Aulas 23 e 24

- A memória cache
- Hierarquia de memória com 2 níveis
- Princípio de funcionamento da cache
  - localidade temporal
  - localidade espacial
- Cache com mapeamento associativo
- Cache com mapeamento direto
- Cache com mapeamento parcialmente associativo

José Luís Azevedo, Arnaldo Oliveira, Tomás Oliveira e Silva, Nuno Lau

Introdução

- A hierarquia de memória combina uma memória adaptada à velocidade do processador (de pequena dimensão) com uma (ou mais) menos rápida, mas de maior dimensão
- A memória rápida armazena um sub-conjunto da informação residente na memória principal
- Uma vez que a memória com que o processador interage diretamente é de pequena dimensão, a eficiência da hierarquia resulta do facto de se mover informação para a memória rápida poucas vezes e de se aceder a essa informação muitas vezes (antes de a substituir)
- Para se tirar partido deste esquema, a probabilidade de a informação (que o processador necessita) estar no nível mais elevado da hierarquia tem que ser elevada
- O que torna essa probabilidade elevada ?

Introdução

- Um programa não acede, tipicamente, a todo o seu código (ou dados) ao mesmo tempo com igual probabilidade
- Um programa acede a uma zona reduzida do espaço de endereçamento num dado intervalo de tempo - princípio da localidade:
  - **Localidade no espaço (spatial locality)**: Se existe um acesso a uma zona de memória então é provável que as zonas contíguas sejam também acedidas
  - **Localidade no tempo (temporal locality)**: Se existe um acesso a uma zona de memória então é provável que essa mesma zona seja acedida novamente num futuro próximo

Introdução

- **Localidade espacial**:
  "a informação que o processador necessita de seguida, tem uma elevada probabilidade de estar próxima da que consome agora"
  - Exemplos: instruções de um programa; processamento de um *array*
  - Quanto maior for o bloco (zona contígua de memória) de informação existente na memória rápida, melhor

- **Localidade temporal**:
  "a informação que o processador consome agora tem uma elevada probabilidade de ser novamente necessária num curto espaço de tempo"
  - Exemplos: variável de controlo de um ciclo, instruções de um ciclo
  - Quantos mais blocos de informação estiverem na memória rápida, melhor

Hierarquia de memória com 2 níveis

- Nível primário (superior) rápido e de pequena dimensão
- Nível secundário (inferior) mais lento mas de maior dimensão
  - O nível primário contém os blocos de memória mais recentemente utilizados
- Os pedidos de informação são sempre dirigidos ao nível primário, sendo o nível secundário envolvido apenas quando a informação pretendida não está nesse nível
  - Se os dados pretendidos se encontram num bloco do nível primário então existe um "**hit**"
  - Caso contrário ocorre um "**miss**"
- Na ocorrência de um "miss" acede-se ao nível secundário e transfere-se o bloco que contém a informação pretendida

Hierarquia de memória com 2 níveis

- A taxa de sucesso (**hit ratio**) é dada por:

$$\text{hit\_ratio} = \frac{\text{Nr\_hits}}{\text{Nr\_total\_acessos}}$$

- A taxa de insucesso (**miss ratio**) é dada por:

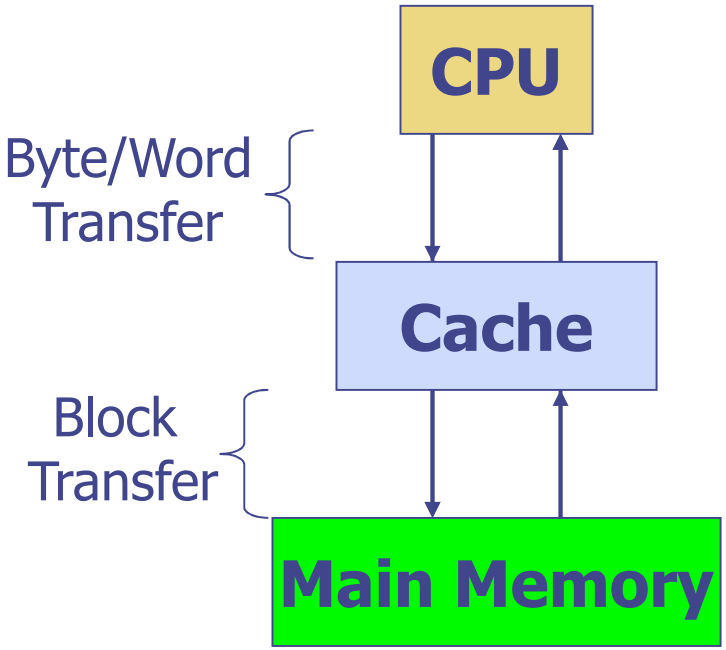
$$\text{miss\_ratio} = 1 - \text{hit\_ratio}$$

- O tempo de acesso no caso de um "hit" designa-se **hit time**
- O tempo de substituir um bloco do nível superior e enviar os dados para o processador é designado por **miss penalty**
- O tempo médio de acesso à informação é então:

$$t_a = \text{hit\_ratio} * \text{hit\_time} + (1 - \text{hit\_ratio}) * \text{penalty\_time}$$

Memória cache

- cache: nível de memória que se encontra entre o CPU e a memória principal (primeiras utilizações no início da década de 60)
- É comum os computadores recentes incluírem 2 ou mais níveis de cache; a cache é muitas vezes integrada no mesmo chip do processador. Exemplo - **cache read**:



Cache recebe endereço (MAddr) do CPU

O bloco que contém MAddr está na cache?

SIM:

Lê a cache e envia para o CPU o conteúdo de MAddr

NÃO:

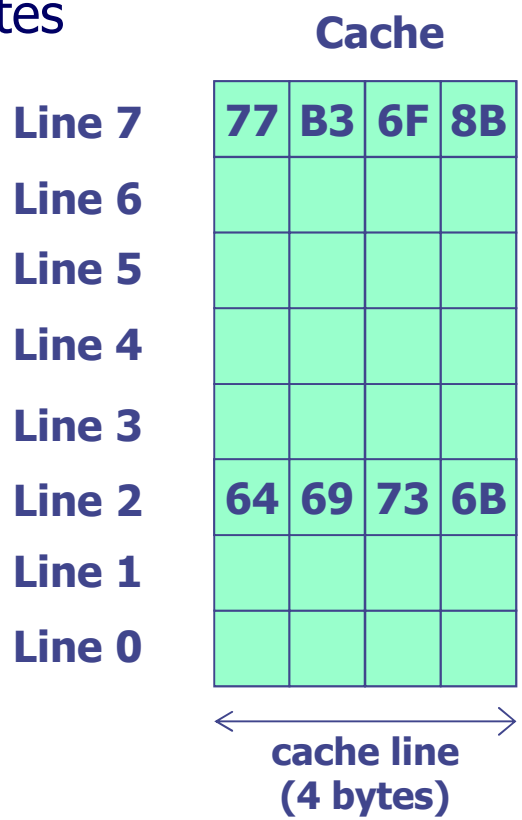
Encontra espaço na cache para um novo bloco

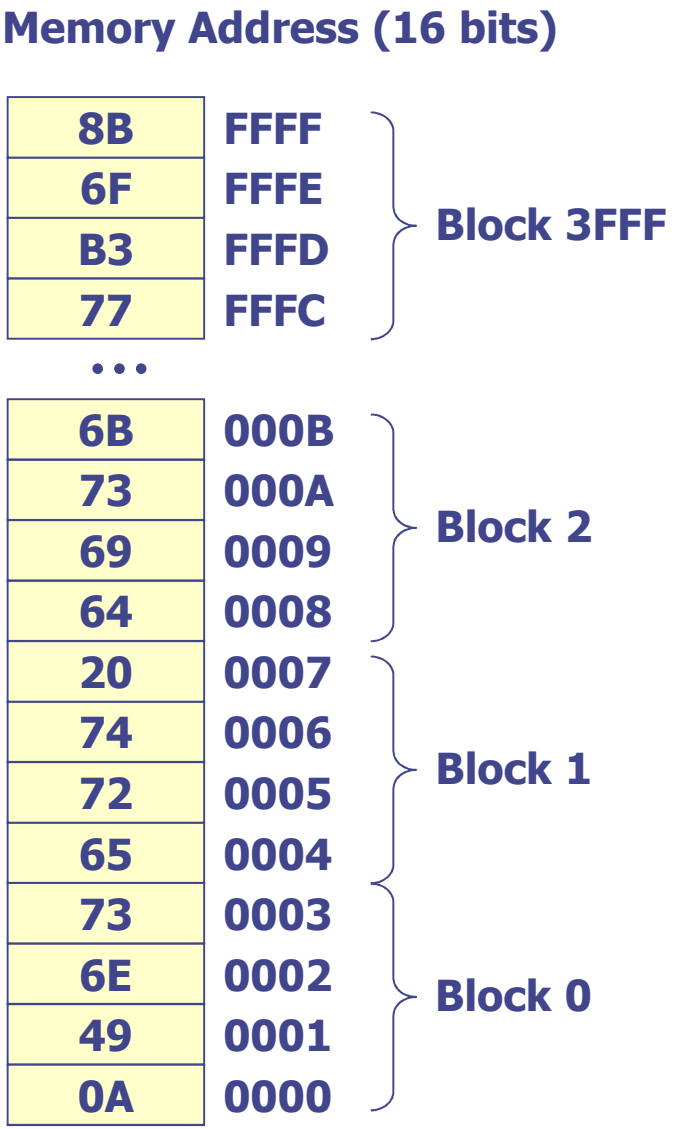
Acede à memória principal e lê o bloco que contém o endereço MAddr

Lê a cache e envia para o CPU o conteúdo de MAddr

Memória cache

- Exemplo: memória de 64K (16 bits de endereço), cache de 8 linhas de 4 bytes





- Quais os endereços da memória principal representados nesta cache?

Memória cache

- As operações da cache são transparentes para o programa; o programa emite um endereço e pede que seja feita uma operação de leitura ou escrita e esse pedido é satisfeito pelo sistema de memória: é desconhecido para o programa se é a cache ou a memória principal a satisfazer o pedido
- Na implementação da unidade de controlo da cache é necessário ter em atenção os seguintes aspetos:
  - Como saber se um determinado endereço está na cache?
  - Se está na cache, onde é que está?
  - Onde colocar um novo bloco na cache?
  - Qual o bloco a retirar da cache quando ocorre um miss?
  - Como tratar o problema das operações de escrita de modo a manter a coerência da informação nos vários níveis?
- Implementação tem de ser eficiente! Realizável em hardware

Organização dos sistemas de cache

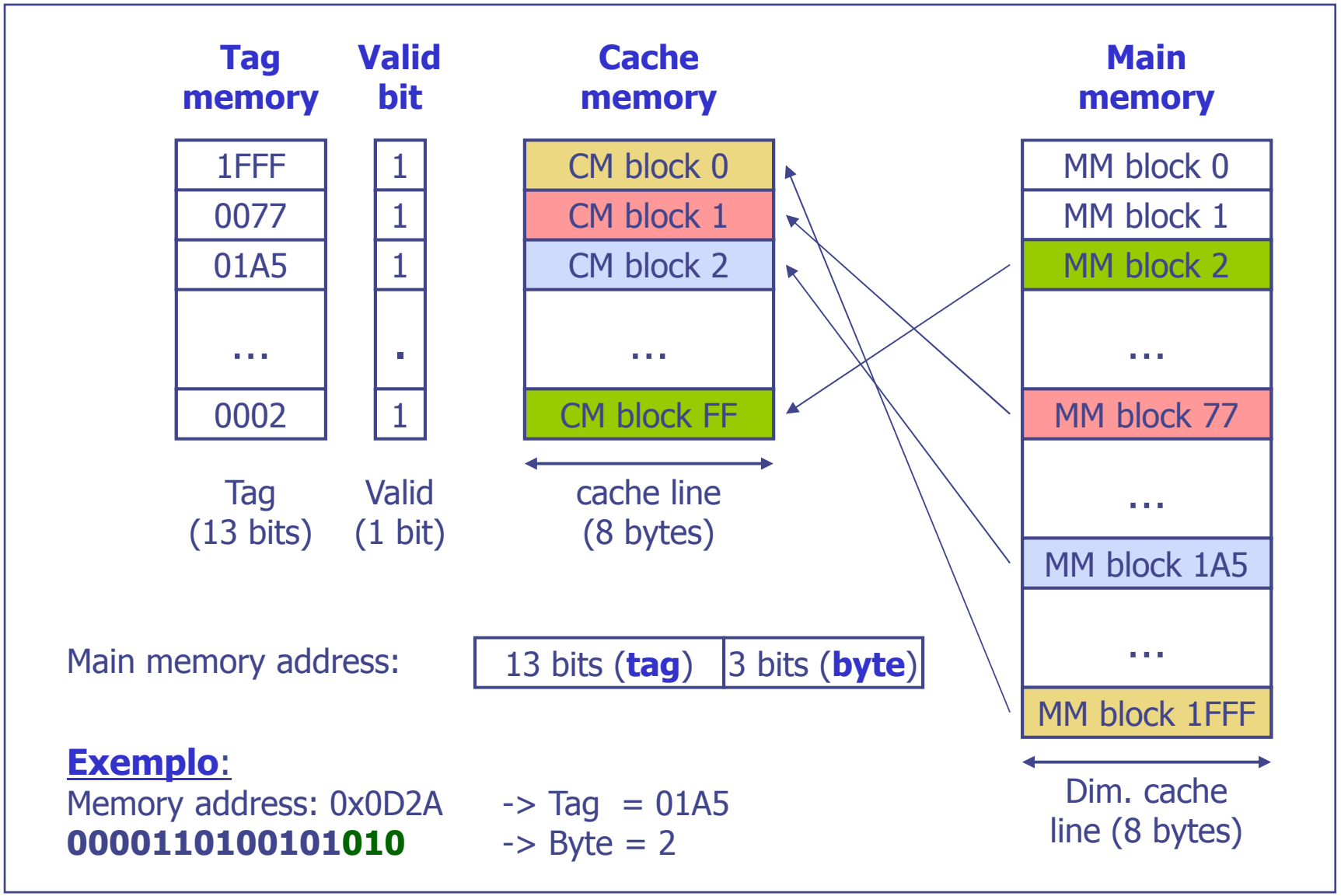
- Há basicamente 3 formas de organizar os sistemas de cache:
  - Cache **totalmente associativa** ("fully associative")
  - Cache **com mapeamento direto** ("direct mapped")
  - Cache **parcialmente associativa** ("set associative")

- Os slides seguintes apresentam a metodologia de organização de cada uma delas, partindo do seguinte conjunto de especificações-base:
  - Espaço de endereçamento de 16 bits (memória principal com 64 kBytes)
  - Memória cache com 256 linhas (posições) de 8 bytes (2^3) cada (ou seja, cada bloco tem uma dimensão de 8 bytes).

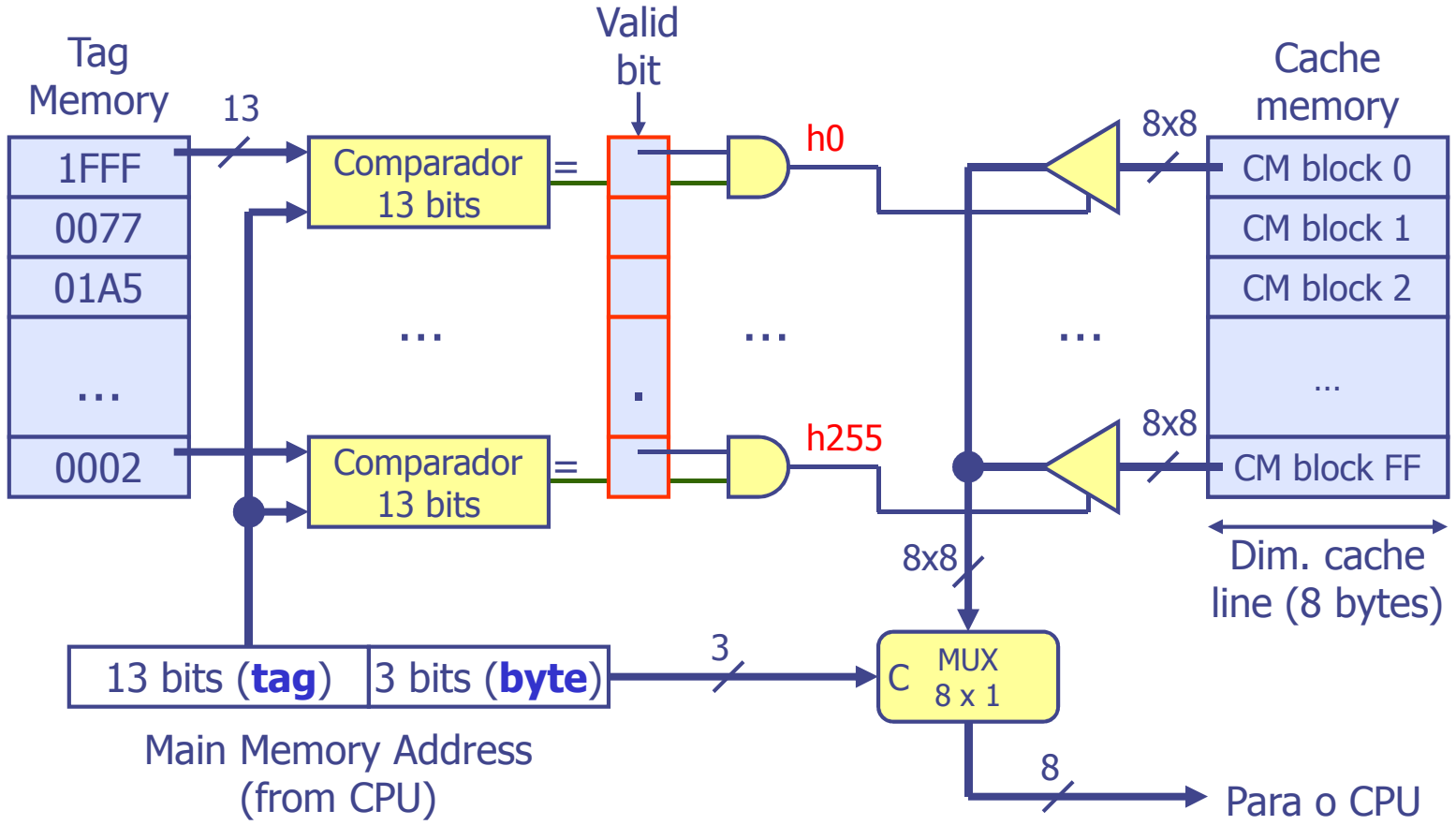
Exemplos de organização

- Especificação-base para os exemplos de organização que se seguem:
  - Espaço de endereçamento de 16 bits
  - Memória cache com 256 posições de 8 bytes
- Com estes valores, a memória principal pode ser vista como sendo constituída por um conjunto de 2^{13} blocos contíguos, de 8 bytes cada ($2^{16}/2^3 = 2^{13}$): 8k blocos (numerados de 0000 a 0x1FFF)
- Assim, no endereço de 16 bits, os
  - 13 bits mais significativos identificam o **bloco**
  - 3 bits menos significativos identificam o **byte dentro do bloco**

Cache com mapeamento associativo



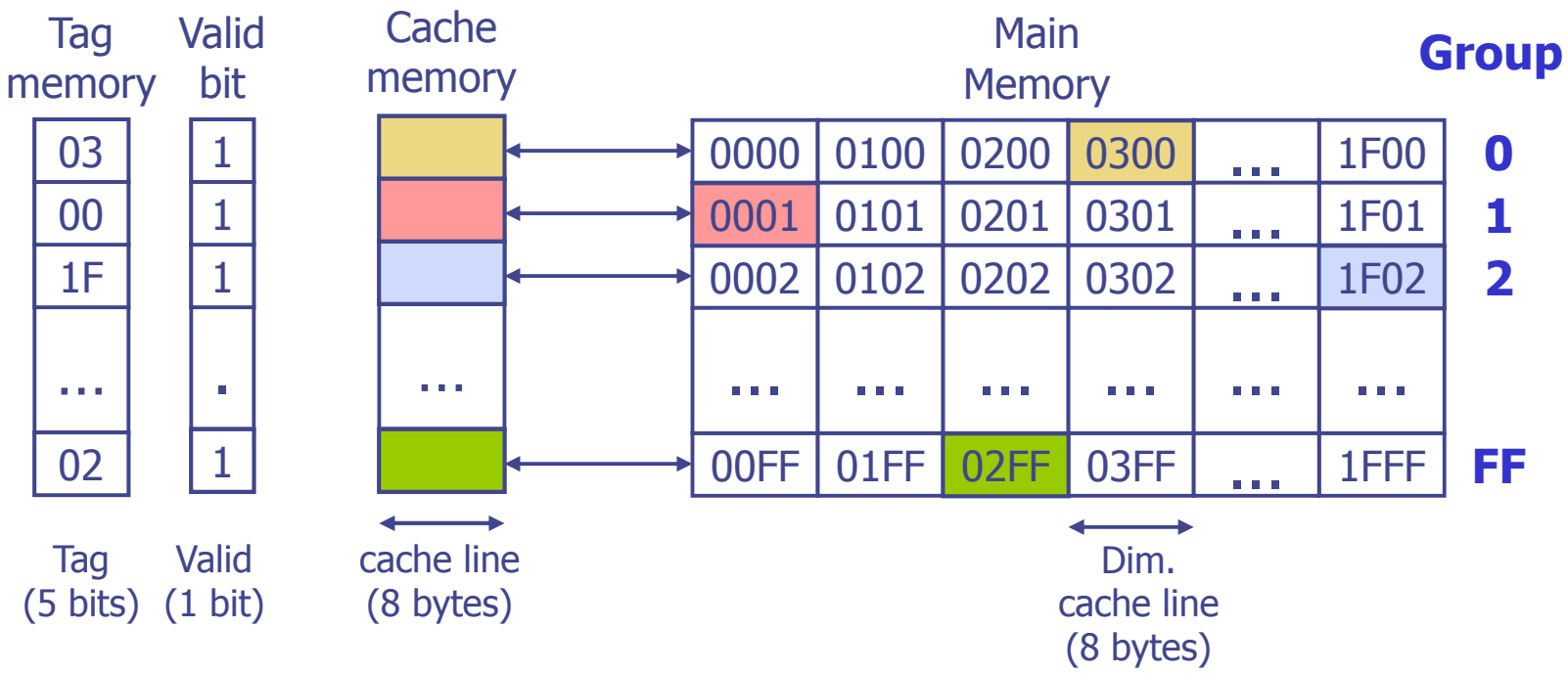
Cache com mapeamento associativo



Cache com mapeamento associativo

- Vantagens:
  - Qualquer bloco da memória principal pode ser colocado em qualquer posição da cache
- Inconvenientes:
  - Todas as entradas da memória "tag" têm de ser analisadas, de forma a verificar se um endereço se encontra na cache
  - Muitos comparadores, alto custo

Cache com mapeamento direto



Main memory address:

5 bits (**tag**)	8 bits (**group**)	3 bits (**byte**)

Exemplo:

Memory address = 0xF816 -> Tag = 0x1F (Block = 0x1F02)

1111100000010110 -> Group = 0x02

-> Byte = 6

Cache com mapeamento direto

- O endereço do bloco da memória é constituído pelos bits dos campos "tag" e "group": endereço do bloco = [tag group], i.e. "tag" concatenado com "group"
- O endereço do bloco é obtido, a partir do endereço real, por:

  Endereço do bloco = endereço real / dimensão do bloco

- A posição da cache (i.e. a linha) associada a um dado endereço é dada pelo campo "group" e também pode ser obtida por:

  Pos = endereço do bloco % número de blocos da cache

- A dimensão do bloco e o número de blocos da cache são potências de 2

- **Exemplo**: endereço=0xF816, dimensão do bloco=8 bytes, 256 linhas

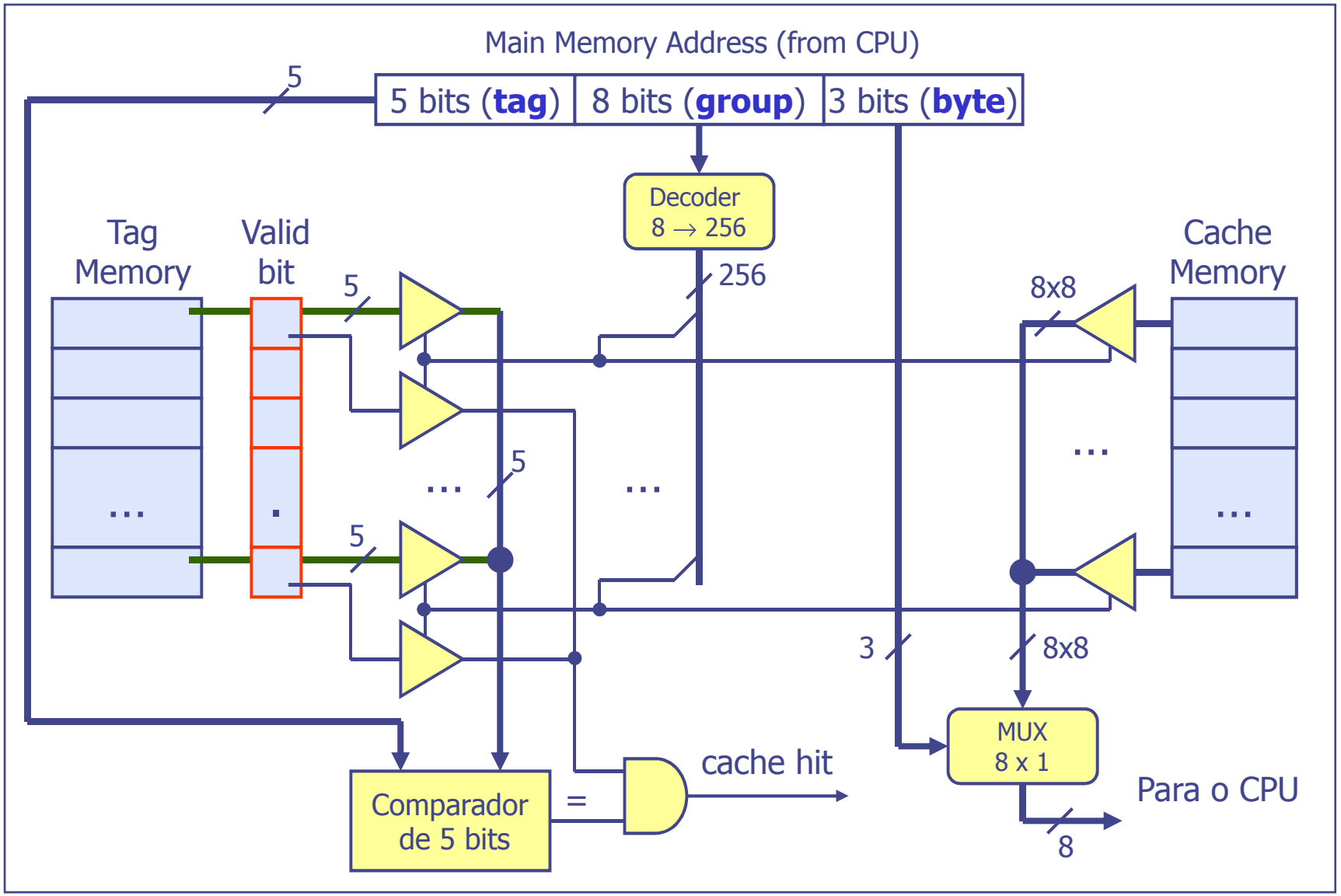
  endereço do bloco:

  0xF816 / 8 = 0x1F02 (13 bits mais significativos do endereço)

  group: 0x02 (8 bits menos significativos do endereço do bloco)

  tag: 0x1F (5 bits mais significativos do endereço do bloco)

Cache com mapeamento direto



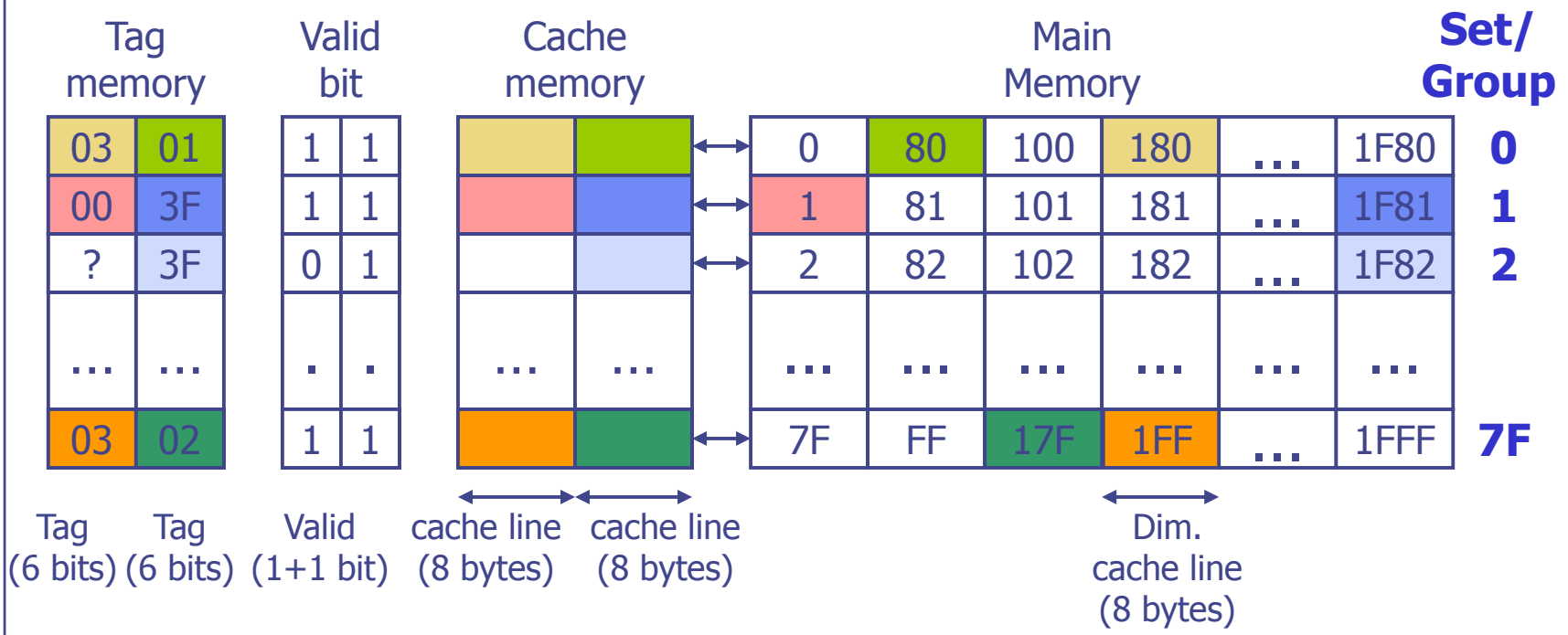
Cache com mapeamento direto

- A cada bloco da memória principal é associada uma linha da cache; o mesmo bloco é sempre colocado na mesma linha
- Vários blocos têm associada a mesma linha
- Maior vantagem:
  - Simplicidade de implementação
- Maior desvantagem:
  - Num dado instante apenas um bloco de um dado grupo pode residir na cache, o que tem como consequência que alguns blocos podem ser substituídos e recarregados várias vezes, mesmo existindo espaço na cache para guardar todos os blocos em uso

- Uma melhoria óbvia deste tipo de cache seria permitir o armazenamento simultâneo de mais que um bloco do mesmo grupo
- É esta melhoria que é explorada na cache **parcialmente associativa**

Cache com mapeamento parcialmente associativo

(exemplo com associatividade de 2)



Main memory address:

6 bits (**tag**)	7 bits (**set**)	3 bits (**byte**)

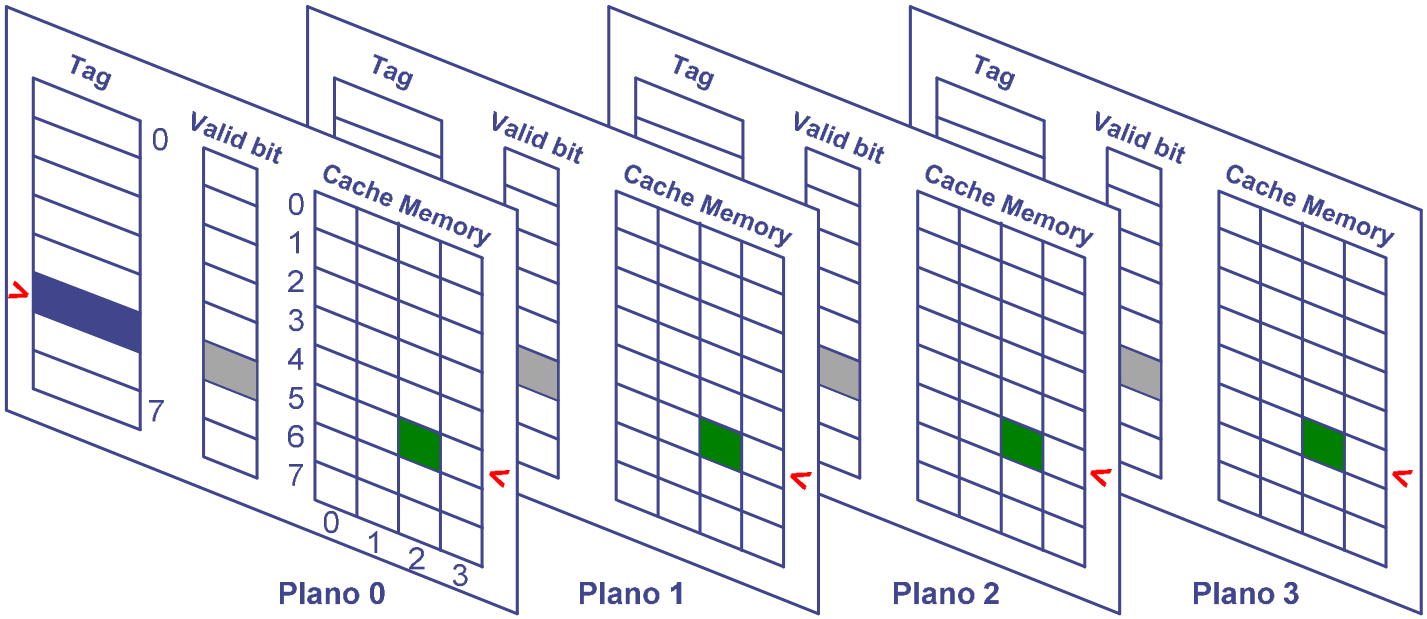
Exemplo:

Memory address = 0x17FC
0001011111111100

-> Tag = 0x05 (Block=0x2FF)
-> Set/Group = 0x7F
-> Byte = 4

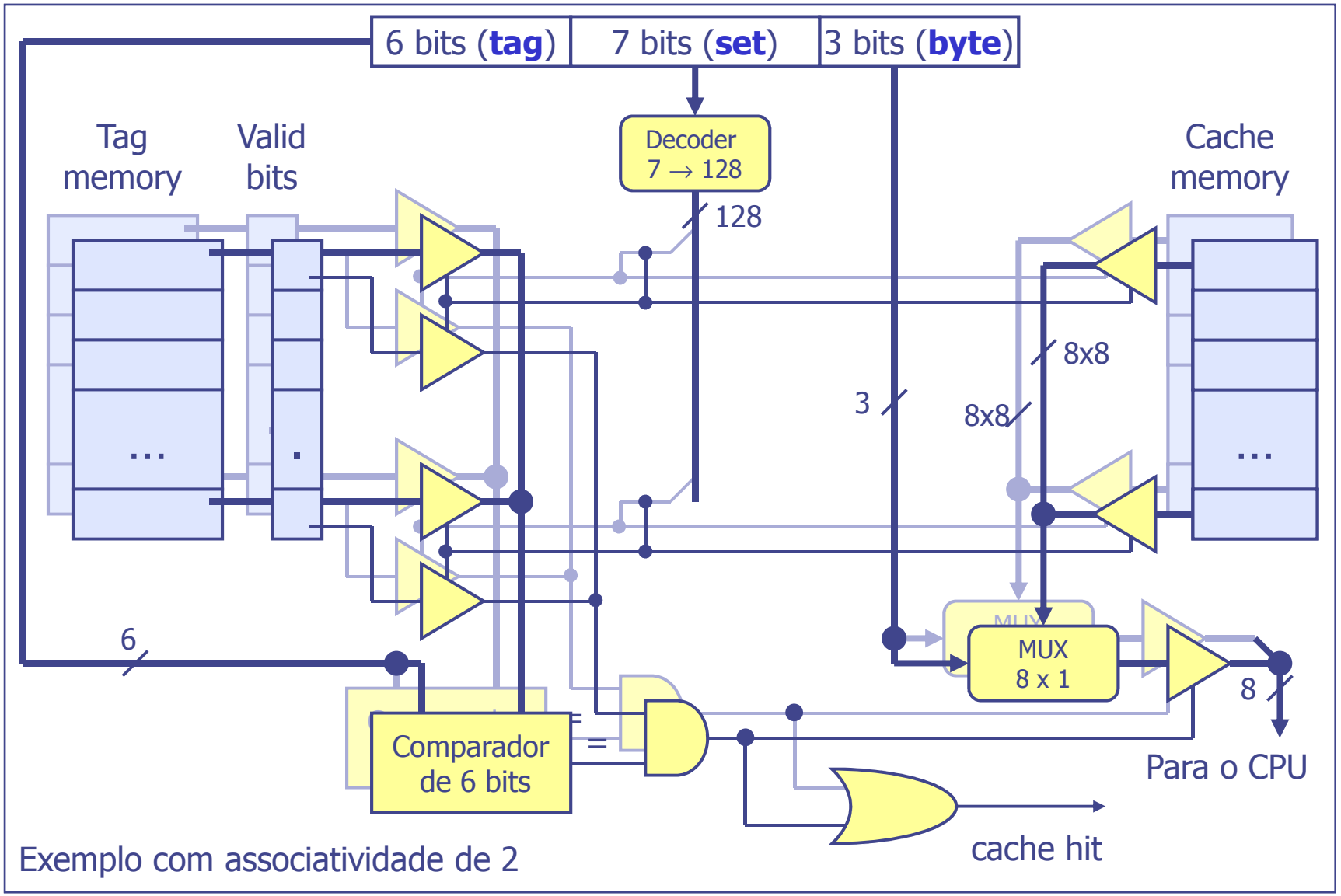
Cache com mapeamento parcialmente associativo

- Esquematização de uma cache com **associatividade de 4**, de **128 bytes**, com **8 linhas** por plano (128 / 4 = 32 bytes por plano, 32 / 8 = 4 bytes por linha)
- Se o endereço gerado pelo CPU for (16 bits): **0110101010010110**



- A comparação dos 11 bits mais significativos (A15-A5) do endereço com as "tags" é feita simultaneamente nos 4 planos
- A posição da memória "tag" onde é feita a comparação é determinada pelos bits A4-A2 do endereço (posição 5 no exemplo)
- O plano onde ocorre o hit (se ocorrer) fornece o byte armazenado na cache na posição determinada pelos bits A1-A0 do endereço (posição 2 no exemplo)

Cache com mapeamento parcialmente associativo



Cache parcialmente associativa

- A cache parcialmente associativa é semelhante à cache com mapeamento direto, mas a primeira permite que mais que um bloco de um mesmo grupo possa estar na cache
- Uma cache com associatividade de 2 permite que 2 blocos do mesmo grupo possam estar simultaneamente na cache
- A divisão do endereço de memória é igual ao do mapeamento direto (o campo "group" é normalmente designado por "set" - conjunto), mas agora há "n" possíveis lugares onde um dado bloco de um mesmo grupo pode residir (para uma cache com associatividade de "n")
- Os "n" possíveis lugares onde o bloco pode residir têm que ser procurados simultaneamente

- (Block-set associative cache / n-way-set associative cache)

Estratégias de substituição

- As estratégias de substituição de blocos na cache, na ocorrência de um miss, são várias; as mais utilizadas são as seguintes:
- **LRU – Least Recently Used**: é substituído o bloco da cache que está há mais tempo sem ser referenciado
- **LFU – Least Frequently Used**: substituído o bloco menos acedido
- **FIFO** – First in first out: é substituído o bloco que foi carregado há mais tempo
- **Random**: substituição aleatória (testes indicam que não é muito pior do que LRU)

Políticas de escrita

- **Write-through**
  - Todas as escritas são realizadas simultaneamente na cache e na memória principal
  - A memória principal está sempre consistente
  - Se endereço ausente na cache, atualiza apenas a memória principal (**write-no-allocate**)
- **Write-back**
  - Valor escrito apenas na cache; novo valor é escrito na memória quando o bloco da cache é substituído
  - "**Dirty bit**" (esta flag é ativada quando houver uma escrita em qualquer endereço do bloco presente na linha da cache)
  - Mais complexo do que "write-through"
  - Se endereço ausente na cache, carrega o bloco para a cache e atualiza-o (**write-allocate**)

Exercícios

1. Qual a posição dos endereços de bloco 1 e 29 numa cache de mapeamento direto com 8 linhas?
2. Considere um sistema computacional com um espaço de endereçamento de 32 bits e uma cache com 256 blocos de 8 bytes cada um.
   - Qual o tamanho em bits dos campos tag, group e byte supondo que a cache é: 1) associativa; 2) mapeamento direto; 3) com associatividade de 2.
3. Para a implementação de uma cache com 64KB de dados e blocos de 4 bytes num sistema computacional com um espaço de endereçamento de 32 bits, quantos bits de armazenamento são necessários supondo:
   - uma cache associativa;
   - uma cache com mapeamento direto;
   - uma cache com associatividade de 2.