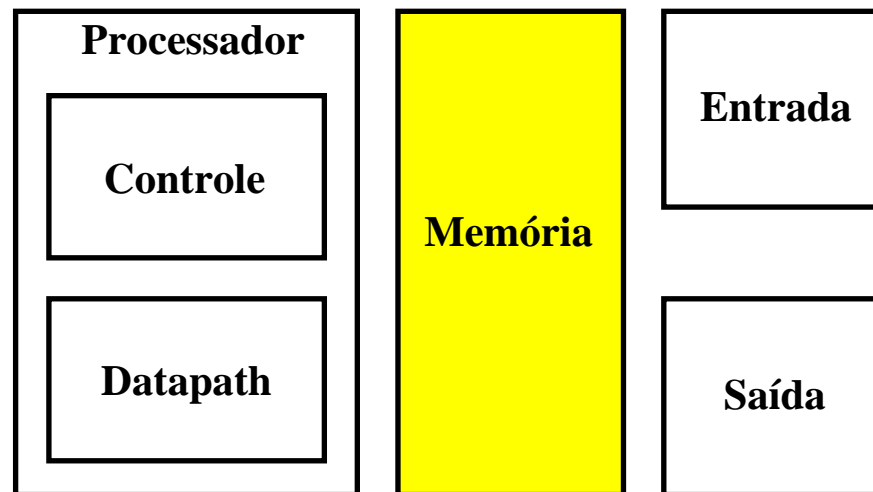


Cache: princípios e mapeamento



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 26 a 250
vezes mais cara
que DRAM

Tecnologia	Tempo de acesso	\$ por Gbyte (2008)
SRAM	0,5-2,5ns	\$5000-\$2000
DRAM	50-70ns	\$20-\$75
Disco magnético	5-20 milhões ns	\$0,20-\$2,00

Princípios básicos

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

SRAM é 10M a 40M
vezes mais rápida
que HD

SRAM é 1000 a
25000 vezes mais
cara que HD

Tecnologia	Tempo de acesso	\$ por Gbyte (2004)
SRAM	0,5-2,5ns	\$5000-\$2000
DRAM	50-70ns	\$20-\$75
Disco magnético	5-20 milhões ns	\$0,20-\$2,00

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 freqüentemente
 - 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

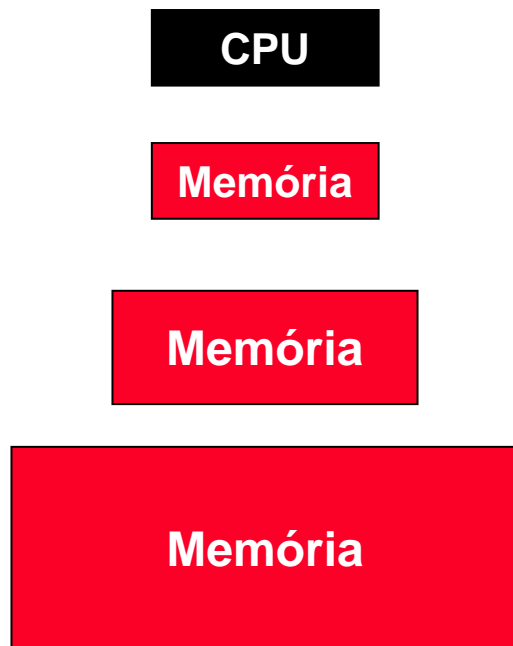
- **Espacial:**

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

- Instruções
- Estruturas c/ elementos contíguos (arranjo, “record”, etc.)

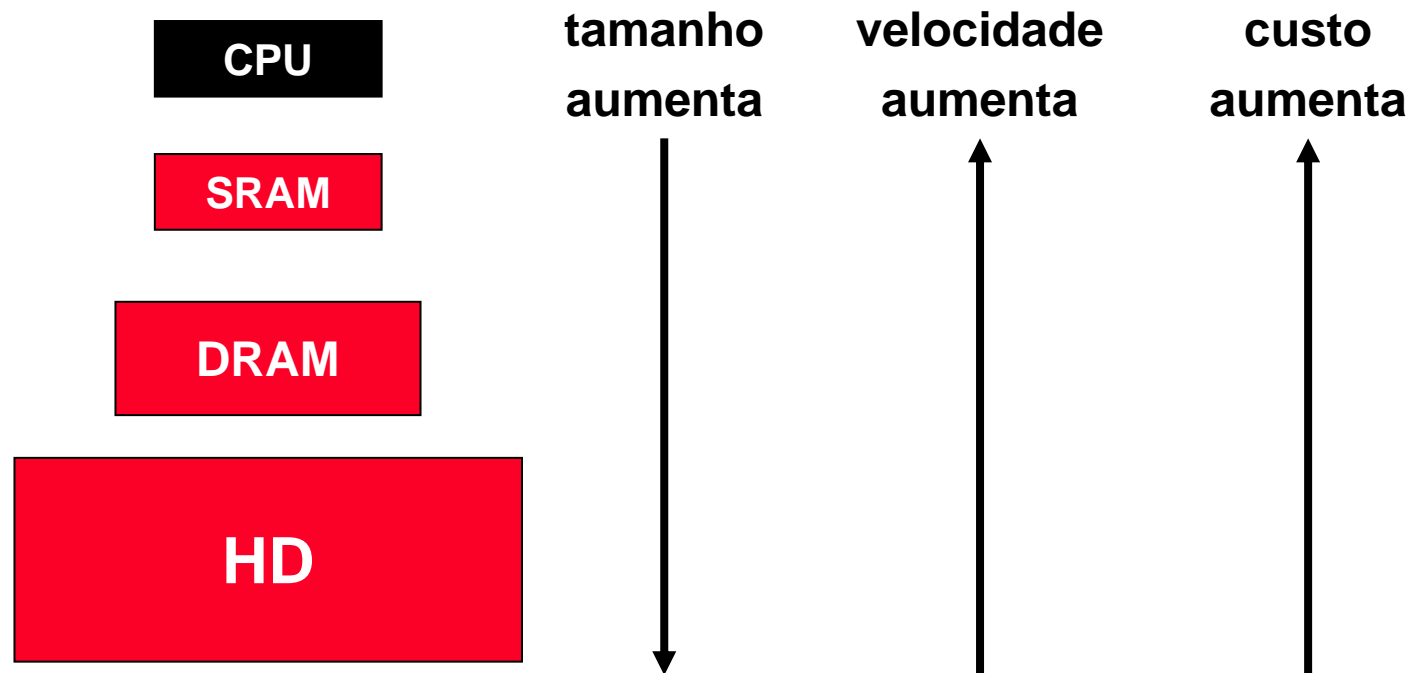
Hierarquia de memória

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



Hierarquia de memória

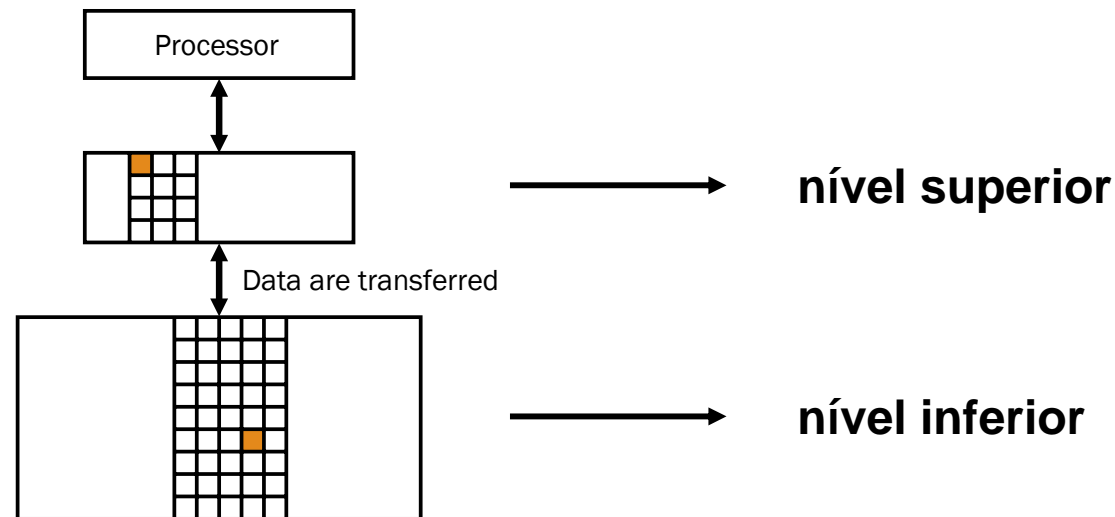
- **Múltiplos níveis de memória**
 - Diferentes custos, velocidades e tamanhos
- **Ilusão de única memória grande e rápida**



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 falha** (“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

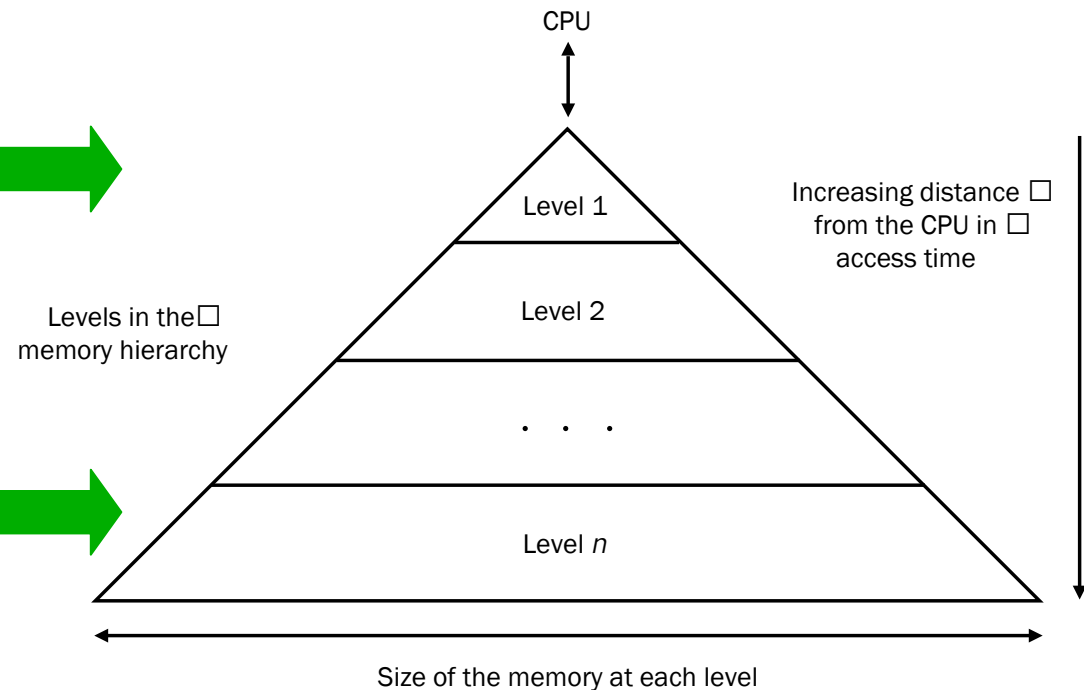
- Taxa de sucesso ou de acertos (“**h**it rate”)
- Taxa de fracasso ou de falhas (“**m**iss rate”)
- Consequência: $h = 1 - m$
- Tempo de sucesso (“hit time”)
 - tempo p/ acessar nível superior
- Penalidade de fracasso (“miss penalty”)
 - tempo para substituir bloco no nível superior
 - tempo de acesso 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 sucesso[↑]: tempo_{acesso} → tempo_{memória rápida}
- Espaço de endereçamento → o da memória grande
- Ilusão de única memória rápida e grande!

Cache: a idéia básica

- Hierarquia: CPU \leftrightarrow **cache** \leftrightarrow memória
- Referências recentes: X_1, \dots, X_n
- CPU requisita X_n
 - Ausente \rightarrow fracasso \rightarrow cache é atualizada

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

a. Before the reference to X_n

Cache: a idéia básica

- Hierarquia: CPU \leftrightarrow **cache** \leftrightarrow memória
- Referências recentes: X_1, \dots, X_n
- CPU requisita X_n
 - Ausente \rightarrow fracasso \rightarrow cache é atualizada

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

a. Before the reference to X_n

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_n
X_3

b. After the reference to X_n

Cache: a idéia básica

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

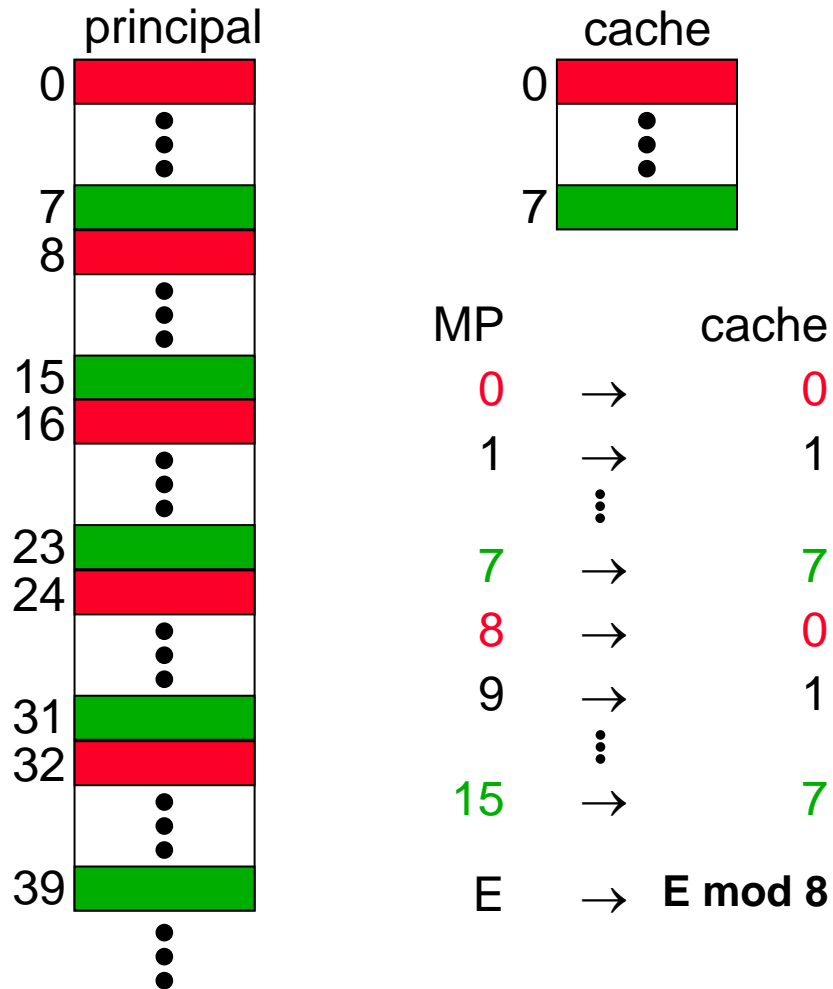
X4
X1
X $n-2$
X $n-1$
X2
X3

a. Before the reference to X n

X4
X1
X $n-2$
X $n-1$
X2
X n
X3

b. After the reference to X n

Mapear para Encontrar



$$\begin{array}{|c|c|c|c|c|} \hline 1 & 0 & 1 & 0 & 1 \\ \hline \end{array} = 21_{\text{dec}}$$

$$\begin{array}{r} 21 \overline{) 8} \\ \underline{5} \\ 2 \end{array}$$

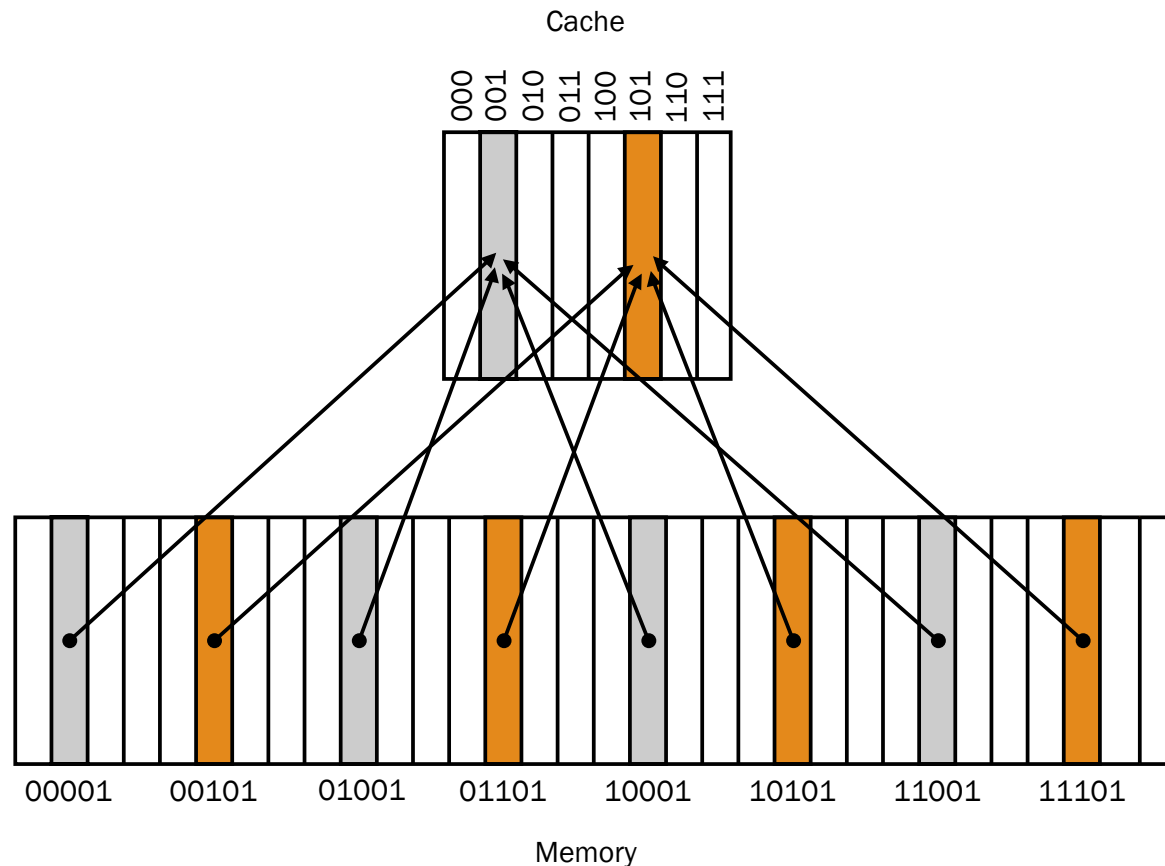
$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline \end{array}$$

$\underbrace{\hspace{1.5cm}}_{\lfloor 21/8 \rfloor} \quad \underbrace{\hspace{1.5cm}}_{21 \bmod 8}$

Conclusão: LSBs do endereço indicam mapeamento.

Mapeamento Direto

- Cada posição de memória é mapeada para uma única posição da cache



Mapeamento

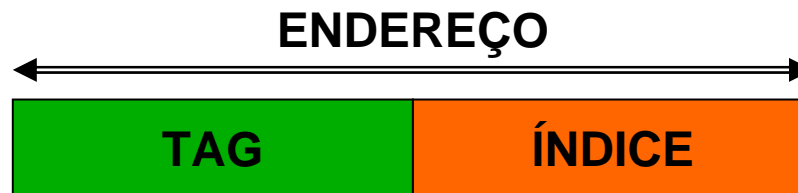
E modulo **N**

E: endereço do
bloco

N: número de
blocos na cache

Conseqüências 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		

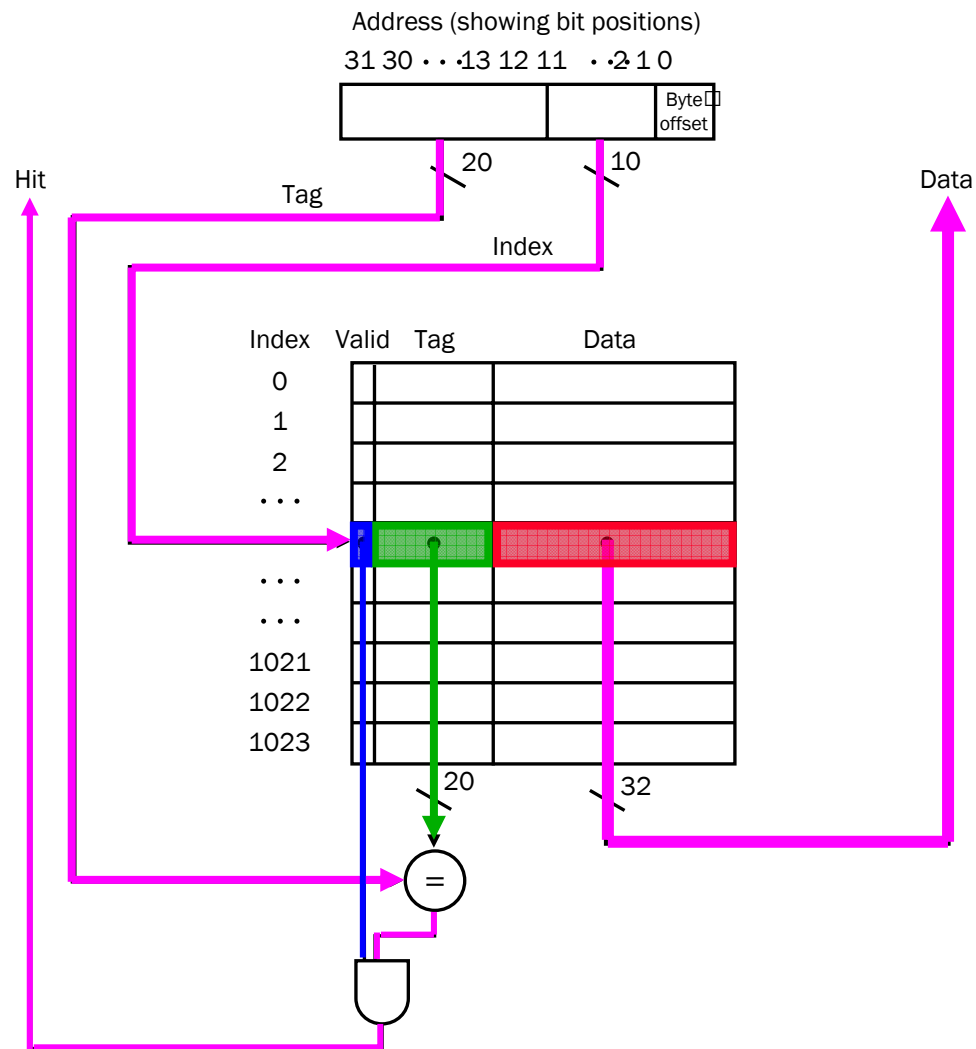
Após fracasso em 10010

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		

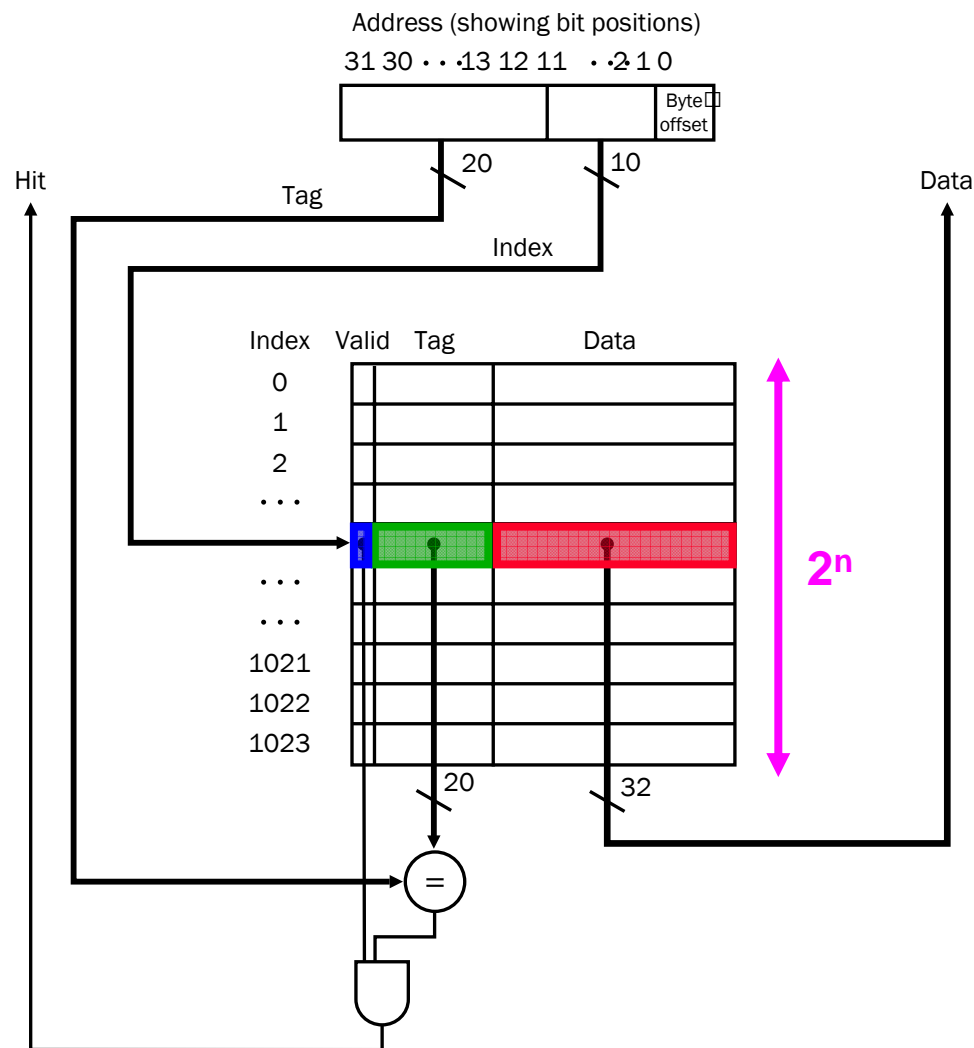


Próximo: 10010

Mapeamento direto: organização

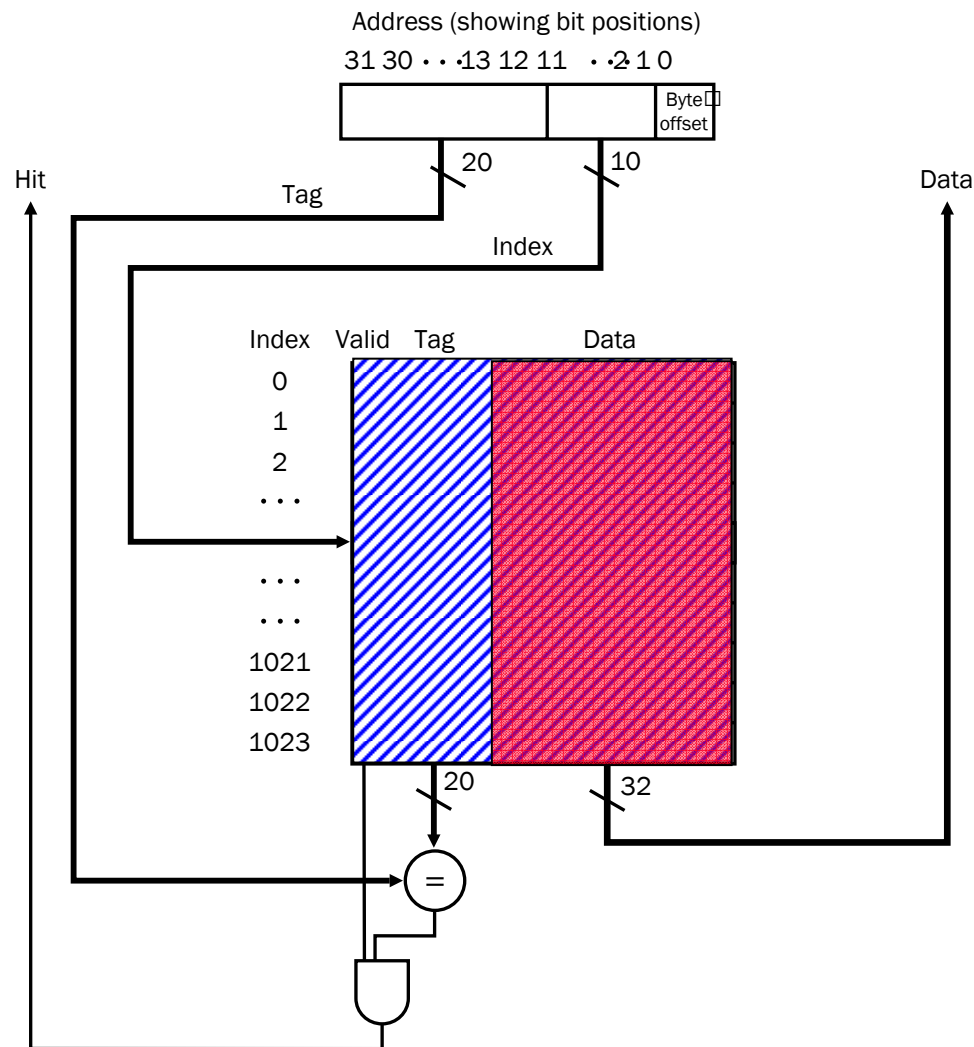


Mapeamento direto: organização



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

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 = $2^{10} \times 147 = 147\text{Kbits}$
 - **147Kbits/128Kbits = 1,19**