Tópicos sobre Arquitectura de Computadores -- 5

José A. Cardoso e Cunha
DI-FCT/UNL

1. Arquitectura de Von Neumann
2. Programação em linguagem ´´máquina´´
3. Sistema de entradas e saídas
4. Hierarquia das unidades de memória
5. Organização interna do processador

Hierarquia de memórias

As memórias: colecções de unidades dispersas no espaço e em função, com tempos de acesso, capacidade e custo variáveis.

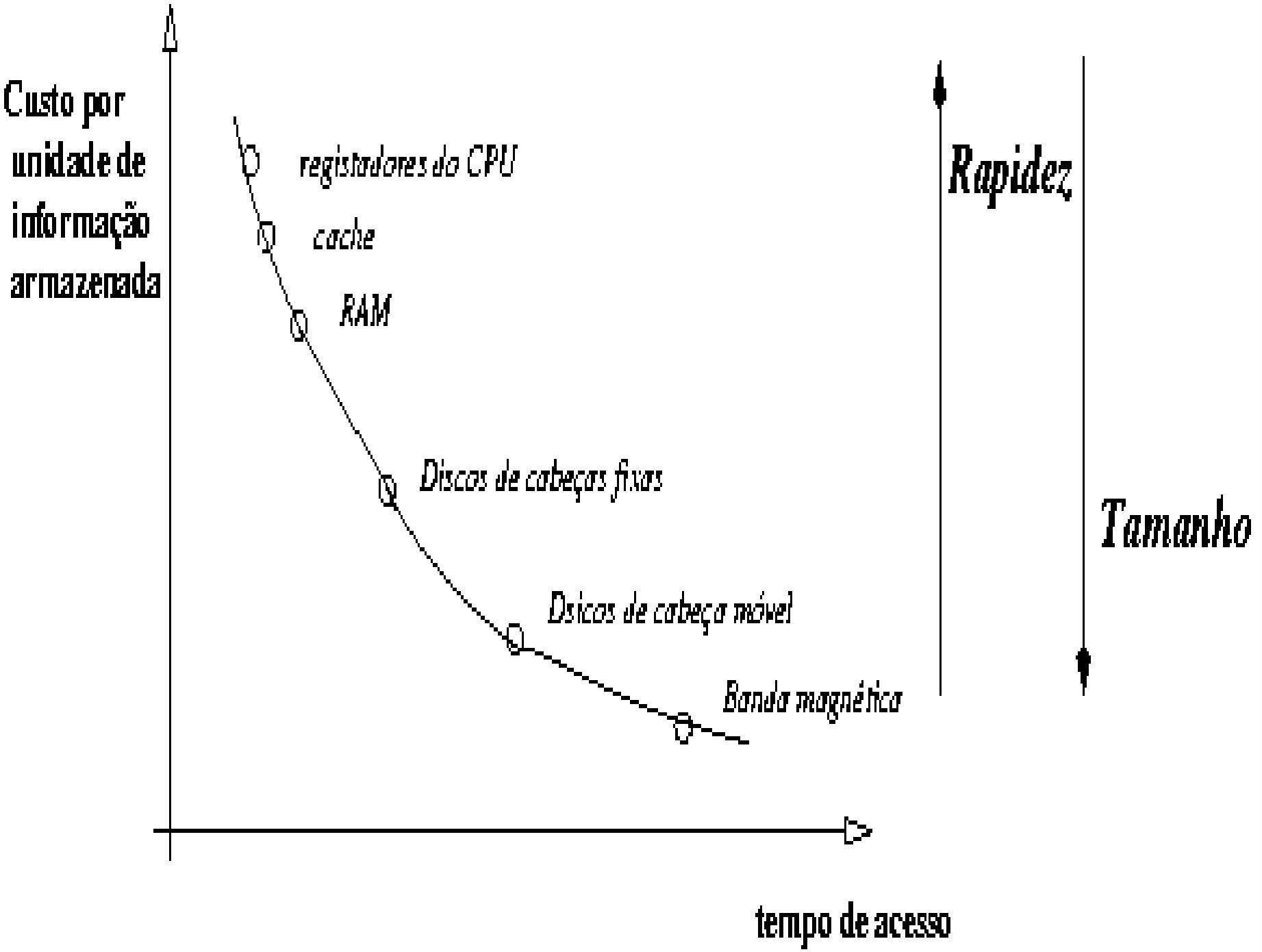
Memória: colecção finita de células, logicamente organizadas.

Cada célula:

- Endereço
- Conteúdo ou valor

Tipos de acesso:

- Por endereço
- Por conteúdo (acesso associativo)


Custo por unidade de informação armazenada
registadores do CPU
cache
RAM
Discos de cabeças fixas
Dsicos de cabeça móvel
Banda magnética
Rapidez
Tamanho
tempo de acesso

Memória central ou principal: acessível directamente pelo CPU, no passo FETCH e na execução de instruções de referência de memória.

Memória secundária ou auxiliar: acessível pelas instruções de entrada e saída; suportadas por dispositivos para armanazemento de ficheiros.

Unidades de acesso:

bytes, múltiplo de bytes

blocos de bytes (páginas, sectores ou blocos)

Modos de acesso:

a) Tempo de acesso constante

À memória central ou RAM: acesso a qualquer célula num intervalo de tempo fixo e conhecido

b) Acesso sequencial:

Só se acede, a cada momento, à célula adjacente relativamente à célula corrente N:

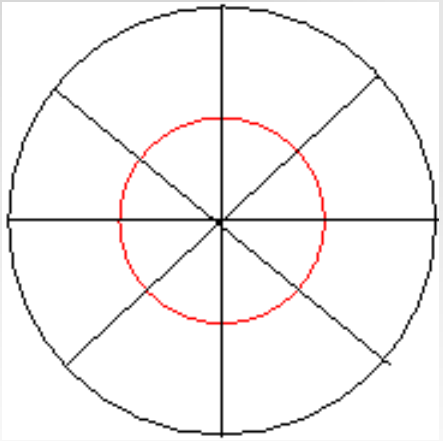
para aceder à célula com endereço M têm de se percorrer todas as células intermédias desde N a M.

O tempo de acesso é uma função linear da distância em endereços de N a M

d * constante

c) Acesso directo:

acede-se a qualquer célula, dando o seu endereço, mas o tempo de acesso é uma função não-linear da distância entre o destino e célula corrente.



Memória central:

ROM (Read Only Memory): para parte do software do SO

RAM:

- Dinâmica (DRAM): exige um sinal periódico de ´´refresh´´
- Estática (SRAM): não exige ´´refresh´´

Comparação entre Memória Dinâmica e Estática

	Capacidade	Tempo Acesso	Custo
Dinâmica	Maior	Maior	Menor
Estática	Menor	Menor	Maior

Rapidez do CPU:

- Depende do ciclo do processador / frequência do relógio
- Frequências de 1 GigaHz → ciclo de 1Nanosegundo

Desempenho da execução de um programa:

- Depende de:
  - Número e tipo de instruções executadas
  - Número de ciclos do processador, por cada instrução

Memória:	estática	dinâmica
Tempo de acesso	10 nanoseg.	50 nanoseg.
Tempo de ciclo	= t. acesso	2 * t. acesso

Para um CPU com relógio interno de 2 GigaHz, exigir-se-ia uma memória RAM com 0.5 Nanoseg…

Não sendo possível,

o CPU entra em estados de espera (´´wait states´´) quando tem de aguardar leituras/escritas de memória.

<u>O que fazer?</u>

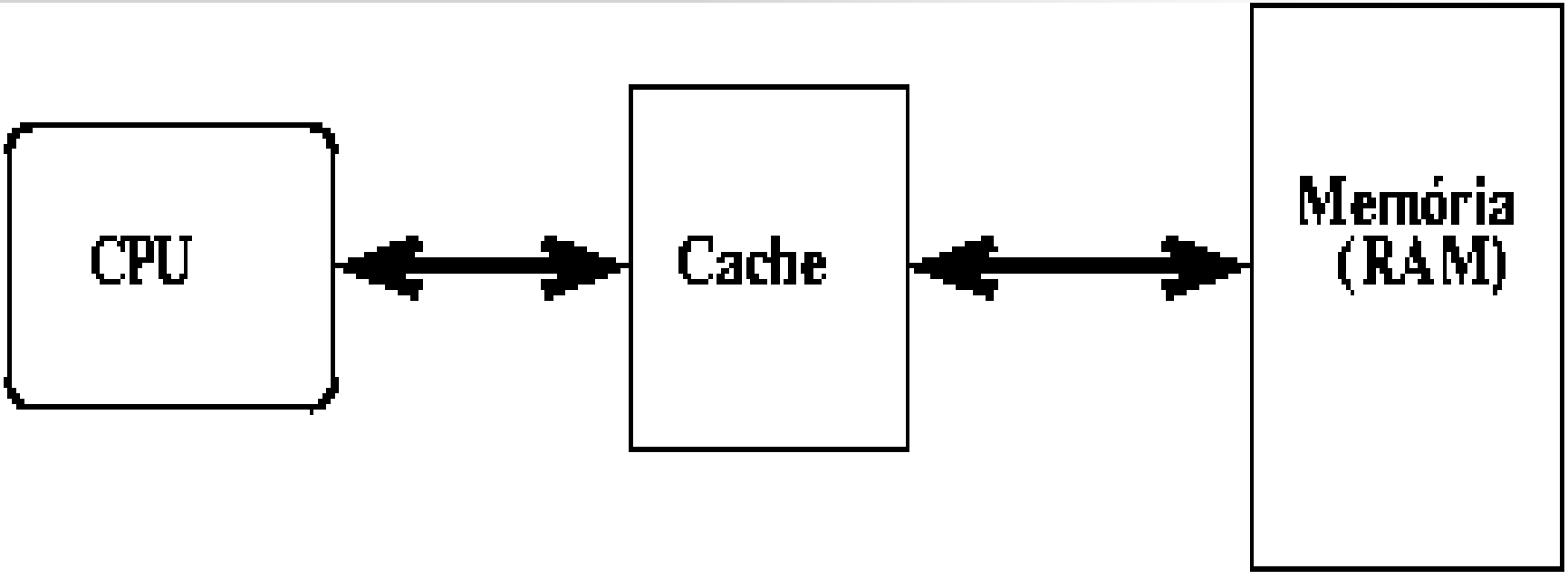
a) Dividir a memória em múltiplos bancos, acessíveis em paralelo (´´interleaved memory´´)

b) Memória ´´Cache´´: uma memória intermédia entre CPU e RAM

Memória central hierarquizada em 2 níveis:

- um tem acesso mais rápido e menor capacidade
- outro tem maior tempo de acesso e maior capacidade

→ O CPU deve ´´ver´´ uma memória de capacidade igual à RAM e com tempo de acesso igual à Cache.



Objectivo da Cache:

tentar manter o mais próximo possível do CPU, a informação que está a ser precisa no momento e a que o vai ser no futuro próximo…

- Explorar o Princípio da Localidade dos Programas:
  - Localidade espacial
  - Localidade temporal

As informações que serão referidas pelo CPU num futuro próximo, estão, muito provavelmente, localizadas em endereços de memória próximos dos das informações que foram referidas, no passado recente.

A memória Cache é carregada, a partir da Memória Central, com antecipação, sendo preenchida em pequenos grupos de bytes (*linhas* ou *blocos* da Cache), de cada vez que o CPU precisa de aceder a um byte em memória RAM.

Os blocos da Cache são pequenos múltiplos das linhas de dados do Bus (8, 16 ou 32 bytes).

Taxa de sucesso (hit ratio): h

-- indicador da probabilidade de que um byte endereçado pelo CPU seja encontrado em Cache

$$h = \frac{\text{número de referências encontradas na Cache}}{\text{número total de referências geradas pelo CPU}}$$

T. Inst. = T.proc. + (h * T.Cache + (1-h) * T.Mem)

Se h = 1 → todas as referências são encontradas na Cache

T.Inst = T.proc. + T.Cache

Se h = 0 → nenhuma referência é encontrada na Cache

T.Inst = T.proc. + T.Mem

Se h = 95% , T. Cache = 10 nanoseg. , T.Mem = 30 nanoseg.

T.Inst = T.proc. + (0.95*10 + 0.05*30) = T.proc + 11 nanoseg.

Em média, o sistema (Cache+Memória) comporta-se como uma memória com:

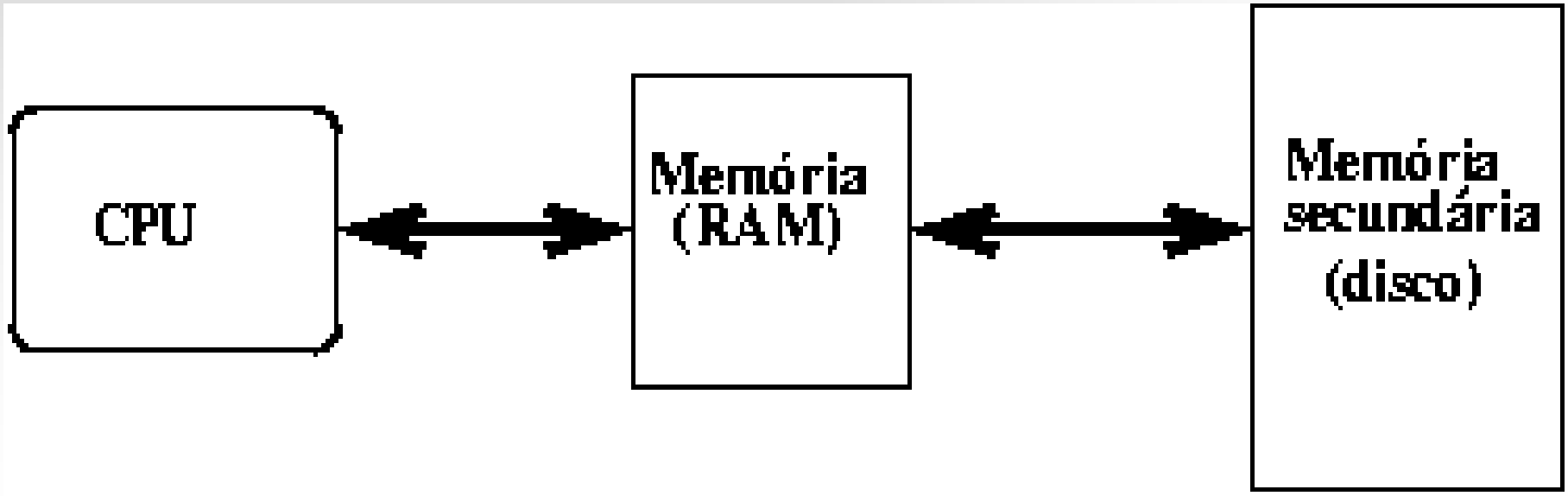
- T. Acesso = 11 nanoseg.
- Capacidade = Capacidade da Memória Central

Memória central hierarquizada em mais 2 níveis

RAM-- Disco:

- um tem acesso mais rápido e menor capaciddade
- outro tem maior tempo de acesso e maior capacidade

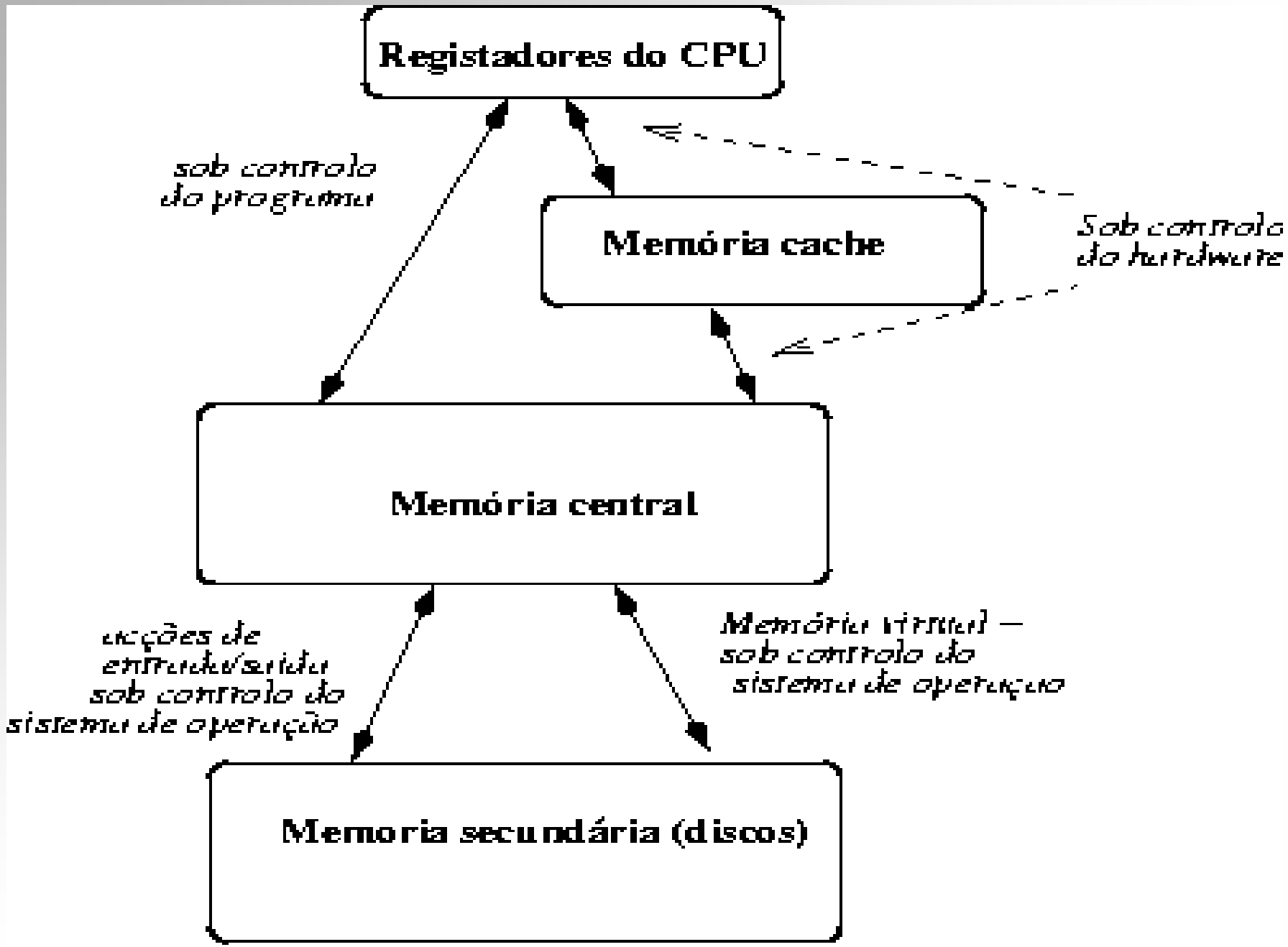
→ O CPU deve ´´ver´´ uma memória de capacidade igual ao DISCO e com tempo de acesso igual à RAM.



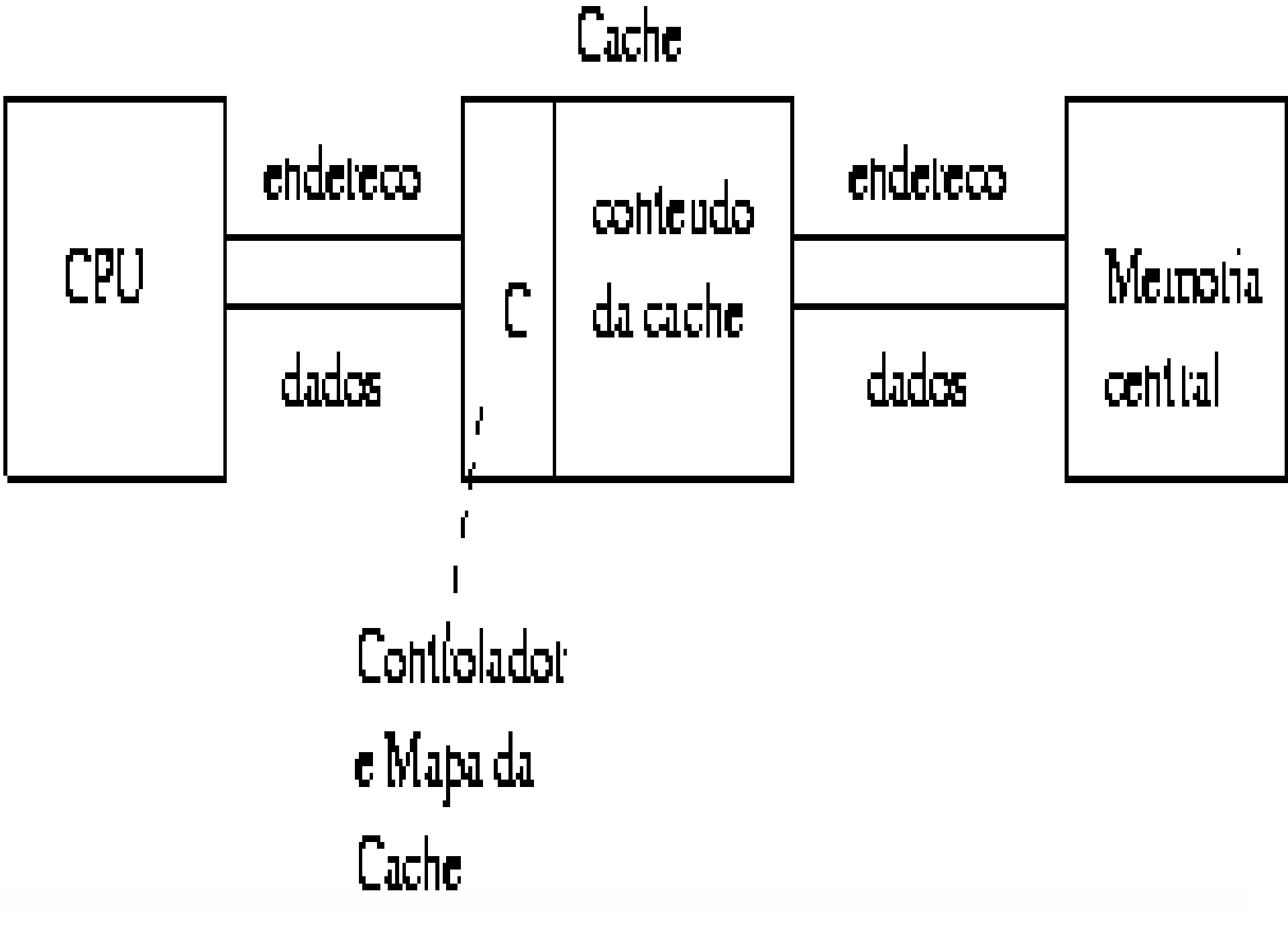
As informações que serão referidas pelo CPU num futuro próximo, estão, muito provavelmente, localizadas em endereços de memória próximos dos das informações que foram referidas, no passado recente.

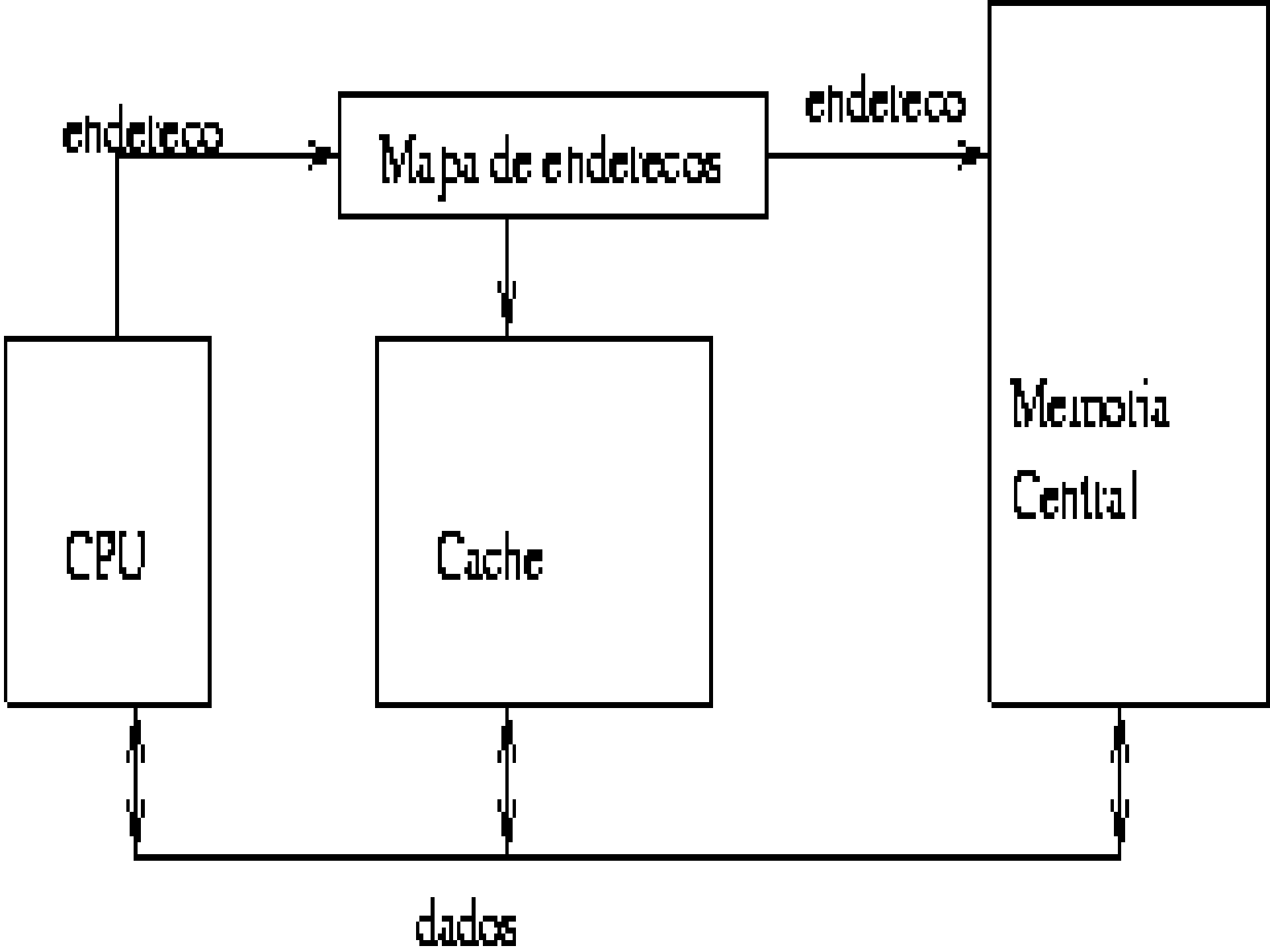
A memória Central é carregada, a partir do DISCO, com antecipação, sendo preenchida em blocos de bytes (*páginas de memória*), de cada vez que o CPU precisa de aceder a bytes em memória RAM que não estejam lá.

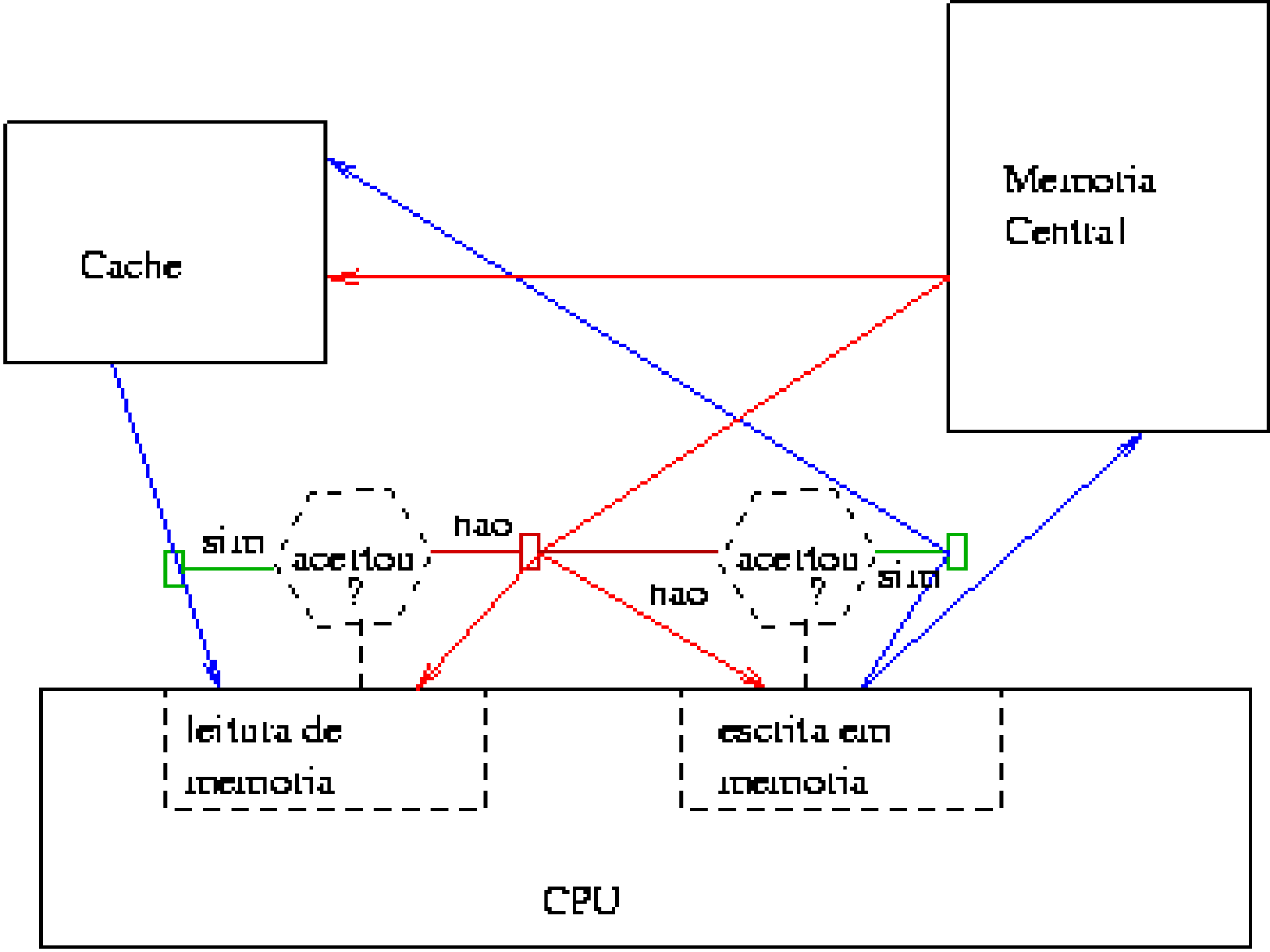
As páginas de Memória são da mesma ordem dos sectores ou blocos de bytes em Disco (4 Kilo Bytes).

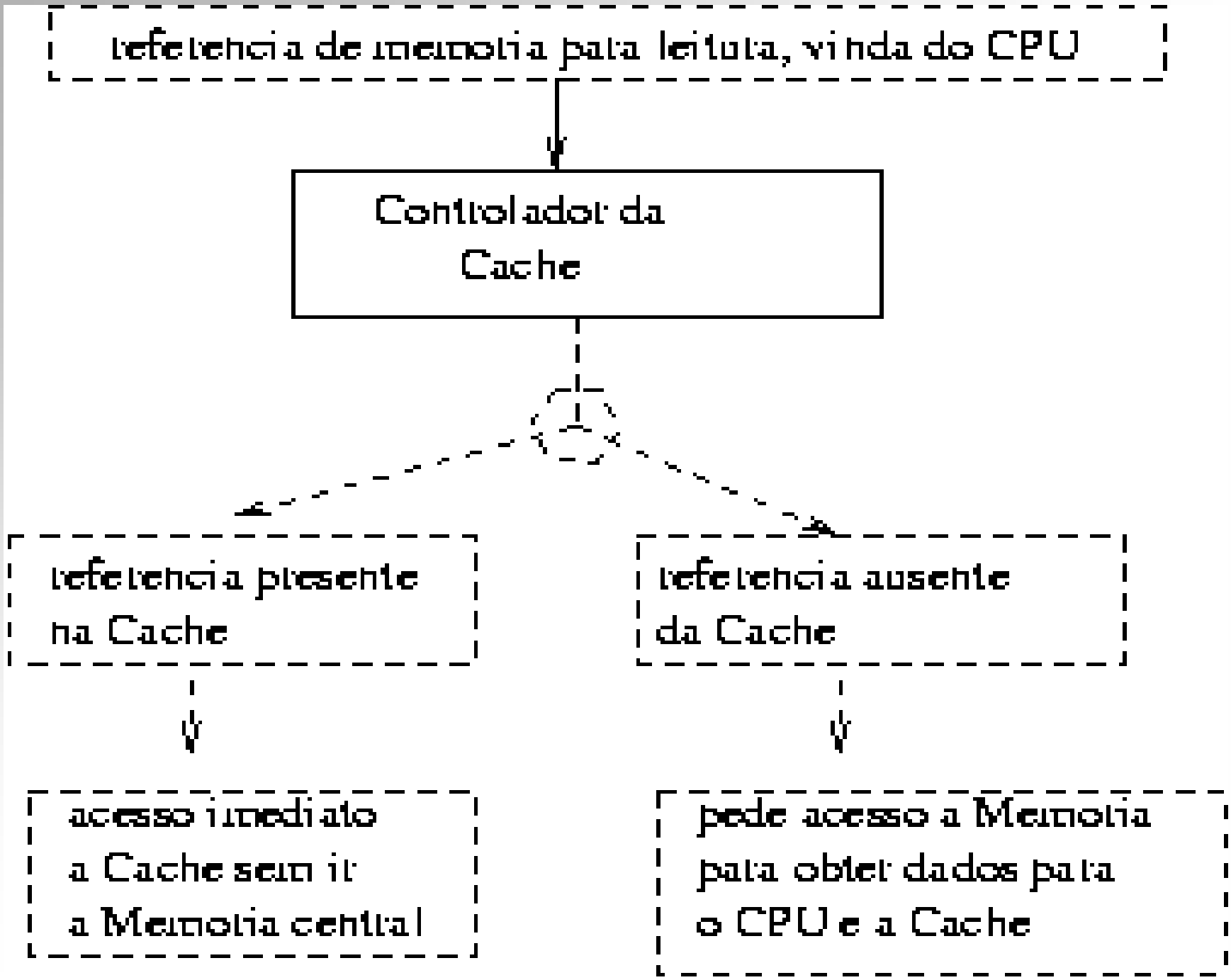

Registadores do CPU
sob controlo do programa
Memória cache
Sob controlo do hardware
Memória central
acções de entrada/saída sob controlo do sistema de operação
Memória virtual — sob controlo do sistema de operação
Memoria secundária (discos)

Memórias Cache


Cache
CPU
endereco
dados
C
conteudo
da cache
endereco
dados
Memoria
central
Controlador
e Mapa da
Cache


endereço
Mapa de endereços
endereço
Memória
Central
CPU
Cache
dados


Cache
Memoria
Central
sim
acertou
?
não
não
acertou
?
sim
leitura de
memoria
escrita em
memoria
CPU


referencia de memoria para leitura, vinda do CPU
Controlador da Cache
referencia presente na Cache
referencia ausente da Cache
acesso imediato a Cache sem ir a Memoria central
pede acesso a Memoria para obter dados para o CPU e a Cache

Referência de memória para escrita, pelo CPU:

1º método: WRITE-THROUGH – ESCRITA IMEDIATA: actualiza a Cache e a Memória

2º método: WRITE-OUT ou WRITE-BACK – ESCRITA ADIADA: só actualiza a Cache e adia a actualização em Memória até que aquele bloco de Cache tenha de ser copiado para memória:

- No fim do programa;
- Quando a Cache está cheia e aquele bloco tem de ser substituído por outro que esteja em Memória.

Limita-se a marcar o bloco, no mapa, com um indicador ´´dirty block´´

A Cache tem um modo de acesso associativo:

Deve detectar se contém uma certa informação

-- recebe o endereço de um bloco e diz se está lá ou não

-- com base numa estrutura que descreve o conteúdo da Cache a cada momento: Mapa de Endereços da Cache

O Mapa da Cache estabelece a correspondência entre:

Endereço real vindo do CPU →

→ Posição da Cache contendo o byte respectivo

→ ou indicação de que não está correntemente em Cache

O Mapa está implementado em hardware, nos circuitos do controlador da Cache

Convém transferir, entre Cache e Memória, blocos de alguns bytes, pequenos múltiplos das linhas de dados do Bus:

Exemplo: Bus com 32 linhas de dados → 4 bytes

blocos da Cache com 8 ou 16 bytes

´´prendem´´ o bus durante 2 ou 4 ciclos de acesso a Memória.

Exemplo: Cache com capacidade de 1 MegaByte e com blocos de 16 bytes

- Quantos blocos pode conter a Cache?
  - → $2^{20} / 2^4 = 2^{16}$ Blocos → 64 Kilo Blocos

E se a Memória Central tiver 1 Gigabyte?

- Quantos ´´blocos da Cache´´ cabem em Memória?
  - → $2^{30} / 2^4 = 2^{26}$ Blocos → 64 Mega Blocos

Há alguma alteração ao modo como o CPU gera o endereço e o envia para a memória?

→ NÃO.

Exemplo: se o Bus tem 32 linhas de endereços

se a Cache tem blocos de 16 bytes

Os endereços gerados pelo CPU são:

31 30 29 ……4 3 2 1 0 endereço de 32 bits

Os endereços gerados pelo CPU **são vistos pelo controlador da Cache**:

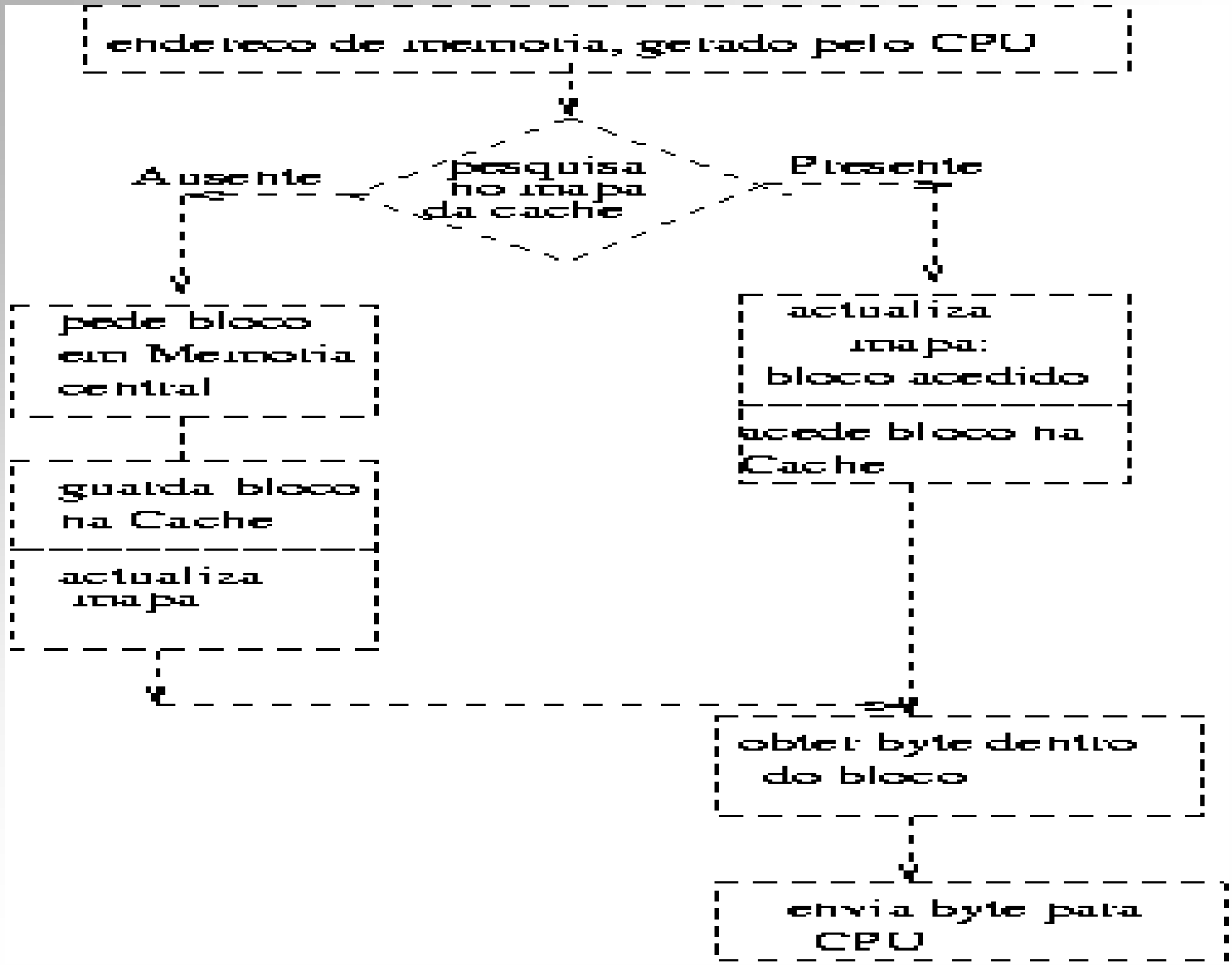
31 30 29 ……4 3 2 1 0

Endereço de bloco:

os 28 bits mais significativos são usados para pesquisar no Mapa

Deslocamento do byte dentro do bloco:

os 4 bits menos significativos são usados só depois de ter localizado o bloco (em Cache ou em Memória)


endereço de memória, gerado pelo CPU
pesquisa no mapa da cache
Ausente
Presente
pede bloco em Memória central
guarda bloco na Cache
actualiza mapa
actualiza mapa: bloco acedido
acede bloco na Cache
obter byte dentro do bloco
envia byte para CPU

Cache associativa

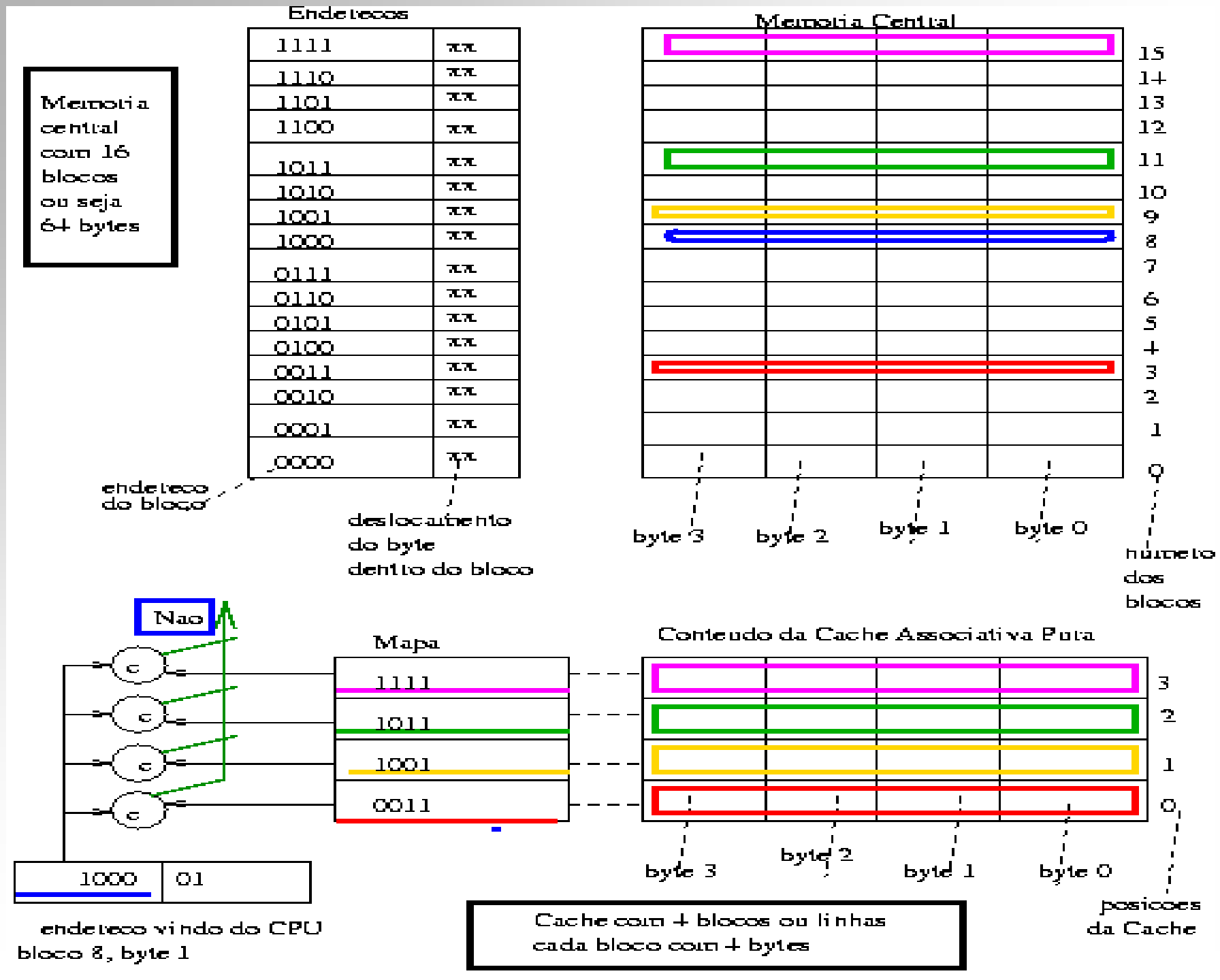
M e C logicamente divididas em Blocos

Seja $n_C = 2^c$ a capacidade da Cache, em blocos

- Define as posições de bloco desde 0 até n_C -1

Seja $n_M = 2^m$ a capacidade da Memória em blocos. Tem-se $n_c <<< n_M$

Dado um endereço de um byte de memória, o correspondente bloco de bytes pode estar localizado em qualquer das n_C posições de blocos da Cache


Memoria central com 16 blocos ou seja 64 bytes
Endereços
1111 xx
1110 xx
1101 xx
1100 xx
1011 xx
1010 xx
1001 xx
1000 xx
0111 xx
0110 xx
0101 xx
0100 xx
0011 xx
0010 xx
0001 xx
0000 xx
endereco do bloco
deslocamento do byte dentro do bloco
Memoria Central
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
byte 3
byte 2
byte 1
byte 0
numero dos blocos
Nao
Mapa
1111
1011
1001
0011
Conteudo da Cache Associativa Pura
3
2
1
0
byte 3
byte 2
byte 1
byte 0
posicoes da Cache
1000 01
endereco vindo do CPU
bloco 8, byte 1
Cache com 4 blocos ou linhas
cada bloco com 4 bytes

As Caches Associativas são muito eficientes:

- qualquer bloco de M pode estar em qualquer posição da C
- a pesquisa do bloco em C faz-se em paralelo, por tantos comparadores quantos os blocos que cabem em C

Mas são muito caras….

- muitos comparadores
- cada comparador é de tantos bits quantos os bits de endereço do bloco
- cada entrada do mapa deve guardar os bits do endereço do bloco carregado nessa posição

Cache de mapa directo

M e C logicamente divididas em Blocos

Seja $n_C = 2^c$ a capacidade da Cache, em blocos

Define as posições de bloco (k) desde 0 até $n_C - 1$

Seja $n_M = 2^m$ a capacidade da Memória em blocos.

Dado um endereço de um byte de memória

$0 <= i <= n_M - 1$

o correspondente bloco de bytes só pode estar localizado numa posição pré-definida k_i de bloco da Cache

$0 <= k_i <= n_C - 1$ tal que $k_i = i \bmod n_C$

Os blocos de memória de endereços

K_i, $K_i + n_C$, $K_i + 2\,n_C$, …

só podem ser carregados na posição K_i da Cache

Exemplo: Cache só com 2 blocos $n_C = 2^1 = 2$

Só há duas posições de bloco na Cache: k = 0 ou k = 1

Na posição 0 da Cache ficam todos os blocos de memória cujo endereço seja par

$k_i = i \bmod 2 = 0$ Blocos i: 0, 2, 4, 6, ….

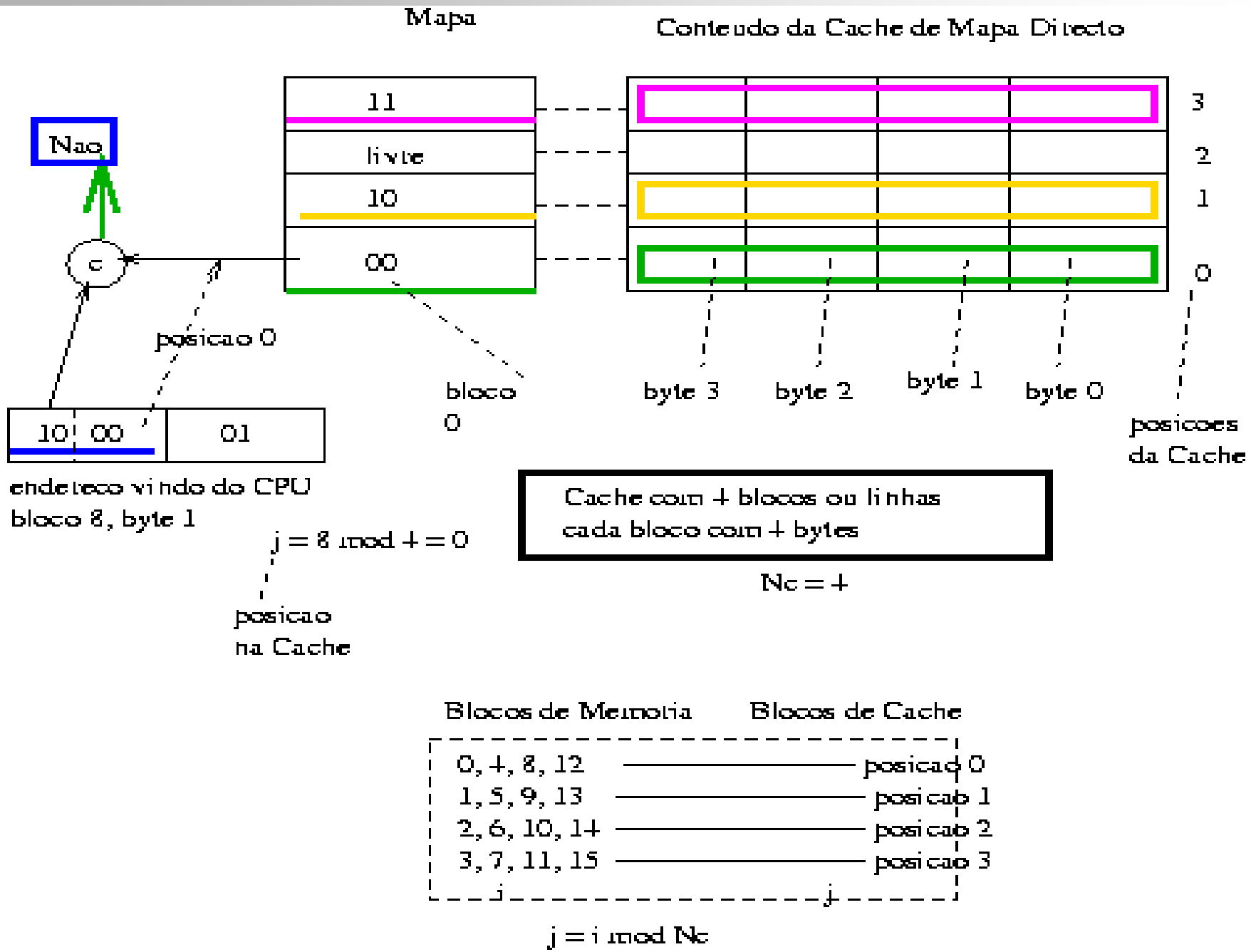
Na posição 1 da Cache ficam todos os blocos de memória cujo endereço seja ímpar

$k_i = i \bmod 2 = 1$ Blocos i: 1, 3, 5, 7, ….

Exemplo: dado este endereço vindo do CPU:

31 30 29 ……4 3 2 1 0

o Controlador da Cache só precisa de testar o **bit 4** para saber em que posição pode estar na Cache! O **bit 4** indica a Classe de equivalência a que o bloco pertence.


Mapa
Conteudo da Cache de Mapa Directo
11
livre
10
00
Nao
c
posicao 0
bloco 0
3
2
1
0
byte 3
byte 2
byte 1
byte 0
posicoes da Cache
10 00 01
endereco vindo do CPU
bloco 8, byte 1
j = 8 mod 4 = 0
posicao na Cache
Cache com 4 blocos ou linhas
cada bloco com 4 bytes
Nc = 4
Blocos de Memoria
Blocos de Cache
0, 4, 8, 12
1, 5, 9, 13
2, 6, 10, 14
3, 7, 11, 15
posicao 0
posicao 1
posicao 2
posicao 3
j = i mod Nc

No mapa do controlador da Cache de mapa directo basta guardar a ´´chave´´ do bloco correntemente ali carregado:

Exemplo: n_C = 256 blocos na Cache → 2^8 posições ou classes de equivalência. Cada bloco 32 bytes = 2^5

Endereço de memória 24 bits: → Espaço de endereços reais: 2^{24} bytes

A memória central suporta $n_M = 2^{24} / 2^5$ blocos = 2^{19} blocos

19 bits (bits 5 a 23) **endereço de bloco em memória**;

5 bits (bits 0 a 4) **deslocamento do byte no bloco**

chave do bloco (bits 13 a 23)

posição do bloco na Cache (bits 5 a 12)

As Caches de mapa directo são mais baratas do que as Associativas:

-- o Mapa da Cache tem menos bits;

-- só precisam de 1 comparador, de tantos bits quantos os da ´´chave´´ dos blocos.

São eficientes na pesquisa em Cache:

-- porque usam os bits da posição do bloco directamente como índice no mapa da Cache.

Originam possíveis colisões no acesso a uma mesma posição da Cache: todos os blocos na mesma classe de equivalência colidem nessa posição.

A probabilidade de colisões diminui à medida que se aumenta a Capacidade da Cache:

a) Cache só com 2 blocos:

todos os blocos pares colidem na posição 0

todos os blocos ímpares colidem na posição 1

b) Cache com 4 blocos:

só colidem na posição 0, os blocos de endereços 0, 4, 8, 12

só colidem na posição 1, os blocos de endereços 1, 5, 9, 13

…

c) Cache com 256 blocos:

só colidem na posição 0, os blocos de endereços 0, 256, 512, …

só colidem na posição 1, os blocos de endereços 1, 257, 513, …

As Caches de Mapa directo são muito usadas na prática.

Cache Associativa por Grupos (*Set Associative*)

Traduzem um compromisso entre as Caches Associativas e as de Mapa Directo.

Seja $n_C = 2^c$ a capacidade da Cache, em blocos.

A Cache está organizada em Grupos de blocos $n_G = 2^g$

Se $n_G = 1$ → Cache Associativa pura → só há 1 grupo.

Se $n_G = n_C$ → Cache de Mapa Directo → cada grupo só tem 1 bloco.

Cada grupo tem uma classe pré-definida (k) desde 0 até $n_G - 1$,

mas cada grupo tem um certo número de posições de blocos: n_C / n_G

Dado um endereço de um byte de memória

$0 <= i <= n_M - 1$

o correspondente bloco de bytes só pode estar localizado nas posições de bloco definidas pela classe k_i de grupo da Cache

$0 <= k_i <= n_C - 1$ tal que $k_i = i \bmod n_C$

Os blocos de memória de endereços

K_i, $K_i + n_C$, $K_i + 2\,n_C$, …

só podem ser carregados nas posições do grupo K_i da Cache

Exemplo: Cache só com 2 Grupos $n_G = 2^1 = 2$

Só há duas classes de grupo na Cache: k = 0 ou k = 1

Nas posições do grupo 0 da Cache ficam todos os blocos de memória cujo endereço seja par

$k_i = i \bmod 2 = 0$ Blocos i: 0, 2, 4, 6, ….

Nas posições do grupo 1 da Cache ficam todos os blocos de memória cujo endereço seja ímpar

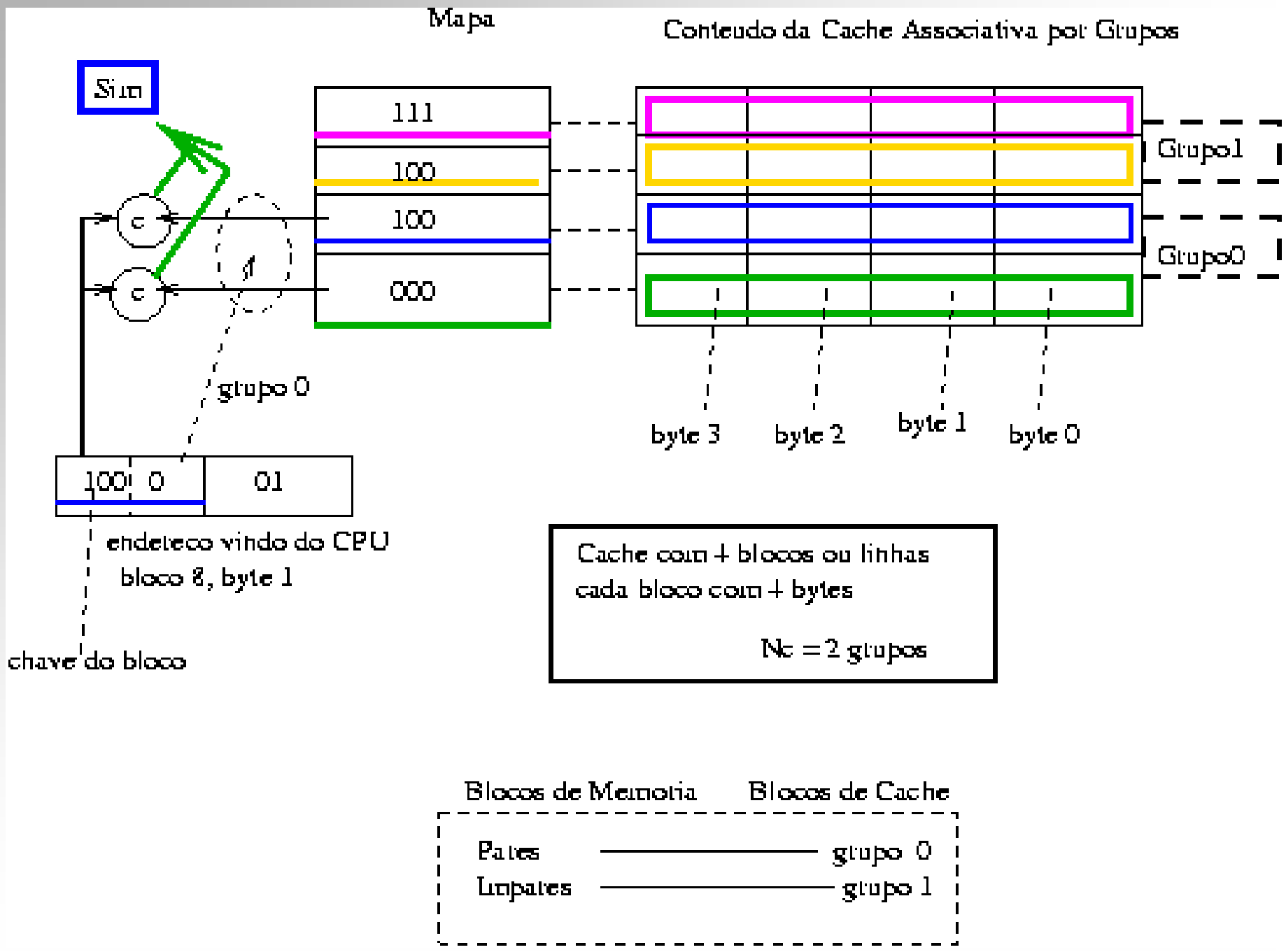
$k_i = i \bmod 2 = 1$ Blocos i: 1, 3, 5, 7, ….

Exemplo: dado este endereço vindo do CPU:

31 30 29 ……4 3 2 1 0

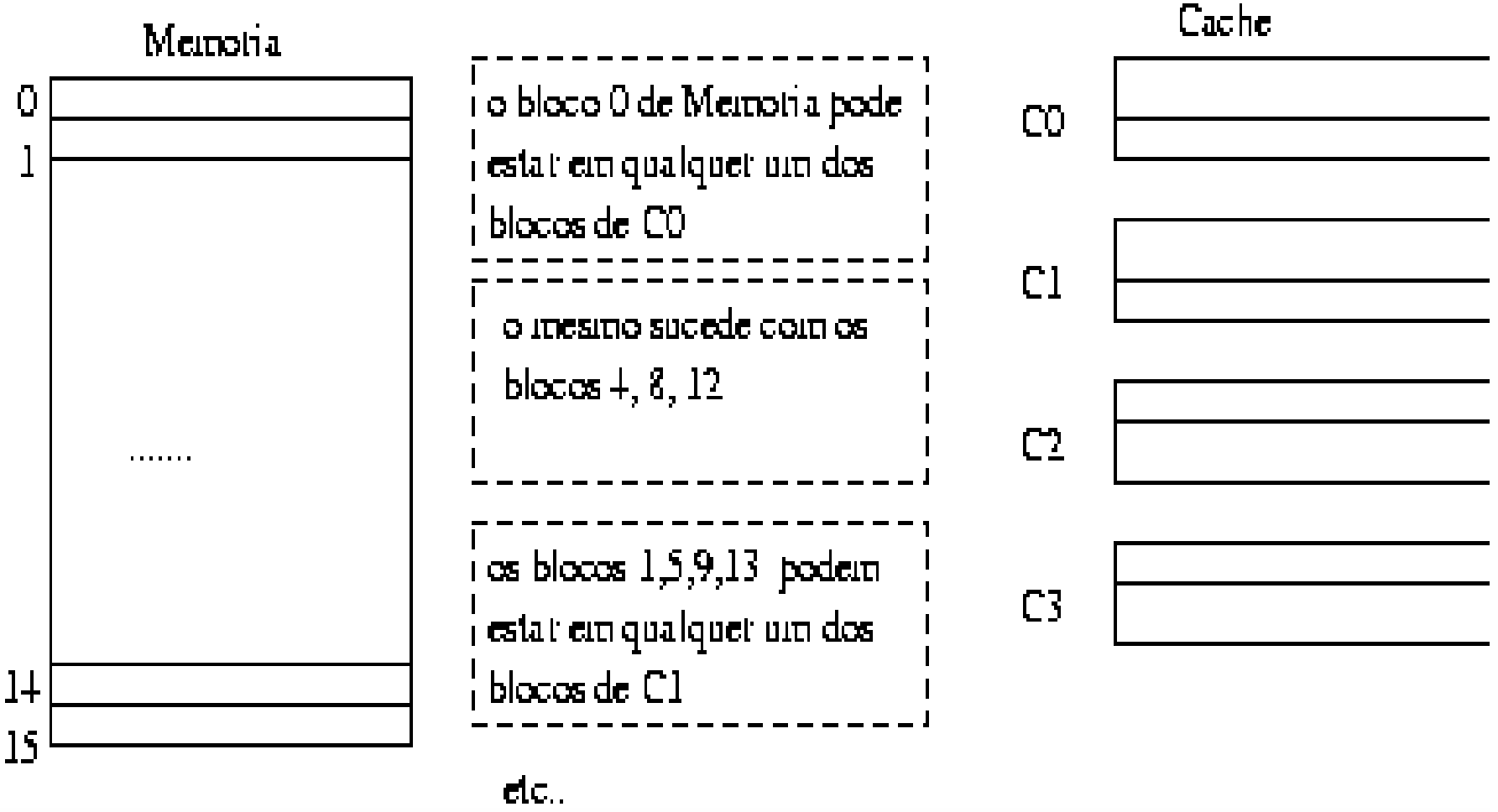
o Controlador da Cache só precisa de testar o **bit 4** para saber em que grupo de posições pode estar na Cache! O **bit 4** indica a Classe de equivalência a que o bloco pertence.

Mas: **se cada grupo tem 2 posições de bloco, podem ter-se dois blocos da mesma classe, simultaneamente carregados em Cache.**


Mapa
Conteudo da Cache Associativa por Grupos
Sim
111
100
100
000
Grupo1
Grupo0
c
c
grupo 0
byte 3
byte 2
byte 1
byte 0
100
0
01
endereco vindo do CPU
bloco 8, byte 1
chave do bloco
Cache com + blocos ou linhas
cada bloco com + bytes
Nc = 2 grupos
Blocos de Memoria
Blocos de Cache
Pares
grupo 0
Impares
grupo 1

$Nm = 16 = 2^{4}$ blocos

$Nc = 4 = 2^{2}$ conjuntos de blocos, cada conjunto com 2 blocos



No mapa do controlador da Cache associativa por grupos basta guardar a ´´chave´´ do bloco correntemente ali carregado:

n_G = 128 grupos na Cache → 2^7 classes de equivalência.

Cada bloco com 32 bytes = 2^5

Endereço de memória 32 bits.