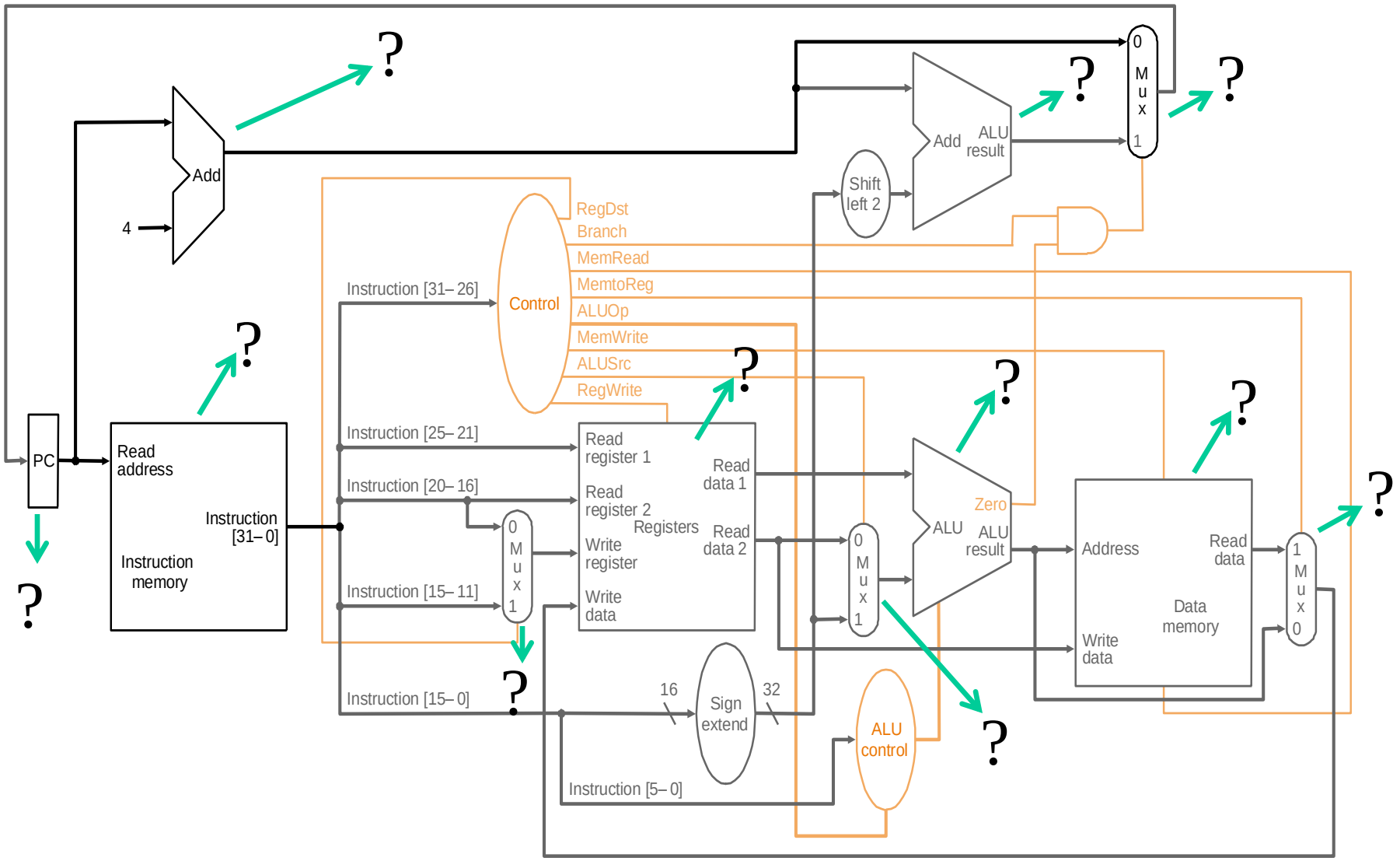
Arquitetura de Computadores III

Parte 2

Elementos, organização e hierarquia de Memória

Uma versão da arquitetura do Processador MIPS



O que é uma palavra de dados?

- Conjunto de bits?
- Quantos bits?
- O que significa processador de 32 bits? E de 64 bits?

- Como representar a palavra de dados?
- Como armazenar uma palavra de dados?

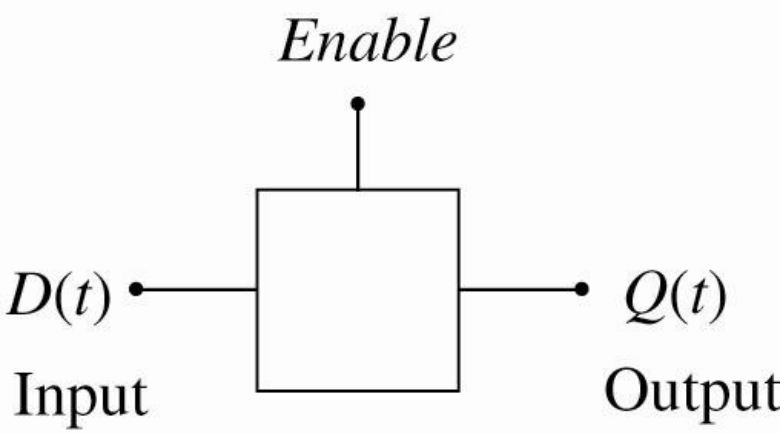
Tipos de Memória

- RAM: Random-Access Memory
- ROM: Read-Only Memory
- PROM: Programmable ROM
  - EPROM: Apagável com radiação ultravioleta
  - EEPROM: Apagável por sinais elétricos.

- Registradores?

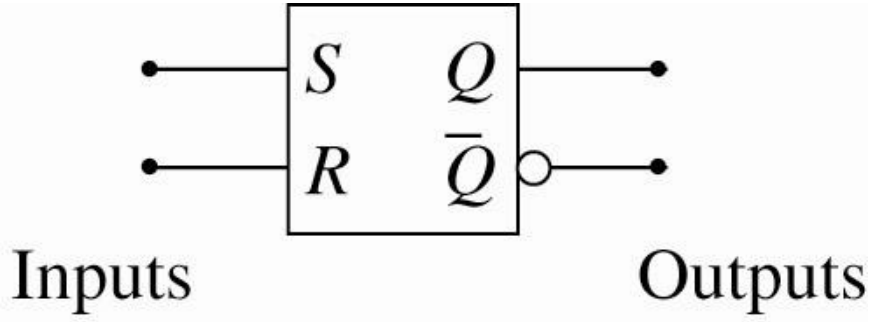
Latch

- Um latch é um elemento lógico que pode acompanhar as variações do dado e transferir estas mudanças para uma linha de saída.
- Circuito biestável: Q pode valer 0 ou 1.



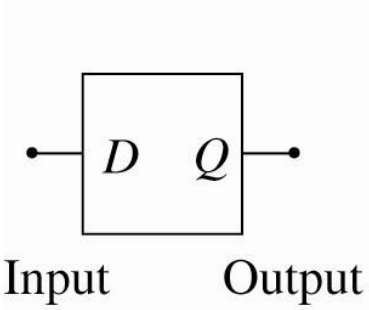
Latch SR (Set-Reset)

- O latch SR é um elemento biestável transparente, ou seja, sensível às variações das entradas.
- Na operação de set a saída é forçada para o valor Q=1
- Na operação reset a saída é forçada para Q=0



Latch D

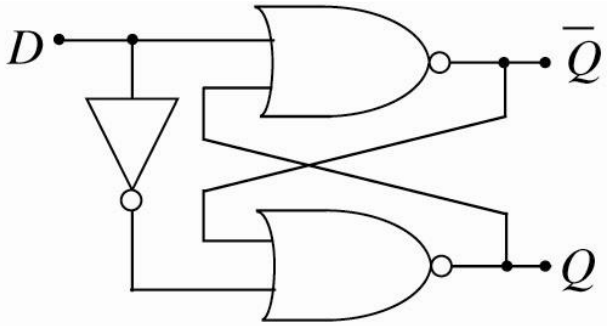
- Um latch tipo D tem uma única entrada D que atua como entrada de um bit de dado.



(a) Element

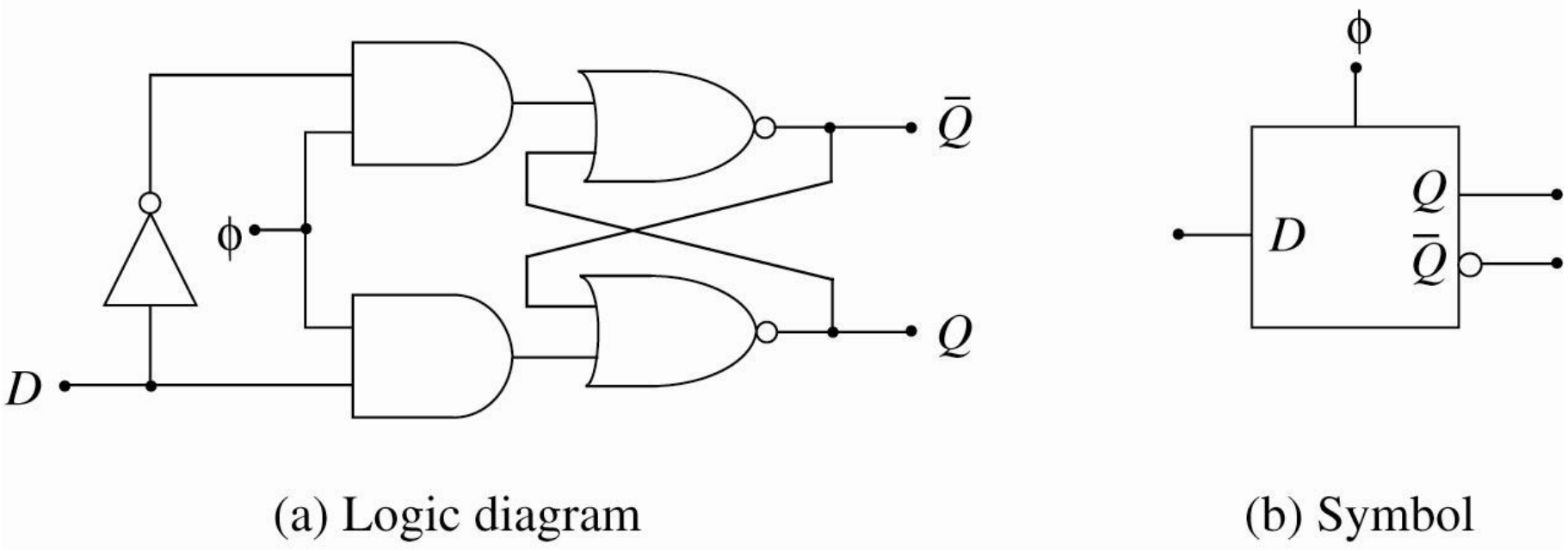
D	Q
0	0
1	1

(b) Operation



(a) Logic diagram

Latch D com Clock

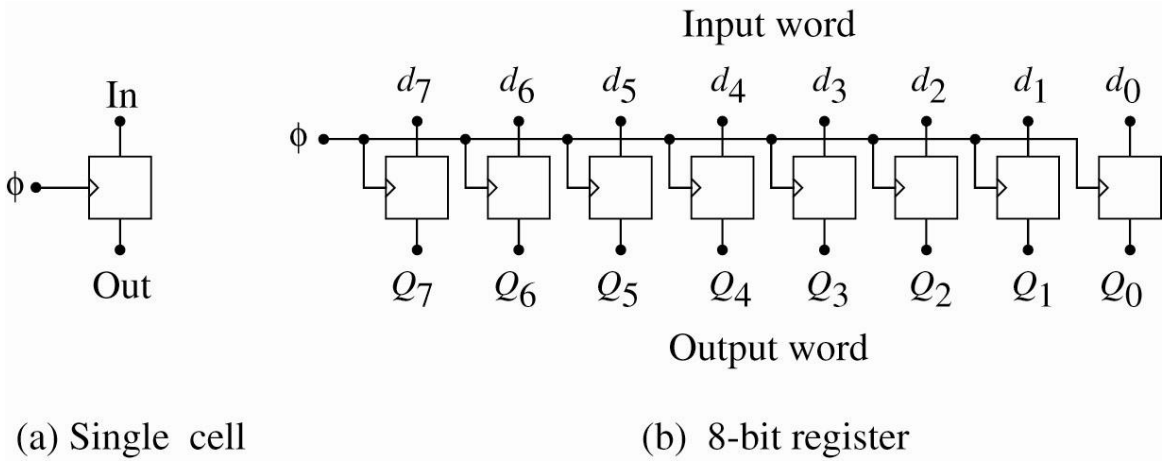


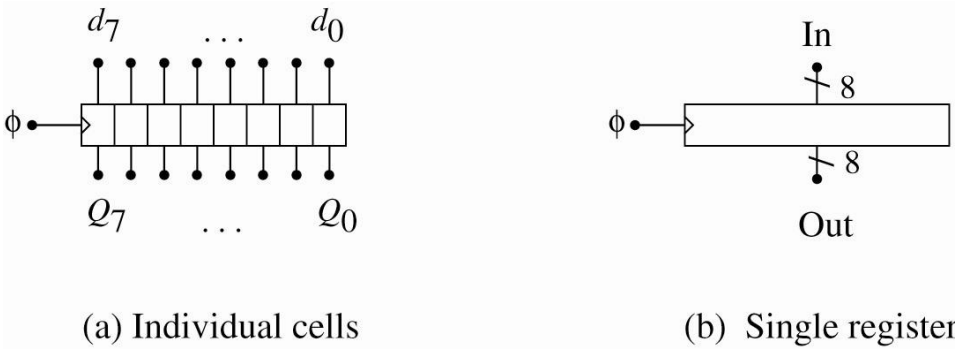
(a) Logic diagram

(b) Symbol

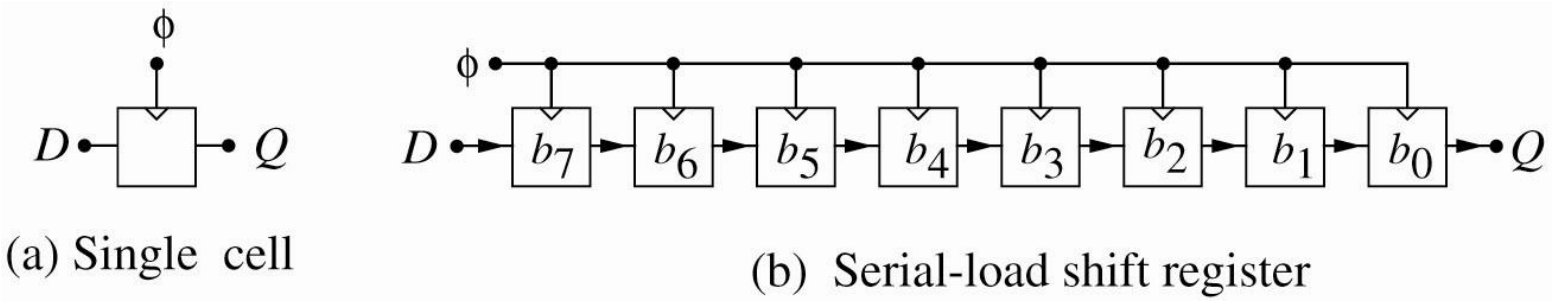
Registradores

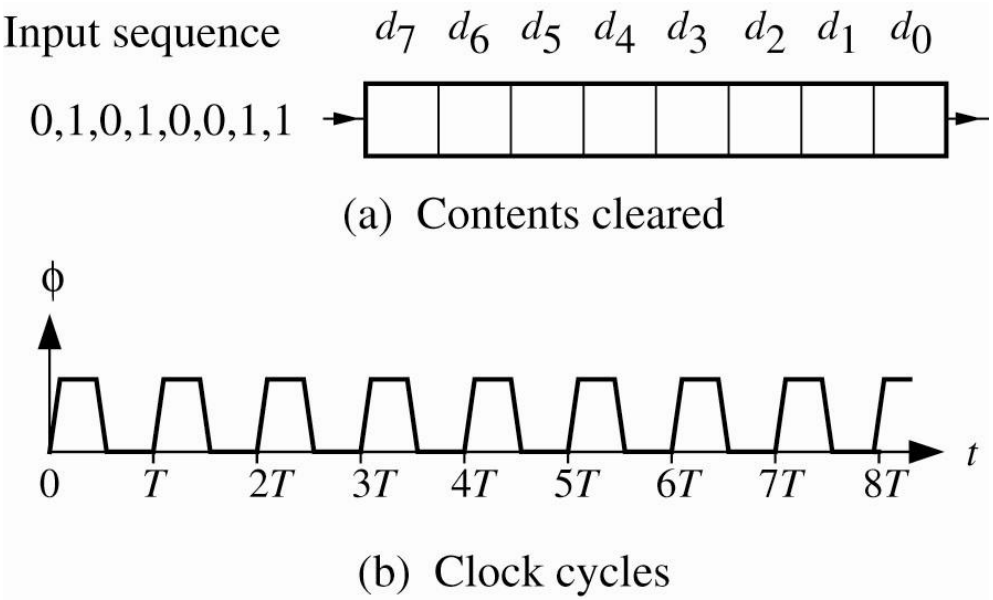
- Um registrador é um elemento lógico utilizado para armazenar uma palavra binária de n-bits.



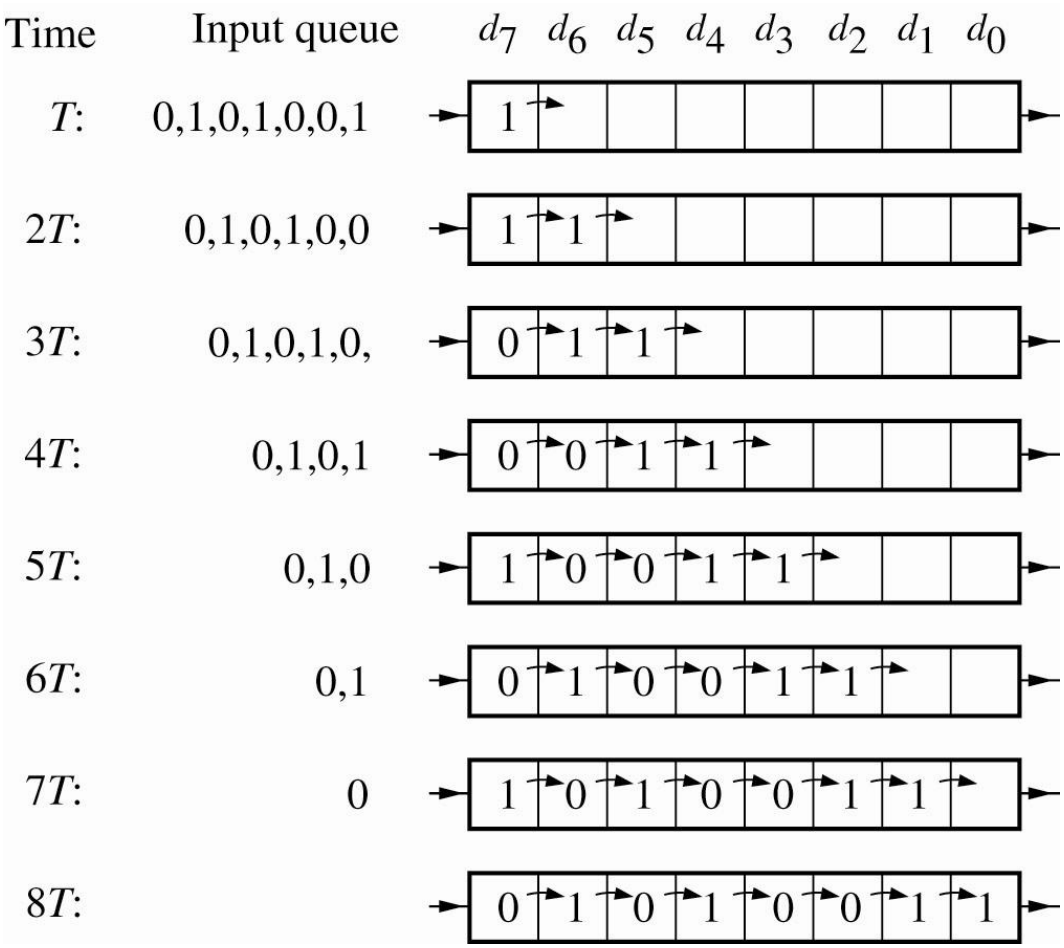


Registradores de Deslocamento

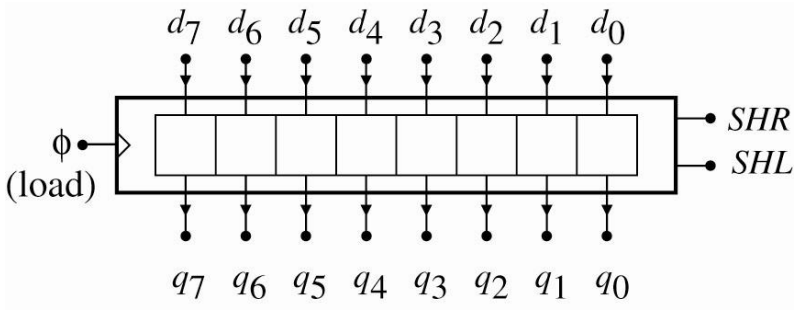


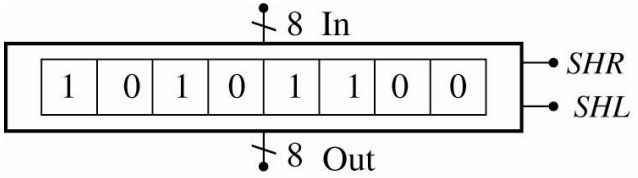


Registradores de Deslocamento

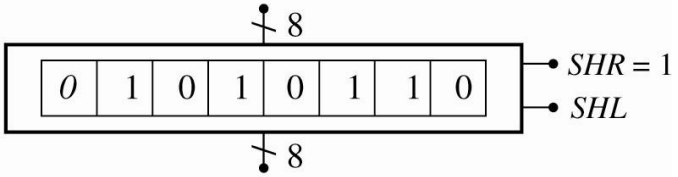


Registradores de Deslocamento

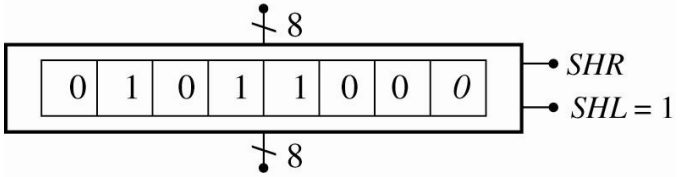




(a) Initial condition



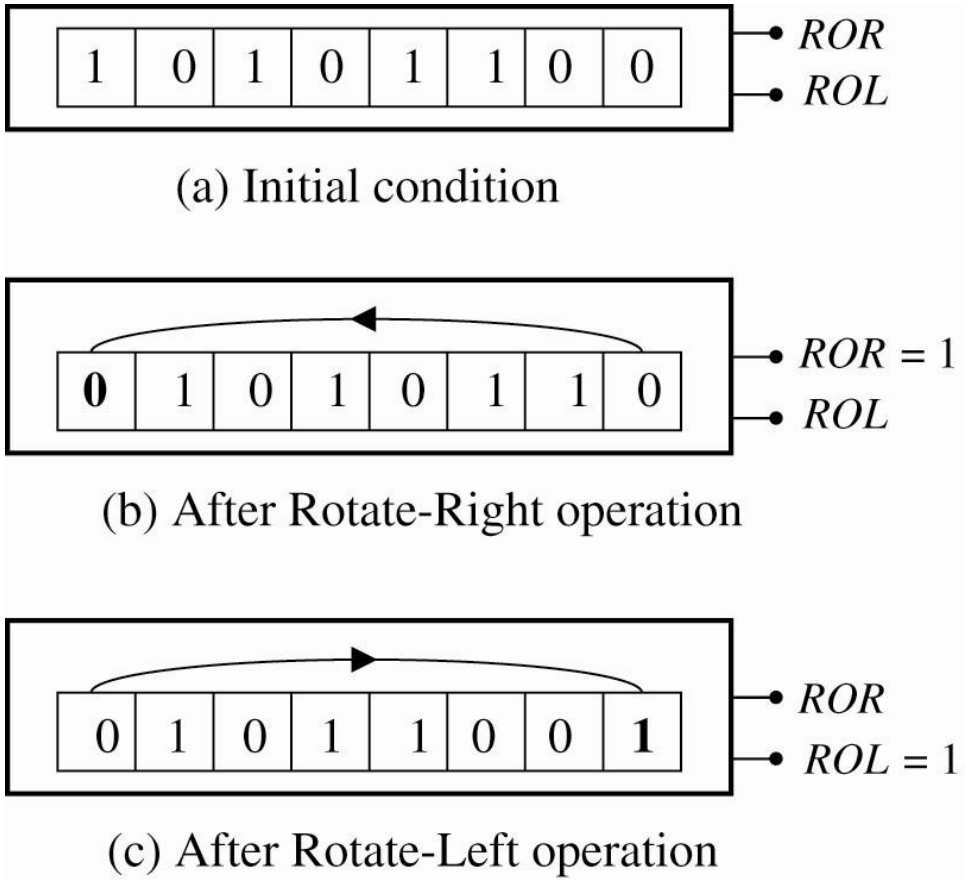
(b) After Shift Right operation



(c) After Shift Left operation

Registradores de Deslocamento

- Operações de rotação



Memória RAM

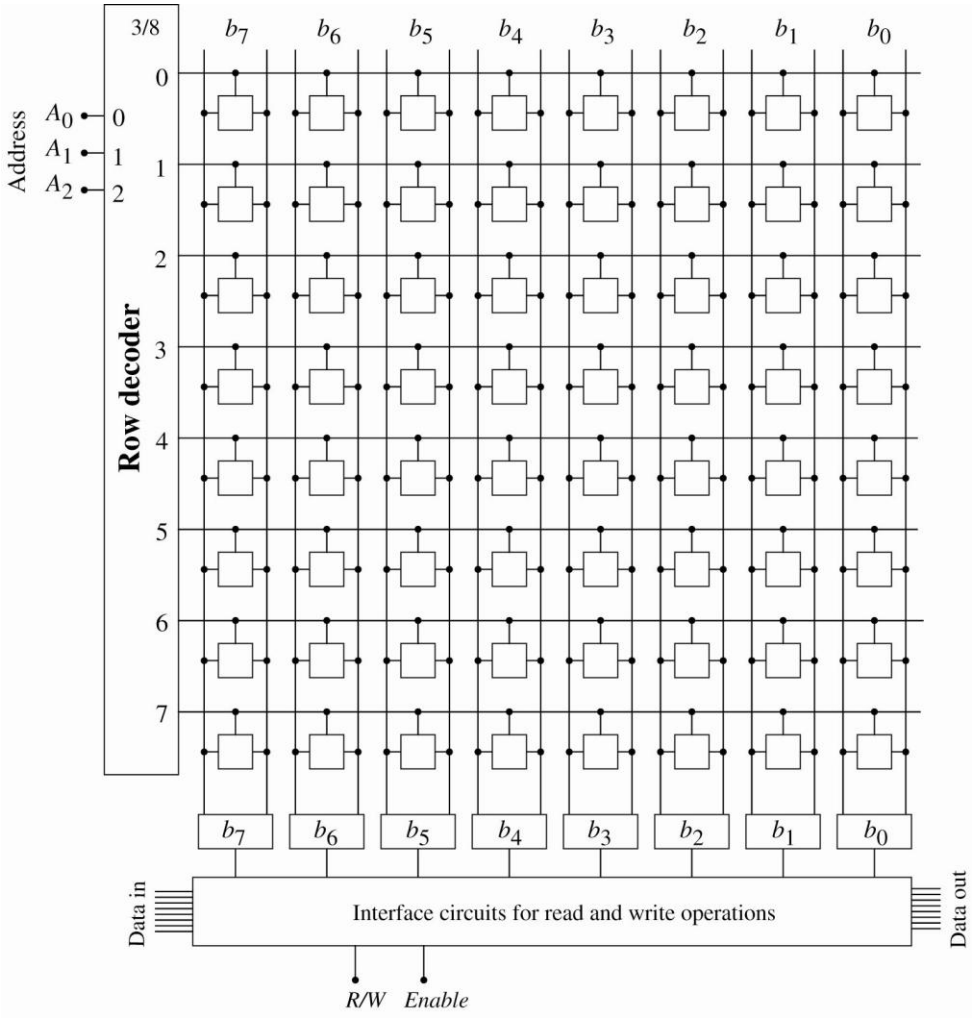
- Célula RAM estática: A memória estática é capaz de manter os bits de dados armazenados apenas enquanto a fonte de alimentação estiver conectada ao circuito. Uma célula SRAM é equivalente ao Latch SR.

Memória RAM

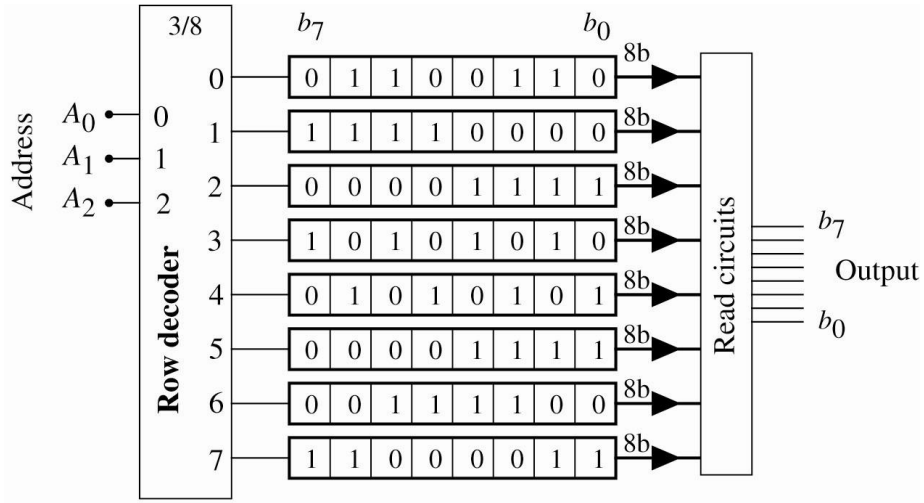
- Célula RAM dinâmica: A DRAM é similar a SRAM. A diferença é o projeto das células. As células dinâmicas são mais simples e necessitam de menos área no chip. Isto permite que a DRAM seja construída com densidades de armazenamento maiores, reduzindo o custo do bit. A DRAM é muito utilizada para memórias principais dos computadores. A desvantagem da DRAM é que as células são mais lentas. Os tempos de leitura e escrita são maiores. Uma célula DRAM é construída a partir de capacitores, demandando *refresh* de memória para manter o os dados armazenados.

Arranjo das SRAMs

Matriz 8x8



Operação de Leitura em uma Matriz RAM

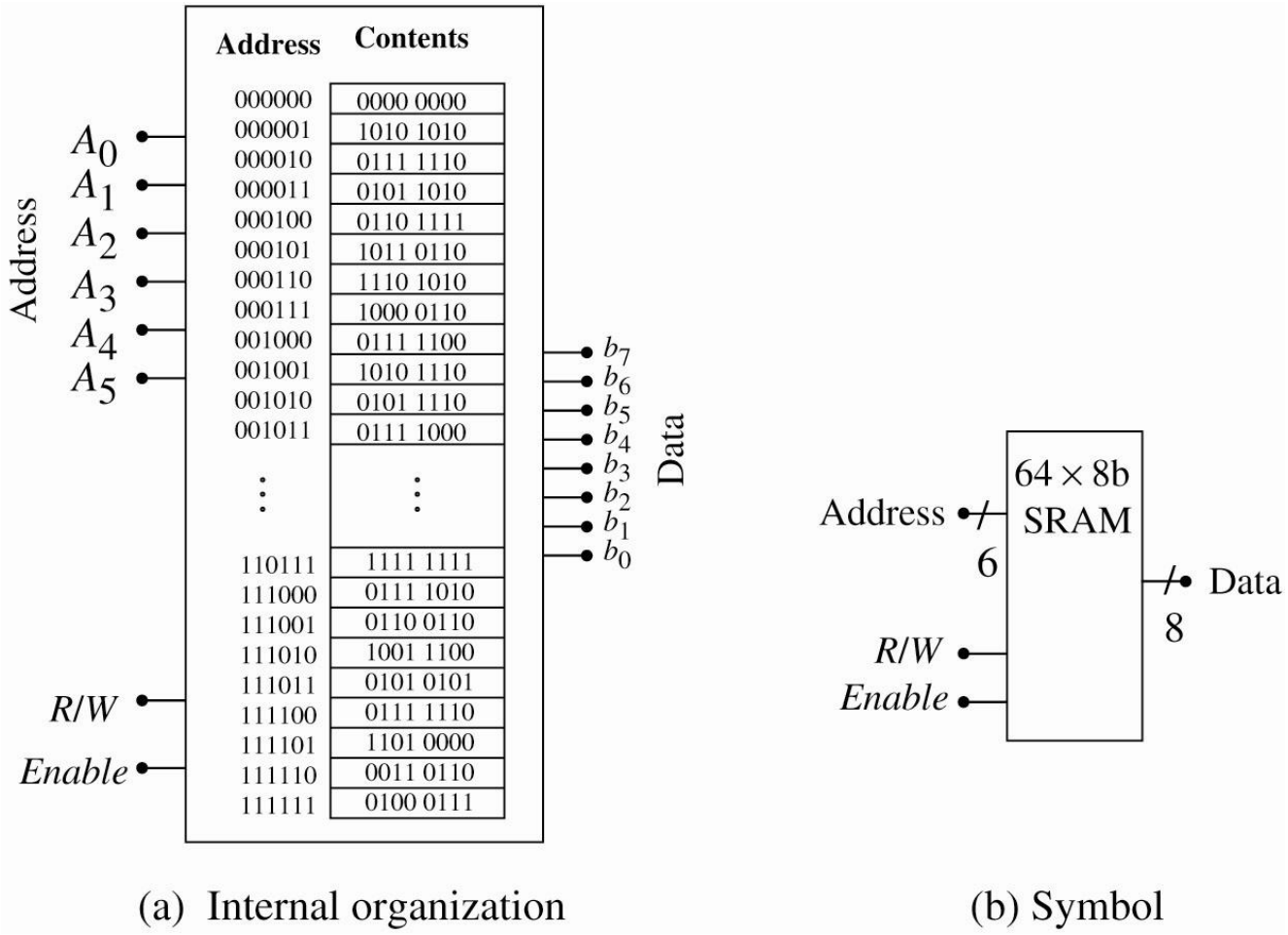


(a) Simplified network

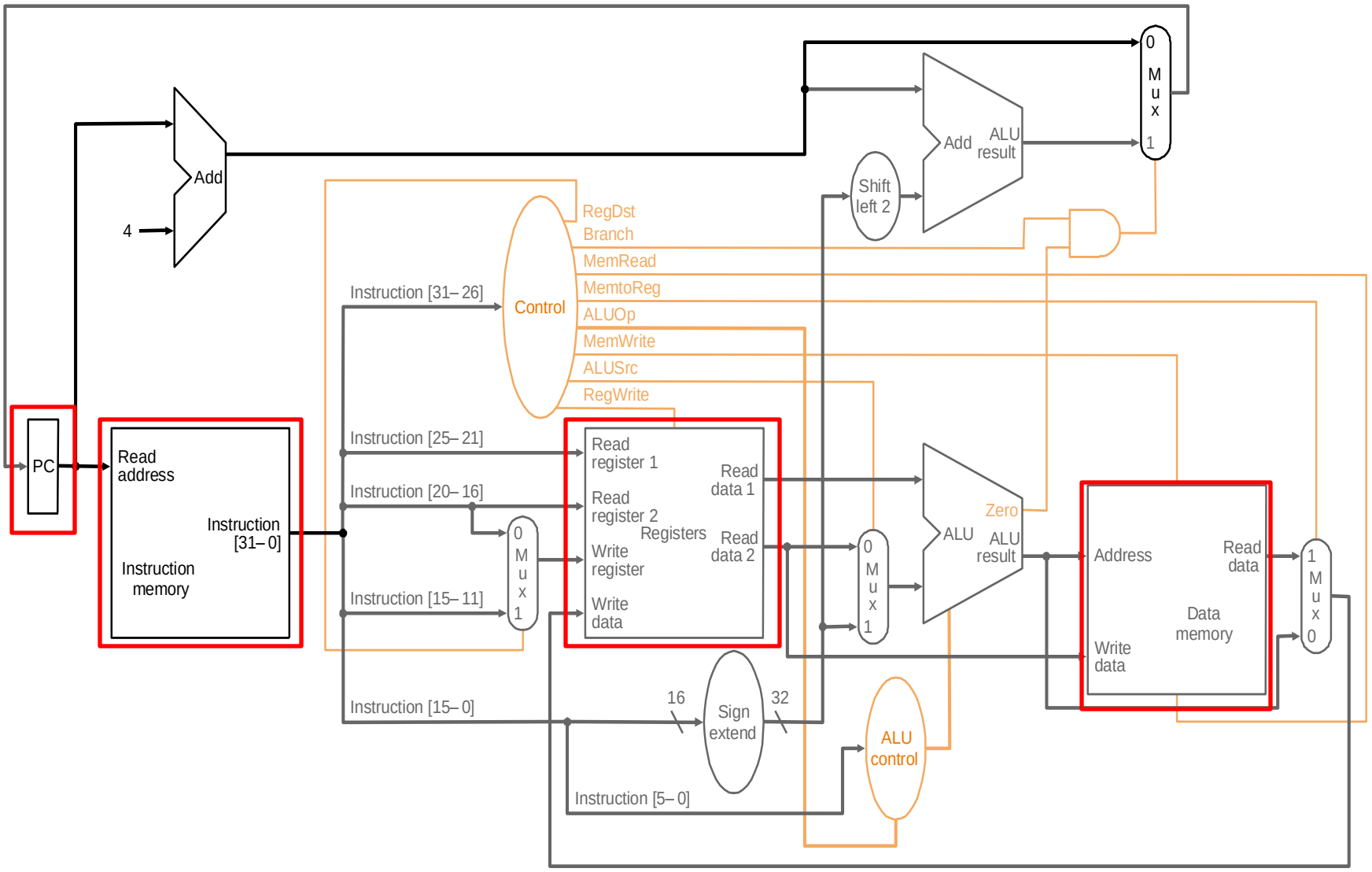
Address $A_2 A_1 A_0$	Row	Output $b_7 \ldots b_0$
0 0 0	0	0110 0110
0 0 1	1	1111 0000
0 1 0	2	0000 1111
0 1 1	3	1010 1010
1 0 0	4	0101 0101
1 0 1	5	0000 1111
1 1 0	6	0011 1100
1 1 1	7	1100 0011

(b) Function table

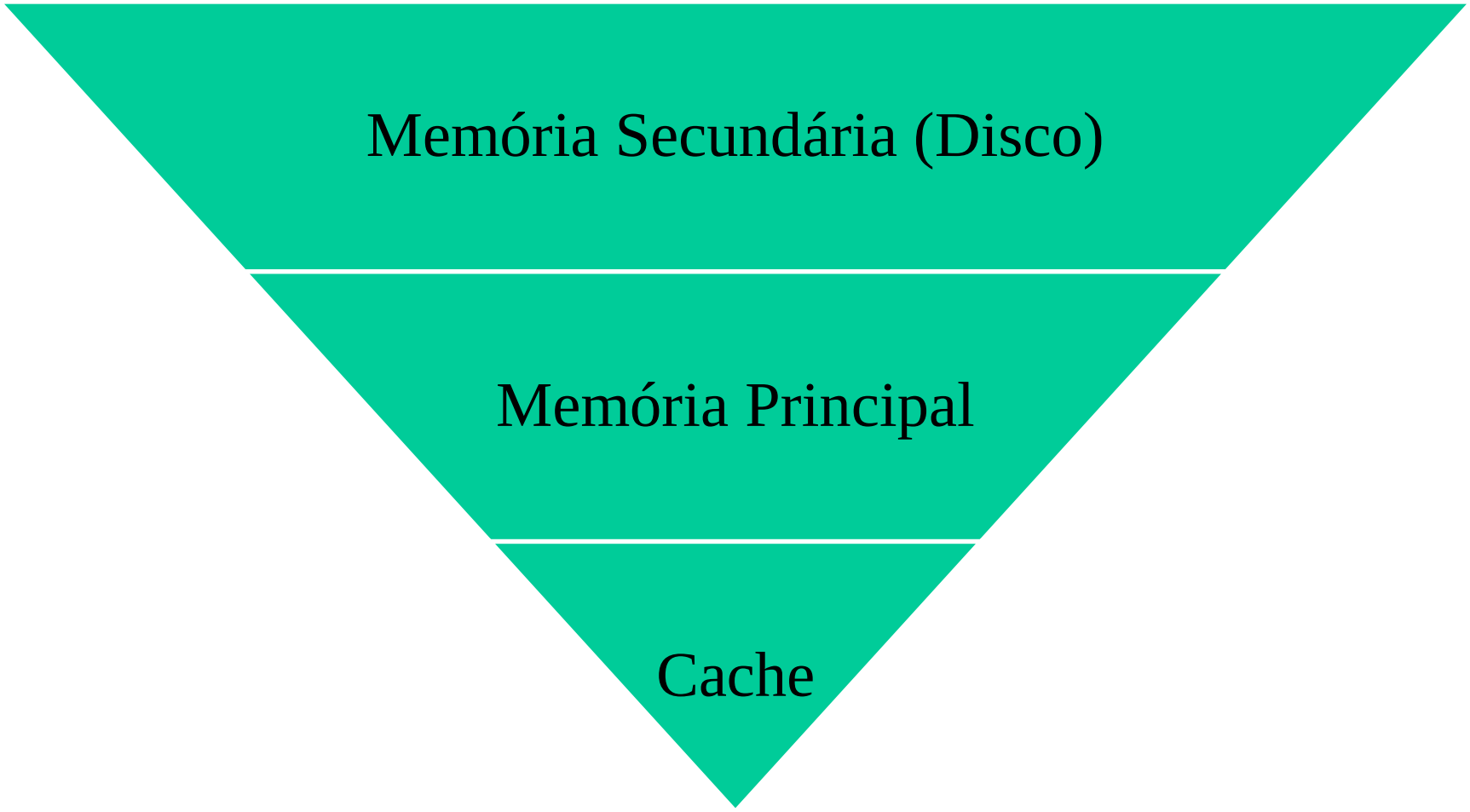
Diagrama em Blocos para SRAM 64x8



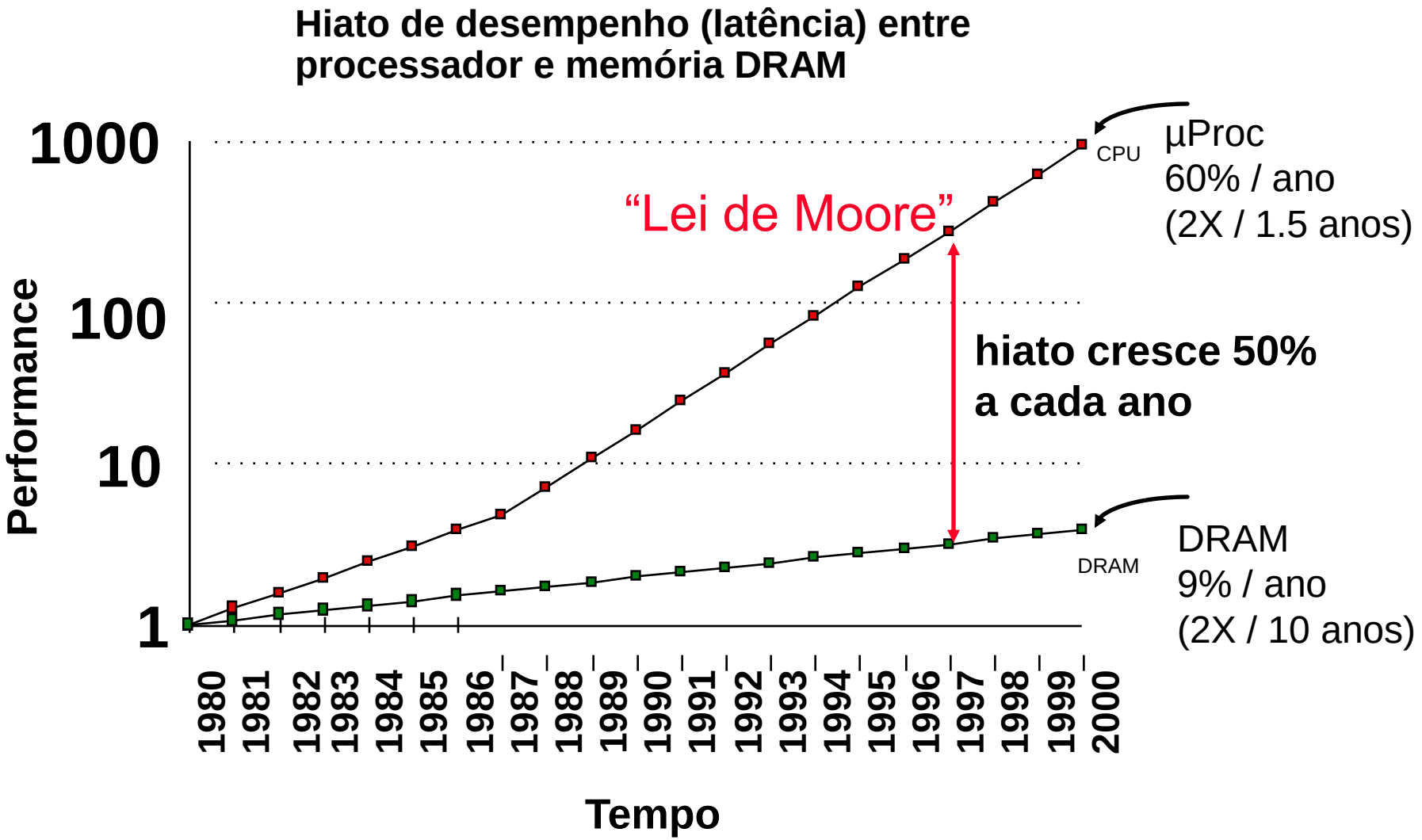
O que nós já sabemos?



Hierarquia de Memória



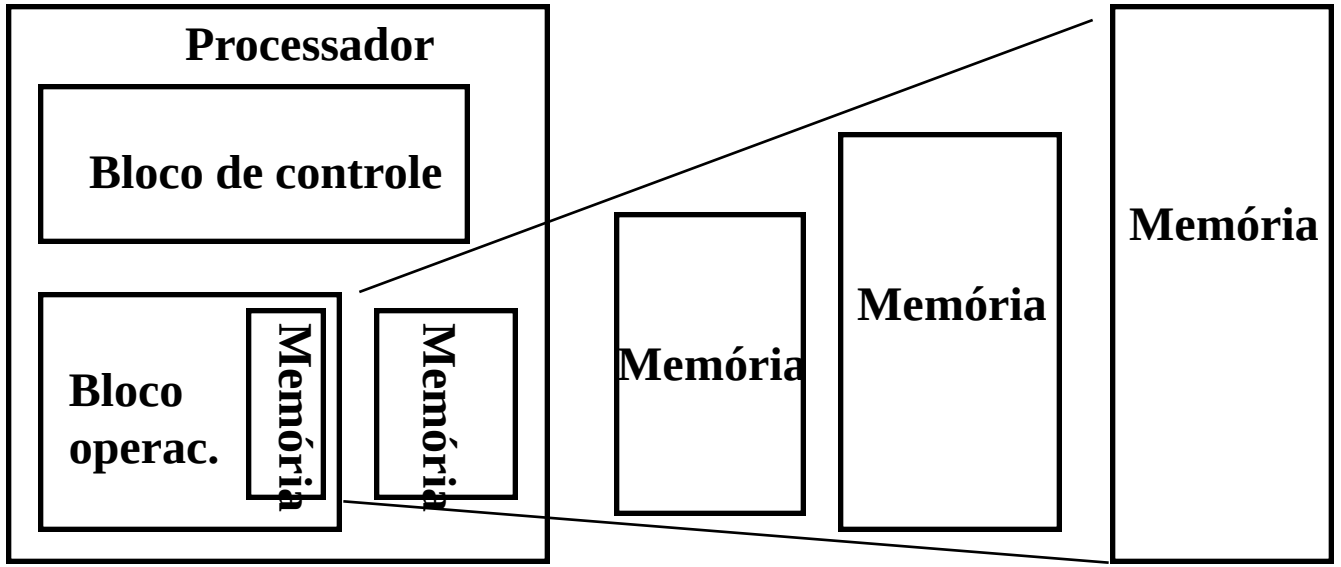
Tendências tecnológicas



Hierarquia de Memória

- **Objetivo: oferecer ilusão de máximo tamanho de memória, com mínimo custo.**
- **Máxima velocidade.**
- **Cada nível contém cópia de parte da informação armazenada no nível superior seguinte.**

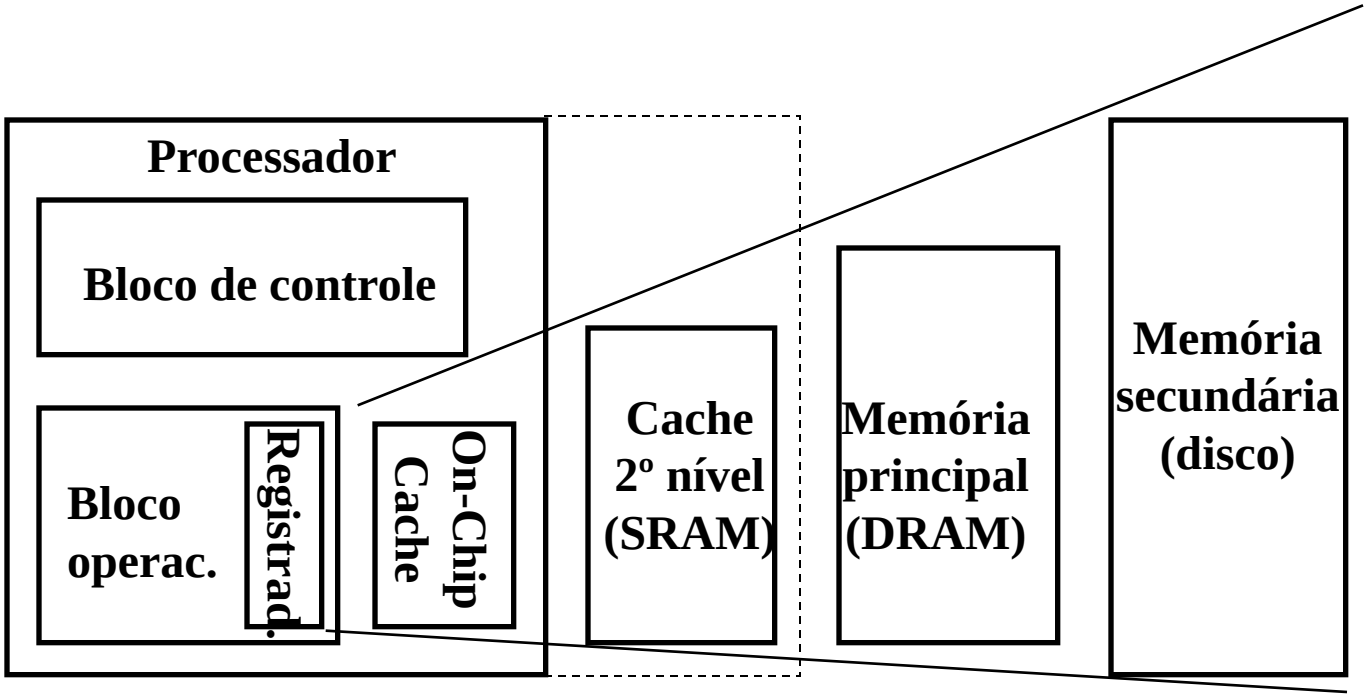
Hierarquia de Memória



Velocidade: Mais rápida
Tamanho: Menor
Custo: Mais alto

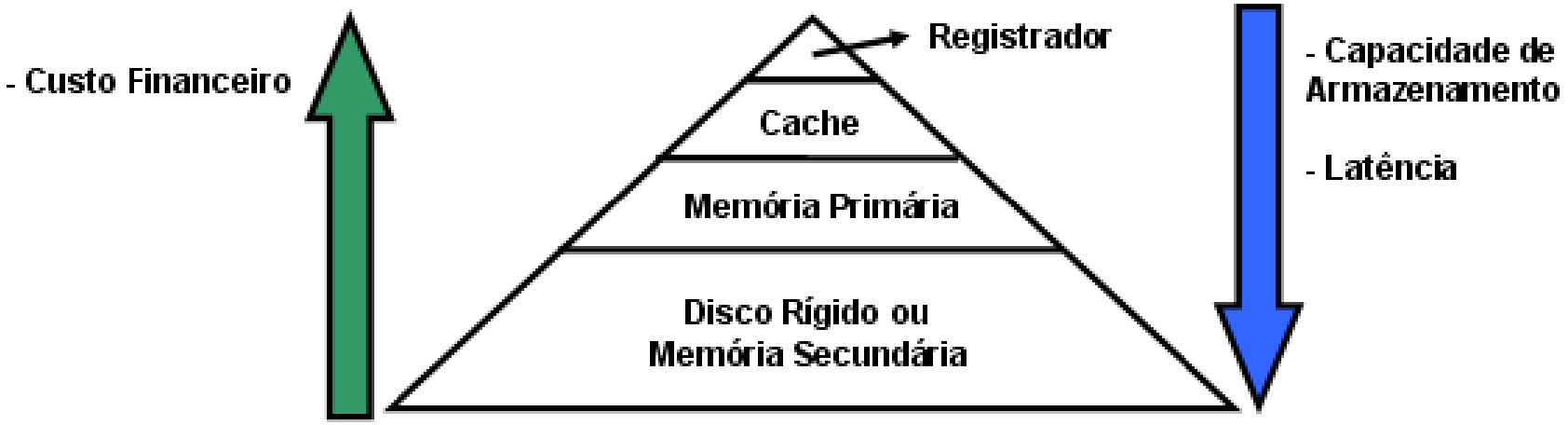
Mais lenta
Maior
Mais baixo

Hierarquia de Memória



Velocidade (ns):	0,1	1	2-5	10-20	10.000.000 (10 ms)
Tamanho (bytes):	100	16 k	512 k ...Ms	256 M ...Gs	Gs

Hierarquia de Memória



Hierarquia de Memória

Como a hierarquia é gerenciada?

- Registradores <-> memória
  - –pelo compilador
- cache <-> memória principal
  - –pelo hardware
- memória principal <-> disco
  - –pelo hardware e pelo sistema operacional (memória virtual)
  - –pelo programador (arquivos)

Hierarquia de Memória

Princípio da Localidade

- Espacial: se um dado é referenciado, seus vizinhos tendem a serem referenciados logo.
- Temporal: um dado referenciado, tende a ser referenciado novamente.

Como explorar o princípio de localidade numa hierarquia de memória?

- Localidade Temporal

  => Mantenha itens de dados mais recentemente acessados nos níveis da hierarquia mais próximos do processador

- Localidade Espacial

  => Mova blocos de palavras contíguas para os níveis da hierarquia mais próximos do processador

Organizações de Memória Cache

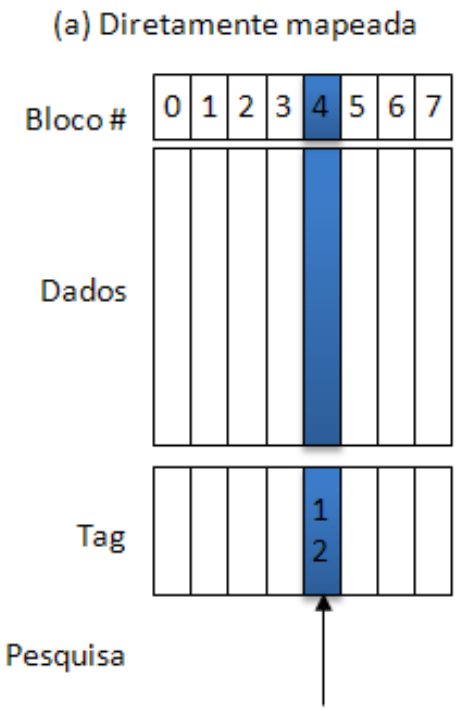
- processador gera endereço de memória e o envia à cache
- cache deve
  - verificar se tem cópia da posição de memória correspondente
  - se tem, encontrar a posição da cache onde está esta cópia
  - se não tem, trazer o conteúdo da memória principal e escolher posição da cache onde a cópia será armazenada
- *mapeamento* entre endereços de memória principal e endereços de cache resolve estas 3 questões
  - deve ser executado em hardware
- estratégias de organização (mapeamento) da cache
  - mapeamento completamente associativo
  - mapeamento direto
  - mapeamento set-associativo

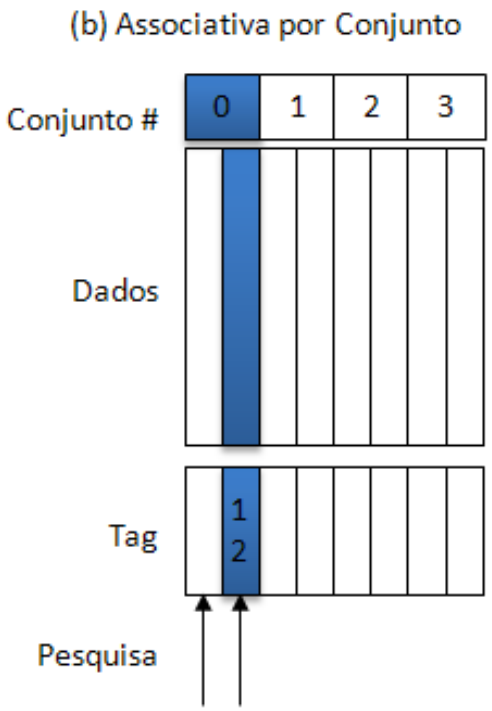
Memórias Cache

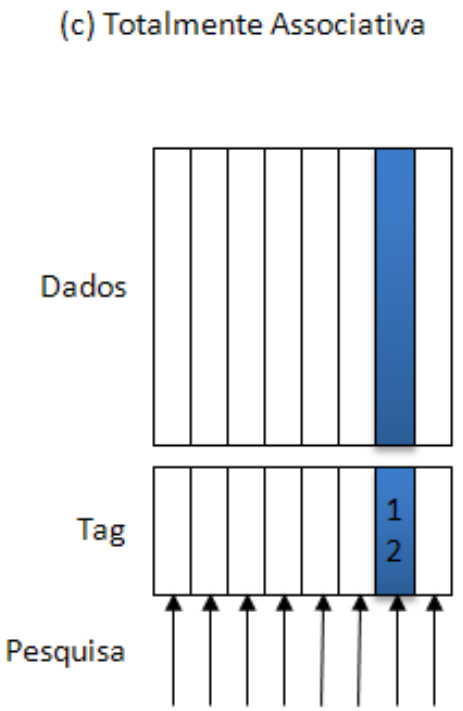
- Características
  - Pouco espaço de armazenamento
  - Alto custo financeiro
  - Baixo tempo de acesso

- Conceitos
  - **Palavra:** conjunto de um ou mais bytes.
  - **Bloco:** conjunto de uma ou mais palavras (unidade da cache)
  - **Bit de Válido**: indica se o dado ou bloco está válido
  - **Tag ou rótulo**: parte do endereço de uma palavra na memória principal
  - **Slot**: cada linha de uma cache, que pode armazenar um ou mais blocos dependendo da organização da cache.
  - **Comparador:** compara a tag de um endereço de uma palavra, com as tags dos endereços armazenados na cache

Modos de Mapeamento





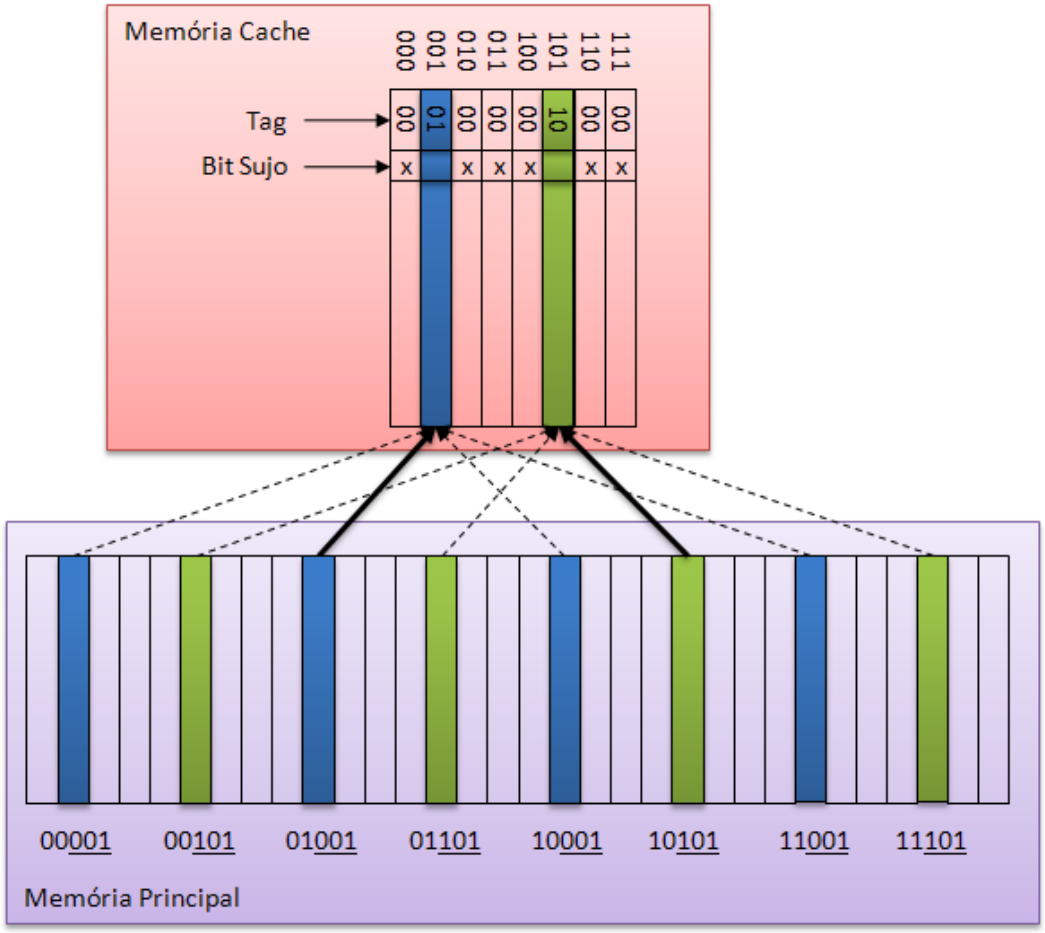


12 modulo 8 = 4
Bloco 4

12 modulo 4 = 0
Bloco 0

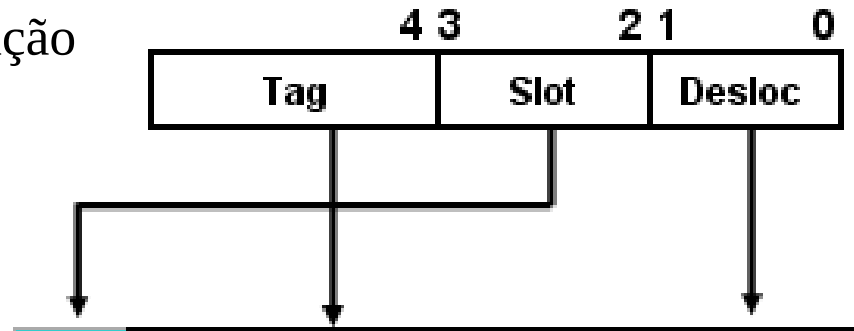
Qualquer end.

Mapeamento Direto



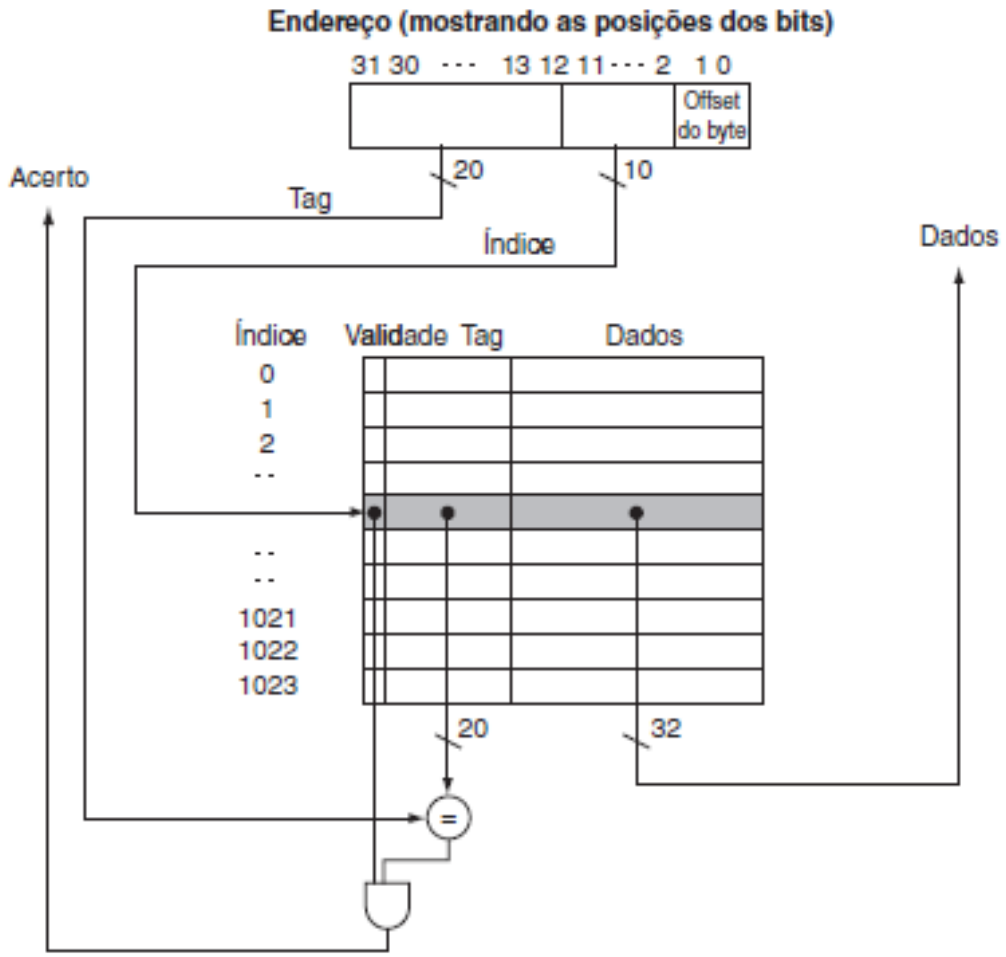
Organizações da Cache

- **Mapeamento Direto**
  - Um bloco pode ser colocado em apenas um slot na cache
  - Necessita de apenas uma comparação
  - Exemplo:
    - Endereços de 5 bits (32 palavras)
    - 4 slots
    - Blocos de 4 palavras

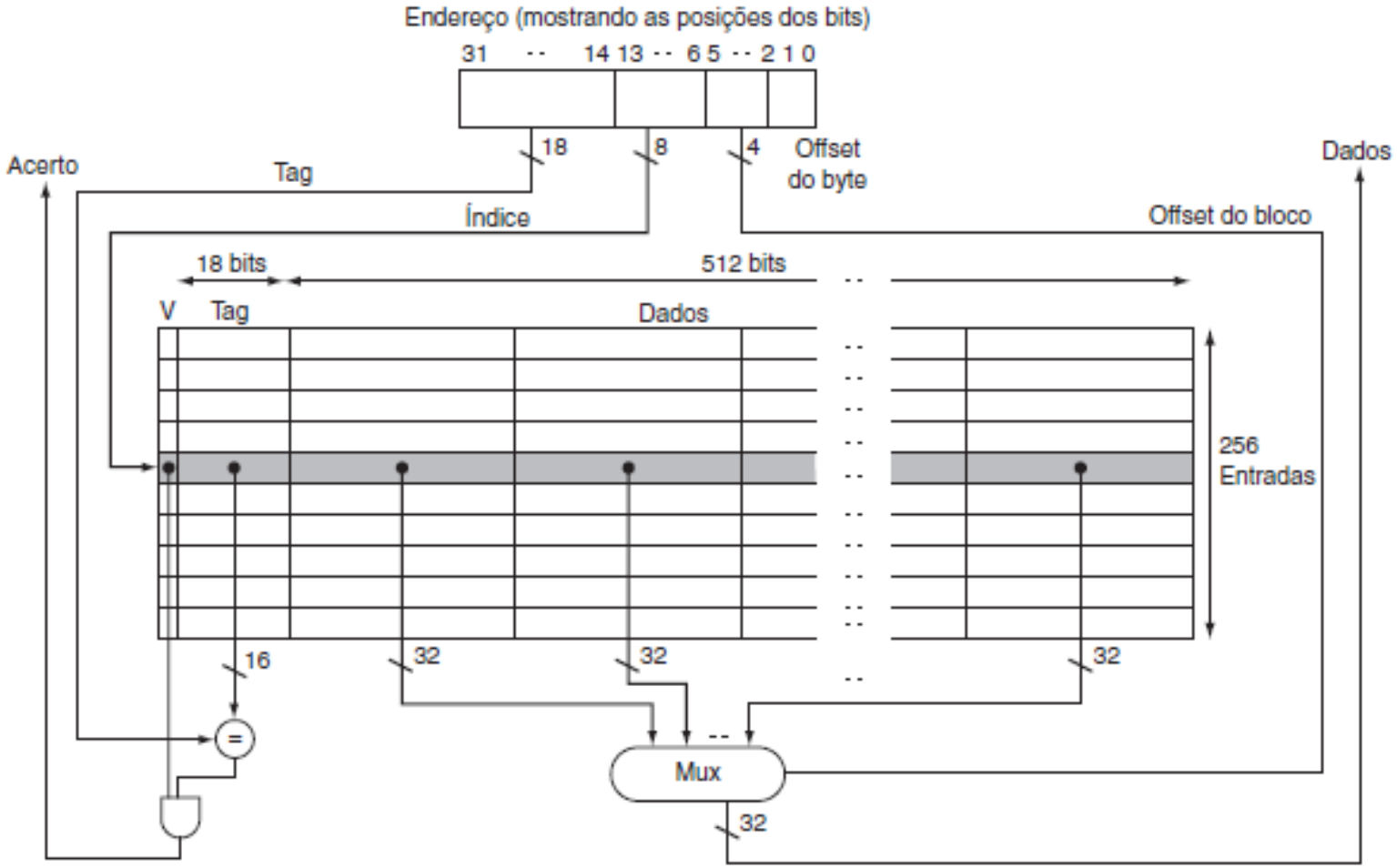


Slot	V	Tag	Dado (Bloco)			
0	1	0	A_0	B_0	C_0	D_0
1	0	-	-	-	-	-
2	0	-	-	-	-	-
3	1	0	A_3	B_3	C_3	D_3

Tamanho da linha

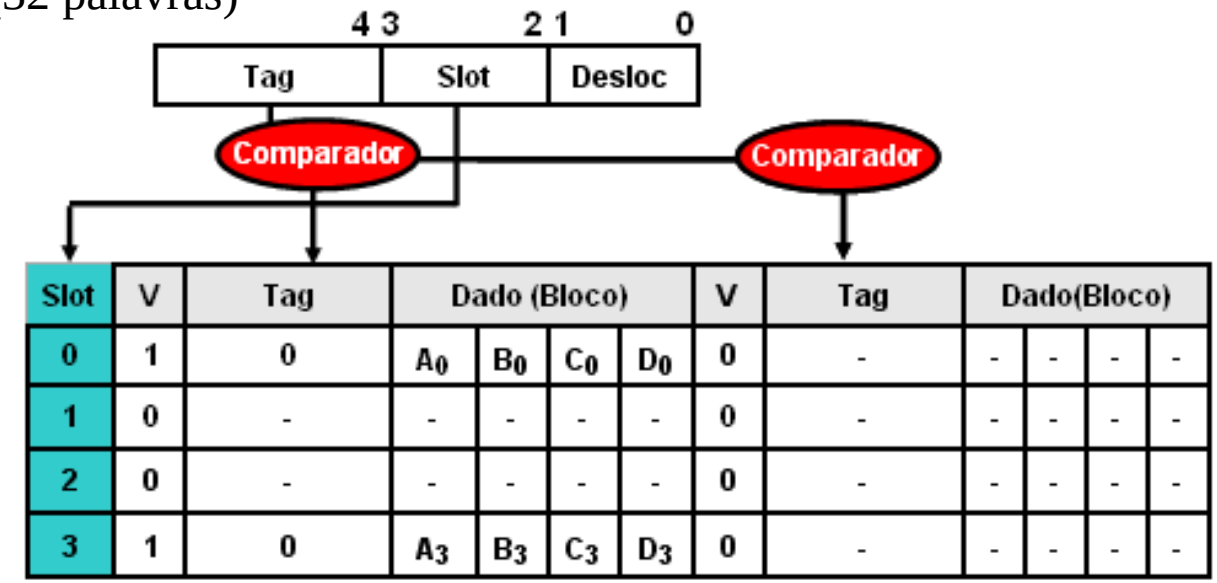


Tamanho da linha tirando vantagem da localidade espacial



Organizações da Cache

- **Associativa por Conjunto N-way**
  - N blocos podem ser colocados em um mesmo slot
  - Necessita de N comparações
  - Exemplo:
    - Endereços de 5 bits (32 palavras)
    - 4 slots
    - 2-way
    - Blocos de 4 palavras

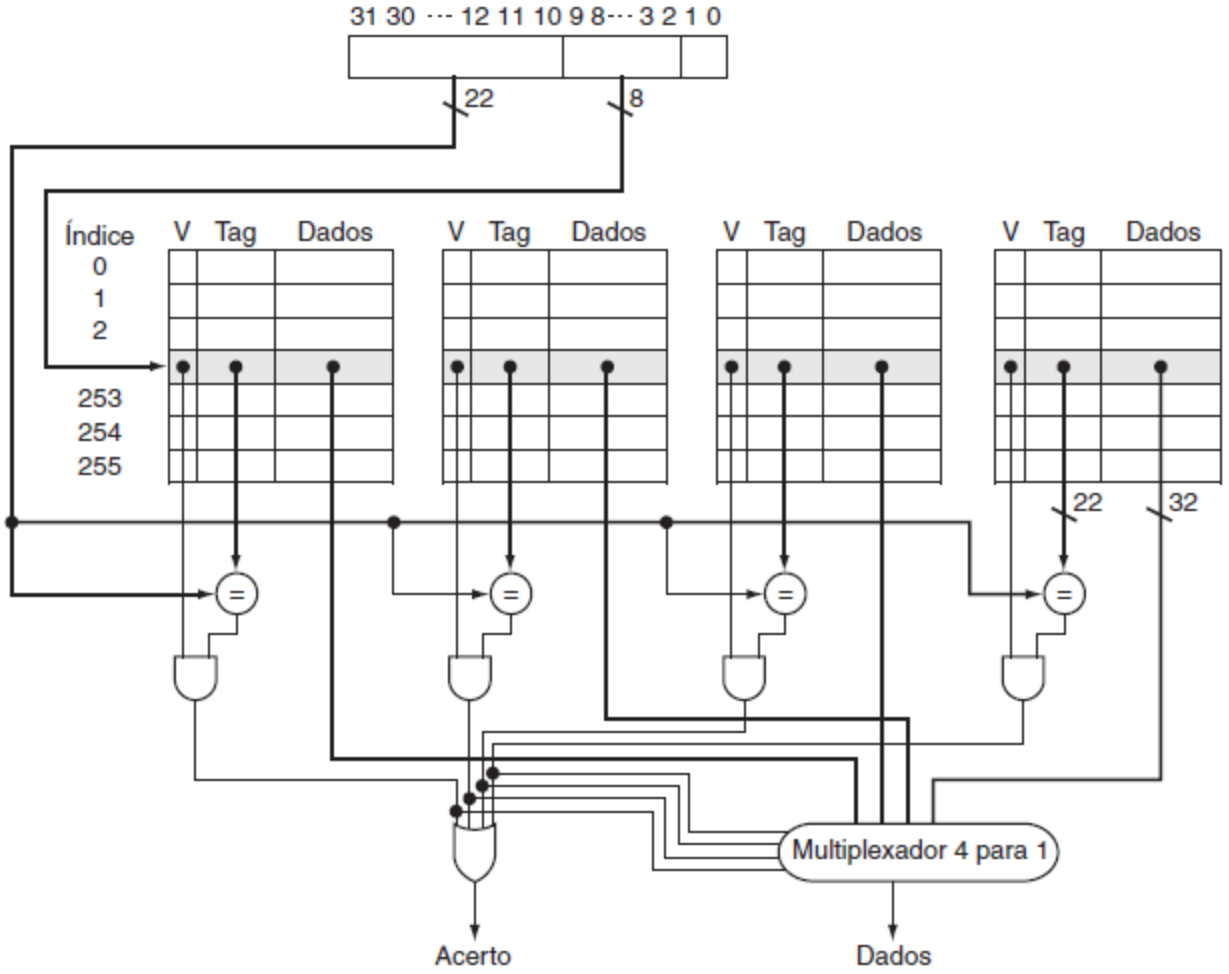


Slot	V	Tag	Dado (Bloco)				V	Tag	Dado(Bloco)			
0	1	0	A_0	B_0	C_0	D_0	0	-	-	-	-	-
1	0	-	-	-	-	-	0	-	-	-	-	-
2	0	-	-	-	-	-	0	-	-	-	-	-
3	1	0	A_3	B_3	C_3	D_3	0	-	-	-	-	-

Organizações da Cache

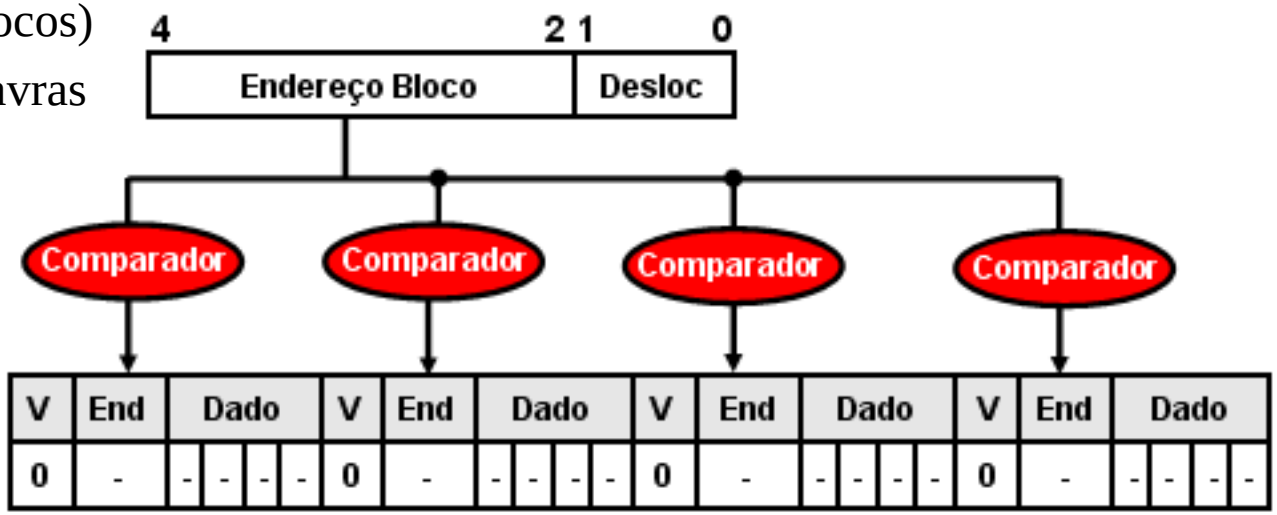
Associativa por conjunto

4 vias



Organizações da Cache

- **Completamente Associativa**
  - Possui um slot com N blocos
  - Necessita de N comparações
  - Endereça o bloco diretamente
  - Exemplo:
    - Endereços de 5 bits (32 palavras)
    - 1 slot (com 4 blocos)
    - Blocos de 4 palavras



Acesso à Cache

- Quando a cache estiver sem espaço, qual bloco será substituído?
  - Mapeamento Direto: o bloco que estiver no slot
  - Associativa por conjunto e Completamente Associativa: usar uma política de substituição
    - LRU: substituir o bloco menos recentemente utilizado
    - FIFO: substituir o primeiro bloco que entrou na cache
    - Aleatório: escolher um bloco qualquer

Acesso à Cache

- Quando ocorrer uma escrita, como manter a coerência com a memória principal?
  - Write-through: a palavra é escrita tanto no bloco da cache, quanto no bloco da memória principal.
  - Write-back: a palavra é escrita somente no bloco da cache. Quando este bloco for substituído, então a palavra será escrita na memória principal.

Cache hit e Cache miss

- Cache Hit: acerto na cache, ou seja, o dado procurado já se encontra carregado na cache
  - Hit Time: tempo de acesso ao nível **superior**, que consiste de tempo de acesso + tempo para determinar hit/miss
- Cache Miss: falta na cache, ou seja, o dado procurado ainda tem que ser buscado na memória principal.
  - Miss Penalty: tempo gasto para substituir um bloco no nível **superior** + tempo para fornecer o bloco ao processador

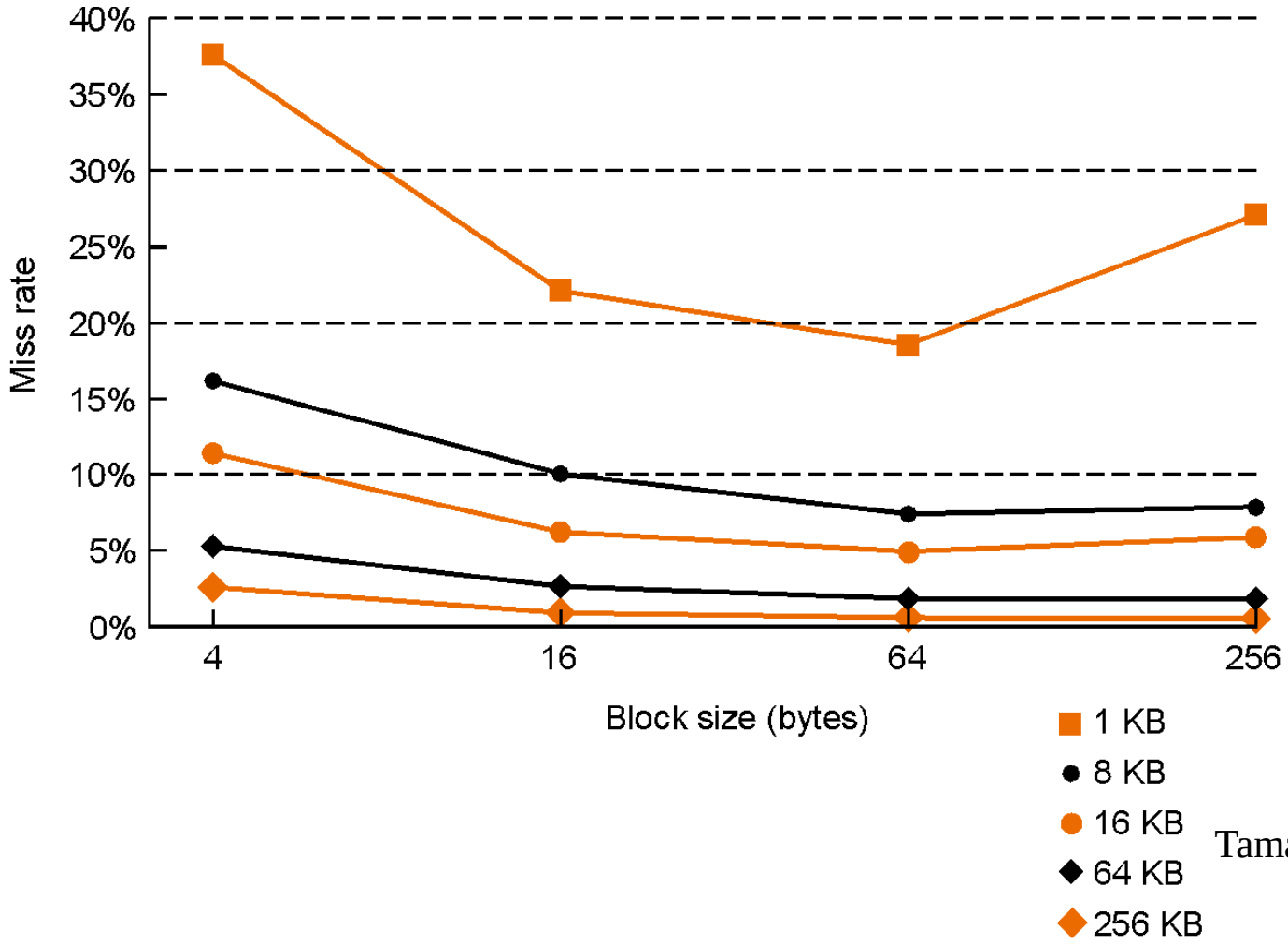
- Métrica
  - Taxa de acerto: dado um número de acessos a cache, qual a porcentagem de cache hit.

$$\text{Taxa de Acerto} = \frac{\text{Nº Cache Hit}}{\text{Nº de Acessos}}$$

Tipos de cache miss

- compulsórios (*cold start* ou chaveamento de processos, primeira referência): primeiro accesso a uma linha
  - é um “fato da vida”: não se pode fazer muito a respeito
  - se o programa vai executar “bilhões” de instruções, *misses* compulsórios são insignificantes
- de conflito (ou colisão)
  - múltiplas linhas de memória acessando o mesmo conjunto da cache conjunto-associativa ou mesma linha da cache com mapeamento direto
  - solução 1: aumentar tamanho da cache
  - solução 2: aumentar associatividade
- de capacidade
  - cache não pode conter todas as linhas accessadas pelo programa
  - solução: aumentar tamanho da cache
- invalidação: outro processo (p.ex. I/O) atualiza memória

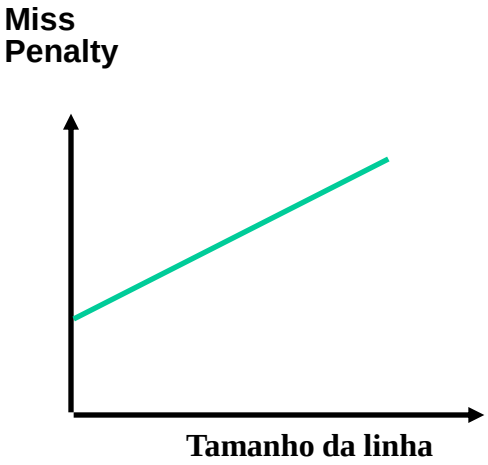
Tamanho da linha vs. miss ratio

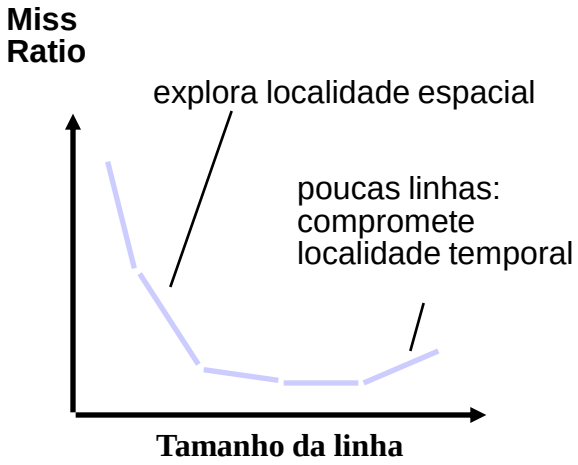


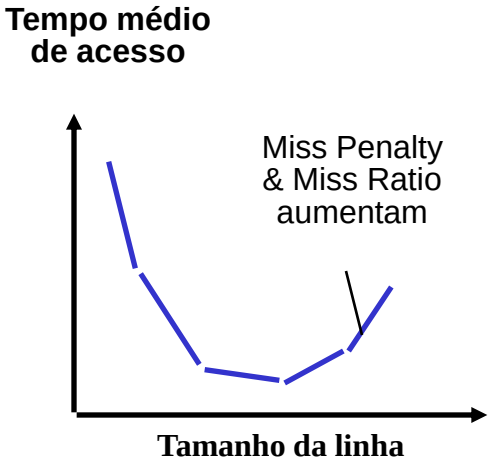
Tamanho da linha

- em geral, uma linha maior aproveita melhor a localidade espacial MAS
  - linha maior significa maior *miss penalty*
    - demora mais tempo para preencher a linha
  - se tamanho da linha é grande demais em relação ao tamanho da cache, *miss ratio* vai aumentar
    - muito poucas linhas
- em geral, tempo médio de acesso =

  Hit Time x (1 - Miss Ratio) + Miss Penalty x Miss Ratio







Quantos bits tem a cache no total?

- supondo cache com mapeamento direto, com 64 kB de dados, linha com uma palavra de 32 bits (4 bytes), e endereços de 32 bits
- 64 kB -> 16 kpalavras, 2^{14} palavras, neste caso 2^{14} linhas

Índice e offset

- cada linha tem 32 bits de dados mais um tag (32-14-2 bits) mais um bit de validade:

  2^{14} x (32 + 32 -14 -2 +1) = 2^{14} x 49 = 784 x 2^{10} = 784 kbits
- 98 kB para 64 kB de dados, ou 50% a mais

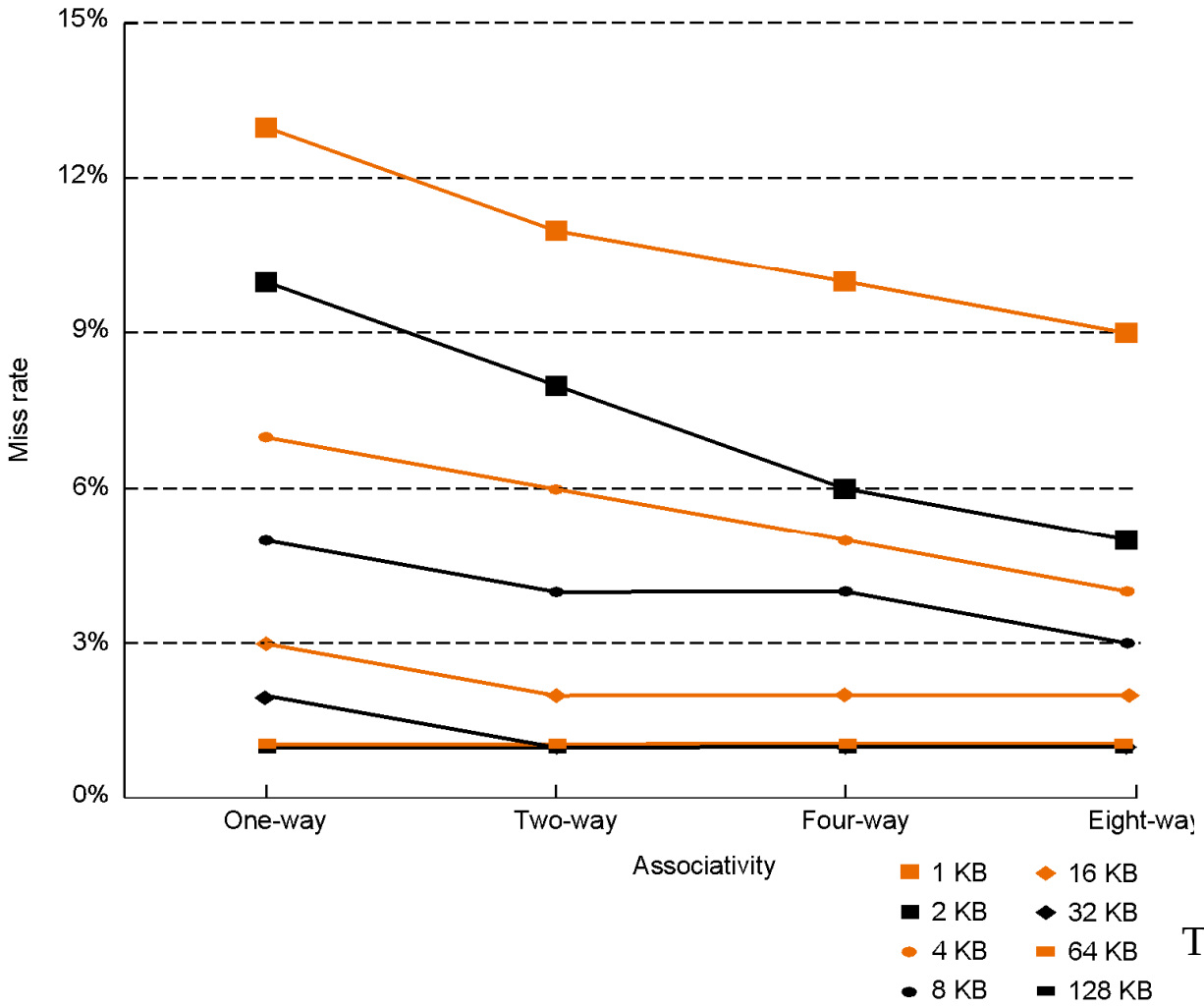
Quantos bits tem a cache no total?

- supondo cache com mapeamento direto, com 16 kB de dados, blocos de 4 palavras, sendo cada palavra de 32 bits e endereços de 32 bits
- 16 kB -> 4 kpalavras, 2^{12} palavras
- Bloco de 4 palavras (2^2), 2^{10} blocos (linhas)
- cada bloco tem 32 bits x 4 = 128 bits de dados mais um tag (32-10-2-2 bits) mais um bit de validade:

  2^{10} x (128 + 32 -10 -2 -2 +1) = 2^{10} x 147 = 147 kbits

- 18.4 kB para 16 kB de dados, ou 15% a mais

Impacto da associatividade



Localidade temporal

- usualmente encontrada em laços de instruções e acessos a pilhas de dados e variáveis
- é essencial para a eficiência da memória cache
- se uma referência é repetida N vezes durante um laço de programa, após a primeira referência a posição é sempre encontrada na cache
- quanto maior o número de acessos, menor o tempo médio de acesso.

Tc = tempo de acesso à cache
Tm = tempo de acesso à memória principal
Tce = tempo efetivo de acesso à cache

$$Tce = \frac{N\,Tc + Tm}{N} = Tc + \frac{Tm}{N}$$

se Tc = 1 ns, Tm = 20 ns, N = 10 ⇨ Tce = 3 ns
N = 100 ⇨ Tce = 1,2 ns

Localidade espacial

- memória principal é entrelaçada
- uma linha é transferida num único acesso entre a memória principal e a cache, através de um largo barramento de dados
- casamento entre tempo de acesso da cache e da memória principal

M = número de módulos da memória principal

Tc = tempo de acesso à cache

Tm = tempo de acesso à memória principal

Ideal: Tm = M Tc

Localidade espacial

- tempo médio de acesso a um byte, na primeira referência
  = $MTc_{memória} + MTc_{cache} = (2\ M\ Tc) / M = 2\ Tc$
- se cada byte é referenciado N vezes na cache, então o tempo efetivo (médio) de acesso *Tce* a cada byte é

$$\mathbf{Tce = \frac{2\ Tc + (N - 1)\ Tc}{N} = \frac{(N + 1)\ Tc}{N}}$$

se **Tc = 1 ns**
Tm = 20 ns
N = 10 **N = 100**

⇩ ⇩

então **Tce = 1,1 ns** **Tce = 1,01 ns**

Impacto no desempenho

Medindo o impacto do *hit ratio* no tempo efetivo de acesso

Tc = tempo de acesso à memória cache
Tm = tempo de acesso à memória principal
Tce = tempo efetivo de acesso à memória cache, considerando efeito dos misses

Tce = h Tc + (1 – h) Tm

se Tc = 1 ns, Tm = 20 ns

h =	**0.85**	**0.95**	**0.99**	**1.0**
	⇩	⇩	⇩	⇩
então Tce =	**3.85 ns**	**1.95 ns**	**1.19 ns**	**1 ns**

Impacto no desempenho

- Supondo um processador que executa um programa com:
  - CPI = 1.1
  - 50% aritm/lógica, 30% load/store, 20% desvios
- Supondo que 10% das operações de acesso a dados na memória sejam *misses.* Cada *miss* resulta numa penalidade de 50 ciclos.

CPI ideal	**1.1**
Data misses	**1.5**
Instr.misses	**0.5**

CPI = CPI ideal + nº médio de stalls por instrução
= 1.1 ciclos + (0.30 acessos à memória / instrução
x 0.10 misses / acesso) x (50 ciclos / miss)
= 1.1 ciclos + 1.5 ciclos
= 2. 6

- 58 % do tempo o processador está parado esperando pela memória!
- um miss ratio de 1% no fetch de instruções resultaria na adição de 0.5 ciclos ao CPI médio

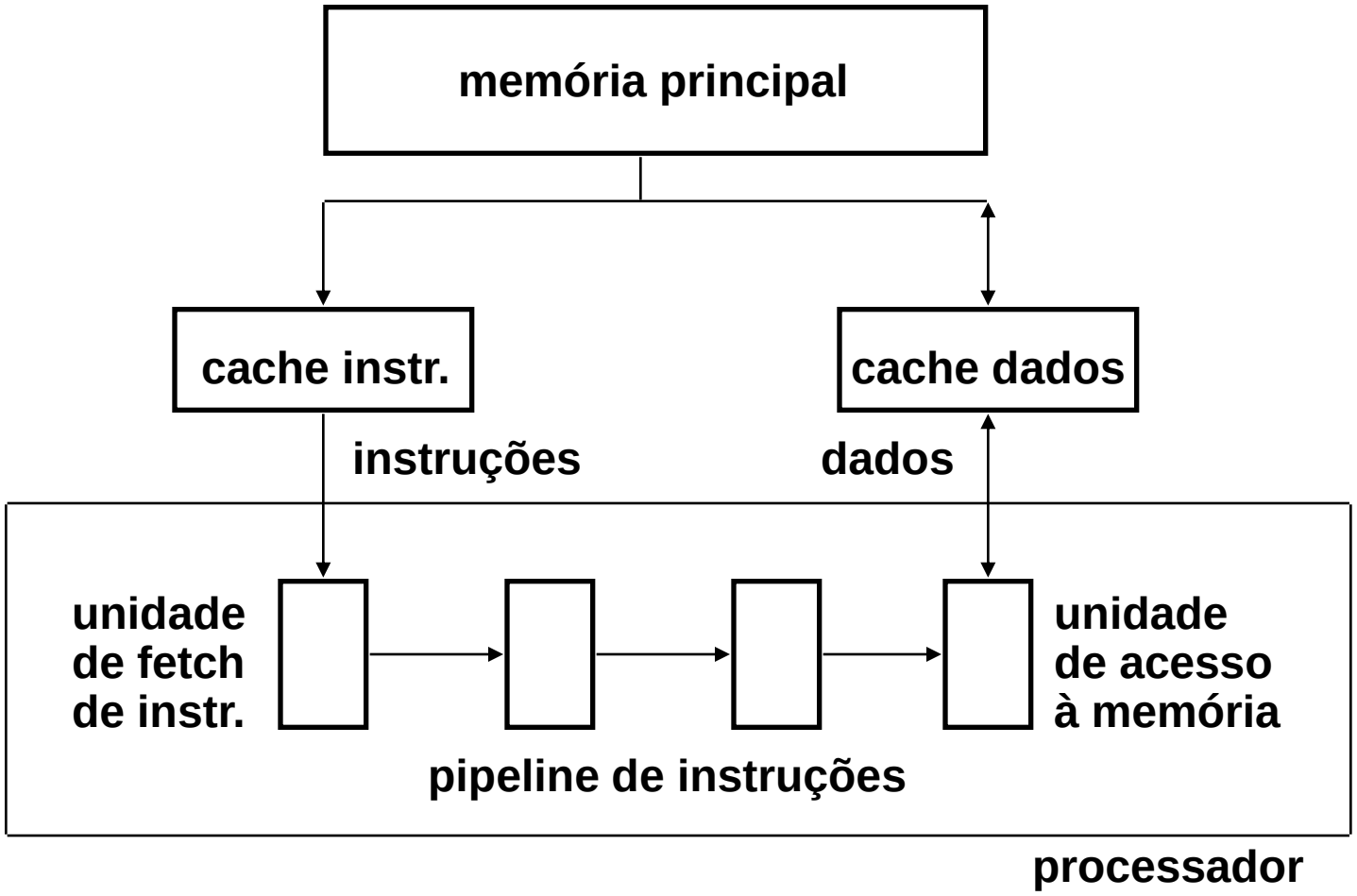
Hierarquia de caches

- caches integradas dentro de um processador têm limitação de tamanho
- *miss penalty* na cache é muito grande, pela diferença entre os tempos de acesso da cache e da memória principal
- solução: caches em dois ou três níveis
  - cache integrada (L1, de primeiro nível) é de tamanho pequeno, p.ex. 8 kbytes, e tempo de acesso menor
  - cache secundária (L2) tem tamanho maior, p.ex. 256 kbytes, e tempo de acesso maior
- cache de terceiro nível (L3)
  - cache L3 “fora” do chip do processador, cache L2 dentro
- *misses* podem ocorrer em referências a qualquer nível de cache
- transferências entre níveis de cache apresentam mesmos problemas e possíveis soluções já discutidos

Caches de dados e instruções

- dados e instruções: cache unificada x caches separadas
- vantagens das caches separadas
  - política de escrita só precisa ser aplicada à cache de dados
  - caminhos separados entre memória principal e cada cache, permitindo transferências simultâneas (p.ex. num pipeline)
  - estratégias diferentes para cada cache: tamanho total, tamanho de linha, organização
- caches separadas são usadas na maioria dos processadores, no nível L1
- caches unificadas nos níveis L2 e L3

Caches de dados e instruções



Desempenho em caches multinível

- Suponha que o processador tenha um CPI de 1,0 e que todas as referencias acertem na cache primária a uma velocidade de clock de 5GHz (0,2ns). O tempo de acesso à memória principal é de 100ns com todos os tratamentos de faltas. Taxa de falhas por instrução na cache primária é de 2%. O quanto mais rápido será o processador se acrescentarmos uma cache secundária com tempo de acesso de 5ns para um acerto ou uma falha e que seja grande o suficiente para que a taxa de falhas na L2 seja de 0,5%?

Desempenho em caches multinível

- Penalidade de falha para memória principal:
  - 100ns/0,2ns = 500 ciclos de clock.
- Para processador com apenas L1:
  - CPI total = 1,0 + ciclos de stall de memória por isntrução = 1,0 + 2% x 500 = 11,0
- Em relação a L1, penalidade de falha para L2:
  - 5ns / 0,2ns = 25 ciclos de clock
- Para cache de dois níveis:
  - CPI total = 1 + stall L1 + stall L2 = 1 + 2%x25 + 0,5%x500 = 1+0,5+2,5 = 4,0
- Portanto, com cache L2:
  - 11,0 / 4,0 = 2,8 vezes mais rápido

Memória Virtual

Introdução

- memória principal semicondutora
  - capacidade limitada
  - tempo de acesso entre 10 e 20 ns
- memória secundária em disco
  - capacidade muito maior
  - tempo de latência entre 10 e 30 ms

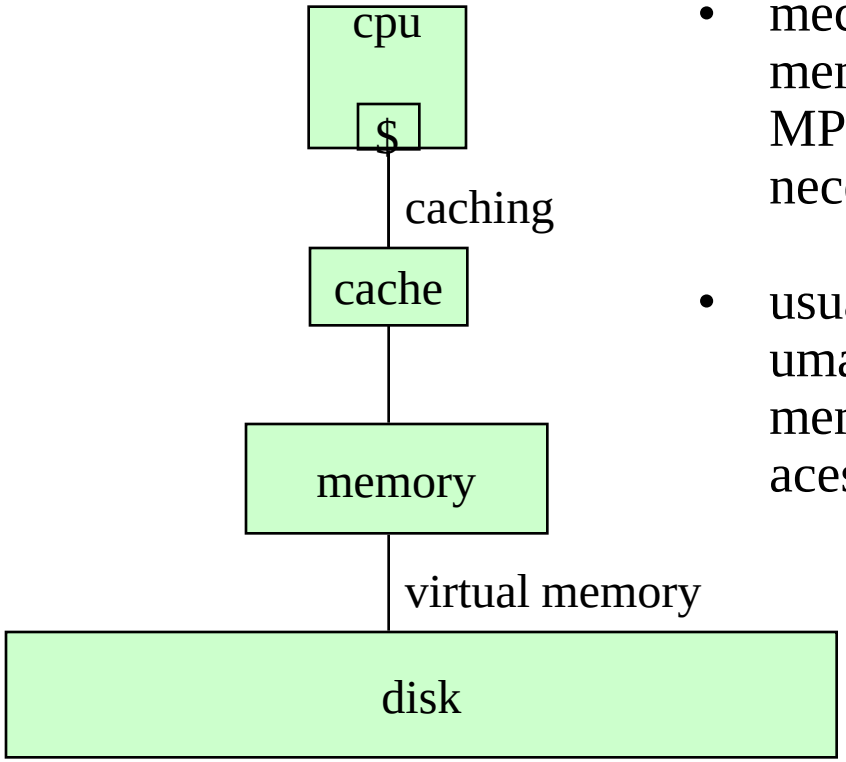
O problema

- Nosso computador tem 32 kbytes de memória principal

- Como podemos:
  - rodar programas que usam mais do que 32 kbytes?
  - permitir que vários usuários usem o computador?
  - executar vários programas ao mesmo tempo?

Memória Virtual: a solução!

- *Memória Virtual :* técnica que nos permite ver a memória principal como uma cache de grande capacidade de armazenamento
- É apenas mais um nível na hierarquia de memórias



- mecanismo automático de gerência de memória, que traz automaticamente para a MP os blocos de informação (do disco) necessários
- usuário tem a impressão de trabalhar com uma memória única, do tamanho da memória secundária, mas com tempo de acesso próximo do tempo da MP

Tempo de acesso

Tempo médio de acesso Tma é dado por

$$Tma = Tm + (1 – h) Ts$$

onde Tm = tempo de acesso à MP
Ts = tempo de acesso ao disco
h = hit ratio

p.ex. se Tm = 20 ns, Ts = 20 ms, h = 0.9999
então Tma = 2,02 µs (100 x maior do que Tm)

Por que MV é diferente das caches?

- Miss penalty é MUITO maior (milhões de ciclos)! Se informação não está na memória, está no disco!

- Logo:
  - *miss ratio* precisa ser bem menor do que em cache
  - alta penalidade do *miss* => necessário buscar blocos maiores em disco
  - princípio de localidade opera sobre blocos maiores de dados ou instruções e leva a *hit ratios* bem mais elevados
  - Mapeamento totalmente associativo das páginas
  - *misses* são tratados por software (há tempo disponivel)
  - técnica de escrita *write-through* não é uma opção. Usa-se *write-back.*

Terminologia

- **mesma idéia da cache, mas com terminologia diferente**

cache	**MV**
bloco	**página (ou segmento)**
cache miss	***page fault***
endereço	**endereço virtual (ou lógico)**
índice	**endereço real (ou físico)**

- **endereço *virtual (lógico)*: gerado pelo programa**
  - **deve endereçar todo espaço em disco**
  - **maior número de bits**

- **endereço *real (físico)*: endereço na memória principal**
  - **menor número de bits**

Unidade de Gerenciamento de Memória

- MMU (*Memory Management Unit*)
  - gerência da hierarquia de memória
  - proteção de memória
  - usualmente integrada dentro do microprocessador

- MMU deve fazer mapeamento do endereço virtual para endereço real

- SO usa a MMU

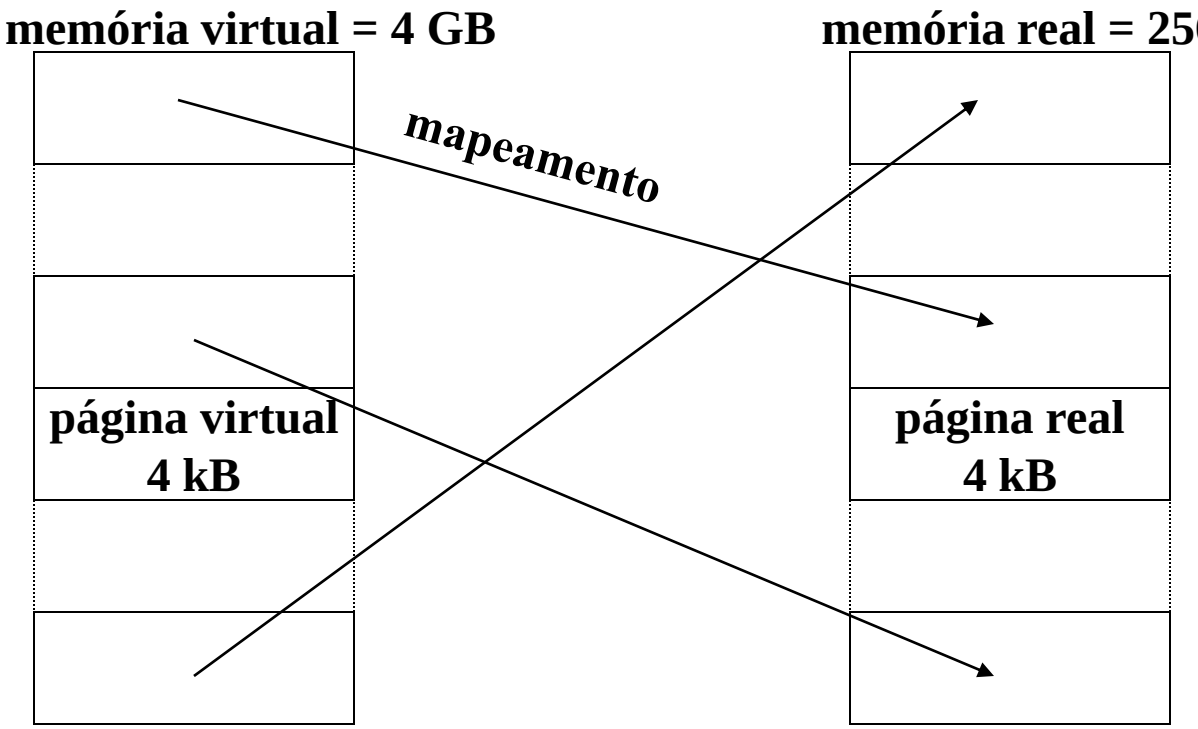
Paginação

- Por que paginação? Resposta: mecanismo simples para tradução de endereços virtuais em reais e para gerenciamento do espaço de memória

- espaços de memória real e virtual divididos em blocos chamados de *páginas*
  - páginas tem tipicamente de 4 kbytes a 16 kbytes
  - Páginas para sistemas embarcados são de 1 kbytes

- endereços virtuais e reais divididos em 2 campos
  - endereço da página
  - endereço da linha (ou palavra), dentro da página

Paginação



1 M páginas de 4 kB

nº página = 20 bits	12 bits

Tamanho de página

endereço virtual = 32 bits

64 k páginas de 4 kB

nº página = 16 bits	12 bits

Tamanho de página

endereço real = 28 bits

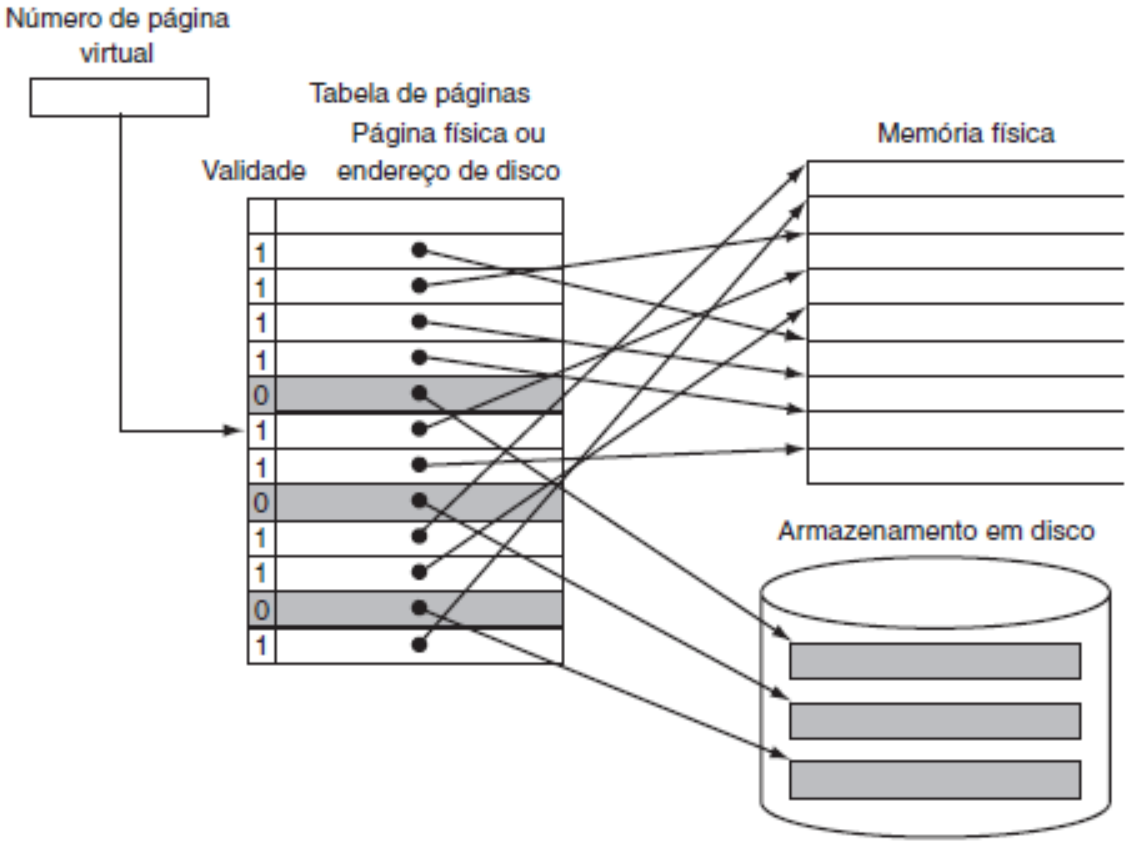
Paginação

- *page fault* ocorre quando a página virtual não está na memória principal
- mapeamento completamente associativo, mais eficiente, ajuda a diminuir alta penalidade dos *page faults*

- *Como transformar endereçamento original do programa no endereçamento real?*

- *page tables*
  - guardam a correspondência entre páginas virtuais e páginas reais
  - permitem a translação de endereços

Paginação



- É apenas uma função de mapeamento dos endereços virtuais (do disco) para endereços reais (físicos) na memória principal

Gerência de processos

- cada processo tem sua própria tabela de páginas
  - processos são compilados para espaços de endereçamento virtuais
  - tabela de páginas define toda a utilização do espaço de endereçamento pelo processo

- sistema operacional é responsável pela alocação de espaço físico para o espaço virtual de cada processo
  - SO carrega tabela de páginas de cada processo

- hardware possui registrador que aponta para início da tabela de páginas do processo atual

- quando novo processo passa a ser ativo, sistema operacional só precisa atualizar valor deste registrador

Paginação

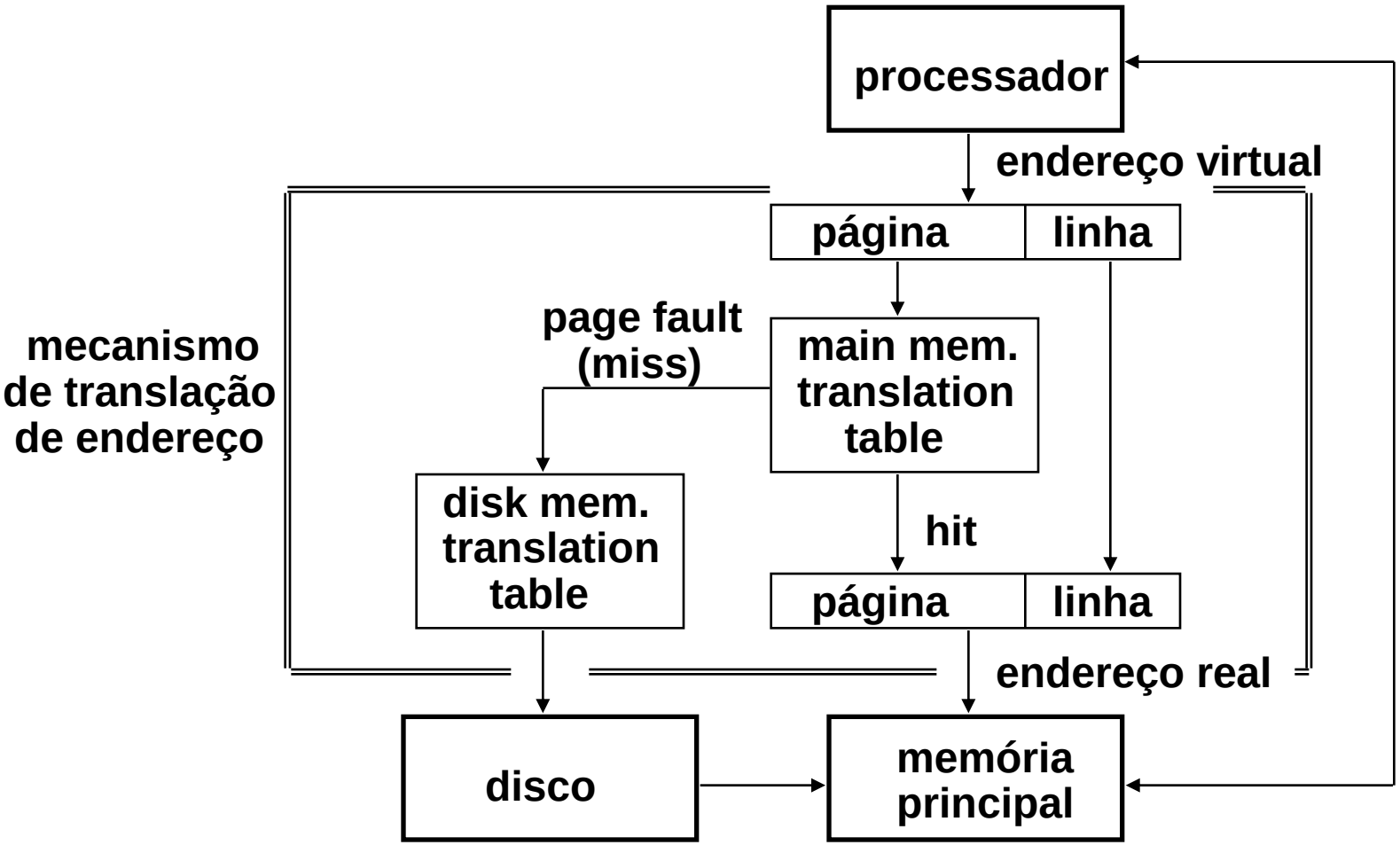
- *main memory translation table (MMTT)*
  - implementada em hardware
  - tamanho = nº de páginas na memória principal

- *disk memory translation table (DMTT)*
  - implementada em software, armazenada na memória principal
  - tamanho = nº de páginas em disco

- algoritmo de substituição, em software, para selecionar página da memória principal a ser substituída em caso de *page fault*

- bom desempenho é garantido pelo princípio de localidade

Paginação


processador
endereço virtual
página
linha
mecanismo
de translação
de endereço
page fault
(miss)
main mem.
translation
table
disk mem.
translation
table
hit
página
linha
endereço real
disco
memória
principal

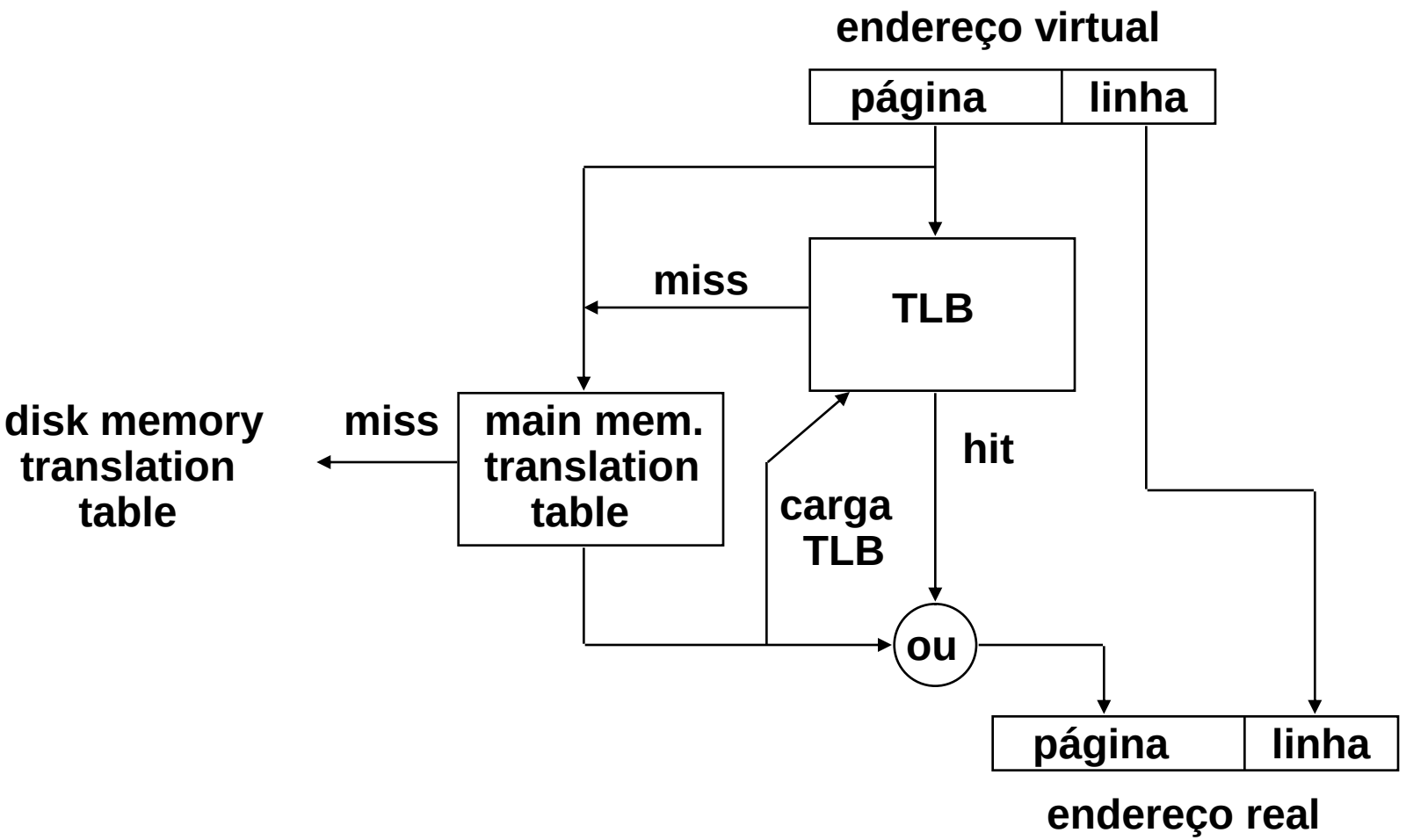
Tamanho de páginas

- tamanhos de páginas variam muito, de 64 bytes a 4 Mbytes
- página de pequeno tamanho
  - tempo curto para transferência de página entre disco e memória
  - muitas páginas de diferentes programas podem estar residentes em memória
  - exige *page tables* muito grandes, que ocupam espaço em memória
  - mais adequada para instruções
- página de grande tamanho
  - *page tables* pequenas
  - tempo longo para transferência de página entre disco e memória
  - mais adequada para dados

Translation Look-Aside Buffer

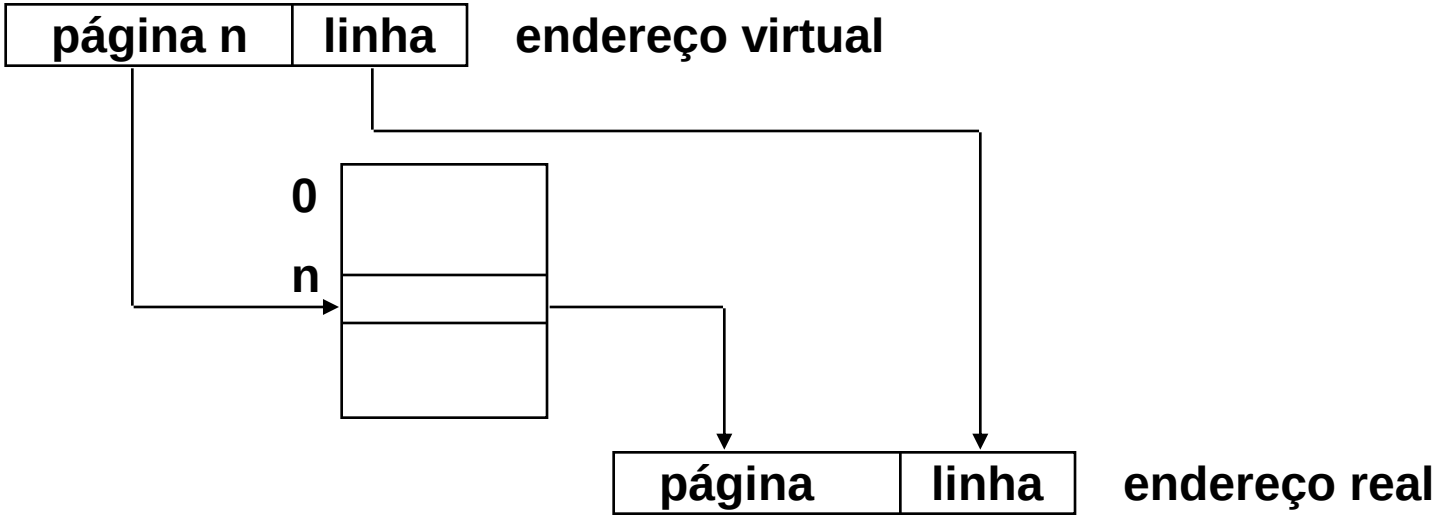
- nº de páginas na memória secundária é muito grande
  - espaço virtual de 2^{32}bytes, páginas de 4k bytes, 4 bytes por entrada na tabela
  - 4 MBytes apenas para a tabela de páginas!!!
  - tamanho excessivo da *main memory translation table*
- se tabela ficar na memória principal => dois acessos à memória a cada *cache miss*
- *working set* = conjunto de páginas mais prováveis de serem acessadas num dado momento, devido ao princípio de localidade
- *Translation Look-Aside Buffer* (TLB)
  - implementado em hardware
  - traduz endereços virtuais para endereços reais
  - só inclui páginas do *working set*
  - pode ser considerado como uma "cache" da MMTT
- Main Memory Translation Table (MMTT)
  - implementada em software

Translation Look-Aside Buffer



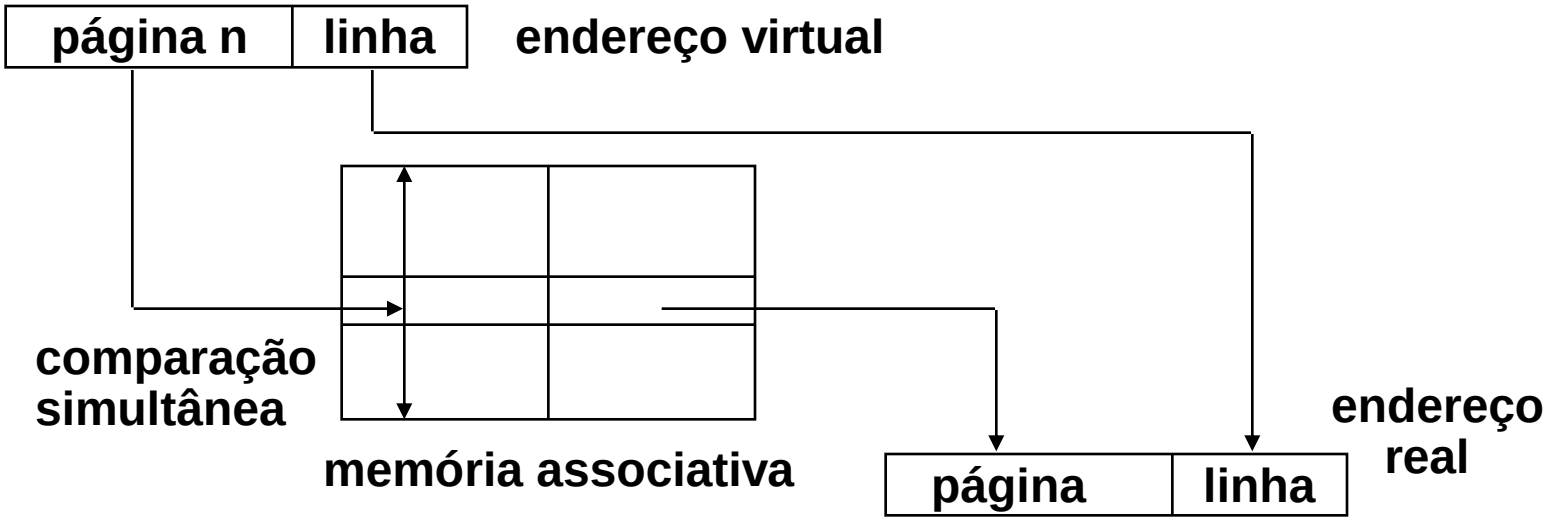
Mecanismos de translação de endereços mapeamento direto

- endereço de página virtual é utilizado como endereço de uma memória cujo conteúdo é o endereço de página real procurado
- tamanho = nº de páginas na memória virtual
- utilizado na MMTT (em software), mas não na TLB

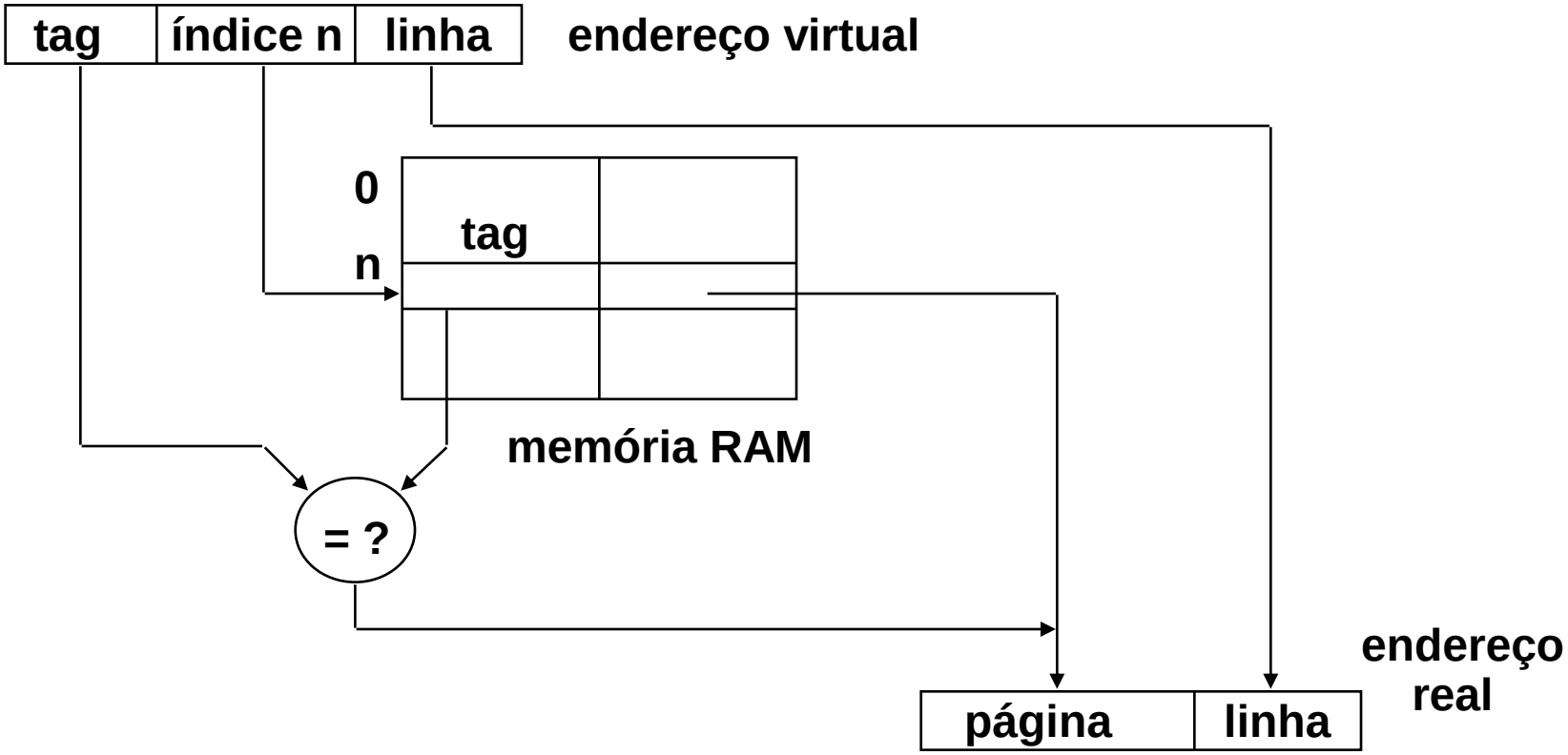


Mecanismos de translação de endereços mapeamento completamente associativo

- memória associativa contém endereços virtual e real
- comparação simultânea com todos os endereços virtuais



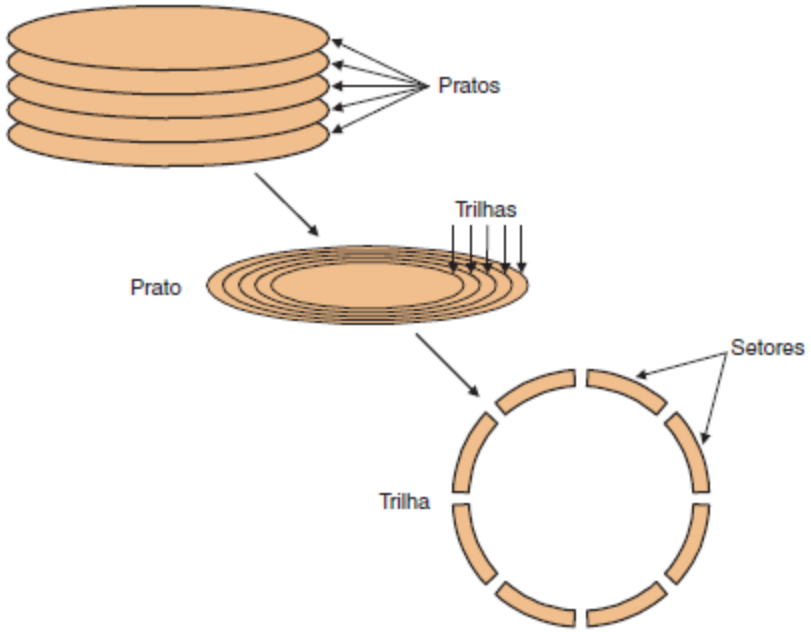
Mecanismos de translação de endereços mapeamento conjunto – associativo (*1-way*)



Mecanismos de translação de endereços mapeamento conjunto – associativo

- endereços divididos em 3 campos: tag, índice, linha
- endereço da página = tag e índice
- 1–way associativo: cada posição da tabela contém um par < end. página virtual, end. página real >
  - apenas um comparador
  - endereços de páginas virtuais armazenados na tabela têm índices diferentes
- n–way associativo: cada posição da tabela contém n pares de endereços de página
  - n comparadores
  - n endereços de páginas virtuais armazenados na tabela têm mesmo índice

Disco



- **Para acessar dados:**
  - **busca: posiciona a cabeça sobre a trilha correta (3 a 14 ms em média)**
  - **latência rotacional: espera pelo setor desejado (0,5 rpm)**
  - **transferência: recupera os dados (um ou mais setores; 30 a 80 MB/seg)**

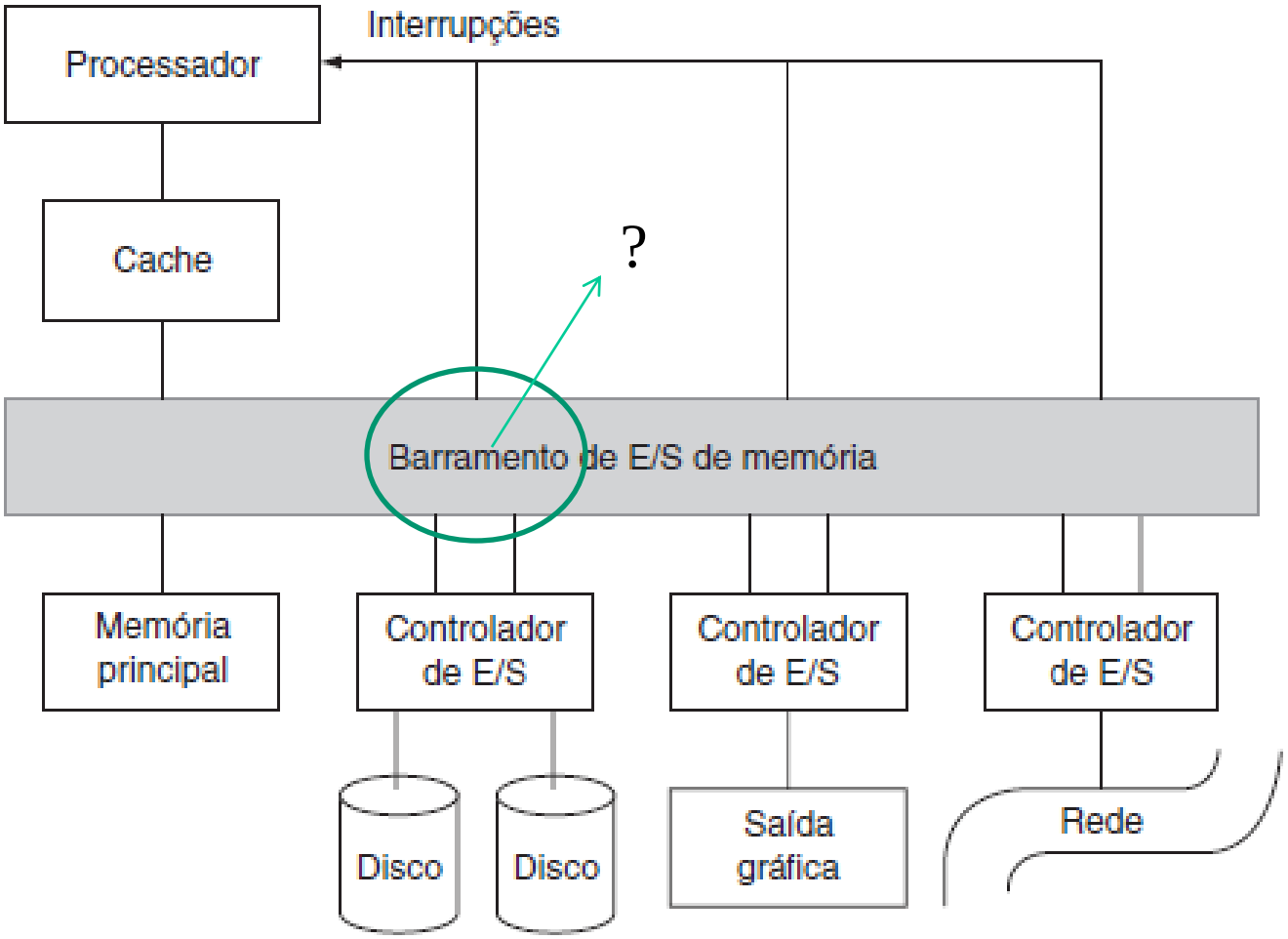
Desempenho do Disco

Tempo de Disco = Tempo de busca + Latência rotacional + Tempo de transferência

A latência rotacional média para a informação desejada está a meio caminho ao redor do disco.

- TB = 1ms
- LRM = $\frac{0{,}5 \text{ rotação}}{5400 \text{ RPM}} = \frac{0{,}5 \text{ rotação}}{5400\text{RPM}/60(\text{seg./min.})} = 0{,}0056\text{s} = 5{,}6\text{ms}$
- TT = 1kB / 100MB/s = $1*2^{10} / 50*10^{6}$ = 20,48µs
- TD = 1ms + 5,6ms + 0,02048ms = 6,62048ms

Visão geral


Processador
Interrupções
Cache
?
Barramento de E/S de memória
Memória principal
Controlador de E/S
Controlador de E/S
Controlador de E/S
Disco
Disco
Saída gráfica
Rede