Arquitetura de Computadores

MIEI – 2018/19

DI-FCT/UNL

Aula 20

Princípio da Localidade

- Os programas têm padrões de acesso a memória:
  - **Localidade temporal**: items referenciados recentemente tendem a voltar a sê-lo.
  - **Localidade espacial**: items com endereços próximos tendem a ser referenciados em conjunto.

Exemplo:

Dados

Elementos do array são referenciados em sequência: **Localidade espacial**

`sum` referenciado em cada iteração:

Instruções **Localidade temporal**

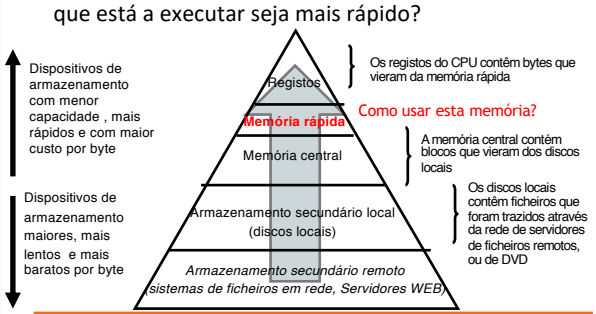
Instruções referenciadas em sequência: **Localidade espacial**

Ciclo : **Localidade temporal**

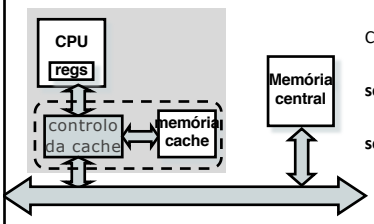
```
sum = 0;
for (i = 0; i < n; i++)
   sum += a[i];
return sum;
```


Hierarquia de níveis de memória

- Poderemos ter mais um nível para que o programa que está a executar seja mais rápido?



Cache de memória



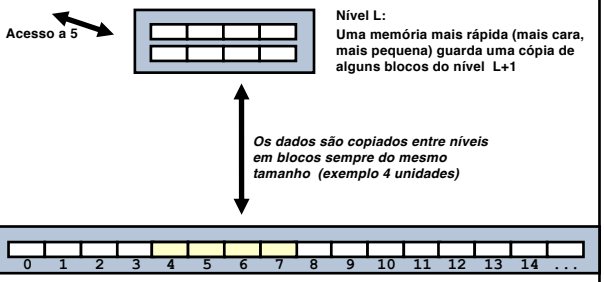
- Funcionamento:

CPU referencia memória:

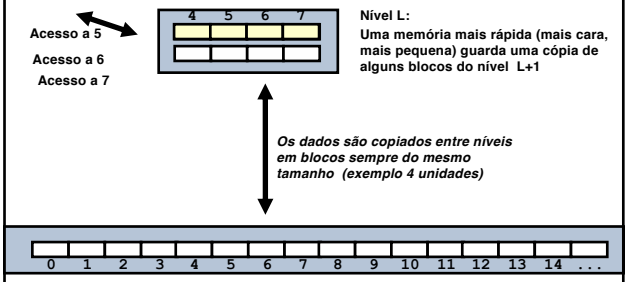
se *está em cache*
  acede à cache
senão
  acede à memória
  e atualiza a cache

A localidade espacial leva a trazer logo um bloco da memória

Caching na hierarquia de memória



Caching na hierarquia de memória



Conceitos gerais de funcionamento

- Para cada acesso à memória de determinado bloco:
- **Cache hit**
  - Este está na cache → tempo de acesso rápido (nível L)
  - Exemplo anterior: acesso aos endereços 6 e 7
- **Cache miss**
  - Este não está na cache. Deve obtê-lo no nível inferior → tempo de acesso lento (nível L+1)
  - Exemplo anterior: primeiro acesso, ao endereço 5
  - Se a cache está cheia, então é preciso arranjar espaço:
    - **Qual o bloco a eliminar?**

Tratamento do miss

- Cache miss
  - Não enconta o que pretende no nível L, então vamos ao nível L+1 (e assim sucessivamente)
  - Guardar uma cópia do bloco no nível L
  - Mas se o nível L está cheio, então é preciso arranjar espaço:
    - Que bloco escolher como vítima? Ao acaso? O menos usado? …?
    - Se o bloco vítima não foi alterado, carregar o novo bloco por cima
    - Se o bloco vítima está alterado (foi escrito), é preciso garantir que no nível L+1 também está alterado antes de o substituir

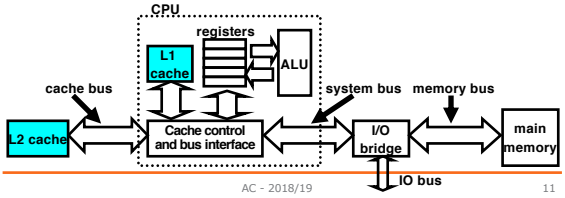
Tempo médio de acesso

- Taxa de sucesso (*hit ratio*)
  - *h* = percentagem de vezes que o dado pretendido está na cache:
    h = núm. cache hit / núm. total acessos
- Tempo médio de acesso:
  - T_L é o tempo de acesso no nível L
  - T_{L+1} é o tempo de acesso no nível L+1
  - Tempo médio de acesso, Ta
    $Ta = h * T_L + (1 - h) * T_{L+1}$

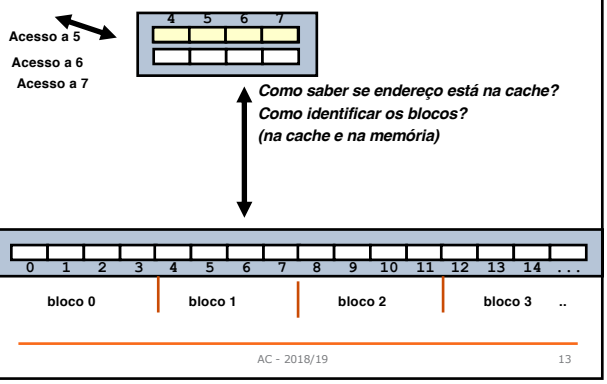
- Exemplo:
  - *T cache* = 2 ns
  - *T mem* = 8 ns
  - *Hit ratio* = 80% numa determinada execução
- $Ta = h * T_L + (1 - h) * T_{L+1}$
  - Ta = 0,8 x 2 + 0,2 x 8 = **3,2 ns**
- Nesta execução a percepção do tempo de acesso à memória é de 3,2 ns

Cache – Exemplo com dois níveis

- Cache são memórias rápidas (SRAM) geridas automaticamente pelo *hardware*.
- CPU procura em L1, depois em L2, finalmente na memória central
  - Podem introduzir algum *overhead,* mas desprezável
- Exemplo de arquitetura com L2 externa ao chip do CPU:

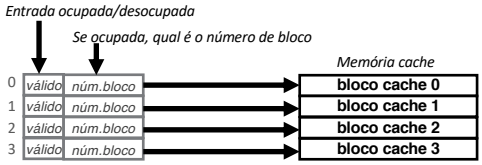


Caching na hierarquia de memória



Gestão da Cache

- É necessário de forma eficiente:
  - Saber que blocos da cache estão ocupados e com que blocos de memória
  - Exemplo de cache com capacidade para 4 blocos de memória:

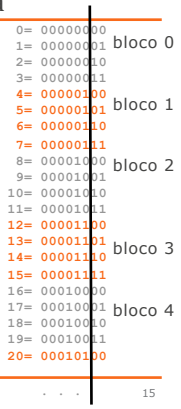


Endereços e blocos de memória

Exemplo:
endereços de 8 bits
blocos de 4bytes

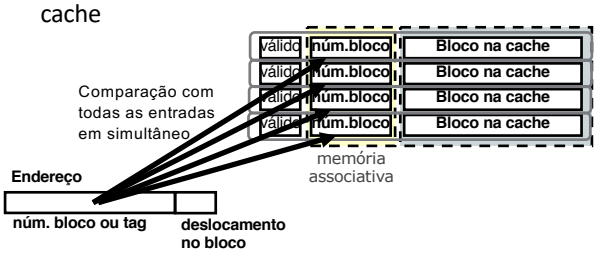
dado um endereço *E*:
E/4 = número do bloco
E%4 = byte dentro do bloco
ou seja:
4 bytes por bloco = 2^2 → 2 bits

os 2 bits menos significativos indicam bytes dentro do mesmo bloco
os restantes correspondem ao número do bloco



Cache associativa pura

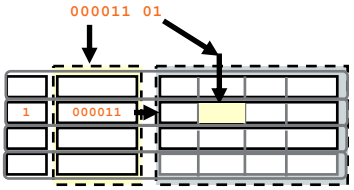
- Usa o número do bloco como **chave** (ou *tag*) na memória associativa para procurar o bloco na cache

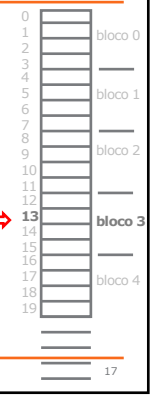


Endereços e a cache

Exemplo: cache com 16bytes, 4 linhas de 4 bytes
Ler endereço 13

Bloco memória = 13/4 = 3
Byte nesse bloco = 13%4 = 1



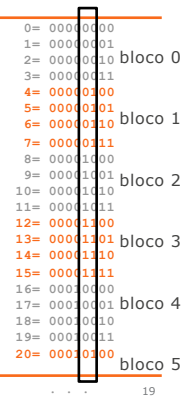


Caches associativas puras

- As Caches Associativas são muito eficientes:
  - qualquer bloco de memória pode estar em qualquer linha da cache
  - a pesquisa do bloco faz-se em paralelo, com tantos comparadores quantos os blocos que cabem na cache
- Mas são complexas e muito caras:
  - cada linha da cache deve guardar todos os bits do número do bloco nessa posição
  - cada comparador é de tantos bits quantos os bits no número do bloco
  - muitos comparadores, um para cada linha da cache

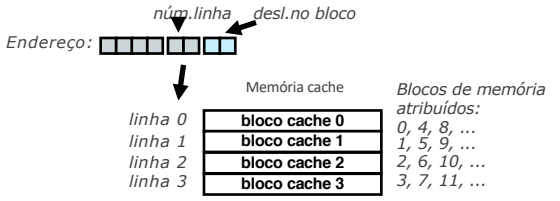
Simplificando

- Pré-definindo uma linha da cache para cada bloco de memória:
  - Linha = nºbloco % nº de linhas da cache
  - Exemplo: 4 linhas --> 2bits



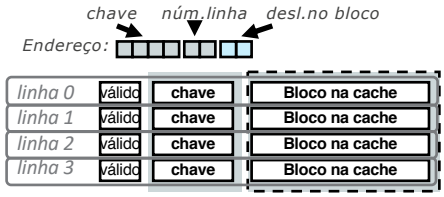
Simplificando

- Pré-definindo uma linha da cache para cada bloco de memória:
  - Linha = nºbloco % nº de linhas da cache
  - Exemplo: 4 linhas --> 2bits



Cache de mapa direto

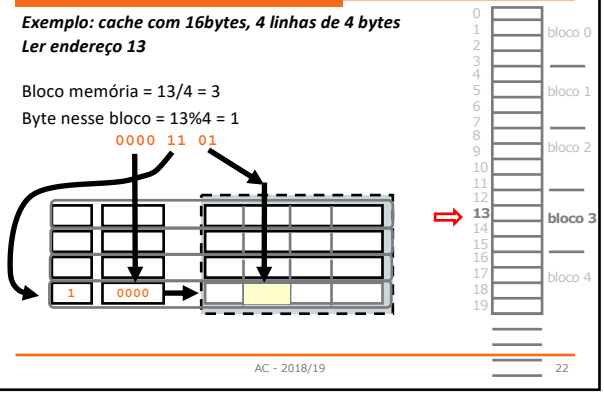
- É necessário manter para cada linha uma chave (ou *tag*) que identifique qual o bloco na cache de entre os vários possíveis:
  - o que distingue os vários blocos são os bits mais significativos restantes do endereço
- Exemplo: end. de 8bits, cache de 4 linhas e 4 bytes p/bloco:



Endereços e a cache

Exemplo: cache com 16bytes, 4 linhas de 4 bytes
Ler endereço 13

Bloco memória = 13/4 = 3
Byte nesse bloco = 13%4 = 1



Características da Cache de mapa direto

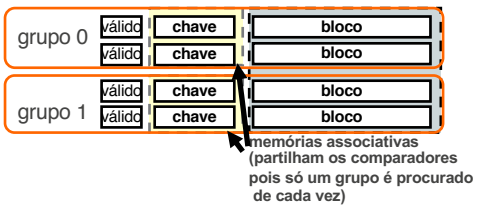
- As Caches de mapa direto são mais baratas do que as Associativas:
  - O mapa da Cache tem menos bits;
  - Não necessita de memória associativa e só precisa de 1 comparador (de tantos bits quantos os na chave dos blocos)
- São eficientes na pesquisa em Cache:
  - usam bits do endereço como índice na Cache e restantes como chave.
- Levam a colisões, mesmo que existam linhas livres!

Cache associativa por grupos (*set associative*)

- Solução híbrida de compromisso
- Procura-se responder às desvantagens das duas técnicas anteriores:
  - evitar as colisões no mapa direto
  - ter menos comparadores que a associativa pura
- Abordagem:
  - cada "linha" de mapa direto dá lugar a um grupo (conjunto) de blocos
  - cada bloco pode ficar em qualquer linha dentro do grupo
  - a busca no grupo é efectuada procurando a chave do bloco usando memória associativa

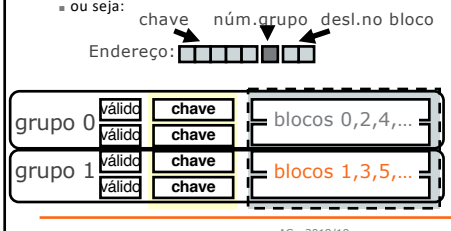
Cache associativa por grupos

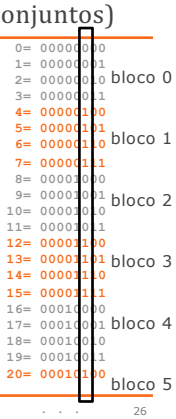
- Os blocos de memória são mapeados diretamente num grupo (ou conjunto de linhas)
  - o núm. do grupo é dado pelo: ***núm.bloco%núm.grupos***
- Dentro do grupo pode ficar em qualquer posição, como se o grupo fosse uma "mini cache associativa pura"



Cache associativa por grupos (ou conjuntos)

- **Exemplo, end. de 8 bits:**
  - **cache com 2 grupos**
  - **4 blocos de 4 bytes, 2 por grupo**
  - núm. do grupo é dado por: **núm.bloco%núm.grupos** (como no mapa direto para obter a linha)
  - ou seja:

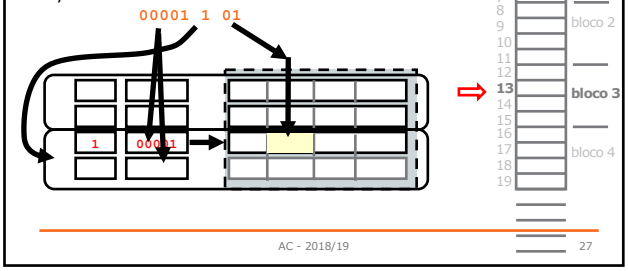




Endereços e a cache

Exemplo: cache com 16bytes, 4 linhas de 4 bytes
Ler endereço 13

Bloco memória = 13/4 = 3
Byte nesse bloco = 13%4 = 1



Quando falha a cache (miss)

- Existe misses quando as localidades não se verificam:
  - por acesso a dados não contíguos em memória
    - ou estruturas que não cabem na cache
  - muda o conjunto de instruções em execução (*working-set*): devido a jumps, calls,…
    - ou este conjunto não cabe todo na cache
  - o SO troca de programa em execução
    - muda de código e de dados…

Tratamento dos misses

- Se ***Cache miss***: é preciso arranjar espaço para o novo bloco. Se associativa por grupos o conflito tem de ser resolvido apenas no grupo.
- Se linha disponível (na cache ou no grupo), marca como válida, atualiza a chave (tag) e o conteúdo do bloco
- Se cache cheia, qual o bloco a ser eliminado da cache ?
  - Um qualquer . . . ou o bloco que no futuro não vai ser necessário. Mas qual é esse?
- Se usando *write-back*, verifica bit *dirty* e atualiza memória, antes de usar a linha vítima