IMPLEMENTAÇÃO DE BASIC PARA OS MICROCOMPUTADORES COBRA/300

VÂNIA APARECIDA DINARDO

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. José L.M. Rangel Netto
(Presidente)

Profa. Sueli Mendes dos Santos

Profa. Ligia Alves Barros

Prof. Edil S. T. Fernandes

RIO DE JANEIRO , RJ - BRASIL

JANEIRO DE 1984

DINARDO, VÂNIA APARECIDA

IMPLEMENTAÇÃO DE BASIC PARA OS MICROCOMPUTADORES COBRA/300 (Rio de Janeiro) 1983.

VII, 100p. , 29,7cm (COPPE-UFRJ, M.Sc. , Engenharia de Sistemas e Computação, 1983).

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

1. Compiladores I.COPPE/UFRJ II. Título (Série).

Aos meus pais
José e Edelfina

AGRADECIMENTOS

Aos meus pais,cuja dedicação e esforços foram fundamentais para a minha formação.

Ao Professor RANGEL pela orientação, pelos conhecimentos ministrados e pela disponibilidade constante.

À Professora LIGIA pelo incentivo, orientação e acompanhamento dado ao trabalho.

Ao INSTITUTO DE PESQUISAS ESPACIAIS, INPE, na pessoa do Dr. CELSO DE RENNA E SOUZA,por propiciar a continuidade deste trabalho.

Ao INSTITUTO TECNOLÓGICO DE AERONÁUTICA , ITA,na pessoa do Professor FERNANDO WALTER, por ceder o equipamento para o término da implementação.

Ao OZIEL,que além de autor das artes gráficas,sempre colaborou com apoio e incentivo.

Ao amigo JONY SANTELLANO, pela contribuição dada à elaboração deste documento.

A todos que direta ou indiretamente me auxiliaram ao longo do tempo para que este meu objetivo fosse alcançado.

Resumo da Tese Apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

IMPLEMENTAÇÃO DE BASIC PARA OS MICROCOMPUTADORES COBRA/300

VÂNIA APARECIDA DINARDO

Janeiro 1984

Orientador: José Lucas Mourão Rangel Netto
Programa: Programa de Engenharia de Sistemas

Texto do Resumo

Este trabalho descreve o projeto e a implementação de um sistema de compilação para a linguagem BASIC, composto de um interpretador com características interativas e com um certo grau de incrementalismo; para os microcomputadores COBRA-300 e COBRA-305 da COBRA Computadores e Sistemas Brasileiros S.A. . A linguagem BASIC implementada é um subconjunto do BASIC-PLUS-2 disponível no sistema PDP-11 da Digital Equipment Corporation.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requeriments for the degree of Master of Science (M.Sc.).

IMPLEMENTAÇÃO DE BASIC PARA OS MICROCOMPUTADORES COBRA/300

VÂNIA APARECIDA DINARDO

Janeiro 1984

Chairman: José Lucas Mourão Rangel Netto
Department: Programa de Engenharia de Sistemas

Text Abstract

This work describes the design and the implementation of a compiling system for the BASIC language. It consists of an interpreter with interactive and incremental features, for the COBRA-300 and COBRA-305 microcomputers of COBRA Computadores e Sistemas Brasileiros S.A. The implemented BASIC language is a subset of BASIC-PLUS-2, wich is available at PDP-11 of Digital Equipment Corporation.

ÍNDICE

CAPÍTULO I

I N T R O D U Ç Ã O

I.1 - OBJETIVO DA TESE

Os microcomputadores estão cada vez mais sendo utilizados para um vasto universo de aplicações. Esta diversidade de uso implica numa demanda por parte dos usuários por linguagens de fácil aprendizado que permitam uma rápida familiarização com o ambiente de programação. A disponibilidade de uma linguagem de aprendizado simples pode possibilitar, já a curto prazo, uma utilização eficiente da máquina.

O objetivo deste trabalho é oferecer a linguagem BASIC para os microcomputadores COBRA-300 e COBRA 305 da COBRA COMPUTADORES E SISTEMAS BRASILEIROS S.A. O SISTEMA BASIC-C , assim denominado, consta de um interpretador com características interativas e com um certo grau de incrementalismo, onde toda a ênfase foi dada na máxima utilização das facilidades oferecidas pelo sistema operacional SOM, (Sistema Operacional Monoprogramável), (COBRA[1]) disponível nos equipamentos em questão.

Um sistema interativo (BROWN[2]; MARTINS[3]; REES e OPPENHEIMER[4]) é aquele que interage com o usuário. Os comandos são enviados pelo teclado e quando o sistema detecta um erro, imediatamente indica esta ocorrência. Assim após a correção, a execução continua. Nos últimos tempos, o crescente uso de sistemas do tipo interativo contribuiu para o desenvolvimento de uma nova ferramenta empregada no processo de compilação. Essa ferramenta, utilizada nos chamados "compiladores incrementais", permite ao usuário fazer alterações no código interno gerado para o programa fonte durante o processo de compilação. Isso representou uma melhora ao processo conhecido anteriormente que permitia corrigir os erros somente depois que todo o programa fonte fosse analisado. Um compilador incremental analisa uma linha do programa fonte, se a linha estiver correta , o código intermediário gerado é adicionado ao programa interno como sendo um incremento; se a linha estiver errada, é ignora

da e o usuário deve dar uma nova entrada. Quando um incremento é adicionado ao programa interno, se já existir outro com o mesmo rótulo que já tenha sido analisado anteriormente, este será eliminado. Se o usuário simplesmente eliminar uma linha do programa fonte, o código intermediário correspondente será removido.

Estas idéias foram incorporadas ao SISTEMA BASIC a fim de permitir que o usuário possa, através de uma interação com a máquina, obter seus resultados em poucas execuções do sistema.

I.2 - CONSIDERAÇÕES SOBRE O PROJETO

Todo o projeto foi elaborado buscando uma utilização mínima de memória, sendo assim o SISTEMA BASIC-C foi dividido em módulos, utilizando para a sua execução o mecanismo de segmentação de programas oferecido pelo sistema operacional SOM . A parte básica denominada "raiz" é carregada na memória no início da execução do sistema, cada segmento fica armazenado em disco sendo carregado para a área de segmentação da memória no momento em que esta ação tenha sido comandada na raiz ou dentro de outro segmento anteriormente carregado.

De acordo com este esquema, o tamanho de memória necessário à execução do SISTEMA BASIC-C se reduz ao tamanho da raiz somado ao tamanho dos segmentos não paralelos.

Esta solução, apesar de elevar o tempo de processamento, é adequada quando se está diante de uma limitação de memória, o que acontece quando se trata de microcomputadores.

A Figura 1 apresenta a estrutura de segmentação utilizada, esquematizada através de uma árvore.

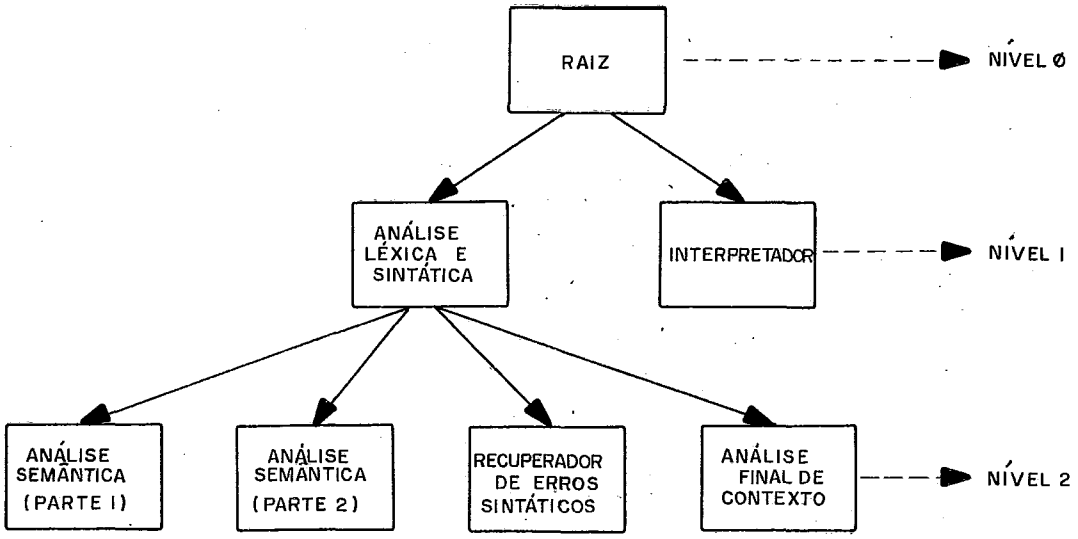


FIGURA 1 - ESQUEMA DE SEGMENTAÇÃO DO SISTEMA BASIC-C

Na Figura 1, os módulos de mesmo nível são denominados segmentos paralelos, sendo que apenas um destes módulos está na memória num dado instante. As ramificações que partem de um nó representam os segmentos ativados por este.

O SISTEMA BASIC-C foi implementado utilizando a linguagem LPS (Linguagem para Programação de Sistemas) - (COBRA[5]), disponível nos equipamentos COBRA, o mesmo oferece aos usuários as seguintes facilidades:

- Compilação por meio interativo, onde na ocorrência de um erro, o sistema pede nova entrada da linha que está sendo analisada.

- Compilação por meio lote, "BATCH", onde todo o programa fonte é montado previamente num arquivo em disco.

- Montagem de um arquivo com o código intermediário gerado pelo sistema. Mediante esta facilidade, o usuário poderá, no futuro, apenas interpretar este arquivo, sem ter que passar novamente pelos módulos de análise.

O usuário determina o modo de execução do sistema, através de parâmetros passado ao SISTEMA BASIC-C, quando da sua ativação.

A Figura 2 apresenta o esquema geral do projeto, evi

denciando suas entradas e saídas.

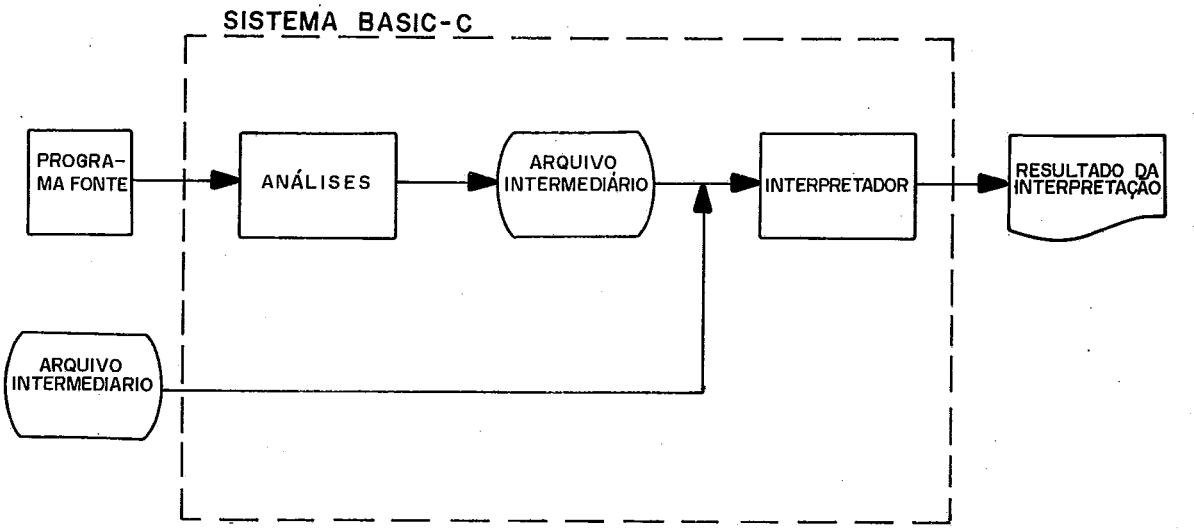


FIGURA 2 - ESQUEMA GERAL DE EXECUÇÃO DO SISTEMA BASIC-C

CAPÍTULO II

ESPECIFICAÇÃO DA LINGUAGEM

A linguagem implementada neste trabalho é um subconjunto do BASIC-PLUS-2 (DIGITAL[6]), disponível no sistema PDP 11 da "Digital Equipment Corporation".

II.2 - NOTAÇÃO, TERMINOLOGIA E VOCABULÁRIO

A notação usada para apresentar a linguagem é o diagrama de sintaxe (KOWALTWSKI[7]) .

As figuras ( , ) envolvem os símbolos terminais da linguagem.

Os elementos sintáticos que são os não terminais , (tais como expressão, comando,etc) estão nas figuras retangulares.

A seqüência no qual devem estar dispostos os componentes da linguagem é dada por uma semi-reta orientada(); as repetições são indicadas pelo símbolo  . O símbolo  indica as alternativas.

A linguagem emprega um vocabulário de palavras e símbolos reservados que não podem ser usados no programa para outros propósitos que não sejam os descritos pela sintaxe.

Apresentamos abaixo os elementos básicos da linguagem BASIC-C na notação BNF (BACKUS - NAUR - FORM), (NAUR[8]) .

<LETRA> :: = A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|
T|U|V|W|X|Y|Z

<DÍGITO> :: = 0|1|2|3|4|5|6|7|8|9

<SÍMBOLO> :: = +|-|*|.|=|&|;|,|!|"|<|>|<>|>=|<=|**|
$|%|(|)|/|##

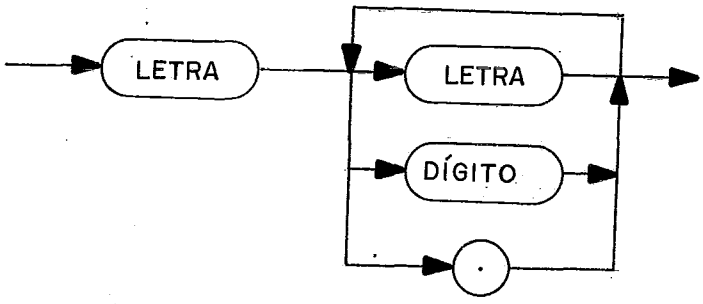
<PALAVRAS RESERVADAS> :: = AND|DIMENSION|DATA|DIM|
DEF|ELSE|END|FNEND|FOR|
GOSUB|GOTO|INTERPRET|
INPUT|IF|LET|NEXT|NOT|

ON|OR|PRINT|RESTORE|
REMARK|RETURN|READ|REM|
STEP|STOP|THEN|TO|UNTIL|
WHILE

II.2 - IDENTIFICADORES E CONSTANTES

Os identificadores servem para nomear constantes, variáveis e funções. Como os identificadores não são declarados explicitamente no programa, então sua ocorrência é tomada como uma declaração.

II.2.1 - IDENTIFICADOR



Um caractere específico colocado no fim de um identificador denota o seu tipo.

Para denotar que um identificador é do tipo inteiro, o último caractere deverá ser "%". Um caractere "$" no fim da cadeia do identificador denota um identificador do tipo alfanumérico. Os identificadores que não terminam com estes caracteres, são considerados do tipo real.

Exemplo: PONTEIRO% - identificador do tipo inteiro
LISTA$ - identificador do tipo alfanumérico
VALOR - identificador do tipo real

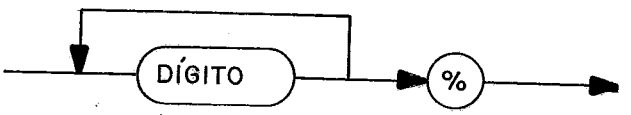
II.2.2 - IDENTIFICADOR DE FUNÇÃO

Todo identificador de função deve ser iniciado pelos caracteres "FND", podendo ter no fim da cadeia o caracter " % " denotando um identificador de função do tipo inteiro.

Exemplo: FNDCOSSENO - denota uma função do tipo real
FNDFATORIAL% - denota uma função do tipo inteiro

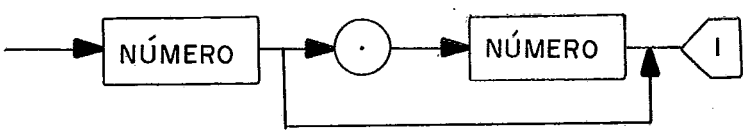
Na linguagem BASIC-C, um identificador pode ter no máximo 30 caracteres. Se este limite for ultrapassado, serão considerados os 29 primeiros concatenados com o último caractere da cadeia.

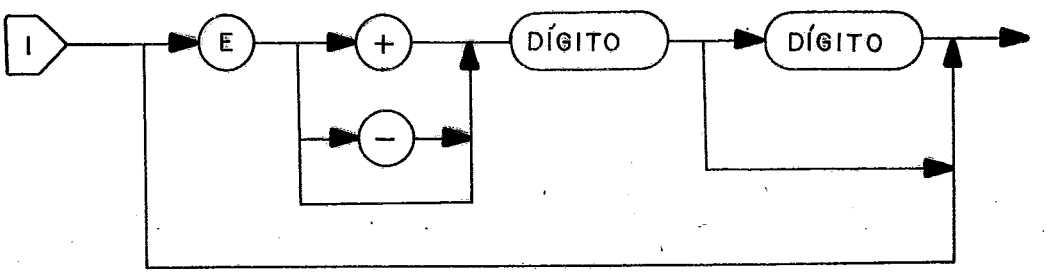
II.2.3 - INTEIRO SEM SINAL



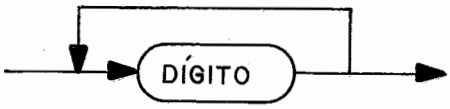
Os inteiros são avaliados e armazenados em palavras de 16 bits. O domínio dos inteiros é (-32768 ..32767)

II.2.4 - REAL SEM SINAL





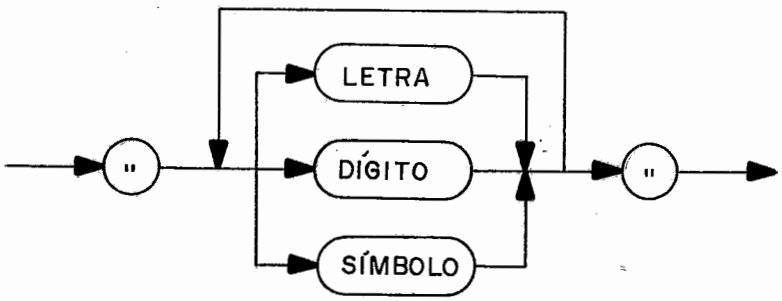
<NÚMERO>



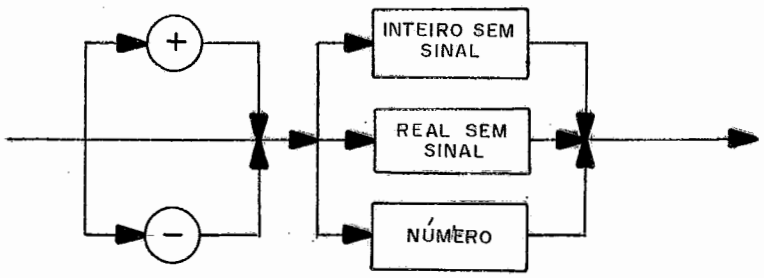
II.2.5 - CONSTANTE ALFANUMÉRICA

Uma constante alfanumérica é uma série de caracteres ASCII (letras, dígitos, símbolos), delimitados pelo caractere (") , este caractere não pode pertencer à constante, uma vez que é o símbolo delimitador.

Exemplo: "Este número não pode existir==> 2"



II.2.6 - CONSTANTE NUMÉRICA



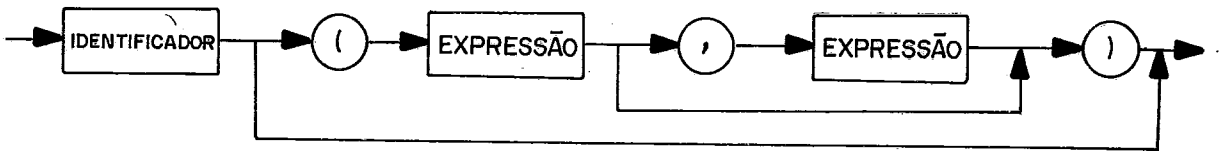
II.3 - VARIÁVEIS

Uma variável simples é referenciada pelo seu identificador.

A variável indexada é referenciada pelo identificador de arranjo, seguido das expressões de índice, entre parênteses.

A linguagem Basic-C permite a utilização de variáveis indexadas de uma ou duas dimensões. Se utilizarmos uma variável indexada com limites 10 ou 10x10, a variável não precisará ser declarada.

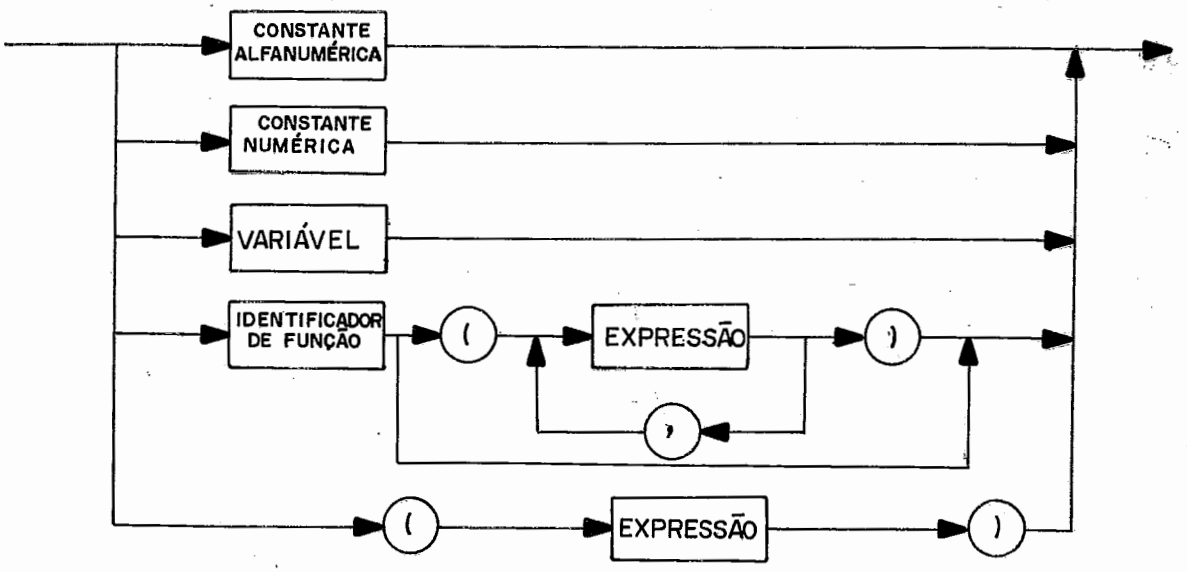
II.3.1 - VARIÁVEL



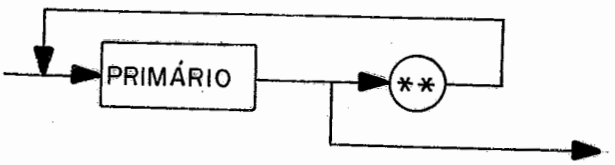
Quando uma variável alfanumérica é utilizada, reserva-se a ela um espaço de 80 bytes.

II.4 - EXPRESSÃO

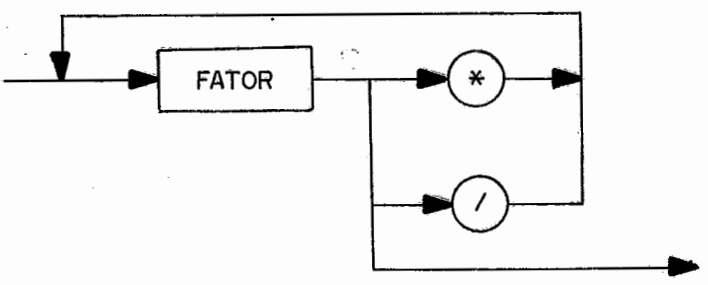
II.4.1 - PRIMÁRIO



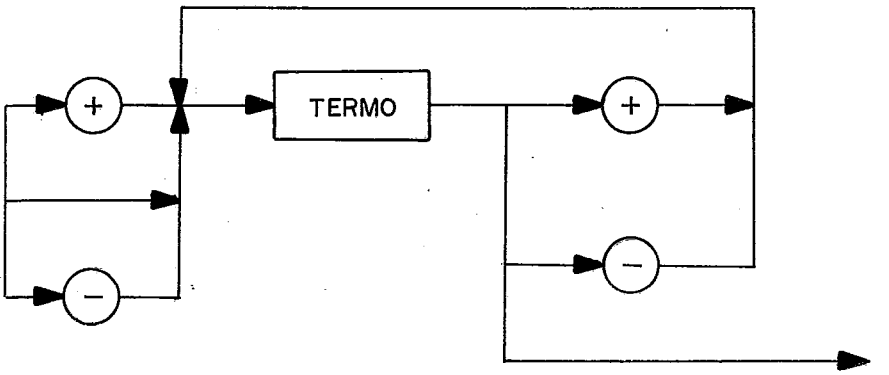
II.4.2 - FATOR



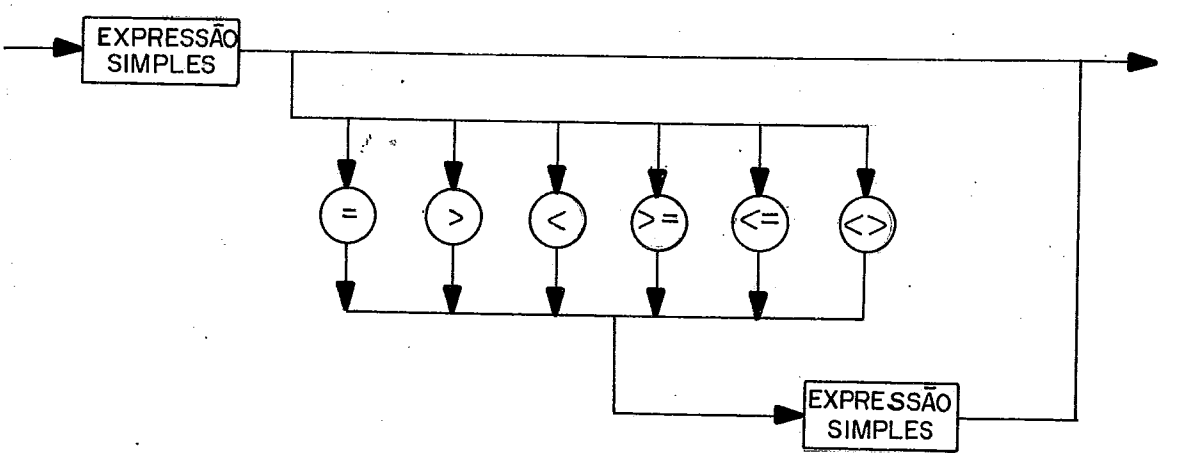
II.4.3 - TERMO



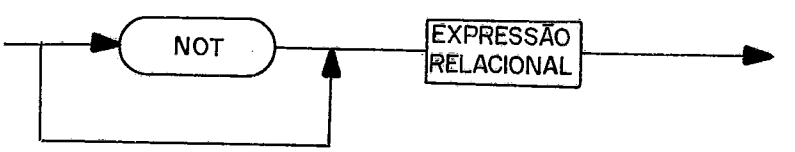
II.4.4 - EXPRESSÃO SIMPLES



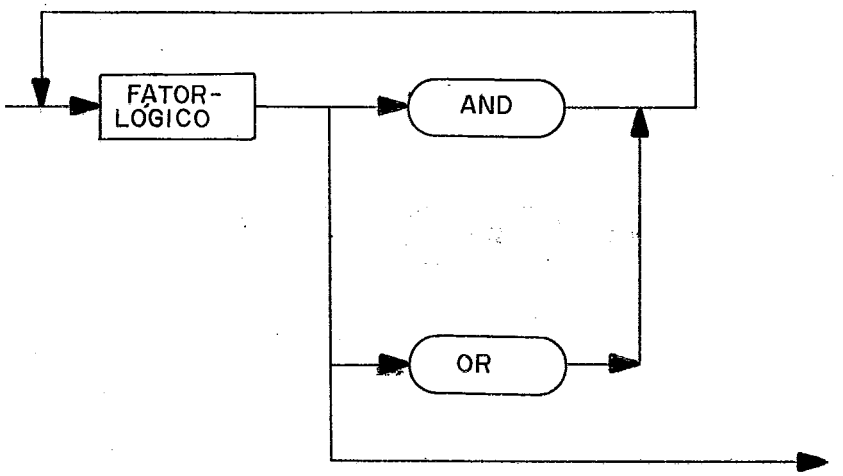
II.4.5 - EXPRESSÃO RELACIONAL



II.4.6 - FATOR LÓGICO



<EXPRESSÃO>



II.4.7 - TABELA I - PRECEDÊNCIA DOS OPERADORES

** (EXPONENCIAÇAO)	(MAIOR PRIORIDADE)
- (MENOS UNÁRIO)	
+,-	
+ (CONCATENAÇÃO)	
=,>,<,>=,<=,<>	
NOT	
AND	
OR	(MENOR PRIORIDADE)

A precedência entre operadores que estão na mesma linha da TABELA-I (operadores com igual precedência) é resolvida pela avaliação da expressão da esquerda para a direita.

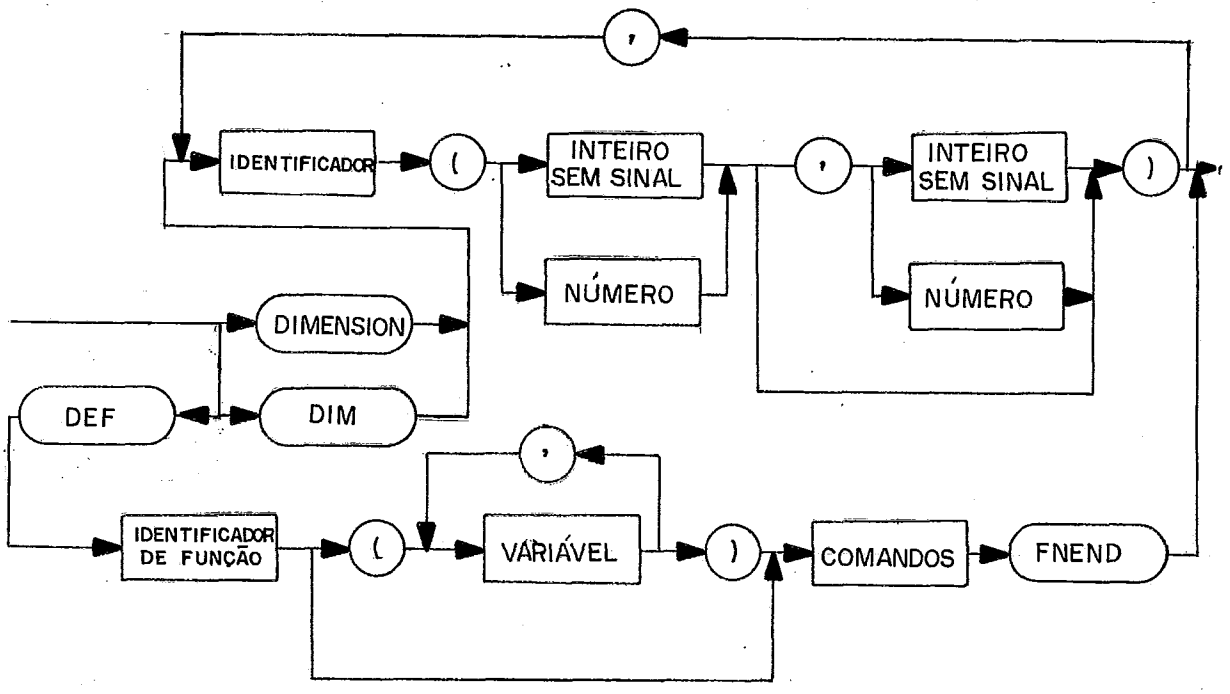
II.5 - COMANDO

Os comandos representam as ações a serem executadas no programa.

Para maior clareza, dividimos os comandos em 9 classes:

II.5.1 - COMANDOS DECLARATIVOS

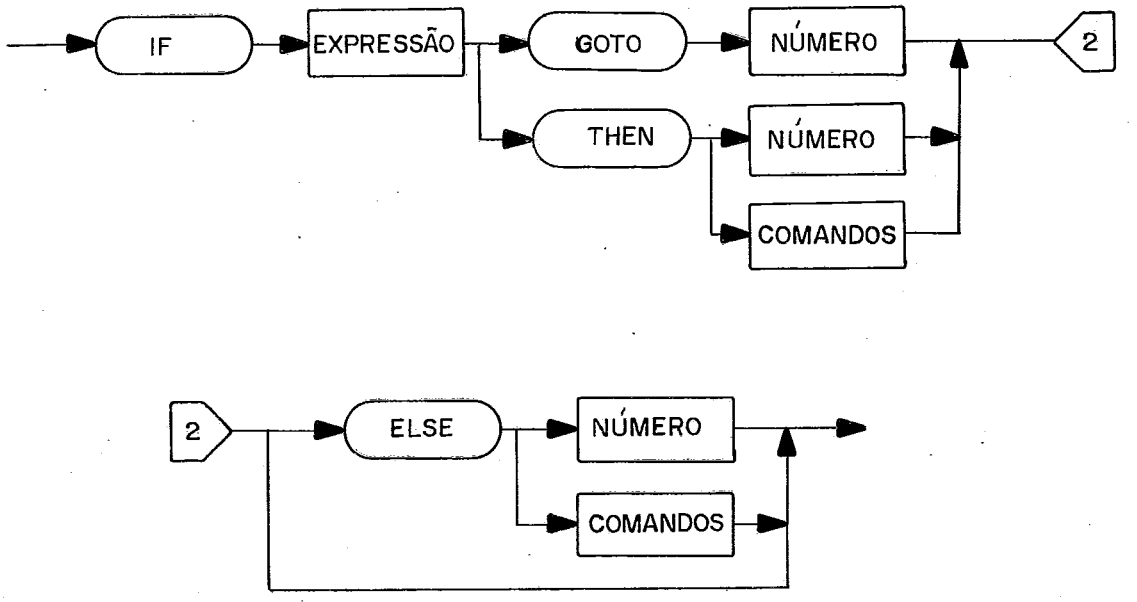
A linguagem BASIC-C possui dois comandos de declaração: o comando DIMENSION cria e dimensiona um vetor e o comando DEF que define uma função, declarando os parâmetros e ações executada.



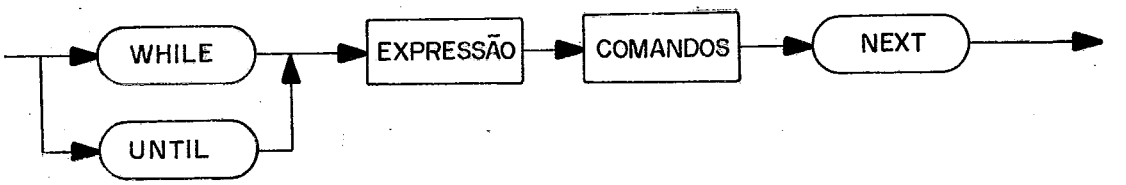
II.5.2 - COMANDOS CONDICIONAIS

Os comandos condicionais alteram a seqüência de execução do programa de acordo com o valor de uma expressão.

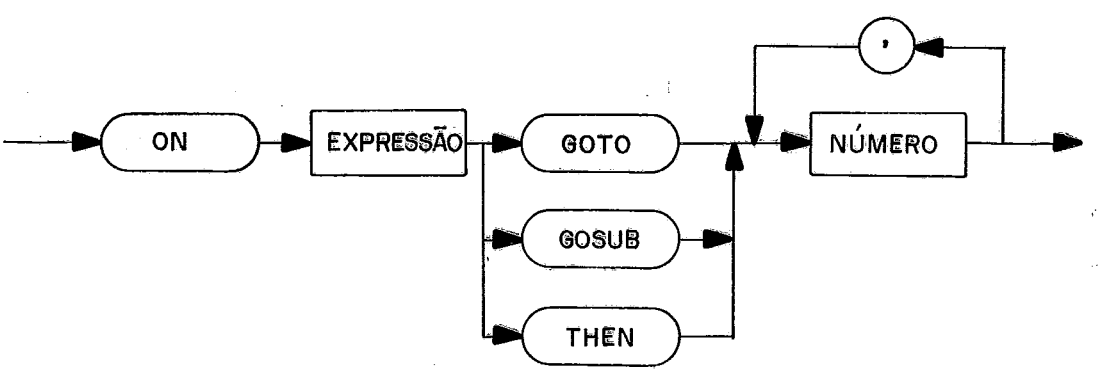
II,5,2,1 - COMANDO - IF



II,5,2,2 - COMANDO - WHILE-UNTIL



II,5,2,3 - COMANDO ON



Esse comando especifica vários números de linha como destino da transferência; a linha é selecionada de acordo com o valor da expressão. Quando este comando é executado, primeiramente a expressão é avaliada, se o valor da expressão for 1, o controle é transferido para o primeiro número de linha da lista; se o valor for 2, é transferido para o segundo número de linha da lista, e assim por diante.

Exemplo: 200 ON A GOTO 50,20,20,300

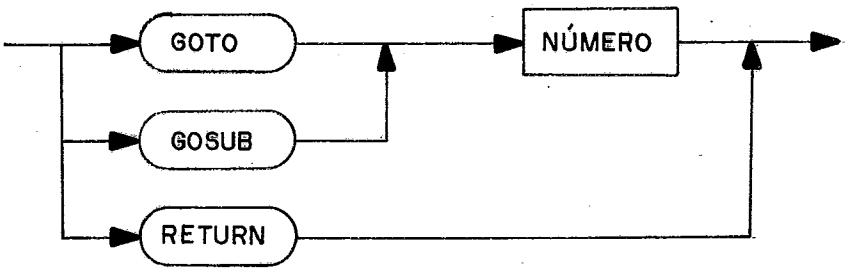
O controle será transferido para a linha:

50 , se o valor de A for 1
20 , se o valor de A for 2
20 , se o valor de A for 3
300 , se o valor de A for 4

II.5.3 - COMANDOS DE DESVIO INCONDICIONAL

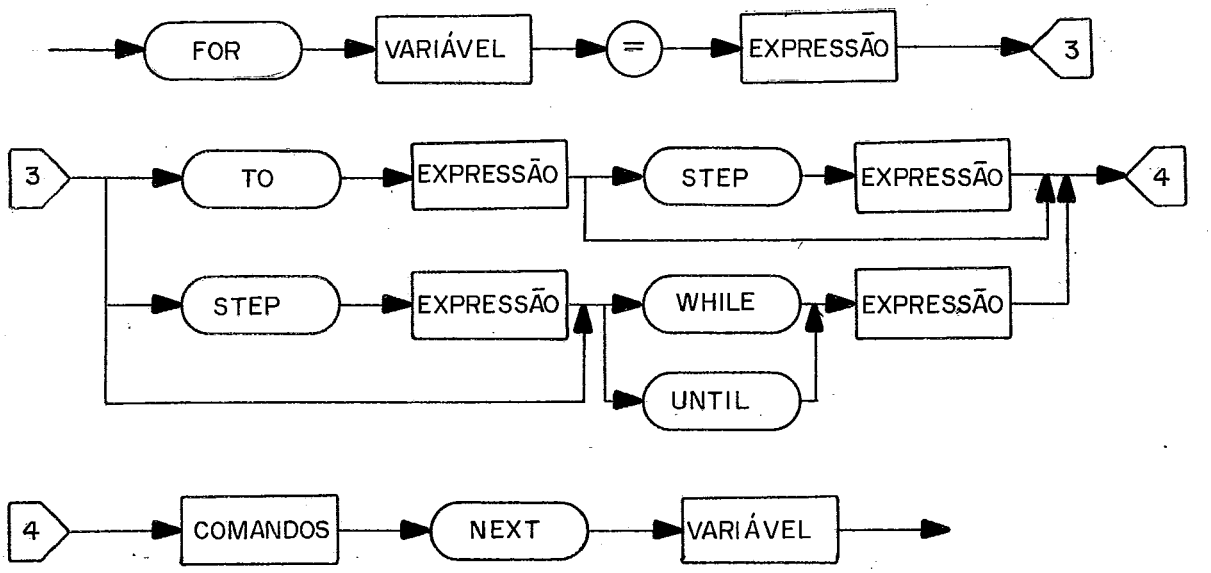
Estes comandos especificam a próxima linha do programa a ser executada.

No caso do comando GOSUB, o controle é transferido para a linha especificada como objetivo, a execução continua a partir desta linha, até que seja encontrado um comando RETURN ; quando isto acontece, o controle volta para o primeiro comando após o GOSUB.



II.5.4 - COMANDOS ITERATIVOS

Os comandos iterativos causam a execução dos comandos do ciclo até que a condição de fim seja alcançada.



Caso a parte relativa ao passo não for especificada, o passo é considerado igual a 1.

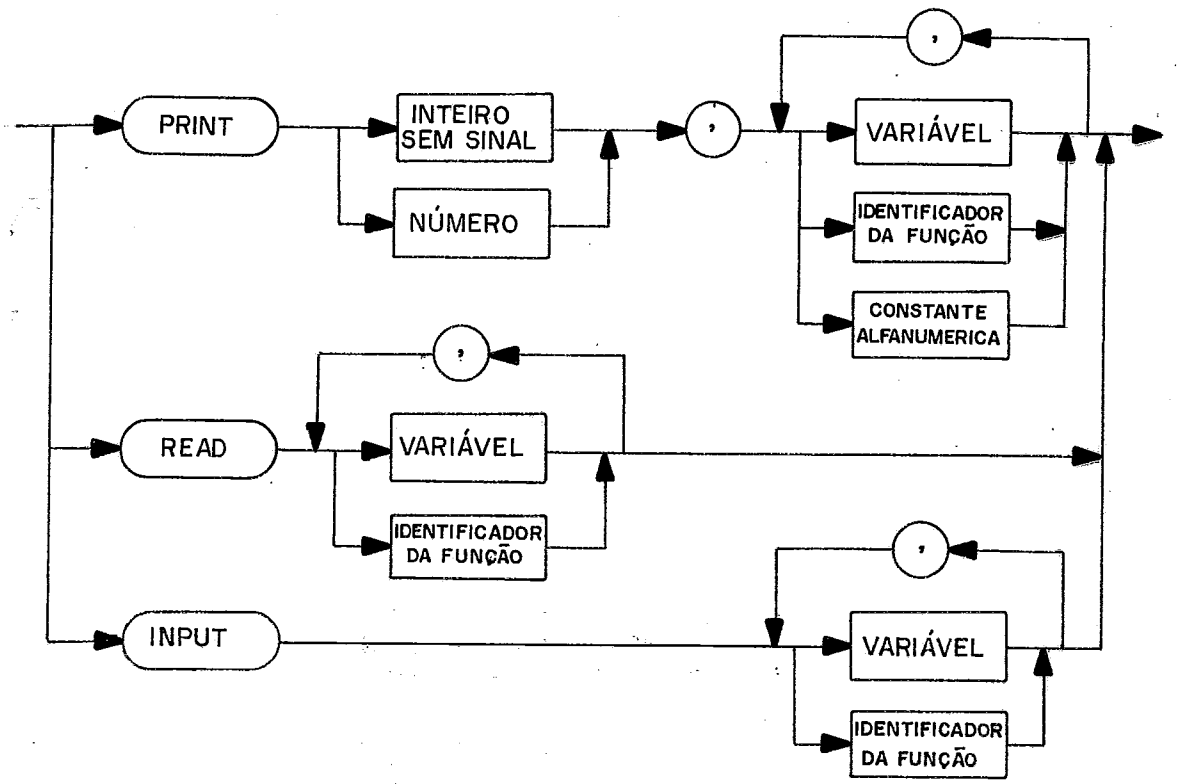
II.5.5 - COMANDOS DE ENTRADA E SAÍDA

Os comandos para entrada de dados são dois:

- COMANDO READ, que faz a entrada a partir de dados especificados no próprio corpo do programa, através do COMANDO DATA.

- COMANDO INPUT, que faz a entrada a partir do teclado, este periférico é especificado ao SISTEMA BASIC-C através da unidade lógica 8, o próprio sistema designa esta unidade lógica ao periférico.

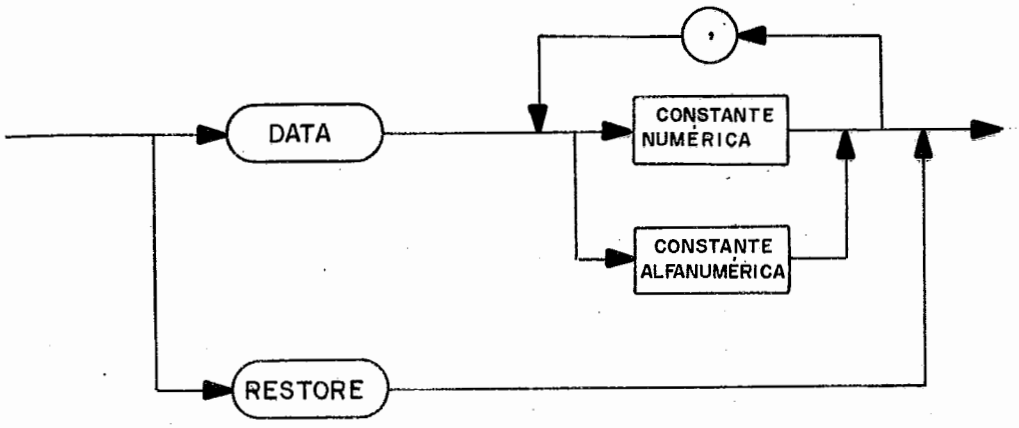
No comando de saída PRINT, o usuário especifica no comando o número da unidade lógica que deve ser previamente associada a uma das unidades físicas de saída (IMPRESSORA E VÍDEO).



II.5.6 - COMANDOS DE DADOS

Conforme menção feita no item anterior, o comando DATA evidencia os dados para o comando READ.

O comando RESTORE a cada ocorrência faz o ponteiro da área de dados, criada pelo comando DATA, apontar para o início desta área.



Exemplo:

```
10 READ B,C,D
15 PRINT 2,B,C,D
20 RESTORE
25 READ E,F,G
30 PRINT 2,E,F,G
40 DATA 6,3,4,7,9,2
50 END #
```

O comando READ (linha 10) lê os três primeiros valores do comando DATA:

```
B = 6
C = 3
D = 4
```

O comando RESTORE restaura o ponteiro para o início da área de dados (linha 20). Durante o segundo READ (linha 25), os três primeiros valores são lidos novamente:

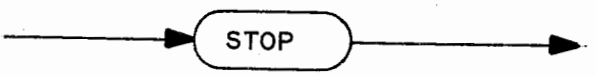
```
E = 6
F = 3
G = 4
```

Caso não fosse colocado o comando RESTORE, os valores lidos seriam:

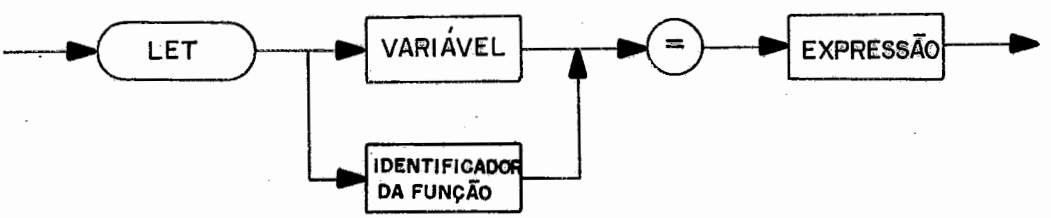
E = 7
F = 9
G = 2

II.5.7 - COMANDO DE PARADA

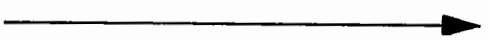
O comando STOP suspende a execução do programa.



II.5.8 - COMANDO DE ATRIBUIÇÃO

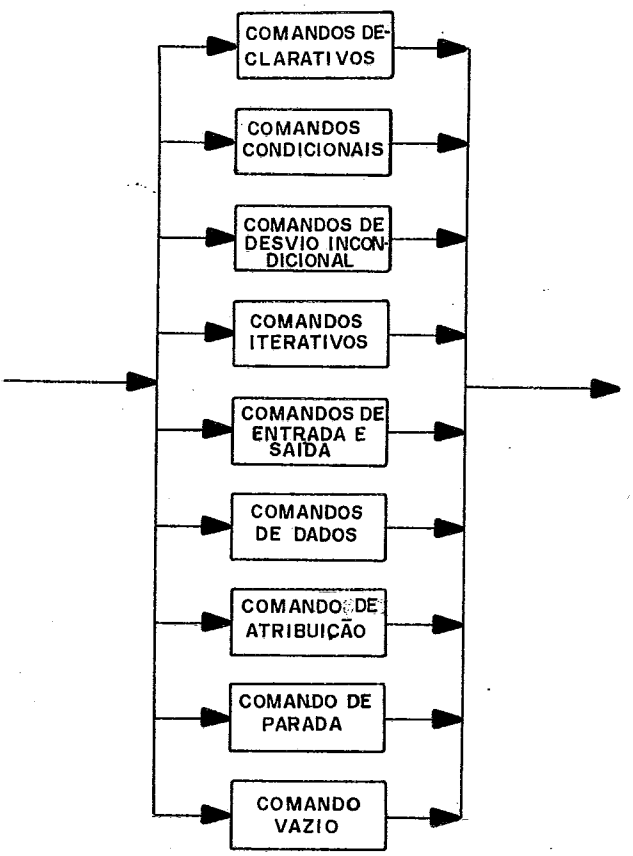


II.5.9 - COMANDO VAZIO

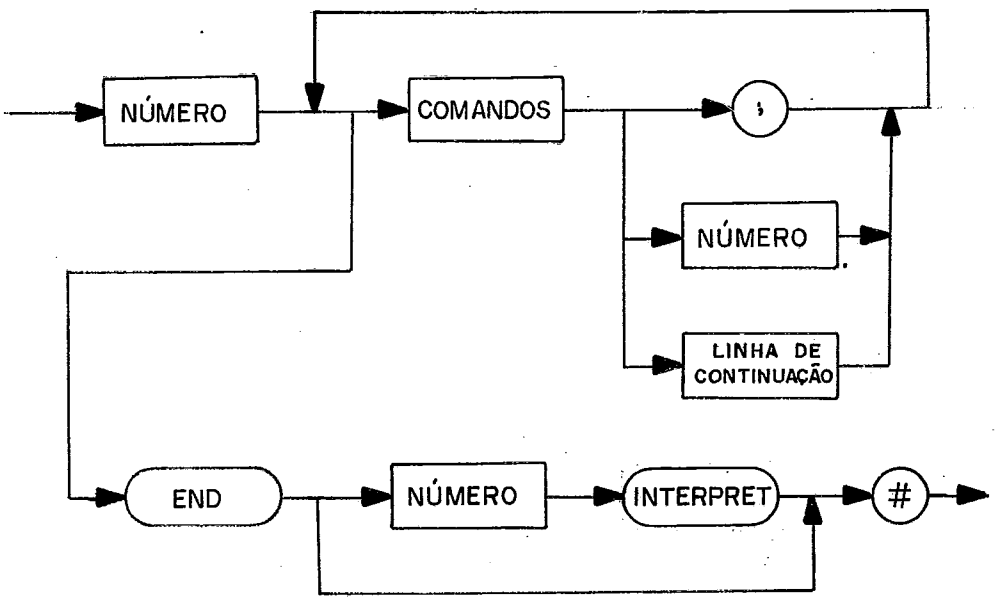


Em resumo temos:

<COMANDOS>



II.6 - PROGRAMA



A linguagem BASIC-C aceita vários comandos na mesma linha, neste caso, eles devem estar separados pelo caractere " ; " .

Se o último caractere de uma linha de comando for " & " , o primeiro caractere da linha seguinte deve ser " / " ,para que esta seja uma linha de continuação.

II.7 - CONSIDERAÇÕES FINAIS

- Cada linha de comando deve ser numerada, a não ser que seja uma linha de continuação. Tal numeração deve ser dada em ordem crescente, dando a seqüência na qual o programa deverá ser executado.

- Toda conversão de tipos, quando necessária, é feita pelo SISTEMA BASIC-C.

- Nas operações lógicas (AND, OR, NOT) , os operandos devem ser inteiros, caso contrário, será notificada a ocorrência de erro.

- O SISTEMA BASIC-C determina se uma expressão é verdadeira ou falsa, testando o resultado: -1 é verdadeira, 0 é falsa.

- O SISTEMA BASIC-C aceita qualquer valor diferente de zero como verdadeiro.

- A ação INTERPRET causa a ativação do módulo que executa a interpretação.

- Para possibilitar a troca, inserção e remoção de linhas do programa fonte, o usuário tem à disposição os comandos INSERT e ENDIN. São comandos de controle do sistema,não fazendo parte da sintaxe da linguagem BASIC-C.

Quando o comando INSERT é encontrado, o sistema salva as variáveis correntes da análise sintática e o estado da análise passa a esperar o início de uma linha de comando. Assim, o primeiro símbolo que deve aparecer após este comando é um número de linha. Desta forma, só podem ser alteradas linhas de programa que sejam numeradas.

Com o propósito de possibilitar simplificações semânticas, não se permite alterar comandos de declaração, como DIM e DEF. O comando ENDIN causa o término do estado de alteração ,

restaurando o analisador sintático para o estado anterior à aparição do comando INSERT. Como estes comandos são de controle e não pertencem à linguagem BASIC-C, eles devem aparecer sozinhos numa linha fonte, não sendo numerados.

Exemplo:

```
 5 LET  B%  = 20%
10 LET  A%  = C%+3
20 LET  C%  = B%+A%
INSERT
 8 LET  C%  = 5
20 LET  C%  = B%+2*A%
ENDIN
30 END  #
```

Neste exemplo, o programa será executado com os seguintes comandos:

```
 5 LET B% = 20
 8 LET C% = 5
10 LET A% = C%+3
20 LET C% = B%+2*A%
30 END  #
```

CAPÍTULO III

ANALISADOR LÉXICO E ANALISADOR SINTÁTICO

O esquema geral de compilação é o clássico onde a tradução é orientada pela sintaxe. Neste enfoque, o analisador léxico é feito com a forma de um procedimento que é chamado pelo analisador sintático toda vez que desejar obter um novo símbolo de entrada. Concomitante a análise sintática, são executadas as análise semântica e a geração de código intermediário . Se for detectado um erro sintático, o analisador tenta recuperar o erro através da chamada ao procedimento recuperador de erros sintáticos, tendo em vista permitir a análise de uma parte maior do programa fonte.

III.1 - ANALISADOR LÉXICO

Para particionar o texto fonte em unidades léxicas e identificá-las visando devolver ao analisador sintático o item léxico, o analisador léxico inicialmente classifica cada caractere numa das seguintes classes:

- classe 0 , para as letras
- classe 1 , para os dígitos
- classe 2 , para os símbolos simples, utilizando para tanto a tabela SIMBSIMP, que contém o caractere relacionado com o item léxico.
- classe 3 , para os símbolos duplos, utilizando a tabela SIMBDUP, que contém os caracteres que podem dar origem a símbolos duplos.

Exemplo: SIMBDUP = (>, <, *)

símbolos duplos = (>= , <= , <>, **).

Após esta classificação, o analisador atua com quatro classes onde um item léxico é determinado a cada chamada, sendo devolvido ao analisador sintático.

III.1.1 - CLASSE DOS IDENTIFICADORES

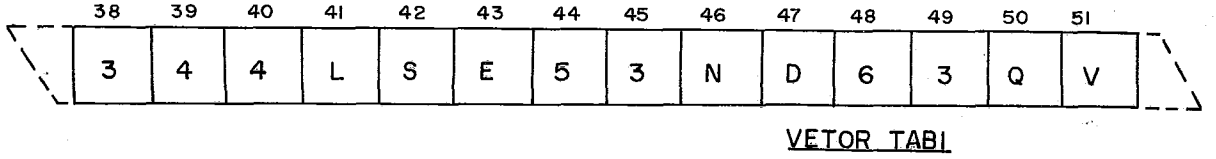
Nesta classe, o analisador após ter identificado toda a cadeia de um identificador, passa a verificar se o último caractere é um denotador de tipo (%) ou ($). Se for, é gerado o item léxico para essa entidade, caso contrário, é necessário verificar se essa cadeia é uma palavra reservada da linguagem BASIC-C.

A identificação das palavras reservadas é feita através de uma estrutura de árvore, sendo o primeiro nível representado pelo vetor VLETRA, que contém ponteiros para cada lista de palavras reservadas.

A cabeça de uma lista especifica o número de elementos da lista, as informações seguintes especificam:

- item léxico
- o tamanho do identificador, isto é, o número de caracteres que formam a palavra reservada.
- os caracteres do identificador, exceto o primeiro caractere, que é especificado no vetor VLETRA.

Exemplo de acesso:



Vamos supor que o analisador, no momento, deseja verificar se o identificador ELSE é uma palavra reservada da linguagem BASIC-C.

Através do código ASCII do caractere "E" , obtém-se o endereço 38 no vetor VLETRA, este endereço indica o início da lista em TABI de todas as palavras reservadas que iniciam com o caractere "E" .

- TABI(38) = 3 - indica que existem 3 palavras reservadas co

meçadas por "E" .

- TAB1(38+2) = 4 - indica que a maior destas palavras tem tamanho 4. Se o tamanho é igual ao da palavra procurada, como no caso, compara-se os 3 últimos caracteres. Caso o tamanho seja diferente, temos duas situações:
  1) O tamanho da palavra procurada é maior que 4, então não existe nenhuma palavra reservada que comece pela letra "E" que tenha o número de caracteres maior que 4.
  2) O tamanho é menor que 4, deve-se pesquisar a próxima palavra, se existir.

Todas as listas contidas em TAB1 são ordenadas de forma decrescente, a fim de tornar a pesquisa mais rápida.

Se depois dessa pesquisa, o analisador verificar que o identificador não é uma palavra reservada, passa a verificar se é um identificador de função, testando se os três primeiros caracteres são "FND" .

III.1.2 - CLASSE DAS CONSTANTES

Nesta classe, o analisador calcula o valor decimal da cadeia de caracteres que formam as constantes numéricas, identifica o tipo (real ou inteiro) e testa se ocorreu "overflow".

III.1.3 - CLASSE DOS SÍMBOLOS SIMPLES

Nesta classe, são analisados os símbolos simples e é gerado o item léxico relativo a cada um deles.

Quando o caractere "!" é reconhecido, o analisador está diante de um comentário, então o analisador avança até encontrar outro caractere "!" que indica fim de comentário. Todos esses símbolos são eliminados, não sendo gerado nenhum item léxico para esta entidade.

Se o caracter (") for reconhecido, o analisador identifica uma constante alfanumérica, avança até encontrar outro caractere ("), armazenando todos os demais caracteres num bloco

de dados, gera o item léxico correspondente.

III.1.4 - CLASSE DOS SÍMBOLOS DUPLOS

Nesta etapa, o analisador estando de posse de um símbolo, lê o próximo , testando se ocorre a formação de um símbolo duplo, se ocorreu, gera o item léxico correspondente ao símbolo duplo.

Todos os caracteres inválidos, isto é, não pertencentes ao alfabeto da linguagem BASIC-C são substituídos pelo analisador por caracteres "branco".

III.2 - ANALISADOR SINTÁTICO

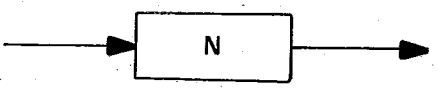
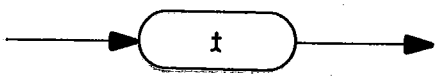
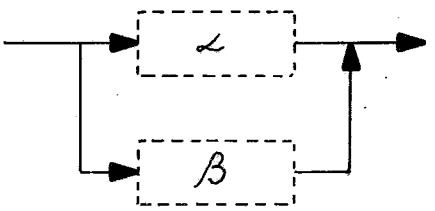
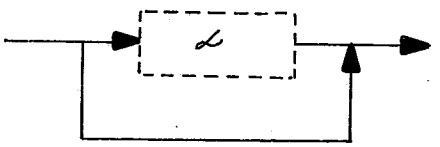
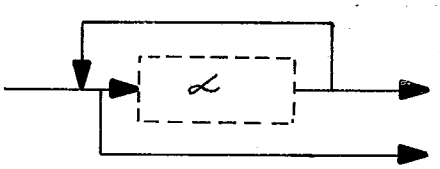
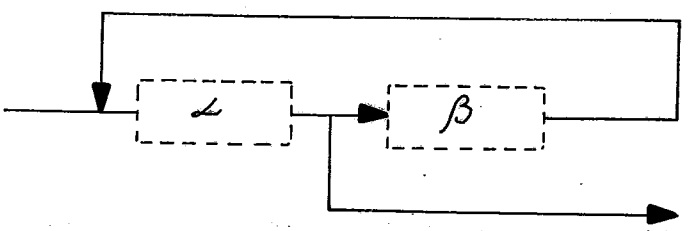
Os métodos de análise sintática podem ser classificados em ascendentes e descendentes, esses termos determinam a estratégia de análise. Nos métodos ascendentes, a árvore sintática é construída a partir da cadeia a ser analisada (folhas) até atingir o símbolo inicial da gramática (raiz); enquanto que nos métodos descendentes, ocorre o inverso, parte-se da "raiz" tendo como objetivo atingir as "folhas".

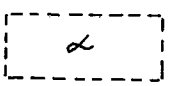
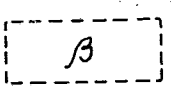
O analisador sintático do SISTEMA BASIC-C foi implementado utilizando o método das gramáticas ESLL(1) (Extended Simple L L (1)) , (SETZER e MELLO[7]) ; (SETZER[8]). Trata-se de um método descendente, simples e eficiente, tanto em velocidade de processamento como em espaço ocupado, possibilitando tratamento automático dos erros sintáticos.

III.2.1 - GRAMÁTICAS ESLL(1)

As GRAMÁTICAS ESLL(1) são uma sub-classe das ERE- gramáticas (Gramáticas das Expressões Regulares Estendidas).

Inicialmente, definiremos as ERE - Gramáticas usando grafos sintáticos. A seguir são dadas as regras para a construção do ERE - grafo.

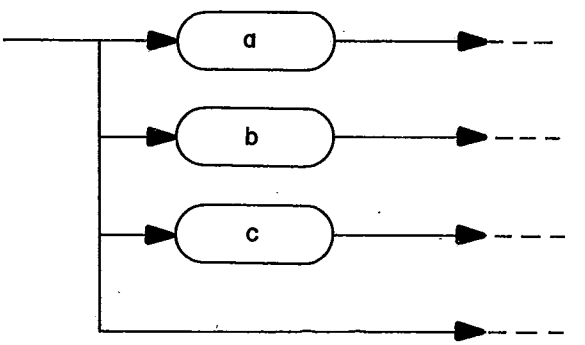
EXPRESSÃO REGULAR ESTENDIDA	ERE-GRAFO	DENOMINAÇÃO
N, não-terminal		"Nó não-terminal"
t, terminal		"Nó-terminal"
α β		
α/β		
[α]		
{α}*		
{α||β}*		

onde  e  são os grafos de α e β

As definições a seguir introduzem noções adicionais que se aplicam a esses grafos.

Definição 1 - Dado um nó m de um ERE - Grafo, dizemos que o nó n é uma alternativa de m se e somente se o arco orientado que chega a m tem bifurcação para baixo antes de m ,e n é o nó a pontado por essa bifurcação.

Por exemplo: Seja a gramática S → [a] | [b] | [c]



b é alternativa de a
c é alternativa de b

O nó "c" tem uma alternativa especial, constituída de um arco que aponta para o vazio. Essas alternativas são chamadas de λ - alternativas.

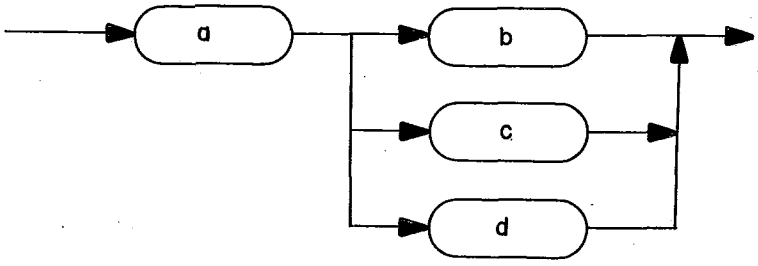
Dada a produção: S → b|λ que corresponde à produção S → [b] , no primeiro caso, temos explicitamente um λ-nó , isto é, um nó terminal contendo λ ; a alternativa de "b" é esse λ-nó. No segundo caso, "b" tem uma λ-alternativa. Em termos de implementação para cada λ-alternativa será gerado implicitamente um λ-nó, sendo as duas estruturas implementadas da mesma maneira.

Definição 2 - Se os nós n_1 , n_2 ,... n_m (m >=2) são tais que n_i é alternativa de n_{i-1} , com i = 2,3,..., m , diremos que estes nós constituem uma seqüência de alternativas. Neste caso, n_1 é

o nó inicial da seqüência e n_m não tem alternativa, podendo ter uma λ-alternativa.

Definição 3 - Dado um nó m de um grafo sintático, dizemos que o nó n é um sucessor de m se e somente se o arco que sai de m a ponta diretamente para n ; se este arco leva à uma seqüência de alternativas, o nó sucessor de n é o primeiro nó da seqüência.

Por exemplo: S → ab | ac | ad



o nó b é sucessor de a.

Partindo dessas noções, chegamos à definição de gramáticas do tipo ESLL(1): Uma ERE - gramática $G = (V_N, V_T, S, P)$ é uma gramática do tipo ESLL(1) se o grafo g de G construído a partir de P segundo as regras dadas anteriormente satisfaz as seguintes propriedades:

1) Para cada nó n_1 de g que pertence a uma seqüência de alternativas $n_1, n_2, \ldots, n_m$, $m > 1$, não existe n_i, $i = 2,3,\ldots,m$, tal que n_i é o próprio n_1.

2) Para cada não terminal $N \in V_N$, $FIRST(N) \neq 0$ onde $FIRST(N) = \{t \in V_T \mid N^* \rightarrow t\ \alpha, N \in V_N, \alpha \in (V_N \cup V_T)^*\}$.

3) Se n for um nó isolado do grafo, isto é, n não pertence a uma seqüência de alternativas e n tem uma λ-alternativa, então n deve conter $t \in V_T, t \neq \lambda$.

4) Para cada não-terminal $N \in V_N$, cada uma das seqüências de alternativas $n_1, n_2, \ldots, n_m$, $m > 1$ do sub-grafo de N deve satisfazer às seguintes condições, onde $i,j = 1,2,\ldots, m-1$:

4.1) n_i deve ser um nó terminal, isto é, n_i contém $t_i \neq \lambda$ e $t_i \neq t_j$ para $i \neq j$;

4.2) n_i não tem uma λ-alternativa;

4.3) Seja $T = \{t_1, t_2, \ldots, t_{m-1}\}$:
se n_m não tem uma λ-alternativa e

4.3.1) Se n_m contém $t_m \in V_T$ e $t_m \neq \lambda$ então $t_m \notin T$

4.3.2) Se n_m contém $M \in V_N$ e não existe geração $M \overset{*}{\rightarrow} \lambda$ então $T \cap FIRST(M) = 0$;

4.3.3) Se n_m contém λ e n_m não tem sucessor então
$T \cap FOLLOW(N) = 0$
onde $FOLLOW(x) = \{t \in V_T \mid S \overset{*}{\rightarrow} \alpha x t \beta$,
$x \in (V_N \cup V_T)$, $t \neq \lambda$, $\alpha, \beta \in (V_N \cup V_T)^* \}$;

4.3.4) Se n_m contém $M \in V_N$ e $M \overset{*}{\rightarrow} \lambda$ então $T \cap FIRST(M) = T \cap FOLLOW(N) = 0$

4.4) Se n_m tem uma λ-alternativa então n_m contém $t_m \in V_T$, $t_m \neq \lambda$, $t_m \notin T$ e $(T \cup \{t_m\}) \cap FOLLOW(N) = 0$

4.5) Se n_m tem λ , n_m não contém uma λ-alternativa e n_m tem um nó sucessor n_{m+1} então a seqüência de alternativas $n_1, n_2, \ldots, n_{m-1}, n_{m+1}$ deve satisfazer as condições 4.3, 4.4 e 4.5

A denominação ESLL(1) surgiu do fato dessas gramáticas serem extensões das gramáticas denominadas "SIMPLE-LL(1)" , que não possuem expressões regulares do lado direito das produções. Logo as restrições impostas para as gramáticas "SIMPLE-LL(1)" são válidas para as gramáticas ESLL(1), tais como: (Aho e Ullmann).

1) Para cada não-terminal N, as produções de suas alternativas devem começar com um terminal distinto.

2) Nenhuma alternativa pode constituir-se exclusivamente de λ .

III.2.2 - FUNCIONAMENTO DO ANALISADOR SINTÁTICO

Vamos mostrar o funcionamento através de um exemplo.

III.2.2.1 - EXEMPLO DE FUNCIONAMENTO

Seja a gramática G:

S → a(b | Sc) | d M

M → {f|| , }*

O grafo desta gramática é apresentado na Figura 3 , tendo-se numerado os nós para se fazer referências posteriores.

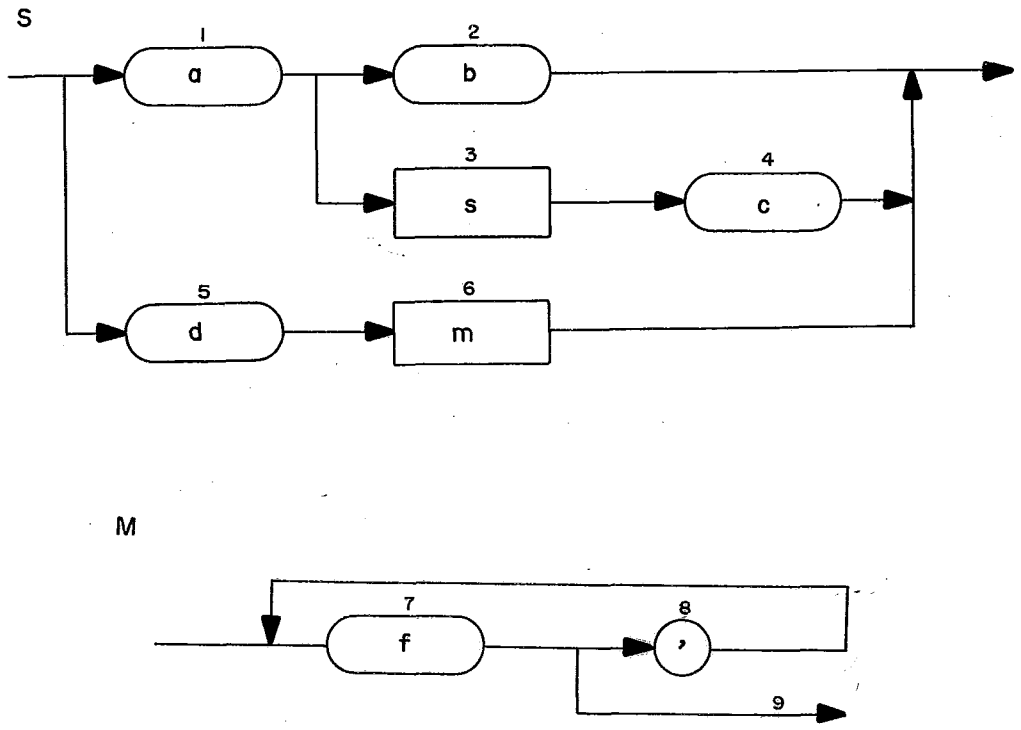


FIGURA 3 - GRAFO DA GRAMÁTICA G

Seja a cadeia de entrada:

a d f , f c

e uma pilha K, sendo TOPOK um ponteiro para esta estrutura.

O analisador léxico aqui representado por AL , sempre que chamado, lê um símbolo da cadeia de entrada, e o coloca na

variável ENT.

A variável NOGR indica o nó do grafo que está sendo analisado. Começamos a análise a partir do sub-grafo referente ao símbolo inicial da gramática (no caso "S"), tomando o nó mais acima e mais à esquerda deste sub-grafo.

SITUAÇÃO INICIAL

NOGR=1 : → primeiro nó do sub-grafo do símbolo inicial da gramática
K = 0 ; pilha vazia
TOPOK = 0;
ENT = a; → primeiro símbolo da cadeia de entrada
CADEIA= df,fc → símbolos ainda não lidos da cadeia de entrada.

ANÁLISE:

1) Conteúdo de NOGR = ENT → símbolo "a" reconhecido
chama AL → ENT = d
NOGR : = sucessor de NOGR = 2
K = 0 ;
TOPOK = 0 ;
CADEIA = f,fc

2) Conteúdo de NOGR ≠ ENT → NOGR:= alternativa NOGR =3;
ENT = d;
K = 0 ;
TOPOK = 0 ;
CADEIA:= f,fc

3) Conteúdo de NOGR = não-terminal → empilha número de NOGR em K
ENT = d ;
K = 0 3 ;
TOPOK = 1 ;
CADEIA = f,fc
NOGR:= primeiro nó do sub-grafo do não-terminal "S"=1 ;

4) Conteúdo de NOGR ≠ ENT → NOGR:= alternativa de NOGR=5;
   ENT = d;
   K = 0 3 ;
   TOPOK = 1 ;
   CADEIA = f,fc

5) Conteúdo de NOGR = ENT → símbolo "d" reconhecido
   chama AL → ENT = f
   NOGR:= sucessor de NOGR= 6;
   K = 0 3 ;
   TOPOK = 1;
   CADEIA = ,fc

6) Conteúdo de NOGR= não-terminal → empilha número de NOGR em K
   ENT = f;
   K = 0 3 6 ;
   TOPOK = 2 ;
   CADEIA = ,fc
   NOGR:= primeiro nó do sub-grafo de não-terminal "M" =7 ;

7) Conteúdo de NOGR= ENT → símbolos "f" reconhecido
   chama AL → ENT= ,
   NOGR:= sucessor de NOGR= 8 ;
   K = 0 3 6 ;
   TOPOK = 2 ;
   CADEIA = fc

8) Conteúdo de NOGR=ENT → símbolo "," reconhecido
   chama AL → ENT = f
   NOGR := sucessor de NOGR= 7;
   K = 0 3 6 ;
   TOPOK = 2 ;
   CADEIA = c

9) Conteúdo de NOGR= ENT → símbolo "f" reconhecido
   chama AL → ENT = c
   NOGR:= sucessor de NOGR = 8;
   K = 0 3 6 ;

CADEIA = VAZIA

10)Conteúdo de NOGR ≠ ENT → NOGR:= alternativa de NOGR=9;
ENT = c ;
K = 0 3 6 ;
TOPOK = 2 ;
CADEIA = VAZIA;

11)NOGR = λ-alternativa = 0 → fim do lado direito de uma produção
NOGR:= K(TOPOK), nó do topo da pilha K = 6;
NOGR:= sucessor de NOGR = 0;
ENT=c ;
K = 0 3 ;
TOPOK = 1 ;
CADEIA = VAZIA;

12)NOGR= 0 → fim do lado direito da uma produção;
NOGR:= K(TOPOK) = 3
NOGR:= sucessor de NOGR = 4
ENT = c ;
K = 0 ;
TOPOK = 0 ;
CADEIA = VAZIA;

13)Conteúdo de NOGR=ENT → símbolo "c" reconhecido
chama AL → acabou entrada
NOGR:= sucessor de NOGR = 0 ;
K = 0 ;
TOPOK = 0 ;
CADEIA = VAZIA;

14)NOGR= 0 , pilha K vazia e cadeia de entrada vazia, então fim de análise.
cadeia → a d f, fc reconhecida

III.2.3 - REPRESENTAÇÃO INTERNA DO GRAFO SINTÁTICO

O grafo sintático foi implementado na forma de quatro vetores:

1) VETOR TABGRSIM : No caso do nó representar um terminal , contém o "token" referente ao símbolo terminal. Se for um não-terminal,contém um código especial para a sua identificação; este código é usado para acessar a tabela de não-terminais.

2) VETOR TABGRALT : Contém o número do nó que é alternativa desse nó.

3) VETOR TABGRSUC : Contém o número do nó sucessor.

4) VETOR TABGRSEM : Contém o número de uma rotina semântica que será executada após o reconhecimento deste nó.

A tabela de não terminais contém o primeiro nó do sub-grafo correspondente ao não-terminal em questão.

A FIGURA 4 apresenta o grafo da gramática G (FIGURA 3) na forma de tabelas

NÓ	TABGRSIM	TABGRALT	TABGRSUC	TABGRSEM
1	a	5	2	0
2	b	3	0	0
3	S	0	4	0
4	c	0	0	0
5	d	0	6	0
6	M	0	0	0
7	f	0	8	0
8	,	9	7	0
9	λ	0	0	0

TABELA DE NÃO-TERMINAIS

S	1
M	7

III.2.4 - ALGORÍTMO DE ANALISADOR SINTÁTICO

Definições:

K → pilha que conterá o número do nó não-terminal , após ter sido desviado para o seu sub-grafo.

PS → pilha sintática, guardará todos os símbolos reconhecidos pela análise.

KR → pilha que conterá a próxima posição livre de PS.

NOGR → nó do grafo a ser analisado.

NOERRO → variável que guarda o sucessor de um nó reconhecido. É usado pelo recuperador de erro descrito no Capítulo V.

TOPO → ponteiro para as pilhas K e Kr .

TOPPS → ponteiro para a pilha PS .

CONTINUE → variável de controle .

SUCESSO → se o valor for 1 indica que a análise sintática acabou sem problemas, senão a pilha de análise está vazia mas a cadeia de entrada não está.

Com estas definições, temos o algorítmo do analisador sintático:

```
INÍCIO
      CONTINUE := 1 ;
      Lê o primeiro símbolo da cadeia de entrada;
      NOGR := primeiro nó do sub-grafo correspondente ao símbo
              lo inicial;
      TOPO:= 1 ; K(TOPO) := 0 ; KR(TOPO) := 1;
      TOPPS:= 0 ;
      ENQUANTO  CONTINUE = 1
      FAÇA INÍCIO
           SE  NOGR ≠ 0
           ENTÃO SE NOGR É UM TERMINAL
                 ENTÃO SE NOGR É UM  λ-NÓ
                       ENTÃO NOGR:= SUCESSOR DE NOGR
```

```
        SENÃO SE NOGR = SÍMBOLO DE ENTRADA
              ENTÃO INÍCIO
                    - Símbolo reconhecido
                      PS(TOPPS):= SÍMBOLO DE EN
                                  TRADA;
                      lê próximo símbolo;
                      TOPPS:=TOPPS+1 ;
                      NOGR:= SUCESSOR DE NOGR;
                      NOERRO:= NOGR;
                    FIM
              SENÃO  SE NOGR TEM ALTERNATIVA
                      ENTÃO NOGR:= ALTERNATIVA
                            DE NOGR
                      SENÃO ERRO
SENÃO - é um nó não-terminal
       INÍCIO
        TOPO:= TOPO+1 ;
        K(TOPO):= NOGR;
        KR(TOPO):= TOPPS+1;
        NOGR:= primeiro nó do sub-grafo do
               não-terminal;
       FIM
SENÃO - nó= 0 então é o fim do lado direito de
      - de uma produção, desempilha o não- ter
      - minal
      INÍCIO
       SE TOPO ≠ 0 - pilha não está vazia
       ENTÃO INÍCIO
              NOGR:= K(TOPO);
              TOPPS:= KR(TOPO);
              TOPO:= TOPO-1;
              PS(TOPPS):= CÓDIGO DO NÃO-TERMINAL
                          RECONHECIDO;
              NOGR:= SUCESSOR DE NOGR;
              NOERRO:= NOGR;
             FIM
       SENÃO  INÍCIO
```

```
                    SE ACABOU CADEIA DE ENTRADA
                    ENTÃO SUCESSO:= 1
                    SENÃO SUCESSO:= 0
                    CONTINUE:= 0
                  FIM
              FIM
FIM
```

No algorítmo acima usamos o identificador "ERRO" para indicar a chamada ao módulo de tratamento de erros sintáticos descrito no CAPÍTULO V deste documento.

CAPÍTULO IV

ANALISADOR SEMÂNTICO E GERAÇÃO DE CÓDIGO INTERMEDIÁRIO

Neste capítulo é descrito a técnica de tradução do programa fonte em código intermediário utilizada pelo SISTEMA BASIC-C, sendo apresentado o código intermediário gerado para os comandos.

IV.1 - ESQUEMA DE TRADUÇÃO DO PROGRAMA FONTE

Um interpretador denominado "puro" executa diretamente os comandos fonte.No SISTEMA BASIC-C, o módulo que faz a interpretação atua sobre uma linguagem intermediária, este mecanismo foi adotado visando facilitar, quando desejável, a construção de um gerador de código objeto; além de dar ao sistema facilidades face a possíveis alterações. Através desta técnica, foi possível guardar o código intermediário em disco, dando ao usuário a possibilidade de apenas interpretar um programa que anteriormente já tinha sido analisado, ao invés de analisá-lo novamente.

Deste modo, optou-se pela técnica de tradução dirigida pela sintaxe (AHO e ULLMAN[9]), onde as ações ou rotinas semânticas são colocadas, em pontos adequados, nas produções da gramática e, quando ativadas pelo analisador sintático,realizam a análise semântica e ao mesmo tempo geram o código intermediário.

A forma intermediária adotada é do tipo quádrupla (AHO e ULLMAN[9]), onde cada elemento tem a seguinte forma:

(OPERADOR, ARG1, ARG2, RES)

OPERADOR denota uma operação a ser realizada com os demais argumentos, ARG1 e ARG2 são endereços dos operandos e RES é o endereço do resultado da operação. Algumas operações não necessitam de todos os argumentos, assim,algumas quádruplas podem ter um ou mais argumentos vazios.

IV.2 - ESTRUTURA DE DADOS UTILIZADA

O analisador semântico utiliza a tabela de símbolos, de constantes e de temporárias para guardar informações sobre as entidades analisadas do programa fonte, com essas informações são feitas todas as verificações semânticas.

A tabela de símbolos é formada pelos seguintes vetores:

a) VETOR VTIPO : Neste vetor temos o tipo da entidade que por ser :variável inteira, real ou alfanumérica; identificador de função inteira ou real e constante alfanumérica.

b) VETOR NIVEL : Fornece o nível estático da variável.

c) VETOR PONT: Quando se refere a variáveis, contém um ponteiro para o vetor POCO, onde estão armazenadas as cadeias dos identificadores. Quando se refere a constantes alfanuméricas , contém um ponteiro para o vetor POCOCAR onde estão armazenados os caracteres que compõe essas constantes.

d) VETOR COMPR : Guarda o número de caracteres que compõem a cadeia de um identificador ou de uma constante alfanumérica.

e) VETOR INFOR : Contém informações adicionais necessárias a determinadas estruturas. Quando se trata de uma variável indexada, contém um ponteiro para o vetor DIM, onde estão armazenadas as dimensões do arranjo. No caso detratar-se de um identificador de função, contém o número da primeira quádrupla gerada para esta função.

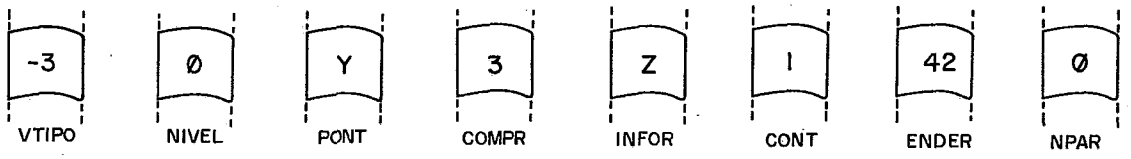
f) VETOR CONT: Guarda o número de ocorrência de cada identificador no programa fonte.

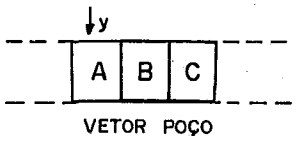
g) VETOR ENDER : Guarda o número de "bytes" que deverá ser alocado para cada estrutura.

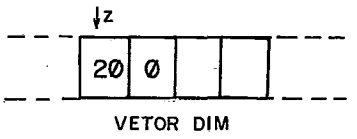
h) VETOR NPAR : Guarda o número de parâmetros, quando se trata de uma função.

Exemplo de acesso a um identificador de vetor real "ABC"

DIM ABC(20)







Como mostra o exemplo acima, o acesso é feito com o auxílio dos vetores PONT e COMPR.

A tabela de constante, assim como a tabela de temporárias, guarda como informações o tipo e o número de bytes a ser reservado para cada elemento e no caso da tabela de constantes guarda também o valor das constantes.

IV.2.1 - ESTRUTURAS UTILIZADAS PARA INSERÇÃO, REMOÇÃO E TROCA DE LINHAS

Como o SISTEMA BASIC-C permite a inserção,remoção e troca de linhas do programa fonte para as quais já tenha sido gerado o código intermediário, esse código não é sequencial, um novo campo denominado LIG é adicionado às quádruplas permitindo o encadeamento das mesmas, de modo a deixá-las na ordem correta para a execução.

Para auxiliar a tradução, é montado um descritor de informações para cada linha de comando que seja numerado. Através deste descritor, são feitas várias análises de contexto que

possibilita a remoção, inserção e troca de código.

Esse descritor está implementado como cinco vetores que contêm as seguintes informações.

a) ROTULO - guarda o número da linha de comando fonte.

b) LQUAD - Ponteiro para a primeira quádrupla gerada para o comando fonte.

c) LREF - Ponteiro para o vetor REFEREN onde é armazenado o endereço das variáveis referenciadas no comando fonte.

d) POSIREG- Número do registro do arquivo em disco onde a primeira linha do comando fonte foi armazenada.

e) NLINHA - Número de registros que o comando está ocupando no arquivo em disco.

```
Exemplo:      12    IF    A% > 2% &
                  / THEN  LET B%:= 2% &
                  / ELSE  LET C%:= 3%
```

Este comando é numerado e ocuparia três registros do arquivo em disco.

Através deste descritor, são feitas também a computação a rótulos ainda não analisados.

IV.3 - CODIGO INTERMEDIÁRIO PARA OS COMANDOS BASIC-C

Nesta parte é apresentada a tradução para alguns comandos da linguagem. A geração de código intermediário para expressões é semelhante ao método descrito por (AHO e ULLMAN[9]).

Visando facilitar o entendimento, a tradução de cada comando não é apresentada em forma de quádruplas, mas sim através de comandos relativos à quádruplas, tornando a tradução direta.

- COMANDO IF

a) IF <EXPRESSÃO> THEN <COMANDO1>
ELSE <COMANDO2>

... código para <EXPRESSÃO> ;
SE <EXPRESSÃO> = 0 ENTÃO GOTO SENÃO;
... código para <COMANDO1> ;
GOTO FIMIF;
SENÃO: ... código para <COMANDO2> ;
FIMIF:

b) IF <EXPRESSÃO> THEN <COMANDO1>
... código para <EXPRESSÃO>
SE <EXPRESSÃO> = 0 ENTÃO GOTO FIMIF;
... código para <COMANDO1> ;
FIMIF:

- COMANDO WHILE

a) WHILE <EXPRESSÃO>
<COMANDOS>
NEXT

INÍCIO: ... código para <EXPRESSÃO> ;
SE <EXPRESSÃO> = 0 ENTÃO GOTO FIMWHILE ;
... código para <COMANDOS> ;
GOTO INÍCIO
FIMWHILE:

- COMANDO UNTIL

a) UNTIL <EXPRESSÃO>
<COMANDOS>
NEXT

INÍCIO: ... código para <EXPRESSÃO> ;
SE <EXPRESSÃO> ≠ 0 ENTÃO GOTO FIMUNTIL;
... código para <COMANDOS> ;
GOTO INÍCIO
FIMUNTIL:

- COMANDO FOR

a)
```
          FOR <VARIÁVEL> = <EXPRESSÃO1> TO <EXPRESSÃO2>
          STEP <EXPRESSÃO3>
          <COMANDOS>
          NEXT <VARIÁVEL>

          <VARIÁVEL> := <EXPRESSÃO1> ;
           SINALPASSO := 0 ;
           SE <EXPRESSÃO3> >= 0 ENTÃO GOTO MAIORIGUAL;
           SINALPASSO := SINALPASSO - 1 ;
           GOTO INÍCIO;
MAIORIGUAL:SE <EXPRESSÃO3> = 0 GOTO INÍCIO;
           SINALPASSO:= SINALPASSO + 1 ;
INÍCIO:    SE (<VARIÁVEL> - <EXPRESSÃO2> * SINALPASSO) >= 0
           ENTÃO GOTO FIMFOR;
           ... código para <COMANDOS> ;
           <VARIÁVEL> := <VARIÁVEL> + <EXPRESSÃO3> ;
           GOTO INÍCIO;
FIMFOR:    <VARIÁVEL> := <VARIÁVEL > - <EXPRESSÃO3> ;
```

b)
```
          FOR <VARIÁVEL> = <EXPRESSÃO1> TO <EXPRESSÃO2>
          <COMANDOS>
          NEXT <VARIÁVEL>

          <VARIÁVEL> := <EXPRESSÃO1> ;
           SINALPASSO:= 0 ;
           SINALPASSO:= SINALPASSO + 1 ;
INÍCIO:    SE (<VARIÁVEL> - <EXPRESSÃO2> * SINALPASSO)>= 0
           ENTÃO GOTO FIMFOR;
           ... código para <COMANDOS> ;
           <VARIÁVEL> := <VARIÁVEL> + 1;
           GOTO INÍCIO
FIMFOR:    <VARIÁVEL> := <VARIÁVEL> - 1;
```

c)
```
          FOR <VARIÁVEL> = <EXPRESSÃO1 > STEP <EXPRESSÃO2>
          WHILE <EXPRESSÃO3>
          <COMANDOS>
          NEXT <VARIÁVEL>
```

```
            <VARIÁVEL> :=  <EXPRESSÃO1> ;
INÍCIO:     SE  <EXPRESSÃO3>  = 0  ENTÃO  GOTO  FIMFOR;
            ... código para <COMANDOS> ;
            <VARIÁVEL> := <VARIÁVEL> + <EXPRESSÃO2> ;
            GOTO  INÍCIO
FIMFOR:
```

d)

```
            FOR  <VARIÁVEL> = <EXPRESSÃO1>  WHILE <EXPRESSÃO2>
            <COMANDOS>
            NEXT <VARIÁVEL>

            <VARIÁVEL> := <EXPRESSÃO1> ;
INÍCIO:     SE  <EXPRESSÃO2> = 0  ENTÃO  GOTO  FIMFOR;
            ... código para <COMANDOS> ;
            <VARIÁVEL> := <VARIÁVEL> + 1 ;
            GOTO  INÍCIO ;
FIMFOR:
```

e)

```
            FOR <VARIÁVEL> = <EXPRESSÃO1>  STEP  <EXPRESSÃO2>
            UNTIL <EXPRESSÃO3>
            <COMANDOS>
            NEXT <VARIÁVEL>

           <VARIÁVEL> := <EXPRESSÃO1> ;
INÍCIO:     SE <EXPRESSÃO3> ≠ 0  ENTÃO  GOTO  FIMFOR;
            ... código para <COMANDOS> ;
            <VARIÁVEL> := <VARIÁVEL> + <EXPRESSÃO2>
            GOTO  INÍCIO
FIMFOR:
```

f)

```
            FOR  <VARIÁVEL> = <EXPRESSÃO1>  UNTIL <EXPRESSÃO2>
            <COMANDOS>
            NEXT <VARIÁVEL>

           <VARIÁVEL> := <EXPRESSÃO1> ;
INÍCIO:     SE <EXPRESSÃO2>  ≠ 0  ENTÃO  GOTO  FIMFOR;
            ... código para  <COMANDOS>
            <VARIÁVEL> := <VARIÁVEL> + 1 ;
```

```
            GOTO  INÍCIO;
FIMFOR:
```

- COMANDO ON

a)

```
ON <EXPRESSÃO>   GOTO     <RÓTULO1> , <RÓTULO2> ...,
                 THEN                 <RÓTULON>
                 GOSUB
```

```
... código para <EXPRESSÃO> ;
    CASO  <EXPRESSÃO>
    1  :  GOTO      <RÓTULO1>  ;
          GOSUB

    2  :  GOTO      <RÓTULO2> ;
          GOSUB

    :

    N  :  GOTO      <RÓTULON> ;
          GOSUB
```

- COMANDO LET

a)

```
LET  <VARIÁVEL> := <EXPRESSÃO> ;
```

```
... código para <EXPRESSÃO> ;
   <VARIÁVEL> := <EXPRESSÃO> ;
```

IV.4 - ANÁLISE FINAL DE CONTEXTO

Depois de todo o programa fonte ser analisado e as análises terminarem sem erro sintático ou semântico, é ativado o módulo que realiza a análise final de contexto.

Este módulo inicialmente percorre o descritor de linhas de comandos, verificando se não foi analisado nenhum desvio a rótulos inexistentes. Após isso, percorre o código intermediário para analisar os comandos FOR, WHILE e UNTIL; esta análise de contexto é feita para completar a geração de código pa

ra estes comandos, todos esses blocos são verificados pois poderá ter ocorrido remoção, inserção ou troca de linhas que levem o programa a uma execução errada.

Finalmente, este módulo transforma o código intermediário em código interpretável. Para isso, são coletadas informações das tabelas de símbolos, constantes e temporárias e adicionadas às quádruplas que logo após são copiadas para o disco.

CAPÍTULO V

TRATAMENTO DE ERROS SINTÁTICOS

O tratamento de erros sintáticos é parte fundamental do analisador sintático, pois seria indesejável que a análise parasse no primeiro erro detectado, não dando oportunidade ao usuário de conhecer o maior número possível de erros, podendo posteriormente corrigí-los todos de uma só vez. Portanto, um sistema de compilação deve detectar os erros e tratar a entrada de maneira a permitir que o analisador sintático verifique todo o programa fonte.

No SISTEMA BASIC-C, só existe tratamento de erros sintáticos quando o sistema está executando de modo lote (Batch) . No modo interativo, quando um erro é detectado, o sistema emite uma mensagem ao usuário e fica à espera de uma nova entrada, onde possivelmente a correção do erro indicado já tenha sido feita.

A maneira utilizada para tratar erros sintáticos foi elaborada com base no método desenvolvido por (SETZER,MELLO[7]) , onde são utilizadas três das quatro estratégias descritas nesta referência, aplicadas dentro de uma nova filosofia, de maneira combinada, como será descrito adiante.

V.1 - DETECÇÃO DE ERROS SINTÁTICOS

No algorítmo do analisador sintático, descrito no Capítulo III - item 2.4 , existem duas possibilidades de detecção de erros:

a) O símbolo inicial da gramática foi encontrado e na cadeia de entrada ainda sobram símbolos que não foram analisados. O SISTEMA BASIC-C pode testar se ocorreu esse erro, testando se é zero o valor da variável SUCESSO.

b) O nó que está sendo analisado é um terminal, mas o seu conteúdo é diferente do símbolo de entrada, não existindo alternativa para esse nó. Esta situação está demarcada, no algorítmo de análise sintática, pelo identificador ERRO.

Logo após a detecção de um erro sintático do tipo descrito no item b), o sistema emite uma mensagem, mostrando exatamente em que símbolo da cadeia de entrada ocorreu o erro, chamando em seguida o módulo recuperador de erro.

V.2 - RECUPERAÇÃO DE ERROS SINTÁTICOS

Sempre que o analisador sintático encontra um erro, deixam de ser chamadas as rotinas semânticas, portanto, deixa de ser gerado código intermediário.

A técnica usada para a recuperação de erro está fundamentada na idéia de que a maioria dos erros cometidos por usuários são decorrentes da falta de símbolos, da troca ou da aparição de símbolos a mais na cadeia de entrada.

O procedimento de análise sintática guarda o índice NOERRO, que indica o nó sucessor do último nó reconhecido antes da detecção de um erro. Este nó do grafo sintático, cujo valor está em NOERRO, é considerado como ponto de partida para o recuperador.

O funcionamento do recuperador de erros sintáticos do SISTEMA BASIC-C está fundamentado em dois procedimentos:

- Seja DELETA(i) o procedimento que remove i símbolos, a partir do símbolo t (último símbolo reconhecido pelo analisador), na cadeia de entrada.

- Seja INSERE(j) o procedimento que insere j símbolos na cadeia de entrada, a partir de t.

Deste modo, o algorítmo de recuperação de erro se apresenta do seguinte modo:

```
INÍCIO
      I:=1;
      CORREÇÃO:= 0
      ENQUANTO  (I <=5)  E  (CORREÇÃO = 0)
      FAÇA INÍCIO
            K:= I;
            J:= 0;
            ENQUANTO  (J <=I) E  (CORREÇÃO = 0)
            FAÇA INÍCIO
```

```
                    DELETA (K);
                    INSERE (J);
                    VERIFICA-SE-CORRIGIU;
                    SE  CORRIGIU
                    ENTÃO  CORREÇÃO=1
                    SENÃO  J:= J+1
                FIM
            I:=I+1;
        FIM
FIM
```

V.3 - DESCRIÇÃO DO MÉTODO

Quando o procedimento DELETA(K) é executado, supomos que o usuário cometeu o engano de escrever K símbolos a mais na cadeia de entrada, logo estes símbolos são supostamente supérfluos e assumimos como novo símbolo de entrada o símbolo K + 1 da cadeia.

A seguir, o procedimento INSERE(J) é executado, supondo que o usuário omitiu J símbolos que devem ser inseridos. A inserção de um símbolo é feita percorrendo o grafo sintático a partir do nó identificado por NOERRO. Seguindo o mesmo raciocínio da análise sintática, todos os nós aqui analisados são assumidos como símbolos inseridos . Portanto, no final temos t_1, $t_2,\ldots,t_j \in (V_N \cup V_T)$ como terminais e não-terminais inseridos na cadeia de entrada. Durante a execução deste procedimento, todas as pilhas sintáticas são atualizadas para que após o término da recuperação a análise sintática possa prosseguir normalmente.

Logo após a execução dos dois procedimentos, o recuperador verifica se o erro foi corrigido, testando se o símbolo considerado como entrada é igual ao conteúdo do nó sucessor do último símbolo inserido t_j, que denominaremos de $t_{próximo}$.

Logo, se símbolo (K+1) = $t_{próximo}$

então CORRIGIU = 1.

V.4 - RETORNO AO ANALISADOR SINTÁTICO

O recuperador de erro finaliza emitindo as mensagens da recuperação realizada; são listados os símbolos que foram retirados e os símbolos que foram inseridos na cadeia de entrada. O retorno ao analisador sintático é feito considerando que o próximo símbolo a ser analisado é K+2 e o nó do grafo, a partir de onde a análise será retomada, é o nó sucessor do nó descrito anteriormente como $t_{próximo}$.

V.5 - DIFERENÇAS BÁSICAS ENTRE O MÉTODO PROPOSTO E O MÉTODO DESENVOLVIDO POR WALDEMAR W.SETZER

O método desenvolvido por Setzer, (SETZER,MELLO[7]) aplica sequencialmente quatro estratégias buscando a recuperação do erro sintático através da eliminação, inserção e troca de um símbolo na cadeia de entrada.

Supondo como cadeia de entrada $e_1, e_2, \ldots, e_n$, e o símbolo e_i , com i <=n , como o símbolo onde um erro sintático foi detectado. Apresentamos a seguir o algorítmo, onde cada passo corresponde a aplicação de uma estratégia.

Passo 1 - Elimina o símbolo e_i - o símbolo e_{i+1} é considerado como entrada. Se não corrigiu executa Passo 2, senão termina.

Passo 2 - Insere um símbolo antes de e_i - se não corrigiu executa Passo 3, senão termina.

Passo 3 - Troca o símbolo e_i pelo símbolo e_k - se não corrigiu executa Passo 4, senão termina.

Passo 4 - Testa se e_i é um delimitador, isto é, se e_i é igual a algum dos símbolos que são sucessores dos não-terminais procurados.

Passo 5 - Caso não corrigiu, ignorar o símbolo e_i, assumir como símbolo de entrada o símbolo e_{i+1} e voltar ao Passo 1.

A cada tentativa de correção por este algorítmo ape

nas um símbolo da cadeia de entrada será eliminado, inserido ou trocado.

O método apresentado neste trabalho, foi idealizado de modo a considerar que nem sempre um só símbolo está errado em cada trecho do programa conforme considera a técnica desenvolvida por Setzer, (SETZER,MELLO[7]).

Os procedimentos DELETA(K) e INSERE(N), quando aplicados causam a eliminação de K símbolos e a inserção de n símbolos respectivamente. A aplicação destes dois procedimentos engloba três das quatro estratégias descritas anteriormente, da seguinte maneira:

1) Eliminação de um símbolo - quando aplicamos DELETA(1) e INSERE(0).

2) Inserção de um símbolo - quando aplicamos DELETA(0) e INSERE(1).

3) Substituição de símbolos - quando aplicamos DELETA(K) e INSERE(N), com $0 <= K <= 5$ e $0 <= N <= 5$.

No método aqui proposto, numa tentativa de correção, podemos inserir, retirar ou trocar de 0 a N símbolos, onde na implementação n foi assumido igual a 5. Além disso, símbolos não-terminais são inseridos permitindo muitas vezes que as mensagens da recuperação fiquem mais claras para o usuário.

CAPÍTULO VI

EXECUÇÃO DA FORMA INTERMEDIÁRIA

Um programa fonte escrito em BASIC-C, depois de convertido para forma intermediária, é executado através de um processo de interpretação, que consiste em simular o comportamento das instruções da máquina hospedeira por meio de " SOFTWARE " (PRATT[10], GRIES[11]). Através desta técnica, cada instrução intermediária é individualmente decodificada, submetida à análise e executada. A simulação das instruções e das estruturas de dados é feita por um programa escrito numa linguagem de alto nível que opera no sistema computacional hospedeiro.

VI.1 - ESTRUTURA DO INTERPRETADOR

Duas partes básicas compõem o módulo interpretador . A primeira parte é responsável pela transferência , do disco para memória principal, do código interpretável. A segunda parte é composta por um conjunto de subprogramas responsáveis pela simulação das instruções da máquina e das estruturas de dados . Essa parte, que denominamos SIMULADOR, executa as seguintes ações:

a) busca do código intermediário apontado pelo contador de programa (PC).
b) decodifica a instrução
c) executa a instrução
d) atualiza o contador de programas (PC).

VI.2 - ESTRUTURA DE DADOS

O programa interpretador, recorrendo ao uso de variáveis globais, simula a memória principal e os registradores da máquina, onde será executado o programa inicialmente escrito em BASIC-C.

A alocação de memória é feita no módulo de análise final de contexto, descrito no CAPÍTULO IV (item IV-4), de ma

neira estática; os endereços são portanto, atribuídos em tempo de compilação.

Além do código interpretável e do vetor MEMÓRIA, outras estruturas são utilizadas para realizar a interpretação.As estruturas relevantes são:

PILHARET - Armazena a quádrupla seguinte à chamada de uma subrotina (COMANDO GOSUB), ou à chamada de uma função. Quando um comando de retorno é decodificado, no caso RETURN ou FNEND, o contador de programa (PC) recebe o número da quádrupla que está no topo desta pilha. A finalidade desta estrutura é garantir que o controle de execução retorne ao programa principal, após o término de uma subrotina ou de uma função, para o comando imediatamente seguinte àquele que realizou a chamada.

POCOCAR - Armazena todas as constantes alfanuméricas encontradas no programa fonte. Esta estrutura é também utilizada para a operação de concatenação entre cadeias.

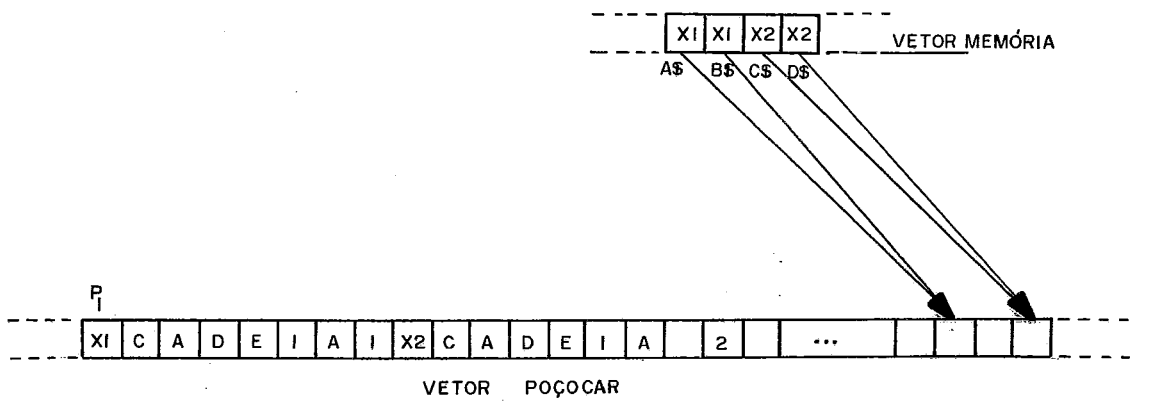
A operação de atribuição entre operandos alfanuméricos, é realizada por meio de ponteiros. O uso de ponteiros torna desnecessária a movimentação física de caracteres (MORRISSON[12]). Desse modo, além da economia de espaço na memória, obtém-se ainda uma outra vantagem, que é a redução do tempo para realizar as operações. A movimentação física de cadeias fica restrita somente para o caso em que houver operação de concatenação.

A estrutura utilizada no armazenamento das constantes alfanuméricas foi projetada com um esquema de controle dos espaços ocupados. Os espaços ocupados pelas constantes não mais ativas poderão ser reaproveitados implementando-se um procedimento para torná-los novamente disponível.

O seguinte exemplo ilustra essa estrutura e a operação de atribuição entre operandos alfanuméricos.

Exemplo:

```
LET A$ = "CADEIA1"
LET B$ = "CADEIA2"
LET C$ = B$
LET D$ = A$
```



Cada elemento do VETOR MEMÓRIA aponta para a posição do VETOR POCOCAR que contém o tamanho da cadeia, em x_{i-1} do VETOR POCOCAR tem-se um ponteiro para o início da cadeia.

O código intermediário obtido pelo processo de compilação foi projetado tendo-se em mente a construção futura de um gerador de código, em detrimento do interpretador ora apresentado. O gerador de código, ao invés de simular as instruções, faria a tradução do código intermediário para linguagem de baixo nível diretamente executável pela máquina. Em decorrência desta filosofia de projeto, foi necessário um grande número de linhas de instrução no código intermediário para criar-se ações mais próximas às instruções de máquina. O módulo interpretador em função disto, tornou-se mais lento, por ser obrigado a executar uma quantidade maior de instruções, por comando da linguagem fonte. Este problema não é muito crítico uma vez que um interpretador é normalmente usado em aprendizado e na depuração de programas onde o tempo de execução não é um ponto essencial.

CAPÍTULO VII

ATIVAÇÃO DO SISTEMA BASIC-C

A execução dos programas escritos em BASIC está a cargo do programa BASIC-C (Interpretador BASIC). Este programa é ativado através do uso de diretivas de Execução de Programas (:EX[13]), associada ao nome do interpretador (BASICC) . Essa diretiva aceita uma linha de parâmetros, cujo número e significado dependem do modo de execução do SISTEMA BASIC-C.

A sintaxe da ativação do sistema, usando a representação BNF, é a seguinte:

a) <EXECUÇÃO EM MODO LOTE - <BATCH> > ::=

:EX,BASICC, "B,<ARQUIVO-1> , <UNIDADE-DE-DISCO> ,
<ARQUIVO-2> , <UNIDADE-DE-DSICO> "

onde <ARQUIVO-1> indica o nome de um arquivo em disco, onde está o programa fonte a ser analisado; <ARQUIVO-2> o arquivo onde será guardado o código interpretável gerado para o programa fonte.

b) <EXECUÇÃO EM MODO INTERATIVO> ::=

:EX,BASICC,"I, <ARQUIVO-1> ,<UNIDADE-DE-DISCO> ,
<ARQUIVO-2> ,<UNIDADE-DE-DISCO> "

onde <ARQUIVO-1> indica o arquivo em disco onde será guardado o programa fonte do usuário, uma vez que a entrada é feita via teclado; <ARQUIVO-2> indica o arquivo intermediário descrito no item a).

c) <EXECUÇÃO DE UM PROGRAMA PREVIAMENTE ANALISADO> ::=

:EX,BASICC,"E, <ARQUIVO-1> , <UNIDADE-DE-DISCO> "

Neste caso, o sistema assume que o <ARQUIVO-1> é um arquivo intermediário gerado anteriormente pelo SISTEMA BASIC-C.

Para os três modos citados temos:

<ARQUIVOS> ::= /<LETRA> <LETRA> | <DÍGITO>

Um nome de arquivo pode ter no máximo 7 caracteres.

<UNIDADE-DE-DISCO> ::= D0 | D1 | D2 | D3

Nos casos a) e b), se o último comando do programa fonte for "INTERPRET", o sistema causa automaticamente a ativação do módulo que faz a interpretação do código intermediário. Caso contrário, o sistema cria apenas o arquivo intermediário.

VII.1 - LISTAGEM DA EXECUÇÃO

O SISTEMA BASIC-C utiliza, para a impressão, a unidade lógica 6. Assim, para que o usuário obtenha a listagem por uma unidade de saída específica, deverá associar esta unidade lógica ao periférico desejado, antes da ativação do sistema.

Os equipamentos COBRA-300 e COBRA-305, por omissão, assumem que a unidade lógica 6 está associada à impressora.

As mensagens de erro e advertências enviadas pelo SISTEMA BASIC-C quando de sua execução, estão listadas no APÊNDICE II.

CAPÍTULO VIII

CONCLUSÕES

Consideramos que este trabalho tenha atingido o seu objetivo sob o ponto de vista prático, uma vez que o SISTEMA BASIC-C está todo implementado, podendo oferecer aos usuários dos microcomputadores COBRA a linguagem BASIC como mais uma ferramenta de compilação.

Com relação à eficiência do sistema, não nos foi possível compará-lo a outro com as mesmas características, o que possivelmente será feito quando de sua utilização em aplicações práticas por entidades acadêmicas que estão interessadas na aquisição do mesmo. Esta utilização será por nós acompanhada, sempre que possível, de modo a medir a eficiência do sistema e testá-lo de maneira mais contínua.

Uma farta documentação sobre o SISTEMA BASIC-C e sobre a linguagem implementada está sendo preparada e estará disponível a qualquer usuário que tenha interesse em sua utilização.

O projeto e a implementação deste trabalho foram de grande importância para o nosso desenvolvimento profissional, uma vez que pudemos pôr em prática alguns conhecimentos teóricos e sentir através da implementação a dificuldade de se desenvolver grandes programas em equipamentos com limitação de memória e sem grandes recursos de "Software".

Para trabalhos futuros podemos sugerir os seguintes tópicos:

- Extensão da linguagem BASIC implementada, oferecendo mais facilidades aos usuários.
- Implementação de um gerador de código objeto para o sistema.
- Aprimoramento desta versão no sentido de otimizar o tempo de execução e o espaço ocupado na memória.

Esperamos que o presente trabalho venha a ser uma contribuição válida como um "software" nacional desenvolvido para um equipamento nacional.

REFERÊNCIAS BIBLIOGRÁFICAS

1. COBRA COMPUTADORES E SISTEMAS BRASILEIROS S/A - COBRA-300 Manual de referência do Sistema Operacional monoprogramável - SOM, Rio de Janeiro, 1980.

2. BROWN, P.J. - Writing Interactive Compilers and Interpreters, John Wiley & Sons, 1979.

3. MARTINS, E. - Um Compilador Basic Incremental para o terminal inteligente, Dissertação de Mestrado em Ciênicas , COPPE-UFRJ, 1982.

4. REES, M.J. ; OPPENHEIMER,A.W. - SOBS - An Incremental Basic System - Software - Practice and Experience, Vol.7: 631-643, 1977.

5. COBRA COMPUTADORES E SISTEMAS BRASILEIROS S/A - COBRA-300 Manual de programação LPS, Rio de Janeiro, 1980.

6. DIGITAL EQUIPMENT CORPORATION - PDP-11 BASIC-PLUS-2(V1.6) - Language Reference Manual, Maynard, MA, 1979.

7. SETZER,V.M. e MELLO,I.S.H. - A Construção de um Compilador, Editora Campus Ltda, Rio de Janeiro, 1983.

8. SETZER,V.M. - Non recursive Top-down syntax analysis - Software - Practice and Experience, Vol.7:237-245,1979.

9. AHO,A.V.; ULLMANN,J.D. - Principles of Compiler Design, Addison - Wesley, 1978.

10. PRATT,T.W. - Programing Language design and implementation, Prentice . Hall, Inc., 1975.

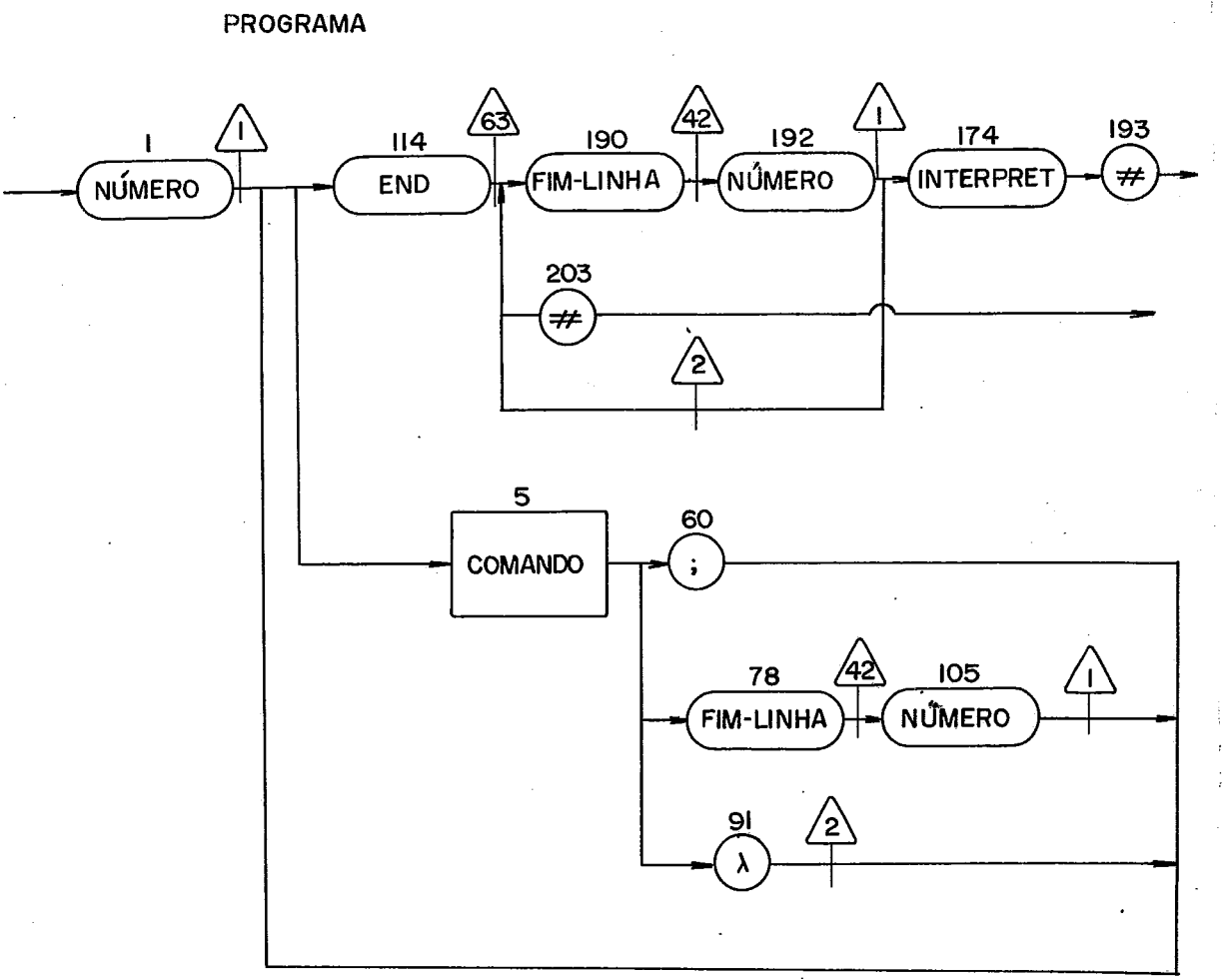
11. GRIES,D. - Compiler Construction for Digital Computers,John

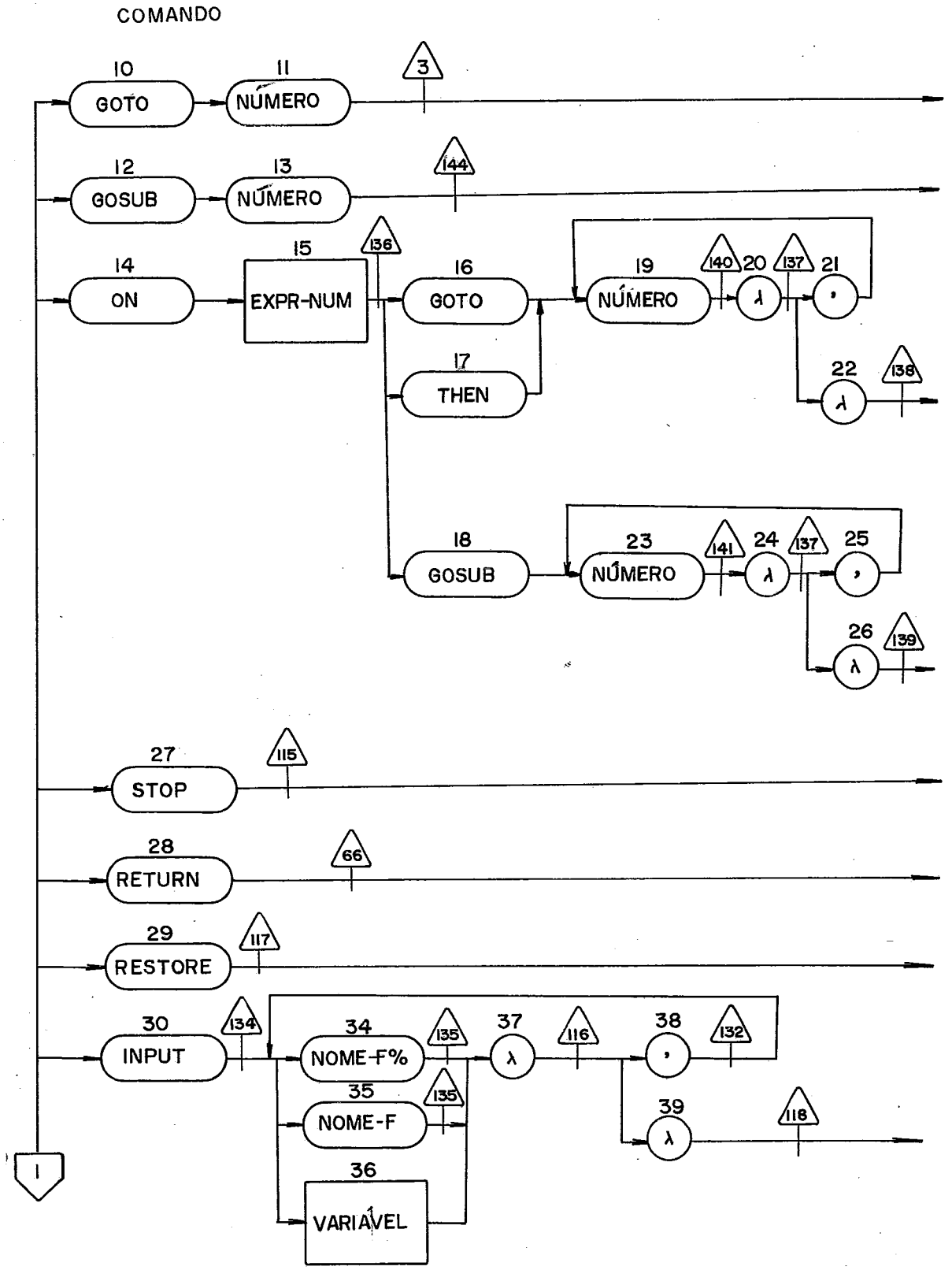
Wiley & Sons, Inc., 1971.

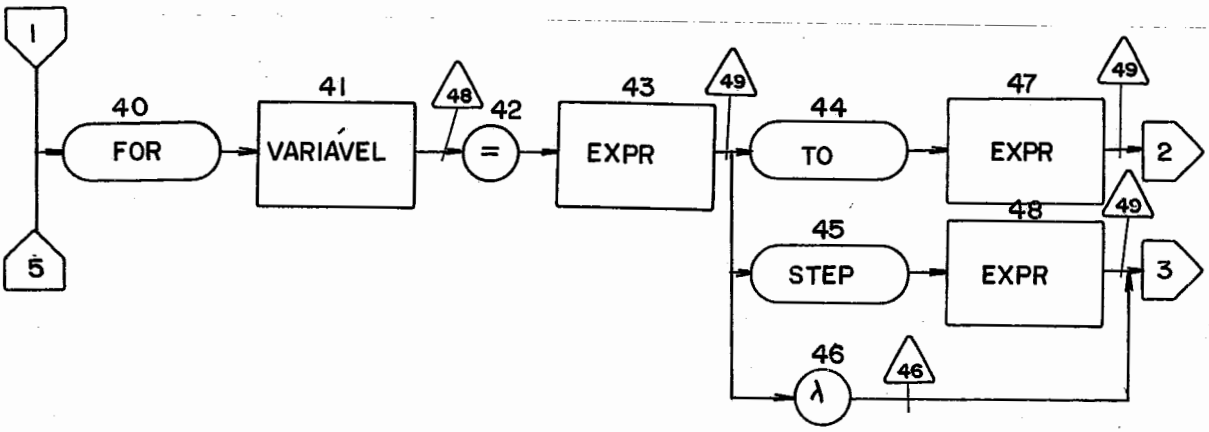
12. MORRISSON, R. - The string as a simple data type - Sigplan Notices , Vol.17, Nº3, 46-52, 1982.

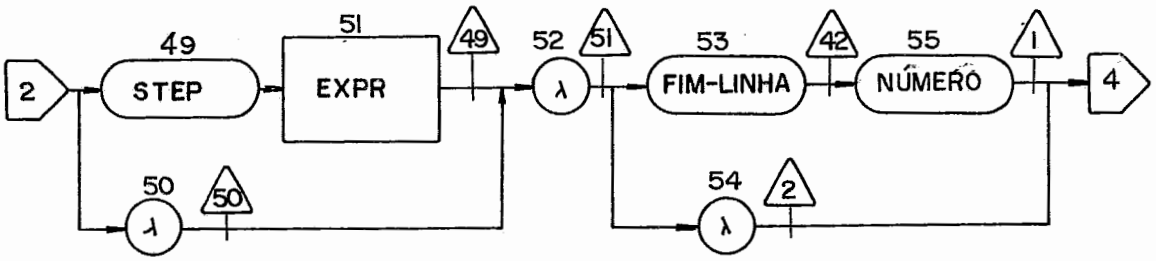
APÊNDICE I

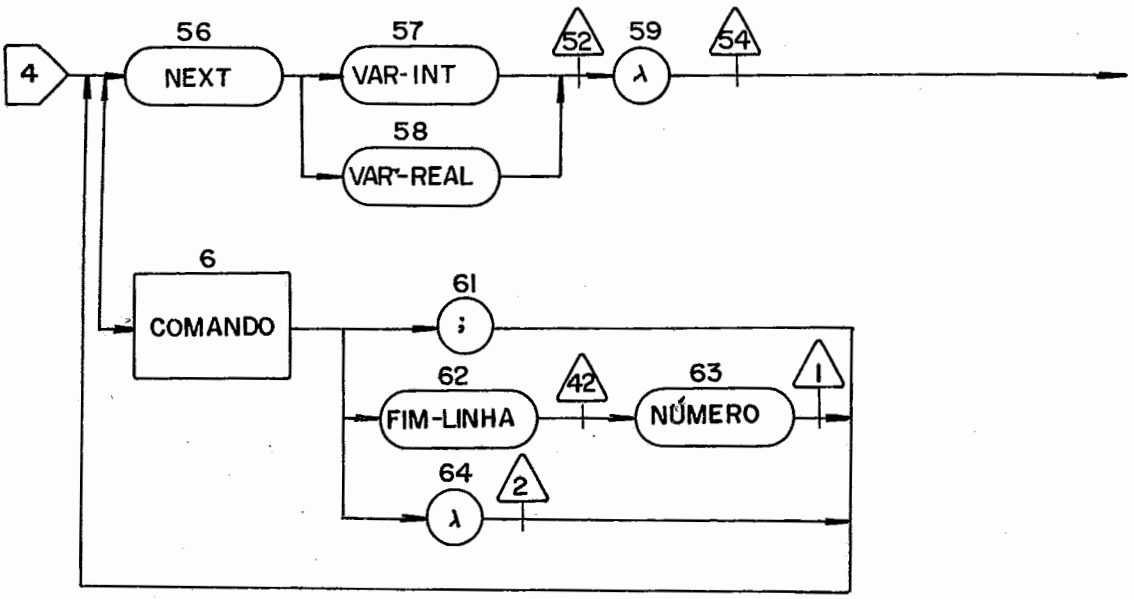
ERE-GRAFO DA LINGUAGEM BASIC-C

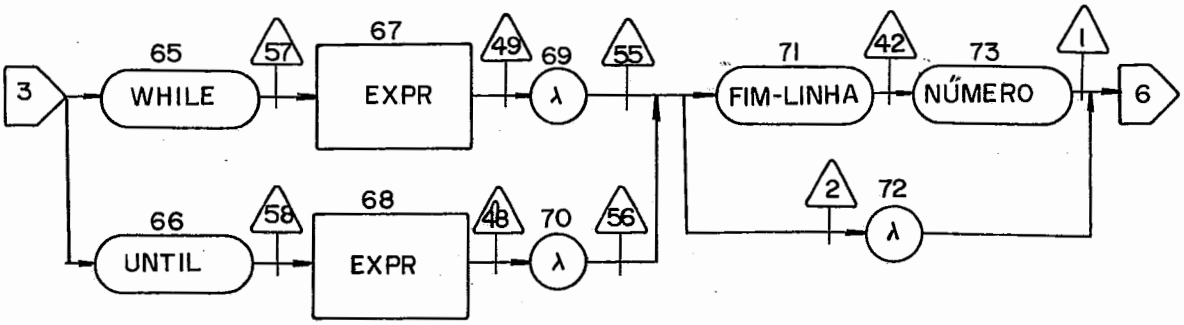


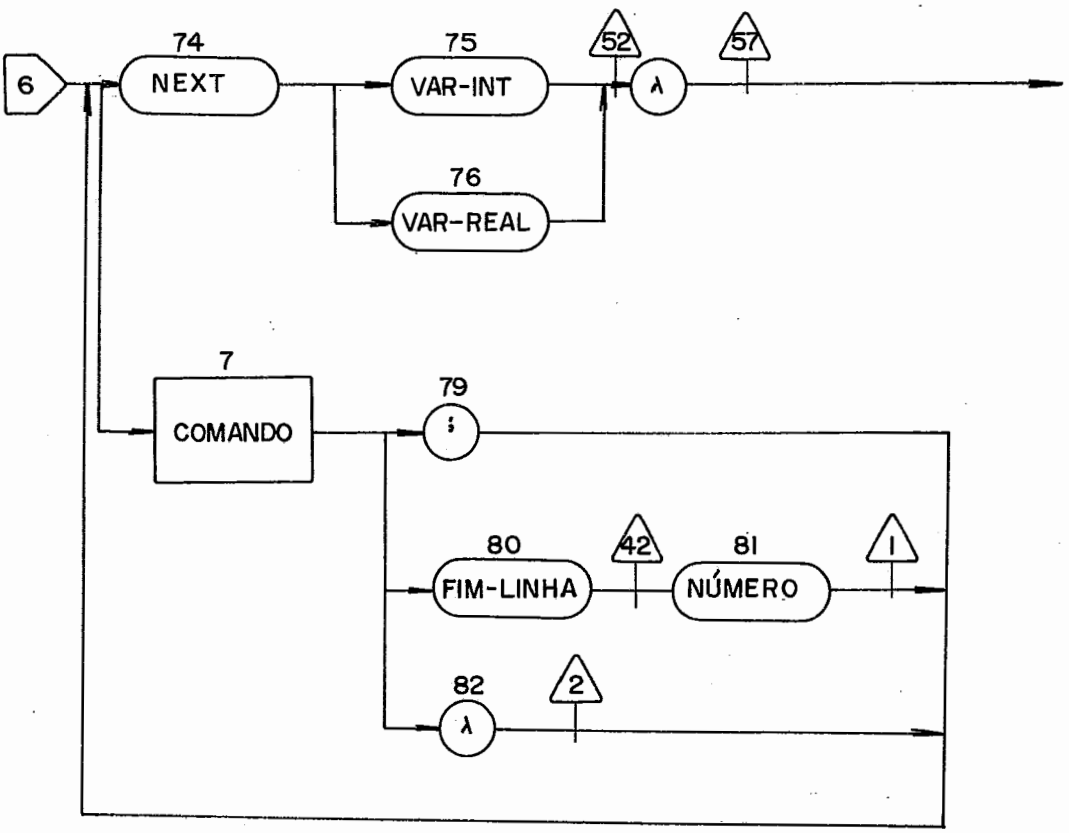

COMANDO
10
GOTO
11
NÚMERO
3
12
GOSUB
13
NÚMERO
144
14
ON
15
EXPR-NUM
136
16
GOTO
17
THEN
19
NÚMERO
140
20
λ
137
21
,
22
λ
138
18
GOSUB
23
NÚMERO
141
24
λ
137
25
,
26
λ
139
27
STOP
115
28
RETURN
66
29
RESTORE
117
30
INPUT
134
34
NOME-F%
135
35
NOME-F
135
36
VARIÁVEL
37
λ
116
38
,
132
39
λ
118
1

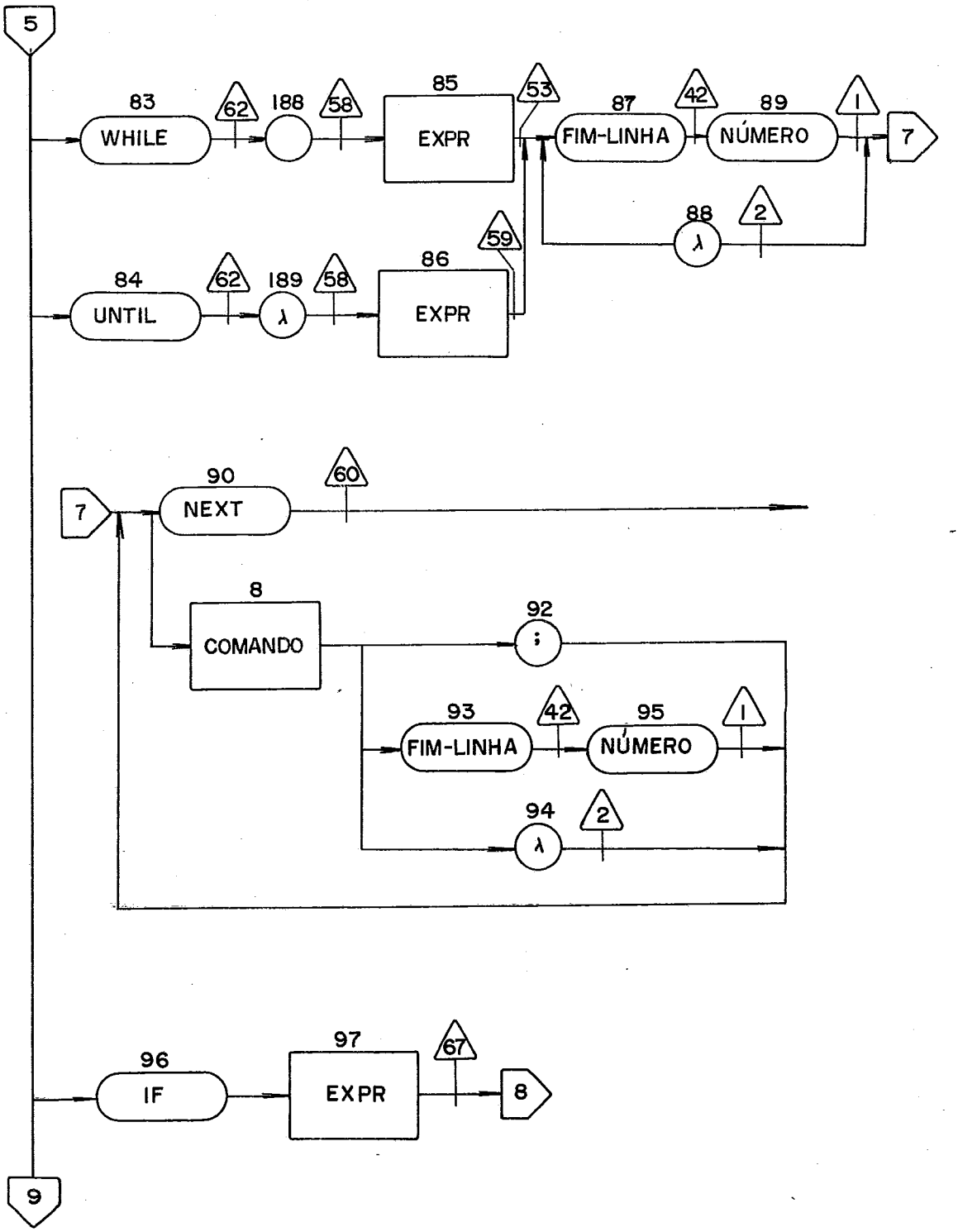

1
5
40
FOR
41
VARIÁVEL
48
42
=
43
EXPR
49
44
TO
47
EXPR
49
2
45
STEP
48
EXPR
49
3
46
λ
46

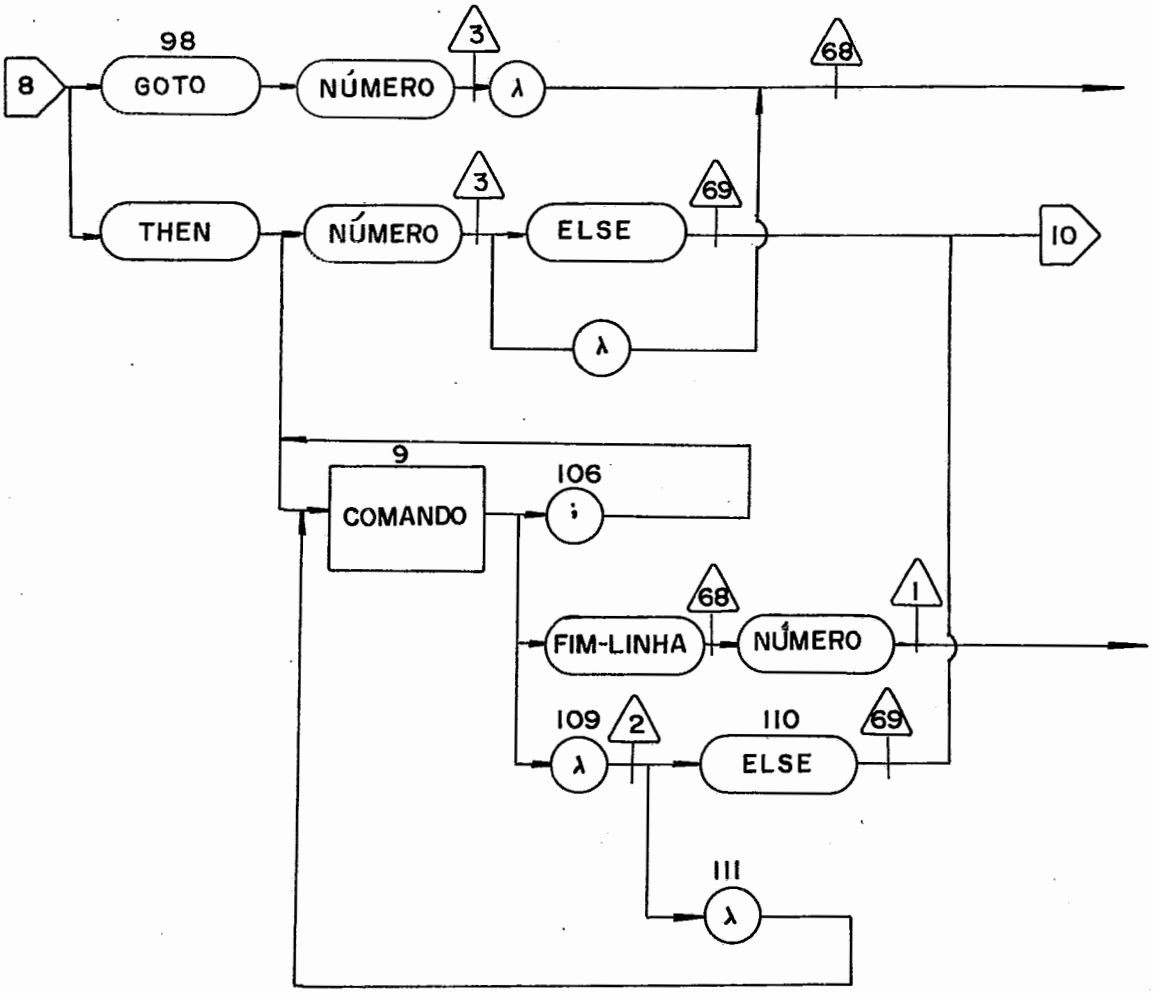

2
49
STEP
51
EXPR
49
52
51
λ
53
FIM-LINHA
42
55
NÚMERO
1
4
50
λ
50
54
λ
2

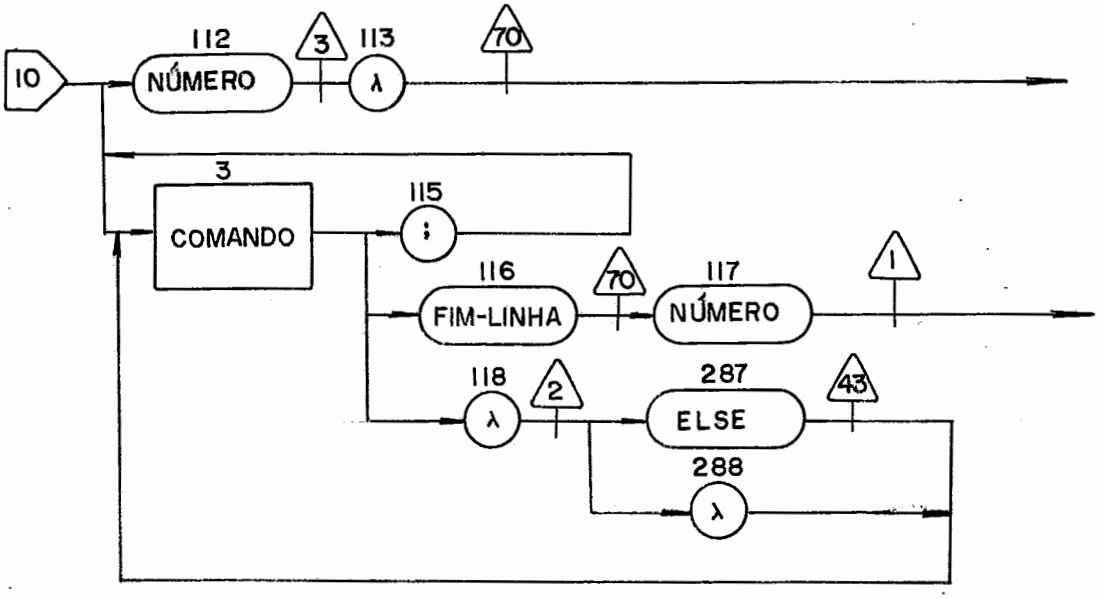

4
56
NEXT
57
VAR-INT
52
59
λ
54
58
VAR-REAL
6
COMANDO
61
;
62
FIM-LINHA
42
63
NÚMERO
1
64
λ
2

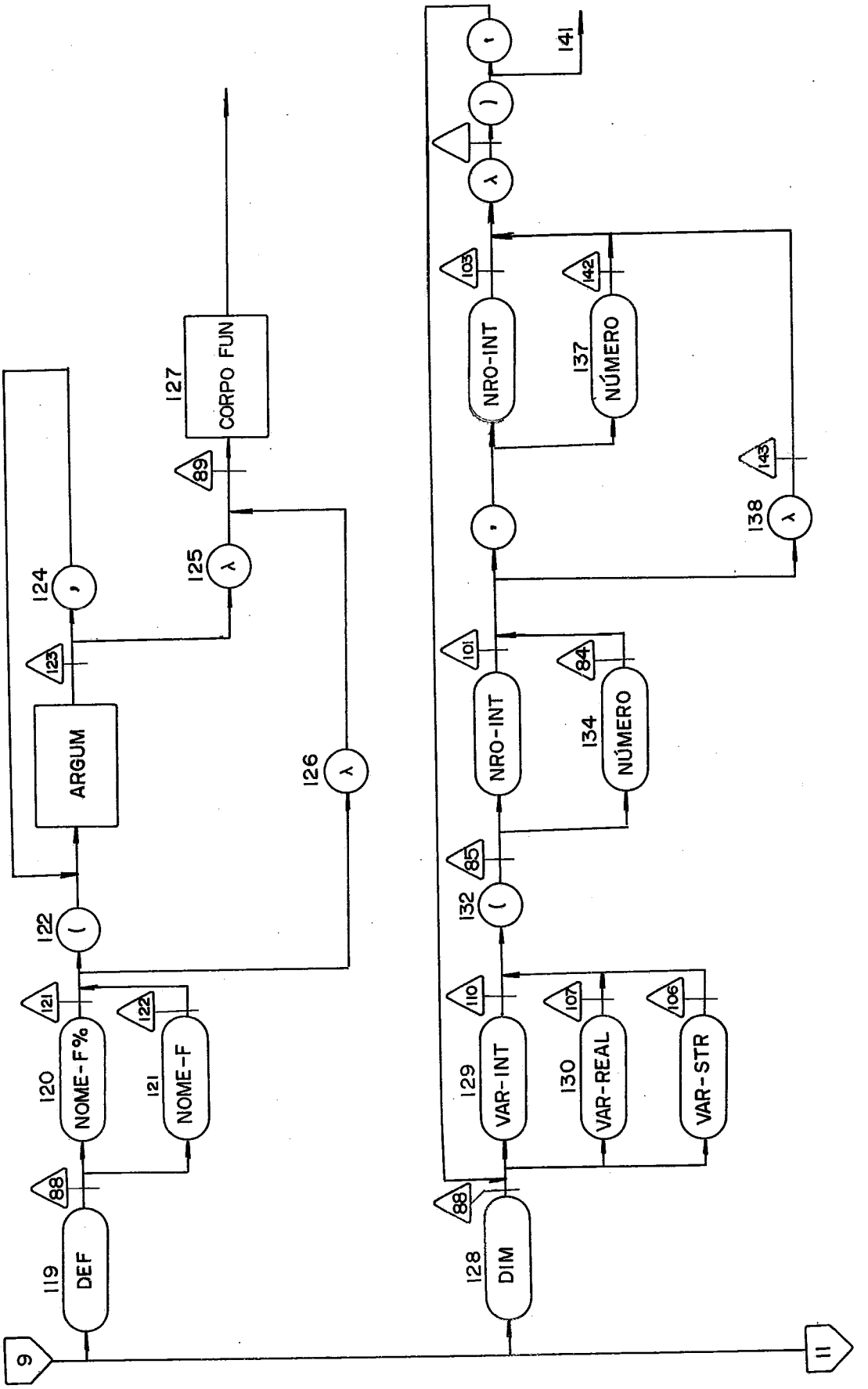

3
65
WHILE
57
67
EXPR
49
69
λ
55
71
FIM-LINHA
42
73
NÚMERO
1
6
66
UNTIL
58
68
EXPR
48
70
λ
56
2
72
λ

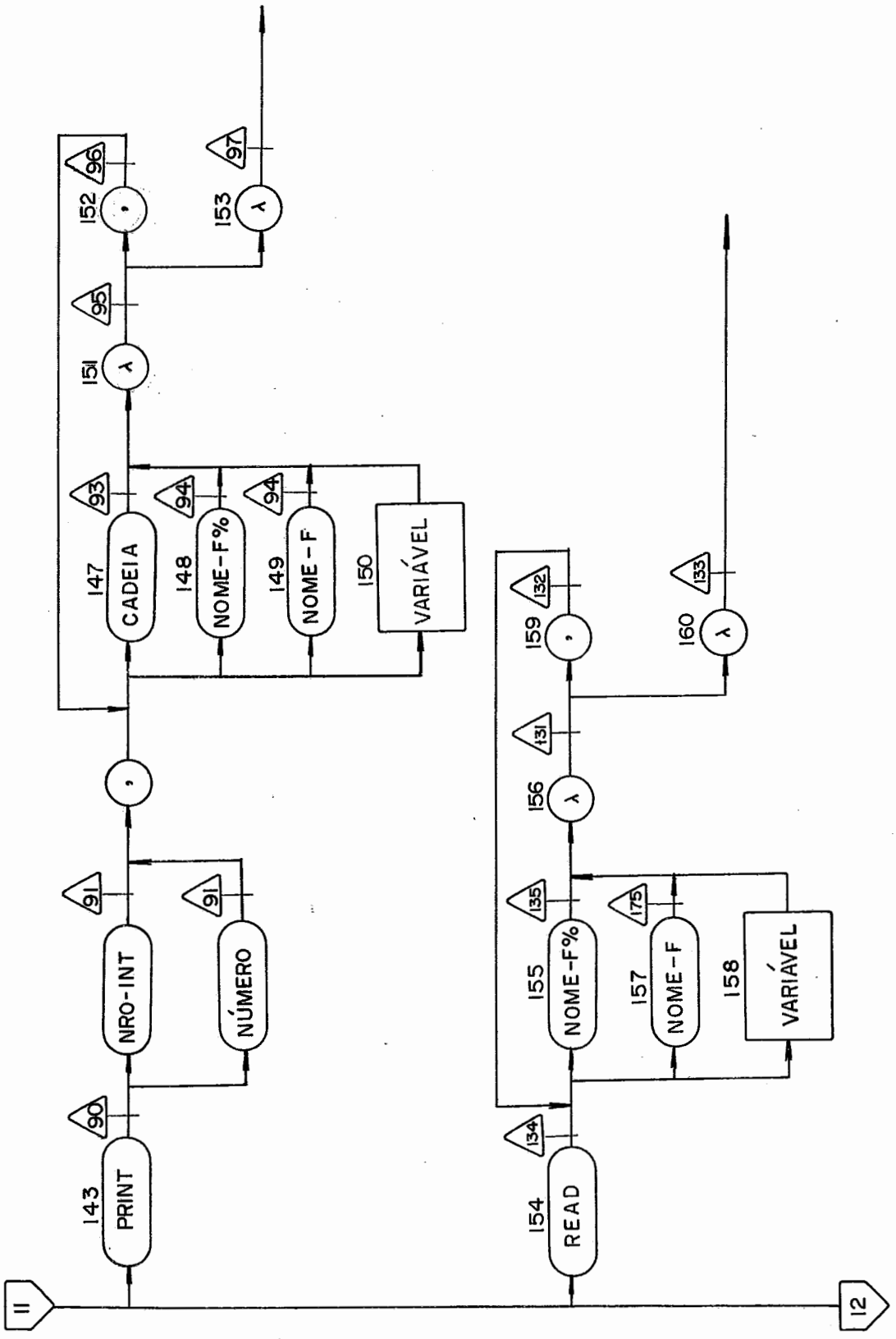

6
74
NEXT
75
VAR-INT
52
λ
57
76
VAR-REAL
7
COMANDO
79
;
80
FIM-LINHA
42
81
NÚMERO
1
82
λ
2

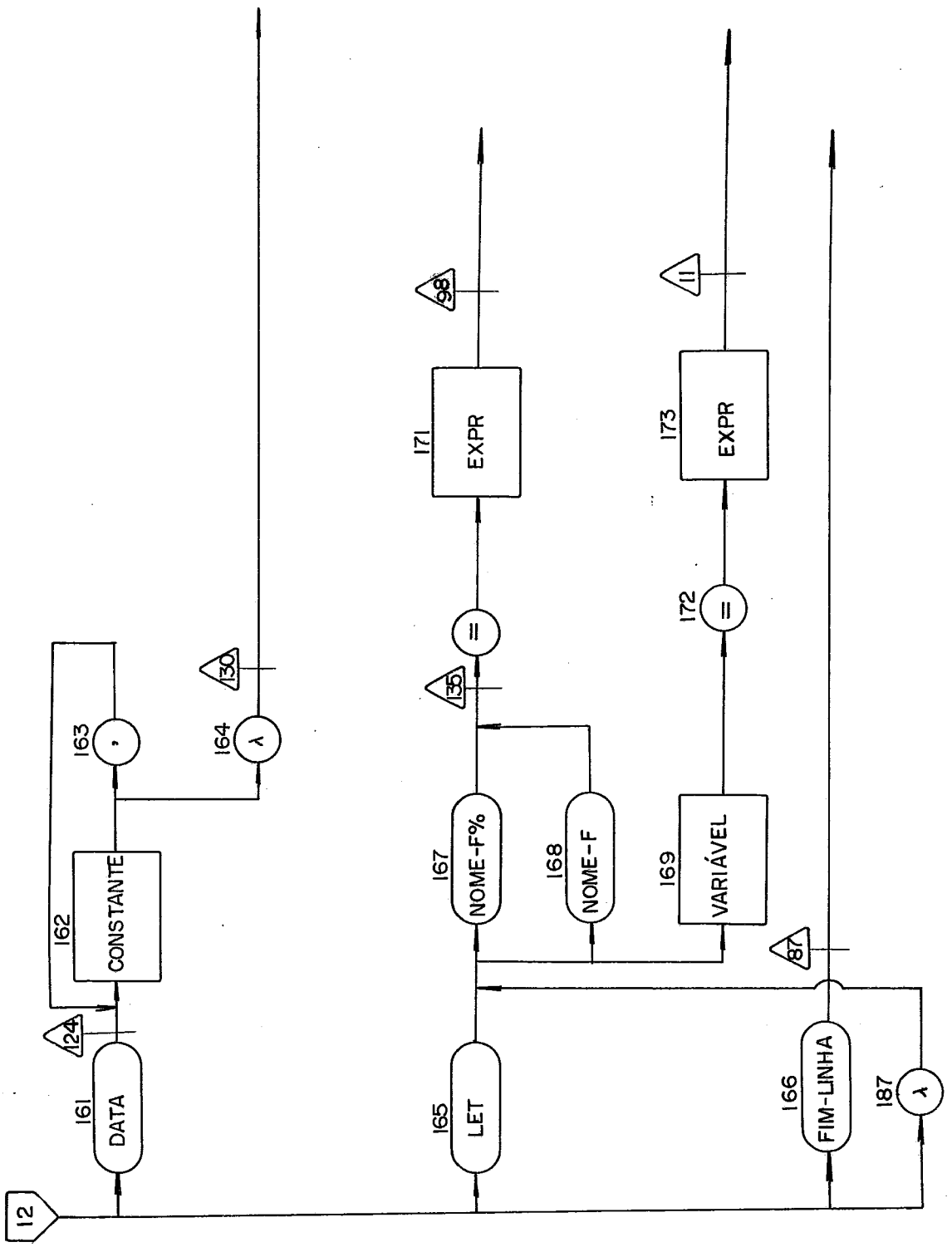

5
83
WHILE
62
188
58
85
EXPR
53
87
FIM-LINHA
42
89
NÚMERO
1
7
88
λ
2
59
84
UNTIL
62
189
λ
58
86
EXPR
7
90
NEXT
60
8
COMANDO
92
;
93
FIM-LINHA
42
95
NÚMERO
1
94
λ
2
96
IF
97
EXPR
67
8
9


8
98
GOTO
NÚMERO
3
λ
68
THEN
NÚMERO
3
ELSE
69
10
λ
9
COMANDO
106
;
68
FIM-LINHA
NÚMERO
1
109
λ
2
110
ELSE
69
111
λ

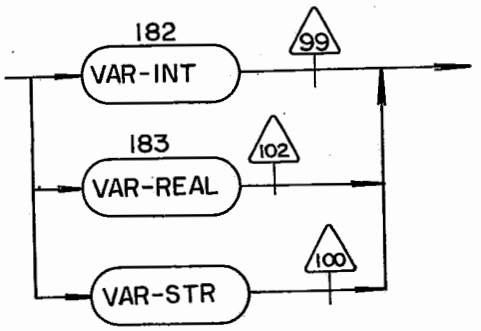

10
112
NÚMERO
3
113
λ
70
3
COMANDO
115
;
116
FIM-LINHA
70
117
NÚMERO
1
118
λ
2
287
ELSE
43
288
λ


9
119
DEF
88
120
NOME-F%
121
NOME-F
122
121
122
(
123
ARGUM
124
,
125
λ
89
127
CORPO FUN
126
λ
128
DIM
88
132
(
85
129
VAR-INT
110
130
VAR-REAL
107
VAR-STR
106
101
NRO-INT
84
134
NÚMERO
,
103
NRO-INT
142
137
NÚMERO
138
λ
143
λ
)
'
141
11

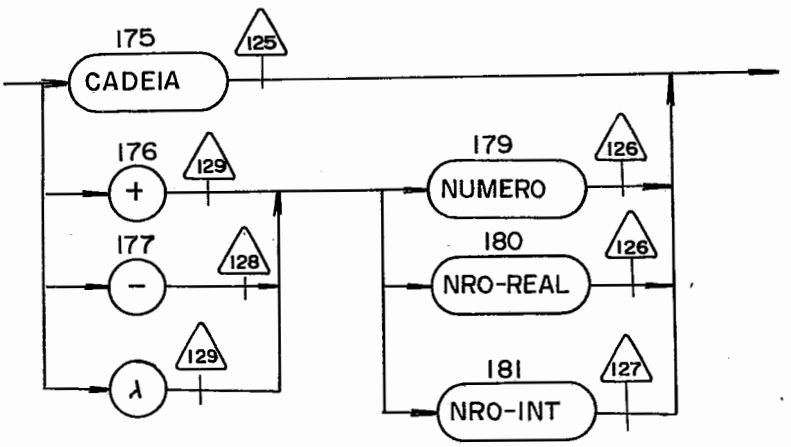

11
143
PRINT
90
NRO-INT
91
NÚMERO
91
,
147
CADEIA
93
148
NOME-F%
94
149
NOME-F
94
150
VARIÁVEL
151
λ
95
152
,
96
153
λ
97
154
READ
134
155
NOME-F%
135
157
NOME-F
175
158
VARIÁVEL
156
λ
131
159
,
132
160
λ
133
12


12
161
DATA
124
162
CONSTANTE
163
'
164
λ
130
165
LET
166
FIM-LINHA
187
λ
87
167
NOME-F%
168
NOME-F
169
VARIÁVEL
172
=
173
EXPR
11
171
EXPR
98
=
135

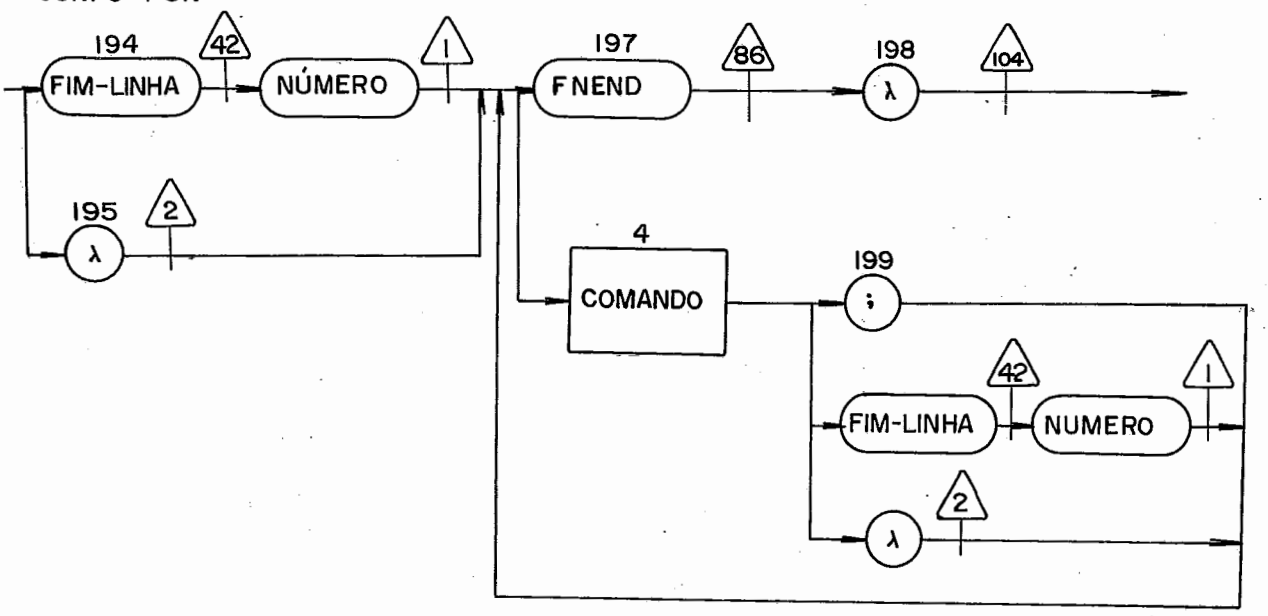
ARGUM



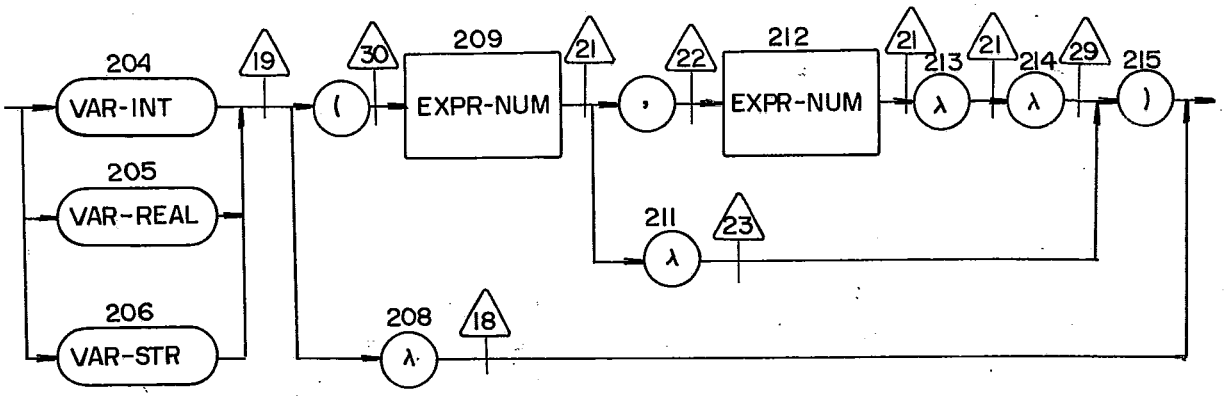
CONSTANTE



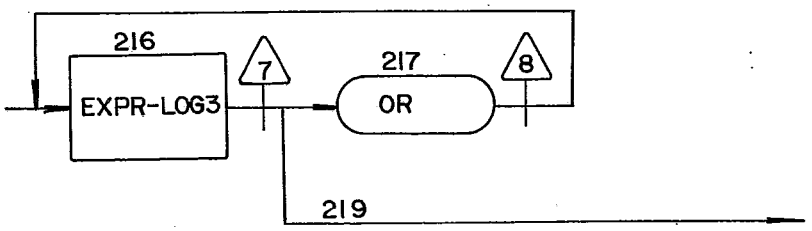
CORPO-FUN



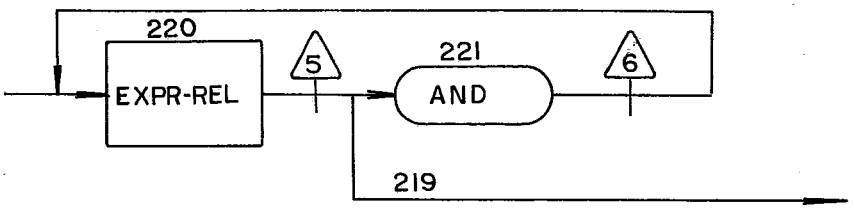
VARIAVEL



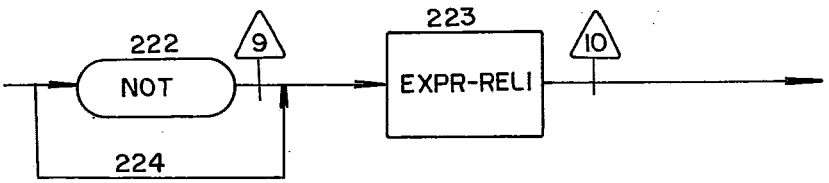
EXPR



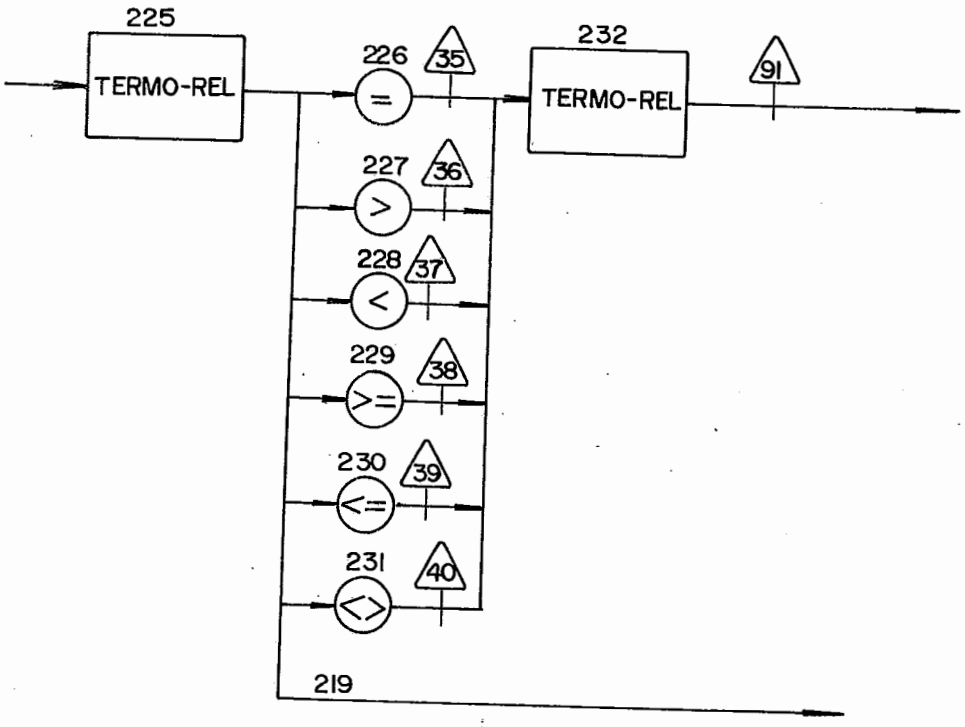
EXPR-LOG3

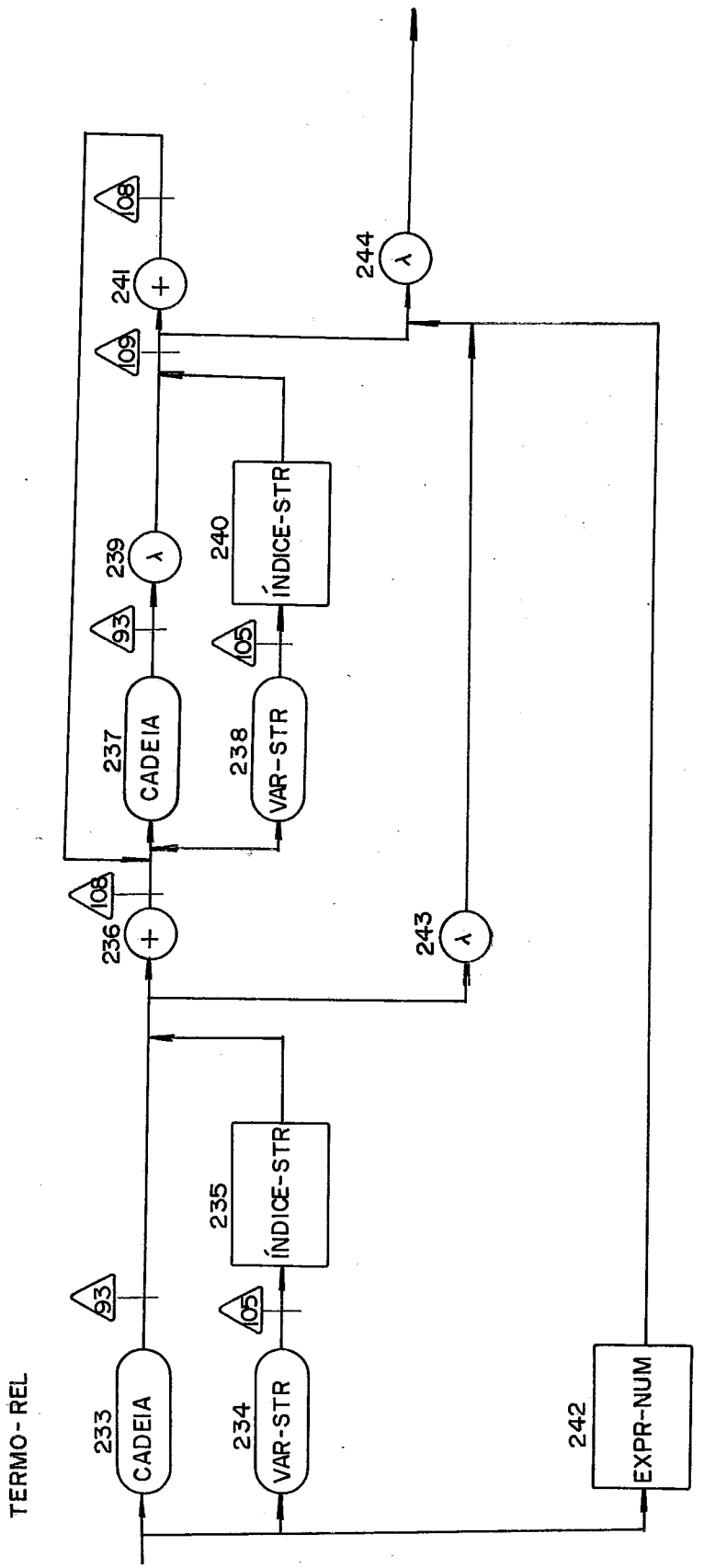


EXPR-REL

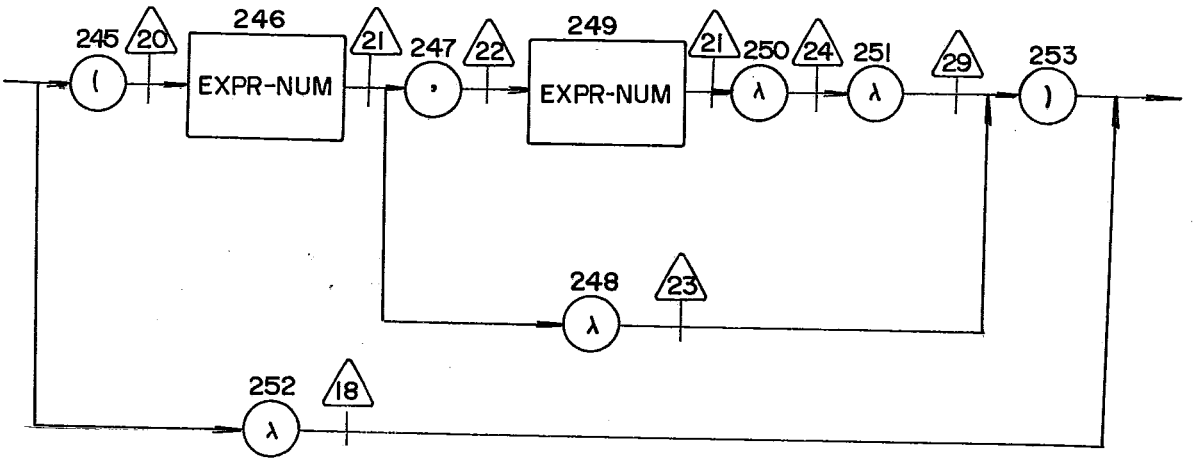


EXPR-REL I

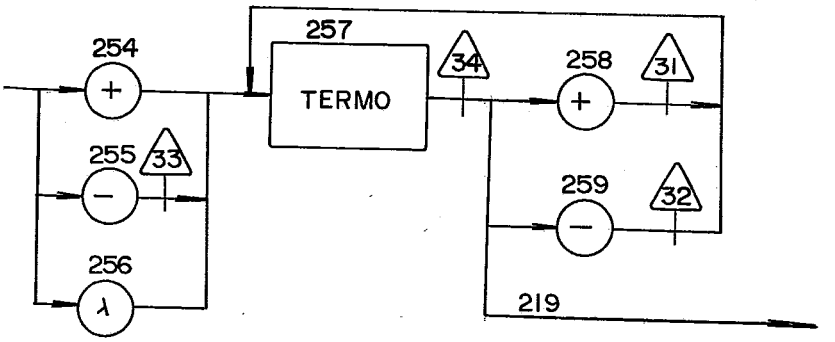

225
TERMO-REL
226
=
35
232
TERMO-REL
91
227
>
36
228
<
37
229
>=
38
230
<=
39
231
<>
40
219


TERMO-REL
233
CADEIA
234
VAR-STR
235
ÍNDICE-STR
242
EXPR-NUM
236
237
CADEIA
238
VAR-STR
240
ÍNDICE-STR
239
241
243
244
93
105
108
109

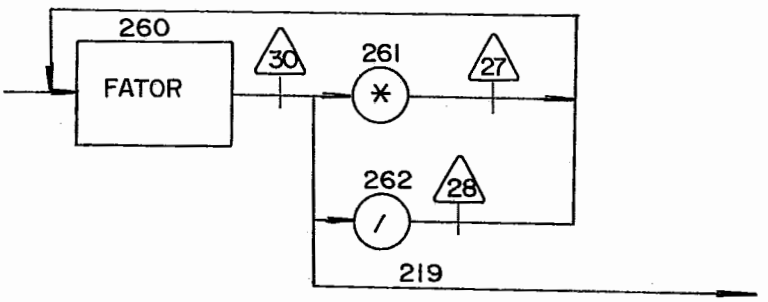
INDICE - STR



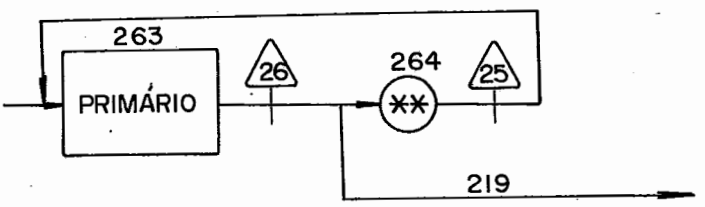
EXPR-NUM

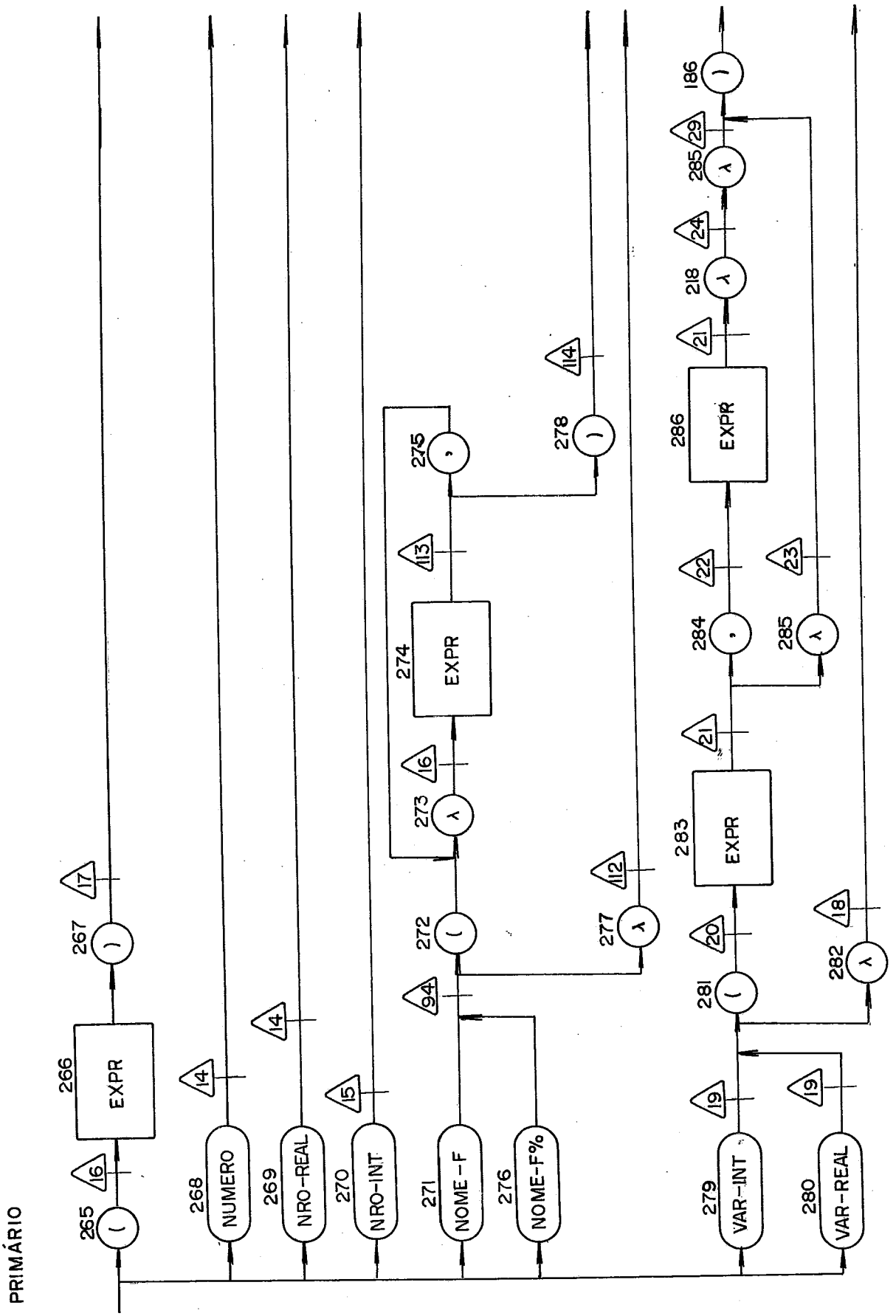


TERMO



FATOR




PRIMÁRIO
265
267
266
EXPR
268
NUMERO
269
NRO-REAL
270
NRO-INT
271
NOME-F
276
NOME-F%
272
273
274
EXPR
275
278
277
281
283
EXPR
284
285
286
EXPR
279
VAR-INT
280
VAR-REAL
282
186

APÊNDICE II

MENSAGENS DE ERRO DO SISTEMA DURANTE A EXECUÇÃO

ANALISADOR LÉXICO

FORMATO : ADVSCAN ==> CÓDIGO

CÓDIGO	MENSAGEM
1	Identificador usado com mais de 30 caracteres, serão considerados os 29 primeiros e o último
3	Apareceu mais de um caractere "." numa constante real, será considerada apenas 1
4	Apareceu mais de um sinal no expoente de uma constante real, será considerado apenas o primeiro.
5	Mais de um caracter "E" numa constante real
6	Caracter inválido, foi ignorado

ANALISADOR SEMÂNTICO

a) ADVERTÊNCIAS SEMÂNTICAS

FORMATO: ADVSEMANT ==> CÓDIGO

CÓDIGO	MENSAGEM
1	Rótulo de uma linha de comando deve ser positivo, será considerado positivo

2	Deve aparecer o símbolo de linha de continuação "&.../" .

b) ERROS SEMÂNTICOS

FORMATO: ERROSEMAN ==> CÓDIGO

CÓDIGO	MENSAGEM
1	Número de parâmetros reais diferente do número de parâmetros formais da função .
2	Tipo do parâmetro real não compatível com o tipo do parâmetro formal da função .
3	Chamada de uma função que não foi declarada.
4	Função declarada com parâmetros,sendo chamada sem parâmetros .
5	Expressão do tipo alfanumérico não pode ser usada como índice de vetor.
6	Variável já declarada num comando "DIMENSION" .
7	Função já declarada.
8	Parâmetro formal da função já declarado .
9	Uma expressão numérica não pode ter operandos do tipo alfanumérico .
10	A operação de concatenação, só admite operandos do tipo alfanumérico .

CÓDIGO	MENSAGEM
12	Numa operação relacional os dois operandos devem ser aritméticos ou ambos alfanuméricos .
13	Um operando do tipo alfanumérico não pode ser usado numa operação lógica.
14	Numa operação lógica os operandos devem ser do tipo inteiro.
15	Variável de iteração de um comando "FOR" não pode ser do tipo alfanumérico.
16	Variável que segue o comando "NEXT" de um comando "FOR" deve ser a variável de iteração deste comando.
18	Expressão condicional do comando "WHILE" ou "UNTIL" não pode ser do tipo alfanumérico.
19	Expressão condicional do comando "IF" não pode ser do tipo alfanumérico .
20	O primeiro membro de uma atribuição entre operandos alfanuméricos não pode ser uma cadeia.
21	Atribuição a um identificador de função fora do corpo da função.
22	Variável já usada com outro tipo.
23	Atribuição a um identificador do tipo alfanumérico só pode ter como 2º mem

CÓDIGO	MENSAGEM
	bro uma expressão alfanumérica.
24	Vetor declarado com duas dimensões e usado com um índice.
25	Vetor declarado com uma dimensão e usado com 2 índices.
26	Rótulo já analisado, deveria aparecer um comando "INSERT".
27	Não pode alterar um comando "DIM" , "DEF" e "FNEND".
28	Não existe recursividade.

ANALISADOR FINAL DE CONTEXTO

FORMATO: ***ERROCONTEXTO ==> CÓDIGO

CÓDIGO	MENSAGEM
1	A variável que segue o comando"NEXT", não corresponde à variável de iteração.
2	Falta um comando de fim de ciclo - "NEXT".
3	Comando "NEXT" de fim de ciclo está a mais.
4	Desvio para um rótulo não existente.
5	Não foi encontrada a quádrupla de retorno ao início do ciclo.

CÓDIGO	MENSAGEM
6	Um bloco de ciclo "FOR", "WHILE" ou "UNTIL" está mal formado.

ERROS DE EXECUÇÃO

FORMATO: ***ERRO DE EXECUÇÃO ==> CÓDIGO

CÓDIGO	MENSAGEM
1	Primeiro índice do vetor é menor que zero.
2	Primeiro índice do vetor é maior que a dimensão declarada.
3	Segundo índice do vetor é menor que zero.
4	Segundo índice do vetor é maior que a dimensão declarada
6	Expressão do comando"ON" é menor ou igual a zero.
7	Expressão do comando "ON" é diferente do número de objetivos.
8	Erro na impressão, deseja imprimir mais de 80 bytes.
9	As constantes alfanuméricas podem ter no máximo 80 caracteres.

APÊNDICE III

LISTAGEM DE PROGRAMAS EXECUTADOS

```
   REM                - PROGRAMA TESTE 1
   REM       ORDENACAO DE UM VETOR ALFANUMERICO PELO METODO BUBLESO
   REM
4     DIM FONTE$(10%)
5     PRINT 7,"********** ENTRE COM O VETOR FONTE"
6     LET AUX1$= "                "
7     FOR I% = 0% TO 10%
8     INPUT FONTE$(I%)
9     PRINT 7,AUX1$,FONTE$(I%)
10    NEXT I%
11    LET NROCHAVE% = 10%
12    LET INDICE% = -1%
13    WHILE INDICE%
14    LET INDICE% = 0%
15    LET NROCHAVE% = NROCHAVE% - 1%
16    FOR VARLOOP%= 0% TO NROCHAVE%
17    IF FONTE$(VARLOOP%) > FONTE$(VARLOOP%+1%) &
      / THEN    LET AUX$ = FONTE$(VARLOOP%)          &
              / LET FONTE$(VARLOOP%) = FONTE$(VARLOOP%+1%) &
              / LET FONTE$(VARLOOP%+1%) = AUX$         &
              / LET INDICE% = -1%
18    NEXT VARLOOP%
19    NEXT
20    PRINT 7,"********** VETOR ORDENADO"
21    FOR I% = 0%  TO 10%
22    PRINT 7,AUX1$,FONTE$(I%)
23    NEXT I%
24    END
25    INTERPRET #
FIM DA ANALISE
```

********** QUADRUPLAS **********

1(	11	0	0	0	-2
2(	56	0	0	2	3
3(	57	7	1	0	-4
4(	55	4	0	3	-5
5(	63	0	0	5	6
6(	46	5001	0	5	7
7(	47	0	0	10000	8
8(	12	10000	0	10000	9
9(	73	0	0	0	10
10(	7	5	5000	10001	11
11(	9	10001	10000	10002	12
12(	67	10002	0	26	13
13(	74	0	0	0	-14
14(	9	5	5002	10003	15
15(	10	1	10003	10004	16
16(	60	0	0	-10004	17
17(	61	0	1	0	-18
18(	56	0	0	3	19
19(	9	5	5002	10005	20
20(	10	1	10005	10006	21
21(	56	0	0	-10006	22
22(	57	7	2	0	-23
23(	64	0	0	5	24
24(	12	5	0	5	25
25(	49	0	0	9	26
26(	48	5	0	5	-27
27(	46	5000	0	6	-28
28(	20	5002	0	10007	29
29(	46	10007	0	7	-30
30(	63	0	0	0	31
31(	73	0	0	0	32
32(	5	7	0	74	33
33(	74	0	0	0	-34
34(	46	5001	0	7	-35
35(	7	6	5002	10008	36
36(	46	10008	0	6	-37
37(	63	0	0	8	38
38(	46	5001	0	8	39
39(	47	0	0	10009	40
40(	12	10009	0	10009	41
41(	73	0	0	0	42
42(	7	8	6	10010	43
43(	9	10010	10009	10011	44
44(	67	10011	0	71	45
45(	74	0	0	0	-46
46(	9	8	5002	10012	47
47(	10	1	10012	10013	48
48(	13	8	5002	10014	49
49(	9	10014	5002	10015	50
50(	10	1	10015	10016	51
51(	26	-10013	-10016	10017	52
52(	5	10017	0	68	53

```
53(      9          8       5002      10018        54
54(     10          1      10018      10019        55
55(     55     -10019          0          9        56
56(      9          8       5002      10020        57
57(     10          1      10020      10021        58
58(     13          8       5002      10022        59
59(      9      10022       5002      10023        60
60(     10          1      10023      10024        61
61(     55     -10024          0     -10021        62
62(     13          8       5002      10025        63
63(      9      10025       5002      10026        64
64(     10          1      10026      10027        65
65(     55          9          0     -10027        66
66(     20       5002          0      10028        67
67(     46      10028          0          7       -68
68(     64          0          0          8        69
69(     12          8          0          8        70
70(     49          0          0         41        71
71(     48          8          0          8       -72
72(     64          0          0          0        73
73(     49          0          0         31       -74
74(     56          0          0         10        75
75(     57          7          1          0       -76
76(     63          0          0          5        77
77(     46       5001          0          5        78
78(     47          0          0      10029        79
79(     12      10029          0      10029        80
80(     73          0          0          0        81
81(      7          5       5000      10030        82
82(      9      10030      10029      10031        83
83(     67      10031          0         93        84
84(     74          0          0          0       -85
85(     56          0          0          3        86
86(      9          5       5002      10032        87
87(     10          1      10032      10033        88
88(     56          0          0      10033        89
89(     57          7          2          0       -90
90(     64          0          0          5        91
91(     12          5          0          5        92
92(     49          0          0         80        93
93(     48          5          0          5       -94
94(     75          0          0          0       -95
```

********** TABELASIMB **********

TP	NIV	COMP	NPAR	ENDE	CONT	ABET	PONT	INF
-2	0	6	64	0	10	64	0	1
6	0	33	64	11	65	64	0	16448
2	0	5	64	12	3	64	6	0
6	0	12	64	13	65	64	33	16448
1	0	2	64	14	7	64	11	0
1	0	9	64	15	4	64	13	0
1	0	7	64	16	4	64	22	0
1	0	8	64	17	8	64	29	0
2	0	4	64	18	2	64	37	0
6	0	24	64	19	65	64	45	16448

********** TABECONSTE **********

VCONST	TIPOCONST	ENDCONST
10	1	20
0	1	21
1	1	22

********** TABELATEMP **********

ENDTEM	TIPOTEMP
23	1
24	1
25	1
26	1
27	2
28	1
29	2
30	1
31	1
32	1
33	1
34	1
35	1
36	2
37	1
38	1
39	2
40	1
41	1
42	2
43	1
44	2
45	1
46	1
47	2
48	1
49	1
50	2
51	1
52	1
53	1
54	1
55	1
56	2

NRO. DE ERROS = 000, ADVERTENCIAS = 000

```
********** ENTRE COM O VETOR FONTE
               CELIA
               ROSA
               AMELIA
               ANA
               VERA
               VILMA
               SUELI
               OLGA
               VANIA
               HELOISA
               CLARA
********** VETOR ORDENADO
               AMELIA
               ANA
               CELIA
               CLARA
               HELOISA
               OLGA
               ROSA
               SUELI
               VANIA
               VERA
               VILMA
***** FIM DA EXECUCAO *****
```

```
REM                    TESTE   2
REM            PESQUISA BINARIA NUMA TABELA ORDENADA.
1       DIM TABELA%(9%)
2       FOR I% = 0% TO 9%
3       INPUT TABELA%(I%)
4       NEXT I%
5       PRINT 7,"--- TABELA ---"
6       FOR I% = 0% TO 9%
7       PRINT 7,TABELA%(I%)
8       NEXT I%
9       LET ENCONTROU% = 0%
10      LET INDICE% = -1%
11      LET INF% = 0%
12      LET SUP% = 9%
13      INPUT ELEMENTO%
14      PRINT 7," ELEMENTO PROCURADO"
15      PRINT 7, ELEMENTO%
16      WHILE INF% < SUP%  AND (NOT ENCONTROU%)
17      LET MEIO% = (INF% + SUP%) / 2%
18      IF ELEMENTO% = TABELA%(MEIO%)    &
        /  THEN   LET ENCONTROU% = -1%    &
                / LET INDICE% = MEIO%     &
        /  ELSE  IF ELEMENTO% < TABELA%(MEIO%)  &
                / THEN LET SUP% = MEIO%         &
                / ELSE LET INF% = MEIO% + 1%
19      NEXT
20      IF ENCONTROU%    &
      / THEN  PRINT 7, " ELEMENTO ESTA NA TABELA "   &
      / ELSE  PRINT 7, " ELEMENTO NAO ESTA NA TABELA "
21      END
22      INTERPRET #
FIM DA ANALISE
```

********** QUADRUPLAS **********

1(	11	0	0	0	-2
2(	63	0	0	2	3
3(	46	5001	0	2	4
4(	47	0	0	10000	5
5(	12	10000	0	10000	6
6(	73	0	0	0	7
7(	7	2	5000	10001	8
8(	9	10001	10000	10002	9
9(	67	10002	0	18	10
10(	74	0	0	0	-11
11(	9	2	5002	10003	12
12(	10	1	10003	10004	13
13(	60	0	0	-10004	14
14(	61	0	1	0	-15
15(	64	0	0	2	16
16(	12	2	0	2	17
17(	49	0	0	6	18
18(	48	2	0	2	-19
19(	56	0	0	3	20
20(	57	7	1	0	-21
21(	63	0	0	2	22
22(	46	5001	0	2	23
23(	47	0	0	10005	24
24(	12	10005	0	10005	25
25(	73	0	0	0	26
26(	7	2	5000	10006	27
27(	9	10006	10005	10007	28
28(	67	10007	0	37	29
29(	74	0	0	0	-30
30(	9	2	5002	10008	31
31(	10	1	10008	10009	32
32(	56	0	0	-10009	33
33(	57	7	1	0	-34
34(	64	0	0	2	35
35(	12	2	0	2	36
36(	49	0	0	25	37
37(	48	2	0	2	-38
38(	46	5001	0	4	-39
39(	20	5002	0	10010	40
40(	46	10010	0	5	-41
41(	46	5001	0	6	-42
42(	46	5000	0	7	-43
43(	60	0	0	8	44
44(	61	0	1	0	-45
45(	56	0	0	9	46
46(	57	7	1	0	-47
47(	56	0	0	8	48
48(	57	7	1	0	-49
49(	63	0	0	0	50
50(	73	0	0	0	51
51(	4	6	7	10011	52
52(	78	4	0	10012	53

```
53(     44      10011      10012      10013       54
54(      5      10013          0         77       55
55(     74          0          0          0      -56
56(     13          6          7      10014       57
57(     17      10014       5003      10015       58
58(     46      10015          0         10      -59
59(      9         10       5002      10016       60
60(     10          1      10016      10017       61
61(     31          8     -10017      10018       62
62(      5      10018          0         67       63
63(     20       5002          0      10019       64
64(     46      10019          0          4       65
65(     46         10          0          5       66
66(     49          0          0         75       67
67(      9         10       5002      10020       68
68(     10          1      10020      10021       69
69(      4          8     -10021      10022       70
70(      5      10022          0         73       71
71(     46         10          0          7       72
72(     49          0          0         75       73
73(     13         10       5002      10023       74
74(     46      10023          0          6      -75
75(     64          0          0          0       76
76(     49          0          0         50      -77
77(      5          4          0         81       78
78(     56          0          0         11       79
79(     57          7          1          0       80
80(     49          0          0         83       81
81(     56          0          0         12       82
82(     57          7          1          0      -83
83(     75          0          0          0      -84
```

********** TABELASIMB **********

TP	NIV	COMP	NPAR	ENDE	CONT	ABET	PONT	INF
-1	0	7	64	0	5	64	0	1
1	0	2	64	10	6	64	7	0
6	0	14	64	11	65	64	0	16448
1	0	10	64	12	4	64	9	0
1	0	7	64	13	2	64	19	0
1	0	4	64	14	4	64	26	0
1	0	4	64	15	4	64	30	0
1	0	9	64	16	4	64	34	0
6	0	19	64	17	65	64	14	16448
1	0	5	64	18	6	64	43	0
6	0	25	64	19	65	64	33	16448
6	0	29	64	20	65	64	58	16448

********** TABECONSTE **********

VCONST	TIPOCONST	ENDCONST
9	1	21
0	1	22
1	1	23
2	1	24

********** TABELATEMP **********

ENDTEM	TIPOTEMP
25	1
26	1
27	1
28	1
29	1
30	1
31	1
32	1
33	1
34	1
35	1
36	1
37	1
38	1
39	1
40	1
41	1
42	1
43	1
44	1
45	1
46	1
47	1
48	1

NRO. DE ERROS = 000, ADVERTENCIAS = 000

```
----- TABELA ---
      12
      14
      16
      18
      20
      22
      24
      26
      28
      30
  ELEMENTO PROCURADO
      24
  ELEMENTO ESTA NA TABELA
***** FIM DA EXECUCAO *****
```

```
----- TABELA ----
      12
      14
      16
      18
      20
      22
      24
      26
      28
      30
  ELEMENTO PROCURADO
      29
  ELEMENTO NAO ESTA NA TABELA
***** FIM DA EXECUCAO *****
```

```
                              TESTE 3
          CALCULO DO NUMERO DE COMBINACOES POSSIVEIS
          DE "Y" OBJETOS TOMADOS "X" A X"
      1   INPUT Y%
      2   INPUT X%
      3   LET C% = Y%
      4   IF (Y% - X% ) ) X%          &
      /   THEN LET X% = Y% -X%
      5   FOR LOOP% = 1% TO X% STEP 1%
      6   LET C% = C% / LOOP%
      7   IF (Y% - LOOP%) ) (Y% - X% ) &
      /   THEN LET C%= (C% * Y%) - (C% *LOOP%)
      8   NEXT LOOP%
      9   PRINT 7, Y%,"OBJETOS COMBINADOS ", X%," A ",X%
      10  PRINT 7, "        TOTAL DE COMBINACOES =",C%
      11  END
      12  INTERPRET #
 FIM DA ANALISE
```

```
********** QUADRUPLAS **********
     1(     60          0          0          1          2
     2(     61          0          1          0         -3
     3(     60          0          0          2          4
     4(     61          0          1          0         -5
     5(     46          1          0          3         -6
     6(      7          1          2      10000          7
     7(     33      10000          2      10001          8
     8(      5      10001          0         11          9
     9(      7          1          2      10002         10
    10(     46      10002          0          2        -11
    11(     63          0          0          4         12
    12(     46       5000          0          4         13
    13(     47          0          0      10003         14
    14(     68       5000          0         17         15
    15(     48      10003          0      10003         16
    16(     49          0          0         19         17
    17(      5       5000          0         19         18
    18(     12      10003          0      10003         19
    19(     73          0          0          0         20
    20(      7          4          2      10004         21
    21(      9      10004      10003      10005         22
    22(     67      10005          0         37         23
    23(     74          0          0          0        -24
    24(     17          3          4      10006         25
    25(     46      10006          0          3        -26
    26(      7          1          4      10007         27
    27(      7          1          2      10008         28
    28(     33      10007      10008      10009         29
    29(      5      10009          0         34         30
    30(      9          3          1      10010         31
```

31(	9	3	4	10011	32
32(	7	10010	10011	10012	33
33(	46	10012	0	3	-34
34(	64	0	0	4	35
35(	13	4	5000	4	36
36(	49	0	0	19	37
37(	7	4	5000	4	-38
38(	56	0	0	1	39
39(	56	0	0	5	40
40(	56	0	0	2	41
41(	56	0	0	6	42
42(	56	0	0	2	43
43(	57	7	5	0	-44
44(	56	0	0	7	45
45(	56	0	0	3	46
46(	57	7	2	0	-47
47(	75	0	0	0	-48

********** TABELASIMB **********

TP	NIV	COMP	NPAR	ENDE	CONT	ABET	PONT	INF
1	0	2	64	0	8	64	0	0
1	0	2	64	1	9	64	2	0
1	0	2	64	2	7	64	4	0
1	0	5	64	3	5	64	6	0
6	0	19	64	4	65	64	0	16448
6	0	3	64	5	65	64	19	16448
6	0	29	64	6	65	64	22	16448

********** TABECONSTE **********

VCONST	TIPOCONST	ENDCONST
1	1	7

```
********** TABELATEMP **********
    ENDTEM        TIPOTEMP
         8             1
         9             1
        10             1
        11             1
        12             1
        13             1
        14             1
        15             1
        16             1
        17             1
        18             1
        19             1
        20             1
 NRO. DE ERROS =  000, ADVERTENCIAS = 000


       5 OBJETOS COMBINADOS       3  A       3

        TOTAL DE COMBINACOES =     10

***** FIM DA EXECUCAO *****


       5 OBJETOS COMBINADOS       5  A       5

        TOTAL DE COMBINACOES =     1

***** FIM DA EXECUCAO *****
```

```
                              TESTE 4
                 CALCULO      DO          FATORIAL
          2      DEF FNDFATORA%(X%)
          4      LET FATORA% = 1%
          6      FOR Z% = 1% TO X%
          8      LET  FATORA% = FATORA% * Z%
          10     NEXT Z%
          12     LET FNDFATORA% = FATORA%
          14     FNEND
          16     PRINT 7, "  NUMERO   FATORIAL"
          18     FOR K% = 1% TO 5%
          20     INPUT W%
          22     LET FATOR% = FNDFATORA%( W%)
          24     PRINT 7, W%,FATOR%
          25     NEXT K%
          26     END
          28     INTERPRET #
 FIM DA ANALISE
```

```
********** QUADRUPLAS **********
     1(     49          0          0         21          2
     2(     76          0          0          0         -3
     3(     46       5000          0          3         -4
     4(     63          0          0          4          5
     5(     46       5000          0          4          6
     6(     47          0          0      10000          7
     7(     12      10000          0      10000          8
     8(     73          0          0          0          9
     9(      7          4          2      10001         10
    10(      9      10001      10000      10002         11
    11(     67      10002          0         18         12
    12(     74          0          0          0        -13
    13(      9          3          4      10003         14
    14(     46      10003          0          3        -15
    15(     64          0          0          4         16
    16(     12          4          0          4         17
    17(     49          0          0          8         18
    18(     48          4          0          4        -19
    19(     46          3          0          1        -20
    20(     77          0          0          0        -21
    21(     56          0          0          5         22
    22(     57          7          1          0        -23
    23(     63          0          0          6         24
    24(     46       5000          0          6         25
    25(     47          0          0      10004         26
    26(     12      10004          0      10004         27
    27(     73          0          0          0         28
    28(      7          6       5001      10005         29
```

29(	9	10005	10004	10006	30
30(	67	10006	0	43	31
31(	74	0	0	0	-32
32(	60	0	0	7	33
33(	61	0	1	0	-34
34(	1	7	0	2	35
35(	2	1	1	0	36
36(	46	1	0	8	-37
37(	56	0	0	7	38
38(	56	0	0	8	39
39(	57	7	2	0	-40
40(	64	0	0	6	41
41(	12	6	0	6	42
42(	49	0	0	27	43
43(	48	6	0	6	-44
44(	75	0	0	0	-45

********** TABELASIMB **********

TP	NIV	COMP	NPAR	ENDE	CONT	ABET	PONT	INF
4	0	10	1	0	3	0	0	2
1	0	2	64	1	2	64	10	0
1	1	7	64	2	4	64	12	0
1	1	2	64	3	3	64	19	0
6	1	19	64	4	65	64	0	16448
1	1	2	64	5	2	64	21	0
1	1	2	64	6	3	64	23	0
1	1	6	64	7	2	64	25	0

********** TABECONSTE **********

VCONST	TIPOCONST	ENDCONST
1	1	8
5	1	9

********** TABELATEMP **********

ENDTEM	TIPOTEMP
10	1
11	1
12	1
13	1
14	1
15	1
16	1

NRO. DE ERROS = 000, ADVERTENCIAS = 000

```
   NUMERO    FATORIAL
      1          1
      2          2
      4         24
      5        120
      6        720
***** FIM DA EXECUCAO *****
```

```
   NUMERO    FATORIAL
      3          6
      1          1
      4         24
      7       5040
      5        120
***** FIM DA EXECUCAO *****
```

```
   !                          TESTE 5
   !       TESTE DO RECUPERADOR DE ERROS SINTATICOS
   !
     1     INPUT Y%
     2     INPUT X%
     3     LET C%   Y%
                    *
)))ERRO)))
     4     IF (Y% - X% ) ) X%          &
*SIMBOLO INSERIDO ===)           =

     /     THEN LET X% = Y% -X%
     5     FOR LOOP% = 1% TO X% STEP 1%
     6     LET     = C% / LOOP%
                   *
)))ERRO)))
*SIMBOLO RETIRADO ===)           =

                         *
)))ERRO)))
     7     IF (Y% - LOOP%) ) (Y% - X% ) &
*SIMBOLO RETIRADO ===)           /

*SIMBOLO INSERIDO ===)           =

     /     THEN LET C%= (C%    Y%) - (C% *LOOP%)
                                 *
)))ERRO)))
*SIMBOLO RETIRADO ===)           VARIAVEL

     8     NEXT LOOP%
     9     PRINT 7   Y%,"OBJETOS COMBINADOS ", X%," A ",X%
                    *
)))ERRO)))
*SIMBOLO RETIRADO ===)           VARIAVEL

     10    PRINT 7, "        TOTAL DE COMBINACOES =",C%
     11    END
     12    INTERPRET #
 FIM DA ANALISE
 NRO. DE ERROS =  005, ADVERTENCIAS = 000
```