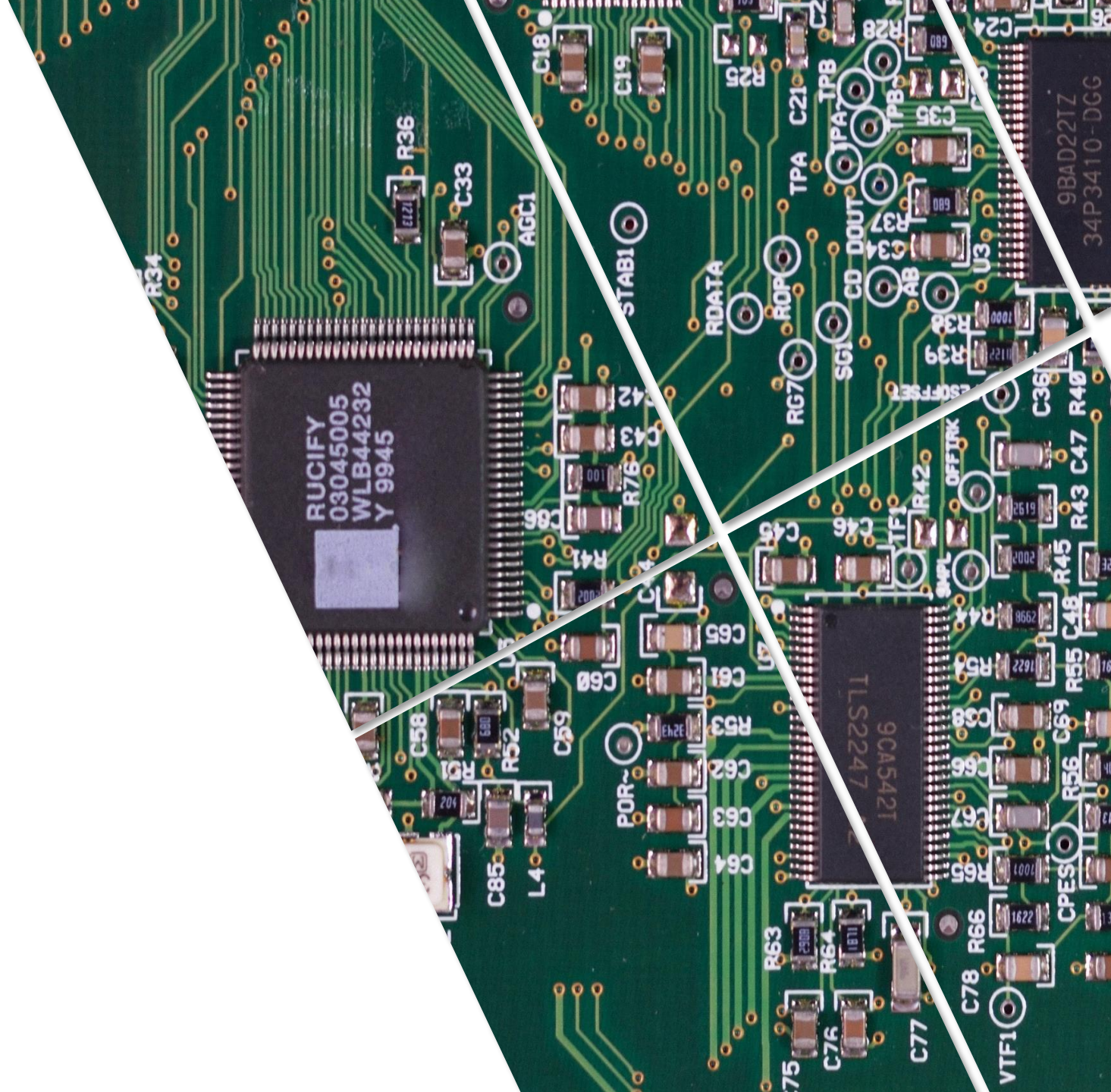
Performance em Sistemas Ciberfísicos

Aula 5 – CPU CACHE



Frank Coelho de Alcantara – 2023 -1

Dois Níveis de Hierarquia

Considere que nossa hierarquia de memória tem apenas dois níveis:

- Nível 2, onde estão todas as instruções e todos os dados;
- Nível 1, muito rápido, mas não comporta muitas instruções nem muitos dados.

CACHE EM SISTEMAS COMPUTACIONAIS

1. **Desafio:** Diferença de velocidade entre CPU e RAM.
2. **Solução:** Memória cache - memória rápida e temporária.
3. **Objetivo:** Armazenar dados e instruções frequentemente utilizados.
4. **Resultado:** Aumento no desempenho e eficiência do sistema.

Sir. Maurice Wilkes - st John’s College Cambridge

Como isso funciona

Se 95% de todos os acessos forem feitos na memória mais rápida ($1 \times 10^{-6} s$) e o restante na memória mais lenta ($100 \times 10^{-6} s$) o tempo médio de acesso será:

$$T_{med} = \frac{95 \times (1 \times 10^{-6}) + 5 \times (100 \times 10^{-6})}{100}$$

$$\boldsymbol{T_{med} = 5,95 \times 10^{-6}}$$

Cache

Uma quantidade pequena de memória, muito rápida:

- De acesso simples;
- De melhor qualidade;
- Próxima do processador;
- Controlada eficientemente.

Como o Cache Opera

A CPU requisita um dado, ou instrução da memória;

Verifica se o conteúdo necessário está no cache;

Se estiver, usa o que está no cache (rápido);

Se não estiver:

a) Lê o dado da memória principal, coloca no cache e acessa o cache novamente;

b) Lê o dado da memória principal e simultaneamente coloca no cache e passa para a cpu.

Como o Cache Opera

Os erros de cache (*misses*) são inevitáveis. Todos os fragmentos de código e dados serão carregados, no mínimo, uma vez.

Durante o tratamento de um *miss* o processador espera todo o processo de recuperação do dado necessário.

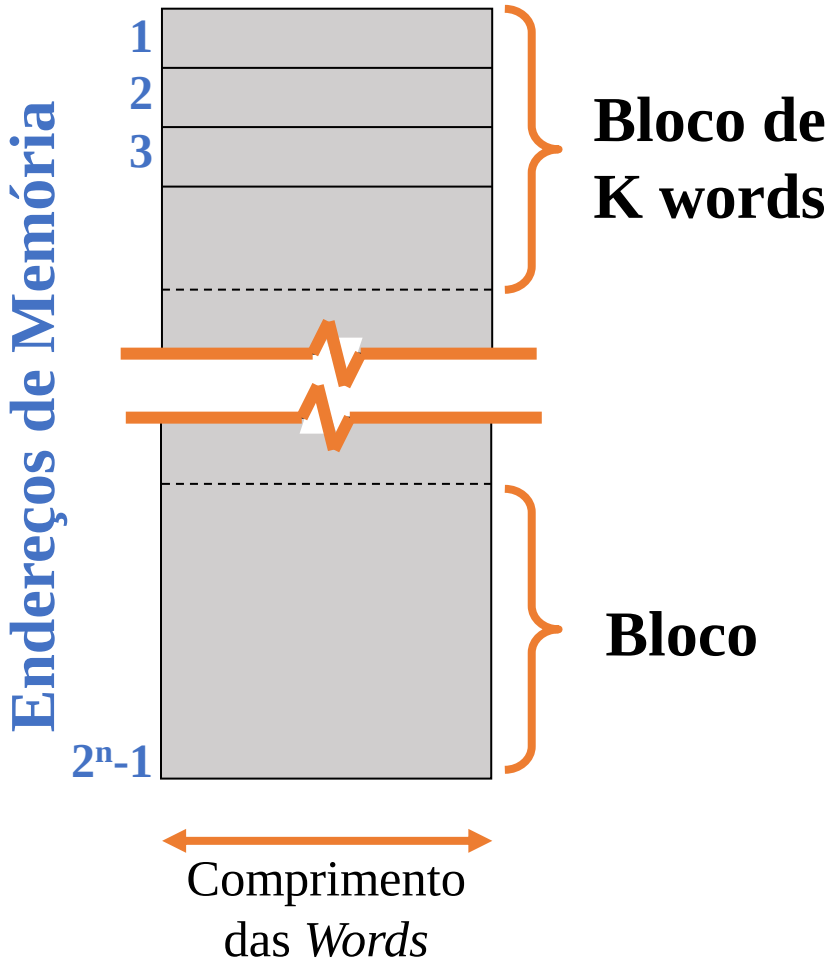
A potência consumida varia de forma linear de acordo com a frequência de *clock* e a tensão de alimentação da cpu.

Diminuir o número de *misses* além de tornar o programa mais eficiência, economiza energia.

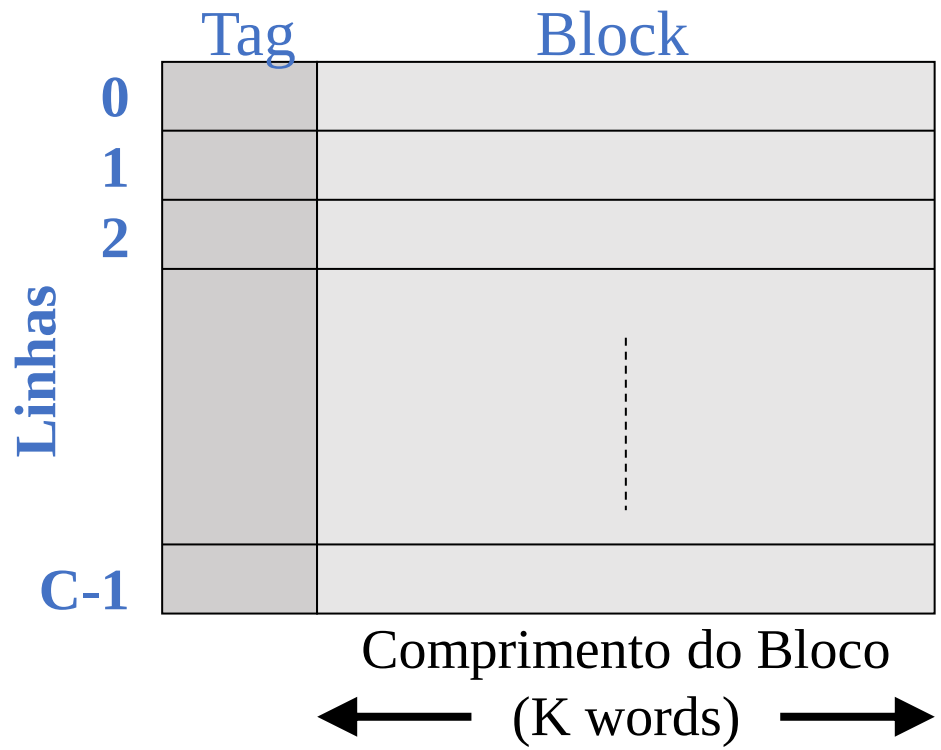
Estrutura do Cache

- O cache inclui um conjunto de *tags* para identificar o endereço do bloco da memória principal contido em uma linha de cache.
- Cada *word* na memória principal tem um endereco único de $n - bits$;
- Existem $M = 2n/k$ blocos de k *words* na memória principal
- O cache contém C linhas de k words mais uma tag que identifica de forma única e exclusive o bloco de k *words* da memória principal.

Memória Principal



Estrutura do Cache



Considerações sobre Design de Cache

- Tamanho;
- Função de Mapeamento;
- Algoritmo de Reposição;
- Política de Escrita
- Tamanho do Bloco
- Número de Caches

Considerações sobre Design de Cache

- Custo – Memória Cache é cara!
- Velocidade:
  - Com Caches maiores seremos mais rápidos (até um limite);
  - Circuitos de decodificação complexos diminuem a velocidade do cache;
  - Se for necessário um algoritmo para mapear a memória principal no cache isso será mais lento que o acesso direto a memória principal.

Considerações de Design

Um problema ocorre quando temos que encontrar um dado que pode, ou não estar no cache.

1. O Cache ser capaz de manter todos os endereços da memória principal.

2. Podemos fazer uma varredura sequencial do cache para ver se o dado já está lá.

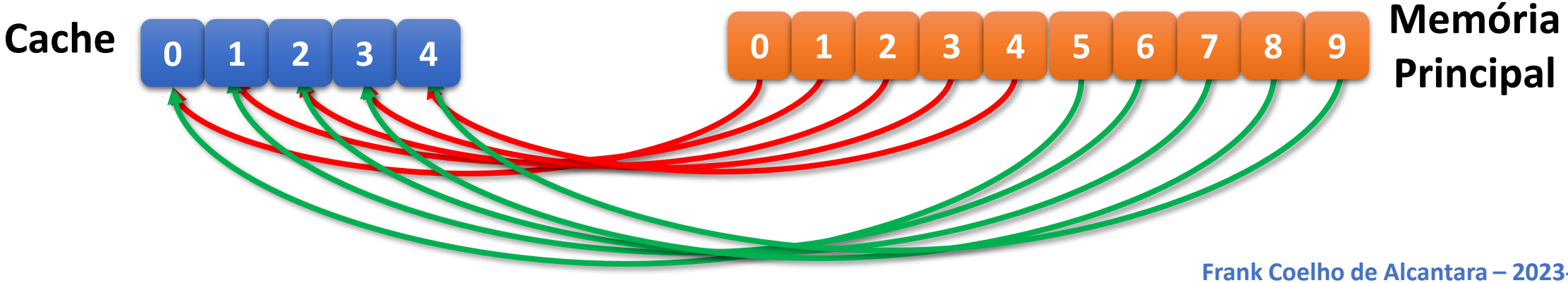
Talvez existam soluções mais práticas que não incluam aumentar o cache, ou varrer todo o cache a cada solicitação de dados.

Design de um Cache – Funções de Mapeamento

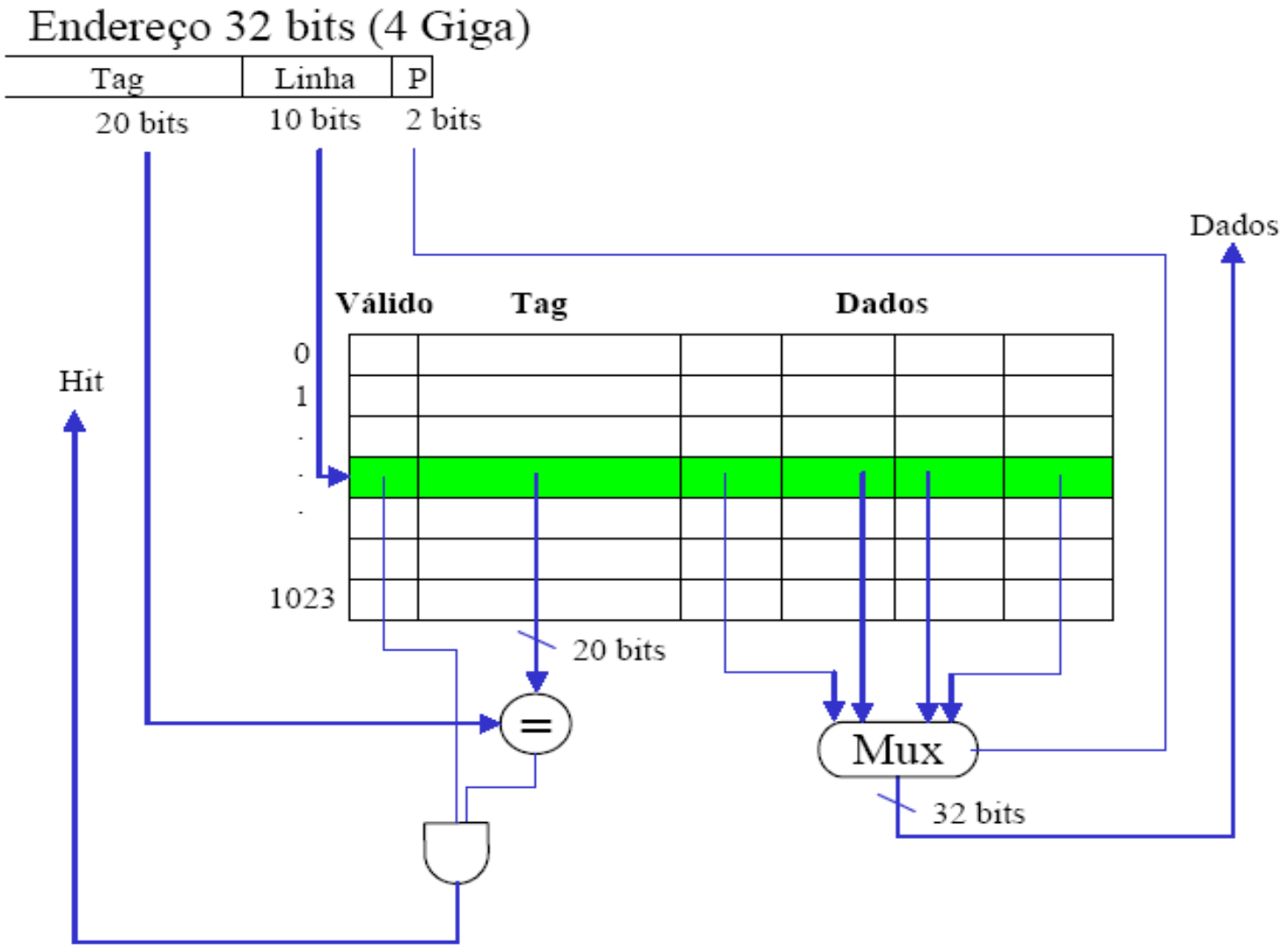
- A função de mapeamento é um dos métodos para localizar um endereço de memória dentro do Cache. Usamos uma função de mapeamento sempre que copiamos um bloco da memória para o Cache, ou quando precisamos encontrar um dado no Cache. Existem três tipos de Funções de Mapeamento:
  - Direto;
  - Associativo;
  - Associativo por Conjunto.

Mapeamento Direto

Por exemplo se o cache contem 5 blocos, o bloco 0 da memória principal mapeia para o bloco 0 do cache, o bloco 1 da memória principal para o bloco 1 do cache, o bloco 2 da memória principal para o bloco 2 do cache... O bloco 5 da memória principal para o bloco 0 do cache...



Exemplo Divisão de Blocos



- Exemplo da divisão de blocos em um Cache com $1024\ (2^{10})$ linhas e blocos com 4 palavras de 32 bits.
- Bit de validade e Tag

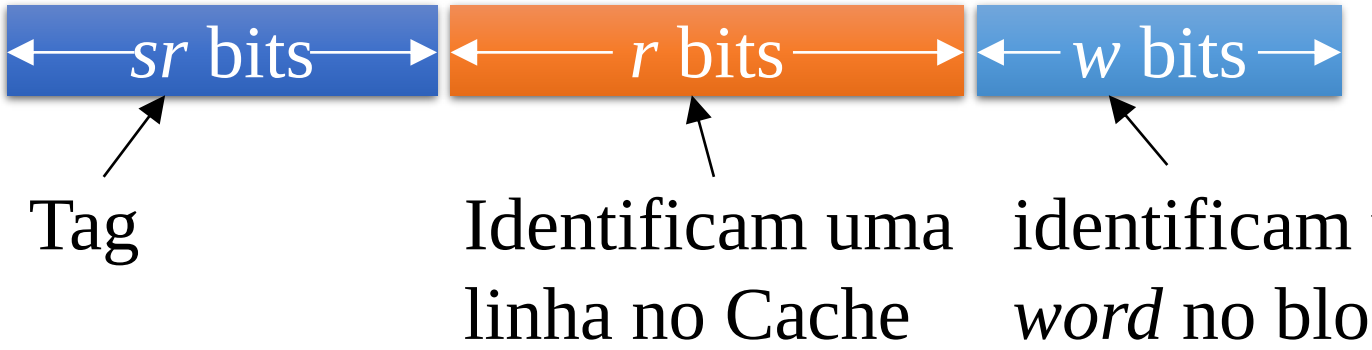
Como Acessar a Memória Principal

1. $Endereço\ no\ Cache = X\ módulo\ Y$ onde Y é o número de linhas do cache (tamanho da cache): Se tamanho do cache for uma potência de 2, módulo corresponderá a alguns dos bits menos significativos do endereço;
2. Se bit de validade da linha do Cache for válido, verificar *Tag*. Se Tag & Endereço no Cache é o endereço X, fim. Senão, se o bit de validade inválido ou se Tag & End Cache não produzir X. Miss;
3. Buscar dado no nível de memória inferior da hierarquia;
4. Colocar dado no Endereço da cache;
5. Efetuar operação (leitura/escrita).

Mapeamento Direto – Estrutura de Endereços

- Cada endereço da memória principal será dividido em três campos:
  - Um grupo de bits menos significantes, w, que identificará uma *word* em um bloco;
  - O resto dos bits irá identificar o bloco da memória. Estes bits são divididos em dois campos:
    - Os bits menos significantes, r, identificam a linha do Cache;
    - Os bits mais significantes, sr, identificam o bloco em uma linha de cache, a nossa *Tag*.

Mapeamento Direto – Estrutura de Endereços



- O tamanho de cada campo depende das características físicas da memória principal e do Cache.

Mapeamento Direto – Exemplo

Considere um espaço de endereçamento da Memória Principal de 1G Byte. Como fica a divisão de bits para um Cache de 2048 posições que trabalha com blocos de 8 palavras de 4 Bytes?

1. Espaço de endereçamento: 1 Giga, 2^{30} bytes.
2. Cache com 2048 linhas e blocos de 8 palavras.
3. Cada palavra com 4 Bytes, ou 32 bits.

Mapeamento Direto – Exemplo

1. Espaço de endereçamento: 1 Giga, 2^{30} bytes.
2. Cache com 2048 linhas e blocos de 8 palavras.
3. Cada palavra com 4 Bytes, ou 32 bits.

- Podemos calcular:
  1. Tamanho do Bloco: 8 palavras * 4 bytes/palavra = 32 Bytes = 2^5 Bytes.
  2. Agora podemos determinar os bits do endereço do Cache.

Mapeamento Direto – Exemplo

1. Bits de offset: Como temos 8 palavras por bloco, precisamos de $\log^2(8) = 3$ bits .
2. Bits de Linha: Como o Cache tem 2048 posições, precisamos de $\log_2(2048) = 11$ bits para as linhas.
3. Bits de tag: Os bits restantes são utilizados para a tag. Como temos um espaço de endereçamento de 2^{30} bytes e usamos 3 bits para o offset e 11 bits para o índice, ficamos com: 30 bits (endereço total) - 3 bits (offset) - 11 bits (índice) = 16 bits para a tag..

Mapeamento Direto – Exemplo

Efetivamente quantos dados temos neste Cache, ainda considerando palavras de 32 bits?

Linha da *Cache*

1 (validade)	16 (tag)	8*32 (bloco de dados)

(Bit de validade + Tag + Dados) / Dados

$$(1 + 16 + (8 * 32)) / (8 * 32)$$

$$273\ bits\ /\ 256\ bits$$

$$256 / 273 * 100\% = 93.77\%$$

Mapeamento Direto - Exercício

- (POSCOMP 2011 - 30) Um sistema de computador possui um mapa de memória de 4 Gbytes, usando endereçamento a byte e uma memória cache com organização de mapeamento direto. A cache tem capacidade de armazenar até 1.024 palavras de 32 bits provenientes do mapa de memória. Assuma que a cache sempre é escrita de forma atômica com quatro bytes vindos de um endereço de memória alinhado em uma fronteira de palavra de 32 bits, e que ela usa 1 bit de validade por linha de cache. Neste caso, as dimensões do rótulo (tag) da cache, do índice e o tamanho da cache são?

Mapeamento Direto – Exercício - Resposta

- Mapa de memória de 4 Gbytes, ou seja, 2^{32} bytes.
- Memória cache com mapeamento direto.
- Cache com capacidade de armazenar 1.024 palavras de 32 bits.
- Escrita atômica de 4 bytes (1 palavra de 32 bits).
- 1 bit de validade por linha de cache.

Mapeamento Direto – Exercício - Resposta

- Tamanho do bloco: como a escrita é atômica de 4 bytes (1 palavra de 32 bits), cada bloco terá 1 palavra, ou seja, 4 bytes = 2^2bytes.
- Número de linhas na cache: o Cache pode armazenar 1.024 palavras de 32 bits. Como cada bloco possui 1 palavra, o número de linhas na cache será de 1.024. Então, precisamos de $\log^2(1.024) = 10$ bits para o índice.
- Bits de offset: como cada bloco tem 2^2 bytes, precisamos de 2 bits para representar o offset dentro do bloco.
- Bits de tag: os bits restantes são utilizados para a tag. Como temos um espaço de endereçamento de 2^32 bytes e usamos 2 bits para o offset e 10 bits para o índice, ficamos com: 32 bits (endereço total) - 2 bits (offset) - 10 bits (índice) = 20 bits para a tag.