MÉTODO DE ACESSO INDEXADO

PARA O TERMINAL INTELIGENTE

SERGIO ZARUR FAISSOL

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIENCIAS (M.SC)

APROVADA por:

.....

(Presidente)

.....

.....

.....

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 1977

FAISSOL, SEPGIO ZAPUR

MÉTODO DE ACESSO INDEXADO PAPA O TERMINAL INTELIGENTE (RIO DE JANEIPO) 1977

VII, 98P. 29,7 CM (COPPE-UFRJ, M.SC, ENGENHAPIA DE SISTEMAS , 1977)

TESE - UNIVERSIDADE FEDERAL DO RIO DE JANEIPO FACULDADE DE ENGENHARIA

1. IMPLEMENTAÇÃO DE UM MÉTODO DE ACESSO INDEXADO NO SISTEMA OPEPACIONAL DO TERMINAL INTELIGENTE DA UNIVERSIDADE DO RIO DE JANEIRO I. COPPE/UFRJ II. TÍTULO (SÉRIE)

RESUMO

Esta tese apresenta um Método de Acesso Indexado para o Terminal Inteligente da Universidade Federal do Rio de Janeiro, que fara' parte do seu Sistema Operacional em Disco (SOCO).

E' feita uma análise das diversas técnicas de indexação e organização de dados. Em seguida e' apresentada a organização adotada, que utiliza para índice uma estrutura chamada ARVORE B.

O método permite inserções e remoções em qualquer ordem praticamente sem degradação na performance e um processamento sequencial eficiente.

E' tambem apresentada a maneira de utilizar o método de acesso e uma previsão de performance baseada em simulação.

ABSTRACT

This thesis presents an Indexed Access Method for the Intelligent Terminal of Federal University of Rio de Janeiro, which will be a component of it's Operating Sistem (SOCO).

An analysis of indexing and data organizations is made. Then the adopted organization, which uses the b-tree data structure for the index is presented.

This access method allows insertions and remotions in any order with almost no performance degradation and efficient sequential processing.

The commands for the user are also presented, together with performance prediction, obtained by simulation.

ÍNDICE

APENDICES

I- INTRODUÇÃO

Este trabalho apresenta um Método de Acesso Indexado para o Terminal Inteligente do Nucleo de Computação Eletronica da Universidade Federal do Rio de Janeiro.

O Terminal Inteligente (TI) e' um microcomputador com ate' 16 kbytes de memória e uma CPU INTEL 8008, com a possibilidade de controlar uma linha de comunicação, leitora de cartões, impressora, video, fita cassete, disco, disquete ou qualquer outro periférico compativel com a linha PDP.

O presente método de acesso sera' parte do "software" básico do Terminal Inteligente e sera' usado com o sistema operacional em disco (SOCO). Devera' permitir acesso direto, com chaves, e um acesso sequencial eficiente em disco. A flexibilidade do sistema operacional de permitir a substituição do periférico em tempo de execução sera' mantida no caso de processamento sequencial. Tambem sera' permitida a extensão do arquivo no mesmo ou em outro disco, vedando-se no entanto a concatenação de arquivos indexados.

O método de acesso sera' implementado em PLTI, uma linguagem de alto nivel projetada para o Terminal Inteligente. Como o compilador PLTI ainda não estava terminado na época de desenvolvimento deste trabalho, foi feita uma simulação do PLTI usando o compilador PL/I. Os

"interfaces" com outras partes do SOCO, tais como carregador de programas e rotina de E/S física, foram tambem simulados em PL/I.

II- ARQUIVOS INDEXADOS

Ha' basicamente tres tipos de metodos de acesso em um sistema operacional moderno, para atender aos diversos tipos de aplicacoes:

- sequencial
- direto
- indexado

O metodo de acesso mais simples e' o sequencial, sendo o mais indicado para o processamento de arquivos em que a percentagem de registros recuperados e' muito alta. O sistema operacional do TI ja' dispoe de um metodo de acesso sequencial, com as funcoes basicas READ, WRITE, REWIND e ENDFILE.

Dispoe tambem de um metodo de acesso direto em que os registros sao recuperados pelo seu endereco relativo no arquivo, atraves da funcao "SEEK". Em geral, o usuario devera' usar uma tecnica de "HASHING" para obter os enderecos dos registros. Este tipo de metodo de acesso e' adequado 'as aplicacoes em que o numero de recuperacoes e' baixo em relacao ao tamanho do arquivo. Um criterio tipico e' abaixo de 10%. O metodo tambem permite acesso sequencial, embora com uma eficiencia reduzida, uma vez que os dados nao estao ordenados.

Uma grande classe de aplicacoes tem uma taxa de recuperacao de registros intermediaria, em que seria desejavel acesso sequencial eficiente, e acesso direto rapido. Para estas aplicacoes, um metodo de acesso indexado e' o mais indicado. O indice e' necessario porque os registros de dados devem ser mantidos ordenados a fim de permitir processamento sequencial, que seria inviavel se o acesso fosse por "HASHING". Neste trabalho estamos implementando este metodo de acesso, sem o qual o sistema operacional do TI estaria incompleto. Alem da implementacao, em PLTI, apresenta-se a maneira de utilizar o metodo de acesso, sua estrutura interna, e uma previsao da performance obtida atraves de simulacao. Tambem sao apresentadas as diversas rotinas e suas descricoes, assim como os "interfaces" com outras partes do SOCO , a fim de permitir a manutencao ou futuras alteracoes no sistema.

II-1 Caracteristicas Basicas

Um metodo de acesso indexado destina-se, como ja' foi dito, 'a classe de aplicacoes em que e' necessario processamento sequencial e direto, devendo atuar com eficiencia em ambos os casos. Obviamente, nao tera' a mesma eficiencia dos metodos sequencial e direto no processamento 100% sequencial ou 100% direto, caso contrario eles nao

seriam necessarios. Podemos medir a eficiencia de um tal metodo de acesso na medida em que ele se aproxima dos outros dois nos extremos da sua faixa de atuacao, isto e': Quantas operacoes de e/s fisicas sao realizadas no metodo de acesso indexado, e quantas seriam realizadas nos outros dois.

A seguir estao as caracteristicas basicas que um metodo de acesso indexado deve atender.

1- Os registros logicos devem estar armazenados na sequencia das chaves, para permitir um processamento sequencial eficiente.

2- Devem ser permitidas insercoes, remocoes ou atualizacoes em qualquer quantidade e em qualquer ordem sem degradacao na performance.

3- O acesso direto deve ser feito com um minimo de operacoes de e/s.

4- O espaco liberado por remocoes deve ser totalmente reutilizado.

5- O arquivo deve poder crescer sem a necessidade de reorganizacoes e sem perda na eficiencia.

6- A utilizacao do metodo devera' ser simples para o usuario.

7- Um arquivo indexado devera' poder substituir, em tempo de execucao, um sequencial, sem necessidade de alteracao ou recompilacao no programa, desde que seja feito apenas acesso sequencial.

8- Todas as operacoes devem poder ser executadas em qualquer ordem, isto e': Processamento sequencial e direto podem ser intercalados livremente, assim como operacoes de leitura e gravacao.

II-2 Tipos de Arquivos

Arquivos indexados sao divididos em duas areas: Indice e dados. E' na organizacao e interligacao destas duas areas que os diversos metodos de acesso indexados diferem entre si.

Este capitulo descreve as tecnicas de indexacao e de organizacao de dados mais encontradas nos metodos de acesso indexados existentes. No capitulo iii e' apresentada a organizacao adotada no metodo de acesso indexado do TI.

II-2.1 Tecnicas de Indexacao

O elemento basico de um indice e' uma ENTRADA. Ela e' composta de um valor (CHAVE) e um "pointer" para o registro que contem aquele valor. Para facilitar a busca e a manutencao, as entradas sao colocadas no indice por ordem de chave. Indices que usam "hashing" nao tem esta caracteristica e por isto nao permitem acesso ao sucessor de um registro, operacao indispensavel 'a nossa aplicacao. Tambem nao permitem a extensao do indice, o que elimina a sua utilizacao em arquivos indexados.

A ordenacao dos registros permite a construcao de indices **NAO DENSOS**. A ideia e' dividir o arquivo indexado em grupos, com varios registros por grupo (na pratica um grupo pode ser uma trilha, um setor ou qualquer outra unidade conveniente de memoria). O indice contem uma entrada para cada grupo, contendo em geral a maior chave do grupo. Dentro de cada grupo os registros estao em sequencia. O termo **NAO DENSO** significa que o indice contem entradas somente para alguns registros do arquivo, ao contrario dos indices **DENSOS**. Uma segunda decorrencia da ordenacao dos registros e' a possibilidade de construir indices com diversos niveis.

O objetivo de um indice e' evitar a pesquisa sequencial do arquivo. No entanto e' necessaria uma pesquisa sequencial ou binaria no indice. Consideremos, por exemplo, um indice com 40000 entradas em 400 registros de 100 entradas cada. Para localizar uma entrada neste indice, usando uma pesquisa binaria seria necessario examinar ate' 16 entradas em 8 registros diferentes. Isto representa um "overhead" muito grande porque, em geral, ha' uma operacao de E/S para cada grupo examinado.

A solucao e' construir um indice para o indice. Este segundo indice contem uma entrada para cada registro do indice original. No exemplo anterior este segundo nivel de indice contera' 400 entradas divididas em 4 registros. Podemos tambem construir um terceiro nivel de indice sobre o

segundo. Assim, no exemplo anterior apenas 3 registros seriam examinados para localizar uma chave.

Um exemplo muito comum desta tecnica e' o seguinte: O arquivo e' dividido em grupos de uma trilha e o indice de trilhas contem uma entrada para cada trilha. Este indice tambem e' dividido em grupos, cada um dos quais contem entradas para um cilindro e um indice de cilindros contem entradas para tais grupos. Pode haver outros niveis que indexem grupos de indices de cilindros. Tal tecnica e' usada pelo **ISAM** (Indexed Sequential Access Method) da IBM.

Um outro tipo de indice muito utilizado e' a arvore binaria, particularmente a arvore binaria balanceada. Esta classe de indices, no entanto, e' mais adequada ao caso em que todo o indice esta' na memoria principal. Estruturas em arvore comportam-se muito bem quando armazenadas em disco se escolhermos uma representacao adequada para os nos. Grupando-se os nos em paginas, tem-se as arvores conhecidas como arvores **"MULTIWAY"**, nas quais cada pagina tem **N+1** filhos, onde **N** e' o numero de nos por pagina.

Enquanto o numero de operacoes de E/S em uma arvore binaria e' proporcional ao logaritmo base 2 do numero de entradas, em uma arvore com **N** entradas por pagina o numero de operacoes e' proporcional ao logaritmo em base **N**, o que representa uma grande melhora. Um importante exemplo deste

tipo de estrutura e' o indice do VSAM (Virtual Storage Access Method) da ibm.

Um novo tipo de indice com estrutura em arvore "MULTIWAY" foi proposto em 1971 por R. Bayer e E. Mccreight (1), denominado ARVORE B. Esta estrutura torna possivel buscas e atualizacoes em um arquivo de grandes proporcoes com eficiencia garantida no pior caso, usando algoritmos relativamente simples. Esta e' a estrutura de indice escolhida para o metodo de acesso indexado do TI e esta' descrita na secao III-1.

II-2.2 Manutencao dos Dados

Quando dados apontados por um indice sao alterados, seja por insercao, remocao ou atualizacao, pode ser necessaria uma atualizacao no indice. Portanto a manutencao do indice deve seguir-se 'a manutencao dos proprios dados, cujas tecnicas sao analisadas nesta secao.

Para permitir um acesso sequencial eficiente e' necessario que os registros estejam ordenados dentro de cada bloco fisico, e que estes blocos estejam tambem ordenados. Sob este aspecto os diversos metodos de acesso indexados sao iguais. As diferencas ocorrem nas insercoes e remocoes.

INSERCOES

Ha' basicamente duas estrategias de insercao: Local e externa, e a utilizacao de uma ou outra e' de certa forma dependente do tipo de indice usado. Caso o indice esteja ordenado na mesma sequencia que os registros de dados e se utilize desta vantagem, isto e', seja **NAO DENSO**, necessita-se uma estrategia de insercao local para manter a caracteristica nao densa do indice.

A estrategia de insercao mais simples e' a externa, e requer uma unica area de "overflow", geralmente no fim do arquivo. As insercoes sao colocadas nesta area de forma sequencial. Este procedimento funciona bem com indices **DENSOS**, mas ha' uma grande degradacao na busca sequencial em caso de grande numero de insercoes. O metodo de acesso **ISAM** e' um exemplo desta estrategia. No entanto, para evitar o uso de indices **DENSOS**, faz-se um encadeamento dos registros na area de "overflow".

A insercao local e' mais complexa e requer areas de insercao multiplas. Este tipo de insercao normalmente requer deslocamento de dados e encadeamento ou divisao de blocos de dados.

O metodo de acesso **VSAM** e' um exemplo de uso desta estrategia. A area de dados esta' dividida em "control areas" e estas em "control intervals", que sao apontados pelo

indice, e contem registros em sequencia. Para facilitar as insercoes pode ser deixado espaco livre nos "control intervals". Se um "control interval" esta cheio, e' alocado um novo, dentro da "control area", e os registros sao divididos igualmente. Esta e' a estrategia de insercao adotada para o TI.

REMOCOES

A maneira mais simples de remover um registro de um arquivo e' colocar uma marca especial que o coloca fora de uso, mas nao libera o espaco. Neste caso nao ha' necessidade de alteracao no indice.

Uma estrategia que libera o espaco pode ser usada removendo o registro do bloco e deslocando as demais entradas. Assim este bloco podera receber outro registro no lugar do que foi removido. Se for removido o ultimo registro do bloco, a sua entrada no indice sera' tambem removida.

III- ORGANIZACAO DO ARQUIVO

A organizacao do disco do TI e' em setores de 512 bytes, com enderecos sequenciais. Cada operacao de E/S fisica transmite um setor inteiro, o que sugere a utilizacao do setor como unidade fisica de dados.

Os setores de um arquivo indexado serao divididos em tres categorias:

- indice
- dados
- controle

O primeiro setor fisico do arquivo contem informacoes de controle necessarias 'a criacao da Tabela De Arquivos Fisicos (TAF) e esta descrito na proxima secao.

Um certo numero de setores e' reservado em tempo de alocacao do arquivo para conter o indice, que sera' concentrado primariamente em uma regiao continua para reduzir o movimento do braco do disco nas pesquisas ao indice. Se durante o processamento a regiao reservada pelo usuario for insuficiente serao usados setores da area de dados para o indice.

Cada entrada no indice contem o endereco de um setor de dados e a chave do ultimo registro logico deste setor.

Os setores seguintes sao reservados para os dados, cada um contendo um numero variavel de registros logicos. Uma percentagem do espaco de cada setor de dados pode ser reservada para futuras insercoes.

III-1 Indice

O indice esta' organizado em paginas de tamanho fixo, que podem estar apenas parcialmente cheias. Cada pagina esta' contida em um setor do disco do TI. Estas paginas sao os nos de uma ARVORE B (1) descrita na proxima secao.

Esta arvore aumenta ou diminui de tamanho de apenas uma forma: Um no' se divide em dois irmaos, ou dois irmaos se juntam para formar um no'. O processo de divisao ("split") ou uniao ("catenation") inicia-se nas folhas e pode se propagar ate' a raiz da arvore sendo esta a maneira pela qual a sua altura pode aumentar ou diminuir.

Este esquema oferece diversas vantagens em relacao aos indices tradicionais de arquivos sequencial indexados, baseados em indices de cilindros e trilhas, e tambem apresenta vantagens em relacao aos indices do tipo "hashing", destacando-se as seguintes:

1- A memoria auxiliar e' alocada e liberada com o crescimento ou retracao do arquivo, nao havendo

degradacao na performance em caso de taxas altas de ocupacao de memoria, (como no caso de "hashing") nem a utilizacao de areas de "overflow".

2- A utilizacao de memoria e' de 50% no minimo sendo em media muito maio.

3- A ordem natural das chaves e' mantida, o que permite intercalar processamento direto e sequencial livre -mente.

4- Se as insercoes, remocoes, ou buscas vierem ordenadas e em grupos, o processamento e' muito eficiente, praticamente sequencial.

III-1.1 Arvores B

Sejam H e K naturais, K nao nulo. Uma arvore T pertence 'a classe T(H,K) de ARVORES B se T e' vazia (H=0) ou se tem as seguintes propriedades:

1- Cada caminho da raiz ate' uma folha tem o mesmo tamanho H , chamado altura da arvore.

2- Cada no' exceto a raiz e as folhas tem no minimo K+1 filhos. A raiz tem no minimo 2 filhos.

3- Cada no' tem no maximo 2K+1 filhos.

Cada no' de uma ARVORE B contem de K a 2K elementos de indice. Seja I o numero de elementos no indice. A altura da arvore, que determina o tempo de pesquisa e' limitada por (1):

$$\log_{2K+1}(I+1) \leq H \leq \log_{K+1}((I+1)/2)+1$$

III-1.2 Estrutura do Indice

Cada pagina contem L entradas do indice, $K \leq L \leq 2K$, exceto a raiz para a qual temos $1 \leq L \leq 2K$. Cada entrada no indice e' composta de uma tripla (X,E,P) onde X e' uma chave, E e' o endereco do setor correspondente na area de dados e P e' um "pointer" para outra pagina, nulo nas folhas. K e' um numero natural que depende do tamanho da chave, tal que caibam no maximo 2K chaves em um setor com 512 bytes.

As entradas em uma pagina estao ordenadas por chave e cada "pointer" aponta para a sub-arvore com chaves menores. A direita da ultima entrada ha' um "pointer" para a sub-arvore com chaves maiores. A figura 1 mostra a organizacao logica de uma pagina.

P1	E1	X1	P2	E2	X2	.....	PL	EL	XL	PD	espaco vazio

figura 1 . Organizacao de uma Pagina

A figura 2 e' um exemplo de uma **ARVORE B T(3,2)**. Na figura os enderecos na area de dados E nao estao mostrados e os "pointers" P estao representados graficamente.

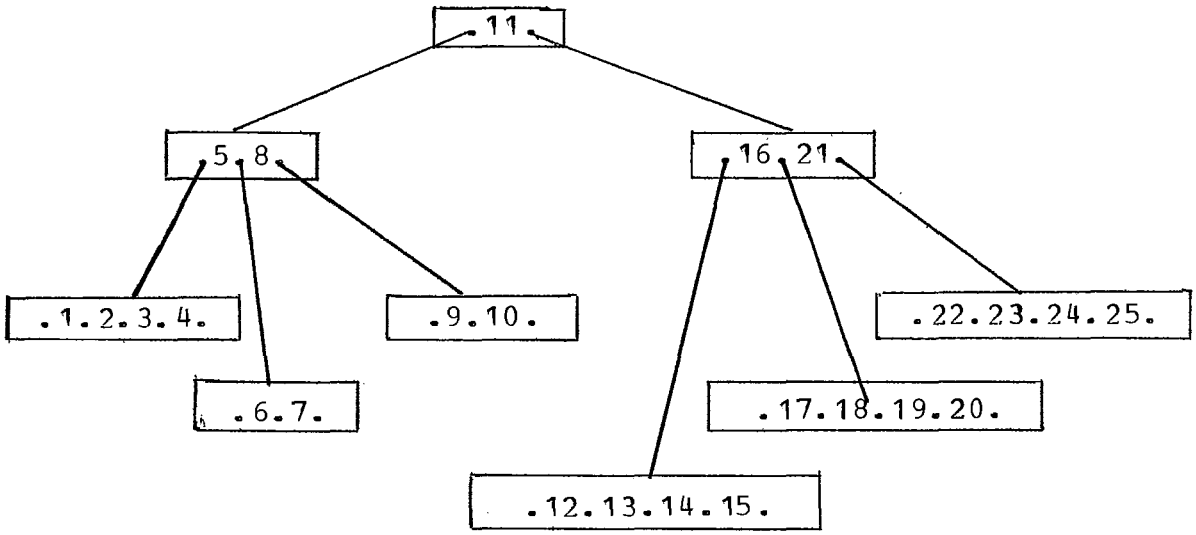

Figura 2. Indice com h=3 e k=2

III-1.3 Algoritmo de Busca

Nem todos os registros logicos tem entradas no indice, apenas o de chave maior em cada setor de dados. A recuperacao de um registro atraves do indice consiste em percorrer a arvore desde a raiz ate' encontrar a chave dada, ou a sua sucessora, caso nao seja encontrada. Em ambos os casos o setor de dados correspondente e' lido para a memoria e a chave e' procurada dentro do setor. O posicionamento no indice e na area de dados, assim como o caminho pela arvore, e' guardado para acelerar as operacoes seguintes.

A ARVORE B e' percorrida da seguinte maneira:

1- Inicia na raiz.

2- Percorre a pagina ate' encontrar uma entrada com chave maior ou igual 'a chave dada.

3- Se for igual passa para a etapa 5.

4- Desce na arvore pelo "pointer" correspondente ou pelo "pointer" 'a direita se nao encontrar uma entrada maior. Ao descer pela esquerda, guarda o endereco E da area de dados. Volta 'a etapa 2 se nao for uma folha.

5- Neste ponto o indice esta posicionado na chave procurada ou na sua sucessora e e' retornado o endereco E do setor de dados.

III-1.4 Algoritmo de Insercao

A insercao de uma chave e' feita sempre em uma folha. Na gravacao sequencial o posicionamento do indice estara' sempre na sucessora da chave a ser inserida. Na gravacao direta e' feita uma busca no indice da chave em questao, que o deixara' posicionado na entrada sucessora. Havendo espaco na pagina a insercao e' feita, deslocando-se as entradas com chaves maiores para a direita a fim de manter a ordenacao.

Se a pagina estiver cheia ocorre o "SPLIT" como esta descrito a seguir.

III-1.5 "Split"

A chave em questao e' inserida logicamente na memoria. A pagina e' dividida em duas, sendo a segunda parte regravada no mesmo setor. E' alocado um novo setor, nele sendo gravada a primeira parte da pagina original. A entrada do indice que estiver no ponto medio da pagina original sera' inserida na pagina pai e o seu "pointer" P apontara' para a nova pagina alocada. A insercao desta entrada na pagina pai pode provocar "split" nela tambem e assim sucessivamente ate' a raiz.

O "split" e' feito de forma recursiva, utilizando uma pilha que foi construida no processo de busca. A altura da arvore aumenta sempre que houver "split" na raiz.

Apos o processo de "split" o posicionamento no indice e' invalidado. A operacao seguinte devera' chamar a rotina de busca para refaze-lo.

III-1.6 Remocao

A remocao de uma entrada no indice ocorre com muito menos frequencia do que a remocao de um registro de dados, e so' ocorre quando um setor de dados desaparece.

Se o indice nao estiver posicionado na chave a ser removida e' feita uma busca nesta chave, que fara' o

posicionamento. A fim de manter a estrutura da arvore, a entrada e' removida se estiver em uma folha. Caso contrario sera' substituida pela sua sucessora, que e' removida da folha. Como consequencia de remocoes em uma folha, ela pode ficar com menos de K entradas. Neste caso efetuamos o processo de "CATENATION" ou "UNDERFLOW".

III-1.7 "Catenation"

Se a soma do numero de entradas em duas paginas adjacentes (paginas irmas) for menor do que 2K, elas podem se juntar para formar uma. Neste caso, a entrada de indice que esta' na pagina pai e que aponta para a pagina da esquerda, sera' removida e inserida na uniao das duas.

Este e' o processo inverso do "SPLIT", que ocorre na insercao, e tambem pode se propagar ate' a raiz, sendo esta a maneira pela qual a arvore diminui de altura.

III-1.8 "Underflow"

Se, apos uma remocao, a soma do numero de entradas em duas paginas adjacentes for maior ou igual a 2K, as entradas podem ser redistribuidas igualmente entre as duas, sendo o processo chamado de "UNDERFLOW". Este processo, ao contrario da "CATENATION", nao se propaga.

III-2 Dados

A area de dados contem os registros logicos do arquivo, organizados sequencialmente dentro de cada setor. Cada setor, alem do numero de bytes ocupados, contem um "pointer" para o seguinte, para que a leitura sequencial possa ser feita sem acesso ao indice. A chave de cada registro logico ocupa os primeiros bytes e seu tamanho e' especificado em tempo de alocacao do arquivo. Somente a maior chave do setor tem uma entrada no indice. Na gravacao sequencial pode ser deixado espaco livre em cada setor, para futuras insercoes que serao feitas sem alteracao no indice. Se este espaco nao for suficiente ocorre "split" no setor, com a insercao de uma nova entrada no indice.

Da mesma forma, as remocoes sao feitas sem alteracao no indice enquanto o setor contiver pelo menos a metade do seu espaco ocupado. Assim como no indice, pode ocorrer **"CATENATION"** ou **"UNDERFLOW"** nos setores de dados. Note-se que nem o **"SPLIT"** nem a **"CATENATION"** se propagam se propagam nos setores de dados.

III-3 Controle

O sistema de E/S do TI contem dois blocos de controle:

- Tabela Resolvida de Arquivos Logicos (TRAL)
- Tabela de Arquivos Fisicos (TAF)

A TRAL contem as informacoes referentes ao arquivo logico selecionando o arquivo fisico a ser afetado, sendo criada em tempo de compilacao. Ela tem um formato fixo e aponta para uma lista de TAFS. A cada TAF esta' associado um arquivo fisico. A lista de TAFS e' criada em tempo de carga do programa pelo carregador de programas (CPROG). A TAF tem um formato diferente para cada tipo de arquivo. No caso do metodo de acesso indexado a primeira TAF contem as seguintes informacoes:

endereco da proxima TAF
endereco do primeiro setor fisico
numero de setores
endereco da raiz do indice
tamanho da chave
tamanho do registro
ultima operacao executada
posicionamento no indice
posicionamento nos dados
maxima ocupacao para gravacao sequencial
endereco do "buffer" de E/S

As TAFS seguintes contem apenas o endereco da proxima TAF, o endereco do primeiro setor fisico e o numero de setores.

Como estas informacoes nao cabem em uma entrada no Diretorio do disco, elas sao gravadas no primeiro setor fisico do arquivo. A gravacao e' feita pelo utilitario de alocacao do arquivo, e e' atualizada pelo metodo de acesso atraves do comando **ENDFILE** . Este setor e' lido pelo **CARREGADOR DE PROGRAMAS** para formar a TAF.

IV- UTILIZACAO DO METODO DE ACESSO INDEXADO

O **METODO DE ACESSO INDEXADO** sera' usado em programas escritos em **PLTI**, atraves dos mesmos comandos usados para os outros metodos de acesso. Havera pequenas alteracoes em alguns, para que seja fornecida a chave do registro (palavra chave **KEY**) e tambem a inclusao do comando **DELETE**.

Programas que usam acesso sequencial podem substituir, em tempo de carga, um arquivo sequencial por um indexado.

IV-1 Regras Gerais de Utilizacao

Um arquivo indexado e' composto de tres partes basicas:

- bloco de controle
- indice
- dados

As duas primeiras sao totalmente transparentes ao usuario, sendo a sua criacao e manutencao feitas de forma automatica pelo sistema. Do ponto de vista do usuario o arquivo indexado e' uma sucessao de registros logicos, de tamanho fixo, que podem ser recuperados, inseridos ou removidos diretamente, atraves de uma chave, ou sequencialmente. A chave ocupa as primeiras posicoes do registro, e seu tamanho e' especificado em tempo de alocacao do arquivo.

O espaco liberado por remocoes e' reaproveitado automaticamente, e o crescimento do arquivo e' praticamente ilimitado, quase sem degradacao na performance . A necessidade de reorganizacao do arquivo e' minima.

As operacoes podem ser feitas em qualquer sequencia. O sistema mantem automaticamente um "pointer" para o posicionamento atual no arquivo. Assim, apos uma operacao com acesso direto, pode-se prosseguir sequencialmente, e vice-versa.

IV-2 Criacao do Arquivo

A criacao de um **ARQUIVO INDEXADO** e' feita pelo utilitario geral de alocacao de arquivos em disco, cujas funcoes serao extendidas. Devem ser especificados os seguintes parametros, pelo usuario:

tipo do arquivo - indexado
tamanho do registro logico
tamanho da chave
numero total de setores
numero de setores para o indice
maximo de utilizacao

Estas informacoes serao guardadas no primeiro setor do arquivo. O usuario devera, no seu programa, fornecer uma area de E/S no minimo igual ao tamanho do registro em todos os comandos que envolverem transmissao de dados.

Para calcular o numero de setores necessarios ao indice o usuario deve levar em conta que cada entrada tem um comprimento igual ao comprimento da chave somado a 4 bytes de "pointers". Ha' ainda mais 4 bytes de controle para cada setor, e uma entrada adicional, com chave ffff.... Marcando o fim do arquivo. Deve considerar tambem que somente o ultimo registro de cada setor de dados tem uma entrada no indice.

Os registros de indice sao alocados preferencialmente na area de indice reservada pelo usuario, a fim de reduzir o movimento do braco do disco, mas, se esta for insuficiente, setores da area de dados serao usados.

O usuario pode deixar espaco livre em todos os setores da area de dados, especificando em tempo de alocacao o parametro "maximo de utilizacao", que e' o numero do ultimo byte a ser ocupado na gravacao sequencial. Este espaco reduz a ocorrencia de "SPLIT" na area de dados.

Dos 512 bytes de cada setor, 508 sao utilizados para os registros de dados, que devem ser de tamanho fixo, como nos demais metodos de acesso do SOCO.

IV-3 Extensao do Arquivo

O esquema de E/S do TI nao permite o uso de arquivos "multivolume". Esta restricao e' superada pelo uso da concatenacao de arquivos, que e' feita atraves do CPROG.

O **METODO DE ACESSO INDEXADO** nao permite a concatenacao de arquivos criados independentemente, o que envolveria a manutencao de dois indices. Ao terminar o espaco alocado inicialmente, o arquivo pode ser extendido, no mesmo volume ou em outros, pela alocacao de novas areas (arquivos sequenciais comuns) e a sua concatenacao com o arquivo original em tempo de execucao. Como o sistema usa enderecamento relativo, tanto o arquivo original como as extensoes podem ser movidas para outros volumes ou outras posicoes.

IV-4 Comandos

Nesta secao descreveremos cada comando do **METODO DE ACESSO INDEXADO**. Os caracteres em negrito sao as palavras chave da linguagem e os demais sao variaveis. Os caracteres sublinhados sao opcionais.

IV-4.1 Read

READ arquivo INTO area KEY campo EOF label-1 ERROR label-2;

Nao sendo especificada a palavra chave KEY sera' feita uma leitura sequencial, sem pesquisa no indice. Caso contrario sera' feita uma leitura direta, com busca no indice da chave especificada em "campo".

Em ambos os casos, apos a execucao o arquivo fica posicionado no proximo registro.

Em caso de fim de arquivo o programa desvia para "label-1" e em caso de erro para "label-2".

IV-4.2 Write

WRITE arquivo FROM area KEY campo EOF label-1 ERROR label-2;

Se a palavra chave KEY nao for especificada e' feita uma gravacao sequencial no final do arquivo mesmo que o posicionamento atual seja outro. A chave do registro, que esta nas primeiras posicoes devera' ser maior do que a ultima chave ja' gravada. Cada setor sera' preenchido ate' a maxima utilizacao que foi especificada na alocacao do arquivo.

Na gravacao direta a chave em "campo", que deve ser igual aos primeiros bytes de "area", podendo ser o mesmo campo, determinara' a posicao de insercao do registro. Neste caso o setor e' expandido ate' o fim do seu espaco livre, ocorrendo "SPLIT" se nao houver mais espaco. O posicionamento e' colocado no sucessor do registro inserido.

Um programa que usou o comando WRITE nao deve terminar sem emitir o comando **ENDFILE** para atualizar as informacoes de controle no primeiro setor do arquivo.

Nao havendo mais espaco no arquivo o programa desvia para "label-1" e em caso de erro para "label-2".

IV-4.3 Rewrite

REWRITE arquivo FROM area EPPOP label-2;

Este comando reescreve o ultimo registro tratado com os dados que estao em "area". Se houver erro, o programa desvia para "label-2". A chave contida nas primeiras posicoes de area deve ser a mesma que a do ultimo registro.

IV-4.4 Delete

DELETE arquivo KEY campo EPPOR label-2;

Este comando e' usado para remover um registro logico do arquivo. O espaco e' liberado e pode ser reutilizado totalmente, de forma transparente ao usuario. Em caso de erro o programa desvia para "label-2".

IV-4.5 Rewind

REWIND arquivo;

Este comando posiciona o arquivo no inicio.

IV-4.6 Endfile

ENDFILE arquivo;

Este comando e' obrigatorio antes de terminar o programa, sempre que o arquivo for alterado.

Serve para atualizar as informacoes de controle no primeiro setor do arquivo, e para gravar o ultimo "buffer" na gravacao sequencial. Pode ser usado a qualquer momento para posicionar no fim do arquivo.

IV-5 Codigos de Retorno

000- Operacao realizada normalmente

001- Fim de arquivo

002 a 099- Erro na rotina de E/S fisica

100- Operacao invalida (SITI)

101- Gravacao sequencial fora de ordem ou duplicada (WRITES)

102- Sequencia invalida de operacoes (WRITES)

103- Chave nao encontrada (RETRVE)

104- Indice nao inicializado (RETRVE)

105- Insercao de chave duplicada no indice (INSERT)

106- Erro no indice (INSERT)

107- Nao ha mais espaco no arquivo (ALLOC)

108- Endereco logico nao encontrado (LOG_RW)

109- Registro nao encontrado (READD)

110- Rewrite cum chave diferente da anterior (REWRTE)

111- Insercao de registro duplicado (WRITED)

112- Registro nao encontrado (DELETE)

113- Erro no indice (WRITED) 114- Erro no indice (DELETE)

115- Erro no indice (REMOVE)

V- INTERFACES DO METODO DE ACESSO INDEXADO COM O SOCO.

Neste capitulo detalhamos as especificacoes e alteracoes para as partes do SOCO que interagem diretamente com o METODO DE ACESSO INDEXADO.

V-1 Carregador de Programas - CPROG

A funcao do CPROG e' carregar o programa a ser executado e as rotinas de E/S, na hora da execucao, resolvendo as referencias externas. Ele devera' criar as TAFS dos diversos arquivos necessarios e encadea-las 'a TRAL.

No METODO DE ACESSO INDEXADO as informacoes de controle sao colocadas na TAF do primeiro arquivo fisico. O CPROG obtera' estas informacoes no primeiro setor fisico do arquivo e as usara' para montar esta TAF. Esta tera' o tipo 19. As demais terao o tipo 20 e o mesmo formato das TAFS normais de disco.

A TAF tipo 19 devera' ter um desvio para a rotina SITI, que e' a rotina principal do metodo de acesso e se encarregara' de chamar as outras.

No presente momento as rotinas de E/S nao podem estar em "overlay". Se esta restricao for levantada, o metodo de acesso podera' ser usado de acordo com a seguinte estrutura:

Modulo principal - SITI

Segundo nivel - READS, READD, WRITES, WRITED, ENDFLE,
REWIND, DELETE

Terceiro nivel - PETPVE, INSERT, REMOVE

Quarto nivel - MOVE, COMP, LOG_RW

E' o seguinte o "layout" do primeiro setor:

endereco da raiz

tamanho da chave

primeiro setor de indice vazio

primeiro setor de dados vazio

tamanho do registro

num maximo de bytes ocupados

Todos estes campos sao numeros binarios com 2 bytes cada, e serao movidos para a TAF. Os demais campos deverao ser gerados. O "layout" da TAF esta' na listagem da rotina principal SITI.

V-2 Rotina de Alocacao em Disco

A rotina de alocacao em disco tera' um trabalho adicional ao alocar um arquivo indexado. Este trabalho consiste em formar o primeiro setor do arquivo, segundo o "layout" especificado na secao anterior, i criar uma chave "dummy" no segundo setor do arquivo, com ffff.... . As

informacoes necessarias serao fornecidas pelo usuario sob a forma de parametros, descritos no capitulo iv-2.

E' o seguinte o formato do setor com a chave "dummy":

CAMPO	TAMANHO	VALOR
N_BYTES	2	TAM_CH + 6
POINTEP ESQ	2	0
ENDERECO	2	0
CHAVE	TAM_CH	FFFF....
POINTER DIR	2	0

V-3 Compilador PLTI

As alteracoes no compilador sao pequenas, consistindo basicamente na inclusao da palavra chave opcional **KEY** nos comandos **READ** e **WRITE**, conforme especificado nas secoes iv-4.1 e iv-4.2, para indicar leitura e gravacao direta.

Nestes casos o compilador passara' os codigos de operacao 8 e 9 respectivamente, indicando leitura ou gravacao direta.

Sera' criado um comando adicional **(DELETE)** que o compilador devera' reconhecer, com o formato descrito na secao iv-4.4 e que devera' gerar o codigo de operacao 7.

A passagem de parametros do compilador para o metodo de acesso indexado e' feita atraves dos registradores da mesma forma que nos outros metodos de acesso.

V-4 Interpretador PLTI

O interpretador PLTI utiliza uma tabela (TRI) com as rotinas correspondentes 'as instrucoes usadas no programa. Instrucoes nao usadas nao tem entradas nesta tabela. Como o metodo de acesso indexado sera' carregado em tempo de execucao e a tabela de instrucoes e' unica, algumas das instrucoes que ele utiliza podem nao estar na tabela.

Neste caso temos duas alternativas:

1- o compilador gera um CALL para o metodo de acesso e este e' compilado junto com o programa do usuario. Neste caso as suas instrucoes seriam incluidas na tabela gerada pelo compilador, mas perderiamos a flexibilidade de substituir o metodo de acesso em tempo de execucao.

2- incluir na tabela do interpretador as entradas correspondentes a todas as instrucoes. Esta inclusao nao ira' aumentar em muito o tamanho da tabela, porque em um programa tipico em PLTI poucas sao as instrucoes nao utilizadas.

A segunda alternativa nos parece a melhor, uma vez que eliminara' uma forte restricao ao desenvolvimento de "software" basico em PLTI.

VI- DISCUSSAO E CONCLUSOES

Este capitulo discute alguns pontos do metodo de acesso indexado e apresenta o resultado de alguns testes feitos simulando o PLTI.

VI-1 Simulacao

A fim de obter uma previsao da performance do metodo de acesso foram feitas algumas simulacoes. Foi escolhido um tamanho de registro de 80 bytes e uma chave de 8 bytes. Estes valores sao tipicos para a operacao normal do Terminal Inteligente. O tempo de E/S foi computado, levando-se em conta o numero de operacoes de E/S, os "seeks" e o "rotational delay". O tempo de cpu nao foi considerado por ser muito menor do que o de E/S.

Foi simulado um programa do usuario que carrega e depois le sequencialmente 1000 registros. Em seguida os registros foram lidos diretamente em ordem inversa.

Depois foram incluidos mais 1000, dobrando o tamanho do arquivo. Ele foi entao lido sequencialmente e diretamente, em ordem inversa.

A tabela abaixo mostra, para cada caso, o numero de operacoes de E/S, o numero de "seeks" e o tempo medio por registro.

	Num de registros	oper de e/s	num de "seeks"	tempo med(ms)
gravacao sequencial	1000	522	332	17,3
leitura sequencial	1000	167	8	5,3
leitura direta	1000	3522	2333	117
insercao direta	1000	4929	2249	165
leitura seq (apos inser)	2000	329	320	5,7
leitura dir (apos inser)	2000	10924	6117	184

simulacao do metodo de acesso indexado

VI-2 Discussao

Acesso Sequencial

A simulacao do metodo de acesso mostrou alguns pontos importantes. A eficiencia da leitura sequencial e' praticamente a mesma de um metodo de acesso sequencial. Isto ocorre porque cada bloco contem um "pointer" para o proximo, evitando assim pesquisa no indice. Na gravacao sequencial, o tempo e' um pouco maior porque o indice esta' sendo construido. Ainda assim, a caracteristica nao densa do

indice aumenta em muito a eficiencia, em relacao ao processamento direto, porque so' gravamos um registro de indice por bloco.

Podemos tambem observar que houve pouca degradacao na performance, mesmo apos a duplicacao do tamanho do arquivo. O numero de operacoes de "SEEK" e de E/S aumentou pouco. Esta simulacao foi feita sem deixar espaco livre nos setores para inclusoes e da forma mais desfavoravel, isto e' : Insercoes continuas ao longo de todo o arquivo. Se a percentagem de inclusoes puder ser prevista antes da carga, podemos deixar espaco livre nos setores, reduzindo o tempo de insercao e de leitura. Se as insercoes estiverem ordenadas e em grupos, o milisegundos, permitem o uso do metodo de acesso em

Acesso Direto

A tabela da simulacao, vista na secao anterior mostra o desempenho do acesso direto. Os tempos medios por operacao, da ordem de milisegundos, permite o uso do metodo de acesso em aplicacoes interativas, como se espera de um terminal inteligente. Nesta simulacao com 2000 registros foi criada uma arvore com altura 3, sendo necessarias no maximo tres operacoes de E/S por acesso ao indice. Na operacao normal do sistema, somente no caso de chaves muito grandes ou arquivos muito extensos serao criadas arvores de altura maior do que

3. Mais de 20000 chaves com tamanho de 4 bytes podem ser armazenados em um indice de altura 3.

Pode-se observar que tambem no acesso direto nao houve degradacao significativa na performance quando o tamanho de arquivo foi duplicado. A separacao de uma area para o indice e' responsavel por uma grande reducao no tempo de "SEEK".

Devido 'a limitacao de memoria, nao estamos deixando nenhuma pagina no "buffer". Se isto fosse possivel, (ha' planos para aumentar a memoria do TI) poderiamos manter a raiz e talvez o segundo nivel do indice na memoria, o que iria acelerar ainda mais o acesso direto. O mesmo pode ser feito em relacao 'a area de dados para acelerar o acesso sequencial. Seria necessaria uma pequena alteracao (ja' prevista) na rotina LOG_RW de leitura logica, que nao faria a leitura fisica se o setor ja' estivesse na memoria.

Embora haja uma padronizacao de registros de tamanho fixo no TI ,este metodo de acesso pode ser usado com pequenas alteracoes com registros de tamanho variavel. Os algoritmos funcionam da mesma forma.

Para converter as rotinas de PL/I para PLTI sao necessarias apenas as seguintes alteracoes:

1- Substituir nas declaracoes as variaveis "FIXED BINARY" e "POINTER" por "ADDRESS"

2- Substituir nas declaracoes "CHAR" por "BYTE"

3- Retirar parenteses dos comandos de e/s

4- Substituir a funcao "ADDR" pelo operador " ".

VI-3 Conclusao

O metodo de acesso indexado foi todo programado e a sua execucao simulada usando o compilador PL/I. A simulacao mostrou a viabilidade da sua utilizacao no TI e a performance esperada. Os objetivos propostos na secao II-1 foram totalmente atingidos.

BIBLIOGRAFIA

1- R. Bayer e E. MCcreight, "Organization and Maintenance of Large Ordered Indexes", Acta Informatica 1, (1973).

2- R. E. Wagner, "Indexing Design Considerations", IBM Systems Journal, 12, 4 (1974).

3- D. G. Keehn e J. O. Lacy, "VSAM Data Set Design Parameters", IBM Systems Journal, 13, 3 (1974).

4- C. J. Date, "An Introduction to Data Base Systems", ADDISON-WESLEY (1975).

5- D. Lefkovitz, "File Structures for On-line Systems", SPARTAN BOOKS (1969).

6- D. E. Knuth, "The Art of Computer Programming - Vol 3 / Sorting and Searching", ADDISON WESLEY (1973).

7- IBM System/360 Operating System "Indexed Sequential Access Method", IBM Corporation, Data Processing Division, White Plains, New York.

8- OS/VS "Virtual Storage Access Method (VSAM) Planning Guide, IBM Corporation, Data Processing Division, White Plains, New York.

APENDICE A - DESCRICAO DAS ROTINAS

Este apendice apresenta uma descricao detalhada de todas as rotinas do metodo de acesso, com o objetivo de permitir futuras alteracoes ou correcoes.

As rotinas sao todas externas entre si. A comunicacao e' feita exclusivamente atraves de um bloco de controle comum (TAF) e de parametros passados explicitamente.

Ha' um "buffer" comum a todas as rotinas, que faz parte da rotina principal SITI, com 1024 bytes, dividido em duas partes de 512 bytes chamados de BUF1 e BUF2. Nele sao feitas todas as operacoes de E/S.

E' tomado um cuidado especial para que uma rotina nao apague dados nos "buffers" necessarios a outra.

MOVE - Movimentacao de Campos

Esta e' uma rotina auxiliar no sistema, usada em diversos modulos. Seu objetivo e' mover um campo de tamanho variavel.

Recebe como parametros um "pointer" para o campo a ser movido, um para a posicao final, e o numero de bytes a ser movido. Se este numero for negativo o deslocamento se

processa da direita para a esquerda, o que e' util ao se deslocar um campo sobre si mesmo.

COMP - Comparacao de Campos

Esta rotina se destina a comparar dois campos, com tamanho variavel, e e' necessaria porque o PLTI so admite comparacoes com 1 ou 2 bytes.

Recebe como parametros "pointers" para os dois campos e o tamanho da comparacao e retorna os seguintes valores:

1 se o primeiro campo e' maior
-1 se o primeiro campo e' menor
0 se sao iguais.

SITI - Rotina Principal

Esta rotina recebe o controle do programa do usuario e os seguintes parametros, nos registradores:

REGA - codigo de retorno
REGB - numero de ordem da TRAL
REGC - operacao a ser executada
REGDE - endereco da area de E/S

Inicialmente ela chama a rotina **TRATAF**, que coloca na posicao de memoria 70 os enderecos da **TRAL** e **TAF**.

Em seguida, de acordo com a operacao pedida ela chama a rotina apropriada.

O "buffer" de E/S esta' nesta rotina e para permitir o seu uso pelas outras rotinas ela coloca o seu endereco na **TAF**.

O "interface" com o **PLTI** e' feito atraves desta rotina, unico ponto a ser alterado se este interface for modificado. Ela passa para as demais rotinas o endereco da area de E/S do usuario e recebe um codigo de retorno, que e' devolvido ao programa do usuario no registrador A.

READS - Leitura Sequencial

Esta e' a rotina de leitura sequencial. Sua primeira providencia e' testar o codigo da operacao anterior. Se for **WRITES** ou **ENDFLE**, retorna imediatamente indicando fim de arquivo. Se for 0, esta e' a primeira operacao. Neste caso testa se o arquivo esta' vazio, retornando fim de arquivo em caso afirmativo. Caso contrario desvia para o procedimento de leitura de um novo setor.

Se a operacao anterior for READS, ou READD, o arquivo esta' posicionado no proximo registro a ser lido, que sera' movido para a area do usuario. Se o posicionamento for maior do que o numero de bytes ocupados (N_BYTES), e' lido um novo setor cujo endereco esta' em PROX_SET e o procedimento anterior e' executado. Apos mover o registro para a area do usuario o posicionamento e' atualizado e o codigo de retorno e' zerado.

O fim de arquivo e' detectado quando o campo PROX_SET estiver zerado.

READD - Leitura Direta

Se a operacao anterior foi gravacao sequencial e' chamada a rotina **ENDFLE**. Depois e' chamada a rotina de busca no indice **RETRVE** que retorna o endereco do setor de dados da chave procurada ou da sucessora.

O setor e' lido para o primeiro "buffer" e varrido 'a procura da chave. Se for encontrada, o registro e' movido para a area de E/S do usuario. Caso contrario retorna registro nao encontrado - RC = 109.

O posicionamento e' colocado no proximo registro.

WRITES - Gravacao Sequencial

Se a operacao anterior nao foi uma gravacao sequencial o posicionamento nao esta' no fim do arquivo. Entao e' procurada a chave FFFF...., que posiciona no fim. Se o arquivo esta' vazio e' alocado um setor de dados.

Se a chave do registro for menor ou igual 'a anterior e' retornado o codigo 101 que indica chave duplicada ou fora de sequencia. Se o espaco for suficiente, o registro e' incluido no "buffer", retornando codigo 0.

Se o novo registro ultrapassar a ocupacao maxima permitida, ele nao e' inserido. Um novo setor e' alocado, o "buffer" e' gravado e a chave do ultimo registro e' incluida no indice. Este setor contem um "pointer" para o novo setor alocado.

Entao o novo registro e' inserido no "buffer" e retorna.

ENDFLE - Fim de Arquivo

Se a operacao anterior for um WRITES e' gravado o ultimo "buffer". A chave "dummy" FFFF.... E' procurada e seu endereco de dados passa a apontar para o ultimo setor alocado.

Depois as informacoes do setor 0 sao atualizadas com os dados atuais da **TAF.**

REWRITE - Rotina de Atualizacao

Esta rotina compara a chave da area do usuario com a do ultimo registro. Se for igual, o registro do usuario e' movido para o "buffer" e este e' regravado. Caso contrario retorna o codigo 110.

REWIND - Posiciona no Inicio

Se a ultima operacao for WRITES e' chamada a rotina de **ENDFILE.** Depois o posicionamento de setor e' colocado no inicio, isto e': Primeiro setor de dados e primeiro byte do setor.

ALLOC - Alocacao de Setores

Se o tipo da alocacao for 1 e o campo AVAIL_IND for menor do que o inicio dos dados este sera' o novo setor alocado. AVAIL_IND e' entao incrementado de 1.

Se o tipo for 2 ou nao houver mais setores na area de indice o proximo setor logico sera' o conteudo do campo AVAIL-DAD este numero e' comparado com o numero de setores, que esta' na primeira TAF. Se for menor o setor esta alocado. Se nao o numero logico do setor e' decrementado do numero de setores nesta TAF e e' testado com o numero de setores da proxima TAF. Isto e' feito com as TAFS sucessivas ate' encontrar um setor disponivel ou ate' retornar a TAF original. Neste caso retorna o codigo 107, que indica que nao ha' mais espaco.

LOG_RW - E/S Logica

Esta rotina recebe um endereco logico e o transforma em fisico, varrendo as TAFS. Para cada TAF percorrida ele e' decrementado do numero de setores desta TAF. Quando o setor logico resultante for menor do que o numero de setores de uma TAF ele e' somado ao endereco do primeiro setor fisico, que contem tambem o endereco do DRIVE.

Entao e' chamada a rotina SDISK que faz a operacao de E/S fisica.

Se o tipo for 2 temos um WRITE. Do contrario e' READ. Tipo negativo indica E/S do segundo "buffer".

WRITED - Gravacao Direta

A rotina RETRVE e' chamada para procurar a chave. O setor de dados correspondente e' lido para "buffer" e e' varrido ate' encontrar uma chave maior ou ate' chegar ao seu fim. Se a chave for encontrada retorna registro ja' existente (rc = 111).

O registro e' inserido no "buffer" e gravado se o espaco for suficiente.

Se o espaco nao for suficiente, ocorre o "SPLIT". E' alocado um novo setor e seu endereco e' colocado em PROX-SET. O "buffer" e' dividido ao meio. A primeira parte e' regravada no mesmo setor e a segunda no novo setor alocado. A entrada de indice correspondente ao setor original e' alterada e passa a apontar para o novo. A ultima chave do novo setor e' incluida no indice.

DELETE - Remocao de Registro

Esta rotina inicialmente chama a rotina RETRVE para procurar no indice a chave do registro a ser removido. E' lido o setor de dados correspondente. O registro e' procurado no setor, acusando erro (rc = 112) se nao encontrar. Se encontrar, o registro e' removido, na memoria. Se este for o ultimo registro do setor, a entrada de indice correspondente tem a chave substituida pela do antecessor.

Se, apos a remocao, o setor estiver com mais da metade dos bytes ocupados ele e' regravado e o controle e' retornado. Do contrario o setor seguinte e' lido em buf2. Se a soma dos bytes ocupados no primeiro e segundo setores for menor do que 509 bytes e' efetuado o processo de "catenation" e se for menor e' feito um "underflow".

"Catenation"

Os registros de BUF2 sao movidos para BUF1, o valor de N_BYTES e' atualizado e a pagina e' regravada no primeiro setor. A entrada de indice correspondente ao segundo setor e' alterada, passando a apontar para o primeiro. Depois e' chamada a rotina REMOVE para remover a entrada do indice correspondente ao primeiro settor.

"Underflow"

Os registros de BUF1 e BUF2 sao divididos igualmente entre os dois e regravados. A entrada de indice correspondente ao primeiro setor tem a chave substituida pela do ultimo registro deste setor.

RETRVE - Busca no Indice

Esta rotina consta de dois "loops". O externo varre a arvore e o interno varre a pagina.

O "loop" externo inicia na raiz e termina numa folha, quando o "pointer" para a proxima pagina for nulo.

O "loop" interno varre a pagina ate' encontrar uma chave maior ou igual 'a procurada. Se for igual e' retornado o endereco da area de dados e rc=0. Quando e' encontrada uma chave maior o controle volta ao "loop" externo, que desce na arvore, ate' chegar 'as folhas. Quando a descida e' pela esquerda, o campo END_SET recebe o endereco da area de dados correspondente, e quando e' pela direita permanece igual. Assim, quando a pesquisa termina 'a direita de uma folha, o ender de dados retornado e' ultimo da pagina pai. Se, ao chegar 'a folha a chave nao foi encontrada e' retornado rc=103 e o posicionamento fica na entrada sucessora da chave procurada.

INSERT - Insercao no Indice

Se o posicionamento no indice estiver valido e' lido o setor correspondente para BUF1. Se nao, e' chamada a rotina RETRVE, que procura a chave no indice e acerta o posicionamento.

A seguir a entrada e' inserida, com o deslocamento das demais. E' colocado o endereco da area de dados e o "pointer" correspondente a esta entrada, que e' nulo no caso de ser uma folha.

Se o numero de bytes ocupados (n_bytes) for menor do que 508 o setor e' regravado e o posicionamento e' colocado na entrada sucessora da que foi inserida.

Nao havendo espaco e' efetuado o procedimento de "split", de forma recursiva.

SPLIT

E' alocado um novo setor de indice. A entrada de indice que ocupa a posicao media e' encontrada. As entradas 'a sua direita sao deslocadas para o BUF2. O numero de bytes ocupados em BUF1 e' acertado ele e' regravado no setor original.

O numero de bytes em BUF1 e' acertado e ele e' gravado no novo setor alocado.

A seguir a entrada do meio e' salva no final do segundo "buffer". A pagina pai da que sofreu "split" e' obtida atraves da pilha que contem o percurso atraves da arvore.

Se o "split" foi na raiz, e' alocado um setor para a nova raiz, nele sendo incluido apenas o "pointer" para o filho 'a direita. Entao e' feito um desvio para o procedimento normal de insercao de uma entrada, que ira' incluir a entrada na raiz e grava-la.

Se o "split" foi no meio a pagina PAI lida e' lida para buf1 e e' varrida ate' encontrar a posicao de insercao da nova entrada. Neste ponto ha' um desvio para o procedimento de insercao de entrada.

Esta nova insercao pode provocar um novo "split" e se propagar ate' a raiz. O procedimento e' repetido recursivamente.

Apos um "split" o posicionamento e' invalidado colocando-se o topo da pilha igual a zero. A chave do ultimo registro da segunda parte e' incluida no indice.

REMOVE - Remocao de Entrada no Indice

POSICIONAMENTO

Inicialmente e' chamada a rotina RETRVE para procurar a entrada a ser removida. Se ela nao estiver em uma folha e' substituida pela sua sucessora, a qual sera' entao removida.

REMOCAO

A entrada e' removida da folha, deslocando-se as demais para a esquerda. Se apos a remocao a folha ficar com menos da metade ocupada sera' feito o procedimento de "underflow" ou "catenation". Se nao, o controle e' retornado (rc=0). O irmao da esquerda e da direita sao obtidos, atraves da pagina pai, sendo feito o procedimento de "catenation" se a soma dos espacos das duas paginas for menor ou igual a 508 bytes e "underflow" se for maior.

"CATENATION"

As duas paginas sao colocadas em BUF1 e BUF2. As entradas da pagina em BUF2 sao deslocadas para BUF1, deixando a entrada pai no meio. O numero de bytes ocupados e' atualizado e buf1 e' regravado no setor da pagina da direita. O controle entao e' passado para a etapa de remocao de entrada, para remover a entrada pai. Esta remocao pode

provocar um novo "catenation" ou "underflow" na pagina pai, e se propagar de forma recursiva ate' a raiz.

Ao se remover a ultima entrada da raiz por um processo de "catenation", a arvore tera' sua altura diminuida.

"UNDERFLOW"

As duas paginas sao colocadas em BUF1 e BUF2, com a entrada pai no meio, e as entradas sao distribuidas igualmente entre as duas. O numero de bytes ocupados e' atualizado e ambas sao regravadas nos seus setores originais.

A seguir a entrada pai e' substituida pela que ficar na ultima posicao da pagina da esquerda, e esta e' removida. O posicionamento no indice e' invalidado.

APENDICE B - LISTAGEM DAS ROTINAS

```
CPROG:   PROC (INPARM)   OPTIONS (MAIN); /* SIMULADOR DO
CPROG       */

  DECLARE    INPARM     CHAR(100) VAR,
             COMANDO    CHAR(100),
           1 BAS_COM    UNAL BASED(P_CMD),
           2 INIT       CHAR(1),
           2 LOGICO     PIC'99',          /*  ARQUIVO   LOGICO     */
           2 NUM_ARQ    PIC'99',          /*  NUM DE ARQS FISICOS  */
           2 FISICOS(10),
           3 PRIM_SET   PIC'999',         /*  PRIMEIRO  SETOR      */
           3 N_SET      PIC'999';         /*  NUM  DE  SETORES     */

  DECLARE    VLP_70  POINTER EXTERNAL STATIC; /*  VALOR 70          */

  DECLARE  1 TPTA       BASED(VLP_70), /*  REGIAO FIXA - ENDER 70   */
           2 P_TRAL          POINTER,  /* POINTER DA TRAL           */
           2 P_TAF           POINTER;  /* POINTER DA TAF            */

  DECLARE  1 SETOR_0    UNAL,          /* SETOR ZERO                */
           2 RAIZ       FIXED BIN(15), /* SETOR INICIO DO INDICE    */
           2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE          */
           2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE   */
           2 S_IN_DAD   FIXED BIN(15), /* SETOR INICIO DADOS        */
           2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS   */
           2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTRO       */
           2 MAX_UT     FIXED BIN(15), /* MAX BYTE UTILIZAVEL       */
           2 RESTO      CHAR(498);     /* RESTO                     */

  DECLARE  1 TAF        BASED(P) UNAL, /* TABELA DE ARQUIVOS        */
                                       /*       FISICOS             */
           2 JMP_ES(3)  CHAR  ,        /* JUMP PARA ROTINA DE E/S   */
           2 PROX_TAF   POINTER,       /* POINTER PARA PROXIMA TAF  */
           2 TYPE        BIT(8),       /*  TIPO DA TAF = 19         */
           2 S_IN_FIS   FIXED BIN(15), /* SET DE IN FISICO-COM DRIVE */
           2 NUM_SET    FIXED BIN(15), /* NUM DE SETORES            */
           2 STATUS     CHAR  ,        /* STATUS                    */
           2 RAIZ       FIXED BIN(15), /* SETOR INICIO DO INDICE    */
           2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE          */
           2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE   */
           2 TOPO       FIXED BIN(15), /* TOPO DA PILHA DO INDICE   */
           2 PILHA(10)  FIXED BIN(15), /* PILHA - CAMINHO NO INDICE */
           2 POS_REG_I  FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
           2 S_IN_DAD   FIXED BIN(15), /* SETOR INICIO DADOS        */
           2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS   */
           2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTRO       */
```

```
          2 POS_SET_D FIXED BIN(15), /* POSICION  - SETOR -DADOS   */
          2 POS_REG_D FIXED BIN(15), /* POSICION  - REG   -DADOS   */
          2 OPEP_ANT  FIXED BIN(15), /* ULTIMA OPERACAO  FEITA     */
          2 P_BUF     POINTER,       /* END DO BUFFER GERAL        */
          2 MAX_UT    FIXED BIN(15); /* MAX BYTE UTILIZAVEL        */

DECLARE   1 TRAL      BASED(0) UNAL,     /* TABELA RESOLVIDA DE    */
                                         /*      ARQUIVOS  LOGICOS */
          2 LBI       FIXED BIN(15),  /* NUM DE ORDEM DA TRAL      */
          2 JMP_TAF(3) CHAR  ,        /* JUMP  PARA TAF            */
          2 RECSIZE   FIXED BIN(15), /* TAMANHO DE REGISTRO        */
          2 RES       CHAR  ;        /* RESERVADO                  */

DECLARE     TAF_EXT   CHAR(13) BASED (P_TAF_EXT), /* TAF DE EXT   */
            END_SET    FIXED BIN(15); /* ENDERECO DO SETOR         */

DECLARE   1 BUF1(512)  CHAR;          /*    BUFFER DE E/S          */

DECLARE     APQ01 FILE RECORD ENV(REGIONAL(1) F RECSIZE(512));
DECLARE     TP(5) BIT(1) EXTERNAL;
DCL         HEX    EXTERNAL ENTRY;   /* IMPRESSAO EM HEXA          */
DECLARE      PGMUS      EXTERNAL  ENTRY;
DECLARE     I FIXED BIN(15);
DECLARE    (P,P_CMD,P_TAF_EXT,Q,P_SO) POINTER,
           N_BYTES   FIXED BIN(15) BASED(P_BUF),
           KEY PIC '99999999';
           ON ERROR  GOTO  FINAL;
           TR='0'B;
```

```
      /*   I N I C I O   D O   C P P O G          */

COMANDO=INPARM;
P_CMD=ADDP(COMANDO);
IF LENGTH(INPARM) < 2          /* DEFAULT DO COMANDO          */
THEN DO;
     INIT='N';
     LOGICO,NUM_APQ=1;
     PRIM_SET(1)=0;
     N_SET(1)=100;
     END;

ALLOC TRTA;                    /* APEA  DE POINTERS TAF-TRAL */

ALLOC TRAL;                    /* ALOCA  TRAL                 */
P_TRAL=Q;                      /* CARREGA POINTER             */
LBI=LOGICO;                    /* PPEENCHE TRAL               */

ALLOC TAF;                     /* ALOCA PRIM  TAF             */
P_TAF=P;                       /* CARREGA  POINTER            */
P_BUF=ADDR(BUF1);              /* COLOCA ENDER NA TAF         */

/* INICIALIZACAO DO APQUIVO  */

IF INIT ¬= 'S'                 /* APQUIVO NAO INICIALIZADO    */
THEN DO;
     OPEN FILE (APQ01) DIPECT OUTPUT;
     SETOP_0.PAIZ=1;           /*      FORMATA                */
     SETOR_0.TAM_CH=116;
     SETOR_0.AVAIL_IND=2;      /*      SETOR                  */
     SETOR_0.S_IN_DAD=25;
     SETOR_0.AVAIL_DAD=25;
     SETOP_0.TAM_REG=120;      /*       ZERO                  */
     SETOR_0.MAX_UT=400;
     PUT SKIP DATA(SETOP_0); GET DATA;
     KEY=PRIM_SET(1);
     WRITE   FILE(ARQ01) FROM(SETOR_0) KEYFROM(KEY);

     DO I=1 TO 512;       /*  ZERA   O   BUFFER              */
     UNSPEC(BUF1(I))='0'B;
     END;
     DO I=7 TO 6+SETOR_0.TAM_CH; /* INCLUI CHAVE FFFF...     */
     UNSPEC(BUF1(I))='11111111'B;
     END;
     N_BYTES=SETOR_0.TAM_CH+6;        /*NUM DE BYTES OCUP    */
     KEY=PRIM_SET(1) + 1;
     WRITE FILE(APQ01) FROM(BUF1) KEYFROM(KEY);
     CLOSE FILE(ARQ01);
     END;
```

```
      OPEN FILE(ARQ01) DIRECT UPDATE;
      KEY=PRIM_SET(1);
      READ FILE(ARQ01) INTO(SETOR_0) KEY(KEY);   /*   LE SETOR ZERO*/
      TAF=SETOR_0 , BY NAME;                     /* PREENCHE PRIMEIRA TAF      */
      TAF.TYPE='00010011'B;                      /* TIPO 19 - SITI             */
      TAF.S_IN_FIS=PRIM_SET(1);                  /*    INCLUI   INFORMACOES    */
      TAF.NUM_SET=N_SET(1);                      /*          FISICAS           */
      TOPO,POS_REG_I,POS_SET_D, POS_REG_D,OPER_ANT=0;
      PILHA=0;
      IF NUM_ARQ=1 THEN DO;                      /*    UM ARQUIVO APENAS       */
                        TAF.PROX_TAF=P_TAF;
            IF TP(1) THEN PUT SKIP(3) DATA(TAF);
            II=61; IF TR(1) THEN CALL HEX(P_TAF,II);
                        GOTO CARGA;
                        END;

      DO  I= 2   TO   NUM_ARQ;                   /* CRIA OUTRAS TAFS       */
      ALLOC TAF_EXT SET (P_TAF_EXT);             /* ALOCA TAF DE EXTENSAO  */
      TAF.PROX_TAF=P_TAF_EXT;                    /* COLOCA ENDEPECO DESTA  */
                                                 /* TAF  NA TAF ANTERIOR   */
      P=P_TAF_EXT;                               /* POSICIONA MASC NA TAF  */
                                                 /*  DE EXTENSAO           */
      TAF.TYPE='00010100'B;      /* TIPO 20  -  SITI EXT                   */
      TAF.S_IN_FIS=PRIM_SET(I); /*  MOVE INICIO E TAMANHO                 */
      TAF.NUM_SET=N_SET(I);      /*     DO    ARQUIVO                      */
      IF TR(1) THEN PUT SKIP(2) DATA(TAF);
      II=13; IF TR(1) THEN CALL HEX(P_TAF_EXT,II);
      END;
      TAF.PROX_TAF=P_TAF;        /* FECHA  LISTA  CIRCULAR                 */


 CARGA:  CALL  PGMUS;                        /* CHAMA PROG DO USUARIO      */

 FINAL:                PUT PAGE DATA(TAF);
       REVERT ERROR;
     II=1024; IF TP(1) THEN CALL HEX(P_BUF,II);
       P_S0=ADDR(BUF1);
       DO I=0 TO N_SET(1) - 1;
       KEY=I; READ FILE(ARQ01) INTO(BUF1) KEY(KEY);
       PUT  SKIP(2) DATA(KEY);
       II=512;  CALL HEX(P_S0,II);
       END;
END CPROG;
```

```
PGMUS:  PPOC;          /*  PROGRAMA  DO  USUARIO     */

   DECLARE  1 REGS   UNAL STATIC EXTERNAL,/* REGISTRADORES GERAIS    */
            2 REGA       BIT(8),         /*ACUMULADOR  -  COD RETORNO */
            2 REGB       BIT(8),         /* NUM DE ORDEM DA TRAL      */
            2 REGC       BIT(8),         /* FUNCAO A SER EXECUTADA    */
            2 REGDE      POINTER;        /* ENDER DA AREA DE E/S      */

   DCL        IOAREA(300) CHAR,
              CHAVE  PIC'99999999' DEF IOAREA,
              BYTE FIXED BIN(8),
              TR(5)  BIT(1) EXTERNAL,
              NUM_REGS FIXED BIN(15) INIT(100),
              (NUM_ES,CIL_ANT,TEMPO,SEEKS)
              FIXED BIN(31)  EXTERNAL STATIC,
              SITI EXTERNAL ENTRY;

       PUT SKIP DATA(NUM_REGS,TR); GET DATA;


       REGB='00000001'B;                /* IDENT DA TRAL             */
       REGDE=ADDR(IOAREA);
       DO I=1 TO 255;
       BYTE=I; UNSPEC(IOAREA(I))=BYTE;
       END;
       NUM_ES,CIL_ANT,TEMPO,SEEKS=0;
       ON ENDFILE(SYSIN) GOTO FIM;
       IF NUM_REGS=0 THEN GOTO ONL;
```

```
 BYTE=2; REGC=BYTE;                /* WRITES                          */
 PUT PAGE LIST ('GRAVACAO SEQUENCIAL');
 PUT SKIP(2);
 DO  I=1 TO 2 * NUM_REGS BY 2;  /* INCLUI NUM_REGS REGISTROS  */
 CHAVE=I;                       /*  IMPARES SEQUENCIALMENTE   */
 CALL SITI;
 IF PEGA ¬= '0'B  THEN PUT SKIP EDIT(' **PC=',REGA) (A,F(4));
 PUT EDIT (CHAVE) (F(5));
 END;
 PUT SKIP(5) DATA(NUM_ES,CIL_ANT,TEMPO,SEEKS);

BYTE=6; REGC=BYTE;                 /*  ENDFILE                        */
CALL SITI;

 BYTE=9; REGC=BYTE;                /* WRITED                          */
 PUT PAGE LIST ('GRAVACAO DIRETA    ');
 PUT SKIP(2);
 NUM_ES,CIL_ANT,TEMPO,SEEKS=0;
 DO  J=2  TO 2*NUM_REGS BY 2;   /* INCLUI NUM_REGS  REGISTROS */
 CHAVE=J;                      /*   PARES   DIRETAMENTE        */
 CALL SITI;
 IF REGA ¬= '0'B  THEN PUT SKIP EDIT(' **PC=',REGA) (A,F(4));
 PUT EDIT (CHAVE) (F(5));
 END;
 PUT SKIP(5) DATA(NUM_ES,CIL_ANT,TEMPO,SEEKS);

 NUM_ES,CIL_ANT,TEMPO,SEEKS=0;
 BYTE=8; REGC=BYTE;                 /* READD                          */
 PUT PAGE LIST('LEITURA DIRETA'); PUT SKIP(2);
 DO I= 2*NUM_REGS TO 1 BY -1;  /*  LE  2 * NUM_REGS REGISTROS */
 CHAVE=I;                      /*           DIRETAMENTE       */
 CALL SITI;
 IF PEGA ¬= '0'B  THEN PUT SKIP EDIT(' **PC=',REGA) (A,F(4));
 PUT EDIT (CHAVE) (F(5));
 END;
 PUT SKIP(5) DATA(NUM_ES,CIL_ANT,TEMPO,SEEKS);
 NUM_ES,CIL_ANT,TEMPO,SEEKS=0;

 PUT PAGE LIST('LEITURA SEQUENCIAL'); PUT SKIP(2);
 BYTE=5; REGC=BYTE;                 /*  REWIND     */
 CALL SITI;

 BYTE=1; REGC=BYTE;                 /*  READS      */
 DO WHILE (REGA='0'B);
 CALL SITI;
 PUT EDIT (CHAVE) (F(5));
 END;
 PUT SKIP(5) DATA(NUM_ES,CIL_ANT,TEMPO,SEEKS);
 NUM_ES,CIL_ANT,TEMPO,SEEKS=0;
```

```
        BYTE=7;  REGC=BYTE;              /* DELETE                          */
        PUT PAGE LIST ('REMOCAO DIRETA     ');
          GET DATA;
        PUT SKIP(2);
        NUM_ES,CIL_ANT,TEMPO,SEEKS=0;
        DO  J=1  TO 2*NUM_REGS BY 2;    /* REMOVE NUM_REGS  REGISTROS */
        CHAVE=J;                        /*  IMPARES  DIRETAMENTE       */
        CALL SITI;
        PUT EDIT (CHAVE) (F(5));
        IF REGA ¬= '0'B  THEN GOTO FIM;
        END;
        PUT SKIP(5)  DATA(NUM_ES,CIL_ANT,TEMPO,SEEKS);

        BYTE=8; REGC=BYTE;               /* READD                           */
        DO I= 1 TO 2*NUM_REGS;          /*  LE 2*NUM_REGS REGISTROS   */
        CHAVE=I;                        /*  DIRETAMENTE               */
        CALL SITI;
        IF REGA ¬= '0'B  THEN PUT SKIP EDIT(' **PC=',REGA)  (A,F(4));
        PUT EDIT (CHAVE) (F(5));
        END;
        PUT SKIP(5)  DATA(NUM_ES,CIL_ANT,TEMPO,SEEKS);
        NUM_ES,CIL_ANT,TEMPO,SEEKS=0;

        PUT PAGE LIST('LEITURA SEQUENCIAL'); PUT SKIP(2);
        BYTE=5; REGC=BYTE;               /*  REWIND     */
        CALL SITI;

        BYTE=1; REGC=BYTE;               /*  READS      */
        DO WHILE (REGA='0'B);
        CALL SITI;
        PUT EDIT (CHAVE) (F(5));
        END;
        PUT SKIP(5)  DATA(NUM_ES,CIL_ANT,TEMPO,SEEKS);

ONL:      PUT SKIP(2)  LIST('ENTRADA ONLINE');
          PUT SKIP DATA(TR); GET DATA;


LOOP:   GET LIST(CHAVE,BYTE,NUM_REGS);
        IF CHAVE=0 THEN DO;
                        PUT SKIP DATA(TR);
                        GET DATA;
                        GOTO LOOP;
                        END;
        REGC=BYTE;              /*       ENTRADA  ONLINE                  */
        DO I=0 TO NUM_REGS;
        CHAVE=CHAVE+2*I;
        CALL SITI;
        PUT SKIP EDIT(CHAVE,' PC=',REGA)  (F(4),A,F(4));
        END;
       GOTO LOOP;
```

```
FIM :  PUT  SKIP(3) EDIT('FIM DO PGMUS.     PC =',REGA)
                                          (A,F(4));
  END PGMUS;
```

```
MOVE:    PPOC(P_1,P_2,TAM);      /*  DESLOCAMENTO DE CAMPOS     */

 /*  ESTA POTINA DESLOCA O CAMPO APONTADO POP P_1 PAPA O
     CAMPO APONTADO POR P_2.  O NUMERO DE BYTES MOVIDOS
     E TAM.  SE TAM FOR NEGATIVO O MOVIMENTO E DA DIREITA
     PAPA A ESQUERDA.                                          */

   DECLARE     (P_1,P_2)          POINTER,
               TAM              FIXED BIN(15);

   DECLAPE     CAMPO1(512)      CHAP    BASED(P_1),
               CAMPO2(512)      CHAP    BASED(P_2),
               (INICIO,FIM,INCR,I)  FIXED BIN(15);


        IF TAM < 0 THEN DO;
                        INICIO=-TAM;
                        FIM=1;
                        INCP=-1;
                        END;

                   ELSE DO;
                        INICIO=1;
                        FIM=TAM;
                        INCR=1;
                        END;

        DO I=INICIO TO FIM BY INCP;
        CAMPO2(I) = CAMPO1(I);
        END;
END MOVE;
```

```
COMP:   PPOC(P_1,P_2,TAM) RETUPNS(FIXED BIN(15)); /*COMP DE CAMPOS    */

        /* ESTA POTINA COMPAPA OS CAMPOS APONTADOS POP P_1 E P_2
        RETORNANDO OS VAPORES SEGUINTES:
              -1   PRIMEIPO  E  MENOP
               0   CAMPOS    IGUAIS
               1   PRIMEIPO  E  MAIOP
        O NUM DE BYTES COMPARADOS ESTA EM TAM.                        */

   DECLAPE     (P_1,P_2) POINTER,
               TAM       FIXED BIN(15);

   DECLAPE     CAMPO1(512)     CHAP       BASED(P_1),
               CAMPO2(512)     CHAP       BASED(P_2),
               I               FIXED BIN(15);


       DO I=1 TO TAM;
       IF CAMPO1(I)  <  CAMPO2(I) THEN PETURN(-1);    /* CAMPO1 MENOP*/
       IF CAMPO1(I)  >  CAMPO2(I) THEN RETURN(1);     /* CAMPO1 MAIOP*/
       END;
       PETURN(0);                                     /*CAMPOS IGUAIS*/
END COMP;
```

```
SITI:    PROC;       /*  ROTINA     PRINCIPAL      */

   DECLARE  1 REGS   UNAL STATIC EXTERNAL,/* REGISTRADORES GERAIS    */
            2 REGA       BIT(8),         /*ACUMULADOR  -  COD RETORNO */
            2 REGB       BIT(8),         /* NUM DE ORDEM DA TRAL      */
            2 REGC       BIT(8),         /*  OPER  A SER EXECUTADA    */
            2 PEGDE      POINTER;        /* ENDER DA AREA DE E/S      */

   DECLARE    P_ES       POINTER,        /* ENDER DA AREA DE E/S      */
              RC         FIXED BIN(15),  /*   CODIGO  DE  RETORNO     */
              OPER       FIXED BIN(08);  /*  CODIGO DA OPERACAO       */

   DECLARE    VLR_70     POINTER EXTERNAL,/*    ENDERECO  FIXO  70    */
              P_70       POINTER ,
            1 TRTA       BASED(P_70),    /* REGIAO FIXA DE MEMORIA    */
            2 P_TRAL     POINTER,        /* PTR  DA  TRAL             */
            2 P_TAF      POINTER;        /* PTR  DA  TAF              */

   DECLARE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS    */
                                             /*      FISICOS          */
            2 JMP_ES(3)  CHAR  ,          /* JUMP PARA ROTINA DE E/S   */
            2 PROX_TAF   POINTER,         /* POINTER PARA PROXIMA TAF  */
            2 TYPE        BIT(8),         /*  TIPO DA TAF = 19         */
            2 S_IN_FIS   FIXED BIN(15),   /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET    FIXED BIN(15),   /* NUM DE SETORES            */
            2 STATUS     CHAR  ,          /* STATUS                    */
            2 RAIZ       FIXED BIN(15),   /* SETOR INICIO DO INDICE    */
            2 TAM_CH     FIXED BIN(15),   /* TAMANHO DA CHAVE          */
            2 AVAIL_IND  FIXED BIN(15),   /* PRIM SETOR VAZIO INDICE   */
            2 TOPO       FIXED BIN(15),   /* TOPO DA PILHA DO INDICE   */
            2 PILHA(10)  FIXED BIN(15),   /* PILHA - CAMINHO NO INDICE */
            2 POS_REG_I  FIXED BIN(15),   /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15),   /* SETOR INICIO DADOS        */
            2 AVAIL_DAD  FIXED BIN(15),   /* PRIM SETOR VAZIO -DADOS   */
            2 TAM_REG    FIXED BIN(15),   /* TAMANHO DO REGISTRO       */
            2 POS_SET_D  FIXED BIN(15),   /* POSICION  - SETOR -DADOS  */
            2 POS_REG_D  FIXED BIN(15),   /* POSICION  - REG   -DADOS  */
            2 OPER_ANT   FIXED BIN(15),   /* ULTIMA OPERACAO  FEITA    */
            2 P_BUF      POINTER,         /* END DO BUFFER GERAL       */
            2 MAX_UT     FIXED BIN(15);   /* MAX BYTE UTILIZAVEL       */

   DECLARE   BUFFER(1024) CHAR STATIC; /*  BUFFER  DE E/S             */

   DECLARE  (READS,WRITES,REWRTE,REWIND, ENDFLE,DELETE,READD,WRITED,
            TRATAF)     EXTERNAL ENTRY; /*       ROTINAS  EXTERNAS   */


     /*   R O T I N A   P R I N C I P A L          */
```

```
        P_70=VLR_70;
        P_BUF=ADDR(BUFFER);              /* ENDEP DO BUFFEP GERAL      */
        CALL TRATAF;                     /* ATUALIZA ENDER TRAL E TAF  */
        P_ES=REGDE;                      /* ENDEP DA AREA DE E/S       */
        OPEP=REGC;                       /* CODIGO  DE  OPEPACAO       */

             IF OPEP = 1    THEN CALL READS (P_ES,RC);
        ELSE IF OPEP = 2    THEN CALL WRITES(P_ES,RC);
        ELSE IF OPEP = 4    THEN CALL PEWRTE(P_ES,RC);
        ELSE IF OPEP = 5    THEN CALL REWIND(RC);
        ELSE IF OPEP = 6    THEN CALL ENDFLE(PC);
        ELSE IF OPEP = 7    THEN CALL DELETE(P_ES,RC);
        ELSE IF OPEP = 8    THEN CALL PEADD (P_ES,RC);
        ELSE IF OPER = 9    THEN CALL WPITED(P_ES,RC);

        ELSE RC=100;                 /*   CODIGO DE OPERACAO INVALIDO   */
        OPEP=RC; REGA=OPER;
END SITI;
```

```
TRATAF:   PROC;
END;
```

```
PETRVE:  PROC(P_CH,S_DAD,RC);      /*  BUSCA NO INDICE   */

   DECLARE    P_CH       POINTER,        /*  POINTER PARA A CHAVE      */
              S_DAD      FIXED BIN(15),  /*  ENDER DO SETOR DE DADOS   */
              RC         FIXED BIN(15);  /*  CODIGO DE RETORNO         */

   DECLARE    VLR_70     POINTER STATIC EXTERNAL,/*   VALOR 70         */
              P_70       POINTER ,
            1 TRTA       BASED(P_70),    /*  REGIAO FIXA - ENDER 70    */
            2 P_TRAL     POINTER,        /*  PTR   TRAL                */
            2 P_TAF      POINTER;        /*  PTP   TAF                 */

   DECLARE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS     */
                                             /*       FISICOS          */
            2 JMP_ES(3)  CHAR  ,          /* JUMP PARA ROTINA DE E/S    */
            2 PROX_TAF   POINTER,         /* POINTER PARA PROXIMA TAF   */
            2 TYPE        BIT(8),         /*  TIPO DA TAF = 19          */
            2 S_IN_FIS   FIXED BIN(15),   /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET    FIXED BIN(15),   /* NUM DE SETORES             */
            2 STATUS     CHAR  ,          /* STATUS                     */
            2 RAIZ       FIXED BIN(15),   /* SETOR INICIO DO INDICE     */
            2 TAM_CH     FIXED BIN(15),   /* TAMANHO DA CHAVE           */
            2 AVAIL_IND  FIXED BIN(15),   /* PRIM SETOR VAZIO INDICE    */
            2 TOPO       FIXED BIN(15),   /* TOPO DA PILHA DO INDICE    */
            2 PILHA(10)  FIXED BIN(15),   /* PILHA - CAMINHO NO INDICE  */
            2 POS_REG_I  FIXED BIN(15),   /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15),   /* SETOR INICIO DADOS         */
            2 AVAIL_DAD  FIXED BIN(15),   /* PRIM SETOR VAZIO -DADOS    */
            2 TAM_REG    FIXED BIN(15),   /* TAMANHO DO REGISTRO        */
            2 POS_SET_D  FIXED BIN(15),   /* POSICION  - SETOR -DADOS   */
            2 POS_REG_D  FIXED BIN(15),   /* POSICION  - REG   -DADOS   */
            2 OPER_ANT   FIXED BIN(15),   /* ULTIMA OPERACAO  FEITA     */
            2 P_BUF      POINTER,         /* END DO BUFFER GERAL        */
            2 MAX_UT     FIXED BIN(15);   /* MAX BYTE UTILIZAVEL        */

   DECLARE  1 BUFFER BASED(P_BUF) UNAL,/*     BUFFER                    */
            2 N_BYTES    FIXED BIN(15), /*     BYTES  OCUPADOS           */
            2 BUF1(510)  CHAR  ;        /*         RESTO                 */

   DECLARE  1 ENT_IND    BASED(P_1) UNAL,/*  ENTRADA NO INDICE          */
            2 PTR_E      FIXED BIN(15),  /* ENDER PROX SETOR DE INDICE*/
            2 END_DAD    FIXED BIN(15),  /* ENDER DO SETOR DE DADOS   */
            2 CH_ENT     CHAR  ;         /* CHAVE DA ENTRADA NO INDICE*/

   DECLARE    P_1        POINTER,  /* PTP DO REG MOVEL DE INDICE      */
              P_2        POINTER,  /* POINTER  DE USO GERAL           */
              (K,J,T,I,L) FIXED BIN(15); /* CONTADORES DE USO GERAL  */

   DCL        HEX    EXTERNAL ENTRY,    /* IMPRESSAO EM HEXA          */
              TP(5)  BIT(1)  EXTERNAL;
   DECLARE    LOG_RW EXTERNAL ENTRY,    /* ROTINA DE LEITURA/GRAVACAO */
              COMP EXTERNAL ENTRY RETURNS(FIXED BIN(15));
```

```
    /*   B U S C A   N O   I N D I C E                    */

      P_70=VLR_70;                      /* CAPPEGA PTP DA TRAL E TAF  */
      IF TP(3) THEN PUT SKIP(2) LIST('RETRVE');
      IF TR(4) THEN CALL HEX(P_CH,TAM_CH);

      J = RAIZ;                         /* ENDERECO DA RAIZ           */
      TOPO=0;

BTPEE:DO WHILE (J >  0);                /* PEPCORRE A ARVORE          */
      T=1;
      CALL LOG_RW(J,T,RC);              /* LE SETOR DE ENDERECO  J    */
      IF RC ¬= 0 THEN RETUPN;           /* ERRO NA LEITURA            */
      TOPO=TOPO+1;
      PILHA(TOPO)=J;                    /* COLOCA SETOR NA PILHA      */
      POS_REG_I=1;                      /* POSICIONA INICIO DO SETOR  */

     SETOR:  DO WHILE (POS_REG_I<N_BYTES-1); /*   VARRE  O SETOR     */
             IF TP(2) THEN PUT SKIP DATA(J);
             IF TR(2) THEN PUT LIST(POS_REG_I,TOPO,PILHA(TOPO));
             P_1 = ADDR(BUF1(POS_REG_I)); /*POSICIONA ENTRADA NO IND */
             P_2=ADDR(CH_ENT);          /* ENDER CHAVE DA ENTRADA     */
             K=COMP(P_CH,P_2,TAM_CH);   /* COMPARA CHAVE COM ENTRADA  */
             IF K > 0                   /* A CHAVE E MAIOR            */
             THEN POS_REG_I=POS_REG_I+TAM_CH+4; /* DESLOCA P/DIREITA */
             ELSE DO;
                  S_DAD = END_DAD;      /* ENDER DO SETOR DE DADOS    */
                  IF K < 0 THEN GOTO DESCE;  /* A CHAVE E MENOR       */
                           ELSE DO;     /* A CHAVE E IGUAL            */
                                RC=0;   /*     CHAVE ENCONTRADA       */
                                PETUPN;
                                END;
                  END;
     END SETOR;                         /* FIM DO BLOCO VARRE SETOR   */

      P_1=ADDR(BUF1(POS_REG_I));        /* DESCE PELA DIREITA         */
DESCE:J = PTR_E;                        /* DESCE  NA  ARVORE          */
END BTREE;                              /* FIM DO BLOCO VARRE ARVORE  */

      RC=103;                           /* CHAVE NAO ENCONTRADA       */
      PETURN;
END PETRVE;
```

```
INSEPT:  PROC(CHAVE,ENDER,RC); /* INSERCAO  NO  INDICE     */

   DECLARE    CHAVE      POINTER,      /*  POINTER PARA A CHAVE      */
              ENDER      FIXED BIN(15), /*  ENDER DO SETOR DE DADOS   */
              RC         FIXED BIN(15); /*  CODIGO DE RETORNO         */

   DECLARE    VLR_70     POINTER STATIC EXTERNAL,/*   VALOR 70       */
              P_70       POINTER ,
            1 TRTA       BASED(P_70),   /*  REGIAO FIXA - ENDER 70    */
            2 P_TRAL     POINTER,       /*  PTR   TRAL                */
            2 P_TAF      POINTER;       /*  PTR   TAF                 */

   DECLARE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS    */
                                             /*      FISICOS          */
            2 JMP_ES(3)  CHAR  ,          /* JUMP PARA ROTINA DE E/S   */
            2 PROX_TAF   POINTER,         /* POINTER PARA PROXIMA TAF  */
            2 TYPE        BIT(8),         /*  TIPO DA TAF = 19         */
            2 S_IN_FIS   FIXED BIN(15), /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET    FIXED BIN(15), /* NUM DE SETORES             */
            2 STATUS     CHAR  ,        /* STATUS                     */
            2 RAIZ       FIXED BIN(15), /* SETOR INICIO DO INDICE     */
            2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE           */
            2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE    */
            2 TOPO       FIXED BIN(15), /* TOPO DA PILHA DO INDICE    */
            2 PILHA(10)  FIXED BIN(15), /* PILHA - CAMINHO NO INDICE  */
            2 POS_REG_I  FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15), /* SETOR INICIO DADOS         */
            2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS    */
            2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTRO        */
            2 POS_SET_D  FIXED BIN(15), /* POSICION  - SETOR -DADOS   */
            2 POS_REG_D  FIXED BIN(15), /* POSICION  - REG   -DADOS   */
            2 OPER_ANT   FIXED BIN(15), /* ULTIMA OPERACAO  FEITA     */
            2 P_BUF      POINTER,       /* END DO BUFFER GERAL        */
            2 MAX_UT     FIXED BIN(15); /* MAX BYTE UTILIZAVEL        */

   DECLARE  1 BUFFER BASED(P_BUF) UNAL,/*     BUFFER                  */
            2 N_BYTES    FIXED BIN(15), /*     BYTES  OCUPADOS         */
            2 BUF1(510)  CHAR,          /*         RESTO               */
            2 BUF2(512)  CHAR;          /*   SEGUNDO BUFFER            */

   DECLARE  1 ENT_IND    BASED(P_1) UNAL,/*  ENTRADA NO INDICE         */
            2 PTR_E      FIXED BIN(15),  /* ENDER PROX SETOR DE INDICE*/
            2 END_DAD    FIXED BIN(15),  /* ENDER DO SETOR DE DADOS   */
            2 CH_ENT     CHAR  ;         /* CHAVE DA ENTRADA NO INDICE*/

   DECLARE    P_1        POINTER,  /* PTR DO REG MOVEL DE INDICE      */
              P_2        POINTER,  /* POINTER  DE USO GERAL           */
              P_CH       POINTER,       /*  POINTER PARA A CHAVE      */
              S_DAD      FIXED BIN(15), /*  ENDER DO SETOR DE DADOS   */
              (I,J,K,L,T) FIXED BIN(15), /* CONTADORES DE USO GERAL  */
              ADR  FIXED BIN(15)  BASED(P_2); /* USO GERAL            */
```

```
DCL         HEX    EXTERNAL ENTRY,    /* IMPRESSAO EM HEXA          */
            TR(5) BIT(1) EXTERNAL;
DECLARE     LOG_RW EXTERNAL ENTRY,    /* ROTINA DE LEITURA/GRAVACAO */
            ALLOC  EXTERNAL ENTRY,    /* ALOCACAO DE SETOR          */
            MOVE   EXTERNAL ENTRY,    /* ROTINA DE MOVER DADOS      */
            RETRVE EXTERNAL ENTRY,    /* ROTINA DE BUSCA NO INDICE  */
            COMP EXTERNAL ENTRY RETURNS(FIXED BIN(15));


      /*            I N S E R C A O   N O   I N D I C E          */

      IF TR(3) THEN PUT SKIP(2) LIST('INSERT');
      P_70=VLR_70;                    /*   CARREGA ENDERECO TAB TRAL*/
      IF TOPO < 1                     /* O POSICION.  ESTA INVALIDO */
      THEN DO;                        /* POSICIONA NO SUCESSOR      */
           CALL RETRVE(CHAVE,S_DAD,RC); /*    PROCURA CHAVE         */
           IF RC ¬= 103 THEN DO;
                           RC=111; /* A CHAVE JA  EXISTE - ERRO  */
                           RETURN;
                           END;
           END;
      ELSE DO;                        /* POSICIONAMENTO DE INDICE OK*/
           T=1;
           CALL LOG_RW(PILHA(TOPO),T,RC); /* LE SETOR DE IND ATUAL */
           IF RC ¬= 0 THEN RETURN;
           END;

      P_CH=CHAVE;                     /*   CHAVE A INSERIR          */
      S_DAD=ENDER;                    /*   ENDER DOS DADOS          */
      K,L=0;

INCL:   P_1=ADDR(BUF1(POS_REG_I));     /* POSICIONA REGISTRO MOVEL   */
   IF TR(2) THEN PUT SKIP LIST(TOPO,PILHA(TOPO),K,L,POS_REG_I);
   IF TR(4) THEN CALL HEX(P_CH,TAM_CH);
      P_2=ADDR(BUF1(POS_REG_I+TAM_CH+4)); /* DESLOCA PARA DIREITA  */
      J=POS_REG_I-N_BYTES-1;          /* NUM DE BYTES A DESLOCAR    */
      CALL MOVE(P_1,P_2,J);           /* DESLOCA FINAL P/DIREITA    */
      END_DAD=S_DAD;                  /* INCLUI NOVA ENTRADA NO     */
      PTR_E=L;                        /* INDICE NA POSICAO ATUAL    */
      P_2=ADDR(CH_ENT);               /* NO INDICE.  - ENDER DOS    */
      CALL MOVE(P_CH,P_2,TAM_CH);     /* DADOS,PTR ESQUERDA E CHAVE */
      N_BYTES=N_BYTES+TAM_CH+4;       /* ATUALIZA NUM BYTES OCUPADOS*/

      IF N_BYTES < 508                /* HA ESPACO NO SETOR         */
      THEN DO;                        /* INCLUSAO  OK               */
           T=2;
           CALL LOG_RW(PILHA(TOPO),T,RC); /*    REGRAVA O SETOR     */
           IF RC ¬= 0 THEN RETURN;    /*   ERRO NA GRAVACAO         */
           POS_REG_I=POS_REG_I+TAM_CH+4;/* ATUALIZA POSICIONAMENTO */
```

```
      IF L ¬= 0 THEN TOPO=0;  /* HOUVE SPLIT - INVALIDA POS.    */
      RC=0;
      RETUPN;
      END;


       /* S P L I T   N O   I N D I C E                          */

IF TP(3) THEN PUT SKIP(2) LIST('SPLIT INSERT');
T=1;
CALL ALLOC(L,T,RC);              /* ALOCA SETOR PAPA INDICE     */
IF RC ¬= 0 THEN RETURN;
I=(N_BYTES-2)/(TAM_CH+4);        /* NUM DE REGS DE INDICE       */
I=(I+1)/2;                       /* NUM DO REGISTRO DO MEIO     */
J=I*(TAM_CH+4)-N_BYTES;          /* NUM DE BYTES A MOVER (NEG)  */
I=I*(TAM_CH+4)+1;                /* POSICIONA REG MOVEL NO      */
P_1=ADDR(BUF1(I));               /* PRIMEIPO REG A MOVER        */
P_2=ADDP(BUF2(3));               /* POSICAO NO SEGUNDO BUFFER   */
CALL MOVE(P_1,P_2,J);            /* MOVE SEG PAPTE DO SETOR     */
P_2=ADDR(BUF2(1));
ADR=-J;                           /* NUM DE BYTES OCUP N_BYTES  */
T=-2;                             /* GPAVACAO - SEGUNDO BUFFEP  */
CALL LOG_PW(PILHA(TOPO),T,PC);   /*    PEGRAVA  SEGUNDA PAPTE  */

N_BYTES=N_BYTES+J-TAM_CH-2;      /*BYTES PESTANTES NA PRIM PAP */
T=2;
CALL LOG_RW(L,T,RC);             /* REGRAVA PRIMEIPA PAPTE      */
IF RC ¬= 0 THEN RETUPN;

      /* SOBE COM REGISTRO DO MEIO PELA APVORE                  */

I=I-TAM_CH-4;                    /* POSICIONA REG MOVEL NO PEG  */
P_1=ADDR(BUF1(I));               /* DO MEIO QUE VAI SUBIP       */

S_DAD=END_DAD;                   /* SALVA ENDER DOS DADOS       */
P_CH=ADDR(BUF2(513-TAM_CH));     /* SALVA CHAVE NO              */
P_2=ADDR(CH_ENT);                /*      FINAL DO               */
CALL MOVE(P_2,P_CH,TAM_CH);      /*   SEGUNDO   BUFFER          */
TOPO=TOPO-1;                     /* SOBE NA ARVORE              */
IF TOPO < 1                      /* SPLIT   NA  RAIZ            */
THEN DO;
      K=PILHA(1);                /* SALVA END DA RAIZ ANTEPIOR  */
      TOPO=1;
      T=1;
      CALL ALLOC(PILHA(TOPO),T,RC);      /* ALOCA NOVO SETOP    */
      IF RC ¬= 0 THEN RETURN;            /*   PARA A RAIZ       */
      POS_REG_I=1;                       /* POSICIONA NO INICIO*/
      RAIZ=PILHA(TOPO);                  /* ATUALIZA PTR P/RAIZ*/
      P_2=ADDR(BUF1(1));         /*     INCLUI POINTER          */
      ADR=K;                     /*      PARA A DIREITA         */
      N_BYTES=2;
```

```
            GOTO INCL;                  /* DESVIA P/INCLUSAO DA CHAVE */
            END;

         ELSE DO;                       /*    SPLIT  NO MEIO DA ARVORE */
            T=1;
            CALL LOG_RW(PILHA(TOPO),T,PC); /*    LE SETOR ACIMA      */
            IF PC ¬= 0 THEN RETURN;
            POS_REG_I=1;                /* POSICIONA NO INICIO         */
            IF TR(3) THEN PUT PAGE LIST('TESTE1');
            II=1024; IF TP(4) THEN CALL HEX(P_BUF,II);
            II=61;   IF TP(4) THEN CALL HEX(P_TAF,II);
                                        /* POSICIONA NO PRIM REG COM   */
            DO WHILE(POS_REG_I < N_BYTES-2); /* CHAVE MAIOR            */
            P_2=ADDR(BUF1(POS_REG_I+4));  /*  PTR CHAVE DA ENTRADA     */
            IF TP(4) THEN CALL HEX(P_CH,TAM_CH);
            IF TR(4) THEN CALL HEX(P_2,TAM_CH);
            IF COMP(P_CH,P_2,TAM_CH) > 0        /* DESLOCA P/DIREITA*/
            THEN POS_REG_I=POS_REG_I+TAM_CH+4;  /* SE CHAVE E MAIOR */
            ELSE GOTO INCL;             /* DESVIA P/INCLUSAO DA CHAVE */
            END;                        /* FIM DO BLOCO VARRE SETOR    */

            GOTO INCL;                  /* INCLUSAO NO FIM DO SETOR    */
            END;
END INSEPT;
```

```
WRITES:PROC(P_ES,RC);                    /*  GRAVACAO SEQUENCIAL        */


   DECLARE    P_ES       POINTER,        /* PTR- AREA DE E/S- USUARIO  */
              RC         FIXED BIN(15); /* CODIGO  DE RETORNO         */

   DECLARE  1 BUFFER     BASED (P_BUF), /* BUFFER  DE  E/S            */
            2 N_BYTES    FIXED BIN(15), /* NUM DE BYTES OCUPADOS      */
            2 PROX_SET   FIXED BIN(15), /* ENDER DO PROX SETOR        */
            2 BUF1(508)  CHAR  ,        /* AREA DISPONIVEL NO BUFFER  */
            2 BUF2(512)  CHAR;          /* SEGUNDO BUFFER             */

   DECLARE    VLR_70     POINTER EXTERNAL,/*    ENDERECO  FIXO  70     */
              P_70       POINTER ,
            1 TRTA       BASED(P_70),   /* REGIAO FIXA DE MEMORIA     */
            2 P_TRAL     POINTER,       /* PTR  DA  TRAL              */
            2 P_TAF      POINTER;       /* PTR  DA  TAF               */

   DECLARE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS    */
                                             /*       FISICOS         */
            2 JMP_ES(3)  CHAR  ,        /* JUMP PARA ROTINA DE E/S    */
            2 PROX_TAF   POINTER,       /* POINTER PARA PROXIMA TAF   */
            2 TYPE        BIT(8),       /*  TIPO DA TAF = 19          */
            2 S_IN_FIS   FIXED BIN(15), /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET    FIXED BIN(15), /* NUM DE SETORES             */
            2 STATUS     CHAR  ,        /* STATUS                     */
            2 RAIZ       FIXED BIN(15), /* SETOR INICIO DO INDICE     */
            2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE           */
            2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE    */
            2 TOPO       FIXED BIN(15), /* TOPO DA PILHA DO INDICE    */
            2 PILHA(10)  FIXED BIN(15), /* PILHA - CAMINHO NO INDICE  */
            2 POS_REG_I  FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15), /* SETOR INICIO DADOS         */
            2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS    */
            2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTRO        */
            2 POS_SET_D  FIXED BIN(15), /* POSICION  - SETOR -DADOS   */
            2 POS_REG_D  FIXED BIN(15), /* POSICION  - REG   -DADOS   */
            2 OPER_ANT   FIXED BIN(15), /* ULTIMA OPERACAO  FEITA     */
            2 P_BUF      POINTER,       /* END DO BUFFER GERAL        */
            2 MAX_UT     FIXED BIN(15); /* MAX BYTE UTILIZAVEL        */

   DECLARE   (P_1,P_2)   POINTER,       /* POINTER DE USO GERAL       */
             (I,J,T)     FIXED BIN(15); /* CONTADORES-USO GERAL       */

   DCL        HEX    EXTERNAL ENTRY,    /* IMPRESSAO EM HEXA          */
              TR(5)  BIT(1)  EXTERNAL;
   DECLARE   (PETRVE,LOG_RW,ALLOC, MOVE,INSERT,ENDFLE) EXTERNAL ENTRY,
              COMP      EXTERNAL ENTRY RETURNS(FIXED BIN(15));
```

```
/*   I N I C I O   D A   R O T I N A  W R I T E S        */
      P_70 = VLR_70;                  /*  CAPPEGA ENDER TAF E TRAL  */

      IF  OPEP_ANT ¬= 2               /* PRIMEIPO WRITE             */
      THEN  DO;
      OPEP_ANT=2;
            DO J=1 TO TAM_CH;         /* FOPMA CHAVE COM  FF        */
            UNSPEC(BUF2(J))='11111111'B;
            END;
            P_1=ADDR(BUF2(1));
            CALL PETRVE (P_1,J,RC);      /* POSICIONA FIM DO APQUIVO*/
            IF J=0 THEN DO;              /*  O ARQUIVO ESTA VAZIO   */
                        T=2;
                        CALL ALLOC (POS_SET_D,T,RC);
                        IF RC ¬= 0 THEN RETURN;
                        POS_PEG_D=1;
                        GOTO INCLUI;
                        END;
            T=1;
            CALL LOG_RW(J,T,RC);          /* LE SETOR CORRESPONDENTE*/
            IF RC ¬= 0 THEN PETURN;       /*      ERRO NA LEITURA   */
            POS_SET_D=J;                  /*     ATUALIZA           */
            POS_PEG_D=N_BYTES + 1;        /*         POSICIONAMENTO */
            END;

      I=POS_REG_D-TAM_REG;            /* COMPARA CHAVE COM ANTEPIOR */
      P_1 = ADDP(BUF1(I));            /* ENDER DA CHAVE ANTERIOR    */
      IF COMP(P_1,P_ES,TAM_CH)   >= 0
      THEN  DO;                       /* CHAVE ANT MAIOR OU IGUAL   */
            RC=101;
            RETURN;
            END;
% SKIP;
INCLUI:  I=POS_REG_D + TAM_PEG;          /* INCLUSAO DO REGISTRO       */
      IF I <= MAX_UT+1
      THEN DO;                        /* HA ESPACO NESTE SETOR      */
           P_1 = ADDR(BUF1(POS_REG_D));
           CALL MOVE(P_ES,P_1,TAM_REG); /* MOVE REGISTRO PAPA BUFF */
           POS_PEG_D = I;               /* ATUALIZA POSICIONAMENTO */
           RC=0;
           RETURN;
           END;

            /*     NAO HA MAIS ESPACO NO SETOP  */

           T=2;
           CALL ALLOC(J,T,PC);          /*  ALOCA  NOVO  SETOR        */
           IF RC ¬= 0 THEN DO;
                           CALL ENDFLE(J);
                           PETURN;
                           END;
```

```
            N_BYTES=POS_REG_D - 1;     /* NUM DE BYTES OCUPADOS      */
            PPOX_SET=J;                /* ENDEP DO PROXIMO SETOP     */
            T=2;
            CALL LOG_RW(POS_SET_D,T,PC); /*    GRAVA ESTE SETOR      */
            IF RC ¬= 0 THEN PETURN;      /*    ERRO NA GRAVACAO      */

            I=POS_PEG_D - TAM_REG;      /*    SALVA CHAVE DO ULTIMO  */
            P_1=ADDR(BUF1(I));          /*    PEGISTRO GRAVADO       */
            I=513-TAM_CH;               /*    NO FINAL DO SEGUNDO    */
            P_2=ADDR(BUF2(I));          /*    BUFFER  E INCLUI       */
            CALL MOVE(P_1,P_2,TAM_CH); /*    NO INDICE              */
            CALL  INSERT(P_2,POS_SET_D,PC); /* ROT INSERCAO INDICE */
            IF RC ¬= 0 THEN PETURN;

            POS_SET_D=J;                /*  ATUALIZA POS DE SETOP    */
            POS_REG_D=1;                /*  POS INICIO DO SETOR      */
            GOTO INCLUI;               /* FAZ A INCLUSAO NO NOVO SET */


END WPITES;
```

```
PEADS:   PROC(P_ES,RC);        /* LEITURA SEQUENCIAL                    */

   DECLARE    P_ES       POINTER,        /* PTR- AREA DE E/S- USUARIO  */
              RC         FIXED BIN(15); /* CODIGO  DE RETORNO         */

   DECLARE  1 BUFFER     BASED (P_BUF), /* BUFFER  DE  E/S            */
            2 N_BYTES    FIXED BIN(15), /* NUM DE BYTES OCUPADOS      */
            2 PROX_SET   FIXED BIN(15), /* ENDER DO PROX SETOP        */
            2 BUF1(508)  CHAR  ,        /* AREA DISPONIVEL NO BUFFER  */
            2 BUF2(512)  CHAR;          /* SEGUNDO BUFFER             */

   DECLARE    VLP_70     POINTER EXTERNAL, /* ENDERECO FIXO 70         */
              P_70       POINTER ,
            1 TRTA       BASED(P_70),   /*  REGIAO FIXA - ENDER 70     */
            2 P_TRAL     POINTER,       /* PTR  DA  TRAL              */
            2 P_TAF      POINTER;       /* PTR  DA  TAF               */

   DECLARE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS     */
                                             /*       FISICOS          */
            2 JMP_ES(3)  CHAR  ,        /* JUMP PARA ROTINA DE E/S    */
            2 PROX_TAF   POINTER,       /* POINTER PARA PROXIMA TAF   */
            2 TYPE        BIT(8),       /*  TIPO DA TAF = 19          */
            2 S_IN_FIS   FIXED BIN(15), /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET    FIXED BIN(15), /* NUM DE SETORES             */
            2 STATUS     CHAR  ,        /* STATUS                     */
            2 RAIZ       FIXED BIN(15), /* SETOR INICIO DO INDICE     */
            2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE           */
            2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE    */
            2 TOPO       FIXED BIN(15), /* TOPO DA PILHA DO INDICE    */
            2 PILHA(10)  FIXED BIN(15), /* PILHA - CAMINHO NO INDICE  */
            2 POS_REG_I  FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15), /* SETOR INICIO DADOS         */
            2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS    */
            2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTRO        */
            2 POS_SET_D  FIXED BIN(15), /* POSICION  - SETOR -DADOS   */
            2 POS_REG_D  FIXED BIN(15), /* POSICION  - REG   -DADOS   */
            2 OPER_ANT   FIXED BIN(15), /* ULTIMA OPERACAO  FEITA     */
            2 P_BUF      POINTER,       /* END DO BUFFER GERAL        */
            2 MAX_UT     FIXED BIN(15); /* MAX BYTE UTILIZAVEL        */

   DECLARE    P_1        POINTER,       /* POINTER DE USO GERAL       */
              T          FIXED BIN(15);

   DCL        HEX    EXTERNAL ENTRY;   /* IMPRESSAO EM HEXA          */
   DECLARE    (LOG_RW,MOVE,ENDFLE) EXTERNAL ENTRY;
```

```
        /*  I N I C I O   D A   P O T I N A   P E A D S                */

        P_70=VLR_70;                    /* CARREGA ENDER TAF E TRAL    */

        IF OPEP_ANT = 1 THEN GOTO OK;  /* OPERACAO ANT FOI READS      */
        IF OPEP_ANT = 2 THEN CALL ENDFLE(RC);/* OPER ANT FOI WRITES   */
        OPEP_ANT=1;
             IF S_IN_DAD=AVAIL_DAD      /*   APOUIVO VAZIO             */
             THEN DO;
                  RC=1;                 /* RC  DE FIM DE ARQUIVO       */
                  RETURN;
                  END;
         T=1;
         CALL LOG_RW(POS_SET_D,T,PC);
         IF PC ¬= 0 THEN PETURN;

   OK:    IF POS_REG_D <= N_BYTES        /* POSICIONAMENTO  VALIDO      */
        THEN DO;
             P_1=ADDR(BUF1(POS_PEG_D));  /* MOVE REGISTRO PAPA         */
             CALL MOVE(P_1,P_ES,TAM_PEG);/*    APEA  DO  USUARIO       */
             POS_REG_D = POS_PEG_D + TAM_PEG; /* ATUALIZA POSICIONAM */
             RC=0;
             RETURN;
             END;
  PROX:  IF PROX_SET = 0
        THEN DO;                        /* PTR DE PROX SETOR NULO      */
             RC=1;                      /* PC DE FIM DE ARQUIVO        */
             RETURN;
             END;
        POS_SET_D = PROX_SET;           /* ATUALIZA POS DE SETOR       */
        T=1;
        CALL LOG_PW(PROX_SET,T,PC);     /* LE PROXIMO SETOR            */
        IF PC ¬= 0 THEN PETURN;         /* EPPO NA LEITURA             */
        POS_REG_D = 1;                  /* ATUALIZA POS DE REGISTRO    */
        GOTO OK;
END READS;
```

```
PEADD:   PPOC(P_ES,RC);          /* LEITURA  DIRETA                    */


   DECLARE    P_ES       POINTER,        /* PTR- AREA DE E/S- USUARIO  */
              RC         FIXED BIN(15); /* CODIGO  DE RETORNO         */

   DECLARE  1 BUFFER     BASED (P_BUF), /* BUFFER  DE  E/S            */
            2 N_BYTES    FIXED BIN(15), /* NUM DE BYTES OCUPADOS      */
            2 PROX_SET   FIXED BIN(15), /* ENDER DO PROX SETOR        */
            2 BUF1(508)  CHAR  ,        /* AREA DISPONIVEL NO BUFFER  */
            2 BUF2(512)  CHAR;          /* SEGUNDO BUFFER             */

   DECLARE    VLR_70     POINTER EXTERNAL, /* ENDERECO FIXO 70         */
              P_70       POINTER ,
            1 TRTA       BASED(P_70),    /*  REGIAO FIXA - ENDER 70    */
            2 P_TRAL     POINTER,        /* PTR  DA  TRAL              */
            2 P_TAF      POINTER;        /* PTR  DA  TAF               */

   DECLARE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS     */
                                             /*       FISICOS          */
            2 JMP_ES(3)  CHAR  ,        /* JUMP PARA ROTINA DE E/S    */
            2 PROX_TAF   POINTER,       /* POINTER PARA PROXIMA TAF   */
            2 TYPE        BIT(8),       /*  TIPO DA TAF = 19          */
            2 S_IN_FIS   FIXED BIN(15), /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET    FIXED BIN(15), /* NUM DE SETORES             */
            2 STATUS     CHAR  ,        /* STATUS                     */
            2 PAIZ       FIXED BIN(15), /* SETOR INICIO DO INDICE     */
            2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE           */
            2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE    */
            2 TOPO       FIXED BIN(15), /* TOPO DA PILHA DO INDICE    */
            2 PILHA(10)  FIXED BIN(15), /* PILHA - CAMINHO NO INDICE  */
            2 POS_REG_I  FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15), /* SETOR INICIO DADOS         */
            2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS    */
            2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTRO        */
            2 POS_SET_D  FIXED BIN(15), /* POSICION  - SETOR -DADOS   */
            2 POS_REG_D  FIXED BIN(15), /* POSICION  - REG   -DADOS   */
            2 OPER_ANT   FIXED BIN(15), /* ULTIMA OPERACAO  FEITA     */
            2 P_BUF      POINTER,       /* END DO BUFFER GERAL        */
            2 MAX_UT     FIXED BIN(15); /* MAX BYTE UTILIZAVEL        */

   DECLARE    P_1        POINTER,        /* POINTER DE USO GERAL       */
              (I,J,T)    FIXED BIN(15); /* CONTADORES-USO GERAL       */

   DCL        HEX    EXTERNAL ENTRY;   /* IMPRESSAO EM HEXA           */
   DECLARE    (PETRVE,LOG_RW,MOVE,ENDFLE) EXTERNAL ENTRY,
               COMP      EXTERNAL ENTRY RETURNS(FIXED BIN(15));
```

```
        /* I N I C I O   D A   P O T I N A   R E A D D      */

        P_70=VLP_70;                    /* CARREGA ENDER TAF E TRAL   */

        IF OPEP_ANT = 2 THEN CALL ENDFLE(RC);
        OPEP_ANT=8;
        CALL RETPVE(P_ES,J,PC);         /* PROCURA CHAVE NO INDICE    */
        T=1;
        CALL LOG_RW(J,T,RC);            /* LE SETOR APONTADO          */
        IF PC ¬= 0 THEN RETURN;         /* ERRO NA LEITURA            */
        POS_SET_D=J;                    /*     ATUALIZA               */
        POS_REG_D=1;                    /*           POSICIONAMENTO   */
        DO WHILE (POS_REG_D < N_BYTES); /* VARRE O SETOR              */
        P_1=ADDR(BUF1(POS_REG_D));
        I=COMP(P_ES,P_1,TAM_CH);         /* COMP CHAVE COM ENTRADA    */
        IF I > 0                        /*  CHAVE MAIOR QUE ENTRADA   */
        THEN POS_REG_D=POS_REG_D+TAM_REG; /* DESLOCA PARA DIREITA     */
        ELSE IF I = 0                   /* CHAVE IGUAL                */
             THEN DO;                   /* MOVE PARA AREA DO USUARIO  */
                  CALL MOVE(P_1,P_ES,TAM_REG);
                  POS_REG_D=POS_REG_D+TAM_REG; /* ATUALIZA POSICION  */
                  RC=0;
                  RETURN;
                  END;
             ELSE DO;                   /*  CHAVE MENOR QUE ENTRADA   */
                  RC=109;               /*  REGISTRO NAO ENCONTRADO   */
                  RETURN;
                  END;
        END;
        RC=109;                         /*       FIM DO REGISTRO.     */
        RETURN;                         /*  CHAVE NAO ENCONTRADA      */
END READD;
```

```
WPITED:  PPOC(P_ES,PC);          /*  GRAVACAO  DIRETA                    */

   DECLARE    P_ES       POINTER,          /* PTP- APEA DE E/S- USUARIO  */
              PC         FIXED BIN(15);    /* CODIGO  DE RETORNO         */

   DECLARE  1 BUFFER     BASED (P_BUF),    /* BUFFER  DE  E/S            */
            2 N_BYTES    FIXED BIN(15),    /* NUM DE BYTES OCUPADOS      */
            2 PPOX_SET   FIXED BIN(15),    /* ENDER DO PROX SETOP        */
            2 BUF1(508)  CHAP  ,           /* AREA DISPONIVEL NO BUFFER  */
            2 N_BYTES2   FIXED BIN(15),    /* NUM DE BYTES OCUPADOS      */
            2 PROX_SET2  FIXED BIN(15),    /* ENDEP DO PROX SETOP        */
            2 BUF2(508)  CHAP;             /* SEGUNDO BUFFER             */

   DECLARE    VLR_70     POINTEP EXTEPNAL, /* ENDERECO FIXO 70           */
              P_70       POINTEP ,
            1 TRTA       BASED(P_70),      /*  PEGIAO FIXA - ENDER 70    */
            2 P_TPAL     POINTEP,          /* PTP  DA  TRAL              */
            2 P_TAF      POINTER:          /* PTP  DA  TAF               */

   DECLARE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS       */
                                             /*      FISICOS             */
            2 JMP_ES(3)  CHAP  ,           /* JUMP PARA ROTINA DE E/S    */
            2 PROX_TAF   POINTEP,          /* POINTEP PARA PROXIMA TAF   */
            2 TYPE        BIT(8),          /*  TIPO DA TAF = 19          */
            2 S_IN_FIS   FIXED BIN(15),    /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET    FIXED BIN(15),    /* NUM DE SETORES             */
            2 STATUS     CHAP  ,           /* STATUS                     */
            2 RAIZ       FIXED BIN(15),    /* SETOR INICIO DO INDICE     */
            2 TAM_CH     FIXED BIN(15),    /* TAMANHO DA CHAVE           */
            2 AVAIL_IND  FIXED BIN(15),    /* PPIM SETOR VAZIO INDICE    */
            2 TOPO       FIXED BIN(15),    /* TOPO DA PILHA DO INDICE    */
            2 PILHA(10)  FIXED BIN(15),    /* PILHA - CAMINHO NO INDICE  */
            2 POS_REG_I  FIXED BIN(15),    /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15),    /* SETOP INICIO DADOS         */
            2 AVAIL_DAD  FIXED BIN(15),    /* PRIM SETOR VAZIO -DADOS    */
            2 TAM_PEG    FIXED BIN(15),    /* TAMANHO DO REGISTRO        */
            2 POS_SET_D  FIXED BIN(15),    /* POSICION  - SETOP -DADOS   */
            2 POS_REG_D  FIXED BIN(15),    /* POSICION  - REG   -DADOS   */
            2 OPEP_ANT   FIXED BIN(15),    /* ULTIMA OPERACAO  FEITA     */
            2 P_BUF      POINTEP,          /* END DO BUFFER GERAL        */
            2 MAX_UT     FIXED BIN(15);    /* MAX BYTE UTILIZAVEL        */

   DECLARE  1 ENT_IND    BASED(P_1) UNAL,/*  ENTRADA NO INDICE           */
            2 PTP_E      FIXED BIN(15),   /* ENDEP PROX SETOR DE INDICE*/
            2 END_DAD    FIXED BIN(15),   /* ENDER DO SETOR DE DADOS    */
            2 CH_ENT     CHAP  ;          /* CHAVE DA ENTRADA NO INDICE*/

   DECLARE   (P_1,P_2,P_3) POINTER,     /* POINTEPS DE USO GEPAL         */
             (I,J,T,K)  FIXED BIN(15); /* CONTADORES-USO GERAL          */
```

```
DCL         HEX     EXTERNAL ENTPY,    /* IMPRESSAO EM HEXA          */
            TR(5)  BIT(1)  EXTERNAL;
DECLARE    (RETRVE,LOG_RW,MOVE, ENDFLE,ALLOC,INSERT) EXTERNAL ENTPY,
           COMP     EXTERNAL ENTRY RETURNS(FIXED BIN(15));
```

```
        /* I N I C I O   D A   P O T I N A   W R I T E D    */

        P_70=VLR_70;                    /* CARPEGA ENDER TAF E TRAL   */

        IF OPER_ANT = 2 THEN CALL ENDFLE(RC);
        OPEP_ANT=9;
        IF TP(3) THEN PUT SKIP(5) LIST('WRITED');
        IF TP(4) THEN CALL HEX(P_ES,TAM_CH);
      CALL RETPVE(P_ES,J,PC);          /* PROCURA CHAVE NO INDICE    */
      T=1;
    CALL LOG_RW(J,T,RC);              /* LE SETOR APONTADO          */
    IF RC ¬= 0 THEN RETURN;           /* ERRO NA LEITURA            */
    POS_SET_D=J;                      /*     ATUALIZA               */
    POS_REG_D=1;                      /*          POSICIONAMENTO    */
    DO WHILE (POS_REG_D < N_BYTES); /* VARRE O SETOR                */
    P_1=ADDP(BUF1(POS_REG_D));
  I=COMP(P_ES,P_1,TAM_CH);          /* COMP CHAVE COM ENTRADA     */
  IF I > 0                          /*  CHAVE MAIOR QUE ENTRADA   */
  THEN POS_REG_D=POS_REG_D+TAM_PEG; /* DESLOCA PARA DIREITA       */
  ELSE IF I = 0                     /* CHAVE IGUAL                */
       THEN DO;                     /* PEGISTPO JA EXISTE         */
            PC=111;
            PETUPN;
            END;
     ELSE GOTO INCL;              /*  INCLUI  PEGISTRO           */
END;
P_1=ADDP(BUF1(POS_REG_D));       /*  INCLUI NO FIM DO REGISTRO */

INCL:    I=POS_REG_D-N_BYTES-1;          /* NUM DE BYTES A DESLOC (NEG)*/
        P_2=ADDP(BUF1(POS_REG_D+TAM_PEG));
      CALL MOVE(P_1,P_2,I);           /* DESLOCA PARA  DIREITA      */
      CALL MOVE(P_ES,P_1,TAM_PEG);    /*  MOVE NOVO REG PARA BUFFEP */
      N_BYTES=N_BYTES+TAM_PEG;
    IF N_BYTES < 508                /* HA ESPACO NO SETOR         */
     THEN DO;
          T=2;
        CALL LOG_RW(POS_SET_D,T,RC); /* REGRAVA  SETOR            */
        RC=0;
        PETURN;
        END;
```

```
SPLIT:    T=2;
        IF TP(3) THEN PUT SKIP(2) LIST('SPLIT WRITED');
         CALL ALLOC(J,T,RC);              /*  ALOCA  NOVO  SETOR        */
         IF PC ¬= 0 THEN RETURN;

          I=N_BYTES/TAM_REG;              /*   POSICIONA                */
           I=(I+1)/2;                     /*  NO  MEIO                  */
           I=I*TAM_REG+1;                 /*  PRIMEIRO BYTE A  MOVER    */
           P_1=ADDR(BUF1(I));
           P_2=ADDR(BUF2);
         K=I-N_BYTES-1;                 /* NUM DE BYTES A MOVER (NEG)*/
         CALL MOVE(P_1,P_2,K);          /*  DESLOCA SEG PARTE P/BUF2 */
         N_BYTES2=-K;
         PPOX_SET2=PROX_SET;
       T=-2;
     CALL LOG_RW(J,T,RC);            /* REGPAVA SEGUNDA PARTE NO  */
     IF PC ¬= 0 THEN PETURN;          /* NOVO SETOR                */

           N_BYTES=N_BYTES+K;
           PROX_SET=J;
         T=2;
         CALL LOG_RW(POS_SET_D,T,RC); /*  REGPAVA PRIMEIRA PARTE   */
         IF PC ¬= 0 THEN PETURN;      /*  NO SETOR ANTIGO          */

           I=N_BYTES2-TAM_REG+1;         /*  POINTER PARA ULTIMA CHAVE*/
           P_1=ADDP(BUF2(I));            /*  DO SEGUNDO SETOR         */

           I=N_BYTES-TAM_REG+1;
           P_2=ADDR(BUF1(I));            /*  SALVA ULTIMA CHAVE       */
           P_3=ADDR(BUF2(508-TAM_CH));   /*  DO PPIMEIRO SETOR NO     */
           CALL MOVE(P_2,P_3,TAM_CH);    /*  FINAL  DE  BUF2          */

           CALL RETPVE(P_1,I,PC);        /*  SUBSTITUI ENDEPECO DO    */
           IF RC¬=0   RC¬=103 THEN PETUPN;/*NOVO SETOP NA ENTRADA    */
           P_1=ADDR(BUF1(POS_PEG_I-2)); /*  DO INDICE CORRESPONDENTE */
           END_DAD=J;                    /*  AO  SEGUNDO  SETOR       */
           T=2;
           CALL LOG_RW(PILHA(TOPO),T,PC);/* PEGRAVA SETOP DE INDICE  */
           TOPO=0;                       /* INVALIDA POSICIONAMENTO   */

           CALL INSERT(P_3,POS_SET_D,PC): /* INSERE  ULTIMA CHAVE DO */
           RC=0;                          /*  PRIMEIRO SETOR NO INDICE*/
           PETURN;
END WPITED;
```

```
DELETE:  PROC(P_ES,PC);      /* REMOCAO  DE REGISTRO     */

   DECLARE    P_ES       POINTER,        /*  POINTER PARA A CHAVE      */
              RC         FIXED BIN(15); /*  CODIGO DE RETORNO         */

   DECLARE    VLR_70     POINTER STATIC EXTERNAL,/*   VALOR 70        */
              P_70       POINTER ,
            1 TRTA       BASED(P_70),    /*  REGIAO FIXA - ENDER 70    */
            2 P_TRAL     POINTER,        /*  PTR   TRAL                */
            2 P_TAF      POINTER;        /*  PTP   TAF                 */

   DECLARE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS     */
                                             /*      FISICOS           */
            2 JMP_ES(3)  CHAR  ,          /* JUMP PARA ROTINA DE E/S    */
            2 PROX_TAF   POINTER,         /* POINTER PARA PROXIMA TAF   */
            2 TYPE        BIT(8),         /*  TIPO DA TAF = 19          */
            2 S_IN_FIS   FIXED BIN(15), /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET    FIXED BIN(15), /* NUM DE SETORES             */
            2 STATUS     CHAR  ,         /* STATUS                     */
            2 RAIZ       FIXED BIN(15), /* SETOR INICIO DO INDICE     */
            2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE           */
            2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE    */
            2 TOPO       FIXED BIN(15), /* TOPO DA PILHA DO INDICE    */
            2 PILHA(10)  FIXED BIN(15), /* PILHA - CAMINHO NO INDICE  */
            2 POS_REG_I  FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15), /* SETOR INICIO DADOS         */
            2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS    */
            2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTRO        */
            2 POS_SET_D  FIXED BIN(15), /* POSICION  - SETOR -DADOS   */
            2 POS_REG_D  FIXED BIN(15), /* POSICION  - REG   -DADOS   */
            2 OPER_ANT   FIXED BIN(15), /* ULTIMA OPERACAO  FEITA     */
            2 P_BUF      POINTER,       /* END DO BUFFER GERAL        */
            2 MAX_UT     FIXED BIN(15); /* MAX BYTE UTILIZAVEL        */

   DECLARE  1 BUFFER BASED(P_BUF) UNAL,/*     BUFFER                  */
            2 N_BYTES    FIXED BIN(15), /*     BYTES  OCUPADOS         */
            2 PROX_SET   FIXED BIN(15), /*     PROXIMO SETOR DE DADOS  */
            2 BUF1(508)  CHAR,          /*     BUF1                    */
            2 N_BYTES2   FIXED BIN(15), /*     BYTES  OCUPADOS         */
            2 PROX_SET2  FIXED BIN(15), /*     PROXIMO SETOR DE DADOS  */
            2 BUF2(508)  CHAR;          /*     BUF2                    */

   DECLARE  1 ENT_IND    BASED(P_1) UNAL,/*MASCARA DE ENTR  NO INDICE  */
            2 PTR_E      FIXED BIN(15),  /* ENDER PROX SETOR DE INDICE*/
            2 END_DAD    FIXED BIN(15),  /* ENDER DO SETOR DE DADOS   */
            2 CH_ENT     CHAR  ;         /* CHAVE DA ENTRADA NO INDICE*/

   DECLARE    P_1        POINTER,  /* PTR MASCARA DE ENTRADA NO INDICE*/
              P_2        POINTER,  /* POINTER  DE USO GERAL            */
              P_3        POINTER,  /* POINTER  DE USO GERAL            */
              (I,J,K,L,T) FIXED BIN(15); /* CONTADORES DE USO GERAL  */
```

```
  DCL         HEX    EXTEPNAL ENTRY,    /* IMPPESSAO EM HEXA          */
              TR(5) BIT(1) EXTERNAL;
DECLARE     LOG_RW EXTERNAL ENTRY,    /* ROTINA DE LEITURA/GRAVACAO */
            PEMOVE EXTEPNAL ENTRY,    /* PEMOCAO  NO INDICE         */
            MOVE   EXTERNAL ENTRY,    /* ROTINA DE MOVER DADOS      */
            PETRVE EXTERNAL ENTRY,    /* ROTINA DE BUSCA NO INDICE  */
            COMP EXTERNAL ENTPY RETUPNS(FIXED BIN(15));
```

```
          /* I N I C I O   D A   P O T I N A   D E L E T E          */

      P_70=VLR_70;                    /* CARPEGA ENDER TAF E TRAL    */
      OPEP_ANT=7;
      IF TP(3) THEN PUT SKIP(2) LIST('DELETE');
      CALL RETRVE(P_ES,J,PC);         /* PROCURA CHAVE NO INDICE     */
      T=1;
    CALL LOG_RW(J,T,PC);            /* LE SETOR APONTADO           */
    IF PC ¬=0 THEN RETUPN;          /* ERRO NA LEITURA             */
    POS_SET_D=J;                    /* ATUALIZA                    */
    POS_REG_D=1;                    /*         POSICIONAMENTO      */

      DO WHILE(POS_REG_D < N_BYTES);/*  VARRE  O  SETOR            */
      P_1=ADDR(BUF1(POS_REG_D));
    I=COMP(P_ES,P_1,TAM_CH);        /*  COMPAPA  CHAVE COM REG     */
    IF I>0                          /*  CHAVE MAIOR                */
    THEN POS_REG_D=POS_REG_D+TAM_PEG;/* DESLOCA PARA DIRETA        */
    ELSE IF I<0 THEN GOTO ER;       /*  CHAVE MENOR                */
                ELSE GOTO PEMOV;    /*  CHAVE IGUAL                */
    END;

EP:     PC=112;                         /* CHAVE NAO ENCONTRADA    */
      RETUPN;

REMOV:  P_2=ADDR(BUF1(POS_PEG_D+TAM_REG));   /*  REGISTRO SEGUINTE  */
      I=N_BYTES-POS_PEG_D-TAM_PEG+1; /*  NUM DE BYTES A MOVER      */
      IF TP(2) THEN PUT SKIP DATA(I);
      CALL MOVE(P_2,P_1,I);           /* DESLOCA RESTO PARA ESQUERDA*/
      N_BYTES=N_BYTES-TAM_REG;        /*  NUM DE BYTES OCUPADOS     */
      IF N_BYTES  > 253                /* MAIS DA METADE OCUPADA*/
      THEN DO;
         T=2;
         CALL LOG_RW(POS_SET_D,T,PC);   /* PEGPAVA SETOR          */
         IF RC¬=0 THEN RETURN;
       IF I=0                        /* ULTIMO PEGISTRO DO SETOR  */
       THEN DO;/* SUBSTITUI CHAVE NO INDICE PELA DO ANTECESSOR */
            T=-1;
            CALL LOG_PW(PILHA(TOPO),T,PC); /*LE SETOR DE INDICE*/
            IF RC ¬= 0 THEN RETURN;     /*     EM  BUF2          */
            P_2=ADDR(BUF1(POS_REG_D-TAM_REG)); /*CH DO ANTECESS*/
            P_3=ADDP(BUF2(POS_REG_I+2)); /*    CHAVE ATUAL      */
            CALL MOVE(P_2,P_3,TAM_CH); /*   SUBSTITUI  CHAVE    */
            T=-2;
            CALL LOG_RW(PILHA(TOPO),T,RC); /*REGRAVA SET IND    */
            IF PC ¬= 0 THEN PETURN;
            END;
      PC=0;                         /* DELETE  OK                */
      RETUPN;
    END;
```

```
   IF I=0 THEN P_3=P_ES;                          /*  PTR PARA ULTIMA */
          ELSE P_3=ADDP(BUF1(N_BYTES +1 -TAM_REG));/*CH PRIM SET*/
   T=-1;
   CALL LOG_RW(PROX_SET,T,RC) ;     /* LE PPOXIMO SETOR PARA BUF2 */
   IF PC ¬= 0 THEN RETURN;
   P_1=ADDP(BUF1(N_BYTES+1)) ;      /* FINAL DO PRIMEIRO SETOR    */
   P_2=ADDP(BUF2) ;                 /* INICIO DO SEGUNDO SETOR    */
   K=N_BYTES2;                      /* SALVA INFORMACOES DE       */
   L=PROX_SET2;                     /* CONTPOLE DE BUF2           */
   CALL MOVE(P_2,P_1,K) ;           /* MOVE SEGUNDO SETOR         */
   N_BYTES=N_BYTES+K;               /* ACERTA NUM DE BYTES OCUP   */
   IF TP(4)  THEN CALL HEX(P_BUF,N_BYTES);
IF N_BYTES > 508 THEN GOTO UNDERF;
```

```
/*              " C A T E N A T I O N "                    */

      IF TP(3) THEN PUT SKIP(3) LIST('CATENATION_DADOS');
      PROX_SET=PROX_SET2;
      T=2;
    CALL LOG_RW(POS_SET_D,T,RC);    /* PEGRAVA NO PRIMEIRO SETOR  */
    IF PC ¬= 0 THEN RETURN;

      P_2=ADDR(BUF2(509-TAM_CH));     /* SALVA ULT CHAVE DO PRIMEIRO*/
      CALL MOVE(P_3,P_2,TAM_CH);      /* SETOR NO FINAL DE BUF2     */

      P_3=ADDR(BUF2(509-2*TAM_CH));   /*      SALVA   ULT   CHAVE   */
      P_1=ADDR(BUF1(N_BYTES+1-TAM_REG)); /*    DO  SEGUNDO         */
      CALL MOVE(P_1,P_3,TAM_CH);      /* SETOR NO FINAL DE BUF2     */

      CALL RETRVE(P_3,I,RC);             /* SUBSTITUI  ENDERECO    */
      P_1=ADDR(BUF1(POS_REG_I-2));       /* DO PRIMEIRO SETOR NA   */
      END_DAD=POS_SET_D;                 /* ENTRADA  DO  INDICE    */
      T=2;                               /* CORRESP AO SEGUNDO     */
      IF TP(4) THEN CALL HEX(P_1,TAM_CH);
      CALL LOG_RW(PILHA(TOPO),T,RC);     /* REGRAVA SETOR DE INDICE*/
      TOPO=0;                            /* INVALIDA POSICIONAMENTO*/

      CALL  REMOVE(P_2,RC);           /* REMOVE ENTRADA DE INDICE   */
      IF RC ¬= 0 THEN RETURN;         /* CORRESP AO PRIMEIRO SETOR  */

      RC=0;                           /* "CATENATION"  OK           */
      RETURN;


/*              " U N D E R F L O W "                      */

UNDERF:  I=N_BYTES/TAM_REG;              /* NUM DE REGS NO PRIM SETOR  */
      IF TP(3) THEN PUT SKIP LIST('UNDERFLOW_DADOS');
      I=I/2;
    I=I*TAM_REG;                    /* BYTES A FICAR NO PRIM SET  */
    J=I-N_BYTES;                    /* BYTES NO SEG SETOR (NEG)   */
    IF TP(3) THEN PUT SKIP DATA(I,J);
    P_1=ADDR(BUF1(I+1));            /*  DESLOCA SEGUNDA  PARTE    */
    CALL MOVE(P_1,P_2,J);           /*  PARA  BUF2                */
    N_BYTES=N_BYTES+J;              /*  NUM DE BYTES  EM  BUF1    */
    N_BYTES2=-J;                    /*  NUM DE BYTES  EM  BUF2    */
    PROX_SET2=L;

      T=2;
    CALL LOG_RW(POS_SET_D,T,RC);    /*  REGRAVA PRIMEIRO SETOR    */
    IF PC ¬= 0 THEN RETURN;
    T=-2;
    CALL LOG_RW(PROX_SET,T,RC);     /*  PEGRAVA  SEGUNDO  SETOR   */
    IF PC ¬= 0 THEN RETURN;
```

```
   /* SUBSTITUI CHAVE CORRESP PELA DO ULTIMO REGISTRO DO PRIM SETOR  */

        T=-1;
        CALL LOG_RW(PILHA(TOPO),T,PC); /* LE SETOR DE INDICE EM BUF2 */
        IF PC ¬= 0 THEN RETURN;
        P_1=ADDR(BUF2(POS_REG_I+2));    /* CHAVE DA ENTRADA DE INDICE */
        P_2=ADDR(BUF1(N_BYTES+1-TAM_REG));/* SUBSTITUI CHAVE ANTER   */
        CALL MOVE(P_2,P_1,TAM_CH);       /* MOVE NOVA CHAVE          */
        IF TR(4) THEN CALL HEX(P_2,TAM_CH);
        T=-2;
        CALL LOG_RW(PILHA(TOPO),T,PC);  /* REGRAVA SETOR DE INDICE   */
END DELETE;
```

```
PEMOVE:  PROC(CHAVE,RC); /* INSERCAO  NO  INDICE     */

   DECLARE    CHAVE      POINTER,        /*  POINTER PARA A CHAVE       */
              RC         FIXED BIN(15); /*  CODIGO DE RETORNO          */

   DECLARE    VLR_70     POINTER STATIC EXTERNAL,/*   VALOR 70         */
              P_70       POINTER ,
            1 TRTA       BASED(P_70),    /*  REGIAO FIXA - ENDER 70     */
            2 P_TRAL     POINTER,        /*  PTR   TRAL                 */
            2 P_TAF      POINTER;        /*  PTR   TAF                  */

   DECLARE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS      */
                                             /*       FISICOS           */
            2 JMP_ES(3)  CHAR  ,          /* JUMP PARA ROTINA DE E/S     */
            2 PROX_TAF   POINTER,         /* POINTER PARA PROXIMA TAF    */
            2 TYPE        BIT(8),         /*  TIPO DA TAF = 19           */
            2 S_IN_FIS   FIXED BIN(15), /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET    FIXED BIN(15), /* NUM DE SETORES              */
            2 STATUS     CHAR  ,         /* STATUS                      */
            2 RAIZ       FIXED BIN(15), /* SETOR INICIO DO INDICE      */
            2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE            */
            2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE     */
            2 TOPO       FIXED BIN(15), /* TOPO DA PILHA DO INDICE     */
            2 PILHA(10)  FIXED BIN(15), /* PILHA - CAMINHO NO INDICE   */
            2 POS_REG_I  FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15), /* SETOR INICIO DADOS          */
            2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS     */
            2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTRO         */
            2 POS_SET_D  FIXED BIN(15), /* POSICION  - SETOR -DADOS    */
            2 POS_REG_D  FIXED BIN(15), /* POSICION  - REG   -DADOS    */
            2 OPER_ANT   FIXED BIN(15), /* ULTIMA OPERACAO  FEITA      */
            2 P_BUF      POINTER,       /* END DO BUFFER GERAL         */
            2 MAX_UT     FIXED BIN(15); /* MAX BYTE UTILIZAVEL         */

   DECLARE  1 BUFFER BASED(P_BUF) UNAL,/*     BUFFER                   */
            2 N_BYTES    FIXED BIN(15), /*     BYTES  OCUPADOS          */
            2 BUF1(510)  CHAR,          /*         RESTO                */
            2 N_BYTES2   FIXED BIN(15), /*     BYTES  OCUPADOS          */
            2 BUF2(510)  CHAR;          /*   SEGUNDO BUFFER             */

   DECLARE  1 ENT_IND    BASED(P_1) UNAL,/*  ENTRADA NO INDICE          */
            2 PTR_E      FIXED BIN(15),  /* ENDER PROX SETOR DE INDICE*/
            2 END_DAD    FIXED BIN(15),  /* ENDER DO SETOR DE DADOS    */
            2 CH_ENT     CHAR  ;         /* CHAVE DA ENTRADA NO INDICE*/

   DECLARE    P_1        POINTER,  /* PTR DO REG MOVEL DE INDICE       */
              P_2        POINTER,  /* POINTER  DE USO GERAL            */
              P_3        POINTER,  /* POINTER  DE USO GERAL            */
              P_CH       POINTER,        /*  POINTER PARA A CHAVE      */
              S_DAD      FIXED BIN(15), /*  ENDER DO SETOR DE DADOS    */
              (I,J,L,P,T) FIXED BIN(15), /* CONTADORES DE USO GERAL    */
              ADR  FIXED BIN(15)  BASED(P_2); /* USO GERAL             */
```

```
DCL         HEX    EXTERNAL ENTRY,    /* IMPRESSAO EM HEXA          */
            TR(5) BIT(1) EXTERNAL;
DECLARE     LOG_RW EXTERNAL ENTRY,     /* ROTINA DE LEITURA/GRAVACAO */
            ALLOC  EXTERNAL ENTRY,     /* ALOCACAO DE SETOR          */
            MOVE   EXTERNAL ENTRY,     /* ROTINA DE MOVER DADOS      */
            RETRVE EXTERNAL ENTRY,     /* ROTINA DE BUSCA NO INDICE  */
            COMP EXTERNAL ENTRY RETURNS(FIXED BIN(15));
```

```
    /*   R E M O C A O   N O   I N D I C E                    */
  IF TP(3) THEN PUT SKIP(2) LIST ('REMOVE');

    P_70=VLR_70;                    /* CARPEGA PTR TRAL E TAF     */
    CALL RETRVE(CHAVE,J,RC);        /* PROCURA CHAVE NO INDICE    */
    IF PC ¬=0 THEN RETURN;
  P_1=ADDR(BUF1(POS_PEG_I));      /* POSICIONA MASCARA DA ENTRAD*/
  IF PTP_E = 0 THEN GOTO REM;     /* ESTA EM UMA FOLHA          */

    /* SUBSTITUI ENTRADA PELA SUCESSORA E PEMOVE ESTA           */

    J=PILHA(TOPO);  I=POS_PEG_I;    /* SALVA POSICIONAMENTO       */
    P_1=ADDR(BUF1(POS_REG_I+TAM_CH+4)); /* PPOCURA ENTR SUCESSORA*/
    IF TR(3) THEN PUT SKIP LIST('DESCE ATE FOLHA');
    DO WHILE (PTP_E > 0);           /*   DESCENDO ATE UMA FOLHA   */
    TOPO=TOPO+1;
  PILHA(TOPO)=PTR_E;              /* COLOCA SETOR NA PILHA      */
  T=-1;
  CALL LOG_RW(PTR_E,T,PC);        /* LE SETOP DE INDICE EM BUF2 */
  P_1=ADDR(BUF2(1));              /* POSICIONA MASC DE ENTRADA  */
  END;

    POS_REG_I=1;
  P_1=ADDR(BUF2(POS_REG_I+2));    /*  MOVE   ENDERECO   DOS     */
  P_2=ADDR(BUF1(I+2));            /*  DADOS   E  CHAVE   DA     */
  P=TAM_CH+2;                     /*  ENTRADA    SUCESSORA      */
  CALL MOVE(P_1,P_2,P);           /*  PARA  A  ORIGINAL         */
  T=2;
CALL LOG_RW(J,T,PC);            /* PEGPAVA  SETOP  ORIGINAL   */
IF PC ¬= 0 THEN RETURN;
P_2=ADDR(N_BYTES2);
P=N_BYTES2+2;
CALL MOVE(P_2,P_BUF,P);         /* MOVE BUF2 PARA BUF1        */
P_1=ADDR(BUF1(POS_PEG_I));      /*  PTP ENTPADA SUCESSORA     */
```

```
        /*                  REMOVE   ENTRADA                          */

PEM:     I=N_BYTES+1-POS_REG_I;          /* NUM DE BYTES A DESLOCAR    */
       IF TP(4) THEN CALL HEX(P_1,TAM_CH);
       P_2=ADDR(BUF1(POS_REG_I+TAM_CH+4));
       CALL MOVE(P_2,P_1,I);           /* DESLOCA PARA A ESQUERDA    */
       N_BYTES=N_BYTES-TAM_CH-4;       /* DECREMENTA NUM DE BYTES    */

        IF N_BYTES > 254               /* MAIS DA METADE OCUPADA     */
        THEN DO;
             T=2;
           CALL LOG_RW(PILHA(TOPO),T,RC);/* REGRAVA SETOR           */
           TOPO=0;                    /*  INVALIDA  POSICIONAMENTO  */
           RC=0;                      /* REMOVE OK                  */
           RETURN;
           END;


        /*   TENTA  "CATENATION"  OU  "UNDERFLOW"          */

        TOPO=TOPO-1;                   /* SOBE NA ARVORE             */
        IF TOPO < 1
      THEN DO;                        /* REMOCAO NA RAIZ            */
           IF TR(3) THEN PUT SKIP LIST('REMOCAO NA RAIZ');
           IF N_BYTES < 1
        THEN DO;                      /* RAIZ FICOU VAZIA           */
             RAIZ=R;                  /* NOVA RAIZ                  */
             END;
        ELSE DO;
           TOPO=TOPO+1;
           T=2;
           CALL LOG_RW(PILHA(TOPO),T,RC); /* REGRAVA SETOR   */
           IF RC ¬= 0 THEN RETURN;
         END;
     RC=0;                     /* REMOCAO  NA  RAIZ  OK      */
     TOPO=0;
     RETURN;
     END;

        /*                  PROCURA   ENTRADA  PAI                    */

        T=-1;
      CALL LOG_RW(PILHA(TOPO),T,RC);/* LE PAGINA PAI EM BUF2      */
      IF RC ¬= 0 THEN RETURN;
    POS_REG_I=1;
    P_1=ADDR(BUF2(POS_REG_I));
    DO WHILE  (POS_REG_I < N_BYTES2);  /* VARRE A PAGINA PAI     */
    IF PTR_F = PILHA(TOPO+1)       /* ENCONTROU ENTRADA PAI      */
    THEN GOTO IRMAOS;
  POS_REG_I=POS_REG_I+TAM_CH+4; /* DESLOCA PARA A DIREITA      */
  P_1=ADDR(BUF2(POS_REG_I));
```

```
END;
RC=115;                          /* ENTRADA PAI NAO ENCONTRADA */
RETURN;
```

```
        /*                  PPOCUPA  IPMAOS                              */

IRMAOS:   IF POS_REG_I  < N_BYTES2-1    /*  OBTEM IRMAO DA DIPEITA     */
        THEN DO;
             P_1=ADDP(BUF2(POS_REG_I+TAM_CH+4));
             R=PTR_E;                   /* ENDEP DO IRMAO 'A DIREITA  */
             L=PILHA(TOPO+1);           /* ENDEP DA PAGINA ORIGINAL   */
             P_2=ADDP(BUF2(POS_REG_I+2)); /*    MOVE    ENTRADA      */
             P_3=ADDR(BUF1(N_BYTES+1));   /*    PAI  PAPA  O         */
             I=TAM_CH+2;                  /*    FINAL   DA           */
             CALL MOVE(P_2,P_3,I);        /*    PAGINA  ORIGINAL     */
             T=-1;
             CALL LOG_RW(R,T,PC);         /* LE IRMAO DA DIP EM BUF2*/
             IF RC ¬= 0 THEN RETUPN;
             P_2=ADDR(BUF1(N_BYTES+TAM_CH+5)); /*  JUNTA    IRMAO   */
             P_3=ADDR(BUF2(1));           /*  DA   DIREITA         */
             I=N_BYTES2;                  /*  'A  PAGINA           */
             CALL MOVE(P_3,P_2,I):        /*  ORIGINAL             */
             N_BYTES=N_BYTES+TAM_CH+2+I;
             IF TR(2) THEN PUT SKIP LIST(L,P,PILHA(TOPO),POS_REG_I);
             IF TR(4) THEN CALL HEX(P_BUF,N_BYTES);
           END;

        ELSE DO;                        /* NAO TEM IRMAO 'A DIREITA   */
             POS_REG_I=POS_REG_I-TAM_CH-4;/* NOVA ENTRADA PAI          */
             P_1=ADDR(BUF2(POS_REG_I));
           L=PTP_E;                   /* ENDER DO IRMAO 'A ESQUERDA */
           R=PILHA(TOPO+1);           /* ENDEP DA PAGINA ORIGINAL   */
           P_2=ADDP(BUF1(1));         /*  DESLOCA ENTRADAS DE BUF1  */
           P_3=ADDR(BUF1(TAM_CH+3));/*  PAPA  A  DIREITA  PAPA      */
           I=-TAM_CH-2;               /*  DEIXAP ESPACO PARA  A     */
           CALL MOVE(P_2,P_3,I);      /*  ENTRADA PAI               */
           P_3=ADDR(BUF2(POS_REG_I+2));/*  MOVE  ENTRADA PAI        */
           CALL MOVE(P_3,P_2,I);       /* PAPA O INICIO DE BUF1     */
           P_3=ADDR(BUF2(1));
         N_BYTES2=N_BYTES-I;
         CALL MOVE(P_2,P_3,N_BYTES2);/* MOVE BUF1 PARA BUF2       */
         T=1;
       CALL LOG_PW(L,T,RC);      /* LE IRMAO DA ESQUERDA  BUF1 */
       IF PC ¬= 0 THEN RETURN;
       P_2=ADDR(BUF1(N_BYTES+1));/*  JUNTA   PAGINA  ORIGINAL */
       I=N_BYTES2;               /*          AO  IRMAO        */
       CALL MOVE(P_3,P_2,I);     /*        DA   DIREITA       */
       N_BYTES=N_BYTES+I;
     IF TR(2) THEN PUT SKIP LIST(L,P,PILHA(TOPO),POS_REG_I);
     IF TR(4) THEN CALL HEX(P_BUF,N_BYTES);
   END;

        IF N_BYTES > 510 THEN GOTO UNDEPF;
```

```
        /*                " C A T E N A T I O N "                    */

          IF TR(3) THEN PUT SKIP LIST('CATENATION_INDICE');
        T=2;
        CALL LOG_RW(P,T,PC);          /* REGRAVA NO SETOR DA DIREITA*/
        IF RC ¬= 0 THEN RETUPN;
      GOTO PEM;                      /* PEMOVE ENTPADA PAI         */

        /*                  " U N D E R F L O W "                    */

UNDEPF:        I=N_BYTES/(TAM_CH+4);     /* NUM TOTAL DE ENTPADAS      */
              IF TR(3) THEN PUT SKIP LIST('UNDERFLOW_INDICE');
            I=(I+1)/2;
          I=I*(TAM_CH+4);             /* NUM DE BYTES RESTANTES-BUF1*/
          IF TP(2) THEN PUT SKIP DATA(I);
          N_BYTES2=N_BYTES-I;
          P_2=ADDR(BUF1(I+1));        /*PPIMEIRO BYTE A MOVER P/BUF2*/
          P_3=ADDR(BUF2(1));            /* MOVE  SEGUNDA  PARTE    */
          CALL MOVE(P_2,P_3,N_BYTES2);/* PAPA  BUF2                */
          T=-2;
          CALL LOG_PW(R,T,PC);        /* REGRAVA PAGINA DA DIREITA  */
          IF RC ¬= 0 THEN RETURN;
          N_BYTES=I-TAM_CH-2;
          T=2;
        CALL LOG_RW(L,T,PC);       /* PEGPAVA PAGINA DA ESQUEPDA */
        IF PC ¬= 0 THEN RETUPN:
        T=-1;
        CALL LOG_RW(PILHA(TOPO),T,PC); /* LE PAGINA PAI EM BUF2*/
        IF RC ¬= 0 THEN RETURN;
        P_1=ADDR(BUF1(N_BYTES+1));    /*   SUBSTITUI  ENTRADA   */
        P_2=ADDR(BUF2(POS_REG_I+2)); /*   PAI  PELA   QUE      */
        I=TAM_CH+2;                   /*   FICOU  NO  MEIO      */
        CALL MOVE(P_1,P_2,I);         /*   DAS DUAS PAGINAS     */
        T=-2;
        CALL LOG_RW(PILHA(TOPO),T,PC);  /* PEGPAVA PAGINA PAI  */
        IF PC ¬= 0 THEN RETURN;
        TOPO=0;                    /* INVALIDA POSICIONAMENTO    */
        RC=0;                      /* " UNDERFLOW" OK            */
        PETUPN;                                                  END
PEMOVE;
```

```
ENDFLE:PPOC(PC);       /* FIM  DE  APQUIVO              */

   DECLAPE    RC          FIXED BIN(15); /*  CODIGO DE RETORNO          */

   DECLAPE    VLR_70      POINTER STATIC EXTEPNAL,/*   VALOR 70         */
              P_70        POINTER ,
            1 TRTA        BASED(P_70),   /*  PEGIAO FIXA - ENDER 70     */
            2 P_TRAL      POINTEP,       /*  PTP   TRAL                 */
            2 P_TAF       POINTER;       /*  PTR   TAF                  */

   DECLAPE  1 TAF         BASED(P_TAF) UNAL,  /* TABELA DE APQUIVOS     */
                                              /*       FISICOS          */
            2 JMP_ES(3)   CHAP  ,          /* JUMP PARA ROTINA DE E/S    */
            2 PROX_TAF    POINTER,         /* POINTEP PARA PROXIMA TAF   */
            2 TYPE         BIT(8),         /*  TIPO DA TAF = 19          */
            2 S_IN_FIS    FIXED BIN(15),   /* SET DE IN FISICO-COM DRIVE */
            2 NUM_SET     FIXED BIN(15),   /* NUM DE SETORES             */
            2 STATUS      CHAP  ,          /* STATUS                     */
            2 RAIZ        FIXED BIN(15),   /* SETOR INICIO DO INDICE     */
            2 TAM_CH      FIXED BIN(15),   /* TAMANHO DA CHAVE           */
            2 AVAIL_IND   FIXED BIN(15),   /* PRIM SETOR VAZIO INDICE    */
            2 TOPO        FIXED BIN(15),   /* TOPO DA PILHA DO INDICE    */
            2 PILHA(10)   FIXED BIN(15),   /* PILHA - CAMINHO NO INDICE  */
            2 POS_REG_I   FIXED BIN(15),   /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD    FIXED BIN(15),   /* SETOP INICIO DADOS         */
            2 AVAIL_DAD   FIXED BIN(15),   /* PRIM SETOR VAZIO -DADOS    */
            2 TAM_PEG     FIXED BIN(15),   /* TAMANHO DO REGISTRO        */
            2 POS_SET_D   FIXED BIN(15),   /* POSICION  - SETOR -DADOS   */
            2 POS_REG_D   FIXED BIN(15),   /* POSICION  - REG   -DADOS   */
            2 OPEP_ANT    FIXED BIN(15),   /* ULTIMA OPEPACAO  FEITA     */
            2 P_BUF       POINTER,         /* END DO BUFFER GERAL        */
            2 MAX_UT      FIXED BIN(15); /* MAX BYTE UTILIZAVEL          */

   DECLAPE  1 BUFFEP BASED(P_BUF) UNAL,/*     BUFFER                     */
            2 N_BYTES     FIXED BIN(15), /*     BYTES  OCUPADOS          */
            2 PROX_SET    FIXED BIN(15), /*  ENDER DO PROX SETOR         */
            2 BUF1(508)   CHAP,          /*        RESTO                 */
            2 BUF2(512)   CHAR;          /*  SEGUNDO BUFFEP              */

   DFCLAPE    S_0(8)      FIXED BIN(15) UNAL BASED(P_BUF);/*SETOP 0      */

   DECLAPE  1 ENT_IND     BASED(P_1) UNAL,/*  ENTRADA NO INDICE          */
            2 PTR_E       FIXED BIN(15),  /* ENDER PROX SETOR DE INDICE*/
            2 END_DAD     FIXED BIN(15),  /* ENDER DO SETOR DE DADOS    */
            2 CH_ENT      CHAP  ;         /* CHAVE DA ENTRADA NO INDICE*/

   DECLARE    P_1         POINTER,  /* PTP DO PEG MOVEL DE INDICE       */
              P_2         POINTER,  /* POINTER  DE USO GERAL            */
              (I,J,T)     FIXED BIN(15); /* CONTADORES DE USO GERAL     */
```

```
   DCL         HEX    EXTERNAL ENTPY;    /* IMPRESSAO EM HEXA          */
   DECLARE     TR(5)  BIT(1) EXTERNAL;
 DECLARE     LOG_RW EXTERNAL ENTPY,    /* ROTINA DE LEITURA/GRAVACAO */
             PETRVF  EXTERNAL ENTPY;

       P_70=VLR_70;
     IF TP(3) THEN PUT SKIP(3) LIST('ENDFILE');

       IF OPER_ANT = 2
       THEN DO;                          /* GRAVA ULTIMO SETOR         */
            OPER_ANT=6;
          IF POS_REG_D > 1
          THEN N_BYTES=POS_PEG_D-1; /* COMPLETA INFORMACOES DO SET*/
          PROX_SET=0;                /* MARCA FIM DE ARQUIVO       */
          T=2;
          CALL LOG_RW(POS_SET_D,T,PC); /* GRAVA ULTIMO SETOR      */
          DO I=1 TO TAM_CH;          /* FORMA CHAVE COM FFFF...     */
          UNSPEC(BUF2(I))='11111111'B;/*E ATUALIZA ULTIMO REGISTRO*/
          END;                       /* DE INDICE                  */
          P_1=ADDR(BUF2);
        CALL RETRVE(P_1,J,PC);     /* PROCURA CHAVE FFFF...       */
        IF PC ¬= 0 THEN RETURN;
      P_1=ADDP(BUF1(POS_PEG_I-2)); /*  POSICIONA REG MOVEL    */
      END_DAD=POS_SET_D;         /* ENDER ULTIMO SETOR DE DADOS*/
      T=2;
      CALL LOG_RW(PILHA(TOPO),T,RC); /* REGRAVA SETOR DE IND  */
      IF PC ¬= 0 THEN RETURN;
     END;

       T=1; J=0;
       CALL LOG_RW(J,T,RC);           /* LE  SETOR  ZERO            */
       S_0(1)=PAIZ;                   /* ATUALIZA   INFORMACOES     */
       S_0(3)=AVAIL_IND;
       S_0(4)=S_IN_DAD;
      S_0(5)=AVAIL_DAD;
      J=0; T=2;
      CALL LOG_RW(J,T,PC);           /* PEGPAVA   SETOR   ZERO     */
      RC=0;
END ENDFLE;
```

```
PEWRTE:  PROC(P_ES,RC);        /* ALTERACAO  EM  PEGISTRO                  */

   DECLARE    P_ES       POINTEP,        /* PTP- AREA DE E/S- USUARIO  */
              PC         FIXED BIN(15); /* CODIGO  DE RETORNO         */

   DECLARE  1 BUFFEP     BASED (P_BUF), /* BUFFER  DE  E/S            */
            2 N_BYTES    FIXED BIN(15), /* NUM DE BYTES OCUPADOS      */
            2 PROX_SET   FIXED BIN(15), /* ENDER DO PROX SETOR        */
            2 BUF1(508)  CHAP  ,        /* APEA DISPONIVEL NO BUFFEP  */
            2 BUF2(512)  CHAR;          /* SEGUNDO BUFFER             */

   DECLAPE    VLR_70     POINTEP EXTERNAL, /* ENDERECO FIXO 70         */
              P_70       POINTER ,
            1 TRTA       BASED(P_70),    /*  PEGIAO FIXA - ENDEP 70    */
            2 P_TRAL     POINTER,        /* PTP  DA  TRAL              */
            2 P_TAF      POINTEP;        /* PTP  DA  TAF               */

   DECLAPE  1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE APQUIVOS     */
                                             /*       FISICOS          */
            2 JMP_ES(3)  CHAP  ,         /* JUMP PARA ROTINA DE E/S    */
            2 PROX_TAF   POINTEP,        /* POINTEP PAPA PROXIMA TAF   */
            2 TYPE        BIT(8),        /*  TIPO DA TAF = 19          */
            2 S_IN_FIS   FIXED BIN(15), /* SET DE IN FISICO-COM DPIVE */
            2 NUM_SET    FIXED BIN(15), /* NUM DE SETORES             */
            2 STATUS     CHAP  ,        /* STATUS                     */
            2 PAIZ       FIXED BIN(15), /* SETOP INICIO DO INDICE     */
            2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE           */
            2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE    */
            2 TOPO       FIXED BIN(15), /* TOPO DA PILHA DO INDICE    */
            2 PILHA(10)  FIXED BIN(15), /* PILHA - CAMINHO NO INDICE  */
            2 POS_REG_I  FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
            2 S_IN_DAD   FIXED BIN(15), /* SETOP INICIO DADOS         */
            2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS    */
            2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTPO        */
            2 POS_SET_D  FIXED BIN(15), /* POSICION  - SETOP -DADOS   */
            2 POS_REG_D  FIXED BIN(15), /* POSICION  - REG   -DADOS   */
            2 OPEP_ANT   FIXED BIN(15), /* ULTIMA OPERACAO  FEITA     */
            2 P_BUF      POINTER,       /* END DO BUFFER GERAL        */
            2 MAX_UT     FIXED BIN(15); /* MAX BYTE UTILIZAVEL        */

   DECLARE    P_1        POINTER,       /* POINTEP DE USO GERAL       */
              J          FIXED BIN(15); /* CONTADORES-USO GERAL       */

   DCL        HEX    EXTERNAL ENTRY;   /* IMPRESSAO EM HEXA          */
   DECLAPE    (LOG_RW,MOVE) EXTERNAL ENTRY,
               COMP     EXTEPNAL ENTPY PETUPNS(FIXED BIN(15));
```

```
        /*   I N I C I O   D A   P O T I N A   P E W R T E          */

        P_70=VLP_70;                    /* CAPPEGA ENDER TAF E TRAL   */
        OPEP_ANT=4;

        J=POS_PEG_D - TAM_PEG;          /* POSICIONA NO  ULTIMO       */
        P_1=ADDR(BUF1(J));              /*        REGISTRO            */
        IF COMP(P_ES,P_1,TAM_CH) = 0    /* MESMA  CHAVE               */
        THEN DO;
           CALL MOVE(P_ES,P_1,TAM_PEG); /* MOVE NOVO REGISTPO      */
           J=2;
           CALL LOG_RW(POS_SET_D,J,RC); /* REGRAVA O SETOR         */
           RC=0;
         PETURN;
       END;
       RC=110;                      /*  CHAVE DIFERENTE        */ END
PEWRTE;
```

```
REWIND:  PROC(RC);        /* POSICIONA INICIO DO ARQUIVO                  */

   DECLARE    RC          FIXED BIN(15); /* CODIGO  DE  RETORNO           */

   DECLARE    VLR_70      POINTER EXTERNAL, /* ENDERECO FIXO 70           */
              P_70        POINTER ,
            1 TRTA        BASED(P_70),    /*  REGIAO FIXA - ENDER 70      */
            2 P_TRAL      POINTER,        /* PTR  DA  TRAL                */
            2 P_TAF       POINTER;        /* PTR  DA  TAF                 */

   DECLARE  1 TAF         BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS       */
                                              /*       FISICOS            */
            2 JMP_ES(3)   CHAR  ,           /* JUMP PARA ROTINA DE E/S      */
            2 PROX_TAF    POINTER,          /* POINTER PARA PROXIMA TAF     */
            2 TYPE         BIT(8),          /*  TIPO DA TAF = 19            */
            2 S_IN_FIS    FIXED BIN(15),    /* SET DE IN FISICO-COM DRIVE   */
            2 NUM_SET     FIXED BIN(15),    /* NUM DE SETORES               */
            2 STATUS      CHAR  ,           /* STATUS                       */
            2 RAIZ        FIXED BIN(15),    /* SETOR INICIO DO INDICE       */
            2 TAM_CH      FIXED BIN(15),    /* TAMANHO DA CHAVE             */
            2 AVAIL_IND   FIXED BIN(15),    /* PRIM SETOR VAZIO INDICE      */
            2 TOPO        FIXED BIN(15),    /* TOPO DA PILHA DO INDICE      */
            2 PILHA(10)   FIXED BIN(15),    /* PILHA - CAMINHO NO INDICE    */
            2 POS_REG_I   FIXED BIN(15),    /* POSICIONAMENTO-REG  -INDICE  */
            2 S_IN_DAD    FIXED BIN(15),    /* SETOR INICIO DADOS           */
            2 AVAIL_DAD   FIXED BIN(15),    /* PRIM SETOR VAZIO -DADOS      */
            2 TAM_REG     FIXED BIN(15),    /* TAMANHO DO REGISTRO          */
            2 POS_SET_D   FIXED BIN(15),    /* POSICION  - SETOR -DADOS     */
            2 POS_REG_D   FIXED BIN(15),    /* POSICION  - REG   -DADOS     */
            2 OPER_ANT    FIXED BIN(15),    /* ULTIMA OPERACAO  FEITA       */
            2 P_BUF       POINTER,          /* END DO BUFFER GERAL          */
            2 MAX_UT      FIXED BIN(15);    /* MAX BYTE UTILIZAVEL          */

   DECLARE ENDFLE EXTERNAL ENTRY;

        P_70=VLR_70;                        /* CARREGA ENDER TAF E TRAL     */

        IF OPER_ANT = 2 THEN CALL ENDFLE(RC);
        POS_SET_D=S_IN_DAD;                 /* PRIM SETOR DE DADOS          */
        POS_REG_D=1;

                                            /* PRIM REGISTRO                */
        OPER_ANT=5;
        RC=0;
END REWIND;
```

```
ALLOC:   PROC(END_SET,TIPO,RC);     /* ALOCACAO DE SETOR                  */

        /* ESTA ROTINA ALOCA UM NOVO SETOR COLOCANDO O ENDERECO
           EM END_SET.   SE O TIPO FOR 1 ALOCA NA AREA DE INDICE.
           SE FOR 2 ALOCA NA AREA DE DADOS.
           NAO HAVENDO ESPACO NA AREA DE INDICE PROCURA NA AREA
           DE DADOS                                                    */

    DECLARE    END_SET    FIXED BIN(15), /* ENDER DO SETOR ALOCADO     */
               TIPO       FIXED BIN(15),/*  TIPO DE ALOCACAO           */
               RC         FIXED BIN(15); /* CODIGO DE RETORNO          */
    DECLARE   (I,K)       FIXED BIN(15), /* CONTADOR DE USO GERAL      */
               P_1        POINTER;       /* POINTER  DE USO GERAL      */

   DECLARE     VLR_70     POINTER STATIC EXTERNAL,/*   VALOR 70        */
               P_70       POINTER ,
             1 TRTA       BASED(P_70),   /*  REGIAO FIXA - ENDER 70     */
             2 P_TRAL     POINTER,       /*  PTR   TRAL                 */
             2 P_TAF      POINTER;       /*  PTR   TAF                  */

   DECLARE   1 TAF        BASED(P_TAF) UNAL,  /* TABELA DE ARQUIVOS     */
                                              /*       FISICOS          */
             2 JMP_ES(3)  CHAR  ,          /* JUMP PARA ROTINA DE E/S    */
             2 PROX_TAF   POINTER,         /* POINTER PARA PROXIMA TAF   */
             2 TYPE        BIT(8),         /*  TIPO DA TAF = 19          */
             2 S_IN_FIS   FIXED BIN(15), /* SET DE IN FISICO-COM DRIVE */
             2 NUM_SET    FIXED BIN(15), /* NUM DE SETORES             */
             2 STATUS     CHAR  ,        /* STATUS                     */
             2 RAIZ       FIXED BIN(15), /* SETOR INICIO DO INDICE     */
             2 TAM_CH     FIXED BIN(15), /* TAMANHO DA CHAVE           */
             2 AVAIL_IND  FIXED BIN(15), /* PRIM SETOR VAZIO INDICE    */
             2 TOPO       FIXED BIN(15), /* TOPO DA PILHA DO INDICE    */
             2 PILHA(10)  FIXED BIN(15), /* PILHA - CAMINHO NO INDICE  */
             2 POS_REG_I  FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
             2 S_IN_DAD   FIXED BIN(15), /* SETOR INICIO DADOS         */
             2 AVAIL_DAD  FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS    */
             2 TAM_REG    FIXED BIN(15), /* TAMANHO DO REGISTRO        */
             2 POS_SET_D  FIXED BIN(15), /* POSICION  - SETOR -DADOS   */
             2 POS_REG_D  FIXED BIN(15), /* POSICION  - REG   -DADOS   */
             2 OPER_ANT   FIXED BIN(15), /* ULTIMA OPERACAO  FEITA     */
             2 P_BUF      POINTER,       /* END DO BUFFER GERAL        */
             2 MAX_UT     FIXED BIN(15); /* MAX BYTE UTILIZAVEL        */

    DECLARE 1 TAF_EXT    BASED(P_1) UNAL,
          2 JMP(3)     BIT(8),
          2 PROX       POINTER,       /* ENDER DA PROXIMA TAF       */
          2 TIP        BIT(8),        /*  TIPO  DE TAF              */
          2 SIN        FIXED BIN(15), /*   SETOR DE INICIO COM DRIVE*/
          2 NSET       FIXED BIN(15); /*   NUM DE SETORES           */
```

```
        P_70=VLR_70;                        /*   CARREGA ENDERECO TAF TRAL*/

        IF TIPO=1                           /* ALOCA SETOR NA AREA DE IND */
        THEN IF AVAIL_IND< S_IN_DAD         /* HA ESPACO NA AREA DE INDICE*/
             THEN DO;
                  END_SET=AVAIL_IND;        /* ALOCA  SETOR               */
                  AVAIL_IND=AVAIL_IND+1; /* POSICIONA NO PROXIMO     */
                  RC=0;
                  RETURN;
                  END;

                                          /* ALOCA NA AREA DE DADOS     */
        IF AVAIL_DAD < NUM_SET THEN GOTO ALC_D; /* HA ESPACO-PRIM ARQ*/

        P_1=PROX_TAF;                       /* POSICIONA PRIMEIRA TAF_EXT */
        I=AVAIL_DAD - NUM_SET;
      DO K=1 TO 10 WHILE (P_1 ¬= P_TAF); /* VARRE OUTRAS TAFS      */
      IF I < NSET THEN GOTO ALC_D;   /* HA ESPACO NESTE ARQUIVO    */
                  ELSE DO;
                     P_1=PROX;      /* ENDER DA PROX TAF          */
                     I=I-NSET;      /* DECREMENTA ENDER DO SETOR  */
                     END;
    END;
  RC=107;                          /* NAO HA MAIS ESPACO         */
  RETURN;

 ALC_D:  END_SET=AVAIL_DAD;              /* ALOCA  SETOR NA AREA DADOS */
        AVAIL_DAD=AVAIL_DAD+1;          /* POSICIONA AVAIL NO PROXIMO */
        RC=0;
        RETURN;
END ALLOC;
```

```
LOG_RW:PROC(END_SET,TIPO,RC);      /*   LEITURA LOGICA DE SETOR        */

      /* ESTA POTINA RECEBE EM END_SET UM ENDERECO LOGICO DE SETOR,
         TRANSFORMA EM FISICO E, DE ACORDO COM O TIPO CHAMA A
         ROTINA SDISK PARA LEITURA/GRAVACAO FISICA.
         TIPO 1 - READ          TIPO  2 - WRITE
         SE TIPO FOR NEGATIVO A E/S E NO SEGUNDO BUFFER                */

   DECLARE   END_SET   FIXED BIN(15), /* ENDER DO SETOR ALOCADO      */
             TIPO      FIXED BIN(15),/*  TIPO DE ALOCACAO           */
             RC        FIXED BIN(15); /* CODIGO DE RETORNO          */

  DECLARE    VLR_70    POINTER STATIC EXTERNAL,/*   VALOR 70        */
             P_70      POINTER ,
           1 TRTA      BASED(P_70),   /*  REGIAO FIXA - ENDER 70    */
           2 P_TRAL    POINTER,       /*  PTR   TRAL                */
           2 P_TAF     POINTER;       /*  PTR   TAF                 */

  DECLARE  1 TAF       BASED(P)      UNAL,  /* TABELA DE ARQUIVOS    */
                                            /*      FISICOS          */
           2 JMP_ES(3) CHAR  ,          /* JUMP PARA ROTINA DE E/S   */
           2 PROX_TAF  POINTER,         /* POINTER PARA PROXIMA TAF  */
           2 TYPE       BIT(8),         /*  TIPO DA TAF = 19         */
           2 S_IN_FIS  FIXED BIN(15), /* SET DE IN FISICO-COM DRIVE */
           2 NUM_SET   FIXED BIN(15), /* NUM DE SETORES             */
           2 STATUS    CHAR  ,        /* STATUS                     */
           2 RAIZ      FIXED BIN(15), /* SETOR INICIO DO INDICE     */
           2 TAM_CH    FIXED BIN(15), /* TAMANHO DA CHAVE           */
           2 AVAIL_IND FIXED BIN(15), /* PRIM SETOR VAZIO INDICE    */
           2 TOPO      FIXED BIN(15), /* TOPO DA PILHA DO INDICE    */
           2 PILHA(10) FIXED BIN(15), /* PILHA - CAMINHO NO INDICE  */
           2 POS_REG_I FIXED BIN(15), /* POSICIONAMENTO-REG  -INDICE*/
           2 S_IN_DAD  FIXED BIN(15), /* SETOR INICIO DADOS         */
           2 AVAIL_DAD FIXED BIN(15), /* PRIM SETOR VAZIO -DADOS    */
           2 TAM_REG   FIXED BIN(15), /* TAMANHO DO REGISTRO        */
           2 POS_SET_D FIXED BIN(15), /* POSICION  - SETOR -DADOS   */
           2 POS_REG_D FIXED BIN(15), /* POSICION  - REG   -DADOS   */
           2 OPER_ANT  FIXED BIN(15), /* ULTIMA OPERACAO  FEITA     */
           2 P_BUF     POINTER,       /* END DO BUFFER GERAL        */
           2 MAX_UT    FIXED BIN(15); /* MAX BYTE UTILIZAVEL        */

  DECLARE    BUFFER(1024) CHAR   BASED(P_1),     /*  BUFFER E/S    */
             N_BYTES   FIXED BIN(15) BASED(P_1), /* NUM DE BYTES    */
             END_FIS   FIXED BIN(15), /*  ENDER FISICO COM DRIVE    */
             (I,K)     FIXED BIN(15), /*  CONTADORES  USO  GERAL    */
            (P,P_1)    POINTER,
           REGA       BIT(8) EXTERNAL,/* REGISTRO  A                */
           SDISK      EXTERNAL ENTRY; /*  ROT DE E/S FISICA         */
DCL        HEX    EXTERNAL ENTRY,   /* IMPRESSAO EM HEXA            */
           TR(5) BIT(1) EXTERNAL;
```

```
      P_70=VLP_70;                        /*   CAPPEGA ENDERECO TAF TRAL*/
      P=P_TAF;                            /*              DA  TAF       */
      P_1=P_BUF;                          /*   ENDEPECO   DO  BUFFER    */
      IF TP(3) THEN PUT SKIP(2) LIST('LOG_RW');
      IF TR(2) THEN PUT SKIP DATA(TIPO,END_SET);
    IF TIPO < 0 THEN DO;                  /*  SEGUNDO BUFFER            */
                     TIPO=-TIPO;
                  P_1=ADDR(BUFFER(513));
                  END;

      IF TIPO=2 THEN REGA='00000001'B;    /*   WRITE                    */
                ELSE REGA='00000010'B;    /*   READ                     */

      I=END_SET;
    IF I < NUM_SET THEN GOTO SEGUE;/* ESTA NO PRIM ARQUIVO              */
    I=I-NUM_SET;                        /* DECREMENTA END LOG DO SET   */
    P=PROX_TAF;                         /* PROXIMA TAF                 */
    DO K=1 TO 10 WHILE(P ¬= P_TAF);/* VARRE LISTA DE TAFS              */
    IF I < NUM_SET THEN GOTO SEGUE;/* ESTA NESTE ARQUIVO               */
    I=I-NUM_SET;                        /* DECREMENTA END LOG DO SET   */
    P=PPOX_TAF;                         /* PROXIMA TAF                 */
    END;

      RC=108;                           /* ENDER LOGICO NAO ENCONTRADO*/
      RETURN;

SEGUE:  END_FIS = S_IN_FIS + I;         /* OBTEM END FISICO COM DRIVE */
      I=512;
    CALL SDISK(BUFFER,I,END_FIS);   /*     CHAMA E/S FISICA        */
    RC=REGA;                        /* MOVE STATUS P/ RC           */
   II=N_BYTES+4;  IF TR(5) THEN CALL HEX(P_1,II);
END LOG_RW;
```

```
SDISK:   PROC(BUFFEP,TAM_SET,END_FIS);  /* SIMULACAO DA E/S FISICA     */

    DECLAPE   BUFFER CHAR(512),          /*  BUFFEP  DE  E/S            */
              TAM_SET   FIXED BIN(15), /* TAMANHO DO SETOR              */
              END_FIS   FIXED BIN(15); /* ENDEP FISICO COM DRIVE        */

    DECLAPE   APQ01 FILE EXTEPNAL,
           (NUM_ES,CIL_ANT,TEMPO,SEEKS)
            FIXED BIN(31) EXTERNAL STATIC,
         CIL   FIXED BIN(15),
       REGA      BIT(8) EXTERNAL, /*  PEGISTPADOR   A          */
       KEY       PIC'99999999';


        NUM_ES=NUM_ES+1;
        CIL=END_FIS/24;
      IF CIL ¬= CIL_ANT THEN SEEKS = SEEKS+1;
    TEMPO=TEMPO + 10 + 0.4*ABS(CIL-CIL_ANT) + 20 + 2.8;
  CIL_ANT=CIL;
  KEY=END_FIS;
  IF PEGA='00000001'B
THEN WPITE FILE (APQ01) FROM(BUFFER) KEYFROM(KEY);
ELSE READ  FILE (APQ01) INTO(BUFFER) KEY (KEY);
REGA='0'B;                                                      END
SDISK;
```

```
HEX:  PPOC(P,TAM);  /* LISTA EM HEXADECIMAL   */
   DCL    P      POINTER,
          TAM FIXED BIN(15),
        TABELA CHAR(16) INIT('0123456789ABCDEF'),
        TAB(0:15) CHAR DEF TABELA,
      AREA(2048) BIT(4) BASED(P),
      J      FIXED BIN(4),
    I  FIXED BIN(15);
   PUT SKIP;
 DO I=1 TO TAM*2;
 J=APEA(I);
 PUT  EDIT (TAB(J))  (A(1));
 END;                                                    END HEX;
```