Gerência de memória

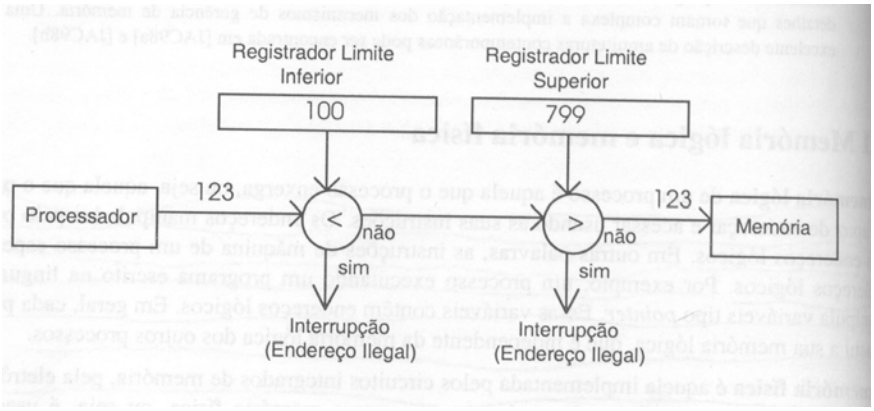
Memória lógica e memória física

A **memória lógica** de um processo é aquela que o processo enxerga, ou seja, aquela que o processo é capaz de endereçar e acessar usando as suas instruções. Os endereços manipulados pelo process são endereços lógicos. Em outras palavras, as instruções de máquina de um processo especificam endereços lógicos. Por exemplo, um processo executando um programa escrito na linguagem C manipula variáveis tipo *pointer*. Essas variáveis contém endereços lógicos. Em geral, cada processo possui a sua memória lógica, que é independente da memória lógica dos outros processos.

A **memória física** é aquela implementada pelos circuitos integrados de memória, pela eletrônica do computador. O endereço físico é aquele que vai para a memória física, ou seja, é usado para endereçar os circuitos integrados de memória.

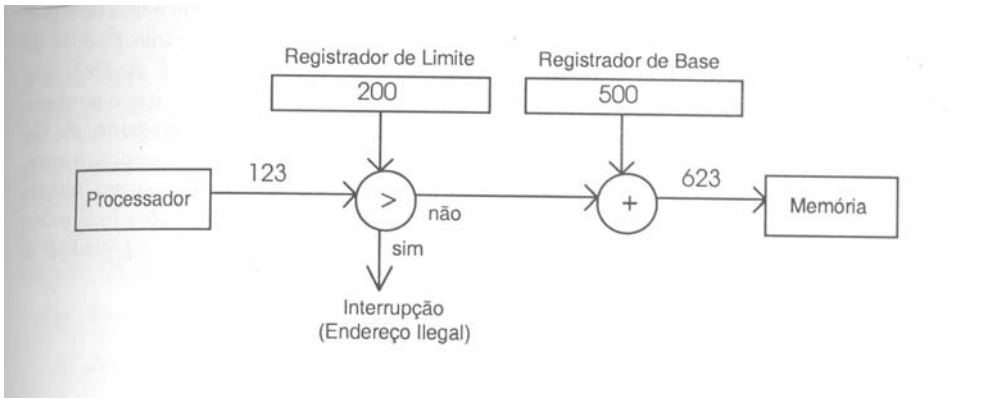
O **espaço de endereçamento lógico** de um processo é formado por todos os endereços lógicos que esse processo pode gerar. Existe um espaço de endereçamento lógico por processo. Já o **espaço de endereçamento físico** é formado por todos os endereços aceitos pelos circuitos integrados de memória.

A **unidade de gerência de memória** (*Memory Management Unit, MMU*) é o componente do hardware responsável por prover os mecanismos básicos que serão usados pelo sistema operacional para gerenciar a memória. Entre outras coisas, é a MMU que vai mapear os endereços lógicos gerados pelos processos nos correspondentes endereços físicos que serão enviados para a memória. Na verdade, o processador e a MMU formam, na maioria das vezes, um único circuito integrado. Podemos considerar os dois registradores de limite como uma MMU muito simples.



Existe uma outra forma de MMU simples, onde o endereço lógico gerado pelo processo é primeiro comparado com um limite superior. Caso seja menor ou igual, ele então é somado ao valor do registrador de base. O resultado da soma é o endereço físico que vai para a memória. Nesse esquema, o endereço lógico é transformado

em endereço físico através da soma do valor da base. Temos então o endereço lógico diferente do respectivo endereço físico.



Nesse esquema de MMU, o espaço de endereçamento lógico vai de zero até o valor limite. Esses são os endereços de memória manipulados pelo processo. O processo pode gerar endereços lógicos entre zero e 200. Qualquer valor fora desse intervalo será considerado ilegal. Os endereços lógicos são mapeados pela MMU para uma área do espaço de endereçamento físico. Essa área da memória física inicia no valor indicado pelo registrador de base e tem o mesmo tamanho da memória lógica do processo. A proteção de memória é conseguida, pois o processo de usuário está restrito a essa área da memória física.Tanto os registradores de limite inferior e superior quanto os registradores de base e limite devem ser protegidos. Eles não podem ser acessados em modo usuário. Obviamente, eles podem ser acessados em modo supervisor. Quando ocorre um chaveamento de processo, os valores são copiados do DP para os registradores da MMU.

Uma diferença entre as soluções apresentadas está na carga dos programas. No esquema que emprega apenas registradores de limite, os programas são gerados para o endereço zero da memória. Dessa forma, no momento da carga, os endereços dop programa devem ser corrigidos para que o programa execute corretamente no lugar onde foi colocado. Esse processo de correção de endereços é chamado **relocação**. Um carregador que efetua uma relocação do programa em tempo de carga é chamado de carregador relocador.

No esquema que emprega registradores de base e limite, todos os programas são também gerados para o endereço zero da memória. Entretanto, eles podem ser carregados em qualquer lugar da memória física.

Um carregador de programas que não precisa corrigir os endereços durante a carga é chamado de **carregador absoluto**. Nesse esquema, podemos considerar que ocorre uma relocação em tempo de execução, pois cada endereço sofre uma correção automática ao ser somado com o conteúdo do registrador de base.

Partições Fixas

Partições fixas são a forma mais simples de gerência de memória para multiprogramação. A memória é primeiramente dividida em uma parte para uso do SO e uma parte para uso dos processos de usuários. A seguir, a parte dos usuários é dividida em várias partições de tamanhos diferentes porém fixos.Quando um programa deve ser carregado, é escolhida uma partição ainda livre. Obviamente a partição deve ter um tamanho igual ou maior que o programa.

Existem, porém, dois problemas com esse tipo de gerência de memória. Dificilmente o programa a ser carregado terá o tamanho exato de uma partição. Ele será carregado em uma partição que é um pouco maior que o necessário. Isso resulta em um desperdício de memória que é chamado de **fragmentação interna**, isto é, memória perdida dentro da área alocada para um processo. Outra possibilidade é termos duas partições livres, digamos, de 25 e 100 kbytes. Nesse momento é criado um processo para executar um programa de 110 kbytes. Observe que a memória total livre no momento é de 125 Kbytes, mas ela não é contígua. O programa não pode ser executado devido à forma como a memória é gerenciada. Esse tipo de problema é chamado de **fragmentação externa**, isto é, memória perdida fora da área ocupada por um processo.

Partições variáveis

Quando variáveis são empregadas, o tamanho das partições é ajustado dinâmicamente às necessidades exatas dos processos. Essa é uma técnica de gerência de memória mais flexível que partições fixas.

O SO mantém uma lista de lacunas, ou seja, espaços livres na memória física que são percorridos quando um processo é criado.

Existem 4 formas básicas de percorrer a lista de lacunas atrás de uma lacuna de tamanho suficiente. Os algoritmos **first-fit** e **circular fit** são os mais usados.

- **First-fit** – utiliza a primeira lacuna que encontrar com tamanho suficiente
- **Best-fit** – utiliza a lacuna que resultar na menor sobra
- **Worst-fit** – utiliza a lacuna que resultar na maior sobra
- **Circular-fit** – Igual ao first-fit, mas inicia a procura na lacuna seguinte à última sobra.

Quando um processo termina, a memória que ele ocupava é liberada. Isso corresponde à criação de uma nova lacuna. Caso a nova lacuna criada seja adjacente a outras lacunas, elas são unificadas.

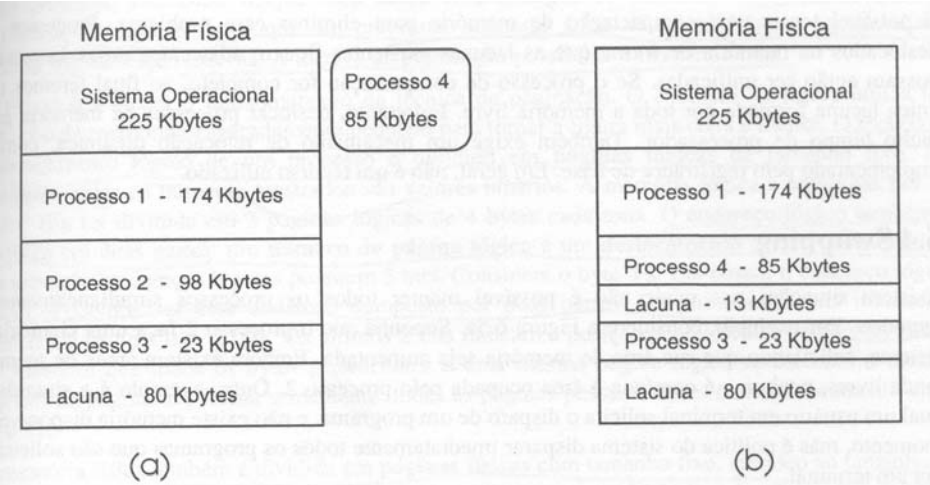


Figura 6.5 - Memória física dividida em partições variáveis.

Alguns sistemas organizam a memória por blocos de, por exemplo, 32 bytes. Esses blocos são muitas vezes chamados de **parágrafos**. A unidade de alocação passa a ser o parágrafo e o tamanho da área alocada por um processo deve ser um número inteiro de parágrafos, ou seja, um múltiplo exato de 32 bytes. Dessa forma, a menor lacuna possível terá o tamanho de um parágrafo ou 32 bytes. Nesse caso, poderemos ter uma fragmentação interna de até 31 bytes por processo. Em algumas arquiteturas, as variáveis do tipo inteiro devem ficar alinhadas corretamente na memória.

Partições variáveis são tipicamente implementadas através de uma lista encadeada de lacunas. Cada lacuna é representada por um descritor de lacuna, que contém basicamente o seu endereço, tamanho e apontadores para as colunas adjacentes. Como cada lacuna tem o tamanho mínimo de um parágrafo, ela pode hospedar o seu próprio descritor.

Com partições variáveis a fragmentação externa é um problema grave. Á medida que áreas de memória são alocadas e liberadas, muitos fragmentos são gerados. A memória adquire uma aparência de “queijo suíço”. É possível tentar usar compactação de memória para eliminar esse problema, mas em geral não é um processo muito usado.

Swapping

Existem situações nas quais não é possível manter todos os processos simultâneamente na memória.

Uma solução para essas situações é o mecanismo de **swapping**. Em determinadas situações, um processo é completamente copiado da memória para o disco. Sua execução é suspensa, ou seja, seu descritor de processo é removido da fila do processador e colocado em uma fila de processos suspensos. É dito que esse processo sofreu um **swap-out**. Mais tarde, ele sofrerá um **swap-in**, ou seja, será copiado novamente para a memória. Seu descritor de processo volta então para a fila do processador, e sua

execução será retomada. O resultado desse revezamento no disco é que o SO consegue executar mais processos do que caberia em um mesmo instante na memória.

Em sistemas nos quais uma pessoa interage com o programa durante a sua execução, o mecanismo de swapping somente é utilizado em último caso, quando não é possível manter todos os processos na memória. A queda no desempenho do sistema é imediatamente sentida pelo usuário no terminal.

Paginação

A técnica de partições fixas gera muita perda de memória e não é mais utilizada na prática. Se essa restrição for eliminada, ou seja, permitir que um programa ocupasse áreas não contíguas de memória, não haveria fragmentação externa. A técnica da **paginação** possibilita exatamente isso.

O espaço de endereçamento lógico de um processo é dividido em **páginas lógicas** de tamanho fixo. O endereço lógico também é dividido em duas partes: um **número de página lógica** e um **deslocamento** dentro dessa página.

A memória física também é dividida em **páginas físicas** com tamanho fixo, idêntico ao tamanho da página lógica. Os endereços de memória física também podem ser vistos como compostos de duas partes. Os 3 primeiros bits indicam um número de página física. Os 2 últimos bits indicam um deslocamento dentro dessa página física.

Um programa é carregado página a página. Cada página lógica do processo ocupa exatamente uma página física da memória física. Entretanto, a área ocupada pelo processo na memória física não precisa ser contígua. Mais do que isso, a ordem em que as páginas lógicas aparecem na memória física pode ser qualquer, não precisa ser a mesma da memória lógica.

Durante a carga é montada uma **tabela de páginas** para o processo. Essa tabela informa, para cada página lógica, qual a página física correspondente. No exemplo, a tabela é formada por 3 entradas, uma vez que o processo possui 3 páginas lógicas.

Quando um processo executa, ele manipula endereços lógicos. O programa é escrito com a suposição que ele vai ocupar uma área contígua de memória, que inicia no endereço zero, ou seja, vai ocupar a memória lógica do processo. Para que o programa execute corretamente, é necessário transformar o endereço lógico especificado em cada instrução executada, no endereço físico correspondente. Isso é feito com o auxílio da tabela de páginas.

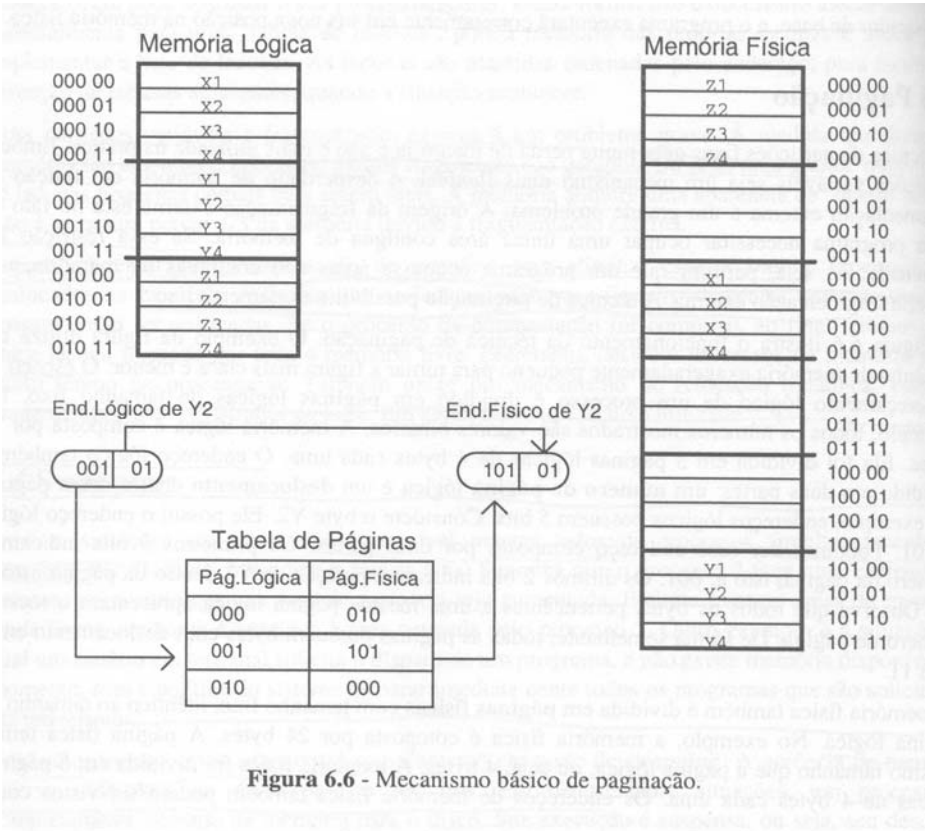


Figura 6.6 - Mecanismo básico de paginação.

O endereço lógico gerado é inicialmente dividido em duas partes: um número de página lógica e um deslocamento dentro da página. O número da página lógica é usado como índice no acesso à tabela de páginas. Cada entrada da tabela de páginas possui o mapeamento de página lógica para página física. Dessa forma, é obtido o número de página física correspondente. Já o deslocamento do byte dentro da página física será o mesmo deslocamento desse byte dentro da página lógica, pois cada página lógica é carregada exatamente em uma página física.

Na prática, os tamanhos de página variam entre 1K e 8Kbytes. Espaços de endereçamento lógico variam de 64K para sistemas antigos e muitos Gbytes nas máquinas atuais. Espaços de endereçamento físico também ficam, em geral, na ordem de Gbytes. Note que o espaço de endereçamento físico denota a capacidade de endereçamento do processador, e não a quantidade de memória realmente instalada na máquina.

Na paginação, uma página lógica pode ser carregada em qualquer página física que esteja livre, dessa forma não existe fragmentação externa.

Existem vantagens e desvantagens em utilizar páginas grandes. Páginas maiores significam que um processo terá menos páginas, a tabela de páginas será menor, a leitura do disco será mais eficiente. Em geral, páginas maiores resultam em um custo menor imposto pelo mecanismo de gerência de memória, ou seja, um **overhead** menor. Por outro lado, páginas maiores resultam em uma fragmentação interna maior. Normalmente não é o sistema

operacional que escolhe o tamanho das páginas. Esse valor é fixado pelo hardware que suporta a gerência de memória, ou seja, pela MMU do computador em questão.

Quando a tabela de páginas é pequena, ela pode ser completamente colocada em registradores de acesso rápido.

Quando a tabela de páginas é muito grande, não é possível mantê-la em registradores. Uma outra solução é manter a tabela de páginas na própria memória. A MMU possui então dois registradores para localizar a tabela na memória. O **registrador de base da tabela de páginas** (page table base register, PTBR) indica o endereço físico de memória onde a tabela está colocada. O **registrador de limite da tabela de páginas** (page Table Limit Register, PTLR) indica o número de entradas da tabela. O problema desse mecanismo é que agora cada acesso que um processo faz à memória lógica transforma-se em dois acessos à memória física. No primeiro acesso, a tabela de páginas é consultada, e o endereço lógico é transformado em endereço físico. No segundo acesso, a memória do processo é lida ou escrita.

Uma forma de reduzir o tempo de acesso à memória no esquema anterior é adicionar uma memória cache especial que vai manter as entradas da tabela de páginas mais recentemente utilizadas. Essa memória cache interna à MMU é chamada normalmente de **Translation Lookaside Buffer** (TLB).

Normalmente, a memória cache é implementada através de um componente de hardware conhecido como **memória associativa**, que inclui algumas células de memória e toda eletrônica necessária para fazer uma pesquisa paralela, incluindo as células. O problema aí é o preço do hardware.

Quando a memória cache é usada e ocorre um chaveamento de processos, novamente os valores do PTBR e do PTLR para a tabela de páginas de processo que recebe o processador devem ser copiados do DP para os registradores na MMU. Além disso, a memória cache deve ser esvaziada (flushed)

É importante observar que a proteção entre processos é facilmente conseguida com uma MMU que suporte paginação. Em primeiro lugar, o mecanismo de paginação garante que cada processo somente tenha acesso às páginas físicas que constam em sua tabela de páginas. Essa tabela de páginas é construída pelo SO e fica em uma região da memória à qual apenas o SO tem acesso. O acesso aos registradores PTBR e PTLR é privilegiado, isto é, restrito ao código do SO, que executa em modo supervisor.

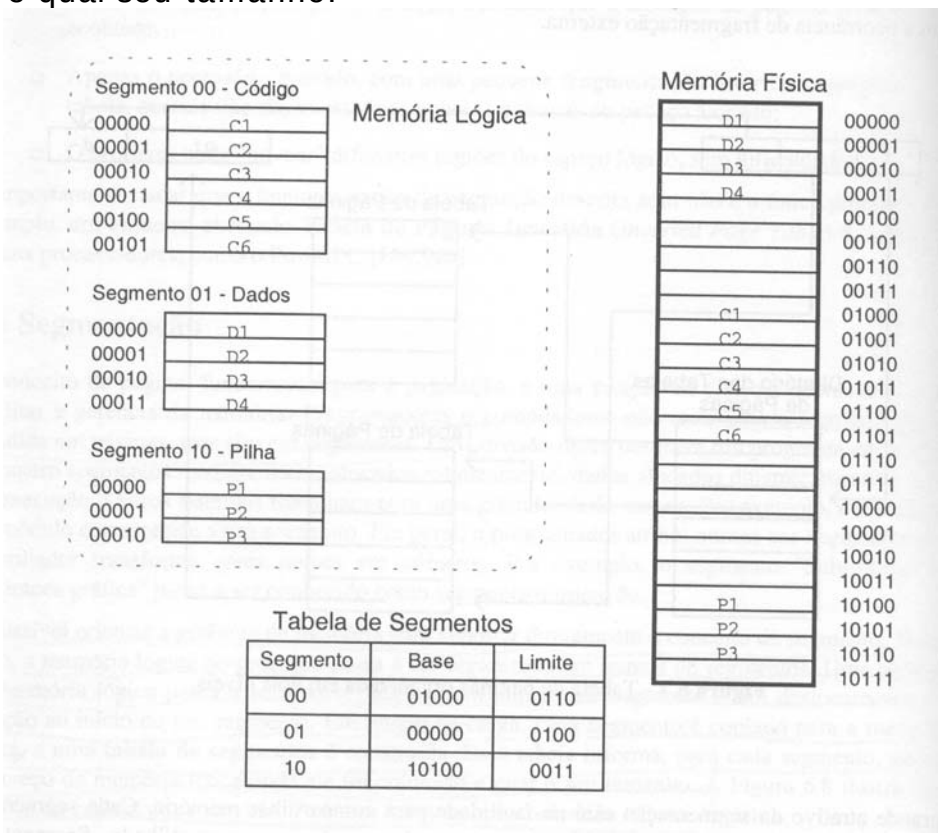
Em sistemas atuais, a tabela de páginas pode ser muito grande. Assim sendo, é muito raro vê-la sendo totalmente usada. Na prática, as tabelas de páginas possuem um tamanho variável, ajustado à necessidade de cada processo. Ocorre que, se as tabelas puderem ter qualquer tamanho, então teremos fragmentação externa novamente (e a maior razão para usar paginação foi a eliminação da fragmentação externa).

Para evitar isso, são usadas tabelas com dois níveis. As tabelas de páginas crescem de pedaço em pedaço, e uma tabela auxiliar chamada diretório mantém o endereço de cada pedaço. Para evitar a fragmetação externa, cada pedaço da tabela de páginas deve ter um número inteiro de páginas físicas, mantendo assim toda alocação de memória física em termos de páginas, não importando a sua finalidade. Entradas desnecessárias em cada pedaço são marcadas como inválidas.

Segmentação

Programadores e compiladores não enxergam a memória lógica dividida em páginas, mas sim em segmentos. Uma divisão típica descreve um programa em termos de 4 segmentos: Código, dados alocados estaticamente, dados alocados dinamicamente e pilha e execução.

É possível orientar a gerência de memória para suportar diretamente o conceito de segmento. Nesse caso, a memória lógica do processo passa a ser organizada em termos de segmentos. Uma posição da memória lógica passa a ser endereçada por um número de segmento e um deslocamento em relação ao início do seu segmento. Em tempo de carga, cada segmento é copiado para a memória física, e uma tabela de segmentos é construída. Essa tabela informa, para cada segmento, qual o endereço da memória física onde ele foi colocado e qual seu tamanho.



Segmento	Base	Limite
00	01000	0110
01	00000	0100
10	10100	0011

Os processos geram endereços lógicos compostos por um número de segmento e um deslocamento dentro do segmento. A MMU

inicialmente utiliza o número de segmento fornecido para indexar a tabela de segmentos.

O grande atrativo da segmentação está na facilildade para compartilhar memória. Cada segmento representa uma parte específica do programa, podendo ou não ser compartilhado.

Por exemplo, suponha que o código das rotinas de biblioteca de uma linguagem de programação é compilado como sendo um segmento único. Esse segmento é marcado como “para apeas execução”, ou seja, não pode ser lido nem escrito. Todos os programas escritos nessa linguagem utilizam esse segmento. Entretanto, apenas uma cópia dele é necessária na memória física. Todos os processos executando programas escritos nessa linguagem terão em sua em sua respectiva tabela de segmentos uma referência à posição desse segmento na memória física.

Segmentação Paginada

Na segmentação paginada o espaço lógico é formado por segmentos, e cada segmento é dividido em páginas lógicas. Cada segmento possui uma tabela de páginas associada. No momento de endereçar a memória, a tabela de segmentos indica, para cada segmento, onde a respectiva tabela de páginas está. Essa tabela de páginas é usada para transformar o endereço de página lógica de determinado segmento em endereço de página física, como é feito normalmente na paginação. Sempre lembrando que este não é o único meio de trabalhar a segmentação paginada. Abaixo, vemos o esquema clássico de segmentação paginada.