

## Aula 10

### Professores:

Felipe M. G. França  
Valmir C. Barbosa

### Conteúdo:

#### Gerenciamento de memória

- Projeto dos sistemas de paginação
- Segmentação

## Modelo de conjunto funcional

- ➡ Um processo possui, em geral, várias **fases de execução**:
  - Em uma fase da execução, o processo acessa somente um subconjunto do conjunto de páginas.
  - Este subconjunto, em geral, se mantém estável, até que uma nova fase de execução seja iniciada (**referência localizada**).
  - Um exemplo de um processo com várias fases de execução seria um compilador com múltiplos passos.
- ➡ O conjunto de páginas atualmente acessadas por um processo é chamado de o seu **conjunto funcional (working set)**.
- ➡ Se todo o conjunto funcional estiver na memória, então o processo executará rapidamente até que passe para uma nova fase.
- ➡ Se a memória não puder armazenar todo este conjunto, então o processo executará lentamente, pois muitas falhas de página serão geradas (**criação de lixo - trashing**).

## Modelo de conjunto funcional

- Na **paginação sobre demanda**, as páginas são copiadas somente quando o acesso à página gerar uma falha de página:
  - Quando o processo é inicializado, nenhuma página é copiada, e todos os bits ausente/presente da tabela são limpos.
  - Falhas de páginas são geradas até que todas as páginas do conjunto funcional da fase de execução atual sejam copiadas.
- O **modelo do conjunto funcional** pode ser usado se a paginação sobre demanda for ineficiente:
  - O sistema tenta monitorar o conjunto funcional para garantir que este esteja na memória ao executar o processo.
  - Neste caso, temos uma **pré-paginação**, pois as páginas são copiadas antes da execução do processo começar.
  - Para implementar o modelo, o sistema pode usar o algoritmo de idade, para escolher somente as últimas páginas referenciadas.

# Políticas de alocação

- Ao escolher as páginas podemos usar as seguintes políticas:
- Na política de alocação **global**, todas as páginas de todos os processos são consideradas ao executar o algoritmo.
  - Na política de alocação **local**, somente as páginas do processo que gerou a falha são consideradas ao executar o algoritmo.

Idade da página	A0	10
A1	7	
A2	5	
A3	4	
A4	6	
A5	3	
B0	9	
B1	4	
B2	6	
B3	2	
B4	5	
B5	6	
B6	12	
C1	3	
C2	5	
C3	6	

Molduras de  
pagina



Exemplo de uma memória com 16 molduras de página. No momento, temos 6 páginas do processo A (A0, A1, A2, A3, A4, e A5), 7 páginas de B (B0, B1, B2, B3, B4, B5, e B6), e 3 páginas de C (C1, C2, C3). Ao lado de cada página, é dado o valor da idade, que pode ser, por exemplo, aquele contador do algoritmo de idade que vimos anteriormente.

Vamos ver o que acontece em cada política quando o processo A acessa a página A6 que não está na memória.



A/C

# Políticas de alocação

- Ao escolher as páginas podemos usar as seguintes políticas:
- Na política de alocação **global**, todas as páginas de todos os processos são consideradas ao executar o algoritmo.
  - Na política de alocação **local**, somente as páginas do processo que gerou a falha são consideradas ao executar o algoritmo.

Idade da página	A0	10
	A1	7
	A2	5
	A3	4
	A4	6
	A6	25
	B0	9
Molduras de página →	B1	4
	B2	6
	B3	2
	B4	5
	B5	6
	B6	12
	C1	3
	C2	5
	C3	6



B/C

# Políticas de alocação

- Ao escolher as páginas podemos usar as seguintes políticas:
- Na política de alocação **global**, todas as páginas de todos os processos são consideradas ao executar o algoritmo.
  - Na política de alocação **local**, somente as páginas do processo que gerou a falha são consideradas ao executar o algoritmo.

Idade da página	A0	10
	A1	7
	A2	5
	A3	4
	A4	6
	A5	3
	B0	9
Molduras de página →	B1	4
	B2	6
	A6	25
	B4	5
	B5	6
	B6	12
	C1	3
	C2	5
	C3	6



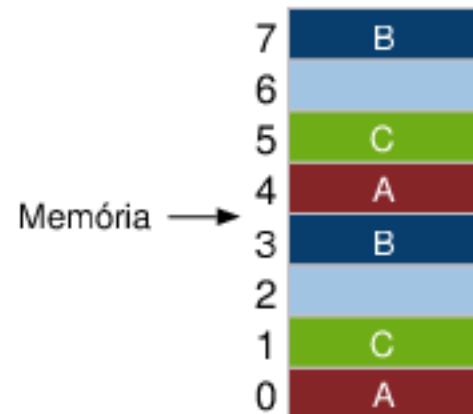
C/C

## Políticas de alocação

- ➡ Se usarmos a política de alocação **global**, o número de molduras de página alocadas a um processo varia dinamicamente.
- ➡ Mas se usarmos a política de alocação **local**, estaremos na verdade alocando uma quantidade fixa de molduras para cada processo.
- ➡ As **políticas globais** são em geral **melhores** do que as locais:
  - Se o tamanho do conjunto funcional do processo variar, então esta política aloca sempre a quantidade ideal de molduras.
  - Com a política local, ou desperdiçaríamos memória, ou não poderíamos copiar todo o conjunto funcional para a memória.
- ➡ Se usarmos a política **global**, o sistema deverá decidir quantas molduras devem ser alocadas a cada um dos processos:
  - Podemos distribuir as molduras entre os processos de acordo com os tamanhos de seus conjuntos funcionais.
  - Podemos usar um algoritmo que aloca dinamicamente as molduras de páginas entre os processos.

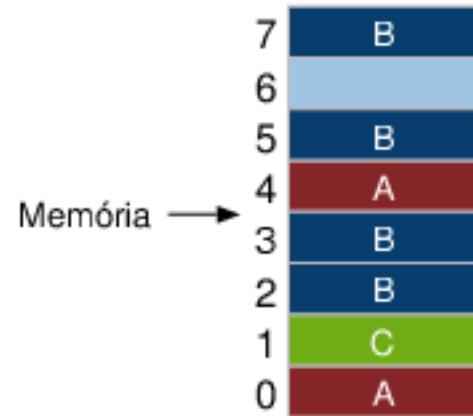
# Políticas de alocação

→ Um possível algoritmo para alocar as molduras é o da **alocação igualitária**, que divide igualmente as molduras entre os processos.



Se tivermos 8 molduras na memória e 3 processos, A, B, e C, então cada processo receberá 2 molduras de página. As 2 molduras restantes serão usadas quando os processos gerarem falhas de página.

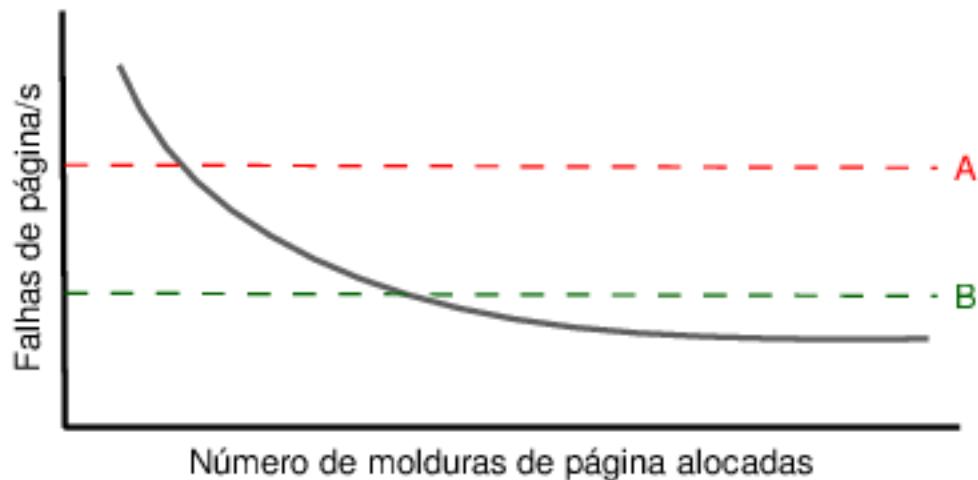
→ Um outro algoritmo é o da **alocação proporcional**, em que o número de molduras alocadas depende do tamanho do processo.



Neste caso, se tivermos as mesmas 8 molduras e os três processos A, B e C, sendo que o tamanho de A é 2K, o de B, 4K, e o de C, 1K, então o A poderia receber 2 molduras, o B, 4 molduras, e o C, receberia somente uma moldura. A moldura restante também pode ser usada quando um dos processos gerar uma falha de página.

# Políticas de alocação

- O algoritmo por **freqüência de falha de página** divide as molduras de tal modo que a geração de lixo seja evitada:
- Baseado no fato que a taxa de falha de página é reduzida quando alocamos mais molduras a um processo.
  - Mais molduras são alocadas a um processo se a taxa de falhas de página for muito alta.
  - As molduras podem ser retiradas de um processo se a taxa de falhas de página for muito baixa.



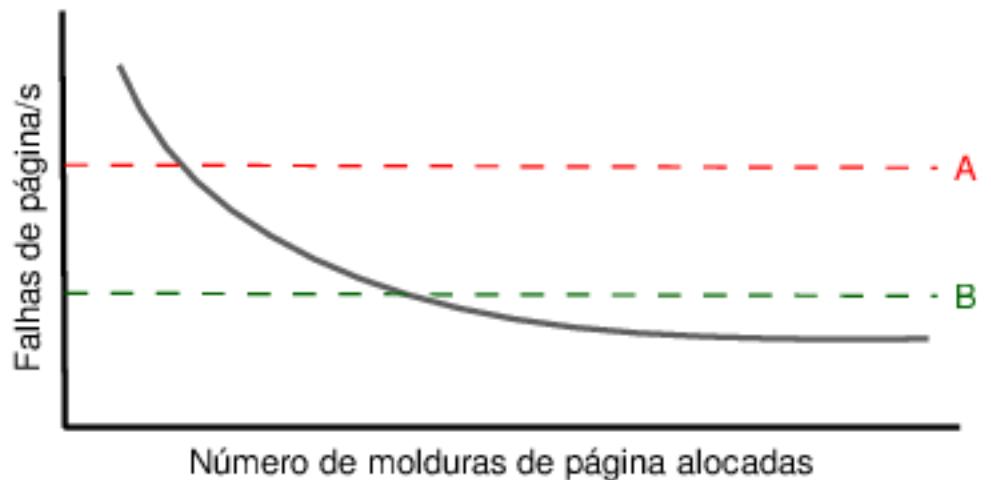
Neste algoritmo, as páginas são alocadas ao processo, ou retiradas deste processo, de acordo com o número de falhas de página por segundo geradas por este processo.



A/C

# Políticas de alocação

- O algoritmo por **freqüência de falha de página** divide as molduras de tal modo que a geração de lixo seja evitada:
- Baseado no fato que a taxa de falha de página é reduzida quando alocamos mais molduras a um processo.
  - Mais molduras são alocadas a um processo se a taxa de falhas de página for muito alta.
  - As molduras podem ser retiradas de um processo se a taxa de falhas de página for muito baixa.



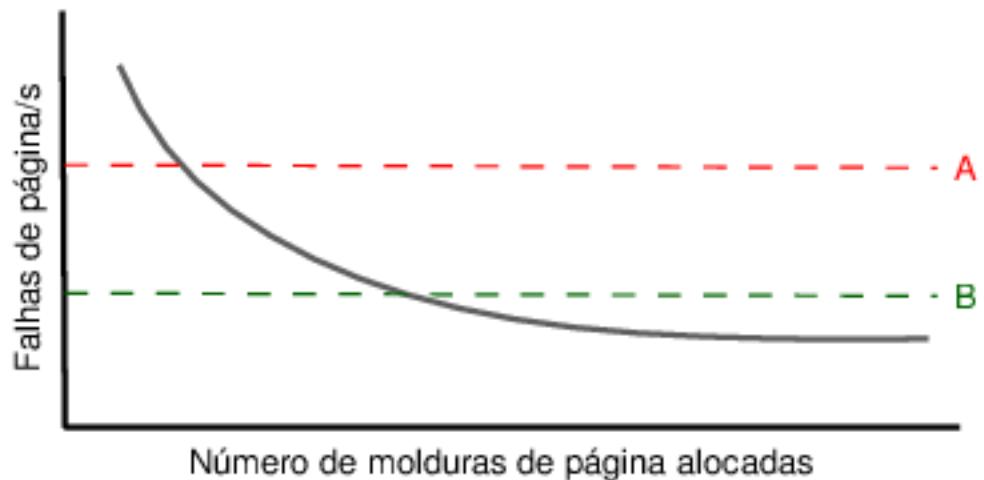
Se a taxa de falhas de um processo estiver acima da linha A, então mais molduras serão alocadas para este processo. Se não temos mais molduras, então algum processo será escolhido, removido da memória, e algumas das suas molduras serão alocadas a este processo, para reduzir a sua taxa de falhas, de tal modo que esta fique abaixo da linha A.



B/C

# Políticas de alocação

- O algoritmo por **freqüência de falha de página** divide as molduras de tal modo que a geração de lixo seja evitada:
- Baseado no fato que a taxa de falha de página é reduzida quando alocamos mais molduras a um processo.
  - Mais molduras são alocadas a um processo se a taxa de falhas de página for muito alta.
  - As molduras podem ser retiradas de um processo se a taxa de falhas de página for muito baixa.



Agora, se a taxa de falhas de um processo estiver abaixo da linha B, então a taxa de falhas é considerada pequena, e, se for necessário, molduras de páginas podem ser retiradas do processo para, por exemplo, reduzir as taxas de falhas que estejam muito altas.



C/C

## Tamanho da página

- ➡ A escolha do tamanho ideal da página (e da moldura) pelo sistema operacional é importante. Se este tamanho for igual a  $p$ :
  - Em média, a metade da última página alocada a uma área de um processo não é usada (**fragmentação interna**):
    - Se existem, por exemplo,  $n$  áreas de memória no sistema, então o desperdício de espaço será de  $np/2$ .
    - Neste caso, o valor de  $p$  deve ser o **menor possível**.
  - O tamanho da tabela de páginas depende de  $p$ :
    - Se o processo tem  $s$  bytes, o espaço gasto pela sua tabela será de  $se/p$ , onde  $e$  é o tamanho da entrada da tabela.
    - Neste caso, o valor de  $p$  deve ser o **maior possível**.
- ➡ Como podemos ver, o tamanho ideal para uma página deve levar em conta estes dois objetivos contraditórios.

## Tamanho da página

Logo, o overhead de usar uma página com  $p$  bytes, com uma entrada com  $e$  bytes, e um processo com  $s$  bytes será:

$$\text{overhead} = \frac{se}{p} + \frac{p}{2}$$

- Para obter o número ideal de páginas, basta derivar a função acima, em relação a  $p$ , e igualar a equação resultante a 0:

$$\frac{-se}{p^2} + \frac{1}{2} = 0 \implies p = \sqrt{2se}$$

- Por exemplo, se  $s$  é igual a 128K e  $e$  é igual a 8 bytes, então o tamanho  $p$  ideal da página é 1448 bytes.
- Como, em geral, o tamanho da página deve ser uma potência de 2, uma página de 1K ou de 2K é usada.

# Interface de memória virtual

- ➡ Usada para permitir que o programador tenha controle sobre o mapa de memória do processo que executa o seu programa:
  - Pode-se facilitar a cooperação entre os processos definindo uma **área de memória comum** mapeada em todos estes processos.
  - A troca de dados entre estes processos será muito mais rápida do que se usarmos outras **técnicas de compartilhamento**.
- ➡ Uma técnica de gerenciamento de memória avançada é a **memória compartilhada distribuída**:
  - Permite que processos em diversas máquinas de uma rede compartilhem um espaço de endereçamento virtual comum.
  - As páginas podem estar localizadas em qualquer computador da rede, e quando uma falha de página é gerada:
    - O computador com a página é localizado, e uma mensagem é enviada a este, pedindo para desmapear a página.
    - Quando a página for enviada, esta é mapeada, e a instrução que gerou a falha é reiniciada.

# Segmentação

- ➡ O gerenciamento através da memória virtual é **unidimensional**, pois cada processo possui somente um espaço de endereçamento.
- ➡ Algumas aplicações seriam mais facilmente programadas se fosse possível termos vários espaços de endereçamento virtual:
  - Um exemplo seria um compilador com múltiplas tabelas, cujos tamanhos podem variar no tempo:

Espaço de endereço virtual



Exemplo de um compilador que usa 5 tabelas ao compilar um programa, distribuídas pelo espaço de endereçamento:

- A tabela com os símbolos do programa.
- A tabela com o código fonte do programa anotado.
- A tabela com as constantes definidas no programa.
- A tabela com a árvore de análise obtida ao avaliar a sintaxe do código do programa.
- A pilha de chamadas, que contém todo o histórico de chamadas de procedimento ainda não terminadas.

A/C



Memória livre

# Segmentação

- O gerenciamento através da memória virtual é **unidimensional**, pois cada processo possui somente um espaço de endereçamento.
- Algumas aplicações seriam mais facilmente programadas se fosse possível termos vários espaços de endereçamento virtual:
  - Um exemplo seria um compilador com múltiplas tabelas, cujos tamanhos podem variar no tempo:

Espaço de endereço virtual



Cada tabela pode crescer até atingir a próxima tabela, ou o final do espaço de endereçamento virtual. Suponha que a tabela de símbolos cresceu até atingir a tabela com o código fonte. Agora não existe mais espaço para a tabela crescer. Se o programa gerar novos símbolos, o compilador poderia simplesmente abortar a compilação, gerando um erro informando que o programa possui muitos símbolos.

Memória livre

# Segmentação

- ➡ O gerenciamento através da memória virtual é **unidimensional**, pois cada processo possui somente um espaço de endereçamento.
- ➡ Algumas aplicações seriam mais facilmente programadas se fosse possível termos vários espaços de endereçamento virtual:
  - Um exemplo seria um compilador com múltiplas tabelas, cujos tamanhos podem variar no tempo:

Espaço de endereço virtual



O compilador poderia alternativamente usar o espaço não usado de alguma outra tabela, como por exemplo, a tabela de constantes, mas isso faria com que o gerenciamento das tabelas do compilador fosse tão complicado quanto o gerenciamento de overlays, o que, como vimos, não é algo com que os programadores (do compilador) desejam lidar. Do ponto de vista do gerenciamento de memória, existiria uma **fragmentação externa** dentro da área de dados da aplicação.

C/C



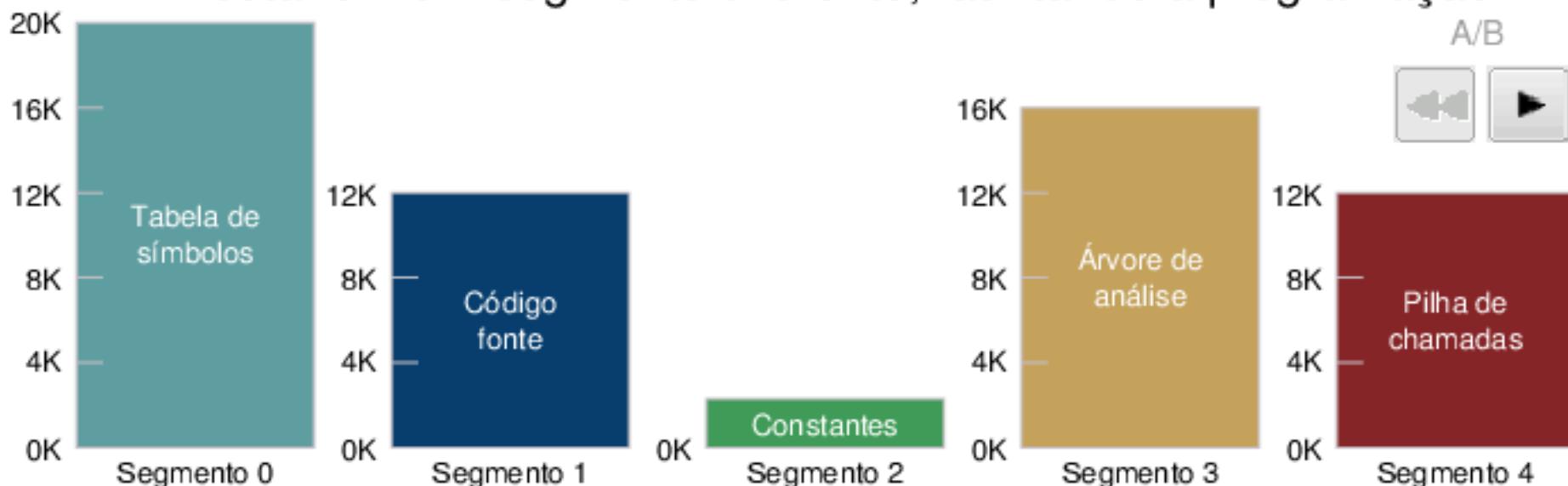
Memória livre

# Segmentação

- ➡ A solução é usar o gerenciamento de memória baseado em **segmentação**:
  - O sistema fornece vários espaços de endereçamento diferentes, chamados de **segmentos**:
    - Cada segmento tem uma **seqüência linear de endereços**, que começa sempre no endereço 0.
    - Cada segmento tem um **comprimento**, que pode variar, durante a execução, entre 0 até um valor máximo permitido.
  - Os tamanhos dos segmentos podem variar independentemente, pois os segmentos são **espaços de endereçamento diferentes**.
  - O espaço de um segmento também pode acabar, mas isso é raro, pois em geral o tamanho máximo é bem grande.
- ➡ Quando o sistema usa uma memória segmentada, o programador sabe da sua existência, isto é, um segmento é uma **unidade lógica**.

# Segmentação

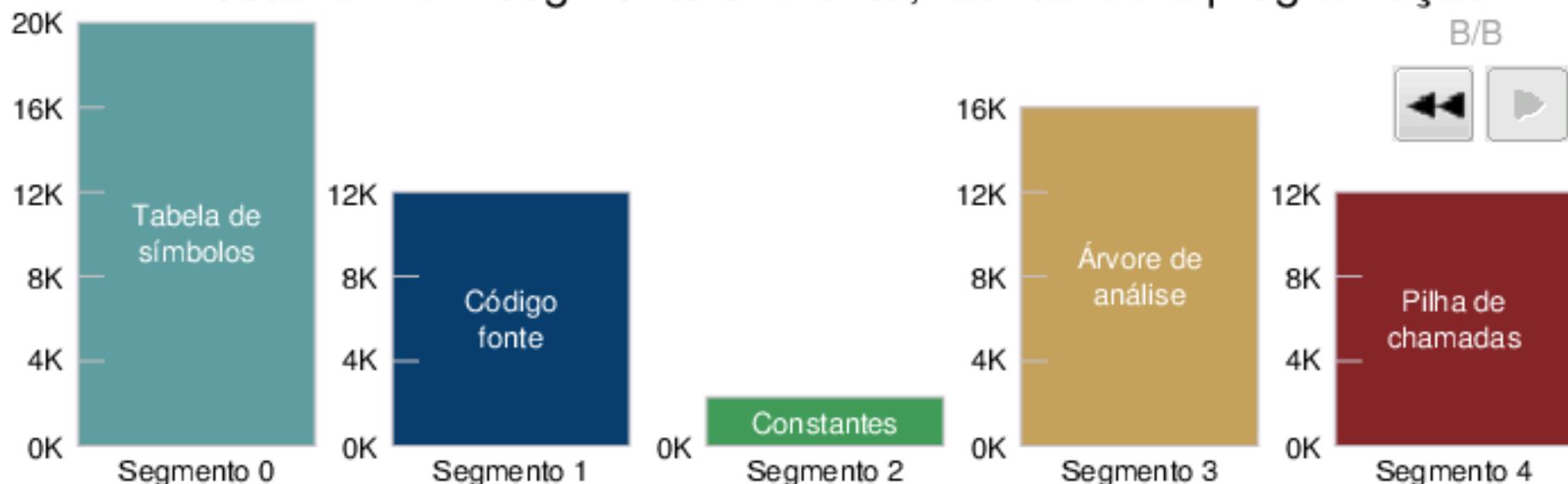
- ➡ Neste caso, a memória é segmentada, ou seja, **bidimensional**, e com isso, os endereços serão compostos de duas partes [(s,d)]:
  - O **número do segmento** (s) que contém este endereço.
  - O deslocamento (d) do endereço dentro deste segmento.
- ➡ No exemplo anterior do compilador, cada uma das tabelas pode estar em um segmento diferente, facilitando a programação:



Exemplo do compilador anterior, mas agora cada tabela está em um segmento diferente, e com isso, as tabelas podem crescer sem colidir umas com as outras. Além disso, nenhum espaço é desperdiçado em uma tabela.

# Segmentação

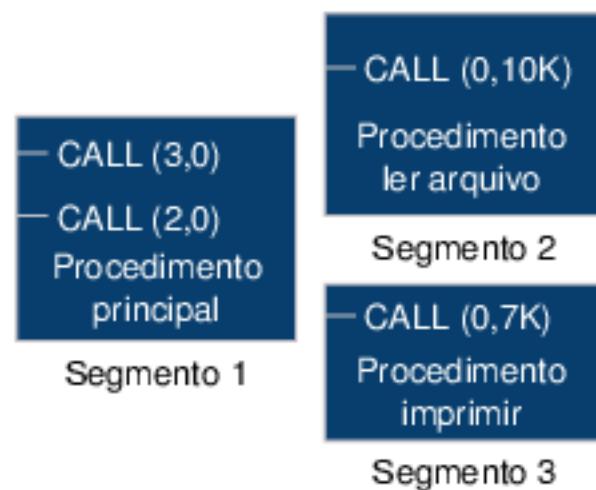
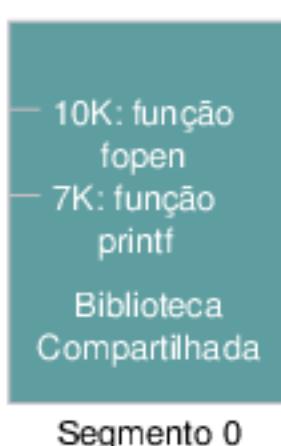
- ➡ Neste caso, a memória é segmentada, ou seja, **bidimensional**, e com isso, os endereços serão compostos de duas partes  $((s,d))$ :
  - O **número do segmento** ( $s$ ) que contém este endereço.
  - O deslocamento ( $d$ ) do endereço dentro deste segmento.
- ➡ No exemplo anterior do compilador, cada uma das tabelas pode estar em um segmento diferente, facilitando a programação:



Para o compilador acessar, por exemplo, a palavra da posição 8K da tabela com a árvore de análise, este deveria usar o endereço  $(3, 8K)$ , que acessa a palavra que está a 8K do início do segmento 3.

# Segmentação

- A memória segmentada facilita a realocação dos processos:
- Colocamos cada procedimento num segmento separado.
  - Para chamar um procedimento do segmento  $n$ , deveremos executar a chamada ao endereço  $(n, 0)$ .
  - A linkedição do programa é facilitada, pois cada procedimento começa no endereço 0 de algum segmento.
  - A modificação de um procedimento é facilitada, pois a sua recompilação não afeta o endereço dos outros procedimentos.
  - O compartilhamento de dados e procedimentos é facilitado.



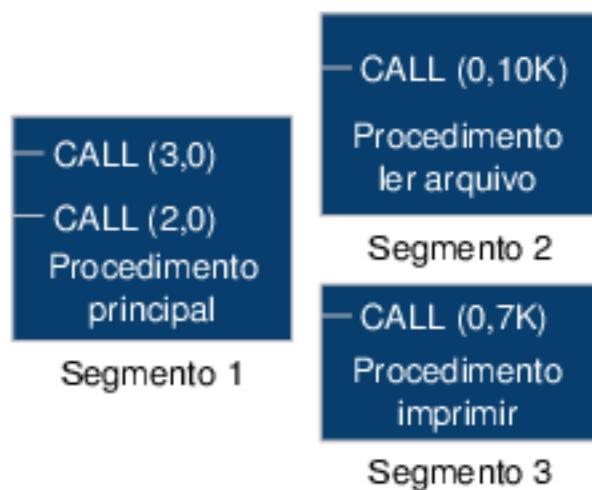
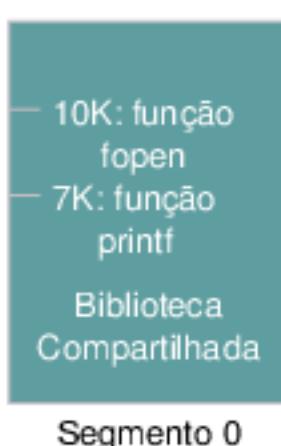
Exemplo de um programa com três segmentos, e de um segmento especial com uma biblioteca compartilhada. O procedimento *principal*, no segmento 1, usa o procedimento do segmento 2 - CALL (2,0) - para ler um arquivo, e o do 3 - CALL (3, 0) - para imprimir este arquivo. Estes procedimentos podem ser alterados sem que seja necessário recompilar todos os outros procedimentos.



A/B

# Segmentação

- A memória segmentada facilita a realocação dos processos:
- Colocamos cada procedimento num segmento separado.
  - Para chamar um procedimento do segmento  $n$ , deveremos executar a chamada ao endereço  $(n, 0)$ .
  - A linkedição do programa é facilitada, pois cada procedimento começa no endereço 0 de algum segmento.
  - A modificação de um procedimento é facilitada, pois a sua recompilação não afeta o endereço dos outros procedimentos.
  - O compartilhamento de dados e procedimentos é facilitado.



O procedimento *ler arquivo* chama a função *fopen* no endereço 10K da biblioteca - CALL (0, 10K) - para abrir o arquivo, e o procedimento *imprimir* chama a função *fprintf* no endereço 7K - CALL (0, 7K) - para imprimir uma linha do arquivo na tela. Neste caso, se a biblioteca for alterada, os procedimentos deverão ser recompilados, pois dependem dos endereços das funções *fopen* e *fprintf*.



B/B

# Segmentação

- ➡ A memória segmentada também facilita a proteção:
  - Como o programador sabe da existência dos segmentos, pode definir o tipo de proteção do segmento:
    - Um segmento de código deve somente ser executado.
  - Na memória paginada este tipo de proteção não é interessante:
    - Para implementar a proteção, o programador deveria saber os limites das páginas, algo não desejável.
- ➡ A tabela abaixo compara a memória paginada com a segmentada:

Consideração	Paginação	Segmentação
É visível ao programador?	Não	Sim
Número de espaços de endereçamento lineares	1	Muitos
O espaço pode ser maior do que a memória?	Sim	Sim
Fornece proteção a dados e procedimentos?	Não	Sim
Facilita o uso de estruturas com tamanhos variáveis?	Não	Sim
Facilita o compartilhamento de procedimentos?	Não	Sim
Por que a técnica foi inventada?	Usar mais espaço	Facilitar a programação

# Implementação da segmentação

- A implementação da segmentação é diferente da paginação:
- Ao contrário das páginas, que possuem um tamanho fixo, o tamanho dos segmentos é variável.
  - O gerenciamento neste caso é similar ao do gerenciamento da memória por troca que vimos anteriormente:
    - Os segmentos devem ser movidos ao aumentar de tamanho, e podem ser salvos e depois lidos do disco.
    - **A fragmentação externa** também pode ocorrer.



Exemplo de uma memória segmentada. A memória real do computador possui, neste exemplo, 32K. Esta memória está totalmente ocupada por 5 segmentos, numerados de 0 até 4, sendo que os seus tamanhos, são, respectivamente, 4K, 8K, 5K, 8K, e 7K.



Memória livre



A/E

# Implementação da segmentação



A implementação da segmentação é diferente da paginação:

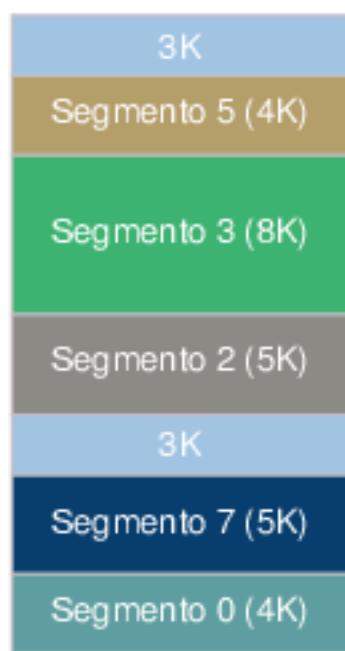
- Ao contrário das páginas, que possuem um tamanho fixo, o tamanho dos segmentos é variável.
- O gerenciamento neste caso é similar ao do gerenciamento da memória por troca que vimos anteriormente:
  - Os segmentos devem ser movidos ao aumentar de tamanho, e podem ser salvos e depois lidos do disco.
  - A **fragmentação externa** também pode ocorrer.



O segmento 1 foi salvo no disco, pois o segmento 7, que não estava na memória, foi acessado. Este segmento é copiado no mesmo endereço do 1, e como o seu tamanho, de 5K, é menor do que o do 1, de 8K, então 3K de memória estão agora disponíveis entre os segmentos 7 e 2.

# Implementação da segmentação

- A implementação da segmentação é diferente da paginação:
- Ao contrário das páginas, que possuem um tamanho fixo, o tamanho dos segmentos é variável.
  - O gerenciamento neste caso é similar ao do gerenciamento da memória por troca que vimos anteriormente:
    - Os segmentos devem ser movidos ao aumentar de tamanho, e podem ser salvos e depois lidos do disco.
    - **A fragmentação externa** também pode ocorrer.



Agora o segmento 4 foi salvo, e no lugar dele foi copiado o segmento 5, que também não estava na memória. Assim como antes, este segmento é copiado no mesmo endereço do 4, e uma área 3K é liberada no final da memória, pois o tamanho do segmento 5 é de 4K, e o do 4 é de 7K.



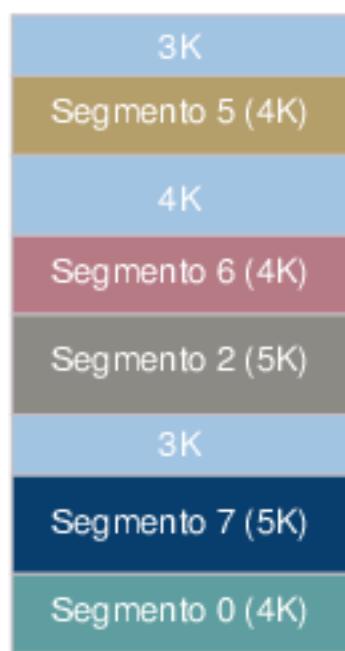
Memória livre



C/E

# Implementação da segmentação

- A implementação da segmentação é diferente da paginação:
- Ao contrário das páginas, que possuem um tamanho fixo, o tamanho dos segmentos é variável.
  - O gerenciamento neste caso é similar ao do gerenciamento da memória por troca que vimos anteriormente:
    - Os segmentos devem ser movidos ao aumentar de tamanho, e podem ser salvos e depois lidos do disco.
    - **A fragmentação externa** também pode ocorrer.



Depois de algum tempo, o segmento 3 foi trocado pelo segmento 6, com 4K, pois este foi acessado. Como o seu tamanho é 4K menor do que o do 3 (cujo tamanho é de 8K), então 4K de memória foram liberados entre os segmentos 6 e 5.

Note que, assim como no caso do gerenciamento por troca, agora a memória está fragmentada, sendo composta por pequenos fragmentos dispersos entre os segmentos.



Memória livre



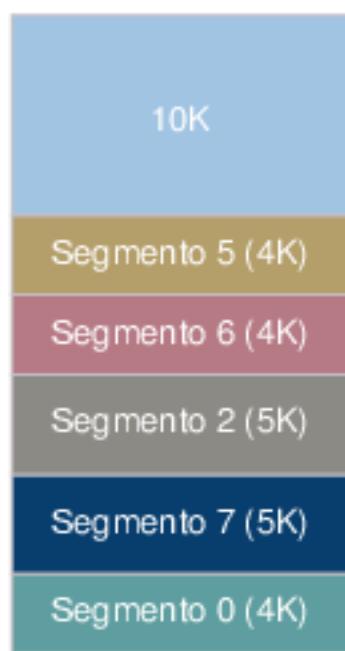
D/E

# Implementação da segmentação



A implementação da segmentação é diferente da paginação:

- Ao contrário das páginas, que possuem um tamanho fixo, o tamanho dos segmentos é variável.
- O gerenciamento neste caso é similar ao do gerenciamento da memória por troca que vimos anteriormente:
  - Os segmentos devem ser movidos ao aumentar de tamanho, e podem ser salvos e depois lidos do disco.
  - A **fragmentação externa** também pode ocorrer.



Este problema, que como vimos é chamado de **fragmentação externa**, pode, assim como antes, ser resolvido pela compactação da memória, mas, como já sabemos, este processo não é viável, na maior parte dos casos, por ser muito lento.



Memória livre



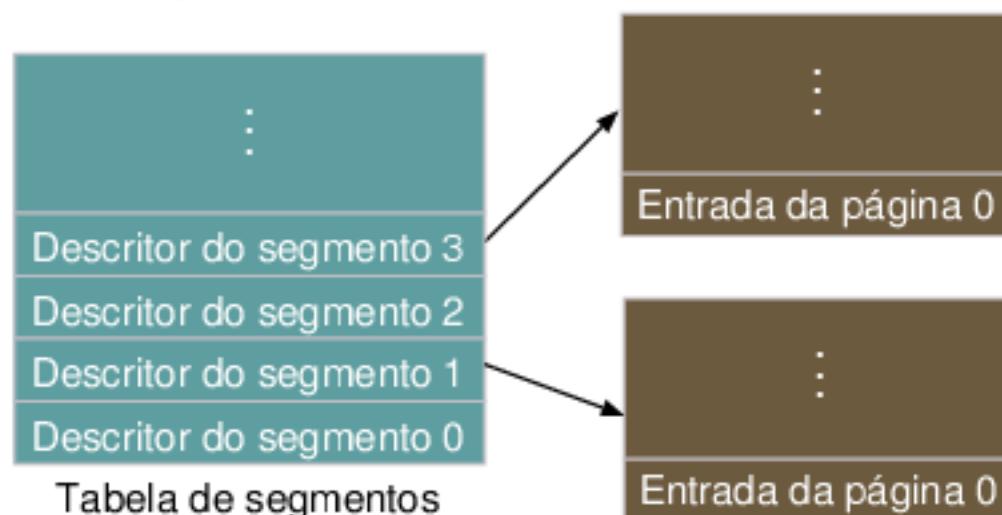
E/E

# Segmentação com paginação

- ➡ Usada para permitir que o tamanho do segmento seja maior do que o da memória:
  - Cada segmento é dividido em um conjunto de páginas, assim como ocorre com a paginação da memória.
  - Otimiza o uso da memória, pois somente as partes atualmente acessadas dos segmentos precisam estar na memória.
- ➡ Como ainda usamos a segmentação, podemos usar as vantagens de programação e de proteção que vimos anteriormente.
- ➡ Como um segmento é similar ao espaço de endereçamento virtual, podemos usar os mesmos algoritmos de substituição de página.
- ➡ O sistema MULTICS usava a segmentação com paginação:
  - Fornecia uma memória virtual composta por 250.000 segmentos com tamanho máximo de 64Kb (com palavras de 36 bits).
  - Cada segmento era tratado como uma memória virtual diferente, e era paginado.

# Segmentação com paginação

→ No MULTICS, cada programa tem uma tabela de segmentos:



Exemplo de uma tabela de segmentos. As tabelas de páginas dos segmentos 1 e 3, indicadas por uma seta, estão na memória, pois estes segmentos estão na memória. Os outros segmentos não estão atualmente na memória. Como esta tabela pode ser grande, pois podemos ter até 250.000 segmentos, esta tabela será armazenada em um segmento que também é paginado, e que é chamado de **segmento descriptor**.

— Cada descriptor possui as informações de um dos segmentos:



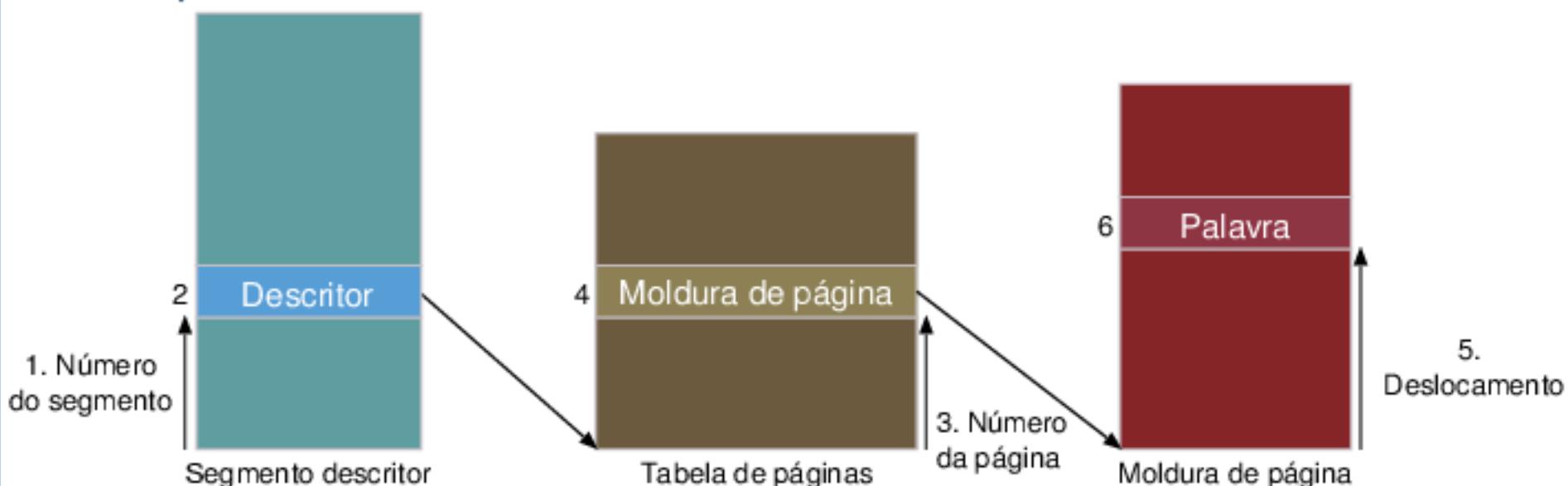
Descriptor de um segmento do MULTICS. O tamanho do descriptor é de 36 bits, pois as palavras do computador usado pelo sistema tinham 36 bits. Pressione o mouse sobre cada um dos campos para ver uma descrição resumida do seu significado.

# Segmentação com paginação

→ O endereço virtual do MULTICS é dividido do seguinte modo:



→ O MULTICS converte o endereço virtual no físico do seguinte modo:



Vamos ver em mais detalhes o que ocorre quando um endereço virtual cujo formato foi dado acima é acessado por alguma instrução.

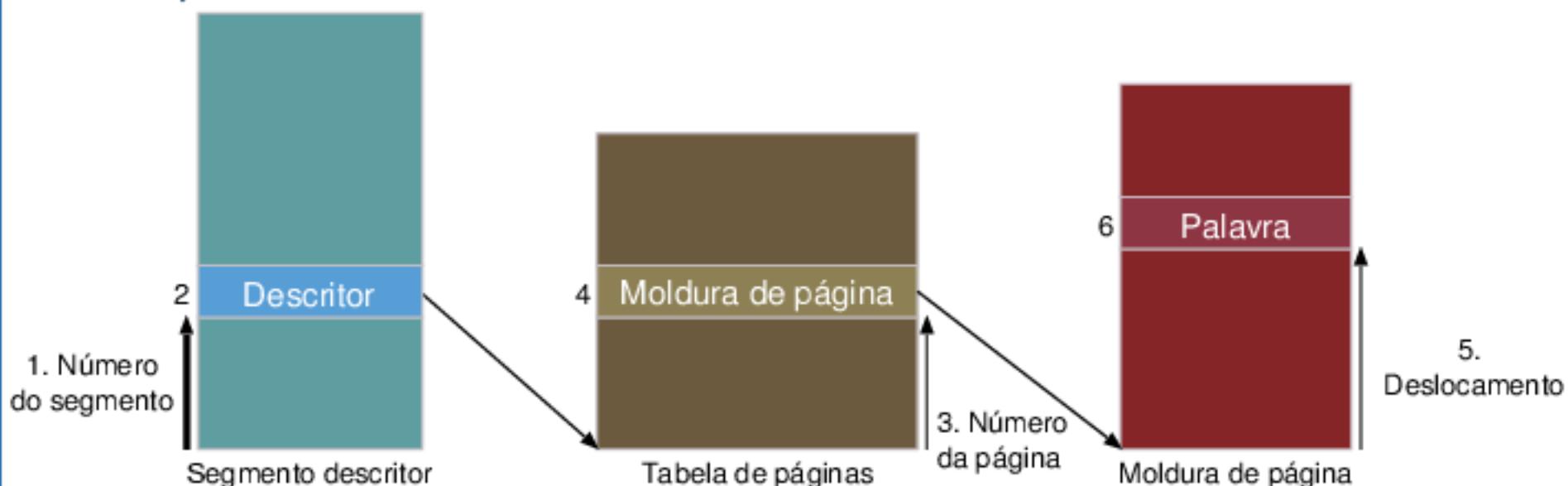


# Segmentação com paginação

→ O endereço virtual do MULTICS é dividido do seguinte modo:



→ O MULTICS converte o endereço virtual no físico do seguinte modo:



1. O número do segmento, definido pelo endereço virtual, é usado como um índice no segmento descriptor do programa, para obter o descriptor do segmento dado por este número.

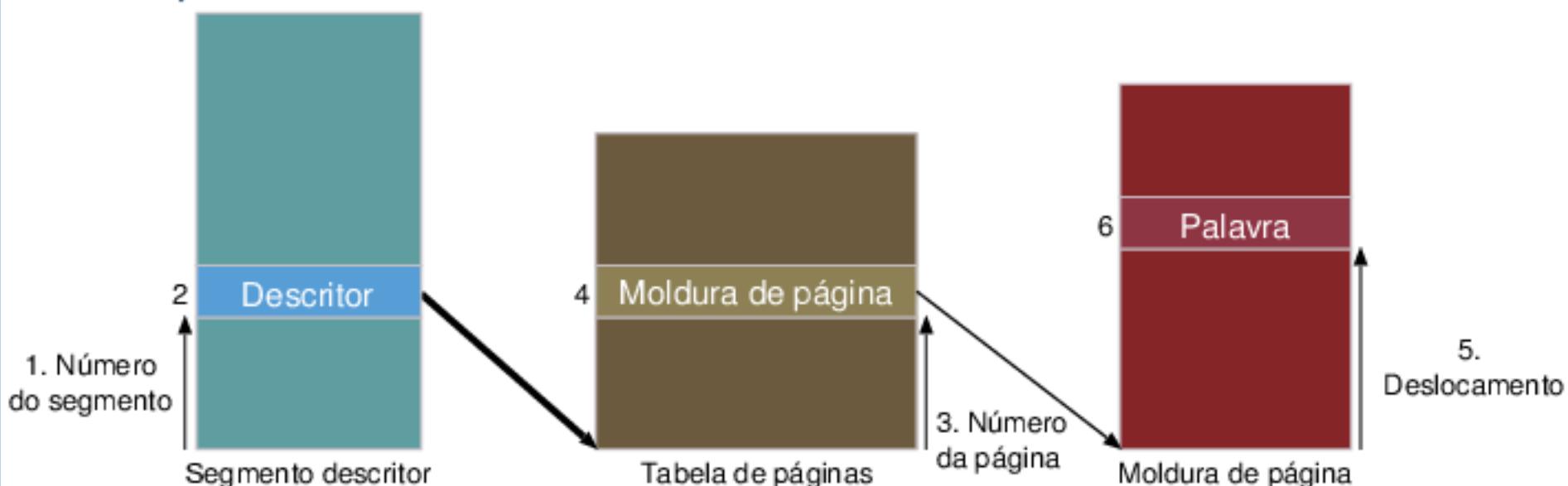


# Segmentação com paginação

→ O endereço virtual do MULTICS é dividido do seguinte modo:



→ O MULTICS converte o endereço virtual no físico do seguinte modo:



2. Depois que o descritor do segmento é localizado, verificamos se este está na memória. Caso este esteja na memória, e o acesso ao segmento não viole a sua proteção, a tabela de páginas do segmento é localizada. Caso contrário, uma falha de violação de proteção é gerada se o acesso for inválido, ou uma falha de segmento é gerada se o segmento não estiver na memória.

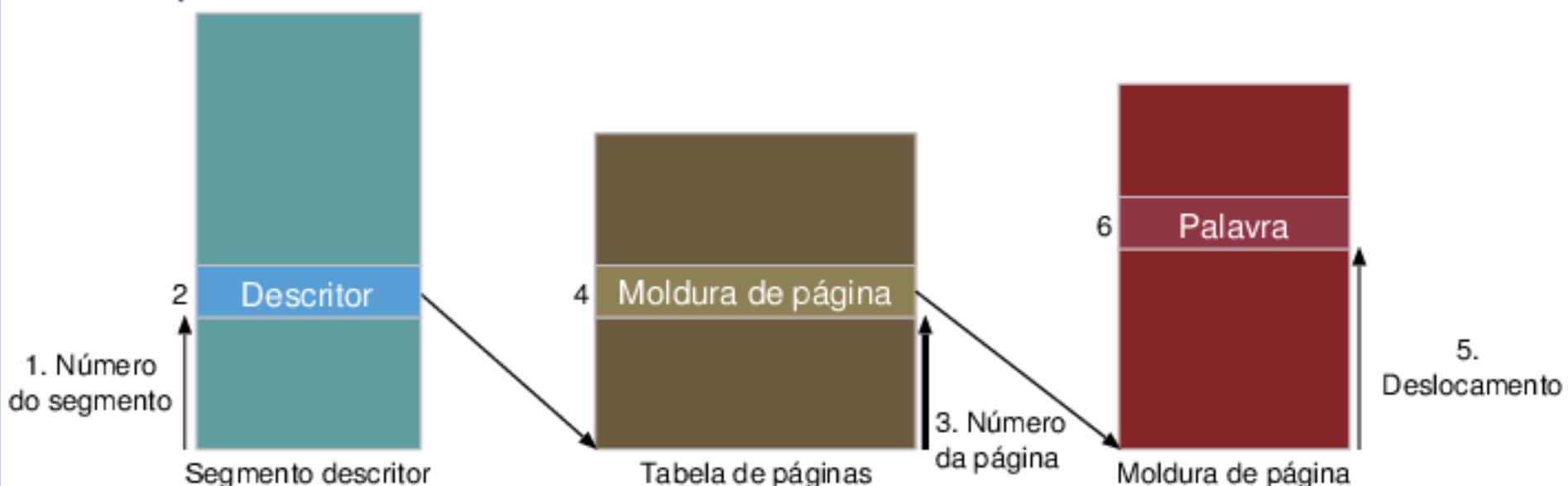


# Segmentação com paginação

→ O endereço virtual do MULTICS é dividido do seguinte modo:



→ O MULTICS converte o endereço virtual no físico do seguinte modo:



3. Agora o número da página dado no endereço virtual é usado como um índice na tabela de páginas para localizar a entrada desta página.

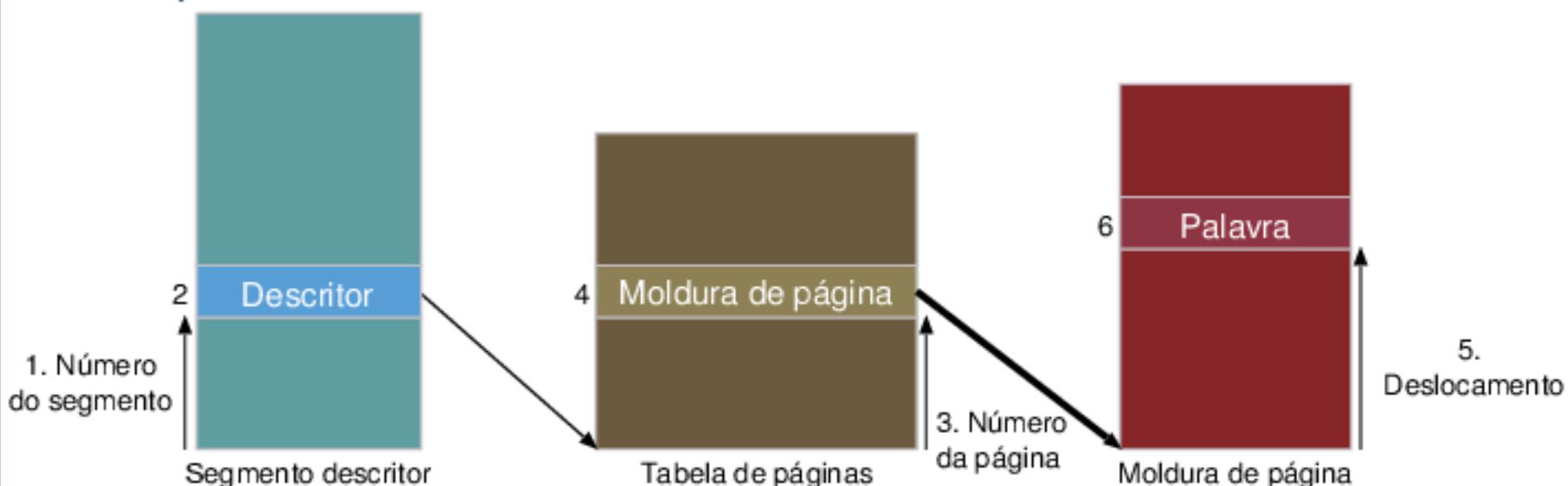


# Segmentação com paginação

→ O endereço virtual do MULTICS é dividido do seguinte modo:



→ O MULTICS converte o endereço virtual no físico do seguinte modo:



4. A entrada na tabela para a página dada no endereço virtual é verificada, e caso a página esteja na memória, a moldura de página em que a página foi copiada é determinada, o que nos permite determinar o endereço inicial desta página na memória principal. Caso contrário, uma falha de página é gerada.

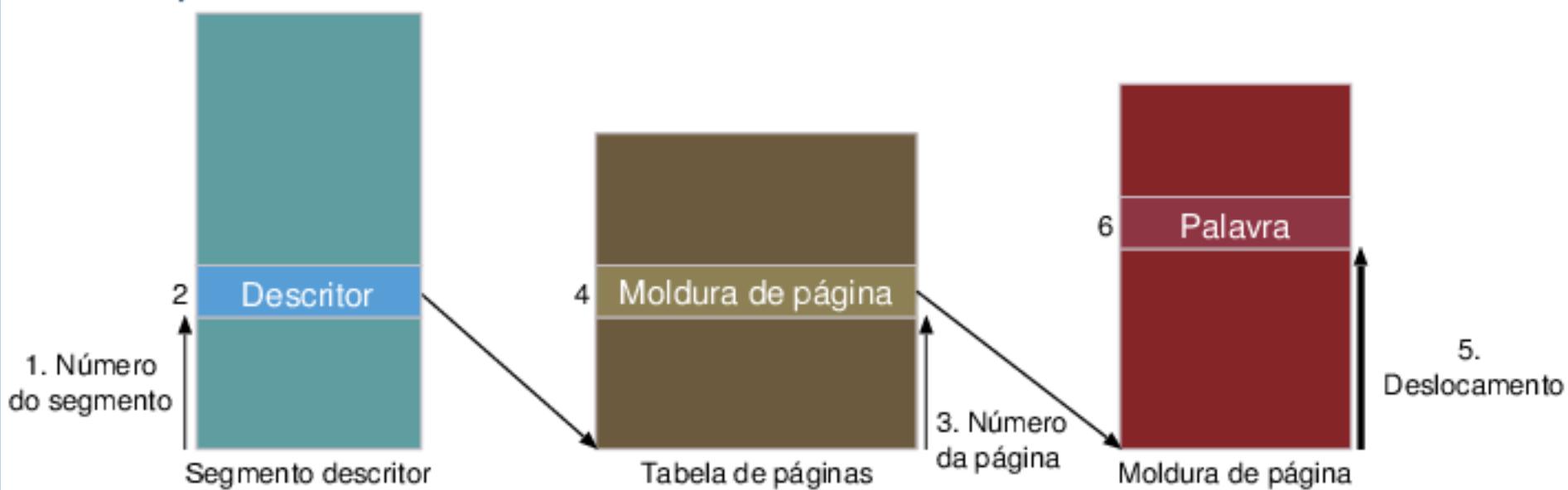


# Segmentação com paginação

→ O endereço virtual do MULTICS é dividido do seguinte modo:



→ O MULTICS converte o endereço virtual no físico do seguinte modo:



5. Depois de determinado o início da página na memória, o deslocamento do endereço virtual é então adicionado ao endereço inicial da página, o que nos dará o endereço da palavra da memória que a instrução desejava acessar.

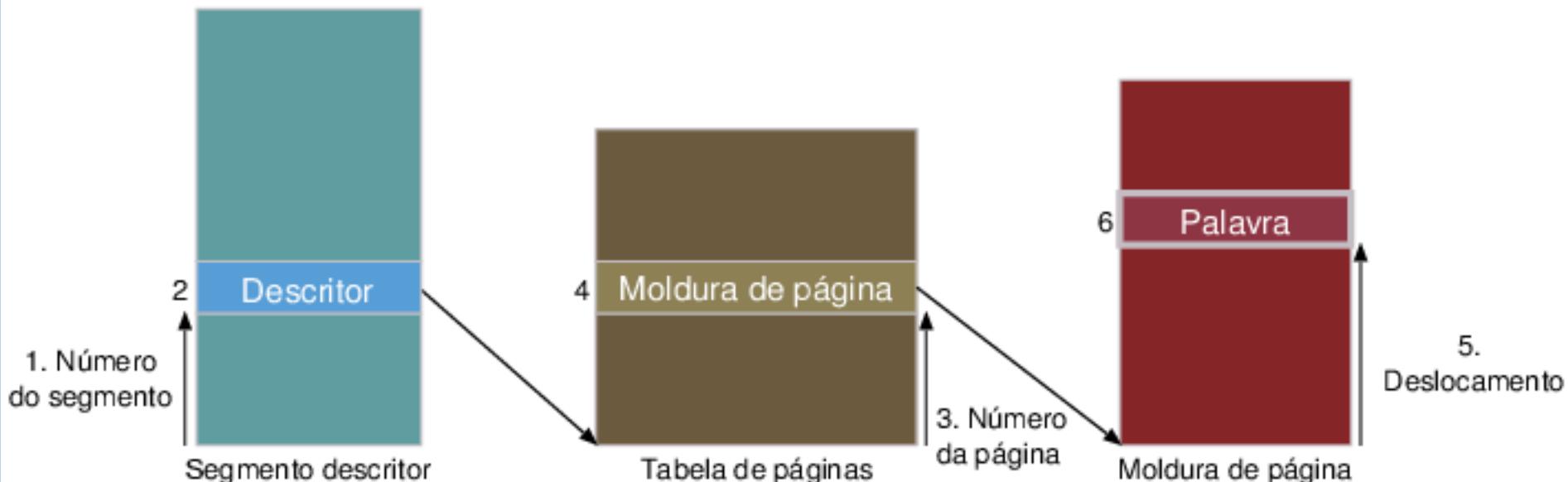


# Segmentação com paginação

→ O endereço virtual do MULTICS é dividido do seguinte modo:



→ O MULTICS converte o endereço virtual no físico do seguinte modo:



6. A palavra da memória agora pode ser lida (se a instrução desejava ler a memória), ou ser gravada (se a instrução desejava alterar a memória).



G/G

# Segmentação com paginação

→ Como o acesso à memória seria ineficiente, podemos, assim como antes, usar uma TLB:

Número do segmento	Página virtual	Moldura de página	Proteção	Idade	Válida
4	1	7	RW	13	1
6	0	2	R	10	1
12	3	1	RW	2	1
-	-	-	-	-	0
2	1	0	X	7	1
2	2	12	X	9	1
:	:	:	:	:	:

A TLB do MULTICS possui 16 entradas, com os endereços mais usados no sistema. Quando um endereço virtual é convertido, o sistema primeiro verifica se a entrada para este endereço está na TLB, procurando pelos campos número do segmento e página virtual paralelamente em todas as entradas da TLB, e o obtendo diretamente da TLB, caso a busca tenha sucesso. A TLB contém todas as entradas necessárias do descritor, além dos campos **Válida**, que indica se a entrada está sendo usada, e **Idade**, que é usado ao escolhermos uma entrada da TLB quando esta está cheia, porque foi acessado um endereço que não está na TLB.

# Segmentação com paginação

- Um outro exemplo é o da memória segmentada do PENTIUM:
  - Possui 16K de segmentos independentes, cujos tamanhos podem ser de até 4Gb (as palavras são de 32 bits).
  - São usadas dois tipos de tabelas de descritores:
    - A **LDT (Local Descriptor Table)** com os segmentos locais do programa (como o código, os dados, e a pilha).
    - A **GDT (Global Descriptor Table)** com os segmentos do sistema, como o código do sistema operacional.
  - Existe uma única GDT no sistema, mas existe uma LDT para cada processo em execução no sistema.
- Ao acessar um segmento, um programa no PENTIUM copia um seletor do segmento para um dos registradores de segmento:

Formato do seletor de um segmento. Existem vários registradores de segmento no processador, sendo que os principais são o CS para o código, o DS para os dados, e o SS para a pilha. O valor 0 é reservado, e é usado para indicar que um registrador não está em uso. Pressione o mouse sobre os campos para ver o seu significado.



# Segmentação com paginação

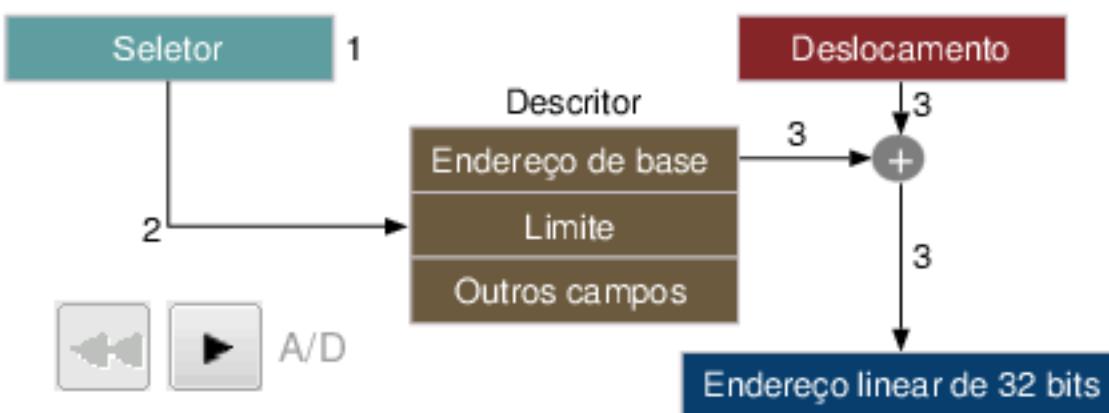
→ Quando um seletor é carregado em um registrador de segmento, o seu descritor:

- É buscado na LDT ou na GDT, e armazenado em registradores internos do processador, para um acesso mais rápido.
- O formato geral de um descritor de código é dado abaixo:



O tamanho do descritor de 8 bytes facilita a busca pelo seletor na GDT ou na LDT. Pressione o mouse sobre cada um dos campos para ver uma breve descrição do seu significado.

→ O endereço virtual é convertido em um linear do seguinte modo:



Modo como o processador Pentium converte um endereço virtual, composto pelo par seletor,deslocamento, em um endereço linear de 32 bits.

# Segmentação com paginação

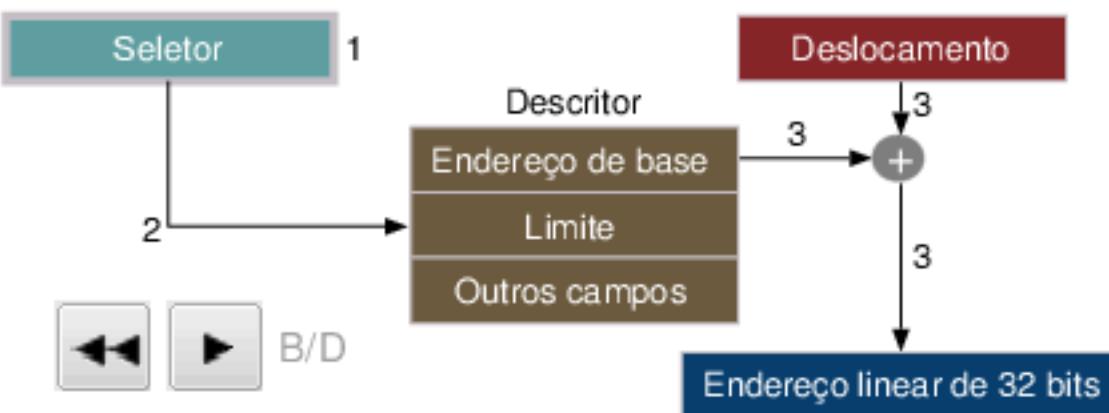
→ Quando um seletor é carregado em um registrador de segmento, o seu descritor:

- É buscado na LDT ou na GDT, e armazenado em registradores internos do processador, para um acesso mais rápido.
- O formato geral de um descritor de código é dado abaixo:



O tamanho do descritor de 8 bytes facilita a busca pelo seletor na GDT ou na LDT. Pressione o mouse sobre cada um dos campos para ver uma breve descrição do seu significado.

→ O endereço virtual é convertido em um linear do seguinte modo:



1. O programa carrega o seletor para o segmento em um dos registradores de segmento. Se o segmento não existe (seletor 0), uma interrupção é gerada.

# Segmentação com paginação

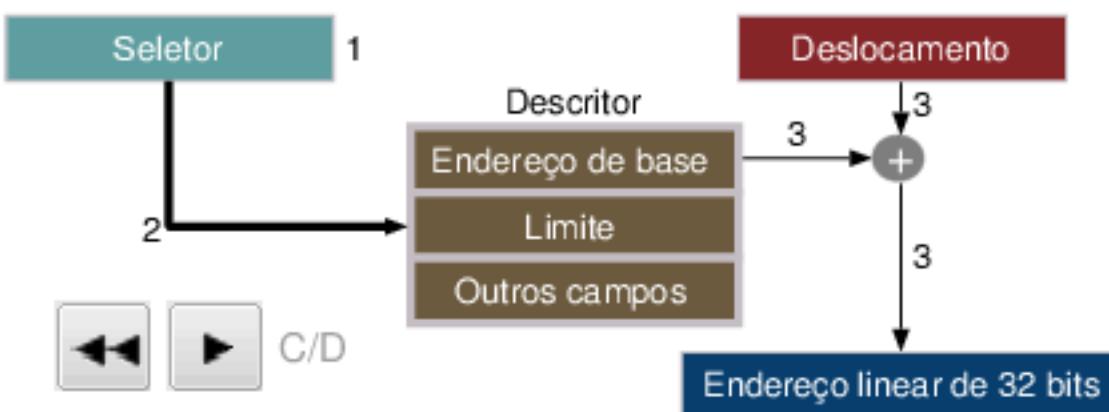
→ Quando um seletor é carregado em um registrador de segmento, o seu descritor:

- É buscado na LDT ou na GDT, e armazenado em registradores internos do processador, para um acesso mais rápido.
- O formato geral de um descritor de código é dado abaixo:



O tamanho do descritor de 8 bytes facilita a busca pelo seletor na GDT ou na LDT. Pressione o mouse sobre cada um dos campos para ver uma breve descrição do seu significado.

→ O endereço virtual é convertido em um linear do seguinte modo:



2. O descritor do segmento é localizado, e copiado para um registrador interno. Uma interrupção será gerada se este segmento não estiver na memória, se o tipo de acesso não for permitido, ou se o deslocamento for maior do que o limite.

# Segmentação com paginação

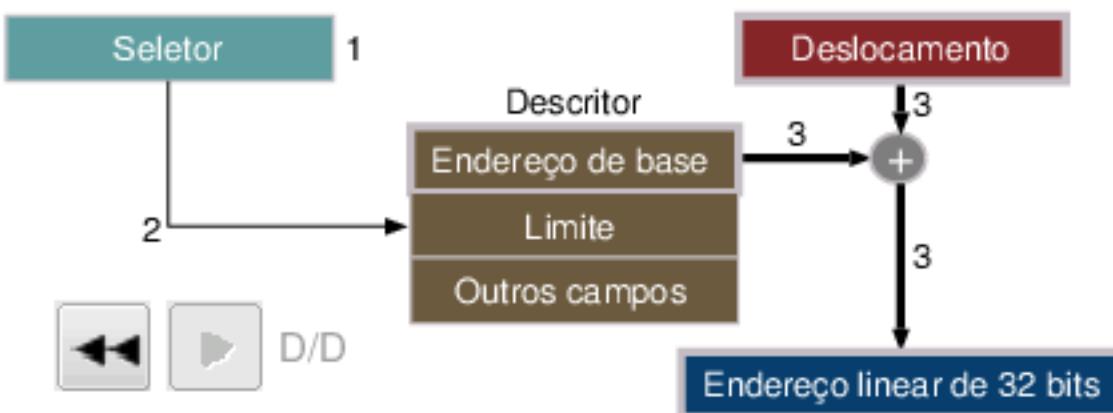
→ Quando um seletor é carregado em um registrador de segmento, o seu descritor:

- É buscado na LDT ou na GDT, e armazenado em registradores internos do processador, para um acesso mais rápido.
- O formato geral de um descritor de código é dado abaixo:



O tamanho do descritor de 8 bytes facilita a busca pelo seletor na GDT ou na LDT. Pressione o mouse sobre cada um dos campos para ver uma breve descrição do seu significado.

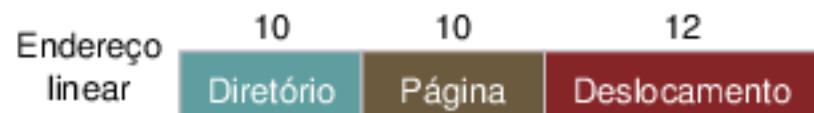
→ O endereço virtual é convertido em um linear do seguinte modo:



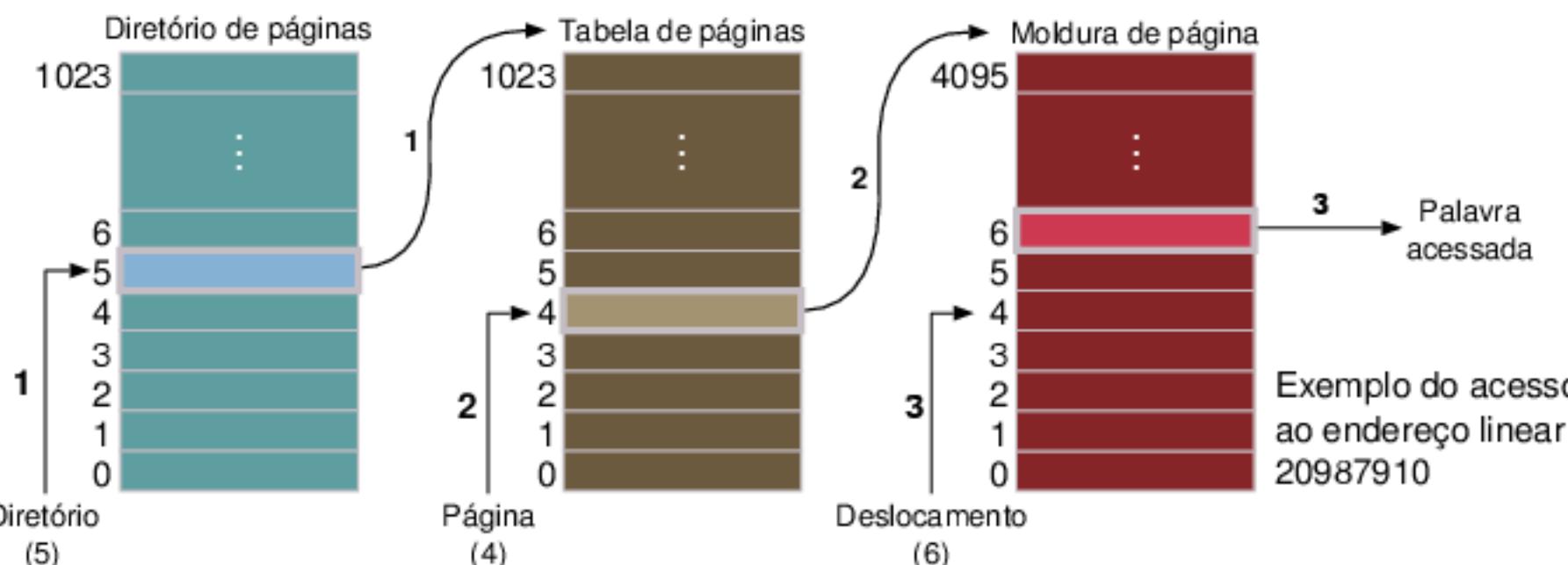
3. Se uma interrupção não foi gerada, o deslocamento é então somado ao endereço base do segmento para obter um endereço linear de 32 bits. Agora, se a paginação não estiver habilitada, este é o endereço enviado à memória.

# Segmentação com paginação

→ O endereço linear é convertido em um físico usando uma tabela de páginas de dois níveis, se a paginação estiver habilitada:



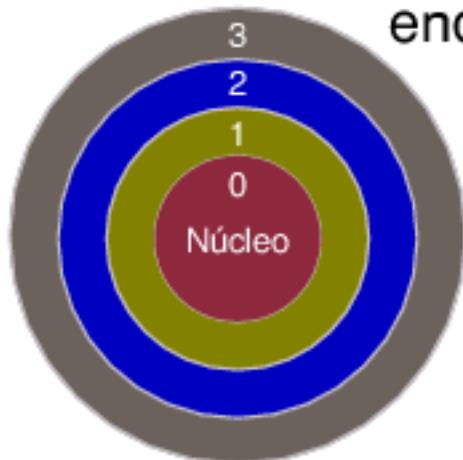
A divisão do endereço linear do modo dado ao lado é devido ao fato de o tamanho das tabelas serem de 1K, e do tamanho das páginas serem de 4K.



1. O campo diretório é usado para obter o endereço da tabela de páginas.
2. O campo página é usado para obter a moldura com o endereço linear.
3. O deslocamento é adicionado ao endereço da moldura, e o endereço linear é finalmente acessado.

# Segmentação com paginação

- ➡ O esquema de proteção do Pentium possui quatro níveis, variando do 0 (o mais privilegiado) ao 3 (o menos privilegiado).
- ➡ Cada segmento e cada seletor possui um nível de proteção. Agora, um programa em execução:
  - Está em um nível dado pelo maior valor entre a sua PSW e o seletor do segmento acessado.
  - Não pode acessar os dados de níveis mais privilegiados.
  - Pode chamar qualquer função em seu nível.
  - Chamadas a funções de outros níveis são feitas usando seletores especiais chamados de **portões de chamada**:
  - Não pode-se fazer chamadas a endereços arbitrários, pois o endereço é dado no descritor indexado pelo seletor.



Um possível uso para a proteção baseada em níveis seria o seguinte:

Nível 0: núcleo do sistema operacional.

Nível 1: chamadas ao sistema operacional.

Nível 2: as bibliotecas compartilhadas pelos programas do sistema.

Nível 3: os programas do usuário.