EDITOR DE REFERÊNCIAS EXTERNAS PARA MICROCOMPUTADOR

Milton de Albuquerque Bezerra

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 CIÊNCIAS (M.Sc.).

Aprovada por:

Prof. Guilherme Chagas Rodriques
(Presidente)

Prof. Jayme Luiz Szwarcfiter

Prof. Ivan da Costa Marques

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 1978

BEZERRA, Milton de Albuquerque

Editor de Referencias Externas para Microcomputador [Rio de Janeiro] 1978

VIII, 86p 29,7cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas e Computação. 1978)

Tese - Universidade Federal do Rio de Janeiro, Faculdade de Engenharia

1 - Assunto: Editor de Referencias Externas
I. COPPE/UFRJ II. TITULO: Editor de Referencias Externas para Microcomputador (série)

AGRADECIMENTOS

A todos os colegas e amigos que direta ou indiretamente contribuíram para a conclusão deste trabalho; em especial ao orientador Guilherme Chagas Rodrigues pelo apoio e incentivo, aos colegas José Carlos Vida Cura e José Antonio dos Santos Borges pelas sugestões e participação na implantação do sistema, e ainda Marta Vinhas pela dedicação no trabalho de diagramação e datilografia.

S U M Á R I O

Neste trabalho é apresentado um Editor de Referências Externas desenvolvido como parte de um sistema operacional em disco para microcomputadores. O sistema permite que o Terminal Inteligente desenvolvido no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro seja utilizado como um computador de uso geral.

O Editor de Referências Externas é orientado para este sistema operacional e tem como princípios atender as necessidades e características do referido terminal.

São oferecidas facilidades de "overlay" que se tornam indispensáveis em equipamentos que dispõem de pouca memória.

S U M M A R Y

This work presents a Linkage Editor, developped as part of an operational system on disk for microcomputers.

The system allows the Intelligent Terminal, developped at Núcleo de Computação Eletrônica of the Universidade Federal do Rio de Janeiro, to be used as general use computer.

The Linkage-Editor is oriented to this operational system and takes by principle to attend the terminal's necessities and characteristics.

As they are indispensable to low memory equipment, overlay facilities are offered by this environment.

ÍNDICE

P R E F Á C I O

A escolha do tema deste trabalho foi baseada na linha de atuação em que um grupo de pesquisadores do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro vem atuando desde meados de 1973.

As pesquisas desenvolvidas por este grupo de pesquisadores tem levado em consideração o desenvolvimento de projetos de complexidade crescente e que permitem uma integração hardware-software.

A orientação básica que vem norteando a escolha dos projetos a serem desenvolvidos é que seja possível a passagem à indústria da experiência adquirida pelo grupo, efetivando de maneira completa a absorção do know-how pela sociedade.

Dentro desta linha de atuação foram desenvolvidos vários projetos, e dentre eles, o mais importante até o momento foi o desenvolvimento em 1975 de um Terminal Inteligente, isto é um sistema programável, que vem sendo utilizado como linha para uma série de sub-projetos.

Dentro desta filosofia de se explorar ao máximo as potencialidades do referido terminal, temos dado muita ênfase ao desenvolvimento de várias aplicações das mais diversas para o Terminal.

Entre estas aplicações encontra-se uma que visa possibilitar o uso do Terminal Inteligente como um computador de uso geral, permitindo que o próprio usuário programe e processe no terminal uma determinada tarefa.

Esta aplicação foi denominada de S.O.Co, e é um Sistema Operacional em Disco, com todas as facilidades para que o Terminal Inteligente seja utilizado como microprocessador de uso geral.

O presente trabalho constitui uma das células deste sistema operacional.

Procuraremos no primeiro capítulo apresentar o Terminal Inteligente e o próprio sistema S.O.Co, de modo a determinar o contexto para o qual o "EDITOR DE REFERÊNCIAS EXTERNAS" foi definido.

Os capítulos seguintes tratam do EDITOR DE REFERÊNCIAS EXTERNAS - (REFEX) propriamente dito.

Nos capítulos finais são tratados problemas de implantação de todo o S.O.Co, e conclusões sobre o REFEX.

1º CAPÍTULO

1.1 Terminal Inteligente

A denominação TERMINAL INTELIGENTE é usada no mercado de terminais de computador para designar terminais com facilidades para executar um conjunto variado de tarefas quando devidamente programados.

Este tipo de terminal vem sendo disseminado com grande rapidez a partir da evolução tecnológica que vem ocorrendo na indústria eletrônica. A contribuição efetiva para consolidar a posição dos terminais inteligentes foi o aparecimento dos microprocessadores, ou seja, toda uma unidade central de processamento em uma única pastilha. Estes processadores possuem todos os atributos de uma UCP (Unidade Central de Processamento), com velocidade, conjunto de instruções de Entrada/Saída de poderio reduzidos, sendo o seu custo muito menor.

É fácil concluir que ao se levar inteligência a um terminal, através da introdução de um microprocessador, damos grande flexibilidade ao terminal, pois suas características podem ser facilmente alteradas por nova programação, não sendo necessário alterar o hardware.

Além disto é obvio que ao ligarmos a um microprocessador um conjunto razoável de periféricos e desenvolvermos um pequeno sistema operacional, teremos a nossa disposição um computador de pequeno porte para uso geral, dentro de certas limitações. A seguir apresentaremos a arquitetura do hardware utilizado.

O Terminal Inteligente é composto de um microprocessador, uma memória de até 16K bytes, uma pilha com 1K bytes, um canal de acesso direto a memória e um conjunto de periféricos.

Entre os periféricos temos os lentos, que executam entrada/saída através do microprocessador, e os rápidos que executam transferência em bloco através do canal.

O microprocessador é um INTEL 8008 com ciclo de instrução entre 12.5 e 27.5 microsegundos. As operações de entrada/saída requerem por volta de 10 instruções, o que limita a taxa de transferência a 10 Kbytes/segundo aproximadamente. Quando o periférico exige uma taxa de transferência maior, como por exemplo o disco, o canal de acesso direto é utilizado, o que permite taxas de transferências de ordem de 600 Kbytes/segundo.

As principais características do microprocessador são as seguintes:

- 48 instruções
- ciclo de instrução entre 12.5 e 27.5 µs
- endereçamento até 16 Kbytes
- pilha para chamada de subrotinas com 8 registros de 14 bits
- 7 registros de uso geral com 8 bits cada
- uma linha de interrupção.

A figura 1 apresenta o esquema principal com fluxo de dados e endereços.

FLUXO DE DADOS E ENDEREÇOS TI

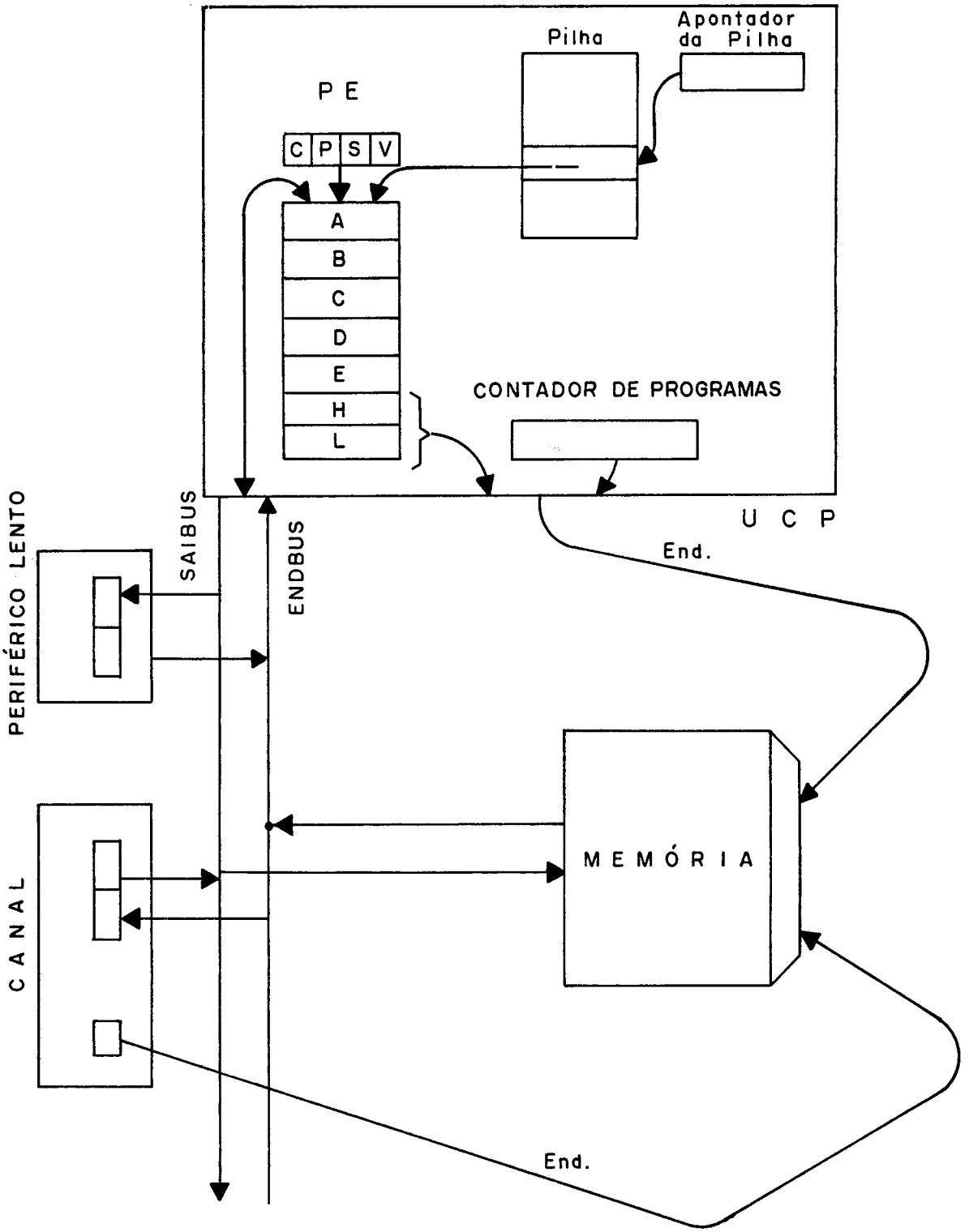


Fig. 1

Ligado a esta unidade central de processamento temos os periféricos que passaremos a descrever. Todo este conjunto forma o hardware básico utilizado pelo sistema operacional S.O.Co.

1.1.1 Painel

- 4 teclas para controle do sistema
- 8 chaves para entrada de dados
- 8 indicadores luminosos
- 2 indicadores hexadecimais

1.1.2 Console

- Teclado de uso geral com 74 teclas cobrindo todo o código ASCII
- Vídeo com tela de 12 polegadas com capacidade máxima de 24 linhas de 80 caracteres

1.1.3 Leitora de Cartões

- Velocidade de 300 cartões por minuto

1.1.4 Impressora

- Velocidade de 60 linhas por minuto
- Apresenta caracteres em matriz de pontos 5 x 7
- Imprime 64 caracteres em código ASCII

1.1.5 Disco

- Disco tipo cartucho
- Capacidade de armazenamento de 1.228.000 palavras de 16 bits.

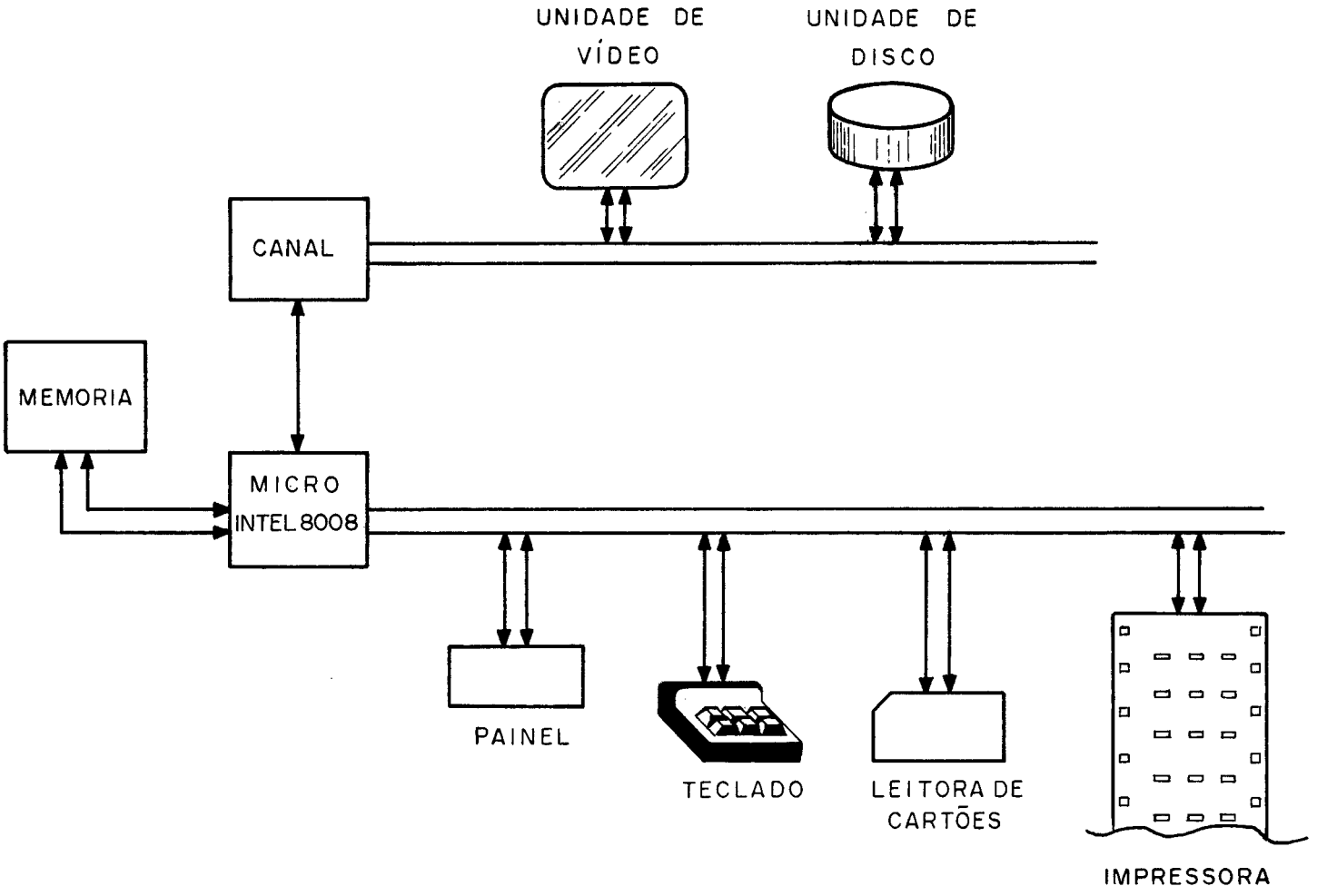

UNIDADE DE VÍDEO
UNIDADE DE DISCO
CANAL
MEMORIA
MICRO INTEL 8008
PAINEL
TECLADO
LEITORA DE CARTÕES
IMPRESSORA

Fig. 2

1.2 Sistema S.O.Co

O desenvolvimento do software para o Terminal Inteligente passou por várias etapas, sendo uma delas a implantação de um Sistema Cruzado ("Cross-Software"), processado no sistema Burroughs B6700, com objetivo de facilitar a confecção de software para o terminal. Este sistema dispõe de todos os recursos para o desenvolvimento de aplicações, com o único inconveniente de exigir um sistema hospedeiro do porte do B6700.

Partindo do princípio de que ao dotarmos o Terminal Inteligente de capacidade de desenvolver o seu próprio software, teríamos uma ampliação considerável na sua área de atuação, então partimos para a implantação de um Sistema Operacional, que permitisse:

- desenvolvimento de software básico
- desenvolvimento de aplicações
- processamento das aplicações

1.2.1 Filosofia

A definição do sistema levou em consideração as características do terminal, como: pouca memória, rápida obsolescência e uso iterativo, bem como a diversificação do tipo de uso.

Baseado nestes fatos, desenvolveu-se um Sistema Operacional funcionando em monoprogramação, e postulou-se:

(i) Prioridade para economia de memória, em detrimento da velocidade de processamento. Isto é possí

vel pois as aplicações serão interativas, e portanto com pouco processamento.

(ii) Escrever o próprio sistema operacional em uma linguagem de alto nível, de modo que ele seja independente da unidade central de processamento.

(iii) Ser o mais simples possível, pois as pessoas que processarão as aplicações não serão especializadas.

(iv) Não impor restrições quanto a técnicas sofisticadas, pois os usuários que utilizarem o sistema para desenvolvimento irão necessitar destas técnicas.

(v) Gerar código interpretável, em função do item (i) e dos recursos limitados do microprocessador utilizado.

(vi) Dentro do possível ser compatível com o Sistema Cruzado já em funcionamento.

1.2.2 Descrição

Podemos dividir o Sistema Operacional em Disco nos seguintes módulos:

- Compiladores
- Editor de Referências Externas
- Carregador de Programas
- Núcleo Residente

- Rotinas de Apoio

  Rotinas de E/S

  Intrínsecos

- Utilitários

Os módulos serão escritos, tanto quanto possível, em uma linguagem de alto nível especialmente desenvolvida (PLTI).

O sistema utilizará uma memória do tipo ROM ("Read Only Memory") para ativar o sistema operacional.

Ao ser apertada no painel do Terminal Inteligente a tecla "CARGA" será provocado um desvio para o programa em ROM. Este programa por sua vez trará para memória o programa de inicialização do sistema (programa de partida fria - PPF).

Existe também no painel a tecla ERRO, que ao ser apertada provocará uma interrupção, isto é, um desvio para rotina EXIT do núcleo residente, permitindo portanto que programas sejam cancelados.

A figura 3 mostra o esquema de funcionamento.

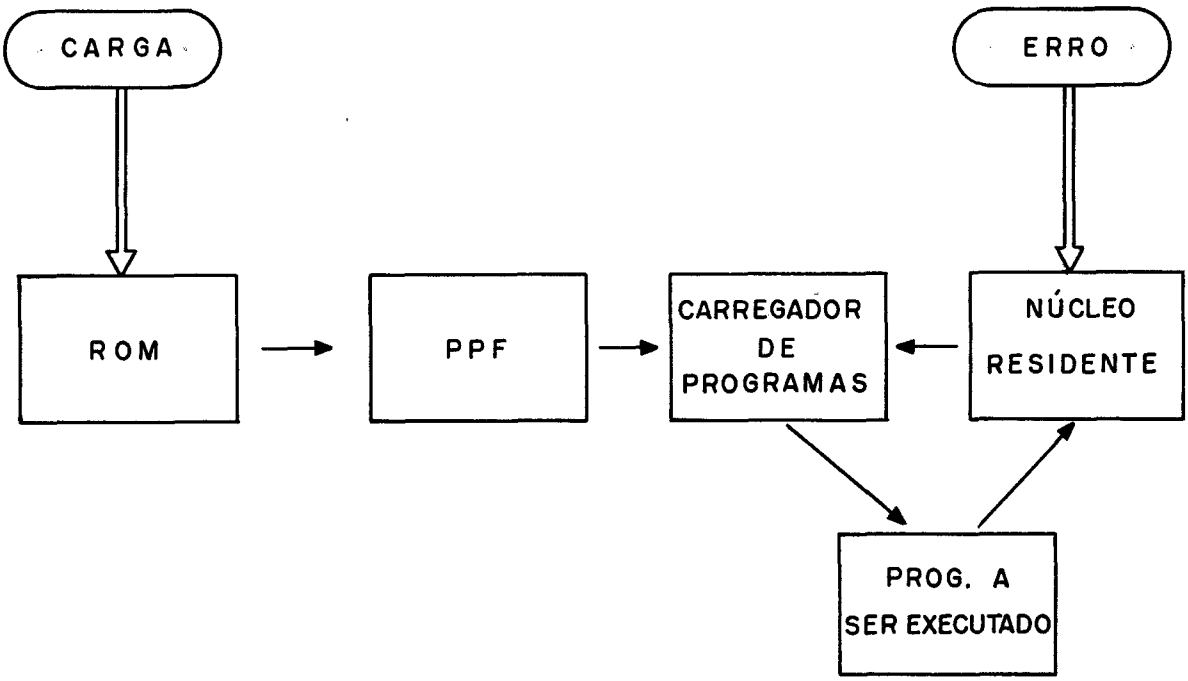

CARGA
ERRO
ROM
PPF
CARREGADOR DE PROGRAMAS
NÚCLEO RESIDENTE
PROG. A SER EXECUTADO

Fig. 3

Ao ser apertada a tecla CARGA é provocado um desvio para a ROM que busca no disco o programa de inicialização do sistema. Este programa inicializa as partes residentes e chama o Carregador de Programas (CPROG). O CPROG por sua vez traz para memória o programa a ser executado, bem como todas as rotinas de entrada/saída utilizadas, e transfere o controle ao programa. O programa ao executar o comando EXIT, que corresponde a retornar ao núcleo residente na parte referente à chamada do CPROG para que o processo se repita.

É importante observar que programas como o REFEX, CPROG e os compiladores são tratados como programas comuns.

1.3 Editor de Referências Externas

O Editor de Referências Externas desenvolvido para o S.O.Co, é um dos programas básicos do Sistema Operacional. Sua função é unir módulos que tenham sido criados por montadores e compiladores incorporados ao sistema, gerando módulos absolutos prontos para serem carregados.

Em função das características de todo o sistema, o REFEX procura ser o mais flexível possível, permitindo a definição de estruturas de "overlay" pelo próprio usuário.

O presente trabalho tratará, em detalhes, o seu funcionamento.

2º CAPÍTULO

Como já foi dito anteriormente o REFEX é parte integrante do sistema S.O.Co e portanto, os seus arquivos de entrada/saída devem estar compatíveis com os demais módulos do sistema.

2.1 Comandos de Controle

O REFEX recebe como informações de controle o nome do programa a ser editado; a estrutura de overlay utilizada, quando necessária e algumas opções de edição.

Todos estes comandos estão organizados de modo a garantir uma das premissas básicas do sistema que é a de oferecer um uso o mais simples possível. Portanto os comandos oferecidos também possuem um grau de dificuldade em função do tipo de edição solicitada, isto é, para uma edição simples, um comando também simples.

A estrutura de overlay utilizada deve ser fornecida ao REFEX em notação de parênteses, obedecendo a sintaxe fornecida na figura 4 e 5. Observe que esta notação representa claramente a estrutura em árvore adotada pelo usuário para a construção dos módulos que atenda as suas necessidades.

Para uma melhor visualização, apresentaremos alguns exemplos de comandos de controle.

(i) PROG.

(ii) PROG [VOLØ1].

(iii) APLIC<ROTINA>.

COMANDO DE CONTROLE DO REFEX

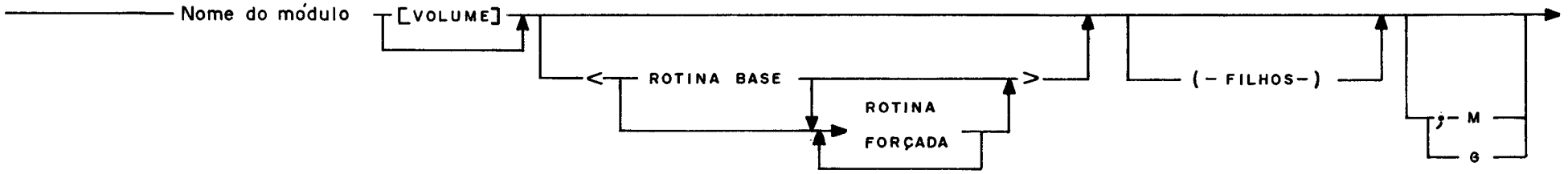
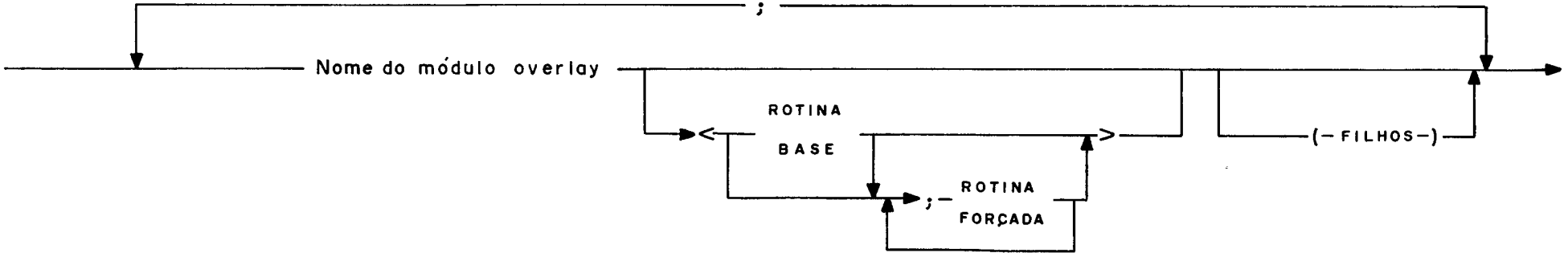


Fig. 4

FILHOS



DESCRIÇÃO DE ELEMENTOS

Citados na figura 4

Nome de Módulo	É o nome com que será reconhecido o módulo depois de editado. No caso de FILHO este módulo será do tipo OVERLAY.
Nome do Volume	Indica em que volume serão criados os arquivos de saída (módulos).
Rotina Base	Nome da rotina que será utilizada como base para resolução das Referências Externas. No caso de ser omitido o nome da rotina base o REFEX assume que é igual ao nome do módulo.
Rotina Forçada	Nome de uma rotina que fará parte do módulo mesmo que não seja referenciada por nenhuma rotina componente.
Filhos	Nome de módulos "OVERLAY's" utilizados, sendo válido em vários níveis.
M	Implica que o usuário ativou a opção MAPA e será fornecido um mapa de alocação de memória.
G	Implica que o usuário ativou a opção GERA e os módulos serão gerados mesmo que exista uma referência não-resolvida.

Fig. 5

(iv) PROG;M.

(v) A(B,C(F,G),D);MG.

(vi) MODA<A,A1>(MODB<B,F1,F2>,MODC,MODD).

Observe que as ordens ao REFEX podem ser simples, como no caso (i) quando se pede para editar um programa sem overlay e que tem o mesmo nome tanto em módulo objeto ou de carga, ou complexas como nos casos (v) e (vi) em que são definidas estruturas de overlay e algumas rotinas são forçadas em determinados módulos.

O REFEX dispõe de duas formas para receber os comandos de controle.

2.1.1 Através de Parâmetros

O REFEX é carregado na memória através do CPROG como um programa comum e portanto admite que o campo de parâmetro seja utilizado para transmissão do comando de controle para o REFEX (programa que será carregado e executado). A figura 6 mostra o comando de controle analisado pelo CPROG, onde:

(i) nome - nome do programa a ser preparado para execução ou geração de formato imagem.

(ii) volume - identificação do dispositivo de Entrada/Saída que contém o arquivo referenciado.

(iii) parâmetro - cadeia de caracteres a ser passado ao programa que será executado.

Quando a cadeia contiver apóstrofo este deverá ser o primeiro caracter da cadeia.

(iv) arquivo interno (1) - nome pelo qual o programa referencia os arquivos nos comandos de Entrada/Saída

externo - nome externo do arquivo

(v) Periférico - nome da rotina de Entrada/saída a ser utilizada no tratamento do arquivo físico.

(vi) nome imagem - nome do módulo imagem, para caso de geração (2)

Um exemplo desta forma de executar o REFEX é: REFEX 'PROG;M.'.

Onde o programa PROG deve ser editado e o REFEX deverá emitir o Mapa de Alocação.

2.1.2 Através de um Arquivo

Outra forma do REFEX receber o comando de controle é através de um arquivo inicialmente definido como DUMMY, que pode ser redefinido, conforme o interesse do usuário, durante a execução do CPROG. Por exemplo: REFEX (DUMMY=:CARTAO). indicando que o comando a ser analisado pelo REFEX, encontra-se em um arquivo em cartões.

(1) No caso do REFEX serve para se fazer "override"
(2) No caso do REFEX não é necessário.

COMANDO DE CONTRÔLE DA EXECUÇÃO

(Analizado pelo CPROG)

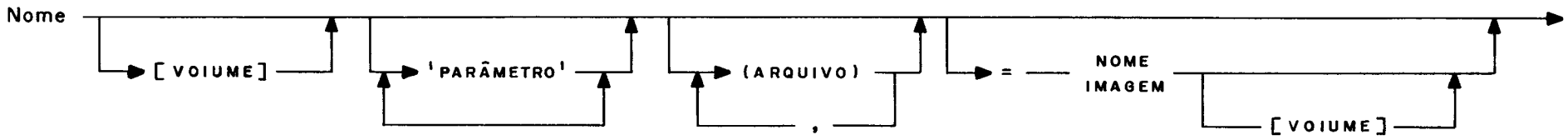
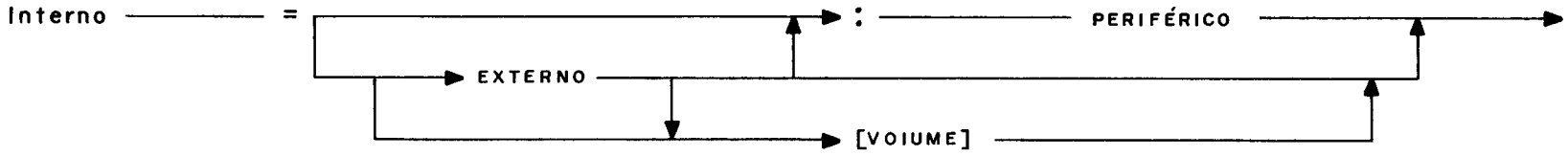


Fig. 6

Onde:

Arquivo:



Parâmetro:

Cadeia de caracteres onde só o primeiro pode ser apostrafo (').

2.2 Módulo Objeto

Os montadores e compiladores do sistema S.O.Co criam arquivos, que denominamos módulos objetos obedecendo a um formato bem definido.

O conteúdo de um módulo objeto é o seguinte:

(i) Tabela de Arquivos Lógicos (TAL)

Fornece dados sobre os arquivos definidos pelo programa.

(ii) Tabela de Símbolos Externos (TSE)

Esta tabela contém todas as referências externas ao programa, bem como possíveis entradas que possam ser referenciadas por outros programas.

(iii) Tabela de Referência ao Interpretador (TRI)

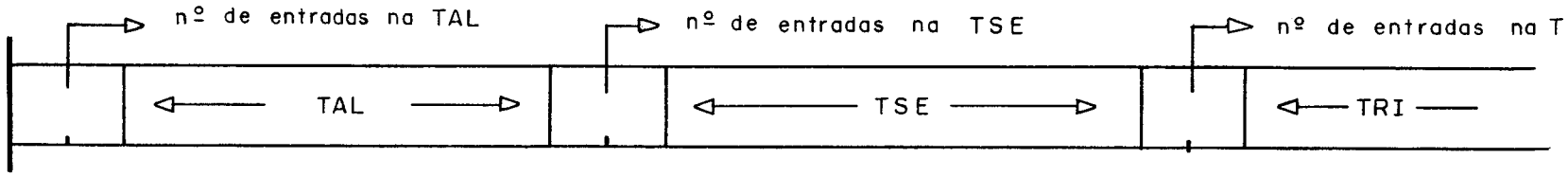
O compilador PLTI, que é um dos módulos do S.O.Co, gera código interpretável e através desta tabela indica um procedimento especial a ser adotado pelo REFEX para que as rotinas de seu interpretador possam ser incorporadas ao módulo resultante da edição.

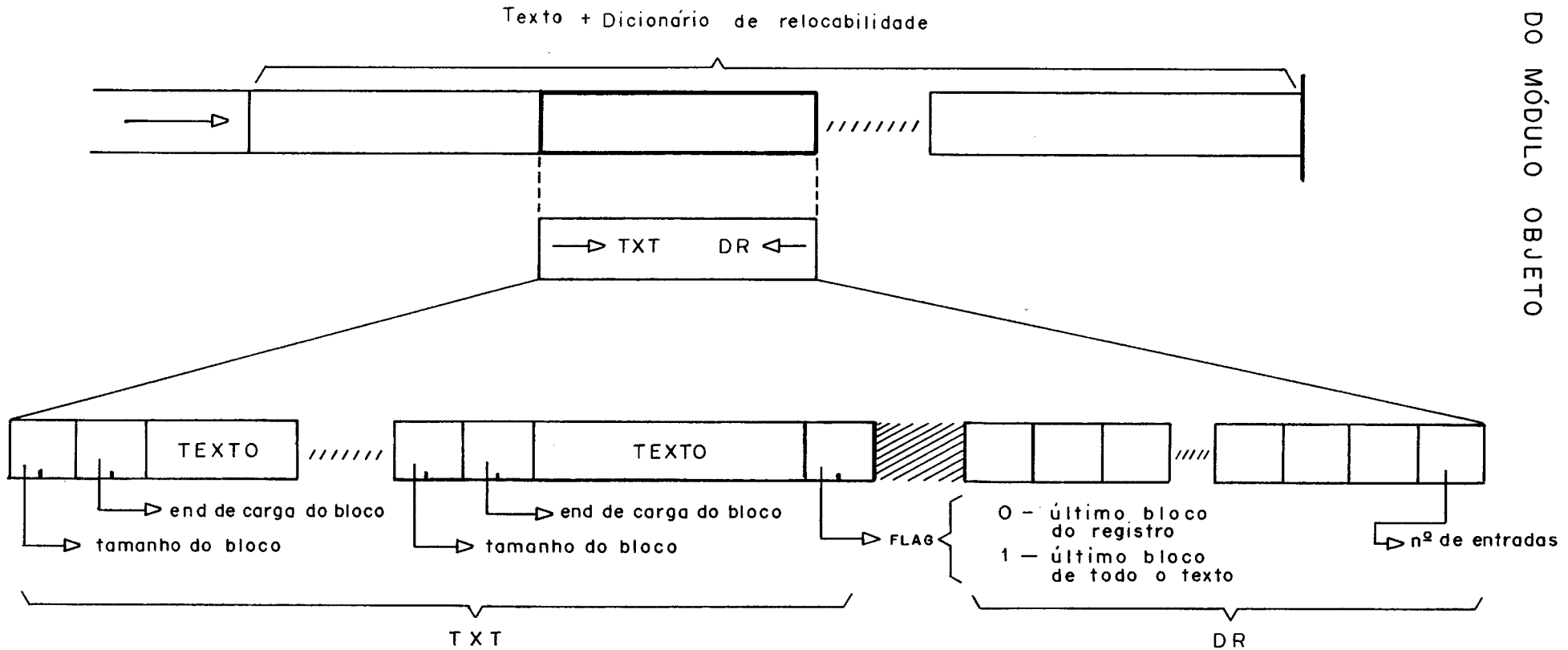
(iv) Dicionário de Relocabilidade e Texto (DR e TXT)

O dicionário de relocabilidade indica quais os endereços do código gerado que dependem de relocação e o texto corresponde ao código gerado por montadores e compiladores do sistema. O TXT e o DR são montados em registros fixos. Dentro de cada registro são colocados o texto e as entradas do DR correspondente ao texto.

A figura 7 apresenta o esquema de um módulo objeto.

O processo de relocação fica simplificado porque o texto e os endereços das posições a serem relocadas, bem





FORMATO DO MÓDULO OBJETO

Fig. 7

como o tipo de relocação (DR) ficam no mesmo registro. É utilizado um esquema de pilha dupla de modo que haja o máximo de aproveitamento sem a necessidade de áreas independentes para texto e dicionário de relocabilidade.

2.3 Módulo de Carga

O REFEX gera dois tipos de módulo de carga: um que denominamos módulo REFEX e outro que chamamos de módulo OVERLAY.

O módulo REFEX é o módulo raiz ou principal e será carregado pelo CPROG, já o módulo OVERLAY corresponde aos vários descendentes da estrutura de overlay definida e será carregado sob o controle do usuário, através da chamada de uma rotina auxiliar do sistema (rotina de LOAD)

2.3.1 Módulo de Carga Tipo REFEX

O conteúdo do módulo REFEX é o seguinte:

(i) Texto

Contém todo o código a ser executado, devidamente relocado, exceto em relação às rotinas de Entrada/Saída que serão incluidas pelo CPROG.

(ii) Tabela de Arquivos Lógicos (TAL)

Fornece dados sobre os arquivos definidos pelo programa. Observe que esta tabela é imagem da fornecida no módulo objeto e será analisada pelo CPROG para permitir um recurso do S.O.Co., que é o de admitir em tempo de execução do programa a redefinição dos arquivos utilizados, bem como do meio em que se encontram. Por exemplo: um arquivo de saída definido inicialmente com o periférico de

saída impressora, poderá, para efeito de testes, ser redefinido como vídeo durante uma determinada execução.

A figura 8.a mostra o esquema do módulo de carga tipo REFEX.

2.3.2 Módulo de Carga Tipo OVERLAY

O conteúdo do módulo OVERLAY é o seguinte:

(i) Tabela de Acesso

Quando o código gerado se refere a programas em PLTI, como o seu código é interpretável, é necessário que ao se carregar um novo módulo na memória seja atualizada a tabela de acesso direto utilizada pelo núcleo do interpretador, pois é possível que o módulo carregado utilize rotinas do interpretador que ainda não tenha sido utilizadas em um ancestral e que portanto constará do módulo recentemente carregado.

(ii) Texto

Contém todo o código a ser executado em formato imagem de memória. No caso de módulos de módulos de carga tipo overlay não existe o problema de rotinas de Entrada/Saída, pois foram resolvidos pelo CPROG quando do carregamento do módulo raiz.

A figura 8.b mostra o esquema do módulo de carga tipo overlay.

MÓDULO DE CARGA

(a) MÓDULO REFEX

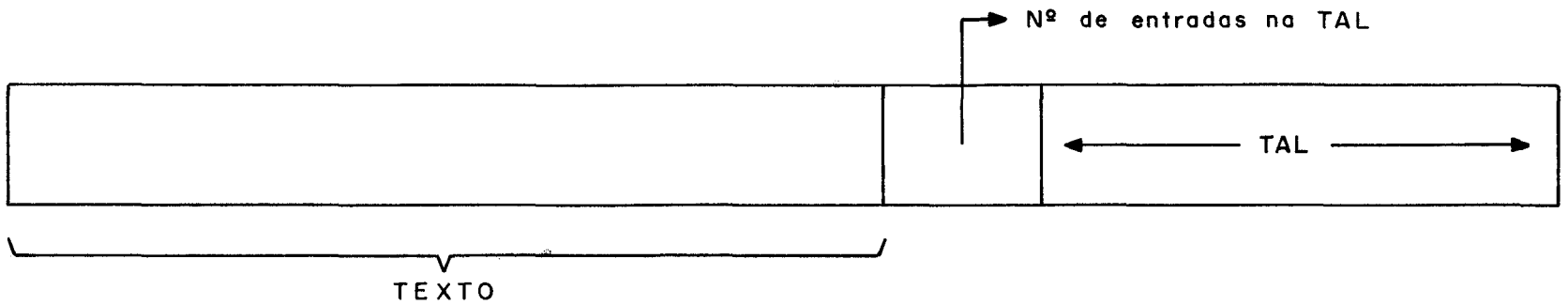
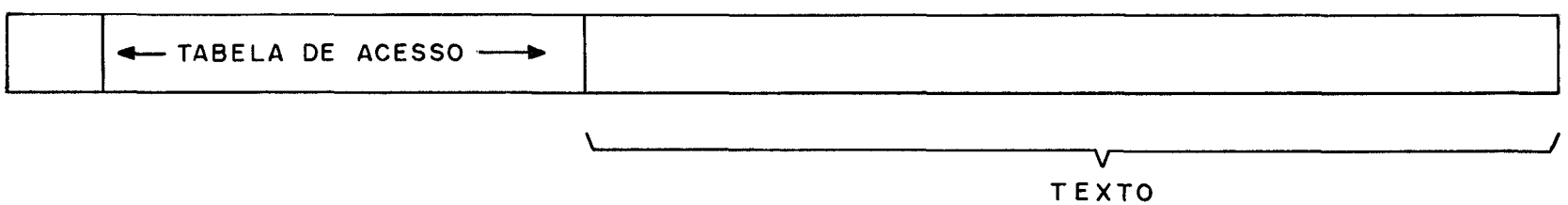


Fig. 8

(b) MÓDULO OVERLAY



3º CAPÍTULO

O REFEX é um programa de serviço do S.O.Co., com funções bem determinadas.

3.1 Descrição Geral

A edição consiste em unir módulos objetos gerados por montadores e compiladores. O REFEX combina os módulos de acordo com os requerimentos definidos pelos comandos de controle, gerando um único módulo já relocado e pronto para execução, exceto em relação as rotinas de Entrada/Saída. Cada módulo objeto processado pelo REFEX teve origem em uma montagem ou durante uma compilação e em cada um destes módulos existem referências a outros módulos. Estas referências são chamadas de Referências Externas, e são resolvidas pelo REFEX.

3.2 Edição no S.O.Co

Como já foi descrito anteriormente o S.O.Co é um sistema operacional para microprocessadores e portanto a definição de um Editor de Referências Externas para este sistema deverá levar em consideração as características do micro-computador utilizado. Portanto um dos pontos básicos a ser considerado é o de restrição de memória, o que implica na necessidade de incorporar recursos de overlay ao REFEX para garantir uma maior flexibilidade no desenvolvimento de aplicações.

3.3 Estrutura

O sistema REFEX está dividido em fases e cada uma destas fases age sobre uma estrutura de dados bem definida que permite um controle rígido e de fácil compreensão.

A escolha deste esquema de funcionamento permite que possamos descrever o sistema REFEX, simplesmente pelo algoritmo que se segue:

1. Montar árvore
2. Analisar indefinidos
3. Criticar árvore e listas
4. Determinar bases e endereços de execução dos módulos
5. Montar módulos fazendo relocação
6. Fornecer mapa de alocação e dados complementares, quando necessário.

Observe que cada item do algoritmo corresponde a uma parte do sistema com tarefa bem determinada e independente, exceto em relação a ordem de execução.

Todas as fases por sua vez se utilizam de uma estrutura de dados especialmente projetada para intercambiar informações no decorrer do processo de edição. Cada fase age somente sobre esta estrutura de dados não existindo nenhuma outra comunicação entre elas, portanto garantindo a modularidade do sistema.

A estrutura de dados utilizada é baseada em uma árvore binária (de Knuth) construída a partir das informações de controle fornecidas pelo usuário. Cada nó desta árvore funciona como cabeça de uma lista de apontadores correspondente aos elementos do módulo.

A figura 9 apresenta um esquema de uma possível estrutura de dados criada e tratada pelo REFEX. Cada nó da árvore correspondente a um módulo overlay e possui apontadores para o filho esquerdo, pai, irmão, além de um apontador para uma lista encadeada com referências externas correspondentes ao módulo.

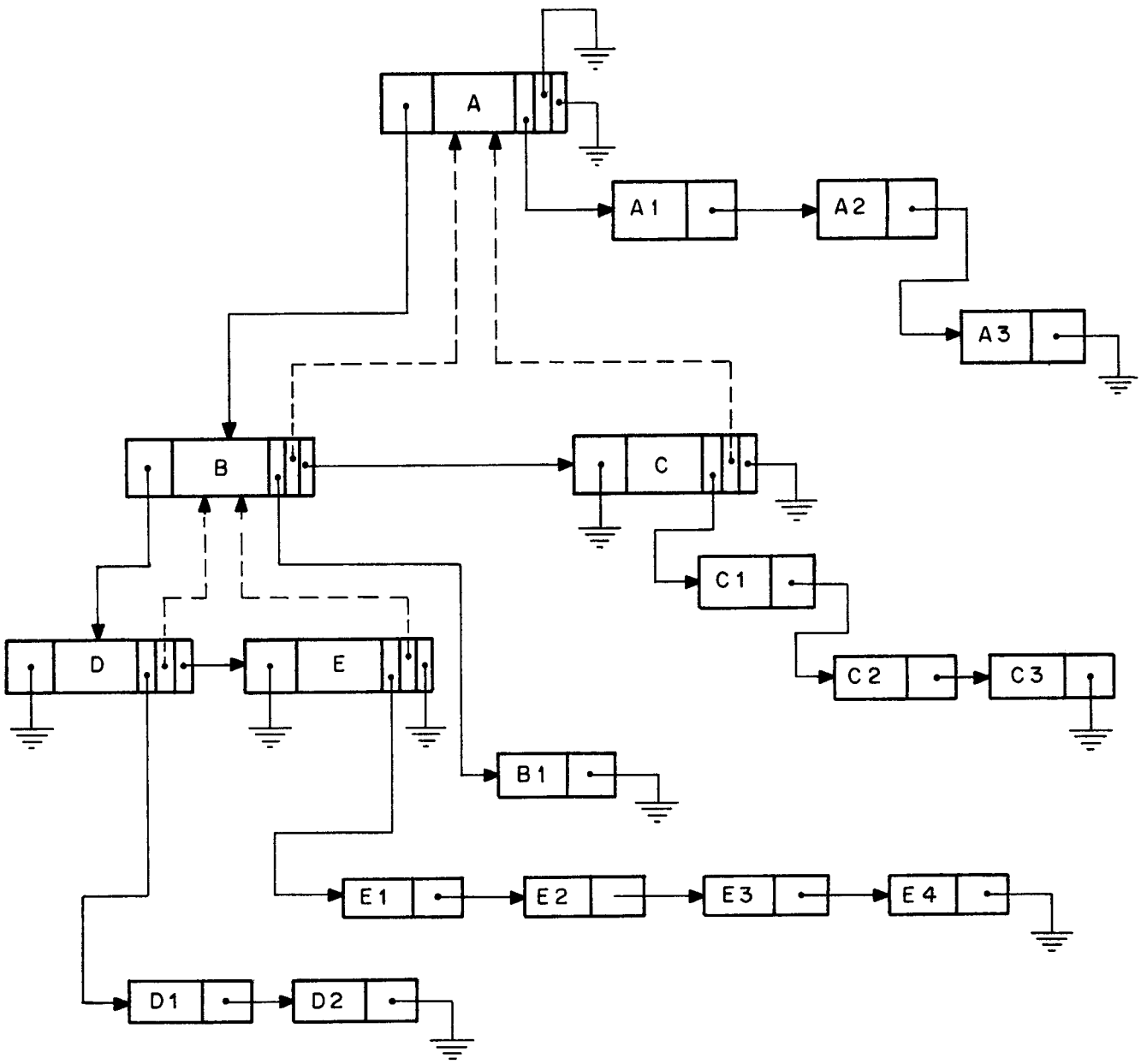


Fig. 9

3.4 As Fases

(i) Montagem da Árvore

Em função da estrutura de overlay definida pelo usuário, os módulos são agrupados em pré-ordem na Área Livre (1) do S.O.Co.

(ii) Análise da Estrutura

A fase de análise da estrutura constitui a principal parte do Editor de Referências Externas. Ela é responsável pela resolução, propriamente dita, de todas as referências não resolvidas.

A estrutura de dados é analisada procurando resolver referências externas e gerando novas informações quando necessário.

(iii) Crítica a Árvore e Listas

Nesta fase a estrutura é percorrida procurando criticar os dados para detetar incongruências.

(iv) Determinações de Bases

Depois da estrutura devidamente construída, ela é novamente percorrida sendo atualizados os endereços de carga e de execução de cada elemento em função dos dados obtidos na fase de análise.

(v) Criação dos Módulos

Esta fase é responsável pela geração em disco dos módulos de carga. Novamente a estrutura é percorrida e os elementos são aglutinados e devidamente relocados.

(1) Denominamos Área Livre à memória disponível entre o fim do programa que está sendo executado e as tabelas do Sistema Operacional que ficam no fim da memória.

(vi) Relatórios

A última fase fornece informações pertinentes à edição como, por exemplo, mapa de alocação.

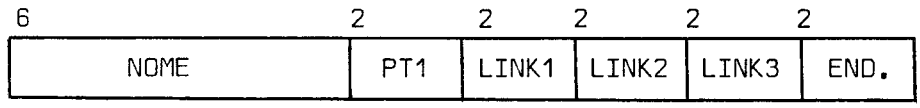
4º CAPÍTULO

4.1 Estrutura de Dados Utilizada

Como foi visto na figura 9, a estrutura de dados utilizada pelo REFEX baseia-se em uma árvore em que cada nó corresponde a um módulo da estrutura de overlay utilizada e que aponta para uma lista encadeada, onde são representados todos os símbolos (pontos de entrada e referencias externas) de um determinado módulo.

4.1.1 Árvore

A árvore é constituida de nós, e cada nó ocupa 16 bytes e obedecendo ao seguinte formato:



Sendo:

Nome - o símbolo correspondente ao nome do módulo

PT1 - ponteiro para a lista encadeada com as referencias (ponto de entrada e referencias externas) do módulo.

LINK1 - ponteiro para o módulo que na árvore ocupa o nó correspondente ao filho mais à esquerda.

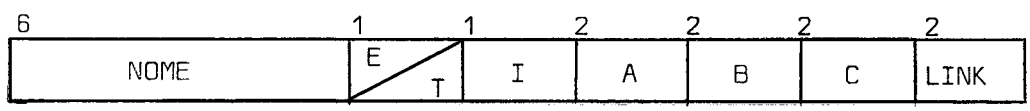
LINK2 - ponteiro para pai do nó.

LINK3 - ponteiro para o irmão à direita adjacente.

END. - endereço de carga do módulo.

4.1.2 Lista

As listas também são constituidas de nós, com tamanho de 16 bytes e obedecendo ao formato abaixo.



Sendo:

Nome - Nome do símbolo referenciado como ponto de entrada ou referência externa em uma tabela de símbolos externos de um determinado módulo.

E - Estado (bit 7 do campo), indicador se a referência já foi resolvida ou não, quando:

E = 0, indica que a referência está indefinida.

E = 1, indica que a referência está definida.

T - Tipo (bits 6-0 campo), indicadores auxiliares para controle da edição, quando tipo:

= 0 indeterminado

uma rotina referencia um símbolo que não consta do módulo objeto tratado.

= 1 arquivo

o símbolo referenciado é um arquivo.

= 2 ponto de entrada secundário

são denominados secundários, todos os pontos de entrada de uma rotina, exceto o primeiro.

= 3 ponto de entrada principal

é o primeiro ponto de entrada de uma rotina.

= 4 nome de rotina BASE

rotina base é a rotina central de um módulo. Todas as referências de um ancestral a um módulo overlay são feitas para os pontos de entrada da rotina BASE.

= 5 nome de rotina FORÇADA

rotina incluida em um módulo por comando explícito do usuário.

= 6 ponto de entrada privilegiado

são os pontos de entrada secundários de uma rotina base.

= 7 variável

são símbolos que no PLTI, foram indicados como variáveis globais.

= 8 especiais

são símbolos automaticamente inseridos na edição. Por exigência da estrutura do sistema S.O.Co, por exemplo podemos citar o Núcleo do Interpretador.

I - indica que o símbolo corresponde a uma rotina do interpretador PLTI, se diferente de zero é rotina do interpretador, em caso contrário o conteúdo é o número que identifica a rotina.

A,B,C- os campos A,B e C tem conteúdos diversos em função do tipo, a tabela abaixo relaciona os tipos com os respectivos conteúdos A,B e C.

TIPO	A	B	C
3,4,5,8	Endereço de carga ou Tamanho da rotina	Endereço de execução	Endereço no Disco
2,6,7	Endereço de carga ou ponteiro para rotina a que pertence	Endereço de Execução	Endereço no Disco
1	-	Endereço da entrada da TRAC correspondente	-

Para os tipos 2,3,4,5,6 e 8 o conteúdo de A é mútuamente exclusivo e dependente da fase de edição que está sendo processada.

Para o tipo 1 o endereço da TRAL é relativo, ou seja, relativo ao fim da área ocupada pelo maior ramo da árvore.

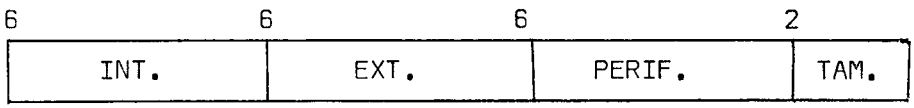
LINK - apontador para o próximo elemento da lista.

4.2 Outras Tabelas

O REFEX consulta algumas tabelas que são constituidas pelos compiladores e montadores e fazem parte integrante do módulo objeto.

4.2.1 Tabela de Arquivos Lógicos (TAL)

A TAL contém a definição de todos os arquivos definidos em um programa. Cada entrada nesta tabela ocupa 20 bytes e obedece ao seguinte formato:



Sendo:

INT. - nome do arquivo dentro do programa

EXT. - nome externo do arquivo

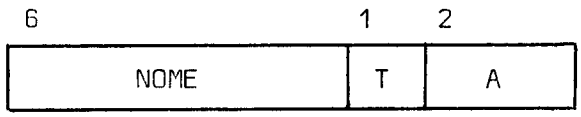
PERIF. - nome do periférico que contém o arquivo

TAM. - tamanho do registro lógico do arquivo

Durante a edição somente a TAL do programa principal é levada em consideração.

4.2.2 Tabela de Símbolos Externos (TSE)

A TSE contém informações de todas as referências de uma rotina (pontos de entrada e referências externas). Cada entrada nesta tabela ocupa 9 bytes e obedece ao seguinte formato:



Sendo:

Nome - nome do símbolo referenciado

T - tipo de referência

= 0, para referências externas

= 1, para referência a arquivos

= 2, para pontos de entrada

= 3, para declarações de variáveis globais

= 4, para referência a variáveis globais

A - o campo A tem um conteúdo que depende do tipo, conforme a tabela abaixo:

TIPO	A
0	-
1	-
2	Endereço de Execução
3	Endereço de Variável
4	-

4.2.3 Tabela de Referência ao Interpretador (TRI)

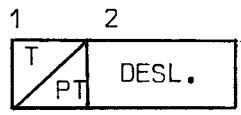
A tabela de referência ao interpretador é uma relação de todas as rotinas do interpretador que um determinado módulo ob

jeto utiliza. Cada entrada nesta tabela ocupa apenas 1 byte e o conteúdo de cada entrada é o código da rotina interpretadora utilizada.

4.2.4 Dicionário de Relocabilidade (DR)

O DR contém informações sobre todos os endereços relocáveis do texto.

Cada entrada ocupa 3 bytes e obedece ao seguinte formato:



Sendo:

T - tipo de relocação (os dois primeiros bits do primeiro byte)

= 00 CAL e JMP

= 01 LRS

= 10 DC

= 11 RESERVADO

As inscrições citadas são do Assembler do microcomputador utilizado que possuem forma de relocação distinta entre si. A instrução RST não foi considerada, pois para elas serão geradas códigos absolutos.

PT - ponteiro para TSE que indica a base de relocação (bits 0 a 5 do campo)

DESL - deslocamento dentro do texto em relação ao início do módulo.

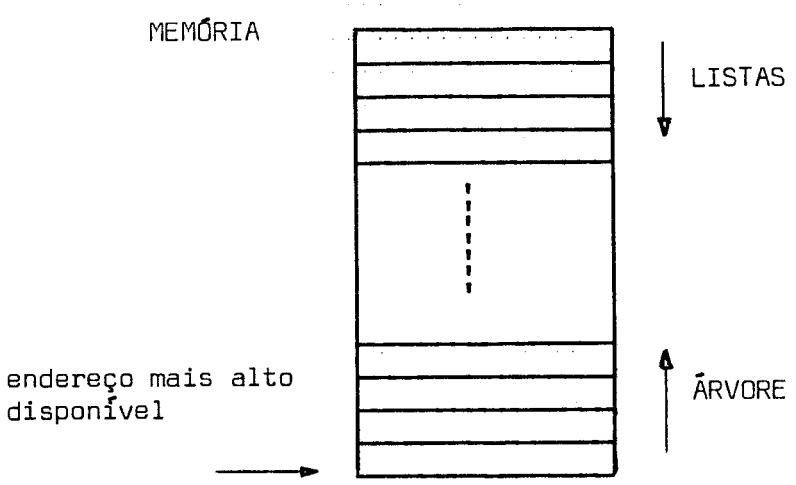
5º CAPÍTULO

Neste capítulo detalharemos as fases do REFEX.

5.1 Montagem da Árvore

A estrutura em árvore adotada é fornecida através de cartões de controle com os módulos grupados em pré-ordem. É função desta fase inicializar a área utilizada para árvore e para as listas correspondentes.

A estrutura é alocada na Área Livre do sistema obedecendo a seguinte disposição:



A Área Livre é dividida em segmentos de 16 bytes, que correspondem ao tamanho dos nós da árvore e das listas.

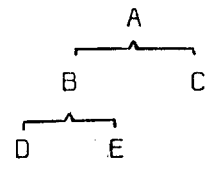
Quando se deseja ocupar um nó da árvore é alocado o segmento disponível de maior endereço, já quando se deseja ampliar uma das listas é alocado o segmento disponível de menor endereço.

Analisando o esquema pode-se observar que a Área Livre é considerada uma pilha dupla onde os nós das listas são alocados de cima para baixo e os nós da árvore de baixo para cima.

Para termos uma ideia precisa do processo de inicialização, analizaremos o seguinte exemplo:

A<AB, F1>(B (D<DB>, E), C<, F2>)

O exemplo corresponde a seguinte estrutura de edição.



Rotina Base	Rotina Forçada
AB	F1
B	
C	F2
DB	
E	

Na figura 10, mostramos a estrutura ao término da fase de inicialização, bem como os dados alocados na Área Livre.

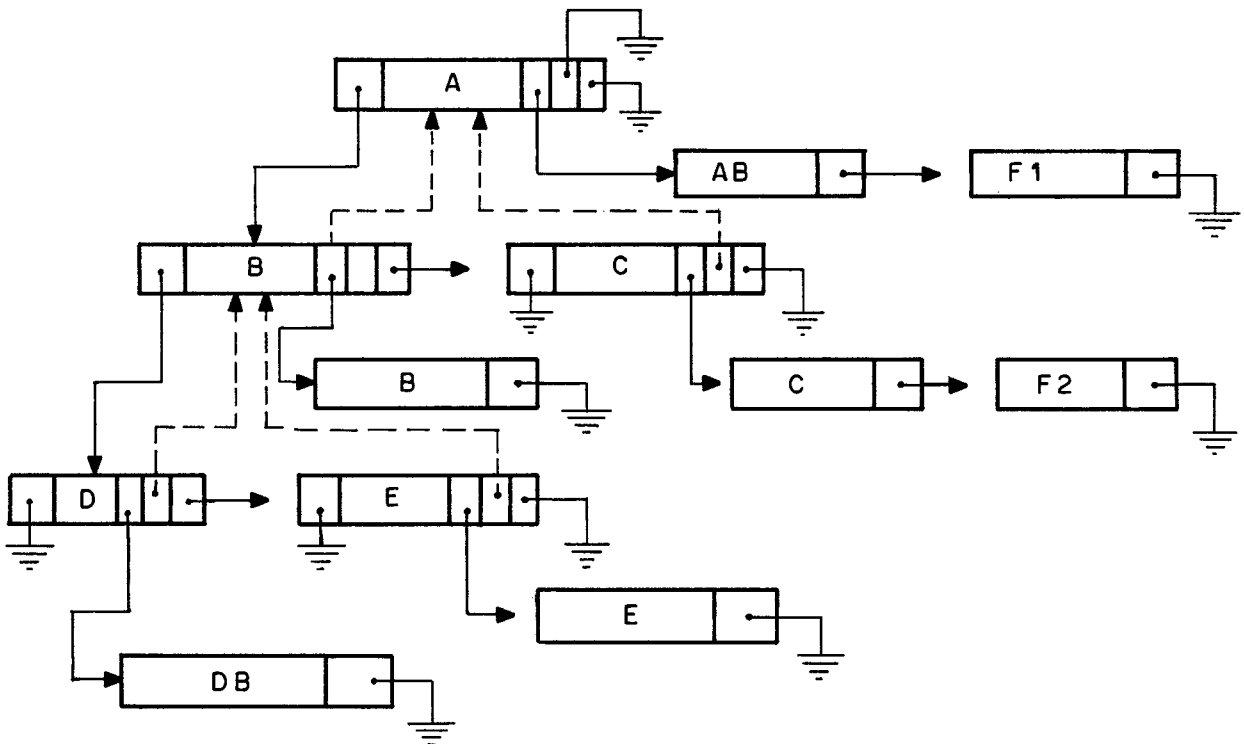
5.2 Análise da Estrutura

Esta é a principal fase do editor, pois nesta fase todas as referencias são efetivamente resolvidas.

A árvore de módulos é atravessada em pré-ordem e para cada nó a sua lista encadeada correspondente é percorrida. Para cada nó, das listas, marcado como indefinido é feita uma busca nos diretórios dos discos à procura do mesmo. Quando um módulo indefinido é encontrado a sua TSE é percorrida sendo as listas devidamente atualizadas.

Para cada entrada de uma TSE analisada é verificado se o símbolo indicado consta: da lista que está sendo percorrida, da lista de algum ancestral, ou pertence a lista de um filho direto com indicação de ponto de entrada previlegiado ou rotina base.

Considerando o exemplo cuja inicialização está detalhada na figura 10, podemos observar que o tratamento dado a rotina AB, que está



End. Hex.	NOME	E/T	I	A	B	C	LINK
1010	AB	0/4					1020
1020	F1	0/5					↑
1030	B	0/4					↑
1040	DB	0/4					↑
1050	E	0/4					↑
1060	C	0/4					1070
1070	F2	0/5					↑
1080							
1090							
10A0							
10B0							
10C0							
10D0							
10E0							
10F0							
1100							
1110							
1120							
1130							
1140							
1150	C	1060		↑	1190	↑	
1160	E	1050		↑	1180	↑	
1170	D	1040		↑	1180	1160	
1180	B	1030		1170	1190	1150	
1190	A	1010		1180	↑	↑	
	NOME	PT1		LINK1	LINK2	LINK3	END.

Fig. 10

marcada como indefinida na fase de análise é o seguinte: a rotina AB é procurada no disco e quando é encontrada a sua TSE é percorrida. Observe que a TSE não precisa ser totalmente montada na memória, já que ela será tratada sequencialmente, portanto a própria área de Entrada/Saída poderá ser utilizada,

A figura 11 mostra as alterações provocadas na estrutura pela análise da TSE de AB.

Comparando a estrutura após a fase de análise da rotina AB (figura 11) com a figura antes da análise (figura 10), podemos observar:

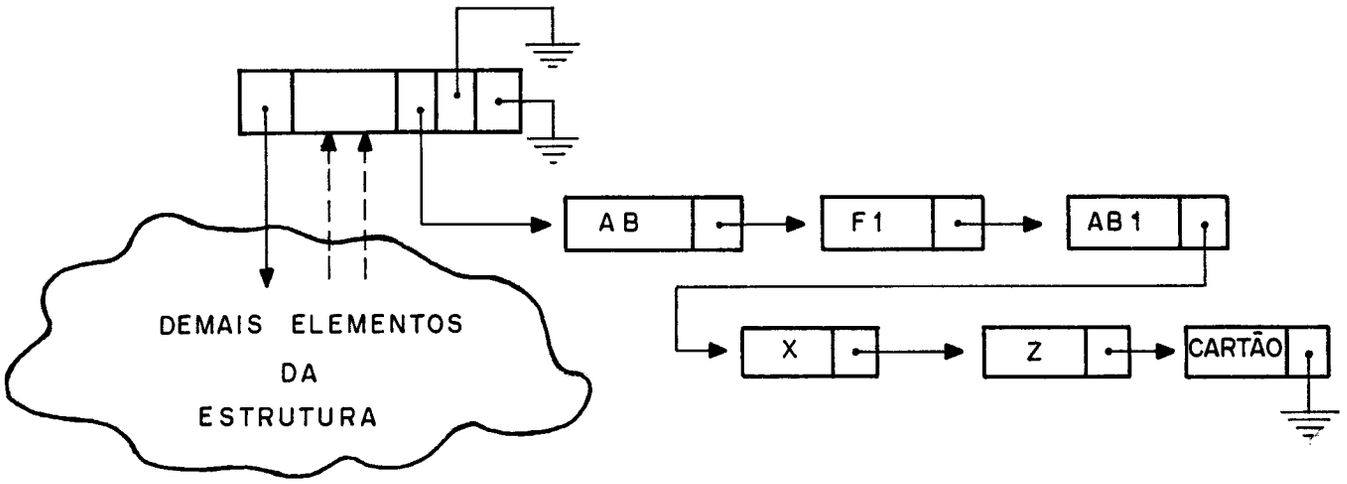
(i) que os símbolos ao serem colocados nas listas são marcados conforme o tipo que os gerou, no exemplo o símbolo AB1 foi marcado como tipo 6 (ponto de entrada previlegiado) porque pertencia a TSE de um símbolo tipo 4 (rotina BASE) e portanto todos os seus pontos de entrada são previlegiados.

(ii) que o conceito de ponto de entrada principal e secundário é utilizado para que uma mesma rotina não seja carregada duas vezes. O ponto de entrada principal implicará na leitura da TSE da rotina, os demais serão denominados pontos de entrada secundários.

(iii) que as listas são formadas a partir da TSE, porém não existe uma relação explícita entre a lista e a TSE porque todas as entradas de uma TSE não ficam na mesma lista. Portanto na fase de relocação será necessário acessar novamente a TSE, de tal modo que os símbolos que constem da mesma sejam localizados em várias listas.

TSE DE AB

AB	2		→ PONTO DE ENTRADA
AB1	2		→ PONTO DE ENTRADA
X	0		→ REFERÊNCIA EXTERNA
B	0		→ REFERÊNCIA EXTERNA
Z	0		→ REFERÊNCIA EXTERNA
CARTÃO	1		→ ARQUIVO



AS ALTERAÇÕES NA ALOCAÇÃO DE ÁREA LIVRE SERIAM:

	NOME	E/T	I	A	B	C	LINK
1010	AB	1/4					1020
1020	F1	0/5					1080
1030	B	0/4					^
1040	DB	0/4					^
1050	E	0/4					^
1060	C	0/4					1070
1070	F2	0/5					^
1080	AB1	1/6					1090
1090	X	0/0					10A0
10A0	Z	0/0					10B0
10B0	CARTÃO	1/1					^
10C0							
10D0							
10E0							
10F0							
1100							
1110							
1120							
1130							
1140							
1150							
1160							
1170							

Fig. 11

(iv) se um símbolo está na TSE e não está nas listas, ele será inserido. Se ele já consta de alguma lista é feita uma análise que é descrita no item 5.2.1 para analisar a ação a ser tomada.

5.2.1 Relações entre TIPOS

Uma das funções básicas da fase de análise consiste em determinar qual a ação a ser tomada em função do tipo que se encontra na TSE e o tipo que conste na estrutura.

Para facilitar este controle temos as seguintes tabelas:

a) Quando na estrutura consta como DEFINIDO

TIPO NA TSE \ TIPO NA ESTRUTURA	0	1	2	3	4	5	6	7	8
REF. EXTERNA (0)	☒		■	■	■	■	■	■	■
ARQUIVO (1)	☒	■							
PONTO DE ENTRADA-(02)	☒							■	
VAR. DECLARAÇÃO (3)	☒	☒	☒	☒	☒	☒	☒		☒
VAR. REFERENCIA (4)	☒	☒	☒	☒	☒	☒	☒	■	☒

CONVENÇÃO:

☐ A RELAÇÃO PROVOCA ERRO

☒ NÃO EXISTE A RELAÇÃO

■ EXISTE AÇÃO

Ø - indeterminado
1 - arquivo
2 - ponto de entrada secundário
3 - ponto de entrada principal
4 - rotina BASE
5 - rotina FORÇADA
6 - ponto de entrada previlegiado
7 - variável
8 - nós especiais

DESCRIÇÃO DOS ERROS E AÇÕES

TSE	EST.	DESCRIÇÃO
0	1	ERRO-referência externa definida como Arquivo
	2	ignora
	3	
	4	
	5	
	6	
1	1	ignora
	2	ERRO-arquivo não foi definido globalmente
	3	
	4	
	5	
	6	
	7	
	8	
2	1	ERRO-ponto de entrada definido como Arquivo
	2	ERRO-ponto de entrada duplicado
	3	
	4	
	5	
	6	
	7	incluir como ponto de entrada e definido
	8	ERRO-ponto de entrada com nome reservado
3	7	ERRO-variável duplicada
4	7	ignora

b) Quando na estrutura consta como INDEFINIDO

TIPO NA TSE \ TIPO NA ESTRUTURA	Ø	1	2	3	4	5	6	7	8
REFERÊNCIA EXTERNA (0)	■		■	■	■	■	■	☒	☒
ARQUIVO (1)								☒	☒
PONTO DE ENTRADA (2)	■				■	■		☒	☒
VARIÁVEL DECLARAÇÃO (3)	☒	☒	☒	☒	☒	☒	☒	☒	☒
VARIÁVEL REFERÊNCIA (4)	☒	☒	☒	☒	☒	☒	☒	☒	☒

CONVENÇÃO:

□ A RELAÇÃO PROVOCA ERRO

☒ NÃO EXISTE A RELAÇÃO

■ EXISTE AÇÃO

Ø - indeterminado
1 - arquivo
2 - ponto de entrada secundário
3 - ponto de entrada principal
4 - rotina BASE
5 - rotina FORÇADA
6 - ponto de entrada previlegiado
7 - variável
8 - nós especiais

DESCRIÇÃO DE ERROS E AÇÕES

TSE	EST.	DESCRIÇÃO
0	0	ignora
	1	erro-referência externa definida como arquivo
	2	ignora
	3	
	4	
	5	
	6	
1	0	erro-arquivo não foi definido globalmente
	1	
	2	
	3	
	4	
	5	
	6	
2	0	atualiza conforme o caso
	1	erro-ponto de entrada definido globalmente como file
	2	
	3	erro-ponto de entrada duplicado
	4	atualiza conforme o caso
	5	
	6	erro-ponto de entrada duplicado

Observação:

Na árvore não existe variáveis indefinidas.

5.3 Crítica a Árvore e Listas

Esta fase é responsável pela verificação da estrutura construida nas fases anteriores.

Neste processo de verificação, é feita uma advertência quando uma rotina forçada já conste de um ancestral, implicando portanto que esta rotina e todas as rotinas por ela referenciada aparecerão em dois módulos.

É testado se uma rotina base é referenciada por um módulo que não seja o seu PAI. Esta situação é considerada como erro, porque se a rotina aparecer em um ancestral não direto ela será montada toda neste ancestral e portanto não terá sentido também constar como módulo independente.

E finalmente é verificado se algum nó ficou indefinido, e em caso afirmativo é suprimida a edição, a menos que o comando que indica "geração mesmo com indefinidos" tenha sido fornecido.

5.4 Determinação de Bases

Nesta fase a árvore é atravessada em pré-ordem e para cada nó da árvore a lista encadeada é percorrida para que os endereços de carga e execução (campos A e B) sejam atualizados.

Quando os símbolos são do tipo 3, 4 ou 5 (respectivamente: ponto de entrada principal, rotina base ou rotina forçada), a atualização é obtida pela soma da base do início de carga do módulo mais o tamanho de todas as rotinas atualizadas no módulo. Ao término de uma lista são atualizadas as bases de início de carga de todos os FILHOS.

Quando os símbolos são do tipo 2 ou 6 (respectivamente: ponto de entrada secundário ou ponto de entrada previlegiado), o endereço de carga é o da rotina para onde o nó aponta (quando um ponto de entrada é atualizado o nó para o qual ele aponta já foi atualizado).

Já os endereços de execução são obtidos pela soma do endereço de execução relativo com o endereço de carga da rotina.

5.5 Criação dos Módulos

A árvore é novamente percorrida em pré-ordem e para cada nó da árvore a lista encadeada é percorrida, selecionando os nós de tipos 3, 4 ou 5 (respectivamente: ponto de entrada principal, rotina base ou rotina forçada).

Para cada nó deste tipo a partir de sua TSE cria-se um vetor de apontadores para as listas. Este vetor de apontadores relaciona os símbolos da TSE com os nós correspondentes. Ele é obtido da seguinte forma: para cada elemento da TSE procura-se na lista atual e em todos os ancestrais até encontrar.

Com o auxílio do vetor de apontadores os registros dos módulos objetos referentes a parte de texto e dicionário de relocabilidade são lidos e o texto é devidamente relocado e gerando em disco os módulos de carga correspondentes.

Cuidados especiais devem ser tomados quando da relocação de um endereço tipo arquivo, pois é necessário somar ao valor do campo B um valor correspondente ao fim da memória ocupada pelo maior dos ramos da estrutura de overlay considerada. Este endereço base refere-se ao início da tabela TRAL que será criada pelo CPROG a partir da tabela TAL for

necida.

Nesta fase é incluido no módulo raiz a tabela TAL de sua rotina base, pois somente serão considerados os arquivos definidos na rotina do módulo raiz.

5.6 Relatórios

Esta fase é responsável pela preparação do mapa de alocação. A árvore é atravessada em pré-ordem e as listas são percorridas.

Para cada nó da lista, o nome, tipo, disco em que se encontra, endereço de carga, endereço de execução e indicadores de definição são editados e impressos em ordem conveniente.

6º CAPÍTULO

Como já foi dito anteriormente todos os módulos do Sistema S.O.Co., exceto as rotinas que dependessem de velocidade ou restrição de memória, foram escritas na linguagem de alto nível denominada PLTI.

A implementação do S.O.Co. foi definida para se realizar em duas fases. A primeira que denominamos de "SOCO MINIMO" e a segunda o Sistema completo.

6.1 SOCO MINIMO

O S.O.Co. Minimo é parte mínima do sistema que permitirá que seja possível continuar o desenvolvimento do S.O.Co. no próprio Terminal Inteligente. Ele é composto dos seguintes módulos:

(i) Carregador de Programas (CPROG)

(ii) Editor de Referências Externas (REFEX)

(iii) Compilador/Interpretador (PLTI)

(iv) Núcleo Residente

(v) Programa em ROM

(vi) Utilitários

- Partida Fria (PPF)
- Inicialização de disco
- Rotina de carga de programa

A implementação passou por diversas fases.

Inicialmente o compilador PLTI, escrito em PLTI, foi convertido para linguagem PL/I e implantado no computador Burroughs B6700. Esta conversão foi facilitada pelo uso do Macro Expansor do PL/I.

Este compilador convertido foi denominado PLTI-1, e serviu como ferramenta para o desenvolvimento dos módulos do sistema escrito em PLTI.

Como a saída do compilador PLTI-1 não gera código compatível com o Sistema Cruzado SOS, foi necessário a utilização de programas filtros para compatibilizá-los de tal modo que o Editor de Referências Externas e o Carregador de Programas do SOS pudessem ser utilizados.

Os programas depois de convertidos para o formato do SOS, são montados e editados com o seu auxílio, sendo incorporadas neste ponto as partes desenvolvidas em Assembler. Os programas assim obtidos são autosuficientes ("Stand-alone") e portanto independentes do Sistema Operacional.

Desta forma, estes módulos independentes podem ser depurados no próprio Terminal Inteligente.

Após a depuração dos módulos independentes que denominaremos de PLTI', REFEX' e CPROG', passa-se para fase final de implementação.

Os programas PLTI, REFEX e CPROG são compilados com o auxílio do compilador independente que roda no Terminal Inteligente, PLTI' e os módulos objetos gerados em disco.

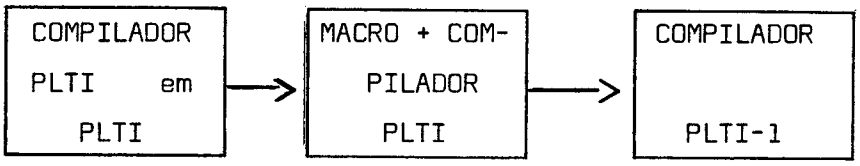
A seguir os programas compilados são editados com o auxílio do Editor de Referências Externas independente REFEX', gerando módulos de carga em disco.

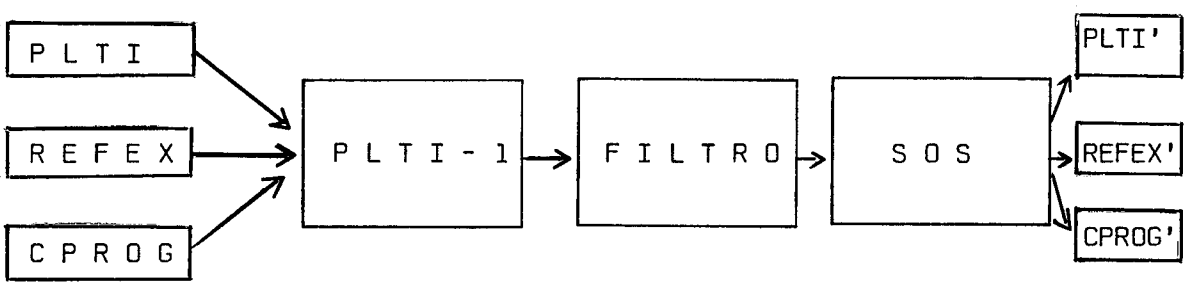
E finalmente o Carregador de Programas independente CPROG' é utilizado para carregar, anexando as rotinas de Entrada/Saída, o módulo de carga do CPROG gerando um módulo imagem, e concluindo a implementação do S.O.Co. Mínimo.

A partir deste momento o CPROG em disco pode ser utilizado para carregar qualquer outro programa, inclusive o PLTI e o REFEX, que para o Sistema Operacional são considerados programas comuns.

A figura 12 mostra o esquema de implementação descrito.

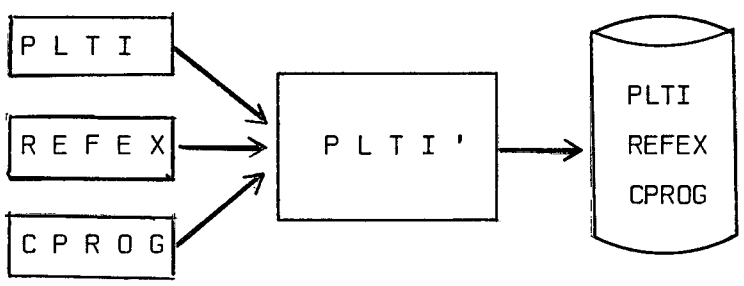
FASE QUE RODA NO BURROUGHS-6700





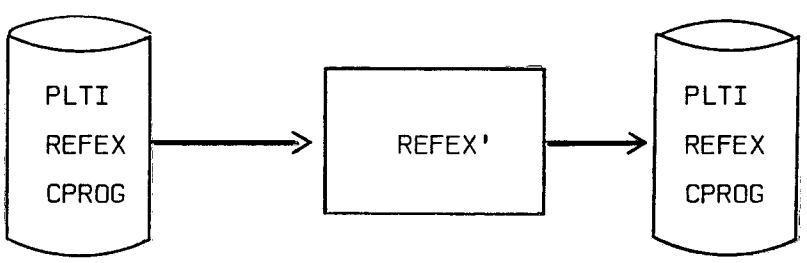
FASE QUE RODA NO TERMINAL INTELIGENTE

a - COMPILAÇÃO



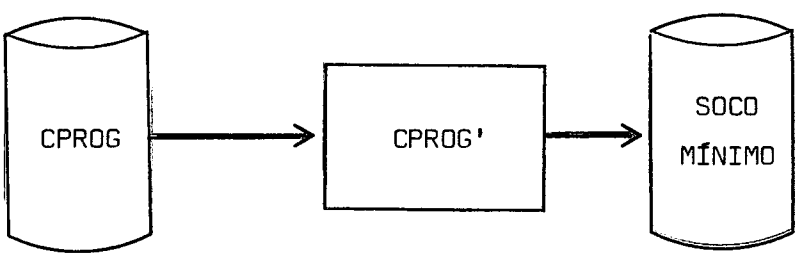
Programas Compilados em Disco

b - EDIÇÃO



Programas Montados a menos das Rotinas de E/S.

c - MONTAGEM DAS ROTINAS DE ENTRADA/SAÍDA (só CPROG)



CPROG já executável com rotinas de E/S.

FIG. 12

7º CAPÍTULO

O sistema S.O.Co. se encontra implantado em configuração do Terminal Inteligente, como a descrita na fig. 2, nas dependencias do Núcleo de Computação Eletrônica da UFRJ. Analisando a implantação do sistema e posteriormente observando o seu uso por vários usuários, podemos destacar alguns pontos em relação ao REFEX.

7.1 ESTRUTURA DO REFEX

O esquema de dividir o REFEX em fases independentes foi bastante positivo, pois facilitou bastante a depuração do sistema.

Observamos também que as fases 3(Criticar Árvores e Listas) e 4(Determinar Base) poderiam se fundir em um único bloco, manteve-se a divisão apenas por questão de documentação.

7.2 EDIÇÃO DE ROTINAS DO INTERPRETADOR

A idéia inicial, como foi descrita no corpo deste trabalho era que em cada módulo só fossem incluidas as rotinas do interpretador efetivamente utilizadas. Verificou-se que por menor que fosse o programa a edição envolveria cerca de 40 a 50 rotinas e passaria a gastar um tempo de edição não compatível. Por outro lado, verificou-se que 95% das rotinas do interpretador são usadas pela maioria dos programas.

Desta forma, o compilador PLTI, deixou de gerar a TRI, ficando todas as rotinas do interpretador agrupadas em um único módulo que é sempre forçado no módulo raiz.

7.3 PARTES CRÍTICAS EM TEMPO

Foi detetado dois pontos críticos em termos de tempo.

7.3.1 BUSCA NA ÁRVORE

A rotina de busca utilizada e a que consome o maior tempo da edição, correspondendo a cerca de 15 a 20% do tempo de edição.

Para melhorar um pouco a performance do sistema, basta converter esta rotina para Assembler.

7.3.2 GERAÇÃO DE MÓDULOS

Outro ponto crítico é a parte de movimentação de texto na fase de geração de módulos, que é feito byte a byte, com a devida relocação.

Para melhorar este ponto foi necessário alterar o esquema de movimentação/relocação, passando a movimentar todo o texto e depois acessar diretamente as posições a serem relocadas. Esta alteração resultou em uma economia de tempo de cerca de 45% na fase 5.

7.4 DESEMPENHO

A seguir apresentaremos alguns casos de edição para que possamos avaliar o desempenho do REFEX em termos de tempo de execução.

DESCRIÇÃO	TAMANHO BYTES	TEMPO (SEG.)						
		FASE1	FASE2	FASE3	FASE4	FASE5	FASE6	TOTAL
Programa sem rotinas externas	500	5	35	3	2	105	10	160
Programa sem rotinas externas	3500	5	35	3	2	200	10	255
Programa chamando uma rotina	1000	5	45	4	2	130	11	197
Programa chamando várias rotinas	2500	5	64	5	3	195	13	285
Programa com 2 módulos "overlay"	3000	8	57	5	4	235	17	326
Programa com vários módulos "overlay"	3200	12	68	6	5	295	21	407

Os tempos foram obtidos com o disco razoavelmente organizado (poucas colisões na busca).

Por outro lado os tempos de fase 5 dependem de localização dos arquivos no disco, pois esta fase é responsável pela manutenção do texto.

Considerando o porte da UCP utilizada podemos concluir que os tempos estão adequados.

APÊNDICE 1

Neste apêndice apresentaremos a estrutura de programação utilizada, bem como a descrição de todas as rotinas do sistema, e quando necessário os algoritmos respectivos.

A1.1 Estrutura de Programação

O sistema adota uma estrutura modular que facilita a depuração, permitindo também ampla flexibilidade para o uso de overlay.

As rotinas recebem nomes padronizados da seguinte forma:

- Os nomes estão no formato RNxxxx, onde:
  - R - designa o subsistema REFEX
  - N - é um número de 0 a 6 que indica a fase onde a rotina será usada. Quando 0 representa rotinas gerais.

A figura 13 apresenta o fluxo geral de chamadas.

A1.2 Programa Principal - RCONT

Descrição:

Controla todo REFEX, sendo responsável pelas chamadas de todas as fases e pelo controle de overlay.

Algoritmo:

1. Montar árvore
2. Analisar indefinidos
3. Criticar árvores e listas
4. Determinar bases e end. de execução
5. Montar módulos fazendo relocação se necessário
6. Fornecer mapa de alocação se necessário
7. Fim

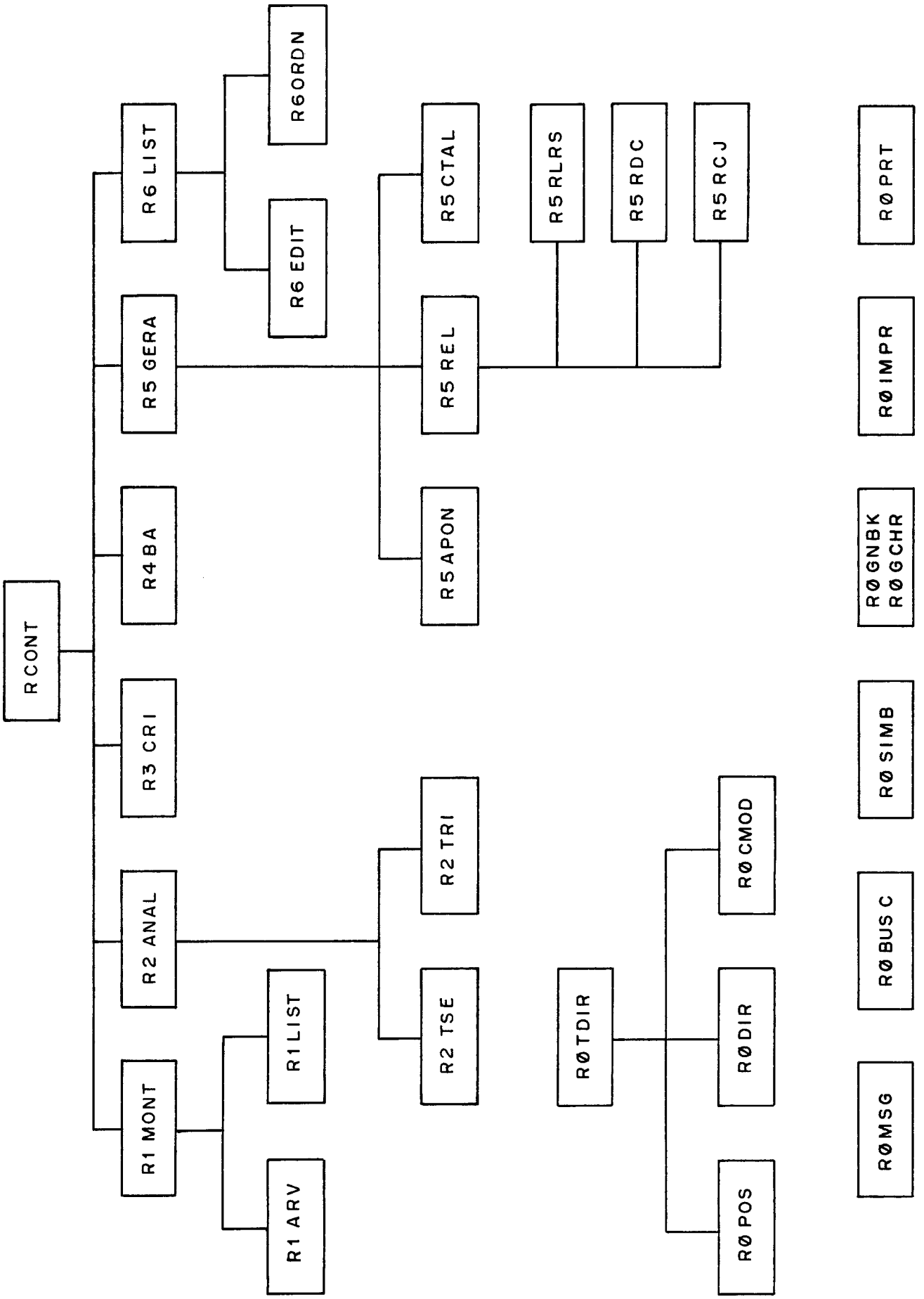

RCONT
R1 MONT
R1 LIST
R1 ARV
R2 ANAL
R2 TRI
R2 TSE
R3 CRI
R4 BA
R5 GERA
R6 LIST
R6ORDN
R6 EDIT
R5 REL
R5 CTAL
R5 RLRS
R5 RDC
R5 RCJ
R5APON
RØTDIR
RØ CMOD
RØDIR
RØ POS
RØPRT
RØIMPR
RØGNBK
RØGCHR
RØ SIMB
RØBUS C
RØMSG

Fig. 13

A1.3 Rotinas de Fase 1

A1.3.1 Rotina R1MONT

Descrição:

Analisa os parâmetros de entrada para o REFEX, inicializando o POOL de trabalho em função das informações fornecidas.

Algoritmo:

1. Inicialização
2. SE existe parâmetro
   ENTÃO atualiza início e fim de parâmetro
   SENÃO lê comando de entrada e atualiza início e fim do registro de entrada
3. Extrai símbolo
4. Inclui no POOL a raiz da árvore
5. Extrai volume se houver
6. SE caracter é '<'
   ENTÃO extrai lista de base e forçadas
   extrai caracter
7. Inclui nós especiais
8. SE caracter é '('
   ENTÃO
   8.1 NÍVEL=1, PARENTESCO=filho
   8.2 ENQUANTO Not (; ou .) FAÇA
   8.2.1 ENQUANTO nível diferente 0 FAÇA
   8.2.1.1 Extrai símbolo
   8.2.1.2 Inclui na árvore conforme parentesco
   8.2.1.3 Extrai caracter

8.2.1.4 SE caracter é '<'

ENTÃO analisa lista de base e forçadas

8.2.1.5 SE caracter é '('

ENTÃO avança ponteiro e extrai caracter

NIVEL=*+1 e parentesco=filho

SENÃO

8.2.1.5.1 PARA=falso

8.2.1.5.2 ENQUANTO ')' ou ','

e não PARA FAÇA

SE ')' ENTÃO NÍVEL=*-1

. avança ponteiro

. extrai caracter

. volta ponteiro que varre

a árvore

SENÃO SE é ',' ENTÃO

. parentesco=irmão

. avança ponteiro

. extrai caracter

. PARA=verdade

8.2.2 SE NÍVEL Not = 0 ENTÃO erro

9. SE é ';' ENTÃO

9.1 Extrai caracter

9.2 SE é 'M' ENTÃO MAPA=1 e extrai caracter

9.3 Se é 'G' ENTÃO GERA=1 e extrai caracter

10. SE diferente de '.' ENTÃO erro-comando incompleto

11. Retorna

A1.3.2 Rotina R1ARV

Descrição:

Inclui um nó na árvore conforme a sua característica (raiz, filho ou irmão)

Algoritmo:

1. Aloca nó para árvore (salvando o anterior)
2. Aloca nó HEAD correspondente ao nó da árvore (inicializa campos com o default)
3. EXECUTA conforme o tipo

   RAIZ

   3.1 LINK1=LINK2=LINK3=-1

   FILHO

   3.1 LINK1 do nó anterior nó atual (pai aponta p/ filho)

   3.2 LINK1=LINK3=-1 (do nó atual)

   3.3 LINK2 do nó atual = nó anterior

   IRMÃO

   3.1 LINK3 do nó anterior = nó atual

   3.2 LINK1=LINK3=-1 (do nó atual)

   3.3 LINK2 do nó atual = LINK2 do nó anterior
4. Retorna

A1.3.3 Rotina R1LIST

Descrição:

Analisa uma lista com o nome da rotina base e de todas as forçadas

Algoritmo:

1. Extrai caracter

2. SE não é letra ou ','

ENTÃO erro na lista

SENÃO

2.1 SE é letra ENTÃO

2.1.1 Volta ponteiro de caracteres

2.1.2 Extrai símbolo

2.1.3 Inclui como rotina base

2.1.4 Extrai caracter

2.2 SE é ',' ENTÃO

2.2.1 ENQUANTO é ',' FAÇA

2.2.1.1 Extrai símbolo

2.2.1.2 Inclui como forçada

2.2.1.3 Extrai caracter

2.3 SE não é '>' ENTÃO erro na lista

3. Retorna

A1.2 Rotina da Fase 2

A1.4.1 Rotina R2ANAL

Descrição:

Controla a fase de análise e símbolos indefinidos. Para cada símbolo indefinido torna-o definido e quando necessário gera novos símbolos indefinidos.

Algoritmo:

1. Inicialização

PT$ARVORE aponta para raiz da árvore

2. ENQUANTO PT$ARVORE dif. flag FAÇA

2.1 Aponta nó da lista

(PONTALISTA=PT1)

2.2 Determina fim da lista (FIMLISTA)

2.3 ENQUANTO PONTALISTA dif. flag. FAÇA

2.3.1 SE marcado como indefinido

ENTÃO

2.3.1.1 Procura nome do diretório

(se não for variável)

2.3.1.2 SE existe

ENTÃO

2.3.1.2.1 Atualiza campos

2.3.1.2.2 Analisa TSE da rotina

2.3.1.2.3 Analisa TRI da rotina

2.3.2 Avança Ponteiro

(PONTALISTA=LINK)

2.4 SE filho esq. dif flag

ENTÃO PT$ARVORE=Filho esq.

SENÃO

2.4.1 SE irmão dir. dif flag

ENTÃO PT$ARVORE=irmão dir.

SENÃO

2.4.1.1 ENQUANTO pai dif. flag e irmão=flag

FAÇA

PT$ARVORE=pai

2.4.1.2 PT$ARVORE=irmão dir.

3. Retorna

A1.4.2 Rotina R2TSE

Descrição:

Controla a fase de análise e símbolos indefinidos. Para cada símbolo indefinido a rotina procura torná-lo defi do e gera novos nós quando necessário.

Algoritmo:

1. Inicialização
2. Lê módulo pulando TAL (se não for base)
3. PARA todas as entradas da TSE FAÇA
   (se não é nome de rotina)
   3.1 SE é variável
       ENTÃO liga flag de tipo de busca
       SENÃO desliga
   3.2 Procura na lista atual e em todas as listas
       de módulo ancestrais
   3.3 SE encontrou
       ENTÃO
       3.3.1 SE definido
             ENTÃO trata possibilidades para definido
             SENÃO trata possibilidades para indefinido
             SENÃO conforme o tipo inclui ou manda
             mensagem conveniente.
4. Retorna

As possibilidades indicadas foram detalhadas no item 5.2.1

A1.4.3 Rotina R2TRI

Descrição:

Inclui na árvore rotinas do interpretador que não constem da lista atual ou das listas dos ancestrais.

Algoritmo:

1. PARA todas as entradas da TRI
   1.1 Converte o código para nome da rotina do interpretador

1.2 Procura na lista atual ou nós ancestrais

1.3 SE não encontrou ENTÃO inclui no fim da lista

2. Retorna

A1.5 Rotinas da Fase 3

A1.5.1 Rotina R3CRI

Descrição:

Critica a árvore gerada na fase 2, fazendo consistência.

Algoritmo:

1. Inicialização

(PT$ARVORE aponta para raiz da árvore)

2. ENQUANTO PT$ARVORE dif flag FAÇA

2.1 Aponta nó

(PONTALISTA=PT1)

2.2 ENQUANTO especiais, base ou forçadas

2.2.1 SE base ou forçada

ENTÃO procura nos ancestrais

SE achou ENTÃO erro

2.2.2 Aponta para o próximo nó da lista

2.3 SE filho esq. dif flag

ENTÃO PT$ARVORE aponta filho esq.

SENÃO

2.3.1 SE irmão dir. dif flag

ENTÃO PT$ARVORE aponta irmão direita

SENÃO

2.3.1.1 ENQUANTO pai dif flag e

irmão dif = flag

FAÇA

2.3.1.1.1 PT$ARVORE aponta pai

2.3.1.2 PT$ARVORE = irmão dir

3. Varre sequencialmente até o fim ou até encontrar uma referência não resolvida. Se houver pelo menos um símbolo indefinido liga flag.

4. Retorna

A1.6 Rotinas da Fse 4

A1.6.1 Rotina R4BA

Descrição:

Atualiza endereço de carga e de execução de todos os símbolos.

Algoritmo:

1. Inicialização

   PT$ARVORE aponta raiz da árvore

   BASE=BASEINICIAL (primeira pos. disponível depois do residente)

2. ENQUANTO PT$ARVORE dif. flag FAÇA

   2.1 Aponta nó da lista e pega base do módulo

   2.2 ENQUANTO PONTALISTA dif. flag FAÇA

   2.2.1 SE entry principal ou base ou forçada

   ENTÃO

   2.2.1.1 Calcula end. de carga e atualiza end. de execução

   2.2.1.2 Atualiza base para próxima rotina

   SENÃO

2.2.1.1 SE entry previlegiado ou secundário ou especial

ENTÃO

2.2.1.1.1 End. de carga de um entry é o mesmo end. de toda rotina (pega end. da rotina através do ponteiro que consta do nó (A) e atualiza end. de execução

2.2.2 Passa para o próximo da lista

2.3 Trata overflow de memória

2.4 SE filho esq. dif. flag

ENTÃO

2.4.1 Coloca no filho esq. e em todos os seus irmãos o end. de carga dos módulos

2.4.1 PT$ARVORE = filho esq.

SENÃO

2.4.1 SE irmão dir. dif. flag

ENTÃO PT$ARVORE = irmão dir.

SENÃO

2.4.1.1 ENQUANTO pai dif. flag e irmão dir=flag

FAÇA

PT$ARVORE = pai

2.4.1.2 PT$ARVORE = irmão dir.

3. Retorna

A1.7 Rotinas da Fase 5

A1.7.1 Rotina R5GERA

Descrição:

Controla a geração dos módulos

Algoritmo:

1. Inicialização

(PT$ARVORE aponta raiz da árvore)

2. ENQUANTO PT$ARVORE diferente flag FAÇA

2.1 PONTALISTA = PT1

2.2 ENQUANTO PONTALISTA dif flag FAÇA

2.2.1 SE entry principal ou

base ou

forçada ENTÃO

2.2.1.1 Lê rotina, posiciona na TSE criando o vetor de apontadores para árvore.

2.2.1.2 Faz relocação do texto a partir do vetor de apontadores.

2.2.2 PONTALISTA = Link

2.3 SE módulo gerado foi a raiz

ENTÃO copia a TAL do programa principal para o fim do módulo

2.4 SE filho esq. diferente flag

ENTÃO PT$ARVORE = filho esquerda

SENÃO

2.4.1 SE irmão dir. dif flag

ENTÃO PT$ARVORE = pai

SENÃO

2.4.1.1 ENQUANTO pai dif flag e irmão = flag FAÇA

PT$ARVORE = pai

2.4.1.2 PT$ARVORE = irmão dir.

3. Retorna

A1.7.2 Rotina R5APON

Descrição:

Cria vetor de apontadores a ser utilizado na relocação para cada rotina.

Algoritmo:

1. Localiza início da TSE
2. Inicializa vetor de apontadores
3. Para cada entrada da TSE procura na lista atual e em todos os ancestrais até encontrar, colocando o endereço no vetor de apontadores
4. Retorna

A1.7.3 Rotina R5CTAL

Descrição:

Cria a TAL para um módulo tipo REFEX

Algoritmo:

1. Atualiza arquivo objeto para ler programa principal (atualização da TRAL)
2. Lê TAL copiando para arquivo de saída
3. Retorna

A1.7.4 Rotina R5REL

Descrição:

Controla a relocação de uma determinada rotina com o auxílio do vetor de apontadores e do DR.

A1.7.4 Rotina R5REL

Descrição:

Controla a relocação de uma determinada rotina com o auxílio do vetor de apontadores e do DR.

Algoritmo:

1. Inicialização
2. ENQUANTO tam. bloco maior ou igual 0 FAÇA
   - 2.1 Lê registro com TEXTO + DR
   - 2.2 ENQUANTO tam. > 0 FAÇA
     - 2.2.1 SE houver quebra (location-counter diferente do end. de carga do bloco)
       ENTÃO gera buracos com zero qté location counter=end. de carga do bloco
     - 2.2.2 PARA todos os bytes do bloco
       - 2.2.2.1 SE location-counter diferente desl. no DR+BASE da Rotina
         ENTÃO não é relocavel (gera código)
         SENÃO é relocavel
         CONFORME tipo no DR
         FAÇA
         - Relocação tipo CAL ou JMP
         - Relocação tipo LRS

. Relocação tipo DC

2.2.3 Passa para o próximo bloco

3. Retorna

A1.7.5 Rotina R5RCJ

Descrição:

Faz relocação de instruções do tipo CAL e JMP.

Instruções com o seguinte formato:

COD	LOW	HIGH

Algoritmo:

1. Pega 3 bytes do código objeto
2. Monta endereço
3. Soma endereço com end. da lista conforme vetor de apontadores
4. Desmonta endereço
5. Coloco os 3 bytes no buffer de saída
6. Retorna

A1.7.6 Rotina R5RDC

Descrição:

Faz relocação de endereços marcados como do tipo DC.

Endereços que obedecem ao seguinte formato:

HIGH	LOW

Algoritmo:

1. Pega 2 bytes do código objeto
2. Atualiza ponteiro para árvore conforme vetor de apontadores e soma end. de carga que consta da lista
3. SE tipo na árvore é FILE ENTÃO soma também end. da TRAL

4. Coloca os 2 bytes relocados no buffer de saída

5. Retorna

A1.7.7 Rotina R5RLRS

Descrição:

Faz relocação das instruções do tipo LRS

Instruções com o seguinte formato:

COD	LOW	COD	HIGH

Algoritmo:

1. Pega 4 bytes do código objeto
2. Monta endereço
3. Soma endereços com end. da lista conforme vetor de apontadores
4. Desmonta endereço
5. Coloca os 4 bytes no buffer de saída
6. Retorna

A1.8 Rotinas da Fase 6

A1.8.1 Rotina R6LIST

Descrição:

Rotina que lista a árvore e as listas do REFEX

Algoritmo:

1. Aponta para raiz da árvore
2. ENQUANTO PT$ÁRVORE dif. flag. FAÇA
   2.1 Ordena a lista de rotinas do módulo por end. de carga e tipo
   2.2 Percorre a lista, listando as características de cada rotina
   2.3 Segue para o próximo nó da árvore
3. Retorna

A1.8.2 Rotina R6EDIT

Descrição:

Edita para cada nó da lista, o nome, tipo, disco, end. de carga, end. de execução, end. da rotina mãe, dizendo se a rotina está ou não definida, listando outras informações.

Algoritmo:

1. Escreve o nome do módulo e seu end. de carga
2. Coloca o cabeçalho especificando que tipo de informações se seguem
3. ENQUANTO PONTALISTA dif. flag FAÇA
   - 3.1 Dá as seguintes informações
     - 3.1.1 Nome do entry-point
     - 3.1.2 Tipo do entry
     - 3.1.3 Disco
     - 3.1.4 End. carga
     - 3.1.5 End. execução
     - 3.1.6 SE entry-secundário, dá o end. da rotina mãe
     - 3.1.7 Diz se foi ou não encontrado no disco
   - 3.2 Avança para o próximo da lista
4. Retorna

A1.8.3 Rotina R6ORDN

Descrição:

Ordena uma lista por endereço de carga e tipo.

Algoritmo:

1. TROCOU = verdade
2. ENQUANTO trocou
   2.1 Inicializa ponteiros
   2.2 TROCOU = falso
   2.3 ENQUANTO elem. seguinte dif. flag FAÇA
      2.3.1 SE elemento seguinte devia preceder o atual
            ENTÃO
            2.3.1.1 Troca esses elementos
            2.3.1.2 TROCOU = verdade
      2.3.2 Avança ponteiros
3. Retorna

A1.9 Rotinas Gerais

A1.9.1 Rotina ROBUSC

Descrição:

Procura um símbolo nas listas de módulos ancestrais quando encontrar informa o endereço do nó onde se encontra.

A busca é feita incluindo a lista que está sendo processada em função de PT$ÁRVORE. A busca leva em consideração se é uma variável ou símbolo (entry) que está sendo procurado.

Algoritmo:

1. Inicialização

   Ponteiro aponta módulo que está sendo tratado

   Achou = falso

```
2. ENQUANTO ponteiro dif. flag e não achou FAÇA
   2.1 PONTALISTA = PT1
   2.2 ENQUANTO PONTALISTA dif. flag e não achou FAÇA
       2.2.1 SE nome do nó = nome procurado
             ENTÃO
             2.2.1.1 SE BUSCA$VAR
                     ENTÃO SE é variável
                           ENTÃO achou = verdade
                           SENÃO PONTALISTA = Link
                     SENÃO SE é variável
                           ENTÃO PONTALISTA = Link
                           SENÃO ACHOU = verdade
             SENÃO PONTALISTA = Link
   2.3 PONTEIRO = pai do nó
3. SE achou ENTÃO END$NO = PONTALISTA
            SENÃO
            3.1 ENS$NO = 0
            3.2 SE BUSCAVAR e não foi crítica
                ENTÃO
                3.2.1 PONTEIRO = PT$ÁRVORE
                3.2.2 PONTEIRO = Link1
                3.2.3 ENQUANTO PONTEIRO = -1 e não
                      achou FAÇA
                      3.2.3.1 PONTALISTA = PT1
                      3.2.3.2 SE nome do nó = no
                              me procurado
```

ENTÃO END$NO = PONTALISTA

ACHOU = verdade

SENÃO PONTEIRO = Link3

4. Retorna

A1.9.2 Rotina ROGNBK e ROGCHR

Descrição:

Extrai um caracter ou o primeiro caracter diferente de branco do buffer apontado pelo ponteiro PT$PARAMETRO fornecendo o código ASCII e o tipo correspondente.

Algoritmo:

SE ROGNBK

1 - Procura o primeiro caracter diferente de branco

2 - Analisar caracter

3 - Retorna

SE ROGCHR

1 - Verifica o tipo através de acesso direto e uma tabela

2 - Se tipo inválido

Então avança ponteiro e chama-se recursivamente

A1.9.3 Rotina ROIMPR

Descrição:

Rotina de auxílio a impressão

Algoritmo:

1. Se linha > 60 ou car. controle é para pular página

ENTÃO

1.1 Lê cabeçalho de página no disco

1.2 Coloca data no cabeçalho

1.3 Coloca num. na página

1.4 Imprime cabeçalho

1.5 Troca caracter de controle para saltar 2 linhas

2. Imprime a mensagem

3. Retorna

A1.9.4 Rotina ROMSG

Descrição:

Rotina que pega mensagens no disco, montando-as para impressão.

Algoritmo:

1. Posicione o arquivo no registro que contém a mensagem e lê.
2. Executa inserção de símbolo se necessário
3. Retorna

A1.9.5 Rotina ROPRT

Descrição:

Rotina que altera a TAF de disco de modo a permitir a troca durante a execução de vários arquivos físicos para o mesmo arquivo lógico.

A1.9.6 Rotina ROSIMB

Descrição:

Extrai um identificador de até 6 caracteres.

Algoritmo:

1. Extrai um caracter
2. SE não é letra ENTÃO erro - símbolo inválido

2.1 VERD=True; I=1 e coloca letra no vetor

2.2 ENQUANTO VERD FAÇA

2.2.1 Extrai caracter

2.2.2 SE letra ou dígito

ENTÃO

2.3.2.1 Se I = 6 ENTÃO coloca caracter no vetor

SENÃO erro - simb. muito longo e ignora o resto do símbolo

2.2.3 Incrementa contador (I)

3. Volta ponteiro e retorna

(o pont. aponta último caracter do símbolo)

4. Retorna

A1.9.7 Rotina para manuseio do diretório (ROTDIR)

Possui 3 entry-points:

- RODIR - Localiza uma entrada no diretório
- ROPOS - Posiciona o arquivo "CARGA" na SOCOWS
- ROCMOD - Coloca um módulo no diretório

APÊNDICE 2

Lista de mensagens enviada pelo sistema REFEX

R-01 - CARACTER INVALIDO

R-02 - COMANDO INCOMPLETO

R-03 - SIMBOLO INVALIDO → XXXXXX

R-04 - SIMBOLO MUITO LONGO TRUNCADO → XXXXXX

R-05 - AREA DE TRABALHO INSUFICIENTE

R-06 - ERRO NA LISTA DE BASE E FORCADOS

R-07 - SINTAXE INVALIDA NA ESTRUTURA DE OVERLAY

R-08 - OPCAO INVALIDA APOS ';' → DEVE SER 'I1' OU 'G'

R-09 - ESPECIFICAÇAO DE VOLUME INCORRETA

R-10 - XXXXXX → NAO CONSTA DO DIRETORIO

R-11 - XXXXXX → REFERENCIA EXTERNA DEFINIDA COMO FILE

R-12 - XXXXXX → FILE NAO FOI DEFINIDO GLOBALMENTE

R-13 - XXXXXX → ENTRY FOI DEFINIDO COMO FILE

R-14 - XXXXXX → SIMBOLO COM NOME RESERVADO

R-15 - XXXXXX → ENTRY POINT COM NOME DUPLICADO

R-16 - XXXXXX → VARIAVEL DUPLICADA

R-17 - XXXXXX → VARIAVEL INDEFINIDA

R-18 - ERRO DE EDICAO

R-19 - XXXXXX → ROT. BASE OU FORCADA JA CONSTA DE UM ANCESTRAL

R-20 - O PROGRAMA EDITADO EXCEDEU A CAPACIDADE DE MEMORIA

R-21 - ERRO NO POSICIONAMENTO DA SOCOWS - VERIFIQUE DISCOS MONTADOS

R-22 - XXXXXX → MODULO JA EXISTE NO DIRETORIO

R-23 - ERRO DURANTE A INCLUSAO DO MODULO XXXXXX

R-24 - RESERVADO PARA USO FUTURO

R-25 - RESERVADO PARA USO FUTURO

R-26 - LINKEDICAO COMPLETADA NORMALMENTE

R-27 - LINKEDICAO FORCADA POR OPCAO 'G'

R-28 - LINKEDICAO DESCONTINUADA

R-29 - ERRO DE ENTRADA E SAIDA

R-30 - DETETADO FIM DE ARQUIVO EM LUGAR INVALIDO

R-31 - FALTOU AREA DE TRABALHO EM DISCO

APÊNDICE 3

Variáveis globais utilizadas no sistema REFEX, com suas definições respectivas. Os indicadores B, A e L representam respectivamente: byte, adress e label.

Rótulos utilizados em "ABORT"

1. TERMINACAO$ANORMAL (L)
   - Label de desvio para finalização anormal do REFEX.
2. ERRO$ES (L)
   - Label de desvio para quando é detetado um erro de entrada e saída.
3. ERRO$EOF1 (L)
   - Label de desvio para quando ocorre "END OF LINE" na leitura do cartão de controle.
4. FALTA$ESPAÇO (L)
   - Label de desvio para quando ocorre um "END OF LINE" no arquivo de saída do REFEX.

Endereços do Residente Utilizado

5. ENDTEMPARM (A)
   - Endereço do "flag" do residente que indica se tem ou não parâmetro.
6. ENDPTPARM (A)
   - Endereço do ponteiro que aponta o parâmetro, quando este existe.

Endereços dos Buffers Utilizados

7. END$BUFFER (A)

   - Endereço do buffer de entrada.

8. END$BUF1 (A)

   - Endereço do primeiro buffer de 512 bytes de E/S.

9. END$BUF2 (A)

   - Endereço do segundo buffer de 512 bytes de E/S.

10. END$BUFF$MSG (A)

    - Endereço do buffer de mensagens.

Ponteiros para o Pool de Nós

11. PT$POOL1 (A)

    - Ponteiro que desce no pool. É responsável pela alocação de nós para as listas.

12. PT$POOL2 (A)

    - Ponteiro que sobe no pool. É responsável pela alocação de nós da árvore.

Ponteiros para o Parâmetro

13. PT$PARAMETRO (A)

    - Ponteiro para o último caracter, do parâmetro ou do buffer de entrada, que foi extraido.

14. PTFIMPARM (A)

    - Ponteiro para o fim do parâmetro ou do buffer de entrada.

Ponteiros para Árvore e Listas

15. PT$RAIZ (A)

    - Ponteiro para o nó raiz da árvore.

16. PT$ARVORE(A)

    - Ponteiro que varre a árvore.

17. PONTALISTA(A)

    - Ponteiro que varre as listas.

18. FIMLISTA(A)

    - Ponteiro que aponta o fim da lista de símbolos referente a um nó tratado, é utilizado para facilitar insenções.

Indicadores Gerais

19. BUSCA$VAR(B)

    - Indicador do tipo de busca na estrutura. Se 1, só são consideradas as variáveis. Se 0 os demais tipos.

20. MODULO$RAIZ(B)

    Indicador informando que o módulo que está sendo gerado é um módulo Raiz.

21. GERA(B)

    - Indicador da presença da opção GERA.

22. MAPA(B)

    - Indicador da presença da opção MAPA.

23. TEM$INDEFINIDOS(B)

    - Indica se na fase de crítica foi detetada alguma rotina indefinida.

24. NUMDAPAG(B)

    - Contador de páginas

25. LINHADAPAG(B)

    - Contador de linhas.

26. SETORES$GRAVADOS(A)

    - Contador de setores gravados.

27. CRITICA(B)

- Indica que está sendo executada a fase de crítica da estrutura.

28. LOCATION$COUNTER(A)

- Utilizado para geração do texto relocado na imagem da memória.

Outros Endereços

29. END$NOME$VOL

- Endereço do vetor que contém o nome do volume onde serão colocados os módulos de saída.

30. PONT$BUF(A)

- Ponteiro para o buffer utilizado para leitura do módulo objeto (tabelas)

31. BASE$INICIAL(A)

- Início da memória disponível para carga.

32. TAM$MEMORIA(A)

- Tamanho de memória RAM disponível.

33. TAM$TRAL(A)

- Valor colocado em nós referente a arquivos. Para cada arquivo é colocado um valor que para o primeiro é zero e para os seguintes são obtidos pelo valor anterior mais oito.

34. INICIO$TRAL(A)

- Endereço do primeiro byte livre depois do texto, endereço onde será montada a TRAL.

35. END$CABEÇALHO(A)

- Endereço do buffer a ser utilizado para impressão do cabeçalho.

36. END$SIMBOLO$MSG(A)

- Endereço do vetor do símbolo que será utilizado em uma mensagem.

APÊNDICE 4

ESTATÍSTICA

Para um acompanhamento do desempenho do sistema e para que sejam detectados pontos críticos, o indicador luminoso do TI é utilizado para informar qual a rotina do sistema que está sendo executada. A seguir apresentamos os códigos utilizados para cada rotina integrante do sistema.

RCONT	77	R2ANAL	20
ROAVAN	0A	R2TRI	21
ROBUSC	01	R2TSE	22
ROCMOD	0D	R3CRI	30
RODIR	06	R4BA	40
ROGCHR	03	R5APON	51
ROGNBK	02	R5GERA	50
ROIMPR	09	R5CTAL	52
ROMSG	08	R5RCJ	54
ROPOS	0C	R5RDC	55
ROPRT	04	R5REL	53
ROSIMB	05	R5RLRS	56
R1ARV	10	R6EDIT	61
R1LIST	11	R6LIST	60
R1MONT	12	R6ORDN	62

REFERÊNCIAS BIBLIOGRÁFICAS

1. Marques, Ivan da Costa
"Computação na UFRJ - Uma Perspectiva"
Anais do VII CNPD - SUCESU
S. Paulo - 1974

2. Marques, Ivan da Costa
"Momento Decisivo para os Computadores Brasileiros"
Anais do VII CNPD - SUCESU
S. Paulo - 1975

3. Marques, Ivan da Costa
"A Opção Urgente - Autonomia ou Dependencia Tecnológica?"
Revista Dados e Idéias - Volume I nº 3
Dezembro/Janeiro - 1975/76

4. Granja, E. P. e outros
"Terminal Inteligente"
Anais VIII CNPD - SUCESU
S. Paulo - 1974

5. Faller, Newton e outros
"Terminal Inteligente - Implementação e Desempenho"
Pesquisa Tecnológica em Computação
Publicação interna NCE/UFRJ - 1975

6. Rodrigues, Guilherme Chagas e Araújo, J. Fábio M.
"Desenvolvimento de Software para um Terminal Inteligente"
Anais do VII CNPD - SUCESU
S. Paulo - 1974

7. Rodrigues, Guilherme Chagas e outros
"Sistema para Desenvolvimento de Software - Implementação"
Pesquisa Tecnológica em Computação
Publicação interna NCE/UFRJ - 1975

8. "Sistema Operacional de Simulação - S.O.S"
Manual do Usuário
Publicação interna NCE/UFRJ - 1975

9. Bezerra, Milton A.
"Implementação e Otimização de um CROSS-SOFTWARE"
Anais III Seminário de Hardware e Software
Rio Grande do Sul - 1976

10. Rodrigues, Guilherme C.
"Sistema Operacional para Terminal Inteligente"
Anais III Seminário de Hardware e Software
Rio Grande do Sul - 1976

11. Knut, Donald E.
"The Art of Computer Programming"
Volume 1-3

12. Presser, L. White J. R.
Linkers and Loaders
ACM Computing Surveys
Set. 1972

13. Barron, D. W.
Assemblers and Loaders

14. Operation System Linguage Editor - IBM System/360
Program Logic Manual GY286667

15. Melo, P. C. Moraes
"Uma Linguagem para Microcomputadores-PLTI"
Anais IX CNPD - SUCESU
Rio de Janeiro - 1976

16. Vida Cura, J. Carlos
"Sistema de Entrada e Saída para o Terminal Inteligente"
Anais IX CNPD - SUCESU
Rio de Janeiro - 1976

17. Rodrigues, Guilherme C.
"A Função Administrativa dos Terminais Inteligentes"
Revista Dados e Idéias - Volume 2 nº 4
Fevereiro/Março - 1977