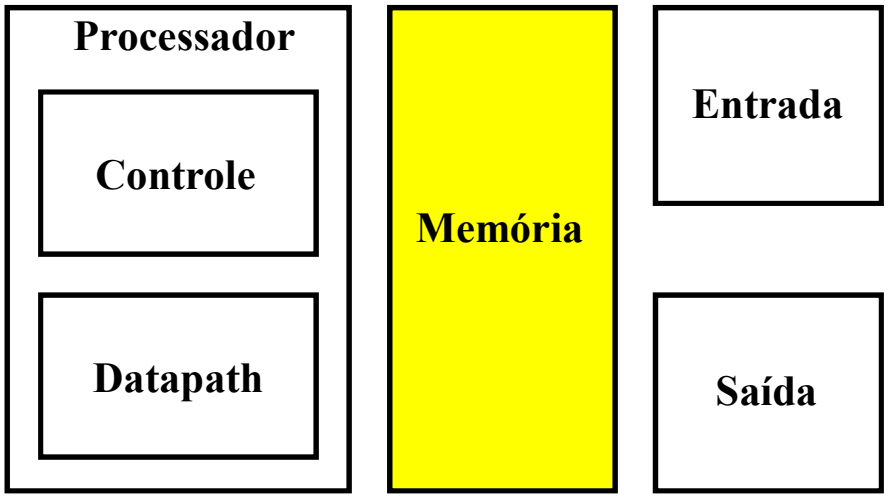
Cache: princípios e mapeamento



Princípios básicos

- **Programador quer memória grande e rápida**
  - **As grandes são lentas.**
  - **As rápidas são caras.**

SRAM é 20 a 140 vezes mais rápida que DRAM

SRAM é 25 a 100 vezes mais cara que DRAM

Tecnologia	Tempo de acesso	$ por Gbyte (2012)
SRAM	0,5-2,5 ns	$500-$1000
DRAM	50-70 ns	$10-$20
FLASH	5.000-50.000 ns	$0,75-$1,00
HD	5.000.000-20.000.000 ns	$0,05-$0,10

Princípios básicos

- **Programador quer memória grande e rápida**
  - **As grandes são lentas.**
  - **As rápidas são caras.**

SRAM é 2M a 40M vezes mais rápida que HD

SRAM é 5000 a 20000 vezes mais cara que HD

Tecnologia	Tempo de acesso	$ por Gbyte (2012)
SRAM	0,5-2,5 ns	$500-$1000
DRAM	50-70 ns	$10-$20
FLASH	5.000-50.000 ns	$0,75-$1,00
HD	5.000.000-20.000.000 ns	$0,05-$0,10

Princípios básicos

- **Programador quer memória grande e rápida**
  - **As grandes são lentas.**
  - **As rápidas são caras.**

SRAM é 2K a 100K vezes mais rápida que FLASH

SRAM é 500 a 1300 vezes mais cara que FLASH

Tecnologia	Tempo de acesso	$ por Gbyte (2012)
SRAM	0,5-2,5 ns	$500-$1000
DRAM	50-70 ns	$10-$20
FLASH	5.000-50.000 ns	$0,75-$1,00
HD	5.000.000-20.000.000 ns	$0,05-$0,10

Princípios básicos

- **Dimensionamento:**
  - **Pouca memória SRAM (centenas de KB)**
  - **Mais memória DRAM (centenas de MB)**
  - **Muita memória em HD (centenas de GB)**

Princípios básicos

- **Critério de uso da memória:**
  - **Itens a serem usados mais frequentemente**
  - **Mantidos na memória mais rápida.**
- **Problema:**
  - **Como distinguir os itens a serem mais usados ?**

Princípio da Localidade

- **Temporal**:

  "Um item tende a ser novamente referenciado em breve."

  - **Laços**
  - **Procedimentos**

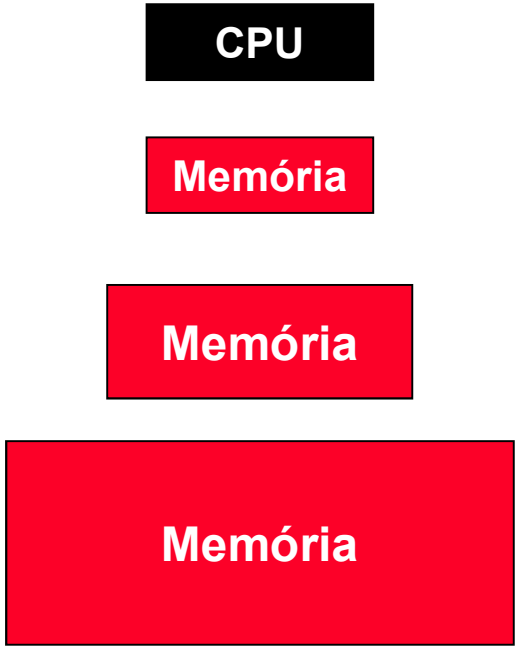
- **Espacial**:

  "Itens com endereços próximos a um item referenciado tendem a ser referenciados em breve."

  - **Instruções**
  - **Estruturas c/ elementos contíguos (arranjo, "record", etc.)**

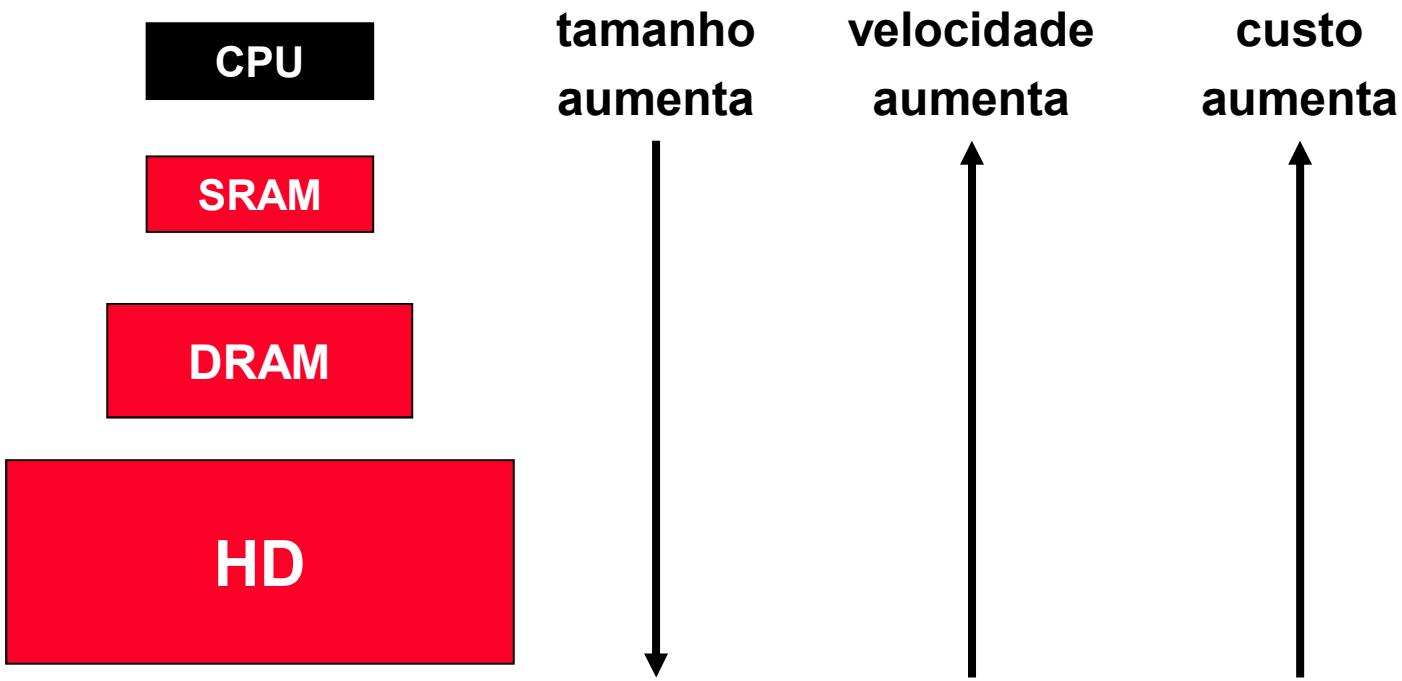
Hierarquia de memória

- **Múltiplos níveis de memória**
  - **Diferentes custos, velocidades e tamanhos**
- **Ilusão de única memória grande e rápida**



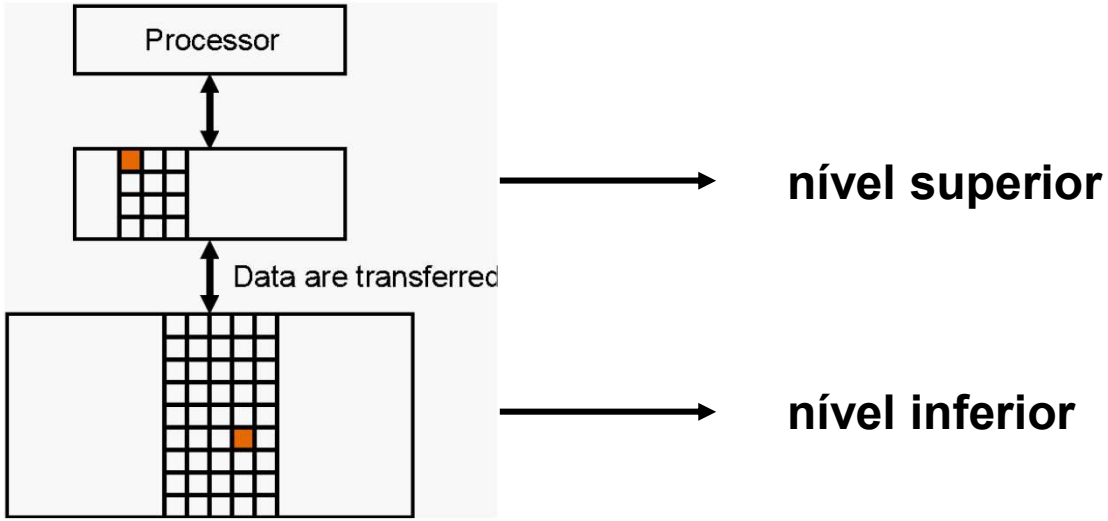
Hierarquia de memória

- **Múltiplos níveis de memória**
  - **Diferentes custos, velocidades e tamanhos**
- **Ilusão de única memória grande e rápida**



Hierarquia de memória

- **Bloco:**
  - **Quantum de informação (presente/ausente)**



- **Dado requisitado pela CPU...**
  - **Presente no nível superior → sucesso ou acerto ("hit")**
  - **Ausente no nível superior → fracasso ou falta ("miss")**

Hierarquia de memória

- **Explora localidade temporal**
  - **Itens mais recentemente referenciados são mantidos nos níveis superiores**
- **Explora localidade espacial**
  - **Palavras contíguas na memória são movidas para níveis superiores**

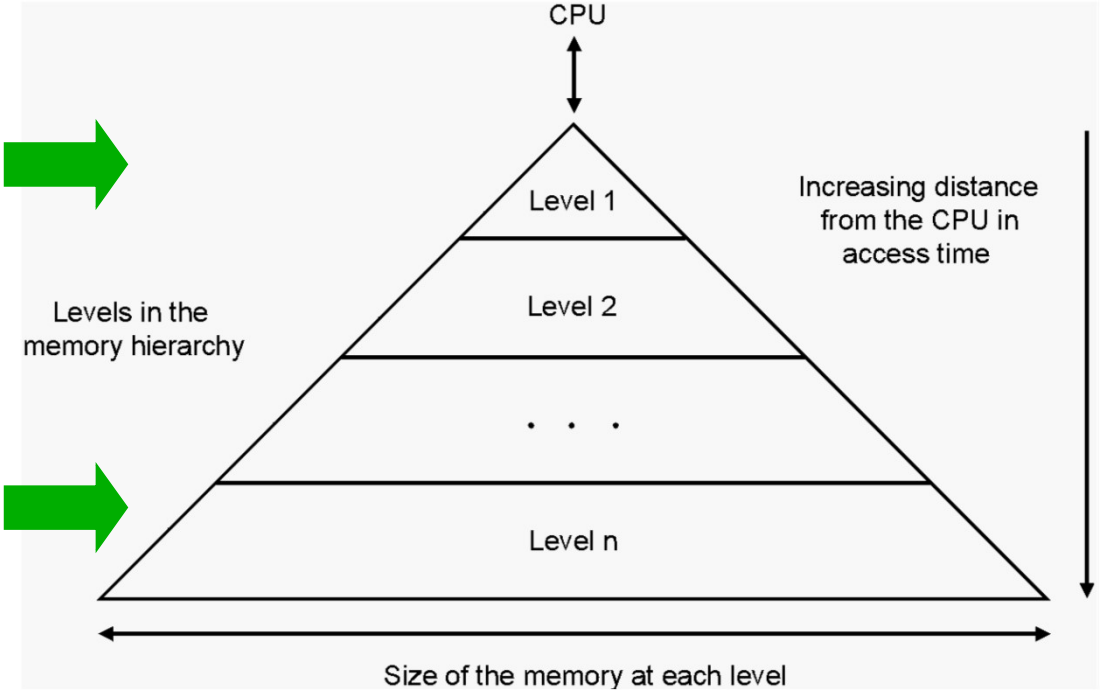
Métricas de Desempenho

- Taxa de sucesso ou de acertos (“hit rate”)
- Taxa de fracasso ou de faltas (“miss rate”)
- Consequência: h = 1 - m
- Tempo de acerto (“hit time”)
  - tempo p/ acessar nível superior
- Penalidade de falta (“miss penalty”)
  - tempo para substituir bloco no nível superior
  - tempo de acesso ao nível inferior é dominante

Hierarquia de memória

Memórias pequenas perto da CPU (itens mais usados acessados rapidamente)

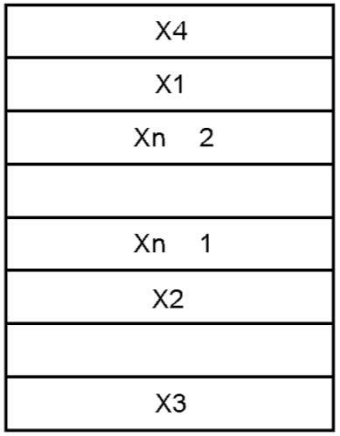
Memórias grandes longe da CPU (itens raramente acessados na memória mais lenta)



- **Taxa de acertos↑: $tempo_{acesso}$→ tempo memória rápida**
- **Espaço de endereçamento → o da memória grande**
- **Ilusão de única memória rápida e grande!**

Cache: a ideia básica

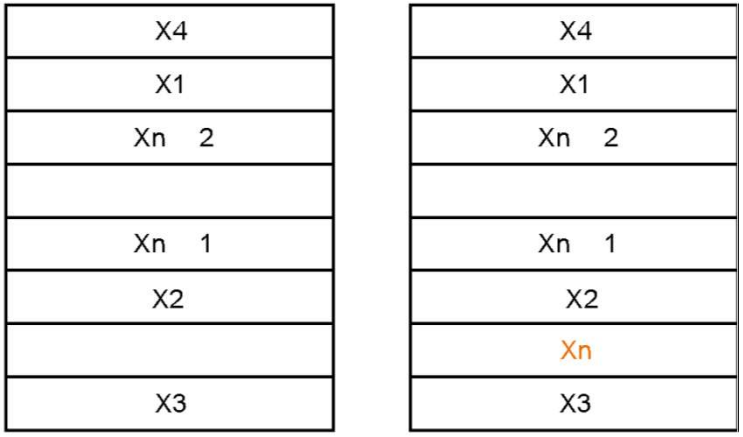
- **Hierarquia: CPU ↔ cache ↔ memória**
- **Referências recentes: X1, ... , Xn**
- **CPU requisita Xn**
  - **Ausente → falta → cache é atualizada**



a. Before the reference to Xn

Cache: a ideia básica

- **Hierarquia: CPU ↔ cache ↔ memória**
- **Referências recentes: X1, ... , Xn**
- **CPU requisita Xn**
  - **Ausente → falta→ cache é atualizada**

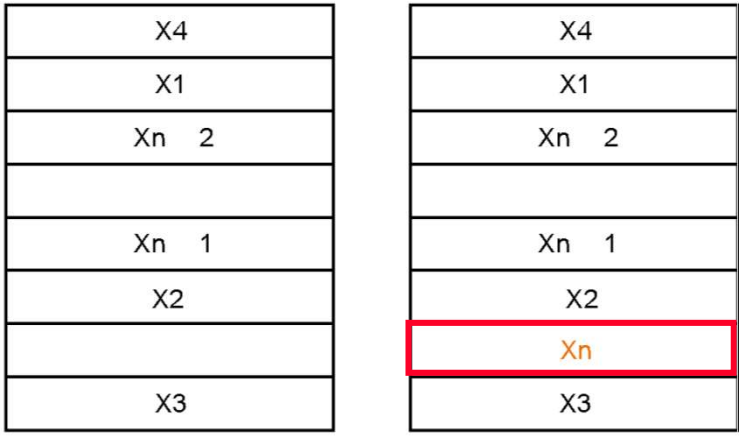


a. Before the reference to Xn

b. After the reference to Xn

Cache: a ideia básica

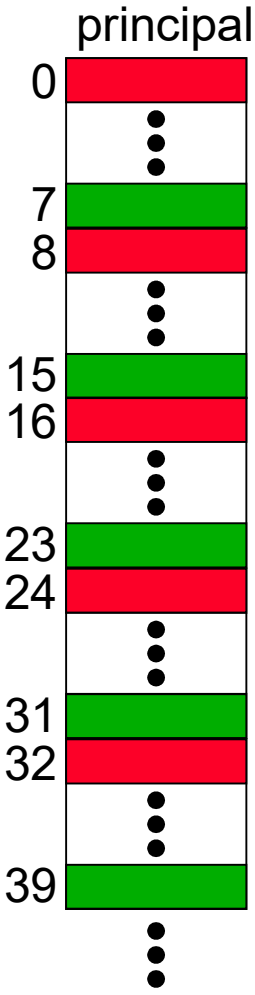
- **Como se sabe se um item está na cache ?**
- **Se está na cache, como encontrá-lo ?**

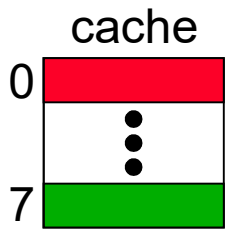


a. Before the reference to Xn

b. After the reference to Xn

Mapear para Encontrar





MP		cache
0	→	0
1	→	1
7	→	7
8	→	0
9	→	1
15	→	7
E	→	**E mod 8**

1	0	1	0	1

$= 21_{dec}$

21 | 8

5 2

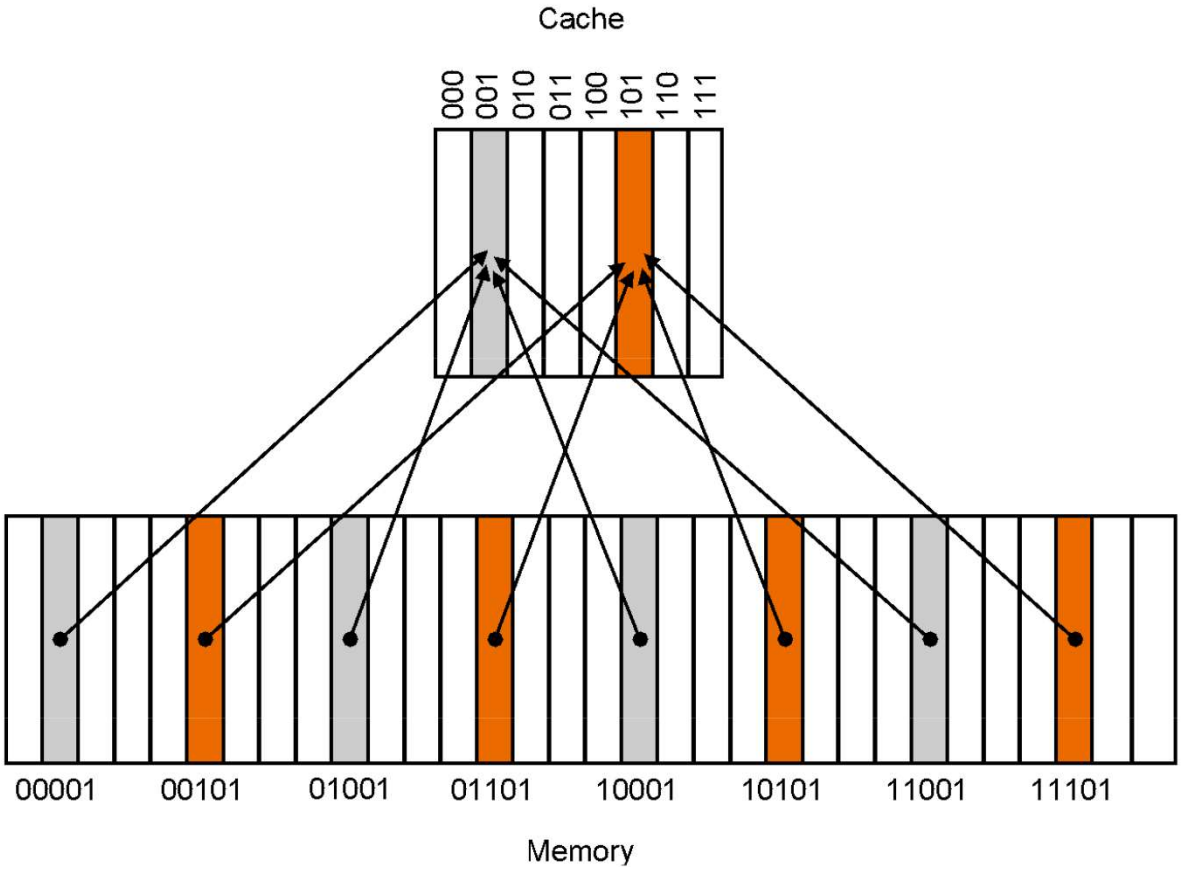
0	0	0	1	0	1	0	1

⌊21/8⌋ **21 mod 8**

Conclusão: LSBs do endereço indicam mapeamento.

Mapeamento Direto

- **Cada posição de memória é mapeada para uma <u>única</u> posição da cache**



Mapeamento

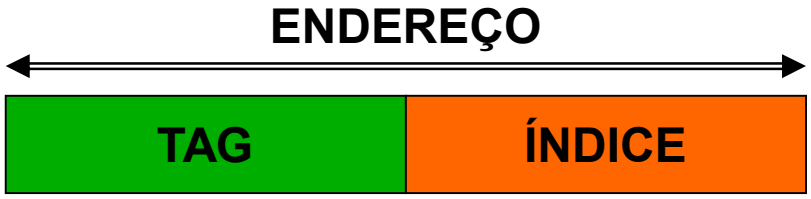
E modulo N

E: endereço do bloco

N: número de blocos na cache

Consequências do Mapeamento

- **Se $N = 2^n$:**
  - **Posição na cache indexada pelos n LSBs do endereço de memória**
- **Conteúdo de cada posição da cache pode vir de diferentes posições da memória.**
- **Como ter certeza se item endereçado é o item armazenado ?**
  - **Etiquetas ou "tags" → MSBs do endereço**



Inicialização da Cache

- **No início, cache vazia**
  - **"Vazia" = conteúdo inválido ("lixo")**
- **Como inicializar a cache ?**
  - **Reset de todos os bits da cache ?**
    - » **Solução cara e inútil**
  - **Bit de validade.**
    - » **1: tag válido**
    - » **0: tag inválido**
- **Cada posição da cache tem vários campos:**
  - **Tag, dado(s), bit de validade.**

Acessando a Cache

Estado inicial

Indice	V	Tag	Dados
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Próximo: 10110

Após fracasso em 10110

Indice	V	Tag	Dados
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	S	10	MEM(10110)
111	N		

Próximo: 11010

Acessando a Cache

Após fracasso em 11010

Indice	V	Tag	Dados
000	N		
001	N		
010	S	11	MEM(11010)
011	N		
100	N		
101	N		
110	S	10	MEM(10110)
111	N		

Próximo: 10000

Após fracasso em 10000

Indice	V	Tag	Dados
000	S	10	MEM(10000)
001	N		
010	S	11	MEM(11010)
011	N		
100	N		
101	N		
110	S	10	MEM(10110)
111	N		

Próximo: 00011

Acessando a Cache

Após fracasso em 00011

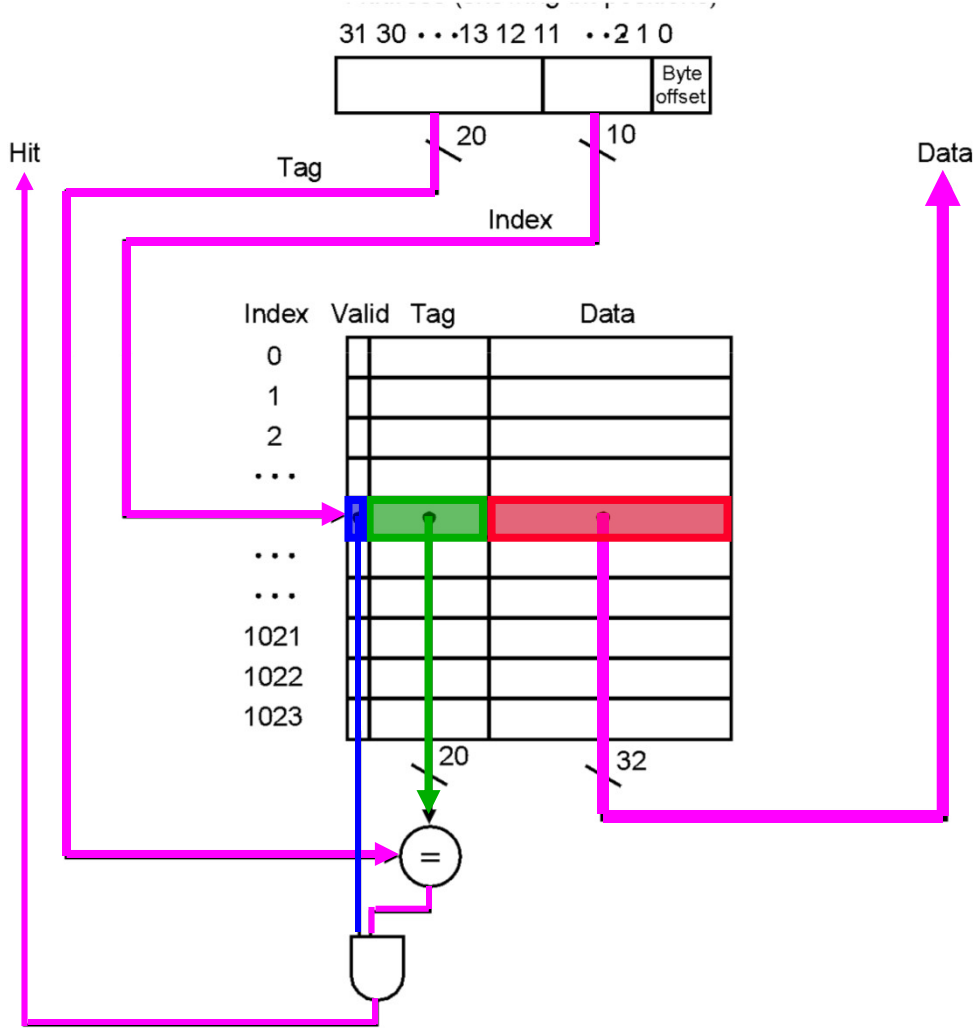
Indice	V	Tag	Dados
000	S	10	MEM(10000)
001	N		
010	S	11	MEM(11010)
011	S	00	MEM(00011)
100	N		
101	N		
110	S	10	MEM(10110)
111	N		

Próximo: 10010

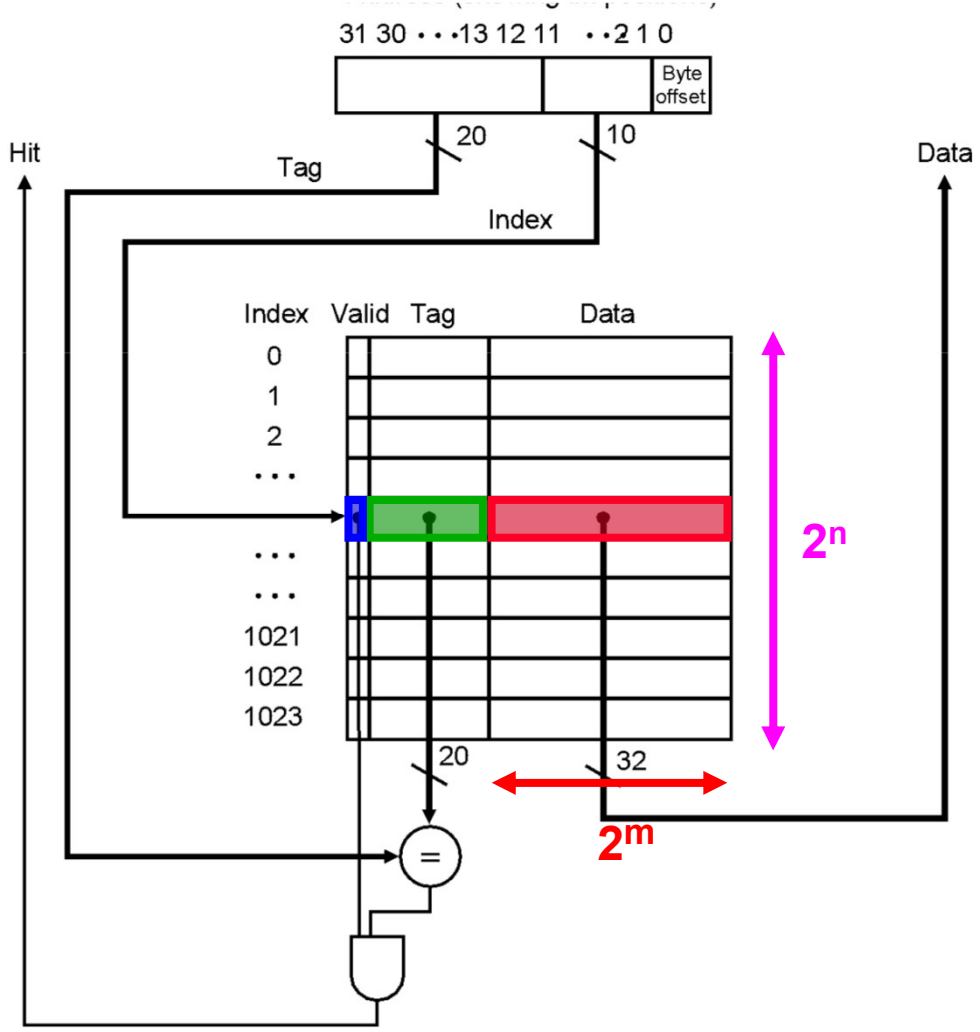
Após fracasso em 10010

Indice	V	Tag	Dados
000	S	10	MEM(10000)
001	N		
010	S	10	MEM(10010)
011	S	00	MEM(00011)
100	N		
101	N		
110	S	10	MEM(10110)
111	N		

Mapeamento direto: organização

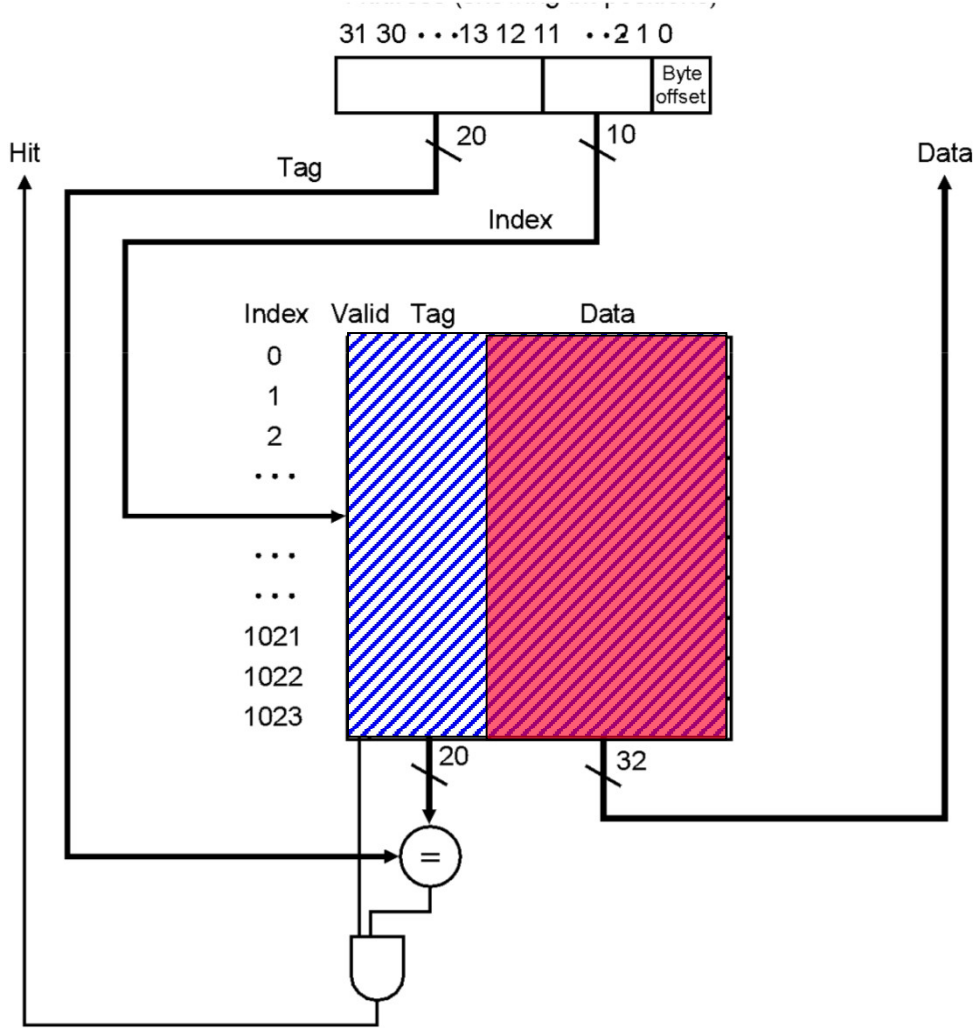


Mapeamento direto: organização



- **Num. bits da cache**
  - **endereço: 32 bits**
  - **tamanho: 2^n blocos**
  - **bloco = 2^m palavras**
  - **bloco = 2^{m+2} bytes**
  - **tag = 32 - (n+m+2)**
  - **bit de validade**
  - **$2^n \times (2^m \times 32 + 32 - (n+m+2) + 1)$**
  - **$2^n \times (2^m \times 32 + (32-n-m-2) + 1)$**
- **Exemplo**
  - **cache: 16KB (dados)**
  - **bloco = 4 = 2^2 palavras**
  - **16KB = 4K = 2^{12} palavras = 2^{10} blocos**
  - **$2^{10} \times (2^2 \times 32 + 32 - (10+2+2) + 1)$**
  - **# bits = $2^{10} \times 147$ = 147Kbits**
  - **147Kbits/128Kbits = 1,19**

Mapeamento direto: organização



- **Exemplo**
  - **cache: 16KB (dados)**
  - **bloco = 4 = 2^2 palavras**
  - **16KB = 4K = 2^{12} palavras = 2^{10} blocos**
  - **$2^{10} \times (2^2 \times 32 + 32 - (10+2+2) + 1)$**
  - **# bits = $2^{10} \times 147$ = 147Kbits**
  - **147Kbits/128Kbits = 1,19**