C-PASCAL: SUPORTE PARA DESENVOLVIMENTO DE PROJETOS EM MICROCOMPUTADORES

José Carlos Martins Leite

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

Aprovada por:

Profª LÍGIA ALVES BARROS

Profª SUELI MENDES DOS SANTOS

Prof. JOSÉ LUCAS MOURÃO RANGEL NETTO

Prof. EBER ASSIS SCHMITZ

RIO DE JANEIRO, RJ - BRASIL
FEVEREIRO DE 1981

LEITE, JOSÉ CARLOS MARTINS

C-PASCAL: Suporte para Desenvolvimento de Projetos em Microcomputadores |Rio de Janeiro| 1981.

VII, 123 p. 29,7 cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1981)

Tese - Univ. Fed. Rio de Janeiro. Fac. Engenharia

1. Assunto: Compiladores e Linguagens Formais I. COPPE/UFRJ II. Título (série).

Ā

Ana Leite.

AGRADECIMENTOS

À Profª LÍGIA ALVES BARROS, pela excelente orientação, apoio, dedicação e todo esforço empregado no desenvolvimento deste trabalho.

À Profª SUELI MENDES DOS SANTOS, pelos incentivos e todos os conhecimentos transmitidos durante os cursos.

Aos Profs. JOSÉ LUCAS MOURÃO RANGEL NETO, ESTEVAM DE SIMONE, JANO MOREIRA DE SOUZA, LÍDIA MICHELA DE ANDA, pelos conhecimentos ministrados, bem como pelas valorosas orientações prestadas nos trabalhos do curso.

Ao CEPEL, e em particular aos Engs. ANTÔNIO LUIS BOGADO e JOÃO GUEDES DE CAMPOS BARROS, pelos incentivos e recursos recebidos para elaboração deste trabalho.

Aos colegas do CEPEL, MOTTA, LÚCIA, PAULO ROBERTO, CRISTINA, pelo apoio desinteressado dado a este trabalho.

À MARIA LIA, pela dedicação e empenho na datilografia desta tese.

RESUMO

A crescente utilização dos microcomputadores para aquisição e controle de dados gerados por equipamentos de simulação, requer grandes esforços na programação das rotinas para o tratamento destes resultados. Tais sistemas, e em particular os de baixo custo, necessitam de métodos mais simples, que a linguagem de máquina, para uma programação rápida, prática e segura sem prejuízo da eficiência do sistema. As linguagens de programação de alto nível (ALGOL, PL/I, PASCAL, etc...) são bastantes dispendiosas para serem implementadas nesta categoria de microcomputadores, por terem sido projetadas para equipamentos de maior porte.

O objetivo desta tese é oferecer uma linguagem estruturada e recursiva para ser usada em microcomputadores de baixo custo. A ênfase dada neste trabalho foi a tradução do código intermediário para um código executável diretamente pelo microprocessador. A linguagem C-PASCAL, elemento básico do SUPORTE (COMPILADOR, INTERPRETADOR e TRADUTOR), foi especificada com base no PASCAL (WIRTH) e implementada para o microprocessador INTEL 8Ø85.

ABSTRACT

Microcomputers in recent years have largely been used for acquisition and control of data generated by equipments of simulation. This utilization requires a great amount of effort in programming for the management of the results obtained. Microcomputer systems, particularly low cost systems, require programming methods, simpler, faster and safer than machine languages without affecting the performance of the system. Most of the high level programming languages (ALGOL, PL/I, PASCAL, etc...) are too costly to be implemented in such systems.

The main aim of this work is to make available a structured and recursive language to be used in low cost microcomputer systems. This thesis emphasizes the translation of intermediate code into a code which runs directly in the microcomputer. C-PASCAL language, the basic element of the SUPORT (compiler, interpreter and translator), was specified with basis in the PASCAL language (WIRTH) and implemented in the INTEL 8085.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

I.1 - OBJETIVO DA TESE

A Tese de Mestrado C-PASCAL: SUPORTE PARA DESENVOLVIMENTO DE PROJETOS EM MICRO-COMPUTADORES tem como objetivo imediato, colocar a disposição dos usuários de micro-computadores, uma linguagem estruturada que permita o desenvolvimento de projetos que utilizem micro-processadores. O segundo objetivo é oferecer condições para projetar um compilador completo da linguagem PASCAL[1] , trabalhando no próprio micro-computador.

Este trabalho foi desenvolvido com apoio, técnico e científico, do CENTRO DE PESQUISAS DE ENERGIA ELÉTRICA - CEPEL, motivado pela inexistência de 'software' nacional para micro-computadores, bem como da necessidade de uma linguagem estruturada para elaboração de projetos nesta área.

Em particular, o Suporte C-PASCAL foi desenvolvido para equiparar sistemas de baixo custo[2], capazes de suprir as necessidades dos laboratórios das Universidades Brasileiras, que necessitam de aparelhagem para pesquisa e desenvolvimento na área de microprocessadores e na maioria das vezes não dispõem de recursos suficientes para importar equipamentos completos. Portanto, o Suporte C-PASCAL é uma alternativa viável para sistemas de baixo custo, a fim de gerar condições para desenvolvimento de montadores 'assembler', "macro expansor", editor de texto, compiladores,etc..., visto que sua implementação exige ' um mínimo de 16KB de memória principal para funcionar normalmente.

Finalmente, o Suporte C-PASCAL é um 'pacote' aberto, sem nenhuma reserva de direitos, e com farta documentação por se tratar de um trabalho acadêmico. O Suporte pode ser facilmente adaptado a qualquer sistema de microprocessador, sem que haja prejuizo no desempenho de suas funções. Estas característi-' cas dão ao Suporte C-PASCAL um destaque especial visto que os

outros sistemas comerciais, 'packages', não permitem o acesso aos programas fontes e são projetados para um determinado tipo de máquina. Estes fatores fazem do Suporte C-PASCAL uma ferramenta bastante confiável para desenvolvimento e pesquisa de novas técnicas utilizando o micro-computador.

I.2 - DESENVOLVIMENTO DA TESE

O Suporte C-Pascal teve como ponto de partida o sistema "TINY" PASCAL[3], projetado por K.M.Chung e H.Yuen em BASIC (versão 'NORTH STAR BASIC'). Este sistema foi projetado para o micro-computador 'ALTAIR 8800', equipado com 36KB de memória e um sistema operacional 'NORTH STAR DISK'.

Dando continuidade ao sistema acima, desenvolvemos o Suporte C-PASCAL, que teve entre as principais fontes de idéias os seguintes sistemas: PASCAL[1], PASCAL UCSD[4], PASCAL DO PDP 11/70[5], LPM da COBRA[6], e o "TINY PASCAL entre outros. Durante o desenvolvimento da tese destacamos as seguintes tarefas:

1- Estudo de alguns microprocessadores, e em particular do INTEL 8Ø85[7];

2- Análise de alguns códigos intermediários, com maior atenção nos do "TINY"PASCAL e PASCAL UCSD;

3- Especificação da linguagem C-PASCAL;

4- Projeto de máquina C-PASCAL (virtual);

5- Estudo e escolha das melhores estruturas de dados e algorítmos a serem utilizados nos módulos do Suporte;

6- Programação e depuração dos módulos (Compilador, Interpretador e Tradutor);

7- Implementação no micro da INTEL 8Ø85.

O desenvolvimento do projeto Suporte C-PASCAL teve dois ambientes de trabalho: o primeiro no PDP 11/70, usando como linguagem de programação um sub-conjunto do PASCAL (existente no PDP 11/70) compatível com o C-PASCAL; e o segundo no microcomputador (INTEL 8085) usando a linguagem C-PASCAL.

A programação, depuração e implementação dos módulos, fases 5 e 6 descritas acima, tiveram os seguintes passos:

PASSO 1 - Programação do primeiro Compilador C-PASCAL no PDP 11/70 em PASCAL. Este módulo, que identificaremos por compilador/PDP, recebe como entrada um programa C-PASCAL e gera como saída uma listagem de compilação e um arquivo com os códigos intermediários do programa fonte.

PASSO 2 - Programação do Interpretador C-PASCAL no PDP 11/70,em PASCAL. Este módulo, que identificaremos por Interpretador/PDP, recebe como entrada os códigos gerados pelo compilador/PDP, e executa-os simulando a máquina virtual C-PASCAL. Neste estágio podemos testar a geração de código do compilador /PDP e avaliar o desempenho dos códigos intermediários.

PASSO 3 - Programação do Compilador C-PASCAL em C-PASCAL no PDP 11/70. Este passo consistiu, basicamente, da adaptação do programa Compilador/PDP em PASCAL para C-PASCAL. A seguir, compilamos este programa usando o compilador/PDP, e efetuamos exaustivos testes do código obtido com o interpretador/PDP. Os testes de 'performance' e a depuração da lógica do compilador C-PASCAL foram realizados durante esta fase com a compilação de vários programas e até mesmo do próprio compilador C-PASCAL.

PASSO 4 - Programação do Tradutor C-PASCAL no PDP 11/70, em PASCAL. Este módulo, que identificaremos por tradutor/PDP, recebe como entrada o arquivo de código intermediário e gera como saída um arquivo de código de máquina 8080 ou 8085 e uma listagem com os mnemônicos das instruções geradas (assembler). Com a listagem 'assembler' estudamos e testamos pequenos programas' C-PASCAL, a fim de avaliar o código 8085 gerado para determinadas estruturas. Com este estudo fizemos uma série de modificações no tradutor/PDP com o objetivo de reduzir o código de máquina 8085.

PASSO 5 - Programação do Interpretador e do Tradutor em C-PASCAL' no PDP 11/70. Este passo consistiu, basicamente, da adaptação dos programas Interpretador/PDP e Tradutor/PDP em PASCAL para C-PASCAL. A seguir, compilamos estes programas usando o compilador/PDP, e efetuamos exaustivos testes dos códigos intermediários com apoio do interpretador/PDP.

PASSO 6 - Montagem dos módulos do Suporte C-PASCAL para carga no micro-computador 8Ø85. Esta fase consistiu em submeter os códigos intermediários do Suporte (passos 3,5) ao tradutor/PDP, a fim de obtermos três arquivos de códigos de máquina 8085 no PDP 11/70. Para o compilador usamos o arquivo de código intermediário gerado no passo 3, e para o interpretador e tradutor os códigos obtidos no passo 5.

PASSO 7 - Programação do pacote de rotinas em assembler do 8085 no PDP 11/70, com auxílio do Montador da INTEL. Este pacote será usado pelos módulos do Suporte C-PASCAL no micro-computador em tempo de execução. O código de máquina 8Ø85 gerado pelo montador para estas rotinas, é gravado em disco, juntamente com os outros arquivos do passo 6.

PASSO 8 - Programação, no PDP 11/70, de uma rotina auxiliar para efetuar a transferência dos arquivos gerados nos passos 6 e 7 para o micro-computador 8Ø85. Este utilitário foi programado em PASCAL e utiliza um sistema de protocolo simplificado para executar a transmissão. Programação no micro-computador de uma rotina auxiliar para receber e montar na memória do micro cada módulo do Suporte C-PASCAL transmitido do PDP 11/70.

PASSO 9 - Carga do Suporte C-PASCAL no micro-computador INTEL 8085. A conexão foi feita através de uma linha física entre os dois equipamentos, e a cópia realizada diretamente na memória volátil ('RAM'). Ao final de cada transmissão, a região de memória, utilizada para receber cada módulo do Suporte, é salva em disco com auxílio dos utilitários do sistema CP/M (BIOS-BASIC I/O SYSTEM).

PASSO 10 - Depuração do Suporte C-PASCAL no micro-computador. Iniciamos a depuração pelo pacote de rotinas do Suporte com a ajuda do módulo 'DDT - DYNAMIC DEBBUGING TOOL ' existente no CP/M. Por fim, testamos exaustivamente cada módulo do Suporte C-PASCAL (Compilador, Tradutor e Interpretador) diretamente no micro-computador.

O Suporte C-PASCAL está sendo implementado no micro-computador do laboratório da Engenharia Eletrônica da UFRJ. Em paralelo, estão sendo desenvolvidos, por alunos do curso acima, trabalhos de final de curso usando a linguagem C-PASCAL, para

futuramente serem utilizados no laboratório.

O CEPEL já utiliza o Suporte no laboratório de simulação, numa versão compacta (Compilador/Tradutor) para desenvolvimento de programas de aplicação. A versão completa está gravada em disco flexível e foi bastante testada no desenvolvimento deste trabalho.

I.3 - DESCRIÇÃO DA TESE

A descrição do Suporte C-PASCAL, apresentada nesta monografia, se encontra nos capítulos (II,III,IV,V) que resumidamente apresentaremos abaixo:

CAPÍTULO II - COMPILADOR C-PASCAL: Descrevemos a máquina virtual C-PASCAL, o código intermediário e a linguagem C-PASCAL, bem como o método de compilação, as estruturas de dados e a técnica usada no recuperador de erros.

CAPÍTULO III - INTERPRETADOR C-PASCAL: Apresentamos o segundo módulo do Suporte, onde descrevemos sus estrutura, seus comandos e damos um exemplo de aplicação.

CAPÍTULO IV - TRADUTOR C-PASCAL: Apresentamos a técnica de emulação da máquina virtual C-PASCAL, usando os códigos alinhavados. Apresentamos também, o roteiro geral para se projetar um tradutor, bem como o projeto específico do tradutor para o INTEL 8080 ou 8085.

CAPÍTULO V - SUPORTE C-PASCAL: Apresentamos uma visão em conjunto do Suporte C-PASCAL, e mostramos uma aplicação em equipamento de pequeno porte.

CAPÍTULO II

COMPILADOR C-PASCAL

II.1 - ESPECIFICAÇÃO DA LINGUAGEM

O desenvolvimento da linguagem C-PASCAL teve como objetivo oferecer ao usuário de micro-computador uma ferramenta para desenvolvimento de projetos. De um modo geral, os usuários destas máquinas dispõem apenas de linguagens de montagem ou linguagens de médio nível[8] sem o mecanismo de recursão[9] .

Estando o projeto desta linguagem baseado no PASCAL[1], ela assimilou algumas das suas características, tais como:clareza recursão e estruturação. Entretanto para atender ao objetivo de facilitar a programação em micro-computadores, onde a interação usuário-máquina é mais estreita,foram introduzidos novos conceitos à linguagem, tais como: chamada de sub-rotinas externas ao programa (- comando CALL), acesso direto à memória (variável MEM), a não definição rígida de tipos e outros que serão explicados mais adiante. Além disto, para permitir sua utilização em equipamentos com pequena memória (mínimo 16KB) foram suprimidas certas características da linguagem base, que onerariam muito o compilador em tempo e espaço, e cujo percentual de utilização não justificam tal custo[10] para um projeto que é apenas a base para desenvolvimento de outros, inclusive o de um compilador da linguagem PASCAL completa.

II.1.1 - SUMÁRIO DA LINGUAGEM

Um programa C-PASCAL está caracterizado por duas partes:

a-) A parte descritiva, onde são definidos os dados a serem manipulados pelo programa, e que é feita através das declarações de LABEL, CONST, VAR,e das declarações dos sub-programas (PROCEDURE ou FUNCTION)

b-) A segunda parte, corpo do programa, com as ações a serem executadas, que são descritas através dos comandos.

Os dados são representados por variáveis, e seus no

mes devem constar nas declarações. Entretanto as declarações nesta linguagem não vinculam, rigidamente, o tipo à forma de usá-la, isto é, uma variável do tipo inteiro pode receber atribuição de um valor do tipo caracter ('B'), já que o valor do código ASCII do caracter está dentro do domínio dos valores possíveis de uma variável inteira. Portanto a declaração determina o domínio dos valores que a variável pode assumir, mas não restringe a forma de representá-la. Desta forma, o tipo básico de dado é o inteiro, e assume-se como definido implicitamente (mas não declarado) os tipos caracter e lógico. O caracter, representado por um símbolo entre aspas, é armazenado internamente pelo valor do seu código ASCII. O valor lógico é representado pelo 'bit' de mais baixa ordem de qualquer variável, sendo o valor 1 avaliado como condição verdadeira e o valor Ø como condição falsa.

O tipo estruturado é o arranjo linear de variáveis, todas do tipo inteiro. Este vetor tem seus limites definidos na declaração. O acesso aos elementos é feito através do cálculo de um índice, que é testado para verificar se está dentro dos limites declarados.

Para facilitar a interação usuário-máquina, a linguagem considera a memória do equipamento como um vetor predefinido de limites entre o menor e o maior endereço acessável, tal como o intervalo (Ø..16K-1), cujo nome é MEM. Portanto, o acesso a qualquer posição de memória se faz referindo-se a MEM [<expressão>] , onde a expressão irá representar um endereço absoluto. Para este vetor não há nenhum esquema de proteção à memória, que impeça o acesso a áreas do monitor ou do próprio código de programa. Deste modo, MEM é um instrumento poderoso , cômodo e ao mesmo tempo perigoso, para o usuário de micro-computador menos experiente.

As variáveis declaradas no programa principal, são ditas globais, e seu escopo está definido em todo programa. Variáveis definidas nos sub-programas só podem ser referenciadas dentro do corpo destes, e são chamadas de variáveis locais. A referência às variáveis não locais ao corpo do procedimento são resolvidas pela regra do escopo estático[11].

As ações, a serem efetuadas, são expressas sob a forma de comandos de atribuição. O controle destas ações é feito por

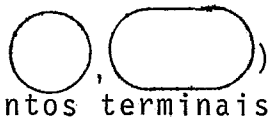
comandos condicionais e iterativos. A interação com o usuário é feita através dos comandos de leitura e escrita.

O comando de atribuição especifica um novo valor a ser recebido por uma variável. Este valor é obtido pela avaliação da expressão a direita do sinal de atribuição (:=).

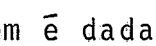
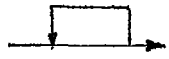
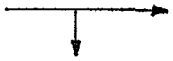
As expressões são constituidas de variáveis, constantes, operadores e funções que se relacionam e atuam segundo regras de precedência definidas na linguagem. O resultado final da avaliação é o novo valor da variável associada ao comando ' de atribuição. A linguagem C-PASCAL permite o uso de expressões 'mistas', onde podemos efetuar operações entre uma variável inteira e o resultado de uma sub-expressão lógica, bem como o tipo do resultado é determinado pelo contexto em que está a expressão.

II.1.2 - NOTAÇÃO, TERMINOLOGIA E VOCABULÁRIO

A notação utilizada para representar a sintaxe da linguagem é o Diagrama de Sintaxe [1].

As figuras circulares () envolvem os símbolos especiais, que são os elementos terminais da linguagem.

Os elementos sintáticos que são os não-terminais, como variável, expressão, etc, estão nas figuras retangulares.

A sequência na qual devem estar dispostos os componentes da linguagem é dada por uma semi-reta orientada (), as repetições indicadas por  e as alternativas por  .

O vocabulário consiste de elementos básicos classificados por letras, dígitos e símbolos especiais. Para definir os elementos básicos apresentamos abaixo na forma padronizada de BNF (BACKUS-NAUR FORM).

```
          <letra>  ::= A | B | C | D | E | F | G | H | I | J | K | L | M
                       N | O | P | Q | R | S | T | U | V | W | X | Y | Z
          <dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<símbolo especial> ::= + | - | * | = | <> | < | > | <= | >= | ( | ) | [ | ]
                       (* | *) | := | . | , | ; | : | & | % | # | $ | DIV
                       MOD | OR | AND | NOT | IF | THEN | ELSE |
                       CASE | OF | REPEAT | UNTIL | WHILE | DO |
                       FOR | TO | DOWNTO | BEGIN | END | GOTO |
                       CONST | VAR | LABEL | ARRAY | FUNCTION |
                       PROCEDURE | PROGRAM
```

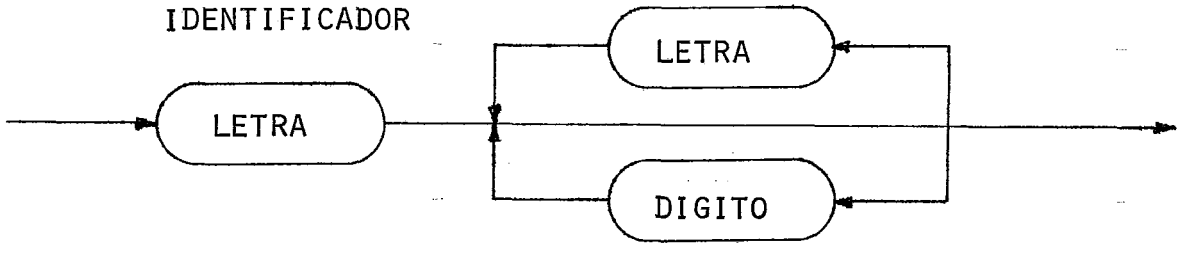
<dígito hexa> ::= Ø | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F

A construção

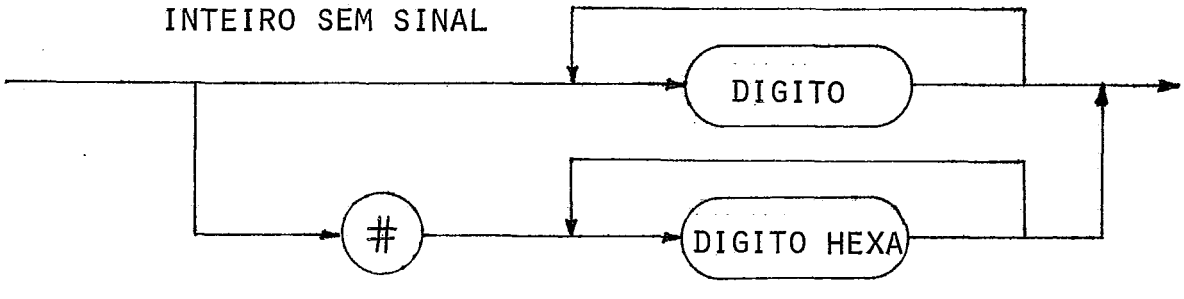
(* qualquer sequência de símbolos,menos '*)' *)

pode ser inserida entre dois identificadores, números ou símbolos especiais, sem alterar o significado do programa. Esta construção é usada para inserir comentários no programa fonte C-PASCAL.

II.1.3 - IDENTIFICADORES, NÚMEROS E CONSTANTES



Os identificadores servem para nomear constantes, variáveis, procedimentos e funções, não sendo permitido usar palavras reservadas como identificadores, e sua declaração deve ser única dentro do escopo de validade. No C-PASCAL não existe limite para o tamanho do identificador, e todos os símbolos (letras ou dígitos) são usados para sua identificação.

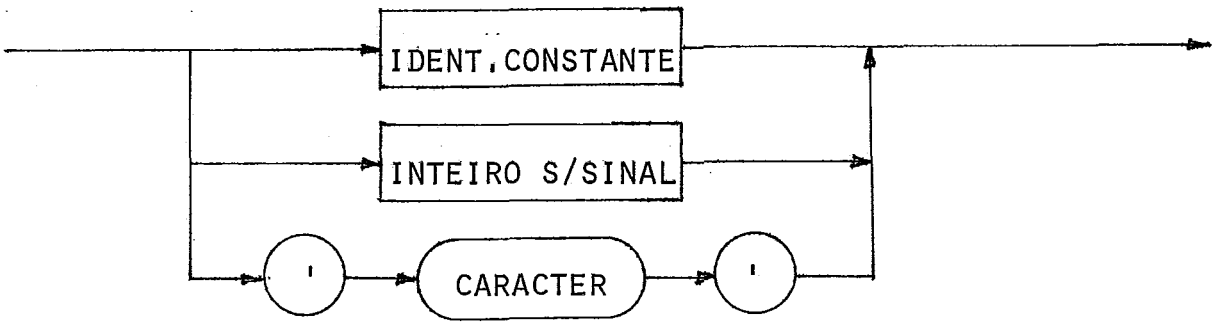


A notação decimal é usada para representar os números, assim como o formato hexadecimal. Os inteiros são avaliados e armazenados internamente na forma binária em palavras de 16-bits. O domínio dos inteiros (-32.768.. 32.767).

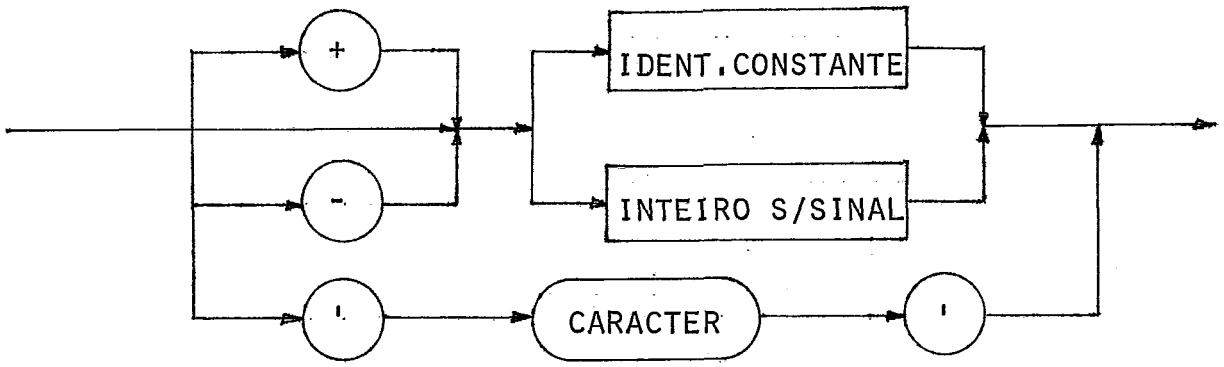
O formato hexadecimal, indicado pelo símbolo ' # ', foi introduzido para faciliatr o endereçamento de memória, bem

como o uso de máscaras em campos de 'bits' de uma variável. Os inteiros hexas são avaliados no intervalo (#ØØØØ.. #FFFF).

CONSTANTE SEM SINAL



CONSTANTE

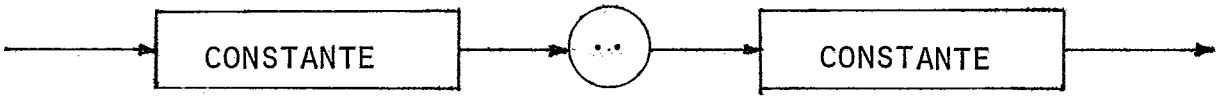


A declaração de uma constante introduz no programa um identificador como sinônimo de uma constante. As constantes negativas são representadas internamente pelo complemento a 2. Um caracter entre apóstrofos ('A') é denotado como uma constante 'CHAR' , e seu valor interno é obtido pelo respectivo código ASCII .

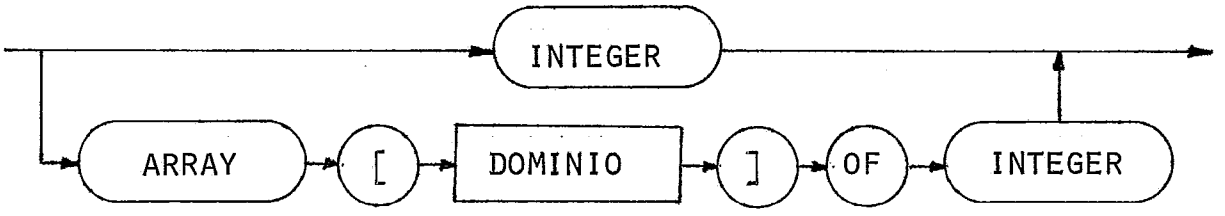
Por exemplo:	'*'	'G'	'3'	'A'
	#Ø	#FF		#A
	Ø	255	-1	10

II.1.4 - DEFINIÇÃO DE TIPOS

DOMÍNIO



TIPO



Os elementos de dados (numéricos, lógicos, caracteres, rótulos, etc...) são todos representados internamente na forma binária, em palavras de 16-bits e complemento a 2.

As linguagens de programação permitem a abstração desta representação interna, por meio da especificação dos conjuntos de elementos de dados com os respectivos operadores. A linguagem C-PASCAL não faz restrição quanto ao tipo e à forma de uso dos seus operadores com suas variáveis, que inicialmente são declaradas como inteiras.

Por exemplo:
```
FOR  I := 'A'  TO  '2'  DO  WRITE  (TTY,&I) ;
     X :=  B<>C ;
     Y :=  B = C ;
     Z :=  B AND C ;
     IF  X  THEN  <comando > ELSE < comando > ;
```

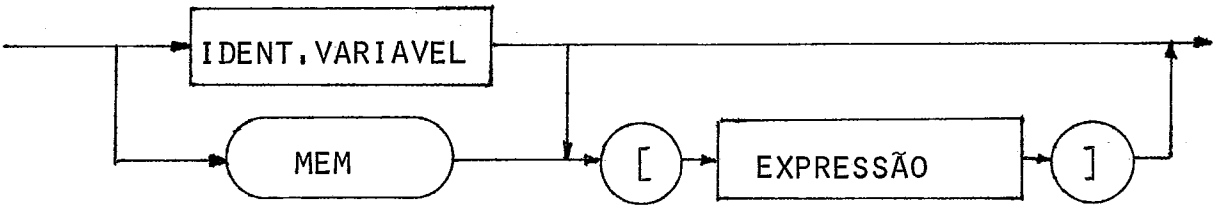
Os tipos padrões CHAR e BOOLEAN não existem explicitamente na sintaxe da linguagem, porém estas duas características podem ser usadas normalmente em programação C-PASCAL. A condição 'booleana' é avaliada pelo dígito binário de mais baixa ordem, sendo condição verdadeira o valor 1 e falsa o valor Ø.O resultado de uma expressão lógica pode ser Ø ou 1, porém o resultado de uma expressão aritmética quando analisado como lógico somente o 'bit' de mais baixa ordem é observado.

O tipo estruturado ARRAY consiste de um número fixo de elementos agrupados numa estrutura vetorial, sendo todos do tipo inteiro. Cada componente deste arranjo, pode ser explicita

mente referenciado e manipulado pelo programador.

II.1.5 - VARIÁVEIS

VARIÁVEL



As variáveis são todas representadas internamente em palavras de 16-bits, sendo um 'bit' para sinal e o restante magnitude em complementação a 2.

A variável simples é especificada pelo seu identificador.

A variável indexada é referenciada pelo identificador do arranjo seguido da expressão de índice, entre colchetes ' IDENT [<exp>] , que especificará um único elemento do vetor. O índice da variável indexada é obtido pela avaliação da expressão, e em seguida testado dentro do domínio. Se ocorrer um índice inválido, durante a execução do programa, será dada mensagem de erro correspondente. O teste de índice é oneroso em espaço e tempo de execução, portanto o seu uso é controlado pelo programador que pode optar pela inserção ou remoção deste teste no código,gerado durante a fase de compilação, com a aplicação da diretiva (*?*).

Por exemplo:
```
PROGRAM TESTE-DE-INDICE :
VAR VETOR : ARRAY [15..30] OF INTERGER; I:INTERGER;
BEGIN
     VETOR [I]:= <exp> ; (* COM TESTE DE ÍNDICE*)
(*?*)
     VETOR [I]:= <exp> ; (* SEM TESTE DE ÍNDICE*)
(*?*)
     VETOR [I]:=  10   ; (* COM TESTE DE ÍNDICE*)
END.
```

Com o objetivo de permitir o acesso direto à memória do micro-computador, criamos um identificador para especificar

o vetor memória denotado por MEM[<exp>] . Cada posição de memória é referenciada como uma variável indexada, sendo o resultado da expressão de índice o endereço absoluto de memória a ser acessada. Cada componente deste vetor representa, no caso do 8Ø8Ø/8Ø85 da INTEL, uma palavra de 8-bits (1'byte'), sendo possível ler ou escrever valores no domínio [Ø..255] ou[#ØØ..#FF]. Na avaliação de uma expressão, o valor de um elemento do vetor MEM é estendido para dezesseis 'bits', ficando compatível com as demais variáveis, sendo seu valor mantido na parte baixa da cadeia estendida.

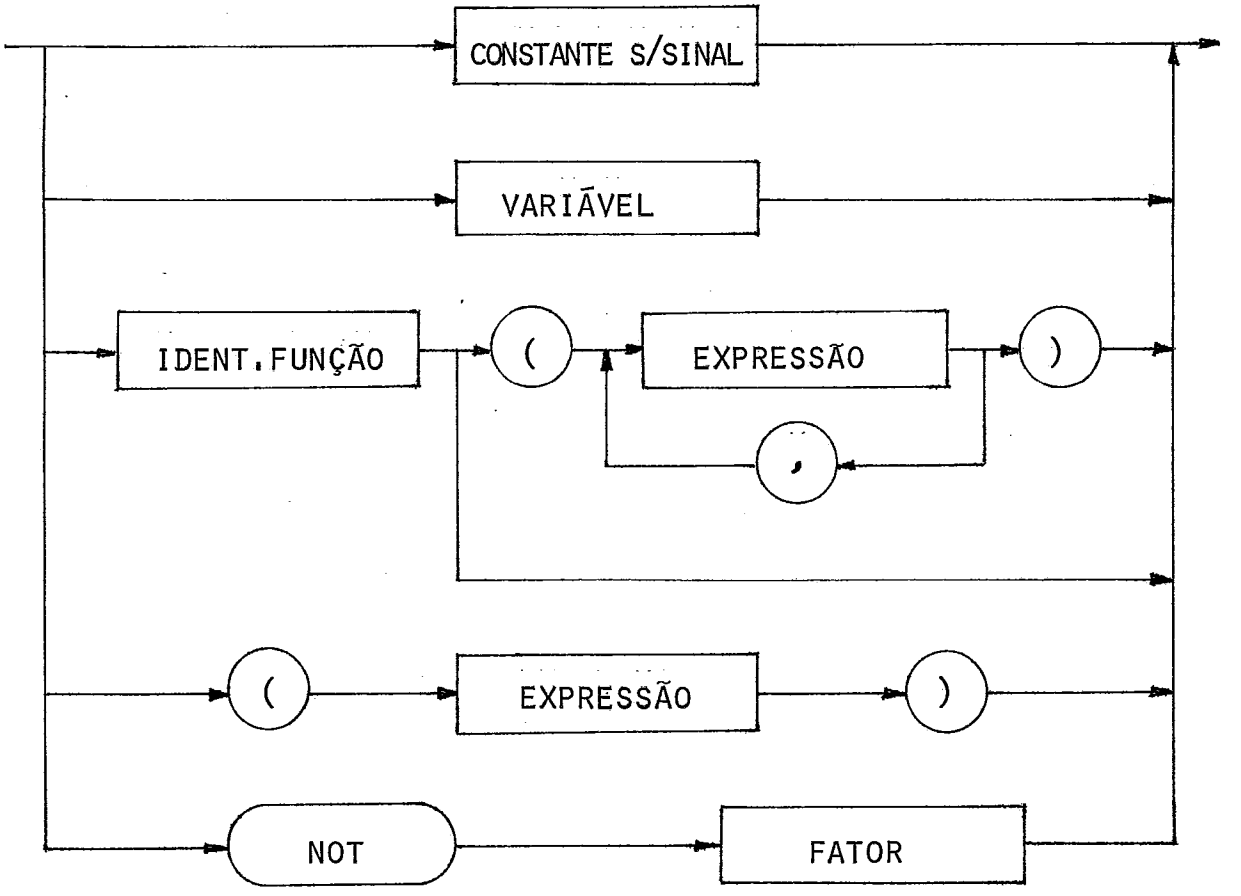
Por exemplo:
```
I := I * MEM [15] + MEM[18] ;
IF MEM [I] = 'A' THEN <comando> ;
```

```
PROGRAM ZERARMEMÓRIA ;
VAR I : INTERGER;
BEGIN
I := #EFFF ;
REPEAT  I := I+1 ;
        MEM [I] := Ø  ;
UNTIL   MEM [I] <> Ø  ;
END .
```

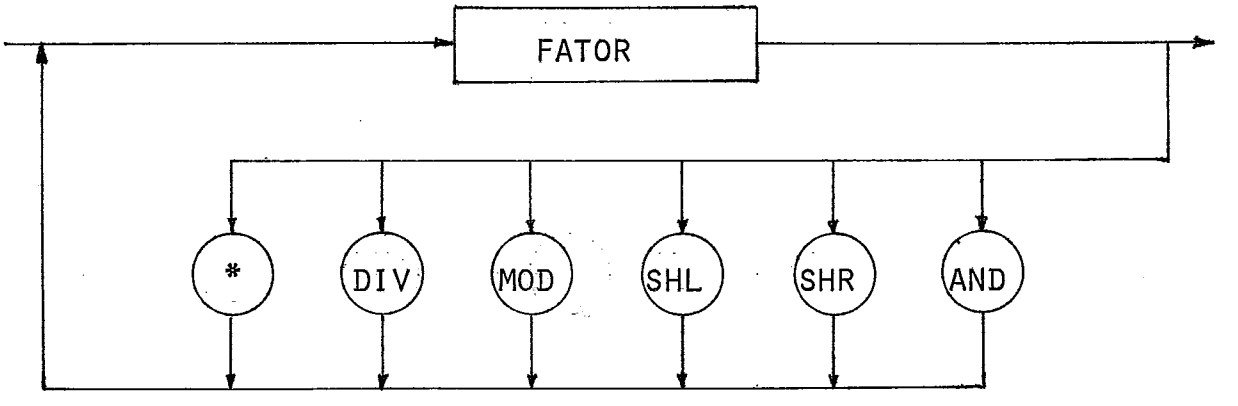
Certas concepções de 'hardware' fornecem a constante 255 às leituras feitas fora da memória real, o programa ZERAR MEMORIA, descrito acima, fará uma varredura da memória, preenchendo-a com zero, a partir do endereço (#EFFF) até encontrar o final da memória real, que corresponde a ler um valor diferente de zero escrito na ação anterior.

II.1.6 - EXPRESSÃO

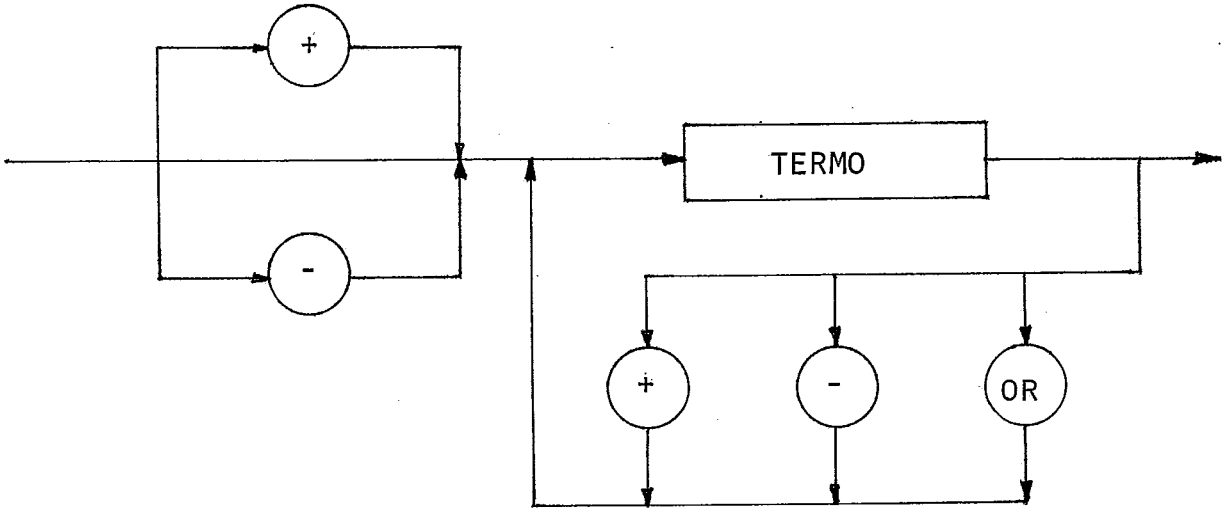
FATOR



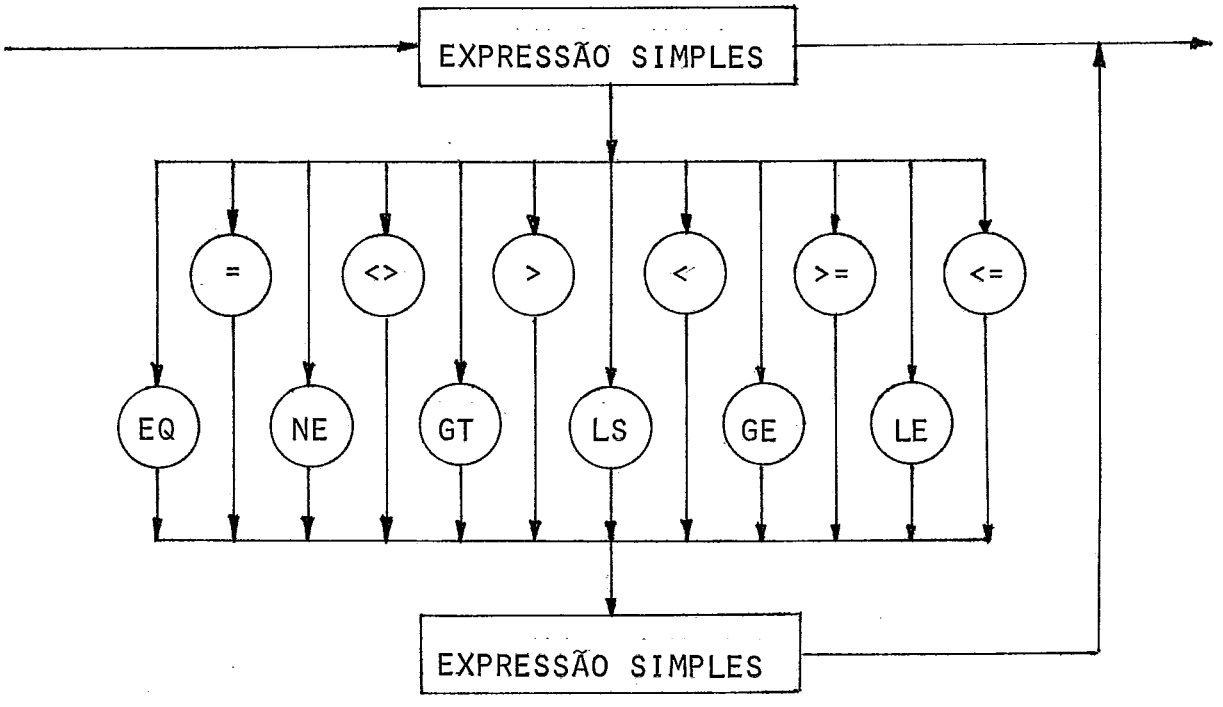
TERMO



EXPRESSÃO SIMPLES



EXPRESSÃO

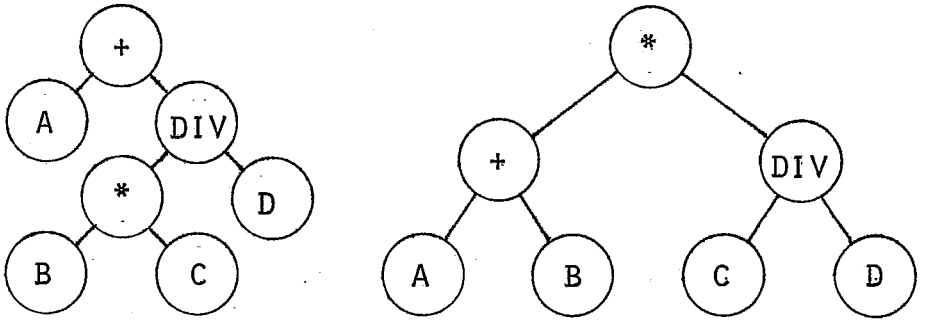


A expressão é uma construção que obedece determinadas regras de formação a fim de obter valores de variáveis, e gerar novos valores com a aplicação dos operadores. As expressões são constituidas de operadores e operandos, isto é, variáveis, constantes, e funções.

As regras de avaliação especificam as precedências dos operadores, classificados em quatro grupos: O operador "NOT" tem a maior precedência, seguido pelo grupo dos operadores de multiplicação (*, DIV, MOD, SHL, SHR, AND), depois pelos operadores de adição (+, -, OR) e com mais baixa precedência o grupo dos operadores relacionais (=,<>,>=,<=,> ,< ,). A precedência entre operadores do mesmo grupo é resolvida pelo sentido esquerda para direita, na avaliação da expressão.

As regras de precedência estão definidas na própria sintaxe da linguagem, como apresentamos nos diagramas referentes a expressão (FATOR, TERMO, EXPRESSÃO SIMPLES, EXPRESSÃO). Por exemplo:

A + B * C DIV D e (A+B) * (C DIV D)



2 + 5 * 8 DIV 4 = 12 (2+5) * (8 DIV 4) = 9

OPERADOR NOT

O operador NOT representa a negação lógica do seu operando. Na linguagem C-PASCAL, este operador quando aplicado a expressão acarretará uma complementação a 1 do resultado.

OPERADORES DE MULTIPLIÇACÃO

Os operadores de multiplicação são todos binários, e usam o (topo) e (topo-1) da pilha de avaliação como seus operandos, e após a operação armazenam o resultado no (topo).

A operação de multiplicação (*) é executada entre

dois operandos da pilha, e o produto calculado com precisão simples (16-bits).

A operação de divisão (DIV) é executada entre o dividendo (topo) e o divisor (topo-1), e seu quociente é truncado (não é feito arredondamento)[1] . Em tempo de execução são detectados dois erros nesta operação: DIVISÃO POR ZERO e DIVISÃO POR -32.768

Por exemplo:

```
 5 DIV 3 =  1        5 DIV -3 = -1
-5 DIV 3 = -1       -5 div -3 =  1
```

A operação resto da divisão (MOD) obedece a seguinte regra: a MOD b = a - (a DIV b) * b

Por exemplo:

```
 5 MOD 3 =  2        5 MOD -3 =  2
-5 MOD 3 = -2       -5 MOD -3 = -2
```

As operações de deslocamento (SHL,SHR) foram introduzidas no C-PASCAL para aproveitar com mais eficiência as instruções de 'SHIFT' disponíveis nos micro-computadores, bem como permitir o deslocamento lógico de uma cadeia de'bits' associada a uma variável. A característica do deslocamento lógico é o preenchimento com zeros da parte deslocada.

Por exemplo:

```
5 * 2 = 5 SHL 1 =  1Ø
#ØØFF SHL 4 = #ØFFØ
15 DIV 4 = 15 SHR 2 = 3
```

A operação AND gera, como resultado, uma cadeia de 16-bits proveniente de um AND lógico feito entre os dois operandos 'bit' a 'bit'.

Por exemplo:

```
#FAC5 AND #ØØFØ = #ØØCØ
WHILE (A>B) AND (B<>C) DO <comando> ;
WHILE (B AND #ØØFF) < 15 DO  <comando> ;
```

OPERADORES DE ADIÇÃO

A operação soma (+) é executada diretamente pela maioria dos microprocessadores, porque normalmente eles dispõem de instruções de soma em 16-bits.

O operador menos (-) pode ser usado como operador

unário que irá acarretar uma complementação a 2 do seu operando, ou como operador binário de subtração.

O operador de adição lógica pode ser aplicado como alternativa entre expressões 'booleanas', bem como entre variáveis ' inteiras.

Por exemplo:

```
IF (A > B) OR (C=D)  THEN <comando> ;
#FØFØ  OR  #Ø1ØØ  = #F1FØ
```

OPERADORES RELACIONAIS

O primeiro conjunto de operadores relacionais ' (=, <>, > , < , >=, <=) é aplicado para comparar variáveis, levando em consideração o sinal, no domínio (-32.768.. + 32.767).

O segundo conjunto (EQ,NE,GT,LS,GE,LE) é usado para testar variáveis, sem levar em conta o sinal, no intervalo (Ø..65.535), que foi introduzido na linguagem para comparar endereços.

Por exemplo:

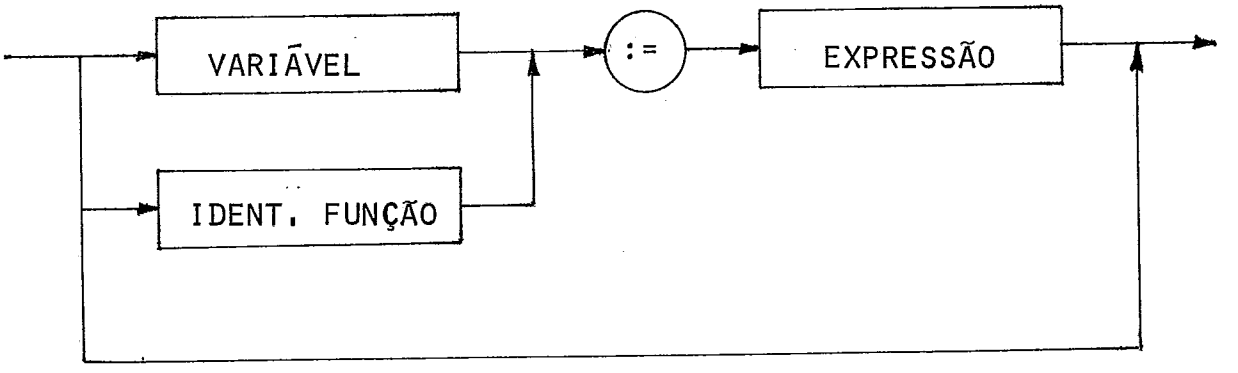
```
A := #FFFF ; B := Ø ;     A > B  é falso
                          A GT B  é verdadeiro

A :=  5ØØ ; B := -1 ;     A > B  é verdadeiro
                          A GT B  é falso

A :=  3ØØ ; B := 5ØØ ;    A > B  é falso
                          A GT B  é falso

A :=  5ØØ ; B := 3ØØ ;    A > B  é verdadeiro
                          A GT B  é falso
```

II.1.7 - COMANDOS

Os comandos representam as ações a serem executadas ' no programa. Eles podem ser prefixados por rótulos numéricos, para que sejam referenciados pelo comando 'GOTO'.

COMANDOS SIMPLES

Os comandos simples são aqueles que não geram outros' comandos. O comando vazio não executa nenhuma ação, já que não possui nenhum símbolo.

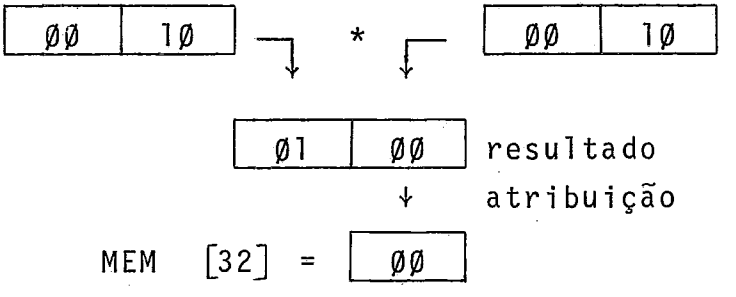


O comando de atribuição serve para trocar o valor corrente de uma variável (ou função) por um novo, especificado por uma expressão.
Por exemplo:

```
A := 'B' ;              A := 5 * B + C DIV 15 ;
A := (B > C) ;          A := (B > C) OR (C = Ø) ;
A := #FF ;              A := B AND C ;
```

A atribuição feita a um elemento do vetor MEM será parcial, sendo apenas a parte baixa do resultado da expressão, usada para carregar o novo valor na posição de memória acessada.
Por exemplo:

```
MEM [32] :=     16      *      16
```



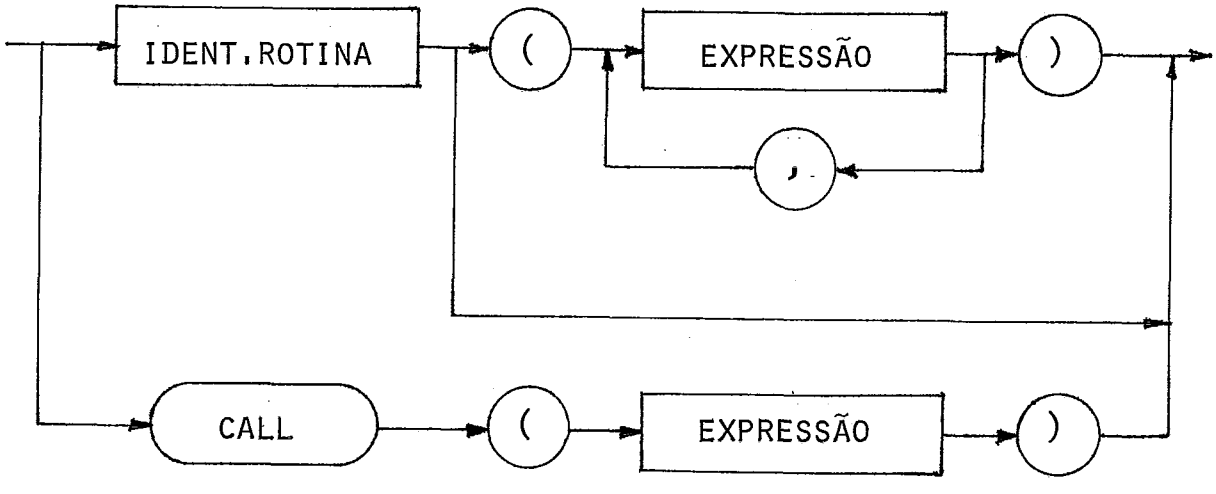
Para armazenar uma variável de 16-bits diretamente na memória, primeiro temos que reservar duas posições de memória e depois efetuarmos duas atribuições parciais (parte baixa e parte alta da variável).

Por exemplo:

```
                                        alta  baixa
VAR1  := 256 ;                        | 01 | 00 |
                                          ↓
MEM [5] := VAR1;          MEM 5 →     | 00 |
                                          ↓
MEM [6] := VAR1 SHR 8;                | 00 | 01 |   resultado da ex
                                          ↓          pressão
                          MEM 6 →     | 01 |
```

para recuperar uma variável diretamente da memória, temos:

```
VAR1  := MEM [5] + MEM [6]  SHL 8 ;
```



O comando 'PROCEDURE' serve para ativar um sub-programa declarado anteriormente. A chamada de um procedimento pode conter uma lista com os parâmetros reais, que irão substituir seus correspondentes parâmetros formais, definidos na declaração do procedimento. A chamada de uma 'PROCEDURE' é precedida da avaliação dos seus parâmetros reais, representados por expressões que têm seus resultados armazenados na pilha, e da criação do registro de ativação para depois efetuar a execução dos comandos da rotina. A rotina se encarrega, no início, de alocar uma área de dados para suas variáveis locais, bem como no final de restaurar a pilha pela deslocação do segmento de dados, do registro de ativação e dos parâmetros reais.

O comando CALL (< exp >) foi introduzido, na sintaxe da linguagem, para permitir o acesso às rotinas externas. Estas

rotinas podem ser provinientes de uma programação assembler pura (rotinas do monitor,tratamento de interrupções por 'software', etc...), bem como de uma programação C-PASCAL. A chamada de sub-programas, compilados separadamente, é feita de uma forma simbólica por um identificador de procedimento, que irá gerar o comando CALL (< exp >) para o específico endereço da rotina. Este procedimento especifica no programa todos os parâmetros formais necessários à rotina que será utilizada.

A técnica utilizada na passagem de parâmetros, juntamente com os recursos do comando CALL (< exp >) tornaram possível o uso da programação modular.

A programação modular consiste em dividir um programa em sub-programas (PROCEDURES ou FUNCTIONS) e compilá-los separadamente, ficando o programa principal com as referências destes sub-programas para acessá-los em tempo de execução.

Na compilação do programa principal fazemos simplesmente referências a estas rotinas externas, indicando o endereço de memória em que elas se encontram.

Por exemplo:

```
PROGRAM MODULO1 ;
   FUNCTION SOMA (X,Y : INTEGER) : INTEGER ;
   BEGIN
        SOMA := X+Y
   END ;
BEGIN END.

PROGRAM MODULO2 ;
   FUNCTION SUB(X,Y : INTEGER) : INTEGER ;
   BEGIN
        SUB := X-Y
   END ;
BEGIN END.
```

As duas funções SOMA e SUB serão compiladas separadamente, sendo que cada uma utiliza dois parâmetros formais.O programa principal, que irá usá-las, deverá explicitar estes parâmetros na parte declarativa para que a associação dos parâme

tros reais seja correta, bem como o endereço de desvio.
Por exemplo:

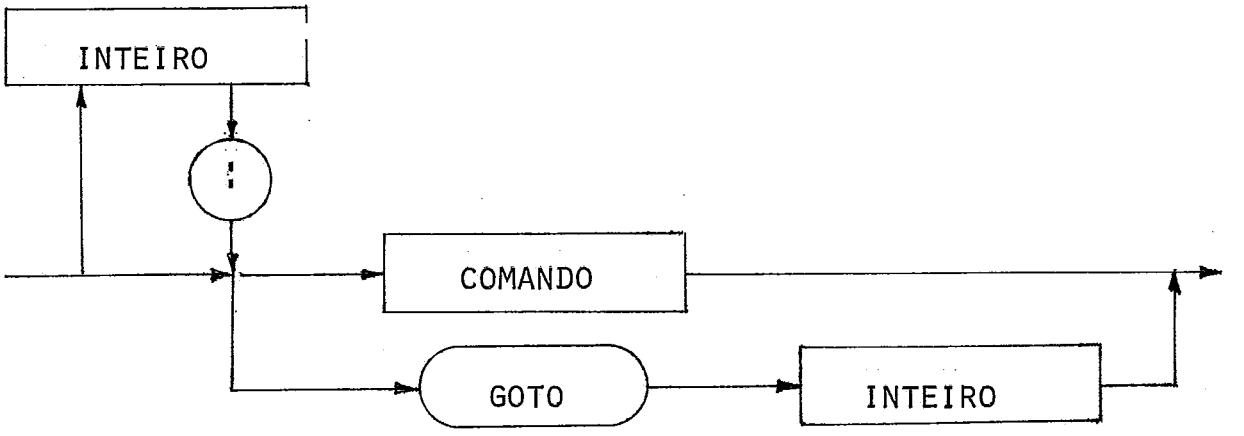
```
PROGRAM MODULAR ;
   VAR   VARG1 , VARG2 : INTEGER ;
   FUNCTION SOMA (X,Y : INTEGER) : INTEGER ;
   BEGIN CALL (SOMA) END ;
   FUNCTION SUB (X,Y : INTEGER) : INTEGER ;
   BEGIN CALL (SUB) END ;
BEGIN
   VARG1 := SOMA (5,6) ;
   VARG2 := SUB  (10,7) ;
END.
```

O programador pode usar esta técnica para executar programas maiores que a memória disponível no equipamento.

A primeira etapa para aplicação desta ferramenta é a divisão do programa em rotinas mutuamente exclusivas. Em segundo lugar, temos que programar todas as rotinas e reservar uma área para 'OVERLAY',dimensionada pela maior rotina. Cada rotina, compilamos separadamente, e o seu código armazenamos em memória secundária (fita, disco, etc...).

No programa principal declaramos todas as rotinas que iremos usar, bem como acrescentamos um sub-programa para efetuar a carga de cada uma delas na área de 'OVERLAY'.
Por exemplo:

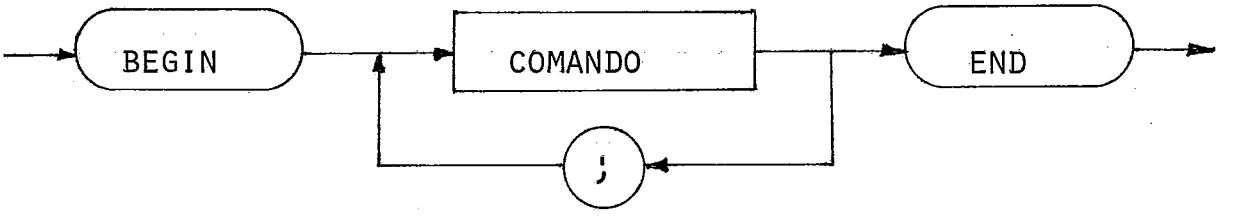
```
PROGRAM OVERLAY ;
   VAR   VARG1 , VARG2 : INTEGER ;
   PROCEDURE CARGA (DATASET : INTEGER) ;
   BEGIN LOADERTAPE (DATASET) END ;
   FUNCTION SOMA (X,Y : INTEGER) : INTEGER ;
   BEGIN    CALL (#2000) END ;
   FUNCTION SUB  (X,Y : INTEGER) : INTEGER ;
   BEGIN    CALL (#2000) END ;
BEGIN
   CARGA (1) ; (* CARREGOU A ROTINA SOMA*)
   VARG1 := SOMA (5,6) ;
   CARGA (2) ; (* CARREGOU A ROTINA SUB *)
   VARG2 := SUB (10,3) ;
END.
```



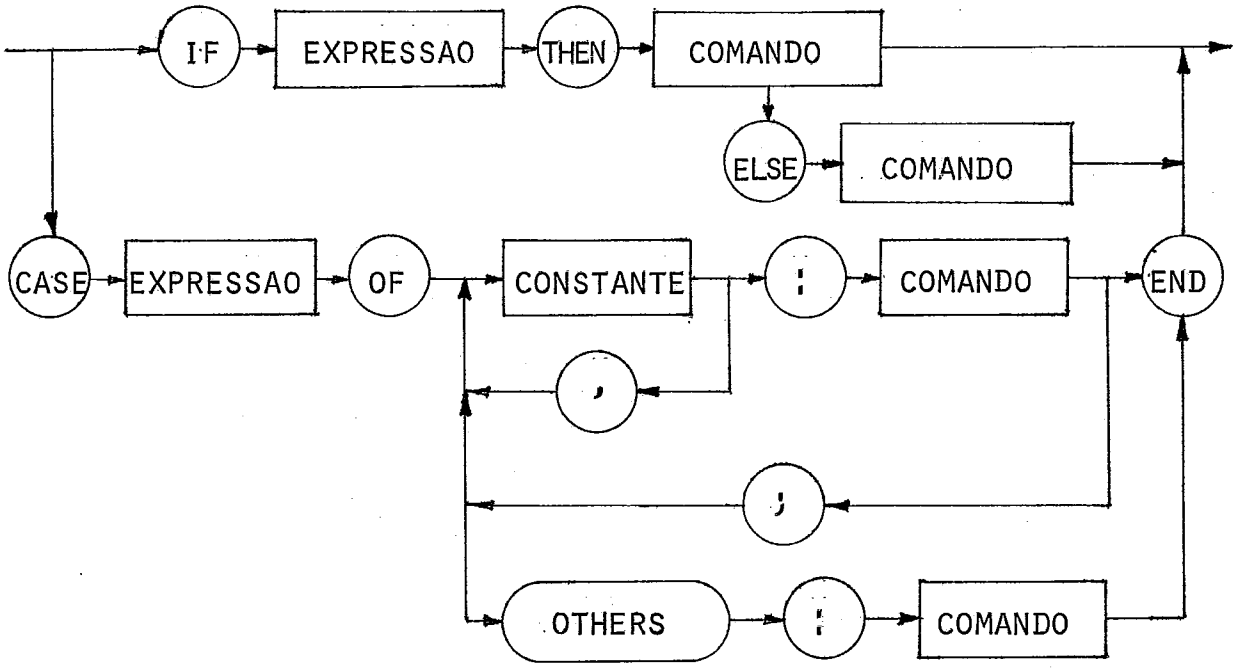
Os comandos podem ser rotulados com um 'label' numérico previamente declarado, para ser referenciado pelo comando GOTO . O escopo de um rótulo é o do procedimento que está declarado, não sendo possível desviar o programa para dentro de uma sub-rotina.

COMANDO ESTRUTURADO

Os comandos estruturados são construções compostas de outros comandos, que podem ser executados na forma sequencial ' (comando composto), condicionalmente (comando condicional), ou repetidamente (comando repetitivo).



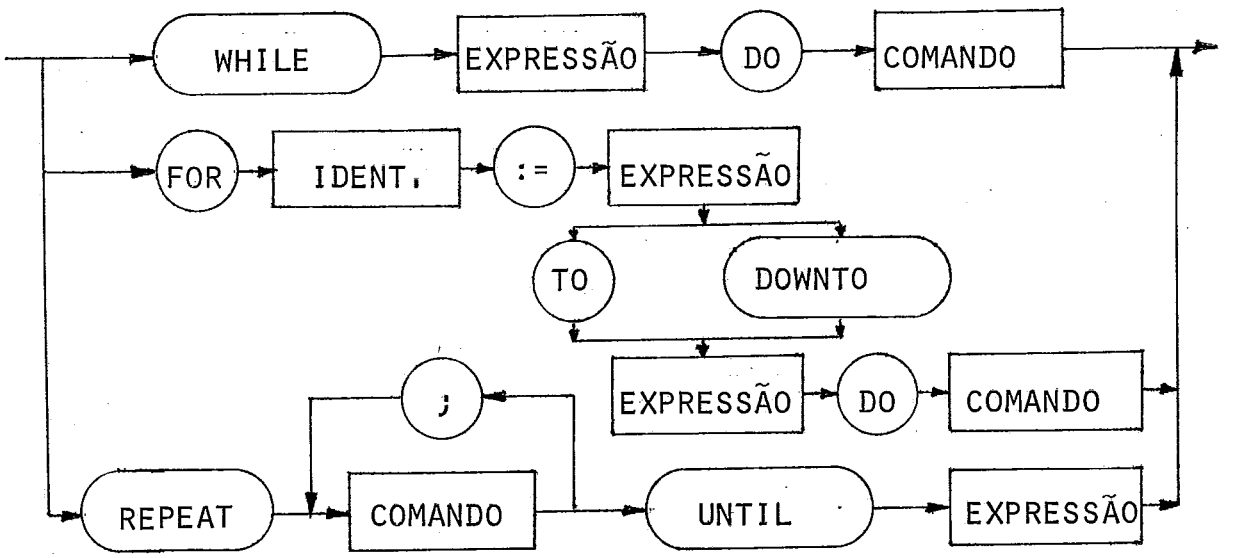
O comando composto especifica um conjunto de comandos que devem ser executados na mesma ordem na qual foram escritos. Os símbolos BEGIN e END atuam como delimitadores de início e fim do conjunto. O símbolo ponto-e-vírgula (;) é usado como separador de comandos e não para determinar o seu fim, isto é, o ponto-e-vírgula não faz parte do comando. Entre dois ponto-e-vírgulas seguidos é assumido um comando vazio, assim como entre um ponto-e-vírgula e um símbolo 'END' .



Os comandos condicionais (IF e CASE) selecionam um dos seus comandos para ser executado. A expressão do comando IF é avaliada como uma expressão lógica, sendo a condição testada pelo 'bit' de mais baixa ordem do resultado (Ø-falso e 1- verdadeiro). O comando CASE requer uma expressão aritmética para selecionar uma das listas rotuladas que coincida com o resultado da avaliação. A claúsula opcional 'OTHERS'[5] foi introduzida para resolver a seleção de resultados não especificados pelos rótulos anteriores.

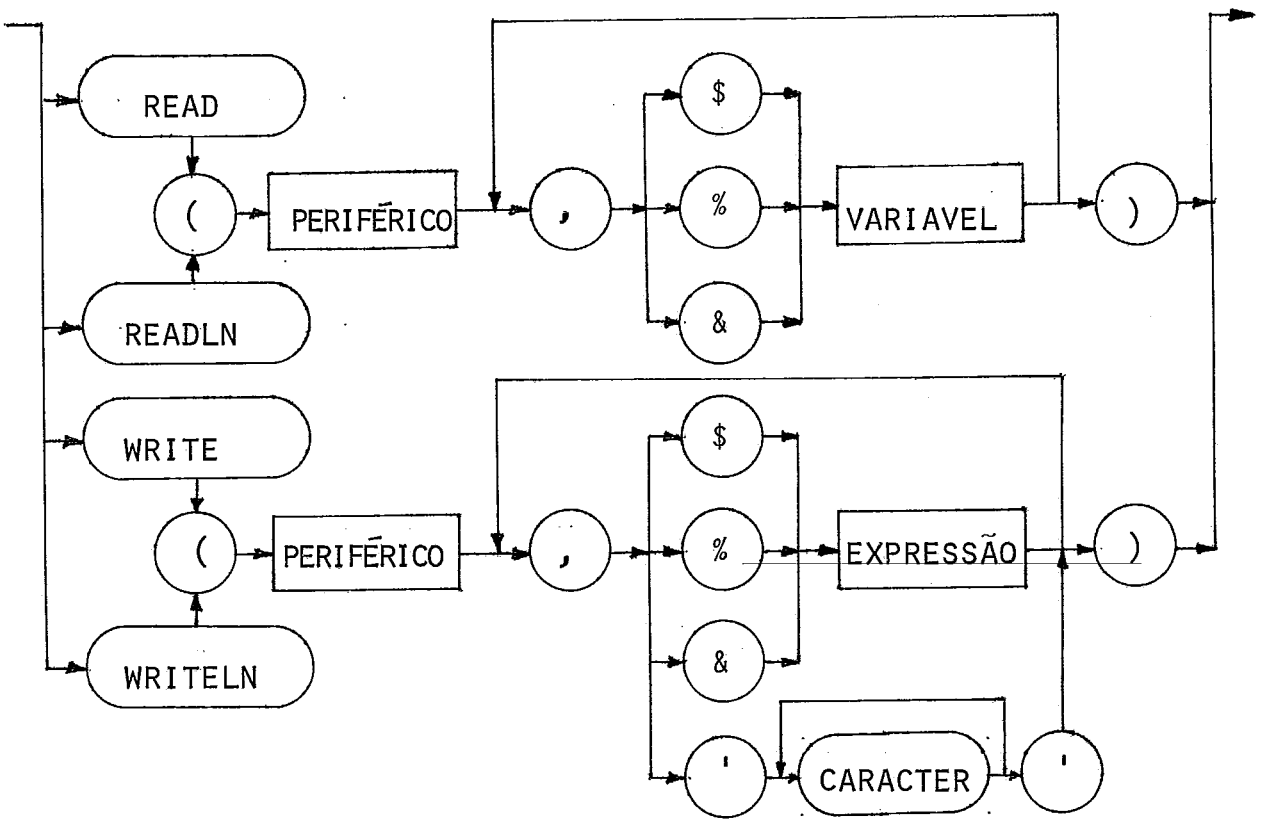
Por exemplo:

```
CASE  < exp >   OF
'A','B', #FF : < comando > ;
 1, 2, 3, 7  : < comando > ;
 158         : < comando > ;
'G'          : < comando > ;
 #A          : < comando > ;
OTHERS       : < comando > ;
END ;
```



Os comandos repetitivos executam um comando determinado número de vezes. No WHILE a expressão de controle é avaliada logicamente, antes da execução do comando precedido pelo símbolo DO , até que a condição verdadeira seja satisfeita. O comando REPEAT é executado pelo menos uma vez, e será repetido até que a expressão atinja uma condição falsa. O comando FOR controla o número de repetições, do comando, pelo incremento ou decremento de sua variável de controle, até que esta atinja o valor limite dado pela expressão precedida pelo TO ou DOWNTO.

II.1.8 - ENTRADA / SAÍDA



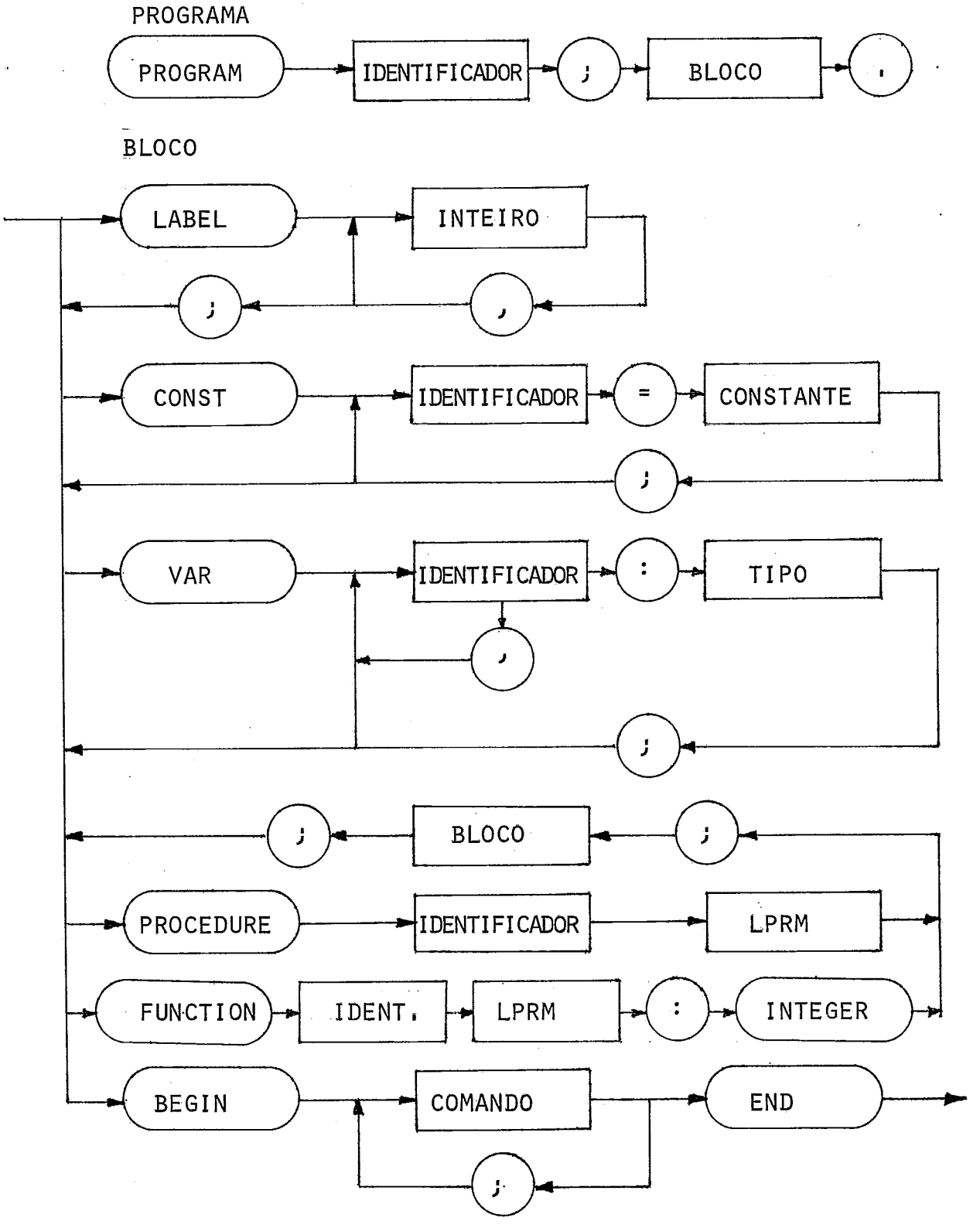
O procedimento de leituta (comando READ ou READLN) requer como parâmetros, o periférico e as variáveis a serem lidas. Cada variável deve ser precedida por um símbolo de formato de leitura, isto é, o $ para valores decimais, o % para formato hexadecimal e o símbolo & para caracter.

O procedimento de saída (comando WRITE ou WRITELN) admite mais uma modalidade de formato, que são as mensagens representadas por uma cadeia de caracteres entre aspas.

Tanto as rotinas READLN como WRITELN assumem automaticamente um novo 'buffer' após a leitura ou escrita de suas variáveis.

II.1.9 - PROGRAMA C-PASCAL

O programa C-PASCAL é dividido em cabeçalho, que representa a identificação do programa, e o corpo (bloco) que contêm as declarações e as ações.



Os rótulos são declarados no início de cada bloco, como uma lista de números inteiros, precedida pela palavra reservada LABEL. Os rótulos têm como escopo somente o bloco em que foram declarados, não sendo permitido o uso rótulo global.

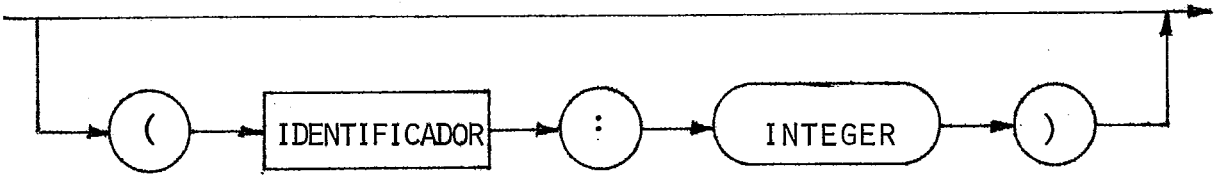
As constantes são declaradas em uma lista separada por ponto-e-vígula, e precedida da palavra chave CONST.
Por exemplo:

```
LABEL     1, 5, 7, 9 ;
CONST     CINCO = 5 ;
          CHARA = 'A' ;
          BRANCO = ' ' ;
```

As variáveis são declaradas no início do bloco, e seu escopo é exatamente o início e o fim do próprio bloco.
Por exemplo:

```
VAR       X1, X2, X3 : INTEGER ;
 ou
VAR       X1 : INTEGER ;
          X2 : INTEGER ;
          X3 : INTEGER ;
```

LISTA DE PARAMETROS (LPRM)



Os parâmetros no C-PASCAL, são todos variáveis simples, e passados somente por valor.

II.2 - MÁQUINA VIRTUAL C-PASCAL

A compilação no suporte C-PASCAL é dirigida para uma hipotética máquina de pilha, que está mais próxima da arquitetura dos micro-computadores.

A máquina virtual é formada por um grupo de registradores, um conjunto de instruções, uma estrutura de pilha para os dados e uma área de programa.

II.2.1 - REGISTRADORES

Os registradores da máquina virtual são todos de 16-bits, e utilizados como controle, isto é, não é permitido o acesso pelo programador.

Os registradores são manipulados pelas instruções e organizados como duas palavras de 8-bits, que corresponde a parte alta (high-h) e a parte baixa (low-l) do registro.

CONTADOR DE PROGRAMA (PC ou PCh e PCℓ)

O registrador PC é usado para orientar o fluxo lógico da execução do código intermediário.

PONTEIRO DA PILHA (SP ou SPh e SPℓ)

O registrador SP é usado para endereçar o topo da pilha.

REGISTRO DE BASE (BR ou BRh e BRℓ)

O registro de BR contém a base de endereçamento estático dos dados.

REGISTRO TEMPORÁRIO (TR ou TRh e TRℓ)

O registro TR é usado como variável auxiliar na execução de uma instrução.

REGISTRO DO OPERANDO (OR ou ORh e ORℓ)

O registro OR contém o operando da instrução que está no registro de instrução.

REGISTRO DE INSTRUÇÃO (IR ou IRh e IRℓ)

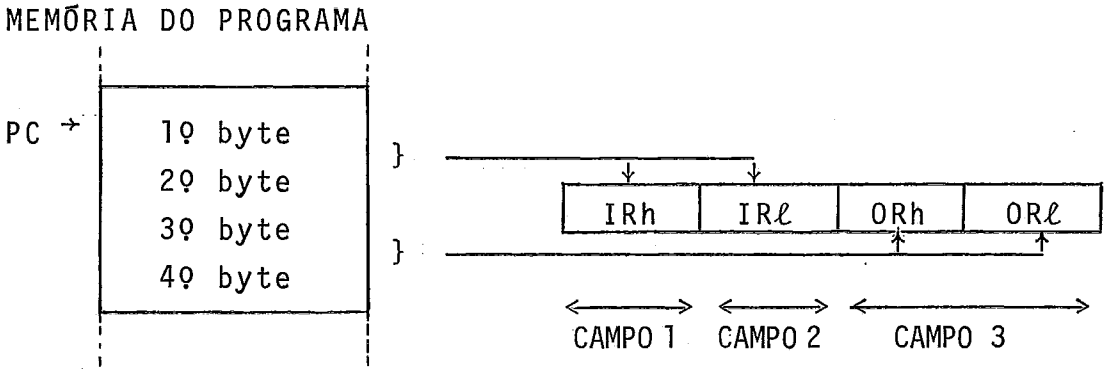
O registro IR contém a instrução a ser executada.

II.2.2 - CONJUNTO DE INSTRUÇÕES

As instruções da máquina virtual são de tamanho fixo (32-bits), que corresponde a instrução (IR-16bits) e seu operando (OR-16bits).

As instruções são armazenadas, temporariamente, em IR e OR antes de serem executadas.

Por exemplo:



CÓDIGO INTERMEDIÁRIO (32-bits)

A instrução pode ser dividida para análise em três campos, sendo dois de 8bits e o outro de 16bits.

- CAMPO 1 - corresponde ao IRh que contém o código da instrução.
- CAMPO 2 - corresponde ao IRℓ que pode conter o nível estático, uma subdivisão do código ou uma condição 'booleana'.
- CAMPO 3 - corresponde ao OR que contém o parâmetro' da instrução (valor, endereço, deslocamento).

As instruções disponíveis na máquina virtual são:

CAMPO 1	CAMPO 2	CAMPO 3	DESCRIÇÃO
ØØ LDI	-	VALOR	'LOAD' IMEDIATO DO VALOR
Ø1 LOD	NÍVEL	DESLOCAMENTO	'LOAD' VARIÁVEL SIMPLES
11 LODX	NÍVEL	DESLOCAMENTO	'LOAD' VARIÁVEL INDEXADA
Ø2 LDM	-	-	'LOAD' DIRETO DE MEMÓRIA
Ø3 STO	NÍVEL	DESLOCAMENTO	'STORE' VARIÁVEL SIMPLES
13 STOX	NÍVEL	DESLOCAMENTO	'STORE' VARIÁVEL INDEXADA
Ø4 SDM	-	-	'STORE' DIRETO EM MEMÓRIA
Ø5 CAL	NÍVEL	ENDEREÇO	CHAMADA DE SUB-PROGRAMA
15 CALX	-	-	CHAMADA DE ROTINA EXTERNA
Ø6 RET	PARÂMETROS	-	RETORNO DE SUB-PROGRAMA
Ø7 JMP	-	ENDEREÇO	DESVIO INCONDICIONAL
17 JMP	-	ENDEREÇO	DESVIO INCONDICIONAL
ØB JPC	CONDIÇÃO	ENDEREÇO	DESVIO CONDICIONAL (Ø ou 1)
Ø9 OPE	TIPO	-	OPERAÇÃO - o campo 2 especifica o operador
ØA RES	PERIFÉRICO	ROTINA	ENTRADA/SAÍDA - o campo 3 especifica a rotina de I/O.
ØB DPI	-	VALOR	DESLOCAMENTO NO SP
ØC OPI	TIPO	-	OPERAÇÃO NA PILHA - o campo 2 especifica a operação

II.2.3 - DESCRIÇÃO DAS INSTRUÇÕES

Para descrver as ações das instruções da máquina virtual, iremos utilizar a seguinte notação:

SIMBOLOGIA		SIGNIFICADO
←	:	símbolo usado para atribuição.
(reg)	:	o conteúdo da posição de memória (8-bits) apontada pelo enderaçamento contido em reg.
{reg}	:	o conteúdo das posições de memória apontadas pelos endereços contidos em reg e reg+1, correspondendo a uma palavra de 16-bits.
regh e regℓ	:	parte alta do registro (regh) e parte baixa do registro (regℓ).
P↑{X}	:	POP - se X for uma posição de memória dada por um endereço entre parêntesis; (< endereço >) ou a parte baixa de um registrador (regℓ) então o desempilhamento será parcial, isto é, somente a parte baixa do elemento apontado por SP (topo) será armazenado na memória ou na parte baixa do registrador. - se X for uma região de memória dada por um endereço entre chaves {endereço} ou um registrador, então o desempilhamento será total, isto é, o elemento do topo será armazenado em duas posições consecutivas da memória ou no registrador (16-bits).
P↓{X}	:	PUSH - se X for uma posição de memória dada por um endereço entre parêntesis, (< endereço >), ou a parte baixa de um registrador (regℓ) então o empilhamento será parcial, isto é, so

SIMBOLOGIA		SIGNIFICADO (CONT.)
		mente a parte baixa do topo receberá o conteúdo de memória ou da parte baixa do registrador.
		- se X for uma região de memória dada por um endereço entre chaves {<endereço>} , ou um registrador (16-bits) então o empilhamento será total, isto é, o elemento do topo receberá os conteúdos de duas memórias consecutivas ou do registrador (16-bits)
|... | X	:	repetir X vezes a execução das ações entre as barras.

DESCRIÇÃO DAS INSTRUÇÕES

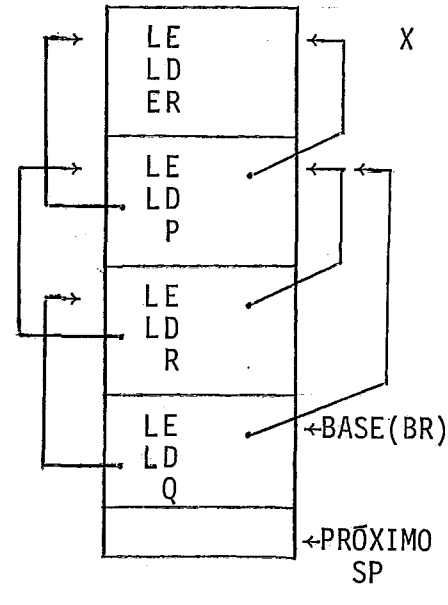
INSTRUÇÃO	IR	OR	AÇÃO
LDI VAL	ØØ -	VALh VALℓ	P↓{OR}
LOD NIV END	Ø1 NIV	ENDh ENDℓ	TR ← BR |TR←{TR}|IRℓ P↓{{TR+OR}}
LODX NIV END	11 NIV	ENDh ENDℓ	TR ← BR |TR←{TR}|IRℓ OR←TR+OR P↑{TR} P↓{{TR+OR}}
LDM	Ø2 -	- -	P↑{TR} P↓{(TR)}
STO NIV END	Ø3 NIV	ENDh ENDℓ	TR ← BR |TR←{TR}|IRℓ P↑{{TR+OR}}
STOX NIV END	13 NIV	ENDh ENDℓ	TR ← BR |TR←{TR}|IRℓ OR←TR+OR P↑{TR} P↑{{TR+OR}}
STM	Ø4 -	- -	P↑{TR} P↑{(TR)}
CAL NIV END	Ø5 NIV	ENDh ENDℓ	TR ← BR |TR←{TR}|IRℓ P↓{TR} TR ← BR BR ← SP P↓{TR} P↓{PC} PC ← OR
CALX	15 -	- -	P↑{OR} P↓{PC} PC ← OR

INSTRUÇÃO	IR	OR	AÇÃO
RET NPRM	Ø6 NPRM	- -	TR ← BR BR←{TR-2} SP←TR+IRℓ PC←{TR-4}
JMP END	Ø7 -	ENDh	ENDℓ PC ← OR
JPC STA END	Ø8 STA	ENDh	ENDℓ P↑{TR} if TRℓ=IRℓ then PC ← OR
OPE TIP	Ø9 TIP	- -	P↑{TR} P↑{OR} case IRℓ of ØØ:P↓{OR} P↓{-TR} Ø1:P↓{OR} P↓{NOT TR} Ø2:P↓{TR*OR} Ø3:P↓{TR DIV OR} Ø4:P↓{TR MOD OR} Ø5:P↓{TR SHL OR} Ø6:P↓{TR SHR OR} Ø7:P↓{TR AND OR} Ø8:P↓{TR=OR} Ø9:P↓{TR<>OR} ØA:P↓{TR<OR} ØB:P↓{TR>=OR} ØC:P↓{TR>OR} ØD:P↓{TR<=OR} ØE:P↓{TR or OR} ØF:P↓{TR-OR} 1Ø:P↓{TR+OR}
RES PER ROT	ØA PER	ROTh	ROTℓ P↓{PC} PC←OR+IRℓ
DPI VAL	ØB -	VALh	VALℓ SP←SP+OR

INSTRUÇÃO	IR	OR	AÇÃO
OPI TIP	ØC TIP	- -	case IRℓ of ØØ: {SP}←{SP}-1 Ø1: P↓{{SP}} Ø2: {SP}←{SP}+1 Ø3: P↓{{SP+2}} Ø4: P↑{TR} P↑{OR} P↓{TR} P↓{OR} Ø5: SP←SP-2 Ø6: SP←SP+2 Ø7: P↑{TR} P↑{OR} if {SP}>TR then erro if {SP}<OR then erro

Por exemplo: Para o programa a seguir, temos, durante a execução, a configuração ao lado da pilha para a correspondente sequência de chamadas:
X → P → R → Q (a pilha cresce para baixo)

```
PROGRAM X ;
   PROCEDURE P ;
        PROCEDURE Q ;
        BEGIN ..... END ;
        PROCEDURE R ;
        BEGIN...Q ;...END ;
   BEGIN...R ;...END ;
BEGIN...P ;...END ;
```



II.3 - ESTRUTURA DO COMPILADOR

A sintaxe da linguagem C-PASCAL tem uma estrutura típica das linguagens orientadas para facilitar uma análise sintática de cima para baixo, e compilação em um passo. Entende-se' por compilação em 1 passo a transformação em apenas uma passagem no texto do programa original para a linguagem alvo. No nosso caso, a linguagem alvo é a definida pela máquina virtual apresentada na seção II.2. Como esta máquina não está implementada em nenhum 'hardware' tem-se duas alternativas para "executá-la": a interpretação ou a emulação[11] no micro-computador.

Tendo sido o projeto desenvolvido originalmente na linguagem PASCAL, no PDP 11/70[5] , que possui mecanismo de recursão e como o próprio C-PASCAL dispõe deste mecanismo, foi possível escrever o compilador na própria linguagem. Entre os métodos 'top-down' optou-se pela descida recursiva[12], por se tratar do método no qual as características do C-PASCAL são mais eficientemente utilizadas.

O esquema geral de compilação é o clássico, onde a análise léxica é feita por um procedimento que é chamado a cada vez que se deseja obter um novo símbolo ("token"). Durante a análise sintática também é feita a análise semântica e geração de código. No caso de se detectar um erro de sintaxe é chamada uma rotina que analisa o contexto local[13] e o modifica de modo a permitir que o analisador sintático possa prosseguir a análise do texto, mas sem gerar novos códigos.

O compilador emite a listagem do programa fonte e ao lado de cada linha indica o endereço do primeiro código gerado para os comandos da linha. Além disso, no caso de erros, são indicados os erros encontrados e os eventuais símbolos abandonados para ser possível o prosseguimento da análise. Ao final da compilação é possível obter a listagem simbólica dos códigos gerados pelo compilador.

II.3.1 - ANÁLISE LÉXICA

Para desempenhar suas funções básicas de particionar' o texto fonte em unidades atômicas e identificá-las, devolvendo ao analisador sintático o 'token', o analisador léxico utiliza duas tabelas que serão descritas resumidamente a seguir.

Como informação preliminar, o analisador léxico classifica cada caracter em uma das três seguintes classes: classe (1) para as letras, classe (2) para os dígitos decimais, e classe (3) para os demais símbolos.

Com esta classificação, o analisador passa a trabalhar sobre os três grupos, que representam: grupo (1) dos identificadores e palavras reservadas, grupo (2) dos números e grupo (3) dos símbolos simples e duplos. Cada grupo usa da melhor' forma a tabela de caracteres da qual coletam informações com significados distintos.

i-) TABELA DE CARACTERES

Esta tabela contém uma informação para cada caracter do conjunto ASCII. Ela é mantida diretamente no código objeto e carregada na memória de programa, quando da execução do compilador. Cada elemento da tabela é representado por uma palavra de 8-bits (1-byte). As informações da tabela são acessadas através do vetor MEM [< exp >] , indexado pelo valor do código ASCII do caracter, cujo conteúdo pode significar: símbolo ("tokens") para os símbolos simples, ou índice de acesso a tabela de palavras reservados para as letras.

Por exemplo:

a-) MEM [BASE1 + ')'] = 51
onde 51 é o lexema ("token") do símbolo simples ')' .

b-) MEM [BASE1 + 'C'] = 2Ø
onde 2Ø é o índice da lista de palavras começadas por 'C', na tabela de palavras reservadas.

c-) MEM [BASE1 + '1'] = 32
onde 32 é o lexema do número inteiro.

A constante BASE1 representa o endereço absoluto de memória onde está armazenada a Tabela de Caracteres , sendo o código ASCII usado como um deslocamento.

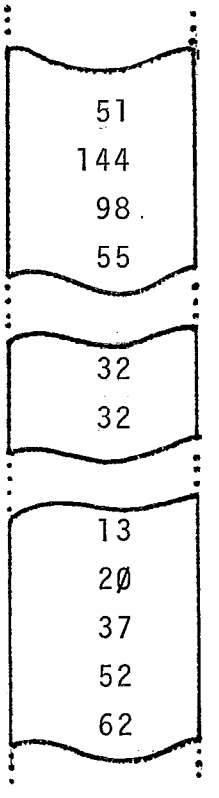


TABELA DE CARACTERES

ii-) TABELA DE PALAVRAS RESERVADAS

A identificação das palavras reservadas é feita por uma estrutura em árvore, com o primeiro nível representado pelas letras.

O primeiro nível é formado pelo sub-vetor de letras da Tabela de Caracteres. Este vetor contém ponteiros para cada lista de palavras reservadas.

A cabeça de cada lista contém o número de elementos existentes nela. Cada elemento é formado pelo código do 'token', número de caracteres da palavra (tamanho) e a cadeia dos caracteres do identificador menos o primeiro.

Por exemplo:

- acesso a lista 'B': MEM [BASE1 + 'B'] = 13
  com o código ASCII de 'B', obtemos na tabela de caracteres o endereço (13) da lista de palavras começadas por esta letra.
- tamanho da lista : MEM [BASE2 + 13] = 1
  o primeiro elemento (cabeça da lista) nos informa que só existe uma palavra reservada começada por 'B' .
- pesquisa na lista : MEM [BASE2+13+2] = 5
  a maior palavra da lista 'B' tem tamanho 5. Se o tamanho for igual ao da palavra procurada, fazemos uma comparação dos quatro últimos caracteres, pois o primeiro já foi testado no acesso a lista. Se o tamanho for diferente temos dois casos:
  - tamanho > 5 : não existe palavra reservada com este tamanho na lista 'B'.
  - tamanho < 5 : pesquisar a próxima palavra se existir, no caso da lista 'B' a pesquisa será sem sucesso , e indicará como resultado o lexema de identificador.

13	1
14	71
15	5
16	E
17	G
18	I
19	N
20	3
21	66
22	5
23	O
24	N
25	S
26	T
27	163
28	4
29	A
30	S
31	E
32	177
33	4
34	A
35	L
36	L

A tabela de palavras reservadas faz parte do código ' objeto, e é carregada na memória de programa no instante da execução do compilador. Este processo dispensa a 'inicialização' da tabela pelo programa compilador. Cada componente desta tabela ocupa uma posição de memória (palavra de 8-bits), e é acessada através do vetor MEM [<exp>].

As buscas são feitas primeiramente comparando-se o tamanho da palavra, para depois testar cada caracter da cadeia. Além disto, as listas são ordenadas de forma decrescente pelo tamanho, a fim de que a busca sem sucesso dos identificadores ' seja interrompida antes do final da lista.

ESQUEMA GERAL DO ANALISADOR LÉXICO

```
PROCEDURE GTOKEN ;
    PROCEDURE SCAN ;
    BEGIN    - LER ENTRADA
             - ELIMINA CARACTERES DE CONTROLE
             - CLASSIFICA OS CARACTERES EM:
                 1 - ALFABÉTICOS
                 2 - NUMÉRICOS
                 3 - SÍMBOLOS ESPECIAIS,SIMPLES
    END;
    BEGIN    - CHAMA A ROTINA SCAN
             - ELIMINA BRANCOS
             - CLASSIFICA AS PALAVRAS RESERVA/IDENTIFICADOR
             - IDENTIFICA E CONCATENA AS CONSTANTES NUMÉRICAS
             - CLASSIFICA OS SÍMBOLOS EM:
                 - SÍMBOLOS SIMPLES
                 - SÍMBOLOS DUPLOS
                 - ELIMINA COMENTÁRIOS
                 - CONCATENA AS CADEIAS DE CARACTERES
    END ;
```

Os comentários são eliminados pelo analisador léxico, bem como os caracteres inválidos (não pertencentes ao alfabeto C-PASCAL) que são substituidos por brancos e emitidas mensagens de erro.

A constante BASE2, utilizada para acessar a Tabela de

Palavras Reservadas, representa o endereço absoluto de memória onde está armazenada a tabela. Os índices, obtidos na Tabela de caracteres, representam deslocamentos relativos a serem acrescentados a BASE2.

II.3.2 - ANÁLISE SINTÁTICA, SEMÂNTICA E GERAÇÃO DE CÓDIGO

A partir do diagrama de sintaxe da linguagem,descrito na seção II.1 ,escrevemos o analisador sintático que é constituido basicamente de um procedimento para cada elemento sintático, tal como: comando, expressão, bloco,etc...

Para cada comando da linguagem o analisador verifica a estrutura sintática por etapas, e a medida que partições são analisadas corretamente, são gerados códigos correspondentes na linguagem intermediária alvo. Na geração de código, os desvios para frente são tratados por uma lista de referências pendentes e resolvidos pelo procedimento 'FIXAR' quando atingimos o endereço especificado na cabeça da lista.

ESTRUTURA DO ANALISADOR SINTÁTICO

```
PROCEDURE INSERIR ;
     BEGIN armazena um identificador na tabela de símbolos END ;
PROCEDURE BUSCAR  ;
     BEGIN busca um identificador na tabela de símbolos END ;
PROCEDURE FIXAR   ;
     BEGIN resolve as referências, atuais e pendentes END ;
PROCEDURE BLOCO   ;
          PROCEDURE EXPRESSÃO ;
                    PROCEDURE FATOR ;
                    PROCEDURE TERMO ;
                    PROCEDURE EXPRESSAOSIMPLES ;
          PROCEDURE COMANDO ;
     BEGIN analisa as estruturas END ;
```

TABELA DE SÍMBOLOS

A tabela de símbolos é formada pelos seguintes vetores:

a-) VETOR TIPO : Neste vetor temos o tipo do identificador , que pode ser: constante, variável simples, variável indexada, variável função, identificador de função e identificador de procedimento. Cada elemento deste vetor ocupa um 'byte', e é manipulado pela estrutura MEM [<exp>].

b-) VETOR NÍVEL: Este vetor fornece o nível estático da variável ou o nível do procedimento, sua informação ocupa um 'byte' e é armazenada diretamente na memória.

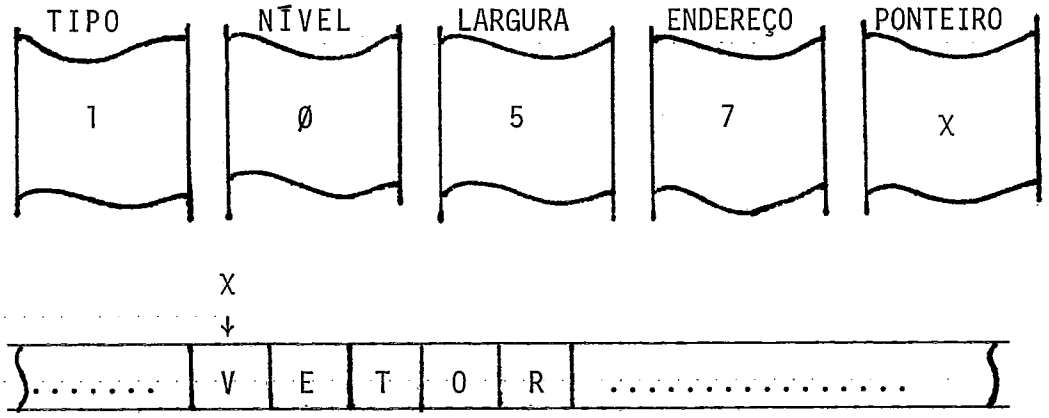
c-) VETOR PONTEIRO: Consiste de uma estrutura de inteiros, ' onde são armazenados os ponteiros para o 'poço' das cadeias de caracteres dos identificadores.

d-) VETOR LARGURA: O número de caracteres de cada identificador é armazenado neste vetor, e como cada componente ocupa apenas um 'byte' sua organização é feita diretamente' na memória com a ajuda da estrutura MEM [<exp>].

e-) VETOR ENDEREÇO: Consiste de um vetor de inteiros, cuja a informação pode ser: o deslocamento relativo à base de dados de uma variável, ou o endereço no código intermediário de uma sub-programa (PROCEDURE ou FUNCTION).

f-) POÇO DE CARACTERES: Esta é uma estrutura de armazenamento sequencial das cadeias de todos os identificadores do programa.

Por exemplo:



POÇO DE CARACTERES

O acesso é feito com a ajuda do VETOR PONTEIRO e do VETOR LARGURA. O exemplo acima mostra as informações armazenadas para um identificador de uma varável simples nomeada de ' 'VETOR'.

A classificação e codificação dos 'tokens' foram estruturados segundo agrupamentos homogêneos (operadores relacionais, declarações, blocagem, etc...), e dentro do grupo na ordem mais adequada que auxiliasse a geração de código.
Por exemplo:

'TOKEN'				CÓDIGO INTERMEDIÁRIO
OPERADORES DE MULTIPLICAÇÃO				INSTRUÇÕES DE MULTIPLICAÇÃO
*	-(9Ø)	ou 144	→	(9 , 2)
DIV	-(91)	ou 145	→	(9 , 3)
MOD	-(92)	ou 146	→	(9 , 4)
SHL	-(93)	ou 147	→	(9 , 5)
SHR	-(94)	ou 148	→	(9 , 6)
AND	-(95)	ou 149	→	(9 , 7)

```
PROCEDURE TERMO ;
VAR   AUX : INTEGER ;
BEGIN
      FATOR ;
      WHILE  (TOKEN AND #FØ) = #9Ø   (**DIV MOD SHL SHR AND*)
      DO BEGIN
            AUX := TOKEN - 142 ;
            FATOR ;
            CODIGO ( 9 , AUX )
      END ;
END ;
```

A classificação adequada permitiu a compactação do programa, como podemos observar na PROCEDURE TERMO descrita acima.

II.3.3 - RECUPERAÇÃO DE ERRO

O método utilizado para recuperar erros sintáticos foi elaborado com base nas estatísticas dos erros que mais ocorrem entre programadores da linguagem PASCAL [14].

Entre os erros mais prováveis apontados pelas estatísticas, nós procuramos tratar os seguintes casos:

a-) erro de digitação ou perfuração, que corresponde as trocas no ato de teclar o programa. Esta classe representa 5% dos erros observados nas estatísticas com programas em PASCAL ;

b-) erro de troca, que corresponde a troca de um símbolo por outro decorrente do costume em outras linguagens (atribuição FORTRAN '=' e PASCAL ':='), ou por distração do programador ('(' por '['). Este grupo de erros é responsável por 39% das ocorrências nas estatísticas utilizadas;

c-) erro por omissão, são aqueles ocasionados por esquecimento de um ponto-e-vírgula, de um fecha parêntesis (')'), da omissão de um END no final de um comando composto, etc... Esta classe é a mais frequente, sendo sua participação de 41% nas estatísticas.

Dentro desta metodologia, estaremos cobrindo aproximadamente 85% dos erros mais prováveis em programação PASCAL, e que são possíveis de serem detectados numa análise de cima para baixo pelo método da descida recursiva.

O procedimento do recuperador sintático é iniciado quando observamos um símbolo não admissível na entrada, e a partir deste ponto o analisador transfere o controle do programa para o recuperador. A ação tomada pelo recuperador está basicamente fundamentada por duas tabelas, organizadas em forma de listas, que são analisadas sequencialmente até que se encontre uma construção aceitável no texto do programa fonte. Todas as decisões de avançar, aceitar ou trocar a entrada são documentadas na listagem de compilação para que o programador tenha uma posição real daquilo que o recuperador executou para contornar um determinado erro.

A lista que primeiramente é analisada contém todos os prováveis símbolos possíveis de serem trocados (erro de digitação ou troca), para um determinado erro (símbolo esperado). No caso de uma busca sem sucesso com o símbolo da entrada, passa

mos a pesquisar uma segunda lista, que contêm os mais prováveis símbolos que podem seguir a entrada esperada ('follows' que contornam os erros por omissão)[13] .

A ação tomada pelo recuperador quando encontra um símbolo da entrada pertencente a primeira lista é :

a-) fornecer uma mensagem de erro, especificando que houve uma troca e que o símbolo em análise foi ignorado, e assumido o correto para continuação da análise sintática;

b-) restaura todas as variáveis do compilador com os atributos do símbolo correto.

A segunda lista só é analisada quando esgotamos todas as possibilidades da primeira. Esta é formada pelos mais prováveis símbolos continuadores da frase em análise ('follows'), acrescida, em alguns casos, dos 'tokens' 'END' e ';' que em última estância funcionarão como símbolos de parada no esquema de 'panic-mode'. A ação tomada com a identificação de um símbolo da entrada pertencente a esta lista é:

a-) fornecer uma mensagem de erro, especificando o símbolo que foi omitido na entrada;

b-) introduzir o símbolo esperado e restaurar todas as variáveis com os atributos corretos;

c-) atrasar a entrada, desligando a rotina que analisa os 'tokens',por uma chamada;

d-) continuar a análise sintática pelo novo símbolo introduzido.

O objetivo do método empregado é desprezar o menor número de símbolos ('tokens') do programa fonte, durante a recuperação de um erro. Para os 'tokens' desprezados fornecemos uma mensagem, assinalando-os com uma seta (↑), informando que foram abandonados durante a recuperação do erro. Com estas mensagens o usuário tem condições de verificar a confiabilidade do restante da análise sintática.

O recuperador de erros é composto por três rotinas:

a-) ROTINA ABORTA: Esta rotina é usada para erros irrecuperáveis, tais como: número de erros acima do limite, memória esgotada, transbordo na tabela de símbolos, final inesperado, etc...

b-) ROTINA MSGERRO: Esta rotina imprime todas as mensagens de erros, assinalando o símbolo em questão com uma seta (↑). As mensagens são solicitadas pela rotina RECERRO, ou diretamente pelo analisador sintático quando se trata de simples advertências.

c-) ROTINA RECERRO: Este procedimento executa a recuperação do programa com o auxílio das listas primárias e secundárias. A rotina recebe dois parâmetros do analisador sintático, que representam o símbolo esperado e o número do erro. Com a entrada de um símbolo diferente do esperado, o recuperador calcula, a partir do número do erro, os endereços das listas primárias e secundárias. Para cada símbolo da entrada desprezado, a rotina RECERRO fornece uma mensagem correspondente através da rotina MSGERRO. Durante a análise do contexto, pela rotina RECERRO, podemos observar seis casos:

1- encontrar um símbolo na entrada igual ao esperado;
2- encontrar um símbolo na entrada pertencente a lista primária;
3- encontrar um símbolo na entrada pertencente a lista secundária;
4- encontrar na entrada um símbolo de parada 'END' ou ';' ;
5- o programa é abortado por excesso de erro;
6- o programa é abortado por final não esperado.

As listas são testadas sequencialmente até ser encontrada uma marca de fim. O endereço inicial é calculado a partir do número do erro, e o mesmo endereço pode ser calculado a partir de quatro erros diferentes. Portanto, podemos ter quatro mensagens diferentes e apenas duas listas (primária e secundária) .

Por exemplo:

```
ERRO 12 : (:=) esperado no comando de atribuição
FRASE   : ....   := <exp>
ERRO 13 : (:=) esperado na inicialização da variável'
                de control do 'FOR'
FRASE   : 'FOR' ID := <exp>  TO/DOWNTO
```

f (12|13|14|15) LPrimária = L5; LSecundária = L1

=	:	fim	MEM	NOT	INTEIRO	ID	'	-	+	(	;	END	fim

↑ L5 ↑ L1

A lista L5 nós temos as seguintes trocas:

:= por = (vícios de outras linguagens)

:= por =: (erro de perfuração ou digitação)

:= por : (omissão do símbolo =)

A lista L1 representa os prováveis inícios de uma expressão, e mais os símbolos 'END' e ';' para serem usados num esquema de 'panic-mode' .

CAPÍTULO III

INTERPRETADOR C-PASCAL

III.1 - INTRODUÇÃO

O projeto do Suporte C-PASCAL não foi dirigido para o modelo compilador/interpretador puro, como a maioria dos sistemas existentes para micro-computador. O código intermediário serviu para reduzir o processo de compilação, assim como tornar o suporte mais facilmente adaptável em outros equipamentos.

O Suporte C-PASCAL admite duas opções de execução do código intermediário. O primeiro corresponde à máquina virtual constituida por simulação[11] (interpretação por 'software'), onde são representados os algarítmos e as estruturas de dados por um programa escrito em alguma linguagem existente no micro-computador hospedeiro. O segundo método consiste na implementação da máquina virtual usando um tradutor do código intermediário e um suporte de rotinas, programado em assembler, para simular o conjunto de instruções e as estruturas de dados.

A Simulação é a técnica que decodifica e executa cada código intermediário. Este processo foi implementado pelo programa INTERPRETADOR, descrito neste capítulo.

A Emulação, análoga aos sistemas simulados por micro-programação[11], corresponde ao método que decodifica e traduz cada código intermediário para código executável pelo micro-computador hospedeiro. Este método, representado pelo programa TRADUTOR, será descrito no próximo capítulo.

II.2 - ESTRUTURA DO INTERPRETADOR

O programa Interpretador consiste basicamente em dois blocos:

a-) PROGRAMA PRINCIPAL - Este bloco funciona como 'interface' homem-máquina, e como monitor da máquina virtual durante a execução de programas na forma intermediária;

b-) SIMULADOR - Este bloco, formado por um conjunto ' de sub-programas, executa as instruções e simula as estruturas' de dados da máquina virtual.

Cada chamada do simulador feita pelo programa princi pal, representa um ciclo de instrução da máquina virtual, que é composto das seguintes ações:

a- buscar o código intermediário apontado pelo conta dor de programa (PC) ;
b- testar a condição de rastreamento;
c- atualizar o vetor de rastreamento;
d- incrementar o contador de programa (PC) ;
e- decodificar a instrução intermediária;
f- executar a instrução intermediária;
g- testar o vetor de interrupção;

Os registradores da máquina virtual são simulados, no Interpretador, por variáveis globais tais como: PC- contador ' de programa; BR- registrador de base; SP- ponteiro da pilha; ' COD, NIV, IDX- registrador de instrução; e VAL- registrador de operando.

A estrutura de pilha é simulada por um vetor, com ta manho máximo definido pela constante MAXSTACK .

O Interpretador de código intermediário foi incorpora do ao Suporte C-PASCAL, para ser usado como ferramenta de depu ração dos erros de lógica no desenvolvimento de projetos no micro-computador.

Os recursos iterativos de depuração de programas mais usuais (execução passo a passo, interrupção,rastreamento,etc..), foram introduzidos no interpretador. A lista dos comandos de de puração e suas funções, estão descritas na seção III.3 deste capítulo.

A estrutura do Interpretador, programado em C-PASCAL, apresenta os seguintes sub-programas:

```
PROGRAM INTERPRETADOR

    FUNCTION    BASE  : INTEGER;
    PROCEDURE   INICIA ;
    PROCEDURE   MNEMÔNICO   (ADR:INTEGER);
    PROCEDURE   EXECUTA ;
BEGIN PROGRAMA PRINCIPAL   END..
```

A função BASE, calcula a base de endereçamento para um determinado nível estático, fornecido pela variável global NIV. Esta função é usada como primitiva da máquina virtual, para executar instruções intermediárias tais como: LOD,STO e CAL.

O procedimento INICIA restaura todas as variáveis globais com seus valores iniciais. Esta rotina é utilizada no início da interpretação, ou quando o usuário desejar reiniciar o programa após uma interrupção por ponto de quebra.

O procedimento MNEMÔNICO fornece a representação simbólica do código intermediário, apontado pelo endereço de memória passado como parâmetro (ADR). Esta rotina é usada para listar sequências de instruções no programa, ou para apresentar um rastreamento ('trace') do fluxo de execução.

O procedimento EXECUTA simula o processador da máquina intermediária. A lógica deste sub-programa foi baseada na especificação das instruções intermediárias descritas na seção II.2 .

III.3 - COMANDOS DO INTERPRETADOR

O interpretador C-PASCAL executa instruções intermediárias, e não comandos escritos diretamente em C-PASCAL.Portanto, o usuário deve conhecer a máquina e o conjunto de instruções intermediárias, descritos na seção II.2 , para poder usar as opções de depuração oferecidas durante a interpretação.

O interpretador pode ser usado entre a compilação e a tradução de um programa C-PASCAL, sem prejuízo do código intermediário.

Durante o processo de compilação, as linhas do programa fonte são impressas com os endereços dos códigos intermediários gerados. Além desta listagem de compilação, o usuário pode solicitar a impressão (mnemônico da instrução) de todos os códigos intermediários gerados.

As listagens de apoio, códigos intermediários e compilação, nos permite fazer referências cruzadas entre o código intermediário e o programa fonte. A correlação entre código intermediário e o programa fonte nos dá condições de acompanhar a interpretação, bem como a possibilidade de selecionar endereços 'chaves' para programar o vetor de interrupção por ponto de

quebra ('break-point').

O primeiro passo, na interpretação do código intermediário, é programar o vetor de parada de execução, com os endereços selecionados nas listagens de apoio.

O interpretador ao detectar erros em tempo de execução (divisão por zero, índice inválido, transbordo na pilha, etc...), fornece mensagens acompanhadas de um histórico dos últimos dezesseis códigos executados.

Os comandos do interpretador estão agrupados em três classes:

i-) COMANDOS DE INTERRUPÇÃO
ii-) COMANDOS DE EXECUÇÃO
iii-) COMANDOS DE DEPURAÇÃO

COMANDOS DE INTERRUPÇÃO

a-) Programar o Vetor de Interrupção (I+ XXXX)

O comando 'I+' insere o endereço (hexadecimal) 'XXXX' no vetor de pontos de quebra. O vetor tem capacidade para programar até dez endereços de parada. A programação do vetor pode ser verificada pelo comando ST (status) descrito mais adiante.

b-) Desprogramar o Vetor de Interrupção (I- XXXX)

O comando 'I-' remove o endereço (hexadecimal) 'XXXX' do vetor de interrupção.

c-) Terminar Interpretação (TI)

O comando TI encerra a interpretação do programa e devolve o controle da máquina para o monitor do sistema.

COMANDOS DE EXECUÇÃO

a-) INICIAR EXECUÇÃO (EI)

O comando EI restaura o contador de programas (PC) com o endereço do primeiro código intermediário, e passa a executar as instruções até encontrar uma interrupção por ponto de quebra ou final de programa. Este comando é usado quando se deseja reiniciar a execução do programa intermediário após uma interrupção.

b-) CONTINUAR EXECUÇÃO (EX)

O comando EX faz com que o programa seja executado a partir do valor corrente do Contador de Programa (PC). Este comando é utilizado para continuar a execução após uma parada por ponto de quebra ('break-point').

c-) EXECUTAR COM RASTREAMENTO (ER)

O comando ER executa o programa, a partir do valor corrente do PC, com o rastreamento das instruções. Este comando executa somente dezesseis instruções de cada vez, caso não ocorra uma interrupção por ponto de quebra ou final de programa. Este método permite o acompanhamento visual do fluxo lógico da execução do programa.

d-) EXECUTAR PASSO A PASSO (EP)

O comando EP executa a instrução corrente, e apresenta o mnemônico do próximo código intermediário apontado pelo contador de programa.

COMANDOS DE DEPURAÇÃO

a-) RASTREAMENTO DO PROGRAMA (RP)

O comando RP apresenta no vídeo os mnemônicos das dezesseis últimas instruções executadas, com seus respectivos endereços. Esta opção é normalmente aplicada após uma interrupção, para analisar o fluxo lógico do programa.

b-) STATUS (ST)

O comando ST apresenta os valores correntes de todos os registradores da máquina intermediária, das variáveis de controle, do vetor de interrupção e do topo da pilha. Com este comando podemos avaliar o comportamento do programa.

c-) LISTAR PROGRAMA (LP)

O comando LP apresenta no vídeo os mnemônicos das dezesseis instruções a partir do valor corrente do contador de programa, e a cada caractere branco digitado o processo se repetirá para as próximas dezesseis instruções. A interrupção da listagem é feita com um caractere diferente de branco.

d-) 'DUMP' DE MEMÓRIA (DP XXXX)

O comando DP apresenta no vídeo os conteúdos de 256 posições de memória, através de duas matrizes de 16 x 16 ,a partir do endereço (hexadecimal) XXXX . A primeira matriz apresenta cada conteúdo de memória no formato hexadecimal, e a segunda o mesmo conteúdo pelo código ASCII. As duas matrizes são apresentadas uma ao lado da outra, e os caracteres de controle' substituidos por ponto ('.') na matriz ASCII .

e-) DESMONTA CÓDIGO (DC)

O comando DC fornece uma listagem com os endereços (hexadecimal) e os mnemônicos das instruções intermediárias do programa C-PASCAL. Esta listagem de apoio é utilizada ' pelo usuário para programar o vetor de interrupções, para acompanhar a interpretação, e para correlacionar áreas do código intermediário com fragmentos do programa C-PASCAL fornecido pela listagem de compilação.

CAPITULO IV

TRADUTOR C-PASCAL

IV.1 - INTRODUÇÃO

Neste capítulo descrevemos a técnica de emulação da máquina virtual C-PASCAL, e a tradução do código intermediário. No desenvolvimento do Suporte C-PASCAL esta fase corresponde ao projeto do módulo de geração e otimização do código, para um determinado equipamento. Esta técnica é mais eficiente quando trabalhamos com equipamentos cuja configuração não se distancie muito da máquina virtual C-PASCAL portanto, a especificação do tradutor requer do projetista total conhecimento das características dos dois sistemas (C-PASCAL e MICRO-COMPUTADOR).

O termo emulação é utilizado devido a grande semelhança com os sistemas simulados por micro-programação[11] . Esta técnica consiste da adaptação do micro-computador, por rotinas programadas em assembler, para que este se comporte de maneira análoga à máquina virtual C-PASCAL (figura IV.I). Este processo corresponde ao método que decodifica e traduz o código intermediário para um código de máquina executável, diretamente pelo micro-computador hospedeiro, com apoio de rotinas auxiliares.

Como exemplo, apresentamos na seção IV.3 , a implementação da máquina C-PASCAL no micro-computador INTEL 8Ø8Ø e 8Ø85.

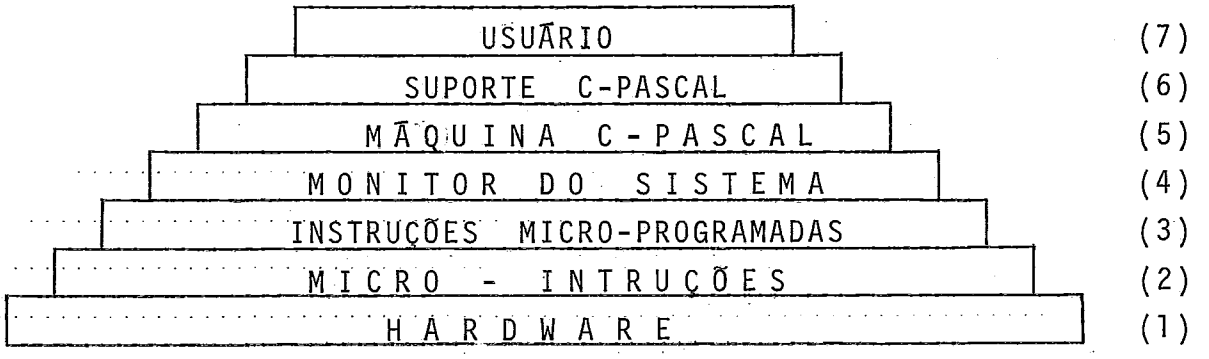


FIGURA IV.1

NÍVEL (1) - Neste plano temos o projeto do 'hardware' em função de um determinado grupo de micro-instruções.

NÍVEL (2) - Neste plano temos a primeira ferramenta de 'software', que são as micro-instruções definidas pelo'hardware', e que serão usadas para micro-programar as instruções do equipamento ('firmware'). Nesta fase podemos associar micro-instruções, definidas pelo 'hardware', que possam ser executadas no mesmo ciclo de máquina sem compartilhar recursos comuns. As micro-instruções associadas otimizam em espaço e tempo o conjunto de instruções do equipamento.

NÍVEL (3) - Neste patamar temos a 'Memória de Controle' do processador, formada por um conjunto de micro-rotinas, ('firmware') que serão endereçadas pelas instruções de máquina, através da decodificação interna feita pelo 'hardware'.

NÍVEL (4) - Neste plano temos o'Monitor de Sistemas'('software' básico) que irá gerenciar o equipamento através de um grupo de rotinas programadas em 'assembler', a fim de controlar o micro-computador em determinadas aplicações.

NÍVEL (5) - Conjunto de rotinas programadas em assembler capaz de emular a Máquina C-PASCAL. Estas rotinas serão endereçadas através da decodificação das instruções intermediárias feita pelo TRADUTOR .

NÍVEL (6) - Grupo de programas que formam o Suporte C-PASCAL (COMPILADOR, INTERPRETADOR, TRADUTOR e UTILITÁRIOS), programados em C-PASCAL, que forma a 'interface' do micro-computador com o usuário.

NÍVEL (7) - Neste plano encontramos os programas feitos em C-PASCAL pelos usuários do sistema.

IV.2 - PROJETO DE IMPLEMENTAÇÃO

O desenvolvimento do projeto de implementação da máquina virtual em um determinado equipamento,pode ser dividido em três etapas:

1 - ORGANIZAÇÃO E ALOCAÇÃO DOS REGISTRADORES
2 - PROGRAMAÇÃO DO CONJUNTO DE ROTINAS
3 - PROGRAMAÇÃO DO TRADUTOR

ORGANIZAÇÃO E ALOCAÇÃO DOS REGISTRADORES

Nesta fase fazemos um estudo comparativo entre as características dos processadores, C-PASCAL descrito na seção II.2 e do equipamento em questão. Como resultado deste estudo escolhemos o melhor sub-conjunto de registradores para simular a máquina virtual, sem prejuízo da eficiência do funcionamento do micro-computador. Esta etapa se torna mais simples e sua solução mais eficiente à medida que cresce a potencialidade do equipamento escolhido.

Com a conclusão desta fase do projeto, ficam estabelecidos:

a-) A simulação dos registradores da máquina virtual, e a alocação dos registradores do micro-computador.

b-) A forma de implementação das estruturas especiais, tais como: pilha, área de dados, registro de ativação, etc...

c-) A organização do pacote de rotinas.

PROGRAMAÇÃO DO CONJUNTO DE ROTINAS

O conjunto de rotinas simula as instruções da máquina C-PASCAL que se distanciem muito das características do micro-computador.

O primeiro passo desta etapa é a simulação de cada instrução isoladamente. As instruções mais complexas são resolvidas por rotinas que simularão as ações em tempo de execução, e para as instruções mais simples são criadas macro-instruções, em assembler, que serão expandidas no código de programa objeto.

Após a programação isolada, passamos a analisar grupos de instruções que possam ser avaliados conjuntamente pelo TRADUTOR, e executados por uma única rotina. Este tratamento dará ao código objeto uma melhor otimização em espaço e tempo de execução, e será tanto melhor quanto mais complexo for o suporte de rotinas.

Os agrupamentos de instruções durante o processo de tradução, envolvem conhecimentos da forma em que são gerados os códigos intermediários pelo compilador, bem como sensibilidade' e bom senso por parte do projetista para julgar a validade destas compactações no código objeto. Para isto, devemos levar em

conta: frequência dos agrupamentos, redução no tempo de execução, reduções do espaço e sobrecargas na tradução do código intermediário.

Com a conclusão desta etapa temos:

a-) Conjunto de macro-instruções que serão introduzidas diretamente no código objeto para simularem determinadas instruções.

b-) Conjunto de rotinas (simples) que serão chamadas diretamente pelo código objeto para resolver determinadas instruções.

c-) Conjunto de rotinas (agrupadas) que serão chamadas diretamente pelo código objeto para resolver determinados grupos de instruções compactados pelo tradutor.

PROGRAMAÇÃO DO TRADUTOR

A última etapa do projeto consiste em programar o TRADUTOR do código intermediário para código objeto do micro-computador. Esta programação tem como base as duas primeiras etapas (alocação dos registradores e determinação das rotinas), e sua lógica é semelhante ao interpretador (capítulo III) .

A implementação do Suporte num determinado micro-computador necessita de um tradutor, escrito em qualquer linguagem existente (até mesmo assembler) ou desenvolvido em outro equipamento que já possua o suporte C-PASCAL. Neste segundo caso, teremos os módulos básicos (compilador, interpretador e tradutor) em linguagem de máquina prontos para serem carregados no micro-computador.

O programa tradutor pode ser logicamente dividido em:

a-) Rotina que identifica e armazena todos os endereços referenciados pelo código intermediário.

b-) Rotina que decodifica cada código intermediário para ser traduzido pelo programa principal.

c-) Rotina que monta os códigos objeto fornecidos pelo programa principal.

d-) Programa principal que utilizando as rotinas acima executa a tradução do código intermediário, bem como resolve todas as referências entre o programa intermediário e o programa objeto.

IV.3 - TRADUTOR PARA O INTEL 8Ø8Ø E 8Ø85

A máquina virtual C-PASCAL é formada por um conjunto de instruções, uma estrutura de pilha e seis registradores: contador de programas (PC), ponteiro da pilha (SP), registro de base (BR), registro de temporárias (TR), registro de operando(OR), e registro de instrução (IR).

O projeto do tradutor para o micro-processador INTEL 8Ø8Ø e 8Ø85 será feito com base nas etapas descritas na seção IV.2 , que são: alocação dos registradores, programação do suporte de rotinas, e programação do tradutor.

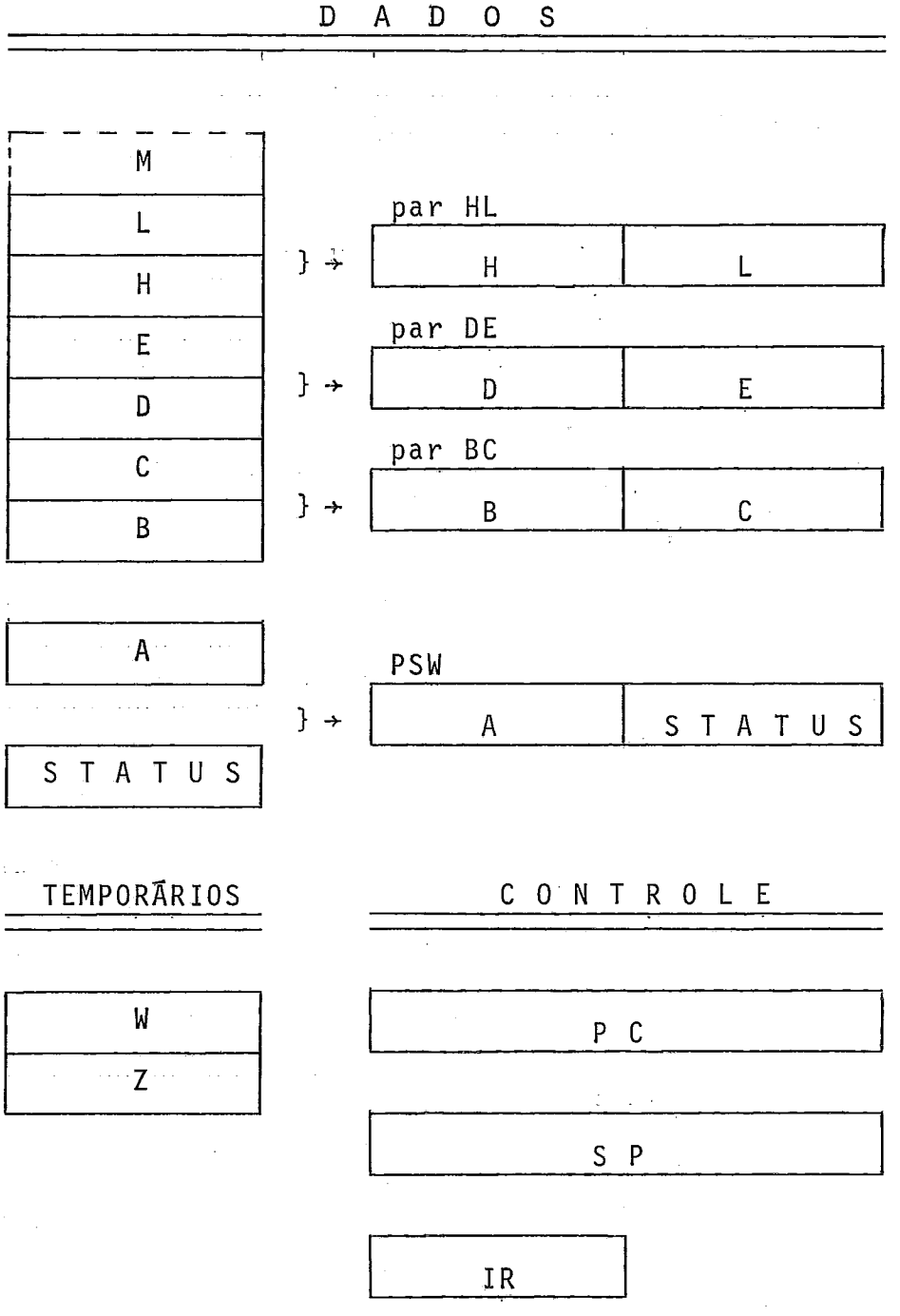
A UCP 8Ø8Ø / 8Ø85 é constituida basicamente por uma memória local, uma unidade aritmética/lógica e uma unidade de controle.

A memória local é formada por seis registradores de 16-bits, organizados da seguinte forma: um par de registradores para armazenamento temporário de 8-bits cada; seis registradores de 8-bits para uso geral; um registrador de 16-bits usado como ponteiro da pilha (Stack Pointer), e um registrador de 16-bits como contador de instrução. A unidade aritmética/lógica dispõe de mais dois registros: 'A' o acumulador de 8-bits e 'F' registrador de 'flags' de 8-bits relativo a última operação executada pela ALU. Estes dois registradores podem ser tratados juntamente como um registrador de 16-bits que representa a palavra de status do programa (PSW - program status word).

Os registradores temporários (W e Z de 8-bits) e o registrador de instruçõn (IR- 8-bits) são utilizados pela unidade de controle (UC) durante a execução das instruções, e não são acessáveis pelo programador.

Os seis registradores de uso geral (B,C,D,E,H e L) são diretamente endereçáveis por instruções, e podem ser utilizados isoladamente, ou aos pares, formando registradores de 16-bits para uso geral. Uma posição de memória é tratada como um registro de 8-bits, denotado de registro M . Quando uma instrução usa o registro M, o conteúdo do registrador par HL é tratado como endereço de memória do operando da instrução.

REGISTROS DO 8Ø8Ø OU 8Ø85



A	acumulador	
B	B - registro, dado	M - posição de memória,endereçada por HL
C	C - registro, dado	STATUS - resultado de status gerado pela ALU
D	D - registro, dado	PC -'Program' 'Counter' (16-bits)
E	E - registro, dado	IR -'Instruction' 'Register'
H	H - registro, dado	SP -'Stack' 'Pointer' (16-bits)
L	L - registro, dado	PSW - 'Program Status Word'(16-bits)

FIGURA IV.2

IV.3.1 - SIMULAÇÃO DOS REGISTROS C-PASCAL

Após estudo comparativo entre a máquina virtual C-PASCAL e o micro-computador INTEL 8Ø8Ø / 8Ø85 chegamos a seguinte simulação dos registradores:

CONTADOR DE PROGRAMA (PC)

O registrador contador de programa (PC-16 bits) será representado pelo seu equivalente no 8Ø8Ø (Program Counter),por se tratar de registrador de controle. Quando necessário serão feitos ajustes no PC do 8Ø8Ø, pelo suporte de rotinas C-PASCAL, para que o programa siga o fluxo normal de execução.
Por exemplo: No fragmento de um programa traduzido abaixo,temos:

```
   ⋮
500    PUSH    H
501    LXI     H, 532
504    CALL    ROTINA (X)
507    DW      PRMB        (1º parâmetro)
509    DW      PRMA        (2º parâmetro)
511    CALL    ROTINA (Y)
   ⋮
```

O fluxo correto do programa acima é executar a ROTINA (X) e em seguida executar a ROTINA (Y) . Para isto, a ROTINA (X), que representa uma instrução do C-PASCAL, é responsável pela atualização do endereço de retorno, para que o programa, no 8Ø8Ø, tenha prosseguimento normal.

ESTRUTURA E PONTEIRO DA PILHA (SP)

Para melhor utilização das instruções especiais do micro-processador (PUSH,POP,XTHL,SPHL,etc...), a pilha da máquina virtual será implementada na própria organização da 'Stack'-8Ø8Ø ou8Ø85. O ponteiro da pilha será simulado pelo registrador de controle 'Stack Pointer-SP' do 8Ø8Ø (16-bits).

As instruções C-PASCAL são dirigidas para manipular a pilha, e devido ao grande número de acesso, operações e restaurações do topo, este será mantido no registrador par HL(16bits), criando uma capacidade adicional no 8Ø8Ø para operar diretamente

o topo da pilha.

Com esta estrutura, conseguimos uma redução substancial em tempo e espaço no suporte de rotinas, bem como no código dos programas traduzidos.

Por exemplo:

OPERAÇÃO	CÓDIGO COM TOPO NA 'STACK/8080'		CÓDIGO COM TOPO NO PAR HL	
SOMA →	POP	H	→ POP	D
	POP	D	DAD	D
	DAD	D		
	PUSH	H		
NOT →	POP	H	→ CALL	NOT
	CALL	NOT		
	PUSH	H		
INVERSÃO →	POP	H	→ XTHL	
(top) ⇆ (top-1)	POP	D		
	PUSH	H		
	PUSH	D		

REGISTRADOR DE BASE

O registrador de base (BR-16 bits), da máquina virtual C-PASCAL, será implementado usando quatro registros M (32 bits de memória), representando, na realidade, dois registradores de 16-bits: Base Principal (BP) e Base Ativa (BA). Este desdobramento do BR permitirá, um acesso mais rápido em tempo de execução às variáveis globais.

A Base Principal (BP) indicará o endereço da área de dados do programa principal. O valor de BP é determinado pelo usuário em tempo de tradução, e carregado no próprio código do programa. Este endereço será usado para acessar as variáveis globais indexadas, em tempo de execução, causando assim uma redução no tempo de busca da base endereçamento.

A Base Ativa (BA) representará a base para o endereçamento dinâmico, em tempo de execução, das variáveis locais ao procedimento ativo. O endereço inicial é carregado pelo próprio código com o mesmo valor de BP, e é atualizado pelo suporte de rotinas à medida que são chamados os sub-programas.

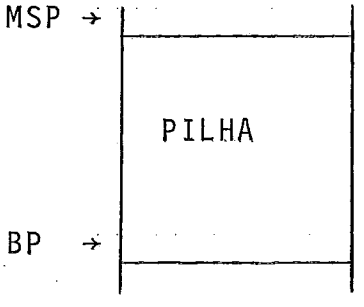
Por exemplo: Reserva dos 32-bits para BA e BP

```
BP   :DS   2
BA   :DS   2.
Início Padrão de Programa Traduzido 8Ø8Ø
      LXI  H,  <endereço do início da pilha>
      SHLD  BP
      SHLD  BA
      SPHL
      LXI  H,  <endereço do fim da pilha>
      SHLD  MSP .
```

LIMITE DA PILHA (MSP)

O endereço limite da pilha (MSP) é um valor fornecido pelo usuário em tempo de tradução para delimitar a região de memória organizada como pilha no 8Ø8Ø. Este parâmetro foi introduzido na implementação para o micro 8080/8085, visto que a pilha se organiza na própria memória, e é necessária uma proteção do mapeamento de memória do programa C-PASCAL em tempo de execução.

O suporte de rotinas testa o valor do ponteiro com o MSP em tempo de execução, e aborta a execução do programa quando ocorre uma invasão na área protegida.



O valor do MSP é carregado pelo próprio programa C-PASCAL, juntamente com os endereços iniciais de BP e BA.

REGISTRO DE TEMPORÁRIAS

Além dos registros temporários internos do micro 8Ø8Ø / 8Ø85, os registradores pares de uso geral BC e DE (16-bits cada) serão utilizados para simular o registro TR da

máquina virtual C-PASCAL.
Por exemplo:

```
SOMA → POP  D   ( registro temporário )
       DAD  D   ( HL ← HL + DE        )
```

REGISTRO DE INSTRUÇÃO E REGISTRO DE OPERANDO

Após a tradução do código intermediário os registradores IR e OR não são mais necessários. Durante o processo de tradução é feita a decodificação da instrução intermediária e gerado um código objeto 8Ø8Ø / 8Ø85 com todas as informações referentes aos registradores IR e OR.
Por exemplo:

CÓDIGO INTERMEDIÁRIO → CÓDIGO 8080/8085

IR	OR
LOD NIV	DESLOC
(1) (2)	(3)

```
(1)  CALL  LOD
(2)  DB    NIV
(3)  DW    DESLOC
```

No exemplo acima podemos observar que todas as informações de IR e OR são transferidas para o código 8Ø8Ø/8Ø85 pelo tradutor.

IV.3.2 - PROGRAMAÇÃO DAS ROTINAS

O Suporte de Rotinas executa as instruções C-PASCAL , assim como a memória de controle micro-programada executa as instruções do 8Ø8Ø / 8Ø85 (figura IV.I). Por este motivo, o pacote de rotinas deve ser mantido em memória principal, de preferência residente em 'ROM' (Read Only Memory).

A programação assembler das rotinas foi feita de maneira clara, para facilitar eventuais modificações, e tendo como objetivos: ocupar pouca memória (aproximadamente 1KB),e ser eficiente em tempo de execução. Para sua programação foram estudados vários algorítmos [4] apropriados para micro-processador 8Ø8Ø / 8Ø85, que executassem operações aritméticas e lógicas em menor ciclo de tempo.

IV.3.2.1 - PROGRAMAÇÃO SIMPLES

O primeiro passo, no projeto do suporte de rotinas, é programar em assembler, cada instrução da máquina virtual. Estes sub-programas executam exatamente os algorítmos descritos na seção II.2 , e têm suas estruturas de dados baseadas na seção IV.3.1 .

Ao final desta etapa, teremos para cada instrução C-PASCAL uma rotina assembler que irá simular sua execução no micro 8Ø8Ø / 8Ø85. A ligação, entre a instrução intermediária e rotina assembler, é feita pelo tradutor por uma instrução de "CALL <endereço rotina>", introduzida no código objeto 8080 / 8085 juntamente com os parâmetros necessários.

" LOADER "

As instruções de 'loader' da máquina virtual foram classificadas e simuladas por oito diferentes rotinas: carregar constantes no topo (LCTE); variáveis simples de programa principal (LODP); variáveis simples locais (LODL); variáveis simples não-locais (LODG); variáveis indexadas do programa principal (LODPX); variáveis indexadas locais (LODLX); variáveis indexadas não-locais (LODGX); e 'loader' direto de memória (LODM).

ROTINA	INSTRUÇÃO		TRADUÇÃO
LCTE:	LDI Ø CTE	→	PUSH H LXI H,CTE
LODP:	LOD 255 DES	→	PUSH H LHLD (BP-2DES)
LODL:	LOD Ø DES	→	CALL LODL DW DES
LODG:	LOD NIV DES (Ø<NIV>255)	→	CALL LODG DB NIV DW DES
LODPX:	LODX 255 DES	→	CALL LODPX DW DES

ROTINA	INSTRUÇÃO		TRADUÇÃO	
LODLX:	LODX Ø DES	→	CALL	LODLX
			DW	DES
LODGX:	LODX NIV DES	→	CALL	LODGX
			DB	NIV
			DW	DES
LODM:	LDM Ø Ø	→	MOV	L,M
			MVI	H,Ø

" STORE "

As instruções de 'store' da máquina virtual foram classificadas e simuladas por sete rotinas: variáveis simples do programa principal (STOP); variáveis simples locais (STOL); variáveis simples não-locais (STOG); variáveis indexadas do programa principal (STOPX); variáveis indexadas locais (STOLX); variáveis indexadas não-locais (STOGX); e 'store' direto em memória (STOM).

ROTINA	INSTRUÇÃO		TRADUÇÃO	
STOP:	STO 255 DES	→	SHLD	(BP-2DES)
			POP	H
STOL:	STO Ø DES	→	CALL	STOL
			DW	DES
STOG:	STO NIV DES	→	CALL	STOG
	(Ø NIV 255)		DB	NIV
			DW	DES
STOPX:	STOX 255 DES	→	CALL	STOPX
			DW	DES
STOLX:	STOX Ø DES	→	CALL	STOLX
			DW	DES
STOGX:	STOX NIV DES	→	CALL	STOGX
	(Ø NIV 255)		DB	NIV
			DW	DES

ROTINA	INSTRUÇÃO		TRADUÇÃO
STOM:	STM Ø Ø	→	MOV A,L
			POP H
			MOV M,A
			POP H

" INSTRUÇÕES DE CHAMADA - CAL "

A chamada de sub-programa foi classificada em três casos: sub-programas locais (CALLL); sub-programa não-locais (CALLG); e sub-programas externos (CALLE).

ROTINA	INSTRUÇÃO		TRADUÇÃO
CALLL:	CAL Ø END	→	CALL CALLL
			DW END"
CALLG:	CAL NIV END	→	CALL CALLG
			DB NIV
			DW END"
CALLE:	CALX Ø Ø	→	LXI D,(P+5)
			PUSH D
			PCHL
			POP H

" INSTRUÇÃO DE RETORNO - RET "

As instruções de retorno dos sub-programas foram classificadas em três casos: retorno de sub-programa sem parâmetro (RETØ); retorno de sub-programa com parâmetros (RETN); e final do programa principal (RETA).

ROTINA	INSTRUÇÃO		TRADUÇÃO
RETØ:	RET Ø Ø	→	CALL RETØ
RETN:	RET NPRM Ø	→	CALL RETN
			DB NPRM
RETA:	RET 255 Ø	→	JMP MONITOR

" INSTRUÇÕES DE DESVIO - JMP "

As instruções de desvio (incondicional e condicional) foram programadas em três casos: desvio incondicional (DINC); ' desvio se o topo é par (DFALSE); e desvio se o topo é impar ' (DTRUE).

ROTINA	INSTRUÇÃO		TRADUÇÃO	
DINC:	JMP Ø END	→	JMP	END"
DFALSE:	JPC Ø END	→	XRA	A
			ADD	L
			POP	H
			JZ	N
DTRUE:	JPC 1 END	→	XRA	A
			ADD	L
			POP	H
			JNZ	END"

" OPERADORES "

As operações, aritméticas e lógicas, foram programadas usando-se algorítmos para operar com variáveis de 16-bits.' As instruções DAD, INX, DCX do 8Ø8Ø / 8Ø85 foram largamente ' utilizadas por manipularem diretamente com variáveis de 16-bits, além do POP, PUSH, XTHL, etc...

ROTINA	INSTRUÇÃO		TRADUÇÃO	
NEG:	OPE Ø	→	CALL	NOT
			INX	H
NOT:	OPE 1	→	CALL	NOT
MUL...OR:	OPE N	→	CALL	ROTINA [N]
	(2≤N≤14)		(MUL,DIV,MOD,SHL,SHR,AND,EQL, NEQ,GEQ,GTR,LEQ,OR)	
SUB:	OPE 15	→	CALL	NOT
			INX	H
			POP	D
			DAD	H
ADD:	OPE 16	→	POP	D
			DAD	D

ENTRADA / SAÍDA

As instruções de entrada/saída foram programadas em função da 'interface'ASCII do monitor.

Para utilizar esta 'interface'foram programadas as rotinas para transformar:

a-) ASCII para decimal
b-) ASCII para hexadecimal
c-) Decimal para ASCII
d-) Hexadecimal para ASCII

Por exemplo: RES N PER → CALL ROTINA [N]
DB PER

Onde: ROTINA [N] corresponde ao endereço da N-ésima rotina de entrada/saída;
PER é o número do periférico (console, fita, disco, impressora, etc...)

DESLOCAMENTO NA PILHA

Esta instrução é usada para reservar ou liberar determinada área na pilha. Apesar de sua simplicidade, esta instrução é executada por uma rotina, que antes testa os limites da região da pilha para proteger as demais posições de memória.

OPERAÇÕES NA PILHA

As instruções do C-PASCAL que operam diretamente com a pilha são todas possíveis de serem traduzidas diretamente para o código objeto 8Ø8Ø, devido a representação do topo no registrador par HL.

OPERAÇÃO	INSTRUÇÃO		TRADUÇÃO
Decrementar o topo :	OPI Ø	→	DCX H
Copiar o topo :	OPI 1	→	PUSH H
Incrementar o topo :	OPI 2	→	INX H
Copiar o (topo-1) no topo :	OPI 3	→	POP D PUSH D PUSH H XCHG

OPERAÇÃO		INSTRUÇÃO		TRADUÇÃO
Trocar o (topo) com (topo-1)	:	OPI 4	→	XTHL
Alocar uma área	:	OPI 5	→	PUSH H
Deslocar uma área	:	OPI 6	→	POP H

IV.3.2.2 - PROGRAMAÇÃO AGRUPADA

O segundo passo, no projeto do suporte de rotinas, é fazer sub-programas em assembler que simulem grupos de instruções C-PASCAL.

Esta fase requer uma análise da geração dos códigos intermediários, pelo compilador, a fim de identificar grupos de instruções que possam ser avaliados em conjunto pelo tradutor. Estes grupos podem ser considerados como novas instruções C-PASCAL mais poderosas, que o tradutor irá gerar uma única chamada no código objeto.

Esta compactação tornará o código objeto 8Ø8Ø/8Ø85 mais eficiente, quanto a espaço e tempo, e o pacote de rotinas mais complexo. Fica para o projetista, decidir que grupos devem ser avaliados conjuntamente sem prejuizo no aumento do pacote de rotinas, nem no tempo de tradução.

Neste ponto apresentamos os agrupamentos considerados ideais nesta implementação.

OPERADOR C/ OPERANDO IMEDIATO

As operações imediatas são encontradas nas expressões ou sub-expressões que terminem por uma constante.

Estes agrupamentos são formados pelas instruções de 'loader' de uma constante seguidas por um operador qualquer. Por exemplo:

Código Intermediário		PROG.SIMPLES		PROG.AGRUPADA
(...+ 453....)	→	PUSH H	→	LXI D,453
LDI 453		LXI H,453		DAD D
OPE ADD		POP D		
		DAD D		
(...* 453....)	→	PUSH H	→	CALL MULI
LDI 453		LXI H,453		DW 453
OPE MUL		CALL MUL		

'LOADER' OU 'STORE' DE MEMÓRIA COM ENDEREÇO FIXO

Os acessos à memória pelo comando MEM [<exp>], cujo o índice é uma constante, podem ser agrupados diretamente no código, tal como:

```
Código INTERMEDIÁRIO          PROG.SIMPLES    PROG.AGRUPADA
(MEM [1500] := <exp>)   →     PUSH  H     →   MVI  M,1500
 LDI 1500                     LXI   H,1500    POP  H
 STM                          MOV   A,L
                              POP   H
                              MOV   M,A
                              POP   H
```

DESVIO RELACIONAL

As instruções de desvio condicional são sempre usadas após a avaliação de uma expressão lógica. Portanto, podemos associar o operador relacional com o desvio condicional, criando assim um conjunto de desvios relacionais, que serão tratados diretamente por uma rotina.

Por exemplo:

```
DESVIA SE IGUAL (JEQL)                →   TRADUÇÃO AGRUPADA
OPE  EQL          OU  OPE NEQ             CALL   JEQL
JPC TRUE ADR          JPC FALSE ADR       DW     ADR"

DESVIA SE MENOR (JLSS)                →   TRADUÇÃO AGRUPADA
OPE LSS               OPE GEQ             CALL   JLSS
JPC TRUE ADR      OU  JPC FALSE ADR       DW     ADR"
```

DESVIO RELACIONAL IMEDIATO

Neste agrupamento efetuamos uma compactação de três instruções: 'loader' constante, operador relacional e desvio condicional.

Por exemplo:

```
IF  A > 458  THEN  →   a- LOD  A
                       b- LDI  458
                       c- OPE  GTR
                       d- JPC  FALSE  ADR
                          :
```

No exemplo dado temos duas possibilidades para compactação simples (b e c ou c e d), porém, neste caso criamos um conjunto de rotinas que executam as três instruções de uma só vez, que são os DESVIOS RELACIONAIS IMEDIATOS.
Por exemplo:

```
DESVIA SE IGUAL A 535          →        TRADUÇÃO AGRUPADA
LDI  535                  LDI 535       CALL   JEQLI
OPE  EQL        OU        OPE NEQ       DW     535
JPC TRUE ADR              JPC FALSE ADR DW     ADR"
```

Os agrupamentos reduzem o código em aproximadamente 50%, por exemplo:

```
CÓDIGO INTERMEDIÁRIO      PROG.SIMPLES          PROG.AGRUPADA
    LDI  1500             PUSH H      (1)       CALL  JEQLI  (3)
    OPE  LSS              LXI  H,1500 (3)       DW    1500   (2)
    JPC  FALSE  ADR       CALL LSS    (3)       DW    ADR"   (2)
                          XRA  A      (1)                    (7 BYTES)
                          ADD  L      (1)
                          POP  H      (1)
                          JZ   ADR"   (3)
                                   (13 BYTES)
```

IV.3.3 - PROGRAMAÇÃO DO TRADUTOR

A programação do TRADUTOR, responsável pela geração do código objeto 8Ø8Ø/8Ø85, tem como estrutura de dados a alocação dos registradores (seção IV.3.1), e com lógica de tradução as rotinas descritas na seção IV.3.2.

A tradução do código intermediário usa, da melhor maneira possível, as opções criadas na seção IV.3.2 , bem como efetua otimizações locais [15] , a fim de gerar um código objeto 8080/8085 compacto e mais eficiente.

O programa TRADUTOR 8080/8085 escrito em C-PASCAL tem a seguinte organização:

```
PROGRAM   TRADUTOR 8080 E 8085;
    PROCEDURE   BUSCACODIGO;
    PROCEDURE   MONTACODIGO;(CODIGO,OPERANDO);
BEGIN   PROGRAMA PRINCIPAL   END.
```

BUSCACODIGO

Esta rotina efetua, a cada chamada, a busca e a decodificação do código intermediário.

O vetor de referências, entre o código intermediário e o código objeto, é atualizado durante a decodificação e as referências pendentes resolvidas.

A decodificação consiste em transformar o código intermediário em novos valores a serem atribuidos às variáveis globais: COD- código de instrução; IDX- tipo de instrução (indexada ou não); NIV- nível estático ou tipo de operador; VAL- valor numérico do operando; REF- se é ou não uma instrução referenciada.

MONTACODIGO (CODIGO, OPERANDO)

Esta rotina recebe dois parâmetros do programa principal, e através de um comando "CASE" reconhece os códigos do 8080/8085 de um, dois ou três'bytes', bem como as pseudos instruções (DB,DW,ORG,etc...). Com estas informações geramos o código objeto 8080/8085 na mesma região de memória do código intermediário.

Este procedimento retém informações sobre o último código objeto montado, para ser usado posteriormente na otimização local do código.

PROGRAMA PRINCIPAL

O programa principal restaura todas as variáveis do TRADUTOR com seus valores iniciais, monta o vetor com os endereços referenciados pelo código intermediário, interage com o usuário para receber os dados necessários para a tradução, bem como efetua a transformação do código C-PASCAL para código 8Ø8Ø/8Ø85.

O tradutor recebe e armazena uma série de dados que irão conduzir a tradução, tais como:

a- Base de endereçamento do código objeto
b- Endereço Inicial da Pilha
c- Endereço Limite da Pilha
d- Tipo de Tradução (M/S)

BASE DE ENDEREÇAMENTO

A base de endereçamento do código objeto é necessária para gerar um código objeto absoluto, esta informação funciona como a pseudo-instrução 'ORG' no assembler.

ENDEREÇO INICIAL / FINAL DA PILHA

Estes dois endereços são necessários para delimitar a região de memória onde será organizada a pilha do programa C-PASCAL .

TIPO DE TRADUÇAO (M/S)

O tipo M ('Master') é usado para traduzir programas ' C-PASCAL. A tradução do tipo 'Master' introduz no início do código objeto o grupo de instruções ('head') que restaura todas ' as variáveis do suporte C-PASCAL (limite do ponteiro, início da pilha, início da base ativa, início da base principal etc...)

O tipo S ('Slave') é usado para traduzir sub-programas (PROCEDURE ou FUNCTION) compilados separadamente do programa principal. A tradução do tipo 'Slave' gera simplesmente o código objeto da rotina, que deverá ser executado por um comando de CALL feito por outro programa C-PASCAL em tempo de execução.

O programa principal é responsável pela otimização local do código objeto[15] 8080/8085. Nesta implementação usamos as seguintes técnicas :

a- 'Loads' e 'Stores' Redundantes
b- Características da Máquina
c- Desvio Imediato.

'LOADS' E 'STORES' REDUNDANTES

Uma instrução de 'load' depois de um 'store' é desnecessária, desde que a segunda não seja referenciada. Para esta otimização o tradutor verifica a variável REF , para depois eliminar os 'loads' redundantes.

Por exemplo:

```
A := B + C ;                          LOD   B
IF A > 15 THEN ≪CMD> ;                LOD   C
                                      OPE   +
                                (a)   STO   A
                                (b)   LOD   A
                                      LDI   1500
                                      OPE   >
                                      JPC   FALSE  L1
                                      <CMD>
                                L1:
```

Podemos observar que as instruções (a) e (b) são redundantes, e como (b) não é referenciada será eliminada na tradução para o código objeto 8Ø8Ø/8Ø85.

Como exemplo, apresentamos a seguir três traduções do código intermediário acima.

TRADUÇÃO PARA 8Ø8Ø/8Ø85

PROG. SIMPLES	PROG.AGRUPADA	POP e PUSH REDUNDANTE	LOAD E STORE REDUNDANTE
(1) PUSH H	(1) PUSH H	(1) PUSH H	(1) PUSH H
(3) LHLD B"	(3) LHLD B"	(3) LHLD B"	(3) LHLD B"
(1) PUSH H	(1) PUSH H	(1) PUSH H	(1) PUSH H
(3) LHLD C"	(3) LHLD C"	(3) LHLD C"	(3) LHLD C"
(1) POP D	(1) POP D	(1) POP D	(1) POP D
(1) DAD D	(1) DAD D	(1) DAD D	(1) DAD D
(3) SHLD A"	(3) SHLD A"	(3) SHLD A"	(3) SHLD A"
(1) POP H	(1) POP H	(3) LHLD A"	(3) CALL JLEQI
(1) PUSH H	(1) PUSH H	(3) CALL JLEQI	(2) DW 1500
(3) LHLD A"	(3) LHLD A"	(2) DW 1500	(2) DW L1
(1) PUSH H	(3) CALL JLEQI	(2) DW L1	
(3) LXI H,1500	(2) DW 1500		
(3) CALL GTR	(2) DW L1		
(1) XRA A			
(1) ADD L			
(1) POP H			
(3) JZ L1"			
31 BYTES	25 BYTES	23 BYTES	20 BYTES

CARACTERÍSTICAS DA MÁQUINA

O 8080/8085 dispõe de instruções bem eficientes para incrementar ou decrementar variáveis de 16-bits; armazenadas nos registradores pares (HL,BC,DE). As operações de soma ou subtração com constantes n ($-6 \leq n \leq 6$) serão tratadas por estas instruções.

Por exemplo:

CÓDIGO INTERMEDIÁRIO	TRAD.SIMPLES			TRAD.OTIMIZADA			
(B := A + 3 ;)							
	PUSH	H	(1) 12	PUSH	H	(1)	(12)
LOD A	LHLD	A"	(3) 16	LHLD	A"	(3)	(16)
LDI 3	PUSH	H	(1) 12	INX	H	(1)	(6)
OPE +	LXI	H,3	(3) 10	INX	H	(1)	(6)
STO B	POP	D	(1) 10	INX	H	(1)	(6)
	DAD	D	(1) 10	SHLD	B	(3)	(16)
	SHLD	B"	(3) 16	POP	H	(1)	(10)
	POP	H	(1) 10				
			(14 BYTES)				(11 BYTES)
			96 CICLOS				72 CICLOS

REDUÇÕES EM ESPAÇO/TEMPO

Constante 5 → 14 BYTES → 13 BYTES
96 CICLOS → 86 CICLOS

Constante 6 → 14 BYTES → 14 BYTES
96 CICLOS → 92 CICLOS

'PUSH' E 'POP' REDUNDANTES

Na seção IV.3.2 definimos, na programação simples, o padrão 8080/8085 para cada instrução do C-PASCAL. Alguns padrões iniciam com a instrução PUSH H e outros terminam com a instrução POP H, e quando gerados em sequência podemos eliminar POP e PUSH redundantes.

Por exemplo:

	CÓDIGO INTERMEDIÁRIO		TRADUÇÃO		OTIMIZAÇÃO	
(A :=....;)	STO	A	SHLD	A"	SHLD	A"
(B := C+D;)	LOD	C	POP	H	LHLD	C"
	LOD	D	PUSH	H	PUSH	H
	OPE	+	LHLD	C"	LHLD	D"
	STO	B	PUSH	H	POP	D
			LHLD	D"	DAD	D
			POP	D	SHLD	B"
			DAD	D		
			SHLD	B"		

CAPITULO V

SUPORTE C-PASCAL

V.1 - INTRODUÇÃO

O Suporte C-PASCAL foi projetado para ser utilizado como ferramenta de 'software' no desenvolvimento de programas em micro-computadores que possuam as seguintes características:

a- Configuração de memória principal a partir de 16KB ('RAM');

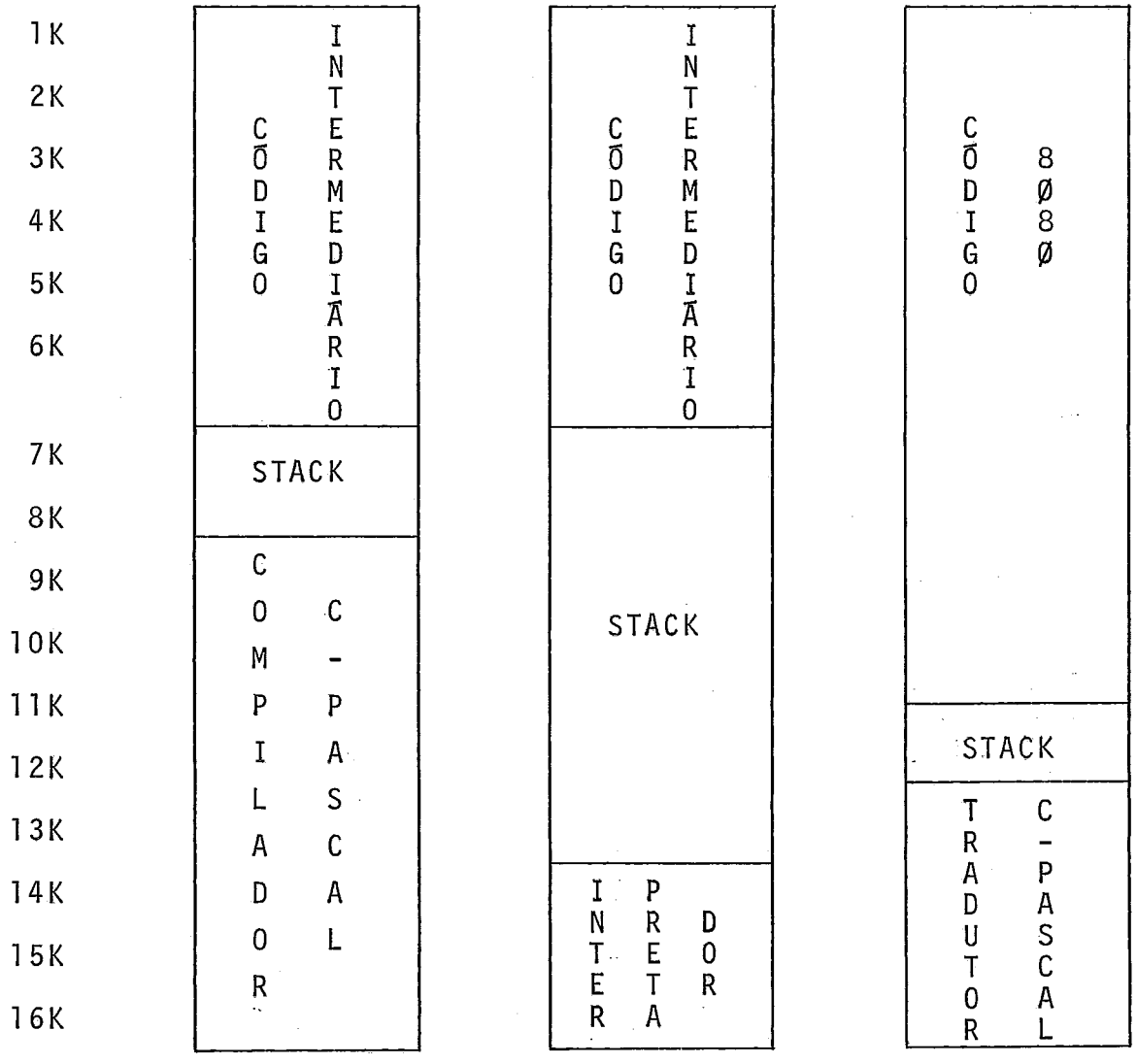
b- Memória secundária (disco flexível ou fita);

c- Monitor do sistema capaz de efetuar carga de programas, e salvar áreas da memória principal na memória secundária.

O Suporte básico é formado por um Compilador, um Interpretador e um Tradutor. Todos os três módulos são carregados na mesma região de memória principal. O mapeamento de memória da figura V.1 apresenta a estrutura de 'overlay' usada pelo suporte no desenvolvimento de programas C-PASCAL.

As operações de entrada/saída foram desenvolvidas da forma mais simples, e sem nenhum compromisso com determinado sistema operacional para manipulação de memórias secundárias. As rotinas de entrada/saída, na atual versão, são feitas caracter a caracter, na codificação ASCII, para uma impressora ou para um terminal de vídeo, ficando a comunicação com a memória secundária por conta do monitor.

MAPEAMENTO DA MEMÓRIA PRINCIPAL (RAM)



O mapeamento dos 16KB de memória 'RAM' acima, apresenta as três fases no desenvolvimento de um projeto no suporte ' C-PASCAL.

O monitor e as rotinas da máquina C-PASCAL são mantidos em memória tipo ROM (Read Only Memory).

FIGURA V.1

V.2 - METÓDO DE COMPILAÇÃO

O programa objeto do compilador C-PASCAL, código do programa para o INTEL 8Ø8Ø ou 8Ø85 e as tabelas (palavras reservadas, símbolos ('tokens'), erro, etc...), ocupa 8KB da memória principal ('RAM'). Para sua execução é necessária uma área adicional de aproximadamente 2KB, para funcionar como pilha. Desta área adicional, os primeiros mil 'bytes' (endereços mais altos) são usados para armazenar a tabela de símbolos e as variáveis globais, e os demais são utilizados para: alocar variáveis temporárias e variáveis locais aos procedimentos, avaliar expressões, salvar contexto, etc...

O compilador gera o código intermediário do programa fonte diretamente na memória, assim o código pode ter no máximo (M-10)KB de tamanho, sendo M a área disponível de memória no equipamento. A configuração considerada como mínima (16KB), apresentada na figura V.I, não pode ultrapassar 6KB de código intermediário para um único programa. Quando ocorre invasão da pilha pela área de código, o programa deverá sofrer uma reestruturação, e se necessário dividí-lo em módulos e compilá-los separadamente usando a técnica de programação modular apresentada na seção II.3 .

A compilação de programas no Suporte C-PASCAL envolve, no atual estágio, três passos:

i-) Editar o programa fonte C-PASCAL. Este procedimento depende estreitamente do equipamento, pois envolve rotinas de leitura/escrita do monitor e a organização dos arquivos em disco ou fita. Para solucionar temporariamente este problema, o compilador dispõe de um modo interativo de compilação que ler o programa fonte diretamente do teclado.

ii-) Carregar o programa compilador (objeto) nos últimos 8KB da memória 'RAM'. Este mecanismo existe em todos os monitores dirigidos para fita ou disco (LOADER).

iii-) Executar o programa compilador. A execução consiste em passar o controle do sistema para o programa carregado na memória, usando um comando do monitor (JMP,GO,etc...) e como ponto de entrada o endereço usado para carregá-lo na memória.

Os passos acima podem ser automatizados por um progra

ma escrito em assembler, que residente em 'ROM' fará o papel do monitor C-PASCAL.

A partir do passo três o compilador irá ler cada registro do programa fonte até o fim. O código intermediário deste programa será armazenado nas primeiras posições da memória e a listagem de compilação será impressa durante a execução.

Ao final da compilação o controle será devolvido ao monitor, para que o usuário possa salvar o código, e chamar o Interpretador ou o Tradutor. Caso o usuário queira uma listagem simbólica do código intermediário, poderá chamar o Interpretador e usar o comando DC (Desmonta código), que simplesmente percorre a área de memória e emite uma listagem com os endereços (hexadecimal) e os mnemônicos das instruções intermediárias.

A listagem V.1 apresenta a compilação do programa TORRE DE HANOI, neste exemplo temos documentado alguns passos do processo de compilação.

V.3 - MÉTODO DE INTERPRETAÇÃO

O programa Interpretador C-PASCAL, traduzido para o código 8Ø8Ø ou 8Ø85 e mais sua tabela de mnemônicos, ocupa aproximadamente 3,5KB de memória principal. Este módulo requer, em tempo de execução, uma área adicional com aproximadamente 4KB da memória para simular a pilha da máquina virtual, juntamente com sua área de trabalho.

O Interpretador trabalha sobre o código intermediário, gerado pelo compilador, sem no entanto destruí-lo e portanto, o interpretador pode ser usado com um estágio intermediário entre a Compilação e a Tradução.

A interpretação de programas no Suporte C-PASCAL envolve, no atual estágio, cinco passos:

i-) Carregar o código intermediário na memória. Caso a interpretação seja subsequente à compilação este passo não é necessário. Códigos intermediários anteriormente armazenados em disco ou fita devem ser carregados na mesma região de memória em que foram gerados pelo compilador.

ii-) Carregar o programa interpretador (objeto) nos últimos 3,5KB da memória 'RAM'. Este mecanismo existe em todos os monitores que operam com disco ou fita (LOADER).

iii-) Executar o programa interpretador. A execução consiste em passar o controle do sistema para o interpretador, usando um comando do monitor tipo 'JMP' ou 'GO', e como ponto de entrada o endereço usado para carregá-lo na memória.

iv-) Imprimir uma listagem dos mnemônicos do programa intermediário, usando o comando do interpretador DC (Desmonta Código). Com ajuda desta listagem e da listagem de compilação, programar o vetor de interrupções com os endereços escolhidos para depuração.

v-) Iniciar a interpretação.

A partir deste ponto o interpretador irá percorrer toda a região de memória executando cada código intermediário. Este procedimento é feito sob supervisão do usuário que interage com a máquina através de um conjunto de comandos descritos na seção III.4 .

Ao final da interpretação, o controle será devolvido

ao monitor, para que o usuário possa chamar o Tradutor ou o Editor para eventuais correções no programa fonte.

A listagem V.2 apresenta a Interpretação do programa TORREDEHANOI, este exemplo anteriormente compilado já possui o código intermediário na memória.

V.3 - MÉTODO DE TRADUÇÃO

O programa Tradutor C-PASCAL, traduzido para o código 8Ø8Ø ou 8Ø85 e mais uma tabela de endereços, ocupa aproximadamente 4KB de memória. A região de memória organizada com pilha em tempo de execução tem tamanho aproximado de 1KB.

O tradutor gera o programa objeto diretamente na memória, sobre o próprio código intermediário. Portanto o programa objeto pode ter no máximo (M-5)KB de tamanho, sendo 'M' a memória disponível no equipamento. Para uma configuração de 16KB temos possibilidade de gerar programas com até 11KB de objeto em 8Ø8Ø ou 8Ø85. Caso ultrapasse a área disponível para código objeto, o programa poderá ser dividido em blocos,e serem traduzidos separadamente, usando a técnica de Programação com 'Overlay', descrita na seção II.3.

O tradutor gera código absoluto para 8Ø8Ø ou 8Ø85 em relação ao endereço fornecido em tempo de tradução, sempre na mesma região de memória. O início da área fica 50 BYTES acima da região do código intermediário, sendo possível,porque o objeto 8Ø8Ø ou 8Ø85 sofre processos de otimização chegando a reduzir em média, 35% em relação ao tamanho do código intermediário.

A tradução de programas no Suporte C-PASCAL envolve , no atual estágio, quatro passos:

i-) Carregar o código intermediário na memória. Caso a tradução seja subsequente à compilação ou à interpretação este passo não é necessário. Códigos intermediários anteriormente gravados em disco ou fita devem ser carregados na mesma região de memória em que foram gerados pelo compilador.

ii-) Carregar o programa tradutor (objeto) nos últimos 4KB da memória. Este mecanismo de carga de programas objeto existe em todos os monitores que operam com disco ou fita como memória de massa (LOADER, INSERT, etc...)

iii-) Executar o programa tradutor. Para executar um programa objeto residente na memória basta passar o controle do sistema usando um comando do monitor tipo'JMP XXXX , ou 'GO XXXX , sendo XXXX o ponto de entrada dado pelo endereço usado para carga.

iv-) Fornecer todos os valores (PC, BASE, tipo de tradução) e iniciar a tradução.

A partir deste ponto o tradutor irá percorrer toda a região do código intermediário e gerar um código 8Ø8Ø ou 8Ø85 equivalente. O programa objeto terá as características determinadas pelo usuário em tempo de tradução, tal como descrito no capítulo IV.

Ao final da tradução o controle será devolvido ao monitor, para que o usuário possa salvar o código 8Ø8Ø gerado ou executá-lo.

A listagem V.3 apresenta a tradução do programa TORREDEHANOI.

CAPÍTULO VI

C O N C L U S Ã O

O objetivo central, oferecer aos usuários de microcomputadores uma linguagem estruturada e recursiva que permita o desenvolvimento de projetos que utilizem microprocessadores de uma forma prática e segura, foi alcançado dentro dos padrões acadêmicos.

A presente versão está sendo usada pelo CEPEL e em fase de implementação na UNIVERSIDADE FEDERAL DE SÃO CARLOS E NA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, e com a continuação do uso desta ferramenta poderemos medir a eficiência de suas aplicações.

A elaboração deste trabalho foi de grande importância para nós, e deixou bem claro as dificuldades encontradas na prática, quando na implementação de um suporte de desenvolvimento em um equipamento sem maiores recursos de 'sottware' e sem uma equipe para dar suporte em 'hardware' e sistemas operacionais.

O suporte C-PASCAL trata-se de um trabalho de cunho acadêmico, e sua documentação está disponível para todo tipo de usuário, a fim de incentivar o intercâmbio de idéias nesta área, principalmente entre universidades e centros de pesquisas.

Pela importância que tem atualmente as aplicações dos microprocessadores sugerimos que a universidade se empenhe em dar continuidade à pesquisa nesta linha, e para tal propósito apresentamos alguns pontos:

a - Implementação do tipo REAL, usando um pacote de rotinas já desenvolvido para manipulação e operação com ponto flutuante;

b - Trabalhar o código intermediário, a fim de otimizá-lo com códigos de tamanho variável, bem como da introdução de novos códigos para manipulação de reais;

c - Desenvolvimento de uma 'interface' mais adequada para manipulação de arquivos no sistema de entrada/saída;

d - Analisar o desempenho do compilador em tempo/espaço e estrutura de suas rotinas, a fim de propor melhorias que otimizem o método de compilação;

e - Desenvolvimento de um editor de textos, em C-PASCAL, para ser adcionado ao suporte com o objetivo de torná-lo mais auto-suficiente;

f - Desenvolvimento de um tradutor para o microprocessador Z-8Ø, seguindo as instruções dadas no capítulo IV, e em conjunto com a equipe do Núcleo de Computação Eletrônica da UFRJ (NCE) que dispõe de tal equipamento.

Outras extensões podem ser adicionadas ao C-PASCAL, a fim de torná-lo mais próximo do PASCAL[1] original, tais como RECORD sem a parte variante, TYPE com os tipos simples e estruturados.

As implementações dos três módulos, COMPILADOR, INTERPRETADOR e TRADUTOR apresentados neste trabalho, foram de grande importância para o nosso desenvolvimento profissional que necessitava de experiências práticas, a fim de medir as reais limitações dos métodos teóricos em relação aos equipamentos de pequeno porte.

O estudo apresentado não pretende ser um modelo sobre implementações em microcomputadores mas sim um conjunto de métodos práticos aplicados a um equipamento real e de poucos recursos, que pode e deve ser melhorado "a posteriori".

BIBLIOGRAFIA

1. JENSEN, K. e WIRTH, N. - PASCAL user Manual and Report. Spring-Verlag, New York, 1975.

2. WILSON, I.R. - PASCAL for School and Hobby use. Software-Particle and Experience, Vol. 10, 659-671, 1980.

3. CHUNG, K.M. e YUEN, H. - A "TINY" PASCAL Compiler. BYTE, (Set., Out. e Nov. 1978).

4. UCSD (Mini-Micro-Computer) PASCAL Version I.5. - Institute for Information Systems - UCSD, 1978.

5. TORSTENDAHL, S. - PASCAL for PDP 11 Under RSX/IAS. LM Ericsson (Report-1977).

6. LAUFER, C. e NOVAES, F. LPM - Linguagem de Programação para Microprocessador. - Trabalho Interno da COPPE-UFRJ, 1979.

7. INTEL CORPORATION - MCS 85 User's Manual - 1977.

8. DE SIMONE, E.G. e CAULA, L.B. - Critérios de Projeto de uma Família de Linguagens de Médio Nível. Seminário Integrado de Software e Hardware, S, 1978.

9. REGHIZZI, S.C. - A Survey of Microprocessor Languagens. Computer, 1980.

10. SCHMITZ, E.A. - PASCAL-Orientated Computer Design. Imperial College, Londres, Tese Ph.D., 1980.

11. PRATT, T.W. - Progranming Languages: Design and Implementation. Prentice-Hall, Englewood Cliffs, N.J., 1975.

12. GRIES, D. - Compiler Construction for Digital Computers. Wiley, 1971.

13. WIRTH, N. - Algorithms + Data Structures = Programs. Prentice-Hall, Englewood Cliffs, N.J., 1975.

14. RIPLEY, G.D. e DRVSEIKIS, F.C. - A Statistical Analysis of Syntaxe Errors. Computer Languages, Vol-3, 227-240, 1978.

15. AHO, A.V. e ULLMAN, J.D. - Principles of Compiler Design. Addison - Wesley, 1978.

LISTAGEM I

```
   COMPILADOR  C-PASCAL VERSAO 1.0        PAGINA      1

[10000]          1  PROGRAM INTERP ;
[10004]          2  CONST PERO   = 0 ;
[10004]          3         EBMNE1 = 5200 ;
[10004]          4         EBMNE2 = 5239 ;
[10004]          5         BASEPC = #2600 ;
[10004]          6         SPMAX  = 2000 ;
[10004]          7  VAR    PC    , (* CONTADOR DE PROGRAMA *)
[10004]          8         BR    , (* BASE DE ENDERECO     *)
[10004]          9         SP    , (* PONTEIRO DA PILHA    *)
[10004]         10         COD   , (* CODIGO C-PASCAL      *)
[10004]         11         NIV   , (* NIVEL ESTATICO       *)
[10004]         12         VAL   , (* VALOR DO OPERANDO    *)
[10004]         13         IDX   , (* FLAG DE INDEXACAO    *)
[10004]         14         TRC   , (* FLAG DE RASTREAMENTO *)
[10004]         15         X1,X2, (* VARIAVEIS AUXILIARES *)
[10004]         16         X3,X4, (* VARIAVEIS AUXILIARES *)
[10004]         17         CMD1 , (* VARIAVEL AUXILIAR    *)
[10004]         18         CMD2 , (* VARIAVEL AUXILIAR    *)
[10004]         19         PTR  , (* PONTEIRO DO TRACE    *)
[10004]         20         FIM    (* VARIAVEL DE CONTROLE *)
[10004]         21                : INTEGER ;
[10004]         22         STACK : ARRAY [ 0..SPMAX ] OF INTEGER ;
[10004]         23         VETTR : ARRAY [ 0..15 ]    OF INTEGER ;
[10004]         24         VETBP : ARRAY [ 0..10 ]    OF INTEGER ;
[10004]         25
[10004]         26  FUNCTION BASE : INTEGER ;
[10004]         27  BEGIN
[10004]         28      IF NIV = 255
[10012]         29      THEN X1 := 0
[10024]         30      ELSE BEGIN
[10032]         31           X1 := BR ;
[10040]         32           WHILE NIV > 0
[10048]         33           DO BEGIN
[10056]         34              X1 := STACK[X1] ;
[10068]         35              NIV := NIV - 1
[10076]         36              END
[10084]         37           END ;
[10088]         38      BASE := X1
[10092]         39  END ;
[10100]         40  (*$P+*)
```

```
   COMPILADOR  C-PASCAL VERSAO 1.0        PAGINA     2

[10100]         41  PROCEDURE INICIA ;
[10100]         42  BEGIN
[10100]         43      PC := BASEPC ;
[10108]         44      BR   :=  0 ;
[10116]         45      STACK[0] :=  0 ;
[10128]         46      STACK[1] :=  0 ;
[10140]         47      STACK[2] := -1 ;
[10156]         48      SP   :=  2 ;
[10164]         49      FIM  :=  0 ;
[10172]         50      FOR X1:=0 TO 15 DO VETTR[X1]:=BASEPC;PTR:=15;
[10240]         51  END ;
[10244]         52
[10244]         53  PROCEDURE CODIGO(ADR : INTEGER) ;
[10244]         54  BEGIN
[10244]         55      X2 := ' ' ;
[10252]         56      X1 := ADR ;
[10260]         57      COD := MEM[ X1 ] ;
[10272]         58      IF COD>15 THEN BEGIN COD:=COD - 16; X2:='X' END ;
[10312]         59      NIV := MEM[ X1+1 ] ;
[10332]         60      VAL := MEM[ X1+2 ] + MEM[ X1+3 ] SHL 8 ;
[10380]         61      WRITE(PER0,%X1,' --> ') ;
[10416]         62      X1 := COD + COD + COD + EBMNE1 ;
[10448]         63      WRITE(0,&MEM[X1],&MEM[X1+1],&MEM[X1+2],&X2,'   ');
[10524]         64      X1 := NIV + NIV + NIV + EBMNE2 ;
[10556]         65      CASE COD OF
[10564]         66      0,11   :WRITELN(PER0,$VAL,'  ',%VAL) ;
[10636]         67      9,10,12:BEGIN
[10680]         68              IF COD = 10 THEN X1 := X1 + 51 ;
[10712]         69              IF COD = 12 THEN X1 := X1 + 78 ;
[10744]         70              WRITELN(0,&MEM[X1],&MEM[X1+1],&MEM[X1+2])
[10800]         71              END
[10800]         72      OTHERS :WRITELN(PER0,%NIV,' / ',%VAL)
[10844]         73      END
[10848]         74  END ;
[10852]         75
[10852]         76  (*$P+*)
```

```
   COMPILADOR  C-PASCAL VERSAO 1.0        PAGINA      3

[10852]          77  PROCEDURE EXECUTA ;
[10852]          78  BEGIN
[10852]          79      IF TRC THEN CODIGO(PC) ;
[10868]          80      PTR := (PTR+1) AND #000F ;
[10892]          81      VETTR[PTR] := PC ;
[10904]          82      IDX := 0 ;
[10912]          83      COD := MEM[ PC ] ;
[10924]          84      IF COD>15 THEN BEGIN COD:=COD-16; IDX:=1 END;
[10964]          85      NIV := MEM[ PC+1 ] ;
[10984]          86      VAL := MEM[ PC+2 ] + MEM[ PC+3 ] SHL 8 ;
[11032]          87      PC := PC + 4 ;
[11048]          88      CASE COD OF
[11056]          89      0  :BEGIN  (* LOAD UMA CONSTANTE INTEIRA *)
[11068]          90          SP := SP + 1 ;
[11084]          91          STACK[SP] := VAL
[11092]          92          END ;
[11104]          93      1  :BEGIN   (* LOAD OU LOADX *)
[11116]          94          IF IDX
[11120]          95          THEN BEGIN
[11124]          96               VAL := VAL + STACK[SP] ;
[11144]          97               SP := SP - 1
[11152]          98               END ;
[11160]          99          SP := SP + 1 ;
[11176]         100          STACK[SP] := STACK[ BASE + VAL ]
[11200]         101          END ;
[11212]         102      2  :BEGIN  (*  LOAD DE MEMORIA DIRETAMENTE *)
[11224]         103          STACK[SP] := MEM[ STACK[SP] ]
[11240]         104          END ;
[11252]         105      3  :BEGIN  (* STORE OU STORE INDEXADO *)
[11264]         106          IF IDX THEN VAL := VAL + STACK[ SP-1 ] ;
[11300]         107          STACK[ BASE+VAL ] := STACK[SP] ;
[11328]         108          SP := SP - 1 - IDX
[11344]         109          END ;
[11360]         110      4  :BEGIN  (* STORE DIRETO EM MEMORIA *)
[11372]         111          MEM[ STACK[SP-1]] := STACK[SP] ;
[11400]         112          SP := SP-2
[11408]         113          END ;
[11424]         114      5  :BEGIN (* CHAMADA DE PROCEDURE OU FUNCTION *)
[11436]         115          IF IDX
[11440]         116          THEN
[11444]         117          BEGIN VAL:=STACK[SP];STACK[SP]:=0;SP:=SP-1 END
[11484]         118          ELSE STACK[SP+1] := BASE ;
[11512]         119          STACK[SP+2] := BR ;
[11532]         120          STACK[SP+3] := PC ;
[11552]         121          BR := SP + 1 ;
[11568]         122          PC := VAL ;
[11576]         123          SP := SP + 3
[11584]         124          END ;
[11600]         125      6  :BEGIN (* RETORNO DE PROCEDURE OU FUNCTION  *)
[11612]         126          SP := BR ;
[11620]         127          BR := STACK[SP+1] ;
[11640]         128          PC := STACK[SP+2] ;
[11660]         129          SP := SP - 1 - NIV
[11676]         130          END ;
[11692]         131      7  :BEGIN  (* DESVIO INCONDICIONAL OU GOTO *)
[11704]         132          PC := VAL
[11708]         133          END ;
```

```
   COMPILADOR  C-PASCAL VERSAO 1.0        PAGINA     4

[11720]        134       8  :BEGIN  (* DESVIO CONDICIONAL *)
[11732]        135           IF STACK[SP] = NIV THEN PC := VAL ;
[11760]        136           SP := SP - 1
[11768]        137           END ;
[11784]        138       9  :BEGIN (* OPERACOES NA PILHA *)
[11796]        139           SP := SP - 1 ;
[11812]        140           X1 := STACK[SP] ;
[11824]        141           X2 := STACK[SP+1] ;
[11844]        142           CASE NIV OF
[11852]        143           0,1:BEGIN SP := SP + 1; X1 := -X2 - NIV END ;
[11924]        144           2  : X1 := X1  *  X2 ;
[11960]        145           3  : X1 := X1 DIV X2 ;
[11996]        146           4  : X1 := X1 MOD X2 ;
[12032]        147           5  : X1 := X1 SHL X2 ;
[12068]        148           6  : X1 := X1 SHR X2 ;
[12104]        149           7  : X1 := X1 AND X2 ;
[12140]        150           8  : X1 := X1  =  X2 ;
[12176]        151           9  : X1 := X1 <>  X2 ;
[12212]        152           10 : X1 := X1  <  X2 ;
[12248]        153           11 : X1 := X1 >=  X2 ;
[12284]        154           12 : X1 := X1  >  X2 ;
[12320]        155           13 : X1 := X1 <=  X2 ;
[12356]        156           14 : X1 := X1 OR  X2 ;
[12392]        157           15 : X1 := X1  -  X2 ;
[12428]        158           16 : X1 := X1  +  X2
[12448]        159           END ;
[12460]        160           STACK[SP] := X1
[12468]        161           END ;
[12480]        162       10 :BEGIN    (* OPERACAO DE ENTRADA E SAIDA *)
[12492]        163           X1 := STACK[SP] ;
[12504]        164           SP := SP - 1 ;
[12520]        165           CASE NIV OF
[12528]        166           0   :READ(PER0,$X1) ;
[12556]        167           1   :READ(PER0,%X1) ;
[12584]        168           2   :READ(PER0,&X1) ;
[12612]        169           3   :BEGIN
[12624]        170                SP := SP + 1 ;
[12640]        171                FOR X2 := 1 TO MEM[PC+2]
[12660]        172                DO BEGIN
[12684]        173                   PC := PC + 4 ;
[12700]        174                   WRITE(PER0,&MEM[PC+2])
[12720]        175                   END ;
[12736]        176                PC := PC + 4
[12744]        177                END ;
[12760]        178           4   :WRITE(PER0,$X1) ;
[12788]        179           5   :WRITE(PER0,%X1) ;
[12816]        180           6   :WRITE(PER0,&X1)
[12836]        181         OTHERS:BEGIN
[12840]        182                SP := SP + 1 ;
[12856]        183                WRITELN(PER0)
[12860]        184                END
[12860]        185           END ;
[12864]        186           IF VAL<3 THEN BEGIN SP:=SP+2;STACK[SP]:=X1 END
[12908]        187           END ;
[12916]        188  (*$P+*)
```

```
   COMPILADOR  C-PASCAL VERSAO 1.0        PAGINA     5

[12916]        189       11 :BEGIN    (* AVANCA O PONTEIRO DA PILHA *)
[12928]        190           SP := SP + VAL ;
[12944]        191           IF SP > SPMAX
[12952]        192           THEN BEGIN
[12960]        193                WRITELN(PERO,'OVERFLOW STACK');
[13028]        194                FIM := 1
[13032]        195                END
[13036]        196           END ;
[13044]        197
[13044]        198       12 :BEGIN  (* OPERACOES NO TOPO DA PILHA *)
[13056]        199           X1 := STACK[SP] ;
[13068]        200           X2 := STACK[SP-1] ;
[13088]        201           CASE NIV OF
[13096]        202           0  : X1 := X1 - 1 ;
[13132]        203           1,5: SP  := SP + 1 ;
[13184]        204           2  : X1 := X1 + 1 ;
[13220]        205           3  :BEGIN X1 := X2; SP := SP + 1 END ;
[13264]        206           4  :BEGIN STACK[SP-1] := X1; X1 := X2 END ;
[13312]        207           6  :BEGIN SP := SP - 1; X1 := X2 END ;
[13356]        208           7  :BEGIN
[13368]        209               X3 := X1 ;
[13376]        210               SP := SP - 2 ;
[13392]        211               X1 := STACK[ SP ] ;
[13404]        212               IF (X1>X2) OR (X1<X3)
[13428]        213               THEN BEGIN
[13436]        214                    WRITELN(PERO,'INDICE INVALIDO ') ;
[13512]        215                    FIM := 1
[13516]        216                    END
[13520]        217               END
[13520]        218           END ;
[13524]        219           STACK[SP] := X1
[13532]        220           END
[13536]        221        END ;
[13540]        222        (* TESTA   'BREAK POINT' *)
[13540]        223        FOR X1:=0 TO 10
[13548]        224        DO IF VETBP[X1] = PC
[13584]        225           THEN BEGIN
[13592]        226                IF PC = -1
[13600]        227                THEN
[13612]        228                BEGIN WRITELN(PERO,'SUCESSO'); PC:=0 END
[13660]        229                ELSE WRITELN(PERO,' PAUSA ') ;
[13704]        230                FIM := 1
[13708]        231                END
[13712]        232  END ;
[13732]        233  (*$P+*)
```

```
   COMPILADOR  C-PASCAL VERSAO 1.0       PAGINA     6

[13732]        234  (*-------------------------------------------------
[13732]        235        P R O G R A M A      P R I N C I P A L
[13732]        236  -------------------------------------------------*)
[13732]        237  BEGIN
[13736]        238      INICIA ;
[13740]        239      FOR X1:=0 TO 10 DO VETBP[X1]:=-1;
[13804]        240      REPEAT
[13804]        241      WRITELN(PER0) ;
[13808]        242      WRITE(PER0,'CMD> ') ;
[13836]        243      READ(PER0,&CMD1,&CMD2) ;
[13852]        244      CASE CMD1 OF
[13860]        245      'E':BEGIN (* EX,EI,ER,EP:MODOS DE EXECUCAO *)
[13872]        246          READLN(PER0) ;
[13876]        247          FIM := 0 ;
[13884]        248          TRC := 1 ;
[13892]        249          CASE CMD2 OF
[13900]        250          'X': TRC := 0 ;                (* EX-EXECUTAR *)
[13928]        251          'I': BEGIN TRC:=0;INICIA END;(* EI-INICIAR  *)
[13960]        252          'R': BEGIN   (* ER-EXECUTAR COM RATREAMENTO *)
[13972]        253               REPEAT EXECUTA; CMD2:=CMD2+1
[13984]        254               UNTIL (FIM) OR (CMD2=97); FIM:=1
[14020]        255               END ;
[14032]        256          'P': FIM := 1  (* EP- EXECUTAR PASSO A PASSO *
[14048]        257          END;
[14056]        258          REPEAT EXECUTA UNTIL FIM
[14064]        259          END ;
[14076]        260      'I':BEGIN(* ISERIR(I+XXXX), RETIRAR (I-XXXX) *)
[14088]        261          READ(PER0,%CMD1) ;
[14096]        262          X2 := CMD1 ;
[14104]        263          READLN(PER0) ;
[14108]        264          X1 := 0 ;
[14116]        265          IF CMD2 = '+' THEN X2:=-1 ELSE CMD1:=-1 ;
[14160]        266          REPEAT X1:=X1+1 UNTIL(VETBP[X1]=X2)OR(X1=10);
[14212]        267          VETBP[X1] := CMD1
[14220]        268          END ;
[14232]        269      'L':BEGIN   (* LISTAR INSTRUCOES (LI) *)
[14244]        270          READLN(PER0) ;
[14248]        271          CMD1 := PC ;
[14256]        272          REPEAT
[14256]        273          FOR CMD2:=1 TO 16
[14264]        274          DO BEGIN CODIGO(CMD1);CMD1:=CMD1+4 END ;
[14328]        275          READLN(PER0,&X3)
[14340]        276          UNTIL X3 = '.'
[14348]        277          END ;
[14364]        278      'R':BEGIN   (* RASTREAMENTO DO PROGRAMA *)
[14376]        279          READLN(PER0) ;
[14380]        280          FOR CMD1:=1 TO 16
[14388]        281          DO CODIGO(VETTR[(PTR+CMD1)AND#000F])
[14440]        282          END ;
[14464]        283  (*$P+*)
```

```
   COMPILADOR  C-PASCAL VERSAO 1.0        PAGINA     7

[14464]        284       'S':BEGIN (* ST-STATUS DA MAQUINA VIRTUAL *)
[14476]        285           READLN(PER0) ;
[14480]        286           WRITELN(PER0,'BR=',%BR,' SP=',%SP,
[14540]        287                        ' IR=',%COD,'/',%IDX,
[14592]        288                        ' OR=',%VAL,' PC=',%PC) ;
[14660]        289           WRITELN(PER0,' * PILHA * ') ;
[14716]        290           FOR X1:=SP DOWNTO SP-2
[14728]        291           DO WRITELN(PER0,'    ',%STACK[X1]) ;
[14808]        292           WRITE(PER0,'INTE --> ') ;
[14852]        293           FOR X1:=1 TO 10
[14860]        294           DO WRITE(PER0,%VETBP[X1] ,';');
[14924]        295           WRITELN(PER0) ;
[14928]        296           CODIGO(PC)
[14936]        297           END ;
[14944]        298       'D':BEGIN    (* DUMP(DP) *)
[14956]        299           IF CMD2 = 'P'
[14964]        300           THEN BEGIN
[14972]        301                READLN(PER0,%X2) ;
[14984]        302                REPEAT
[14984]        303                FOR X1:=1 TO 16
[14992]        304                DO BEGIN
[15016]        305                   WRITE(PER0,%X2,'     ') ;
[15048]        306                   FOR X3 := 0 TO 7
[15056]        307                   DO BEGIN
[15080]        308                      X4 := X2 + X3 + X3 ;
[15104]        309                      WRITE(PER0,
[15104]        310                      %(MEM[X4] SHL 8 OR MEM[X4+1]),'  ')
[15160]        311                      END ;
[15176]        312                   FOR X2:=X2 TO X2+15
[15188]        313                   DO BEGIN
[15216]        314                      X3 := MEM[X2] ;
[15228]        315                      IF (X3<' ')OR(X3>#7A) THEN X3:='.'
[15268]        316                      WRITE(PER0,&X3)
[15276]        317                      END;
[15292]        318                   WRITELN(PER0)
[15296]        319                   END ;
[15312]        320                READLN(PER0,&X4)
[15324]        321                UNTIL X4 = '.'
[15332]        322                END
[15340]        323           ELSE BEGIN
[15344]        324                READLN(PER0) ;
[15348]        325                CMD1 := BASEPC ;
[15356]        326                REPEAT CODIGO(CMD1) ;
[15364]        327                       CMD1 := CMD1 + 4
[15372]        328                UNTIL  MEM[CMD1] = 255
[15392]        329                END
[15400]        330           END ;
[15408]        331       'T':BEGIN (* TI-TERMINO DA SIMULACAO *)
[15420]        332           FIM := 2
[15424]        333           END
[15428]        334       END
[15432]        335       UNTIL FIM = 2
[15440]        336   END .
```

L I S T A G E M II

```
   COMPILADOR  CPASCAL VERSAO 1.0

[10000]       1  PROGRAM ORDENACAO ;
[10004]       2  LABEL  1 ;
[10004]       3  CONST  PERO = 0 ;
[10004]       4  VAR    X,Y,I,J,K,CHAVE,TOTAL,VAL,CMD : INTEGER ;
[10004]       5          VETOR : ARRAY [1..100] OF INTEGER ;
[10004]       6  BEGIN
[10004]       7      TOTAL := 0 ;
[10012]       8      REPEAT WRITELN(PERO,'BUSCA OU ORDENACAO');
[10096]       9             WRITE(PERO,'CMD > ');
[10128]      10             READLN(PERO,&CMD) ;
[10140]      11             CASE CMD OF
[10144]      12             'B': BEGIN
[10160]      13                  WRITE(PERO,'CHAVE ?.>');
[10204]      14                  READLN(PERO,$VAL);
[10216]      15                  I := TOTAL ;
[10224]      16                  J := 1 ;
[10232]      17                  WHILE I>=J
[10240]      18                  DO BEGIN
[10248]      19                     K := (I+J) SHR 1 ;
[10272]      20                     CHAVE := VETOR [K] ;
[10284]      21                     IF CHAVE >VAL
[10292]      22                     THEN I=K-1
*******                                  ^ 12
*******                                  ^380
[10308]      23                     ELSE IF CHAVE < VAL
[10328]      24                          THEN J : K+1
*******                                       ^ 12
*******                                       ^380
[10344]      25                          ELSE BEGIN
[10356]      26                               WRITELN(PERO,'** SUCESSO **',
[10416]      27                               'REG[',$K,'] = ',$VAL);
[10484]      28                               GOTO 1
[10488]      29                               END
[10488]      30                     END ;
[10492]      31                     WRITELN(PERO,'BUSCA SEM SUCESSO',$VAL); 1
[10580]      32                  END ;
[10580]      33             'O': REPEAT I := 0 ;
[10608]      34                       FOR J:=1 TOTAL (* FALTA 'TO' *)
*******                                       ^ 48
*******                                       ^380
[10616]      35                       DO IF VETOR[J] > VETOR[J+1]
[10664]      36                          THEN BEGIN
[10672]      37                               I := ( J + 32 ;
*******                                            ^  6
*******                                            ^380
[10688]      38                               X := VETOR [J] ;
[10700]      39                               VETOR[J] := VETOR[J+1] ;
[10724]      40                               VETOR [J+1] := X
[10740]      41                               END
[10744]      42                  UNTIL I = 0
[10768]      43             END   (* END DO CASE *)
[10780]      44         UNTIL CMD = 'F'
[10788]      45  END .
```

```
   COMPILADOR  CPASCAL VERSAO 1.0

[10000]       1  PROGRAM ORDENACAO ;
[10004]       2  LABEL  1 ;
[10004]       3  CONST  PERO = 0 ;
[10004]       4  VAR    X,Y,I,J,K,CHAVE,TOTAL,VAL,CMD : INTEGER ;
[10004]       5         VETOR : ARRAY [1..100] OF INTEGER ;
[10004]       6  BEGIN
[10004]       7      TOTAL := 0 ;
[10012]       8      REPEAT WRITELN(PERO,'BUSCA OU ORDENACAO');
[10096]       9             WRITE(PERO,'CMD > ');
[10128]      10             READLN(PERO,&CMD) ;
[10140]      11             CASE CMD OF
[10144]      12             'B': BEGIN
[10160]      13                  WRITE(PERO,'CHAVE ?.>');
[10204]      14                  READLN(PERO,$VAL);
[10216]      15                  I := TOTAL ;
[10224]      16                  J := 1 ;
[10232]      17                  WHILE I>=J
[10240]      18                  DO BEGIN
[10248]      19                     K := (I+J) SHR 1 ;
[10272]      20                     CHAVE := VETOR [K] ;
[10284]      21                        CHAVE > VAL (* FALTA DE UM IF *)
*******                                    ^ 12
*******                                    ^370
*******                                       ^380
[10288]      22                     THEN I:=K-1
*******                              ^ 24
*******                              ^370
*******                                ^380
[10300]      23                     ELSE IF CHAVE < VAL
*******                              ^ 24
*******                              ^370
*******                                 ^380
[10316]      24                           THEN J := K+1
[10332]      25                           ELSE BEGIN
[10344]      26                                WRITELN(PERO,'** SUCESSO **',
[10404]      27                                'REG[',$K,'] = ',$VAL);
[10472]      28                                GOTO 1
[10476]      29                                END
[10476]      30                     END ;
[10480]      31                     WRITELN(PERO,'BUSCA SEM SUCESSO',$VAL); 1
[10568]      32                  END ;
[10568]      33             'O': REPEAT I := 0 ;
[10596]      34                       FOR J:=1 TO TOTAL
[10604]      35                       DO IF VETOR[J] > VETOR[J+1]
[10652]      36                          THEN BEGIN
[10660]      37                               I := 1;
[10668]      38                               X := VETOR [J] ;
[10680]      39                               VETOR[J] := VETOR[J+1] ;
[10704]      40                               VETOR [J+1] := X
[10720]      41                               END
[10724]      42                  UNTIL I = 0
[10748]      43               END   (* END DO CASE *)
[10760]      44         UNTIL CMD = 'F'
[10768]      45  END .
```

```
   COMPILADOR  CPASCAL VERSAO 1.0

[10000]       1  PROGRAM ORDENACAO ;
[10004]       2  LABEL  1 ;
[10004]       3  CONST  PERO = 0 ;
[10004]       4  VAR    X,Y,I,J,K,CHAVE,TOTAL,VAL,CMD : INTEGER ;
[10004]       5          VETOR : ARRAY [1..100] OF INTEGER ;
[10004]       6  BEGIN
[10004]       7      TOTAL := 0 ;
[10012]       8      REPEAT WRITELN(PERO,'BUSCA OU ORDENACAO');
[10096]       9             WRITE(PERO,'CMD > ');
[10128]      10             READLN(PERO,&CMD) ;
[10140]      11             CASE CMD OF
[10144]      12             'B': BEGIN
[10160]      13                  WRITE(PERO,'CHAVE ?.>');
[10204]      14                  READLN(PERO,$VAL);
[10216]      15                  I := TOTAL ;
[10224]      16                  J := 1 ;
[10232]      17                  WHILE I>=J
[10240]      18                  DO BEGIN
[10248]      19                     K := (I+J) SHR 1 ;
[10272]      20                     CHAVE := VETOR [K] ;
[10284]      21                     IF CHAVE >VAL
[10292]      22                          I:=K-1  (* FALTA DE 'THEN' *)
*******                              ^ 28
*******                              ^380
[10308]      23                     ELSE IF CHAVE < VAL
[10328]      24                          THEN J := K+1
[10344]      25                          ELSE BEGIN
[10356]      26                               WRITELN(PERO,'** SUCESSO **',
[10416]      27                               'REG[',$K,'] = ',$VAL);
[10484]      28                               GOTO 1
[10488]      29                               END
[10488]      30                     END ;
[10492]      31                     WRITELN(PERO,'BUSCA SEM SUCESSO',$VAL); 1
[10580]      32                  END ;
[10580]      33             'O': REPEAT I := 0 ;
[10608]      34                       FOR J:=1 TO TOTAL
[10616]      35                       DO IF VETOR[J] > VETOR[J+1]
[10664]      36                          THEN BEGIN
[10672]      37                               I := ( J + 32 ;
*******                                           ^  6
*******                                           ^380
[10688]      38                               X := VETOR [J] ;
[10700]      39                               VETOR[J] := VETOR[J+1] ;
[10724]      40                               VETOR [J+1] := X
[10740]      41                               END
[10744]      42                  UNTIL I = 0
[10768]      43               END   (* END DO CASE *)
[10780]      44          UNTIL CMD = 'F'
[10788]      45  (*END*) .
*******                 ^ 24
*******                 ^380
*******  COMPILACAO ABORTADA  ***** ERRO (344)
```

```
   COMPILADOR  CPASCAL VERSAO 1.0

[10000]       1  (* ERRO DE IDENTIFICADOR NAO DECLARADO *)
[10000]       2  PROGRAM THANOI ;
[10004]       3
[10004]       4    CONST PERO = 0 ;
[10004]       5          ORIGEM = 1 ;
[10004]       6          DESTINO = 3 ;
[10004]       7          AUXILIAR = 2 ;
[10004]       8
[10004]       9    PROCEDURE TROCATORRE(ALTURA,TORI,TDES,TAUX:INTEGER) ;
[10004]      10
[10004]      11      PROCEDURE MOVEDISCO (RETIRAR,COLOCAR:INTEGER) ;
[10004]      12      BEGIN
[10004]      13          WRITELN(PERO,%RETIRAR,' ==> ',%COLOCAR)
[10052]      14      END ;
[10056]      15
[10056]      16    BEGIN      (* PROCEDURE TROCATORRE *)
[10056]      17        IF ALTURA > 0
[10064]      18        THEN BEGIN
[10072]      19             TROCATORRE [ALTURA-1,TORI,TAUX,TDES)
*******                              ^  1
*******                              ^380
[10100]      20             MOVEDISCO (TORI,TDES) ;
*******                           ^ 24
*******                           ^380
[10112]      21             TROCATORRE (ALTURA-1,TAUX,TDES,TORI) ;
[10140]      22             END ;
[10140]      23    END ;
[10144]      24
[10144]      25  BEGIN  (* PROGRAMA PRINCIPAL *)
[10144]      26
[10144]      27      WRITE(PERO,'NUMERO DE DISCOS NA ORIGEM ?> ') ;
[10272]      28      READLN(PERO,%NUMDISCOS) ;
*******                                ^306
[10284]      29      TROCATORRE (NUMDISCOS,ORIGEM,DESTINO,AUXILIAR]
*******                                                        ^ 52
*******                                                        ^380
[10304]      30
[10304]      31  END .
```

```
   COMPILADOR  CPASCAL VERSAO 1.0

[10000]       1  PROGRAM ORDENACAO ;
[10004]       2  LABEL  1 ;
[10004]       3  CONST  PERO = 0 ;
[10004]       4  VAR    X,Y,I,J,K,CHAVE,TOTAL,VAL,CMD : INTEGER ;
[10004]       5         VETOR : ARRAY [1.,100] OF INTEGER ;
[10004]       6  BEGIN
[10004]       7      TOTAL := 0 ;
[10012]       8      REPEAT WRITELN(PERO,'BUSCA OU ORDENACAO');
[10096]       9             WRITE(PERO,'CMD > ');
[10128]      10             READLN(PERO,&CMD) ;
[10140]      11             CASE CMD OF
[10144]      12             'B': BEGIN
[10160]      13                  WRITE(PERO,'CHAVE ?.>');
[10204]      14                  READLN(PERO,$VAL);
[10216]      15                  I  = TOTAL ; (* TROCA DE := POR = *)
*******                           ^ 12
*******                           ^380
[10224]      16                  J ;= 1 ;(* ERRO DE DIGITACAO ;/: *)
*******                          ^ 12
*******                          ^370
*******                           ^380
[10232]      17                  WHILE I>=J
[10240]      18                     BEGIN   (* FALTA UM  'DO' *)
*******                                ^ 30
*******                                ^380
[10248]      19                     K := (I+J) SHR 1 ;
[10272]      20                     CHAVE := VETOR [K] ;
[10284]      21                     IF CHAVE > VAL
[10292]      22                     THEN I:=K-1
[10308]      23                     ELSE IF CHAVE < VAL
[10328]      24                          THEN J := K+1
[10344]      25                          ELSE BEGIN
[10356]      26                               WRITELN(PERO,'** SUCESSO **',
[10416]      27                               'REG[',$K,'] = ',$VAL);
[10484]      28                               GOTO 1
[10488]      29                               END
[10488]      30                     END ;
[10492]      31                     WRITELN(PERO,'BUSCA SEM SUCESSO',$VAL); 1
[10580]      32                  END ;
[10580]      33             'O': REPEAT I := 0 ;
[10608]      34                       FOR J:=1 TO TOTAL
[10616]      35                       DO IF VETOR[J] > VETOR[J+1]
[10664]      36                          THEN BEGIN
[10672]      37                               I := 1;
[10680]      38                               X := VETOR [J] ;
[10692]      39                               VETOR[J) := VETOR[J+1]
*******                                        ^ 16
*******                                        ^380
[10712]      40                               VETOR (J+1] := X
*******                                     ^ 24
*******                                     ^380
*******                                       ^  2
*******                                       ^380
[10732]      41                               END
[10736]      42                  UNTIL I = 0
[10760]      43             END   (* END DO CASE *)
[10772]      44         UNTIL CMD = 'F'
[10780]      45  END .
```

L I S T A G E M V.1

```
-----------------------------------#
  CARGA DAS ROTINAS                #
     C-PASCAL                      #
-----------------------------------#

-IROTPAS.COM
-R
NEXT   PC
2600 0100
-----------------------------------#
  CARGA DO COMPILADOR              #
      C-PASCAL                     #
-----------------------------------#

-ICOMPAS.COM
-R500
NEXT   PC
2600 0100
-----------------------------------#
  CARGA DO PROGRAMA                #
    TORRE DE HANOI                 #
-----------------------------------#

-ITHANOI.CPA
-R2600
NEXT   PC
2C00 0100
-SF6FB
F6FB CD C3
2600   0001   PROGRAM TORREDEHANOI ;
2604   0002
2604   0003     CONST PERO = 0 ;
2604   0004           ORIGEM = 1 ;
2604   0005           DESTINO = 3 ;
2604   0006           AUXILIAR = 2 ;
2604   0007     VAR   NUMDISCOS : INTEGER ;
2604   0008
2604   0009     PROCEDURE TROCATORRE(ALTURA,TORI,TDES,TAUX:INTEGER) ;
2604   000A
2604   000B       PROCEDURE MOVEDISCO (RETIRAR,COLOCAR:INTEGER) ;
2604   000C       BEGIN
2604   000D           WRITELN(PERO,%RETIRAR,' ==> ',%COLOCAR)
2634   000E       END ;
2638   000F
2638   0010     BEGIN      (* PROCEDURE TROCATORRE *)
2638   0011         IF ALTURA > 0
2640   0012         THEN BEGIN
2648   0013              TROCATORRE (ALTURA-1,TORI,TAUX,TDES) ;
2664   0014              MOVEDISCO (TORI,TDES) ;
2670   0015              TROCATORRE (ALTURA-1,TAUX,TDES,TORI)
268C   0016              END
268C   0017     END ;
2690   0018
2690   0019   BEGIN  (* PROGRAMA PRINCIPAL *)
2694   001A
2694   001B       WRITE(PERO,'NUMERO DE DISCOS NA ORIGEM ?> ') ;
2714   001C       READLN(PERO,%NUMDISCOS) ;
2720   001D       TROCATORRE (NUMDISCOS,ORIGEM,DESTINO,AUXILIAR)
2734   001E
2734   001F   END .
```

L I S T A G E M V.2

```
-----------------------------------#
  CARGA DO INTERPRETADOR           #
       C-PASCAL                    #
-----------------------------------#

-IINTPAS.COM
-R500
NEXT  PC
2B00 0161
-#
-----------------------------------#
.EXECUTAR A INTERPRETACAO          #
.DESVIAR PARA END. 600H            #
-----------------------------------#

-G600

CMD> DC
2600 --> GTO   0000 / 2690
2604 --> LOD   0000 / FFFE
2608 --> RES   HEX
260C --> RES   STR
2610 --> LCT   00005  0005
2614 --> LCT   00032  0020
2618 --> LCT   00061  003D
261C --> LCT   00061  003D
2620 --> LCT   00062  003E
2624 --> LCT   00032  0020
2628 --> LOD   0000 / FFFF
262C --> RES   HEX
2630 --> RES   CRL
2634 --> RET   0002 / 0000
2638 --> LOD   0000 / FFFC
263C --> LCT   00000  0000
2640 --> OPE   GTR
2644 --> GIF   0000 / 268C
2648 --> LOD   0000 / FFFC
264C --> LCT   00001  0001
2650 --> OPE   SUB
2654 --> LOD   0000 / FFFD
2658 --> LOD   0000 / FFFF
265C --> LOD   0000 / FFFE
2660 --> GSB   0001 / 2638
2664 --> LOD   0000 / FFFD
2668 --> LOD   0000 / FFFE
266C --> GSB   0000 / 2604
2670 --> LOD   0000 / FFFC
2674 --> LCT   00001  0001
2678 --> OPE   SUB
267C --> LOD   0000 / FFFF
2680 --> LOD   0000 / FFFE
2684 --> LOD   0000 / FFFD
2688 --> GSB   0001 / 2638
268C --> RET   0004 / 0000
2690 --> OPT   00001  0001
2694 --> RES   STR
2698 --> LCT   00030  001E
269C --> LCT   00078  004E
26A0 --> LCT   00085  0055
26A4 --> LCT   00077  004D
26A8 --> LCT   00069  0045
26AC --> LCT   00082  0052
26B0 --> LCT   00079  004F
```

```
26B4 --> LCT    00032  0020
26B8 --> LCT    00068  0044
26BC --> LCT    00069  0045
26C0 --> LCT    00032  0020
26C4 --> LCT    00068  0044
26C8 --> LCT    00073  0049
26CC --> LCT    00083  0053
26D0 --> LCT    00067  0043
26D4 --> LCT    00079  004F
26D8 --> LCT    00083  0053
26DC --> LCT    00032  0020
26E0 --> LCT    00078  004E
26E4 --> LCT    00065  0041
26E8 --> LCT    00032  0020
26EC --> LCT    00079  004F
26F0 --> LCT    00082  0052
26F4 --> LCT    00073  0049
26F8 --> LCT    00071  0047
26FC --> LCT    00069  0045
2700 --> LCT    00077  004D
2704 --> LCT    00032  0020
2708 --> LCT    00063  003F
270C --> LCT    00062  003E
2710 --> LCT    00032  0020
2714 --> RES    HEX
2718 --> STO    00FF / 0003
271C --> RES    CRL
2720 --> LOD    00FF / 0003
2724 --> LCT    00001  0001
2728 --> LCT    00003  0003
272C --> LCT    00002  0002
2730 --> GSB    0000 / 2638
2734 --> RET    00FF / 0000

CMD> ST
BR=0000 SP=0002 IR=0006/0000 OR=0000 PC=2600
 * PILHA *
   FFFF
   0000
   0000
INTE --> FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2600 --> GTO    0000 / 2690

CMD> I+2720

CMD> I+2604

CMD> I+2638

CMD> ST
BR=0000 SP=0002 IR=0007/0000 OR=2690 PC=2600
 * PILHA *
   FFFF
   0000
   0000
INTE --> 2720;2604;2638;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2600 --> GTO    0000 / 2690

CMD> EX
NUMERO DE DISCOS NA ORIGEM ?> 0003
 PAUSA

CMD>
```

```
CMD> ST
BR=0000 SP=0007 IR=000A/0000 OR=0000 PC=2720
 * PILHA *
   C172
   0003
   C172
INTE --> 2720;2604;2638;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2720 --> LOD   00FF / 0003

CMD> RP
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2600 --> GTO   0000 / 2690
2690 --> OPT   00001  0001
2694 --> RES   STR
2714 --> RES   HEX
2718 --> STO   00FF / 0003
271C --> RES   CRL

CMD> EX
 PAUSA

CMD> EX
 PAUSA

CMD> RP
2720 --> LOD   00FF / 0003
2724 --> LCT   00001  0001
2728 --> LCT   00003  0003
272C --> LCT   00002  0002
2730 --> GSB   0000 / 2638
2638 --> LOD   0000 / FFFC
263C --> LCT   00000  0000
2640 --> OPE   GTR
2644 --> GIF   0000 / 268C
2648 --> LOD   0000 / FFFC
264C --> LCT   00001  0001
2650 --> OPE   SUB
2654 --> LOD   0000 / FFFD
2658 --> LOD   0000 / FFFF
265C --> LOD   0000 / FFFE
2660 --> GSB   0001 / 2638

CMD> ST
BR=0013 SP=0015 IR=0005/0000 OR=2638 PC=2638
 * PILHA *
   2664
   000C
   0000
INTE --> 2720;2604;2638;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2638 --> LOD   0000 / FFFC

CMD> EP
2638 --> LOD   0000 / FFFC
```

```
CMD> EP
263C --> LCT    00000  0000

CMD> EX
 PAUSA

CMD> EX
 PAUSA

CMD> ST
BR=0021 SP=0023 IR=0005/0000 OR=2638 PC=2638
 * PILHA *
   2664
   001A
   0000
INTE --> 2720;2604;2638;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2638 --> LOD    0000 / FFFC

CMD> I-2720

CMD> I-2604

CMD> I-2638

CMD> ST
BR=0021 SP=0023 IR=0001/0000 OR=FFFC PC=2638
 * PILHA *
   2664
   001A
   0000
INTE --> FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2638 --> LOD    0000 / FFFC

CMD> EX
0001 ==> 0003
0001 ==> 0002
0003 ==> 0002
0001 ==> 0003
0002 ==> 0001
0002 ==> 0003
0001 ==> 0003
SUCESSO

CMD> RP
2670 --> LOD    0000 / FFFC
2674 --> LCT    00001  0001
2678 --> OPE    SUB
267C --> LOD    0000 / FFFF
2680 --> LOD    0000 / FFFE
2684 --> LOD    0000 / FFFD
2688 --> GSB    0001 / 2638
2638 --> LOD    0000 / FFFC
263C --> LCT    00000  0000
2640 --> OPE    GTR
2644 --> GIF    0000 / 268C
268C --> RET    0004 / 0000
268C --> RET    0004 / 0000
268C --> RET    0004 / 0000
268C --> RET    0004 / 0000
2734 --> RET    00FF / 0000

CMD>
CMD>
```

```
CMD> EI
NUMERO DE DISCOS NA ORIGEM ?> 0001
0001 ==> 0003
SUCESSO

CMD> EI
NUMERO DE DISCOS NA ORIGEM ?> 0002
0001 ==> 0002
0001 ==> 0003
0002 ==> 0003
SUCESSO

CMD> EI
NUMERO DE DISCOS NA ORIGEM ?> 0003
0001 ==> 0003
0001 ==> 0002
0003 ==> 0002
0001 ==> 0003
0002 ==> 0001
0002 ==> 0003
0001 ==> 0003
SUCESSO

CMD> I+2690

CMD> ST
BR=0000 SP=FF00 IR=0006/0000 OR=0000 PC=0000
 * PILHA *
   4F43
   5221
   5445
INTE --> 2690;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
0000 --> ≡MZX   0003 / 01F6

CMD> EI
 PAUSA

CMD> ST
BR=0000 SP=0002 IR=0007/0000 OR=2690 PC=2690
 * PILHA *
   FFFF
   0000
   0000
INTE --> 2690;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2690 --> OPT    00001   0001

CMD> EP
2690 --> OPT    00001   0001

CMD> EP
2694 --> RES    STR
NUMERO DE DISCOS NA ORIGEM ?>
CMD> EP
2714 --> RES    HEX
0001
CMD> ST
BR=0000 SP=0006 IR=000A/0000 OR=0000 PC=2718
 * PILHA *
   0001
   0003
   C17F
INTE --> 2690;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2718 --> STO    00FF / 0003
```

```
CMD> I+2734

CMD> I+2604

CMD> EI
 PAUSA

CMD> EX
NUMERO DE DISCOS NA ORIGEM ?> 0003
 PAUSA

CMD> EX
0001 ==> 0003
 PAUSA

CMD> ST
BR=0018 SP=001A IR=0005/0000 OR=2604 PC=2604
 * PILHA *
   2670
   0013
   0013
INTE --> 2690;2734;2604;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2604 --> LOD   0000 / FFFE

CMD> EX
0001 ==> 0002
 PAUSA

CMD> ST
BR=001F SP=0021 IR=0005/0000 OR=2604 PC=2604
 * PILHA *
   2670
   001A
   001A
INTE --> 2690;2734;2604;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2604 --> LOD   0000 / FFFE

CMD> EX
0003 ==> 0002
 PAUSA

CMD> EX
0001 ==> 0003
 PAUSA

CMD> EX
0002 ==> 0001
 PAUSA

CMD> EX
0002 ==> 0003
 PAUSA

CMD> EX
0001 ==> 0003
 PAUSA

CMD> EX
SUCESSO

CMD>
CMD>
```

```
CMD> ST
BR=0000 SP=FF00 IR=0006/0000 OR=0000 PC=0000
 * PILHA *
   4F43
   5221
   5445
INTE --> 2690;2734;2604;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
0000 --> EMZX  0003 / 01F6

CMD> I-2734

CMD> I-2604

CMD> I+2720

CMD> ST
BR=0000 SP=FF00 IR=00B3/0000 OR=01F6 PC=0000
 * PILHA *
   4F43
   5221
   5445
INTE --> 2690;2720;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
0000 --> EMZX  0003 / 01F6

CMD> EI
 PAUSA

CMD> EX
NUMERO DE DISCOS NA ORIGEM ?> 0001
 PAUSA

CMD> ER
2720 --> LOD   00FF / 0003
2724 --> LCT   00001  0001
2728 --> LCT   00003  0003
272C --> LCT   00002  0002
2730 --> GSB   0000 / 2638
2638 --> LOD   0000 / FFFC
263C --> LCT   00000  0000
2640 --> OPE   GTR
2644 --> GIF   0000 / 268C
2648 --> LOD   0000 / FFFC
264C --> LCT   00001  0001
2650 --> OPE   SUB
2654 --> LOD   0000 / FFFD
2658 --> LOD   0000 / FFFF
265C --> LOD   0000 / FFFE
2660 --> GSB   0001 / 2638

CMD> ST
BR=0013 SP=0015 IR=0005/0000 OR=2638 PC=2638
 * PILHA *
   2664
   000C
   0000
INTE --> 2690;2720;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;FFFF;
2638 --> LOD   0000 / FFFC

CMD>
CMD>
CMD>
CMD>
CMD>
```

```
CMD> DP2600
2600    0700  9026  0100  FEFF  0A05  0000  0A03  0000  ...&............
2610    0000  0500  0000  2000  0000  3D00  0000  3D00  ..... ...=...=.
2620    0000  3E00  0000  2000  0100  FFFF  0A05  0000  ..>... ........
2630    0A08  0000  0602  0000  0100  FCFF  0000  0000  ................
2640    090C  0000  0800  8C26  0100  FCFF  0000  0100  .......&........
2650    090F  0000  0100  FDFF  0100  FFFF  0100  FEFF  ................
2660    0501  3826  0100  FDFF  0100  FEFF  0500  0426  ..8&...........&
2670    0100  FCFF  0000  0100  090F  0000  0100  FFFF  ................
2680    0100  FEFF  0100  FDFF  0501  3826  0604  0000  ..........8&....
2690    0B00  0100  0A03  0000  0000  1E00  0000  4E00  ..............N.
26A0    0000  5500  0000  4D00  0000  4500  0000  5200  ..U...M...E...R.
26B0    0000  4F00  0000  2000  0000  4400  0000  4500  ..O... ...D...E.
26C0    0000  2000  0000  4400  0000  4900  0000  5300  .. ...D...I...S.
26D0    0000  4300  0000  4F00  0000  5300  0000  2000  ..C...O...S... .
26E0    0000  4E00  0000  4100  0000  2000  0000  4F00  ..N...A... ...O.
26F0    0000  5200  0000  4900  0000  4700  0000  4500  ..R...I...G...E.

2700    0000  4D00  0000  2000  0000  3F00  0000  3E00  ..M... ...?...>.
2710    0000  2000  0A01  0000  03FF  0300  0A07  0000  .. ............
2720    01FF  0300  0000  0100  0000  0300  0000  0200  ................
2730    0500  3826  06FF  0000  FF00  0000  2020  2020  ..8&........
2740    2020  2020  2020  4F52  4947  454D  203D  2031        ORIGEM = 1
2750    203B  0D0A  2020  2020  2020  2020  2020  4445   ;..            DE
2760    5354  494E  4F20  3D20  3320  3B0D  0A20  2020  STINO = 3 ;..
2770    2020  2020  2020  2041  5558  494C  4941  5220          AUXILIAR
2780    3D20  3220  3B0D  0A20  2020  2056  4152  0D0A  = 2 ;..     VAR..
2790    2020  2020  2020  2020  2020  4E55  4D44  4953              NUMDIS
27A0    434F  5320  3A20  494E  5445  4745  5220  3B0D  COS : INTEGER ;.
27B0    0A20  200D  0A20  2020  2050  524F  4345  4455  .  ..     PROCEDU
27C0    5245  2054  524F  4341  544F  5252  4520  2841  RE TROCATORRE (A
27D0    4C54  5552  412C  544F  5249  4745  4D2C  5444  LTURA,TORIGEM,TI
27E0    4553  5449  4E4F  2C54  4155  5849  4C49  4152  ESTINO,TAUXILIAR
27F0    203A  2049  4E54  4547  4552  2920  3B0D  0A20   : INTEGER) ;..

2800    2020  200D  0A20  2020  2020  2020  2050  524F     ..          PRO
2810    4345  4455  5245  204D  4F56  4544  4953  434F  CEDURE MOVEDISCO
2820    2028  5245  5449  5241  522C  434F  4C4F  4341   (RETIRAR,COLOCA
2830    5220  3A20  494E  5445  4745  5229  203B  0D0A  R : INTEGER) ;..
2840    200D  0A20  2020  2020  2020  2042  4547  494E   ..          BEGIN
2850    200D  0A20  2020  2020  2020  2020  2020  2057   ..              W
2860    5249  5445  4C4E  2850  4552  302C  2552  4554  RITELN(PERO,%RET
2870    4952  4152  2C27  203D  3D3E  2027  2C25  434F  IRAR,' ==> ',%CO
2880    4C4F  4341  5229  0D0A  2020  2020  2020  2020  LOCAR)..
2890    454E  4420  3B0D  0A20  0D0A  2020  2020  4245  END ;.. ..     BE
28A0    4749  4E0D  0A20  2020  2020  2020  2049  4620  GIN..         IF
28B0    414C  5455  5241  203E  2030  0D0A  2020  2020  ALTURA > 0..
28C0    2020  2020  5448  454E  2042  4547  494E  0D0A      THEN BEGIN..
28D0    2020  2020  2020  2020  2020  2020  2054  524F                  TRO
28E0    4341  544F  5252  4520  2841  4C54  5552  412D  CATORRE (ALTURA-
28F0    312C  544F  5249  4745  4D2C  5441  5558  494C  1,TORIGEM,TAUXIL
.

CMD> TI      **********FIM*0161
```

L I S T A G E M V.3

```
-----------------------------------------#
     CARGA DO TRADUTOR                   #
         C-PASCAL                        #
                                         #
.VERSAO SIMPLIFICADA COM OS              #
.VALORES DOS PARAMETROS DE               #
.TRADUCAO FIXOS                          #
-----------------------------------------#

-ITADPAS.COM
-R500
NEXT   PC
2B00 0161
-G600
25D0
      **********FIM*0161
-#
-----------------------------------------#
.O CODIGO 8085 OCUPA AS                  #
.POSICOES DE 2500 A 25D0                 #
-----------------------------------------#

-EXECUTAR O PROGRAMA #

-G2500
NUMERO DE DISCOS NA ORIGEM ?> 0003
0001 ==> 0003
0001 ==> 0002
0003 ==> 0002
0001 ==> 0003
0002 ==> 0001
0002 ==> 0003
0001 ==> 0003
      **********FIM*0161
-G2500
NUMERO DE DISCOS NA ORIGEM ?> 0002
0001 ==> 0002
0001 ==> 0003
0002 ==> 0003
      **********FIM*0161
-G2500
NUMERO DE DISCOS NA ORIGEM ?> 0001
0001 ==> 0003
      **********FIM*0161
```

```
 1                                  ****************************
 2                                  *
 3                                  *   CODIGO 8085 DO PROGRAMA
 4                                  *
 5                                  ****************************
 6
 7                        0000 00000            ORG        2500H
 8
 9  21 00 25              2500 09472            LXI  H,    2500H
10  22 00 01              2503 09475            SHLD       0100H
11  22 02 01              2506 09478            SHLD       0102H
12  F9                    2509 09481            SPHL
13  E5                    250A 09482            PUSH H
14  E5                    250B 09483            PUSH H
15  C3 88 25              250C 09484            JMP        9608
16  CD 18 04              250F 09487            CALL       1048
17  04 00                 2512 09490            DW            4
18  CD 63 03              2514 09492            CALL        867
19  00                    2517 09495            DB            0
20  CD 10 03              2518 09496            CALL        784
21  00                    251B 09499            DB            0
22  05                    251C 09500            DB            5
23  20                    251D 09501            DB           32
24  3D                    251E 09502            DB           61
25  3D                    251F 09503            DB           61
26  3E                    2520 09504            DB           62
27  20                    2521 09505            DB           32
28  CD 18 04              2522 09506            CALL       1048
29  02 00                 2525 09509            DW            2
30  CD 63 03              2527 09511            CALL        867
31  00                    252A 09514            DB            0
32  CD 96 03              252B 09515            CALL        918
33  00                    252E 09518            DB            0
34  CD 9D 02              252F 09519            CALL        669
35  06                    2532 09522            DB            6
36  CD 18 04              2533 09523            CALL       1048
37  08 00                 2536 09526            DW            8
38  CD 00 04              2538 09528            CALL       1024
39  00 00                 253B 09531            DW            0
40  84 25                 253D 09533            DW         9604
41  CD 18 04              253F 09535            CALL       1048
42  08 00                 2542 09538            DW            8
43  2B                    2544 09540            DCX  H
44  CD 18 04              2545 09541            CALL       1048
45  06 00                 2548 09544            DW            6
46  CD 18 04              254A 09546            CALL       1048
47  02 00                 254D 09549            DW            2
48  CD 18 04              254F 09551            CALL       1048
49  04 00                 2552 09554            DW            4
50  CD BC 03              2554 09556            CALL        956
51  01                    2557 09559            DB            1
52  33 25                 2558 09560            DW         9523
53  CD 18 04              255A 09562            CALL       1048
54  06 00                 255D 09565            DW            6
55  CD 18 04              255F 09567            CALL       1048
56  04 00                 2562 09570            DW            4
57  CD 6D 04              2564 09572            CALL       1133
58  0F 25                 2567 09575            DW         9487
```

```
 59  CD 18 04      2569 09577      CALL      1048
 60  08 00         256C 09580      DW           8
 61  2B            256E 09582      DCX  H
 62  CD 18 04      256F 09583      CALL      1048
 63  02 00         2572 09586      DW           2
 64  CD 18 04      2574 09588      CALL      1048
 65  04 00         2577 09591      DW           4
 66  CD 18 04      2579 09593      CALL      1048
 67  06 00         257C 09596      DW           6
 68  CD BC 03      257E 09598      CALL       956
 69  01            2581 09601      DB           1
 70  33 25         2582 09602      DW        9523
 71  CD 9D 02      2584 09604      CALL       669
 72  0A            2587 09607      DB          10
 73  E5            2588 09608      PUSH H
 74  CD 10 03      2589 09609      CALL       784
 75  00            258C 09612      DB           0
 76  1E            258D 09613      DB          30
 77  4E            258E 09614      DB          78
 78  55            258F 09615      DB          85
 79  4D            2590 09616      DB          77
 80  45            2591 09617      DB          69
 81  52            2592 09618      DB          82
 82  4F            2593 09619      DB          79
 83  20            2594 09620      DB          32
 84  44            2595 09621      DB          68
 85  45            2596 09622      DB          69
 86  20            2597 09623      DB          32
 87  44            2598 09624      DB          68
 88  49            2599 09625      DB          73
 89  53            259A 09626      DB          83
 90  43            259B 09627      DB          67
 91  4F            259C 09628      DB          79
 92  53            259D 09629      DB          83
 93  20            259E 09630      DB          32
 94  4E            259F 09631      DB          78
 95  41            25A0 09632      DB          65
 96  20            25A1 09633      DB          32
 97  4F            25A2 09634      DB          79
 98  52            25A3 09635      DB          82
 99  49            25A4 09636      DB          73
100  47            25A5 09637      DB          71
101  45            25A6 09638      DB          69
102  4D            25A7 09639      DB          77
103  20            25A8 09640      DB          32
104  3F            25A9 09641      DB          63
105  3E            25AA 09642      DB          62
106  20            25AB 09643      DB          32
107  CD DF 02      25AC 09644      CALL       735
108  00            25AF 09647      DB           0
109  22 FA 24      25B0 09648      SHLD      9466
110  E1            25B3 09651      POP  H
111  CD 8F 03      25B4 09652      CALL       911
112  00            25B7 09655      DB           0
113  E5            25B8 09656      PUSH H
114  2A FA 24      25B9 09657      LHLD      9466
115  E5            25BC 09660      PUSH H
116  21 01 00      25BD 09661      LXI  H,      1
```

```
117  E5          25C0 09664          PUSH H
118  21 03 00    25C1 09665          LXI  H,     3
119  E5          25C4 09668          PUSH H
120  21 02 00    25C5 09669          LXI  H,     2
121  CD 6D 04    25C8 09672          CALL     1133
122  33 25       25CB 09675          DW       9523
123  CD 3C 01    25CD 09677          CALL      316

ASM -- NAO FOI DETETADO ERRO NO PROGRAMA
```

MENSAGEM DE ERRO

CÓDIGO		MENSAGEM
Ø	: '['	Esperado no subscrito de uma variável < expresão >
1	: '('	esperado na lista de parâmetros de uma função
2	: '['	esperado no subscrito de uma variável na < atribuição >
3	: ') '	esperado na lista de parâmetros de um procedimento
4	: '] '	esperado no subscrito de uma variável na < expressão >
5	: ') '	esperado na lista de parâmetros de uma função
6	: ') '	esperado
8	:	chamada de um procedimento dentro de expressão
9	:	fator ilegal dentro de expressão
12	: ':='	esperado no comando de atribuição
13	: ':='	esperado no comando 'FOR'
16	: '] '	esperado no subscrito de uma variável na < atribuição >
2Ø	:	IDENTIFICADOR de função ou constante no início de comando
24	: ';'	ou 'END' esperado no comando composto
28	:	'THEN' esperado
29	: ':'	esperado na lista de rótulos do 'CASE'
3Ø	: 'DO'	esperado no comando 'FOR'
32	: 'OF'	esperado no comando 'CASE'
36	: 'END'	esperado no comando 'CASE'
37	:	INTEIRO esperado no comando 'GOTO'
40	:	';' ou 'UNTIL' esperado no comando 'REPEAT'
44	:	IDENTIFICADOR ilegal na atribuição do 'FOR'
48	:	'TO/DOWNTO' esperado no comando 'FOR'

CÓDIGO		MENSAGEM
3ØØ	:	Transbordo na área reservada para código
3Ø1	:	Transbordo na área reservada para cadeias
3Ø2	:	Transbordo no poço de identificados
3Ø3	:	Transbordo na tabela de símbolos
3Ø4	:	Transbordo no vetor de rótulos do 'GOTO'
3Ø6	:	Identificador no declarado
312	:	Símbolo ilegal no C-PASCAL
314	:	Declaração 'LABEL' fora de ordem
315	:	Declaração 'CONST' fora de ordem
316	:	Declaração 'VAR' fora de ordem
317	:	Declaração 'PROCEDURE' ou 'FUNCTION' dentro dos comandos
318	:	Transbordo na constante inteira
319	:	Caracteres hexadecimais ilegais
344	:	Final inesperado do programa
346	:	Formato ilegal de entrada/saída
347	:	Declaração 'PROGRAM' fora dos padrões
36Ø	:	Número de erros acima do limite
37Ø	:	Símbolo desprezado
380	:	Assumimos o símbolo esperado (troca ou inserção)

CLASSIFICAÇÃO DOS 'TOKENS'

CÓDIGO HEXADECIMA	'TOKEN'
ØØ	< símbolo ilegal >
1Ø	< identificador >
2Ø	< número INTEIRO >
	< blocagem >
3Ø	'
31	##
32	(
33	)
34	|
35	|
36	:
37	;
38	;
39	.
3A	END
3B	..
	< declarações >
4Ø	PROGRAM
41	LABEL
42	CONST
43	TYPE
44	VAR
45	PROCEDURE
46	FUNCTION
47	BEGIN
	< tipo >
5Ø	PACKED
51	ARRAY
52	INTEGER
53	REAL
54	CHAR
55	BOOLEAN
56	RECORD

CÓDIGO HEXADECIMAL	'TOKEN'
57	FILE
58	SET
	< OPE ADIÇÃO >
6Ø	OR
61	SUB (-)
62	ADD (+)
	< OPE NEGAÇÃO >
7Ø	NOT
	< OPE RELACIONAL >
8Ø	=
81	< >
82	<
83	> =
84	>
85	< =
86	EQ
87	NE
88	LS
89	GE
8A	GT
8B	LE
	< OPE MULTIPLICAÇÃO >
9Ø	MUL (*)
91	DIV (/)
92	MOD
93	SHL
94	SHR
95	AND
	< comandos >
AØ	IF
A1	THEN
A2	ELSE
A3	CASE
A4	OF
A5	OTHERS
A6	GOTO

CÓDIGO HEXADECIMAL	'TOKEN'
A7	:=
A8	WHILE
A9	DO
AA	REPEAT
AB	UNTIL
AC	FOR
AD	TO
AE	DOUNTO
	< procedimentos >
BØ	MEM
B1	CALL
B"	READ
B3	READLN
B4	WRITE
B5	WRITELN
	< formatos >
CØ	$ decimal
C1	% hexadecimal
C2	& ASCII