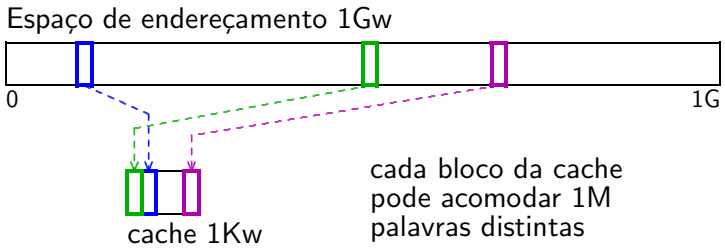
Projeto de Caches

- Mapeamento de endereços (hashing)
- capacidade [bytes]
- tamanho de bloco [palavras]
- associatividade (mais hashing)
- três tipos de faltas
- tempo médio de acesso à memória

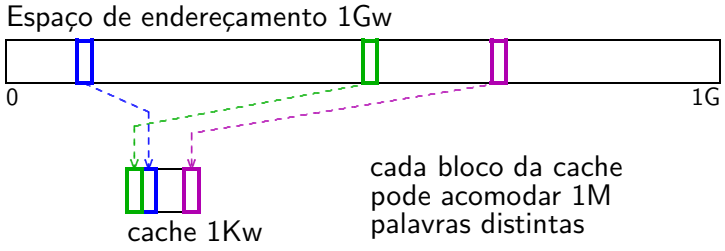
Projeto de memórias cache

Problema #1:
mapear espaço de endereçamento de 4Gbytes em 4Kbytes
necessita função que mapeia 1Gwords em 1Kwords hashing
como implementa função em hardware, eficientemente?

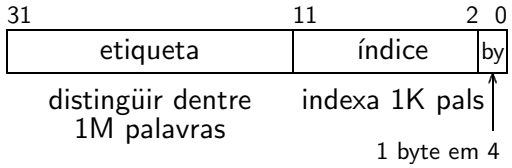


Projeto de memórias cache

Problema #1: mapear espaço de 4Gbytes em 4Kbytes

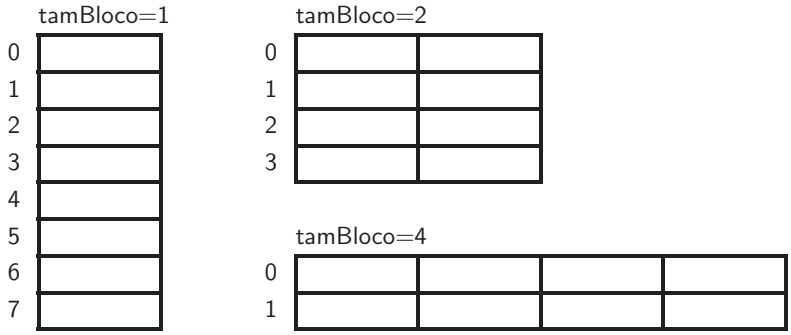


Mapeamento direto:
posição na cache
determinada por
10 bits do endereço
`pos = end_WD % 1024`



Projeto de memórias cache

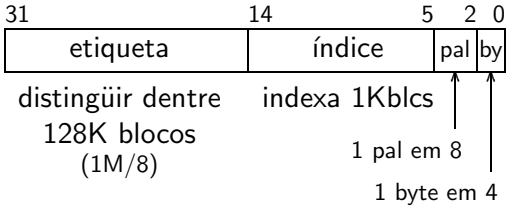
Problema #2: tirar vantagem da localidade espacial
necessita acomodar mais de uma palavra em cada bloco da cache



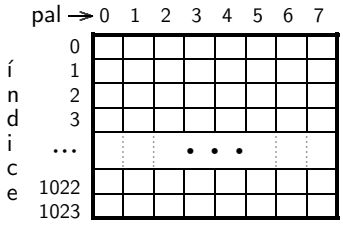
preencha as 3 caches com refs a palavras:
0 1 3 4 5 8 9 12 11 15 19 4 2 16 13 2 2 19 18 3

Projeto de memórias cache

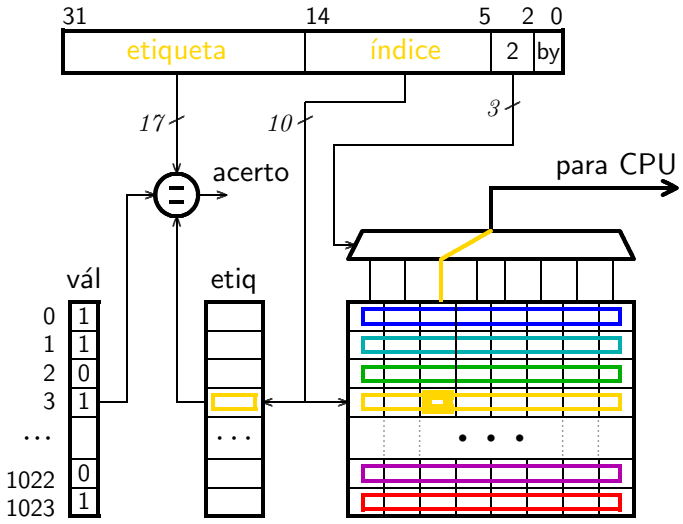
Problema #2: tirar vantagem da localidade espacial



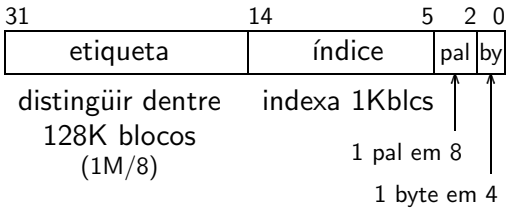
Mapeamento direto:
posição na cache
determinada por
10 bits do endereço
`pos = ⌊end_WD/tamBlc⌋ % númBlc`
bloco acomoda 8 palavras de mesmo etiq-índice



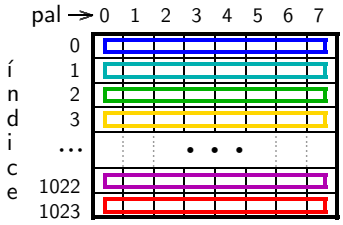
Cache com Mapeamento Direto



Mapeamento Direto e conflitos

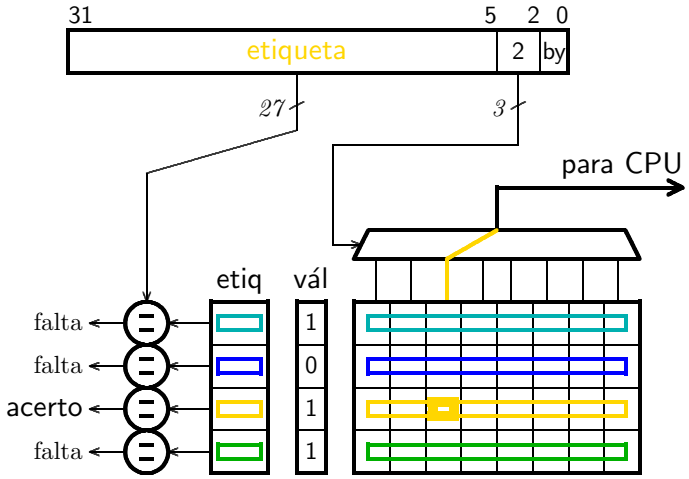


Mapeamento direto:
cada bloco da cache
pode armazenar palavras
de uma "cor" mas
memória contém 128K blocos
de cada cor (nesta cache 1K,8pal/bl)



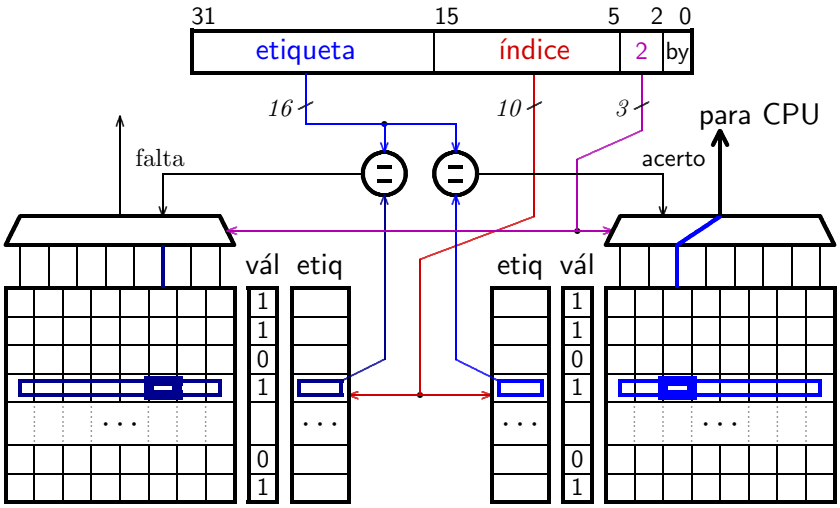
O que acontece se dois blocos da mesma cor são acessados em seqüência? índice igual, etiquetas distintas

Mapeamento Associativo



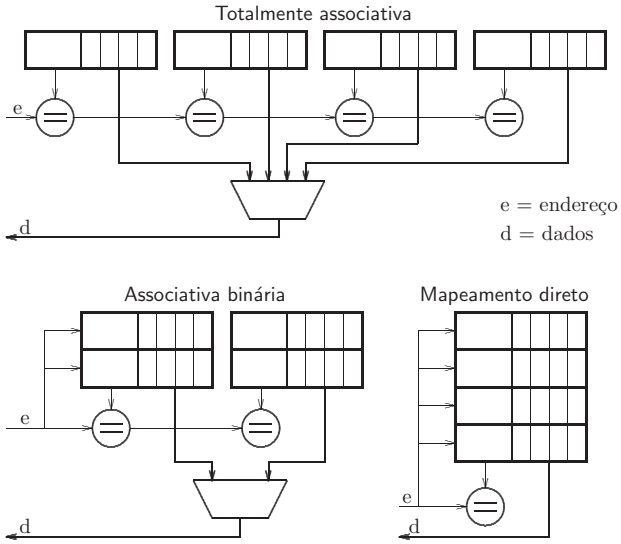
Mapeamento ideal e caro: um comparador em cada bloco da cache

Mapeamento Híbrido: Associatividade por Conjuntos

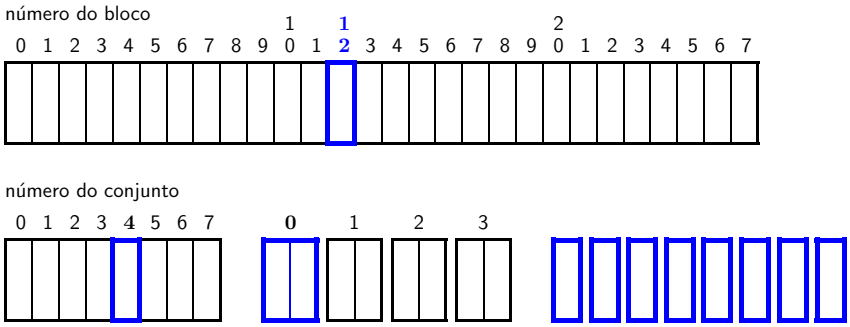


bloco de índice n pode estar em $\forall$ elmto do conjunto N $|N|=2$

Três Mapeamentos



Alocação nos Blocos



mapeamento direto	associativa por conjuntos (binária)	associativa (total)
bloco 12 mapeia somente no bloco 4 12 % 8 = 4	bloco 12 mapeia no conjunto 0 12 % 4 = 0	bloco 12 mapeia em qualquer conjunto

Alocação nos Blocos

`M[12]:= 99; M[13]:= 200; M[14]:= 33`

Associativa - 4 conjuntos

0		99		200		33		

`alocação: 12%1 = ?`

Associativa binária - 2 conjuntos

0		99		33
1		200		

`12%2 = 0`
`14%2 = 0`
`13%2 = 1`

Mapeamento direto - 1 conjunto

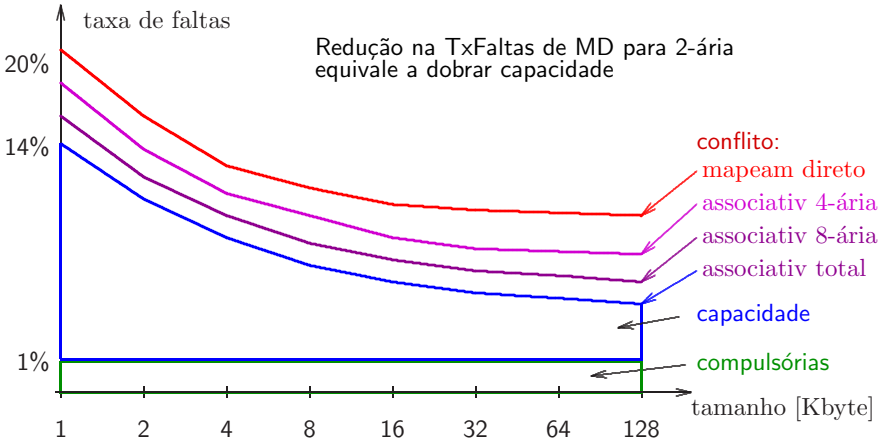
0		99
1		200
2		33
3		

`12%4 = 0`
`13%4 = 1`
`14%4 = 2`
`xx%4 = 3`

Como ficam as etiquetas, em cada um dos 3 casos?

Tipos de Faltas – falta por conflito

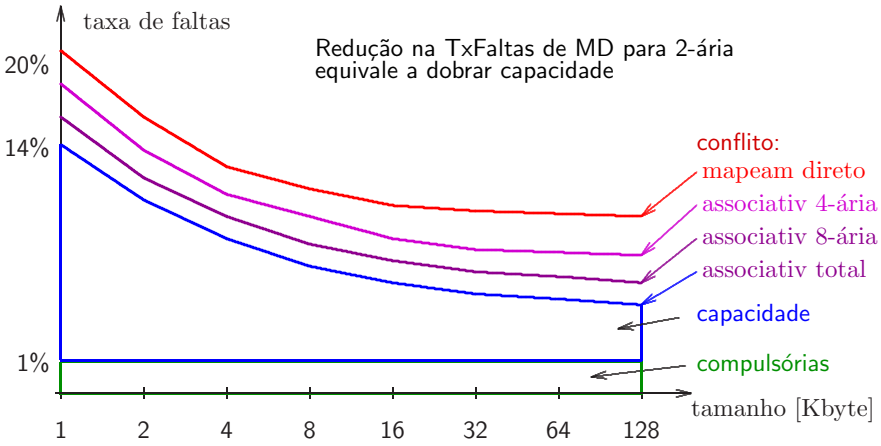
problema: N blocos da mesmo índice (côr) em uso simultâneo, mas cache só acomoda $M < N$ blocos



solução: aumentar associatividade

Tipos de Faltas – falta por capacidade

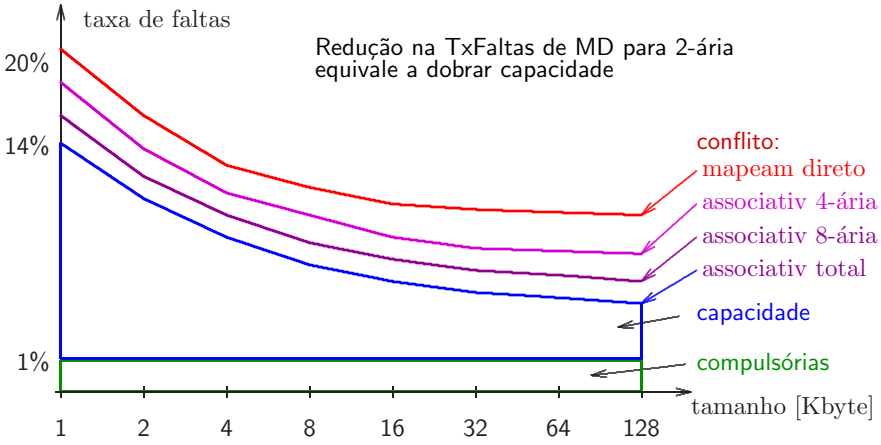
problema: cache não tem capacidade para acomodar todos blocos acessados por programa



solução: aumentar tamanho da cache

Tipos de Faltas – falta compulsória

problema: primeiro acesso a um bloco sempre causa uma falta (que é inevitável)



solução: busca antecipada por hardware ou por software

Tipos de Faltas – 3 Cs

- Que tipos de faltas ocorrem numa cache totalmente associativa de tamanho infinito?
  faltas compulsórias porque deve trazer cada bloco para a cache
- Além destes, que tipos de faltas ocorrem numa cache totalmente associativa, de tamanho finito?
  faltas por capacidade porque programa pode usar mais blocos do que a cache comporta
- Além destes, que tipos de faltas ocorrem numa cache associativa por conjunto ou com mapeamento direto?
  faltas por conflito porque dois endereços $\neq$s mapeiam no mesmo bloco da cache (mesmo índice)

Política de substituição de blocos

Em caches associativas, depois de uma falta de leitura, se não há blocos vazios, qual bloco deve ser expurgado da cache?

bloco usado menos recentemente?
Parece ser o melhor, mas pode ser difícil de implementar

escolher aleatoriamente?
É fácil de implementar, mas como é o desempenho?

TxFaltas Cache Assoc Binária

capac	LRU	aleatória
16KB	5.2%	5.7%
64KB	1.9%	2.0%
256KB	1.15%	1.17%

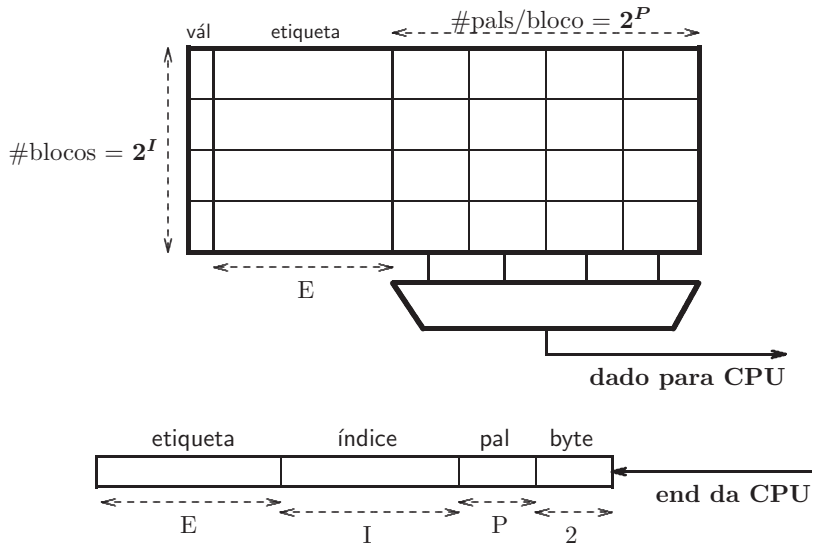
least recently used = LRU

Implementação:

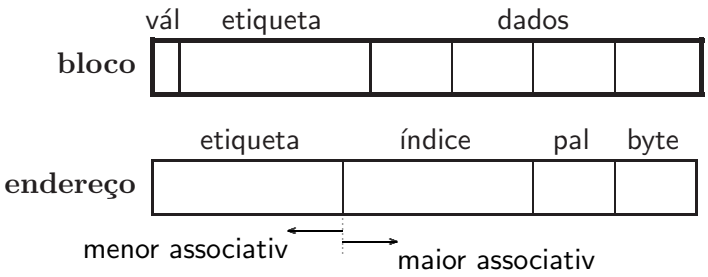
LRU 2-way: bit aponta para LRU
LRU 4-way: 3 bits, hierárquico

Aleatória: (a) somatório de alguns bits do endereço; (b) registrador de deslocamento com realimentação

Organização – Detalhes



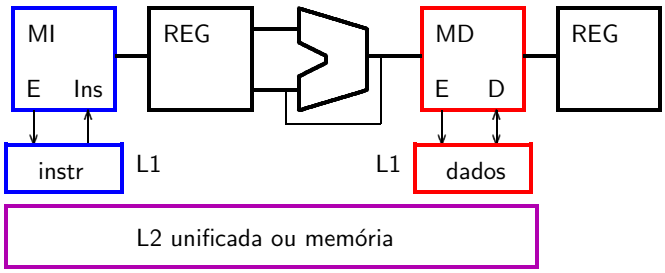
Organização – Detalhes



capacidade = num_blocos × tam_bloco × tam_palavra × associativ
|índice| = $\log_2$ num_blocos — altura da matriz
|pal| = $\log_2$ tam_bloco — # palavras/dados, largura
|byte| = $\log_2$ tam_palavra
|etiqueta| = tam_ender - ((|índice|/associativ) + |pal| + |byte|)

Caches e Pipelines

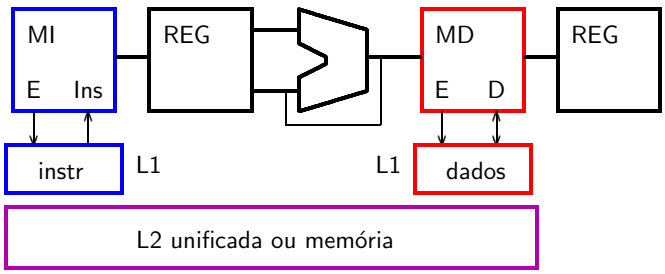
Tempo de Acerto é crucial porque limita o ciclo da CPU
Se muito longo ⟶ mais de um ciclo para acessar a cache nos acertos



Há duas portas para memória para eliminar risco estrutural;
2 caches separadas, uma para instruções, uma para dados porque
localidade no acesso aos dados é diferente do acesso a instruções

Caches e Pipelines

Acertos: se encontrou inst/dado na cache,
entrega para unidade funcional (RegInstr ou MEM) e prossegue



Faltas: segura execução até que memória entregue bloco faltante
instrução: segura busca, outras instruções prosseguem
dado: deve parar a busca até que memória entregue dado

Desempenho do sistema com cache

Equação do desempenho: $\frac{\text{segs}}{\text{prog}} = \frac{\text{instr}}{\text{prog}} \times \frac{\text{ciclos}}{\text{instr}} \times \frac{\text{segs}}{\text{ciclo}}$

Até agora, CPI com tempo de acesso à memória constante:

$\text{CPI}_{\text{real}} = \text{CPI}_{\text{ideal}}$ + ciclos esperando pela memória

CPI_{real} depende do Tempo Médio de Acesso à Memória

TMAM = TempoDeAcerto + (TaxaDeFaltas × PenalidadePorFalta)

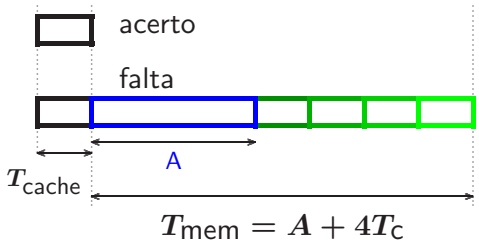
objetivo de projeto: minimizar TMAM para dados e instruções

cuidado: melhorar um termo pode piorar outros e aumentar TMAM

. *AMAT = Average Memory Access Time*

Desempenho do sistema com cache

TMAM = TempoDeAcerto + (TaxaDeFaltas × PenalidadePorFalta)



A = tempo de acesso à memória

T_{C} = tempo para copiar da memória para a cache (4 palavras por bloco)

penalidade = $A + 4T_{\text{C}}$

Compilador GCC com memória perfeita tem CPI=1.2 (dependências), 11% das referências causam faltas que custam 10 ciclos cada.

$$\begin{aligned} \text{CPI}_{\text{real}} &= (\text{CPI}_{\text{ideal}} + \text{faltas} \times \text{penalidade}) \\ &= (\mathbf{1.2 + 0.11 \cdot 10}) \;=\; \mathbf{2.3} \end{aligned}$$

Minimizar Tempo de Acerto

TMAM = TempoDeAcerto + (TaxaDeFaltas × PenalidadePorFalta)

- Tempo de acesso é proporcional à capacidade da memória
  ⟶ caches pequenas
- Mapeamento Associativo é mais lento do que mapeamento direto
  acesso às etiquetas ; comparação ; propagar através do MUX
- Segmentar o acesso à cache (aumenta vazão, mantém latência ≈)
  endereçamento ; acesso aos dados ; transferência 2-3 estágios

  adiciona estágios ao pipeline da CPU:
  2-3 est em BUSCA + 2-3 em MEM

Minimizar Taxa de Faltas

TMAM = TempoDeAcerto + (TaxaDeFaltas × PenalidadePorFalta)

- Minimizar a taxa de faltas, ajustando para os 3Cs
  - aumentar capacidade — tempo de acesso ↗
    capacidade = num_blocos × tam_bloco × tam_palavra
  - aumentar associatividade — tempo de acesso ↗
    capacidade = (n_blocos × associativ) × t_bloco × t_palavra
  - busca antecipada por hardware ou por software
    cuidado para não desalojar blocos úteis — poluição
    traz blocos e armazena em buffer específico

capacidade MD = num_blocos × tam_bloco × tam_palavra
[byte] = [bloco] × [pal/bloco] × [byte/pal]

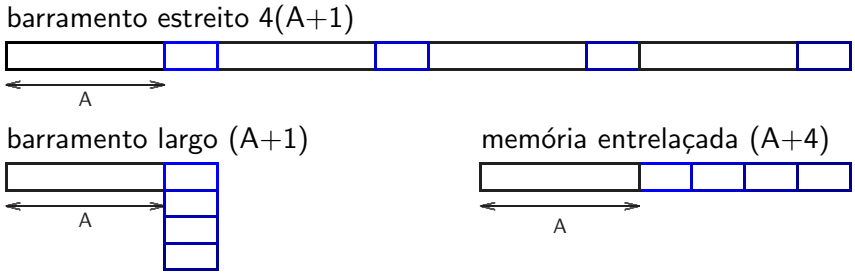
Minimizar Penalidade por Falta (i)

TMAM = TempoDeAcerto + (TaxaDeFaltas × PenalidadePorFalta)

- Minimizar o tempo de transferência com barramento mais eficiente

barramento largo (e caro);
cache organizada em bancos;
barramento CPU-cache com transações;

cache de vítimas: cache associativa pequena (4-8 blcs) mantém blocos expurgados da L1



Minimizar Penalidade por Falta (ii)

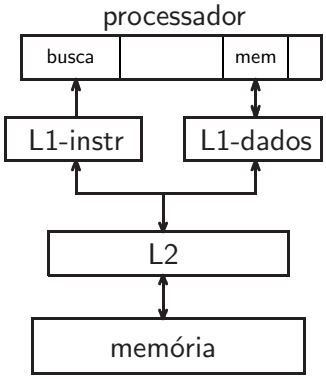
TMAM = TempoDeAcerto + (TaxaDeFaltas × PenalidadePorFalta)

Minimizar tempo de acerto
no nível inferior (L2 ou memória)

$$\text{TxFaltasLocal Ln} = \frac{\text{faltas-Ln}}{\text{refs-Ln}}$$

$$\text{TxFaltasGlobal} = \frac{\text{faltas-L1,2}}{\text{refs-CPU}}$$

Taxa de faltas na L2 é alta porque
L1 filtra referências com boa localidade,
sobram para L2 refs com localidade ruim



$$T_{\text{mam}} = T_{L1} + F_{L1} \cdot [\, T_{L2} + F_{L2} \cdot T_{m} \,]$$

Minimizar Penalidade por Falta (iii)

TMAM = TempoDeAcerto + (TaxaDeFaltas × PenalidadePorFalta)

- **Minimizar tempo de carga:** preenchimento do bloco é demorado
- ...mas não precisa esperar até que todo bloco seja preenchido
- **Early restart** – assim que palavra requisitada chegar da memória, entrega ao processador, que continua a executar (instruções)
- **Critical word first** – busca palavra requisitada primeiro e a entrega ao processador assim que chegar da memória (CPU continua)
  - processador requisita **p2**:

p0	p1	**p2**	p3

  - **p2** é entregue, e bloco é preenchido com p2 → p3 → p0 → p1
  - localidade espacial indica que próximo acesso será na palavra seguinte, que pode não ter chegado ainda

Resumo

- Hashing para mapear espaço grande em armazenador pequeno
  se largura aumenta (tamBloco>1) então altura diminui
- Três formas de hashing:
  - mapeamento direto → posição = ender % num_blocos
  - totalmente associativo → posição = ∀ no conjunto (índice)
  - associativo por conjuntos → posição = ender % num_conjs
- Três categorias de faltas:
  - por capacidade → cache maior
  - por conflitos → associatividade
  - compulsórias → busca antecipada (?)
- TMAM = TempoDeAcerto + (TaxaDeFaltas × PenalidadePorFalta)
  melhorar um termo geralmente piora outro/s

Exercícios

1) Projete uma cache com 512Kbytes de capacidade, e as associatividades indicadas abaixo, com blocos de 32 bytes. (a) mapeam direto, (b) associativ binária, (c) associativ 8-ária. Indique claramente a largura das estuturas (índice, etiquetas). Qual o número de bits total (matriz de dados+etiquetas) de cada cache?

2) Considere um processador com relógio de 500MHz (2ns). Você dispõe de 96kbytes de memória com 1ns de tempo de acesso; 2048kbytes de memória com tempo de acesso de 10ns; 4Gbytes de RAM dinâmica com tempo de acesso de 60ns. Projete uma hierarquia de memória completa, que minimize o tempo médio de acesso à memória. Suponha que a memória necessária para implementar as etiquetas existe em abundância e com tempo de acesso adequado ao seu uso.

Mais Exercícios

3) Considere os três barramentos do slide 26. Para blocos com 4, 8 e 16 palavras, calcule (i) a vazão de pico, (ii) a vazão sustentada (efetiva), para cada um dos três barramentos, considerando relógio de 1GHz (1ns). A unidade da vazão é [byte/s]

4) Suponha um programa que não causa faltas na L1-ins. A taxa de faltas na L1-D é 10%, e a taxa de faltas na L2 é 2%. O tempo de acesso à L1 é 1 ciclo, e o da L2 é 5 ciclos. A penalidade por falta na L2 é 100 ciclos. Qual o tempo médio de acesso à memória? Quais as taxas de falta locais (L1,L2) e global?