Arquitetura de Computadores

MiEI – 2016/17

DI-FCT/UNL

Aula 18

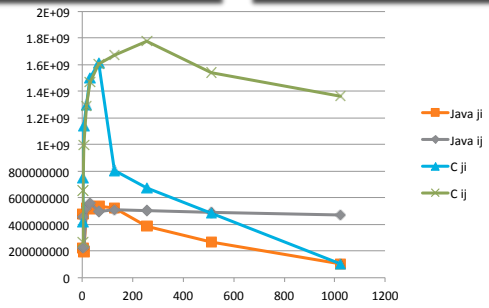
Ideia da Cache

- Vantagens esperadas:
  - A localidade leva a que a maior parte dos acessos (leitura/escrita) sejam aos dados já no nível L em vez de ter de ao nível L+1.
  - O armazenamento no nível L+1 pode ser mais lento, e assim mais barato e muito maior.
  - Efeito global: Um grande conjunto de memória que custa o preço da que se encontra no nível mais baixo, mas que permite o acesso aos dados a um ritmo semelhante à do nível mais elevado.

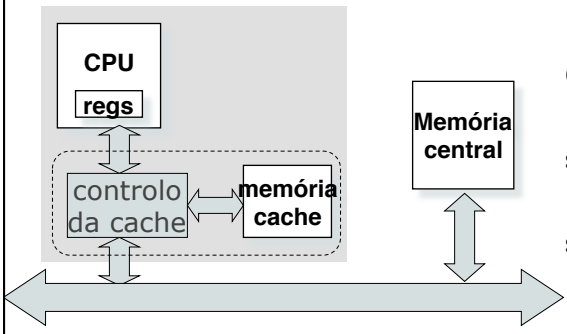
Desempenho (somas/s)

```
/* ij */
sum = 0.0;
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum += a[i][j];
  }
}
```

```
/* ji */
sum = 0.0;
for (j=0; j<n; j++) {
  for (i=0; i<n; i++) {
    sum += a[i][j];
  }
}
```



Organização da cache de memória

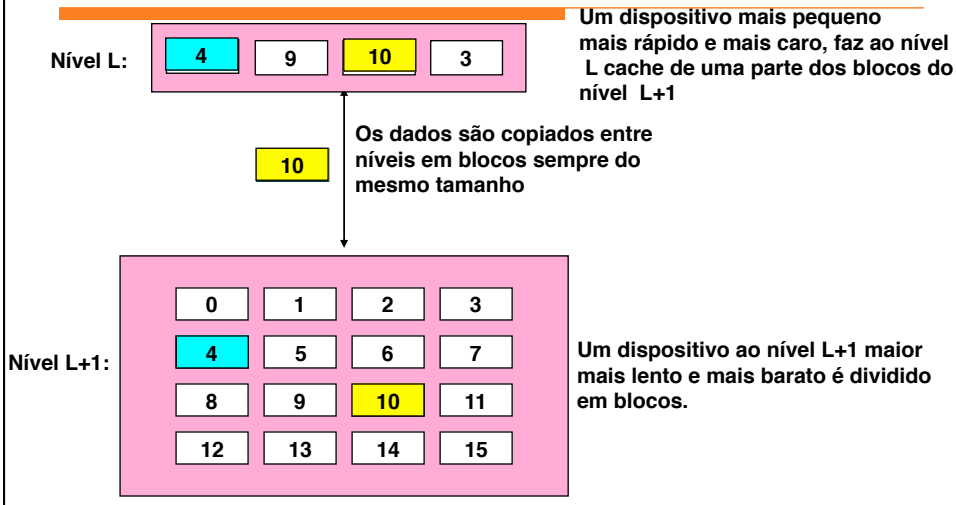


- Funcionamento:

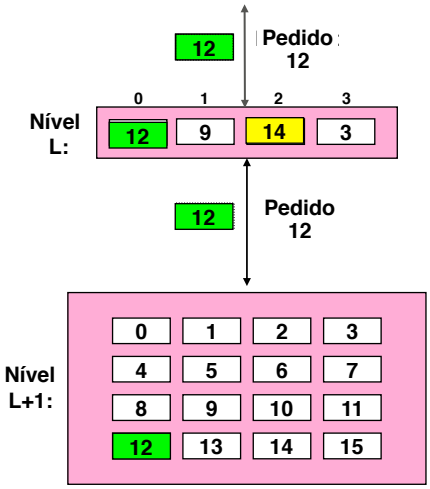
CPU referencia memória:

se *está em cache*
  acede à cache
senão
  acede à memória
  e actualiza cache

Caching na hierarquia de memória



Conceitos gerais da cache



- O programa requer um dado b.
- Cache hit
  - O programa encontra b na cache a nível L. E.g., bloco 14.
- Cache miss
  - b não está no nível L, então a cache de nível L deve obtê-lo no nível L+1. E.g., bloco 12.
  - Se o nível L está cheio, então é preciso escolher uma vítima:
    - Qual o bloco a ser escolhido como vítima?

Conceitos gerais da cache: hit/miss

- ***Cache hit***
  - O programa encontra o dado pretendido na cache (nível L). Não é preciso fazer mais nada!
- ***Cache miss***
  - Não enconta o que pretende no nível L, então a cache de nível L deve obtê-lo no nível L+1 (e assim sucessivamente).
  - Se o nível L está cheio, então é preciso arranjar espaço, **escolhendo uma *vítima***:
    - Qual o bloco a ser escolhido como vítima (ao acaso? LRU - *least recently used* ?)
    - Se o bloco vítima está *limpo*, isto é, **não foi alterado** desde que veio para o nível L, posso carregar o novo bloco por cima
    - Se o bloco vítima está *sujo*, isto é, **foi alterado**, é preciso escrevê-lo no nível L+1 antes de o substituir

Conceitos gerais duma cache

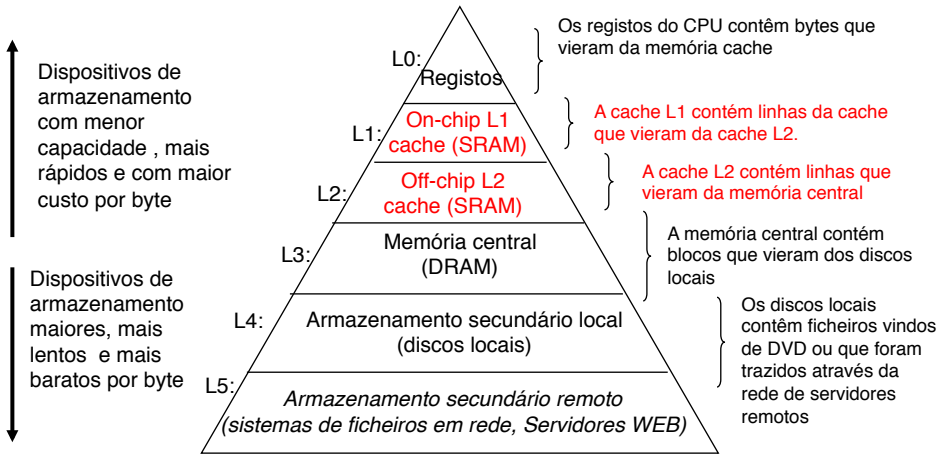
- Taxa de sucesso (*hit ratio*)
  - *h* = percentagem de vezes que o dado pretendido está na cache:

    h = núm. cache hit / núm. total acessos
- Tempo de acesso médio:
  - T_L é o tempo de acesso no nível L
  - T_{L+1} é o tempo de acesso no nível L+1
  - Tempo de acesso médio: Ta

    $Ta = h * T_L + (1 - h) * T_{L+1}$

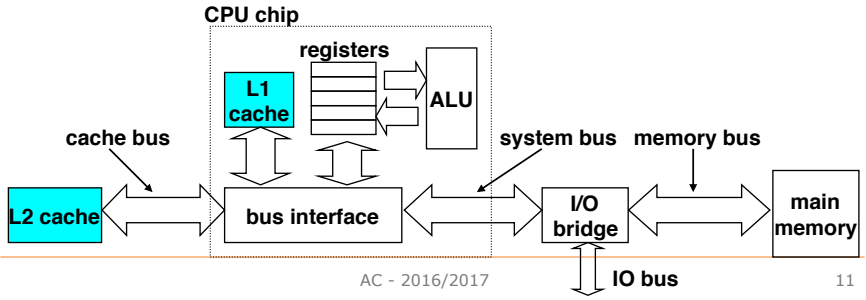
- Exemplo:
  - *T cache* = 2 ns
  - *T mem* = 8 ns
  - *Hit ratio* = 80%
- Ta = $h * T_L + (1 - h) * T_{L+1}$
  - Ta = 0,8 x 2 + 0,2 x 8 = **3,2 ns**

Hierarquia de níveis de memória



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:



Política de escritas com cache

- Quando o CPU altera o conteúdo duma posição de memória, faz sentido que a alteração fique em cache
  - a ideia é que o CPU pode vir ler esse valor em breve (localidade temporal)
- O que acontece na memória de nível inferior, quando um bloco na cache é alterado, depende da **política de escrita**:
  - *Write-through*
  - *Write-back (*ou *delayed)*

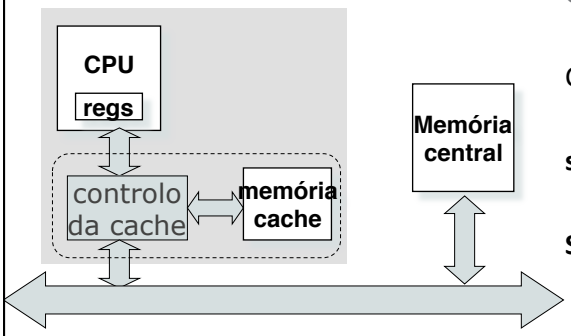
Write-through

- A memória cache e a memória central são actualizadas em cada escrita
- Assegura que a cache e a memória central estão sempre **coerentes**
- Cada escrita demora o tempo de escrita na memória central
- Tolerável, porque normalmente há muito mais acessos em leitura do que em escrita

Write-back

- Quando há uma escrita só é atualizada a cache; mas marca-se esse bloco como alterado (*dirty*)
- Só quando um bloco é removido da cache é que se atualiza o bloco na memória de nível inferior
- Isto é mais rápido do que o *write-through*
- Diminui o tráfego para a memória de nível inferior
- O inconveniente é que a cache e a memória inferior nem sempre estão coerentes
  - isto pode provocar problemas …

Organização da cache de memória

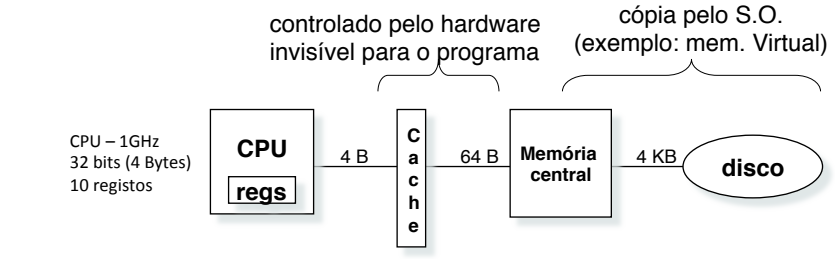


- Funcionamento:

CPU referencia memória:

se *está em cache (hit)*
acede à cache
Senão *(miss)*
acede à memória
e atualiza cache
com o novo *bloco*

Exemplos de blocos usados

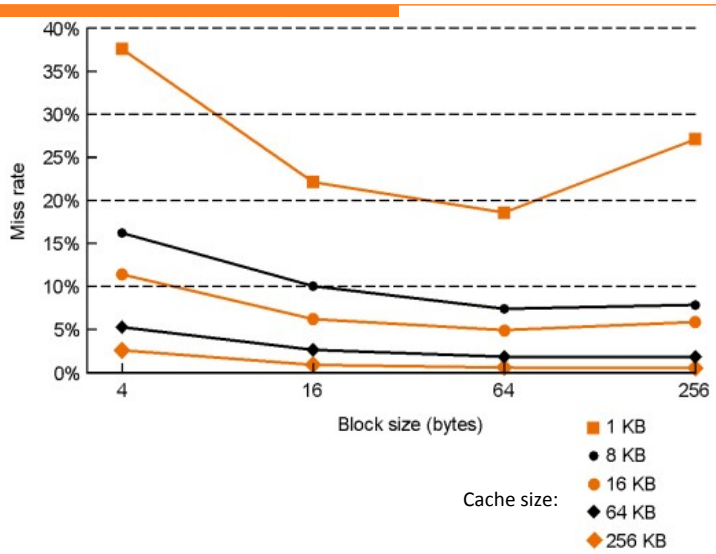


	Registos	Cache	Memória	Disco
tamanho:	40 B	32 KB a 4 MB	1 GB	100 GB
latência:	1 ns	5 ns	50 ns	7 ms
$/Mbyte:	-	$100/MB	$1.25/MB	$0.008/MB
tamanho do bloco:	**4 B**	**64 B**	**4 KB**	

memória mais lenta, mais barata e maior

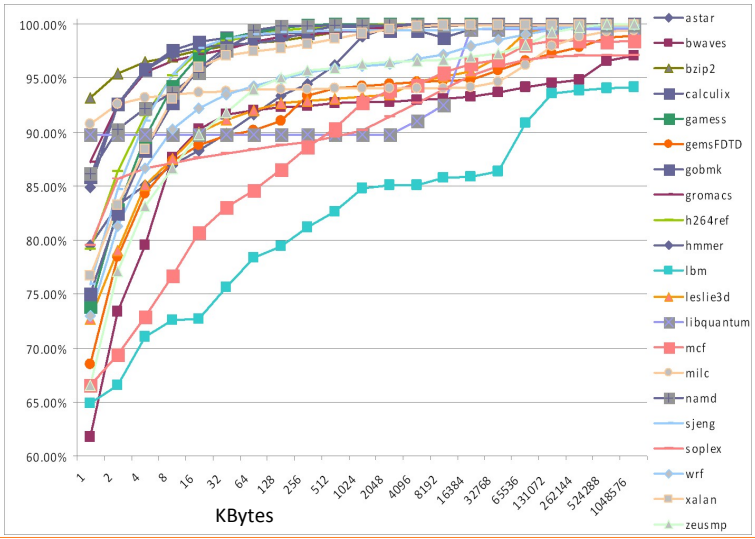
Impacto do tamanho do bloco no *hit-rate*

EXEMPLO para um programa:



Impacto do tamanho da cache no *hit-rate*

EXEMPLOS de vários programas



Organização da cache

- Para além de guardar o conteúdo da memória de nível inferior, também necessita de guardar:
  - Que zonas da cache estão realmente em uso (de inicio a cache está vazia!)
  - Que endereços/blocos da memória de nível inferior estão na cache
  - Se usar *write-back*, que zonas foram escritas
  - Informação que permita escolher os blocos a substituir
- A consulta e atualização desta informação **tem de ser eficiente**!!!

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

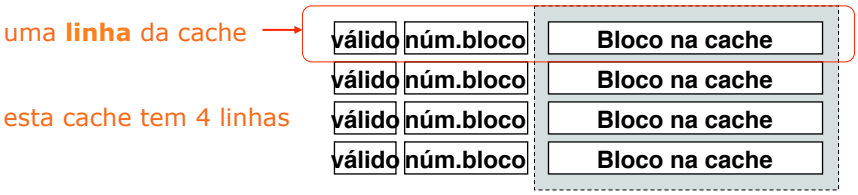
os 2 bits menos significativos indicam bytes dentro do mesmo bloco
os restantes correspondem ao número do bloco

núm. bloco *deslocamento no bloco*

```
 0= 00000000
 1= 00000001   bloco 0
 2= 00000010
 3= 00000011
 4= 00000100
 5= 00000101   bloco 1
 6= 00000110
 7= 00000111
 8= 00001000
 9= 00001001   bloco 2
10= 00001010
11= 00001011
12= 00001100
13= 00001101   bloco 3
14= 00001110
15= 00001111
16= 00010000
17= 00010001   bloco 4
18= 00010010
19= 00010011
20= 00010100
```

Exemplo (*write-through*)

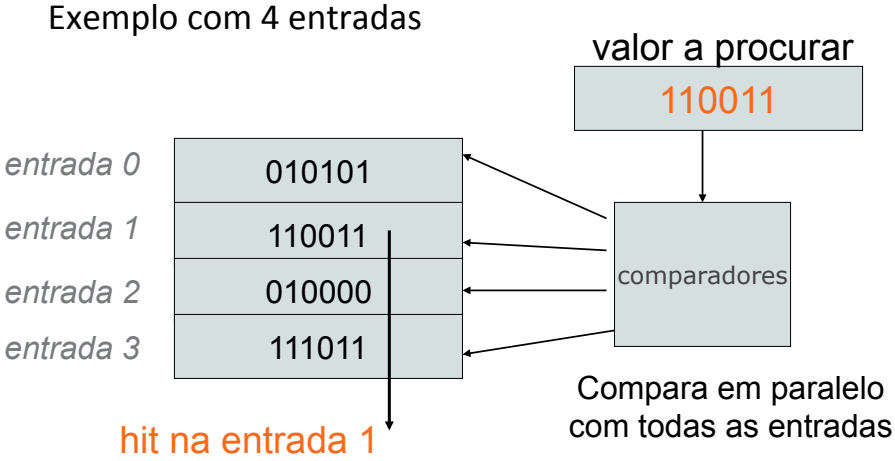
- A cache tem para cada bloco, se está ocupado (válido) ou não e o respectivo número de bloco, quando ocupado



- Para cada acesso do CPU à memória, parte-se o endereço de acordo com o tamanho do bloco e procura-se o seu número na cache
  - só faz sentido se esta procura for rápida!
- Usa-se os bits de deslocamento no bloco para aceder ao byte pretendido

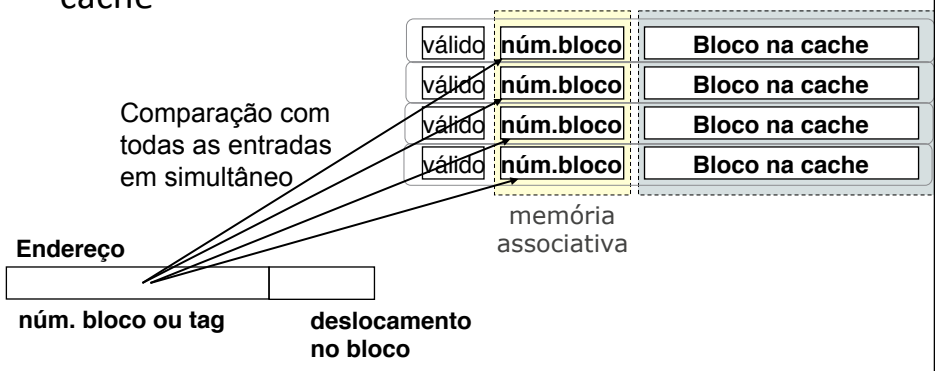
Memória associativa (interrogável pelo conteúdo)

Exemplo com 4 entradas



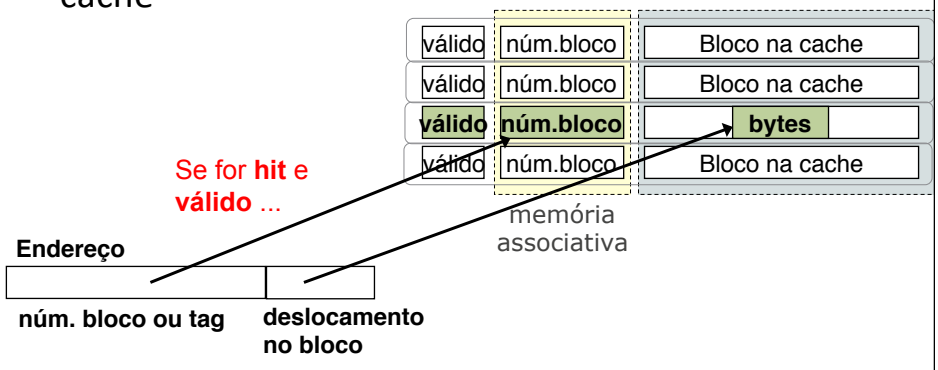
Cache associativa pura

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



Cache associativa pura

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

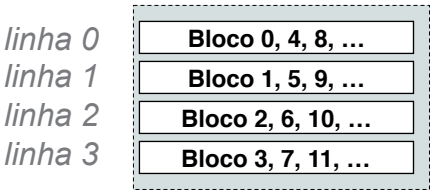


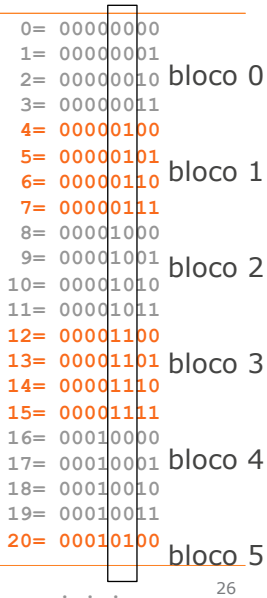
Caches associativas puras

- As Caches Associativas são muito eficientes:
  - qualquer bloco pode estar em qualquer posição
  - 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

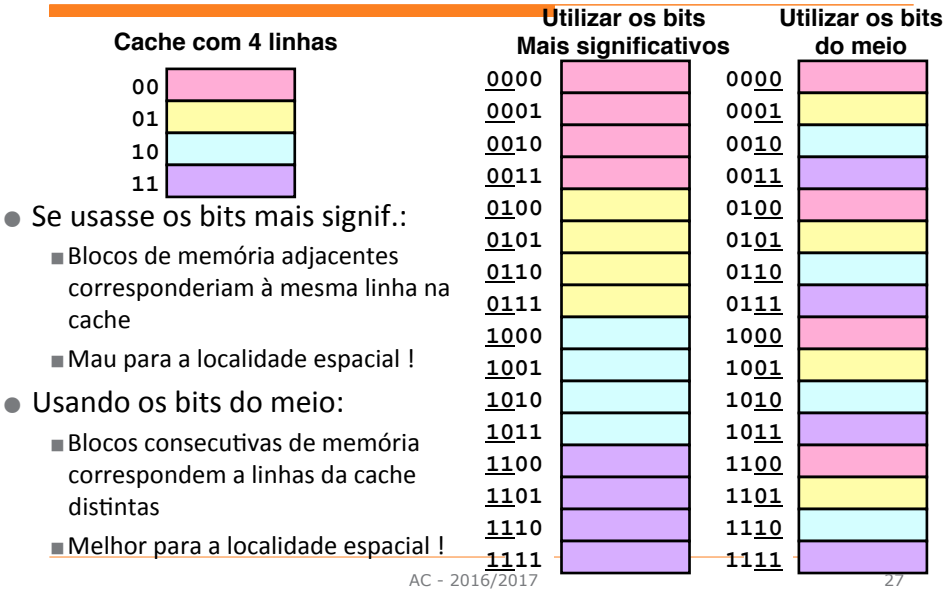
Tentando não usar memória associativa

- Fixamos a posição na cache para cada bloco
- Exemplo:
  - **cache com 4 linhas de blocos de 4 bytes**
  - o núm. da linha é dado pelo *núm.bloco%núm.linhas*
  - ou seja, por 2 bits do endereço:





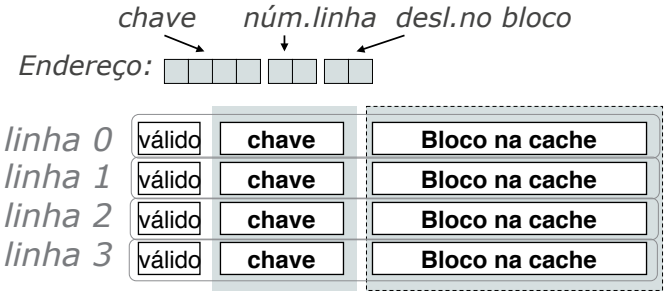
Porque se usam os bits do meio para escolher a linha?



- Se usasse os bits mais signif.:
  - Blocos de memória adjacentes corresponderiam à mesma linha na cache
  - Mau para a localidade espacial !
- Usando os bits do meio:
  - Blocos consecutivas de memória correspondem a linhas da cache distintas
  - Melhor para a localidade espacial !

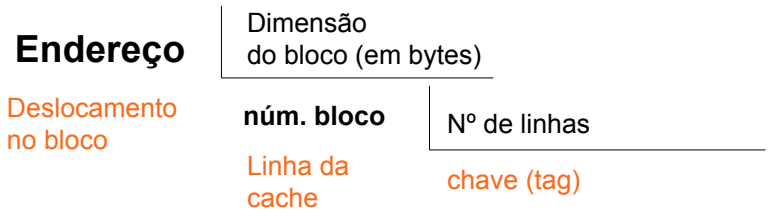
Cache de mapa directo

- É necessário manter para cada linha uma chave (ou *tag*) que identifique qual o bloco realmente 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:

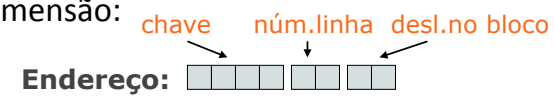


Tratamento do endereço

- Resumo (divisões inteiras):



  - em binário, estas operações correspondem a separar os bits do endereço em 3 partes, de acordo com cada dimensão:



Vantagens da Cache de mapa directo

- 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:
  - porque usam os bits do endereço diretamente como índice no mapa da Cache e como chave, só precisando de comparar essa chave.

Desvantagems da Cache de mapa directo

- Levam a colisões: todos os blocos com a mesma linha atribuída colidem nessa posição
  - mesmo que existam outras linhas na cache livres!
- A probabilidade de colisões diminui à medida que se aumenta a Capacidade da Cache
  - mas o problema mantém-se …