacial

- Os programas são sequenciais, e as instruções sequenciais ficam armazenadas em posições subsequentes na memória.
- Exs: a manipulação de matrizes, vetores e outras estruturas de dados implica em referências a endereços subsequentes.

Localidade Espacial

- Para se usufruir da localidade espacial, um bloco deve conter mais que uma palavra de memória referenciada pela CPU.
- Assim, quando a CPU faz uma referência, um bloco contendo várias palavras é carregado na memória de nível superior, e existe a probabilidade de que outra palavra contida no mesmo bloco seja referenciada subsequentemente.

Localidade Temporal

- Existe pela implementação de *loops*, sub-rotinas ou outros controles que acabam provocando a repetição de um mesmo código.

Hit ou *Miss*

- *Hit*: acerto -> dado requisitado no nível superior.
- *Miss*: falta -> o dado requisitado não está no nível superior.

Memória *cache*

- *Cache*: nível superior; memória rápida (SRAM); contém cópias dos dados da memória principal (nível inferior, DRAM).
- A memória principal contém mais blocos que a *cache*. Assim, diferentes blocos da memória irão compartilhar posições (*slots*) na *cache*.

Memória *cache*

- Questões:
  - Como saber se um dado item está na *cache*?
  - E se estiver, como encontrá-lo?

  Obs.: as respostas irão depender do tipo de mapeamento da memória principal para a *cache*, que pode ser: mapeamento direto, mapeamento associativo e mapeamento associativo por conjunto.

Cache em mapeamento direto

- Para a carga da *cache* com as palavras mais referenciadas pela CPU, a MP é dividida logicamente em certo número de blocos de tamanho fixo.
- Módulo: conjunto de blocos cujo tamanho coincide com o tamanho da *cache*.

Cache em mapeamento direto

- Pode-se dizer, também, que a memória é dividida em certo número de módulos de tamanho fixo.
- *Cache*: dividido em *slots* de mesmo tamanho dos blocos.
- Cada bloco subsequente da memória principal é carregado num *slot* subsequente no *cache* quando referenciado pela CPU.

Cache em mapeamento direto

- Para carregar blocos subsequentes ao módulo anterior, os *slots* começam novamente, a partir do primeiro *slot* da *cache*. Logo, o conteúdo anterior da *cache* é substituído.
- Numeração dos módulos consecutivos: usada como rótulos, *tags*, para identificar os blocos contidos nos *slots* da *cache*.

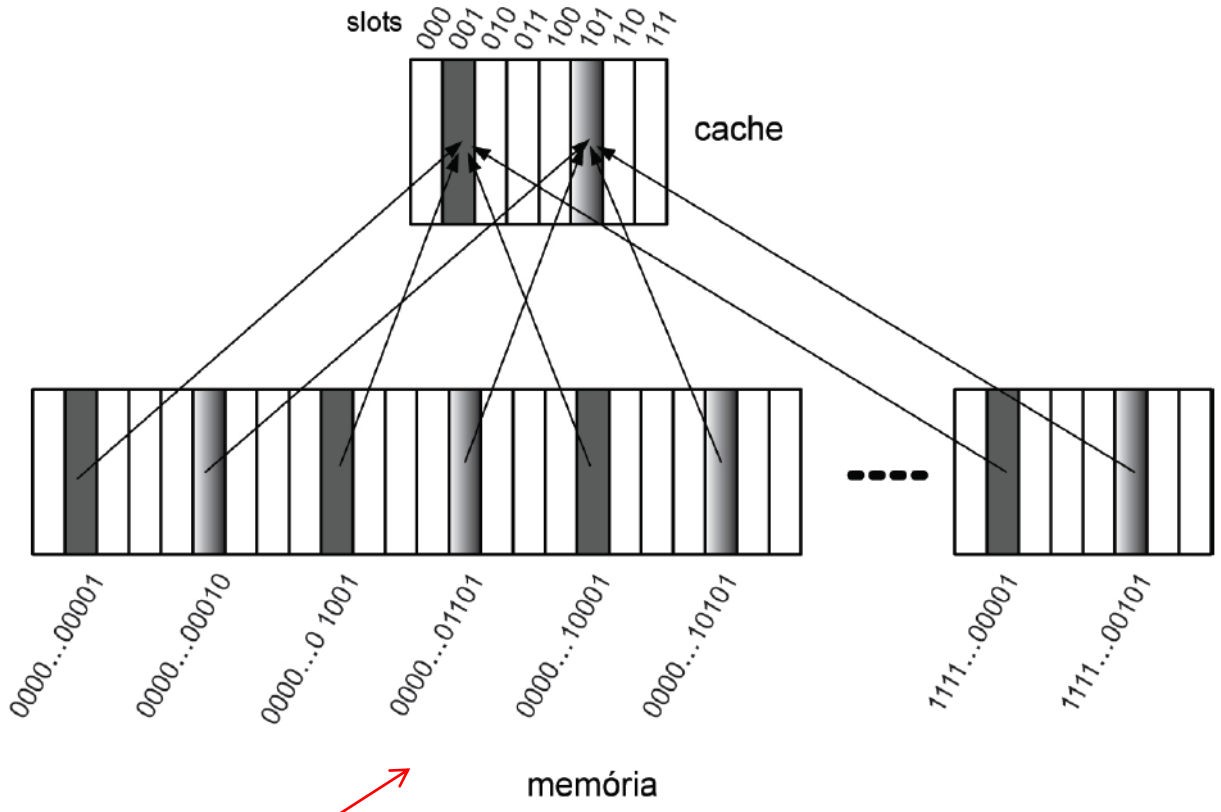
Cache em mapeamento direto

- No mapeamento direto: o número do bloco de memória, dentro de um módulo, coincide com o número do *slot* da *cache*.
- Cálculo do *slot* a ser usado: dividir o endereço de memória pelo número de *slots*. O resto da divisão é o número do *slot*.

Exemplo

- Para uma *cache* de 8 *slots*, o endereço 9 é copiado no *slot* 1, pois:
  - 9 mod 8 = 1.

Outro exemplo

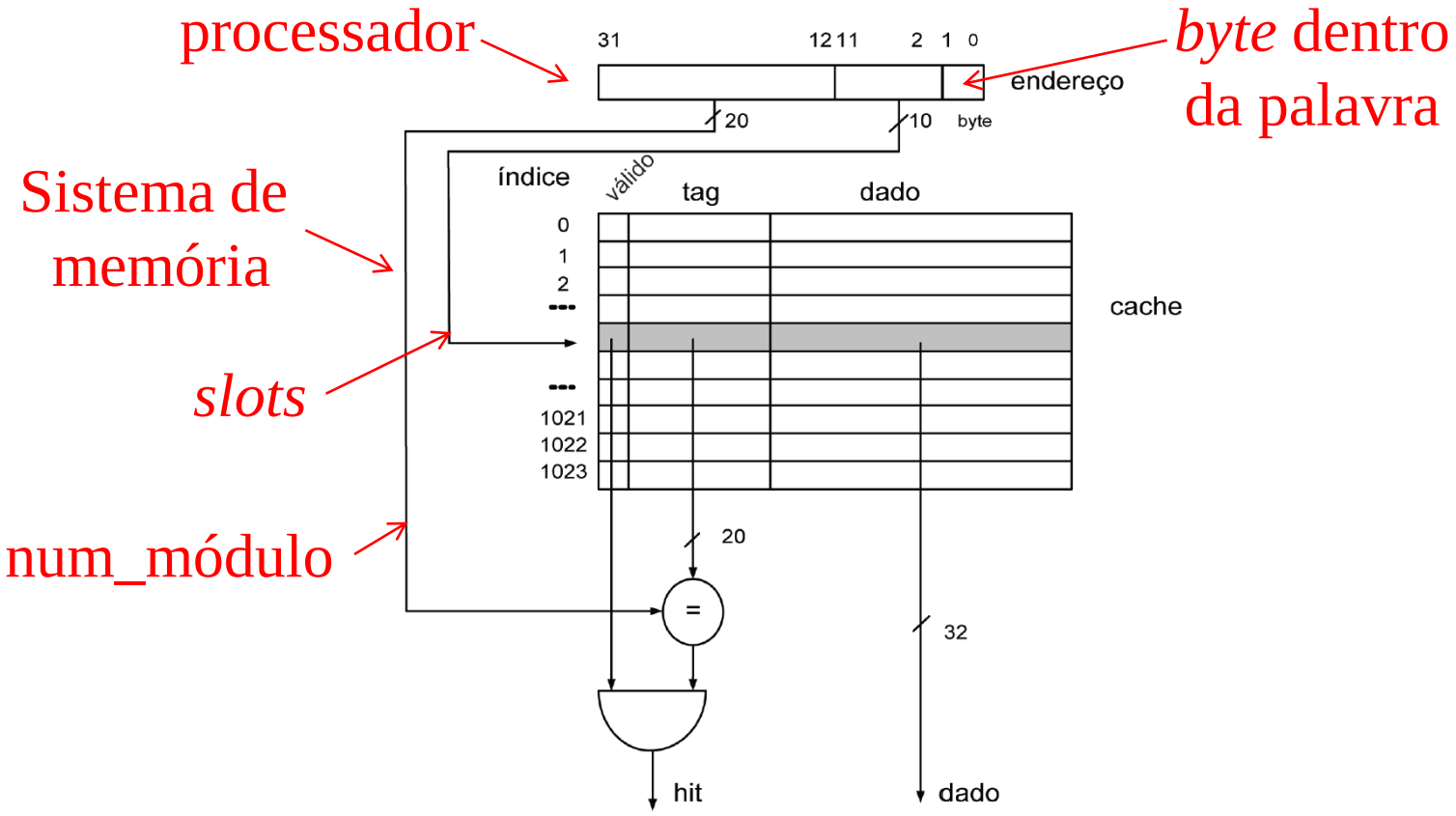


Obs.: diagrama ilustrativo de um mapeamento direto numa cache de 8 *slots*.

Cache em mapeamento direto

- Cada bloco de memória tem apenas um *slot* na *cache* para o mapeamento.
- Como vários blocos de memória compartilham um mesmo *slot*, como saber qual bloco de memória está no *slot*?
  - A resposta está na próxima figura.

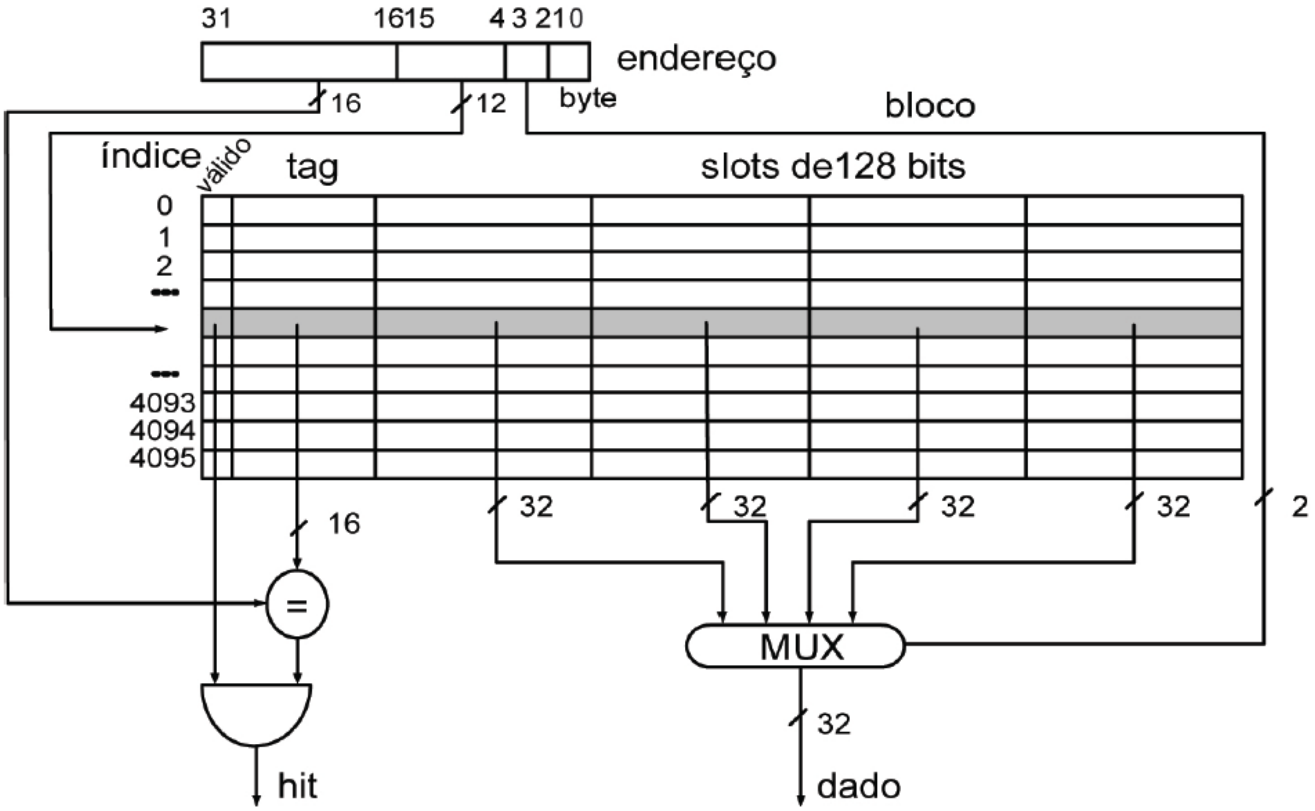
Diagrama da organização da *cache* em mapeamento direto



Cache organizado em mapeamento direto com exploração da localidade espacial

- Para que a localidade espacial seja explorada, um bloco de memória deve conter mais de uma palavra referenciada pela CPU.
- Uma nova referência feita pela CPU apresenta alta probabilidade de ser encontrada no mesmo *slot* da referência anterior.

Cache organizado em mapeamento direto, com 4 palavras por bloco



Observações

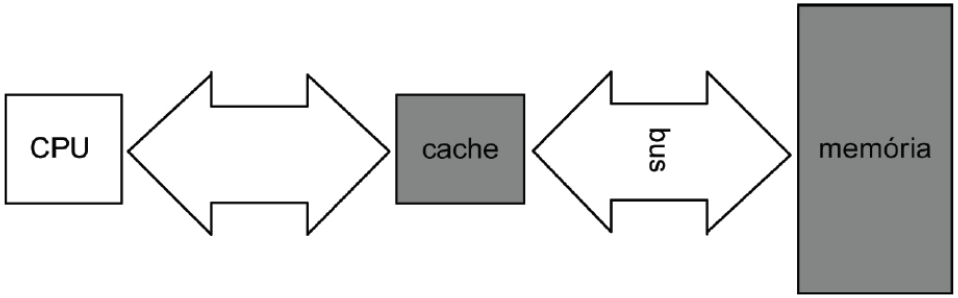
- O que se deseja numa referência à memória é o acerto (*hit*), tanto na leitura quanto na escrita.
- Quando ocorre uma falta (*miss*) de leitura, a CPU para enquanto se faz a busca de um bloco na memória para carregar na *cache*.
- Quando ocorre um acerto de escrita, existem duas alternativas, vistas a seguir.

Observações

- 1. Escrever na *cache* e na memória -> *write-through*.
- 2. Escrever somente na *cache* -> a escrita na memória, *write-back*, é feita mais tarde, na substituição do bloco.
- Quando ocorre uma falta na escrita -> o bloco correspondente deve ser carregado na *cache*, e, depois, a palavra deve ser escrita usando uma das duas alternativas anteriores.

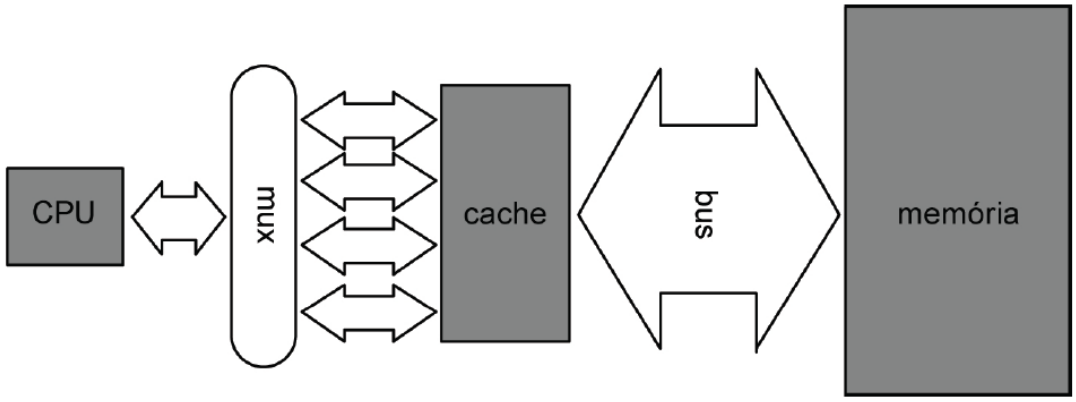
Interconexão da *cache* com a memória principal

- Interconexão da *cache*: importante para que um sistema de memória hierárquico seja eficiente.
- Ex: sistema de interconexão da MP com a *cache*, num barramento do tamanho de uma palavra. Exploração da localidade de referência temporal.



Sistema de interconexão com barramento largo

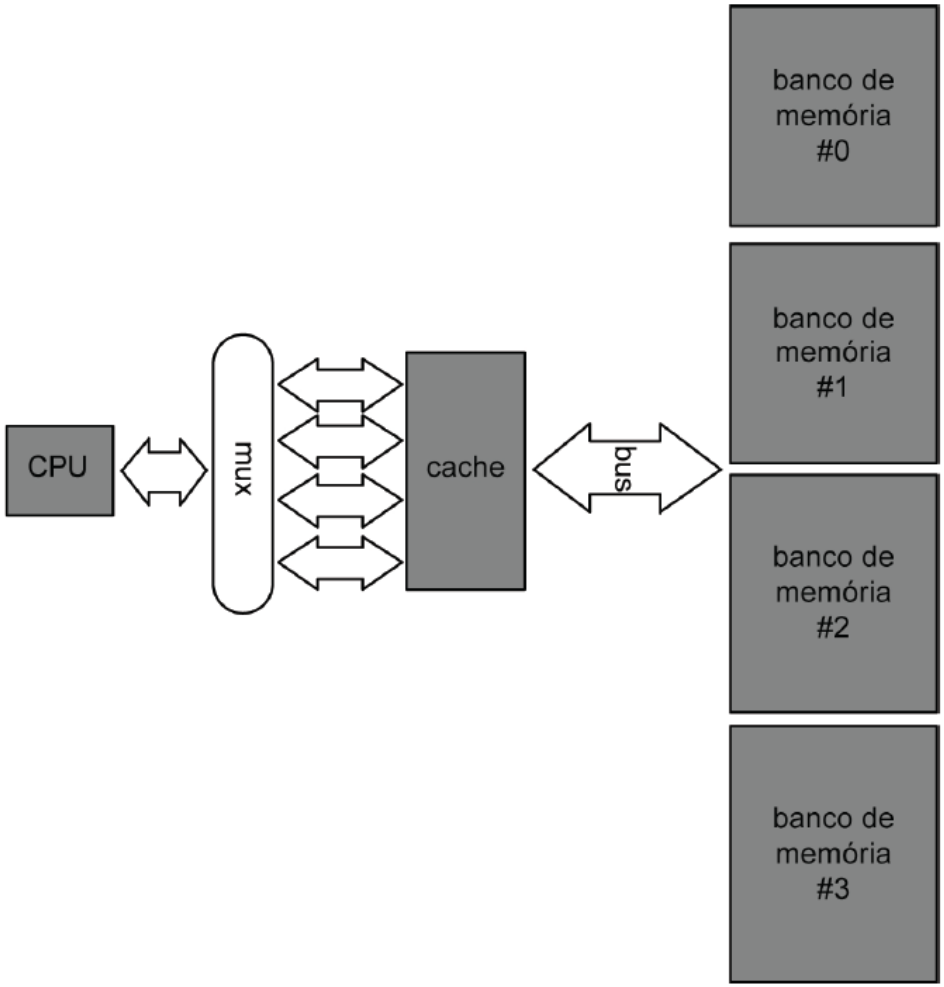
- Ex: sistema de interconexão de 4 palavras. Neste caso a transferência para a *cache* pode ser feita em blocos de 4 palavras. Exploração das localidades de referência espacial e temporal.



Organização de memória entrelaçada (*interleaving*)

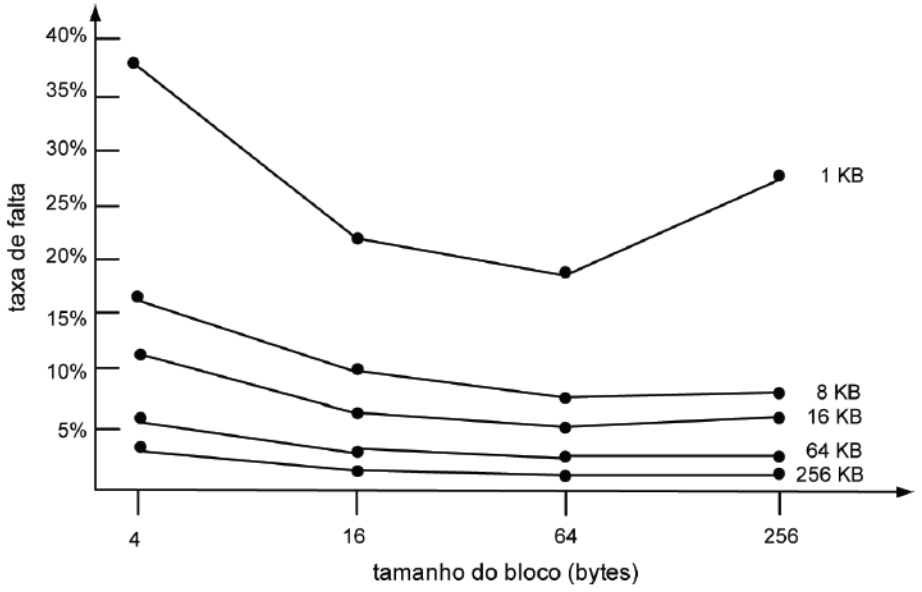
- No próximo exemplo, cada memória do banco, contém uma das palavras do bloco. O bloco é lido simultaneamente.
- Após a leitura, as palavras são transferidas sequencialmente para a *cache*. A latência total é relativamente pequena, pois sua maior parte corresponde ao tempo de acesso à memória (que no caso é feito em paralelo), e não no tempo de transferência.

Exemplo


CPU
mux
cache
bus
banco de memória #0
banco de memória #1
banco de memória #2
banco de memória #3

Desempenho da *cache* em mapemanto direto

- A figura a seguir mostra um diagrama de taxa de falta na *cache*, em função dos tamanhos do bloco e da *cache*.



Interpretações

- Tamanho de 4 *bytes* por bloco -> taxa de falta maior que para outras condições. Há apenas uma palavra de 32 *bits* no bloco, logo, não ocorre a exploração da localidade de referência espacial para um computador de 32 *bits*.

Interpretações

- Para a *cache* de 1 *kbyte*, mesmo a localidade de referência temporal não pode ser explorada, pois a *cache* tem possibilidade de carregar apenas 256 palavras, e os *slots* devem ser compartilhados.

Interpretações

- A taxa de falta diminui à medida que o tamanho do bloco aumenta, até 64 *bytes*. Na *cache* de 1 *kbyte* há apenas 4 *slots* de 64 palavras, logo, há pouca exploração da localidade temporal, do que decorre o aumento da taxa de falta.

Interpretações

- A tabela seguinte mostra o resultado da taxa de falta nas *caches* quanto estes são divididos em: *cache* de instruções e *cache* de dados. Nota-se que, quando o tamanho do bloco é de 4 palavras, a taxa de falta diminui drasticamente para a *cache* de instruções. Isso significa que as instruções se enquadram na localidade de referência espacial, pelo fato de serem sequenciais.

Tabela: taxa de falta em *benchmark* SPEC

Programa	Tamanho do bloco em palavras	Taxa de falta em instruções	Taxa de falta em dados	Taxa de falta combinada
Gcc	1	6,1%	2,1%	5,4%
	4	2,0%	1,7%	1,9%
Spice	1	1,2%	1,3%	1,2%
	4	0,3%	0,6%	0,4%

tempo de execução = (*ciclos de execução* + *ciclos de parada*) x *tempo de ciclo*

ciclos de parada = *número de instruções* x *taxa de falta* x *penalidade*.

Exercício

- Suponha que uma taxa de falhas de instruções para um programa seja de 2% e que uma taxa de falhas de *cache* de dados seja de 4%. Se um processador possui um CPI de 2 sem qualquer *stall* de memória e a penalidade de falha é de 100 ciclos para todas as falhas, determine o número total de ciclos de *stall* da memória e o CPI com *stalls*.
  - Dado: a frequência de todos os *loads* e *stores* no SPECint2000 é de 36%.

Solução

- O número de ciclos de falha da memória para instruções em termos de contagem de instruções (I) é:
  - Ciclos de falha de instrução = I x 2% x 100 = 2 x I.
  - Ciclos de falha de dados = I x 36% x 4% x 100 = 1,44 x I.
  - Portanto, o número total de ciclos de *stall* da memória será: 3,44 I. O CPI com *stalls* da memória será: 2 + 3,44 = 5,44.