# UM EDITOR DE TEXTOS REENTRANTE, ORIENTADO PARA TELA, COM LINGUAGEM DE COMANDO COMPATÍVEL COM O CANDE B 6700

JEAN-MICHEL NAYRAC

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

Aprovada por:

ESTEVAM GILBERTO DE SIMONE
(Presidente)

PIERRE-JEAN LAVELLE

EDIL SEVERIANO T. FERNANDES

RIO DE JANEIRO, RJ - BRASIL

FEVEREIRO DE 1983

NAYRAC, JEAN-MICHEL

Um Editor de Textos Reentrante, Orientado para Tela, com Linguagem de Comando Compatível com o CANDE B 6700 (Rio de Janeiro) 1983.

ix, 213 p. 29,7 cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1983)

Tese - Univ. Fed. Rio de Janeiro, Coordenação dos Programas de Pós-Graduação em Engenharia.

1. Computadores I. COPPE/UFRJ II. Título (série).

Ã Nilse,

Philippe e Magali

## AGRADECIMENTOS

Ao meu orientador, Prof. Estevam de Simone pela competência, dedicação e constância.

Ao Prof. Pierre-Jean Lavelle pela orientação inicial e valiosas sugestões.

Ao Prof. Edil Severiano T. Fernandes pelo interesse e importante ajuda.

Ao Prof. Carlos A. da Silva Franco e ao colega Miguel Argolo pelo interesse e colaboração.

À minha esposa Nilse pelo incentivo constante e a grande ajuda na elaboração do manuscrito.

# R E S U M O

O presente trabalho descreve o projeto e a implementação de um editor de textos no minicomputador CII - Mitra 15 do Laboratório de Sistemas da COPPE/UFRJ.

O editor proposto é reentrante, orientado para tela ("full screen") e permite atender vídeos com características diferentes.

O sistema deve ser compatível com a linguagem do editor CANDE do Burroughs B 6700, visando minimizar a necessidade de estudo adicional para os usuários, uma vez que nosso editor pretende hospedar-se em sistema estação remota de um B 6700.

A sua linguagem de comando é analisada através do método RRP LL(1) e visando agilizar a preparação de textos, decidimos implementar as funções "full screen" mais usadas através de caracteres de controle.

Os tempos de resposta do sistema são excelentes, principalmente devido ao uso de arquivos seqüenciais indexados.

# RÉSUMÉ

Le présent travail décrit le projet et l'implantation d'un éditeur de textes sur minicalculateur CII - Mitra 15 du "Laboratório de Sistemas da COPPE/UFRJ".

L'éditeur proposé est réentrant, orienté écran ("full screen") et permet d'être utilisé à partir de terminaux video de caractéristiques différentes.

Le système doit être compatible avec le langage de l'éditeur CANDE du Burroughs B 6700, afin de minimiser la nécessité d'étude additionnelle de la part des utilisateurs, sachant que notre éditeur prétend être un élément d'un système de traitement par lots à distance d'un B 6700.

Son langage de commande est analysé par la méthode RRP LL(1) et dans le but d'assouplir et d'accélérer la préparation de textes, nous avons décidé d'implanter, au moyen de caractères de contrôle, les fonctions "full screen" les plus utilisées.

Les temps de réponse du système sont excellents, en particulier grâce à l'utilisation de fichiers séquentiels indexés.

# A B S T R A C T

This thesis describes the project and implementation of a text editor in the minicomputer CII - Mitra 15 of the "Laboratório de Sistemas da COPPE/UFRJ".

The proposed editor is re-entrant, screen oriented, and it is able to handle diferent kinds of terminals in the system.

In order to reduce the user's learning effort we decided to make the system compatible with the CANDE editor language of the Burroughs B 6700 since we intend to use the editor's host machine as a remote job entry station of a B 6700.

The editor's command language is analysed by the RRP LL(1) method and to enhance text production we implemented the more frequent screen functions through control characters.

The system's response time is excellent, mainly due to the use of index - sequential files.

# I N D I C E

# I. INTRODUÇÃO

## I.1 - Situação do Laboratório de Sistemas

Os recursos computacionais do Laboratório do Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ no início desse projeto constavam de:

- um computador de grande porte B 6700, palavra de 48 bits, biprocessador, 2,4 Mbytes de memória, 1,155 Gbytes de disco "on line", 4 drives de fitas magnéticas de 320 Kbytes/s (1600 bpi), 3 impressoras de linha (de 750 a 1800 lpm), 4 leitoras de cartões (de 600 a 1200 cmp), situado no NCE (a aproximadamente 1 km do Laboratório) do qual conseguimos inicialmente uma linha tipo CANDE (BURROUGHS[6]) half duplex, assíncrona, loop de corrente de 20 mA, ponto a ponto.

Historicamente essa primeira linha foi ligada numa teletype em 110 bauds, após termos desenvolvido uma interface loop de corrente de 20 mA - RS232C até 1200 bauds. Pouco depois adquirimos um terminal de vídeo Embracomp TB 110-M que substituiu vantajosamente a teletype citada e foi ligado nessa mesma linha em 1200 bauds. A partir daí, o uso diário dessa linha nos permitiu descobrir toda a potencialidade e universalidade do sistema CANDE do B 6700.

- um minicomputador CII MITRA 15 orientado para tempo real, multiprogramável, palavra de 16 bits, com 32 Kbytes de memória, uma console operadora tipo teletype, leitora de cartões de 300 cpm, uma unidade de disco de 5 Mbytes + 5 Mbytes composta de um disco fixo e de um disco removível, dois relógios

tempo real, 4 linhas assíncronas até 1200 bauds ligadas a 3 teletypes e um terminal de vídeo Iriscope 200 (produto OEM CDC) até 300 bauds.

Essa descrição sucinta deixa pressentir o dilema dos recursos disponíveis: de um lado um computador de grande porte que, por possuir importantes recursos (2 CPU, mais de 2 Mbytes de memória, mais de 1 Gbyte de disco "on line", fitas, impressoras rápidas; entre outros compiladores, PASCAL, ALGOL, FORTRAN) tem alta demanda de serviços e conseqüentemente um tempo de resposta relativamente lento devido a inúmeros usuários, e do outro lado um minicomputador com poucos recursos (pouca memória, sem impressora nem fita magnética), pouca demanda e por conseguinte alta disponibilidade.

Dessa constatação surgiu a idéia principal do projeto: transformar o laboratório de Sistemas num laboratório de Ensino usando o minicomputador Mitra 15 como um processador RJE do B 6700 (BURROUGHS[5]) graças à implantação no Mitra 15 de um software de comunicação compatível com o protocolo de RJE da série B 7000/B 6000 da Burroughs (tese de Mestrado de Maria Cano Mendonza (MENDONZA[30])) e de um editor de textos multiusuário permitindo uma edição e armazenamento locais de arquivos no Mitra 15 (presente trabalho).

Durante o desenvolvimento do projeto o laboratório se enriqueceu com:

- um microcomputador COBRA 300, CPU Intel 8080, palavra de 8 bits, com 48 Kbytes de memória, tela, teclado, 4

unidades de disquete 8 polegadas, simples face, simples densidade (total de 1 Mbytes "on line"), uma impressora 180 cps, 3 linhas de comunicação (assíncrona ou síncrona); compilador LPS, editor de textos "full screen" e software de comunicação RJE com B 6700; sistema operacional monoprogramável.

- uma linha B 6700 tipo CANDE, concentrada, "full duplex", assíncrona, loop de corrente 20mA, em 1200 bauds na qual hoje está ligado o terminal Embracomp TB-110M, a primeira linha conseguida, ponto a ponto, sendo atribuída ao terminal Iriscope 200 do Mitra 15 depois de modificações internas aumentando sua velocidade de 300 para 600 bauds.

- uma linha B 6700 tipo CANDE, multiponto, "half duplex", assíncrona, TDI, protocolo "pull/select" em 2400 bauds ligada a um terminal Embracomp TS 800.

- uma linha B 6700 tipo RJE, "half duplex", assíncrona, loop de corrente 20mA para a qual tivemos que melhorar nossa interface loop de corrente - RS 232 C limitada a 1200 bauds para poder operar em 2400 bauds. Essa linha é a usada no projeto para a ligação do Mitra 15 e também pode ligar o Cobra 300 que possui o software adequado.

## I.2 - Importância do Sistema Proposto

Hoje as 4 linhas do Laboratório de Sistemas ligadas ao B 6700 (3 do tipo CANDE com terminais de vídeo e 1 do tipo RJE com Cobra 300) são usadas diariamente das 8 às 19 horas ou seja 11 horas por dia (às vezes até mais). A procura dos terminais é muito grande e os recursos não atendem às necessidades do

programa de Sistemas que recebe cada ano de 30 a 40 alunos novos entre alunos de mestrado e alunos de doutorado, além de manter um corpo docente de aproximadamente 20 professores. Os alunos necessitam normalmente de recursos computacionais durante a execução de trabalhos de cursos e para muitos durante a implementação da tese (de maneira intermitente durante 3 a 5 anos para um aluno de mestrado).

É então indispensável fornecer mais recursos no laboratório para o uso do B 6700.

Ao encerrar a poucos dias atrás a alocação dos últimos recursos de espaço em disco assim como das últimas das 48 linhas de comunicação disponíveis no B 6700, sem nenhuma perspectiva de expansão futura, sentimos toda a importância do nosso projeto como única solução para atender hoje o crescimento de nossas necessidades de uso de um computador de grande porte.

Além disso esse projeto tem mais dois impactos:

O Cobra 300 por ser um microcomputador nacional de grande divulgação e por possuir uma linguagem atrativa (o LPS) é também muito procurado pelos alunos para a execução de trabalhos de cursos ou tese. O seu grande inconveniente é ser monoprogramável e toda a sua programação deve ser feita no seu único conjunto teclado/tela através do seu próprio editor de textos. Uma ligaçao local assíncrona RS 232 C de alta velocidade (9600 bauds) Cobra 300 - Mitra 15, expansão do nosso projeto inicial, permitirá aos usuários do Cobra preparar seus programas pelo editor de

textos implantado no Mitra e transferir arquivos entre os dois computadores.

O terceiro impacto consiste no aumento das potencialidades do próprio Mitra 15 que num futuro próximo atingirá 64 Kbytes de memória, receberá uma impressora de 600 lpm, 6 terminais de vídeo. O desenvolvimento do projeto já lhe trouxe algumas melhorias como:

- aumento da velocidade do terminal de vídeo Iriscope de 300 para 600 bauds;
- aumento da velocidade das linhas assíncronas de 1200 para 9600 bauds por adjunção de um relógio externo;
- aumento da velocidade da leitora de cartões de 300 para 600 cpm;
- implantação de um novo sistema operacional com gestão de arquivos seqüenciais indexados;
- enfim grande vantagem de ter um editor de textos multiusuário "full screen" que achamos poderoso;

Com todos esses novos recursos, prevemos uma procura crescente desse equipamento nos próximos anos, permitindo uma repartição melhor equilibrada do uso do laboratório.

## I.3 - Importância do Editor e de sua Compatibilidade com CANDE

Não vale a pena discorrer muito sobre as vantagens do uso de terminal de vídeo em substituição à tradicional entrada por cartão perfurado. Isso está descrito amplamente em todos os trabalhos apresentando editores e só resumiremos aqui as duas maiores vantagens ao nosso ver:

- conforto do usuário: pela visualização constante dos caracteres teclados com possibilidade de correção imediata em caso de erro de batida.

- conforto dos responsáveis de CPD, pelo elevado MTBF ("Mean Time Between Failures": tempo médio entre falhas) dos terminais de vídeo (eletrônica simples, bem dominada) contrastando com o muito baixo MTBF de uma perfuradora de cartões (por exemplo, o tipo IBM 029, praticamente a única usada na UFRJ, eletromecânica com funções acionadas por relés).

Uma vez escolhida a implantação de um editor como interface privilegiada entre o usuário e o computador percebemos toda a importância desse sistema por ser o software mais usado dentre todos os outros.

Efetivamente hoje e ainda durante os próximos anos, é confeccionando arquivos de programa e de dados, corrigindo erros de programação que o usuário permanece mais tempo em contato direto com o computador. É a fase mais tensa e cansativa e menos gratificante que terá que enfrentar o usuário no desenvolvimento de seus trabalhos. É quando ele precisará ser atendido

pelo computador com o maior cuidado (rapidez, mensagens explícitas, indicação do tipo e da posição dos erros eventuais, ajuda direta sem manual na aprendizagem da linguagem de comando) e com maiores recursos (funções complexas).

Na psicologia do usuário um tempo de espera superior a 3 segundos para um pedido de serviço comum (listagem, remoção, inserção de caracteres ou de linhas) corresponde a um computador "atolado" e em todo caso não atendendo satisfatoriamente as suas necessidades, aumentando assim sua tensão, nervosismo e cansaço. Tempos de espera superiores podem ser aceitos na medida em que o computador oferece um serviço complexo que o usuário poderia obter somando os efeitos de vários comandos elementares consecutivos. Nesse caso o usuário pode encarar o tempo de espera como um tempo de descanso no qual, graças a um único comando, o computador executa para ele o equivalente a vários comandos mais elementares para os quais teria levado muito mais tempo e dispensado muito mais trabalho. Referenciamos o artigo, relativamente recente, de EMBLEY[23] - NAGY que mostra a influência dos tempos de resposta de um editor sobre a psicologia do usuário e a sua conseqüência nos custos de tarefas de edição.

Foram essas observações principais que guiaram a realização do nosso presente trabalho. Escolhemos compatibilizar o editor o máximo possível com o CANDE do B 6700, tentando melhorar ou inovar certos aspectos, principalmente o tratamento "full screen" dos textos. Essa escolha do CANDE deverá facilitar sobretudo a passagem dos usuários desse sistema para o B 6700 e vice-versa. É bom acrescentar que essa compatibilização não foi para nós uma solução de facilidade, mas bem ao contrário uma fon

te de complicações no projeto, levando-se em conta que o CANDE é um produto da "grande informática" com toda a potencialidade que isso implica. Ainda assim, julgamos absolutamente necessário que os usuários do nosso sistema utilizassem uma linguagem única de comando para edição de textos, obrigando-nos a adotar o CANDE como base para o sistema.

Apesar da especificidade do editor proposto nesse trabalho devido à sua compatibilidade com o CANDE, referenciamos algumas das nossas leituras e em primeiro lugar o excelente artigo de VAN DAN[34]-RICE que foi o ponto de partida em 1971 de numerosos editores posteriores. Referenciamos também trabalhos recentes (ELLIOTT[22], EMBLEY[23], SCOWEN[32], FINSETH[24], FRASER[26]) demonstrando que a edição de texto, que teve um grande desenvolvimento na década passada, é ainda um assunto atual.

# II. OBJETIVOS DO EDITOR

## II.1 - Esboço geral do sistema

Uma estação clássica de RJE da Burroughs da linha B 7000/B 6000 (BURROUGHS[5]) se compõe (Veja figura II.1) de uma console operadora (CS/teletype 110 bauds), de uma impressora de linha (LP) e de uma leitora de cartões (CR). Eventualmente pode constar também de uma perfuradora de cartões "on line" (PC).

Os usuários de tal sistema entram por cartões, descrevendo pelo uso da linguagem de WFL as tarefas desejadas do computador central. Os Jobs assim submetidos são tratados por lotes multifilas.

O sistema proposto (veja figura II.2) se apóia também nessa filosofia permitindo inclusive entrada de Jobs pela leitora de cartões do Mitra 15 mas também - e aí está a grande vantagem do sistema - a partir de arquivos em DISCO elaborados e manipulados pelos terminais graças ao editor de textos multiusuários.

Simulando uma perfuradora de cartões "on line", um arquivo em disco poderá também receber um arquivo alfanumérico ou até binário proveniente do B 6700.

A figura II.2 mostra a composição definitiva do sistema tal qual a planejamos para um futuro próximo.

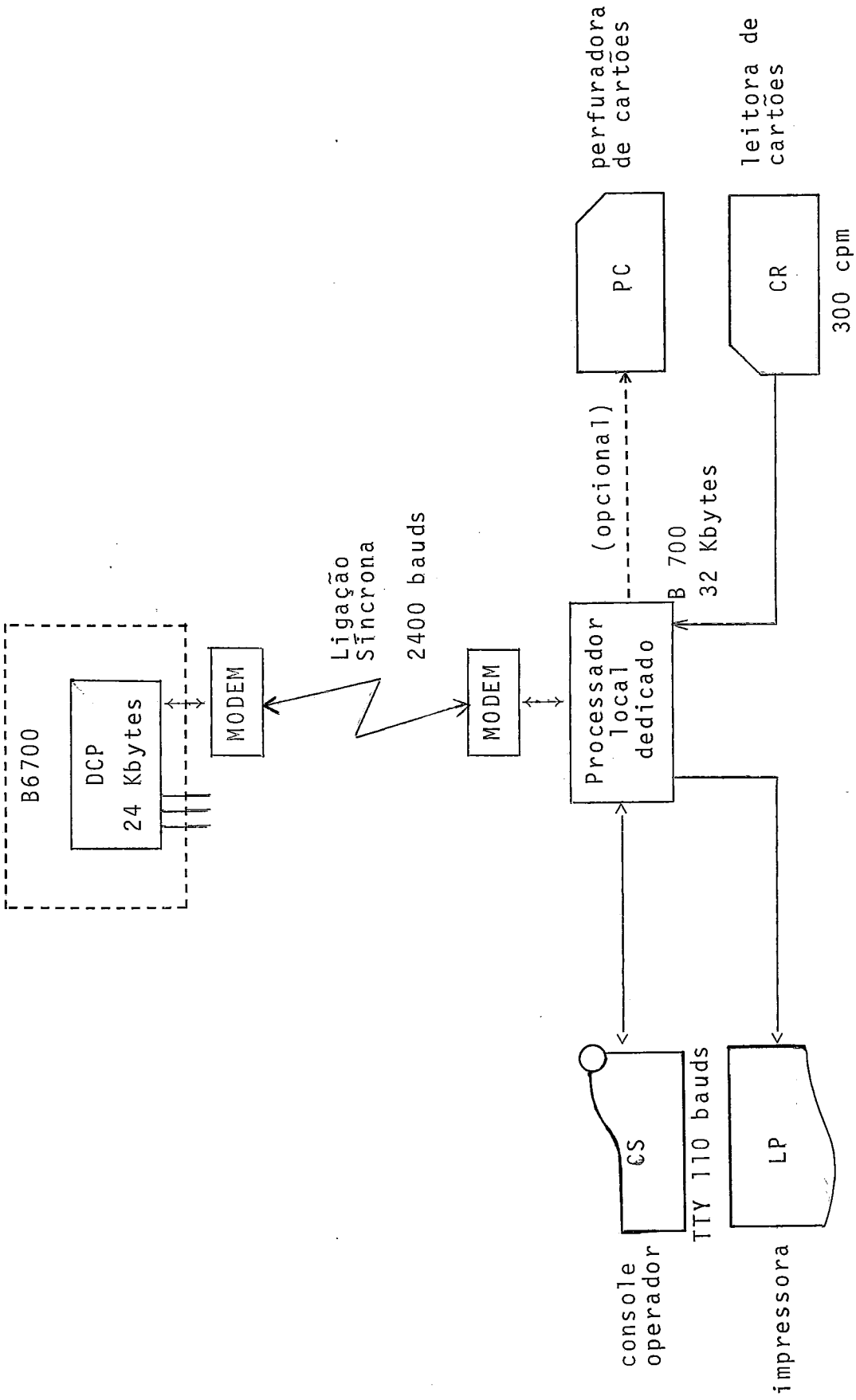



Fig.II-1: Estação RJE clássica da BURROUGHS (os dados técnicos correspondem à configuração implantada no bloco H do Centro de Tecnologia da UFRJ)

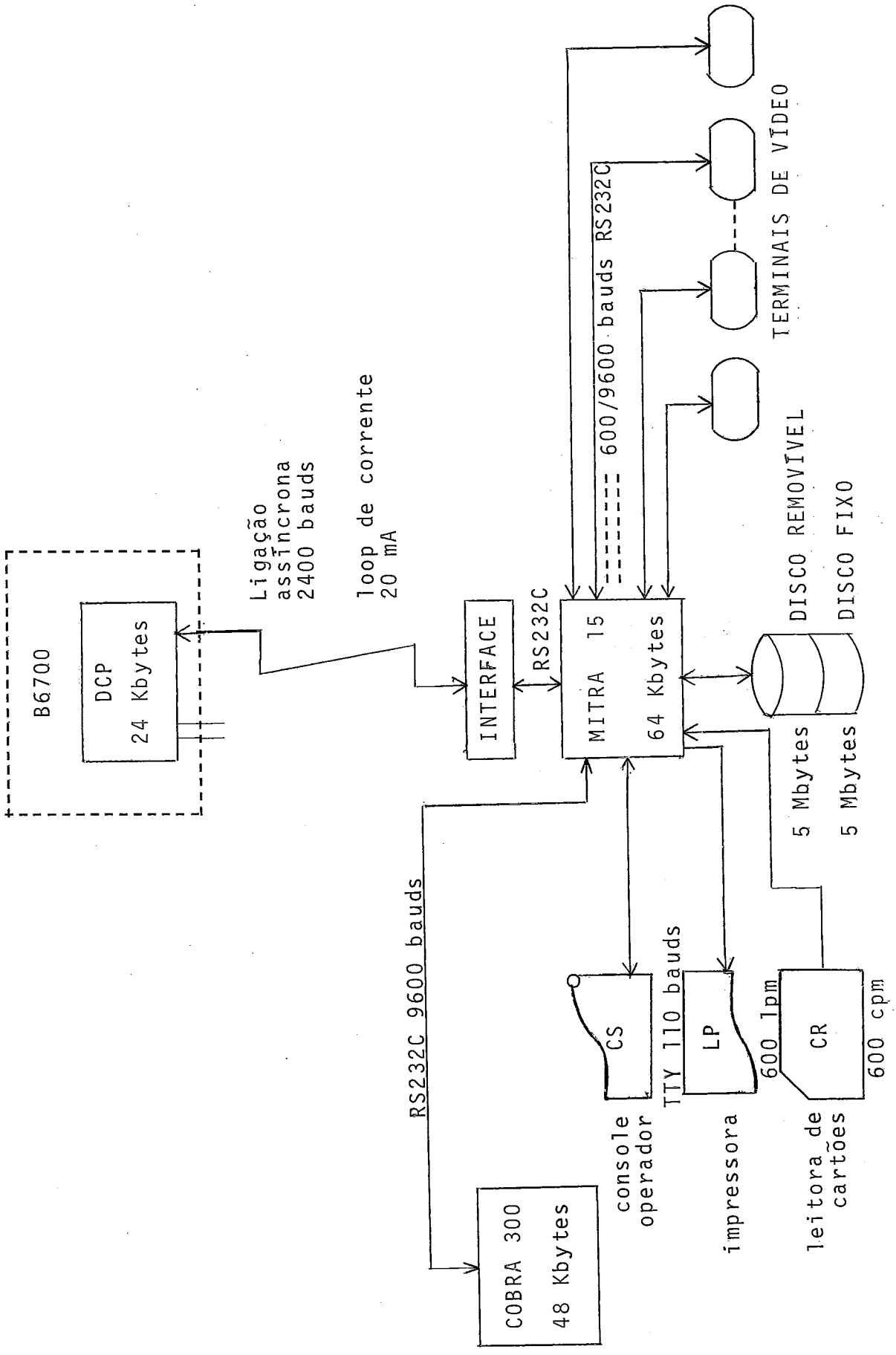



Fig.II.2: Sistema proposto

## II.2 - Escolha de uma Filosofia de Edição

Um editor de textos orientado para a produção de programas, no mínimo, deve permitir:

- criar arquivos
- modificar arquivos
- remover arquivos
- trocar o nome de arquivos
- listar arquivos
- listar biblioteca de usuários

A modificação de arquivos deve ser permitida através de funções de:

- substituição de caracteres dentro de uma linha
- remoção de caracteres dentro de uma linha
- inserção de caracteres dentro de uma linha
- remoção de linha
- inserção de linha

Além desses recursos básicos, que funções mais sofisticadas um editor poderia oferecer aos usuários?

As principais seriam:

- remoção de várias linhas com um só comando

- movimentação de textos dentro do arquivo em edição
- inserção de textos no arquivo em edição provenientes do mesmo ou de outros arquivos
- duplicação de linhas ou de campos de linhas
- listagem de outros arquivos além do que está sendo editado, sem sair da edição
- procura de cadeia de caracteres num trecho de arquivo
- substituição de uma cadeia de caracteres por outra dentro de um trecho de arquivo.

Além da quantidade de recursos oferecidos pesquisamos amplamente a forma de oferecer esses recursos. Sendo nossa meta propor um sistema a implantar e não apenas um modelo de editor mais teórico que prático, analisamos várias realizações implantadas efetivamente e consultamos os seus usuários.

Os comentários destes são às vezes contraditórios e quem não trabalhou um tempo razoável em sistemas diferentes e conhece bem um sistema e pouco outros preferirá o sistema com o qual está acostumado, achando sempre uma função melhor tratada neste ou não existente no outro. Isto quer dizer que qualquer sistema, por suas inovações, será sempre sujeito, no início, a muita crítica.

Assim vimos operar ou nós mesmos operamos aproximadamente 15 editores diferentes hoje usados no Brasil por profissionais da área.

Tais editores são implantados em computadores de grande porte (CANDE da Burroughs num B 6700, XEDIT da CDC num CYBER 750 (CDC[7]) EDIT (IBM[28]), EDGAR (IBM[27]) e INTERACT(OLIVEIRA[31]) em IBM 370), em minicomputadores (editor do COBRA 700 (COBRA[16]), editor do COBRA 530 (COBRA[17]), EDT num PDP 11/70 (DEC[20]), PRETEXTO num PDP 11/10 (FRANÇA[25]), K52 num PDP 11/34 (DEC[19]), editor do MITRA 15 (ALVES[1])) e microcomputadores (editor do COBRA 300 (COBRA[18]) editor do terminal inteligente do NCE (ARAUJO[3]), editor do APPLE II (APPLE[2]), editor do Embracomp SDE 40 (DIGITAL[21]), editores dos Prológica CP 500 e 700).

De todos esses editores se destacam duas formas de tratamento dos textos:

- uma forma orientada para linha (editor de linha)
- uma forma orientada para tela (editor de tela ou "full screen")

É interessante notar que a escolha por um fabricante de uma dessas filosofias é antes de tudo um problema de marketing. Por exemplo os fabricantes de grandes computadores normalmente oferecem editores poderosíssimos mas orientados para linha como é o caso da Burroughs, CDC, CII Honeywell Bull. É possível trabalhar nesses editores com uma teletype em 110 bauds ou uma impressora matricial com teclado, ou também com qualquer terminal de vídeo de qualquer velocidade e tipo, seja do próprio fabricante ou não. Normalmente o terminal proposto pelo fabricante, de sua própria concepção, oferecerá algumas vantagens suplementares de hardware que ajudarão a edição local, tal como substitui-

ção, inserção ou supressão de caracteres, que, acompanhada de uma retransmissão da tela ou de campos da tela, apresentará assim um certo caráter "full screen". É o caso por exemplo do terminal TD-800 da BURROUGHS usado com o CANDE. Isto é um ponto muito importante na aquisição de um computador de grande porte, permitindo assim a recuperação de terminais já à disposição do cliente, possibilitando respeitar também a reserva de mercado em vigor no Brasil. IBM constitui exceção, certamente por possuir a maior porcentagem do mercado mundial, e oferece vários editores "full screen" (EDGAR, XEDIT, INTERACT) que só podem ser usados nos terminais da própria IBM ou em terminais emulando os próprios.

A COBRA oferece editores "full screen" com a mesma restrição de usá-los com um certo tipo de terminal (terminais da Scopus no caso do C700, terminais da própria COBRA no caso da linha C500.

No caso dos microcomputadores que normalmente são monoprogramáveis e então inteiramente dedicados a um só usuário, os editores de qualidade são sempre "full screen". A implantação desses tipos de editores depende estritamente dos recursos de hardware disponíveis e aproveitam toda a memória disponível do microcomputador. A memória de tela faz geralmente parte da memória endereçável diretamente pela CPU, o que permite uma extraordinária velocidade de mudança das informações apresentadas na tela. A velocidade global de edição cai na necessidade de acessar arquivos em disquete.

O editor do sistema operacional CP/M (DIGITAL[21])

o mais comumente usado em microcomputadores, por não querer depender do hardware de um micro particular, é orientado para linha e sua manipulação se torna bem mais incômoda.

Para escolhermos uma dentre as duas filosofias mencionadas, consideramos que hoje em dia não seria astucioso e um pouco arriscado colocar um aparelho eletro-mecânico (teletype, impressora com teclado) nas mãos de muitas pessoas diferentes e isso com uma alta taxa de ocupação (como seria o caso no Laboratório de Sistemas). Levando-se em conta também, além do MTBF baixo, o alto custo desses aparelhos e em contrapartida a boa disponibilidade no mercado interno de terminais de vídeo de excelente qualidade e de alta velocidade (até 9600 e mesmo 19200 bauds), optamos pela filosofia de um editor "full screen", considerando que nosso editor não deverá atender especificações de periféricos eletro-mecânicos.

Por já possuirmos um terminal lento (Iriscope 200 em 600 bauds) e o fabricante do minicomputador utilizado (Mitra 15) não garantir o uso das interfaces de linha assíncrona acima de 1200 bauds (hoje com relógio externo uma linha trabalha em 4800 bauds), desenvolvemos um editor "full screen" com características especiais para atender sem discriminação terminais de vídeo lentos (600 - 1200 bauds) e terminais de vídeo mais rápidos (2400 - 4800 - 9600 bauds). Além disso o editor atende em versão padrão os dois tipos de terminais mais comuns do mercado por simples parametrização.

Essas características a nosso ver são interessantes para o mercado atual pois, apesar de dispor de terminais de vi

deo de bom desempenho, usa comumente para transmissão à distância linhas e modems em 1200 bauds.

A superioridade para nós de um editor "full screen" em relação a um editor orientado para linha (isto poderia ser até uma definição do aspecto "full screen") é poder reaproveitar de forma direta as informações, os textos visualizados na tela do terminal de vídeo graças a recursos internos do terminal de velocidade de ação independente ou pouco dependente da velocidade de transmissão (deslocamento do cursor nos quatro sentidos, HOME/RESET, rolamento da tela (SCROLL), apaga linha a partir do cursor, apaga tela, posiciona cursor no início da linha (CR)).

É esta característica principal que proporcionará ao usuário mais rapidez, eficiência e conforto em relação à outra filosofia.

Aliamos o aspecto "full screen" à poderosa linguagem de edição do CANDE do B 6700, isto para dar toda flexibilidade de operação ao usuário evitando portanto o emprego abusivo de mnemônicos através de caracteres de controle.

Poderíamos também ter escolhido uma seleção de funções por cardápio, mas isso acarretaria o uso somente de terminais rápidos.

Enfim nosso grande interesse na escolha do CANDE é uma compatibilização maior dos recursos oferecidos aos usuários pelo Laboratório de Sistemas.

Esta compatibilização é feita através dos comandos de edição, pois o CANDE não é apenas um editor de textos mas um sistema completo que permite além da edição, principalmente a compilação e execução de jobs "on line" ou assíncrona à sessão, a execução "on line" de task, o uso de comandos de controle mais prioritários que permitem supervisionar os processos ativos, a troca de mensagens entre terminais.

O sistema proposto aqui permitirá apenas, como já vimos, editar e armazenar localmente no Mitra 15 os programas do usuário e submetê-los ao B 6700 entrando numa fila de processamento.

II.3 - Principais Particularidades do Editor

Para dar ao leitor uma visão global do editor listaremos a seguir as suas principais particularidades:

- reentrante, permitindo um atendimento multiusuário, multiconsoles e otimizando o uso da memória;
- atende terminais de vídeo caractere a caractere ligados em linha assíncrona ponto a ponto com protocolo RS 232 C;
- orientado para edição de tela ("full screen") com características especiais para atender com bons compromissos vídeos lentos e vídeos rápidos;
- parametrizado, permitindo atender em versão padrão os dois tipos de vídeo mais comuns (e bara

tos do mercado);

- usa técnica "full duplex" para permitir o controle permanente do cursor nos terminais;

- baseado em arquivos seqüenciais indexados facilitando:

  . numeração intrínseca das linhas;

  . substituição de linhas;

  . inserção de linhas;

  . remoção de linhas;

  . alta velocidade de acessos consecutivos a partes de arquivos distantes;

  . compactação de brancos graças à possibilidade de admitir tamanho de registro variável;

- compatível com o sistema operacional do Mitra 15 (MTRD.74) e organização de arquivos padrão por ele gerenciado;

- uso de um só buffer de uma linha para cada terminal junto com uso de uma tabela de número e status das linhas presentes na tela com a finalidade de economizar memória;

- oferece segurança pelo uso de arquivo de trabalho ("workfile"), cópia automática do arquivo original a editar e arquivo de recuperação ("recovery file") em caso de falha do sistema ou saída anormal de uma sessão;

- número de conta e senha para ter um controle sobre os usuários do sistema e permitir diferenciar arquivos de mesmo nome pertencentes a usuários diferentes;

- três modos de funcionamento: modo edição "full screen", modo entrada de comandos, modo mensagem;

- uso de caracteres de controle (CNTRL / tecla) e caracteres de função (ESCAPE + tecla) em modo edição "full screen";

- analisador de comando compatibilizando os comandos com a linguagem CANDE do B 6700;

- mensagens de erros explícitas;

- em modo entrada de comando, em caso de um eventual erro, posicionamento do cursor sobre o ponto errado detetado pelo analisador;

- memorização dos 10 últimos comandos válidos com possibilidade de reutilização;

- comando de ajuda à aprendizagem das teclas e da linguagem de edição;

- seqüenciamento automático de linhas;

- tabulação horizontal nos dois sentidos por campo e por palavra;

- tabulação vertical;

- opção de indentação automática;

- opção de conversão automática de letras minúsculas em letras maiúsculas;

- possibilidade de depuração na manipulação dos arquivos pelo editor (opção de compilação e uso das chaves do painel do computador);

- escrito em linguagem de montagem ASS2X e LP 15E para minimizar o uso da memória do computador.

Todas essas particularidades do editor serão detalhadas no próximo capítulo.

# III. DESCRIÇÃO DO EDITOR

## III.1 - Operação e Linguagem de Comando do Editor

### Inicialização de uma Sessão de Edição

O editor de textos foi concebido para criar e editar arquivos de programas a partir de terminais de vídeo providos de tela e teclado. A tela dos terminais deve poder conter 80 caracteres por linha, o número total de linhas representadas na tela não constituindo um dado crítico, pois trata-se de um parâmetro ajustável do editor antes da sua compilação.

Após o seu lançamento, o editor deixa aparecer em todos os terminais a ele ligados uma mensagem identificando-o e pedindo conta e senha do usuário. A conta e a senha introduzidas pelo usuário serão procuradas no arquivo "mensagens - número de contas" associado ao editor e caso não sejam encontradas, o usuário será avisado por uma mensagem de "CONTA NÃO AUTORIZADA" e poderá fazer nova tentativa. No caso contrário, ou seja, de conta e senha corretas, uma mensagem de "SESSÃO ABERTA" será apresentada na tela.

A partir daí, o usuário poderá dispor de todos os recursos do editor.

### Os Arquivos do Editor

O usuário terá acesso a todos os arquivos da sua

própria biblioteca identificados pelo sistema por seus nomes e a senha do usuário (número de 1 a 254).

Por razões de segurança o editor não efetua modificações diretamente dentro de um arquivo da biblioteca do usuário. Quando o usuário deseja editar um arquivo já existente, tal arquivo é copiado para um arquivo de trabalho do editor (comando GET) e o arquivo original é preservado intato até o usuário escolher se o arquivo de trabalho irá ou não substituir o arquivo original (comandos SAVE e REMOVE).

No caso de criação de um arquivo novo (comando MAKE) este será montado no arquivo de trabalho e passará ou não a existir na biblioteca a pedido do usuário (comandos SAVE e REMOVE).

Enquanto o arquivo de trabalho for ativo (após os comandos MAKE ou GET e até os comandos REMOVE ou BYE), o nome do arquivo escolhido pelo usuário fica memorizado no arquivo "mensagens - número de conta" associado ao editor, juntamente com a conta e senha correspondentes ao usuário. Em caso de falha no meio de uma sessão, quando o editor for reinicializado ele salvará automaticamente, sob o nome de arquivo RECOVERY (recuperação) na biblioteca de cada usuário, os arquivos de trabalho respectivos então ativos no momento da falha.

No momento em que o usuário vítima da falha abrir uma nova sessão, a leitura do arquivo "mensagem - número de conta" permitirá assinalar a existência de um arquivo RECOVERY que

passará automaticamente a ser o arquivo de trabalho do usuário, após ser emitida a devida mensagem de aviso.

Adotamos, pelo uso de arquivos de trabalho e de recuperação, uma política de edição extremamente segura contra erros de operação e falhas de hardware. O preço pago por isto será um tempo morto para o usuário devido às transferências entre arquivos de trabalho e os da sua biblioteca.

Todos os arquivos manipulados pelo editor são arquivos seqüenciais indexados standard do sistema operacional do Mitra 15 com chave binária de 4 bytes incluída aos registros de tamanho variável entre 4 bytes (linha branca) e 76 bytes (linha com 72 caracteres significativos).

Todas as linhas dos arquivos são numeradas (o número de uma linha corresponde à chave incluída ao registro). É essa numeração que determina a posição lógica das linhas dentro de um arquivo.

Na tela as linhas de um arquivo serão sempre apresentadas com uma numeração de até 6 dígitos na sua frente e permitirão um texto de até 72 caracteres por linha, como veremos com mais detalhe logo a seguir na descrição do "modo texto".

## Os Três Modos de Funcionamento do Editor

O editor trabalha de três modos diferentes correspondentes às três maneiras de apresentação de linhas na tela.

Esses modos, com suas principais funções, são:

- modo comando, que permite digitar um comando do editor;

- modo texto, que permite criar ou editar diretamente na tela linhas de arquivo;

- modo mensagem, que permite ao usuário receber uma mensagem do editor.

O Modo Comando

O editor estará em modo comando, como acontece logo após a abertura de uma sessão, quando o cursor estiver posicionado numa linha na qual aparecerá no campo das colunas 1 a 3 uma seta horizontal. O cursor terá acesso apenas ao campo das colunas 4 a 79 da linha (76 caracteres).

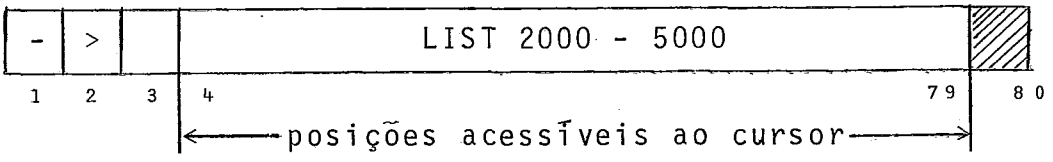



Fig.III-1: Apresentação de uma Linha no MODO COMANDO

Esse modo permite submeter um comando ao analisador do editor. O comando deverá respeitar a gramática da linguagem de edição que será descrita mais adiante. O comando será digitado a partir de qualquer posição permitida ao cursor e submetido ao analisador no momento de a tecla CR (carriage return) ser pressionada. Os dez mais recentes comandos efetivados sem erro

são guardados numa pilha a fim de poderem ser reutilizados posteriormente.

O Modo Texto

O modo texto é caracterizado pela posição do cursor numa linha numerada à frente com até 6 algarismos no campo das colunas 1 a 6. Um espaço em branco (coluna 7) separa este primeiro campo do campo das colunas 8 a 79 que corresponde a um campo de texto a criar ou editar de até 72 caracteres. O cursor manipulado pelo usuário só terá acesso a esse último campo.

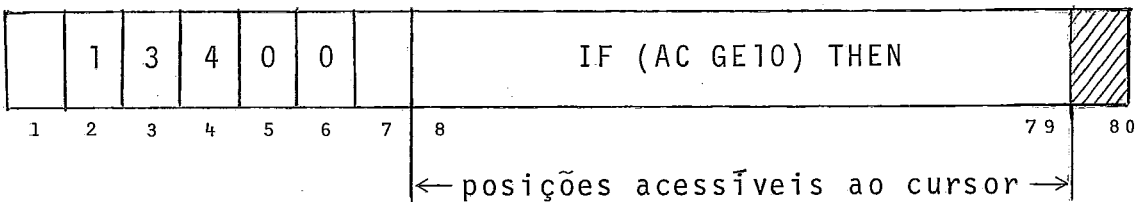



Fig.III.2: Apresentação de uma linha no MODO TEXTO

É nesse modo que o usuário trabalhará mais freqüentemente e foram previstas por isso algumas funções suplementares de edição, certamente as mais usadas durante uma sessão de edição, baseadas no acionamento de teclas de controle.

Para usar estas facilidades, o usuário deverá pressionar simultaneamente a tecla CNTRL (control) e a tecla de uma letra, que foi escolhida para ser a letra inicial de um comando do editor. Em outros casos, o usuário deverá apertar a tecla ESC (escape) e logo depois a tecla de uma letra escolhida também por seu sentido mnemônico. Esses dois tipos de comandos serão descri

tos mais adiante.

É esse modo de funcionamento aliado a esses comandos baseados em caracteres de controle que dão o aspecto "full screen" ao editor.

Por exemplo, após ter pedido uma página de texto através do comando LIST, o usuário poderá deslocar o cursor nos 4 sentidos pelas teclas ←, →, ↑, ↓ até se posicionar na linha desejada a fim de aí introduzir uma modificação, remover caracteres ou inserir um novo texto.

Voltaremos a falar do caráter "full screen" do editor na descrição dos comandos de edição.

A numeração à esquerda do texto, contrária à do tradicional cartão perfurado (numeração à direita do texto), foi escolhida para aumentar a velocidade de listagem na tela, o cursor não precisando assim percorrer os brancos não significativos à direita do texto útil.

No modo texto, assim como nos outros modos, poderemos reparar que a coluna mais à direita de uma linha nunca será acessada para evitar no caso da linha mais inferior da tela um rolamento automático da tela que não pode ser evitado em certos terminais.

O Modo Mensagem

O terceiro modo ou modo mensagem permite ao edi-

tor mandar um aviso ao usuário após ter detetado um erro, ou em resposta a um comando interrogativo, ou no caso de fim de execução de um comando.

Esse modo se caracteriza pelo posicionamento do cursor no final de um texto numa linha não precedida de seta, contrariamente ao modo comando, ou numa linha não numerada, contrariamente ao modo texto.

Em geral, no caso de deteção de erro pelo analisador de comando, a mensagem apresentada é precedida de alguns caracteres "*".

Nos outros casos (erros de execução, mensagens de fins de comando) a mensagem é precedida do caractere "#".

A mensagem apresentada poderá ocupar o campo das colunas 1 a 79.

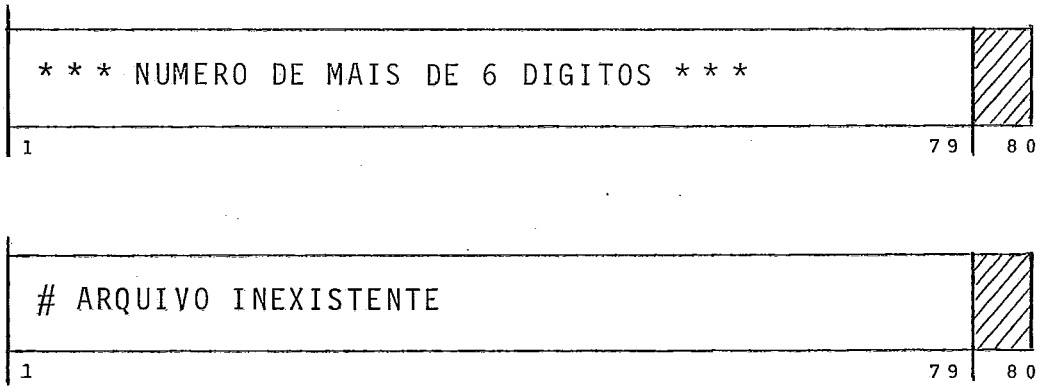



Fig.III-3: Apresentação de linhas no MODO MENSAGEM

Nesse modo o usuário não poderá manipular o cursor e, uma vez captada a informação, apertando-se qualquer tecla

(por exemplo "CR" ou a barra de espacejamento), a mensagem desaparecerá da linha e o editor passará em modo comando (veremos logo a seguir uma exceção no submodo "mensagem página").

Se a mensagem corresponder a um erro detetado pelo analisador de comando (erros de sintaxe ou de semântica) o comando que provocou o erro reaparecerá na linha e o cursor ficará apontando o ponto onde o analisador detetou o erro. Nos outros casos, a linha em modo comando aparecerá vazia.

## Inter-relação entre os Modos; Submodos

Como acabamos de ver na descrição sumária do modo mensagem, os modos não são estanques entre eles. As figuras III.4 e III.5 mostram, graças a diagramas de estados, a inter-relação entre os modos.

Nessas figuras aparecem também o estado correspondente à fase inicial e o estado correspondente à fase de análise dos comandos levando à fase de execução dos comandos que, por sua vez, leva ao modo mensagem ou ao modo texto os quais são divididos em submodos. Devido à subdivisão dos modos e para maior clareza apresentamos dois diagramas do editor em vez de um único diagrama global.

O modo mensagem possui dois submodos. O submodo "mensagem linha" corresponde ao caso de o executivo enviar para o usuário uma mensagem de uma única linha, na própria linha onde foi digitado o comando. Os comandos FILES e HELP por apresenta-

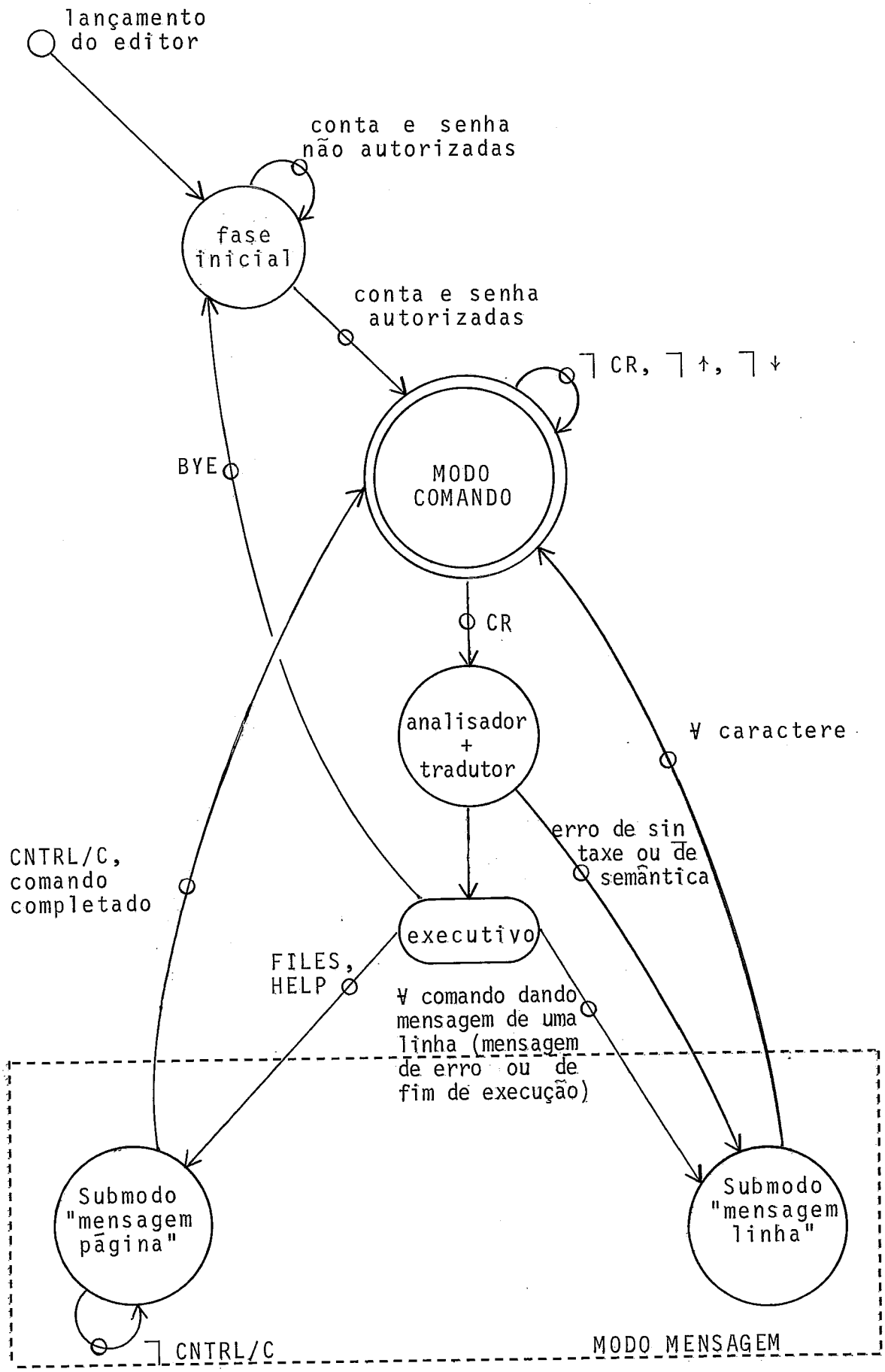



Fig.III-4: Inter-relação entre a fase inicial, o modo comando e o modo mensagem

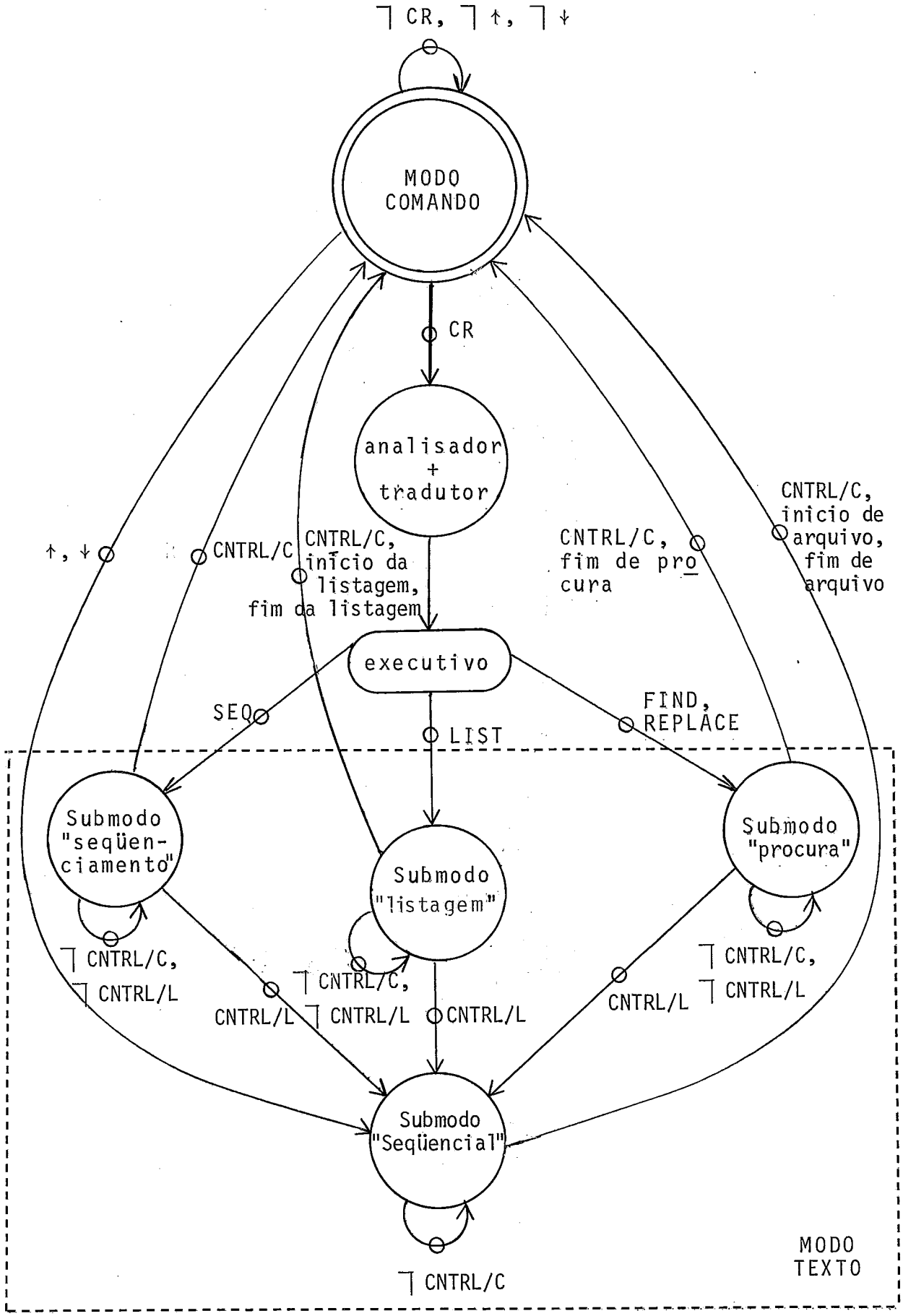



Fig.III-5: Inter-relação entre o modo comando e o modo texto

rem normalmente mensagens de mais de uma linha determinam outro submodo, o submodo "mensagem página" no qual a tela toda é inicialmente apagada. Nesse submodo, se a mensagem não couber inteiramente numa única tela, o cursor ficará esperando, na última linha de mensagem da tela, a digitação de uma tecla qualquer (diferente de CNTRL/C) que permitirá ao executivo prosseguir o comando. A tecla CNTRL/C força a passagem no modo comando e conseqüentemente o abandono do comando. Também o fim da execução do comando provoca automaticamente a passagem em modo comando na linha seguinte à última linha de mensagem apresentada na tela.

O modo texto possui quatro submodos que divergem muito pouco um do outro. Os três submodos "seqüenciamento", "listagem" e "procura" serão descritos em detalhes junto com os respectivos comandos SEQ, LIST e FIND/REPLACE. Pela digitação da tecla CNTRL/L passa-se de qualquer um desses três submodos ao quarto submodo, o submodo "seqüencial". Pode-se também chegar a esse submodo a partir do modo comando pelas teclas ↑ e ↓ que, além de restituirem a linha original do arquivo em edição, movimentam o cursor na direção indicada pelo caractere. Todos esses submodos permitem, como já descrevemos, uma edição "full screen" graças ao deslocamento do cursor e ao uso de caracteres de controle. Os submodos "listagem" e "procura" podem apresentar linhas de arquivo não consecutivas porém em ordem de numeração crescente (considerando-se o sentido de cima para baixo da tela). Após serem inicializados, os submodos "seqüenciamento" e "seqüencial" pelo contrário apresentam sempre linhas de arquivo consecutivas (os textos presentes na tela antes de se entrar nesses submodos podem não ser de linhas consecutivas).

O que diferencia mais os submodos, além de uma elaboração diferente dos textos apresentados na tela, é a ação diferenciada de certos caracteres de controle específicos de edição "full screen". Por exemplo, nos submodos "listagem" e "seqüencial" extremamente semelhantes, o caractere CR no primeiro submodo dará a próxima página da listagem respeitando a lista dos intervalos de linhas especificada no comando LIST, enquanto no segundo submodo dará a próxima página de linhas consecutivas do arquivo. Acontece o mesmo com o comando ESC + CR que no primeiro submodo listará a página anterior respeitando o comando LIST e no segundo submodo, a página anterior de linhas consecutivas. Os tratamentos dos caracteres ↓ na última linha escrita na tela e ↑ na primeira linha escrita da tela são feitos com a mesma diferenciação. A saída automática do sumodo "listagem" para o modo comando ocorre quando a listagem atingir o início ou o fim da lista de intervalos de linhas especificada no camando LIST, enquanto ocorre no submodo "seqüencial" quando a listagem atingir o início ou o fim do arquivo.

A tecla CNTRL/C permite sair de qualquer um dos quatro submodos do modo texto para o modo comando. Como já vimos, a operação inversa, graças aos caracteres ↑ e ↓, conduzirá somente ao submodo "seqüencial". Os outros submodos são acessíveis a partir do modo comando apenas, pelo envio do comando adequado.

A escolha para o editor destes três modos de funcionamento podendo aparecer em qualquer linha da tela corresponde às preocupações de atender, além de vídeos rápidos, vídeos lentos.

Efetivamente não quisemos perder a facilidade de rolamento da tela (scroll) que é uma função interna dos terminais de vídeo e por conseqüência independente da velocidade de comunicação dos terminais.

Se tivéssemos usado duas ou mais linhas, reservadas para entrada de comandos e saída de mensagens, teríamos sido obrigados, para preservar a homogeneidade da tela, a regenerar essas linhas após cada rolamento da tela e reposicionar o cursor no lugar certo do texto em edição. Isto teria implicado uma perda de tempo inversamente proporcional à velocidade de transmissão ou determinado o não uso do rolamento da tela, o que teria degradado o aspecto "full screen" do editor.

## Requisitos Funcionais dos Terminais de Vídeos Usados pelo Editor

Antes de entrar nos detalhes dos comandos do editor seria talvez interessante descrever as características funcionais dos terminais podendo ser conetados ao editor. Isso ajudará certamente o leitor a entender a escolha de certos comandos.

Já dissemos que o editor, e talvez seja esta a sua maior particularidade, foi concebido para poder trabalhar com filosofia "full screen" tanto com terminais de vídeo lentos quanto rápidos, atendendo também aos dois tipos mais comuns do mercado. Entretanto esses devem dispor dos recuros mínimos seguintes:

- 80 colunas por linha

- trasmissão "full duplex"

- comandos especiais para deslocar o cursor nos 4 sentidos. Tais comandos podem ser acionados via computador de duas maneiras: pelo uso de caracteres de controle (por exemplo CNTRL/H para BS [back space]) ou senão do caractere Escape seguido de outro caractere (por exemplo ESCAPE + D para BS)

- scroll up: rolamento da tela para cima

- reset/home: posicionamento do cursor na parte superior esquerda da tela

- line clear: apagamento de uma linha a partir do cursor

- clear: apagamento de toda a tela com posicionamento do cursor idêntico a reset/home

Além desses recursos básicos, os recursos seguintes permitirão um melhor aproveitamento do editor:

- endereçamento direto do cursor

- apagamento da tela a partir do cursor

- scroll down: rolamento da tela para baixo.

Linguagem de Edição

A linguagem de comando do editor foi escolhida para ser a mais compatível possível com a do CANDE do B 6700.

Foram escolhidos 24 dos 65 comandos disponíveis no CANDE e acrescentados 3 comandos novos.

Muitos comandos foram retirados não tendo cabimento no editor proposto por corresponderem a comandos de TASK (só foi escolhido o comando START) ou de utilitários (só foi escolhido o comando WRITE) do B6700.

Outros comandos foram eliminados, como por exemplo o comando FIX e a entrada simples de linha, tendo o editor, pelos seus aspectos "full screen", outros recursos próprios com a mesma finalidade e de uso bem mais fácil e maior potencialidade.

Também alguns parâmetros de comando deixaram de existir, por exemplo o parâmetro FAMILY, sendo os arquivos implantados num único disco do MITRA 15.

Outros mudaram um pouco de sentido, como veremos por exemplo com o parâmetro NEXT.

Muitas dessas mudanças vieram também das características "full screen" do editor devidas à implantação de comandos complementares de edição baseados em caracteres de controle.

Enfim algumas simplificações foram ditadas não para diminuir a complexidade da sintaxe dos comandos, mas para limitar o tamanho do buffer de semântica.

## Diagramas de Sintaxe

Para descrever a sintaxe dos comandos, usamos uma representação por diagramas idêntica à utilizada no manual de referência do CANDE (BURROUGHS[6]), no intuito de facilitar o trabalho dos usuários devendo utilizar os dois sistemas.

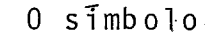
Os diagramas têm por finalidade dar uma representação gráfica que permita facilmente ao usuário deduzir as produções permitidas pela linguagem de comando.

Esses diagramas obedecem a regras muito simples.

São percorridos sempre da esquerda para a direita ou no sentido indicado por uma eventual seta. A continuação de uma linha de diagrama para outra é indicada pelo caractere ">" no fim da linha corrente e no início da próxima linha. O fim de um diagrama é indicado pelo caractere "%". Desmembramentos permitem escolher entre opções, e laços permitem caminhos recorrentes através dos diagramas.

As opções em letras minúsculas e entre os caracteres "<" e ">" são variáveis definidas pelo usuário e descritas logo a seguir, na parte referente à construção de base.

As opções em letras maiúsculas são usadas literalmente e as suas menores abreviações são sublinhadas.

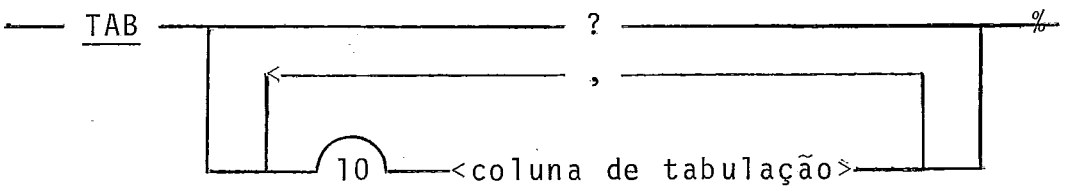
O símbolo  , onde "n" é um número inteiro

positivo e diferente de zero, indica que o diagrama nesse ponto poderá ser percorrido no máximo "n" vezes.

O símbolo —(n*)—, extensão do precedente, indica que o diagrama nesse ponto poderá ser percorrido no máximo "n" vezes e deverá ser percorrido no mínimo uma vez.

A seguir apresentamos um exemplo real de diagrama de sintaxe e algumas produções sintáticas por ele esquematizadas.

Diagrama de Sintaxe




<coluna de tabulação> é um inteiro entre 1 e 72.

Exemplos de Produções Possíveis

TAB?

TAB 50

TAB 8, 16, 32

TAB 5, 9, 13, 17, 21, 25, 29, 33, 37, 41

Construções de Base

As variáveis que o usuário terá que definir mais comumente são as seguintes:

<nome do arquivo> é um nome qualquer começando por uma letra e composto de no máximo 8 caracteres alfanuméricos ou do caractere especial "/". Identifica o nome do arquivo de trabalho ou de um arquivo da biblioteca do usuário.

<tipo> é um dos três tipos possíveis de arquivo SEQ, DATA ou JOB e permitirá ao usuário diferenciar o conteúdo dos seus arquivos. O tipo SEQ será normalmente reservado para arquivos de programa enquanto o tipo DATA, para arquivos de dados. O tipo de arquivo JOB será o único a ser aceito pelo comando START permitindo submeter um job ao computador central.

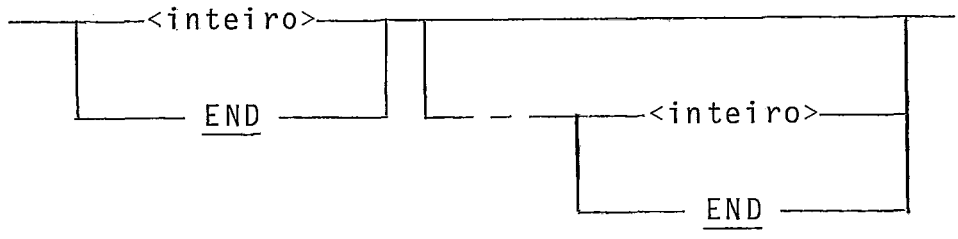
<base> é o primeiro número de uma seqüência de números de linhas. Todos os números de linhas comportam no máximo 6 dígitos significativos.

<inc> é um incremento usado com <base> para gerar uma seqüência de linhas. É composto no máximo de 6 dígitos significativos.

<delimitador> é um caractere especial qualquer, ou seja, um caractere que não é alfanumérico, nem de controle, nem espaço.

<sequence range> é um intervalo de linhas delimitado por seus números de linhas extremos. Pode-se limitar a uma só linha.

Obedece à sintaxe seguinte:



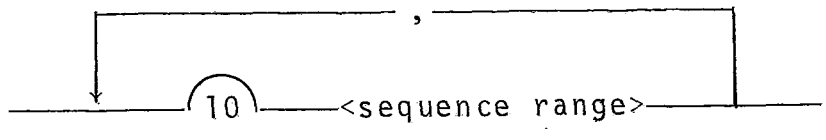

<inteiro> é um número de linha de no máximo 6 dígitos significativos.

Exemplos:

```
1000
END
   0 - 6000
3000 -  END
7000 - 0
END  - 5000
```

<sequence range list> é uma lista de no máximo 10 intervalos de linhas.

Sintaxe:



Exemplo:

100, 6000 - 8000, 10000 - END

Os Comandos

A seguir descreveremos os 27 comandos do editor por ordem alfabética.

BYE

Sintaxe:

—— BYE ——————————————————————————%

Semântica:

O comando BYE fecha a sessão do usuário aberta pela aceitação, na fase inicial, da conta e senha corretas do usuário.

Se o arquivo de trabalho estiver ativo (não salvo e modificado em relação ao arquivo original) será apresentada a mensagem:

# SALVAR OU REMOVER O ARQUIVO DE TRABALHO

e a sessão não será encerrada.

Somente após ter submetido um dos dois comandos SAVE ou REMOVE o usuário poderá submeter de novo o comando BYE a fim de encerrar efetivamente a sessão.

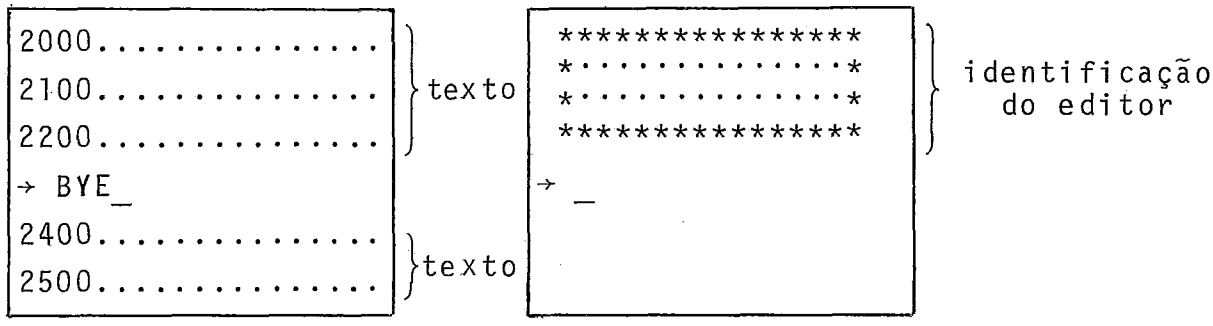
O usuário é avisado do encerramento da sessão pela aparição na tela da mensagem identificando o editor e pedindo conta e senha do usuário.

O comando HELLO do CANDE tem uma função de fechamento de sessão idêntica à do BYE e permite, além disso, introdu

zir conta e senha do usuário.

Por razões de simplificação, não oferecemos o comando HELLO, obrigando o usuário, através do comando BYE, a passar pela fase inicial interativa do editor, o processo de identificação de conta e senha não utilizando o analisador de comandos nesse caso.

Exemplo:

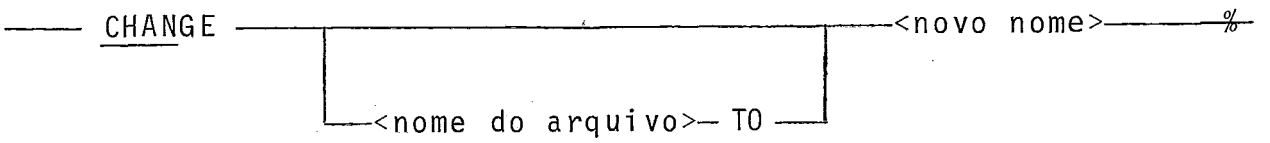



Tela antes da introdução do comando

Tela após a execução do comando

CHANGE

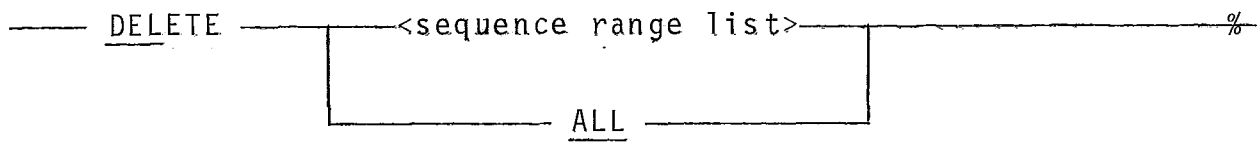
Sintaxe:




CHANGE e TITLE são sinônimos. Referir-se a TITLE

DELETE

Sintaxe:




Semântica:

O comando DELETE permite remover do arquivo de trabalho uma linha ou várias linhas de uma só vez.

<sequence range list> identifica as linhas a serem removidas.

Se ALL for especificado, todas as linhas do arquivo de trabalho serão removidas sem que este deixe de ser ativo.

Após execução completa do comando, o editor passará em modo mensagem na linha onde foi digitado o comando apresentando a seguinte mensagem:

# ATUALIZAÇÕES COMPLETADAS

Serão apagadas da tela as eventuais linhas pertencentes à lista das linhas a serem removidas.

Exemplos:

→ DEL 2000 - 3500, 5000, 8000 - END

→ DELET ALL

FILES

Sintaxe:

—— FILES ——————————————————————————%

Semântica:

Esse comando permite ao usuário listar na tela todos os arquivos da sua biblioteca.

Visualização:

Se o usuário não possuir nenhum arquivo em biblioteca, será avisado pela seguinte mensagem:

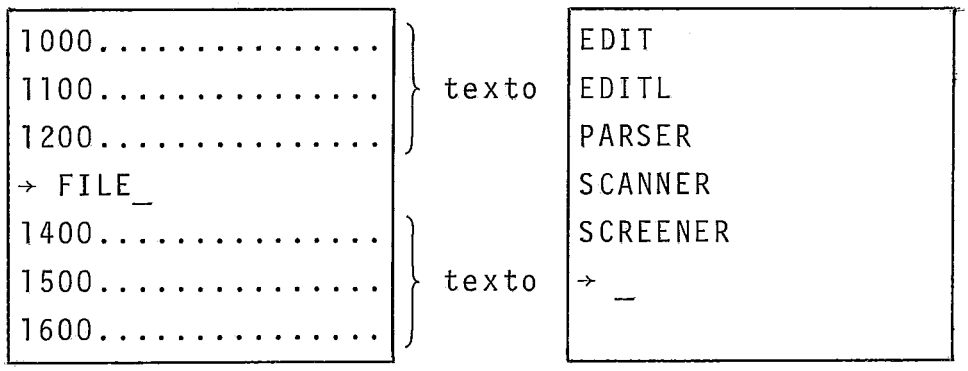
# BIBLIOTECA VAZIA

que aparecerá na linha onde ele entrou o comando.

No caso contrário, após apagamento de toda a tela, os arquivos listados a partir da parte superior da tela.

Se a listagem passar de uma página, na última linha da tela o editor passará em modo "mensagem página" esperando um caractere, qualquer diferente de CNTRL/C, para continuar a listagem. O caractere CNTRL/C permite abandonar o comando, forçando a passagem para o modo comando. Caso contrário, o editor passará em modo comando tendo completado a execução do comando.

Exemplos:

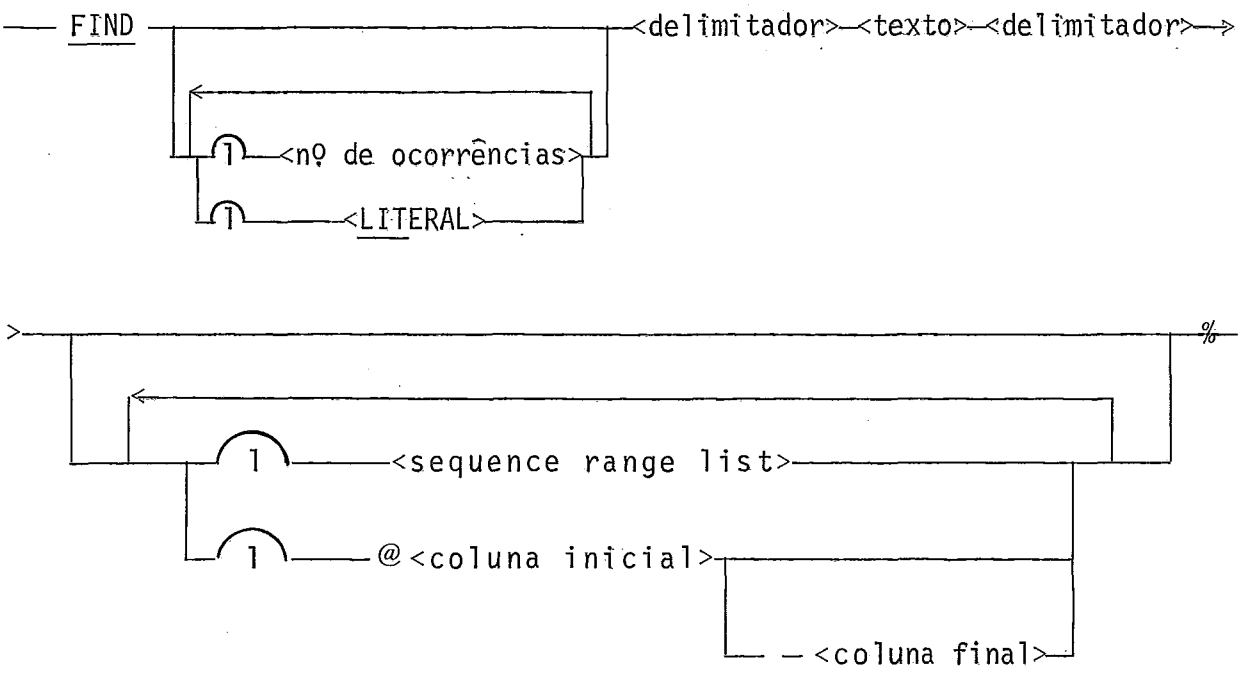



Tela antes da introdução do comando

Tela após a execução do comando

FIND

Sintaxe:




Semântica:

O comando FIND procura no arquivo de trabalho e visualiza as ocorrências de um determinado texto.

<texto> especifica o texto procurado que não deverá conter o delimitador escolhido para caracterizar o início e o fim do texto.

O texto será procurado apenas nos intervalos das linhas especificadas no <sequence range list> se esse parâmetro existir. Caso contrário, a procura se estenderá a todo o arquivo de trabalho.

<coluna inicial> e <coluna final> são inteiros en

tre 1 e 72 que indicam em que intervalo de colunas o texto deverá ser procurado. Se <coluna final> não for especificado, será assumido no lugar a última coluna, ou seja, a coluna 72. Se nenhum desses dois parâmetros for especificado, a pesquisa será feita em toda a linha (das colunas 1 a 72).

<número de ocorrências> é um inteiro entre 1 e 72 que limita o número de ocorrências a serem visualizadas do texto procurado.

LITERAL se for especificado indica que o texto será procurado na sua integralidade, os caracteres em branco do texto sendo significativos. No caso contrário, implícito, a procura será feita em função apenas dos caracteres alfanuméricos e especiais agrupados sem que o número de caracteres em branco, antes ou depois de grupos de caracteres seja significativo. Tal particularidade é muito importante no caso de edição de programas escritos em linguagem com formato livre.

Apresentação:

Se o texto procurado não for encontrado nenhuma vez, aparecerá a seguinte mensagem na linha onde foi digitado o comando:

# TEXTO INEXISTENTE

No caso contrário a tela será apagada e as linhas procuradas, mostradas a partir da primeira linha na tela.

Se o número de linhas encontradas ultrapassar o número total de linhas disponíveis no vídeo, o cursor ficará posi-

cionado na última linha da tela no submodo "procura" do modo texto, permitindo, por deslocamento do cursor, qualquer correção direta na tela (referir-se aos comandos de edição complementar baseados em caracteres de controle).

A digitação do caractere CR permitirá a retomada do algoritmo de procura a partir da linha de digitação de CR e as novas linhas achadas serão apresentadas a partir da primeira linha da tela, após apagamento total da tela.

A digitação da tecla ↓ efetuada na última linha da tela apresentará apenas a próxima linha achada após rolamento da tela para cima.

A digitação da tecla ↑ na primeira linha da tela será sem ação e ecoada pela campainha do terminal.

Em caso de conclusão da pesquisa antes de completar uma página de tela, o editor passará em modo comando. Do modo comando, pela tecla ↑ poderemos passar no submodo "seqüencial" do modo texto a fim de modificar diretamente o texto presente na tela.

Dessa maneira o comando FIND poderá ser usado para listar o arquivo a partir de um dado texto, se for usada a opção <número de ocorrências> igual a 1.

Exemplo:

→ FIND LIT!<5,3/CODE1, TST>!

→ FIND 1/WHILE(CONTINUE NE 0)/2000-5000, 7000-END, @ 1-40

GET

Sintaxe:

—— GET ——————<nome do arquivo>——————————%

Semântica:

O comando GET tem por finalidade transferir um arquivo de nome <nome de arquivo> para o arquivo de trabalho.

Se o arquivo de trabalho estiver ativo, aparecerá a mensagem:

# SALVAR OU REMOVER O ARQUIVO DE TRABALHO

Após providências, o comando deverá ser submetido novamente ao editor.

Se o arquivo não existir na biblioteca do usuário, aparecerá a mensagem:

# ARQUIVO INEXISTENTE

Após transferência aparecerá a mensagem:

# ARQUIVO DE TRABALHO<nome do arquivo>:<tipo>,<N>SETORES,<M>REGISTROS

onde <tipo> é um dos três tipos de arquivos possíveis: SEC, DATA ou JOB, <N> o número de setores ocupados pelo arquivo, <M> o nú-

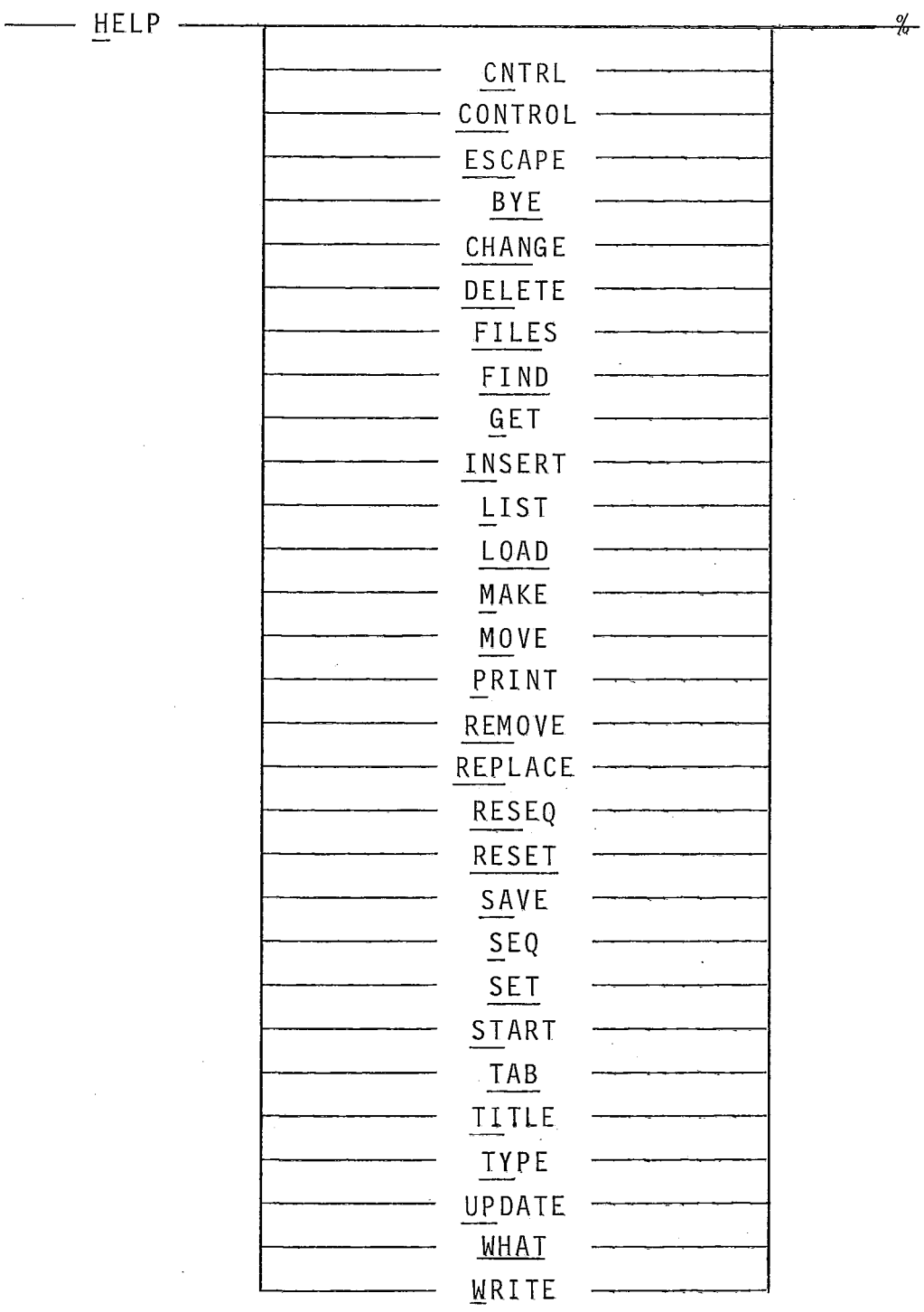
mero total de registros contidos no arquivo.

Exemplo:

→ GET SCREENER

HELP

Sintaxe:




Semântica:

O comando HELP não existe na linguagem do CANDE.

Foi implementado para ajudar o usuário na aprendizagem da linguagem de comandos e evitar o uso de manuais.

O comando HELP, sem parâmetro, apresenta uma lista dos parâmetros possíveis com uma rápida explicação para cada um deles.

O comando HELP associado a um parâmetro dá detalhes de sintaxe e semântica sobre o respectivo comando ou grupo de caracteres complementares de edição (CONTROL e ESCAPE).

As informações são fornecidas por página, o cursor esperando na última linha da tela em modo "mensagem página" no caso de mais de uma página de informações. O fim das informações é assinalado pela passagem em modo comando sob a última linha escrita na tela.

O comando pode ser abandonado pela digitação da tecla CNTRL/C na última linha da tela, o que força a passagem em modo comando. Qualquer outro caractere nas mesmas condições faz prosseguir a execução do comando.

Exemplos:

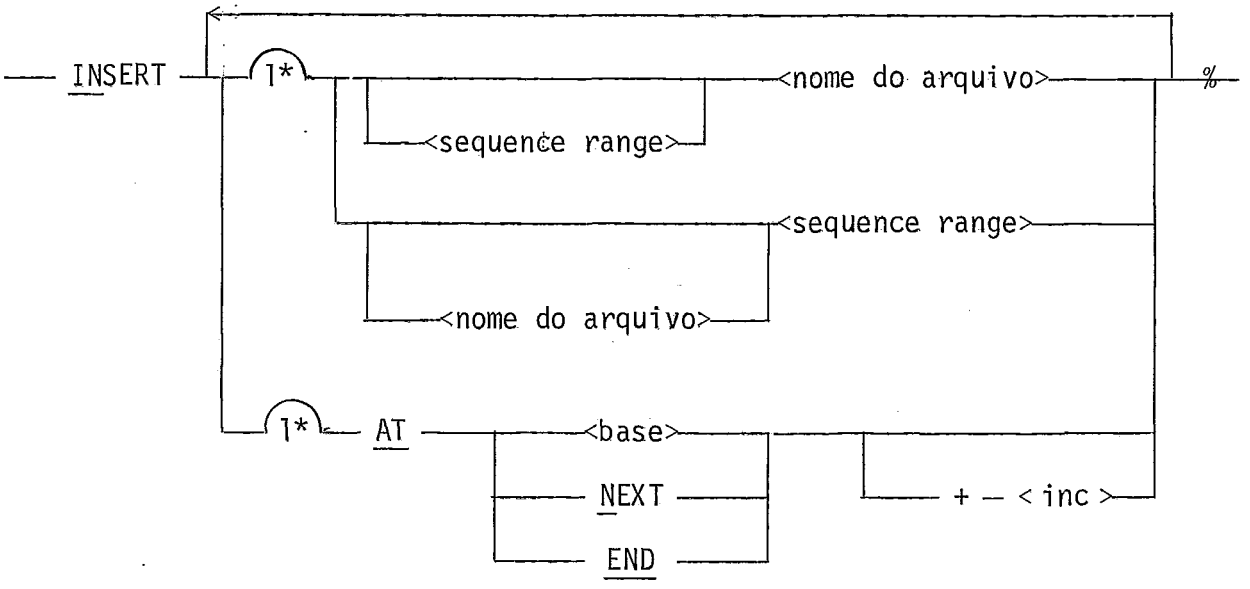
→ HELP

→ HELP ESC

→ HELP WRITE

INSERT

Sintaxe:




Semântica:

O comando INSERT copia linhas de um arquivo da biblioteca do usuário (ou do arquivo de trabalho "por default") para o arquivo de trabalho.

<nome do arquivo> é o nome do arquivo de onde será feita a cópia. Se não for especificado, o próprio arquivo de trabalho será assumido.

<sequence range> especifica o intervalo das linhas a serem copiadas.

<base> indica o número da linha do arquivo de trabalho (obrigatoriamente não existente) a partir da qual será feita a inserção e que por conseqüência dará o número novo da pri-

meira linha inserida.

<inc> é um inteiro que define o incremento usado para determinar os números de seqüência das linhas inseridas. 100 é assumido "por default".

NEXT indica que o número da linha tomada por <base> é o número da linha que precede a em que o comando for digitado, somado ao valor de <inc>.

END indica que o número da linha tomado por <base> é o número da última linha do arquivo somado ao valor de <inc>.

Nenhuma linha do arquivo de trabalho poderá existir entre as linhas a serem inseridas. Essa verificação será feita pelo editor antes de validar o comando.

Em caso de ocorrência será apresentada a mensagem:

# TENTATIVA DE SUPERPOSIÇÃO DE LINHAS; COMANDO REJEITADO

O usuário será avisado do fim correto da execução do comando pela mensagem:

# ATUALIZAÇÕES COMPLETADAS

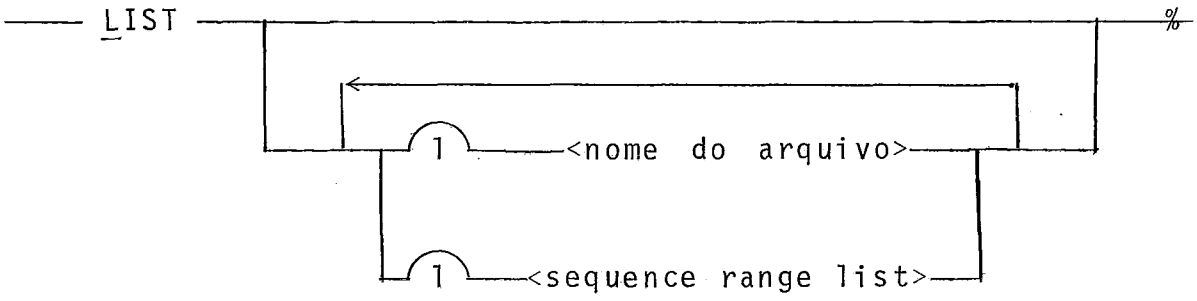
Exemplos:

→ INSERT 10000 - END AT NEXT + 5

→ INSERT OUTROARQ AT END

LIST

Sintaxe:




Semântica:

O comando LIST permite visualizar na tela o conteúdo do arquivo de trabalho ou de outro arquivo da biblioteca do usuário.

<nome do arquivo> indica o nome do arquivo a listar. Se não for especificado, é assumido o arquivo de trabalho.

<sequence range list> especifica a lista dos intervalos (10 no máximo) de linhas a serem listadas. Essa lista deve ser uniforme, ou seja, os intervalos devem ser disjuntos e os números de linhas limitando os intervalos, todos crescentes ou todos decrescentes na sua ordem de aparição no comando. Aqui fizemos uma pequena modificação em relação ao CANDE para facilitar o usuário trabalhando em edição "full screen" e que normalmente listará página por página ou no mínimo por grupo de linhas. Se num intervalo de linhas aparecer um único número de linha, o intervalo considerado consistirá de todo o arquivo a partir dessa linha.

Para listar uma única linha (pouco usado com as facilidades "full screen"), deverá ser indicado duas vezes seguidas o mesmo número de linha.

Se <sequence range list> não for especificado, todo o arquivo será listado.

Outra particularidade em relação ao CANDE consiste em poder listar o arquivo no sentido dos números de linhas decrescentes, se <sequence range list> apresentar seqüências de linhas nessa ordem.

O comando LIST é certamente o mais usado do editor. Os comandos LIST e SEQ apenas, complementados pelo jogo das funções de edição implementadas com caracteres de controle que apresentaremos mais adiante, já permitiriam a constituição de um editor "full screen" completo bastante poderoso.

Apresentação:

Toda listagem começa por um apagamento da tela e as linhas são listadas a partir do canto superior da tela se a seqüência dos intervalos de linhas indicada for crescente.

No caso contrário (ordem decrescente) as linhas são listadas a partir do canto inferior da tela, de baixo para cima.

Se a listagem for maior que uma página, o cursor

pára na última linha (ou primeira se ordem decrescente) no submo do "listagem" do modo texto permitindo, assim, deslocamento do cursor na tela para modificação direta de texto (somente no caso de listagem do arquivo de trabalho).

Se a listagem for inferior a uma página, o editor passa em modo comando na linha seguinte (ou precedente se ordem decrescente) à última linha escrita na tela.

O caractere de controle CR permite pedir a próxima página de listagem sendo que a primeira linha será aquela onde o cursor estava posicionado no envio de CR.

O caractere ESCAPE seguido de CR permite pedir a página anterior de listagem nas mesmas condições.

O caractere ↓ teclado na última linha inferior da tela permite apresentar a próxima linha de listagem, que será a última da tela, após o rolamento desta para cima.

O caractere ↑ teclado na primeira linha da tela permite apresentar a linha de listagem precedente à primeira linha da tela após um rolamento desta para baixo no caso de o terminal o permitir. No caso contrário o caractere ↑ será sem ação e ecoado pela campainha do terminal.

Exemplos:

```
→LIST  ADD100   6000 - 8000, 10000 - 20000, END

→L
```

```
→L 3000          (equivalente a L 3000 - END)

→L 4500-4500 (lista apenas a linha 4500 se existir)

→LIS END-7000(listagem em ordem decrescente)
```

LOAD

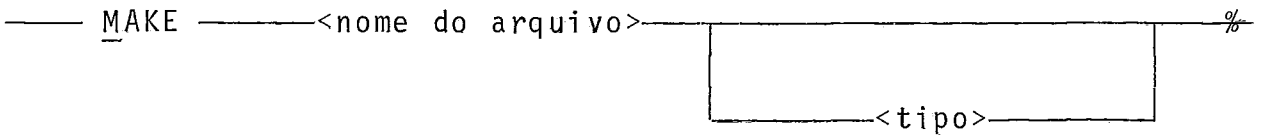
Sintaxe:

——— LOAD ——————<nome do arquivo>——————%

LOAD e GET são sinônimos. Referir-se a GET.

MAKE

Sintaxe:




Semântica:

O comando MAKE cria um novo arquivo de trabalho, se esse for inativo.

No caso contrário é apresentada a mensagem:

# SALVAR OU REMOVER O ARQUIVO DE TRABALHO

<nome do arquivo> é o novo nome. Se este nome já existir na biblioteca do usuário, será apresentada a mensagem:

# ARQUIVO JÁ EXISTENTE

<tipo> é o tipo do novo arquivo escolhido entre SEQ, DATA e JOB. SEQ é o tipo "por default".

O fim do comando correto é assinalado ao usuário pela mensagem:

# ARQUIVO DE TRABALHO <nome do arquivo>:<tipo>

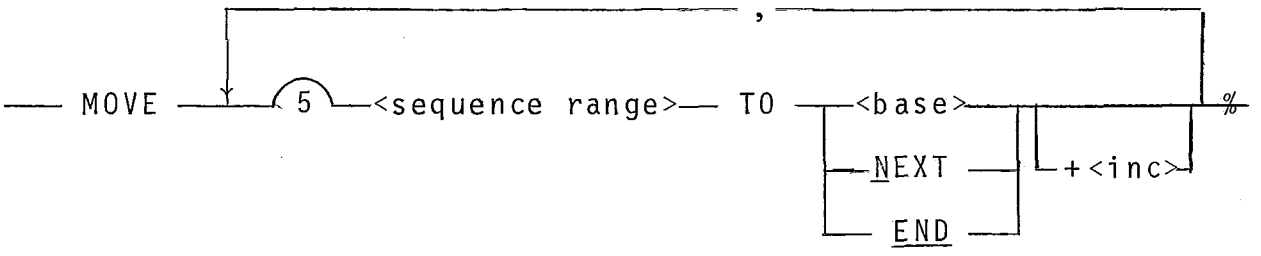
Exemplos:

→ MAKE ARQUIVO1

→ MAKE ARQUIVO2 JOB

MOVE

Sintaxe:




Semântica:

O comando MOVE movimenta uma parte de texto do arquivo de trabalho de um lugar para outro atribuindo às linhas movimentadas um novo número de seqüenciamento.

<sequence range> determina o intervalo das linhas a serem movimentadas.

<base>, NEXT, END e <inc> têm o mesmo sentido que no comando INSERT ao qual pediremos se referir.

Como no caso do comando INSERT nenhuma linha do arquivo de trabalho poderá existir no meio da nova seqüência das linhas movimentadas. No caso contrário será exibida a mensagem:

# TENTATIVA DE SUPERPOSIÇÃO DE LINHAS; COMANDO REJEITADO

O usuário será avisado do fim correto da execução do comando pela mensagem:

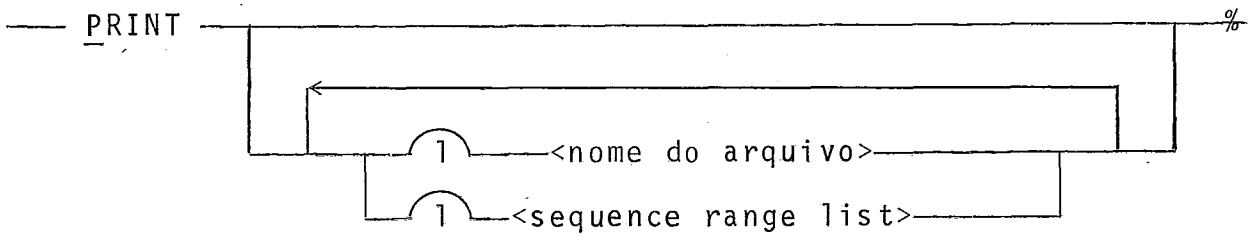
# ATUALIZAÇÕES COMPLETADAS

Exemplo:

→ MOVE 2000-2400 TO 8010 + 10, 0-500 TO NEXT + 20

PRINT

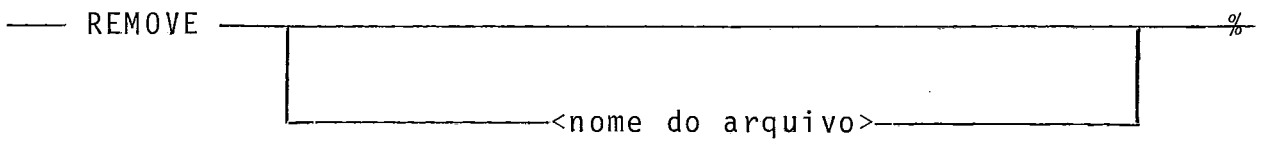
Sintaxe:




PRINT e LIST são sinônimos. Referir-se a LIST.

REMOVE

Sintaxe:




Semântica:

O comando REMOVE permite remover o arquivo de trabalho ou um arquivo da biblioteca do usuário.

<nome do arquivo> indica o nome do arquivo da biblioteca do usuário a remover. Se essa opção for ausente, é assumida a remoção do arquivo de trabalho.

Se <nome do arquivo> não corresponder a nenhum arquivo da biblioteca do usuário, será apresentada a seguinte mensagem:

# ARQUIVO INEXISTENTE

O fim correto da execução do comando é marcado por uma das duas mensagens:

# ARQUIVO DE TRABALHO < nome do arquivo de trabalho>REMOVIDO.

# ARQUIVO < nome do arquivo > REMOVIDO

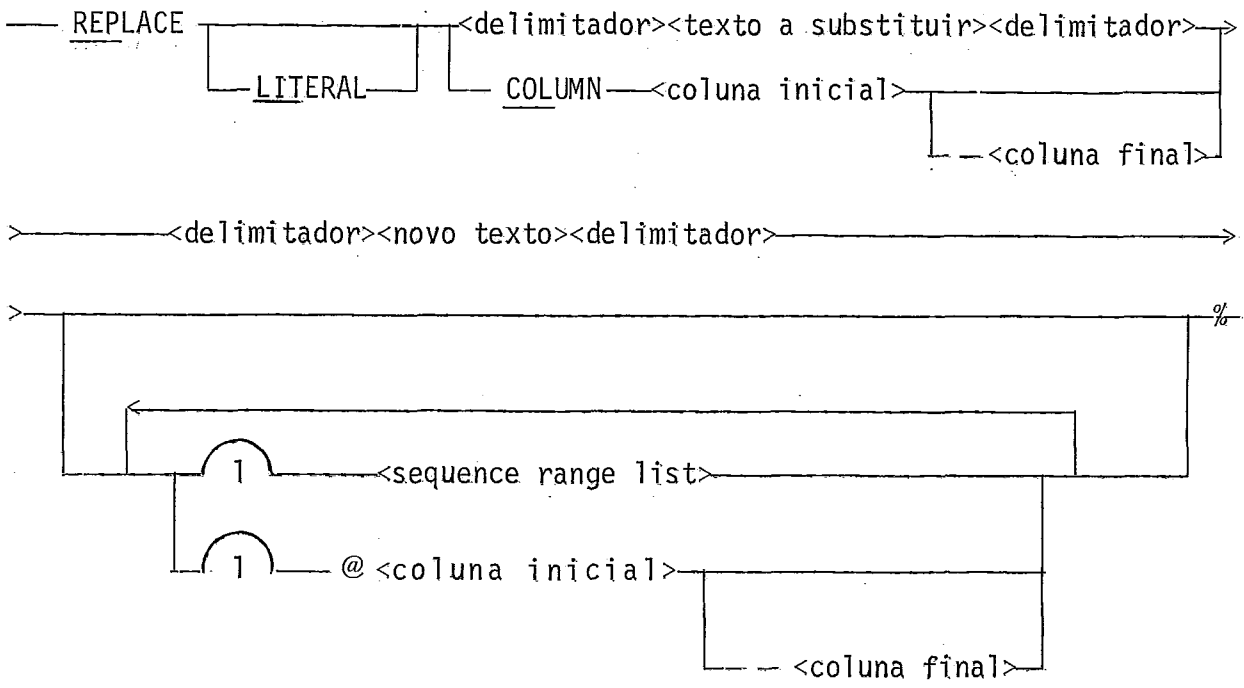
Exemplos:

→ REM

→ REMOVE VELHOARQ

REPLACE

Sintaxe:




Semântica:

O comando REPLACE procura na totalidade ou numa parte do arquivo de trablaho as ocorrências de um determinado texto e o substitui por um novo texto. Pode ter também funções de substituição ou inserção de texto em determinadas colunas.

LITERAL tem o mesmo sentido que no comando FIND ao qual pediremos se referir.

<texto a substituir> é o texto a ser procurado nos limites indicados por <sequence range list>. Não poderá incluir o caractere delimitador.

<novo texto> é o texto que substituirá o texto in

dicado no <texto a substituir>.

COLUMN <coluna inicial>-<coluna final> determina as colunas entre 1 e 72 que deverão ser removidas (inclusive) e substituídas pelo novo texto. Se <coluna final> não for mencionado, o novo texto será inserido logo após a coluna inicial indicada.

@ <coluna inicial>-<coluna final> têm o mesmo sentido que no comando FIND ao qual pedimos se referir.

<novo texto> pode ser de tamanho maior que o texto substituído. A inserção é feita por deslocamento para a direita do texto à direita da inserção. Se o texto resultante ultrapassar a coluna 72, o texto além da coluna 72 será perdido. Nenhum erro será assinalado.

Apresentação:

A apresentação é idêntica à do comando FIND ao qual pedimos se referir.

Exemplos:

Substituição de texto:

→ REP LIT/(A-B)/¦ (A/B)¦ 2000-5000, 8000-END @ 1-20

Supressão do texto:

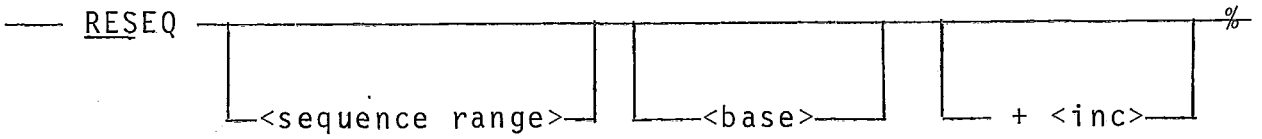
→ REPLACE COL 60-72 / /

Inserção de texto:

→ REP COL 1 / *** / 0-1000

RESEQ

Sintaxe:




Semântica:

O comando RESEQ permite atribuir novos números de seqüência das linhas na totalidade do arquivo de trabalho ou numa parte dele sem modificar a ordem dessas linhas.

<sequence range> indica o intervalo das linhas a serem resseqüenciadas. Se não for indicado, será assumida a totalidade das linhas do arquivo de trabalho.

<base> indica o novo número da primeira linha resseqüenciada. Se esse parâmetro for ausente, será assumido no lugar o número antigo da primeira linha resseqüenciada se <sequence range> for presente. No caso contrário será assumido o valor 100.

<inc> indica o incremento entre as linhas resseqüenciadas. 100 é o valor "default".

Antes de efetuar esse comando, o editor verificará se o intervalo de linhas a resseqüenciar ficará disjunto do resto do arquivo e no caso contrário, será apresentada a mensagem:

# TENTATIVA DE SUPERPOSIÇÃO DE LINHAS; COMANDO REJEITADO

O usuário será avisado do fim correto da execução do comando pela mensagem:

# ATUALIZAÇÕES COMPLETADAS

Exemplo:

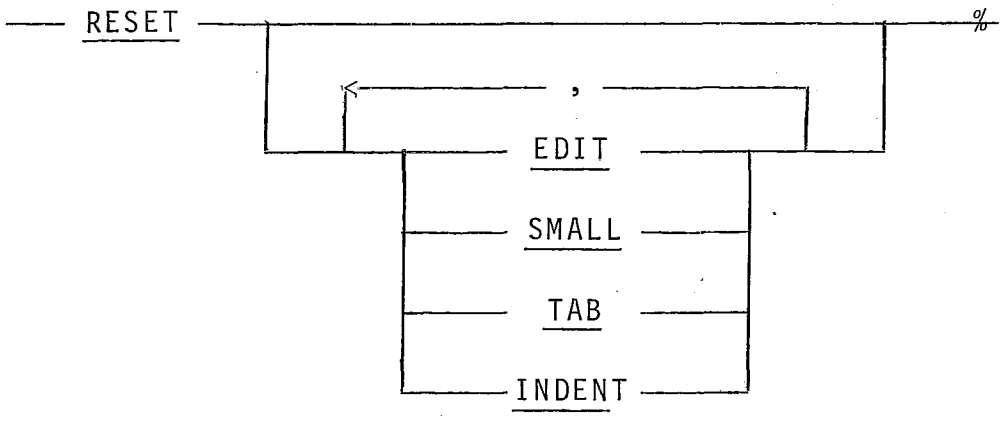
→ RESEQ

→ RES 1000 - 2000 + 10

→ RES 0 - 5000 10 + 10

RESET

Sintaxe:




Semântica:

Os comandos RESET e SET não existem na linguagem do CANDE.

RESET permite eliminar uma ou mais das quatro opções de edição.

A opção EDIT, implícita no lançamento do editor, proíbe toda modificação direta ("full screen") do texto apresentado na tela.

A opção SMALL, implícita no lançamento do editor, provoca a transformação automática de caracteres teclados em letra minúscula para letra maiúscula.

A opção TAB suprime todos os pontos de tabulação de linha com exceção dos pontos correspondentes à primeira (col. 1) e última (col.72) colunas.

A opção INDENT elimina a indentação automática. Por conseqüência, na abertura de uma nova linha de seqüenciamento (no submodo seqüenciamento: ver comando SEQ) o cursor será posicionado sempre na primeira coluna do texto.

O comando submetido sem opção permitirá informar o usuário sobre as opções atuais da edição.

Apresentação:

Sobre a linha onde é digitado o comando aparece a mensagem:

# EDIT:<opção>, SMALL:<opção>, TAB:<opção>, INDENT:<opção>

onde <opção> é a letra S se SET ou a letra R se RESET.

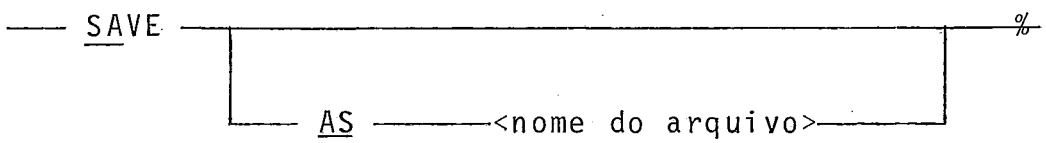
Exemplos:

→ RESET

→ RESET  EDIT, INDENT

SAVE

Sintaxe:




Semântica:

O comando SAVE salva o arquivo de trabalho na biblioteca do usuário.

Sem opção, se o arquivo já existir, ele será removido e substituído pelo arquivo de trabalho. Aparecerá a mensagem:

# ARQUIVO DE TRABALHO<nome do arquivo>SALVO;ANTIGO FONTE REMOVIDO

Ainda sem opção, se o arquivo não existir, ele será criado e aparecerá a mensagem:

# ARQUIVO DE TRABALHO<nome do arquivo>SALVO

Com a opção AS <nome do arquivo> o arquivo é salvo sob um nome que não será seu nome original. Uma verificação sobre o <nome do arquivo> é feita para saber se ele não existe já na biblioteca do usuário. Se for o caso, será apresentada a mensagem:

# ARQUIVO DE TRABALHO<nome do arquivo>SALVO SOB O NOME<novo nome>

No caso contrário aparecerá a seguinte mensagem:

# ARQUIVO JÁ EXISTENTE

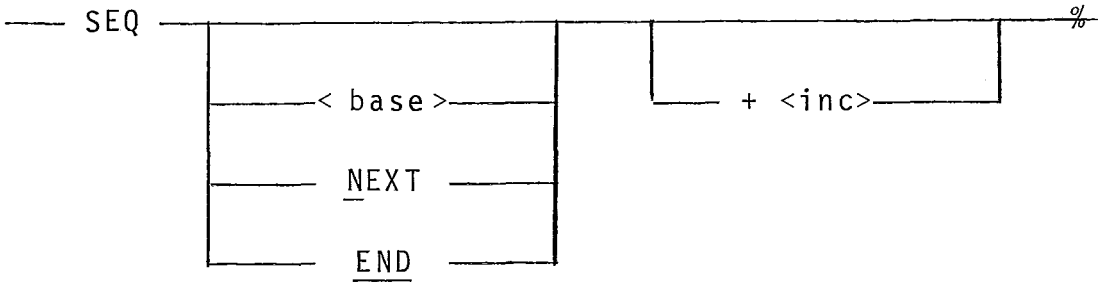
Exemplo:

→ SAVE

→ SA AS NOVONOME

SEQ

Sintaxe:




Semântica:

O comando SEQ é o comando que permite inserir ou criar novas linhas no arquivo de trabalho graças à geração automática pelo editor de seqüências de linhas.

<base> é o número da linha inicial da seqüência de linhas.

NEXT indica que a linha inicial é a linha que precede a linha do cursor.

END indica que a linha inicial é a última linha do arquivo de trabalho.

Se <base>, NEXT ou END não forem especificados, será assumido o número 100.

<inc> é o incremento determinando a seqüência das linhas geradas. Se esse parâmetro for omitido, será assumido o

valor 100.

O comando SEQ sem parâmetro significará que a linha inicial será a linha 100 e o incremento também 100.

Apresentação:

O comando SEQ posiciona o editor no submodo "seqüenciamento" do modo texto.

Quando nenhum parâmetro for precisado ou quando usados os parâmetros <base> ou END a tela será apagada e a primeira linha seqüenciada aparecerá na posição da primeira linha da tela. Se essa primeira linha não existir, aparecerá apenas o número de seqüência dessa nova linha e o cursor ficará posicionado na primeira coluna de texto, esperando a digitação da linha. Digitada a linha, os caracteres CR, LF ou ↓ darão a seqüência da próxima linha. Se nenhum caractere for digitado na linha, a próxima seqüência aparecerá sobre a mesma linha.

Para entrar uma linha branca será necessário teclar no mínimo um espaço em branco na linha.

Quando uma nova seqüência de linhas for gerada pelo editor, o cursor será posicionado sistematicamente na primeira coluna, se a opção INDENT for eliminada. No caso de essa opção ser ligada, o cursor será posicionado na primeira coluna do texto útil (considerando-se não significativos os brancos à esquerda do texto) da última linha inserida ou modificada.

Se a primeira linha seqüenciada existir, essa será apresentada na tela e o cursor ficará esperando no final do texto dessa linha uma eventual modificação da linha. Como no caso precedente CR, LF ou ↓ darão a próxima seqüência de linha.

Se uma seqüência de linha já existir ou se houver outras linhas do arquivo entre duas seqüências, essas linhas serão apresentadas na tela. Tal possibilidade é uma vantagem significativa sobre o CANDE que não a permite.

No caso de utilização do parâmetro NEXT, toda a tela a partir da linha do cursor (inclusive) será apagada e aparecerá na linha de introdução do comando a primeira linha de seqüência ou uma linha intermediária, se houver.

Nesse submodo "seqüenciamento" o cursor pode ser deslocado em todos os sentidos para modificar qualquer linha já na tela, qualquer tipo de edição "full screen" sendo permitido.

Exemplo:

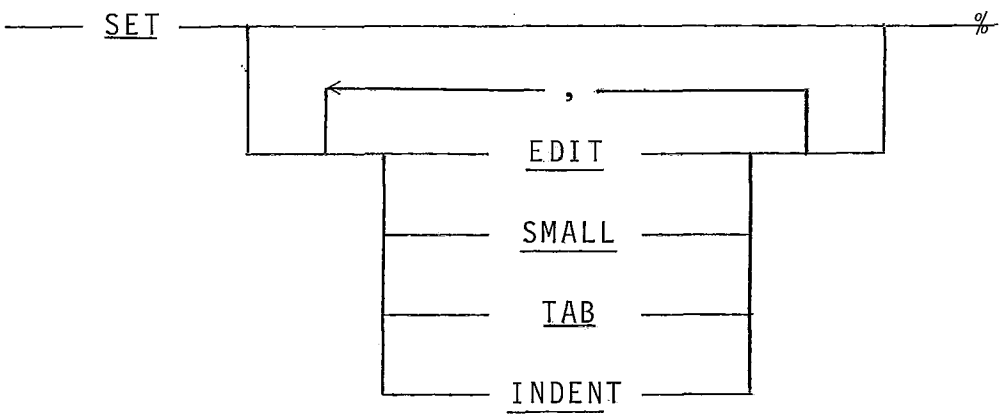
```
→ SEQ

→ S  END + 100

→ SEQ NEXT + 10
```

SET

Sintaxe:




Semântica:

Os comandos SET e RESET não existem na linguagem do CANDE.

SET permite ligar uma ou várias das quatro opções de edição.

A opção EDIT autoriza modificações "full screen" da tela se os textos aí presentes pertencerem ao arquivo de trabalho.

A opção SMALL autoriza a digitação de caracteres minúsculos que deixam de ser automaticamente transformados para caracteres maiúsculos.

A opção TAB substitui todos os pontos de tabulação existentes pelos 10 standard do editor (colunas 1 - 9 - 17 - 25 33 - 41 - 49 - 57 - 65 - 72) implícitos na abertura de uma sessão.

A opção INDENT liga a indentação automática.

Conseqüentemente, na abertura de uma nova linha de seqüenciamento (no submodo seqüenciamento: ver comando SEQ) o cursor será posicionado na primeira coluna de texto útil (considerando-se não significativos os brancos à esquerda do texto) da última linha inserida ou modificada.

O comando inserido sem opção permitirá informar o usuário sobre as opções atuais da edição.

Apresentação:

Idêntica ao comando RESET ao qual pedimos se referir.

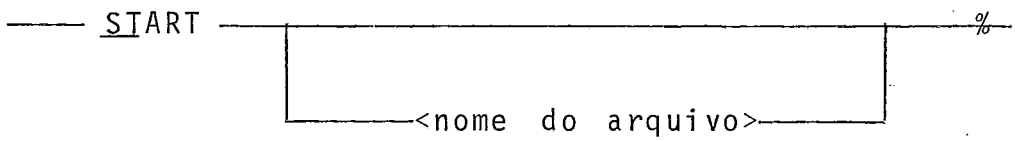
Exemplos:

→ SET

→ SET EDIT, TAB, INDENT

START

Sintaxe:




Semântica:

O comando START permite submeter um job ao computador central através da ligação tipo RJE Mitra 15 - B6700. O job entrará numa fila de processamento do B6700.

<nome do arquivo> é o nome do arquivo a ser enviado. Se esse parâmetro não for indicado, será assumido no lugar o arquivo de trabalho.

O arquivo enviado deverá ser do tipo JOB senão o comando será recusado e aparecerá a mensagem:

# O ARQUIVO NÃO É DO TIPO JOB; COMANDO CANCELADO

Após o fim da operação correta, será apresentada a mensagem:

# ARQUIVO<nome do arquivo>NA FILA DE TRANSMISSÃO

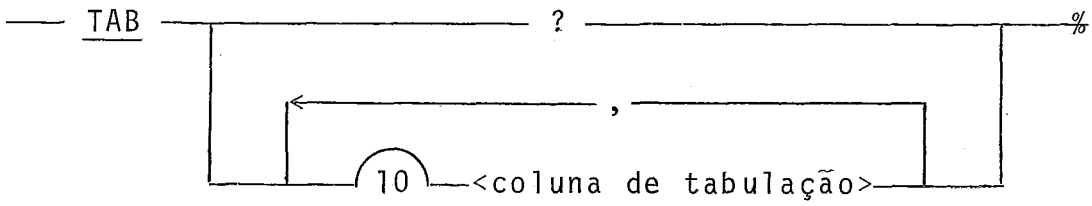
Exemplos:

→ START

→ ST TESE/INI

TAB

Sintaxe:




Semântica:

O comando TAB pela opção "?" permite visualizar qual é a posição dos pontos de tabulação horizontal.

Com a opção <coluna de tabulação> que é um inteiro entre 1 e 72 o comando TAB permite posicionar até 10 pontos de tabulação além dos dois fixos correspondentes às colunas 1 e 72.

Um comando TAB com opção <coluna de tabulação> elimina todos os antigos pontos de tabulação para validar somente os indicados no comando.

Os pontos de tabulação também podem ser manipulados pelos comandos SET TAB e RESET TAB e pelos comandos complementares CNTRL/T e ESCAPE + T aos quais pedimos se referir.

A mensagem emitida é sempre do seguinte tipo:

# PONTOS DE TABULAÇÃO NAS COLUNAS: 1,...,...,...,72

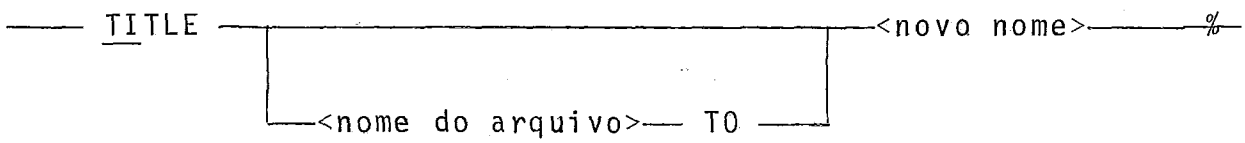
Exemplos:

→ TAB ?

→ TAB 8, 16, 32

TITLE

Sintaxe:




Semântica:

O comando TITLE (sinônimo do comando CHANGE) permite mudar o nome do arquivo de trabalho ou de qualquer arquivo da biblioteca do usuário.

<nome do arquivo> indica o nome do arquivo da biblioteca que terá o nome alterado. Se tal parâmetro não for especificado, será assumida a alteração do nome do arquivo de trabalho.

<novo nome> é o novo nome atribuído ao arquivo.

Resposta do Editor:

Se <nome do arquivo> especificar um arquivo inexistente, será apresentada a mensagem:

# ARQUIVO<nome do arquivo>INEXISTENTE; COMANDO CANCELADO

Se <novo nome> especificar um arquivo já existente, será apresentada a mensagem:

# ARQUIVO<novo nome>JÁ EXISTENTE; COMANDO CANCELADO

Após encerramento do comando, o usuário é avisado por uma das duas mensagens:

# ARQUIVO<nome do arquivo>MUDADO PARA<novo nome>

# ARQUIVO DE TRABALHO<nome do arquivo>MUDADO PARA<novo nome>

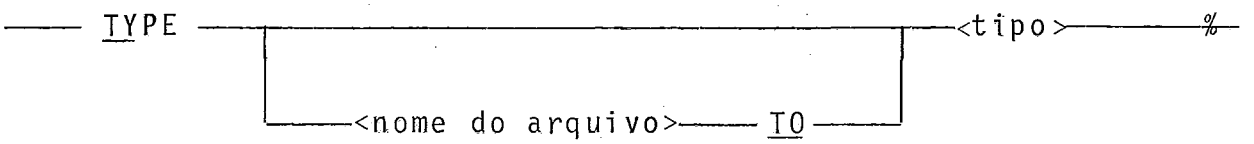
Exemplos:

→ TITLE SCREEN TO SCREENER

→ TI SUPER/1

TYPE

Sintaxe:




Semântica:

O comando TYPE permite mudar o tipo do arquivo de trabalho ou de qualquer arquivo da biblioteca do usuário.

<nome do arquivo> especifica o nome do arquivo da biblioteca do usuário que terá seu tipo mudado.

Se esse parâmetro não for especificado, será assumido no lugar o arquivo de trabalho.

<tipo> é um dos três tipos SEQ, DATA ou JOB.

Completado o comando, o usuário é avisado pela mensagem:

# TIPO TROCADO

Exemplo:

→ TYPE JOB

→ TY BN/DC TO SEQ

UPDATE

Sintaxe:


Semântica:

O comando UPDATE reestrutura o arquivo de trabalho para permitir uma melhor utilização do seu espaço em disco.

O usuário poderá recorrer a esse comando quando sentir atrasos em listagens devidos a remoções importantes de linhas consecutivas.

No fim da execução desse comando, o usuário será avisado pela mensagem:

# FIM DE REESTRUTURAÇÃO DO ARQUIVO DE TRABALHO

Exemplo:

→ UPDATE

WHAT

Sintaxe:

—— WHAT ——————————————————————————%

Semântica:

O comando WHAT permite ao usuário ter as seguintes informações sobre o arquivo de trabalho:

1. nome
2. tipo
3. número total de linhas
4. número da última linha
5. status (salvo ou não na biblioteca do usuário)

Apresentação:

A mensagem recebida pelo usuário é a seguinte:

#ARQUIVO DE TRABALHO<nome>:<tipo>,<n>LINHAS,ATÉ LINHA<m>,<status>

onde:

<nome> é o nome do arquivo de trabalho (1 a 8 caracteres)

<tipo> é um dos três tipos SEQ, DATA ou JOB

<n> e <m> inteiro de 1 a 6 dígitos

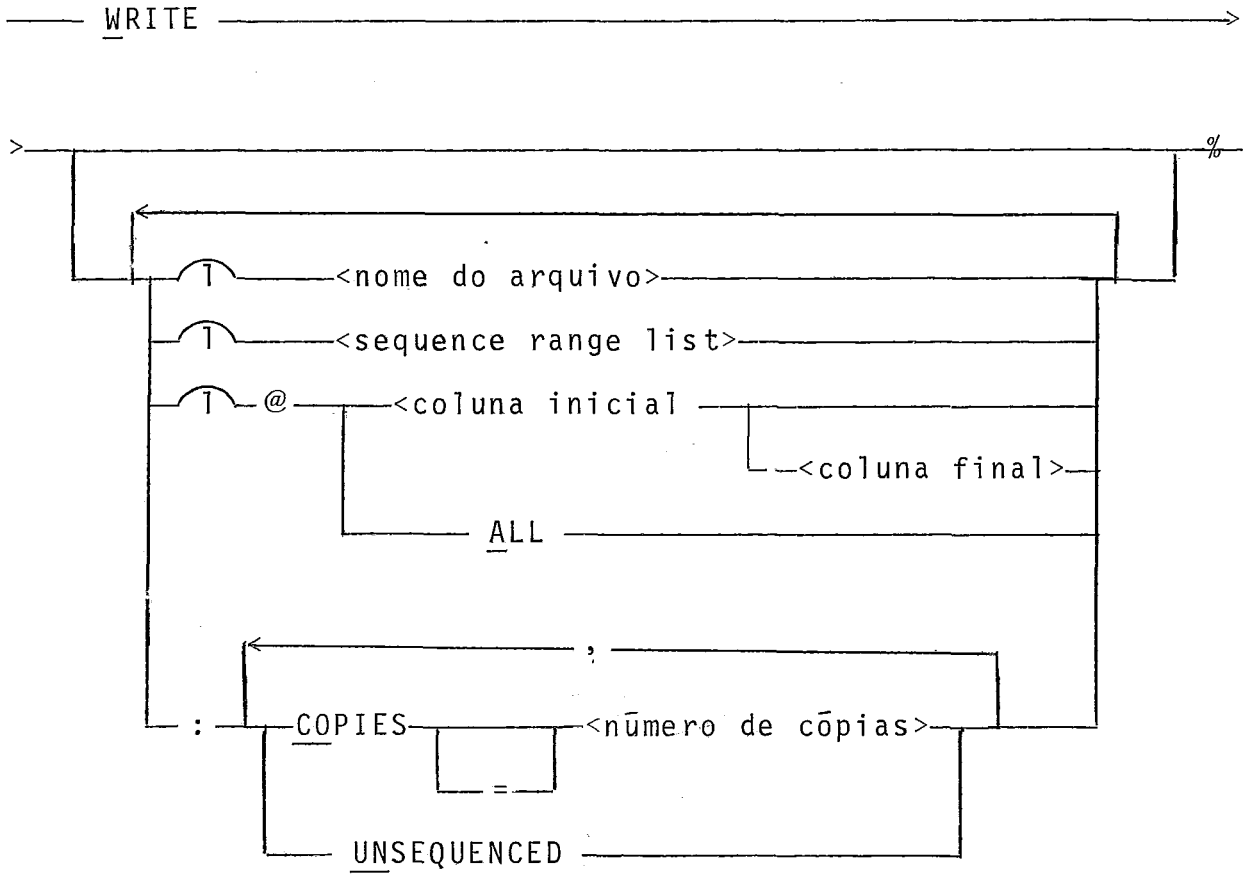
<status> é indicado por SALVO só no caso de o arquivo de trabalho ser salvo. No caso contrário não há indicação de status.

Exemplo:

→ WHAT

WRITE

Sintaxe:




Semântica:

O comando WRITE permite imprimir na impressora do Mitra 15 um arquivo (ou partes) da biblioteca do usuário ou o arquivo de trabalho.

<nome do arquivo> especifica o nome do arquivo da biblioteca do usuário a imprimir. Se esse parâmetro for omitido, será assumido no lugar o arquivo de trabalho.

<sequence range list> determina as seqüências de

linhas do arquivo a imprimir. Se esse parâmetro for omitido, será assumido no lugar todo o arquivo.

A opção @ especifica o intervalo das colunas de cada linha a ser impressa se <coluna inicial> e <coluna final> forem explicitadas. Se <coluna final> for omisso, todas as linhas serão impressas a partir da <coluna inicial>.

ALL indica que a linha toda será impressa mas com compactação de brancos.

Se a opção @ não aparecer no comando, as linhas serão impressas integralmente.

<número de cópias> especifica o número de cópias a serem impressas. O valor "default" é de uma cópia.

UNSEQUENCED indica que a listagem não será numerada.

O fim da execução do comando é marcado pela mensagem:

# ARQUIVO NA FILA DE IMPRESSÃO

Exemplos:

→ WRITE EDITL

→ W 0-2000, 8000-END @ 1 - 25:CO = 2, U

## Comandos Complementares de Edição

A filosofia "full screen" que adotamos exigiu a implementação de comandos complementares de edição baseados em caracteres de controle.

Alguns desses caracteres permitem edição direta de textos já presentes na tela e apontados pelo cursor enquanto outros facilitam a manipulação do próprio cursor ou do texto.

Poder-se-ia mesmo ter pensado num editor "full screen" baseado unicamente em caracteres de controle e simples entrada de argumentos. Um teclado de terminal vídeo oferece normalmente 33 caracteres de controle (códigos ASCII decimais, sem paridade de 0 a 31 e código 131) que permitiriam implementar 33 funções diferentes. O inconveniente de uma tal filosofia consiste na aprendizagem e operação das teclas usadas para gerar os caracteres de controle. Se forem numerosas as funções, a escolha de muitas teclas de controle ficará totalmente arbitrária e de muito difícil memorização, posto que certos caracteres de controle já são reservados para movimentação do cursor (←, ↑, ↓, →, CR, HOME/RESET).

Por outro lado, uma linguagem de edição tal como o CANDE, com uma certa sintaxe a respeitar e muitos parâmetros, pode ser vista pelo usuário como um tanto pesada.

Por isso fizemos um esforço particular na deteção de erros de sintaxe e semântica dos comandos, com mensagens de

erros explícitas e posicionamento do cursor sobre o ponto errado, e além disso, permitimos o reaproveitamento dos 10 últimos comandos efetivados pelo editor, a fim de evitar redigitação de comandos compridos repetitivos.

Mas além dessas facilidades, o editor tenta oferecer um compromisso razoável entre a escolha de um conjunto de caracteres de controle facilmente memorizável e uma verdadeira linguagem de edição.

Tais caracteres de controle não foram escolhidos arbitrariamente mas sim pelo valor mnemônico de alguns (em inglês já que adotamos o CANDE como linguagem de comando) em relação à função que implementam, ou pela posição privilegiada que outros ocupam no teclado dos terminais.

Também, outros caracteres de controle são emitidos por teclas especializadas de função implícita (←, →, ↑, ↓, HOME/RESET) ou convencional (CR, RUB OUT/DELETE).

Enfim, o caractere de controle ESCAPE associado a algumas teclas escolhidas igualmente por seu papel mnemônico implementa funções inversas das precedentes. ESCAPE associado a um número é também usado para restituir elementos da pilha dos 10 últimos comandos efetivados.

A seguir descrevemos as diversas funções complementares de edição que dividimos em quatro grupos:

- funções de deslocamento do cursor e do texto
- funções de edição propriamente dita
- funções inversas
- funções de restituição de elementos da pilha dos comandos.

Funções de Deslocamento do Cursor e do Texto

Movimentação do cursor para a esquerda: tecla especializada ←.

O cursor é deslocado horizontalmente de uma posição para a esquerda. Se o cursor já estiver na primeira posição permitida da linha (col. 4 em modo comando, col. 8 em modo texto), ficará imóvel e será ecoado no lugar o caractere BELL (campainha).

Movimentação do cursor para a direita: tecla especializada →.

O cursor é deslocado horizontalmente de uma posição para a direita. Se o cursor já estiver na última posição permitida da linha (col. 79), ficará imóvel e será ecoado no lugar o caractere BELL.

Movimentação do cursor para baixo, obtenção da próxima linha: teclas especializadas ↓ ou LF (line feed).

Em modo texto o cursor é deslocado verticalmente para baixo até a mais próxima linha eventual.

Se não existir nenhuma linha embaixo da linha do cursor e se o editor estiver no submodo "listagem", será apresentada logo embaixo da linha do cursor a próxima linha de listagem do arquivo. Se o cursor estiver inicialmente na última linha da tela, o rolamento da tela para cima será efetuado.

No caso da última linha de listagem, o cursor passa na próxima linha em modo comando.

No submodo "seqüencial" a atuação da tecla ↓ é semelhante mas ao invés de dar a próxima linha da listagem apresenta a próxima linha seqüencial do arquivo. A saída automática para o modo comando é feita na tentativa de ultrapassar o fim do arquivo.

No submodo "seqüenciamento" e nas mesmas condições aparecerá a próxima seqüência de linha ou a próxima linha intermediária.

Em modo comando numa linha onde já existiu texto, o editor sai do modo comando apagando um eventual comando digitado, restitui a linha original de texto e continua um tratamento idêntico ao do submodo "seqüencial" do modo texto que acabamos de descrever.

Numa linha onde não existiu texto, o cursor fica

imóvel e é ecoado o caractere BELL.

Movimentação do cursor para cima, obtenção da linha precedente: tecla especializada ↑.

Em modo texto o cursor é deslocado verticalmente para cima até a mais próxima linha eventual.

Se não existir nenhuma linha acima da linha do cursor e no submodo "listagem", será apresentada, logo acima da linha do cursor, a linha de listagem precedente do arquivo (em relação à linha do cursor). Se o cursor estiver inicialmente na primeira linha da tela, o rolamento da tela para baixo será efetuado nos terminais que admitem tal facilidade. Nos outros terminais o cursor ficará imóvel e será ecoado o caractere BELL.

Na tentativa de ultrapassar a primeira linha de listagem será efetuada a mudança automática para o modo comando.

No submodo "seqüencial" a atuação da tecla ↑, muito semelhante, permite obter nas mesmas condições a linha anterior do arquivo e proporcionar a passagem automática em modo comando na tentativa de ultrapassar a primeira linha do arquivo.

Nos submodos "sequenciamento" e "procura" a tecla ↑ é sem ação na primeira linha presente na tela e é ecoada pelo caractere BELL.

Em modo comando numa linha onde já existiu texto,

o editor sai do modo comando apagando um eventual comando digitado, restitui a linha original de texto e continua um tratamento idêntico ao do submodo "seqüencial" do modo texto que acabamos de descrever.

Numa linha onde não existiu texto, o cursor fica imóvel e é ecoado o caractere BELL.

Movimentação do cursor para o mais próximo ponto de tabulação à direita: caractere de controle CNTRL/W (escolhido pela sua posição no teclado: ver próximas observações).

O cursor é deslocado horizontalmente para a direita até o próximo ponto de tabulação. Se o cursor já estiver na última posição permitida da linha (col. 79), ficará imóvel e será ecoado no lugar o caractere BELL.

São 10 pontos de tabulação possíveis além dos dois extremos correspondentes à primeira e última colunas acessíveis de uma linha. Os pontos de tabulação são determinados pelo uso dos comandos TAB, SET e RESET e dos comandos complementares CNTRL/T e ESCAPE + T aos quais pedimos se referir.

Movimentação do cursor para o mais próximo ponto de tabulação à esquerda: caractere de controle CNTRL/Q (escolhido pela sua posição no teclado: ver próximas observações).

O cursor é deslocado horizontalmente para a esquerda até o próximo ponto de tabulação. Se o cursor já estiver na

primeira posição permitida da linha (col. 4 em modo comando, col. 8 em modo texto), ele ficará imóvel e será ecoado no lugar o caractere BELL.

Movimentação rápida do cursor numa linha: caractere de controle CNTRL/A (escolhido pela sua posição no teclado: ver próximas observações).

O cursor é deslocado horizontalmente até a primeira posição permitida da linha (col. 4 em modo comando, col. 8 em modo texto).

Se o cursor já estiver na primeira posição permitida, será deslocado para a última posição permitida da linha (col. 79).

Observações sobre os Comandos de Tabulação e de Movimentação Rápida do Cursor

A escolha das teclas para realizar esses três comandos foi feita em função da posição privilegiada dessas teclas entre elas (W à direita para tabulação à direita, Q à esquerda para tabulação à esquerda, A no meio embaixo de Q e W para movimentação rápida do cursor nos dois sentidos) e a proximidade destas da tecla CNTRL.

Colocação de pontos de tabulação: caractere de controle CNTRL/T (mnemônio de "TAB (SET)").

Os 10 pontos de tabulação possíveis podem ser de-

terminados pelo uso do comando TAB que trabalha com números de colunas de tabulação. Mas às vezes os números de colunas de tabulação são pouco significativos para o usuário que determinará visualmente a necessidade ou não de um ponto de tabulação. O presente comando foi criado com essa finalidade.

A função da tecla CNTRL/T é criar um ponto de tabulação na posição do cursor. Se já existirem 10 pontos de tabulação e tentarmos criar um novo ponto, tal operação não será aceita e no lugar será ecoado o caractere BELL.

Como veremos mais adiante existe uma função inversa que retira pontos de tabulação: ESCAPE + T

Movimentação do cursor para o final do mais próximo grupo de caracteres à direita: caractere de controle CNTRL/N (escolhido pela sua posição no teclado: ver próximas observações).

O cursor é deslocado horizontalmente para a direita até o final do mais próximo grupo de caracteres. Se o cursor já estiver no final do grupo de caracteres mais à direita da linha, ficará imóvel e será ecoado no lugar o caractere BELL.

Movimentação do cursor para o início do mais próximo grupo de caracteres à esquerda: caractere de controle CNTRL/B (escolhido pela sua posição no teclado: ver próximas observações).

O cursor é deslocado horizontalmente para a esquer

da até o início do mais próximo grupo de caracteres. Se o cursor já estiver no início do grupo de caracteres mais à esquerda da linha, ficará imóvel e será ecoado no lugar o caractere BELL.

Observações sobre os Comandos de Movimentação por Grupos de Caracteres

Entendemos por grupo de caracteres qualquer conjunto misturado ou não de letras, números ou caracteres especiais não incluídos os brancos. Um ou vários brancos delimitam os grupos de caracteres entre eles.

A escolha das teclas CNTRL/N e CNTRL/B foi feita em função da sua posição privilegiada no teclado, isto é, na metade superior da barra de espacejamento.

Tabulação vertical: tecla especializada HOME/RESET.

No modo texto o cursor é deslocado verticalmente até a primeira linha de texto presente na tela. Se o cursor já estiver nessa primeira linha, será deslocado para a linha do meio do texto presente na tela. Enfim, se o cursor já estiver nessa linha do meio, irá para a última linha de texto presente na tela.

No modo comando essa função é inválida e será ecoado no lugar o caractere BELL.

Obtenção da próxima página, entrada de uma nova linha de seqüenciamento, entrada de um comando: tecla especiali-

zada CR (carriage return).

A tecla CR tem basicamente as mesmas funções que no CANDE.

Em modo texto e no submodo "listagem" será apresentada, após apagamento de toda a tela, a próxima página de listagem do arquivo, a qual começará na linha onde o cursor estava posicionado. Se a listagem apresentada ocupar a tela toda do terminal, o cursor ficará no modo texto na última linha da tela. No caso contrário o cursor ficará em modo comando na linha seguinte à última linha listada na tela.

No submodo "seqüencial" será apresentada a próxima página seqüencial do arquivo ao invés da próxima página de listagem determinada pelo comando LIST.

Em modo texto e no submodo seqüenciamento a tecla CR permitirá inserir no arquivo de trabalho uma nova linha e obter a próxima linha de seqüenciamento ou linhas intermediárias, se houver.

No modo comando a tecla CR submete o comando digitado na linha ao analisador e interpretador de comandos.

Listagem de fim de página: caractere de controle CNTRL/L (mnemônico de "LIST END OF PAGE").

No modo texto em qualquer um dos submodos todas as

linhas abaixo da linha do cursor, esta inclusive, são apagadas e é listada, a partir da linha do cursor, a continuação do arquivo já em parte presente na tela até a linha do cursor. Se não existir nenhuma linha na tela acima da linha do cursor, a própria linha do cursor será assumida como primeira linha de listagem e a listagem será feita a partir da primeira linha da tela, como acontece com a tecla CR em modo texto.

A tecla CNTRL/L tem a mesma função em todos os submodos do modo texto mas força o editor a passar no submodo "seqüencial", se já não estiver nele, e será então utilizada para sair dos outros submodos e em particular, do submodo "seqüenciamento".

No modo comando CNTRL/L é inválido e ecoado pelo caractere BELL.

Passagem em modo comando: caractere de controle CNTRL/C (mnemônico de "COMMAND MODE").

Seja qual for o modo do editor a linha do cursor é apagada e é apresentada uma linha vazia em modo comando. Esse comando pode ser utilizado para zerar um comando ou passar do modo texto para o modo comando.

Funções de Edição Propriamente Dita

Supressão de caractere: caractere de controle CNTRL/S (mnemônico de "SUPPRESS CHARACTER").

Suprime o caractere apontado pelo cursor e desloca de uma coluna para a esquerda todos os caracteres à direita do cursor. Se não existir nenhum caractere à direita do cursor, a função será sem ação.

Ativação do estado de inserção de caracteres: caractere de controle CNTRL/I (mnemônico de "INSERT CHARACTERE MODE").

Ao ser recebido o caractere de controle CNTRL/I o editor passa do seu estado normal ao estado de inserção de caracteres (ver figura III-6). O estado normal do editor é caracterizado pelo posicionamento pelo editor do caractere teclado (não correspondente a uma função) em cima da posição apontada pelo cursor na tela, substituindo o caractere ou o espaço aí presentes.

No estado de inserção de caracteres, quando um caractere (não correspondente a uma função) é teclado, todos os caracteres à direita do cursor, inclusive o caractere apontado pelo cursor, são deslocados de uma coluna para a direita e o caractere teclado aparece na posição apontada pelo cursor. Doze caracteres são conservados e também deslocados além da coluna 79 e poderão reaparecer na tela por supressão de caracteres.

O editor volta ao estado normal quando forem utilizadas as teclas ←, →, quando o cursor sair da linha (com ↑, ↓, Home), com mudança de modo (CNTRL/C) ou pelo uso da função inversa ESC + I (ver mais adiante).

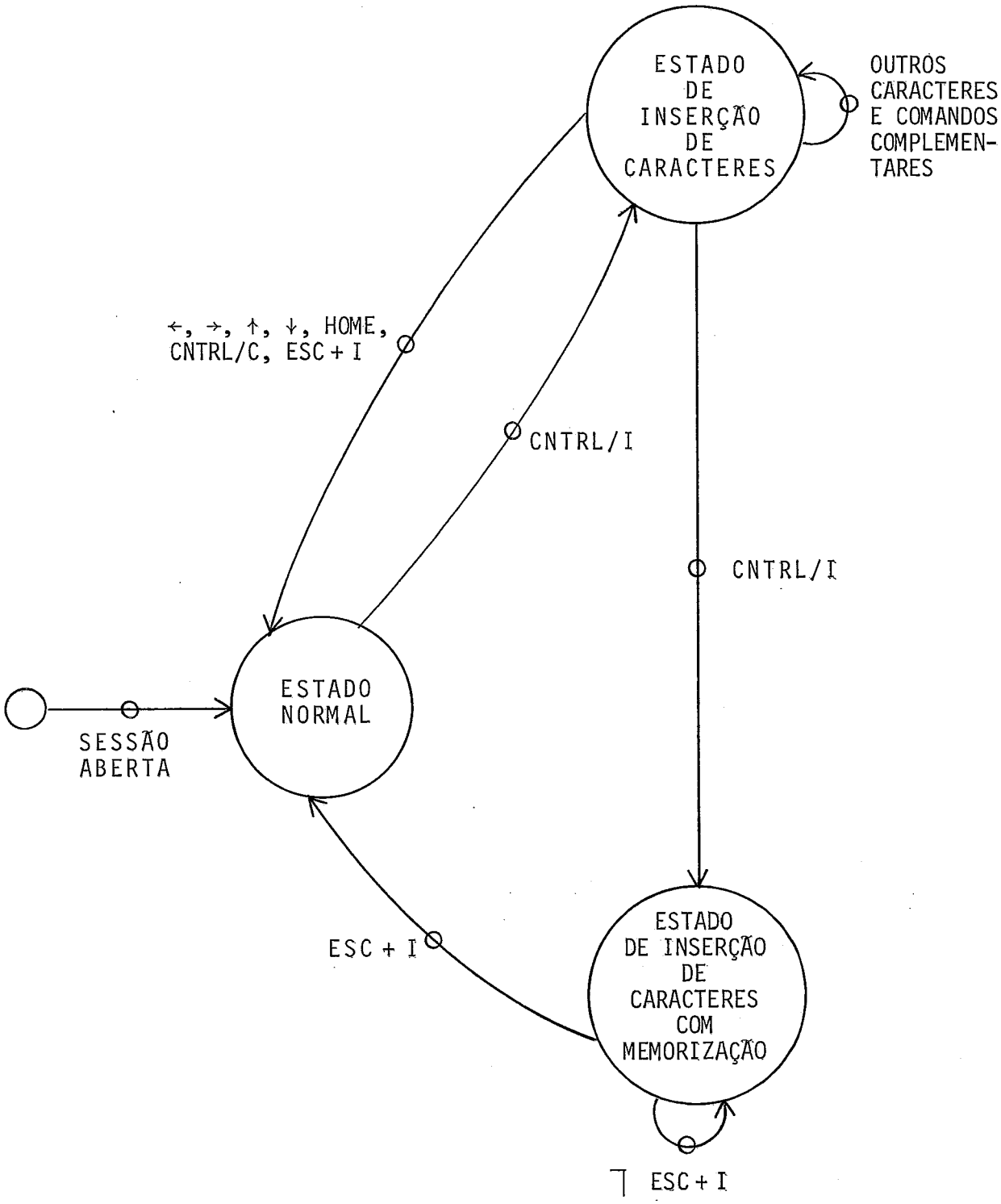



Fig. III-6: OS 3 ESTADOS DO EDITOR
(Independentemente dos modos e submodos)

Pela redigitação do caractere de controle CNTRL/I, o editor passa do estado de inserção de caracteres para o estado de inserção de caracteres com memorização. Esse novo estado de inserção é idêntico ao precedente com a diferença que fica ativo até a utilização da função inversa ESC + I que leva o editor para o modo normal. O editor não sai mais do estado de inserção por simples movimentação do cursor, como acontecia no outro caso.

Reprodução de linha: caractere de controle CNTRL/R (mnemônico de "REPRODUCE").

Permite reproduzir campos da última linha inserida no arquivo de trabalho (no submodo seqüenciado) ou da última linha modificada (modo texto).

A linha de referência será duplicada a partir do cursor e até o próximo ponto de tabulação.

No modo comando tal função é inválida e ecoada pelo caractere BELL.

Supressão de fim de linha: tecla especializada LINE CLEAR ou também caractere de controle CNTRL/E (mnemônico de "END OF LINE (CLEAR)").

Permite apagar o fim de uma linha a partir do cursor inclusive.

Se a operação for efetuada na primeira posição per

mitida da linha, a linha toda será apagada, mas sem deixar de existir: será uma linha branca.

Remoção de linha: caractere de controle CNTRL/D (mnemônico de "DELETE LINE").

Em modo texto permite remover a linha do arquivo de trabalho onde está posicionado o cursor.

Ao pressionar a tecla CNTRL/D a linha do cursor desaparece da tela e o cursor é posicionado no início da mais próxima linha embaixo da linha removida. Se não existir linha embaixo da que foi removida, a próxima linha do arquivo é listada na tela.

A linha removida não mais será acessível ao cursor.

A escolha de não compensação do espaço em branco introduzido na tela por esse método foi feita para não prejudicar os usuários utilizando terminais lentos. Salientamos que a compensação pode ser feita através do comando de listagem de fim de página (CNTRL/L).

Esse comando é inválido em modo comando e no submodo seqüenciamento numa nova linha de seqüenciamento.

Restituição de linha: tecla especializada RUB OUT/DELETE.

Tal tecla é utilizada também no CANDE para cancelar a digitação de uma linha.

Conservamos esse aspecto no modo comando onde a tecla RUB OUT/DELETE tem a mesma função que a tecla CNTRL/C, ou seja, de zerar a linha.

Em modo texto estendemos a idéia e a tecla RUB OUT/DELETE restitui a linha em edição.

Isto quer dizer que se numa linha forem introduzidas modificações e se sem sair dessa linha for apertada a tecla RUB OUT/DELETE, a linha original existente antes de todas essas modificações é restituída.

A saída de uma linha em edição (por exemplo pelas teclas ↑ e ↓) atualiza essa linha no nível do arquivo de trabalho. Após essa operação, não será mais possível recuperar pelo comando RUB OUT/DELETE a linha original. Será necessário usar o comando INSERT.

O presente comando é de grande importância pelo fato de o editor não possuir uma linha reservada para digitar comandos. Os comandos são digitados em qualquer lugar da tela após se ter passado no modo comando pela tecla CNTRL/C. Poderá acontecer então que o usuário, julgando estar em modo comando, comece a digitar um comando no meio de um texto. Poderá facilmente recuperar o seu erro através do presente comando. Também em caso de dúvida sobre um eventual deslocamento das informações apre

sentadas na tela, o usuário poderá recorrer a esse comando.

Funções Inversas

São comandos complementares que necessitam digitar duas teclas, a tecla especializada ESCAPE seguida de outra tecla.

Obtenção da página anterior: teclas ESCAPE + CR.

Função inversa do comando CR nos submodos "listagem" e "seqüencial" do modo texto, permite obter a página anterior a partir da linha do cursor (respectivamente página anterior de listagem e página anterior seqüencial do arquivo).

Toda a tela será apagada. As linhas serão listadas de baixo para cima da tela, sendo que a primeira linha listada (que passa a ficar na última posição da tela) é a linha onde estava posicionado o cursor. Se a listagem apresentada ocupar a tela toda do terminal, o cursor ficará no submodo corrente na primeira linha da tela. No caso contrário (respectivamente ultrapassagem do início da listagem e do início do arquivo) o cursor ficará em modo comando na linha precedente à última linha listada na tela, ou seja, na linha precedente à primeira linha de texto presente na tela.

Nos submodos "seqüenciamento" e "procura" e no modo comando o presente comando é inválido e será ecoado pelo caractere BELL.

Supressão de pontos de tabulação: teclas ESCAPE + T.

Função inversa do comando CNTRL/T, permite suprimir pontos de tabulação entre os 10 possíveis.

O cursor deve ser posicionado no ponto de tabulação a cancelar para o comando ser efetivado. No caso contrário o comando será inválido e será ecoado no lugar o caractere de controle BELL. Lembramos que o comando RESET TAB suprime todos os pontos de tabulação de uma só vez.

Desativação do estado de inserção de caracteres: teclas ESCAPE + I.

Permite ao editor sair do estado de inserção de caracteres e voltar ao estado normal de edição. É o único comando que permite sair do estado de inserção memorizado para o estado normal do editor.

Funções de Restituição de Elementos da Pilha dos Comandos

Passagem em modo comando com restituição de um dos 10 últimos comandos efetivados: teclas ESCAPE + {0,1,...,9}.

Permite forçar o editor a passar no modo texto e restituir um dos 10 últimos comandos efetivados e guardados na pilha dos comandos. ESC + 0 restitui o comando mais recente,

ESC + 9 restitui o comando mais antigo. É claro que o comando restituído que aparecerá na linha em modo comando pode ser modificado à vontade e até abandonado.

Se o número escolhido não corresponder a nenhum comando (em certos casos onde 10 comandos ainda não foram usados) será apresentada uma linha branca em modo comando.

Resumo das Teclas Usadas nos Comandos Complementares de Edição e Justificativa de sua Escolha

Tentaremos mostrar agora os motivos da escolha das teclas no sentido de minimizar o esforço de memorização do usuário. Lembramos também que em caso de dúvida sobre a função de uma tecla, o usuário poderá recorrer ao comando HELP.

As funções complementares de edição foram implementadas através de 4 tipos diferentes de teclas.

Teclas Especializadas

Sete teclas especializadas de ação diferente foram utilizadas.

As teclas ←, →, ↑, ↓ (ou LF) têm uma função óbvia para o usuário.

As teclas CR e RUB OUT/DELETE têm funções baseadas no CANDE, as quais foram apenas estendidas.

A tecla HOME/RESET tem uma função parecida com a função de hardware disponível normalmente num terminal (posicionamento no canto superior ou inferior da tela).

Teclas Escolhidas pela sua Posição Privilegiada no Teclado e entre Si Mesmas

São 5 no total e foram exclusivamente reservadas para tabulação do cursor. O usuário não deverá memorizar o nome delas mas a sua posição (ver figura III-7).

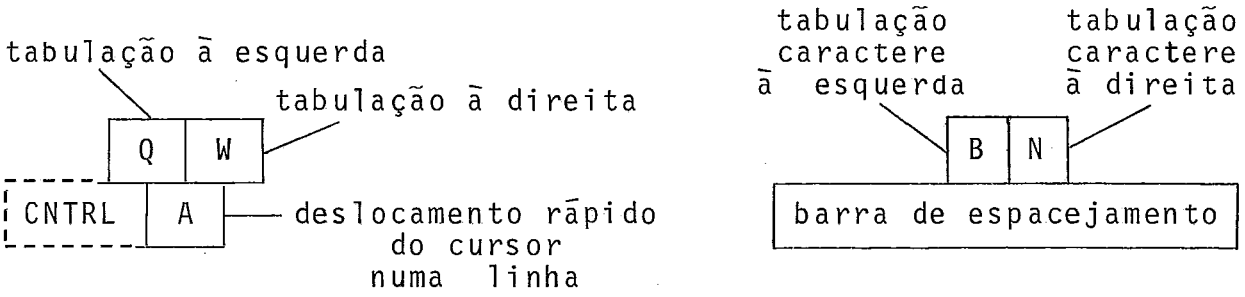



Fig.III-7: Teclas de tabulação escolhidas pela sua posição privilegiada no teclado e entre si

Teclas Escolhidas pelo seu Valor Mnemônico

São limitadas a 8 que deverão ser memorizadas pelo usuário, facilmente a nosso ver, pois os mnemônicos escolhidos pertencem à linguagem comumente usada para descrever editores.

Na tabela seguinte resumimos estas teclas e sua origem mnemônica que indica a função que implementam.

| Tecla | Origem Mnemônica |
|---|---|
| CNTRL/T | TAB (SET) |
| CNTRL/L | LIST END OF PAGE |
| CNTRL/C | COMMAND MODE |
| CNTRL/S | SUPPRESS CHARACTER |
| CNTRL/I | INSERT CHARACTER MODE |
| CNTRL/R | REPRODUCE |
| CNTRL/E | END OF LINE (CLEAR) |
| CNTRL/D | DELETE LINE |

Teclas com ESCAPE para Funções Inversas

São apenas 3, escolhidas também pelo valor mnemônico da função que invertem. São resumidas na seguinte tabela:

| Sentido da Função Direta | Teclas da Função Direta | Teclas da Função Inversa Correspondente | Sentido da Função Inversa |
|---|---|---|---|
| próxima página | CR | ESC + CR | página anterior |
| SET TAB | CNTRL/T | ESC + T | RESET TAB |
| SET INSERT CH.MODE | CNTRL/I | ESC + I | RESET INSERT CH.MODE |

Teclas com ESCAPE para Restituição de Elementos da Pilha dos Comandos

Este tipo de teclas, idêntico ao precedente e que foi colocado nos comandos complementares de edição por usar o ca

ractere ESCAPE, é na realidade mais aparentado com a linguagem de comando do que com as facilidades de edição "full screen".

Já vimos a vantagem de tais funções de restituição de comandos anteriores, fáceis de usar pelo emprego de teclas numéricas que, normalmente, constituem nos terminais um bloco separado do bloco das outras teclas.

## III.2 - Analisador e Tradutor

### Analisador Sintático

O analisador sintático ("PARSER") implantado para interpretar a linguagem de comando do editor é do tipo RRP LL(1) (TELES[33]). Este tipo de analisador foi escolhido por utilizar uma tabela de controle reduzida (e nosso ponto crítico é justamente o espaço em memória) sem prejudicar o desempenho do algoritmo de análise por ser este de ordem n, ou seja, com tempo proporcional ao comprimento da sentença analisada.

Nosso primeiro trabalho consistiu em transformar os diagramas de sintaxe já apresentados na descrição da linguagem de edição em grafos orientados permitindo definir para cada comando um conjunto de nós inter-relacionados e para cada nó um tipo de ação sintática e também de ação semântica.

As regras de formação desses novos grafos são simples. Cada nó pode ter no máximo um nó sucessor e/ou um nó alternado. Cada nó é numerado e a ele é associado um símbolo que corresponde a um nome de entidade sintática.

O algoritmo do analisador deverá achar um caminho de nó em nó, no sentido das setas, a partir do nó inicial de um comando, até chegar ao nó EOFL (end of line) que encerra o comando. Isto corresponde precisamente a criar um autômato finito determinístico para cada regra sintática.

Certas seqüências de nós, por se repetirem várias vezes, foram agrupadas em um só nó chamado nó não terminal que, por sua ação sintática própria, desviará o analisador para um conjunto isolado de nós. Assim reduzimos o número total de nós da mesma forma que a criação de uma categoria sintática reduz o número de regras sintáticas.

Também com a mesma finalidade, dado que todos os comandos começam por uma palavra reservada distinta, eliminamos o nó inicial de cada comando que iria ter por símbolo o código da palavra reservada associada ao comando. O novo nó inicial de cada comando é o nó que teria sido o sucessor do nó eliminado. O número do nó por onde o analisador começará sua pesquisa é dado pelo código do primeiro "token" do comando (determinado pelo "SCREENER").

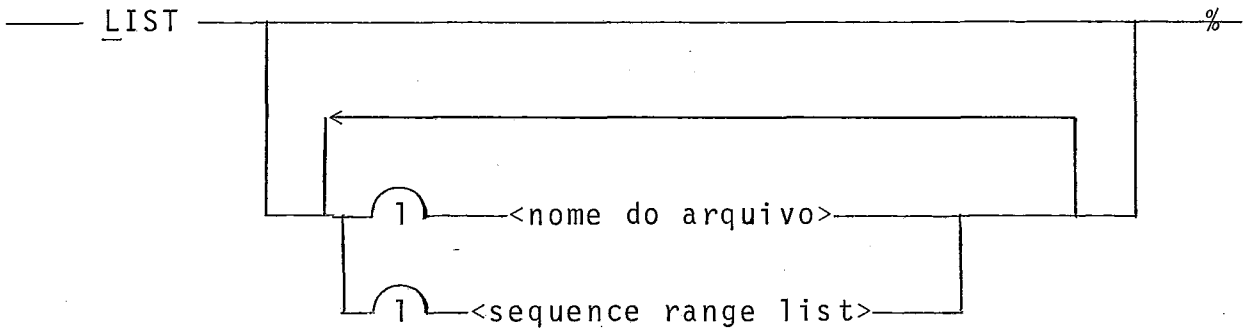
A seguir damos um exemplo de transformação do diagrama de sintaxe do comando LIST em grafo que permitirá compor a tabela de controle do analisador.

Os nós aparecem representados de maneira geral por retângulos identificados pelo número do nó e contendo

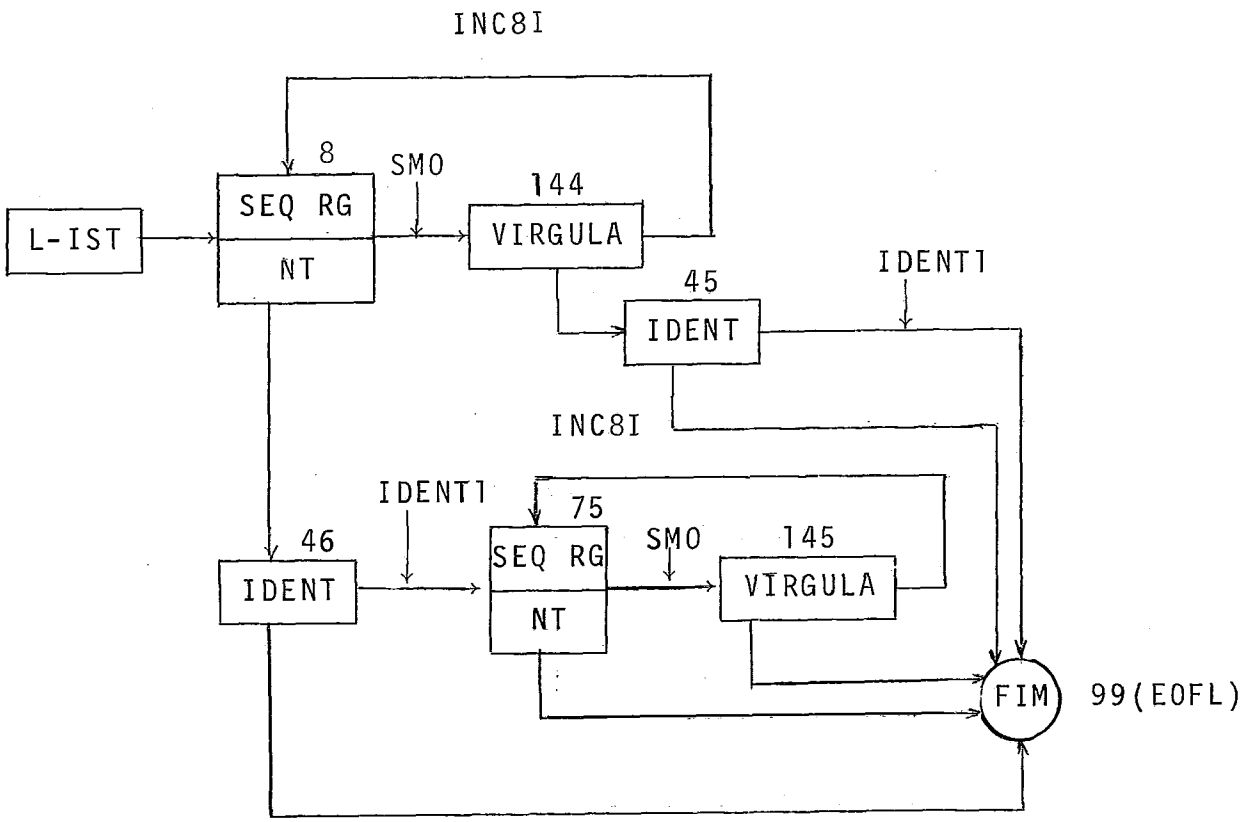
o símbolo do nó. Como a maioria dos nós têm uma ação sintática idêntica (ação TST), essa ação não foi indicada. Apenas no caso dos nós não terminais, aparece dentro do retângulo junto ao símbolo a indicação NT. Os nós representados por formas circulares (nós de saída de grafos ou subgrafos) contêm somente o tipo de sua ação sintática (por exemplo: ação FIM).

Entre cada nó e seu sucessor aparece uma indicação que aponta por uma seta a saída do nó e corresponde à ação semântica específica do nó que detalharemos mais adiante.

Diagrama de Descrição da Sintaxe do Comando LIST

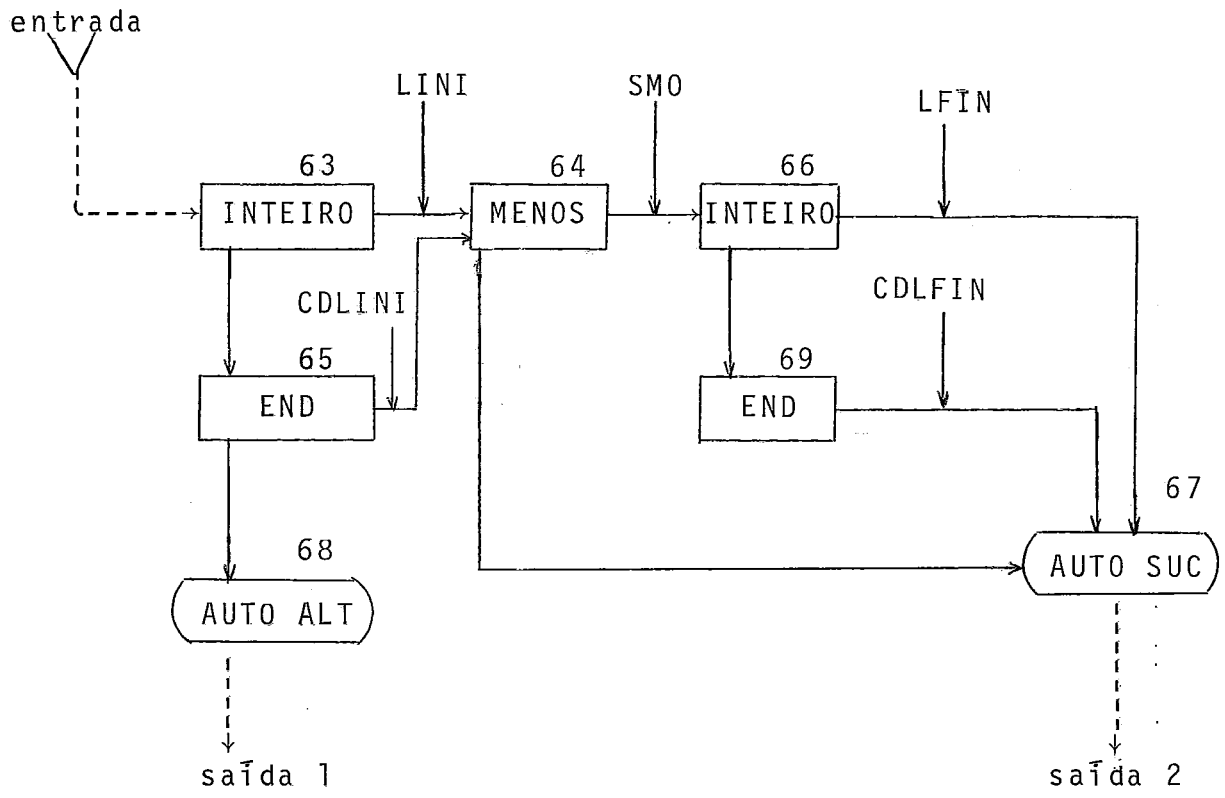


Grafo Correspondente




Nesse grafo aparecem os nós 8 e 75 de símbolo SEQ RG (SEQUENCE RANGE) que são nós não terminais. O subgrafo correspondente é mostrado logo a seguir. Os nós 45 e 46 de símbolo IDENT (IDENTIFICADOR) correspondem ao parâmetro <nome do arquivo>.

Subgrafo Correspondente ao Nó Não Terminal SEQ RG




Nesse subgrafo AUTO ALT e AUTO SUC correspondem a ações sintáticas particulares.

O algoritmo do PARSER trabalha com seis tipos diferentes de ações sintáticas que serão detalhadas juntamente com esse algoritmo.

Codificação do Grafo

Para os grafos apresentados podemos montar, a título de exemplo, a seguinte tabela onde aparecem 5 dos 6 tipos existentes de ações sintáticas.

| Número do Nó | Símbolo do Nó | Nó Alternativo | Nó Sucessor | Ação Sintática do Nó | Ação Semântica do Nó |
|---|---|---|---|---|---|
| NO | SIMB | ALT | SUC | AÇÃOST | AÇÃOSM |
| 8 | SEQRG | 46 | 144 | NT | - |
| 45 | IDENT | EOFL | EOFL | TST | IDENT1 |
| 46 | IDENT | EOFL | 75 | TST | IDENT1 |
| 63 | INTEIRO | 65 | 64 | TST | LINI |
| 64 | MENOS | 67 | 66 | TST | - |
| 65 | END | 68 | 64 | TST | CDLINI |
| 66 | INTEIRO | 69 | 67 | TST | LFIN |
| 67 | - | - | - | AUTOSUC | - |
| 68 | - | - | - | AUTOALT | - |
| 69 | END | λ | 67 | TST | CDLFIN |
| 75 | SEQRG | EOFL | 145 | NT | - |
| (EOFL)99 | - | - | - | FIM | - |
| 144 | VIRG | 45 | 8 | TST | INC8I |
| 145 | VIRG | EOFL | 75 | TST | INC8I |

Algoritmo do PARSER

O algoritmo do analisador é extremamente simples.

Compõe-se de uma fase inicial na qual é tratado o primeiro "token" e são feitas as inicializações, seguida da fase principal onde o grafo correspondente a um comando é percorrido até o seu último nó. Essa fase principal é constituída de uma única instrução WHILE dentro da qual opera uma instrução CASE função do tipo de ação semântica.

O algoritmo é o seguinte, descrito em linguagem aparentada ao ALGOL-60:

```
% fase inicial

SCREENER;              % pede primeiro token (devolve CODE/INFO)
if CODE = FDL          % primeiro código = fim de linha (FDL)
then ERROVAZIO;        % deteção de comando vazio
if CODE > MAXCODE      % o primeiro token do comando não corres-
then ERROINI;          % ponde a um nome de comando
NÓ: = CODE;            % o nó de partida do PARSER no grafo corres
                       % ponde ao código do primeiro token.
SCREENER;              % pede segundo token
ACABOU: = 0;           % inicialização do WHILE
```

```
                % fase principal: percorrendo o grafo!

while not ACABOU
do case  AÇÃOST[NÓ]
of
TST:    if SIMB[NÓ] = IDENT          % resolve o caso geral de um
        and CODE ≤ IDENT             % identificador puro e o caso
        then begin                   % particular de uma palavra cha
             NÓ: = SUC[NÓ];          % ve usada como identificador
             SCREENER;               % pede próximo token
             end
        else begin
             if CODE = SIMB[NÓ];
             then begin
                  NÓ: = SUC[NÓ];
                  SCREENER;          % pede próximo token
                  end
             else begin
                  if ALT[NÓ] = λ     % testa se existe um nó alter
                                     %                       nativo
                  then ERROST;       % deteção de erro de sintaxe
                  else NÓ:=ALT[NÓ]   % o nó alternativo substitui
                  end;               % o nó corrente e a pesquisa
                                     %                   continua
        end;
```

```
NT:         SALVANÓ:=NÓ;              % salva o nó corrente

            NÓ:=SIMB[NÓ];             % SIMB nesse caso é
                                      % o número do nó inicial do
                                      %                  subgrafo
                                      % correspondente ao nó não ter
                                      %                  minal

AUTOSUC:    NÓ:=SALVANÓ;              % restituição do nó corrente
                                      %        do grafo principal

            NÓ:=SUC[NÓ];              % o algoritmo continua no nó
                                      %                  sucessor

AUTOALT:    NÓ:=SALVANÓ;              % restituição nó corrente do
                                      %            grafo principal

            if ALT[NÓ]=λ              % testa se existe um nó alter
                                      %                  nativo

            then ERROST;              % deteção de erro de sintaxe

            NÓ:=ALT[NÓ];              % o algoritmo continua no nó
                                      %                  alternativo

FIM:        ACABOU:=1;                % determina o fim do algoritmo

            if CODE≠EOL               % testa se o código é o de fim
                                      %                  de linha

            then ERROFIM;             % deteção erro de fim de co-
                                      %            mando esperado

TRANSIT:    NÓ:=SUC[NÓ];              % nó "transparente" sem ação
                                      %        sintática própria.

end

            % fim da análise sintática
```

Obs.: No caso de uma gramática qualquer necessitaríamos de uma

pilha sintática para guardar os desvios de um grafo (produção sintática) para outro e seu respectivo retorno. No caso de nossa gramática a pilha sintática não ultrapassará dois elementos, e foi representada pelas variáveis NÓ (topo) e SALVANÓ (sub-topo).

Exemplo:

A seguir damos um exemplo dos passos efetuados pelo analisador no caso da análise do comando:

LIST ARQ/N1 END - 2000

fase inicial

SCREENER → CODE = 8 (LIST)

NÓ:=8

SCREENER → CODE = IDENT (identificador)

ACABOU:=0

fase principal

- AÇÃOST[8] = NT

SALVANÓ:=8

NÓ:=SIMB[NÓ]

(63)

- AÇÃOST[63] = TST detalhando os intermediários

SIMB[63] ≠ IDENT

(INTEIRO)

CODE ≠ SIMB[63]

(IDENT) (INTEIRO)

ALT[63] ≠ λ

(65)

NÓ:=ALT[63]

(65)

- AÇÃOST[65] = TST sem mais detalhar todos os intermediários

CODE ≠ SIMB[65]

(IDENT) (END)

NÓ:=ALT[65]

(68)

- AÇÃOST[68] = AUTOALT

NÓ:=SALVANÓ

(8)

ALT[NÓ] ≠ λ

(46)

NÓ:=ALT[NÓ]

(46)

- AÇÃOST[46] = TST

SIMB[46] = IDENT

CODE ≤ IDENT

(IDENT)

NÓ:=SUC[NÓ]

(75)

SCREENER → CODE = END

- AÇÃOST[75] = NT

SALVANÓ: = 75

NÓ: = SIMB[NÓ]

(63)

- AÇÃOST[63] = TST

CODE ≠ SIMB[63]

(END) (INTEIRO)

NÓ: = ALT[63]

(65)

- AÇÃOST[65] = TST

CODE = SIMB[65]

(END) (END)

NÓ: = SUC[65]

(64)

SCREENER → CODE = MENOS

- AÇÃOST[64] = TST

CODE = SIMB[64]

(MENOS) (MENOS)

NÓ: = SUC[64]

(66)

SCREENER → CODE = INTEIRO

- AÇÃOST[66] = TST

CODE = SIMB[66]

(INTEIRO) (INTEIRO)

NÓ: = SUC[66]

(67)

SCREENER → CODE = EOL (fim de linha)

- AÇÃOST[67] = AUTOSUC

NÓ: = SALVANÓ

(75)

NÓ: = SUC[75]

(145)

- AÇÃOST[145] = TST

CODE ≠ SIMB[NÓ]

(EOL) (VIRG)

NÓ: = ALT[145]

(99)

- AÇÃOST[99] = FIM

ACABOU: = 1

CODE = EOL

(EOL)

Identificador ("SCREENER")

O identificador, chamado pelo analisador sintático, tem por função devolver-lhe as características do próximo elemento sintático ("token") (a partir do apontador PFIN).

Essas características são:

. código do token (CODE)

. apontadores início e fim de token (PINI, PFIN)

. valor binário em 32 bits no caso do token ser um número inteiro de até 6 dígitos decimais significativos (GAUCHE, DROITE).

Todas essas características não são diretamente determinadas pelo SCREENER pois o seu primeiro passo é chamar o analisador léxico que, por sua vez, determina os apontadores início e fim de token e um código provisório do token.

Somente se o código do token devolvido pelo analisador léxico for o de uma suposta palavra chave, o SCREENER tentará identificar o código correspondente ou atribuirá o código de identificador puro no caso de o token não corresponder a nenhuma palavra chave da sua tabela.

Também se o código desenvolvido pelo analisador léxico for de número inteiro, o SCREENER fará a conversão da cadeia decimal do token em número binário de 32 bits.

Para os outros tipos de códigos o SCREENER ficará transparente e transmitirá diretamente para o PARSER as informações provenientes do analisador léxico, sem nenhuma alteração.

O algoritmo de identificação de palavras chaves do SCREENER trabalha com uma tabela de apontadores de 26 vetores, um para cada letra do alfabeto.

Cada vetor é composto de uma sucessão de grupos de bytes, um grupo para cada palavra chave.

Um grupo é constituido de:

. 1 byte determinando o mínimo de letras identificando a palavra chave

. 1 byte determinando o máximo de letras identificando a palavra chave

. 1 byte dando o código da palavra chave

. 1 cadeia de letras do nome da palavra chave menos a sua primeira letra.

O algoritmo, após ter identificado o vetor com o qual vai trabalhar (graças à primeira letra do token), o percorrerá e tentará a identificação letra a letra do token com as cadeias de palavras chaves guardadas no vetor somente se o tamanho do token estiver incluído entre o mínimo e o máximo identificando cada palavra chave.

Sabendo-se também que o máximo de palavras chaves por letra alfabética é de 5 e em média de 1,6, o algoritmo de identificação é extremamente rápido.

Analisador Léxico ("SCANNER")

O analisador léxico tem por função determinar:

. o código provisório (CODE) do próximo token (a partir do apontador PFIN)

. os apontadores início (PINI) e fim (PFIN) desse token.

O SCANNER trabalha a partir de uma tabela de correspondência reduzida de 65 bytes que transforma caracteres alfanuméricos e especiais maiúsculos ASCII sem paridade (foram eliminados caracteres de controle e minúsculos) em símbolos diretamente utilizáveis no autômato finito, base do algoritmo do SCANNER.

A seguir descrevemos o autômato finito que permite determinar os códigos dos tokens:

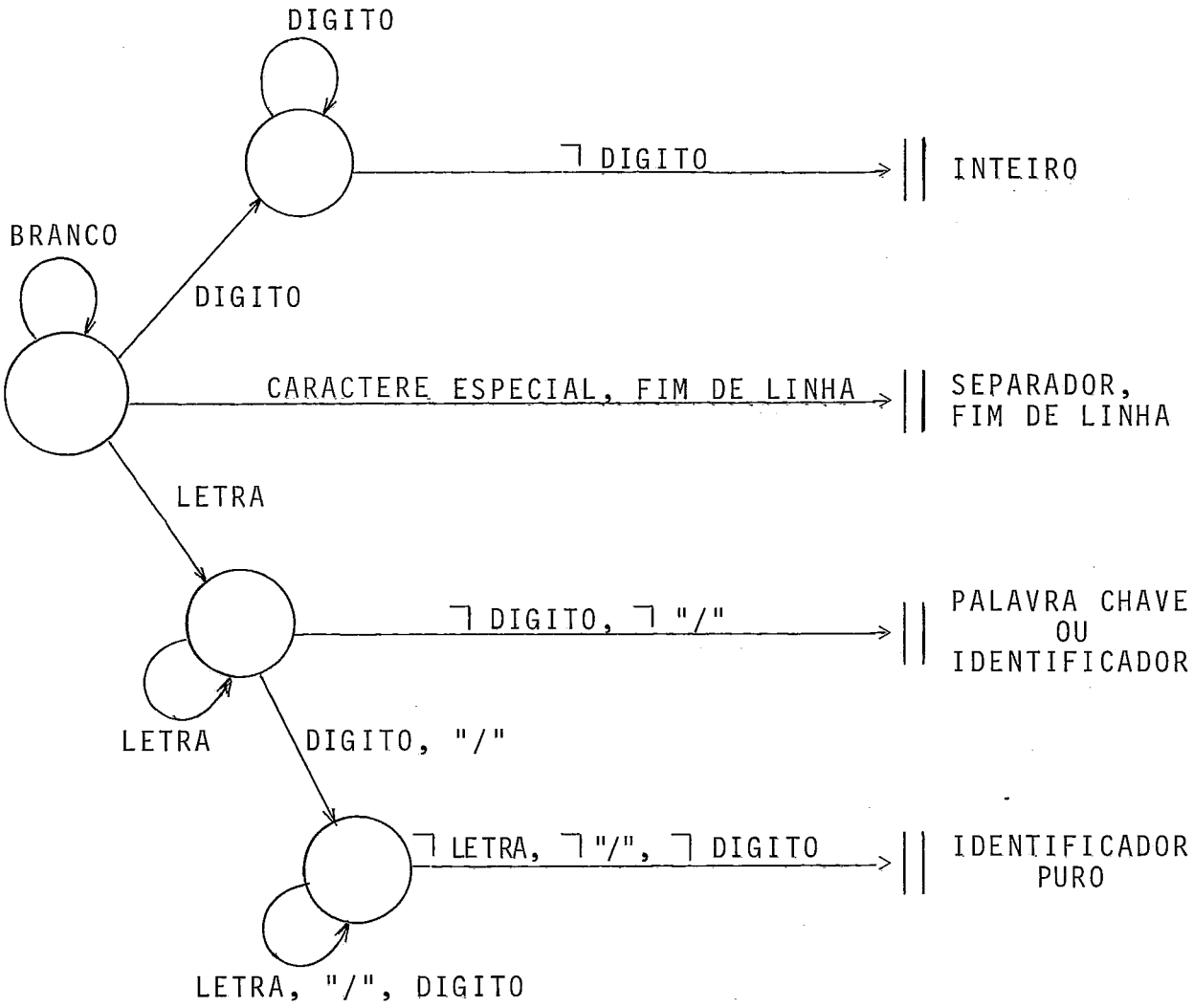



Fig. III-8: Autômato finito do analisador léxico

Tradução

A tradução para cada nó é efetuada logo após a análise sintática, na mesma instrução WHILE graças a uma nova instrução CASE. Por ser costume, chamaremos as ações de tradução de ações semânticas.

Foram isolados 26 tipos de ações semânticas que têm como objetivo montar o buffer de semântica e detetar erros de semântica. Demos um nome a cada um desses 26 tipos de ações, mas na realidade são somente 19 tipos de ações diferentes.

Limitamos o buffer de semântica a 91 bytes. Ele contém, entre outros elementos, uma pilha que permite armazenar em binário até 10 duplas de números de até 6 dígitos decimais significativos (números de linha, incrementos). Representamos a seguir esse buffer com os nomes de ações semânticas utilizadas na descrição dos grafos.

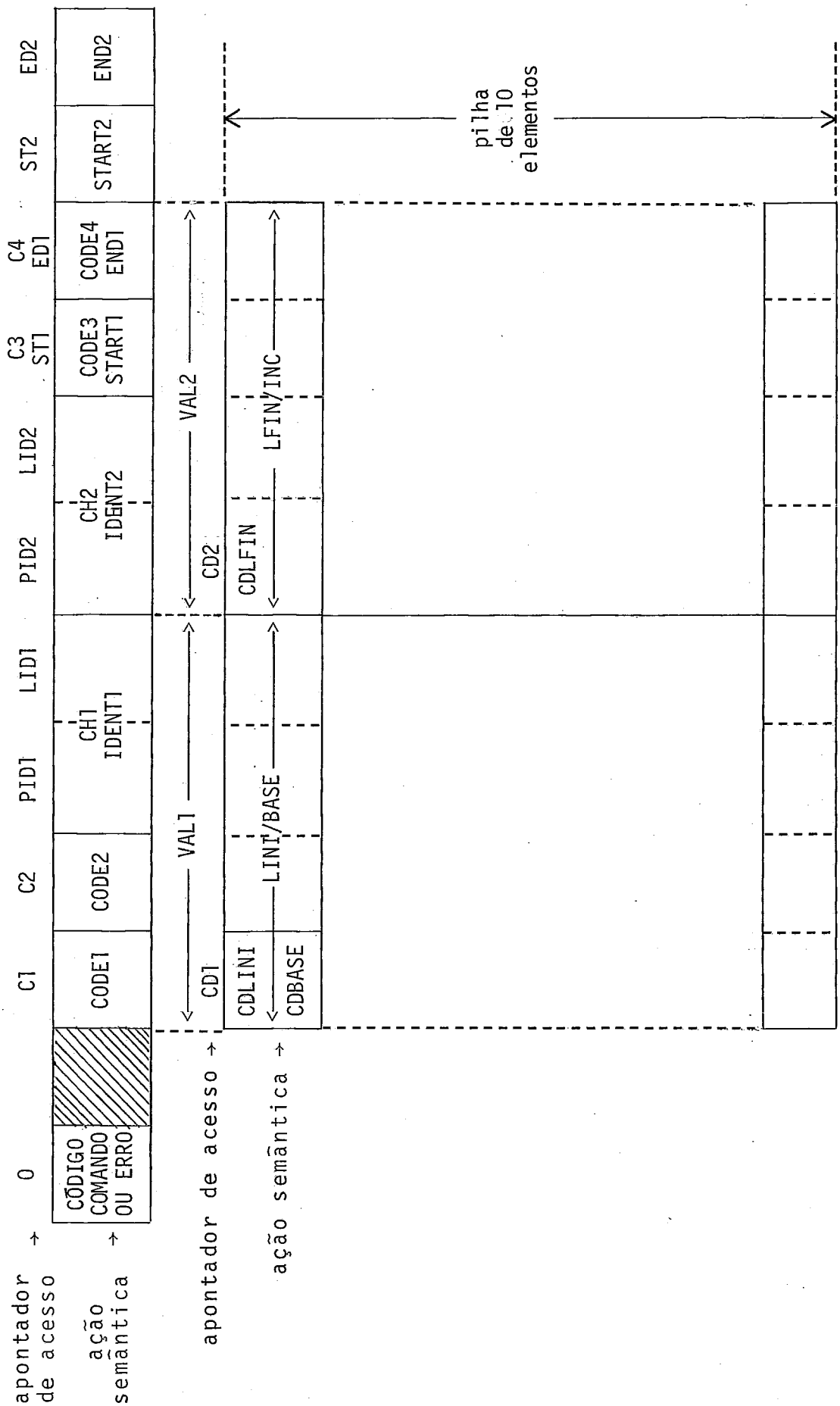



Fig.III-9: Buffer de Semântica

## Algoritmo de Tradução

O buffer de semântica é inicializado por 91 bytes de valor hexadecimal FF.

O primeiro byte do buffer de semântica é função do código do token inicial do comando; recebe o valor zero em caso de deteção de erro. Esse primeiro byte guiará um desvio indexado que permitirá a seleção do executivo do respectivo comando.

No algoritmo de análise semântica detalhado a seguir NOSM é o valor inicial do NÓ a cada passo, essa última variável podendo ser alterada durante a análise sintática anterior.

```
Case      AÇÃOSM[NÓ]
of
CODE1:    BSM(C1):=SIMB[NOSM];          % as posições C1, C2, C3 ou C4 do
CODE2:    BSM(C2):=SIMB[NOSM];          % buffer de semântica recebem o
CODE3:    BSM(C3):=SIMB[NOSM];          % símbolo do nó.
CODE4:    BSM(C4):=SIMB[NOSM];
IDENT1:   % comum também a CH1
          BSM (PID1):=PINI;             % PINI e PFIN são os apontadores
          BSM (LID1):=PFIN-PINI;        % de início e fim de token de
          if LIDENT>8                   % terminados pelo SCANNER
          and CODE≠FIND
          and CODE≠REPLACE
          then ERROIDENT;
IDENT2:   % comum também a CH2
          BSM (PID2):=PINI;
          BSM (LID2):=PFIN-PINI;
```

```
            if LIDENT > 8
            and CODE ≠ FIND
            and CODE ≠ REPLACE
            then ERROIDENT;
START1:     if VALOR < 1 or VALOR > 72 then ERROCOL;
            BSM (ST1):= VALOR;              % VALOR ocupa um só byte
END1:       if VALOR < 1 or VALOR > 72 then ERROCOL;
            BSM (ED1):= VALOR;
STAR2:      if VALOR < 1 or VALOR > 72 then ERROCOL;
            BSM (ST2):= VALOR;
END2:       if VALOR < 1 or VALOR > 72 then ERROCOL;
            BSM (ED2):= VALOR;
LINI:       % comum também a BASE, KEY e COL
            if VALOR > 999999 then ERROOVERFLOW;
            BSM (VAL1 + I):= VALOR;         % VALOR nesse caso ocupa 4 bytes
LFIN:       % comum também a INC
            if VALOR > 999999 then ERROOVERFLOW;
            BSM (VAL2 + I):= VALOR;
CDLINI:     % comum também a CDBASE
            BSM (CD1 + I):= SIMB[NOSM];
CDLFIN:     BSM (CD2 + I):= SIMB[NOSM];
INC8I:      I:= I + 8;                      % incrementa o apontador de acesso
            if I > 72 then ERROPILHA;       % à pilha
INCJ:       J:= J + 1;
            if J > 1 then  ERRO2VEZES;
INCK:       K:=K+1;
            if K > 1 then ERRO2VEZES;
INCL:       L:=L+1;
            if L > 1 then ERRO2VEZES;
MINI1:      if J = 0 then ERRO1MINI;        % J não foi incrementado ne-
                                            % nhuma vez
end
```

## Grafos de Análise Sintática e Tradução dos Comandos

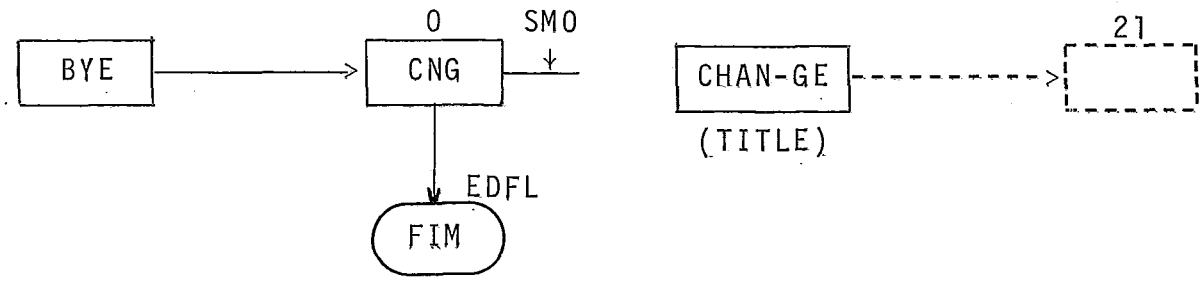
A seguir mostraremos para cada comando o seu grafo sintático e tradução.

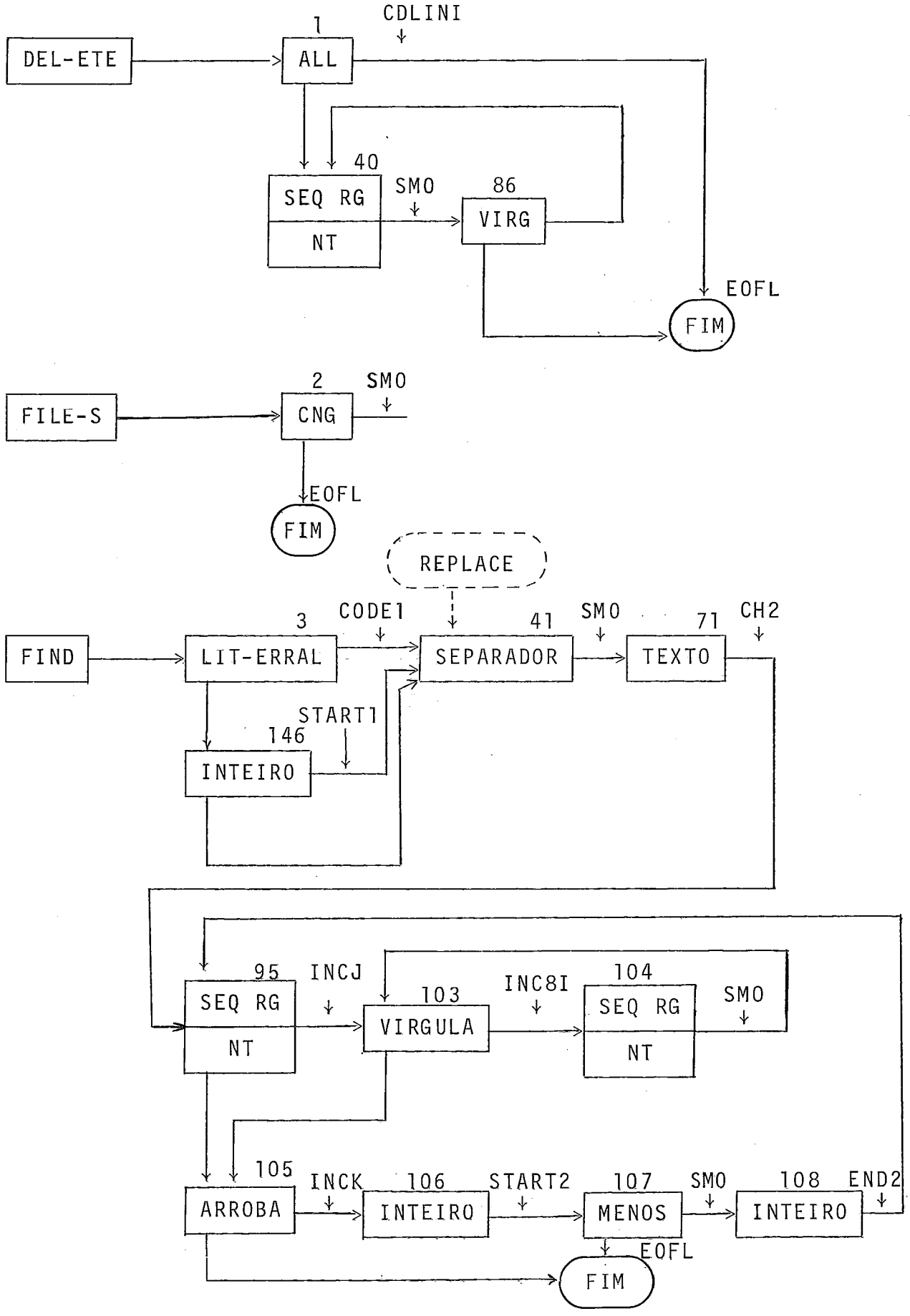
O símbolo de nó CNG (código não gerado) não tem código correspondente gerado pelo SCREENER, o que permite forçar o analisador para o nó alternativo do nó corrente (utilizado nos comandos BYE, FILES, SET, UPDATE, WHAT).

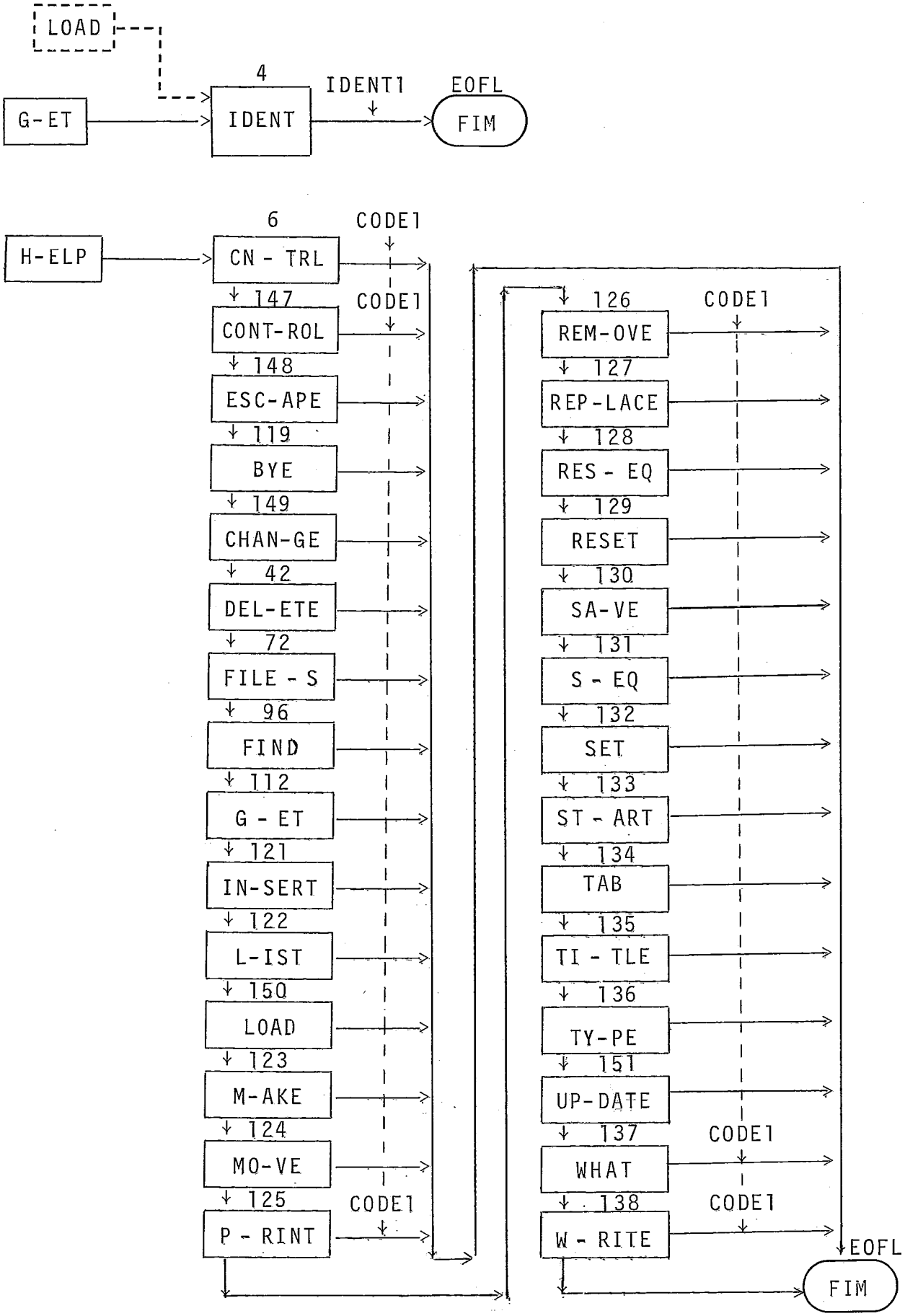
As indicações de ação semântica são sempre colocadas na saída de um nó para seu sucessor. Efetivamente, apenas no caso de o analisador prosseguir para o nó sucessor será realizada uma ação semântica. No caso de o analisador continuar para o nó alternativo não haverá nenhuma ação semântica realizada.

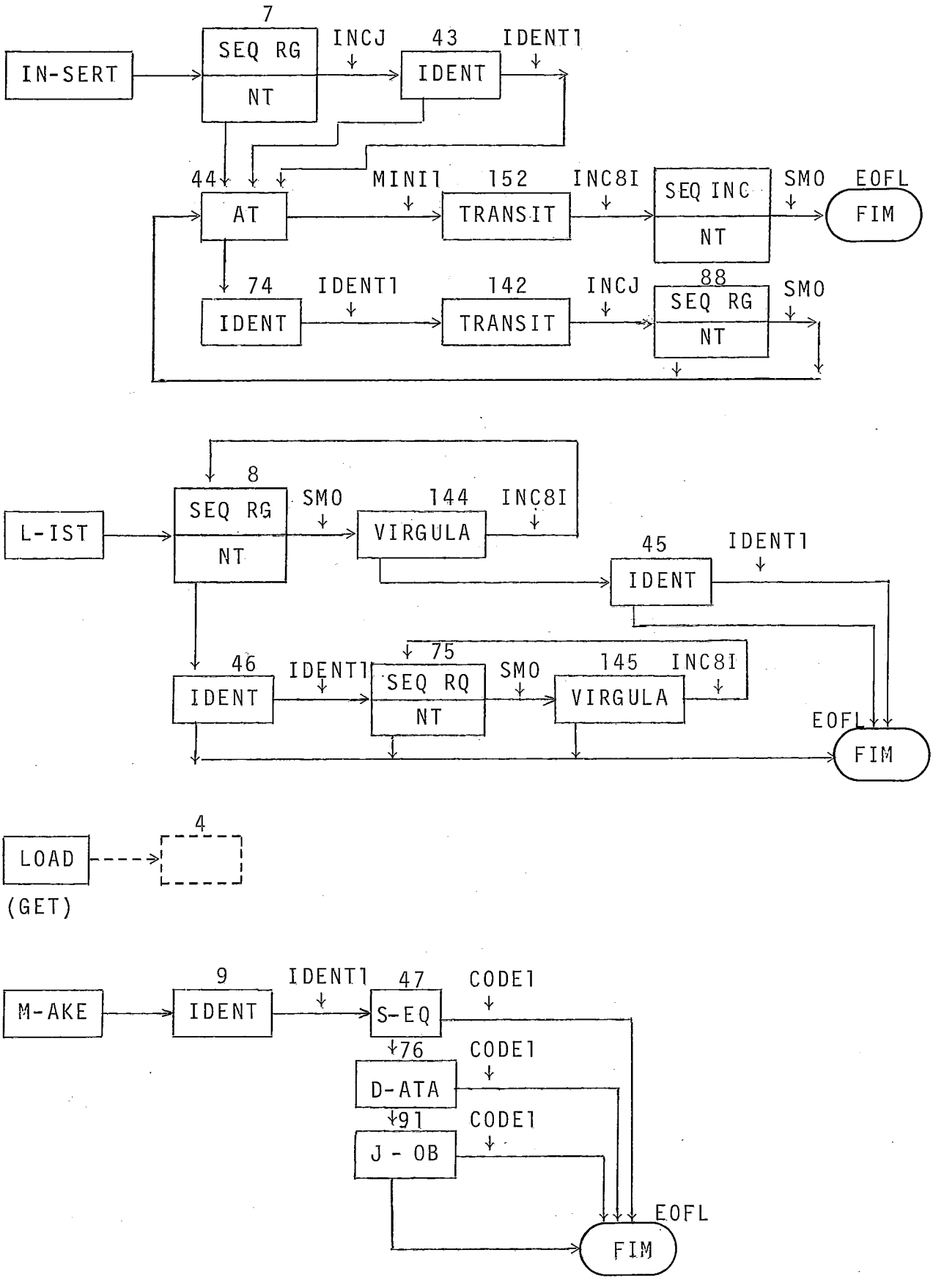
A ação semântica indicada por SMO é uma ação semântica sem efeito.

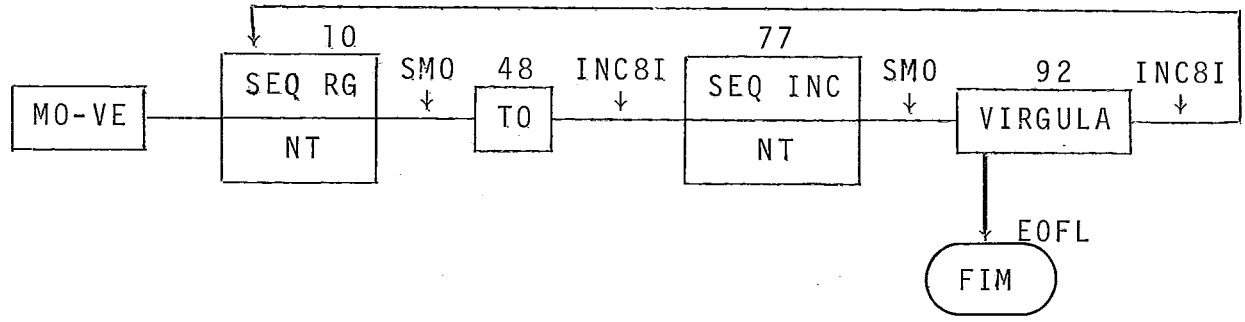
Em certos nós tivemos necessidade de implementar duas ações semânticas, o que o algoritmo de análise não permitia. Por isso desdobramos certos nós e assim aparece em certos comandos o nó chamado TRANSIT cuja única função é permitir acrescentar mais uma ação semântica (utilizado nos comandos INSERT, RESEQ, WRITE).

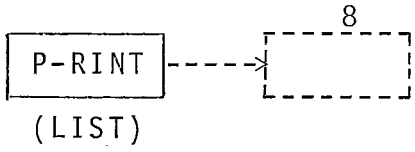


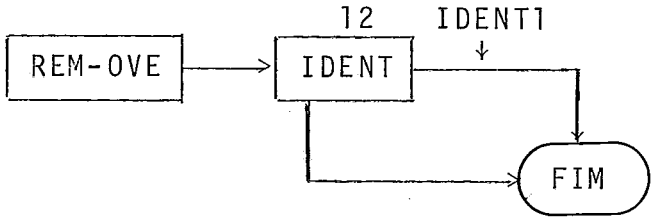

DEL-ETE
1
ALL
CDLINI
40
SEQ RG
NT
SMO
86
VIRG
EOFL
FIM
FILE-S
2
CNG
SMO
EOFL
FIM
REPLACE
FIND
3
LIT-ERRAL
CODE1
41
SEPARADOR
SMO
71
TEXTO
CH2
START1
146
INTEIRO
95
SEQ RG
NT
INCJ
103
VIRGULA
INC8I
104
SEQ RG
NT
SMO
105
ARROBA
INCK
106
INTEIRO
START2
107
MENOS
SMO
108
INTEIRO
END2
EOFL
FIM

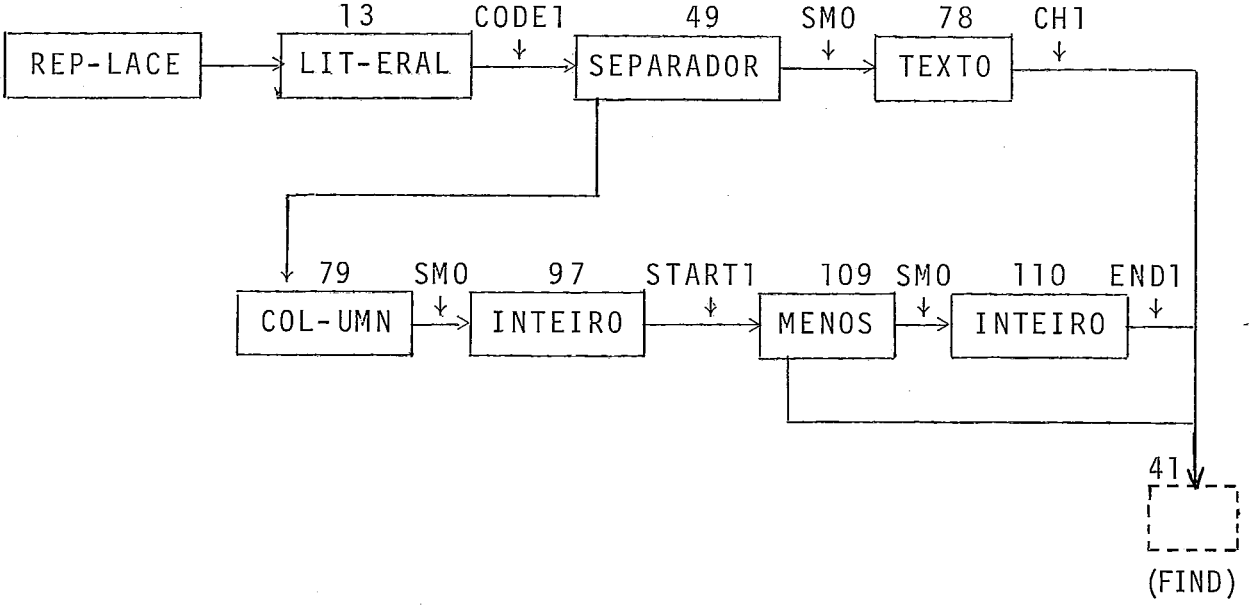

LOAD
G-ET
4
IDENT
IDENT1
EOFL
FIM
H-ELP
6
CN - TRL
CODE1
147
CONT-ROL
148
ESC-APE
119
BYE
149
CHAN-GE
42
DEL-ETE
72
FILE - S
96
FIND
112
G - ET
121
IN-SERT
122
L-IST
150
LOAD
123
M-AKE
124
MO-VE
125
P - RINT
126
REM-OVE
127
REP-LACE
128
RES - EQ
129
RESET
130
SA-VE
131
S - EQ
132
SET
133
ST - ART
134
TAB
135
TI - TLE
136
TY-PE
151
UP-DATE
137
WHAT
138
W - RITE
EOFL
FIM

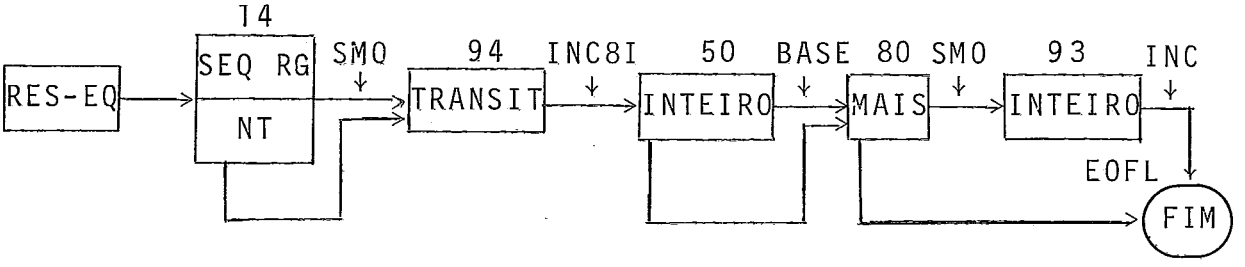

IN-SERT
7
SEQ RG
NT
INCJ
43
IDENT1
IDENT
44
AT
MINI1
152
TRANSIT
INC8I
SEQ INC
NT
SMO
EOFL
FIM
74
IDENT
IDENT1
142
TRANSIT
INCJ
88
SEQ RG
NT
SMO
L-IST
8
SEQ RG
NT
SMO
144
VIRGULA
INC8I
45
IDENT
IDENT1
46
IDENT
IDENT1
75
SEQ RQ
NT
SMO
145
VIRGULA
INC8I
EOFL
FIM
LOAD
4
(GET)
M-AKE
9
IDENT
IDENT1
47
S-EQ
CODE1
76
D-ATA
CODE1
91
J - OB
CODE1
EOFL
FIM

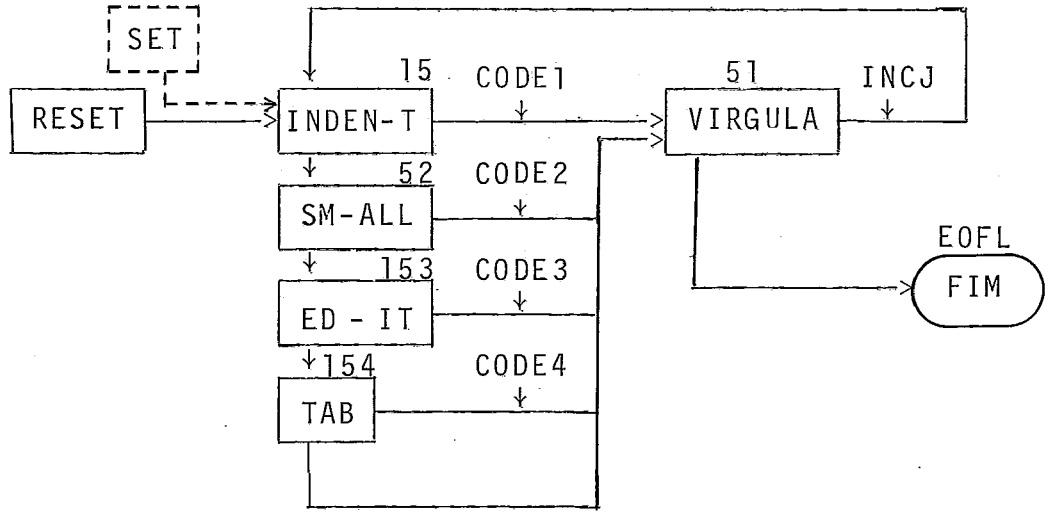

10
SEQ RG
NT
SMO
48
TO
INC8I
77
SEQ INC
NT
SMO
92
VIRGULA
INC8I
MO-VE
EOFL
FIM

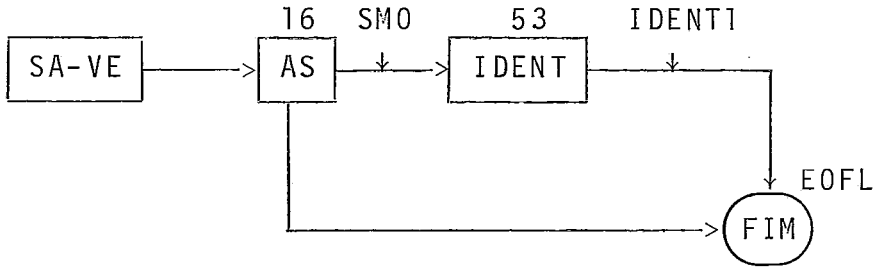

8
P-RINT
(LIST)

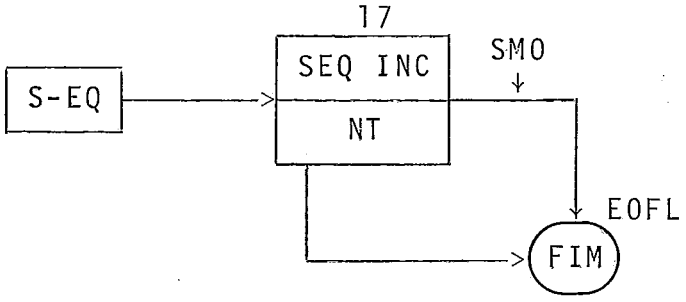

12
IDENT1
REM-OVE
IDENT
FIM

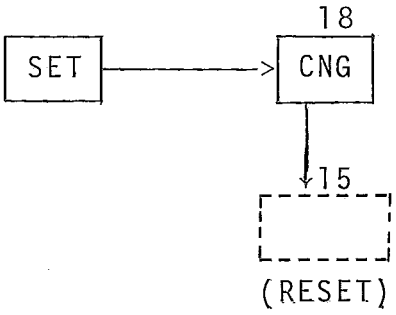

13
CODE1
49
SMO
78
CH1
REP-LACE
LIT-ERAL
SEPARADOR
TEXTO
79
SMO
97
START1
109
SMO
110
END1
COL-UMN
INTEIRO
MENOS
INTEIRO
41
(FIND)

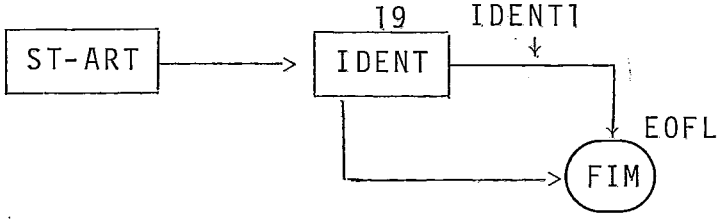

14
SEQ RG
NT
SMO
94
INC8I
50
BASE
80
SMO
93
INC
RES-EQ
TRANSIT
INTEIRO
MAIS
INTEIRO
EOFL
FIM

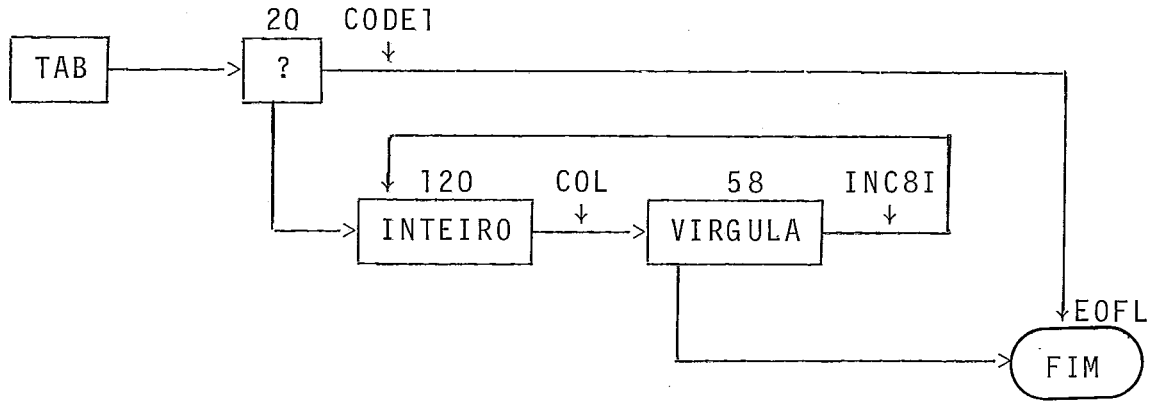

SET
RESET
15
INDEN-T
CODE1
51
VIRGULA
INCJ
52
SM-ALL
CODE2
EOFL
FIM
153
ED - IT
CODE3
154
TAB
CODE4

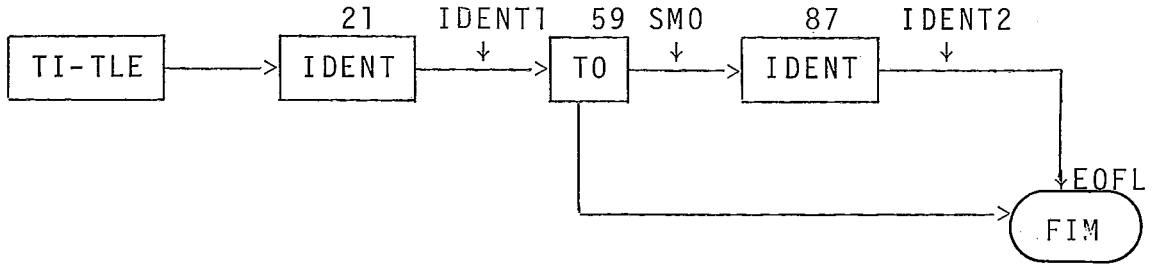

SA-VE
16
AS
SMO
53
IDENT
IDENT1
EOFL
FIM

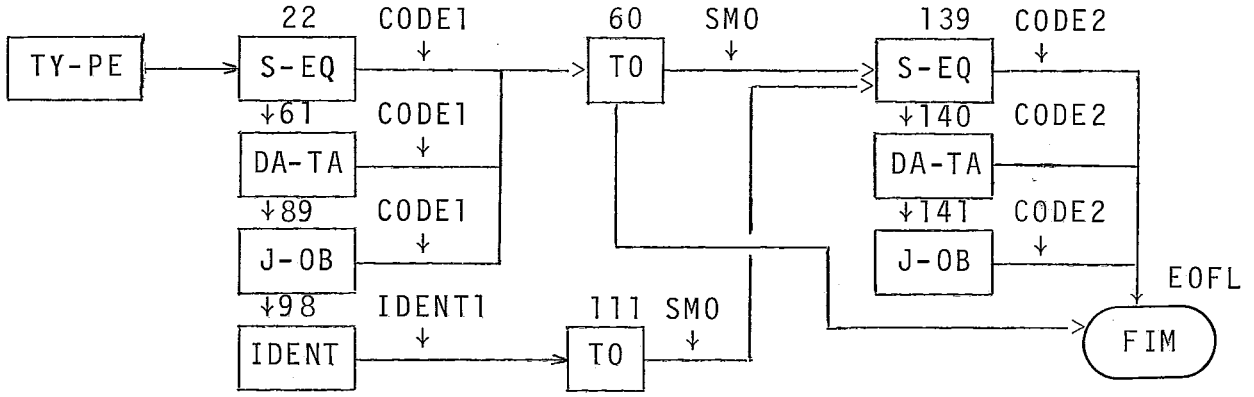

17
S-EQ
SEQ INC
NT
SMO
EOFL
FIM

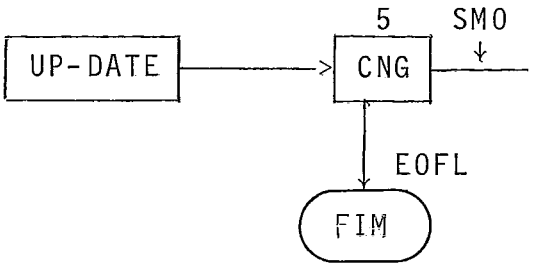

18
SET
CNG
15
(RESET)

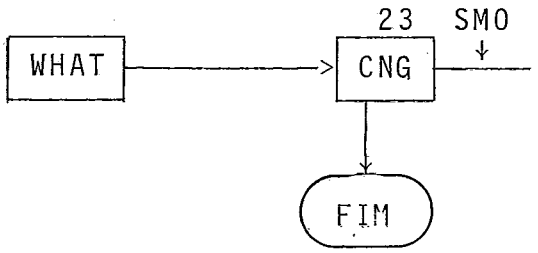

19
ST-ART
IDENT
IDENT1
EOFL
FIM

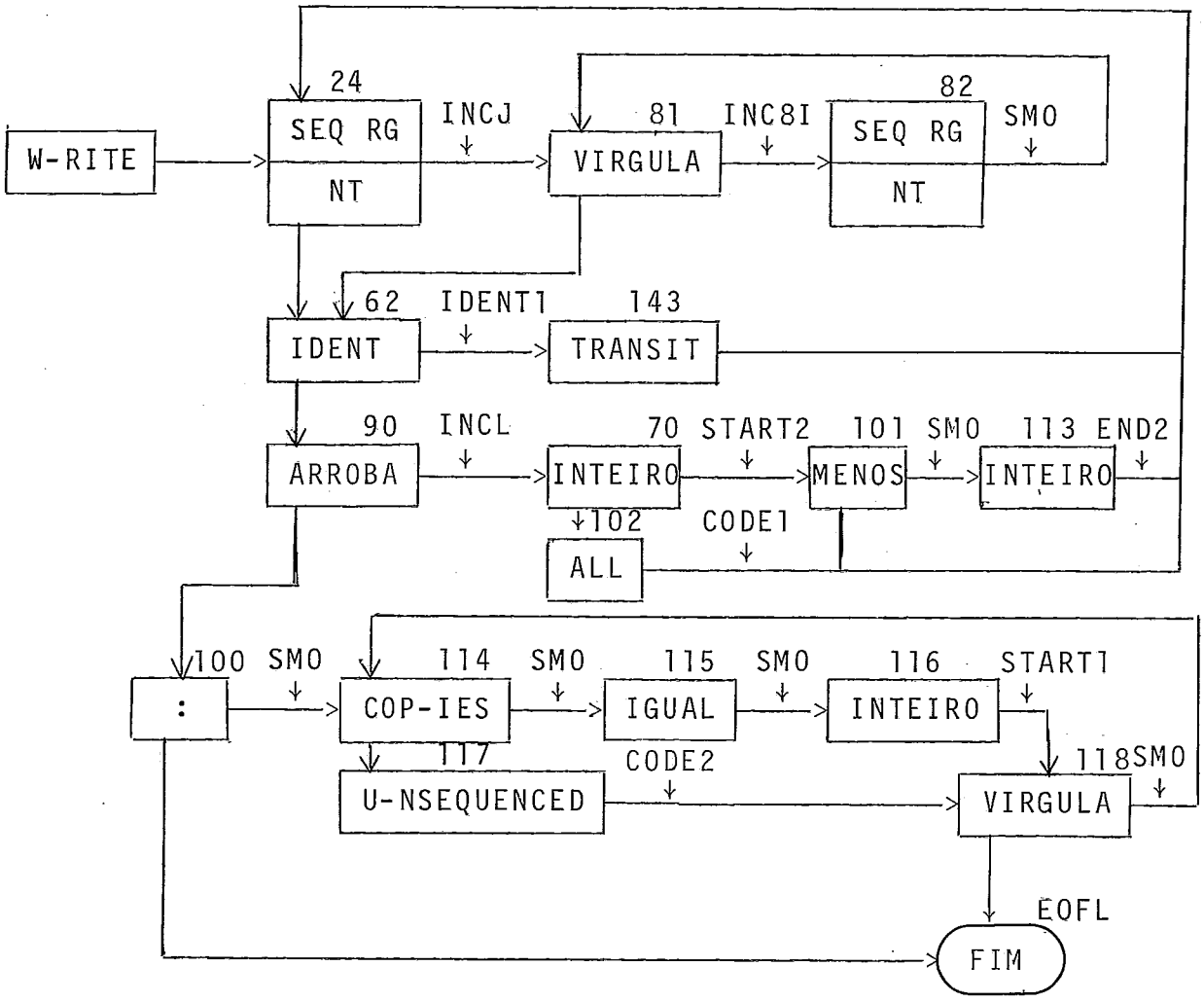

TAB
20
?
CODE1
120
INTEIRO
COL
58
VIRGULA
INC8I
EOFL
FIM


TI-TLE
21
IDENT
IDENT1
59
TO
SMO
87
IDENT
IDENT2
EOFL
FIM


TY-PE
22
S-EQ
CODE1
↓61
DA-TA
CODE1
↓89
J-OB
CODE1
↓98
IDENT
IDENT1
60
TO
SMO
111
TO
SMO
139
S-EQ
CODE2
↓140
DA-TA
CODE2
↓141
J-OB
CODE2
EOFL
FIM

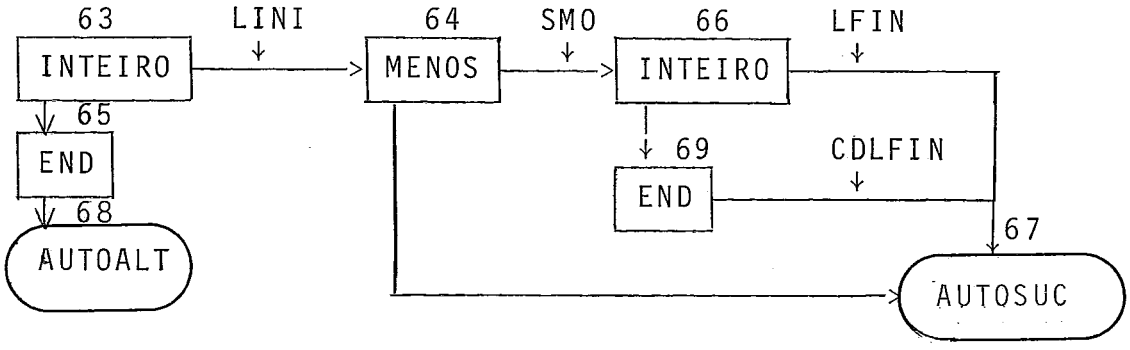

UP-DATE
5
CNG
SMO
EOFL
FIM


WHAT
23
CNG
SMO
FIM

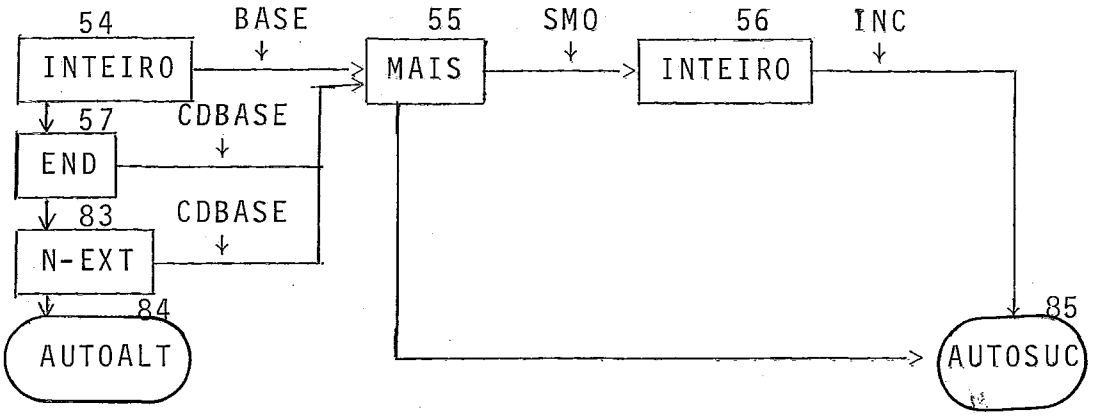



NAO-TERMINAIS

<SEQ RG>



<SEQ INC>




Ocupação Memória e Avaliação do Analisador

Tendo apenas 7 tipos de ações sintáticas (ação nula incluída) e 20 tipos de ações semânticas (ação nula incluída), conseguimos incluir num mesmo byte da tabela do PARSER as duas espécies de ações, graças à distribuição 3 bits/5 bits, entre elas.

Assim, cada nó do PARSER ocupa somente 4 bytes na tabela (símbolo do nó, nó alternativo, nó sucessor, ações).

O analisador foi programado na linguagem estruturada de nível médio LP15E (CII[10]) que tem um bom rendimento na geração de código máquina. Numa das últimas versões do analisador (com 144 nós) foram feitas as medidas de ocupação memória resumidas na seguinte tabela:

| | dados | instruções máquina | observações |
|---|---|---|---|
| PARSER + tradutor | 666 bytes (584 bytes para tabela do analisador) | 301 | análise sintática: 138 (fase inicial: 57; fase principal: 81)<br>análise semântica: 99<br>procedures de tratamento de erro: 64 |
| SCREENER | 338 bytes (309 bytes para as palavras chaves) | 71 | chama também um módulo de comparação de cadeias de 22 instruções do S.O.<br>chama também um subprograma de conversão de cadeia decimal para binário de 55 instruções do editor |
| SCANNER | 76 bytes (65 bytes para tabela de correspondência) | 108 | |
| Total | 1,080 Kbytes | 480 (960 bytes) | |

O analisador ocupando por volta de apenas 2 Kbytes, podemos considerar seu tamanho como excelente. Quanto à sua velocidade, o usuário achará sua ação instantânea. Na verdade é difícil sentir fisicamente a velocidade de um algoritmo simples como o que implementamos, o qual não efetua nenhuma operação de entrada saída e gasta somente CPU de um computador que executa por volta de 300.000 a 400.000 instruções por segundo.

## III.3 - Sistema Operacional e Arquivos

### Sistema Operacional

O sistema operacional do Mitra 15 suportando o editor é do tipo tempo real disco (MTRD) (CII[12]). O presente projeto foi iniciado pela implantação de uma nova versão 7.4 desse tipo de sistema operacional, conseguida nessa época na França, que continua sendo a última versão disponível nesse tipo de computador. Um sistema operacional de tipo multitarefa (MMT), sofisticação do MTRD, não foi escolhido por ser de tamanho bem maior enquanto os sistemas operacionais de base (MOB) e tempo real (MTR) de tamanhos bem menores foram também eliminados por não apresentarem recursos mínimos de gerenciamento de disco (CII[11]).

A grande novidade da versão 7.4 do MTRD em relação à versão 6 então implantada no Mitra 15 do Laboratório de Sistemas da COPPE é a possibilidade de gerenciamento em disco de arquivos seqüenciais indexados.

As principais funções do MTRD são as seguintes:

- controle do computador (tratamento das interrupções, diálogo com operador);
- carregamento e lançamento de programas;
- gerenciamento das entradas/saídas;
- funções de serviço (conversões, dump, tratamento de cadeias, ...).

Essas funções são tratadas de um ponto de vista de tempo real graças a:

- gerenciamento memória de zonas foreground, background e comum;
- conexão de programas a um nível de interrupção;
- multiprogramação pelo dispositivo de prioridade hardware das interrupções;
- filas de espera;
- gerenciamento de recursos (reserva, teste, liberação);
- gerenciamento de eventos;
- módulos reentrantes;
- gerenciamento de um disco sistema e de overlays;
- gerenciamento de arquivos seqüenciais e seqüências indexados.

Todas as entradas/saídas para periféricos padrões (console operador, leitora de cartões, impressora, discos, linhas de comunicações) são microprogramadas (CII[9]).

A inicialização, o lançamento e o controle do microprograma de entrada/saída de cada periférico é deixado a cargo de um módulo específico do sistema operacional chamado "handler" conectado a um nível de interrupção próprio.

Para se liberar de um contexto físico rígido, o acesso às entradas/saídas no nível do usuário é feito através de um contexto lógico graças a um sistema de etiquetas operacionais (OL).

Para o editor geramos um sistema operacional específico (chamado EDIT.74) possuindo unicamente os recursos indispensáveis ao funcionamento do editor (CII[13], CII[14]).

O núcleo MCCF.74 escolhido contém somente 44 módulos reentrantes dos 108 possíveis e é o menor núcleo disponível, permitindo o gerenciamento de arquivos em disco. Junto com a sua PRTS que consiste numa tabela de apontadores de acesso aos diversos módulos, ele ocupa 14,132 Kbytes (sempre consideraremos 1 Kbytes = 1000 bytes).

Os dois handlers "console" e "painel" são sempre indispensáveis em qualquer sistema operacional de qualquer tipo. Juntos com os três handlers "leitora de cartões", "disco" e "linhas assíncronas" ocupam 5,176 Kbytes, mesmo espaço ocupado pelos três últimos módulos, TRAP cujo papel é simular instruções inexistentes, M:IOX que é o módulo específico de tratamento das entradas/saídas sobre arquivos seqüenciais indexados e LPTES, a tabela de gerenciamento das entradas/saídas.

O espaço ocupado pelo sistema operacional totalmente residente atinge 24,484 Kbytes. A necessidade de buffers encadeados obrigou-nos a incorporar o maior handler de linhas assíncronas, ultrapassando em 0,936 Kbytes o menor.

Para não aumentar ainda mais o tamanho do sistema

operacional, não foram incorporadas as seguintes funções:

- simulação de instruções de ponto flutuante (no módulo TRAP);
- módulos de depuração (no núcleo);
- desdobramento de etiquetas operacionais Background / Foreground (na tabela de entradas / saídas);
- handler relógio tempo real;
- módulo de tratamento de queda de tensão;
- módulo de tratamento de volta de tensão.

A tabela de entradas/saídas foi gerada com somente três etiquetas operacionais usuário (UL) além das 16 padrões (OL), limitando também a 5 os recursos para arquivos e a 4 o número de linhas assíncronas.

Uma reserva mínima de 806 bytes é também necessária em zona comum para o bom funcionamento do sistema operacional e dos processadores rodando em Background.

Eliminando totalmente o Foreground e com memória inicial de 32 Kbytes, sobraram para implantar o editor 7,478 Kbytes.

A figura III-10a mostra a organização geral da memória do Mitra 15 e a figura III.10b detalha a posição e o tamanho

dos principais módulos do sistema operacional usados pelo editor.

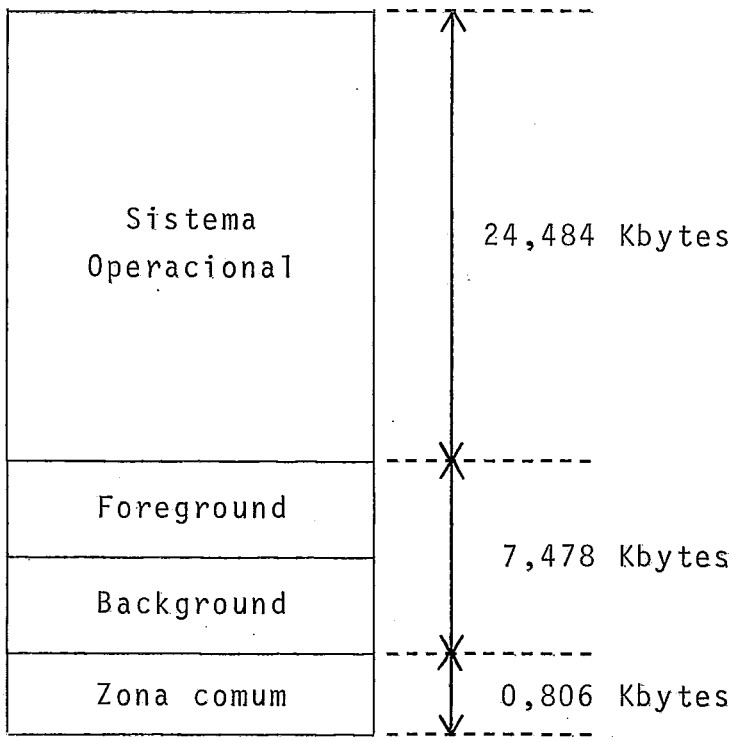



Fig.III.10a: Organização da memória do Mitra 15

Arquivos Seqüenciais Indexados

O sistema operacional EDIT.74 permite gerenciar um disco sistema dividido em 6 zonas distintas (ver fig.III-11):

- SY: zona sistema onde está armazenado o código (IMA: imagem memória absoluta) dos diversos sistemas operacionais disponíveis;
- EP: zona de programas executáveis (IMT: imagem memória trasladável);

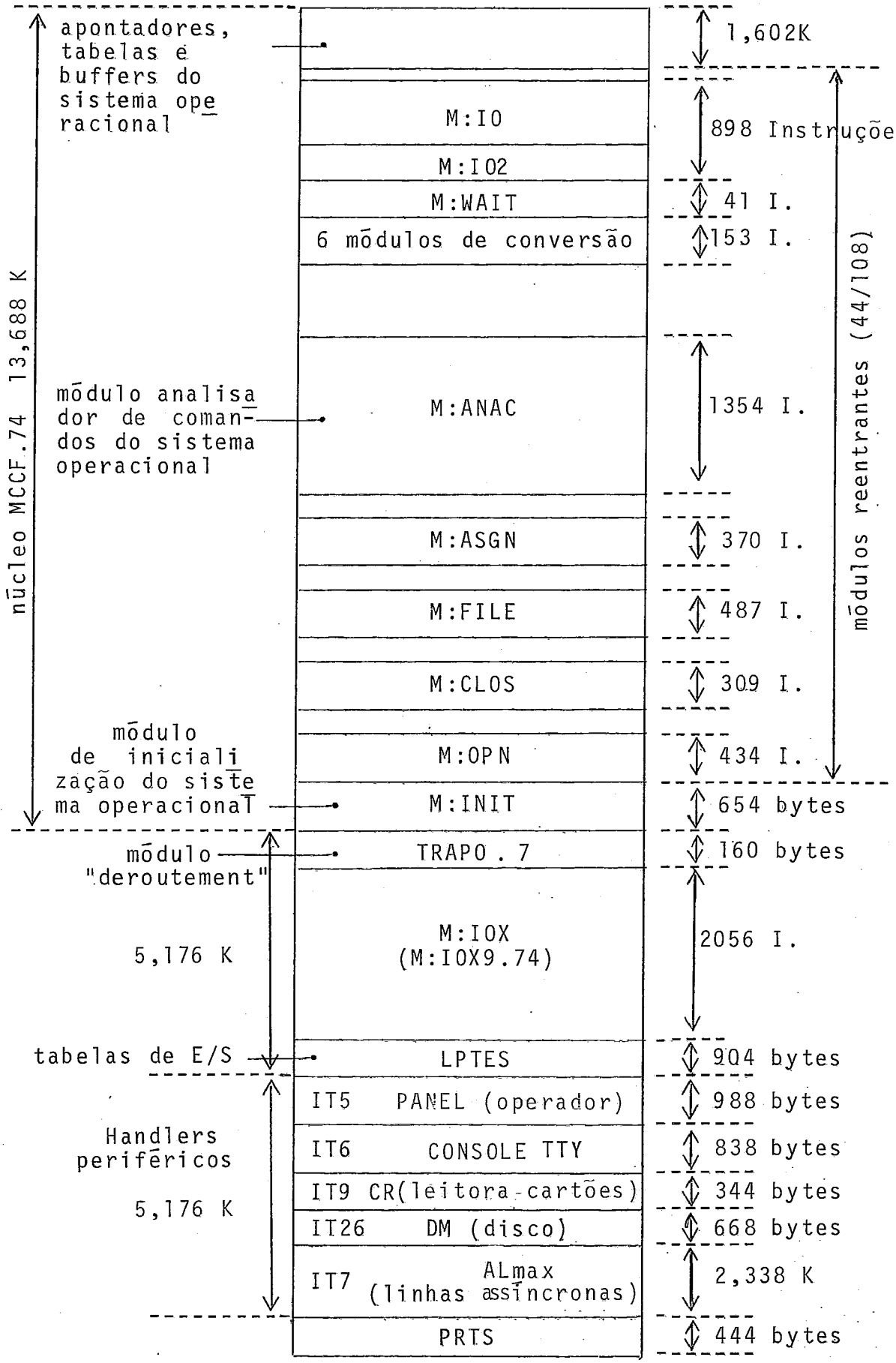



Fig. III.10b: Organização da zona de memória ocupada pelo Sistema Operacional

- SL: zona dos módulos sistema (BT: binário transladável) não linkados;
- UL: zona dos módulos usuário (BT) não linkados;
- GI-GO: zona de trabalho dos processadores (BT-IMT);
- DA: zona de trabalho dos usuários.

Os arquivos usuários são implantados na zona DA e gerenciados pelo sistema SGF 15 E (CII$^{15}$) que consiste num conjunto específico de módulos, ponteiros, tabelas e buffers do sistema operacional.

O SGF 15 E permite dois tipos de organização de arquivos:

- arquivos seqüenciais;
- arquivos seqüenciais indexados.

A alocação de espaço em disco é fixa para os dois tipos.

Os arquivos seqüenciais podem ser bloqueados ou não e permitir acesso seqüencial a registros, aleatório a registros e aleatório a setores de um arquivo.

Nesse tipo de arquivo o tamanho dos registros é fixo. Somente são permitidas a substituição de registros e a es-

crita de registros suplementares no final do arquivo. A remoção ou inserção de registros no meio do arquivo não são permitidas.

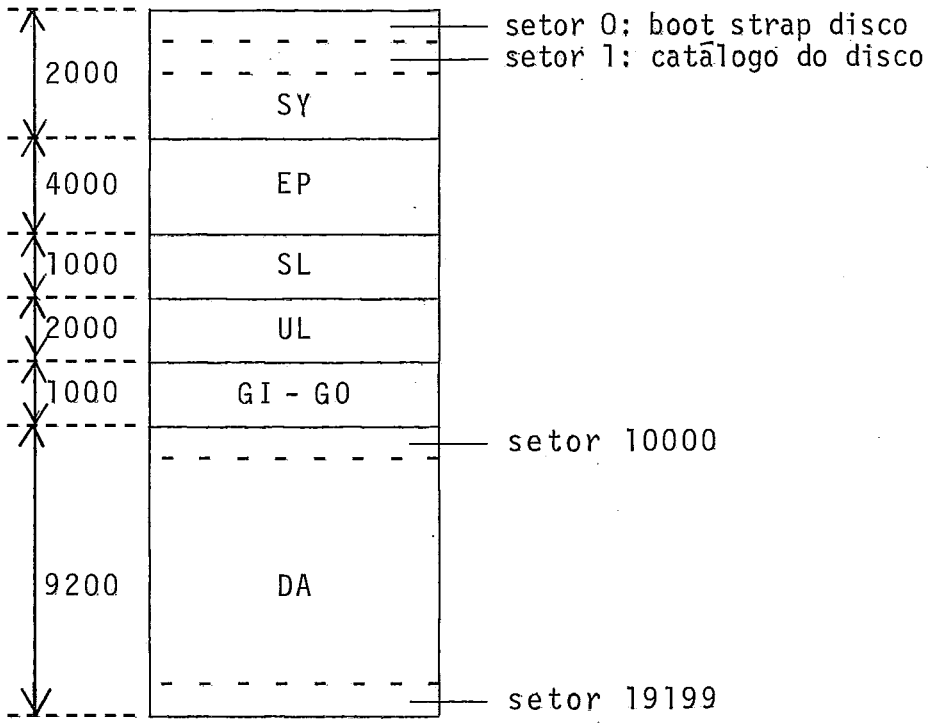



Fig.III-11: Organização do disco sistema
(os tamanhos em setores correspondem aos valores atuais)

Os arquivos seqüenciais indexados oferecem muito mais flexibilidade e apareceram como uma opção adequada para o presente trabalho.

Um arquivo seqüencial indexado é constituído de uma lista de registros, de tamanho variável, agrupados em blocos de registros e ordenados dentro de cada bloco de maneira seqüencial por ordem crescente das suas chaves.

A chave de um registro é definida como um campo do registro, de posição e tamanho determinados pelo usuário.

Os blocos de registros são encadeados entre eles somente no sentido crescente das chaves.

A chave do registro inicial de um bloco é utilizada como índice de acesso aos registros desse bloco.

Os índices são hierarquizados e agrupados também em blocos. Cada bloco de nível i resume os blocos de nível i-1. Os blocos de mais baixo nível (nível 1) apontam os blocos de dados. A cada nível a tabela de índices é constituída de um a vários blocos.

O usuário deve fornecer um buffer de blocagem/deblocagem dos registros e também um buffer de índices.

Uma zona livre ou padding, tanto para blocos de dados quanto para blocos de índices, inocupada na criação do arquivo facilita posteriores inserções de registros.

A alocação de espaço em disco sendo também fixa para esse tipo de arquivo, os blocos de registros são alocados seqüencialmente a partir do início da zona disco reservada para o arquivo. Os blocos de índices são alocados seqüencialmente a partir do fim do arquivo (ver fig.III-12).

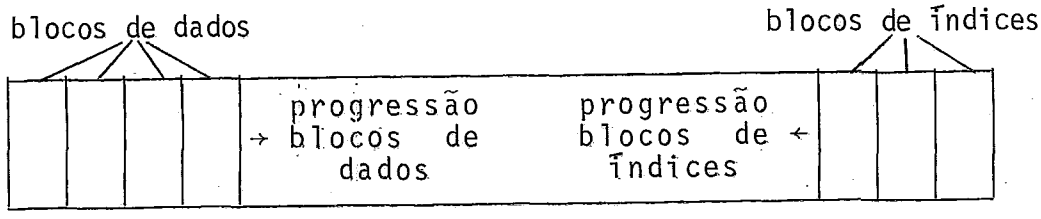



Fig.III-12: Alocação de blocos no arquivo

A organização seqüencial indexada permite substituição, inserção e remoção de registros em qualquer lugar do arquivo, operações que são a base de todo editor. A possibilidade de tamanho de registro variável permite a compactação dos registros a fim de otimizar o uso do disco.

Adotamos então esse tipo de organização para o editor, acertando os diversos parâmetros para o nosso caso específico.

A chave de cada registro foi escolhida como sendo o número da linha correspondente a ser mostrada na tela. A chave é um número binário de 4 bytes amplamente suficiente para representar números de linha de até 6 dígitos decimais significativos.

A posição das chaves foi escolhida no início dos registros para minimizar o tamanho de cada um deles.

Os blocos de dados e de índices foram escolhidos de tamanho igual a 1 setor a fim de limitar ao mínimo permitido os tamanhos dos buffers de trabalho requisitados nas operações de entradas/saídas.

Comprometendo um pouco a velocidade média de inserção de novos registros, não determinamos padding de blocos de dados nem de blocos de índices a fim de minimizar a ocupação do disco e aumentar a velocidade de listagem na tela, função mais usada do editor.

A seguir, uma apresentação da estrutura de um bloco de registros e de um registro usados no editor:

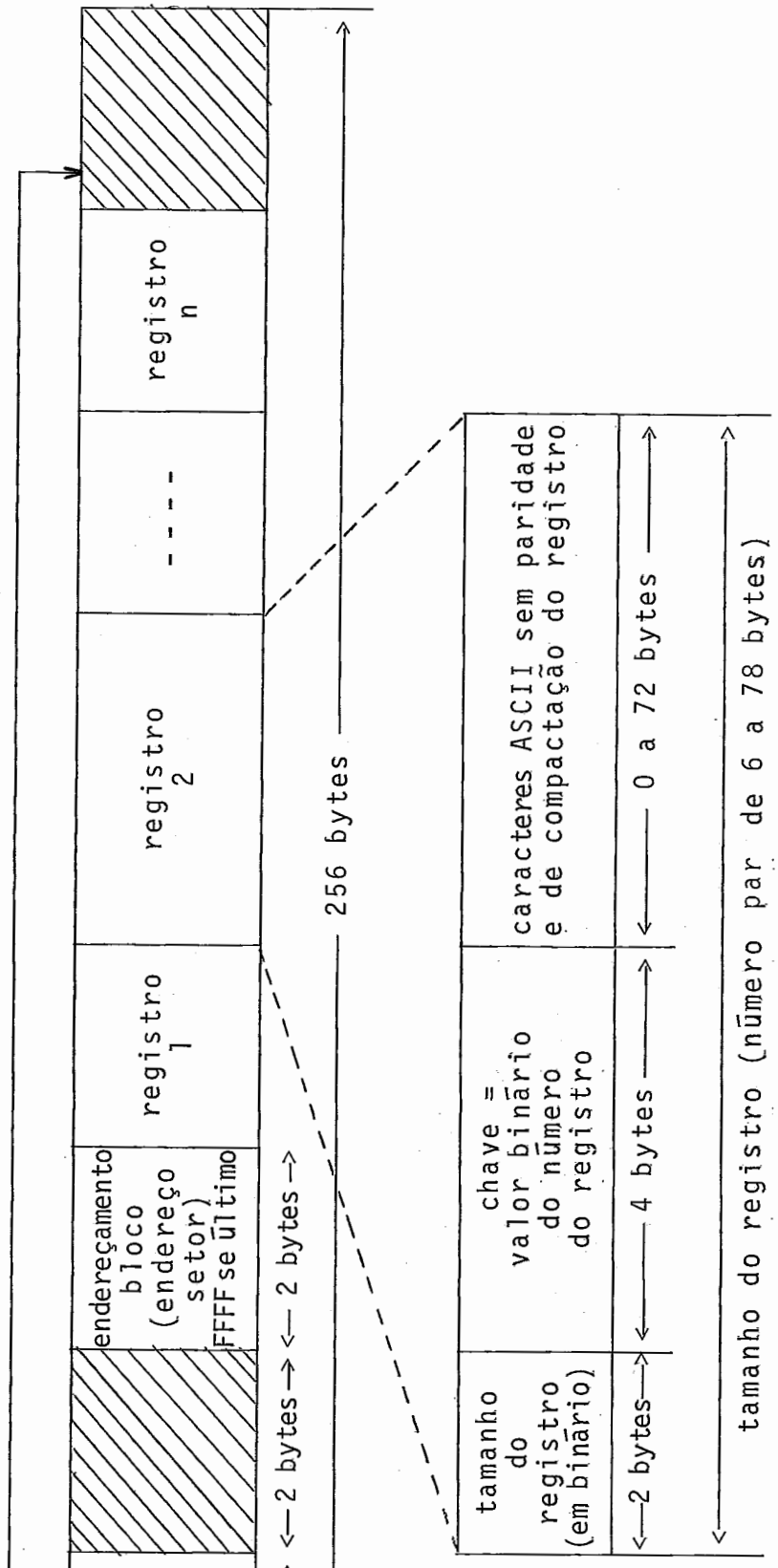



Fig. III-13: Estrutura de um bloco de registros e de um registro

Num bloco de registros cabem de 3 registros (linha com 72 caracteres úteis) a 41 registros (linhas brancas).

Um bloco de índices e um dos seus elementos são organizados da forma seguinte:

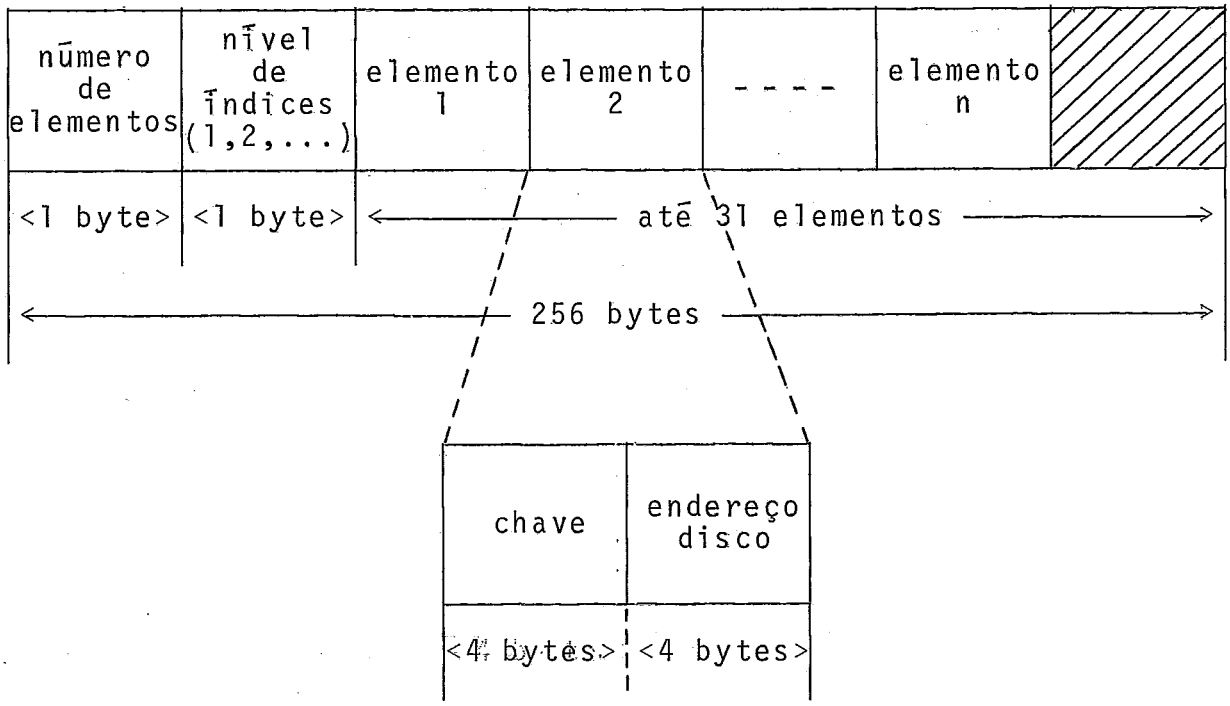



Fig.III-14: Estrutura de um bloco de índices e de um de seus elementos

Num bloco de índices cabem, no máximo, 31 elementos.

Agora daremos um exemplo de organização de um arquivo com uma tabela de índices de 2 níveis, caso mais comum no editor:

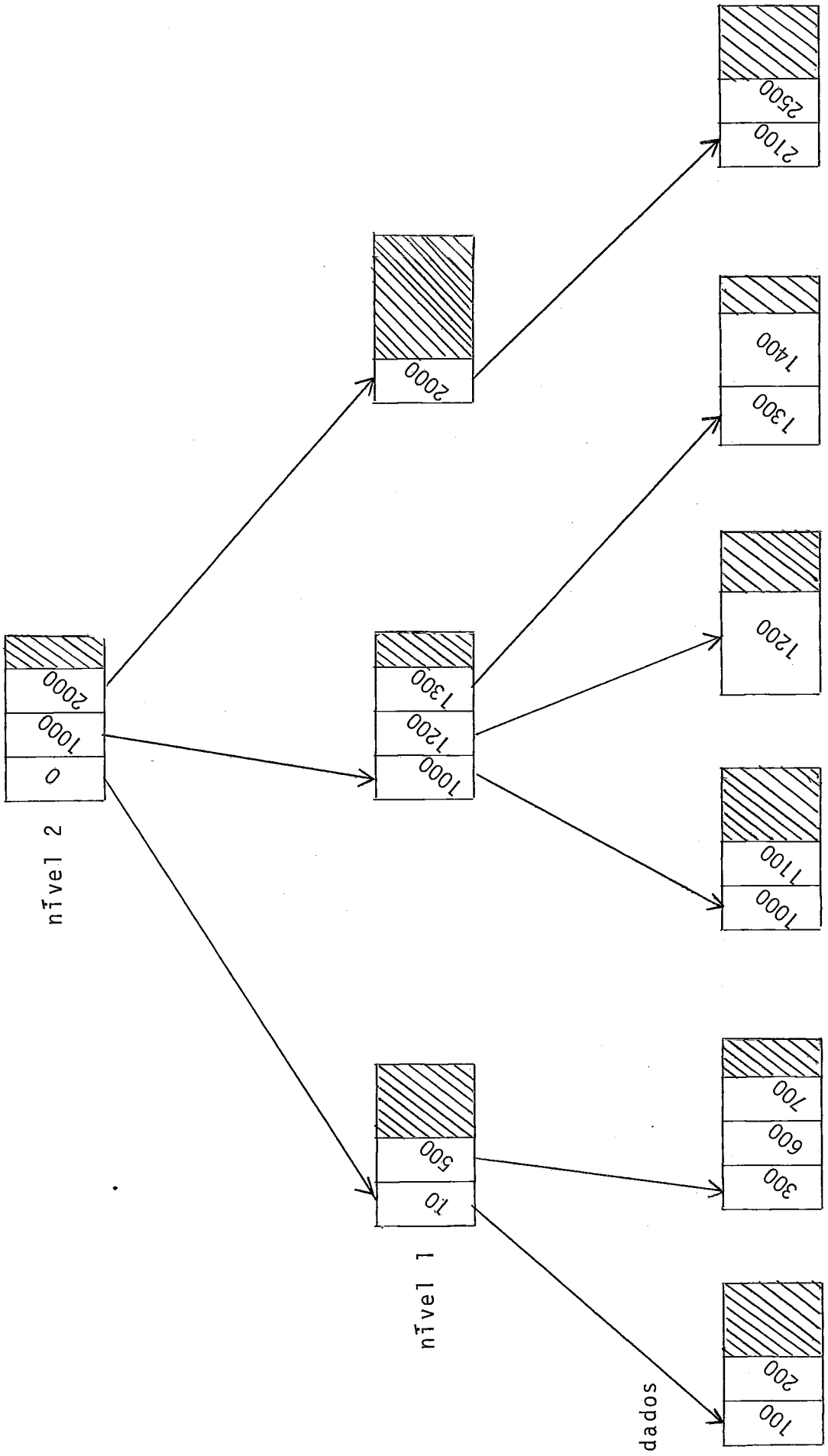



Fig.III-15: Exemplo de organização sequencial indexada

Numa tal estrutura a dois níveis, o nível 1 permite apontar até $31^2 = 961$ blocos de dados. Veremos mais adiante que a compactação utilizada proporciona mais de 7 registros por bloco o que totaliza mais de 6.727 registros.

Com um nível somente chegaríamos a mais de 217 registros (31 x 7).

Esses números mostram a eficiência extremamente alta deste sistema de índices.

## Operações sobre Arquivos Seqüenciais Indexados

Mostraremos, agora, através de exemplos, os diversos processos de manipulação dos arquivos seqüenciais indexados pelo sistema SGF 15 E.

### Remoção de um Registro

Nas figuras seguintes é mostrado um exemplo de remoção de registro:

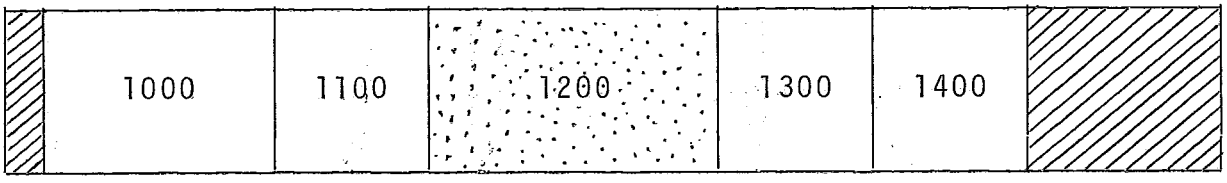



Fig.III-16: Bloco de dados antes da remoção do registro 1200

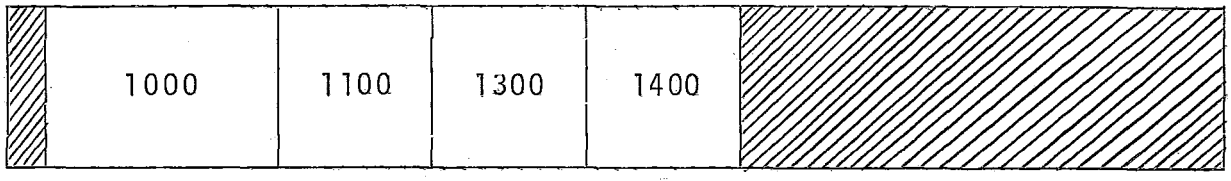



Fig.III-17: Bloco de dados após a remoção do registro 1200

Nesse exemplo o registro "1200" é eliminado do bloco e o espaço vazio assim introduzido, compensado pelo empilhamento dos registros seguintes "1300" e "1400". No caso de remover o único registro presente num bloco, esse fica vazio mas ainda apontado por um elemento de um bloco de índices. Esse bloco de dados será utilizado novamente apenas no caso de inserção de novos registros no mesmo intervalo de chaves que os antigos registros removidos.

Em nenhum caso de remoção de registros é alterada a tabela de índices. Esse procedimento permite evitar a fusão e eliminação dos blocos (que caracterizariam um esquema tipo árvore B) e isto proporciona um gerenciamento muito eficiente em tempo, com pouco gasto adicional de memória.

Inserção de um Registro

Dois casos posem se apresentar:

1º caso: O registro a ser inserido cabe junto com os já presentes no bloco, como mostra a representação seguinte:

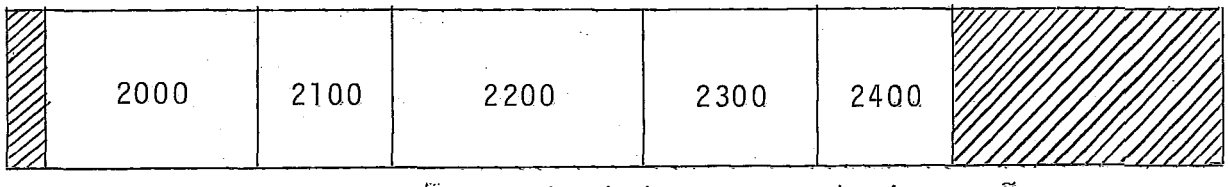



Fig.III-18: Bloco de dados antes da inserção

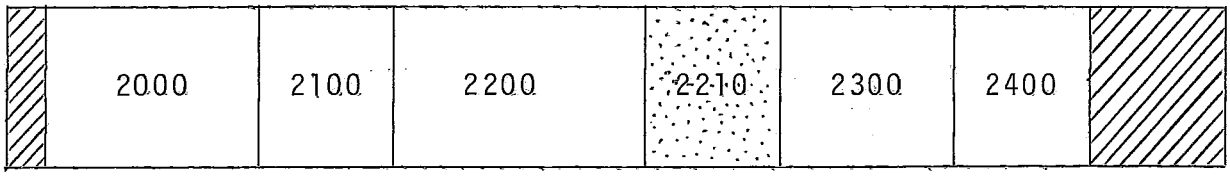



Fig.III-19: Bloco de dados após a inserção

Nesse exemplo o registro "2210" é inserido na sua posição seqüencial de chaves e os registros seguintes "2300" e "2400", deslocados.

O bloco de índices correspondente fica inalterado.

2º caso: O registro a ser inserido não cabe no bloco. É o caso apresentado a seguir:

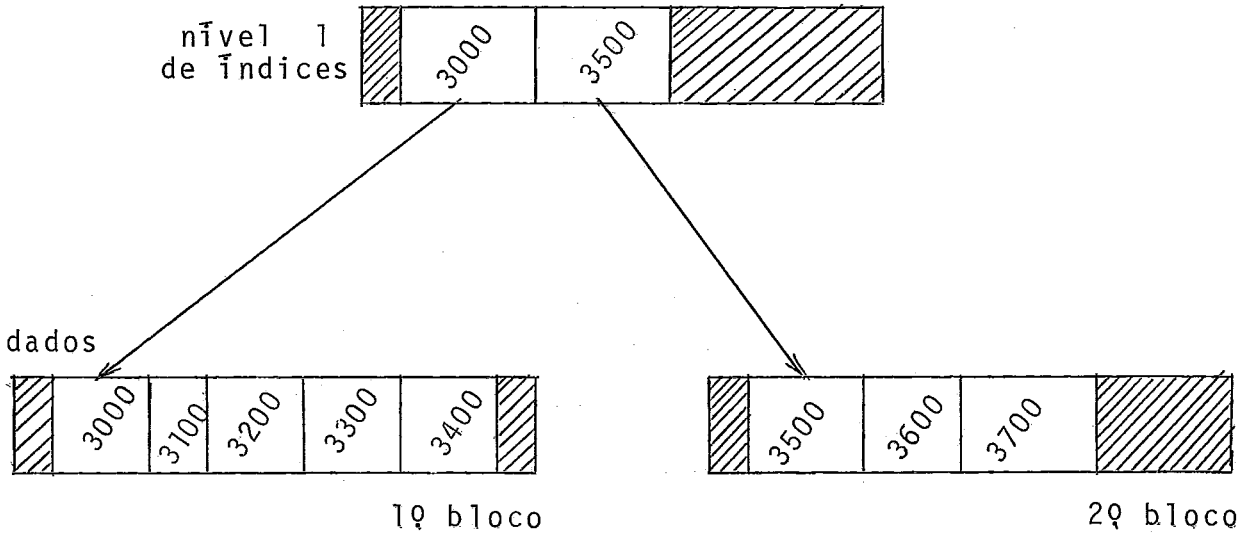



Fig.III-20: Configuração antes da inserção

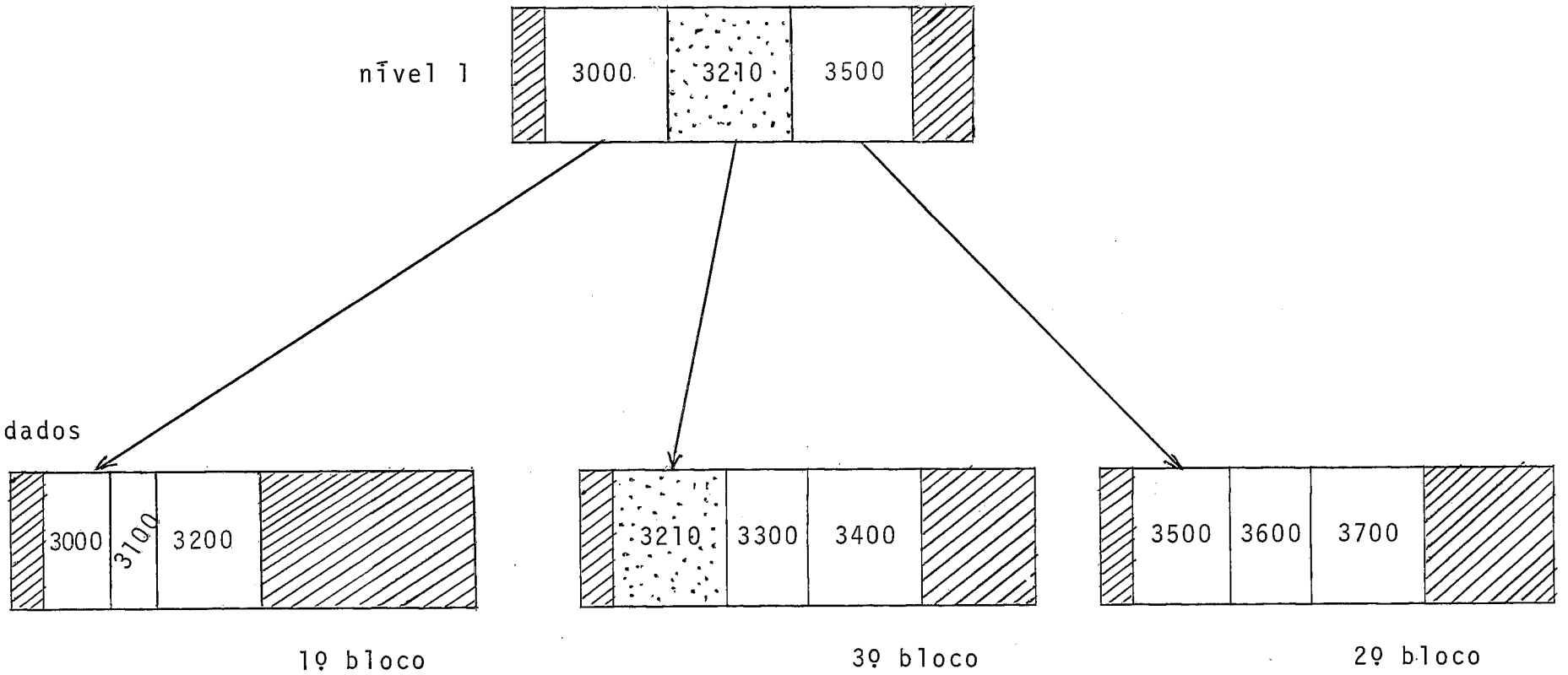



Fig.III-21: Configuração após a inserção

No exemplo acima o registro "3210" a inserir não cabendo no 1º bloco, é criado um novo bloco de dados (3º bloco) que também necessita de um apontador a ser inserido no nível 1 dos índices. Essa última inserção pode também provocar a criação de um novo bloco de índices o que obrigará também a atualização do bloco correspondente de nível 2. Na passagem de 1 nível de índices para 2 níveis há criação de dois novos blocos de índices.

Substituição de Registro

Dependendo do novo tamanho do registro a substituir, caímos num dos dois casos precedentes de inserção e registros.

Observação: Comando "UPDATE"

Lembramos a existência do comando "UPDATE" que permite reestruturar o arquivo de trabalho, eliminando a presença de blocos vazios ou pouco aproveitados (com poucos registros) devido a muitas remoções e/ou inserções e podendo introduzir, em alguns casos, atrasos perceptíveis em listagens na tela do terminal.

Organização da Zona DA

A zona DA é organizada da seguinte maneira:

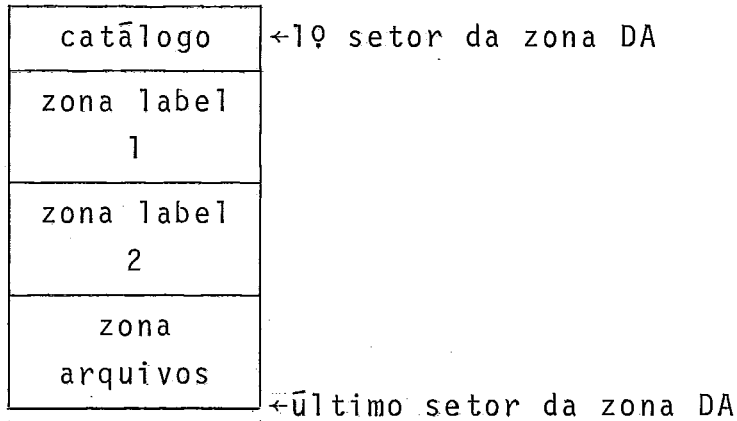



Fig.III-22: Organização da zona DA

O primeiro setor da zona DA é o catálogo dessa zona.

Os principais parâmetros desse catálogo são mostrados na tabela seguinte:

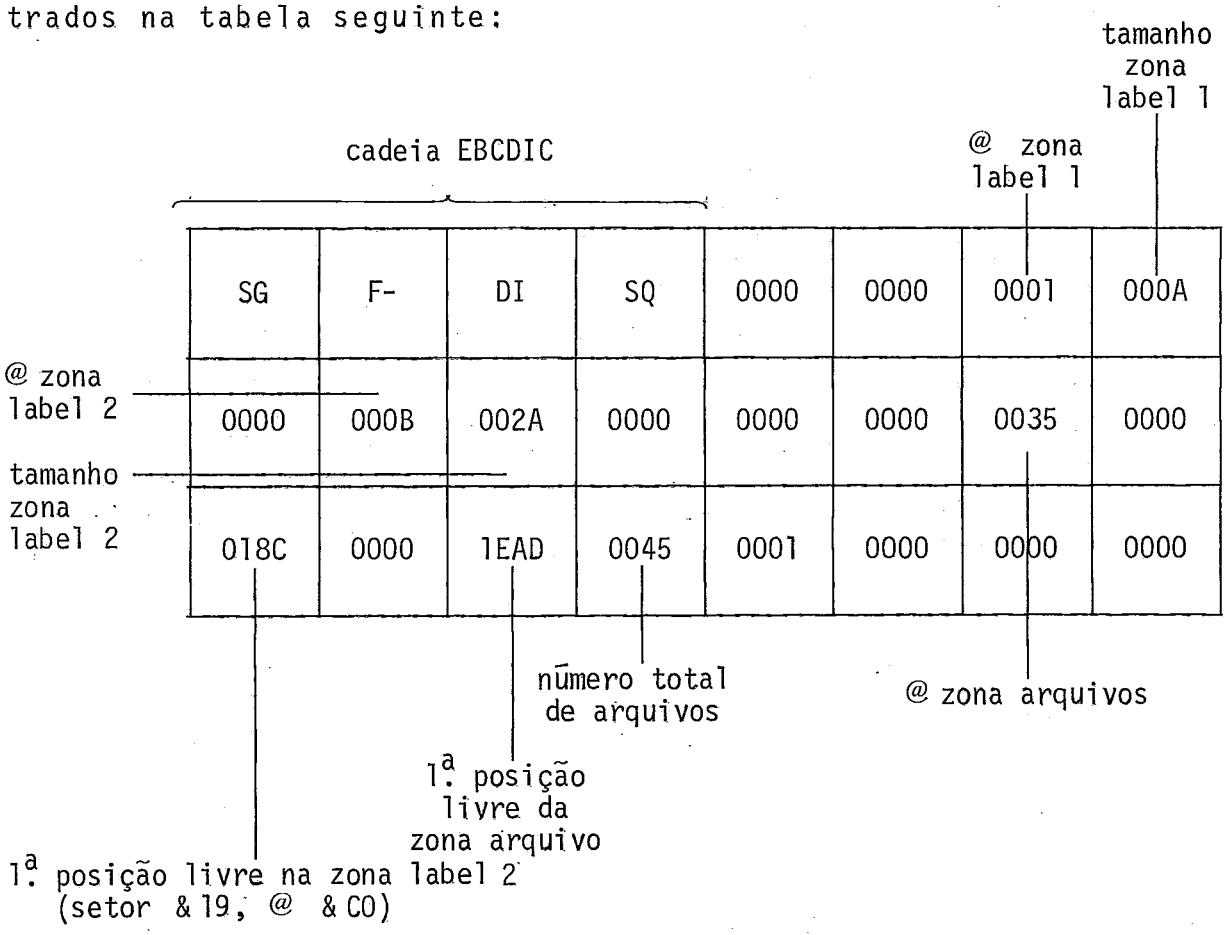



Fig.III-23: Principais parâmetros do setor catálogo da zona DA

A zona label 1 contém o nome dos arquivos com a respectiva chave de proteção (número entre 0 e 255) e apontador para zona correspondente de label 2. Um exemplo é dado a seguir:

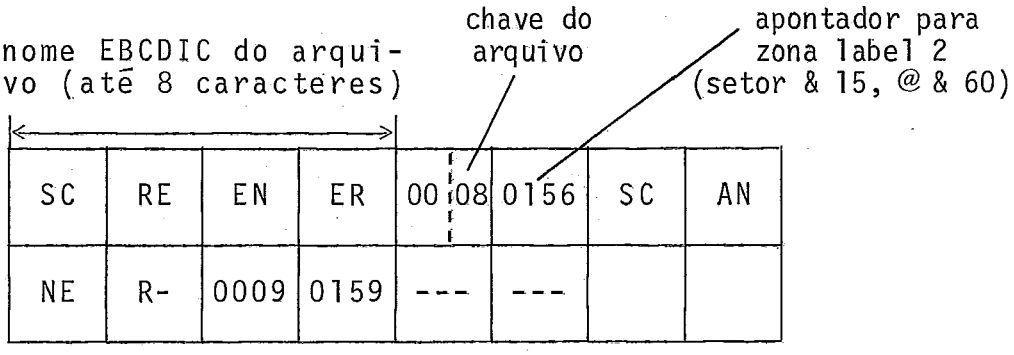



Fig. III-24: Organização da zona label 1

Cada aquivo precisa de 12 bytes nessa zona.

A zona label 2 contém as características dos arquivos.

Um arquivo seqüencial indexado precisa de 48 bytes nessa zona.

As principais características armazenadas nessa zona para um arquivo seqüencial indexado são mostradas a seguir:

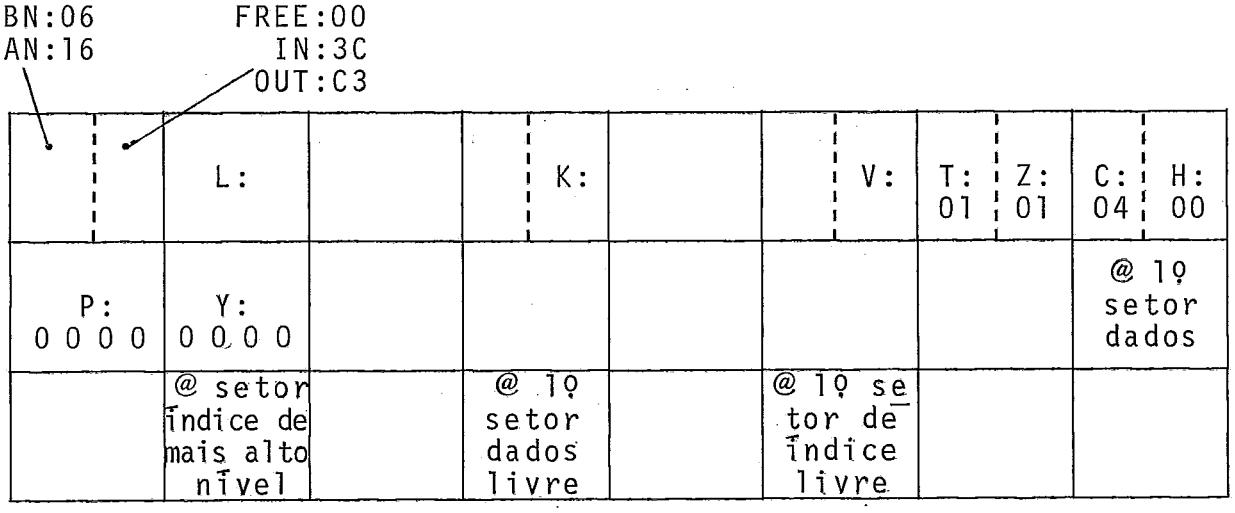



Fig. III-25: Organização da zona label 2

Com:

L: tamanho do arquivo em número de setores

K: número de conta usuário de 0 a 255

V: número de versão de 0 a 255

T: tamanho de um bloco de índices em setores (01 no caso do editor)

Z: tamanho de um bloco de dados em setores (01 no caso do editor)

C: tamanho da chave em bytes (04 no caso do editor)

H: posição da chave nos registros em byte (00 no caso do editor)

P: padding dados em bytes (0000 no caso do editor)

Y: padding índices em bytes (0000 no caso do editor).

O primeiro byte indica se o arquivo é alfanumérico ou binário (binário no caso do editor).

O segundo byte é o modo de acesso e fixa se o arquivo é protegido e acessível apenas em leitura (IN) ou protegido e acessível apenas em escrita ou atualização (OUT) ou enfim se não é protegido (FREE).

Ligação Arquivos - Sistema Operacional

A fim de evitar acessos constantes à zona label 2,

as características dos arquivos utilizados num dado instante são mantidas residentes na memória, na zona LPTES do Sistema Operacional. Apenas um acesso à zona label 2 na associação de um arquivo a uma OL, outro ao fechamento desse arquivo são necessários num processo de entrada/saída.

Cada arquivo ativo dispõe de um grupo de 14 bytes na tabela OLFI. A tabela OLFI é constituída de tantos grupos quantas são as etiquetas operacionais disponíveis no sistema para acesso a arquivos (no caso de EDIT.74: 9 das 16 padrões [OL]+3 usuário [UL]). Para uma OL ou UL um grupo da tabela OLFI armazena as informações mostradas no seguinte exemplo:

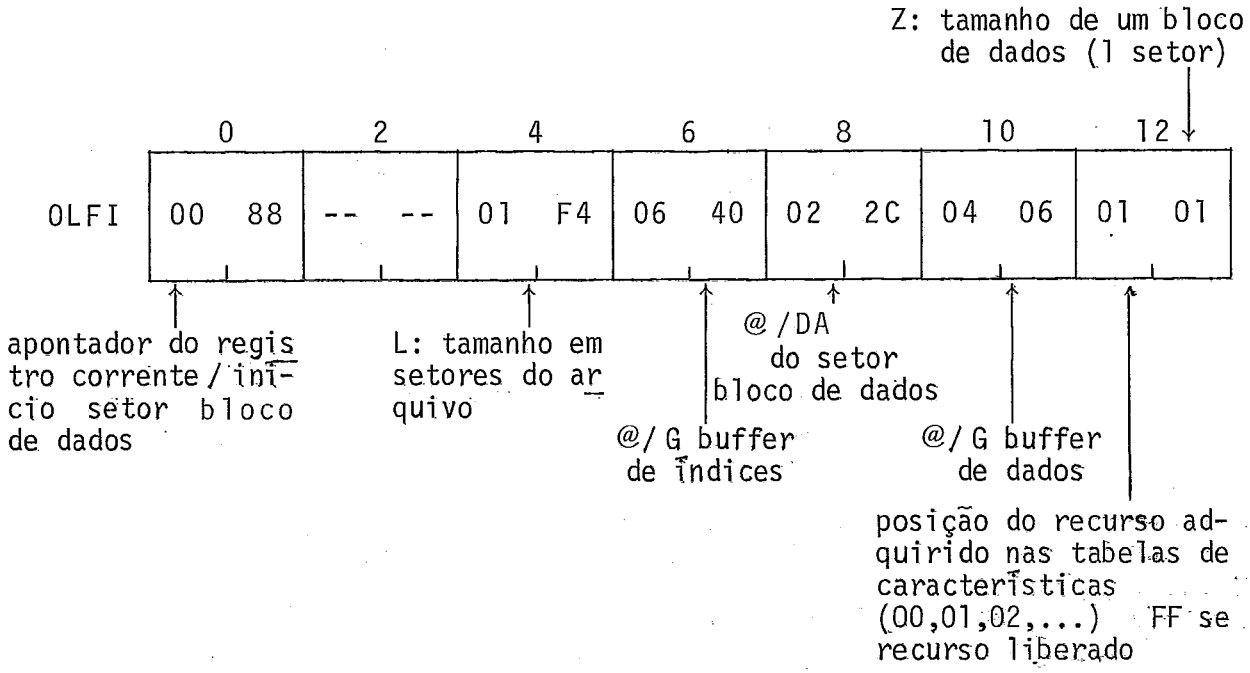



Fig. III-26: Elemento da tabela OLFI do S.O.

As tabelas de características são constituídas de tantos elementos quantos os recursos de arquivo permitidos pelo sistema (5 no caso de EDIT.74). Os elementos podem ser de 1 byte ou de 2.

A seguir daremos, num exemplo, as principais tabelas utilizadas por SGF 15 E. Os nomes das tabelas correspondem aos nomes utilizados na geração do S.O. (CII[13], CII[14]).

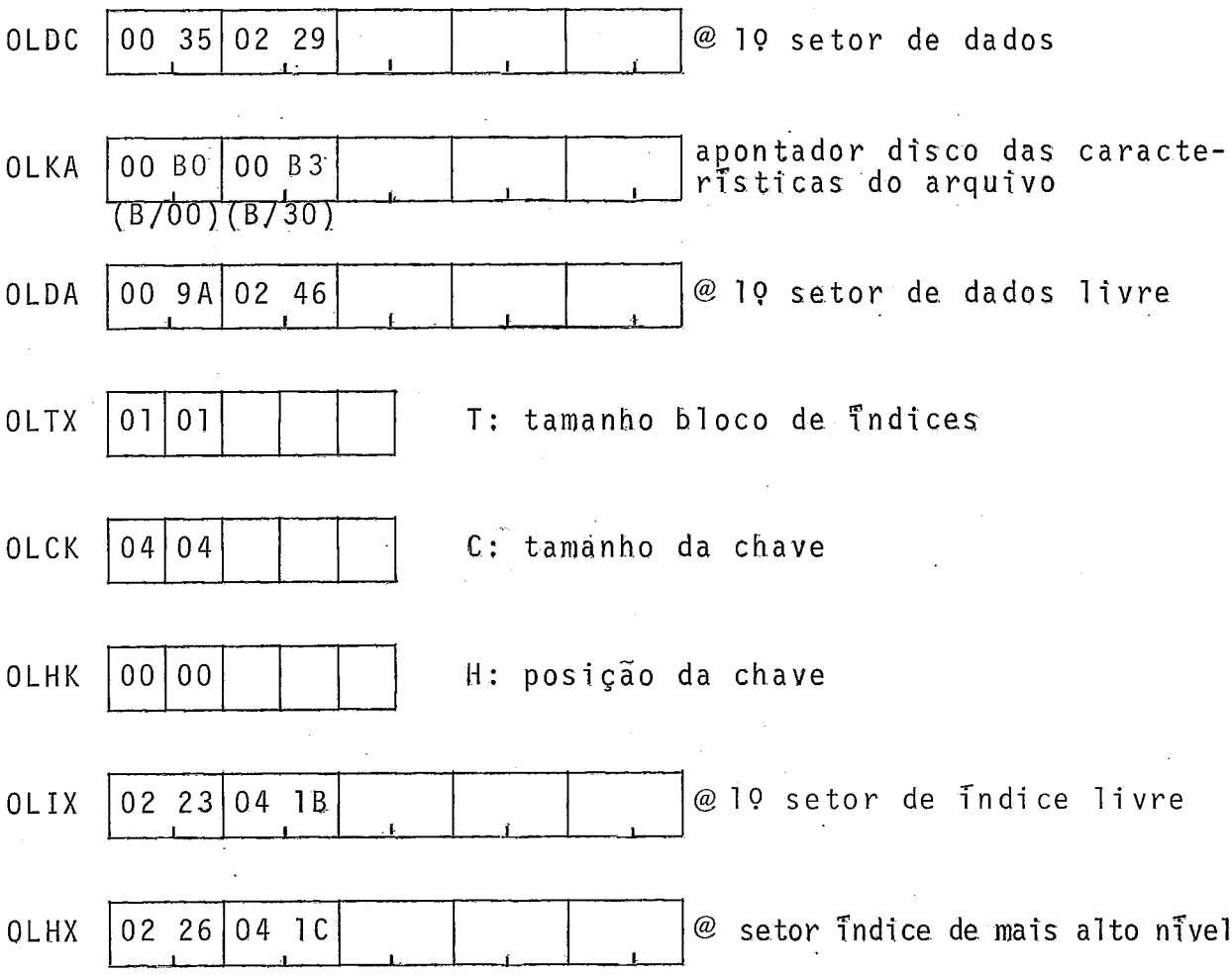



Fig.III-27: Tabelas de características de arquivos do S.O.

Todas as tabelas são apontadas a partir de endereços fixos específicos no início do S.O.

Comandos de Gerenciamento de Arquivos Seqüenciais Indexados (S.I.)

Para efetuar entradas/saídas num arquivo S. I, a

sucessão de comandos é a seguinte:

- declaração do arquivo (se necessário);
- associação do arquivo a uma etiqueta operacional (OL);
- abertura do arquivo e da OL associada;
- operações de entradas/saídas;
- fechamento do arquivo e da OL associada.

Declaração de um Arquivo S.I.

A criação de um arquivo S.I. é assegurada pela chamada do módulo "M:FILE" do S.O.. Na chamada, o usuário deve fornecer uma tabela de 28 bytes (C.B.: control bloco) que precisará o nome, o número de conta, o modo de acesso e outras características do arquivo a ser criado.

O catálogo e as zonas label 1 e 2 da zona DA são atualizados. Um espaço disco é alocado ao arquivo após o último arquivo da zona DA. O nome do arquivo com seu número de conta é implantado na zona label 1 após o último nome de arquivo. O modo de acesso pode ser IN (leitura), OUT (escrita e atualização) ou FREE (deixado em aberto para ser decidido no momento da associação). Os arquivos usuários criados pelo editor são do tipo "FREE".

Associação de um Arquivo S.I. a uma OL

É assegurada pela chamada do módulo "M:ASGN" do S.O.

Tal associação permite a ligação entre um arquivo e uma OL: o número da OL passa a ser o nome interno do arquivo para o S.O.. Na chamada do módulo, o usuário deve fornecer um control bloco de 18 bytes que precisará principalmente o nome da OL, o nome, número de conta e modo de acesso do arquivo.

No caso do editor, o modo de acesso é do tipo RW (read/write), único a permitir atualizações nos arquivos S. I. (substituições, inserções, remoções).

Esse tipo de modo de acesso não é permitido com etiquetas operacionais padrões, o que nos obrigou a utilizar etiquetas operacionais usuário (U:1, U:2, U:3, ...). O método de acesso escolhido é do tipo "misto", ou seja, com primeiro acesso aleatório seguido de acessos seqüenciais.

A função do módulo "M:ASGN" é procurar seqüencialmente na zona de label 1, graças a um buffer de 256 bytes próprio do S.O., o nome do arquivo a associar. Em caso de sucesso, as tabelas de ligações arquivos - S.O. são atualizadas com as características do arquivo (zona label 2) ao qual é atribuída a OL requerida e um recurso arquivo.

## Abertura de um Arquivo S.I.

É assegurada pela chamada do módulo "M:OPN" do S.O. na qual o usuário deve fornecer um C.B. de 12 bytes que precisa principalmente o tamanho e o endereço relativo à base G (base geral do programa) dos buffers de registros e de índices indispensáveis às operações de entrada/saída.

No caso dos arquivos do editor, os buffers utilizados são descritos a seguir:

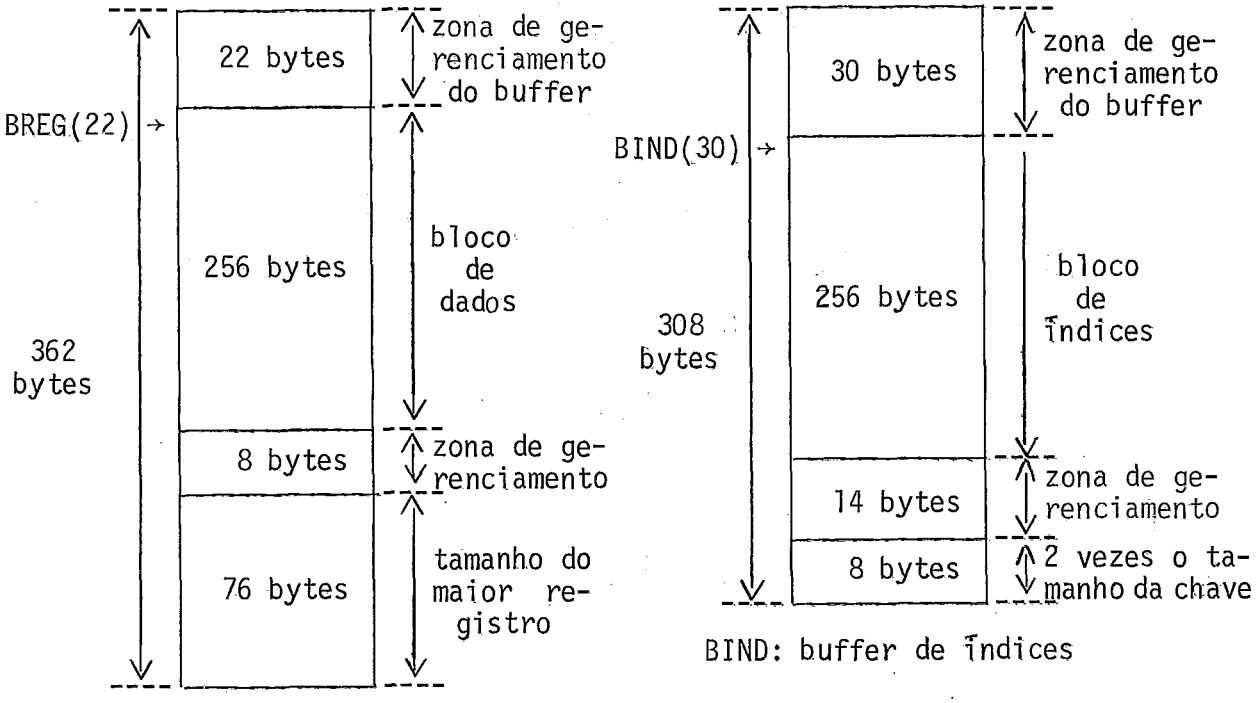



Fig. III.28: Buffers de trabalho com arquivos S.I.

A zona de gerenciamento de 22 bytes do buffer de registro fundamental para todas as operações de entrada / saída é detalhada a seguir:

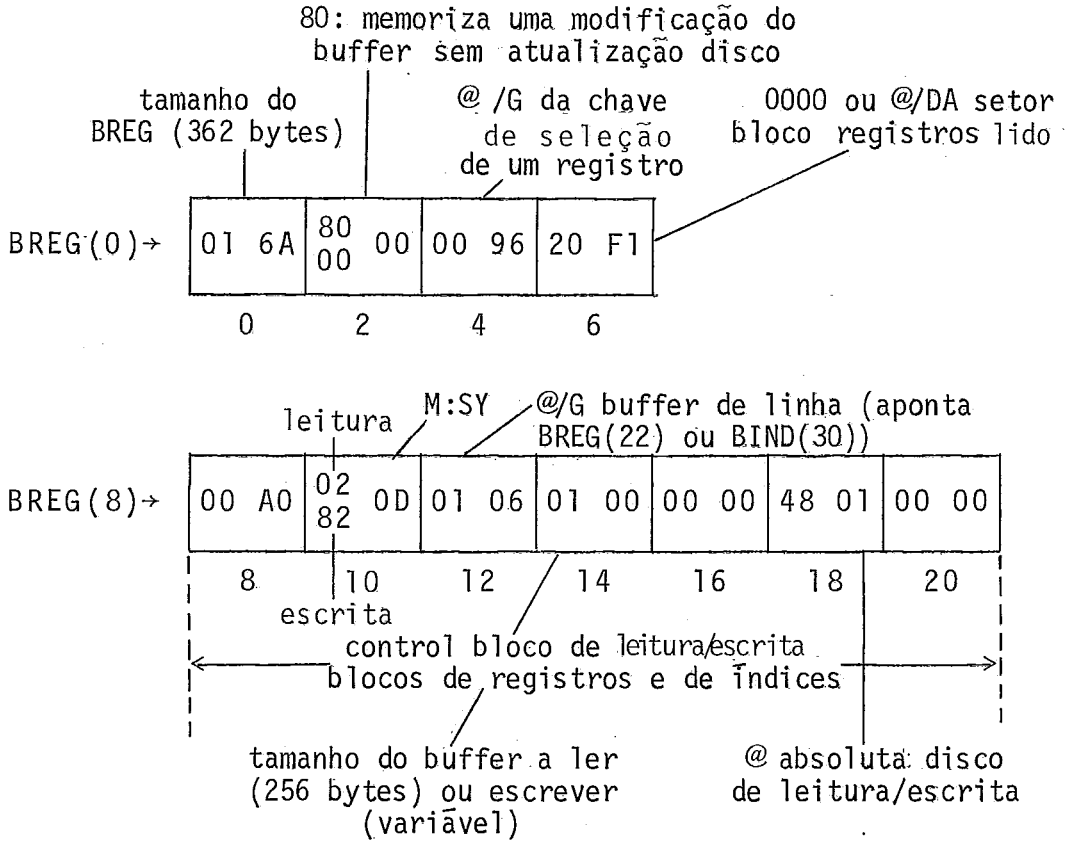



Fig.III-29: Detalhes da zona de gerenciamento do buffer de registros

É constituída principalmente de um control bloco usado para todas as operações de entrada/saída disco, tanto para blocos de registros quanto para blocos de índices.

Operações de Entradas/Saídas com Arquivo S.I.

São asseguradas pela chamada do módulo "M:IOX" do S.O. na qual o usuário deve fornecer um C.B. de 8 bytes onde precisará em particular o tipo de função requerida.

São 7 tipos de funções:

. SLT (select): seleção de um registro através da sua chave

. $GET_0$ : leitura de um registro com progressão do apontador de registro

. $GET_1$ : leitura de um registro sem progressão do apontador de registro

. PUT : criação de registros

. RPL(Replace): substituição de um registro por outro de mesma chave

. DEL(Delete) : remoção de um registro

. INS(Insert) : inserção de um registro

Todas as funções descritas a seguir correspondem ao caso particular do editor com modo de acesso próprio.

SLT: Seleção de um Registro

A seleção de um registro de determinada chave é feita por consulta sistemática do bloco de índices de mais alto nível (apontado pela tabela OLHX e normalmente de nível 2) até o bloco de índices de nível 1 que determina o endereço disco do bloco de dados contendo o registro procurado. Uma verificação preliminar é feita com BREG(6) para determinar se o bloco de dados apontado já está presente no buffer BREG a fim de evitar, nesse caso, um acesso disco inútil.

Após atualização ou não de BREG, o registro é procurado no bloco e os apontadores OLFI(0) e OLFI(8) atualizados.

No caso de o registro procurado não existir,

OLFI(0) apontará o registro de chave diretamente superior ao do registro procurado.

## GET: Leitura de um Registro

A função GET transfere o registro de dados apontado graças a OLFI[OLFI(0) e OLFI(8)] para um buffer fornecido pelo usuário. $GET_0$ atualiza OLFI para apontar o próximo registro enquanto $GET_1$ é sem ação sobre OLFI. No caso do próximo registro pertencer a outro bloco de dados, o endereço setor de encadeamento de bloco é usado.

Veremos mais adiante que reprogramamos a função GET para evitar principalmente que o algoritmo de descompactação introduzido prejudique essa fase de leitura. Criamos também um novo tipo de GET que permite ler uma sucessão de registros de chaves decrescentes.

## PUT: Criação de Registro

A função PUT cria um arquivo S.I. escrevendo registros seqüencialmente. A tabela de índices é criada a cada passo da constituição dos blocos de dados. Os registros devem ser apresentados por ordem crescente das suas chaves.

O modo de acesso é do tipo "OUT" e o método, seqüencial.

O primeiro PUT reinicializa o arquivo.

Essa função é usada no editor para criar o primeiro registro de um arquivo (INS não é aceito num arquivo vazio) e para transferir um arquivo para outro (comandos GET, SAVE, UPDATE, ...).

RPL: Substituição de Registro

A função RPL permite substituir um registro apontado por OLFI[OLFI(0) e OLFI(8)] por outro de mesma chave. Será então necessária a seleção do registro a substituir antes de se efetuar a sua substituição. Se o novo registro for de tamanho menor ou igual ao antigo, não será feita a atualização no nível do arquivo e um flag de memorização de modificação do buffer será posicionado em BREG(2) que passará a ser negativo (8000). No caso contrário, será feita a atualização.

DEL: Remoção de Registro

A função DEL permite remover um registro apontado por OLFI[OLFI(0) e OLFI(8)], ou seja, um registro precedentemente selecionado. A atualização do arquivo nunca é feita no momento e somente o flag de memorização de modificação do buffer será posicionado em BREG(2).

INS: Inserção de Registro

A função INS permite inserir um registro de determinada chave num arquivo não vazio (com no mínimo um registro). O registro será inserido entre os registros de chaves imediata-

mente inferior e superior à chave do registro a inserir. A inserção atualiza imediatamente o arquivo.

Fechamento de um Arquivo e da OL Associada

É assegurado pela chamada do módulo "M:CLOS" do S.O.. Permite:

- fechar a OL associada autorizando uma nova abertura ou associação;

- atualizar eventualmente um bloco de dados do arquivo no caso de o flag de BREG(2) estar posicionado;

- atualizar as características do arquivo na zona label 2 graças às tabelas de ligação arquivo - S.O.;

- liberar o recurso arquivo alocado a OL.

## III.4 - Detalhes sobre Algumas Particularidades do Editor

### Gerenciamento da Tela dos Terminais

Para economizar a memória escassa do Mitra 15, já que os buffers requisitados pela organização S.I. são um pouco grandes (670 bytes), o editor trabalhará com um único buffer de uma linha para cada terminal vídeo (92 bytes).

A tela é gerenciada por uma tabela de 4 bytes por linha que determina para cada linha física da tela o status (1º byte) e a chave (três últimos bytes) de um eventual registro apresentado na linha referente.

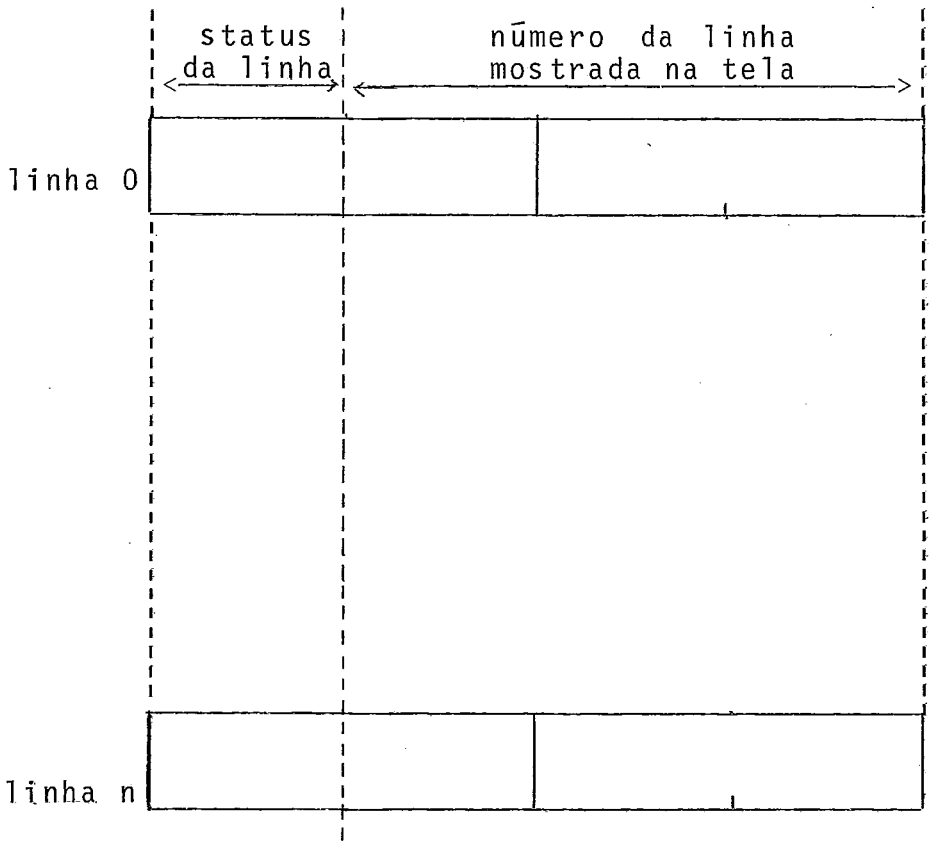



Fig.III-30: Tabela de gerenciamento da tela

Os diferentes valores hexadecimais do status com seus significados são os seguintes:

00: linha presente na tela

01: linha removida da tela e do arquivo (pelo comando CNTRL/D[DELETE])

02: linha nova e vazia em modo SEQ (não será conservada)

FF: linha corrente e seguintes não utilizadas.

Na movimentação nos quatro sentidos do cursor na tela pelo usuário, os ponteiros de acompanhamento do cursor são atualizados mas não o buffer de trabalho. Um indicador aponta a linha da tela presente no buffer de trabalho.

Somente no caso de modificação de uma linha outra que a presente no buffer de trabalho, este último é inicialmente atualizado a partir da seleção (SLT) e leitura (GET) disco do registro correspondente cujo número de chave é determinado pela consulta da tabela de gerenciamento da tela.

Buffers Encadeados dos Teclados; Comunicação Full Duplex

As interfaces eletrônicas de linhas assíncronas do Mitra 15 que atendem os diversos terminais de vídeo do editor não possuem buffer de recepção de caracteres próprio.

Sendo microprogramadas, elas utilizam como buffer de caracteres o próprio buffer de memória central definido pelo usuário. Os caracteres lidos da linha entram diretamente neste buffer (acesso modo canal).

Para efetuar uma recepção de caracteres numa linha, o usuário deverá precisar, além da posição do buffer de recepção, o número máximo de caracteres que deseja receber, um caractere de reconhecimento (KR) predeterminado (parâmetro do S.O.) podendo cancelar antecipadamente a operação no momento em que for recebido.

Todo caractere chegando numa linha não ativa é simplesmente perdido.

No caso do presente editor, devido a seu caráter "full screen", os caracteres teclados podendo modificar diretamente o texto apresentado na tela, é aconselhável evitar perdê-los para não confundir o usuário apesar da existência da função de eco (um caractere perdido não sendo ecoado).

Por outro lado, certos terminais têm teclas que emitem dois caracteres consecutivos (caractere ESCAPE seguido de outro caractere).

Isto nos obrigou a utilizar buffers encadeados de um caractere.

Neste tipo de organização, cada buffer de um caractere aponta seu sucessor. Para assegurar uma leitura perma-

nente das linhas, encadeamos o último buffer da lista com o primeiro.

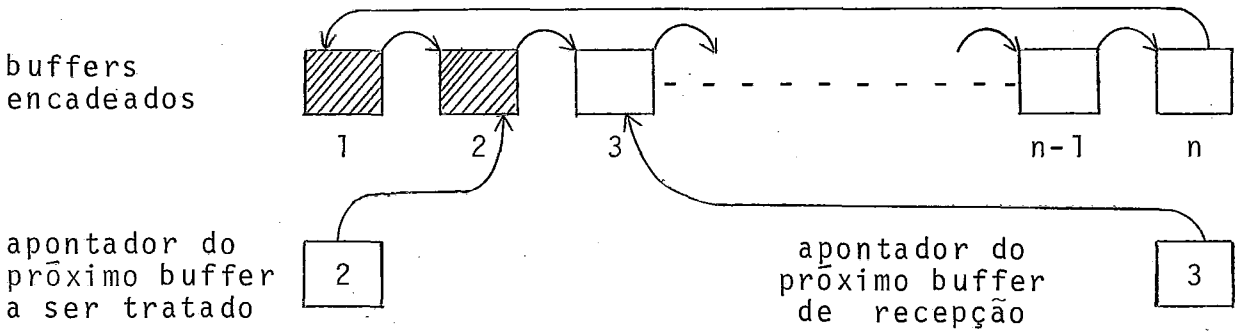



Fig.III-31a: Organização dos buffers encadeados dos teclados

O microprograma de recepção "fica em loop" jogando os caracteres lidos da linha nos sucessivos buffers encadeados, atualizando a cada passo um apontador que determina qual dos buffers receberá o próximo caractere. Este apontador tem uma grande importância para nós pois nos permite saber se foram recebidos caracteres da linha. O editor mantendo um apontador "do próximo buffer a ser tratado", a não igualdade desse apontador com o precedente implicará na chegada de novos caracteres da linha.

O loop introduzido no nível da micromáquina não é problemático pois, na realidade, o microprograma de tratamento da linha é ativado, somente na chegada de caracteres, por uma suspensão similar, no nível da micromáquina, a uma interrupção no nível da máquina.

Com a ativação permanente das linhas de recepção, a comunicação "full duplex" impõe-se para permitir emissão simul

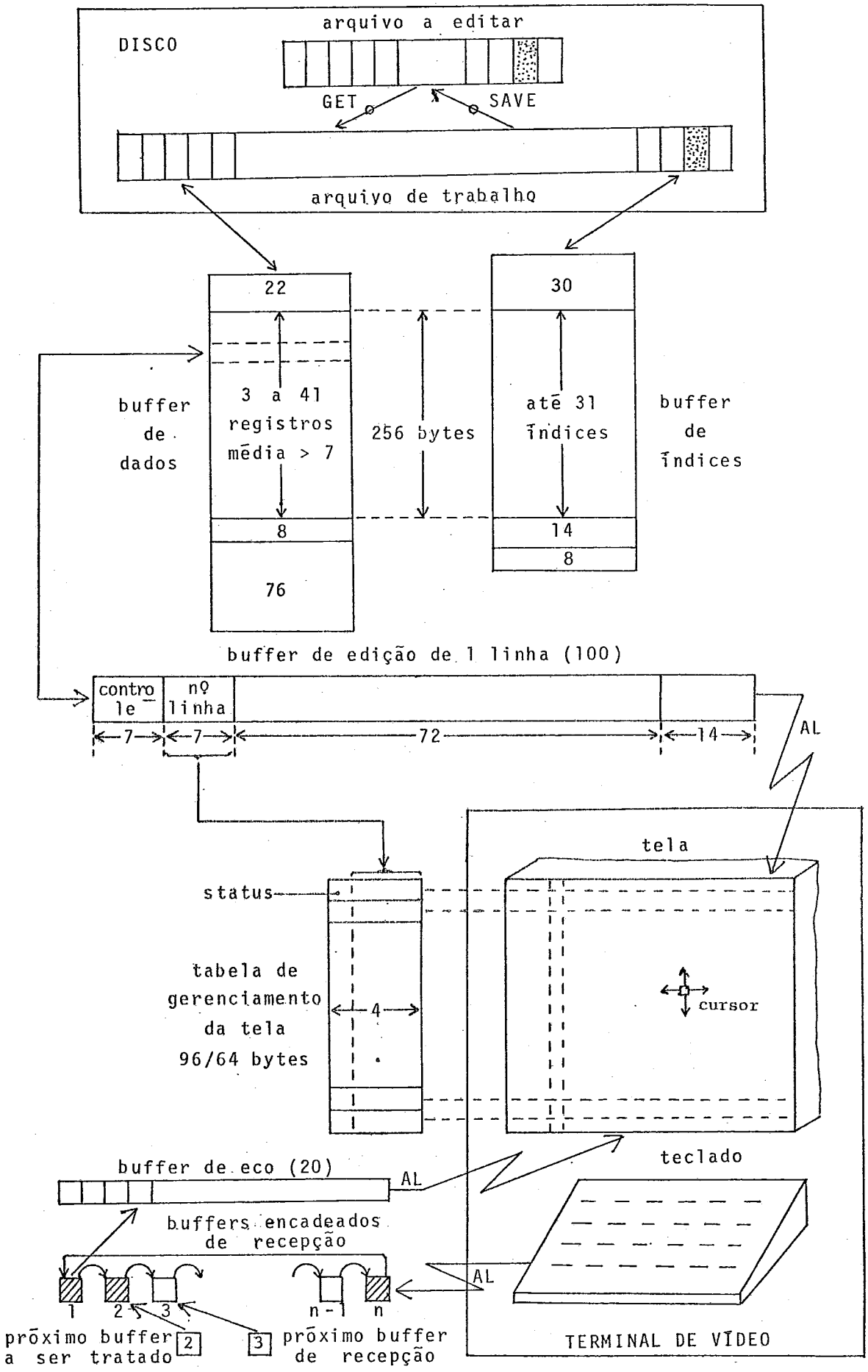



Fig. III-31b: Esquema Global do Editor

tânea e evitar o zeramento e relançamento freqüente do loop de recepção respectivamente antes e depois de uma emissão, como seria o caso numa transmissão "half duplex". Já vimos também a grande vantagem do "full duplex" com possibilidade de pré-tratamento dos caracteres recebidos antes de ecoá-los na tela dos terminais, o que permite, entre outros benefícios, o perfeito controle do cursor manipulado pelo usuário.

A figura III.31b apresenta um esquema global do editor resumindo a inter-relação entre os diversos elementos por este manipulados (disco, memória, terminais).

Reentrância

Para minimizar o uso da memória central, evitando a presença simultânea de várias cópias idênticas do mesmo programa EDITOR, uma para cada terminal ativo, cada uma ocupando um nível de interrupção, implantamos um sistema de reentrância que permite com um único programa EDITOR ocupando um único nível de interrupção atender com simultaneidade aparente todos os terminais.

O Mitra 15 não possui instruções de manipulação de pilha por ser orientado para trabalhar com bases (base geral G e base local L) (CII[8]).

A idéia para implementar uma reentrância eficiente é utilizar instruções da máquina que, além de providenciar um desvio, efetuam no nível da micromáquina várias operações elementares de mudança de base, salvamento de contexto, troca de contexto,... São as instruções denominadas no Mitra 15 "instruções de desvio sistema" (CLS, RTS, CSV, RSV, DIT, DITR).

Como de maneira geral o fim de uma entrada / saída para qualquer periférico pode gerar uma interrupção a desejo do usuário, pensamos inicialmente numa reentrância total baseada num programa supervisor ativado a cada chegada de interrupção de fim de entrada/saída. Este programa supervisor rodaria num nível de interrupção mais prioritário do que o do EDITOR, o que permitir-lhe-ia suspender o programa EDITOR em qualquer ponto e reativá-lo em outro ponto a serviço de outro terminal.

Não foi possível implantar este tipo de reentrância, pois os buffers encadeados de leitura dos teclados não permitem gerar interrupções no fim de preenchimento de cada buffer mas somente no preenchimento do último buffer da lista que, no nosso caso, também não existe devido à forma circular dos buffers encadeados utilizados.

Pensamos então numa reentrância muito parecida com a multiprogramação do sistema operacional do Mitra no caso da chamada do módulo "M:WAIT".

Após efetuar uma operação de entrada/saída, o usuário pode na verdade liberar a CPU pelo uso deste módulo, o que permite ao S.O. ativar um programa à espera de CPU conectado a um nível de prioridade inferior.

A seqüência de instruções de linguagem de montagem utilizada é a seguinte:

```
LEA  CB          * A:=@/G do control bloco caracte-
                 * rizando a entrada/saída a efetuar.
```

```
CSV  M:IO      * chamada do módulo de S.O. lançan
               * do o microprograma de E/S.
CSV  M:WAIT    * devolução da CPU aos níveis de in
               * terrupção menos prioritários
```

No EDITOR substituímos a chamada do módulo M:WAIT do S.O. pela chamada de um subprograma do editor que denominamos supervisor (SUPER).

A seqüência passa a ser a seguinte:

```
LEA  CB        * A:=@/G do control bloco
CSV  M:IO      * chamada do S.O.
LDE  =0        * E:=modo de trabalho do supervisor
               * (0 ou 1)
CLS  SUPER     * chamada do supervisor do EDITOR
```

A chamada do supervisor num ponto do EDITOR, onde normalmente precisa esperar um fim de entrada/saída para continuar em seqüência, permite reativar o EDITOR a serviço de outro terminal de vídeo o qual, nas mesmas condições anteriores, chamou o supervisor e está no momento com entrada/saída concluída e à espera de CPU.

Contrariamente ao caso precedente, este tipo de reentrância ocorre em pontos fixos do programa, determinados no nível da programação pela chamada do subprograma SUPER.

O algoritmo do supervisor é mostrado na fig.III-32.

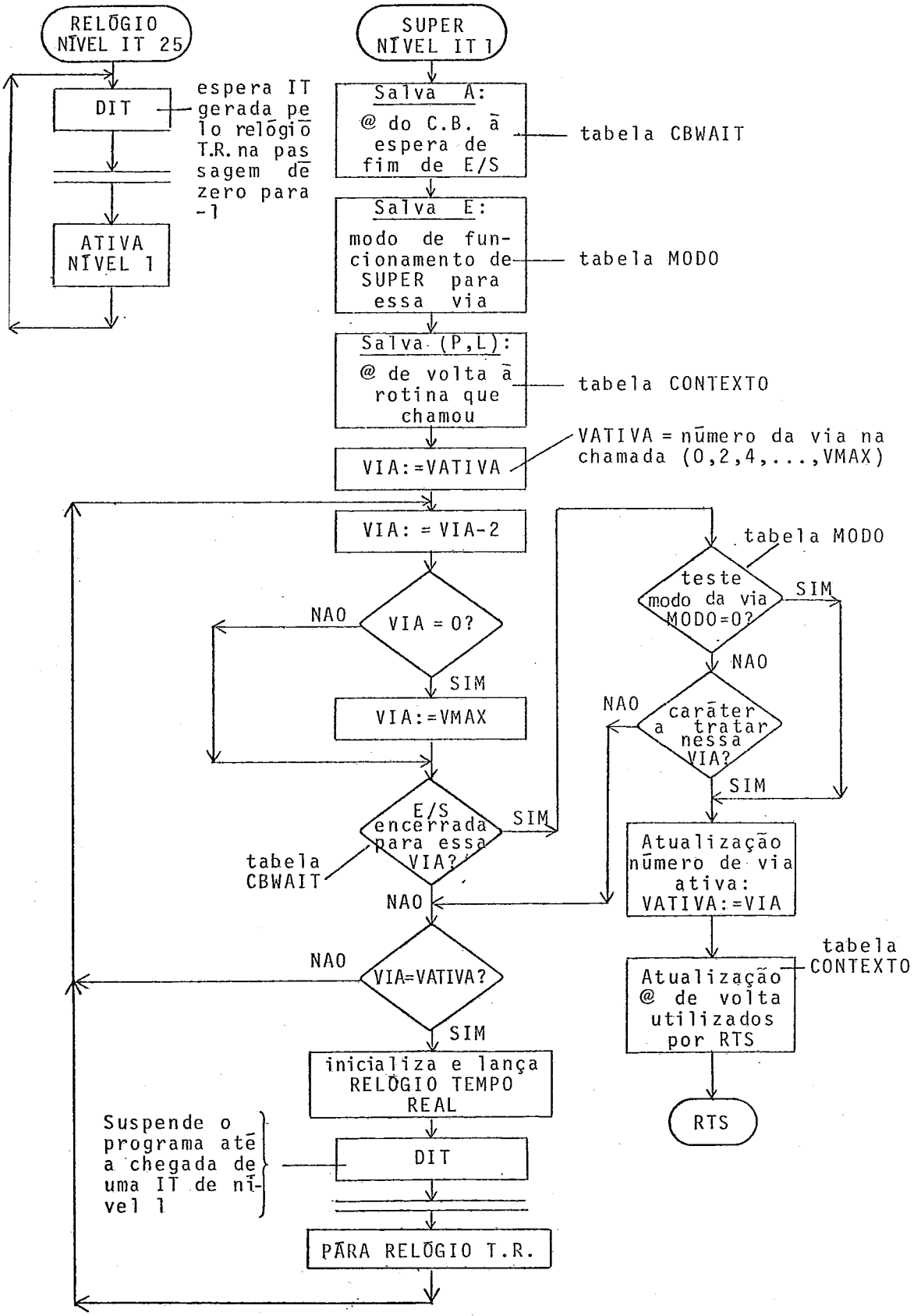



Fig. III-32: Algoritmo do supervisor e do relógio

O teste pelo supervisor do fim de E/S de uma via é feito sobre o 1º byte do C.B. de E/S denominado byte evento (bit mais significativo a 1 no caso de E/S em andamento, senão a 0).

No modo E = 0 (sem espera) o supervisor faz somente este tipo de teste para determinar se uma via deverá ser reativada ou não.

No modo E = 1 (com espera de caractere do teclado) o supervisor, após ter feito este teste, autorizará uma via somente se ela tiver algum caractere recebido do teclado a tratar.

A instrução "CLS" guarda automaticamente nas duas primeiras posições da base local do supervisor os endereços de volta (P,L) de quem chamou. Estes endereços são guardados numa tabela de um par de endereços por via de terminal de vídeo (tabela CONTEXTO). Uma variável em base geral (G) indica a todo instante qual é a via ativa (uma de cada vez) e permite o acesso a esta tabela. Após ter determinado por consulta dos CB qual é o número da via a ativar, o supervisor reatualiza as duas primeiras palavras (endereços de volta) da sua base local a partir da precedente tabela e relança o processo de edição com a via escolhida pela simples instrução "RTS" que desvia para estes endereços.

A varredura das vias pelo supervisor faz-se numa ordem predeterminada mas começando sempre pela via consecutiva à via ativa na chamada do supervisor. Isto permite chamar o supervisor com um CB fictício indicando uma E/S já acabada, somente para dar a vez a outro terminal.

Com o objetivo de rodar a versão definitiva do editor no nível de prioridade 1, a fim de liberar o nível zero, único a suportar os compiladores, com a idéia de editar e compilar simultaneamente, o supervisor manipula (inicializa, lança, pára) um relógio de tempo real que, graças a um programa de nível 25 ativando o nível do editor, permite alocar ao nível zero um tempo de CPU quando o editor está esperando um fim de E/S ou uma entrada de caracteres em todas as suas vias.

A reentrância não se limita somente ao supervisor. Cada subprograma desejando ser reentrante precisa de uma interface específica de apenas 5 instruções de máquina executadas unicamente na sua chamada. A reentrância é feita sobre a base local (L) e cada subprograma reentrante deverá possuir tantos dados locais repetidos quantos terminais atendidos. A interface permite acertar a base local própria a cada terminal no lançamento do subprograma (graças à instrução ICL de incrementação da base L) e salvar os endereços (P,L) de volta ao subprograma que chamou. O retorno a este último é feito simplesmente pela instrução RTS que utiliza para este fim as duas primeiras palavras da base L atualizadas pela interface. A fig.III-33 detalha todo o processo de reentrância no nível de um subprograma.

Salientamos que somente precisam ser reentrantes os subprogramas do editor que efetuam entradas/saídas, podendo os outros se contentar em trabalhar a partir dos registradores para receber e devolver variáveis ou apontadores.

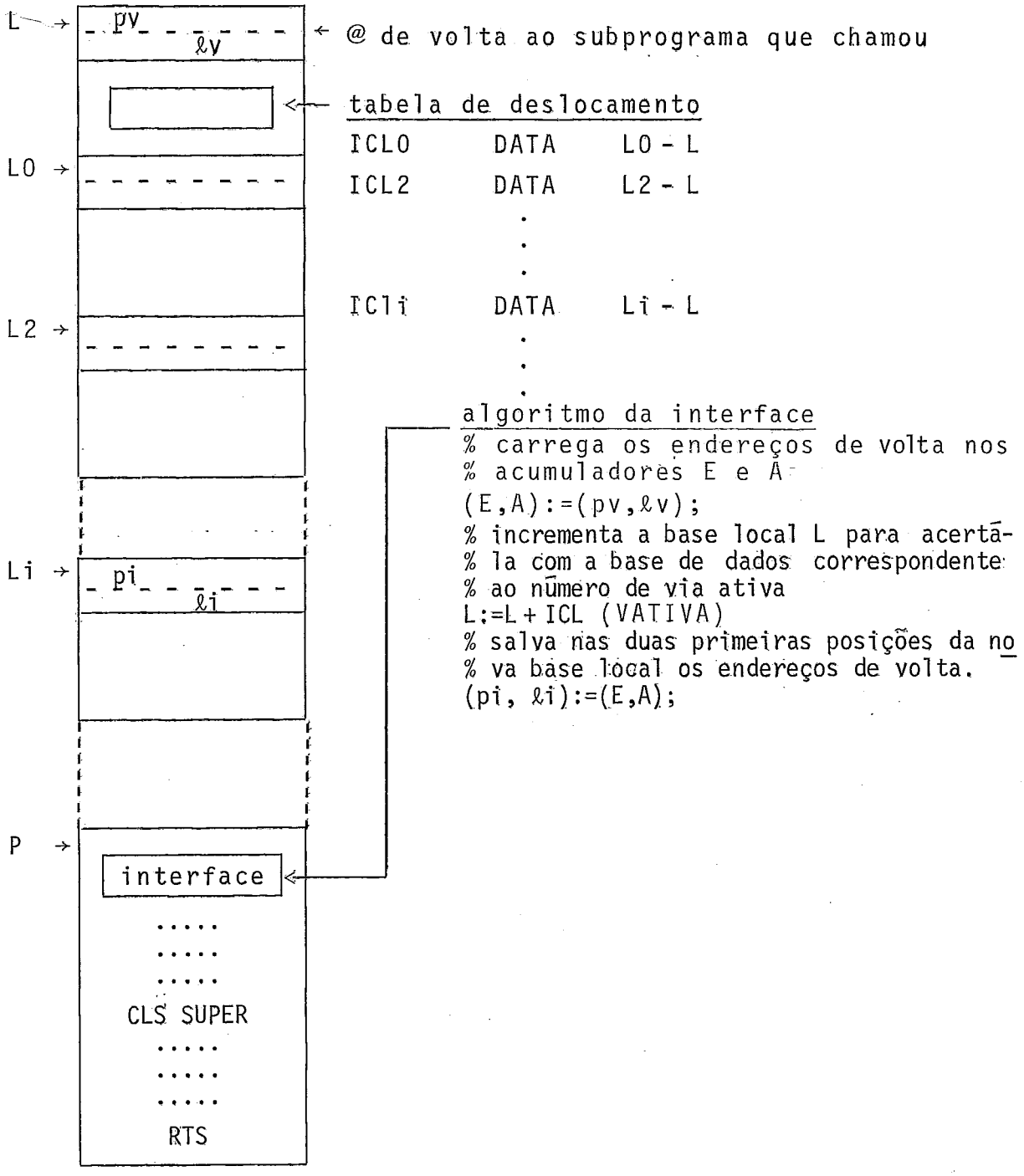



Fig. III.33: Tratamento de reentrância no nível de um subprograma

## Compactação - Descompactação de Brancos

A fim de economizar o espaço disco, reduzindo também o número de acessos, implantamos algoritmos muito simples de

compactação - descompactação de caracteres brancos repetidos e eliminamos os brancos finais dos registros.

Os caracteres armazenados nos arquivos do editor sendo do tipo ASCII sem paridade, utilizamos caracteres com paridade posicionada a 1 (bit mais significativo do caractere a 1) como caracteres de compactação. Este tipo de caractere indica quantos brancos ele compacta.

Os resultados obtidos dependendo do tipo da linguagem do programa editado são resumidos na tabela seguinte:

| linguagem do programa editado | nº de registros do programa | nº de registros por setor sem compactação (com eliminação dos brancos finais) | nº de registro por setor com compactação | ganho em % |
|---|---|---|---|---|
| LP 15 E (linguagem do nível médio do Mitra 15) | 845 | 5,28 | 7,35 | 28,13 |
| PASCAL (B 6700) | 454 | 5,82 | 7,09 | 17,95 |
| ASS 2 X (linguagem de montagem do Mitra 15) | 1731 | 6,12 | 7,40 | 17,31 |

Fig.III-34: Resultados obtidos com simples compactação de brancos nos registros

## III.5 - Algoritmos de Edição

Como apresentado na figura III-35 o editor é constituído de vários níveis de algoritmos de edição. O nível de supervisão possui os algoritmos de gerenciamento da tela dos terminais (apresentação das linhas, controle do deslocamento vertical do cursor) e de execução dos comandos de edição. Este nível se utiliza dos recursos oferecidos pelos outros níveis. O nível de edição de uma linha manipula o buffer de edição de uma linha, permitindo principalmente as funções de substituição, inserção e remoção de caracteres numa linha, e controla e otimiza o deslocamento horizontal do cursor. O nível de gerenciamento dos arquivos seqüenciais indexados (definição, associação, abertura, fechamento de arquivo, E/S sobre arquivo) é assegurado principalmente por algoritmos do sistema SGF 15 E. Enfim um último nível representa os algoritmos dos módulos de serviço tais como do analisador de comandos, das funções de conversões e de compactação e descompactação de brancos.

Obviamente, o editor é constituído de uma grande variedade de algoritmos em todos esses níveis e não caberia aqui descrever todos eles.

No nível de gerenciamento dos arquivos S.I. apresentamos o algoritmo de pré-tratamento da seleção (SLT) de um registro e o algoritmo de leitura (GET) de um registro, função existente também no sistema SGF 15 E e que reprogramamos por razões explicitadas mais adiante.

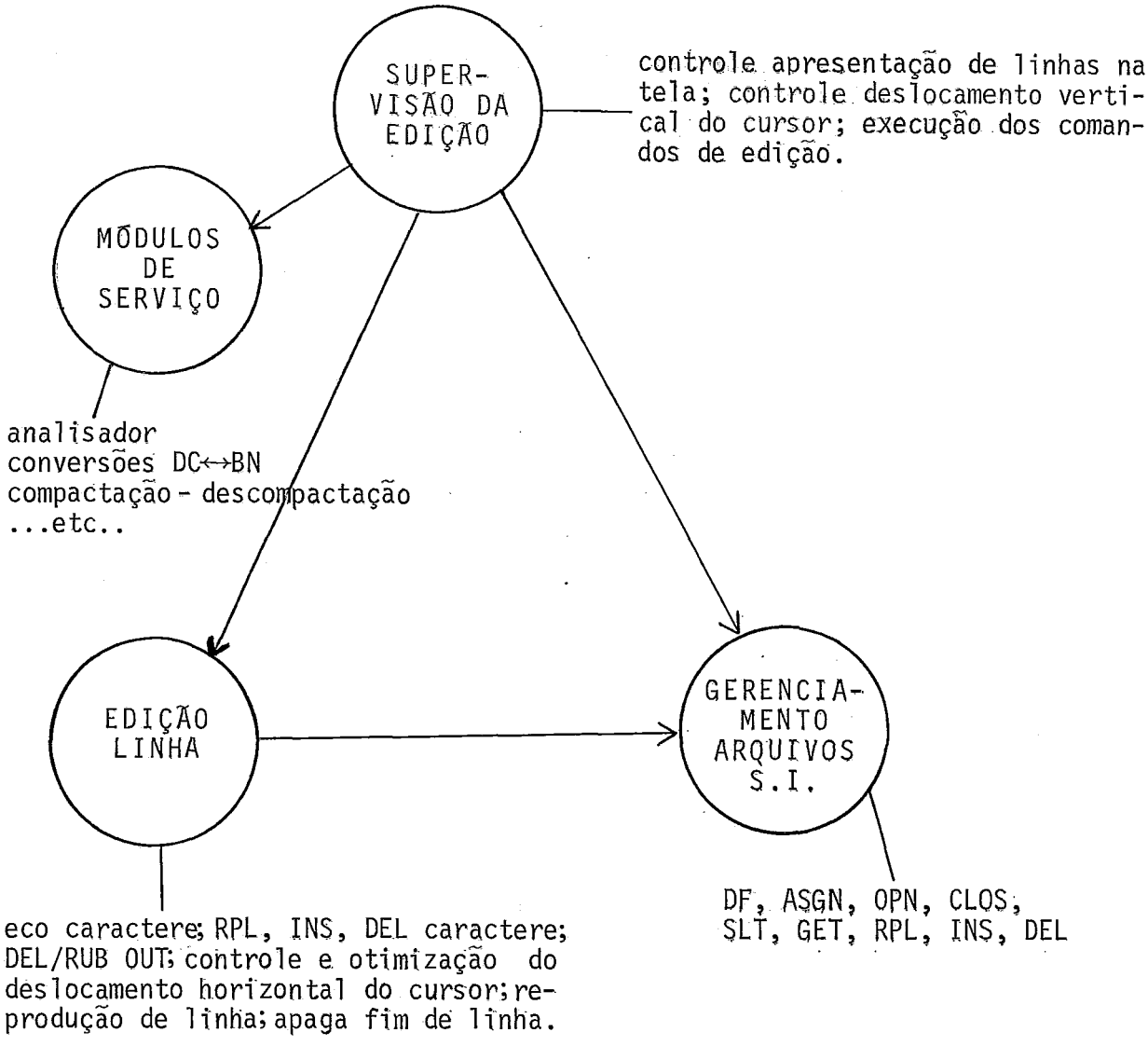



Fig.III-35: Os diferentes níveis de algoritmos do editor

No nível de supervisão da edição, descrevemos o algoritmo de procura de uma cadeia de texto num arquivo.

Algoritmo de Pré-Tratamento da Seleção de um Registro S.I.

A seleção de registro pelo sistema SGF 15 E (função SLT) sendo feita por consulta sistemática do bloco de índices de mais alto nível até o bloco de índices de mais baixo nível

(ver descrição p.169) quisemos, graças a um algoritmo de pré-tratamento da seleção, evitar os inúteis acessos a disco no caso de o registro a selecionar já estar presente no buffer de dados do usuário, o que é muito freqüente no editor.

O algoritmo apresentado manipula parâmetros da zona de gerenciamento do buffer de registros (22 primeiros bytes de BREG) e da tabela OLFI do S.O. mantendo total compatibilidade com o sistema SGF 15 E.

Para facilitar a descrição do algoritmo, os parâmetros principais do buffer de registro receberão os nomes indicados na figura seguinte:

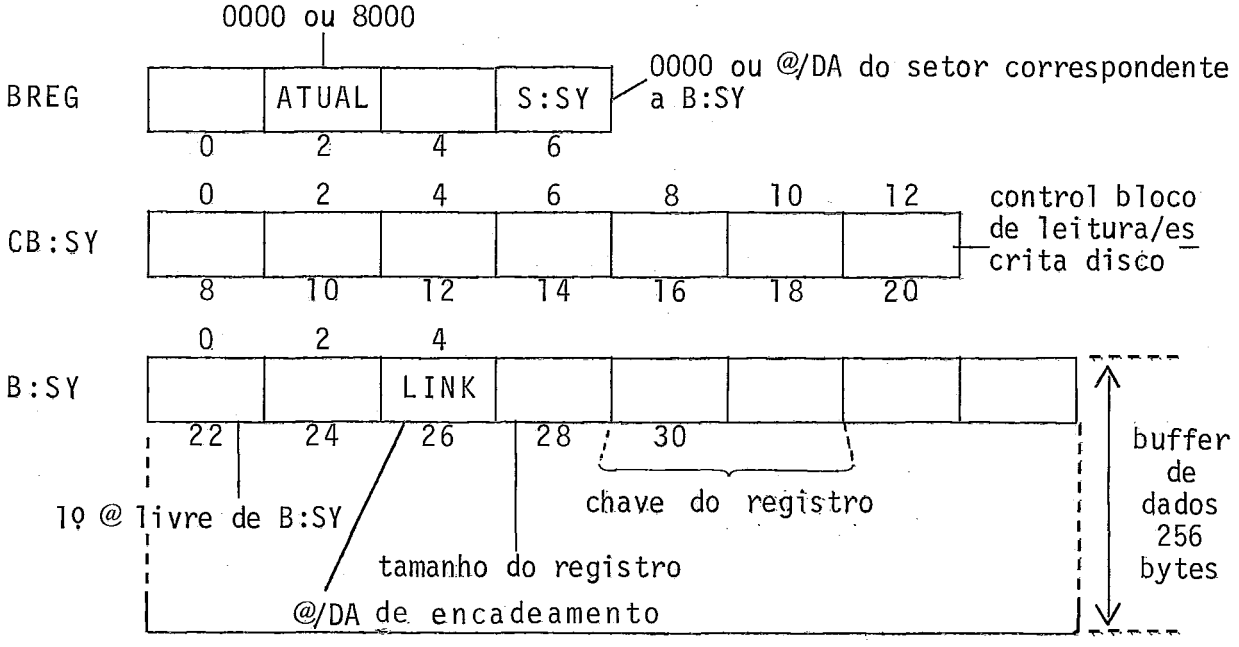



Fig. III-36: Identificação dos parâmetros de BREG utilizados no algoritmo de pré-tratamento da seleção de registro

A função do algoritmo é procurar no buffer de dados (B:SY) a presença do registro a selecionar (LINI) e, em caso

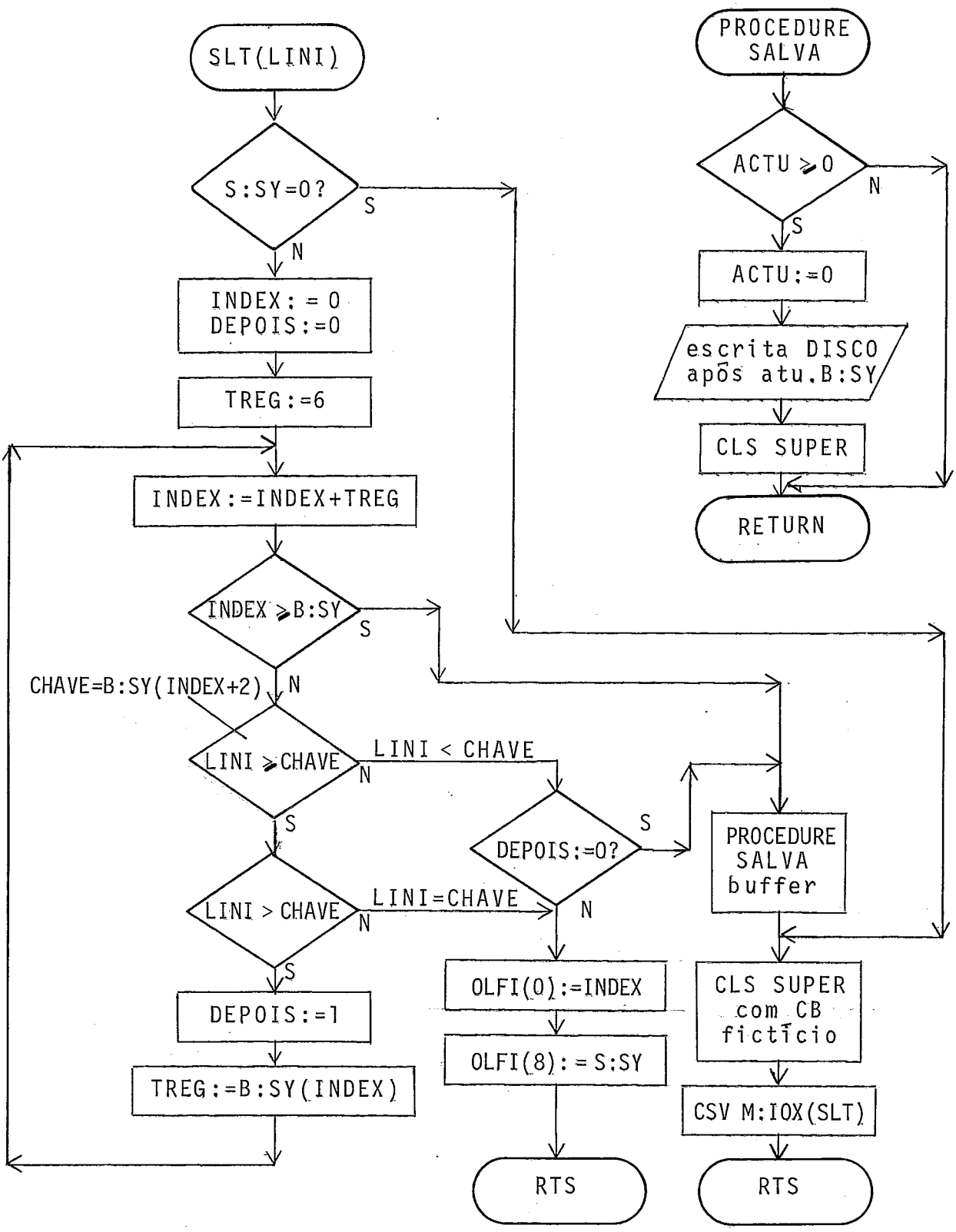



Fig.III-37: Algoritmo de pré-tratamento da seleção de um registro S.I.

de sucesso, atualizar os parâmetros OLFI(0) e OLFI(8) da tabela OLFI. No caso contrário o módulo de seleção de SGF 15 E é chamado.

O algoritmo é mostrado na fig.III-37. Se o parâmetro S:SY é nulo, isto significa que o bloco B:SY não é um bloco de dados e neste caso a seleção do registro deverá ser feita através de SGF 15 E. INDEX é uma variável de acesso ao tamanho [TREG] dos registros de B:SY[B:SY(INDEX)] e à chave [CHAVE] destes registros [B:SY(INDEX + 2)].

Algoritmo de Leitura do Tamanho e da Chave de um Registro S.I.

A função de leitura de registro assegurada por SGF 15 E foi reprogramada pois transferia todo o registro apontado por OLFI(0) no buffer do usuário enquanto o editor, em muitos casos, precisava somente conhecer o tamanho e a chave deste registro. Com a introdução da compactação - descompactação de brancos, a função GET de SGF 15 E revelou-se inútil, a transferência de um registro para o buffer usuário podendo ser efetuada diretamente pela rotina de descompactação, evitando o uso de um buffer intermediário.

O algoritmo apresentado trabalha com a tabela OLFI e o buffer de dados B:SY, como o algoritmo precedente de pré-tratamento da seleção de registro, e fornece, no buffer do usuário, o tamanho e a chave do registro de B:SY apontado por OLFI(0).

Os parâmetros de entrada são os seguintes:

- o buffer B:SY deve ser atualizado;
- OLFI(0) deve apontar o endereço do registro sujeito a leitura em relação ao início de B:SY [mínimo 6, o máximo podendo ultrapassar o valor indicado por B:SY(0)].

Estes dois parâmetros de entrada podem ter sido atualizados pela seleção do registro a ler (ver algoritmo precedente) ou por chamada anterior do próprio algoritmo de leitura.

Na saída, os parâmetros são os seguintes:

- se OLFI(0) aponta um registro dentro de B:SY [OLFI(0) < B:SY(0)] a transferência do tamanho e da chave do registro no buffer usuário é efetuada. O acumulador "AC" é também atualizado com o tamanho do registro;
- se OLFI(0) aponta um registro fora de B:SY [OLFI(0) ⩾ B:SY(0)], o bloco de dados encadeado por LINK é lido (se existente) e OLFI(0) e B:SY atualizados assim como o buffer usuário e o acumulador, como no caso precedente;
- no caso precedente e com LINK=FFFF (não há mais bloco encadeado) assinalando o fim do arquivo, o acumulador e a palavra mais significativa da

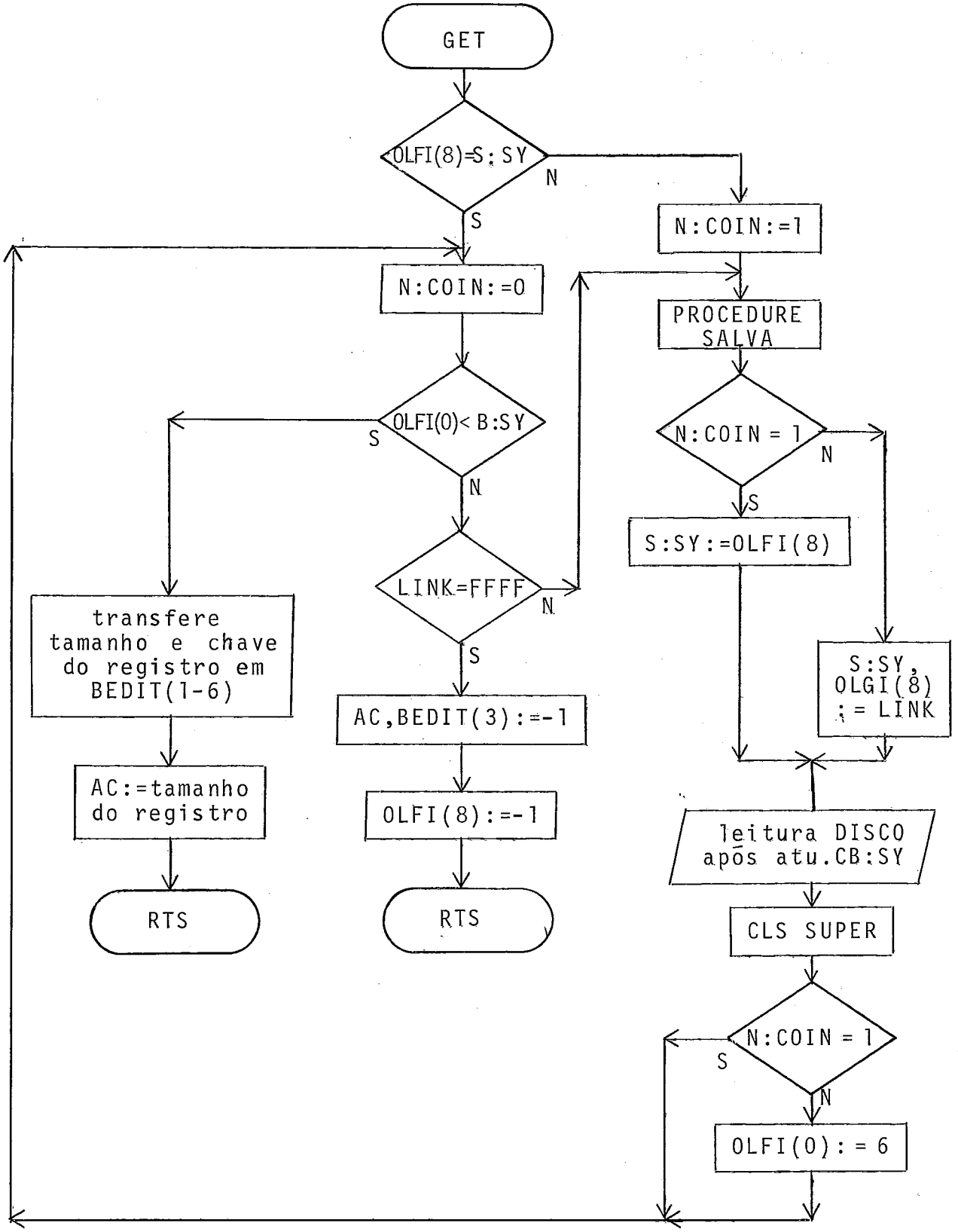



Fig. III-38: Algoritmo de leitura do tamanho e da chave de um registro S.I.

chave do registro são transferidos para o usuário com o valor hexadecimal FFFF(-1).

O algoritmo é apresentado na figura III-38.

N:COIN é um flag de não coincidência setado quando OLFI(8) e S:SY não são idênticos. BEDIT é o buffer usuário (buffer de edição) que recebe o tamanho do registro nos bytes 1 e 2 e a chave do registro nos bytes 3 a 6. SALVA é uma procedure detalhada junto ao algoritmo precedente (ver fig.III-37).

Algoritmo de Procura de uma Cadeia de Texto

O algoritmo apresentado a seguir permite descobrir a primeira ocorrência (na posição I) ou a não ocorrência de uma cadeia de caracteres CAD de tamanho TAMCAD num texto TEXT de tamanho TAMTEXT.

Um dos métodos mais conhecidos para tal algoritmo é o de KNUTH - MORRIS - PRATT (KNUTH[29]) que necessita de I+TAMCAD-1 comparações para achar a cadeia procurada enquanto o algoritmo comum de procura, no pior dos casos, precisaria de IxTAMCAD comparações.

O algoritmo escolhido neste trabalho é baseado no estudo de BOYER - MOORE (BOYER[4]) e pode ser considerado como "sublinear", ou seja, em média precisa efetuar somente C x (I+TAMCAD) comparações para acertar sua busca, onde C < 1.

A idéia do algoritmo é comparar os caracteres da cadeia CAD com os do texto TEXT começando pelo fim da cadeia.

Dependendo do resultado das comparações, o algoritmo poderá efetuar um salto importante no texto antes de reinicializar novas comparações sempre a partir do fim da cadeia.

A eficiência do algoritmo aumenta com o tamanho TAMCAD da cadeia a procurar.

O Algoritmo

As variáveis utilizadas no algoritmo são as definidas anteriormente.

CAD(J) representa o Jº caractere da cadeia a procurar (contando-se a partir de 1 à esquerda) enquanto TEXT(I), o Iº caractere do texto objeto da pesquisa.

O presente algoritmo supõe a existência de duas tabelas Δ1 e Δ2 que determinam o salto de I no decorrer do tratamento. A primeira tabela possui tantas entradas quantos são os caracteres existentes no alfabeto utilizado. A entrada para um caractere CAR será Δ1(CAR). A segunda tabela possui tantas entradas quantos são os caracteres existentes na cadeia CAD. A Jº entrada será Δ2(J).

As duas tabelas são constituídas de inteiros positivos ou nulos.

São inicializadas pelo pré-processamento da cadeia CAD da seguinte maneira:

Tabela Δ1(CAR):

- se CAR for um caractere que não pertence à cadeia CAD então Δ1(CAR):=TAMCAD

- senão Δ1(CAR):=TAMCAD - J onde J é o maior inteiro tal que CAD(J) = CAR.

Exemplo:

cadeia: OS BLOCOS

posição do caractere: 123456789

caracteres aparecendo na cadeia: BCLOS branco

maior inteiro J tal que CAD(J)=CAR: 47589 3

Δ1 correspondente: 52410 6

para todos os outros caracteres do alfabeto Δ1 = 9

Tabela Δ2(J):

Δ2 é a distância da qual se deve avançar a cadeia para alinhar a subcadeia dos últimos TAMCAD - J caracteres da cadeia, identificados no texto com a possível reocorrência desta sub-cadeia mais à direita na cadeia (PRMD).

Para definir esta reocorrência PRMD, o critério usado é o seguinte:

Seja $ um caractere que não pertence à cadeia CAD e que será colocado na frente da cadeia para os valores de J inferior a 1 tal que CAD(J) = $ para J < 1.

Duas seqüencias de caracteres $[C_1 \ldots C_N]$ e $[D_1 \ldots D_N]$ serão ditas "unificadas" se, para todo J de 1 a N, $C_J = D_J$ ou $C_J = \$$ ou $D_J = \$$.

Finalmente, para a posição J da cadeia, PRMD(J)- possível reocorrência mais à direita de uma sub-cadeia que começa na posição J + 1 - é o maior valor de $K \leq$ TAMCAD tal que as duas seqüências [CAD(J+1) ... CAD(TAMCAD)] e [CAD(K) ... CAD(K+TAMCAD--(J+1))] sejam "unificadas" e $K \leq 1$ ou CAD(K - 1) $\neq$ CAD(J).

Podemos enfim definir Δ2 como:

$$\Delta 2(J) = \text{TAMCAD} + 1 - \text{PRMD}(J).$$

Exemplo:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| cadeia: | | | | | | | O | S | | B | L | O | C | O | S |
| introdução dos $ | | | | | | | | | | | | | | | |
| para J < 0 : | $ | $ | $ | $ | $ | $ | O | S | | B | L | O | C | O | S |
| posição: | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| PMRD(J): | | | | | | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 0 | 9 | |
| Δ2 : | | | | | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 10 | 1 | |

A seguir mostramos finalmente o algoritmo apresentado numa linguagem de alto nível e sob forma de uma procedure.

```
procedure PROCURA;
     begin
     FIM: = false;
     I: = TAMCAD;
     while(I < = TAMTEXT) and (not FIM)
     do begin
        J:=TAMCAD;
        ACABOU:=false;
        while (J <> 0) and (not ACABOU)
        do begin
           if TEXT(I) = CAD(J)
           then begin
                J: = J - 1;
                I: = I - 1;
                end
           else ACABOU: = true;
           end;
        if J <> 0
        then I: = I + max[DELTA1(TEXT(I)),DELTA2(J)]
        else FIM: = true;
        end;
     if FIM
     then LISTA;    % cadeia achada na posição I + 1
                    % o texto será listado na tela
     end;
```

Exemplo:

A partir de um exemplo, mostraremos agora o fun-

cionamento do algoritmo. Sejam a seguinte cadeia a procurar e o seguinte texto objeto da procura:

```
cadeia   : OS BLOCOS
texto    : DEPENDENDO DO NOVO TAMANHO DO REGISTRO , OS BLOCOS SERÃO ...
posições: 123456789      18   25     34   41 43   50    56
                           19              42
```

Δ1 e Δ2 já foram calculados em exemplos anteriores.

Mostraremos principalmente a variação de I e J a cada passo do algoritmo.

passo 1:

I = 9, J = 9; TEXT(9) < > CAD(9)

I = I + max [Δ1(TEXT(9)),Δ2(9)]

= 9 + max [9, 1]

= 18

passo 2:

I = 18, J = 9; TEXT(18) <> CAD(9)

I = I + max [Δ1(TEXT(18)), Δ2(9)]

= 18 + max [1, 1]

= 19

passo 3:

I = 19, J = 9; TEXT(19) < > CAD(9)

I = I + max [Δ1(TEXT(19)), Δ2(9)]

= 19 + max [6, 1]

= 25

passo 4:

I = 25, J = 9; TEXT(25) < > CAD(9)

I = I + max [Δ1(TEXT(25)), Δ2(9)]

= 25 + max [9,1]

= 34

passo 5:

I = 34, J = 9; TEXT(34) < > CAD(9)

I = I + max [Δ1(TEXT(34)), Δ2(9)]

= 34 + max [9, 1]

= 43

passo 6:

I = 43, J = 9; TEXT(43) = CAD(9)

I = 42, J = 8; TEXT(42) = CAD(8)

I = 41, J = 7; TEXT(41) < > CAD(7)

I = I + max [Δ1(TEXT(41)), Δ2(7)]

= 41 + max [6, 9]

= 50

passo 7:

I = 50, J = 9; TEXT(50) = CAD(9)

I = 49, J = 8; TEXT(49) = CAD(8)

- - - - - - etc - - - - - -

I = 42, J = 1; TEXT(42) = CAD(1)

I = 41, J = 0; FIM:=true

A cadeia foi encontrada na posição I + 1, ou seja, 42 e será listada na tela.

Neste exemplo vemos que, para encontrar a cadeia na posição 42, foram feitas somente 8 comparações com referência ao texto, mais as 9 finais para confirmar a identificação da cadeia.

# IV. IMPLEMENTAÇÃO E CONCLUSÕES

## IV.1 - Implementação

Como já foi dito, o presente trabalho começou pela implantação de um novo Sistema Operacional com novo sistema de gerenciamento de arquivo e novo handler de linha assíncrona. Esta implantação foi um pouco demorada devido à escassez de informações à nossa disposição, à dificuldade de obtê-las e foi baseada quase exclusivamente em nossa experiência dos sistemas anteriores, o que nos permitiu descobrir, por exemplo, apontadores do S.O. ausentes, outros errados sem poder ter facilidade de acesso à fonte.

Um dos nossos primeiros trabalhos foi também desenvolver um programa de interfaceamento (denominado CANDE) terminal vídeo/linha CANDE - B 6700 através do Mitra 15 permitindo, além do uso normal da tela e do teclado do terminal como se fosse diretamente ligado à linha do CANDE, a recepção em disco do Mitra de arquivos do B 6700 e o envio para o B 6700 de arquivos do Mitra (em disco ou lidos na leitora de cartões).

Foi tal programa que nos permitiu, inicialmente, criar e editar os programas do EDITOR até este último chegar a uma fase de auto-suficiência mínima permitindo boot - straps posteriores.

Para aproveitar ao máximo a pouca memória disponível no início do projeto (somente 7,5 Kbytes), desenvolvemos o e-

ditor em linguagem de montagem do Mitra (ASS 2 X) (CII[8]) com exceção do analisador que foi escrito em LP 15 E, linguagem algoritmica de nível médio de bom rendimento na geração de código máquina (CII[10]).

Durante todo o desenvolvimento do software, o tamanho dos programas foi sempre nossa grande preocupação e sempre tentamos ganhar alguns bytes de um lado ou de outro, optando em todos os casos para as soluções mais econômicas em termos de ocupação memória. Por exemplo, chegamos a utilizar a TWB (task working block) de 72 bytes, normalmente de uso exclusivo do sistema operacional para permitir a reentrância dos seus módulos, como zona de passagem de parâmetros e de trabalho de alguns subprogramas do EDITOR. Na programação do analisador (PARSER + tradutor, SCREENER, SCANNER) medimos um ganho memória de mais de 7% na utilização deste recurso. Também as mensagens emitidas pelo EDITOR são guardadas em arquivo em disco e não residentes em memória.

Apesar de todos estes cuidados, não conseguimos implementar na sua totalidade o editor descrito no presente trabalho por insuficiência da memória.

Era também um pouco ilusório conseguir desenvolver, em somente 7,5 Kbytes, um editor do porte proposto.

Todavia, conseguimos implantar as grandes linhas do editor, testando os algoritmos principais e hoje temos à disposição dos usuários um editor "full screen" reentrante funcio-

nando com dois terminais.

Este editor que podemos chamar de intermediário, no qual o analisador não foi incorporado, trabalha graças às funções implementadas com caracteres de controle e o equivalente dos comandos LIST e SEQ. Ocupa 7,312 Kbytes dos 7,478 Kbytes memória disponíveis.

Pode ser considerado como um editor completo. Uma fase de inicialização interativa pede ao usuário conta e senha e, após abertura de uma sessão, o nome do arquivo a criar ou editar. O encerramento da sessão pelo usuário reativa esta fase inicial. Um programa auxiliar (programa OUT) permite copiar arquivos do editor para o disco fixo a fim de posterior compilação no Mitra (compiladores ASS 2 X e LP 15 E), listagem (programas LISTAR e PRINT) ou transmissão (programa CANDE) por linha assíncrona.

Outro programa auxiliar (programa IN) permite copiar arquivo em disco ou lido na leitora de cartões para um arquivo seqüencial indexado do editor.

A utilização sucessiva dos programas OUT e IN para um arquivo do editor permite seu resseqüenciamento.

As funções de remoção de arquivo e listagem de diretório são asseguradas pelo utilitário de gerenciamento de arquivos UGF 15 E do sistema (CII[15]).

O uso atual dos discos do Mitra é mostrado na se-

guinte figura:

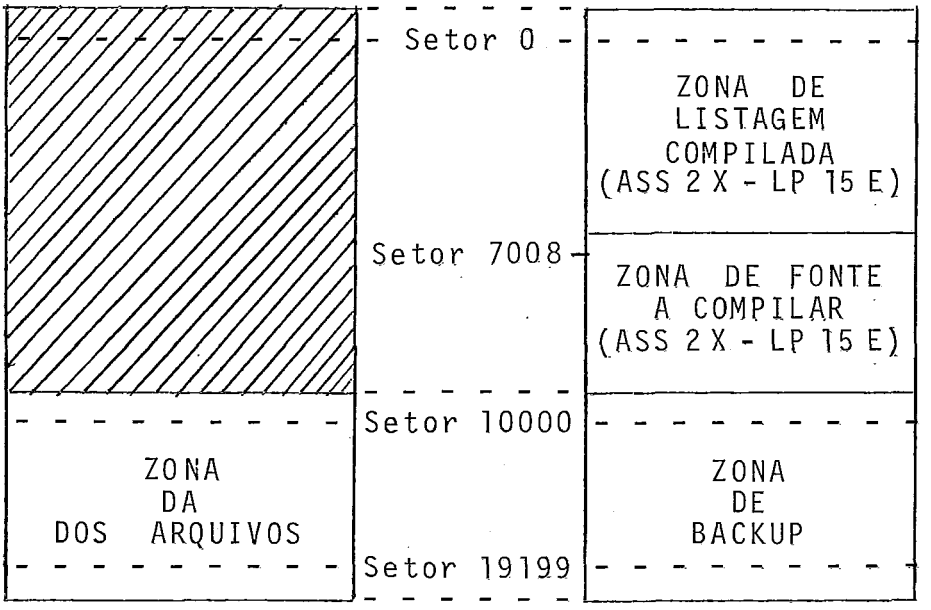



Fig.IV-1: Uso atual dos discos do Mitra com o editor implantado

A escolha das zonas no disco fixo foi feita para minimizar o deslocamento das quatro cabeças magnéticas (uma por face) sendo elas solidárias e levando em conta a existência de 48 setores por cilindro em cada disco.

Tivemos de efetuar algumas modificações nos control blocos dos compiladores ASS 2 X e LP 15 E e no handler disco para permitir aos compiladores compilar e listar simultaneamente no disco fixo.

Nessa fase de constantes modificações com riscos elevados de acidentes nos testes reservamos uma zona de backup no disco fixo idêntica à zona DA do disco removível e desenvolvemos um programa geral de backup (programa COPIA).

A partir de uma versão mais antiga e então mais reduzida desse editor intermediário (sem compactação-descompactação e com algumas funções a menos) e para um só terminal, incorporamos o analisador, permitindo testes de velocidade deste no contexto global do editor assim como testes de deteção e apresentação de erros em comandos. Implementamos também o sistema de restituição dos dez últimos comandos submetidos ao editor.

O espaço ocupado por este editor de teste foi de 7,348 Kbytes.

O software desenvolvido até o momento para o editor é de 8000 linhas (6.650 em ASS 2 X e 1.350 em LP 15 E) gerando 2.550 instruções máquina) sem contar nenhum dos programas anexos. Usamos uma técnica de programação modular (são atualmente 22 subprogramas) com o compromisso de poupar a memória, o que nos obrigou a aceitar subprogramas reentrantes um pouco grandes em alguns casos (até aproximadamente 2.000 linhas). Nos subprogramas deste tipo determinamos procedures para não sobrecarregar a programação e reduzir o código.

Adotamos uma certa metodologia de desenvolvimento, indispensável no uso prolongado de uma linguagem de montagem e principalmente todos os programas foram extremamente bem documentados. O programa principal do EDITOR serve exclusivamente para parametrizar as características diversas dos terminais a serem suportados e acertar apontadores do S.O. para o editor não depender de uma nova geração de S.O.. Para otimizar o código em função dos números e do tipo de terminais a serem atendidos, utili-

zamos uma facilidade do montador ASS 2 X pouco comum em compiladores e que é uma diretiva de compilação condicional. Para facilitar os testes do EDITOR na sua elaboração, introduzimos uma opção de depuração por compilação condicional e seleção de função de depuração nas chaves do painel do Mitra.

## IV.2 - Conclusões

### Avaliação do Editor Implantado

Uma grande motivação no decorrer deste trabalho foi poder beneficiar, a cada nova versão do editor, de novas facilidades para continuar o seu próprio desenvolvimento. A mudança, na elaboração dos programas, do CANDE do B 6700 para uma das primeiras versões auto-suficientes do editor foi para nós um grande passo por nos permitir um ganho em velocidade de trabalho importante. Além das demoras devido às filas no B 6700 e apesar de sua sofisticação, o CANDE é um editor de linha. São as facilidades de edição "full screen" que permitem numa edição habitual dar em termos de velocidade de trabalho uma vantagem considerável ao editor implantado hoje no Mitra em relação ao CANDE.

Diariamente o editor foi usado com dois terminais editando arquivos de 2.000 linhas (2 níveis de índices) e os tempos de resposta quando perceptíveis, foram considerados muito bons.

A geração automática de uma nova linha em modo seqüenciamento após a inserção de uma linha, que constitui a opera

ção de edição mais demorada em alguns casos, nunca passa de meio segundo. O tempo de resposta a um comando de listagem de qualquer parte de arquivo é de alguns décimos de segundo, e a conseqüente visualização na tela não apresenta irregularidade de velocidade.

A expansão de brancos comprimidos, sistemática antes do envio de uma linha na tela, poderia ser considerada como altamente ineficiente. Bem ao contrário, ela permite aumentar a velocidade global do editor diminuindo o número de acessos a disco pela maior quantidade de registros cabendo em cada setor do arquivo. O que pode limitar no nosso caso a velocidade do editor não é o uso da CPU executando de 300.000 a 400.000 instruções por segundo mas o uso do disco permitindo uma média bastante razoável de 19,8 acessos aleatórios por segundo.

A utilização de um buffer de trabalho de somente uma linha para modificações diretas na tela não acrescentou atrasos notáveis devido ao pré-tratamento da seleção de registro que introduzimos limitando, para atualização deste buffer no pior dos casos, a uma média apenas superior a 3 acessos a disco por página de 24 linhas.

Os ganhos obtidos por tal escolha em termos de utilização memória foram apreciáveis e medimos um acréscimo de código de somente 1,304 Kbytes entre uma versão do EDITOR com um terminal Embracomp de 24 linhas e outra versão com dois terminais, o terminal precedente e um terminal Iriscope de 16 linhas.

Em termos de filosofia de edição, os primeiros usuários ou interessados apreciaram o aspecto "full screen" do editor e em particular as facilidades de inserção e supressão de caracteres numa linha, o comando DELETE/RUB OUT que restitui a linha original do cursor e que pode ser usado em caso de erro de digitação, a duplicação de campos de linha, o modo seqüenciamento que permite geração automática de novas linhas numeradas com indentação automática e apresentação de eventuais linhas intermediárias.

## Extensões

No estágio atual deste projeto está sendo instalada a expansão memória do Mitra de 32 Kbytes elevando ao máximo de 64 Kbytes sua capacidade total. Isto vai nos permitir implantar a totalidade do editor planejado no presente trabalho e integrar o programa de comunicação tipo RJE com B 6700. Com o sistema operacional atual a disponibilidade de memória para o usuário cresce de 7,478 Kbytes para 40,246 Kbytes, ou seja, passa a ser 5,4 vezes maior.

Na realidade o ganho para o usuário será um pouco menor pois estamos pensando gerar um novo sistema operacional um pouco maior que o atual com ponto flutuante e handler relógio tempo real para permitir compilar e executar programas em simultaneidade com o editor e RJE.

Para maior eficiência do sistema, evitando transferências supérfluas de arquivo seqüencial indexado para arquivo

seqüencial, pensamos modificar os compiladores do Mitra para que possam trabalhar diretamente com os arquivos seqüenciais indexados gerados pelo editor.

Prevendo a utilização do editor por um número grande e crescente de usuários, o disco fixo deverá ser também empregado como zona DA gerenciada por SGF 15 E e suportando o editor. Assim, a zona disco usuário passará de 2,355 Mbytes (9200 setores) atuais para 7,270 Mbytes (28400 setores), ou seja, tornar-se-á um pouco mais de 3 vezes maior.

O presente trabalho e suas extensões deixarão no Mitra 15 um sistema bastante interessante, permitindo uma produção de software bem mais eficiente e agradável que no passado, contribuindo assim com os planos de melhoria das condições de ensino e de pesquisa no programa de sistemas da COPPE.

# V. BIBLIOGRAFIA

1. ALVES, S. de Brito - Um Editor Conversacional para um Mini-Computador, Rio de Janeiro, Tese de M.Sc., COPPE/UFRJ, 1976. 106 p.

2. APPLE, Apple Computer Inc. - Apple Pascal Operating System Reference Manual. Chapter 4. The Editor [A2L0 028(030-0100-00)], USA, Apple C.I., 1980. p.70-126.

3. ARAUJO, W. Barbosa de - Um Editor Interativo para o Terminal Inteligente, Rio de Janeiro, Tese de M.Sc., COPPE/UFRJ, 1981. 68 p.

4. BOYER, R.S., MOORE, J.S.-A Fast String Searching Algorithm, Communications of the ACM, USA, 20(10):762-772, 1977.

5. BURROUGHS, Burroughs Corporation - B 7000 / B 6000 Series. Remote Job Entry. Reference Manual (5001548), USA, Burroughs, 1977. 106 p.

6. _____ - B 7000/B 6000 Series. CANDE. Reference Manual (5010259), USA, Burroughs, 1978. 205 p.

7. CDC, Control Data Corporation - XEDIT. Version 3.1.00 User Info Manual (76071000), USA, CDC, 1980, 90 p.

8. CII, Compagnie Internationale pour l'Informatique-Mitra 15. Manuel de Référence. Tome 1 (4047 U2/FR), França, CII, 1975. 260 p.

9. _____ - MITRA 15. Manuel de Référence entrées/sorties. Tome 2 (4058 U/FR), França, CII, 1975. 68 p.

10. _____ - MITRA 15. Language LP15, LP15E (4109 U5/FR), França, CII, 1975. 110 p.

11. _____ - MITRA 15. Moniteurs (4557 U/FR), França, CII, 1976. 127 p.

12. _____ - MITRA 15. Moniteur Temps Réel Disque MTRD (4117 U2/FR), França, CII, 1975. 108 p.

13. _____ - MITRA 15. Génération de Système (4255 U1/FR), França, CII, 1975. 37 p.

14. _____ - MITRA 15. GENTES, Générateur de Tables d'Entrées-Sorties, (DPOAS/A/75/38155), França, CII, 1975. 17p.

15. _____ - MITRA 15. SGF15E - UGF15E (4589 U1/FR), França, CII, 1976. 97 p.

16. COBRA, Computadores e Sistemas Brasileiros S/A - COBRA 700. Manual de Operação SOP 700 (70 SUT 60021S79). Editor (7 TED 02, Versão 1), Rio de Janeiro, COBRA S/A, 1982. 16 p.

17. _____ - COBRA 500. Programa Editor de Telas (50 SUT 230021J72), Rio de Janeiro, COBRA S/A, 1982. 31 p.

18. _____ - COBRA 300/TD. Programa Editor (T3SUT090041J72), Rio de Janeiro, COBRA S/A, 1981. 39 p.

19. DEC, Digital Equipment Corporation - RSX-11M/RT11. PDP-11. Keypad Editor User's Guide (AA-J692A-TC), USA, DEC, 1980. 153 p.

20. _____ - RSTS/E. Editing and Text Formating. Volume 4. EDT Editor Manual V2.0 (AA-J726A-TC), USA, DEC, 1980. 241 p.

21. DIGITAL, Digital Research - ED: A Context Editor for the CP/M Disk System. User's Manual, USA, Digital Research, 1978. 18 p.

22. ELLIOTT, B. - Design of a Simple Screen Editor, Software - Practice and Experience, U.K., 12(4):375-384, 1982.

23. EMBLEY, D.W., NAGY, G. - Behavioral Aspects of Text Editors, Computing Surveys, USA, 13(1):33-70, 1981.

24. FINSETH, C.A. - Theory and Practice of Text Editors, USA, B.S.thesis, MIT, 1980. 106 p.

25. FRANÇA, P.M. Bianchi - Sistema Preparador de Textos e Programas, Rio de Janeiro, Tese de M.Sc., COPPE/UFRJ, 1974. 79 p.

26. FRASER, C.W. - A Generalized Text Editor, Communications of the ACM, USA, 23(3):154-158, 1980.

27. IBM, International Business Machines Corporation - EDGAR: A Graphics Editor for CMS/370, Vers.4, Lev.0(5796-PJP), USA, IBM, 1977. 55 p.

28. _____ - IBM Virtual Machine Facility/370. CMS User's Guide. Section 5. The CMS Editor (GC20-1819), USA, IBM, 1979. 34 p.

29. KNUTH, D.E., MORRIS, J.H., PRATT, V.R. - Fast Pattern Matching in Strings, SIAM Journal on Computing, USA, 6(2): 323-350, 1977.

30. MENDONZA, M. Cano -, Rio de Janeiro, Tese de M.Sc., COPPE/UFRJ, a publicar.

31. OLIVEIRA, C. Ipólitto de - INTERACT. Publicação Interna INT-0681, Rio de Janeiro, Rio Datacentro/PUC, 1981. 148 p.

32. SCOWEN, R.S. - A Survey of Some Text Editors, Software - Practice and Experience, U.K., 11(9):883-906, 1981.

33. TELES, A.A.S., SIMONE, E. - Gerador de Analisadores Sintáticos RRP LL(1), Anais do VIII Seminário Integrado de Software e Hardware, Florianópolis: 387-398, 1981.

34. VAN DAN, A., RICE, D.E. - On-line Text Editing: A Survey, Computing Surveys, USA, 3(3):93-114, 1971.