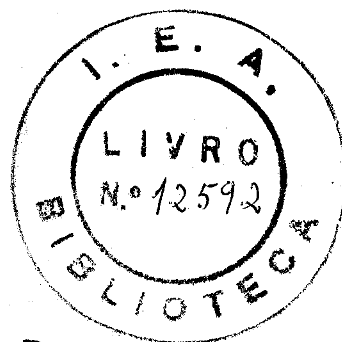
ELENICE MAZZILLI

CONCEITO DE RECURSIVIDADE NA LINGUAGEM FORTRAN - IMPLEMENTAÇÃO NO IBM 1620 - MOD.2

Serviço de Cálculo Analógico e Digital
INSTITUTO DE ENERGIA ATÔMICA



Orientadores:

Prof. Guy Augier
Prof. Ivan de Queiroz Barros

Dissertação de Mestrado apresentada na Escola Politécnica da Universidade de São Paulo

São Paulo, 1970

INSTITUTO DE ENERGIA ATÔMICA

AGRADECIMENTOS

Agradeço

Ao Sr. Diretor do Instituto de Energia Atômica, Prof. Dr. Rômulo Ribeiro Pieroni, pelo apoio dele recebido sempre, desde o início de minhas atividades no I.E.A., e em particular na realização dêste trabalho.

Aos Professores Guy Augier e Ivan de Queiroz Barros, meus orientadores, pelo interêsse e valiosas sugestões.

Ao Chefe do S.C.A.D. e amigo, Engº Cíbar Cáceres Aguilera, pelo constante incentivo e oportunas discussões.

À colega Helena Suzuki, pela sua inestimável ajuda, principalmente na elaboração dos desenhos.

A todos os colegas do S.C.A.D. pelo incentivo e colaboração que sempre me dedicaram.

E como êste trabalho não se teria realizado sem a colaboração dedicada e constante da colega Lúcia Faria Silva, a ela os meus agradecimentos.

ÍNDICE

pag.

Introdução 1

Cap. I - A Recursividade 3

Cap. II - A Compilação 9

Cap. III- Descrição das Modificações Introduzidas no
Compilador 18

Cap. IV - Descrição da Rotina Recurs 24

Cap. V - Descrição da Rotina Pilha 33

Cap. VI - Limitações no uso da recursividade 42

Cap. VII - Exemplos de aplicação 48

Conclusão - 53

Bibliografia - 54

CONCEITO DE RECURSIVIDADE NA LINGUAGEM FORTRAN - IMPLEMENTAÇÃO NO IBM 1620 - MOD.2

INTRODUÇÃO

As linguagens de programação de alto nível tendem a permitir a simplificação cada vez maior na elaboração de um programa. É importante que o pesquisador possa concentrar seus esforços no problema a ser resolvido e no algorítmo a ser utilizado, sem se perder em tortuosos artifícios de programação; que no decorrer de seu trabalho, êle venha a utilizar o computador e as linguagens de programação como ferramentas; que o programa por êle escrito seja tão sòmente um meio de obter o resultado, e, portanto, que a programação seja simplificada tanto quanto possível graças aos recursos oferecidos pela linguagem.

Sob êste ponto de vista, a introdução da recursividade no FORTRAN apresenta grande interêsse, pois de um modo geral as definições recursivas tornam-se de mais fácil compreensão quando escritas como tais, ao invés de se usarem processos iterativos; além disso, existem alguns tipos de funções que só podem ser calculadas recursivamente.

O objetivo dêste trabalho é permitir o uso da recursividade no FORTRAN, mediante certas modificações no compilador, e a utilização de rotinas auxiliares. Mais ainda, a possibilidade de introduzir alterações no compilador sem modificá-lo essencialmente, nos permitirá, em trabalhos futuros, a introdução de novos recursos, entre êles as macro-instruções, que consideramos também de grande interêsse no sentido de simplifi-

car a tarefa de programação.

O computador para o qual foi desenvolvido êste trabalho é um IBM 1620, mod.2, dispondo de duas unidades 1311 de discos magnéticos, e utilizando como sistema operacional o Sistema Monitor I.

Obs.: Em diversas oportunidades usaremos nêste trabalho, têrmos em inglês, embora existindo a tradução dos mesmos para o português; por estarem já consagrados pelo uso, tais têrmos nos parecem mais aconselháveis; consideramos que proporcionam maior clareza ao texto, do que se fôsse usada sua tradução, muitas vezes artificial, como seria o caso de "Loader", Record mark","Flag", etc.

CAPÍTULO I

A RECURSIVIDADE

1.1 - Entende-se por recursividade a técnica de se definir uma função em têrmos de si própria, como por exemplo, no caso do fatorial de um inteiro n:

$Fat(n) = Fat(n-1).n \quad$ para $n \neq 0$

$Fat(0) = 1$

Evidentemente, qualquer função definida recursivamente deve ter uma definição explícita para algum valor do argumento.

Lembremos a notação Mc Carthy para expressões condicionais:

$$x = [b_1 \longrightarrow e_1 \, , \, b_2 \longrightarrow e_2 \, , \, \ldots \, , \, b_{n-1} \longrightarrow e_{n-1} \, , \, e_n]$$

onde os b_i são expressões booleanas; se b_1 é verdadeira, $x = e_1$; senão, examina-se b_2: sendo b_2 verdadeira, $x = e_2$, e assim por diante; se tôdas as b_i forem falsas, $x = e_n$.

Usando esta notação, podemos escrever a definição de fatorial da seguinte maneira:

$$Fat(n) = [(n=o) \longrightarrow 1 \, , \, n.Fat(n-1)] \qquad (1)$$

Um outro exemplo é o algorítmo de Euclides para determinar o máximo divisor de dois números inteiros positivos:

$$Mdc(n,m) = [(m>n) \longrightarrow Mdc(m,n), \, (m=0) \longrightarrow n, Mdc(m, Res(n,m))]$$

onde Res (n,m) é uma função cujo valor é o resto da divisão de n por m.

No caso do fatorial, temos o que se chama uma definição recursiva, e no caso do máximo divisor comum, o uso recursivo de um procedimento. Como é fácil ver, o primeiro pode ser calculado por um processo iterativo, enquanto que com o segundo, isso não é possível.

As vantagens e desvantagens do uso da recursividade em programação têm sido objeto de opiniões divergentes. En -

quanto no LISP ela é considerada obviamente necessária, devido à natureza recursiva da definição da estrutura de listas, em ALGOL e FORTRAN essa necessidade é muitas vezes posta em discussão.

Há quem julgue que a principal razão da superioridade do ALGOL sôbre o FORTRAN seja o fato de o primeiro permitir a recursividade; por outro lado, há os que consideram os processos iterativos como perfeitos substitutos para as técnicas recursivas. Basta, porém, atentarmos para a diferença entre definição recursiva e uso recursivo de um procedimento, conforme apresentado no início dêste capítulo, para vermos que nem sempre isso é verdade.

Um argumento contra a recursividade é o tempo gasto no processo de empilhamento e desempilhamento de parâmetros; porém, conforme o caso, pode dar-se que êste excesso de tempo seja compensado pela facilidade de programação e de compilação. Naturalmente, como em muitas outras situações, o programador deve conhecer as vantagens e as objeções contra o uso da recursividade, a fim de fazer a escolha do método mais adequado ao seu caso. Desde que a recursividade venha a se constituir em um instrumento a mais dentro da linguagem de programação, consideramos interessante a sua implementação no FORTRAN II-D do sistema Monitor I para IBM 1620.

1.2 - Faremos a seguir algumas considerações sôbre Pilhas, que são elementos indispensáveis à execução de um programa recursivo.

Uma pilha é uma sucessão de posições de memória, em que a última é acessível por intermédio de um apontador, cujo valor define a altura da pilha.

As informações são armazenadas em uma certa ordem, e serão posteriormente utilizadas em ordem inversa. Essa estrutura diz-se do tipo LIFO (Last In First Out). As estruturas nas quais as informações são retiradas na mesma ordem em que foram armazenadas, são do tipo LILO (Last In Last Out), como é o caso das filas.

Enquanto que normalmente não existe relação de ordem definida entre os elementos armazenados na memória, uma vez que o acesso a cada um se faz através de seu endereço, o mesmo não acontece em uma pilha: a ordem é determinada pela posição na pilha; assim, entre dois elementos a e b, se dizemos que a é mais alto que b, significa que a está em uma posição que será atingida antes de b.

Podemos efetuar sôbre uma pilha três operações distintas: Incluir, retirar, e apagar um elemento:

Chamando P(p) a palavra da memória correspondente ao tôpo da pilha, e por p o apontador que indica o nível dessa palavra na pilha, as três operações são feitas da seguinte maneira:

1) Introdução de um elemento <u>e</u>

```
p=p+1
P(p)= e
```

2) Retirada de um elemento <u>e</u>

```
e=P(p)
p=p-1
```

3) Apagamento de um elemento <u>e</u>

```
p=p-1
```

Note-se que nêste último caso, ao se redefinir o valor de <u>p</u>, o elemento correspondente será apagado da pilha, embora fìsicamente continue presente na memória.

1.3 Sub- programa recursivo

A redação de um sub-programa FORTRAN para o cálculo do fatorial, correspondendo à definição (1), teria a seguinte forma:

```
  FUNCTION FAT (A)
  IF (A)  1,1,2
1 FAT=0
  RETURN
```

```
2 FAT = FAT (A-1) * A
  RETURN
  END
```

Como se pode observar, o comando 2 não é válido, pois consiste em uma chamada de um sub-programa por êle próprio. Suponhamos, porém, que tal comando possa ser aceito e executado. Obteríamos, na execução, sucessivas chamadas do sub-programa FAT, cada uma com o argumento decrementado de 1, até que êste atingisse o valor 0. Neste ponto, seria feito um desvio para o comando 1.

Porém, a cada chamada de FAT com um novo valor do argumento, o valor anterior seria perdido, uma vez que os sucessivos resultados da operação A-1., ocupariam a mesma área temporária. Surge então a necessidade do uso de uma pilha, para a proteção dêsses argumentos. Sem entrar em detalhes, que serão vistos posteriormente na descrição da rotina PILHA (cap.V), mostraremos em um fluxograma (fig.1) como seria, em princípio, a execu dêsse programa com o uso de uma pilha.

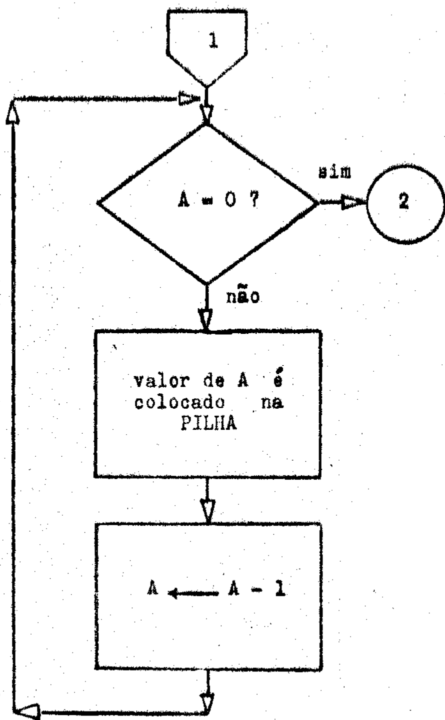
Em um cálculo dêste tipo, com uma chamada do sub-programa para si próprio, dizemos que há a recursividade direta.

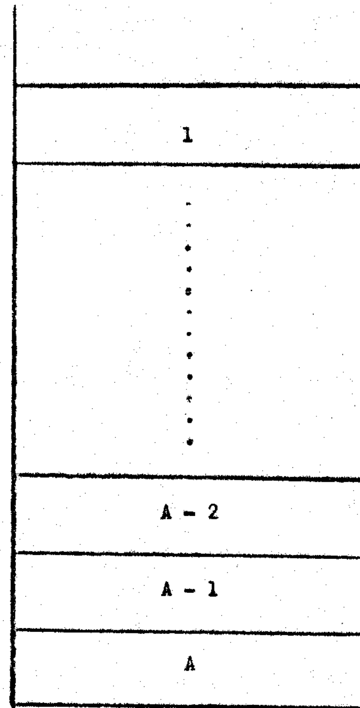
Na chamada recursividade indireta, podemos ter dois ou mais sub-programas S1, S2, S3, ... Sn, em que S1 chama S2, êste chama S3, que por sua vez chama S1, etc. Nêste caso não há uma chamada de sub-programa feita por êle próprio, e na compilação de cada um não seria detetado êrro. Porém, a execução não conduziria ao resultado esperado, pelo mesmo motivo visto anteriormente, no caso da recursividade direta: Os argumentos das sucessivas chamadas iriam se perder, por ocuparem todos a mesma área; novamente aqui aparece a necessidade do uso de uma pilha para a proteção dos argumentos e sua oportuna restauração.

As modificações introduzidas no compilador FORTRAN II-D do Sistema Monitor I, tiveram por objetivo tornar possível

a compilação e execução de programas recursivos, tanto no caso de recursividade direta como indireta.

No compilador ALGOL, o tratamento padrão de todo o programa é feito por intermédio de pilhas, e a recursividade não constitui, portanto, uma noção à parte; ela está naturalmente incluída na linguagem. O compilador FORTRAN, por outro lado, não possui essa modalidade de tratamento das expressões. É por isso que, para se introduzir o conceito de recursividade no FORTRAN, torna-se necessário introduzir o uso de pilhas, para que os programas recursivos possam ser executados convenientemente.





pilha de argumentos

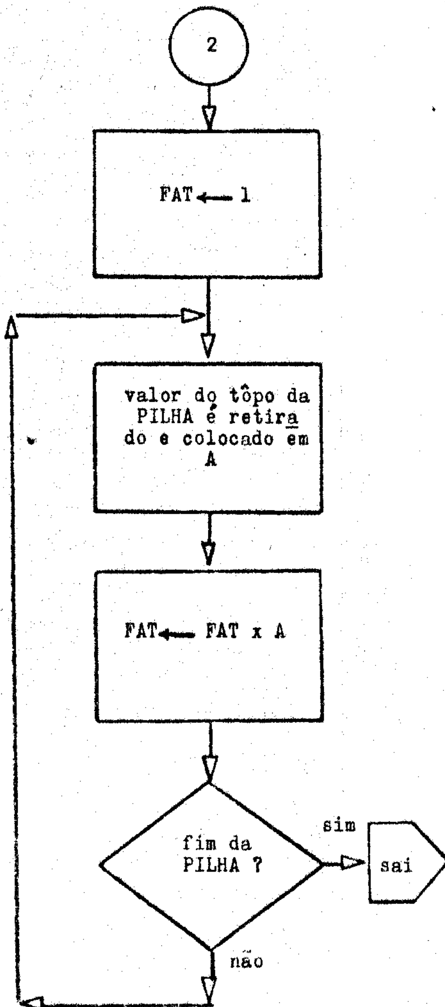


Fig. 1 - Uso de pilha no cálculo do Fatorial

CAPÍTULO II

A COMPILAÇÃO

2.1 Definições

2.1.1 Uma substituição estática é uma operação que permite produzir um texto resultado (objeto) a partir de um texto dado(fonte), em que o mecanismo de substituição independe da execução do texto fonte.

Uma substituição dinâmica é uma operação que permite produzir um texto resultado a partir da execução de um texto fonte.

Na substituição estática, o mecanismo de substituição é definido independentemente do texto fonte, enquanto que na dinâmica, o mecanismo de substituição é definido pelo texto fonte.

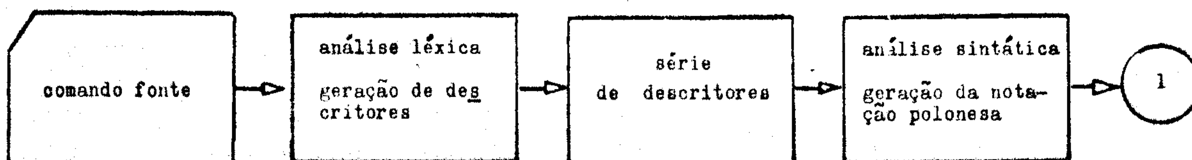
A montagem de um programa escrito em linguagem simbólica é um exemplo de substituição estática, assim como a compilação de um programa FORTRAN e a carga de um programa objeto, isto é, a transformação dos endereços relativos em endereços absolutos; a interpretação e a macro-geração são exemplos de substituição dinâmica.

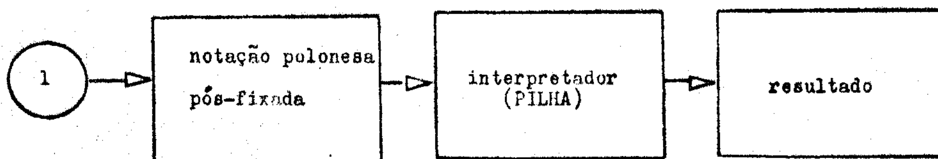
2.1.2 A compilação é um processo de substituição estática, bastante complexa, devido à diferença de nível entre as linguagens fonte e objeto. A substituição pode ser feita em diversas fases, com o texto objeto de cada fase servindo de texto fonte para a seguinte. O produto destas substituições sucessivas será a substituição complexa total, que constitui a compilação

2.2 Fases de um compilador

Veremos a seguir, resumidamente, as diferentes fa-

ses de um compilador do tipo ALGOL, que é completamente modular e onde, como já foi dito, a recursividade é inerente, devido ao tratamento padrão das expressões. O esquema geral da compilação e execução de um programa é o que segue:





2.2.1 - Análise léxica

A análise léxica é realizada por meio de um autômato de número finito de estados. Por exemplo:

a) A unidade sintática "identificador" pode ser definida como:

$I \longrightarrow l \mid Il \mid Id$

onde
l... letra
d... algarismo

O reconhecimento de um identificador será feito com a seguinte tabela de estados:

	l	d
S_1	S_2	êrro
S_2	S_2	S_2

Isto é: Partindo o autômato do estado S_1, uma letra o fará passar ao estado S_2, e um algarismo, a uma condição de êrro; partindo do estado S_2, tanto letra como algarismo farão com que permaneça em S_2.

b) A unidade sintática "número" é definida pelas regras:

N → V | sV
V → D | E | D E
D → J | F | J F
E → eI
F → pJ
I → J | sJ
J → dJ | d

onde:
N ... número
V ... número sem sinal
D ... número decimal
E ... expoente
J ... inteiro sem sinal
F ... fração decimal
I ... inteiro
s ... sinal
e ... 10 (base 10)
p ... ponto decimal
d ... algarismo

A tabela de estados para reconhecimento do número será a seguinte:

	s	d	p	e
S_1	S_2	S_3	S_4	S_6
S_2	êrro	S_3	S_4	S_6
S_3	êrro	S_3	S_4	S_6
S_4	êrro	S_5	êrro	êrro
S_5	êrro	S_5	êrro	S_6
S_6	S_7	S_8	êrro	êrro
S_7	êrro	S_8	êrro	êrro
S_8	êrro	S_8	êrro	êrro

S_1 é o estado inicial e S_8 o final

Após realizar a análise e reconhecimento dos elementos de um comando fonte, será gerada uma série de descritores e êstes serão analisados na fase subsequente.

2.2.2 - Análise Sintática

Nesta fase o texto fonte proveniente da fase anterior é analisado conforme a sintaxe da linguagem fonte. Podem ser usados diferentes algorítmos de análise, que permitem verificar

se uma frase pertence ou não à linguagem. Os algorítmos de análi se sintática utilizam como dados:

a) as regras sintáticas da linguagem fonte;

b) o programa fonte a ser analisado,

e fornecem como resultado final as expressões escritas segundo a notação polonesa pós - fixada.

2.2.3 - Interpretação

O programa objeto resultante não estará em linguagem de máquina, mas em uma linguagem intermediária; esta será submetida a um interpretador que, com o uso de pilhas de trabalho, irá interpretar e calcular as expressões.

Tanto na fase de análise léxica como na sintática, existem na verdade duas partes: a análise, ou reconhecimento, que verifica a validez dos símbolos e das frases, e a geração, que consiste em produzir o texto objeto que entrará como fonte na próxima fase.

2.3 - Modificações em um compilador

Tendo em vista as diversas fases de um compilador, as modificações nele introduzidas podem ser feitas em diferentes níveis:

2.3.1 - Uma modificação no nível da análise léxica seria a inclusão de novos símbolos de base. Por exemplo:

Suponhamos que o número π seja muito utilizado em nossos cálculos, e desejamos poupar-nos o trabalho de definir o seu valor em cada programa; podemos introduzir PI como um novo símbolo de base: Ao se realizar a análise léxica, o aparecimento dêsse símbolo seria reconhecido, e o valor 3.1415926 ... tomaria seu lugar no texto fonte. A análise das expressões, na segunda fase, e correspondente geração em notação polonesa, não sofreriam alteração alguma. O número 3.1415926 iria ocupar no texto objeto o lugar do símbolo PI.

INSTITUTO DE ENERGIA ATÔMICA

2.3.2 - Uma modificação no nível da análise sintática consistiria na inclusão de novas regras na gramática da linguagem fonte.

É o caso da introdução de macro definições, que permitem a criação de novas entidades sintáticas, a partir de entidades mais simples. No ALGOL, por exemplo, ao ser analisada a macro instrução do tipo :

for q:=r step s until t do A [q] := B [q]

essa análise iria ser feita mediante as regras introduzidas através da macro estrutura :

for variável := expressão step expressão until expressão do instrução

isto é, para verificar se (1) é sintàticamente correta, seria verificado se:

q é uma variável
r,s,t são expressões
A [q]:= B [q] é uma instrução

E seria produzido, por substituição dinâmica, o seguinte :

```
begin
     q:= r
   L1 : if (q-t) x sign (s)  0 then
          begin
               A  q := B  q ;
               q:= q+s ;
               go to L1
          end
end
```

2.3.3 - E finalmente, uma modificação no último nível, que podemos chamar de modificação no conceito, seria a introdução de um novo conceito, que não possa ser expresso em fun

ção dos já existentes. Por exemplo, a introdução da recursividade no caso do FORTRAN, que originalmente não a possui.

2.4 - O compilador FORTRAN

O compilador FORTRAN não obedece exatamente a êsse esquema traçado acima; êle não é modular, como o ALGOL, pois as várias fases não são nìtidamente distintas, além de não utilizar autômatos na análise, nem pilhas na interpretação. Mas, para efeito de clareza na exposição dêste trabalho, podemos supor o compilador FORTRAN teòricamente constituído de um modo análogo ao do ALGOL, e localizar as modificações efetuadas entre as diferentes fases da compilação.

Como sabemos, existem dois tipos de declaração de sub-programa FORTRAN:

1) FUNCTION F (A_1 , A_2 ... A_n)

2) SUBROUTINE S (A_1,A_2 ... A_n)

A diferença entre êles reside na forma como o resultado retorna ao programa que chamou. A partir dêste ponto, e apenas por uma questão de ordem didática, iremos tratar sempre do primeiro tipo, embora, " mutatus mutandi ", tudo o que for feito para o primeiro, se aplique também ao segundo.

2.4.1 - Modificação no nível da análise léxica

2.4.1.1 - Para que um sub-programa recursivo possa ser reconhecido e compilado como tal, êle deve conter uma declaração que o identifique. Convencionamos que esta declaração seja um comando do tipo

$ RECURSIVE FUNCTION F (A_1,A_2,...,A_n)

substituindo no programa fonte o comando

FUNCTION F (A_1,A_2, ...,A_n)

e, como acontece com êste, deve ser o primeiro comando diferente de comentário, no programa fonte.

O simbolo $ deve ser perfurado na coluna 1 do car-

e o restante do comando, a partir da coluna 7, seguindo sempre as regras da programação FORTRAN.

Houve, pois, a introdução de novos símbolos, cujo tratamento será feito por uma rotina auxiliar, chamada RECURS, que será descrita detalhadamente no capítulo IV.

Ao encontrar o comando $ RECURSIVE FUNCTION ... a rotina RECURS tomará duas providências:

1a. - Apagará os caracteres $ RECURSIVE, afim de poder devolver ao compilador um comando reconhecível por êle, da forma FUNCTION F (...

2a. - Funcionará como um macro-gerador, criando mais um comando, que será incorporado ao programa fonte e submetido ao compilador para análise, após o primeiro. Êsse comando será da forma

CALL PILHA (1, A_1, A_2, ..., A_n)

em que $A_1, A_2, ..., A_n$ são os mesmos argumentos do sub-programa que está sendo compilado, e PILHA é o nome de uma segunda rotina auxiliar cuja descrição detalhada será feita no capítulo V. Sua função será a de providenciar, na execução, o empilhamento dos argumentos e endereços de retôrno, e posterior desempilhamento. O argumento 1 serve como informação ao programa PILHA, para distinguir esta chamada daquela que será feita em 2.4.1.2.

Após isto, RECURS devolverá o contrôle ao compilador. Para permitir a intervenção desta rotina no decorrer da compilação, foi necessário introduzir-se algumas modificações nas instruções do compilador, as quais serão descritas e explicadas no capítulo III.

2.4.1.2 - No caso da recursividade direta, em que aparece no programa uma chamada para êle próprio, conforme exemplo em 1.3, o compilador iria acusar um êrro, através da mensagem ERR 03, com consequente inibição da execução. Também aqui foi introduzida uma modificação afim de que êsse tipo de comando seja aceito, caso o sub-programa tenha sido inicialmente declarado co-

mo recursivo.

A verificação será feita pela rotina RECURS: se o sub-programa foi declarado como recursivo, RECURS substituirá aquela chamada ilegítima por uma chamada para PILHA, conservando os mesmos argumentos. Caso o sub-programa não tenha sido declarado como recursivo, essa chamada deve ser considerada como êrro, e para tanto o contrôle será devolvido ao compilador, que irá prosseguir na análise do comando.

Considerando estas duas modificações, os sub-programa para cálculo do fatorial deverá ser escrito da seguinte maneira:

```
$       RECURSIVE FUNCTION FAT (A)
        IF (A) 1,1,2
     1  FAT = 1.
        RETURN
     2  FAT = FAT (A - 1) * A
        RETURN
        END
```

Como consequência da intervenção da rotina RECURS na primeira fase da análise, o programa realmente analisado e submetido ao compilador será como se tivesse sido escrito:

```
        FUNCTION FAT (A)
        CALL PILHA (1,A)
        IF (A) 1,1,2
     1  FAT = 1.
        RETURN
     2  FAT = PILHA (A - 1.)  * A
        RETURN
        END
```

A partir dêste ponto, a compilação prosseguirá normalmente. Como se pode notar, o sub-programa PILHA será chamado uma única vez, se for o caso de recursividade indireta, e duas ve-

zes se for direta. Nêste caso, porém, à segunda chamada êle apagará os elementos que foram empilhados na primeira, para evitar duplicação na pilha. À primeira chamada daremos o nome de tipo 1 e à segunda, tipo 2.

2.4.2. - Modificação no nível de conceito

A fase de interpretação vista no resumo sôbre o compilador ALGOL tambem está presente no FORTRAN, mas com uma diferença: no primeiro, o interpretador recebe como texto fonte uma linguagem intermediária gerada pelas fases anteriores, que, após interpretada, irá fornecer o resultado; no segundo, o interpretador é a própria máquina, que tem como texto fonte o programa objeto em linguagem absoluta, gerado pelo compilador e pelo carregador.

No momento da execução (ou interpretação) de um subprograma recursivo, entrará em uso o sub-programa auxiliar PILHA, cuja ação permite a modificação do FORTRAN no nível do conceito; isto é, a introdução do conceito de recursividade, antes inexistente na linguagem FORTRAN, é facultada pela introdução do uso de pilhas para proteger e restaurar os argumentos. A manipulação dessas pilhas é feita, como já foi dito, pela rotina auxiliar PILHA.

CAPÍTULO III

DESCRIÇÃO DAS MODIFICAÇÕES INTRODUZIDAS NO COMPILADOR E NO LOADER

A fim de tornar possível a compilação de um subprograma recursivo, escrito conforme 2.4.1.2, algumas modificações foram introduzidas na fase de análise léxica do compilador, além de uma alteração feita no Loader, para evitar problemas que poderiam surgir com respeito ao comprimento das palavras.

3.1 - Modificações no Compilador

3.1.1 - Quando o compilador está presente na memória, encontram-se, ocupando os endereços 03062 e seguintes, na versão original, as instruções:

03062	14	15139	00043	
03074	46	02878	01200	
03086	15	15148	00000	(1)
03098	43	03130	15139	
03110	31	15138	15140	

Elas correspondem ao início da análise de um cartão fonte lido, e foram substituídas pelas seguintes:

03062	25	03106	00440	
03074	34	03106	00701	
03086	36	03106	00702	(1')
03098	49	17700	01140	
	00	01817	700	

No endereço 03106 inicia-se um "disk control field". O dígito nessa posição será 1 ou 3 conforme a unidade de disco que estiver sendo utilizada no momento; essa informação encontra-se disponível na área de comunicação, endereço 00440, e será utilizada pela primeira instrução de (1').

Estas instruções têm por finalidade chamar do disco

a rotina RECURS, e transferir-lhe o contrôle. RECURS está gravada no disco a partir do setor 14.000, e ocupa 18 setores. Ao ser chamada à memória, irá ocupar o enderêço 17700 e seguintes; a escolha dêste endereço deve-se ao fato de que êle pertence a uma área reservada pelo compilador para conter a tabela de símbolos, à medida em que esta for sendo construída, no decorrer da compilação. Portanto, no momento da análise do primeiro cartão, esta área está disponível, e contém conjuntos de zeros e record-marks, para futuro contrôle do comprimento da tabela de símbolos. Após ser utilizada para analisar o primeiro cartão, que poderá conter ou não a declaração de sub-programa recursivo, RECURS deixará de ser necessária, a não ser por um pequeno trecho: é o que tem por finalidade fazer o tratamento do êrro 03; êste pequeno trecho será transferido para outra área disponível da memória (a partir de 14800); as posições 17700 e seguintes receberão de novo os zeros e record-marks que tinham anteriormente; as instruções (1) do compilador, que trazem RECURS à memória, serão substituídas pelas originais, e dêste momento em diante tudo se passará como se não tivesse havido modificação alguma.

Se o primeiro cartão lido for um comentário, o desvio para RECURS ainda será conservado até que seja lido um cartão diferente de comentário.

3.1.2 - A segunda modificação foi feita na instrução que ocupa os endereços de memória 07468 e seguintes. Originalmente, nesta posição encontra-se a instrução:

17 09350 00073 (2)

Esta seria executada sempre que o compilador detectasse um êrro, devido à chamada de um sub-programa, feito por ele próprio, ou ao uso de índice em variável não pràviamente dimensionada. O compilador interpreta tal fato como êrro mediante as seguintes conclusões.

a) - O nome de variável que está sendo analisado

no momento, e que se encontra na palavra do compilador, SYM, já consta da tabela de símbolos;

b) - Não consta como sendo de uma variável dimensionada;

c) - O próximo caráter no comando fonte após êsse nome é um "(".

A instrução (2) consiste em um desvio para uma rotina de êrro, transmitindo o número 03, que é o código correspondente; essa rotina, além de imprimir a mensagem ERR 03, posiciona um indicador, que será consultado posteriormente, e acarretará o fim do JOB, com uma mensagem de inibição da execução.

Substituindo essa instrução (2), foi colocada a seguinte:

49 14866

14866 é um ponto de entrada do já citado trecho de RECURS, que tem por finalidade verificar se a variável presente em SYM coincide com o nome do sub-programa recursivo que está sendo compilado; em caso afirmativo, essa variável será substituída por PILHA, e em caso negativo, será providenciado o desvio para a rotina de êrro.

Caso não esteja sendo compilado um sub-programa recursivo, RECURS já terá se encarregado de retornar a 07468 a instrução (2) original, para que não seja gasto tempo inùtilmente, com um desvio desnecessário para 14866.

3.1.3 - Quando o programa fonte está entrando pela máquina de escrever do computador, e não por cartões, o início da análise do comando é feito diferentemente. Encontra-se no enderêço 02810 a instrução:

47 03098 01200 (3)

que corresponde a êsse início. Ela foi substituida por:

Com isto, o contrôle será transferido para a primeira instrução de (1'), que é encarregada de chamar RECURS. O compilador, nêste ponto, está informado de que a entrada está sendo feita pela máquina de escrever, devido à presença de um dígito no enderêço simbólico INDIV. Êsse mesmo dígito servirá de informação para RECURS, a fim de, após cumprida a sua tarefa, poder reconstituir em 02810 a instrução original (3).

3.2 - Modificações no LOADER

Quando está sendo efetuada a carga de um programa e de seus sub-programas na memória, o Loader realiza, entre outros, um teste para verificar se a definição do comprimento das palavras, tanto de ponto fixo como flutuante, é a mesma para o programa principal, programas encadeados e sub-programas.

Como sabemos, F representa o número de dígitos da mantissa das palavras de ponto flutuante, e K o número de dígitos da palavra de ponto fixo, e na definição padrão do sistema, F=8 e K=4. Entretanto, mediante o uso de um cartão de contrôle da forma *FANDKffkk, o programador pode alterar êsses comprimentos, desde que o faça em todos os sub-programas e programas encadeados.

O programa objeto, antes de ser transformado para "core image", contêm um cabeçalho onde constam algumas informações necessárias à relocação e carga do programa, e entre elas estão os valores de F e K.

O Loader irá veridicar êsses valores a cada programa a ser carregado, e caso encontre um dêles diferente dos anteriores, acusará um êrro mediante a mensagem ERR L6, o que ocasionará a inibição da execução.

Lembrando, porém, que o sub-programa PILHA é requerido por todos os sub-programas recursivos, sem ser chamado ex-

plicitamemte pelo programador, vemos que seria impraticável para êste a previsão dêsse tipo de êrro.

Quando um sub-programa é escrito em FORTRAN, a construção do cabeçalho no programa objeto é feita pelo compilador; em SPS, porém, é o programador que o faz; isso permitiu que o cabeçalho de PILHA fôsse feito com o valor 99 na posição correspondente a F; no ponto em que o LOADER faz a verificação de F e K , e encontra um dêles diferente dos anteriores, foi introduzida uma modificação para que, se êste valor for 99, seja aceito; caso contrário, a operação deverá prosseguir normalmente.

As instruções originais que fazem o desvio para a rotina de êrro, ocupam na memória os enderêços 06698 e seguintes :

06698	24	07424	02219
06710	47	06746	01200
06722	24	07426	02221
06734	46	06758	01200
06746	17	07620	00076

Foram substituidas por:

06698	25	06742	00440
06710	34	06742	00701
06722	36	06742	00702
06734	49	07990	01142
06746	00	00107	990

Em 06742 está um "disk control field", cujo dígito inicial, obtido da área de comunicação pela primeira instrução , será 1 ou 3, conforme a unidade de disco em uso no momento.

Estas instruções trazem do disco uma pequena rotina auxiliar, AUXL6, que está gravada no setor 14200, e cuja finalidade é verificar se o valor de F é 99; caso o seja, ela restaura

INSTITUTO DE ENERGIA ATÔMICA

as instruções originais (4), e devolve o contrôle ao Loader , saltando o ponto em que seria detectado e registrado um êrro; posteriormente, durante a execução, PILHA se encarregará de obter o verdadeiro valor de F na área de comunicação, e usá-lo adequadamente; se F não fôr 99, será providenciado o desvio para a rotina de êrro, com a consequente inibição da execução.

A rotina AUXL6 ocupa na memória o enderêço 07990 e seguintes, e consta do seguinte:

07990	30	06698	08041
08002	49	06698	0
08010	14	07424	000$\bar{9}$9
08022	46	06758	01200
08034	49	06746	
08041	24	07424	02219
08053	47	08010	01200
08065	24	07426	02221
08077	46	06758	01200
08089	17	07620	0$\bar{0}$0≠

* * *

CAPÍTULO IV

DESCRIÇÃO DA ROTINA RECURS

RECURS foi escrito em SPS, e gravada no disco em "core image", a partir do setor 14000 , ocupando 18 setores. Seu endereço inicial de memória é 17700.

Daremos aqui algumas explicações para complementar o fluxograma da figura 2.

Durante a compilação, cada comando lido é colocado em uma área, de endereço simbólico CH15, e depois transferido para outra, CHI, onde será iniciada a sua análise. CHI corresponde a 15139, e CH15 a 15839.

Após terminar a análise de um elemento do comando fonte, é feito um "shift" do conteúdo de CHI, de maneira que aí estará sempre o próximo elemento a ser analisado.

O compilador, ao iniciar o exame do primeiro comando fonte, chama RECURS do disco, e desvia o contrôle para o endèrêço 17700. São as seguintes as ações de RECURS, a partir dêste momento:

4.1 - Reconstitui na memória as instruções originais do compilador, a partir da posição 03062; isto é necessário para que a compilação possa ser retomada normalmente após a intervenção de RECURS, que, como já dissemos, só se dá na análise do primeiro cartão; entretanto, como êste pode ser um comentário, a primeira das instruções em 03062, será ainda um desvio para 17700.

4.2 - Verifica se é um comentário, e em caso afirmativo desvia para 02878; isto teria sido feito pelas duas primeiras instruções de (1) que foram substituídas.

4.3 - Se não for comentário, verifica se o primeiro caráter é um $; caso o seja, passará a pesquisar o restante do comando:

4.3.1 - As colunas de 2 a 6 devem estar em branco; a partir da 7 deve haver as palavras RECURSIVE FUNCTION ou RECURSIVE SUBROUTINE; em seguida deve vir o nome do sub-programa, iniciado por um caráter alfabético, e contendo no máximo 6 caracteres alfanuméricos.

Ao encontrar o "(", se todas as condições anteriores foram satisfeitas, coloca um flag em SWT, para indicar a presença de um sub-programa recursivo; o nome do sub-programa é transferido para TAB.

A seguir apaga de CHI os caracteres $ RECURSIVE e devolve o contrôle ao compilador, após colocar no enderêço 07602 um desvio para GERAR. A finalidade dêste desvio é explicada a seguir:

A instrução original em 07602 corresponde ao reconhecimento, pelo compilador, da presença de um comando de sub-programa. Ao atingir êste ponto, a situação é a seguinte:

Em CHI se encontra parte do comando que está sendo analisado, isto é, o trecho que contém o "(" e a lista de argumentos; em CHI5 está o próximo comando fonte lido.

GERAR é um ponto de entrada em RECURS, que tem por finalidade transferir para a área BUFFER o registro contido em CHI5, e colocar em CHI5 o registro "CALL PILHA (1," seguido da lista de argumentos, que é transferida de CHI. Após estas providências, a compilação deverá continuar como se o próximo comando lido fôsse CALL PILHA $(1,A_1,A_2,\ldots A_n)$; com êste objetivo, RECURS inclue outra alteração no compilador, para que, terminada a análise de CHI, seja omitida a leitura de um novo cartão; assim, após o conteúdo de CHI5 ser transferido para CHI, o conteúdo de BUFFER será levado para CHI5, substituindo a próxima leitura; ao mesmo tempo, serão restituídas à memória as instruções originais.

A partir dêste ponto, RECURS não deverá mais ser solicitada, a não ser no momento em que for detectado um êrro 03; deve, pois, liberar a área da memória em que se encontra, para permitir ao compilador ocupá-la com a tabela de símbolos. É feita então, uma relocação do trecho de programa necessário a estas duas finalidades: Êle passará a ocupar o endereço 14800 e seguintes, e contém instruções para:

a) - Transmitir zeros e record marks para as posições a partir de 17700, até cobrir totalmente a área ocupada por RECURS.

b) - Fazer, no momento oportuno, o tratamento do erro o3, conforme foi explicado em 3.1.2.

c) - Devolver o contrôle ao compilador.

4.4 - Se alguma das condições expostas em 4.3.1 não for satisfeita, significará que o comando foi mal redigido; o contrôle deve então ser devolvido ao compilador, que se encarregará de acusar o erro; também neste caso é feita a relocação do trecho final para 14800, e a instrução original de detecção do erro 03 é devolvida ao endereço 07468. Dêste modo, o compilador estará totalmente reconstituído, e continuará a compilação dos próximos comandos fonte.

4.5 - Se o primeiro caráter não for um "$", significa que o programa fonte não é recursivo; as instruções originais e o contrôle são, então, devolvidos definitivamente ao compilador, do modo já explicado.

Em resumo: conforme foi salientado anteriormente, RECURS é solicitada na fase de análise léxica, para permitir a inclusão de novos símbolos na programação; no caso da recursividade, eles só aparecem no início de um programa fonte; mas poderiam ser incluídos mais alguns novos símbolos, que tivessem oportunidade de aparecer no decorrer de todo o programa; para isto, bastaria conservar a rotina auxiliar e o desvio para ela,

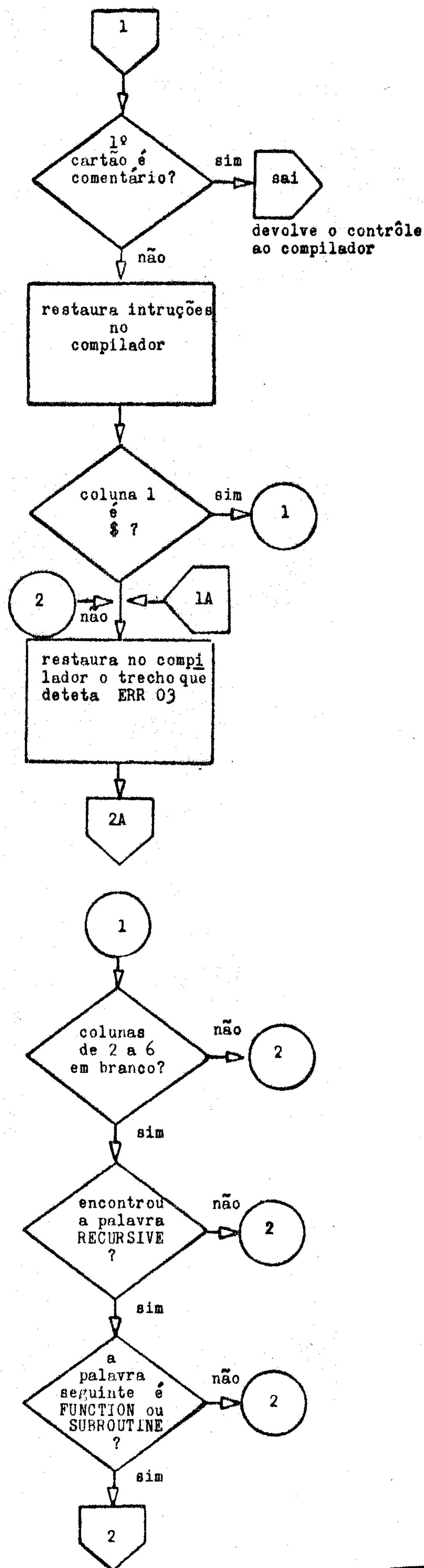
presentes na memória durante todo o tempo; tal fato viria acarretar, naturalmente, a necessidade de um exame cuidadoso da organização da memória durante tôda a compilação, para se aproveitarem as áreas disponíveis.

Por não ser necessário ao objetivo dêste trabalho, tal estudo não foi realizado aqui, mas considerâmo-lo perfeitamente viável, e de interêsse para trabalhos futuros.

A rotina auxiliar para a análise, nesse caso, teria a mesma lógica de RECURS, apenas ampliando-se a relação dos possíveis símbolos.

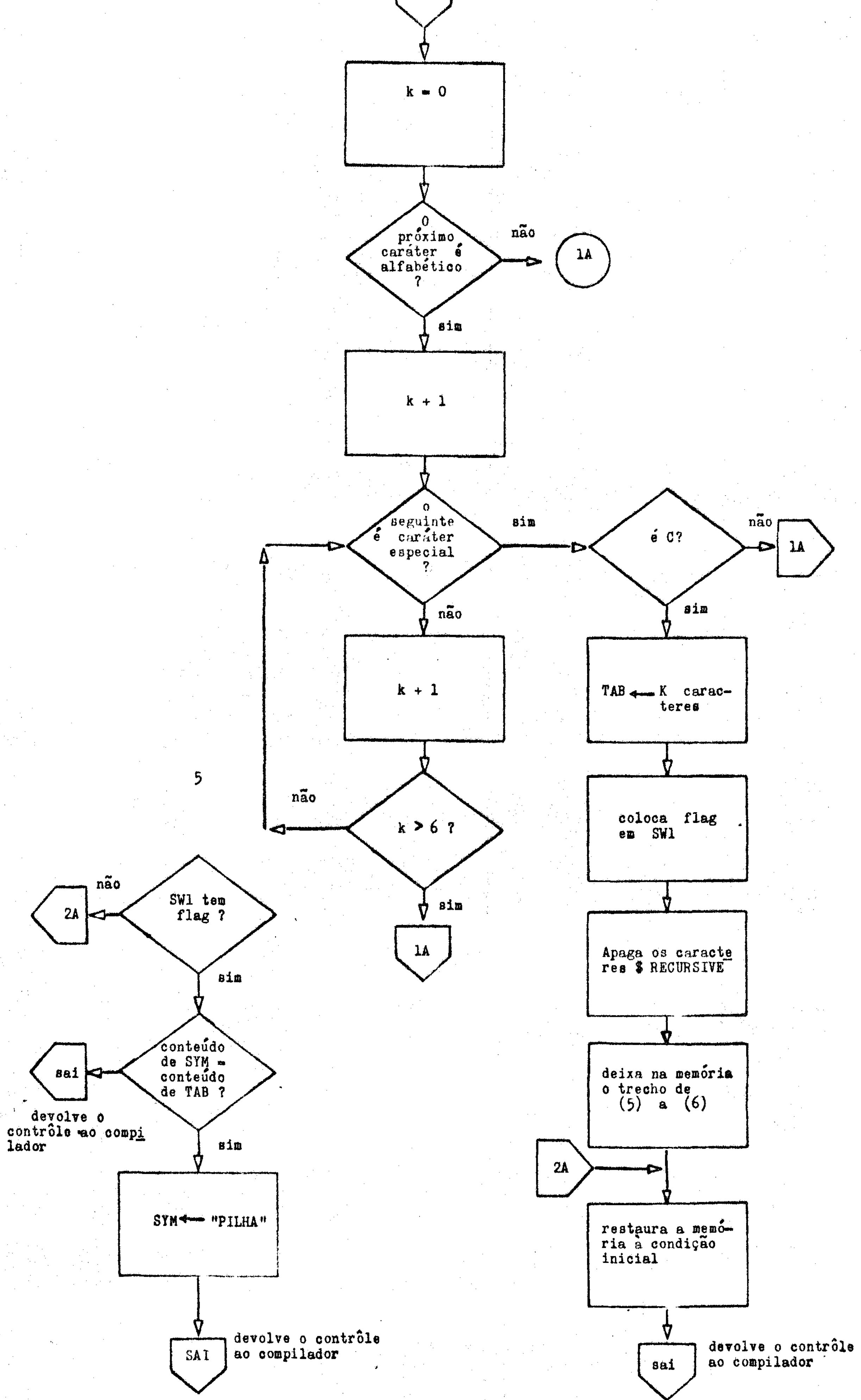
* * *

R E C U R S



2

INSTITUTO DE ENERGIA ATÔMICA


k = 0
O próximo caráter é alfabético ?
não
1A
sim
k + 1
o seguinte é caráter especial ?
sim
é C?
não
1A
sim
não
k + 1
TAB ⟵ K caracteres
5
não
k > 6 ?
coloca flag em SW1
sim
1A
Apaga os caracteres $ RECURSIVE
não
2A
SW1 tem flag ?
sim
sai
conteúdo de SYM = conteúdo de TAB ?
devolve o contrôle ao compilador
sim
deixa na memória o trecho de (5) a (6)
2A
restaura a memória à condição inicial
SYM ⟵ "PILHA"
SAI
devolve o contrôle ao compilador
sai
devolve o contrôle ao compilador

```
*** RECURS - ROTINA AUXILIAR PARA RECURSIVIDADE
       DORG17700
DESV   NOP START
       30   VOLTA,DMOD
       BD   TYPE,INDIV
       TDM DESV+1,9
       B    START
TYPE   TFM B+6,VOLTAT
       TFM 2816,VOLTAT
       B    REC
START CM   CHI ,43,10
       BE   VOLTA3
REC    30   VOLTA,INST
       CM   CHI,13,10
       BE   RECURS
***
HALT   BNF END,SWT
       CF   SWT
       B    B+6,,6
END    SF   ZRMK
       SM   E03+11,2,10
       CF   TAB+2
       CF   TAB+4
       CF   TAB+6
       CF   TAB+8
       CF   TAB+10
       TF   ORG+9,ZRMK+9
       TR   ORG+10,REST
       B7   ORG+10
*** A SER RELOCADO ***
ZRMK   DC   1,'
       DC   1,'
       DC   1,'
       DC   1,'
       DC   1,'
       DSC 5,0
REST   TF   17709,ORG+9,2,REST=ORG+10
       AM   ORG+16,10,10
       CM   ORG+16,19300
       BNP ORG+10
B      B7   VOLTAF
*** ERRO 03
E03    C    SYM,ORG+158,,7,, =ORG+66
       BE   ORG+102
BTERR BTM ERROR,73,09,PERMITE MENS. DE ERRO3
DOP    TD   INSW,0
       SF   SMADD
       SM   SMONT,3,10
       TF   SYM,ORG+178
       B7   CSYM
TAB    DSS 12
       DC   11,5749534841'
***
******  FIM DO REST
RECURSTFM TABELA+6,TAB+1
BRAN26BD  TDM,CHI+1,7
       AM   BRAN26+11,      1
       CM   BRAN26+11,CHI+11
       BNE BRAN26,,,TESTA SE 2 A 6 E BRANCO
```

```
       TFM FLAG+6,CHI+12 ,7
BRANI  CM  FLAG+6,0,610,,TESTA BRANCO NA COL. I
       BNE NOBRAN,,,NAO E BRANCO
I2     AM  I,1,10
       AM  FLAG+6,2,10
       CM  I,72,10
       BE  TDM
       B   BRANI
NOBRANC    J,JREC
       BN  BN
       BH  CJMAX
       TF  APAGA,FLAG+6
       SM  APAGA,2,10
       B   BN
CJMAX  C   J,JMAX
       BH  ISOLA
       BE  NOME
BN     AM  J,1,10
       AM  CFJ+11,2,10
CFJ    C   FLAG+6,FJ-2,67
       BE  I2
J10    CM  J,10,10
       BNE TDM
       CM  FLAG+6,62,610
       BNE TDM
       TFM JMAX,19,10
       TR  FJ+21,ALFSUB-1
       B   I2
NOME   AM  J,1,10
       CM  FLAG+6,40,610,,VER SE E LETRA
       BNH APAGAR
       CM  FLAG+6,70,610
       BNN APAGAR
       B   K1
ISOLA  CM  FLAG+6,40,610 ,,VER SE E CAR. ESPECIAL
       BH  K1
       CM  FLAG+6,24,610,,VER SE E (
       BNE APAGAR
SFT    SF  SWT
       30  7602,BGER
** APAGA $RECURSIVE
APAGARBNF R03,SWT
       B   APAG
R03    TF  DIMERR+11,BTERR+11
       TF  DIMERR+8,BTERR+8
APAG   TFM APAGA,0,2610
       CM  APAGA,CHI
       BNH TDM
       SM  APAGA,2,10
       B   APAGAR
K1     AM  K,1,10
       CM  K,6,10
       BH  APAGAR
       AM  E03+11,2,10
TABELATF      ,FLAG+6,11
       AM  TABELA+6,2,10
       B   I2
FLAG   DSS 7
I      DC  2,7
```

```
J       DC   2,0
K       DC   2,0
JMAX    DC   2,17
JREC    DC   2,9
KONT    DC   3,1'
FJ      DC   2,59
        DC   2,45
        DC   2,43
        DC   2,64
        DC   2,59
        DC   2,62
        DC   2,49
        DC   2,65
        DC   2,45
FUNC    DC   2,46
        DC   2,64
        DC   2,55
        DC   2,43
        DC   2,63
        DC   2,49
        DC   2,56
        DC   2,55
RMK     DC   5,0'
ALFSUBDC     2,42
        DC   2,59
        DC   2,56
        DC   2,64
        DC   2,63
        DC   2,49
        DC   2,55
        DC   2,45
        DC   1,'
CHI     DS   ,15139
TDM     DS   ,HALT
ORG     DS   ,14800
INSW    DS   ,2611
SMADD DS     ,7208
SMONT DS     ,7261
CSYM    DS   ,7202
SYM     DS   ,9979
DMOD    DSC 47,49177000000046028780120015151480000-43031
    30151'
APAGA DS     5
INDIV DS     ,2424
VOLTA DS      ,3062
VOLTA3DS     ,2878
VOLTAFDS     ,3086
VOLTATDS     ,3098
INST    CM   CHI,43,10
        DC   1,'
SWT     DS   ,  SFT+7
GERAR TR     BUFFER,15838,,GUARDA PROX. CARTAO
        30   15850,CP    ,,GERA CALL PILHA
        TR   15874,15140,,LEVA ARGUM.
        30   15840,ZEROF
BRM1    BNR BRM2,15879,7
        SM   BRM1+11,3,10
        30   BRM1+11,ZEROF+4,6
        B    SEGUE
```

```
BRM2   AM   BRM1+11,2,10
       B    BRM1
SEGUE  30   7602,ORIG
       TFM  2544,REC2
       B    7602
REC2   30   15838,BUFFER
       30   1946,M46
       TFM  2544,FIM
       B    532
FIM    TFM  2544,532
       TFM  1959,30,10
       30   1946,OR46
       TFM  B+6,532
       B    TDM
M46    TFM  1959,41,10
       DC   1,'
OR46   TD   1959,1968
       DC   1,'
ZEROF  DSC  9,-0-0-0-0'
CP     DSC  26,M3M1N3N3N7M9N3M8M1K4P1K3-'
BGER   B7   GERAR
       DC   1,'
ORIG   DSC  8,4407634'
BUFFERDSS   160
ERROR  DS   ,9350
DIMERRDS    ,7466
       DEND
```

CAPÍTULO V

DESCRIÇÃO DA ROTINA PILHA

A rotina PILHA tem por finalidade manipular a pilha para proteção e restauração dos argumentos. Foi escrita em SPS, sob a forma de sub-programa relocável, e será chamada, como já mencionamos, por todos os sub-programas recursivos; sendo assim, não tem o número de argumentos determinado a priori: êste dependerá do programa que a chama; no estado atual o número máximo de argumentos é dez , mas pode ser aumentado com a simples introdução de mais uma instrução DSA, que define os endereços simbólicos para os argumentos, em dois pontos do programa : INSUB e ARG (ver listagem).

5.1 - Uma chamada de sub-programa, em FORTRAN, é compilada da seguinte maneira:

```
BTM  SUB, * +11
DSA  A, B, ......
```

onde SUB é o ponto de entrada do sub-programa e A, B, ... são os endereços dos argumentos; * + 11 é o endereço transmitido para o sub-programa, e, posteriormente, com o auxílio do número de parâmetros, será ajustado e se tornará o endereço da instrução seguinte, isto é, o endereço de retôrno.

5.2 - PILHA

5.2.1 - Ao ser feito o desvio para PILHA, a primeira operação consiste em determinar o número de argumentos, o que é feito varrendo o programa que chamou, a partir do endereço transmitido pela instrução BTM.

5.2.2 - A seguir, os endereços dêsses argumentos são transmitidos para a sua posição, como é feito em todo sub-programa, e a rotina passa a pesquisar o endereço do programa que a chamou; isto é importante porque, na recursividade di-

reta, como já vimos em exemplo anterior (FAT), há uma nova chamada para o sub-programa recursivo, com os novos argumentos, isto é, PILHA, em vez de retornar à instrução seguinte à de chamada, deve guardar êsse enderêço para uso futuro, e chamar novamente FAT, com o valor de A-1 para argumento.

5.2.3 -Na área de comunicação, endereços 02219 e 02221 encontram-se, respectivamente, os valores de F e K, que PILHA irá usar para , sabendo qual o comprimento das palavras a serem empilhadas, calcular o incremento para cada nível.

5.2.4- Tôda a área disponível da memória é utilizada para a construção da pilha: Na área de comunicação, endereço 00434, encontra-se o endereço da primeira posição disponível, após a carga de todos os programas na memória; em 02231 está o endereço da última posição disponível, antes do início da área comum. Êstes endereços delimitam o comprimento da pilha.

Caso, no empilhamento, seja requerida mais memória do que a disponível, haverá uma mensagem : PILHA INSUFICIENTE , e o JOB será terminado.

5.2.5 - A pilha a ser construída terá os seguintes elementos:

1) Endereço de retôrno.(5 dígitos)

2) Endereço indireto do primeiro argumento, isto é, do ponto do sub-programa recursivo que contém o endereço do primeiro argumento . (5 dígitos)

3) Primeiro argumento .(F +2 dígitos)

4) Segundo argumento. (F + 2 dígitos)

.

.

.

n+2) N^{ésimo} argumento (F + 2 dígitos)

n+3) Incremento (3 dígitos)

O incremento é igual à soma dos comprimentos dos argumentos, mais

três.

5.2.6 - Uma vez colocados na pilha êstes elementos, há duas possibilidades a serem verificadas:

a) A chamada é do tipo 1: Nêste caso, o retôrno se dá para a instrução seguinte à de chamada, normalmente, pois não se trata de um sub-programa chamando a si mesmo;

b) A chamada é do tipo 2: Trata-se, pois, de uma chamada de sub-programa por êle mesmo. PILHA deve então chamar êsse sub-programa recursivo, levando o valor do argumento empilhado.

E assim, sucessivamente: O sub-programa chamará novamente PILHA com o argumento redefinido; PILHA guardará êste argumento, o seu enderêço no sub-programa, e o enderêço de retôrno.

5.2.7 - Quando fôr encontrado no sub-programa o comando RETURN, o enderêço de retôrno não será mais para o programa principal, e sim para o ponto de entrada em PILHA encarregado de iniciar o desempilhamento:

a) é retirado da pilha o enderêço de retôrno;

b) o enderêço indireto do argumento é utilizado para modificar no sub-programa os enderêços dos argumentos: Êstes apontarão agora para os últimos valores empilhados;

c) o contrôle é transferido para a instrução correspondente ao enderêço de retôrno.

Após executada essa instrução, novamente um comando RETURN fará com que seja feito mais um desempilhamento, e assim por diante, até ser atingido o fundo da pilha, que é assinalado por um record mark.

5.2.8- Nêste ponto, serão restaurados os enderêços iniciais dos argumentos, que estavam guardados na área END1 de PILHA, bem como o enderêço de retôrno do sub-programa

INSTITUTO DE ENERGIA ATÔMICA

recursivo ao programa que o chamou, e que foi conservado no endereço ENDV.

O contrôle é transferido para o endereço de retôrno.

Na figura 3 apresentamos o fluxograma de PILHA, e a seguir a sua listagem.

PILHA ocupa 2875 posições de memória.

* * *

PILHA

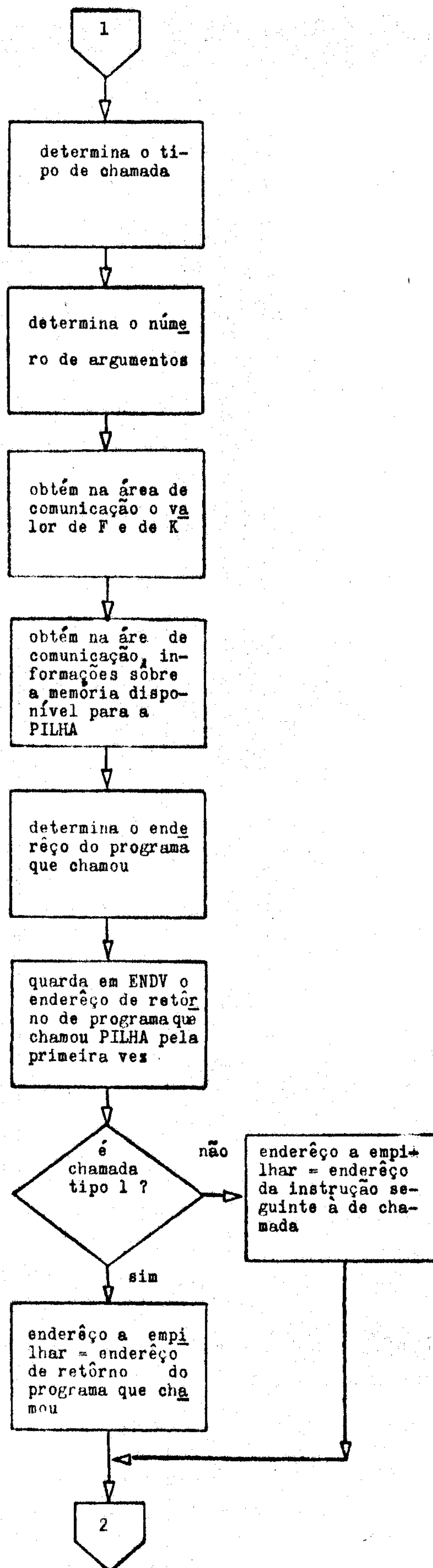
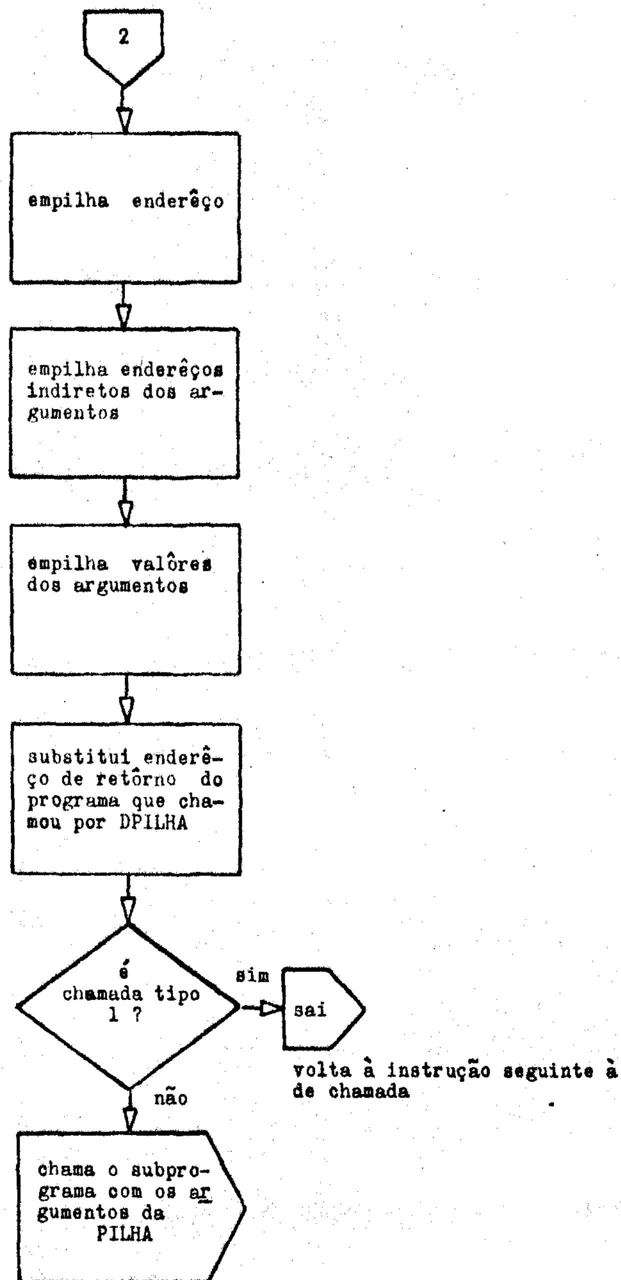
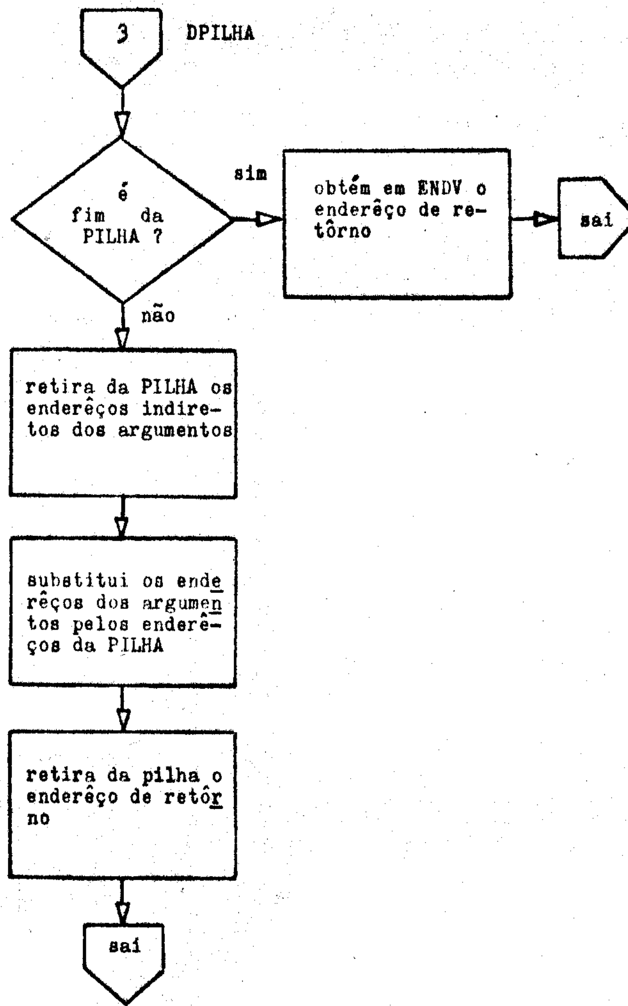


Fig. 3


2
empilha endereço
empilha endereços indiretos dos argumentos
empilha valôres dos argumentos
substitui endereço de retôrno do programa que chamou por DPILHA
é chamada tipo 1 ?
sim
sai
volta à instrução seguinte à de chamada
não
chama o subprograma com os argumentos da PILHA


3
DPILHA
é
fim da
PILHA ?
sim
obtém em ENDV o
endereço de re-
tôrno
sai
não
retira da PILHA os
endereços indire-
tos dos argumentos
substitui os ende
rêços dos argumen
tos pelos endere-
ços da PILHA
retira da pilha o
endereço de retôr
no
sai

```
***** PILHA*****
*** SUBPR. PARA MANIPULACAO DA PILHA DE ARGUMENTOS
ESSE  DS   ,*+101
      DC   6,987898,5-ESSE
      DAC 6,PILHA ,7-ESSE
      DVLC 22-ESSE,5,LAST,2,99,2,5,5,PILHA -6,5,0,30,0
      DSC 17,0,0
      DORG ESSE-100
INSUB DSA 0,0,0,0,0,0,0,0,0,0
RMK   DC   1,'
      DC   5,0
*** DET. DO NUM. DE PARAMETROS ***
*** VERIF. O TIPO DE CHAMADA
PILHA TF   TESTE,PILHA-1
      AM   TESTE,5,10
SFTST SF   TESTE
      TFL BUFFER,TESTE,11
      BNF FIXO,BUFFER-1
SFDIR SF   DIRETA
      TDM DNPAR+11,1
      TDM TRMO+11,2
      B    DNPAR
FIXO  SF   INDIR
CFDIR CF   DIRETA
SFULT SF   ULT
TD2   TDM DNPAR+11,2
      TDM TRMO+11,1
      AM   PILHA-1,5,10
****
DNPAR TDM AM3+11,
      TFM NPAR,1,8
      TF   AUX,PILHA-1
AM6   AM   AUX,6,10
BNF1  BNF TRMO,AUX,11
AM1   AM   NPAR,1,10
      AM   AUX,5,10
      BNF TRM,AUX,11
      AM   AUX,2,10
      BNF N1,AUX,11
B6    B    TRM
N1    AM   NPAR,1,10
      AM   AUX,3,10
B7    B    BNF1
TRMO  TDM AM3+11,
TRM   TFM TRM1+6,INSUB
      MM   NPAR,5,10
SF97  SF   97
      TF   NP5,99
SM1   SM   99,4,10
      A    TRM1+6,99
TRM1  TD   INSUB,RMK,2
**** TRANSM. DOS END. DOS ARGUMENTOS ***
***
      TFM TF+6,INSUB-4
SEG   AM   TF+6,4,10
      AM    PILHA-1,5,10
      TF   CF+11, PILHA-1,11
      BNF TF,  CF+11
CF    CF   CF+11
```

```
ES      TF   EAUX ,CF+11
SFIA    SF   IA
        TF   CF+11,CF+11.11
TF      TF   INSUB.CF+11
        AM   TF+6,1,10
        BNR   SEG .    TF+6.11
AM3     AM    PILHA-1.1.10..CONFORME O NUM. DE PAR.
        TDM TRM1+6.0.6
***
TFM1    TFM CONT,0.10
        TFM TFST+11,INSUB,711
        AM   TFST+11.5.10
NOP     NOP DEV ,,,SO E EXECUTADO NO PRIM. NIVEL
        TF   FST,2231,. GUARDA END. DA ULT. POSICAO OCUPA
     DA
        TF   F2,2219.. MANTISSA DO PTO. FLUT.
        TF   K,2221 ,. COMPRIM. VAR. PTO FIXO
        TF   K2,K
        AM   K2,2,10
        AM   F2,2,10
        TF   STACK,434
        TF   SETA,STACK
        TD   STACK,RMK.6
NZ      TFM NIVEL.0.10
        M    NPAR,F2
SF1     SF   97
        TF   MULT,99
        AM   MULT,3.10
        TDM NOP+1,9
***  DET. DO  END. DO SUBPR. QUE CHAMOU ***
DEV     BNF DET2,IA
CFIA    CF   IA
        TF   SET ,EAUX
        AM   SET ,1.10
TRMK    BNR S5,SET ,11
BO      B    STR1
DET2    TF   SET,INSUB
S1      SM   SET.1.10
        BNR S1 ,SET,11
STR1    TF   TR1+11.SET ..SET E O END. DO R.M.
        S    TR1+11,NP5
        TF   EARGS.TR1+11..GUARDA O END. DO PRIM. ARG. DO
       SUBP.
        AM   EARGS.4.10
NOP1    NOP BUSCA
TR1     TR   EARG1. ...GUARDA OS END.  INICIAIS DOS ARGUM
     .,, NO PRIM.N
        TDM NOP1+1.9
B3      B    BUSCA
S5      AM   SET ,5.10
B2      B    TRMK
BUSCA AM   SET ,1.10
BUSCA1BNF BUSCA,SET .11
BUSCA2TF   RMK1,SET ...RMK1 E O END. INICIAL DA BUSCA
CZ      TFM CON ,0,10
SOM     AM   SET ,1.10
        AM   CON ,1,10
        BNF SOM ,SET .11
C12     CM   CON ,2,10
```

```
       BN   CON16
       BE   CON11
CON4   CM   CON,4,10
       BE   BUSCA2
       B    CON5
CON11  SM   SET,1,10
       CM   SET,11,610
       BE   FINAL
       AM   SET,1,10
       B    BUSCA2
CON16  CM   SET,16,61011
       BE   FINAL
       B    BUSCA2
CON5   CM   CON,5,10
       BE   BUSCA2
       C    CON ,K
       BN   BUSCA
       BE   BUSCA2
       C    CON ,K2
       BN   BUSCA2
       A    RMK1,F2
       TF   SET ,RMK1
B1     B    BUSCA1
FIM    TF   END     ,SET
B5     B    NZ
FINAL  TF   END     ,RMK1
       TF   END1,END
       SM   END1,1,10
BBD    NOP  BD
TENDV  TF   ENDV,END1,11
       TDM  BBD+1,9
*** *** ***
*** EMPILHAMENTO ***
BD     BNF  SULT,DIRETA
       CF   ULT
       BNF  SETA5,INDIR
BCFIN  B    CFIND
SETA5  AM   SETA,5,10
       B    EMP1
CFIND  CF   INDIR
       TF   MULT1,SETA,11
SUB    S    SETA,MULT1
       SM   SETA,5,10
       SM   NIVEL,1,10
EMP1   TF   SETA,PILHA-1,6,EMP. ENDERECO ,NA CHAM. INT.
       B    SOMA1
SULT   SF   ULT
SOMA   AM   SETA,5,10,,EMPILHA ENDERECO ,NA CHAM. EXT.
       TF   SETA,END1,611
SOMA1  AM   SETA,5,10,,EMPILHA EARGS
       TF   SETA,EARGS,6
SOMA2  A    SETA,F2,,EMPILHA ARGUMENTOS
       C    SETA,FST
       BNN  ERRO
AARG   SM   TFST+11, 5,10
TFST   TFL  SETA  , ,6
       AM   CONT,1,10
       C    CONT,NPAR
       BNE  SOMA2
```

```
       AM   NIVEL,1,10   ,
       AM   SETA,3.10
       TF   SETA,MULT,6
       BNF  TFM2,DIRETA
BDIR   B    DIR
TFM2   TFM  END1,DPILHA.67
BI     BI   CHECK,1400
CHECK  BI   *+12,1500
BIND   B    PILHA-1,,6
*** CONSTR. DOS CAMPOS DE END. P/ CHAMAR O SUBP. REC.
    ***
DIR    TF   SETA1,SETA
       SM   SETA1,3,10
       TFM  TF2+6,ARG -5
       S    SETA,MULT
AM2    AM   TF2+6,5.10
       A    SETA,F2
TF2    TF   ,SETA
       C    SETA,SETA1
       BNE  AM2
       AM   SETA,3,10
*** ***
*** CONSTR. DO END. DE RETORNO ***
       TFM  TR2+6,ARG-5
       A    TR2+6,NP5
       A    TR2+6,AM3+11
TR2    30    ,BDP
       BI   CHEKK,1400
CHEKK  BI   *+12,1500
       BTM  END,*+11,67
ARG    DSA  0,0,0,0,0,0.0,0.0.0
DPILHACF    INDIR
       BNF  BNR,ULT,,FLAG INDICA O.  A ULT. FOI INDIRETA
CFULT  CF   ULT
EAUX2  DS   ,SFTST+11
K      DS   ,AM1+9, 2 DIG.
F2     DS   ,AM3+9, 2 DIG.
K2     DS   ,CFULT+11
EAUX   DS   ,SFDIR+11
TESTE  DS   ,BNPAR+11
BUFFERDS    ,INSUB+29
TIPR   DS   ,CFULT+7
ENDV   DS   ,B9+11
ULT    DS   ,TD2+9
DIRETADS    ,TD2+10
END1   DS   ,CFDIR +11
MULT1  DS   ,SFULT+11
       TF   MULT,SETA,11
       S    SETA,MULT
       SM   SETA,10.10
BNR    BNR  AS,SETA,11
B4     B    REC
AS     TF   MULT,SETA,11
       S    SETA,MULT
       SM   SETA,5,10
TF3    TF   PILHA-1.SETA,11
       SM   SETA,5,10
       BNR  TFS,SETA,11
SM2    TF   EAUX2,EARGS
```

```
       SM   EAUX2,4.10   ,
       TR   EAUX2,EARG1.6
B8     B    PILHA-1.,6
*** REC. DOS CAMPOS ***
REC    TDM NOP+1,1
       TDM NOP1+1,1
       TDM BBD+1,1
       TF   END1,ENDV,6
B9     B    ENDV,,6
TFS    TF   SETA1,SETA
       SM   SETA1,3.10
       S    SETA,MULT
       TF   EARGS,SETA.11
       A    SETA,F2
       TF   EAUX2,EARGS
TFS1   TF   EAUX2, SETA.6
       C    SETA,SETA1
       BN   ASF2
       AM   SETA,3.10
VOLTA  B    PILHA-1,,6
ASF2   A    SETA,F2
       AM   EAUX2, 5.10
B10    B    TFS1
ERRO   K    ,951
       PRA MENS
EXIT   CALLEXIT
END    DS   ,B0+11
SET    DS   ,B2+11
CON    DS   ,CZ+9,2DIG.
RMK1   DS   ,SFIA +11
STACK  DS   ,B1+11
FST    DS   ,ERRO+6
CONT   DS   ,TFM1+9. 2DIG.
NPAR   DS   ,BDIR+11
NIVEL  DS   ,SF1+11
MULT   DS   ,B10+11
AUX    DS   ,B7+11
SETA1  DS   ,CFIA+11
NP5    DS   ,SM1+9. 3 DIG.
SETA   DS   , SF98+11
EARG1.DSS 51
EARGS  DS   ,B5+11
MENS   DAC 18,PILHA INSUFICIENTE
IA     DS   ,BCFIN+11
INDIR  DS   ,CFULT+8
BDP    B7   DPILHA
       DC   1,'
LAST   DC   1,'
       DEND
```

CAPÍTULO VI

LIMITAÇÕES NO USO DA RECURSIVIDADE

6.1 - A principal restrição no uso da recursividade implementada no FORTRAN II-D do Sistema Monitor I, deve-se ao modo como são calculadas as expressões em FORTRAN. Esta limitação é mencionada também por Toscani[3], em trabalho análogo.

Usaremos como ilustração o cálculo da função de Ackermann, que é definida por:

$$Ack(a,b)=\left[a=0 \longrightarrow b+1,\ b=0 \longrightarrow Ack(a-1,1),\ Ack(a-1,Ack(a,b-1))\right]$$

O sub-programa recursivo para êste cálculo poderia ser, em princípio, escrito da seguinte maneira:

```
$       RECURSIVE FUNCTION RACKE (A,B)
        IF (A) 1,1,2
  1     RACKE = B+1.
        RETURN
  2     IF (B) 3,3,4
  3     RACKE = RACKE (A-1.,1.)
        RETURN
  4     RACKE = RACKE (A-1.,RACKE (A, B-1.))
        RETURN
        END
```

Examinemos o comando 4:

Será calculado um valôr para A-1., e em seguida entrar-se-á na chamada recursiva de RACKE (A,B-1.). Haverá diversás entradas na função, mas o valor de A-1., que se encontra em uma área temporária, não acompanhará a restauração dos valores do argumento A, a cada nível de recursividade: Isto faz com que o sub-programa forneça um resultado final errado, mas difícilmente percebido pelo usuário.

Êste efeito é devido à própria natureza do FORTRAN,

e só poderia ser corrigido se se fizesse o cálculo de tôdas as expressões por intermédio do uso de pilhas, como no ALGOL, e não apenas as chamadas recursivas; isto corresponderia a mudar todo o processo de compilação.

Para evitar o aparecimento dêsse efeito, o programador deve tomar a seguinte precaução:

Sempre que houver uma chama recursiva, esta chamada não pode ser precedida, na mesma expressão, por nenhum cálculo que envolva os argumentos da função. No exemplo acima, a chamada recursiva de RACKE está precedida pelo cálculo de A-1., envolvendo a variável A, que é argumento de RACKE. Com esta precaução, não ocorrerá o efeito citado.

Vejamos como deve ser, então, escrito o sub-programa para cálculo da Função de Ackermann:

```
$     RECURSIVE FUNCTION RACKE (A,B)
      IF (A) 1,1,2
 1    RACKE = B+1.
      RETURN
 2    IF (B) 3,3,4
 3    RACKE = RACKE (A-1.,1.)
      RETURN
 4    TRACK = RACKE (A,B-1.)
      RACKE = RACKE (A-1.,TRACK)
      RETURN
      END
```

Com isto, cada chamada recursiva cuidará da restauração de seus argumentos, e o resultado final será correto.

6.2 - Número e tipo dos argumentos

O número de argumentos em um sub-programa recursivo está limitado a 10, mas, como já mencionamos no capítulo V , basta introduzir-se mais uma instrução DSA em dois pontos do pro

grama, para aumentar o número permissível de argumentos.

Quando ao tipo, os argumentos devem ser todos de ponto flutuante; também esta restrição é contornável, mediante algumas alterações no programa PILHA, mas achamos interessante conservá-la por simplificar o programa, proporcionando economia de memória, que nêste ponto nos parece mais importante.

* * *

INSTITUTO DE ENERGIA ATÔMICA

CAPÍTULO VII

EXEMPLOS DE APLICAÇÃO

Nêste capítulo apresentamos, como exemplos, os programas para cálculo de algumas funções recursivas, e correspondentes problemas - amostra.

6.1 - Fatorial

6.2 - Máximo Divisor Comum

6.3 - Função de Ackermann

São apresentados ainda alguns programas com erros na redação, com as respectivas mensagens de êrro.

```
C*****FUNCAO PARA CALCULO DO FATORIAL RECURSIVAMENTE-RRFAT
$       RECURSIVE FUNCTION RRFAT(A)
        IF(A)1,1,2
      1 RRFAT=1.
        RETURN
      2 RRFAT=RRFAT(A-1.)*A
        RETURN
        END
```

```
C*****TESTE DE RRFAT
      1 READ 10,A
     10 FORMAT(I3)
        X=RRFAT(A)
        PRINT 20,A,X
     20 FORMAT(2X, 2HA=I3, 5X,7HFAT(A)=E14.8)
        GO TO 1
        END
```

```
00280 CORES USED
39999 NEXT COMMON
END OF COMPILATION

 A=  5     FAT(A)= .12000000E+03
 A=  9     FAT(A)= .36288000E+06
 A= 13     FAT(A)= .62270208E+10
```

```
C*****RESTO DA DIVISAO DE A POR B - RES
      FUNCTION RES(A,B)
      I=A/B
      AI=I
      RES=A-AI*B
      RETURN
      END
```

```
C*****CALCULO DO MAXIMO DIVISOR COMUM DE DOIS NUMEROS - RMD
     CM
$      RECURSIVE FUNCTION RMDCM(A,B)
      IF(A-B)2,2,1
    2 RMDCM=RMDCM(B,A)
      RETURN
    1 IF(B)3,3,4
    3 RMDCM=A
      RETURN
    4 RMDCM=RMDCM(B,RES(A,B))
      RETURN
      END
```

```
C*** TESTE DO RMDCM
    1 READ 10,A,B
   10 FORMAT(2I3)
      X=RMDCM(A,B)
      PRINT 100,A,B,X
  100 FORMAT (2X, 2HA=E14.8, 5X, 2HB=E14.8, 5X, 4HMDC=E14.8
     )
      GO TO 1
      END
```

```
0356 CORES USED
9999 NEXT COMMON
ND OF COMPILATION

A= .18000000E+02      B= .14000000E+02      MDC= .20000000E+01
A= .56000000E+02      B= .28000000E+02      MDC= .28000000E+02
A= .12000000E+02      B= .15000000E+02      MDC= .30000000E+01
```

```
C*****CALCULO DA FUNCAO DE ACKERMANN - RACKE
$       RECURSIVE FUNCTION RACKE(A.B)
        IF(A)1,1,2
      1 RACKE=B+1.
        RETURN
     2  IF(B)11,11,12
     11 RACKE=RACKE ((A-1.),1.)
        RETURN
     12 TRACK=RACKE (A,B-1.)
        RACKE=RACKE (A-1.,TRACK)
        RETURN
        END
```

```
C******       TESTE RACKE... (FUNCAO DE ACKERMAN)
     1  READ 10,A,B
     10 FORMAT(2I3)
        X=RACKE(A,B)
        PRINT 20,A,B,X
     20 FORMAT(2X,2HA=E14.8,5X,2HB=E14.8,5X, 9HACK(A.B)=,
       1E14.8)
        GO TO 1
        END
```

```
00366 CORES USED
39999 NEXT COMMON
END OF COMPILATION

 A= .00000000E-99      B= .10000000E+01      ACK(A.B)= .20000000E+01
 A= .10000000E+01      B= .00000000E-99      ACK(A.B)= .20000000E+01
 A= .20000000E+01      B= .10000000E+01      ACK(A.B)= .50000000E+01
 A= .10000000E+01      B= .40000000E+01      ACK(A.B)= .60000000E+01
 A= .10000000E+01      B= .30000000E+01      ACK(A.B)= .50000000E+01
 A= .30000000E+01      B= .10000000E+01      ACK(A.B)= .13000000E+02
 A= .20000000E+01      B= .20000000E+01      ACK(A.B)= .70000000E+01
```

BIBLIOGRAFIA

1- Barron D.W. - Recursive Techniques in Programming - Mac Donald - London 1969

2- Bolliet L. - Notation et Processus de Traduction des Langages Symboliques (tese) - Faculté des Sciences de Grenoble - 1967

3- Toscani S.S. - Recursividade em FORTRAN - (tese) - P.U.C. - R. de Janeiro - Guanabara

4- IBM 1620 - PR 033 Monitor System (Reference Manual)