


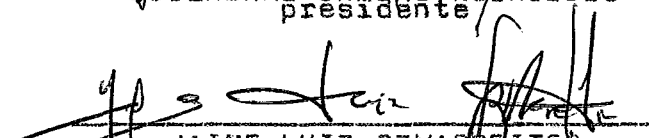
EXPANSOR DE MACRO ASSEMBLER

ZACHARIAS ERNANE DAS CANDEIAS

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

Aprovada por:

  
GUILHERME CHAGAS RODRIGUES  
presidente

  
JAIME LUIZ SZWARZEITER

  
NELSON MACULAN FILHO

ESTADO DO RIO DE JANEIRO - BRASIL  
MARÇO DE 1976

Aos meus saudosos Pais  
A minha esposa TEREZA  
e aos meus filhos  
ANA LÚCIA  
JUNIOR  
ARMANDO  
TASSO  
CESAR

## Agradecimentos

A todas pessoas que direta ou indiretamente contribuíram, quer com seus conhecimentos, quer com seus incentivos na elaboração deste trabalho.

Ao Prof. Guilherme Chagas Rodrigues MSc do Nucleo de Computação Eletrônica e COPPE da UFRJ, pelo seu interesse e sugestões Valiosas durante a orientação desta tese.

Como também nosso reconhecimento ao Dr. Arão Horowitz chefe do Departamento de Energia Nuclear da UFPe. juntamente com todos colegas e funcionários por não ter faltado com seus apoios e incentivos.

Aos colegas Eduardo Doria Silva e Milton Albuquerque Bezerra pelas discursões durante sua realização.

Ao colega Pedro Nogueira Cruz e sua esposa Dra. Maria das Dores Nogueira Cruz pela correções finais dos originais deste trabalho.

As Srtas. Maria Delza de Oliveira Cardoso, Ana Lúcia Bezerra Candeias e Escolástica Pereira de Farias pelos desenhos e a elaboração da parte datilográfico.

À Universidade Federal de Pernambuco e Coordenação de Programação de Pós-Graduação em Engenharia pelo apoio de base , sem o qual era impossivel a realização deste trabalho.

## SUMÁRIO

Este trabalho apresenta um Expansor de Macro Assembler (EMA) que se integra ao Sistema Operacional de Simulação (SOS), cuja finalidade é desenvolver "software" para o Terminal Inteligente (TI), desenvolvido em conjunto pelo Núcleo de Computação Eletrônica (NCE) e a Coordenação dos Programas de Pós-Graduação de Engenharia (COPPE) da Universidade Federal do Rio de Janeiro.

Descrevemos neste trabalho os seguintes tópicos:

- . Generalidades sobre os processadores de Macros
- . Organização das tabelas do EMA
- . Implementação do subsistema EMA/SOS
- . Fluxograma de subsistema

Inicialmente mostramos de forma didática o que vem a ser uma rotina macro e o seu uso num programa Assembler.

Depois criamos uma linguagem de programação para o manuseio do Expansor de Macro Assembler. Nas definições de seus comandos usamos a meta-linguagem BNF (Bakus Naur Form).

Descrevemos também o uso e geração das principais tabelas, facilitando, deste modo, a descrição da implementação do subsistema. Mostramos os passos básicos do processador implementado e, como elucinação, damos um exemplo de uma rotina recursiva, onde expomos o uso prático de todas as tabelas básicas.

## ABSTRACT

This work presents a Macro Assembler Expander (MAE) which is to be added to the Operational Simulation System (OSS), in order to develop software for the Intelligent Terminal (IT). This project was developed in conjunction with the Nucleo de Computação Eletrônica (NCE) and the Coordenação de Programas de Pós-Graduação em Engenharia (COPPE) of Universidade Federal do Rio de Janeiro (UFRJ). The following topics are discussed:

- . General principles of macro processors
- . MAE table organization
- . Implementation of subsystem MAE/OSS
- . MAE flow diagram

At first a macro routine is defined and its use in an Assembler program demonstrated.

A program language is then created in order to use the MAE. This language is based on Bakus Naur Form (BNF).

The use and generation of the main tables are also described in order to facilitate the description of the subsystem implementation. The basic operations of the implemented processor are shown, and example of recursive routine given which demonstrates the practical use of all of the the basic tables.

## I N D I C E

I.	Introdução . . . . .	1
II.	Generalidade sobre macro processadores . . . . .	5
III.	Discriminação da linguagem . . . . .	8
	3.1. Definição formal dos elementos da linguagem . . . . .	8
	3.2. Definição de rotinas no expensor do macro assembler . . . . .	9
	3.3. Comando do expensor de macro assembler . . . . .	10
	3.3.1. Classificação dos comandos . . . . .	10
	3.3.2. Definições . . . . .	11
	3.3.3. Chamada de rotina macro . . . . .	11
	3.3.4. Comando de concatenação . . . . .	14
	3.3.5. Comando de transferência- IF . . . . .	16
	3.3.6.-Comando de transferência - GO TO . . . . .	17
	3.3.7. Comando de atribuição LET . . . . .	17
	3.3.8. Comando NOP . . . . .	18
	3.3.9. Variáveis do Subsistema . . . . .	19
	3.3.10. Declaração de variáveis GLB . . . . .	19
	3.3.11. Comando. STATUS . . . . .	20
	3.3.12. Comando de substituição . . . . .	21
	3.4. Manutenções e consultas as Bibliotecas . . . . .	21
	3.4.1. Comando externo . . . . .	22
IV.	Organização das tabelas no expensor de macro assembler . . . . .	23
	4.1. Introdução . . . . .	23
	4.2. Tabela que guarda os parâmetros formais (ALA) . . . . .	24
	4.3. Tabela de nomes de macros (TNM) . . . . .	24
	4.4. Tabela de definição de macro (TDM) . . . . .	24
	4.5. Tabela de variáveis locais e globais . . . . .	27
	4.5.1. Organização . . . . .	28
V.	Implementação do subsistema expensor macro assembler . . . . .	33
	5.1. Introdução . . . . .	33
	5.2. Reconhecedor de rotinas macros . . . . .	33
	5.3. Reconhecedor dos comandos EMA . . . . .	33
	5.4. Reconhecedor e executor das chamadas de rotina Macro . . . . .	34
	5.5. Consultor de Biblioteca . . . . .	38

VI.	Conclusões . . . . .	40
	6.1. Escolha da linguagem . . . . .	40
	6.2. Observações finais e sugestões para futuras imple- mentações . . . . .	43
	Apêndice A . . . . .	44
	Linguagem de extensão . . . . .	44
	Apêndice B . . . . .	46
	Uso dos comandos . . . . .	50
	Apêndice C . . . . .	50
	Fluxograma do expensor de macro assembler . . . . .	
	Apêndice D . . . . .	56
	Resultados . . . . .	
	Bibliografia . . . . .	85

## I. INTRODUÇÃO

A finalidade deste trabalho é integrar o Sistema Operacional de Simulação (SOS) às facilidades de macro Assembler.

O SOS é um sistema orientado para desenvolver o "software" básico para terminais inteligentes. Como está descrito em /7/ é composto das seguintes facilidades.

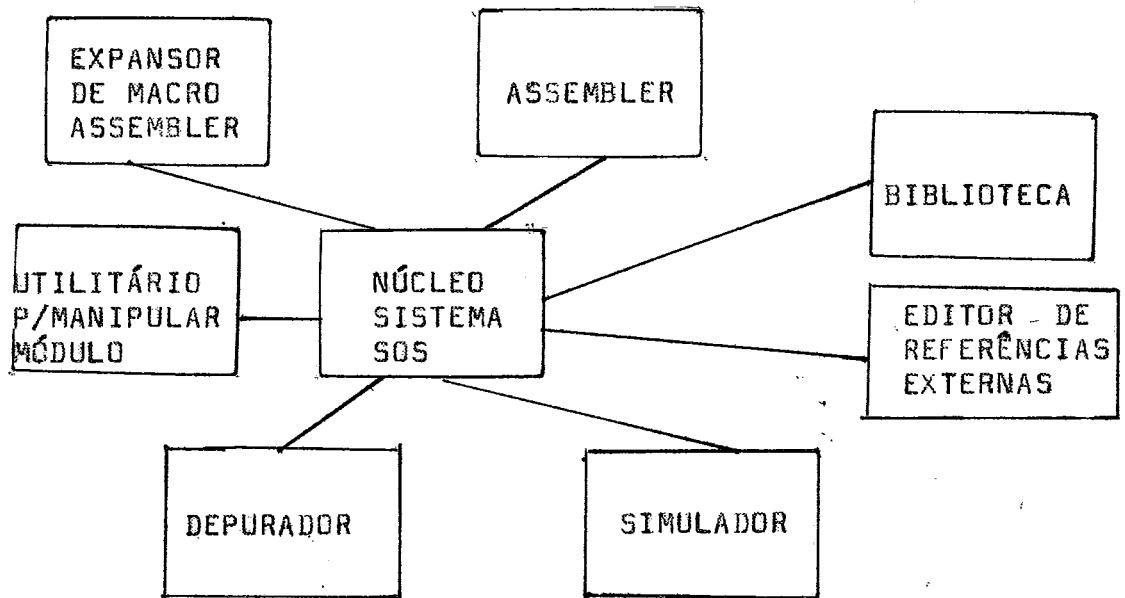
1. Montador Assembler
2. Uma biblioteca onde são guardados os módulos objetos
3. Um Editor de referência, que junta, e resolve as referências entre os módulos.
4. Um Simulador do Terminal Inteligente TI
5. Um módulo de depuração, que permite uma simulação iterativa e simbólica usando uma linguagem de alto-nível própria.
6. Uma série de utilitários para manipulação dos módulos.
7. Um módulo de controle (núcleo central do sistema) para controlar as facilidades acima.

Além desses será incluído um Expansor de Macro Assembler (EMA) que, como os demais, será controlado através do núcleo central.

O Sistema Operacional de Simulação pode ser descrito como mostra a fig. 1.



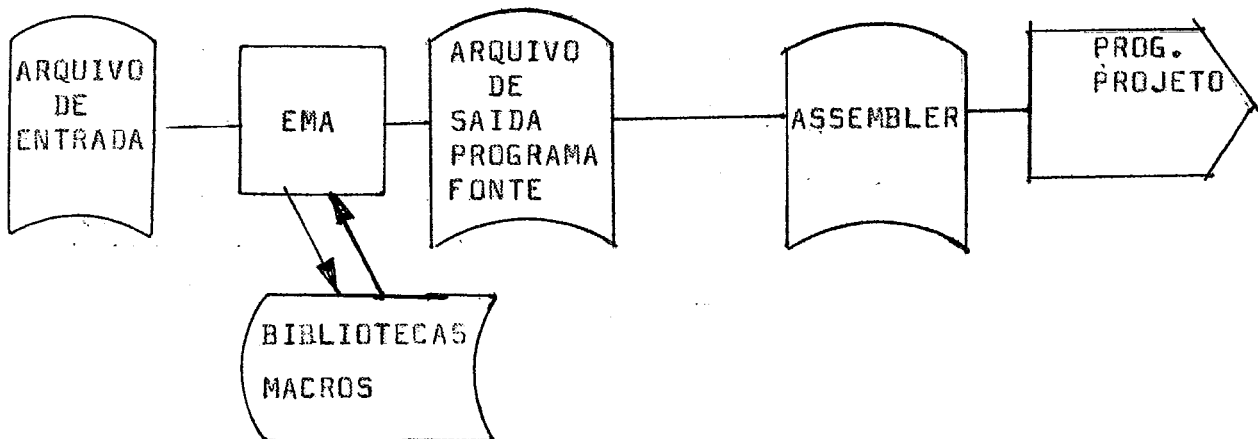
FIG. 1



### Sistema Operacional de Simulação

Gráficamente podemos representar a integração do sistema SOS com o subsistema EMA como sendo fig. 2

FIG. 2



O EMA é um programa que interpreta os comandos de rotinas macros (serão descritos no decorrer deste trabalho).

Podemos definir uma rotina macro informalmente, como sendo um bloco de instruções constituída de um corpo e de dois delimitadores, um inferior e o outro superior §2.2.

O corpo pode ser referenciado para ser incluído, em um programa e ser expandido, através de um comando denominado chamada de macro.

Exemplo (didático)

Seja a rotina macro

EXPAND MACRO (delimitador superior)

X  
Y Corpo de macro  
Z

ENDM (delimitador inferior)

Seja o programa fonte

A  
B  
EXPAND (chamada da rotina EXPAND)  
C  
D  
EXPAND (chamada da rotina EXPAND)  
-----  
-----

aplicando este programa ao subsistema EMA resultará

A  
B  
X  
Y  
Z  
C  
D  
X  
Y  
Z  
E

Programa resultante após a expansão  
ou programa fonte expandido.

Aplicações como esta são muito comuns em programação assembler onde as repetições de blocos são muito frequentes, as vezes mudando apenas um operando, ou operador, ou toda a instrução.

Geralmente o uso das rotinas macros facilita o trabalho de codificação, depuração, além dos usuários poderemos criar novos comandos no Assembler usado.

No capítulo 1 apresentamos generalidades sobre as origens dos macros processadores. Como complementação, apresentamos no apêndice A o uso dos macros processadores nas linguagens de extensões.

No capítulo 2 definimos de uma maneira formal a linguagem do EMA. Para isto usamos a meta-linguagem BNF (Bakus Naur Form). Os exemplos do uso dos comandos estão no apêndice B.

No capítulo 3 apresentamos a organização das tabelas, seus usos e implementações.

No capítulo 4 descrevemos as tarefas básicas para implementação do subsistema EMA, e será complementado com o apêndice C onde se encontra o fluxo do subsistema

## II. GENERALIDADES SOBRE MACRO PROCESSADORES

### 2.1. Introdução

Macro, de acordo com o pequeno dicionário da língua portuguesa, é um prefixo que em grego significa grande. Este prefixo é usado em várias palavras técnicas em todos os ramos dos conhecimentos humanos, por exemplo: macroeconomia, macromolecular, macroregião, etc.

Em linguagem de programação este termo é usado para definir uma instrução que por sua vez, produzirá uma nova sequência de instruções básicas durante sua execução.

Entende-se por instruções básicas todas aquelas que sejam executáveis por circuitos lógicos de uma máquina.

O clássico artigo de McIlroy /1/ mostra de forma sucinta e clara o aumento da versatilidade de uma assembler que admite macros instruções. Uma das grandes vantagens é a montagem condicional controlada por parâmetros.

Cronologicamente a idéia de macro surgiu na literatura científica em 1962 num trabalho de Halpern, M.I. /2/ com o processador de macro XPOP, que tem como linguagem base o FAP. Tentou-se criar uma linguagem de programação, cujos comandos se aproximassem o mais possível de ingles natural, entretanto não contou com o apoio necessário e terminou desistindo.

Em 1964 /3/ Wilkes, M.V. na University Computer Laboratory ' Cambridge, desenvolveu o WISP. Constava de um processador de Macros que explorava duas idéias básicas, o processamento de lista e o "Self-Compiler". Neste trabalho surgiu pela primeira vez a técnica de um reconhecedor usando "template matching" (padrão de comparação) em 1965 com finalidade de ajudar a criação de compilador CPL escrito para o computador TITAN Strachey C. /4/ projeta o clássico dos processadores de macro "General Purpose Macrogenerator" (GPM) que é considerado o mais econômico e elegante sob o ponto de vista de linguagem de programação.

No início o uso dos processadores macros era mais ligado a geração de compiladores.

Com o avanço da tecnologia o uso dos computadores é cada vez mais solicitado, para executar as mais diferentes tarefas das atividades humanas.

Desta necessidade surgiu uma complexidade cada vez mais crescente dos sistemas de computação, sobre todos os aspectos.

Um dos aspectos que enfocaremos será o da linguagem de programação. Para atender as diversas áreas de conhecimento humano foram criadas várias linguagens de programação, aproximando-se o mais possível da linguagem natural dos tipos de problemas enfocados, por exemplo, de uma sentença matemática caso a linguagem seja científica, ou melhor, numérica.

Como nem sempre a tarefa a resolver é numérica daí surgiram linguagens de simulação, comerciais, não numérica etc.

Apesar de todas as linguagens serem criadas para ter aplicações em determinadas áreas é sempre possível resolver problemas que estejam fora do campo para as quais foram criadas. Nestes casos geram códigos ineficientes, além de distanciar da linguagem natural do problema, desencorajando aos programadores usá-las fora de sua área. Exemplo: resolver um problema numérico usando LISP ou resolver um problema de lista usando FORTRAN o ideal seria uma linguagem geral que resolvesse todas as aplicações. Sob o ponto de vista teórico é plenamente possível porém do ponto de vista prático é inviável. Outra solução seria uma linguagem para cada tipo de aplicação. Se o número de usuários for pequeno em cada aplicação, a solução é anti-econômica. Uma solução viável e, em certos casos, chega ser até econômica é a partir de uma linguagem base estender seus comandos para cada aplicação. Isto é possível usando um processador de rotinas macro (Apêndice A). Deste modo poderemos ter uma única linguagem base, e vários conjuntos de extensões projetadas para determinadas tarefas. Assim sendo cada usuário poderá criar novos comandos, com nomes, de forma que mais se ajuste aos seus trabalhos.

Foi seguindo esta filosofia que surgiram em 1966 o LIMP (Lan

guage Independent Macro Processor) desenvolvido por Waite, W.M. na University de Colorado /5/ e o ML/I (Macro Language/I) desenvolvido por Brown, J. na University Mathematical Laboratory Cambridge /6/. São processadores para estender linguagens.

### III. DESCRIÇÃO DE LINGUAGEM

#### 3.1. Definição Formal dos Elementos da Linguagem

Nas definições dos comandos EMA, para auxiliar sua conceituação, usaremos a meta-linguagem BNF /20-21/.

Aqui definiremos os elementos da linguagem mais usados no decorrer deste trabalho.

$\langle \text{comando arit.} \rangle ::= \langle \text{identif} \rangle = \langle \text{expressão} \rangle$   
 $\langle \text{expressão} \rangle ::= \langle \text{sinal opc.} \rangle \langle \text{termo} \rangle \mid \langle \text{expressão} \rangle \langle \text{sinal} \rangle \langle \text{termo} \rangle$   
 $\langle \text{termo} \rangle ::= \langle \text{fator} \rangle \mid \langle \text{termo} \rangle \langle \text{op-mult} \rangle \langle \text{fator} \rangle$   
 $\langle \text{fator} \rangle ::= \langle \text{identif} \rangle \mid \langle \text{n}^\circ \rangle \mid \langle \text{expressão} \rangle$   
 $\langle \text{identif} \rangle ::= \langle \text{letra} \rangle \mid \langle \text{identif} \rangle \langle \text{letra} \rangle \mid \langle \text{identif} \rangle \langle \text{dígitos} \rangle$   
 $\langle \text{n}^\circ \rangle ::= \langle \text{dígitos} \rangle \mid \langle \text{n}^\% \rangle \langle \text{dígitos} \rangle$   
 $\langle \text{expressão booleana} \rangle ::= \langle \text{expressão} \rangle \langle \text{op-bool} \rangle \langle \text{expressão} \rangle$   
 $\langle \text{sinal opc.} \rangle ::= \langle \text{sinal} \rangle \mid \Lambda$   
 $\langle \text{rótulo} \rangle ::= \langle \text{identif} \rangle \mid \Lambda$   
 $\langle \text{rótulo EMA} \rangle ::= \langle \text{identif} \rangle \# \mid \Lambda$   
 $\langle \text{bretulos} \rangle ::= \langle \text{rótulo EMA} \rangle \mid \langle \text{rótulos} \rangle$   
 $\langle \text{bs} \rangle ::= \emptyset \mid \langle \text{bs} \rangle \emptyset$   
 $\langle \text{bn} \rangle ::= \langle \text{bs} \rangle \mid \Lambda$   
 $\langle \text{sinal} \rangle ::= + \mid -$   
 $\langle \text{op-mult} \rangle ::= * \mid / \mid * *$   
 $\langle \text{op-bool} \rangle ::= \neg \mid = \mid < \mid >$   
 $\langle \text{letra} \rangle ::= A \mid B \mid C \mid D \mid E \mid F \mid G \mid H \mid I \mid J \mid K \mid L \mid M \mid N \mid O \mid P \mid Q \mid R \mid S \mid T \mid U \mid V \mid W \mid X \mid Y \mid Z$   
 $\langle \text{dígitos} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

O nulo é representado pelo símbolo " $\Lambda$ ", enquanto que o branco é representado por " $\emptyset$ ".

### 3.2. Definição de Rotinas no Expansor do Macro Assembler

$\langle \text{rotina macro} \rangle ::= \langle \text{identif} \rangle \text{ MACRO } \langle \text{lista de param. formais} \rangle$   
 $\quad \quad \quad \langle \text{corpo da macro} \rangle \text{ ENDM}$   
 $\langle \text{lista de param. formais} \rangle ::= \langle \text{param. formal} \rangle | \langle \text{lista de param. for-}$   
 $\quad \quad \quad \text{mais} \rangle, \langle \text{param. formal} \rangle$   
 $\langle \text{param. formal} \rangle ::= \& \langle \text{identif} \rangle$   
 $\langle \text{corpo da macro} \rangle ::= \langle \text{é o conjunto dos comandos descritos neste ca-}$   
 $\quad \quad \quad \text{pítulo} \rangle$

As palavras reservadas MACRO e ENDM são delimitadores do corpo da rotina.

Formato do primeiro cartão

$\langle \text{identif} \rangle$	$\langle \text{bs} \rangle$	MACRO	$\langle \text{bs} \rangle$	$\langle \text{lista de parâm. formais} \rangle$
----------------------------------	-----------------------------	-------	-----------------------------	--

Os brancos contidos na lista de parâmetros formais serão substitu<sub>g</sub> tidos por nulo. Exemplo:

ENDERE  $\not\equiv$  MACRO  $\not\equiv$  &A, &ELEM, &ACHA, &5

é equivalente a

ENDERE  $\not\equiv$  MACRO  $\not\equiv$  &A,  $\not\equiv$  &ELEM, &A  $\not\equiv$  Cb  $\not\equiv$  HA, &5

$\not\equiv$  equivale a um branco

O último cartão da rotina tem o seguinte formato:

$\langle \text{bs} \rangle$	ENDM	$\langle \text{Comentário} \rangle$
-----------------------------	------	-------------------------------------

ENDM é o delimitador do fim do macro

Devemos salientar que cada comando deve estar contido em um registro.



### 3.3. COMANDOS DO EXPANSOR DE MACRO ASSEMBLER

#### 3.3.1. Classificação dos comandos

Os comandos do sub-sistema EMA podem ser classificados em três grupos:

1. Os que controlam as expansões;
2. Os de manutenção e consulta das Bibliotecas de Macros;
3. Os válidos no sistema SOS.

GRUPO 1	{	Chamada de Macro	
		Concatenação	
		ENDM	
		GO TO IF	
		Declaração de variáveis (.GLB)	
		Atribuição (LET)	
		Não Opera (NOP)	
		. STATUS	
		Substituição	
GRUPO 2	{	Manutenção de	{ /INICIALIZE
		Biblioteca	{ .GUARDE
			{ .APAGUE
			{ .COMPRIMA
		Consulta a	{ .EXTERNO
		Biblioteca	{ .LISTE
GRUPO 3	{	Os que são válidos no SOS	/7/

### 3.3.2. DEFINIÇÕES

Os comandos do EMA podem ser decritos como.

$$\langle \text{Comando} \rangle ::= \langle \text{rótulos} \rangle \langle \text{bs} \rangle \langle \text{operando} \rangle \langle \text{bs} \rangle \langle \text{operador} \rangle \langle \text{bs} \rangle \langle \text{comentário} \rangle$$

É importante frisar que os comandos do grupo I só são válidos dentro das rotinas macros, com exceção das chamadas de macros, que são válidos tanto dentro como fora das rotinas. (A não observação causará erro semântico).

O subsistema EMA, admite dois tipos de rótulo: um do Assembler TI descrito em /7/ que pode ser modificado durante a expansão de uma macro, isto é, pode fazer parte dos parâmetros formais de uma rotina macro, e os rótulo EMA que são caracterizados por:

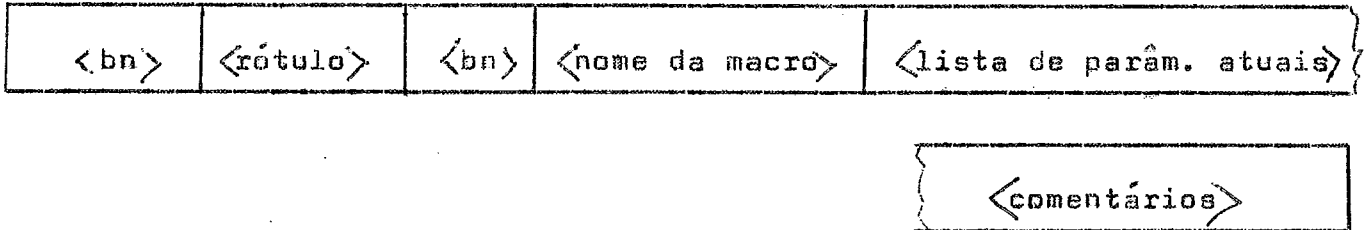
1. Não podem ser modificados durante uma expansão;
2. Podem ser colocados em qualquer coluna de cartão;
3. São formados por um identificador seguido de um caracter reservado do subsistema EMA;
4. Podem ser nulos;
5. São transparentes à expansão, isto é não constam do arquivo de saída.

### 3.3.3. Chamada de rotina macro

Este comando invoca a expansão do corpo da macro em um programa, seguindo as regras definidas nas macros a serem expandidas.

$\langle \text{chamada de rotina} \rangle ::= \langle \text{rótulo} \rangle \langle \text{bn} \rangle \langle \text{nome da macro} \rangle \langle \text{bs} \rangle$   
 $\qquad \qquad \qquad \langle \text{lista de parâm. atuais} \rangle \langle \text{comentários} \rangle$   
 $\langle \text{lista de parâm. atuais} \rangle ::= \langle \text{parâm. atual} \rangle / \langle \text{lista de pa} \rangle$   
 $\qquad \qquad \qquad \text{râm. atuais} \rangle \langle \text{parâm. atual} \rangle$   
 $\langle \text{parâm. atual} \rangle ::= \langle \text{identif} \rangle$

Formato do Comando



Mostraremos através do exemplo abaixo a generalidade do uso das chamadas das rotinas no EMA.

Seja a seguinte rotina macro

```

EXP3  MACRO P1,P2,P3,...,Pn
      - - - - -
      - - - - -
      - - - - -
      ENDM

```

Onde P<sub>1</sub>,P<sub>2</sub>,P<sub>3</sub>,...,P<sub>n</sub> são os parâmetros atuais.

Suponhamos a seguinte chamada. Ex P3 q<sub>1</sub>,q<sub>2</sub>,q<sub>3</sub>,...,q<sub>k</sub>, onde q<sub>1</sub>,q<sub>2</sub>,q<sub>3</sub>,...,q<sub>k</sub> são os parâmetros atuais. Esta chamada serpa processada observando as seguintes regras: o parâmento atual q<sub>1</sub>, substituirea no corpo da macro, o formal p<sub>1</sub>, o q<sub>2</sub> ao p<sub>2</sub>, e assim sucessivamente, isto é, univocamente por posição.

Sendo  $n$  e  $k$  o número de parâmetro formais e atuais, respectivamente, causará as seguintes implicações no uso do comando.

1. Se  $k > n$  os  $k-n$  parâmetros formais a partir de  $q_{n+1}$ , serão ignorados.
2. Se  $k < n$  serão gerados  $n-k$  parâmetros atuais nulos.

#### Exemplo 4

```
EXP4 MACRO    &A,&B,&C,&D,&E
```

```
-----  
-----  
-----
```

São válidas as seguintes chamadas:

- a) . EXP4 A0,A1,A2,A3,A4,A5
- b) . EXP4 , , , , , VAI
- c) . EXP4 SY
- d) . EXP4 1, , 3, FIM
- e) . EXP4 A1,A2,A3,A4,A5

#### Comentário

Como já foi mostrado, o EMA substitui os parâmetros atuais em branco por nulo. Daí teremos:

- a)  $k > n$  será eliminado o quinto parâmetro;
- b) Do primeiro ao quinto serão nulos;
- c) Do segundo ao quinto serão nulos;
- d) O segundo e o quinto serão nulos;
- e)  $k < n$  cada parâmetro atual tem seu correspondente formal.

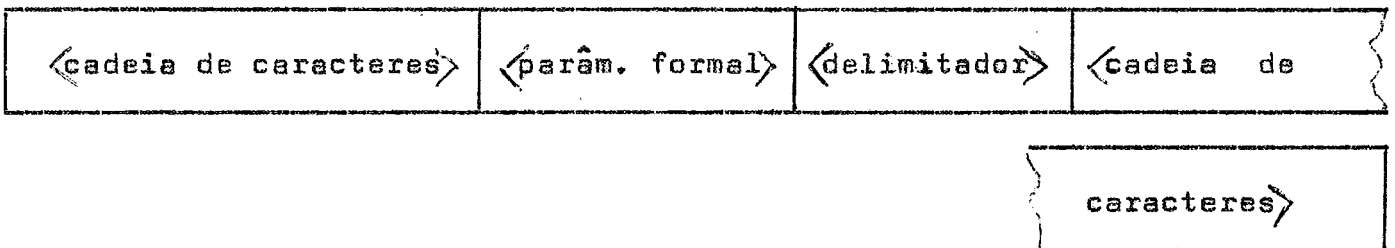
### 3.3.4. Comando de Concatenação

Este comando concatena qualquer cadeia de caracteres com um parâmetro formal ou vice-versa.

<Concatenação> ::= <cadeia de caracteres><parâm. formal>  
                   <delimitadores><cadeia de caracteres>  
                   | <concatenação><cadeia de caracteres>  
                   <parâmetro formal><delimitador><cadeia  
                   de caracteres>  
 <cadeia de caracteres> ::= <Qualquer sequência de caracte  
                                   res> |  $\Lambda$   
 <delimitadores> ::= # | .

O delimitador ". " é transparente à expansão, isto é, não cons<sub>t</sub>ta do arquivo de saída.

Formato do comando



Exemplo

```

EXP4 MACRO  &A1,&A2,&A3,&A4
i   ) LRS &A3.-&A4
      -----
      -----
      -----
ii  ) LB&A1
      MHD
  
```

```
iii ) &A2.B
      ADM
      ENDM
      -----
      -----
      .EXP4 A,LA,COEF,3
      -----
      -----

I   ) LRS COEF 3
      -----
      -----

II  ) LBA
      MHD

III ) LAB
      ADM
      -----
      -----

      END
```

### Comentários

No comando (i) temos como delimitadores no primeiro parâmetro' formal o ponto "." e no segundo o branco, e produz o comando (I).

No comando (ii), o delimitador é o branco e gera o comando(II).

No comando (iii) o delimitador é o ponto e produz (III).

### 3.3.5. Comando de transferência - IF

É um comando de transferência condicional na expansão.

<comando IF> ::= <rótulo EMA> <bn> <expressão booleana> <rótulo>

Formato do comando

<rótulo EMA>	<bs>	IF	<bn>	<expressão booleana>	<bn>	<rótulo>	<comentário>
--------------	------	----	------	----------------------	------	----------	--------------

A expressão booleana deve está contida entre parênteses. Se o resultado da expressão booleana for verdadeira transfere a expansão da rotina para o rótulo EMA, que seja igual ao rótulo indicado no comando.

O domínio de ação de transferência é restrita a macro que contém o comando. A não existência do rótulo de transferência, o controle de rotina passará para o próxima comando depois de ter dado uma advertência.

Operadores booleanos {  
  ≠ diferente  
  = igual a  
  < menor que  
  > maior que

Devemos salientar que o subsistema-EMA tem um limitador de ciclo, cujo valor máximo é igual a 50, para cada rotina macro. Ao atingir o máximo será dado uma advertência, e a expansão da macro é suprimida passando ao comando seguinte à chamada macro.

A finalidade deste limitação é para que não haja ciclos infinitos, que consumirá muito tempo de processamento inútil.

Caso o usuário necessita de mais de 50 ciclos em determinada rotina, pode ser removida a limitação imposta pelo subsistema através

do comando de atribuição LET(%LOOP=N) onde N é uma expressão aritmética ou número inteiro, que passara a ser limite máximo na rotina. Ao ser desativada a rotina, o limite voltará a ser 50 ciclos.

%LOOP é uma variável global definida no sistema. (O uso do comando de transferência IF vide no apêndice B).

### 3.3.6. Comando de Transferência - GO TO

O GO TO é um comando de transferência incondicional.

<comando GO TO> :: <bs> GO TO <bs> <rótulo> <bs> <comentário>

O domínio de ação do comando GO TO, é dentro da rotina na qual está contida, e suas limitações, para que não haja ciclos infinitos, são idênticos as do comando IF.

Formato do comando

<bs>	<GO TO>	<bs>	<rótulo>	<comentário>
------	---------	------	----------	--------------

### 3.3.7. Comando de atribuição LET

LET é um comando de atribuição onde o identificador do lado esquerdo do comando aritmético, recebe os resultados das operações, e guarda numa tabela de variável descrito em 2.3.9.

<comando LET> :: <bn> <rótulo EMA> LET <bn> ( <comando arit.> )  
<bs> <comentário>



Formato do comando

<bs>	<rótulo EMA>	LET	<bn>	((<comando arit.>)	<bs>	<comentário>
------	--------------	-----	------	--------------------	------	--------------

Exemplo:

a) LET (A=A\*B - ( B\*\*2 - 5 )/(A-B))

b) ETIQT# LET ( C=&N )

3.3.8. Comando NOP

É o comando que não altera a lógica da expansão, serve para inserir rótulo.

Formato do comando

<rótulo EMA>	<bs>	<NOP>	<comentário>
--------------	------	-------	--------------

Exemplo

```
EXP5  MACRO
      -----
      -----
      -----
NC#  NOP
LX  CAL  CARRY
      -----
      -----
      -----
ENDM
```

### 3.3.9. VARIÁVEIS DO SUBSISTEMA

Todas as variáveis do subsistema são inicializadas com zero. Há dois tipos de variáveis: GLOBAIS e LOCAIS.

As variáveis GLOBAIS guardam seus valores durante toda a expansão do programa, podendo, portanto servir de transmissão de parâmetro entre as macros que constituem o programa. São definidos pelo usuário de acordo com o parágrafo 2.3.10.

Enquanto que as variáveis LOCAIS só guardam o seu valor durante o tempo que a rotina macro está ativa, isto é, a cada chamada de rotina todas as variáveis LOCAIS são reinicializadas com zero exemplo no apêndice B.

### 3.3.10. DECLARAÇÃO DE VARIÁVEIS .GLB

É um comando que coloca na tabela de variáveis globais' (TVGL) as variáveis que guardarão seu valor após o término da expansão da macro.

<comando .GLB> ::= <bs>.GLB <bn> (<lista de variáveis globais>)  
<bn> <Comentário>

<lista de variáveis globais> ::= <variável global> | <variável global>, <lista de variáveis globais>

<variável global> ::= <identif>

### Formato do Comando

<bs>	.GLB	<bn>	( <lista de variáveis globais> )	<bn>	<comen - tário>
------	------	------	----------------------------------	------	--------------------

Este comando deve ser colocado na primeira rotina do programa ou na primeira rotina em que apareça uma das variáveis que consta da lista de variáveis globais. (Exemplo: Vide o apêndice B).

#### 3.3.11. Comando - .STATUS

<comando .STATUS> ::= <bs> : STATUS <bs> <comentário>

É o comando que dá o estado do programa expensor até aquele ponto que foi processado. É usado como depurador do programa e resultado nas seguintes informações.

- 1.) Diretório das Macros;
- 2.) Tabela de definição de Macro;
- 3.) Tabela de variáveis locais;
- 4.) Tabela de variáveis globais;
- 5.) Início do último bloco de informações a entrar na pilha;
- 6.) Pilha de blocos de informações.

As tabelas serão detalhadas no capítulo sobre Organização de Tabelas no EMA.

### 3.3.12: Comando de Substituição

Este comando substitui um identificador pelo seu valor 'número'. Se não for atribuído nenhum valor ao identificador o resultado será igual a zero.

<comando SUBSTITUIÇÃO> ::= <cadeia de caracteres> "<identif.>" <cadeia de caracteres>  
<Cadeia de caracteres> ::= <Qualquer sequência de caracteres> |  $\wedge$

Formato do Comando:

<cadeia de caracteres>	" <identif.> "	<cadeia de caracteres>
------------------------	----------------	------------------------

Exemplo

- i) DC /"K"
- ii) UM EQU "K"

Suponhamos K como o valor igual a 1 então ii) e i) ficará

DC /1  
UM EQU 1

### 3.4. Manutenção e Consultas as Bibliotecas

A manutenção e a consulta das bibliotecas são feitas através de comandos que criam, apagam, gravam, comprimem, listam rotinas contidas nas bibliotecas, listam o diretório das rotinas, ou carregam rotinas no programa em expansão.

Com exceção do comando que carrega rotinas macros no programa a ser expandido, os demais são feitos através de utilitário que serão descritos no apêndice /7/.

O subsistema EMA dispõe de duas bibliotecas de macros, para consulta: uma primitiva do usuário, e a outra do sistema. Esta terá acesso livre para todos os usuários.

### 3.4.1. Comando EXTERNO

Este comando carrega de uma das bibliotecas, no programa ' que está sendo expandido, as rotinas requeridas.

```
<comando .EXTERNO> ::= <bn> .EXTERNO ( <lista de nomes de
rotinas> ) <comentário>
<lista de nome de rotinas> ::= <nome de rotina> <lista de
nome de rotina> , <nome de rotina>
<nome de rotinas> ::= <identif.>
```

Inicialmente é pesquisado a biblioteca do usuário. Não obtendo sucesso, pesquisa a do sistema. Caso venha a obter novo insucesso emite uma mensagem de advertência.

<bs>	.EXTERNO	<lista de comando de rotinas>	<comentário>
------	----------	-------------------------------	--------------

#### IV. ORGANIZAÇÃO DAS TABELAS NO EXPANSOR DE MACRO ASSEMBLER

##### 4.1. Introdução

Neste capítulo descrevemos as tabelas usadas no subsistema EMA, suas características e suas funções. Devemos salientar inicialmente, que todas as tabelas são alocadas dinamicamente, isto é, variando' tamanho em função do programa a ser expandido. Serão ativadas enquanto durar a expansão e removidas assim que acabar todas as expansões pelo sistema.

##### 4.2. Tabela que guarda os parâmetros formais (ALA)

Esta tabela guarda os parâmetros formais, (como já foi visto, estes parâmetros, entram no sistema através do primeiro comando da rotina macro), e será usada posteriormente na codificação dos comandos, ou melhor na substituição dos parâmetros formais dos demais comandos, pelo seu índice na tabela, seguido do símbolo.

Esta codificação é feita para tornar mais fácil a substituição das variáveis atuais na rotina macro. A tabela ALA é destruída depois da codificação da macro.

##### Exemplo

```
Seja a rotina macro:  
SOMA  MACRO  &ARG1,&ARG2,&ARG3  
-----  
ENDM
```

Na codificação tabela ALA ficará

ÍNDICE	Parâm. formais
1	ARG1
2	ARG2
3	ARG3

TAB. 1  
tabela que guarda  
os parâmetros for-  
mais.  
ALA

#### 4.3. TABELA DE NOMES DE MACROS (TNM)

Esta tabela é o diretório das macros existentes no programa. Guarda em seus registros estruturas, com o nome da macro, o número de parâmetros formais da macro e sua posição dentro da tabela de definição de macro (TNM), cuja tabela será exposta neste capítulo.

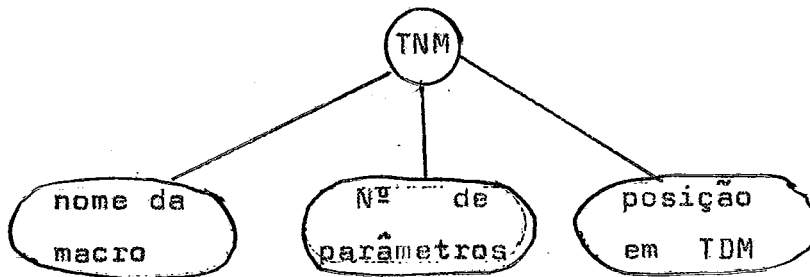


Fig. 3 - Estrutura dos Registros da TNM

Seu acesso é feito sequencialmente

#### 4.4. TABELA DE DEFINIÇÃO DE MACRO (TDM)

A TDM guarda as rotinas macros após serem codificadas. Isto é:

1. Substituir os parâmetros formais pelo símbolo " " seguido de seu índice na tabela ALA;
2. Marca os comandos IF e LET com o símbolo " " a fim de facilitar sua interpretação na hora de execução;
3. Colocar as expressões booleanas e aritméticas na forma polonesa posfixada /22/.

O acesso é feito através das informações contido em TNM, ou seja, acesso direto. Exemplo: sejam as macros:

```
SOMA      MACRO  &ARG1,&ARG2,&ARG3
           LRH   &ARG1
           LAM
           LRH   &ARG2
           &ARG3
           ENDM

REPETE    MACRO  &M,&N
           L# LET (I=&N)
           R# OUT / "I"
           LET (I=I+1)
           IF (I<&M) R
           IF (H>0) FIM
           A# LET(H=H+1)
           CAL DELAY
           IF (H<2) A
           GO TO L
           FIM #NOP
           ENDM

TESTE     MACRO
           RAR
           LDE
           LEA
           ENDM
```



A partir das rotinas macros geramos as seguintes tabelas:

TAB. 2 - Tabela de definições de Macros

ÍNDICE	ROTINAS MACROS CODIFICADAS
1	SOMA      MACRO    ARG1, ARG2, ARG3,
2	LRH#1
3	LAM
4	LRH#2
5	#3
6	ENDM
7	REPETE    MACRO    M,    N
8	a) LET(I,#1,=,)
9	OUT / "I"
10	a) LET(I,I,1,+,=,)
11	a) IF (I,#2,<, ) R
12	a) IF (H,0,>, ) FIM
13	a) LET(H,H,1,+,=,)
14	CAL DELAY
15	a) IF(H,2,<, ) A
16	GO TO L
17	NOP
18	ENDM
19	TESTE    MACRO
20	RAR
21	LDE
22	LEA
23	ENDM

TAB. 3 - Tabela de Nome de Macro (TNM)

Nome da rotina	Nº de parâm.	índice em TDM
SOMA	3	1
REPETE	2	7
TESTE	0	19

Note que na tabela 2 o símbolo seguido de um número são os parâmetros formais codificados, onde o número é a sua ordem na lista de parâmetros formais. Os comandos IF e LET, suas formas codificadas, são inicializadas por "a", e a expressão aritmética ou booleana é colocada na forma polonesa posfixada entre parênteses, com a finalidade de simplificar as suas análises, durante as expansões.

Observamos que na tabela não consta os rótulos "EMA" porque durante a codificação a processador, ao encontrar uma instrução com um rótulo EMA, o comando é guardado em TDM e o rótulo é marcado com o símbolo " ", seguido do índice do nome da rotina em TDM, e colocado na tabela de variáveis globais. (Mais detalhes daremos no parágrafo seguinte)

#### 4.5. TABELAS DE VARIÁVEIS LOCAIS E GLOBAIS

O Expansor de Macro Assembler tem duas tabelas de variáveis: uma das variáveis locais TBS e outra de variáveis globais. Os registros são constituídos das seguintes estruturas:

- a) estrutura dos registros das variáveis locais
- b) estrutura dos registros das variáveis globais.

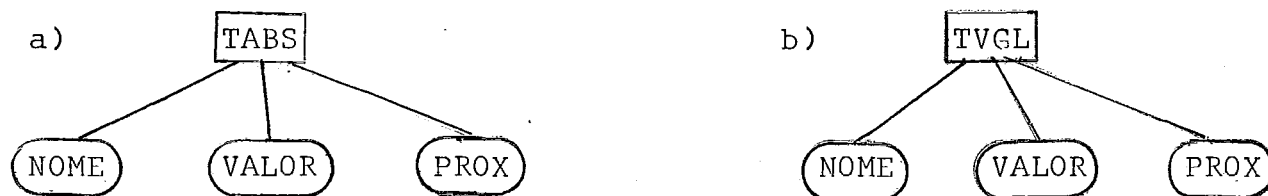


FIG.4

a) Estrutura dos registros TABS    b) Estrutura dos registros TVGL

onde NOME é o identificador da variável, VALOR é o valor numérico atribuído a variável e PROX é o índice de encadeamento da próxima variável que tenha o mesmo "hash". (Mais detalhes no decorrer deste capítulo).

A tabela de variáveis locais (TABS) é apagada todas as vezes que a rotina é desativada do programa de expansão, enquanto que a tabela de variáveis globais TVGL só é apagada quando o EMA retorna o controle ao núcleo central do SOS.

Uma particularidade da TVGL é que guarda, além das variáveis globais, os rótulos EMA depois de codificados, sendo que, nos campos reservados aos valores atribuídos as variáveis, são colocados os índices em TDM, correspondentes aos comandos que os continham.

#### 4.5.1. ORGANIZAÇÃO

A organização das tabelas de variáveis locais e globais foram implementadas segundo as técnicas "Scatter Index Table" que segundo a literatura, é uma das mais eficientes, /8/ /9/ /10/. Baseia-se na geração das chaves em função dos símbolos que não entrar na tabela, onde é usada uma função Hash /11/ /12/.

Estas chaves são associadas a um endereço e guardadas numa "Hash Table". Servirão como apontadores de cabeças de listas que se encontram em outra tabela chamada tabela de variáveis, cujos registros, contém campos para tributos das variáveis e um campo de encadeamento. Um outro importante parâmetro é fazer um apontador para o último registro colocado na tabela.

### CONSULTA A TABELA

Suponhamos que desejássemos inserir, copiar, ou alterar uma determinada variável.

Inicialmente, aplicando esta variável a uma função "hash" (descrita em /11/ função b), geraríamos uma chave, e através de uma pesquisa sequencial na "hash table" verificaríamos se a chave consta ou não.

### PRIMEIRA HIPÓTESE

A chave consta da tabela, então este fornece o apontador da cabeça da lista pesquisada na tabela de símbolo. Se a informação não consta do primeiro elo da lista, houve uma colisão. Então consultaremos o campo de encadeamento que contém o próximo elemento que tem o mesmo hash. Se não obtivermos sucesso houve outra colisão. Então seguiremos sucessivamente até encontrar o símbolo pesquisado ou um zero no campo de encadeamento, onde tomamos a decisão apropriada para o evento.

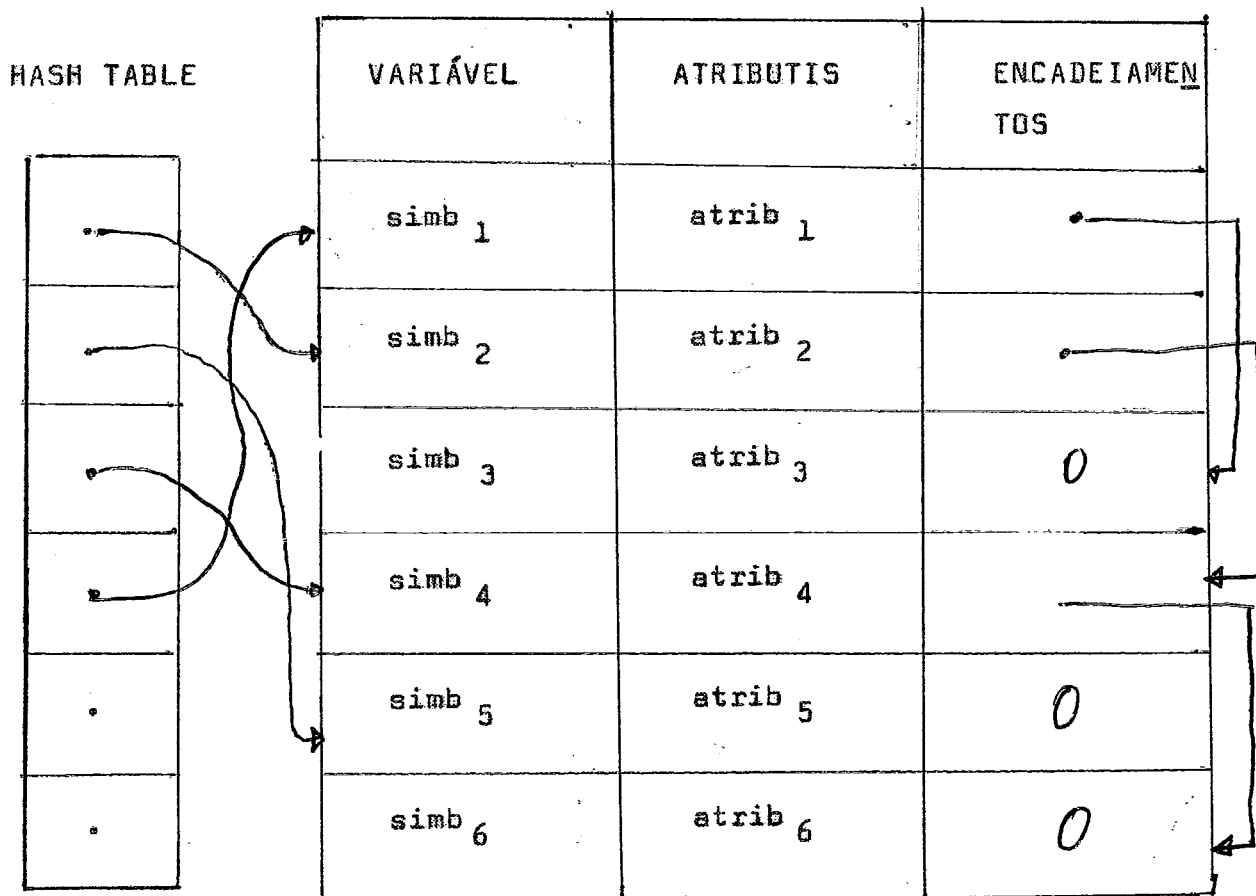
### SEGUNDA HIPÓTESE

A chave não consta de "hash table". Se for uma inserção então colocaremos as variáveis com seus atributos no primeiro registro livre da tabela de variáveis. Depois atualizaremos o "Hash Table", guardando

a chave gerada pela função "hash" e o endereço da variável Tabela de Variáveis. O apontador do último registro colocado nesta tabela, é incrementado de um.

Se for para alterar ou copiar, o valor da variável voltará com insucesso. Nestes casos tem a vantagem de pesquisar apenas a "Hash Table" que será sempre menor ou igual a tabela de variáveis.

TAB. 4 - Tabela de Variáveis



Com essa organização, o número de comparação para encontrar uma variável é sempre menor do que qualquer outro tipo convencional. Em média é dado por:

$$N_c = \frac{NRTS}{2 \times N_K}$$

$N_c$  é o número de comparações média;

NRTS é o número de registros na tabela de variáveis

$N_K$  é o número de chaves na "hash table".

Como vemos a média de comparações é inversamente proporcional ao número de chaves da "hash table". Daí o sucesso depende da função "hash" usada, /12/ /11/.

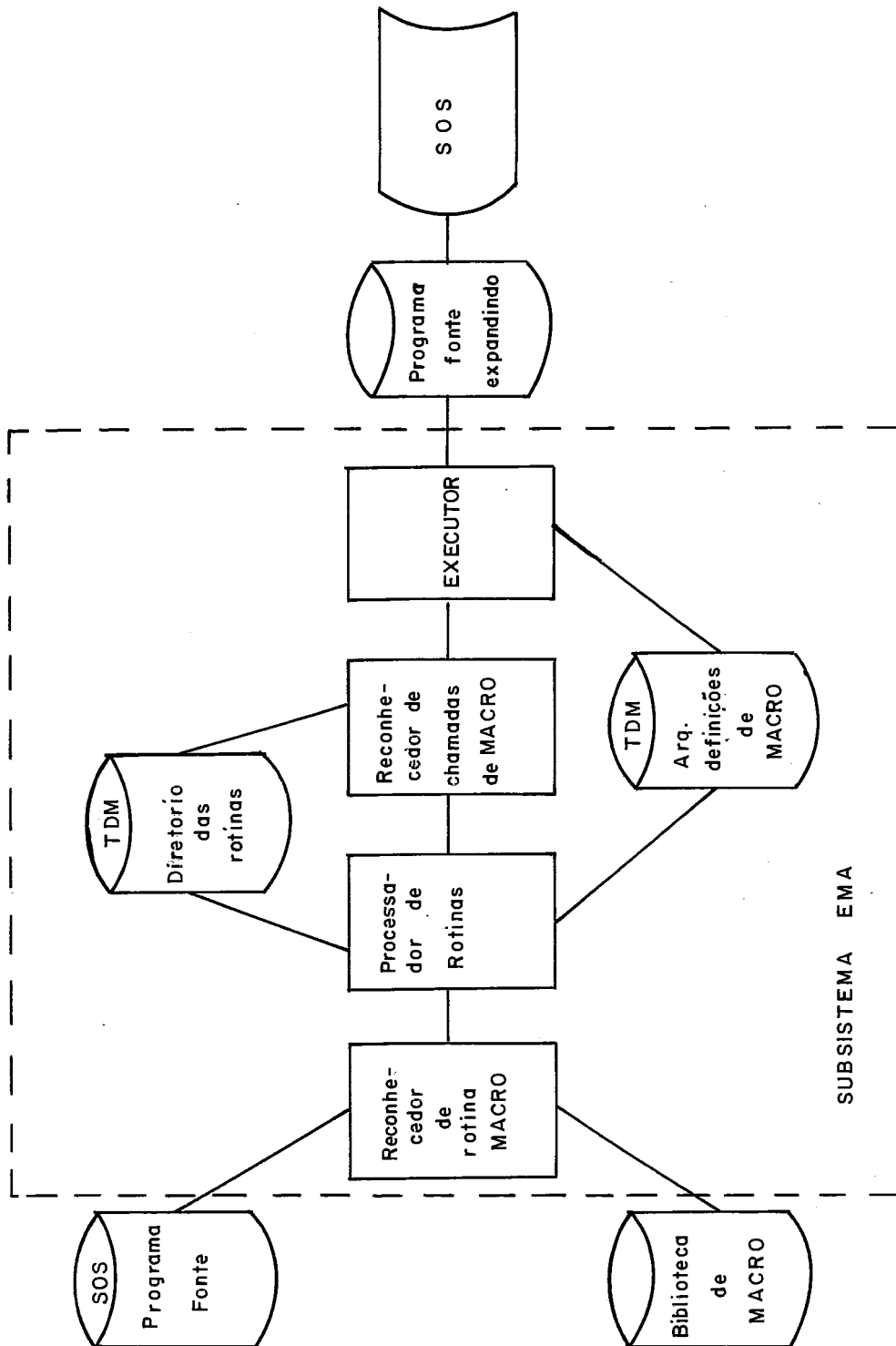


Fig. 5 O expansor de macro assembler com suas entradas e saída

## V. IMPLEMENTAÇÃO DO SUB SISTEMA EXPANSOR MACRO ASSEMBLER

### 5.1. INTRODUÇÃO

O sub sistema recebe o programa a ser expandido do arquivo criado pelo SOS. Codifica as rotinas macros contidas no programa e as que são referenciadas pelo programa na biblioteca. Depois coloca todas na tabela de definições de macro (TDM).

Em seguida será iniciada a expansão, que basicamente é constituído dos seguintes reconhecedores:

- 1) Reconhecedor de rotinas macro;
- 2) Reconhecedor de comandos EMA;
- 3) Reconhecedor e executor das chamadas macros;
- 4) Consultor de biblioteca de Macro;
- 5) Arquivo de programa após a expansão.

### 5.2. Reconhecedor de Rotinas Macros

Ao encontrar a palavra reservada MACRO, em um registro de entrada, este é colocado TDM e em seguida coloca no diretório TNM o nome, sua posição em TDM e número de parâmetros formais. Como também guarda na tabela ALA os parâmetros formais.

### 5.3. Reconhecedor dos Comandos EMA

Inicialmente analisa os comandos e verifica se está sintaticamente correto. Depois processa a codificação que consta dos seguintes passos:

- a) Guarda o rótulo EMA na tabela de variáveis globais;
- b) Substitui os parâmetros formais pelo símbolo seguido do índice do parâmetro na tabela ALA, que chamaremos, depois de



codificado, de parâmetro formal codificado;

- c) Os comandos EMA que tenham expressões aritmética ou booleana; as expressões são colocadas na forma polonesa posfixada, e acrescentado ao início do comando o símbolo " ", (Deve-se ressaltar que esta edição é para diminuir o tempo de processamento caso o comando esteja com erro semântico.);
- d) Coloca em TDM os comandos codificados.

#### 5.4. Reconhecedor e Executor das Chamadas de Rotinas Macro

Este reconhecedor ao encontrar uma chamada de rotina macro verifica se a mesma consta ou não do diretório TDM. Se não consta é emitida uma advertência e volta a ler o próximo comando. Se consta, então é gerado uma pilha de blocos de informações que irá gerenciar a expansão. Cada bloco contém:

- a) O índice do início do último bloco que entrou na pilha;
- b) O índice da última instrução executada em TDM;
- c) Uma lista dos parâmetros atuais da chamada da rotina.

#### OBSERVAÇÕES:

- 1) o ítem "a" do primeiro bloco difere dos demais porque ele contém "flag" de pilha vazia.
- 2) Cada bloco corresponde a uma chamada.

Ao aparecer uma chamada é criado um bloco de informações como foi descrito. Se durante a expansão for encontrado uma segunda chamada, este bloco será guardado na pilha, nas condições em que se encontra, e é criado um novo bloco, que passa a gerenciar a expansão e assim sucessivamente até encontrar o fim da última rotina que está sendo expandida

Testamos então se é o último bloco. (vide neste parágrafo observação 1) Se for terminaremos a expansão, caso contrário, desempilhamos um bloco e continuamos o processo até a pilha ficar vazia.

Conceitualmente este tratamento é conhecido como LIFO (Last-in first-out), ou seja, o último a entrar é o primeiro a sair. Para dar melhor idéia de como funciona a pilha de bloco, vamos mostrar em um exemplo recursivo.

Seja calcular  $n!$

Exemplo:

```
FACT      MACRO &N
          . GLB(F)
          LET( F = 1)
          .FACT1 &N
          DC "F"
          ENDM
FACT1     MACRO &N
          IF (&N = 0) A
          LET(NUM = &N)
          LET (F = F x NUM)
          LET (NUM = NUM - 1)
          ..FACT1 "NUM"
          A# NOP
          ENDM
FACT      3
END
```

TAB. 5 - Tabela de definições de Macro

ÍNDICE	COMANDOS CODIFICADOS
1	FACT           MACRO N
2	ⓐ LET(F,L,=,)
3	.FACTI #1
4	DC " F "
5	ENDM
6	FACT1           MACRO N
7	IF (#1,0,=,) A
8	ⓐ LET(NUM,ⓐ,1,=,)
9	ⓐ LET(F,F,NUMx,=,)
10	.FACT1 "NUM"
11	NGP
12	ENDM
13	-----

O reconhecedor de chamada de rotina macros ao encontrar o comando .FACT 3 será criado o primeiro bloco de informações, (Por questão puramente didática a pilha está com o sentido inverso da real. O último bloco a entrar é o que tem o maior índice na pilha).

TAB. 6 - Pilha de Informações

blocos	índice	pilha inf.	comentários
1	1	-1	"Flag" que indica pilha vazia
	2	3	Índice dos comandos efetuados em TDM
	3	3	Variável da primeira chamada
2	4	1	Índice do início do ult.bloc. a ent. na pilha
	5	6,7,8,9,10	Índice dos comandos efetuados em TDM
	6	3	Variável da segunda chamada
3	7	4	Índice do inic. do últ.bloc.a ent. na pilha
	8	6,7,8,9,10	Índice dos comandos efetuados em TDM
	9	2	Variável da terceira chamada
4	10	7	Índice do inic. do ult.bloc.a ent. na pilha
	11	6,7,8,9,10	Índice dos comandos efetuados em TDM
	12	1	Variável da quarta chamada

blocos	índice	pilha inf.	comentários
5	13	10	índice do inc.do ult. a ent. na pilha
	14	6,7,8,9,10	índice dos comandos efetuados em TDM
	15	0	Variável da quinta chamada
6	16	13	índice do inc. do ult. bloc. a ent. na pilha
	17	6,7,8,9,10	índice dos comandos efetuados em TDM

Observando a pilha notamos que os últimos comandos executados em TDM em cada bloco são iguais a 10, (devido a lógica da expansão) com exceção do primeiro e do sexto. O primeiro está fora da recursão, enquanto que o sexto a lógica faz sair do "loop" ao encontrar ENDM no comando 12º. Começa então desempilhando o bloco em questão, executa os comandos 11º e 12º, voltando a encontrar o comando ENDM. Então desempilha o bloco 4º e segue o processo anterior igualmente para os blocos 3º e 2º no 1º bloco, o próximo comando a ser executado, será o 4º DC "F" e no 5º encontramos um ENDM. Termina a expansão da rotina, voltando a ler outro comando no arquivo da entrada.

### 5.5. Consultor de Biblioteca

O processador de rotinas macros dispõe de duas bibliotecas para serem consultadas: uma do usuário e a outra do sistema. A do usuário é uma biblioteca privada onde somente ele tem acesso as informações, enquanto que na biblioteca do sistema é livre o acesso para consulta a todos os usuários.

O consultor de biblioteca é o processador do comando .EXTERNO. Ao encontrar o comando .EXTERNO, pega cada rotina da lista e verifica se consta da biblioteca do usuário ou da biblioteca do sistema. Se a proposição for verdadeira, isto é, a rotina requerida consta de uma das bibliotecas, então carrega a rotina no programa que está sendo expandido. Caso contrário, emite uma advertência da inexistência dela. E assim segue sucessivamente até processar a última rotina da lista. Devemos ressaltar que a primeira biblioteca a ser consultada é a do usuário.

## VI. CONCLUSÕES

Em virtude de não se ter implementado no sistema Burroughs B 6700 um expensor de macro, não foi possível tirar conclusões comparativas. Entretanto procuraremos dar alguns subsídios para futuras implementações.

### 6.1. Escolha da linguagem

O EMA foi implementado no computador Burroughs B 6700 do NCE/UFRJ, usando como linguagem de programação o PL/I, pelas seguintes razões:

1. Grandes recursos de programação para executar o tipo de trabalho proposto;
2. Tempo de execução.

Sob o ponto de vista de programação o PL/I nos dá grandes facilidades para tratamento de dados não numéricos, por apresentar um conjunto de "built-in" com este fim, o que não acontece com as outras linguagens existentes atualmente no NCE/UFRJ. Dessas facilidades surge um inconveniente, que é o seu tempo de compilação tendo em vista que, em relação as similares é relativamente maior. Entretanto, não consideramos com grande relevância este fato porque o subsistema EMA ficará residente no disco depois de compilada.

Sendo assim um fator importante é o tempo de execução. Inicialmente tentamos, na literatura especializada, estudos comparativos sobre tempo de execução entre as linguagens existentes no B/6700, porém não obtivemos sucesso. Partimos para medir experimentalmente, e para isto, fizemos um programa em PI/I e outro em ALGOL contendo apenas comandos de atribuições, depois de processá-los no B/6700 obtivemos os seguintes resultados:

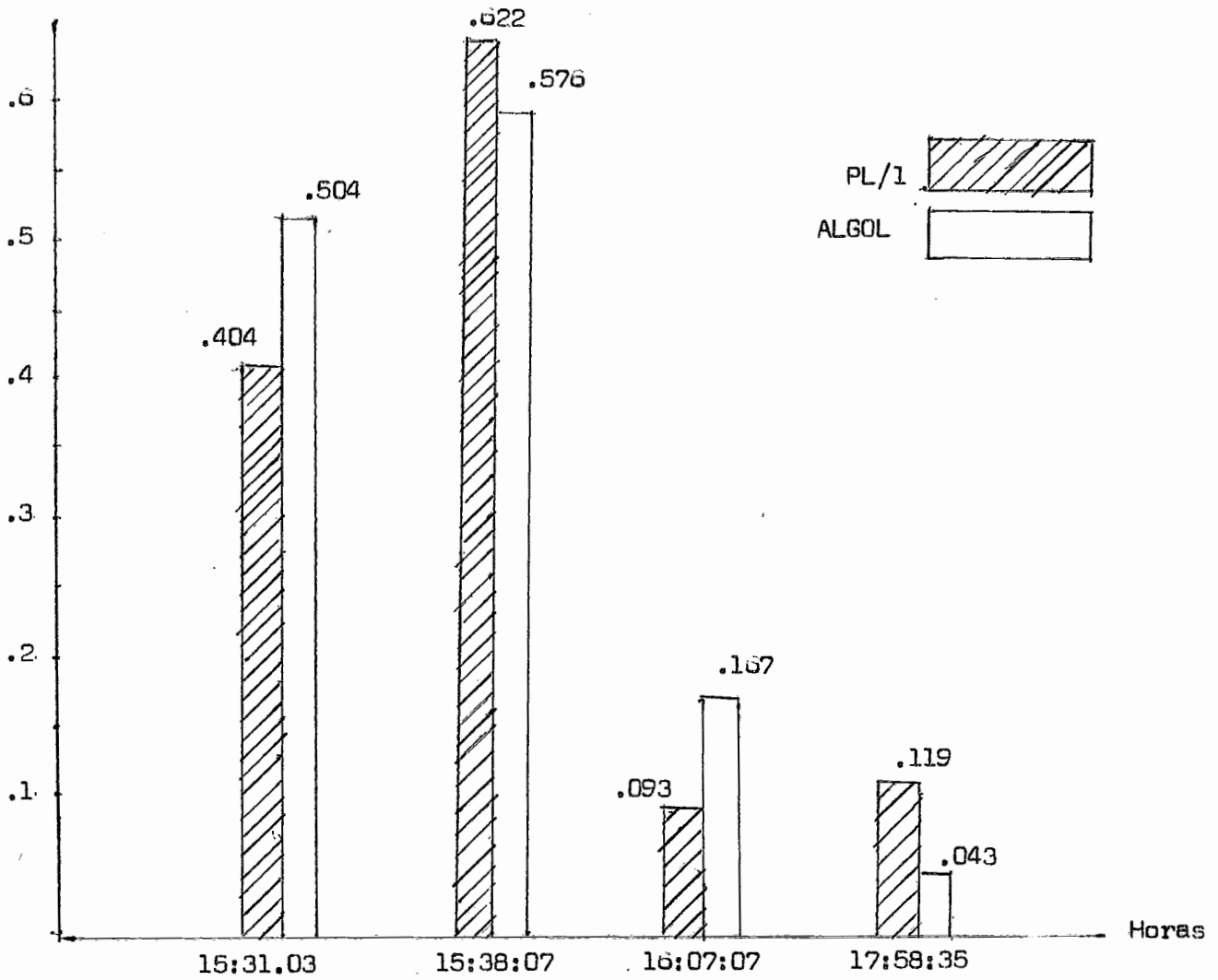


FIG. 6 - Tempo de Execução PL/I e ALGOL



Pelo gráfico podemos tirar as seguintes conclusões:

1. Existe uma grande flutuação no tempo de execução de um horário para outro. Este fato ocorre em função do número de jobs que estão sendo processados naquele mesmo instante.
2. Em média, o tempo de execução não difere significativamente entre o ALGOL e o PL/I.

Ao final do trabalho verificamos que o PL/I do B/6700. Por enquanto não se pode fazer ligações com as outras linguagens existentes no B/6700. Exemplo: FORTRAN ALGOL e vice-versa. Para sanar esta deficiência usamos arquivos auxiliares de ligações do subsistema EMA com o sistema SDS.

## 6.2. Observações finais e sugestões para futuras implementações

Como já era esperado, o processamento das macros torna a linguagem básica um pouco mais lenta, devido as análises feitas a cada comando processado. No EMA procuramos diminuir o número de comparações referenciando as chamadas de macros com um ponto seguido do nome da rotina. Este procedimento limitou a generalidade dos comandos de extensões, porém diminuiu o seu tempo de execução.

O EMA é um processador de macros semi-independente da linguagem de programação básica que está sendo simulada. No nosso caso particular, usamos o assembler do Terminal Inteligente (TI). Para o Expansor de Macro Assembler ser usado com outra linguagem básico, deve ser feita as seguintes modificações:

1. Substituir os delimitadores ( "/", ".", ' "x", e o END) pelos seus equivalentes na linguagem a ser implementada Vide /7/.
2. Observar o uso dos arquivos de entrada e ' saída do EMA.

## APÊNDICE A

### LINGUAGEM DE EXTENSÃO

Como uma das aplicações de um processador de macro é estender uma linguagem tomada de base. Como por exemplo de aplicação técnica exporemos, em linhas gerais, o funcionamento do processador de macro LIMP (Lânuage Independent Macro Processor).

Uma das notáveis características é o uso do padrão de comparação (Template Matching) introduzido por Wilker M.V. /3/.

O padrão de comparação é equivalente no EMA ao primeiro comando de uma rotina macro, só que, no caso LIMP o primeiro comando da rotina é o próprio comando a ser estendido na linguagem com suas variáveis substituídas por um símbolo qualquer. Exemplo: O comando estendido A B+C ficaria %=+%%. Supondo % o símbolo usado a esta é guardado na tabela de padrão de comparação. A chamada é feita através do comando.

Como ilustração do que foi exposto daremos um exemplo usando o processador LIMP.

Tomando como linguagem base o Assembler do SOS, vamos estendê-la para admitir comando do tipo A=B+C. Inicialmente, na tabela de comparações de padrões, colocamos padrão %=+% seguido da rotina:

padrão de comparação

i) %=+%

ou

Template Matching

ii) LRH 1

LAM

iii) LRH 2

ADM

iv) LRH 3

LMA

ENDM fim de macro

v)  $A=B+C$  chamada de macro

LRH A

LAM

LRH B

LDM

LRH C

LMA

Podemos observar que em (i) da comparação é igual a chamada' (v) menos variáveis, que são usadas como argumento de substituição em (ii) e (iv), enquanto (v) é o comando estendido que funciona como chamada de rotina com os parâmetros atuais A,B e C.

O LIMP admite muitas facilidades, quais sejam: recursividade, transferência condicional e incondicional, comandos aritméticos etc, porém sua grande desvantagem é ser muito lento.

APÊNDICE B

USO DOS COMANDOS QUE CONTROLAM A EXPANSÃO

Exemplo b - 1

Uso dos comandos LET e das substituições. Seja o programa :

```
          EXP8          MACRO &X,&Y
i)          LET(Z=&X+3+Z)
ii)         DC "Z"
iii)        LET(&Y=Z x &X)
iv)         DC "&Y"
           ENDM
           -----
           -----
           .EXP8 10, V
           .EXP8 5, G
           END
```

Resultado da primeira chamada:

```
          DC 13
          DC 130
```

Resultado da segunda chamada:

```
          DC 8
          DC 40
```

Na segunda chamada as variáveis serão reinicializadas com Zero, e seguem o mesmo processo mudando apenas as variáveis formais pelas formas atuais:

```
O comando i produzira z 8
"      "  ii           "   DC 8
"      "  iii          "   Y 40
"      "  iv           "   DC 40
```

Exemplo b - 2:

Aplicação de declaração de variável global, seja a macro

```
EXP9      MACRO  &X,&Y
           .GLB(Z)
           LET(Z=X +3+Z)
           DC "&Y"
           ENDM
           -----
           -----
           .EXP9 10, V
           .EXP9 5, G
```

Resultado da primeira chamada:

```
DC 13
DC 130
```

Resultado da segunda chamada:

```
DC 21
DC 105
```

### Comentários

As duas rotinas dos exemplos b-1 e b-2 só diferem entre si porque na última, foi declarada como variável global o z. Teremos então na primeira chamada da EXP8 resultados iguais a EXP9.

Entretanto, na segunda chamada, todas as variáveis serão reiniciadas com zero, exceto z que permanecerá com seu valor obtido na úl-

tima chamada. Daí termos:

```
O comando a) LET(Z=5+3+13)  donde Z = 21
"      "      b) DC 21
"      "      c) LET(G=21 x 5)  donde V = 105
"      "      d) DC 105
```

Exemplo b - 3:

Uso dos comandos de transferência

```
REPETE      MACRO  &N,&M
  L# LET(I &N)
  R# OUT / "I"
    LET(I=I+1)
    IF (I>&M) R
    IF (H<0) FIM
  A LET(H=H+1)
    CAL DELAY
    IF (H>2) A
    GO TO L
FIM# NOP
  ENDM
  .REPETE 10, 13
  -----
  -----
  ENDM
```

Resultado em

```
OUT/10
OUT/11
OUT/12
CAL DELAY
CAL DELAY
OUT/10
OUT/11
OUT/12
```

Para obter esta expansão o processador de macro chamou a rotina de transferência 10 vezes. Supondo que desejássemos executar a chamada .REPETE 5, 30. Neste caso, em condições normais, não executaria toda a expansão como já foi justificada em 2.3.5. Para resolver este impasse deve ser colocado antes do primeiro laço de transferência da rotina ativa (ou de acordo com a conveniência do usuário) onde o comando 'LET(%LOOP: N) N é o número máximo de transferência.



APÊNDICE C

FLUXOGRAMA DO EXPANSOR DE MACRO ASSEMBLER

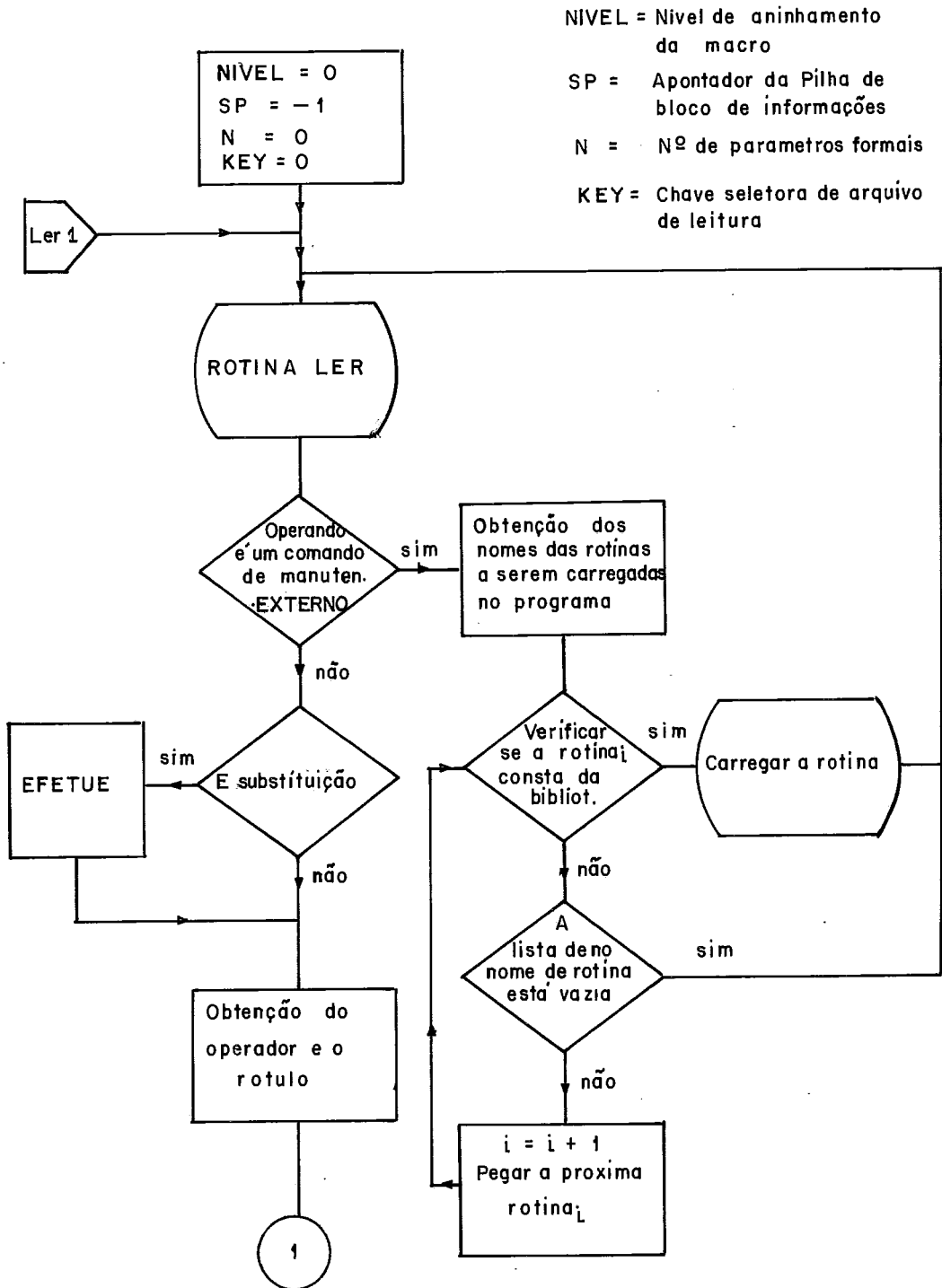


Fig. 7 Fluxograma do expensor de macro assembler

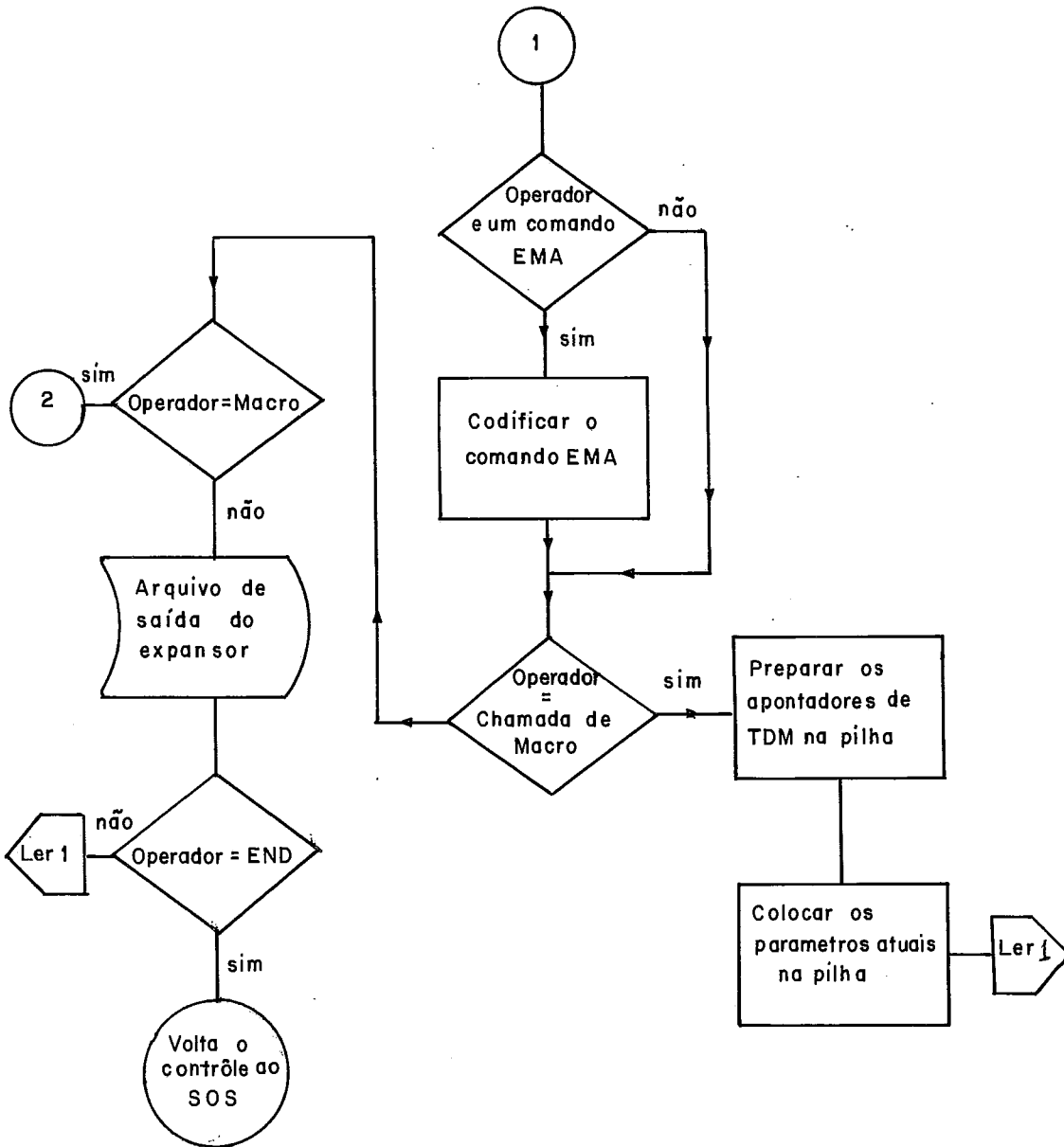


Fig. 8 Continuação da anterior

KEY Chave seletora de leitura

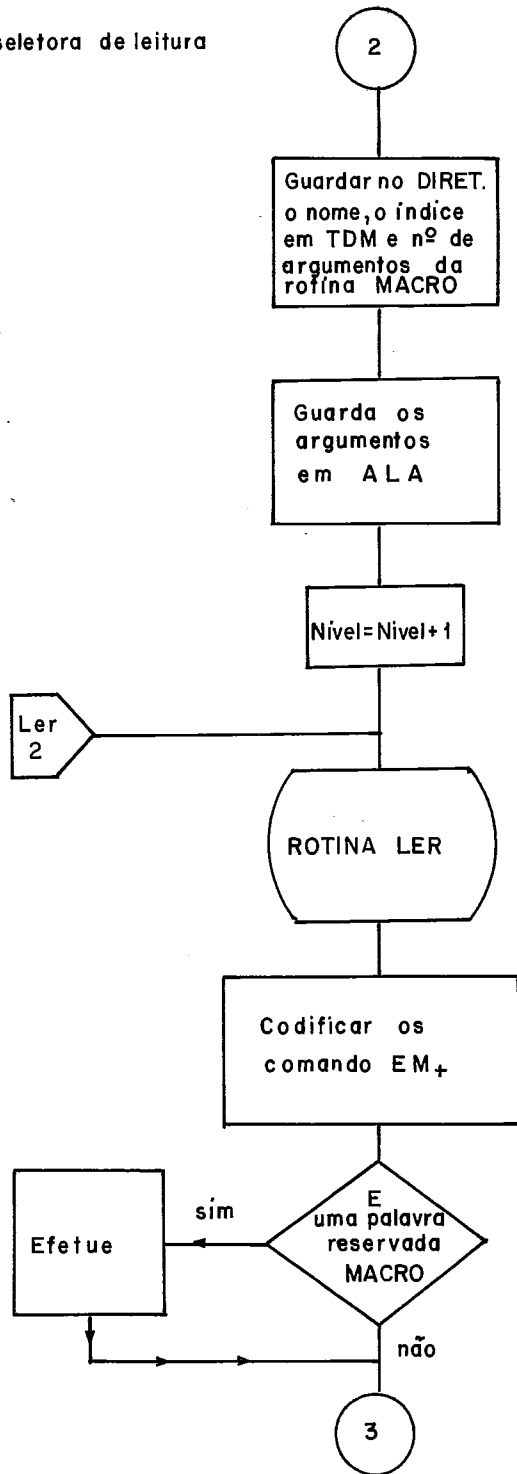


Fig 3 Continuação da anterior

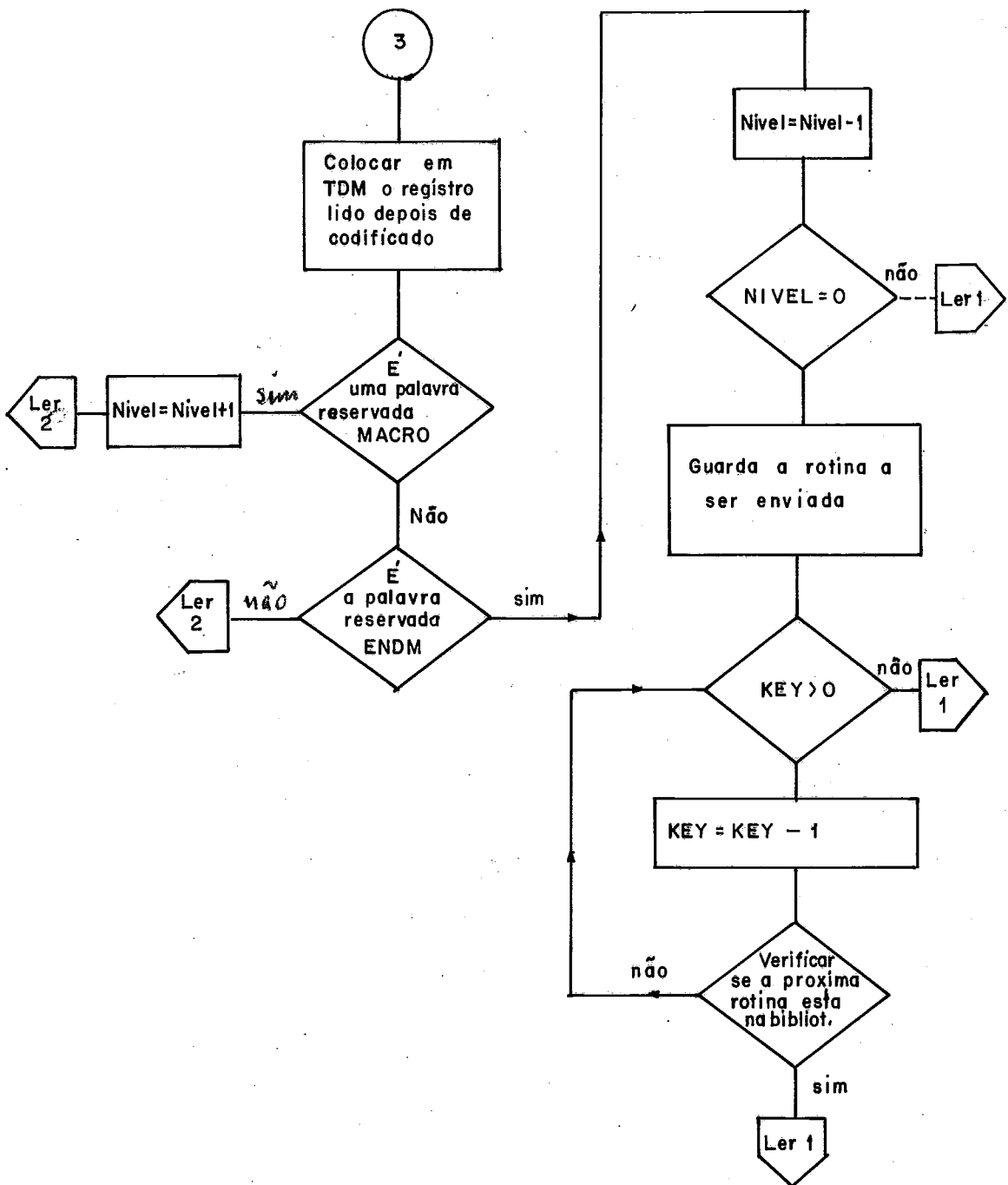


Fig 10 Continuação da anterior

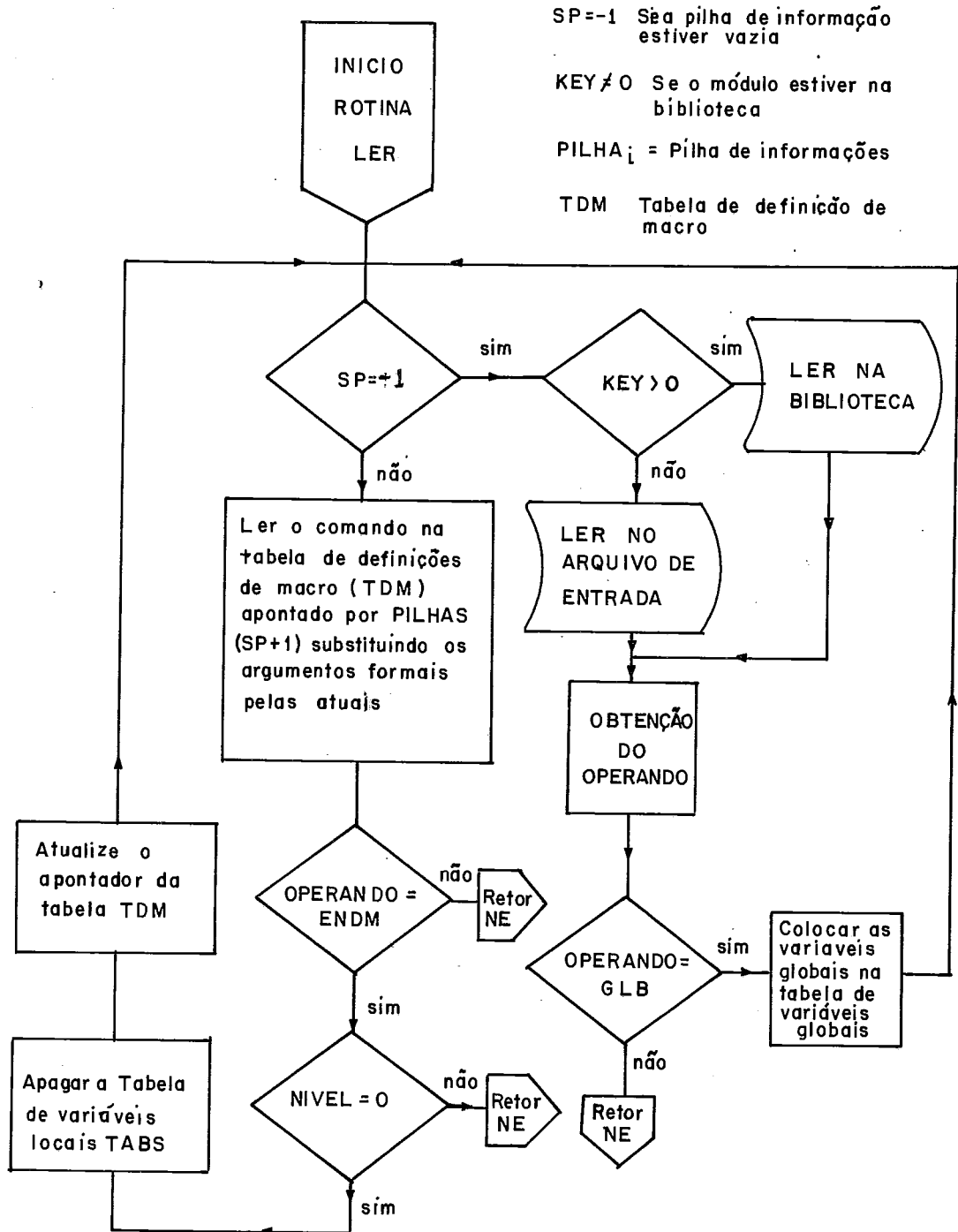


fig 11 Rotina de leitura

Apêndice D

Programas testes usando as facilidades do Expansor Macro Assembler. Neste primeiro programa observaremos que:

- a) Do comando 3 ao 7 constitui a rotina ENDERE1
- b) " " 9 " 13 " " " ENDERE2
- c) " " 15 " 19 " " " ENDERE4
- d) " " 21 " 25 " " " ENDERE3
- e) " " 23 " 33 " " " INCRE
- f) " " 35 " 39 " " " INCRS
- g) " " 40 " 44 " " " FACT
- h) " " 45 " 51 " " " FACT1

Note que a descontinuidade na numeração dos comandos é devido ao fato de se ter usado o comando SALT, na codificação do programa na linguagem do Expansor do Macro Assembler, vide /7/.

Pela lógica do programa, a primeira expansão a ser feita será através do comando 53. FACT 7, fará o fatorial de 7, cujo resultado será colocado no comando 151. Pela característica recruciva da rotina FACT1, daremos mais detalhe no próximo exemplo.

Observe que logo após das demais chamadas, os comandos ' que as precedem têm na coluna LC o sinal "+", denotando que o comando teve origem de uma expansão.

STM LC P1 P2 P3 P4 P5

```

1 SISTEMA EXPANSOR DE MACRO ASSEMBLER
2 VERSAO I/75
3 MACRO &A,&B,&C, &D ,&E,&LABEL
4 &LABEL LRS &A
5 &B &C
6 &D &E
7 ENDM

```

```

9 ENDERE2 MACRO &A,&B,&C,&LABEL
10 &LABEL LRS &A
11 LAM &B
12 &C
13 ENDM

```

```

15 ENDERE4 MACRO &A,&B
16 SUB
17 SMA
18 LAM &B
19 ENDM

```

```

21 ENDERE3 MACRO &A,&B,&C
22 LRS &A
23 LAM &B
24 LBI
25 ENDM

```

```

27 INCR MACRO &ARG1,&ARG2
28 LRS &ARG1
29 LAM &ARG2
30 LBI
31 LADMA
32 LLENDM

```

```

35 INCRS MACRO &X1,&Y1,&X2,&Y2,&Y3
36 INCRE &X1,&Y1
37 INCRE &X2,&Y2
38 INCRE &X1,&Y3
39 ENDM

```



```

STM LC P1 P2 P3 P4 P5
40 MACRO &N
41 MACT(F)
42 LFACT(F=1) &N
43 LFACT1 &N
44 ENDMRO &N
45 IFF(&NUM=&N) A
46 IFF(&NUM=&N)
47 IFF(&NUM=&N)
48 IFF(&NUM=&N)
49 IFF(&NUM=&N)
50 IFF(&NUM=&N)
51 IFF(&NUM=&N)
52 IFF(&NUM=&N)
53 IFF(&NUM=&N)
54 IFF(&NUM=&N)
55 IFF(&NUM=&N)
56 IFF(&NUM=&N)
57 IFF(&NUM=&N)
58 IFF(&NUM=&N)
59 IFF(&NUM=&N)
60 IFF(&NUM=&N)
61 IFF(&NUM=&N)
62 IFF(&NUM=&N)
63 IFF(&NUM=&N)
64 IFF(&NUM=&N)
65 IFF(&NUM=&N)
66 IFF(&NUM=&N)
67 IFF(&NUM=&N)
68 IFF(&NUM=&N)
69 IFF(&NUM=&N)
70 IFF(&NUM=&N)
71 IFF(&NUM=&N)
72 IFF(&NUM=&N)
73 IFF(&NUM=&N)
74 IFF(&NUM=&N)
75 IFF(&NUM=&N)
76 IFF(&NUM=&N)
77 IFF(&NUM=&N)
78 IFF(&NUM=&N)
80 IFF(&NUM=&N)
81 IFF(&NUM=&N)
82 IFF(&NUM=&N)
83 IFF(&NUM=&N)
84 IFF(&NUM=&N)
85 IFF(&NUM=&N)
86 IFF(&NUM=&N)
87 IFF(&NUM=&N)
88 IFF(&NUM=&N)
89 IFF(&NUM=&N)
90 IFF(&NUM=&N)
91 IFF(&NUM=&N)
92 IFF(&NUM=&N)
93 IFF(&NUM=&N)
94 IFF(&NUM=&N)
95 IFF(&NUM=&N)
96 IFF(&NUM=&N)
97 IFF(&NUM=&N)
98 IFF(&NUM=&N)
99 IFF(&NUM=&N)
100 IFF(&NUM=&N)

```

```

36 DC 2E 01
3E 00
0E 0A
36 DC 2E 01
3E 00
E8 F4
36 DB 2E 01
F8 00
60 80 01
36 DB 2E 01
C7

```

```

+0150
+0154
+0156
+0158
+015C
+015D
+015E
+0161
+0162
+0166
+0167
+0168
+016A
+016F

```

STM	LC	P1	P2	P3	P4	P5	Code
84	+0170						JTC SUB FIMDIV
85	+0171	91	A3	01			LMA ENDERE1 NCARY, AI, 1, ADM
86	+0174	F8					L NCARY 1
87	+0175	36	DC	2E	01		LADM
88	+0179	06	01				LADM
89	+0178	87					LADM
90	+0178	F8					LADM
91	+017C	44	6B	01			LADM
92	+017D	44	6B	01			LADM
93	+0180	36	DB	2E	01		JMP VOLTA
94	+0180	0E	0A				JMP VOLTA
95	+0184	0E	0A				JMP VOLTA
96	+0185	81					JMP VOLTA
97	+0187	F8					JMP VOLTA
98	+0188	60	98	01			JMP VOLTA
99	+0189	60	98	01			JMP VOLTA
100	+018C	36	DC	2E	01		JMP VOLTA
101	+0190	C7	0E	01			JMP VOLTA
102	+0190	0E	01				JMP VOLTA
103	+0191	0E	01				JMP VOLTA
104	+0193	91					JMP VOLTA
105	+0193	F8					JMP VOLTA
106	+0194	44	80	01			JMP VOLTA
107	+0195	44	80	01			JMP VOLTA
108	+0198	36	DC	2E	01		JMP VOLTA
109	+0198	C7	0E	01			JMP VOLTA
110	+019C	0E	01				JMP VOLTA
111	+019D	0E	01				JMP VOLTA
112	+019E	91					JMP VOLTA
113	+019F	F8					JMP VOLTA
114	+01A0	0E	0A				JMP VOLTA
115	+01A1	0E	0A				JMP VOLTA
116	+01A3	36	DB	2E	01		JMP VOLTA
117	+01A7	C7	F4				JMP VOLTA
118	+01A7	F8					JMP VOLTA
119	+01AA	F8					JMP VOLTA
120	+01AB	11	B4	01			JMP VOLTA
121	+01AB	68	E6	01			JMP VOLTA
122	+01AC	44	58	01			JMP VOLTA
123	+01AE	44	58	01			JMP VOLTA
124	+01B1	36	DC	2E	01		JMP VOLTA
125	+01B4	C7					JMP VOLTA
126	+01B8						JMP VOLTA
127	+01B8						JMP VOLTA

STM	LC	P1	P2	P3	P4	P5	RET	RETORNA
128	+01B9	07						DECHL
129	01BA	05				0000		
130	01BB	DD	E6					
131	01BD	44	58	01				
132								ROT1
133								JTEST,1
134	+01C0	36	DB	2E	01			NCARY,4,8,8
135	+01C4	C7	01					JTEST
136	+01C5	0E						JTEST
137	+01C7	81						JTEST
138	+01C8	F8						JTEST
139								1
140	+01C9	36	DC	2E	01			NCARY,4
141	+01CD	C7						NCARY
142	+01CE	0E	04					4
143	+01D0	81						JTEST,8
144	+01D1	F8						JTEST
145								JTEST
146	+01D2	36	DB	2E	01			JTEST,8
147	+01D6	C7						JTEST
148	+01D7	0E	08					JTEST
149	+01D9	81						JTEST
150	+01DA	F8						JTEST
151	01DR	B0						JTEST
152	01DC	00						JTEST
153								NCARY DC 5040
								DC 0
								END

MEMORIA UTILIZADA 141 BYTES MONTAGEM BFM SUCEDIDA 0 ADVERTENCIA(S) 0 ERROS(S) 0  
 OPCOES EM EFEITO NXREF LIST

TEMPO DE EXECUCAO DO PASSO1/PASSO2 9.227/ 8.477 SEG

Este próximo exemplo tem como finalidade mostrar alguns detalhes da recursividade do subsistema EMA.

O problema teste será achar o fatorial de 4. Inicialmente usamos o comando .STATUS para que seja listado passo a passo, o estado do programa. Ao executar o comando 25, isto é a chamada da rotina, FACT esta chama a rotina FACT1, e esta última por sua vez chama a se próprio, entrando em recursão.

Obtivemos as seguintes informações:

- a) Do 33 ao comando 40, temos o diretório no qual consta os nomes das rotinas usadas pelo programa com seus respectivos endereço (índice) na Tabela de definição de macro e seu numero de argumentos.
- b) Do comando 42 ao 64 a Tabela de definições de Macros
- c) Do comando 66 ao 81 a tabela de variáveis locais e globais que são constituídas, por índices, por nomes das variáveis, por seus valores numericos e pelos apontadores para o proximo valor que tenha o mesmo "hash".
- d) No comando 86 temos o índice do início do bloco que está sendo processada como já foi descrito.
- e) Em seguida uma pilha de blocos de informações que varia de tamanho dependendo da lógica do programa.

Estas informações serão listadas cada vez que executar o comando .STATUS, note que a informação dada pelo índice do bloco na pilha de gerencia de expansão são: 4,7,10,13,16,13,10,7,4,1,-1; assumidas durante o processamento do programa exemplo, mostrando assim o empilhamento e o desempilhamento dos blocos de informações.

STM LC P1 P2 P3 P4 P5

SISTEMA EXPANSOR DE MACRO ASSEMBLER  
VERSAO I/75

1 2

```
FACT MACRO &N  
  .GLB(F)  
  LET(F=1)  
  .FACT1 &N  
  SALT  
  .STATUS  
  ENDM
```

4 5 6 7 8 9 10

```
FACT1 MACRO &N  
  IF(&N=0) A  
  LET(NUM=&N)  
  LET(F=F*NUM)  
  LET(NUM=NUM-1)  
  SALT  
  .STATUS  
  .FACT1 "NUM"  
  A#NOP  
  SALT  
  .STATUS  
  ENDM
```

12 13 14 15 16 17 18 19 20 21 22 23

```
.FACT 4  
.FACT1 4
```

25 26

29

31

33

35

37

39

40

42

44

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

ESTADO DO PROGRAMA

DIRIGIO DAS ROTINAS MACROS

NOME	INDICE	N. ARGUMENTOS
FACT	1	1
FACT1	7	1

TABELA DE DEFINICOES DE MACROS

INDICE	CONTEUDO
1	FACT N
2	QLET(F,1,=,)
3	.FACT1 #1
4	SALT
5	.STATUS
6	ENDM
7	FACT1 N
8	QIF(#1,0,=, )A
9	QLET(NUM,#1,=,)
10	QLET(F,F,NUM,*,=,)
11	QLET(NUM,NUM,1,*,=,)
12	SALT
13	.STATUS
14	.FACT1 "NUM"
15	NOP
16	SALT
17	.STATUS
18	ENDM
19	OXENTE

TABELA DE VARIAVEIS LOCAIS

68	INDICE	VAR LOCAL	VALOR	PROXIMO
70	1	XSTATUS	0.000000E+00000	0
71	2	F	4.000000E+00000	0
72	3	NUM	3.000000E+00000	0

TABELA DE VARIAVEIS GLOBAIS

74	INDICE	VAR GLOBAL	VALOR	PROXIMO
78	1	XLOOP	5.000000E+00001	2
79	2	XSTATUS	1.000000E+00000	0
80	3	F	4.000000E+00000	0
81	4	*7A	1.500000E+00001	0

INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAD 4 1

64 1

PILHA DE BLOCO DE INFORM. Q GERENCIA A EXPANSAD

86	INDICE	CONTEUDO
88	6	4
89	5	13
90	4	1
91	3	4
92	2	3
93	1	-1

=====

100

102

104

106

108

110

111

113

115

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

ESTADO DO PROGRAMA

DIRETORIO DAS ROTINAS MACROS

NOME	INDICE	N. ARGUMENTOS
FACT	1	1
FACT1	7	1

TABELA DE DEFINICOES DE MACROS

INDICE	CONTEUDO
1	FACT N
2	QLET(F,1,=,)
3	.FACT1 #1
4	SALT
5	.STATUS
6	ENDM
7	FACT1 N
8	QIF(#1,0,=, )A
9	QLET(NUM,#1,=,)
10	QLET(F,F,NUM,*,=,)
11	QLET(NUM,NUM,1,*,=,)
12	SALT
13	.STATUS
14	.FACT1 "NUM"
15	NOP
16	SALT
17	.STATUS
18	ENDM
19	OXENTE

137

TABELA DE VARIAVEIS LOCAIS



INDICE	VAR LOCAL	VALOR	PROXIMO
139			
141	%STATUS	0.000000E+00000	0
142	F	1.200000E+00001	0
143	NUM	2.000000E+00000	0

TABELA DE VARIABEIS GLOBAIS

INDICE	VAR GLOBAL	VALOR	PROXIMO
147			
149	%LOOP	5.000000E+00001	2
150	%STATUS	1.000000E+00000	0
151	F	1.200000E+00001	0
152	*7A	1.500000E+00001	0

INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAO

154	7	1
	6	0
	5	1

PILHA DE BLOCO DE INFORM. 0 GERENCIA A EXPANSAO

INDICE	CONTEUDO
159	3
160	13
161	4
162	4
163	14
164	1
165	4
166	3
167	-1
168	
169	

174

176

178

180

182

184

185

187

189

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

211

ESTADO DO PROGRAMA

DIRETORIO DAS ROTINAS MACROS

NOME	INDICE	N. ARGUMENTOS
FACT	1	1
FACT1	7	1

TABELA DE DEFINICOES DE MACROS

INDICE	CONTEUDO
1	FACT N
2	QLET(F,1,=,)
3	.FACT1 #1
4	SALT
5	.STATUS
6	ENDM
7	FACT1 N
8	QIF(#1,0,=, )A
9	QLET(NUM,#1,=,)
10	QLET(F,F,NUM,*,=,)
11	QLET(NUM,NUM,1,*,=,)
12	SALT
13	.STATUS
14	.FACT1 "NUM"
15	NOP
16	SALT
17	.STATUS
18	ENDM
19	OXENTE

TABELA DE VARIABEIS LOCAIS

STM	LC	INDICE	VAR LOCAL	VALOR	PROXIMO
213					
215		1	%STATUS	0.000000E+00000	0
216		2	F	2.400000E+00001	0
217		3	NUM	1.000000E+00000	0

219 TABELA DE VARIAVEIS GLOBAIS

STM	LC	INDICE	VAR GLOBAL	VALOR	PROXIMO
221					
223		1	%LOOP	5.000000E+00001	2
224		2	%STATUS	1.000000E+00000	0
225		3	F	2.400000E+00001	0
226		4	*7A	1.500000E+00001	0

228 INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAO 10 68 1

231 PILHA DE BLOCO DE INFORM. O GERENCIA A EXPANSAO

STM	LC	INDICE	CONTEUDD
233		12	2
234		11	13
235		10	7
236		9	3
237		8	14
238		7	4
239		6	4
240		5	14
241		4	1
242		3	4
243		2	3
244		1	-1
245			
246			

251

253

255

257

259

261

262

264

266

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

ESTADO DO PROGRAMA

DIRETORIO DAS ROTINAS MACROS

NDME	INDICE	N. ARGUMENTOS
FACT	1	1
FACT1	7	1

TABELA DE DEFINICOES DE MACROS

INDICE	CONTEUDO
1	FACT N
2	@LET(F,1,=,)
3	.FACT1 #1
4	SALT
5	.STATUS
6	ENDM
7	FACT1 N
8	@IF(#1,0,=,)A
9	@LET(NUM,#1,=,)
10	@LET(F,F,NUM,*,=,)
11	@LET(NUM,NUM,1,*,=,)
12	SALT
13	.STATUS
14	.FACT1 "NUM"
15	NDP
16	SALT
17	.STATUS
18	ENDM
19	OXENTE

TABELA DE VARIABEIS LOCAIS

288

INDICE	VAR LOCAL	VALOR	PROXIMO
290			
292	XSTATUS	0.000000E+00000	0
293	F	2.400000E+00001	0
294	NUM	0.000000E+00000	0

TABELA DE VARIAVEIS GLOBAIS

INDICE	VAR GLOBAL	VALOR	PROXIMO
300	XLOOP	5.000000E+00001	2
301	XSTATUS	1.000000E+00000	0
302	F	2.400000E+00001	0
303	*7A	1.500000E+00001	0

INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAO 13

PILHA DE BLOCO DE INFORM. 0 GERENCIA A EXPANSAO

INDICE	CONTEUDO
15	1
14	13
13	10
12	2
11	14
10	7
9	3
8	14
7	4
6	4
5	14
4	1
3	4
2	3
1	-1

=====

331

333

335

337

339

341

342

344

346

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

368

ESTADO DO PROGRAMA

DIRETORIO DAS ROTINAS MACROS

NOME	INDICE	N. ARGUMENTOS
FACT	1	1
FACT1	7	1

TABELA DE DEFINICOES DE MACROS

INDICE	CONTEUDO
1	FACT N
2	@LET(F,1,=,)
3	•FACT1 #1
4	SALT
5	•STATUS
6	•ENDM
7	FACT1 N
8	@IF(#1,0,=, )A
9	@LET(NUM,#1,=,)
10	@LET(F,F,NUM,*=,=,)
11	@LET(NUM,NUM,1,=,=,)
12	SALT
13	•STATUS
14	•FACT1 "NUM"
15	NOP
16	SALT
17	•STATUS
18	•ENDM
19	OXENTE

TABELA DE VARIAVEIS LOCAIS

INDICE	VAR LOCAL	VALOR	PROXIMO
370			
1	ZSTATUS	0.000000E+00000	0
2	F	2.400000E+00001	0
3	NUM	0.000000E+00000	0

TABELA DE VARIAVEIS GLOBAIS

INDICE	VAR GLOBAL	VALOR	PROXIMO
376			
1	ZLOOP	5.000000E+00001	2
2	ZSTATUS	1.000000E+00000	0
3	F	2.400000E+00001	0
4	*7A	1.500000E+00001	0

INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAO 16

PILHA DE BLOCO DE INFORM. 0 GERENCIA A EXPANSAO

INDICE	CONTEUDO
18	0
17	17
16	13
15	1
14	14
13	10
12	2
11	14
10	7
9	3
8	14
7	4
6	4
5	14
4	1
3	4
2	3
1	-1

413 =====

415 ESTADO DO PROGRAMA

417 =====

419 DIRETORIO DAS ROTINAS MACROS

421 NOME INDICE N. ARGUMENTOS

423 FACT 1 1

424 FACT1 7 1

426 TABELA DE DEFINICOES DE MACROS

428 INDICE CONTEUDO

430	1	FACT	N	
431	2		@LEI(F,1,=,)	
432	3		.FACT1 #1	
433	4		SALT	
434	5		.STATUS	
435	6		ENDM	
436	7	FACT1	N	
437	8		@IF(#1,0,=, )A	
438	9		@LEI(NUM,#1,=,)	
439	10		@LEI(F,F,NUM,*,=,)	
440	11		@LEI(NUM,NUM,1,-,=,)	
441	12		SALT	
442	13		.STATUS	
443	14		.FACT1 "NUM"	
444	15		NOP	
445	16		SALT	
446	17		.STATUS	
447	18		ENDM	
448	19		OXENTE	

450 TABELA DE VARIAVEIS LOCAIS



STM LC P1 P2 P3 P4 P5

INDICE	VAR LOCAL	VALOR	PROXIMO
452			
454	XSTATUS	0.000000E+00000	0

TABELA DE VARIAVEIS GLOBAIS

INDICE	VAR GLOBAL	VALOR	PROXIMO
458			
460	XLOOP	5.000000E+00001	2
461	XSTATUS	1.000000E+00000	0
462	F	2.400000E+00001	0
463	*7A	1.500000E+00001	0

INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAO 13

PILHA DE BLOCO DE INFORM. O GERENCIA A EXPANSAO

INDICE CONTEUDO

18	0
17	18
16	13
15	1
14	17
13	10
12	2
11	14
10	7
9	3
8	14
7	4
6	4
5	14
4	1
3	4
2	3
1	-1

=====

465  
468  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489

=====

ESTADO DO PROGRAMA

=====

DIRETORIO DAS ROTINAS MACROS

NOME	INDICE	N. ARGUMENTOS
FACT	1	1
FACT1	7	1

TABELA DE DEFINICOES DE MACROS

INDICE	CONTEUDO
1	FACT N
2	QLET(F,1,=,)
3	•FACT1 #1
4	SALT
5	•STATUS
6	ENDM
7	FACT1 N
8	QIF(#1,0,=,)A
9	QLET(NUM, #1,=,)
10	QLET(F, F, NUM, *, =,)
11	QLET(NUM, NUM, 1, =, =,)
12	SALT
13	•STATUS
14	•FACT1 "NUM"
15	NDP
16	SALT
17	•STATUS
18	ENDM
19	OXENTE

TABELA DE VARIAVEIS LOCAIS

493

495

497

499

501

503

504

506

508

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

530

INDICE	VAR LOCAL	VALOR	PROXIMO
532			
534	%STATUS	0.000000E+00000	0

TABELA DE VARIAVEIS GLOBAIS

INDICE	VAR GLOBAL	VALOR	PROXIMO
540	%LOOP	5.000000E+00001	2
541	%STATUS	1.000000E+00000	0
542	F	2.400000E+00001	0
543	*7A	1.500000E+00001	0

INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAD 10

PILHA DE BLOCO DE INFORM. O GERENCIA A EXPANSAD

INDICE	CONTEUDO
550	
551	0
552	18
553	13
554	1
555	18
556	10
557	2
558	17
559	7
560	3
561	14
562	4
563	4
564	14
565	1
566	4
567	3
568	-1
569	

573

575

577

579

581

583

584

586

588

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

ESTADO DO PROGRAMA

DIRETORIO DAS ROTINAS MACROS

NOME	INDICE	N. ARGUMENTOS
FACT	1	1
FACT1	7	1

TABELA DE DEFINICOES DE MACROS

INDICE	CONTEUDO
1	FACT N
2	QLET(F,1,=,)
3	•FACT1 #1
4	SALT
5	•STATUS
6	ENDM
7	FACT1 N
8	QIF(#1,0,=,)A
9	QLET(NUM,#1,=,)
10	QLET(F,F,NUM,*=,)
11	QLET(NUM,NUM,1,-,=,)
12	SALT
13	•STATUS
14	•FACT1 "NUM"
15	NOP
16	SALT
17	•STATUS
18	ENDM
19	OXENTE

610

TABELA DE VARIAVEIS LOCAIS

INDICE	VAR LOCAL	VALOR	PROXIMO
612			
614	XSTATUS	0.000000E+00000	0

TABELA DE VARIAVEIS GLOBAIS

INDICE	VAR GLOBAL	VALOR	PROXIMO
620	XLOOP	5.000000E+00001	2
621	XSTATUS	1.000000E+00000	0
622	F	2.400000E+00001	0
623	*7A	1.500000E+00001	0

INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAO 7

PILHA DE BLOCO DE INFORM. 0 GERENCIA A EXPANSAO

INDICE CONTEUDO

18	0
17	18
16	13
15	1
14	18
13	10
12	2
11	18
10	7
9	3
8	17
7	4
6	4
5	14
4	1
3	4
2	3
1	-1

630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649

653

655

657

659

661

663

664

666

668

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

690

ESTADO DO PROGRAMA

DIRETORIO DAS ROTINAS MACROS

NOME	INDICE	N-ARGUMENTOS
FACT	1	1
FACT1	7	1

TABELA DE DEFINICOES DE MACROS

INDICE	CONTEUDO
1	FACT N
2	@LET(F,1,=,)
3	•FACT1 #1
4	SALT
5	•STATUS
6	ENDM
7	FACT1 N
8	@IF(#1,0,=, )A
9	@LET(NUM,#1,=,)
10	@LET(F,F,NUM,*,=,)
11	@LET(NUM,NUM,1,=,=,)
12	SALT
13	•STATUS
14	•FACT1 "NUM"
15	NDP
16	SALT
17	•STATUS
18	ENDM
19	OXENTE

TABELA DE VARIAVEIS LOCAIS

STM LC P1 P2 P3 P4 P5

INDICE	VAR LOCAL	VALOR	PROXIMO
1	%STATUS	0.000000E+00000	0

TABELA DE VARIAVEIS GLOBAIS

INDICE	VAR GLOBAL	VALOR	PROXIMO
1	%LOOP	5.000000E+00001	2
2	%STATUS	1.000000E+00000	0
3	F	2.400000E+00001	0
4	*7A	1.500000E+00001	0

INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAO 4

INDICE	CONTEUDO
18	0
17	18
16	13
15	1
14	18
13	10
12	2
11	18
10	7
9	3
8	18
7	4
6	4
5	17
4	1
3	4
2	3
1	-1

PILHA DE BLOCO DE INFORM. 0 GERENCIA A EXPANSAO

INDICE CONTEUDO

INDICE	CONTEUDO
18	0
17	18
16	13
15	1
14	18
13	10
12	2
11	18
10	7
9	3
8	18
7	4
6	4
5	17
4	1
3	4
2	3
1	-1

=====

733

ESTADO DO PROGRAMA

735

737

DIRETORIO DAS ROTINAS MACROS

739

741

NOME	INDICE	N. ARGUMENTOS
FACT	1	1
FACT1	7	1

743

744

746

TABELA DE DEFINICOES DE MACROS

748

INDICE	CONTEUDO
1	FACT N
2	QLET(F,1,=,)
3	.FACT1 #1
4	SALT
5	.STATUS
6	ENDM
7	FACT1 N
8	QIF(#1,0,=, )A
9	QLET(NUM,#1,=,)
10	QLET(F,F,NUM,*,=,)
11	QLET(NUM,NUM,1,=,=,)
12	SALT
13	.STATUS
14	.FACT1 "NUM"
15	NDP
16	SALT
17	.STATUS
18	ENDM
19	OXENTE

750

751-

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

770

TABELA DE VARIAVEIS LOCAIS



INDEXE	VAR LOCAL	VALDR	PROXIMO
772			
774	ZSTATUS	0.000000E+00000	0

TABELA DE VARIAVEIS GLOBAIS

INDEXE	VAR GLOBAL	VALDR	PROXIMO
780	ZLBOP	5.000000E+00001	2
781	ZSTATUS	1.000000E+00000	0
782	F	2.400000E+00001	0
783	*7A	1.500000E+00001	0

INDEXE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAO 1

PILHA DE BLOCO DE INFORM. 0 GERENCIA A EXPANSAO 8

INDEXE	CONTEUDO
790	
791	0
792	18
793	13
794	1
795	18
796	10
797	2
798	18
799	7
800	3
801	18
802	4
803	4
804	18
805	1
806	4
807	5
808	-1

=====

814

816

818

820

822

824

825

827

829

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

851

ESTADO DO PROGRAMA

DIRETORIO DAS ROTINAS MACROS

NOME	INDICE	N. ARGUMENTOS
FACT	1	1
FACT1	7	1

TABELA DE DEFINICOES DE MACROS

INDICE	CONTEUDO
1	FACT N
2	2LET(F,1,=,)
3	•FACT1 #1
4	SALT
5	•STATUS
6	ENDM
7	FACT1 N
8	2IF(#1,0,=,)A
9	2LET(NUM,#1,=,)
10	2LET(F,F,NUM,#,=,)
11	2LET(NUM,NUM,1,=,=,)
12	SALT
13	•STATUS
14	•FACT1 "NUM"
15	NOF
16	SALT
17	•STATUS
18	ENDM
19	OXENTE

TABELA DE VARIAVEIS LOCAIS

STM LC P1 P2 P3 P4 P5

INDICE	VAR LOCAL	VALOR	PROXIMO
853			
855	%STATUS	0.000000E+00000	0

857 TABELA DE VARIAVEIS GLOBAIS

INDICE	VAR GLOBAL	VALOR	PROXIMO
861	%LOOP	5.000000E+00001	2
862	%STATUS	1.000000E+00000	0
863	F	2.400000E+00001	0
864	*7A	1.500000E+00001	0

866 INDICE DO BLOCO NA PILHA DE GERENCIA DE EXPANSAO -1

868

=====

870 0000 18 JTEST DC 24  
871 END

I  
84  
I

MEMORIA UTILIZADA 1 BYTES MONTAGEM BEM SUCEIDIDA 0 ADVERTENCIA(S) 0 ERROS(S)  
OPCOES EM EFEITO NXREF LIST NT

TEMPO DE EXECUCAO DO PASSO1/PASSO2 35.553/ 33.496 SEG

BIBLIOGRAFIA

1. ILROY, M. Macro instructiona extesions of compiler languages. New York, CACM 3: 214-20, 1960.
2. HALPERN, M. I. EXPOP a meta-language without metaphysics. Proc. London, Computer Conference, 26: 57-68, 1964.
3. WILKER, M.V. An experimete with a self-compilar for a simple list processing language, Oxford, Annual Review in Automatic Programming, 4 : 1-48, 1964.
4. STRACHEY, C. A. general purpose macrogenerater (GPM). London, Computer Journal, 8: 225-41, 1965.
5. WAITE, W. M. A language independente macro processor (LIMP) New York, CACM, 10: 18-23, 1967.
6. BROWN, P.J. The ML/I macro processador, New York, CACM, 10:18-23, 1967.
7. Universidade Federal do Rio de Janeiro. Nucleo de Computação Eletrônico. Manual do Usuário Sistema Operacional de Simulação (S.O.S.). Rio de Janeiro 1975.
8. GRIS, David. Compiler construction for digital computers. New York, Wiley, 1971. p. 220-24.
9. KNUTH, Donald E. The art of computar programming. New York Addison Wesley, 1973. p. 516-42.
10. GUIMARÃES, Célio. Tramento de Colisão por Encadeamento (Notas de aulas de Curso de Estrutura de Arquivo do Programa Eng . de Sistemas COPP. outubro 1974.

11. BARRODALE, I. EHLE. B.I. Elementary Computer Applications .  
New York, Jonh Wiley, 1971. p. 203-7.
12. LUM, V.Y. P.S.T. & Dodd. M. Key to. Address transform tecniques: A fundamental hasfunctions. New York  
CACM 14 (4) 228-39, 1971.
13. KELLY. M. Campbel. An indrotuction to macro. New York, Macdonal/  
Elsevier, 1974. p. 55-113.
14. DONAVAN, John J. Systems programming. New York, McGraw Hill,  
1974. p. 111-42.
15. DIJKTRA. E. W. Recusive programming. New York, 1969. p. 221.
16. KENT. Willian. Assembler Language Macropromming Computing  
New York Survey ACM, 1 (4): 183-196, 1969.
17. BURROUGHS, B6700/B7700 PL/I Language referencia Manual 5000201  
1 nov. 1974. New York.
18. IBM System/360 Operating System Assembler Language C 28-6514-5  
New York 1970.
19. IBM System/360 Operantin Sustum Supervison And Management  
Macro-Instruction New York 1971.
20. MAGINNIS, JAMES B.  
Elements of Compiler Construction New York Appleton-Cen-  
tury-Crofts 1975 p. 25-106.
21. NEAUR, Peter (editor) & BACHUS, J. W. Revised  
Report on the Algorithmic Language Algol 60 New York  
CACM 6 (1) p. 1-80 1963.
22. WEGNER, PETER  
*Introduction to system programming - New York*  
*Academic Press 1964 pag 101-136*