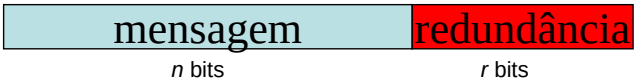
Correção de Erros

- Erros de memória de semicondutores podem ser:
  - Erros graves que constitui um defeito físico permanente;
  - Erros moderados, onde a(s) célula(s) não são capazes de armazenar os dados ou fazem com os dados variem.
- Causa das falhas:
  - Grave: uso excessivo em ambiente inadequado, por defeito de fabricação ou por desgaste.
  - Moderado: problemas com o fornecimento de energia.

Controle de Erros

- Princípio
  - Inclusão de informações redundantes junto com cada bloco de dados gravados.
    - Identificação de um erro
    - Deduzir o conteúdo que deveria estar escrito



m bits = n bits + r bits -> **Palavra de Código**

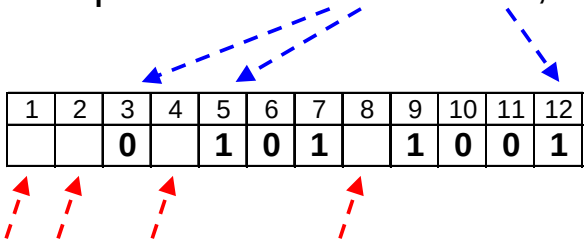
Exemplo:

1	0	0	1	1	

Bit de paridade, pode ser:
=0 para uma soma de bits par
(depende da convenção)

Correção de erros

- Código de Hamming
  - Bits que são potência de 2 são bits de verificação
    - 1,2,4,8,16 etc
  - Os demais bits são preenchidos com os *m* bits de dados.
  - Exemplo: enviar “01011001”, ficaria então

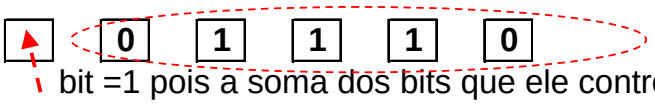


1	2	3	4	5	6	7	8	9	10	11	12
		0		**1**	**0**	**1**		**1**	**0**	**0**	**1**

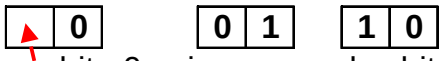
  - Bits de verificação

Correção de erros

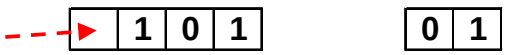
1	2	3	4	5	6	7	8	9	10	11	12
		0		**1**	**0**	**1**		**1**	**0**	**0**	**1**



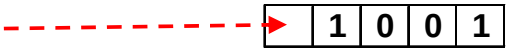
bit =1 pois a soma dos bits que ele controla foi impar (0+1+1+1+0=3)



bit =0 pois a soma dos bits que ele controla foi par (0+0+1+1+0=2)



bit =1 pois a soma dos bits que ele controla foi impar (1+0+1+0+1=3)



bit =0 pois a soma dos bits que ele controla foi impar (1+0+0+1=2)

Então ficaria

1	2	3	4	5	6	7	8	9	10	11	12
1	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**

supor bit 5 com erro, era =1

1	2	3	4	5	6	7	8	9	10	11	12
1	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**

1 **0** **0** **1** **1** **0**

(0+0+1+1+0)=par logo bit deveria ser 0 mas é 1 : erro!

0 **0** **0** **1** **1** **0**

ok sem erro!

1 **0** **0** **1** **0** **1**

(0+0+1+0+1)=par logo bit deveria ser 0 mas é 1 : erro!

0 **1** **0** **0** **1**

ok sem erro!

Então ficaria

bit 1 com erro somar 1
bit 3 com erro somar 4
então bit identificado com problema 1+4=5

SECDED

- single-error-correcting and double-error-detecting (SECDED)
- Com apenas 1 bit a mais é possivel detectar dois erros embora nao se possa corrigi-los por este algoritmo

Correção de um único erro

Bits de dados	Bits de teste	Porcentagem de aumento
8	4	50
16	5	31,25
32	6	18,75
64	7	10,94
128	8	6,25
256	9	3,52

Memória *Cache*

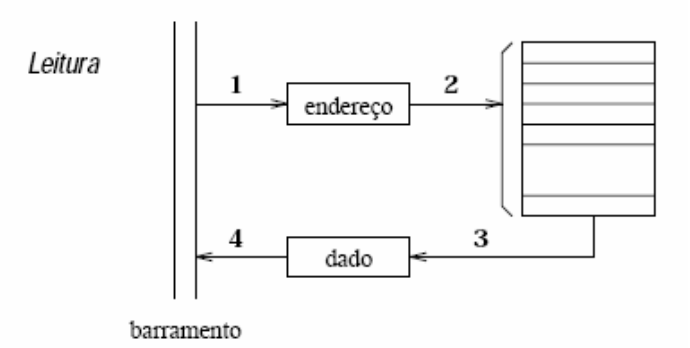
- Princípio de funcionamento:
  - duplicar parte dos dados contidos na memória principal em um módulo menor (o *cache*) composto por dispositivos de memória mais rápidos.

- Quando o processador solicita um item de dado, o gerenciador de memória requisita este item do *cache*.
- Duas situações podem ocorrer:

  - ***cache hit*:** item está presente no *cache*, é retornado para o processador praticamente sem período de latência;
  - ***cache miss*:** item não está presente no *cache*, processador deve aguardar item ser buscado da memória principal.

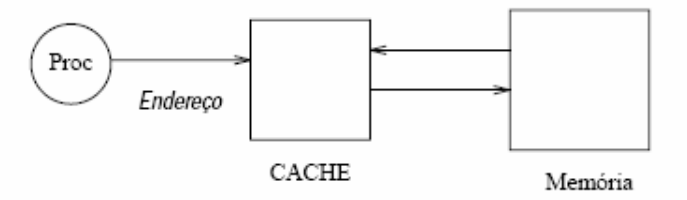
Localização de itens no *cache*

- Em um acesso “convencional” à memória principal, um item buscado é selecionado com base em seu **endereço** na memória:



Localização de itens no *cache*

- A idéia por trás do princípio do *cache* é trazer os dados da memória principal que estão sendo mais utilizados para a memória rápida e mais próxima do processador:



Localização de itens no *cache*

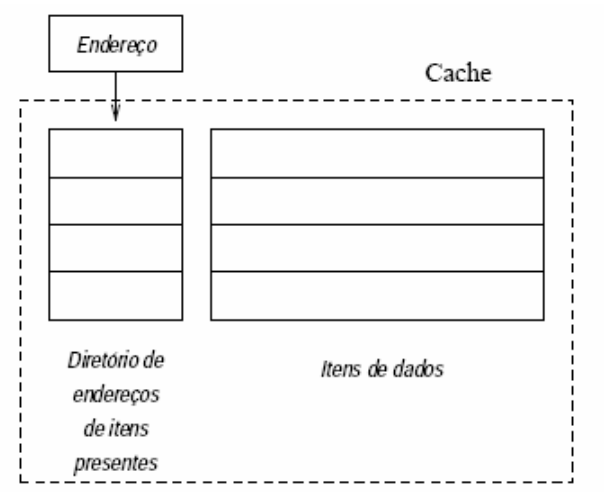
- A presença do *cache* é transparente para o processador.
- Se o item solicitado está presente *(cache hit)*, ele é entregue à CPU com baixa latência.
- Caso esteja ausente, um ciclo de acesso convencional à memória principal é realizado, com o processador recebendo o item após um período de espera (latência alta).

Localização de itens no *cache*

- Há ainda um problema de localização de item no *cache*.
- Ao copiar um item da memória principal para a *cache*, a informação sobre a posição (endereço) do item na memória principal não é válida para a posição do item no *cache*.
- Como saber então se o item buscado está presente ou não no *cache*?

Estrutura Básica

- Para cada item de dado armazenado no *cache*, armazena-se também uma parte do endereço (o *tag*) do item na memória principal.
- O conjunto de *tags* armazenados constitui o **diretório** do *cache*;
- o item é armazenado em uma área de dados associada àquele *tag*:



Estrutura básica

- Quando o módulo de *cache* recebe uma requisição de um item do processador, o endereço do item é buscado entre os endereços dos itens (seus *tags*) armazenados no *cache*.

- *Se o tag* estiver presente no diretório, o item solicitado é entregue.

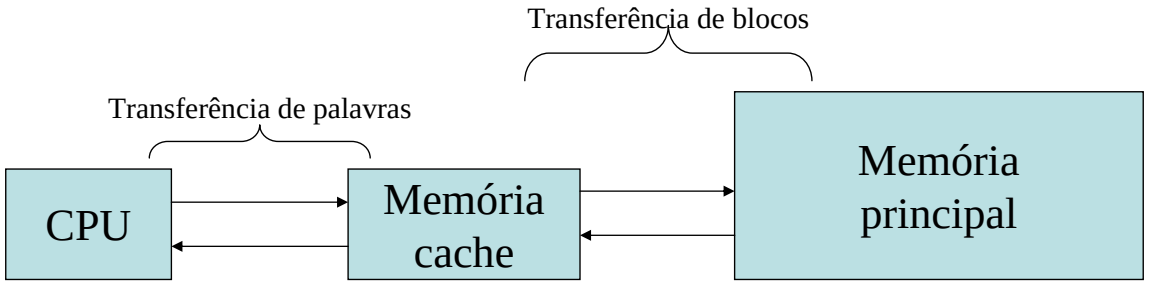
Estrutura básica

- Se o *tag* não estiver no diretório, sinaliza-se um *cache miss* e um bloco da memória principal contendo o item solicitado é transferido para o *cache*.

- Pode ser necessário, neste caso, remover um bloco presente no *cache* para abrir espaço para o novo bloco transferido.

Estrutura básica

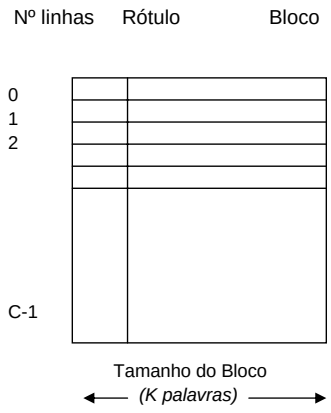
- A busca de um endereço na área de diretório de um *cache* requer uma comparação do endereço buscado com os *tags* armazenados no diretório, o que caracteriza um modo de acesso **associativo**.
  - Quando um item é trazido da memória principal para o *cache*, alguns itens armazenados nas posições vizinhas também são trazidos para o *cache*.

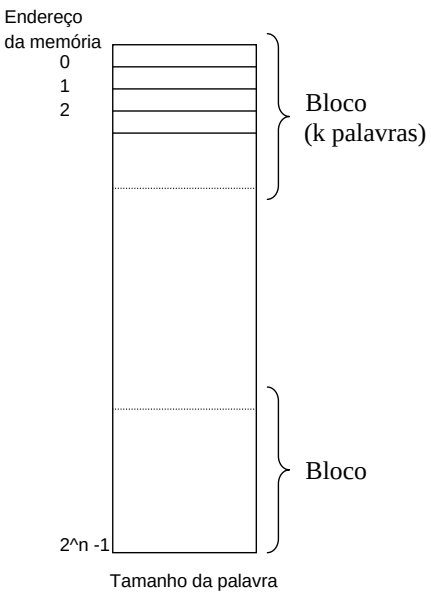


Estrutura básica

- É a forma de aproveitar a propriedade de localidade seqüencial.
- O conjunto de itens que é transferido entre memória principal e *cache* é chamado de bloco ou uma **linha** do *cache*. O número de palavras em uma linha é um dos parâmetros no projeto do *cache*

Estrutura básica





Tempo de acesso efetivo

- A latência para operações de leitura à *cache* varia entre os dois valores correspondentes às situações de *cache hit* ou *miss*.
- Seja t_c o tempo de acesso a um item no módulo *cache*,
- e seja t_l o tempo de acesso a uma linha na memória principal.
- As duas situações possíveis são:
  - **Leitura com *hit*:** neste caso, o tempo de acesso visto pelo processador é t_c;
  - **Leitura com *miss*:** neste caso, o tempo observado pelo processador é $t_c + t_l$.

Tempo de acesso efetivo

- Seja *h* é a probabilidade de item buscado estar presente no *cache*, ou seja,

$$h = \frac{\text{número palavras encontradas no } cache}{\text{número total de referências à memória}}$$

- parâmetro *h* é usualmente chamada de **taxa de acerto**, ou *hit ratio*.
  - Então, o tempo de acesso efetivo à *cache* pode ser expresso por:

$$t_{\text{eff}} = t_c + (1 - h)t_l$$

  - O termo 1 - *h* é usualmente denominado a **taxa de ausência** do *cache* (*miss ratio*).

Tempo de acesso efetivo

- Tipicamente, o valor de h está em torno de **0.85**, embora este valor varie sensivelmente de acordo com o tipo de aplicação.
- Pequenas variações em h podem afetar significativamente o tempo efetivo de acesso.

- Em geral, expressa-se a **latência** em termos de ciclos do processador.
- Tipicamente, a latência para um *hit* está em torno de 1 a 2 ciclos, enquanto que a latência com *miss* pode variar de 5 a 20 ciclos de CPU.

Organização

- É fundamental para o bom desempenho do *cache* que a busca a um item seja realizada em um tempo curto.

- Por este motivo, o uso de **memórias associativas** (em *hardware*) para implementar o diretório do *cache* não é uma solução viável, pois seu tempo de acesso é longo.

Organização

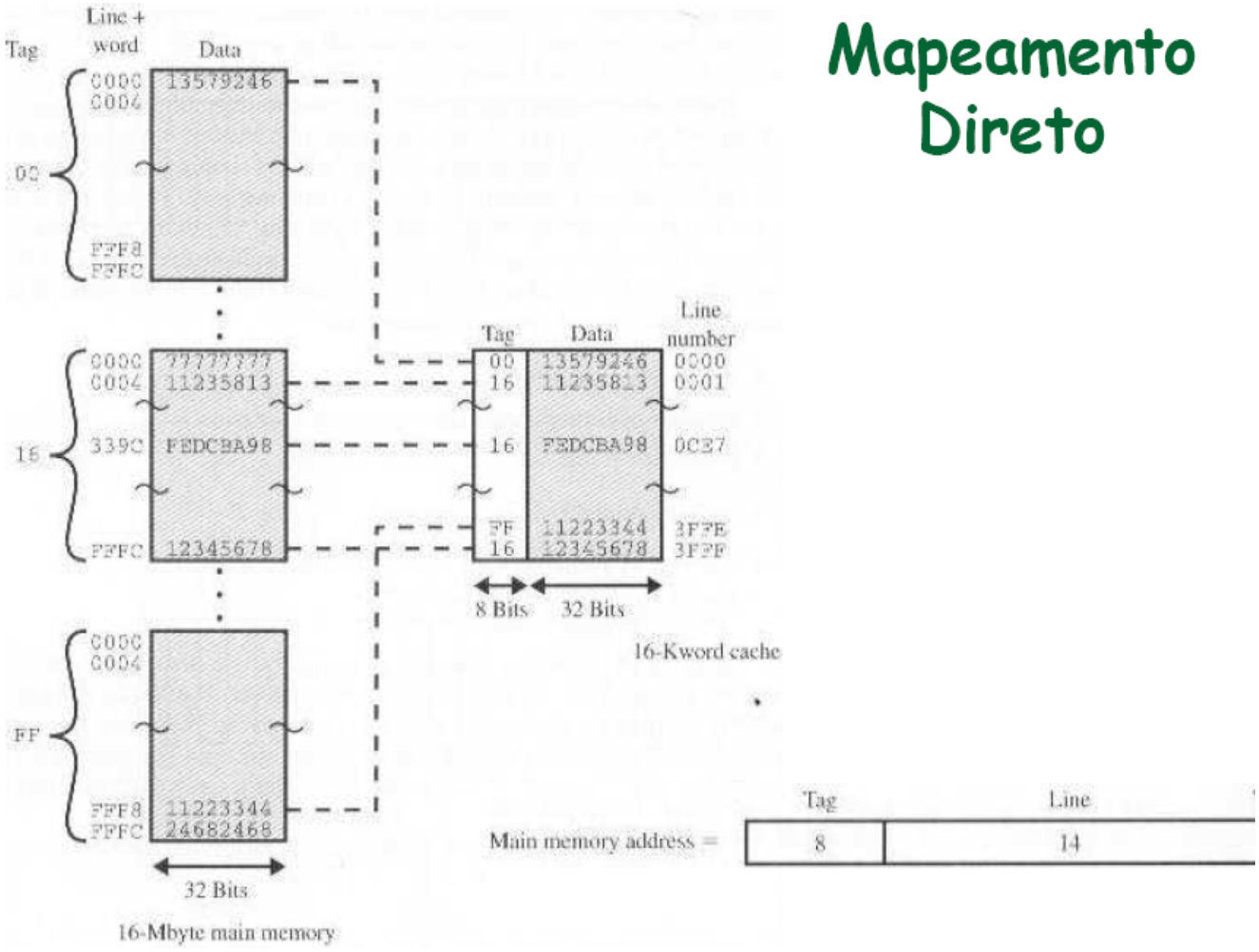
- É preciso utilizar uma organização interna para o *cache* que permita fazer a busca rápida, usando dispositivos convencionais de memória.
- Evidentemente, uma busca seqüencial ao diretório é inconcebível.
- Uma das alternativas para agilizar a busca de um *tag* no diretório do *cache* é reduzir o número de posições que devem ser buscadas.

Organização

- Existem três estratégias básicas de organização do *cache*:
  - 1. Organização por mapeamento direto;
  - 2. Organização completamente associativa;
  - 3. Organização associativa por conjuntos.

Organização por mapeamento direto

- Há apenas uma posição no *cache* para onde um item de dado pode ser transferido ➔ A busca restringe-se a uma inspeção nesta posição, requerendo apenas a presença de um comparador.
- Se o *tag* do item solicitado for igual ao *tag* associado àquela posição de memória, o *cache hit* está caracterizado.
- Caso contrário, como nesta organização o item não pode estar em nenhuma outra posição, o *cache miss* é caracterizado.
  - Então a linha contendo o item solicitado é transferida da memória principal para aquela posição do *cache*.



Organização por mapeamento direto

- Vantagens:
  - simplicidade de *hardware*, e conseqüentemente custo mais baixo;
  - não há necessidade de escolher uma linha para ser retirada do *cache*, uma vez que há uma única opção. Isto dispensa a implementação de um algoritmo de troca de linhas; e
  - operação rápida

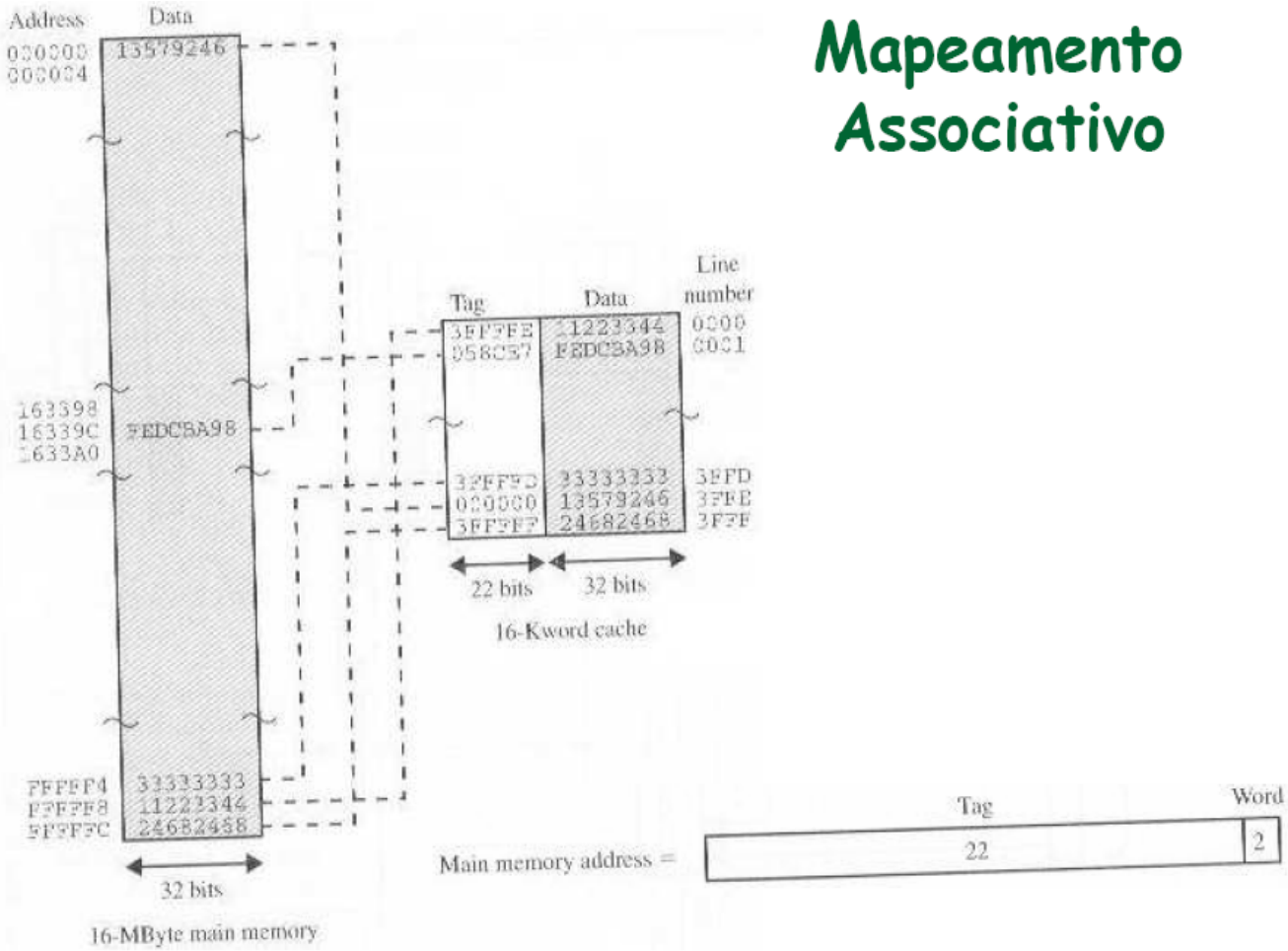
Organização por mapeamento direto

- Apesar de simples, esta estratégia é pouco utilizada devido à sua baixa flexibilidade.
- Como há apenas uma posição no *cache* onde o item solicitado pode estar localizado, a probabilidade de presença no *cache* h pode ser baixa.
- Assim, pode haver queda sensível no desempenho em situações onde o processador faz referências envolvendo itens que são mapeados para uma mesma posição do *cache*.

Organização completamente associativa

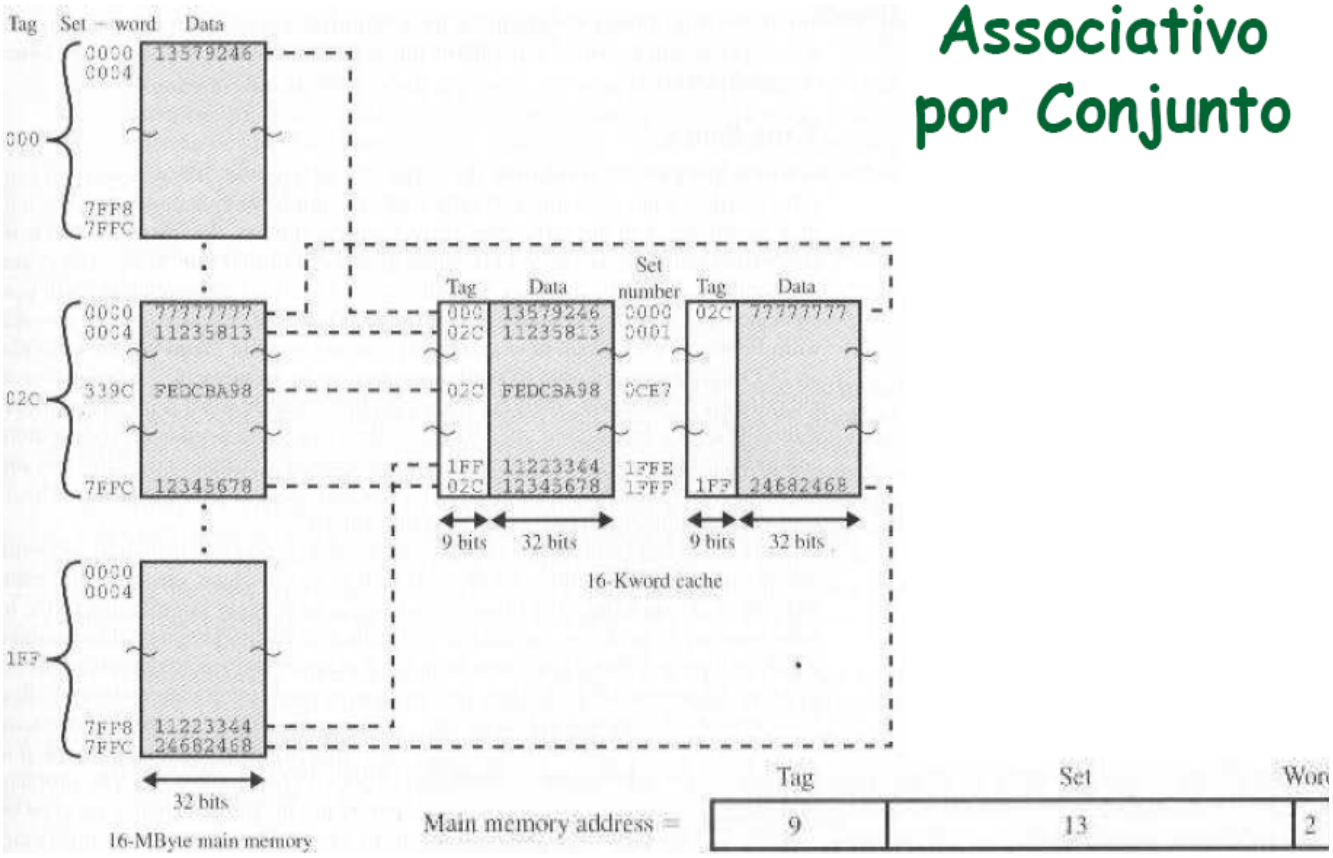
- Um item de memória pode estar localizado em qualquer posição do *cache*.
- O processo de identificar se um item está ou não presente no *cache* requer uma comparação simultânea do *tag* buscado com todas as linhas do diretório.
- A comparação com todas as linhas do diretório requer a utilização de memórias associativas.
  - Por este motivo, a formação de um *cache* totalmente associativo eficiente e economicamente viável é limitado a *caches* de dimensões reduzidas, como em algumas implementações de *cache* interno (no *chip* do processador).

Mapeamento Associativo



Organização associativa por conjuntos

- Oferece um compromisso entre o desempenho da organização por mapeamento direto e a flexibilidade da organização completamente associativa.
- O *cache* é particionado em *N* conjuntos, onde cada conjunto pode conter até *K* linhas distintas.
- *K* é a *associatividade* do *cache*; na prática, *1 < K <= 16*



Organização associativa por conjuntos

- Uma dada linha só pode estar contida em um único conjunto, em uma de suas *K* linhas.
- O conjunto selecionado é determinado a partir do endereço do item sendo referenciado.

Políticas de troca de linha

- Quando todas as posições disponíveis para uma linha requisitada e ausente do *cache* estão preenchidas, uma das linhas deve ceder lugar à linha requisitada.
- Portanto, é necessário estabelecer uma política de troca de linhas.

Algoritmo de Substituição

Os quatro algoritmos mais comuns são:

- Least recently used – LRU: substitui o bloco menos recentemente usado; É usado um bit de USO para indicar quando uma linha foi referenciada
- First-in-First-out – FIFO: substitui o bloco do conjunto que foi armazenado primeiro na memória cache;
- Least frequently used – LFU: substitui o bloco do conjunto menos freqüentemente utilizado; É usado um contado a cada linha da memória cache;
- Substituição aleatória;

Políticas de troca de linha

- LRU *(Least Recently Used)* é a estratégia mais utilizada em sistemas comerciais, sendo passível de implementação em hardware.
- FIFO *(First In, First Out)*, apesar de permitir uma implementação mais simples, apresenta taxas de ausência cerca de 12% maior que para LRU.

LRU com diretório sombra

- O diretório sombra é uma alternativa para contornar deficiência de LRU sob a presença de “ciclos longos”.
- Nesta abordagem, o cache tem duas áreas de diretório, principal e sombra.
- Quando um novo item é trazido para o *cache*, o *tag* da linha retirada do *cache* é armazenado no diretório sombra.
- Quando um item solicitado ao *cache* está ausente, esta ausência pode ser de duas formas:
  - **transitória:** dado não está no *cache* e o *tag* correspondente não está no diretório sombra
  - **sombra:** dado não está no *cache* porém seu *tag* correspondente está no diretório sombra

LRU com diretório sombra

- A ocorrência de uma ausência sombra indica que o dado esteve no *cache* há algum tempo e agora está novamente sendo requisitado.
- Este comportamento pode caracterizar a detecção de um ciclo longo.
- A linha é trazida para o *cache* e sua entrada é marcada como oriunda de uma ausência sombra.
- Na próxima vez que um item tiver que ser selecionado para remoção, este item pode receber maior prioridade de permanência no *cache*.

LRU com diretório sombra

- Um bit é necessário para diferenciar ausência sombra de ausência transitória.
- O custo adicional associado à implementação do diretório sombra é relativamente baixo, pois o maior custo do *cache* corresponde à área de dados.
- A utilização do diretório sombra não acarreta em sobrecarga no tempo de processamento do *cache*.
- Os benefícios associados à utilização do diretório sombra refletem-se na forma de uma redução do número de ausências entre 10 a 30%.

Política de Atualização

Antes que um bloco residente na memória cache possa ser substituído, é necessário verificar se ele foi alterado na memória cache;

Se isso não ocorreu, então o novo bloco pode ser escrito sobre o bloco antigo;

Caso contrário, se pelo menos uma operação de escrita foi feita sobre uma palavra dessa linha de memória cache, então a memória principal deve ser atualizada;

Dois problemas devem ser considerados:

- Alteração da memória principal por um dispositivo de E/S;
- Múltiplos processadores com múltiplas memórias cache;

Políticas de atualização da memória

- Para manter consistência entre *cache* e memória principal, uma das políticas de escrita em hierarquias de memória, *write-through* ou *write-back*, deve ser utilizada quando um dado é atualizado no *cache*.

Write-through

- Nesta estratégia, quando um ciclo de escrita ocorre para uma palavra, ela é escrita na *cache* e na memória principal simultaneamente.
- A principal desvantagem desta estratégia é que o ciclo de escrita passa a ser mais lento que o ciclo de leitura.
- No entanto, em programas típicos a proporção de operações de escrita à memória é pequena - de 5 a 34% do número total de referências à memória.

Write-back

- Nesta estratégia, quando um ciclo de escrita ocorre para uma palavra, ela é atualizada apenas no *cache*.
- A linha onde a palavra ocorre é marcada como alterada.
- Quando a linha for removida do cache, a linha toda é atualizada na memória principal.
- A desvantagem desta estratégia está no maior tráfego entre memória principal e *cache*, pois mesmo os itens não modificados são transferidos do *cache* para a memória.