Hierarquia de Memória

No projeto de um sistema digital, deve-se ter em mente que *hardware* menor geralmente é mais rápido do que *hardware* maior.

A propagação do sinal é uma das principais causas de atrasos. No caso da memória, quanto maior mais atraso de sinal e mais níveis para decodificar endereços.

Na maioria das tecnologias, pode-se obter memórias menores que são mais rápidas do que memórias maiores.

As memórias mais rápidas estão geralmente disponíveis em números menores de bits por integrado e custam substancialmente mais por byte.

O aumento da largura de banda da memória e a diminuição do tempo de acesso à memória são também importantes para o desempenho do sistema.

O princípio da localidade se baseia no fato de que, num intervalo virtual de tempo, os endereços virtuais gerados por um programa tendem a ficar restritos a pequenos conjuntos do seu espaço. Isto se deve a iterações, seqüenciamento das instruções e estruturas em bloco.

Assim sendo, deveríamos manter os itens mais recentemente utilizados na memória mais rápida e o mais próximo possível da CPU.

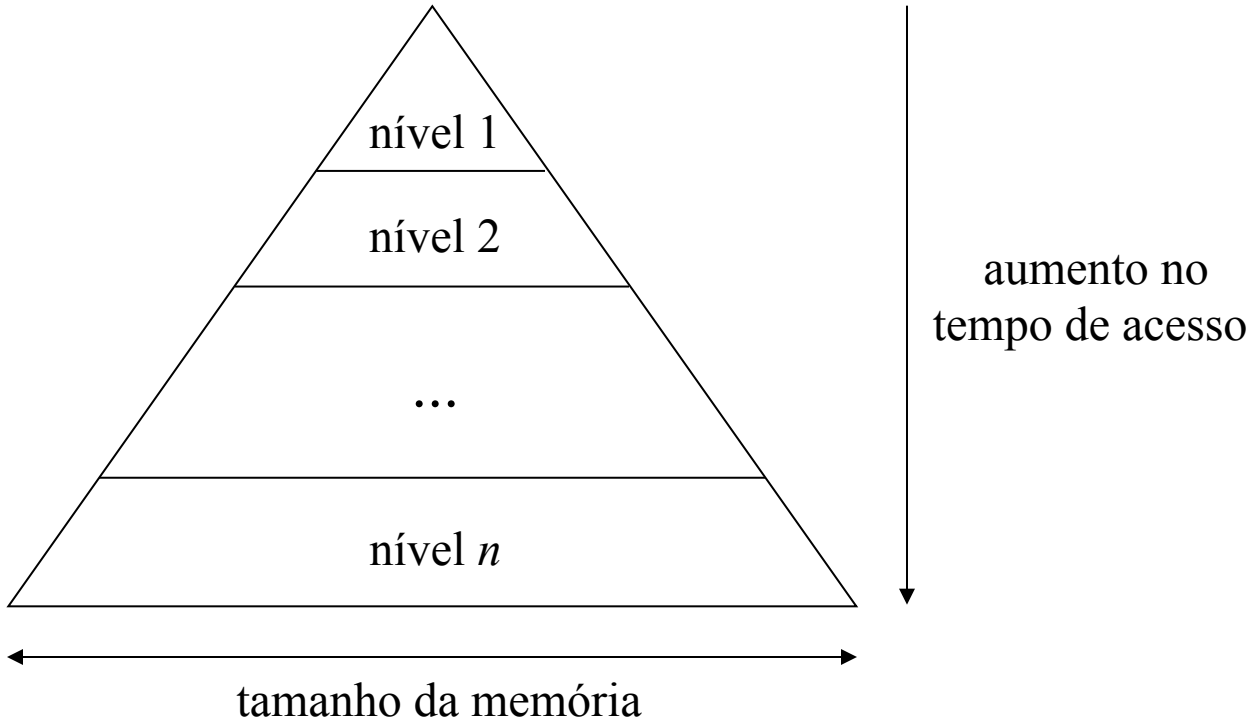
Há três componentes do princípio da localidade, que coexistem num processo ativo:

- localidade temporal: há uma tendência a que um processo faça referências futuras a posições feitas recentemente;

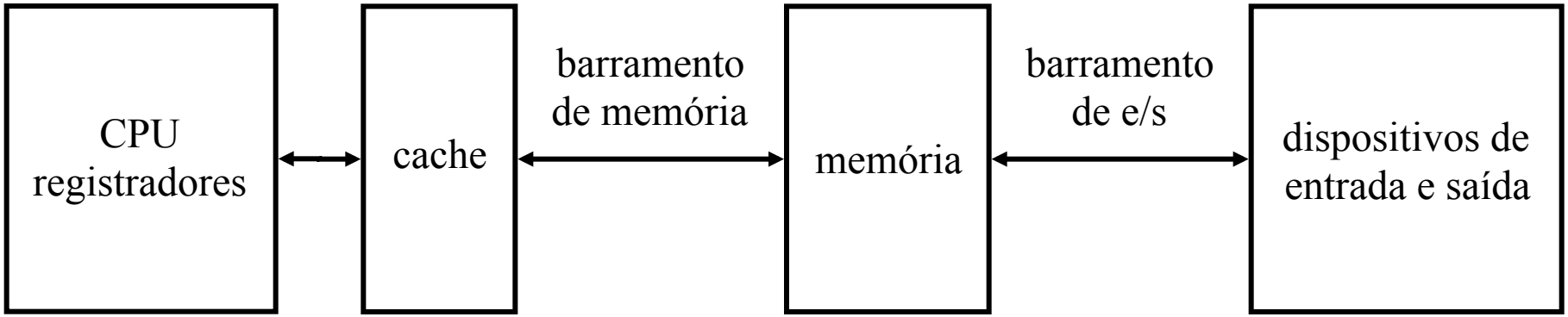
- localidade espacial: há uma tendência a que um processo faça referências a posições na vizinhaça da última referência;

- localidade seqüencial: há uma tendência a que um processo faça referência à posição seguinte à atual.

Hierarquia de memória consiste em diferentes níveis de memória, associados a diferentes velocidades de acesso e tamanhos.



Os níveis da hierarquia são subconjuntos uns dos outros. Todos os dados encontrados em um nível também são encontrados no nível abaixo dele e assim sucessivamente.



< 1 KB	< 16 MB	< 16 GB	> 100 GB
0,25-0,5 ns	0,5-25 ns	80-250 ns	5 ms
20000-100000 MB/s	5000-10000 MB/s	1000-5000 MB/s	20-150 MB/s
CMOS	CMOS SRAM	CMOS DRAM	disco magnético
compilador	*hardware*	sistema operacional	sistema operacional e usuário

Taxa de acerto (*hit ratio - h*) consiste na proporção dos acessos à memória encontrados em um nível da hierarquia.

Taxa de falha (*miss ratio - m*) consiste na proporção dos acessos à memória não encontrados em um nível da hierarquia.

$$m = 1 - h$$

Ciclos de parada por memória (*ncp*) consiste do número de ciclos que a CPU espera por um acesso à memória, quando ocorre uma falha de acesso.

O tempo de execução da CPU deve, então, levar em conta o número de ciclos de parada por memória:

$$tc = (ncc + ncp) \times cc$$

O número de ciclos de parada depende do número de erros (*ne*) e do custo por erro ou penalidade de erro (*pe*):

$$ncp = ne \times pe$$

$$ncp = ni \times \frac{erros}{instrução} \times pe \quad \Rightarrow \quad ncp = ni \times \frac{acessosàmemória}{instrução} \times m \times pe$$

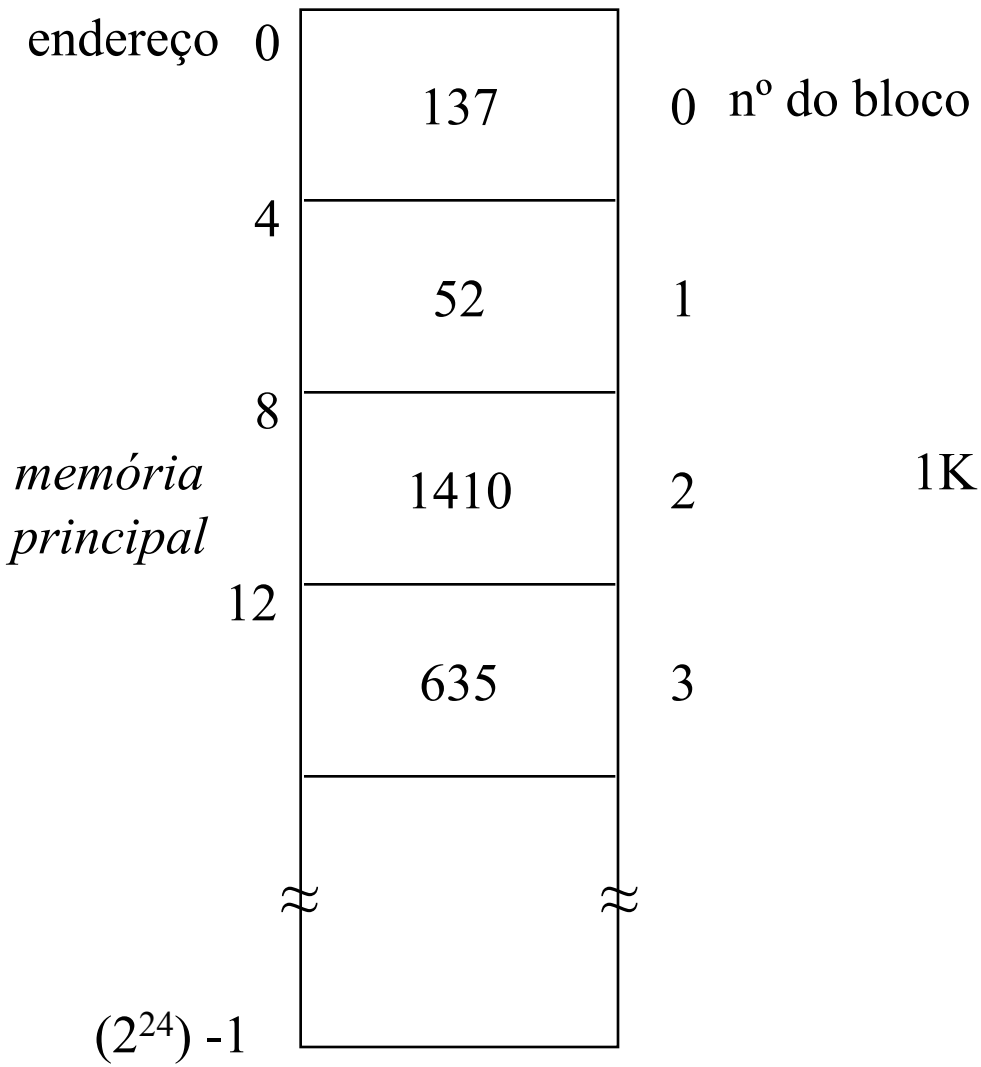
Um bloco consiste da unidade mínima de informação que pode estar presente ou ausente entre dois níveis da hierarquia.

A memória principal, de 2^m bytes, é dividida em blocos consecutivos de b bytes, totalizando $(2^m)/b$ blocos.

Cada bloco tem um endereço, que é um múltiplo de b, e o tamanho do bloco é, normalmente, uma potência de 2.

A *cache* associativa apresenta um número de posições, cada uma contendo um bloco e seu número de bloco, junto com um bit, que diz se aquela posição está em uso ou não. A ordem das entradas é aleatória.

memória principal



cache associativa

v	nº do bloco	valor
1	0	137
1	600	2131
1	2	1410
0		
1	160248	290380

1K posições

1	22 bits	32 bits

Se a *cache* estiver cheia, uma entrada terá que ser descartada para deixar lugar para uma nova.

Quando aparece um endereço de memória, o microprograma deve calcular o número do bloco e, então, procurar aquele número na *cache*.

Para evitar a pesquisa linear, a *cache* associativa pode fazer uso de uma memória associativa, que compara simultaneamente todas as entradas com o número do bloco dado. Isao torna a *cache* associativa cara.

Na *cache* com mapeamento direto, cada bloco é colocado numa posição, cujo número pode corresponder, por exemplo, ao resto da divisão do número do bloco pelo número de posições.

cache com
mapeamento direto

posição	v	*tag*	valor	endereços
0	1	0	137	0, 4096, 8192, 12288, ...
1	1	600	2131	4, 4100, 8196, 12292, ...
2	1	2	1410	8, 4104, 8200, 12296, ...
3	0			
1023	0			4092, 8188, 12284, 16380, ...

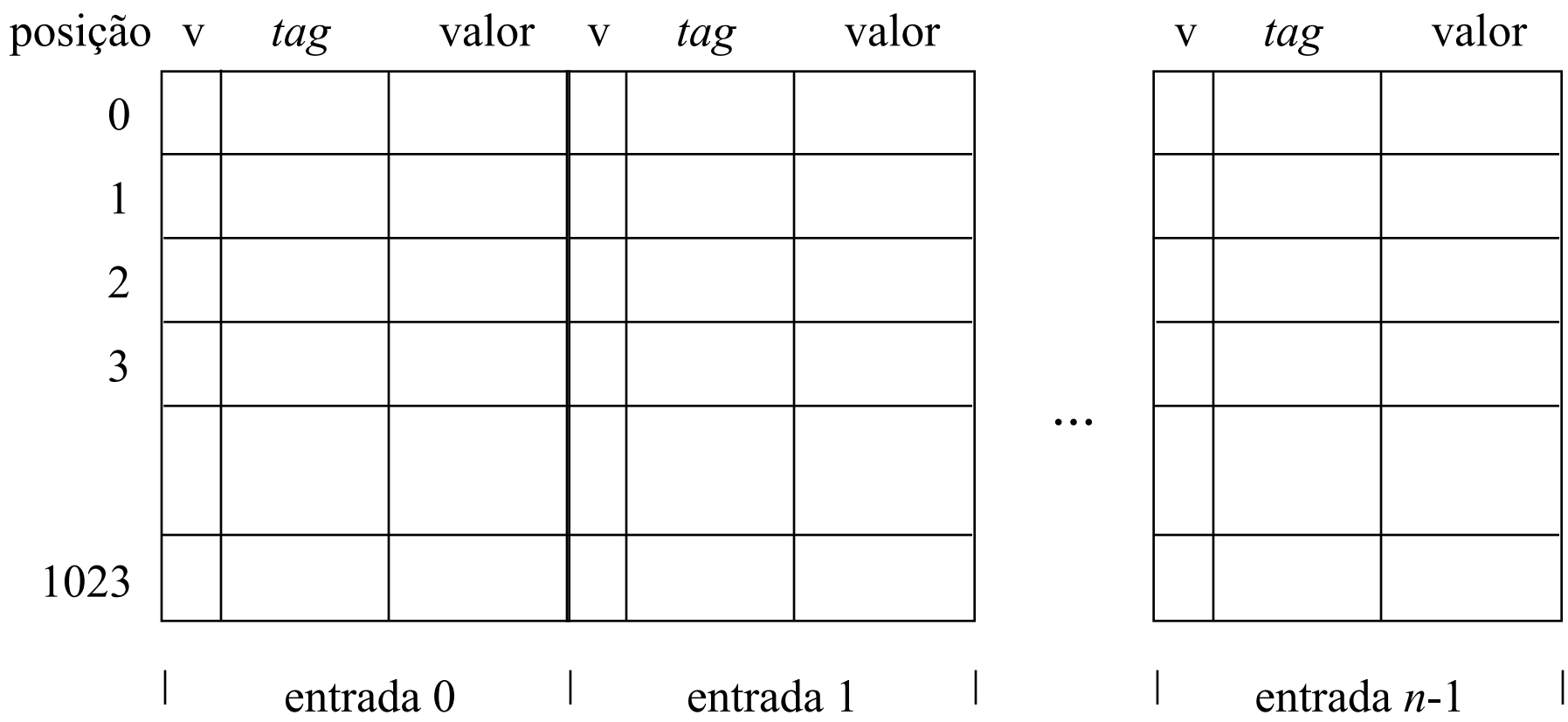
O campo *tag* guarda a parte do endereço que não participa do endereçamento da posição.

Ex: Seja a palavra no endereço 8192.

Os dois bits menos significativos são 0, pois os blocos são inteiros e múltiplos do tamanho do bloco (4 *bytes*).

O fato de que blocos múltiplos mapeiam na mesma posição pode degradar o desempenho da *cache*, se muitas palavras que estiverem sendo usadas mapeiem na mesma posição.

Na *cache* associativa por conjunto, utiliza-se uma *cache* com mapeamento direto com múltiplas entradas por posição.



Tanto a *cache* associativa quanto a com mapeamento direto são casos especiais da *cache* associativa por conjunto.

A *cache* com mapeamento direto é mais simples, mais barata e tem tempo de acesso mais rápido.

A *cache* associativa tem uma taxa de acerto maior para qualquer dado número de posições, pois a probabilidade de conflitos é mínima.

Uma preocupação no uso de hierarquia de memória é quanto a garantir a consistência da informação em todos os níveis da hierarquia.

Essa consistência é comprometida pela operação de escrita, que altera a informação em algum nível da hierarquia.

Uma técnica para manipular escritas em *cache* é denominada *write through*, quando uma palavra é escrita de volta na memória principal imediatamente após ter sido escrita na *cache* (consistência de dados).

Outra técnica é denominada *copy back*, em que a memória só é atualizada quando a entrada é expurgada da *cache* para permitir que outra entrada tome conta da posição (consistência de dados).

A técnica *write through* causa mais tráfego de barramento.

A técnica *copy back* pode gerar inconsistência se o processador efetuar uma transferência entre memória principal e disco, enquanto a primeira não tiver sido atualizada.

Se a razão de leituras para escritas for muito alta, pode ser mais simples usar *write through*.

Se houver muitas escritas, pode ser melhor usar *copy back* e fazer com que o microprograma expurgue toda a *cache* antes de uma operação de entrada/saída.