"ESTUDO DE UM SUPERVISOR PARA UM INTERPRETADOR COBOL"

Bruno Correia da Nôbrega Queiroz

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:

Martin A Diamond
Presidente

RIO DE JANEIRO
ESTADO DA GUANABARA - BRASIL
AGOSTO DE 1974

Aos meus pais .

AGRADECIMENTOS

- Ao Professor Martin A. Diamond pela paciente orientação dada a esse trabalho.

- Aos funcionários da COPPE e da Biblioteca da UFRJ.

- A Luis Nóbrega de Queiroz pelo apoio e colaboração durante minha permanência no Rio de Janeiro.

SUMÁRIO

Esse trabalho é um estudo de um supervisor para um interpretador COBOL, que tem por objetivo utilizar pouco espaço de memória principal.

Consta ainda de uma descrição da linguagem utilizada, da forma interna das instruções e dos dados com todas as tabelas que serão utilizadas pelo sistema.

ABSTRACT

This work consists of the design of the supervisor module for a COBOL interpreter.

The main goal of this interpreter is to use small ammounts of main memory so that it can be implemented on a Mini-computer.

A description of the language, the internal form of instructions and data and the data structure used is also presented.

INDICE

CAPÍTULO I

INTRODUÇÃO

O objetivo deste trabalho é desenvolver um interpretador COBOL para Mini-Computadores. Logo depois de sua criação e posterior divulgação, o COBOL passou a ser sem nenhuma dúvida a mais importante e mais utilizada linguagem de programação para fins comerciais. Segundo a revista francesa L'INFORMATIQUE em uma enquete recentemente levada a efeito nos Estados Unidos, na qual foram visitados 3.000 núcleos de computação, todos com equipamentos da terceira geração, as linguagens de alto nível apresentaram as seguintes percentagens de utilização:

COBOL	77%
FORTRAN	19%
PL1	4%

Em paralelo à rápida difusão do COBOL no mundo inteiro, os mini-computadores foram sendo seguidamente aperfeiçoados e sua utilização nos mais diversos tipos de trabalho foi crescendo de ano para ano. Um dos principais resultados da evolução dos mini computadores foi a acentuada queda do custo desses equipamentos. Isso tornou possível que um grande número de instituições educacionais, firmas comerciais, etc que por razões econômicas não podiam adquirir computadores passasse a ter condições de comprá-los. Verificou-

se que apesar da grande difusão da linguagem COBOL, um número relativamente elevado dessas máquinas não tem compiladores ou interpretadores dessa linguagem.

A idéia deste trabalho surgiu baseada nos fatos acima. Deveria ser construído um interpretador COBOL para o mini-computador MITRA-15, construído pela CII (Compagnie Internationale pour L'Informatique). No entanto a idéia inicial foi abandonada devido à falta de periféricos necessários para um trabalho dessa ordem (principalmente memória auxiliar) no MITRA-15 instalado na COPPE. Ficou então resolvido que seria desenvolvida a estrutura de um interpretador capaz de ser facilmente implementado em qualquer modelo de mini-computador, desde que este seja dotado dos periféricos necessários.

Esse trabalho foi então dividido em duas teses de Mestrado na COPPE, uma tratando do interpretador propriamente dito e a segunda de um supervisor capaz de comandar o interpretador. O segundo trabalho citado acima é esta tese e o outro é :"Estudo de um Interpretador COBOL para um Mini-Computador".

Apresentamos a seguir um resumo do estudo do interpretador.

a - FORMA INTERNA DAS INSTRUÇÕES

Inicialmente foi pesquisada uma forma interna para as instruções do programa fonte. Por ser mais conveniente para o uso em linguagens do tipo do COBOL, a escolha recaiu sobre as QUÁDRUPLAS. As Quádruplas contêm o código das operações no primeiro

campo, com um comprimento de uma palavra e os três campos seguintes fornecem as informações sobre os operandos. Essas informações são:

a-1 a tabela que deverá ser consultada

a-2 o tipo da PICTURE

a-3 se existe subscritos no operando ou não

a-4 um apontador para a tabela indicada em a-1

Todas as quádruplas são alocadas em memória auxiliar (disco) de modo que cada parágrafo fique residente em um setor que tem normalmente uma dimensão de 128 palavras. Quando um parágrafo é maior do que um setor, a última quádrupla do setor é um sinal que indica que o parágrafo continua no setor seguinte.

b - FASE DE ANÁLISE

A fase de análise (estudada em detalhes em "Estudo de um Interpretador COBOL para Mini-Computadores") será encarregada de gerar e alocar as quádruplas no disco, bem como de montar as tabelas indispensáveis ao sistema. Serão necessárias para o supervisor tabelas de variáveis, literais, arquivos e parágrafos.

c - BUFFER

O sistema contará com um 'BUFFER' com um tamanho igual ao de um setor do disco, ou seja, com 128 palavras.

Como cada quádrupla ocupa um total de quatro palavras, será possível alocar nesse 'BUFFER' 32 delas.

d - FREQUÊNCIA DAS INSTRUÇÕES

Foi feito um estudo para estimar as instruções que aparecem com maior frequência nos programas COBOL. Para fazer a contagem, foi escrito um programa FORTRAN que após a leitura de uma pilha de cartões COBOL informa a percentagem de ocorrência das instruções.

e - FORMA INTERNA DOS DADOS

A representação interna dos dados foi feita com a preocupação de utilizar um mínimo possível de memória. As especificações das variáveis e as tabelas de literais e arquivos serão alocadas em memória auxiliar. A maior parte desse trabalho foi desenvolvido no "Estudo de um Interpretador COBOL para um Mini-Computador".

f - ALGORITMO DE GESTÃO DE ROTINAS

Para controlar quais as subrotinas das instruções devem ser colocadas ou retiradas da memória principal, foi construído um algoritmo de gestão de memória.

mod 1 - Depois que o operador de uma quádrupla é identificado, o algoritmo inicia sua tarefa. Esse algoritmo exerce o controle da alocação na memória principal das subrotinas para execução das instruções que entram ou saem da memória de acordo com

as necessidades.

g - ALGORITMO DE GESTÃO DE VARIÁVEIS

As variáveis quando são alocadas na memória principal ficam em uma tabela chamada Tabela de Variáveis em Uso. Para controlar as entradas e saídas de variáveis nesta tabela, foi construído um algoritmo cuja descrição resumida é a seguinte:

g-1 - Procura saber quando a tabela de variáveis em uso está cheia

g-2 - Verifica quais as últimas variáveis que entraram na memória

g-3 - Escolhe as variáveis que serão retiradas

g-4 - Reloca as variáveis que permanecerão na memória para abrir espaços para novas variáveis.

h - SUBROTINA DAS INSTRUÇÕES

O estudo das subrotinas para executar as instruções foi feito em trabalho palalelo (Vide Tese de M.Sc. COPPE - "Estudo de um Interpretador COBOL para um Mini-Computador").

CAPÍTULO II

O MINI - COBOL

O MINI-COBOL a ser utilizado é um sub-conjunto do COBOL-65, capaz de ser implementado em computadores que tenham uma configuração de memória de pelo menos 8.000 bytes.

Um dos objetivos desejados é de que o mini-COBOL seja compatível com a grande maioria dos compiladores existentes, deixando claro que as diferenças residem apenas nas restrições que serão apresentadas mais adiante. Por essa razão, para a escolha do conjunto de instruções para compor o Mini-COBOL, foi tomado por base o COBOL normalizado (ANS) do IBM/360.

Os formatos do mini-COBOL são os seguintes:

2.1 - IDENTIFICATION DIVISION

PROGRAM-ID .	nome do programa
AUTHOR .	nome(s) do(s) autor (es)
INSTALLATION .	nome do Departamento
DATE-WRITTEN .	data em que o programa foi feito
DATE-COMPILED .	data em que o programa foi rodado
SECURITY .	comentários
REMARKS .	comentários

2.2 - ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. nome do computador

OBJECT-COMPUTER. nome do computador

INPUT-OUTPUT SECTION

SELECT nome do arquivo-1 ASSIGN nome do periférico

: :

SELECT nome do arquivo-n ASSIGN nome do periférico

2.3 - DATA DIVISION.

FILE SECTION.

FD nome do arquivo

[BLOCK nº inteiro RECORDS]

DATA RECORDS nome do arq-1 ..... nome do arq-n

número de nível { nome de dados / FILLER }

[OCCURS inteiro TIMES] PICTURE cadeia de caracteres

[VALUE literal]

WORKING-STORAGE SECTION

número de nível { nome de dados / FILLER }

[OCCURS inteiro TIMES] PICTURE cadeia de caracter [VALUE literal]

2.4 - PROCEDURE DIVISION.

nome do parágrafo.

sentença [sentença] ....

nome do parágrafo contendo EXIT .

EXIT.

nome do parágrafo.

[[declaração imperativa] [declaração imperativa] ...]

[[declaração condicional] .....]

[NOTE cadeia de caracteres] ...

IF identificador { GREATER THAN / LESS THAN / EQUAL TO } { literal / identificador }

THEN declarações imperativas [ELSE / OTHERWISE] declarações imperativas

READ nome do arquivo.

ADD { literal-1 / identificador } { literal-2 / identificador-2 } GIVING identificador-n

SUBTRACT { literal / identificador } FROM { literal / identificador } GIVING identificador.

MULTIPLY { literal / identificador } BY { literal / identificador } GIVING identificador.

DIVIDE { literal / identificador } INTO { literal / identificador } GIVING identificador.

MOVE { literal / identificador } TO identificador.

DISPLAY { literal não numérico / identificador }

WRITE nome do arquivo { BEFORE / AFTER } inteiro.

OPEN { INPUT / OUTPUT } nome do arquivo.

CLOSE nome do arquivo.

GO TO nome de parágrafo.

PERFORM nome de parágrafos [THRU nome de parágrafo]

EXIT

STOP { literal / RUN }

CAPÍTULO III

A FORMA INTERNA DAS INSTRUÇÕES

Em todos os compiladores e interpretadores, por razões de economia de espaços na memória e facilidade de implementação, o programa fonte é inicialmente traduzido para uma forma interna intermediária. Como nos interpretadores não é necessário a geração de um código as instruções são executadas na própria forma interna. Para o interpretador estudado foi escolhido o tipo de forma conhecida por quádruplas. A forma geral de uma quádrupla é a seguinte:

<OPERADOR> , <OPERANDO-1> , <OPERANDO-2> , <RESULTADOS>

onde <operando-1> e <operando-2> , especificam os argumentos e <resultados> o resultado da operação .

Houve porém uma ligeira modificação na estrutura da quádrupla apresentada acima. <operando-1>, <operando-2> e <resultados > estão divididos em quatro campos que contêm as seguintes informações:

a) tabela que deverá ser consultada
b) o tipo da picture
c) se uma determinada variável é indexada ou não
d) um apontador para tabela correspondente.

Estes campos são discutidos abaixo.

Neste sistema, cada quádrupla ocupa quatro palavras de 16 bits. A palavra destinada ao operador ficará com alguns bits perdidos porque os valores do código de operação serão sempre menores do que 2^6.

Cada uma das três palavras seguintes serão divididas da seguinte maneira:

1º campo - composto de 3 bits informa qual das tabelas deverá ser consultada. Se o valor contido neste campo é igual a:

a) 000 - está sendo indicado que os campos seguintes contêm o próprio valor de um literal inteiro, logo não será necessário consultar nenhuma das tabelas montadas na análise.

b) 001 - deverá ser consultada a tabela de variáveis.

c) 010 - deverá ser consultada a tabela de literais.

d) 011 - deverá ser consultada a tabela de variáveis temporárias.

e) 100 - deverá ser consultada a tabela de arquivos.

2º campo - este campo também é composto por 3 bits e informa o tipo da picture das variáveis e literais. Os valores contidos nesse campo têm os seguintes significados:

000	indica um ítem grupo
001	indica um ítem elementar alfabético
010	indica um ítem elementar alfanumérico
011	indica um ítem elementar numérico
100	indica um ítem elementar alfanumérico de edição
101	indica um ítem elementar numérico de edição-1
110	indica um ítem elementar numérico de edição-2

3º campo - Este campo contém um único bit e informa quando se trata de variáveis se a mesma é indexada ou não.

4º campo - Este campo é composto de 9 bits e indica o deslocamento em relação ao início de uma tabela, onde se encontra o operando, ou onde o mesmo é descrito.

A palavra destinada ao operador tem um campo contendo o código de operação que identifica as instruções. Como as subrotinas estão agrupadas em blocos, os dois primeiros bits deste campo informam o número de blocos que a subrotina está contida, enquanto que as quatro seguintes indicam o número de ordem da subrotina no bloco.

O código de operações é o seguinte:

00 0000	MOVE
00 0001	WRITE
00 0010	ADD
00 0011	PERFORM
00 0100	READ
00 0101	STOP
00 0110	MULTIPLY
00 0111	DIVIDE
00 1000	OPEN
00 1001	CLOSE
00 1010	DISPLAY

00 1011	SUBTRACT
00 1100	EXIT
00 1101	GO TO
00 1110	DI
00 1111	DSZ
01 0000	DSMZ
01 0001	DSME
01 0010	DSMIZ
01 0011	DSMEI

Todas as instruções a partir do código 11 0011 são usadas na instrução IF , isto será discutido mais adiante.

A representação em forma de quádruplas dessas instruções será mostrada abaixo. Para cada instrução será feito um exemplo de como se usa no programa fonte seguido de sua forma interna. Está sendo assumido em todos os casos que a forma interna começa no endereço 1000 .

3.1 - ADD

FORMA DO PROGRAMA FONTE

```
ADD { identificador-1 } { identificador-2 } [identificador-3]...
    { literal-1       } { literal-2       } [literal-3      ]

..... [identificador-n]        GIVING      identificador-m
      [literal-n      ]
```

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 0010 , ident-1, ident-2, temp-1
1004	00 0010, ident-3, temp-1, ident-m
⋮	⋮
1020	00 0010, ident(n-1), temp(n-2), temp(n-1)
1024	00 0000, temp(n-1), , ident-m

3.2 - CLOSE

FORMA DO PROGRAMA FONTE

CLOSE nome do arquivo

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 1001 , endereço do arquivo

3.3 - DISPLAY

FORMATO DO PROGRAMA FONTE

DISPLAY { literal não numérico / identificador }

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 1010 , nome

3.4 - DIVIDE

FORMATO DO PROGRAMA FONTE

```
DIVIDE { literal-1       } INTO { literal-2       }
       { identificador-1 }      { identificador-2 }

GIVING    identificador-3
```

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 0111 , ident-1, ident-2, temp
1004	00 0000 , temp , , ident-3

3.5 - EXIT

FORMATO DO PROGRAMA FONTE

```
nome do parágrafo.   EXIT
```

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 1100

3.6 - GO TO

FORMA DO PROGRAMA FONTE

```
GO TO    nome de rótulo
```

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 1101 , endereço do parágrafo no disco

3.7 - MOVE

FORMA DO PROGRAMA FONTE

MOVE { identificador-1 / literal-1 } TO identificador

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 0000 , ident-1, , ident-2

3.8 - MULTIPLY

FORMA DO PROGRAMA FONTE

MULTIPLY { identificador-1 / literal-1 } BY { identificador-2 / literal-2 } GIVING identif.

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 0110 , ident-1, ident-2, temp
1004	00 0000, temp, , ident-3

3.9 - OPEN

FORMA DO PROGRAMA FONTE

OPEN, INPUT, ARQ-1, ARQ-2,..., ARQ-N

OUTPUT, ARQS-1, ARQS-2,..., ARQS-N

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 1000, ENDARQ-1
1004	00 1000, ENDARQ-2
⋮	⋮
	00 100 , ENDARQ-N

3-10 - READ

FORMA DO PROGRAMA FONTE

READ nome do arquivo

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 0100, endereço do arquivo

3.11 - STOP

FORMA DO PROGRAMA FONTE

STOP RUN

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 0101 , endereço

3.12 - SUBTRACT

FORMA DO PROGRAMA FONTE

SUBTRACT { identificador-1 / literal-1 } FROM { identificador-2 / literal-2 } GIVING identif.

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 1011, ident-1, ident-2, temp
1004	00 0000, temp, , ident

3.13 - WRITE

FORMA DO PROGRAMA FONTE

WRITE nome do registro

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 0001 , endereço do registro

3.14 - PERFORM

FORMATO DO PROGRAMA FONTE

PERFORM nome do parágrafo-1 THRU nome do parágrafo-2 número inteiros TIMES .

FORMA INTERNA

ENDEREÇO	CÓDIGO
1000	00 0011 , parágrafo-1, parágrafo-2, Nvezes

Com as informações da quádrupla acima, a subrotina PERFORM se encarrega de montar uma pilha contendo as seguintes informações:

a) início do PERFORM

b) fim do PERFORM

c) estado corrente

d) número de vezes

e) endereço de retorno

Este procedimento permite a existência de um PERFORM dentro do outro. A pilha vai sendo atualizada à medida que um ninho de PERFORM's vai sendo executado.

A estrutura da pilha montada pelo PERFORM é a seguinte:

INICIO	FINAL	EST.CORR.	NVEZES	ENDRETORNO

3.15 - DECLARAÇÕES DE DESVIOS

Todas as declarações de desvios

DI - desvio incondicional

DSZ - desvia se igual a zero

DSMZ - desvia se maior do que zero

DSME - desvia se menor do que zero

DSMIZ - desvia se maior ou igual a zero

DSMEI - desvia se menor ou igual a zero

têm o mesmo tipo de forma interna ou seja

OPERANDO, ENDEREÇO logo:

FORMA INTERNA

ENDEREÇO	CÓDIGO		
1000	00 1110 , endereço	——	DI
1000	00 1111 , endereço	——	DSZ
1000	01 0000 , endereço	——	DSMZ
1000	01 0001 , endereço	——	DSME
1000	01 0010 , endereço	——	DSMIZ
1000	01 0011 , endereço	——	DSMEI

3.16 - IF

FORMA DO PROGRAMA FONTE

IF identificador-1 { GREATER THAN / LESS THAN / EQUAL TO } { identificador-2 / literal }

THEN declarações imperativas-1 [{ ELSE / OTHERWISE }]

declarações imperativas-2

FORMA INTERNA

A declaração IF será desdobrada em um conjunto de instruções durante a formação das quádruplas. A declaração

IF não tem uma forma interna própria, porém deverá surgir uma das instruções de desvios listadas acima no conjunto das instruções em que ela será desdobrada: DI, DSZ, DSMZ, DSME, DSMIZ, DSMEI .

O desdobramento da declaração IF é dividido em duas fases: a primeira delas faz a comparação entre o identificado-1 e o identificador-2 ou um literal. É nessa fase que aparece uma das instruções de desvio já citadas acima. A segunda fase executa a declaração ou o conjunto de declarações para o qual foi feito o desvio da primeira fase.

Uma declaração IF desenvolvida em quádruplas apresenta a seguinte estrutura:

```
ENDEREÇO        CÓDIGO
1000            OP, ident-1, ident-2, temp
1004   DESVIO=  OP, 1080,           , temp
1008            OP, ident-3, ident-4, temp-1
1012            OP, ident-5, ident-6, temp-2
 ⋮               ⋮    ⋮       ⋮     ⋮
1080            OP     ..............
```

Será apresentado agora um exemplo do desdobramento da declaração IF , cuja forma no programa fonte é a seguinte:

IF I GREATER THAN J THEN ADD B, C GIVING E, SUBTRACT C FROM D GIVING F, MULTIPLY E BY F , ADD K, F GIVING K ELSE ADD 1, I GIVING I

ENDEREÇO	CÓDIGO
1000	00 1011 , I , J , temp-1 (SUBTRACT)
1004	01 0000 , 1140 , , temp-1 (DSMZ)
1008	00 0010 , B , C , temp-2 (ADD)
1012	00 0000 ,temp-2, , E (MOVE)
1016	00 1011 , C , D , temp-3 (SUBTRACT)
1020	00 0000 ,temp-3, , F (MOVE)
1024	00 0110 , E , F , F (MULTIPLY)
1028	00 0010 , K , F , temp-4 (ADD)
1032	00 0000 , temp-4, , K (MOVE)
1036	00 1110 , 1148 (DI)
1040	00 0010 , I , 1 , temp-5 (ADD)
1044	00 0000 , temp-5, , I (MOVE)
1048	⋮ ⋮ ⋮ ⋮
⋮	

CAPÍTULO IV

TABELAS UTILIZADAS PELO SUPERVISOR

Na fase de análise será montado um conjunto de tabelas onde ficarão as variáveis, literais, arquivos e variáveis temporais. Todas essas tabelas são guardadas em memória auxiliar e somente um pequeno número de entradas é encontrado na memória principal em um determinado instante.* Também serão montadas pelo analisador tabelas que serão utilizadas pelo supervisor para aprontar os operandos antes da execução de uma determinada instrução. O supervisor usará essas tabelas para verificar se os operandos estão na memória principal. Se eles não estão, o supervisor usa as tabelas para localizá-los no disco. Neste Capítulo será feita uma descrição dessas tabelas começando pela tabela de variáveis.

4.1 - TABELA DE VARIÁVEIS

Esta tabela torna possível o endereçamento de variáveis no disco e na memória principal, e também fornece o endereço de uma tabela guardada no disco que contém todas as especificações de uma determinada variável. A estrutura geral desta tabela é a seguinte:

---

* Tabela de variáveis temporárias reside na memória principal.

ENDEREÇO NA MEMÓRIA	TIPO-V	MUD	ORDEM NA TABELA
9 BITS	1 BIT	1 BIT	5 BITS

Logo, todas as informações sobre uma determinada variável são guardadas em uma palavra de 16 BITS.

A função dos campos que dividem uma palavra desta tabela é a seguinte:

1º CAMPO : ENDEREÇO NA MEMÓRIA - Este campo indica se a variável está na memória ou não, e, se está, fornece o seu endereço. Dependendo da informação de um dos campos das quádruplas, que indica se a variável é indexada ou não, o campo "ENDEREÇO NA MEMÓRIA" pode ser interpretado de duas maneiras:

1º CASO

Quando uma variável não é indexada, este endereço representa o deslocamento em relação a uma área de memória onde as variáveis são colocadas. Esta área será chamada "tabela de variáveis em uso". No Capítulo seguinte será apresentado um algoritmo que fará a gestão dessas variáveis, escolhendo as que devem ser retiradas e relocando as que devem permanecer. Como o campo 'ENDEREÇO NA MEMÓRIA' contém 9 bits pode-se concluir que o comprimento da tabela de variáveis em uso é igual a 312 bytes. As entradas desta tabela não terão comprimento fixo e serão formadas por dois campos. Estes campos são os seguintes:

1º CAMPO - (tabela de variáveis em uso) - Este campo tem um tamanho de 1, 2 ou 4 bytes dependendo do tipo da PICTURE. Se a PICTURE é alfabética ou alfanumérica sem edição, o comprimento da entrada

é de um byte. Se a PICTURE é numérica sem edição, o comprimento é de 4 bytes. Os tipos das 'PICTURES' são fornecidos pelas quá - druplas.

2º CAMPO - (tabela de variáveis em uso) - Este campo tem um tamanho variável, indicado pela PICTURE do campo anterior. Neste campo é guardado o próprio valor da variável.

2º CASO

Quando a variável é indexada, o campo "ENDEREÇO DE MEMÓRIA" aponta para uma lista. A primeira entrada desta lista ocupa 1, 2 ou 4 bytes dependendo do tipo da PICTURE fornecida pelas quádruplas. As considerações para esta entrada são as mesmas do primeiro campo da tabela de variáveis em uso.

As outras entradas são formadas por dois campos , sendo que o primeiro fornece o deslocamento da variável em relação ao início do registro que contém a mesma. O segundo campo contém o próprio valor da variável.

Quando uma variável indexada aparece em um programa, calcula-se seu deslocamento em relação à variável de nível 01 à qual a mesma pertence (Vide 1) . Com o deslocamento encon - trado, pesquisa-se a lista sequencialmente. Se o deslocamento for encontrado, a variável em questão se encontra na memória princi - pal. Caso contrário, procura-se a variável no disco e coloca- se na lista. Como procurar a variável será visto mais adiante.

O número máximo de entradas destas tabelas é igual

a dez, enquanto que o número de listas vai apenas a três. Logo , será preciso que o supervisor faça uma gestão para estas listas . Quando uma lista está totalmente ocupada deverão ser retiradas algumas entradas para alocação de outras entradas que se encontram no disco. Quando as três listas estão ocupadas e aparece mais uma variável indexada no programa, uma dessas listas deve ser totalmente desocupada para alocação da nova variável. Esta gestão será vista no Capítulo V .

2º CAMPO - TIPO V - Este campo informa se a variável que está contida no respectivo endereço tem o nível 01 ou não. Deste campo depende o significado do campo ORDEM NA TABELA como veremos abaixo.

3º CAMPO - MUD - Este campo composto de 1 bit informa se a respectiva variável foi alterada ou não. Esta informação será usada na gestão de memória.

4º CAMPO - ORDEM NA TABELA - Este campo tem as seguintes finalidades:

1) Determinar o endereço do registro no disco.

2) Determinar o endereço onde as especificações da variável se encontram no disco.

As especificações indicam a PICTURE e um deslocamento. Esse deslocamento é usado para encontrar o endereço do valor da variável em outro setor no disco (endereço do registro no disco + deslocamento fornecidos nas especificações = endereço da variável no disco). A Figura 1 mostra os apontadores fornecidos

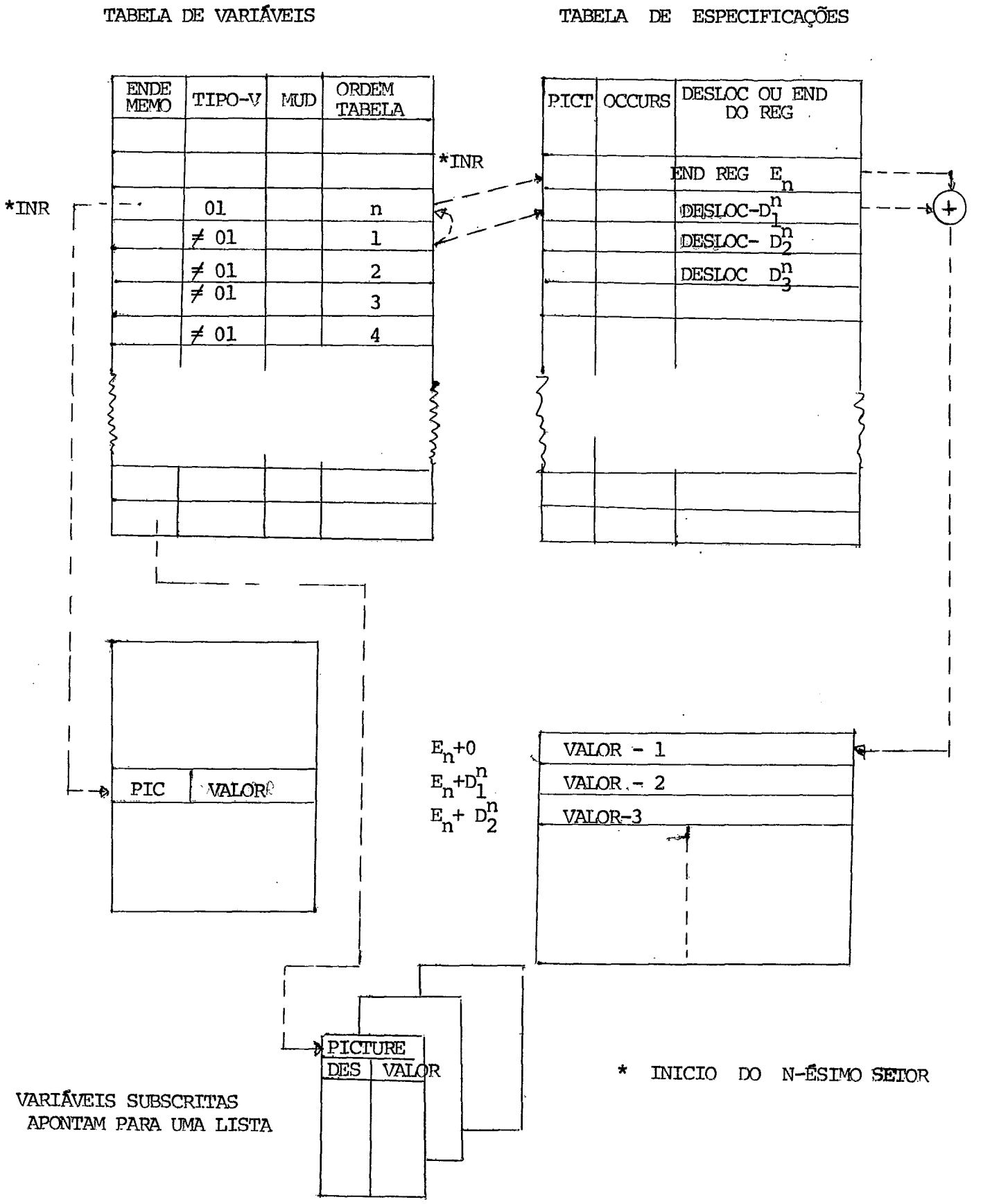


TABELA DE VARIÁVEIS
TABELA DE ESPECIFICAÇÕES
ENDE MEMO
TIPO-V
MUD
ORDEM TABELA
PICT
OCCURS
DESLOC OU END DO REG
*INR
*INR
01
n
≠ 01
1
≠ 01
2
≠ 01
3
≠ 01
4
END REG E_n
DESLOC-D_1^n
DESLOC- D_2^n
DESLOC D_3^n
PIC
VALOR
E_n+0
E_n+D_1^n
E_n+ D_2^n
VALOR - 1
VALOR - 2
VALOR-3
PICTURE
DES
VALOR
VARIÁVEIS SUBSCRITAS APONTAM PARA UMA LISTA
* INICIO DO N-ÉSIMO SETOR

pela tabela de variáveis e facilita a compreensão do campo "ORDEM NA TABELA".

O significado de 'ORDEM NA TABELA' depende do campo TIPO-V. Se o campo TIPO-V indica que a variável tem nível 01, ou seja, se a variável é um registro, este campo indica o número de ordem de entrada do registro da DATA DIVISION. Este número de ordem é a posição das especificações para o registro. Para o n-ésimo registro definido na DATA DIVISION, este campo tem o número n , indicando que o n-ésimo setor da tabela de especificações contêm as especificações deste registro. Como o número de ordem na 'DATA DIVISION' é igual ao número do setor que contém as especificações de um registro, para cada registro existe um setor correspondente contendo as especificações do mesmo.

Se o TIPO-V indica que o número de nível não é 01, 'ORDEM NA TABELA' indica a ordem da variável em relação ao início do registro. Este valor tem duas finalidades. Primeiro aponta para entrada da tabela de variáveis que contém as informações sobre a variável de nível 01 a qual a variável em questão pertence. A segunda utilidade é encontrar a posição da variável dentro do setor da tabela de especificações. Logo o procedimento para encontrar as especificações de uma variável no disco quando a mesma não tem nível 01 é o seguinte:

i) Usa-se o valor do campo 'ORDEM NA TABELA' para achar a entrada da variável de nível 01, da qual a variável em questão faz parte.

ii) Usa-se o campo 'ORDEM NA TABELA' da entrada encontrada

em i) como apontador para o setor da tabela de especificações correspondente. Coloca-se este setor na memória principal.

iii) Encontra-se o deslocamento no setor das especificações que é igual a 4 vezes o conteúdo do campo 'ORDEM NA TABELA'.

Aqui termina a discursão sobre a tabela de variá - veis. Como a tabela de especificações é muito ligada à tabela de variáveis, serão apresentados os campos desta tabela antes das considerações sobre as tabelas de literais, variáveis temporárias e arquivos.

4.2 - TABELA DE ESPECIFICAÇÕES

As especificações de uma determinada variável são encontradas em uma tabela totalmente alocada no disco. Como já foi visto, as especificações de cada variável de nível 01 estão alocadas em um setor do disco, ou seja, em cada setor serão colocadas especificações de apenas um registro. A forma geral desta tabela é a seguinte:

PICTURE	OCCURS	DESLOC
2 PALAVRAS	1 PALAVRA	1 PALAVRA

4.2.1 - PICTURE

Uma descrição completa das 'PICTURES' das variáveis se encontra nas duas primeiras palavras da tabela de especificações.

Esta representação da 'PICTURE' foi desenvolvida em 1 .

Se as informações sobre o tipo das 'PICTURES' nas quádruplas contêm 000, 001 ou 010 apenas um byte das duas palavras será utilizado. Se a quádrupla contêm 011 , dois bytes serão utilizados. E se a quádrupla contêm 100, 101 ou 110 , todos os 4 bytes serão utilizados. Para os sete casos, as informações contidas na PICTURE tem os seguintes significados:

4.2.1.1 - 000 - O byte contêm o tamanho do ítem grupo

4.2.1.2 - 001 - O byte contêm o número de 'A's existentes na PICTURE, trata-se da PICTURE alfabética.

4.2.1.3 - 010 - O byte contêm o número de 'X's existentes na PICTURE, trata-se da PICTURE alfanumérica.

4.2.1.4 - 011 - Trata-se da PICTURE numérica e são necessários 16 dos 32 bits para sua descrição.

A palavra de 16 bits está devidida em seis campos da seguinte maneira:

1º campo - Formado po 1 bit indica como a variável é armazenada (USAGE). Se o valor armazenado no bit é zero, a variável será guardada numa representação decimal externa (DISPLAY). Se o valor armazenado é 1, a variável será guardada em binário (COMPUTACIONAL).

2º campo - Este campo indica o sinal do número '0' = + e '1' = - .

3º campo - Este campo é formado por 5 bits e fornece o número de digitos da parte inteira do valor da variável.

4º campo - Formado por 4 bits indica o número de digitos da parte

decimal.

5º campo - Fornece o número de P's , isto é, o número de zeros que serão inseridos à esquerda ou à direita, dependendo do valor do campo seguinte.

6º campo - Informa em 1 bit se o número de zeros serão inseridos à esquerda ou à direita.

4.2.1.5 - 100 - Quando o tipo é igual a 100, trata-se do ítem elementar alfanumérico de edição. Para a descrição desta PICTURE serão necessários todos os primeiros 32 bits da tabela alocada no disco. Esta PICTURE pode descrever caracteres alfabéticos, alfanuméricos, zeros e brancos.

Os 32 bits estão divididos basicamente em dois campos, sendo um de 2 bits e um de 30 bits que por sua vez é dividido em sub-campos de dimensão e número variados. No primeiro campo de dois bits temos a informação de como serão distribuídos os sub-campos do segundo campo.

00 - Se os dois primeiros bits têm o valor zero, o segundo campo será formado de 5 sub-campos de 6 bits cada um. Os dois primeiros bits de cada sub-campo determinam o tipo do caracter. Se está carregado com

00 = A
01 = X
10 = ZERO
11 = BRANCO

Os demais bits determinam a quantidade dos respectivos caracteres.

01 - Se os dois primeiros bits têm o valor 01, o segundo campo será dividido em 6 sub-campos de 5 bits onde os dois primeiros bits informam o tipo do caracter (como no caso 00) e os seguintes a quantidade deles.

10 - Se os dois primeiros bits têm o valor 10 , o segundo campo será dividido em 3 sub-campos de 10 bits cada um. Os dois primeiros bits dos sub-campos informam o tipo do caracter (como no caso 00) e os oito restantes as respectivas quantidades.

11 - Se o valor dos dois bits é igual a 11, o segundo campo será dividido em 3 sub-campos de 6 bits e um de 12 bits. Também aqui os dois primeiros indicam o tipo do caracter e os demais o número deles.

4.2.1.6 - 101 - Quando o tipo da PICTURE for 101, teremos um ítem elementar numérico de edição 1 . Para a descrição dessa PICTURE será necessária uma palavra dupla (32 bits) dividida em campos.

1º campo - Composto por um bit informa se haverá ou não inserção do ponto decimal no ítem.

2º campo - Também composto por um bit informa se haverá inserção de vírgulas no ítem.

3º campo - Composto de três bits, informa se haverá inserção dos caracteres CB (crédito), DB (débito) , (+), ou (-). O (+) e o (-) podem aparecer à esquerda ou à direita do número. Se neste cam

po está armazenado o valor :

000	será inserido	CB (crédito)
001	será inserido	DB (débito)
010	será inserido	+ (à direita)
011	será inserido	+ (à esquerda)
100	será inserido	- (à direita)
101	será inserido	- (à esquerda)
110	nada a ser inserido	

4º campo - Contém um único bit, informa se haverá inserção fixa do símbolo dólar.

5º campo - Composto de dois bits, informa a quantidade de espaços em branco que existe depois do primeiro símbolo ou antes do último.

6º campo - Composto de 1 bit, informa se os espaços em branco do campo anterior se encontram respectivamente à esquerda ou à direita do número.

7º campo - Contém 3 bits e determina se na PICTURE existe algum dos tipos Z , * , $ múltiplos e (+) ou (-) múltiplos, obedecendo a seguinte convenção:

000	Z	supressão de zeros
001	*	(substituição de zeros por asteriscos)
010	$	múltiplos (substituição de zeros por dólares)
011	+	múltiplos (substituição de zeros por +)
100	-	múltiplos (substituição de zeros por -)

8º campo - Composto de 4 bits,determina a quantidade de um dos símbolos que aparecem no campo 7 .

9º campo - Composto de 4 bits, determina o número de noves na PICTURE.

10º campo - Contém 4 bits que fornecem o número de algarismos da parte decimal.

11º campo - Composto de 4 bits, informa a quantidade de zeros mais à direita da PICTURE.

4.2.1.7 - 110 - Quando o tipo da PICTURE é 110, trata-se do ítem numérico de edição-Z. Para descrição desta PICTURE será também necessário utilizar os 32 bits da tabela de especificações. Com esta PICTURE pode-se descrever os caracteres numéricos zeros e brancos.

Os 32 bits estão divididos em dois campos, sendo um de dois bits e um de trinta que por sua vez é dividido em sub-campos de dimensão e número variados. No primeiro campo estão distribuídos os sub-campos do segundo campo, obedecendo a seguinte convenção:

00 - Se os dois primeiros bits têm o valor 00, o segundo campo será formado por 5 sub-campos de 6 bits cada um. Os dois primeiros bits de cada sub-campo determinam o tipo do caracter. Se está carregado com

00 = 9
10 = ZERO
11 = BRANCO

Os demais determinam a quantidade dos respectivos caracteres.

01 - Se os dois primeiros bits têm o valor 01 , o segundo campo será dividido em 6 sub-campos de 5 bits onde os dois primeiros bits informam o tipo do caracter, como no caso 00, e os seguintes a quantidade deles.

10 - Se os dois primeiros bits têm o valor 10 , o segundo campo será dividido em 3 sub-campos de 10 bits cada um. Os dois primeiros indicam o tipo do caracter e os restantes a quantidade deles.

11 - Se o valor do dois primeiros bits é igual a 11, o segundo campo é dividido em 3 sub-campos de 6 bits e um de 12 bits. Também aqui os dois primeiros indicam o tipo do caracter e os seguintes a quantidade deles.

Isto completa o tipo da PICTURE. O assunto seguinte se refere à descrição dos outros campos da tabela de especificações OCCURS e DESLOC .

4.3 - OCCURS

A palavra que segue a descrição das PICTURES na tabela de especificações informará o número de vezes que uma determinada variável ou estrutura será repetida. Cada variável poderá ser repetida até um número igual a 2^{10} , ou seja, 1024 vezes, Os últimos seis bits da palavra não serão utilizados.

4.4 - DESLOC

Este campo utiliza apenas 10 ou 12 dos seus 16 bits. Quando as especificações são para uma variável com nível 01 serão usados 12 bits, sendo que os oito primeiros fornecem o número da trilha e os quatro seguintes o número do setor onde o registro está armazenado. Se as especificações são para uma variável de nível diferente de 01 serão utilizados 10 bits. Estes 10 bits fornecem o deslocamento da variável em relação ao início do setor onde a mesma está alocada. Quando se trata de variáveis subscritas, o deslocamento é a posição da variável com índices iguais a 1 (Vide 1) .

4.5 - ATUAÇÃO DO SUPERVISOR

Para completar a discursão sobre as tabelas de variáveis, segue um resumo da atuação do supervisor quando uma determinada variável é solicitada numa quádrupla.

O primeiro passo é verificar se a variável está na memória. Se a variável não está na memória, são lidas suas especificações no disco utilizando os apontadores dos campos 'ORDEM NA TABELA'. Na tabela de especificações (no disco) obtem-se a 'PICTURE', um endereço de um registro e um deslocamento. A 'PICTURE' é utilizada no primeiro campo da tabela de variáveis em uso ou no primeiro campo de uma das listas para variáveis subscritas. Obtem-se o endereço da variável no disco somando-se o endereço do registro com o deslocamento. (Quando se trata de variá -

veis subscritas, este deslocamento é uma função do deslocamento fornecido pelas especificações). Encontrado o endereço, a variável é lida e armazenada junto à PICTURE.

Seguem as considerações para as tabelas de variáveis temporárias, literais e arquivos.

4.6 - TABELA DE LITERAIS

Esta tabela guarda apontadores para todos os literais não numéricos e os literais numéricos que não são inteiros. Os literais inteiros terão seus valores nas quádruplas. Os valores dos literais ficarão alocados no disco e a tabela que fica na memória tem a seguinte estrutura:

END MEM	ENDEREÇO DISCO	TIPO	PICTURE	NUM. VEZES
8 bits	20 bits	1 bit	11 bits	8 bits

4.6.1 - END MEM - Este campo é composto de 8 bits e indica o endereço do literal quando o mesmo se encontra na memória.

4.6.2 - ENDEREÇO DISCO - Este campo é composto de 20 bits, sendo 8 bits para o endereço da trilha, 4 bits para o setor e os oito restantes fornecem o deslocamento em relação ao início do setor.

4.6.3 - TIPO - Este campo composto de um bit indica se o literal é numérico ou não numérico.

4.6.4 - PICTURE - Se o campo anterior indica que a PICTURE é numérica, o 1º bit deste campo indicará o sinal do número, os outros dez fornecem o número de nomes, sendo 5 para a parte inteira e 5 para a decimal. Se a PICTURE é alfanumérica, os 11 bits informam o número de X's .

4.6.5 - NUMERO DE VEZES - Este campo indica o número de vezes que um literal é solicitado em um programa. Isto é utilizado na gestão dos literais.

4.7 - TABELA DE VARIÁVEIS TEMPORÁRIAS

Esta tabela guarda o valor das variáveis temporárias criadas durante a execução das operações aritméticas. As informações nesta tabela serão armazenadas em 10 palavras cada uma, sendo uma para a PICTURE e as outras para o valor da variável. Será reservada uma área de 50 palavras, portanto é possível que até cinco variáveis temporárias fiquem na memória no mesmo instante . A estrutura desta tabela é a seguinte:

PICTURE	VALOR
1 PALAVRA	9 PALAVRAS

4.7.1 - PICTURE - No campo destinado à 'PICTURE', o primeiro bit indicará o sinal do número. Dos bits seguinte, 8 informam a quan

tidade de algarismos da parte inteira e 7 da parte decimal.

4.7.2 - VALOR - Este campo tem um comprimento fixo de 9 palavras. Geralmente não é necessária a utilização de todo este campo. Os bytes serão todos utilizados da esquerda para a direita.

4.8 - TABELA DE ARQUIVOS

As informações sobre os arquivos definidos no programa fonte são colocadas em uma tabela onde cada arquivo é descrito em duas palavras. O maior número de arquivos permitidos é 5. Logo, esta tabela ocupará uma área de 10 bytes. A estrutura da tabela é a seguinte:

UNIDADE	TAMANHO DO REGISTRO	E/S	ENDEREÇO NO DISCO	T.ORGANIZ.
4 BITS	10 BITS	1 BIT	12 BITS	2 BITS

4.8.1 - UNIDADE - Os quatro primeiros bits deste campo informam em que unidade se encontra o arquivo (disco, fita magnética, fita de papel, cartões perfurados, etc).

4.8.2 - TAMANHO DO REGISTRO - Este campo, composto de 10 BITS, fornece o tamanho do registro.

4.8.3 - ENTRADA/SAÍDA - Este campo, composto de 1 bit, indica se o registro é de entrada ou de saída.

4.8.4 - ENDEREÇO NO DISCO - Este campo contém 12 bits, onde oito informam a trilha e quatro o setor do disco.

4.8.5 - TIPO DE ORGANIZAÇÃO - Este campo contém 4 bits e informa qual será o tipo da organização em que os arquivos estão dispos - tos. A organização pode ser:

00 - sequencial

01 - acesso direto

10 - indexado sequencial

Cada entrada desta tabela usa 29 dos 32 bits reservados.

No próximo Capítulo será considerada a gestão de memória apresentando métodos de gestão para as subrotinas do in - terpretador, variáveis e os literais. As variáveis temporárias são residentes na memória principal e não precisam de gestão. As variáveis dos arquivos são tratadas como variáveis comuns, não sendo consideradas como um caso diferente.

CAPÍTULO V

FASE DE INTERPRETAÇÃO E SUPERVISÃO

Este capítulo consta de um estudo de um supervisor e um conjunto de subrotinas que são capazes de executar as instruções do MINI-COBOL. Todas essas subrotinas serão alocadas em memória auxiliar, e à medida em que serão solicitadas serão alocadas na memória principal.

Já foi visto nos capítulos anteriores que o analisador fornece como saída o programa fonte em forma de quádruplas. Essas quádruplas são alocadas no disco de maneira que todo parágrafo fique no início de um setor. O disco que está sendo considerado é composto de 256 trilhas,cada uma com 16 setores de 128 palavras de 16 BITS. Logo, cada setor tem capacidade para alocação de 32 quádruplas. Quando aparece um parágrafo composto de um número de quádruplas superior a 32, a última quádrupla do setor onde o parágrafo está alocado é um sinal avisando que o parágrafo continua no setor seguinte e assim sucessivamente.

Todas as quádruplas do programa fonte alocadas no disco serão admitidas na memória principal através de um BUFFER. Este "buffer" terá uma dimensão igual a 128 palavras, ou seja, o tamanho de um setor do disco citado acima. Com o BUFFER do tamanho de um setor do disco, a programação do supervisor será facilitada porque para carregá-lo basta que um setor inteiro seja movido para a memória principal.

Depois que o BUFFER está carregado, o supervisor inicia a execução das instruções contidas no mesmo. Para cada instrução verifica-se se os operandos são válidos. Se sim, o supervisor apronta os operandos e as rotinas do interpretador que serão usadas. Este processo requer uma gestão de memória. Nas próximas seções será discutida esta gestão.

5.1 - GESTÃO DE MEMÓRIA

Depois que o operador de uma quádrupla é identificado, o algoritmo de gestão de memória inicia a sua tarefa. Este algoritmo controla a alocação na memória principal das subrotinas utilizadas para a execução das instruções. Uma área de memória de 2000 bytes foi reservada para a alocação das subrotinas citadas acima. O número de bytes necessários para a alocação de cada uma das subrotinas citadas acima é aproximadamente o seguinte:

CÓDIGO	INSTRUÇÃO	NÚMERO DE BYTES
00 0000	MOVE	1000
00 0001	WRITE	100
00 0010	ADD	250
00 0011	PERFORM	500
00 0100	READ	100
00 0101	STOP	50
00 0110	MULTIPLY	250
00 0111	DIVIDE	250
00 1000	OPEN	50
00 1001	CLOSE	50
00 1010	DISPLAY	50

CÓDIGO	INSTRUÇÃO	NÚMERO DE BYTES
00 1011	SUBTRACT	250
00 1100	EXIT	50
00 1101	GO TO	50
00 1110	DI	50
00 1111	DSZ	50
01 0000	DSMZ	50
01 0001	DSME	50
01 0010	DSMIZ	50
01 0011	DSMEI	50

A soma dos comprimentos de todas as subrotinas é igual a 3800 bytes. Como a área de memória reservada para essas subrotinas não ultrapassa 2000 bytes, está claro que não será possível alocar todas essas subrotinas em um mesmo instante na memória principal. Para resolver esse problema foi necessário um estudo para saber quais as subrotinas que devem permanecer ou serem retiradas da memória em um determinado instante.

À medida que uma determinada instrução é solicitada, o supervisor se encarregará de colocá-la na memória. Como a área reservada para a alocação das subrotinas é 2000 bytes e a soma total dos seus comprimentos é igual a 3800, é possível que em um determinado programa seja necessário retirar algumas subrotinas para alocação de uma nova.

Para a execução desse algoritmo será organizada uma tabela com a seguinte estrutura:

TABELA DE GESTÃO DE MEMÓRIA

	ESTÁ NA MEMÓRIA	COMPRIMENTO DA SUBROTINA	NÚMERO DE VEZES	ENDEREÇO NA MEMÓR.	ENDEREÇO NO DISCO
MOVE		1000			
WRITE		100			
ADD		250			
PERFORM		500			
READ		100			
STOP		50			
MULTIPLY		250			
DIVIDE		250			
OPEN		50			
CLOSE		50			
DISPLAY		50			
SUBTRACT		250			
EXIT		50			
GO TO		50			
DI		50			
DSZ		50			
DSMZ		50			
DSMIZ		50			
DSMEI		50			
	1 bit	10 bits	12 bits	12 bits	12 bits

O primeiro campo desta tabela informa se a subrotina se encontra na memória ou não. O segundo campo indica o número de bytes necessários para alocação da subrotina. O terceiro campo informa o número de vezes que uma subrotina foi chamada na execução de um programa. Os dois campos restantes fornecem respectivamente os endereços na memória e no disco.

Quando a área destinada à alocação das subrotinas está cheia e uma foi solicitada, será inicialmente feita uma pesquisa sequencial no campo "número de vezes". Esta pesquisa indica qual foi a variável menos utilizada. Feito isso,compara-se o campo 'COMPRIMENTO DA SUBROTINA" da subrotina que vai entrar com a subrotina que vai sair. Se o espaço aberto permite a alocação, será iniciada uma relocação na memória e em seguida a subrotina será alocada. Caso não seja suficiente, inicia-se uma nova pesquisa no campo "NUMERO DE VEZES".

Em resumo, este algoritmo faz o seguinte:

a) Testa o campo 'ESTÁ NA MEMÓRIA" : se o valor é 1, vai para F ; se o valor é zero, procura-se o endereço no disco no campo destinado para isso e vai para B .

b) Identifica o campo 'COMPRIMENTO DA SUBROTINA' da subrotina solicitada e soma-se este valor com o comprimento total das subrotinas alocadas. Vai para C .

c) Testa a soma encontrada em B com 2000.
Se maior, vai para D ; se menor, vai para E .

d) Pesquisa o campo 'NUMERO DE VEZES' para encontrar a subrotina menos usada.

d.1- Retira a menos usada da memória e coloca o valor 4096 no campo 'NUMERO DE BYTES'

d.2- Faz a relocação e vai para B .

e) Aloca nova subrotina na memória e atualiza a tabela. Vai para F .

f) Subrotina está pronta para execução.

Para fazer a relocação citada em d-2, inicia-se um contador 'BYTES ELIMINADOS' com zeros e em seguida executa-se os seguintes passos.

A - Testa o campo 'NUMERO DE VEZES' da tabela de gestão. Se o campo contém o valor 4096, soma-se 'COMPRIMENTO DA SUBROTINA' a 'BYTES ELIMINADOS' e vai para a próxima entrada. Se o campo não contém este valor, vai para B .

B - Subtrai 'BYTES ELIMINADOS' do campo 'ENDEREÇO MEMÓRIA' para obter o novo endereço da subrotina. Atualiza o campo 'ENDE - REÇO MEMÓRIA' e reloca a variável.

C - Coloca zeros em todos os campos 'NUMERO DE VEZES'.

5.2 - GESTÃO DE VARIÁVEIS

Após a identificação do operador e da execução do algoritmo de gestão de memória deixando alocada na memória princi pal a subrotina que será chamada para executar a instrução, o su-

pervisor passa a identificar os operandos das quádruplas.

No Capítulo III, onde foi estudada a estrutura das quádruplas, ficou certo que tanto o operador da quádrupla como os operandos ocupariam uma palavra de memória. Também foi visto que as palavras ocupadas pelos operandos estão divididas em sub-campos da seguinte maneira:

1º campo - Composto de 3 bits para indicar se precisa consulta a uma tabela ou não, e se precisa, qual a tabela.

2º campo - Composto de 3 bits, indica o tipo da PICTURE.

3º campo - Composto de 1 bit, indica se uma determinada variável é indexada ou não.

4º campo - Composto de 9 bits, contém um apontador para a tabela indicada no primeiro campo.

Quando se trata de um literal inteiro, a palavra que contém o operando passa a ter apenas dois campos, pois o valor do literal está guardado na própria quádrupla.

Logo,quando um operando vai ser identificado, o primeiro passo é o reconhecimento dos três primeiros bits para saber se será necessária ou não a consulta a uma determinada tabela, e se fôr, qual será a tabela consultada. Essa tabela pode ser uma das seguintes:

a) tabela de variáveis

b) tabela de variáveis temporárias

c) tabela de literais

d) tabela de arquivos

Se nos primeiros bits de um operando de uma quádrupla for encontrado o valor 000 , no campo seguinte está armazenado o próprio valor do operando. Neste caso, o operando é um literal inteiro que já está pronto para ser executado e o conteúdo dos três primeiros bits do operando de uma quádrupla é 001 , o último sub-campo composto de 9 bits é um apontador para tabela de variáveis. Neste caso será feita uma consulta à tabela de variáveis residente na memória principal. Antes, porém, é necessário identificar os outros campos do operando da quádrupla para colher informações sobre o tipo da 'PICTURE' e saber se a variável é indexada ou não.

O apontador da quádrupla permite o acesso à tabela de variáveis vista no Capítulo IV . Agora será feita uma revisão desta tabela, dando ênfase ao seu uso na gestão de memória. A tabela tem a seguinte forma:

ENDEREÇO DE MEMÓRIA	TIPO-V	MUD	ORDEM NA TABELA

Para a identificação de uma variável, o primeiro passo é reconhecer o campo 'ENDEREÇO DE MEMORIA'. Se o endereço desse campo é nulo significa que a variável não se encontra na memória principal. Se a variável não é indexada, 'ENDEREÇO DE MEMÓRIA' representa o deslocamento em relação ao início de uma área de

memória. Essa área chamada 'TABELA DE VARIÁVEIS EM USO' é formada por dois campos. O primeiro campo contém a 'PICTURE' da variável que pode ser descrito em um byte, uma palavra ou duas palavras. O segundo campo tem o seu comprimento determinado pelo primeiro. Logo,cada entrada dessa tabela pode ter uma dimensão diferente. Os registros e ítens grupos em geral nunca serão alocados nessa tabela. Somente serão alocadas as variáveis que são usadas pelas declarações ADD, SUBTRACT, MULTIPLY e DIVIDE . A razão disso é que um ítem grupo pode ter um comprimento muito grande e poderia criar uma série de dificuldades para gestão das variáveis. Logo, instruções como o MOVE, por exemplo, limita-se a tirar um grupo de variáveis do disco, colocar na memória em um lugar que não seja a tabela de variáveis em uso, e em seguida colocá-las em algum outro lugar no disco. Pode acontecer que alguma variável que pertence a um ítem grupo, sendo movida, se encontre na tabela de variáveis em uso com alterações no seu valor. Neste caso, antes da instrução MOVE ser executada, o valor da variável alterada deve ser atualizado no disco. De maneira análoga isso é feito para WRITE, DISPLAY e READ .

Como foi reservada uma área de memória de apenas 512 bytes para alocação da tabela de variáveis em uso, é fácil notar que esta tabela pode ter toda sua área ocupada no decorrer da execução de um programa. Para resolver este problema foi feito um algoritmo para gestão da tabela de variáveis em uso. Depois que a área da tabela está totalmente ocupada, o algoritmo deve ser acionado para retirar da memória as variáveis menos utilizadas que

ocupam uma área maior e fazer uma relocação a fim de abrir vagas para novas variáveis. Em vez de usar todas as variáveis na contagem para saber quais são as que devem ser retiradas, foi decidido que as variáveis que foram recentemente alocadas não serão levadas em consideração.

Para execução do algoritmo será mantida na memória principal uma tabela contendo os seguintes campos:

a) apontador para tabela de variáveis

b) número de bytes utilizados pela variável mais sua respectiva PICTURE

c) número de vezes que a variável é utilizada

A estrutura geral desta tabela é a seguinte:

APONTADOR TAB.VARIÁVEIS	NUMERO DE BYTES	NUMERO DE VEZES
9 BITS	5 BITS	10 BITS

Toda vez que o supervisor encontra na tabela de variáveis (tabela que tem END, MEM., TIPO-V, MUD e ORD TAB) um endereço diferente de zero, significa que a variável está na memória principal. Entçao, na tabela acima será incrementado o campo NUMERO DE VEZES. Para fazer isso é necessário pesquisar os campo 'APONTADOR TAB VARIAVEIS' sequencialmente. Quando o apontador para a variável em questão é encontrado, o campo 'NUMERO DE VEZES corres-

pondente será incrementado. Esta pesquisa sequencial elimina a necessidade de um apontador para a tabela acima.

Se o campo 'NUMERO DE VEZES' atingir a 1022 será reduzido para 512. Este procedimento continua a informar que a variável é muito utilizada e evita 'OVERFLOW' no campo (o valor 1023 tem um sentido especial para gestão).

Depois que toda a área da tabela de variáveis em uso está ocupada, será feito um cálculo para escolher o número de variáveis recentemente alocadas. Este número será igual ao truncamento de 5% do total de variáveis alocadas na tabela em uso.

Logo, se exitem n' variáveis na tabela de variáveis em uso, o cálculo será o seguinte x = TRUN (5n/100) . Estas últimas x variáveis não serão levadas em conta no cálculo da frequência. Com as variáveis restantes, o algoritmo vai inicialmente procurar a que foi utilizada o menor número de vezes. Caso essa variável seja única, será retirada da memória. Se existe mais de uma variável com o menor índice de ocorrências, inicialmente serão procuradas as que não foram alteradas e dessas,a que tem maior dimensão,será retirada da memória. Se a tabela ainda não ficou com a metade de sua área desocupada, o processo será repetido. Quando o número de bytes ocupados é menor ou igual a 256, esta fase termina e a relocação das variáveis restantes começa.

Em resumo, o algoritmo de gestão de variáveis é o seguinte:

A) No campo 'NUMERO DE VEZES' da tabela de gestão de variáveis em uso procura-se a variável com menor número de frequência.

B) Testa se existe mais de uma variável com o menor número de ocorrências.

C) Se existe mais de uma, vai para D .

C.1)Se existe apenas uma, verifica no campo 'NUMERO DE BYTES' da tabela de gestão a área que a variável ocupa.

C.2)Vai para E .

D) Existe mais de uma variável com o mesmo número de vezes.

Se existe alguma variável que não foi modificada vai para D.2.

D.1 - São todas variáveis modificadas

D.1.1 - Procura variável de maior dimensão no campo número de bytes.

D.1.2 - Atualiza variável no disco

D.1.3 - Vai para E .

D.2 - Existem variáveis que não foram modificadas

D.2.1 - Procura variável de maior dimensão

D.2.2 - Vai para E .

E)

E.1 - Coloca 1023 no campo 'NUMERO DE VEZES' para indicar que a variável não deve ser considerada no passo seguinte.

E.2 - Subtrai o número de bytes ocupados pela variável do número de bytes que estão ocupados na tabela.

E.3 - Testa o total atual da tabela com um valor igual à metade da tabela.

E.3.1 - Se maior vai para A .

E.3.2 - Se menor ou igual inicia a relocação e atualização das variáveis.

RELOCAÇÃO

Depois que a metade da tabela está vazia, será dado início à relocação. Para fazer a relocação será mantido um contador com o número de bytes que as variáveis devem ser relocadas. O processo será o seguinte:

A - Inicializa dois contadores , BYTES ELIMINADOS e NOVA com zeros.

B - Para cada entrada na tabela executa os passos descritos em 1, 2 e 3 :

1 - Testa o campo 'NUMERO DE VEZES'. Se o campo contém 1023 soma 'NUMERO DE BYTES' a 'BYTES ELIMINADOS' e vai para a próxima entrada. Se o campo não contém 1023, vai para 2 .

2 - Usando o apontador para a tabela de variáveis acha o endereço na memória da variável. Atualiza 'ENDEREÇO NA MEMORIA'na tabela de variáveis e reloca a variável.

3 - Muda os campos 'APONTADOR' e 'NUMERO DE BYTES' para a posição 'NOVA' da tabela. Incrementa NOVA por um e vai para a próxima entrada.

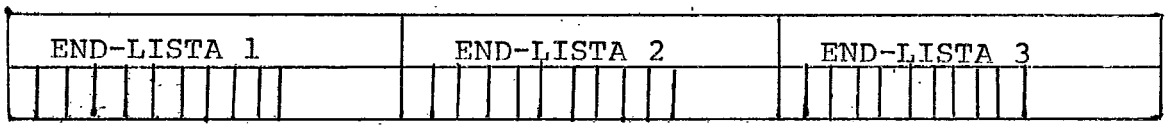
C - Coloca zeros em todos os campos NUMERO DE VEZES e nos campos 'APONTADOR' que não são usados.

Quando a variável é indexada, o campo 'ENDEREÇO DE MEMORIA' da tabela de variáveis aponta para uma lista. Esta lista tem como primeira entrada o valor da 'PICTURE' ocupando um espaço de 1 byte, 1 palavra ou 2 palavras como já foi visto anteriormente. O restante das entradas contêm 2 campos. O primeiro

fornece o deslocamento em relação à variável de nível 01; o segundo campo contém o valor. A dimensão deste campo depende do valor da 'PICTURE'.

Serão montadas apenas 3 destas listas com 10 entradas cada uma. Como estas listas podem ser preenchidas no decorrer de um programa. O supervisor também se encarregará de fazer uma gestão para elas. Toda vez que uma dessas listas está com suas 10 entradas ocupadas e aparece uma nova entrada para ser alocada, o supervisor liberará todas as entradas da tabela. Antes porém, todas as variáveis que foram modificadas deverão ter seus valores devidamente atualizados no disco. A mesma coisa acontece quando as três listas estão ocupadas e aparece uma nova variável no programa fonte.

Para controlar esta gestão, será alocada na memória uma tabela de gestão de variáveis indexadas. Esta tabela tem a seguinte forma:



Como se vê na figura acima, esta tabela é composta de duas entradas, sendo que a primeira fornece o endereço no disco do registro que contem a variável e a segunda informa quais das 10 variáveis da lista foram modificadas.

Se as três listas estão sendo utilizadas e aparece uma nova variável indexada, uma das listas deve ser desocupada. Para escolher esta lista, procura-se na tabela acima qual das três tem um menor número de variáveis modificadas.

Para atualizar os valores das variáveis subscritas no disco, foi escolhido um método para reduzir tanto quanto possível o número de saídas para o disco. O método é o seguinte: primeiro identifica e junta em conjuntos as variáveis com subscritos consecutivos. Feito isso, determina-se a primeira e a última variável modificada para cada conjunto. Finalmente escreve todas as variáveis começando com a primeira modificada e terminando com a última modificada. Logo, para cada conjunto é necessária apenas uma saída. Sabendo-se que é comum os elementos de um "Array" aparecerem sequencialmente, este método precisa de muito menos saídas do que um método que atualiza uma variável modificada por cada saída.

O endereço usado na saída para o disco é calculado da seguinte maneira:

ENDEREÇO DE SAÍDA =	endereço do registro dado na tabela de gestão de variáveis subscritas	+	deslocamento dado na lista junto com o valor

GESTÃO DOS LITERAIS

Se o conteúdo dos 3 primeiros bits de um operando de uma quádrupla é 010 significa que os últimos 9 bits da palavra armazenam um apontador para tabela de literais. Esta tabela também já foi vista anteriormente e tem a seguinte forma:

END MEM	ENDER DISCO	TIPO	PICTURE	NUM VEZES
8 BITS	20 BITS	1 BIT	11 BITS	8 BITS

Será reservada uma área na memória de 128 bytes para alocação dos literais na memória. Como esta tabela pode ser totalmente ocupada e seja necessária a alocação de uma nova variável, o supervisor deverá fazer uma gestão para os literais. Cada entrada da tabela de literais será formada de dois campos: o primeiro contendo a 'PICTURE' e o segundo o valor do literal.

Quando um literal não inteiro é solicitado em um programa, o primeiro passo será verificar se o mesmo está ou não na memória. Se END MEM é igual a zero indica que o literal não se encontra na memória; então o supervisor deve encontrar o literal no disco e colocá-lo na tabela de literais alocada na memória. Se a tabela está cheia, o algoritmo de gestão dos literais será acionado.

Este algoritmo inicialmente procura no campo NUMERO DE VEZES da tabela de especificações dos literais qual o literal com menor número de frequência. Se este valor não é único, será retirado o literal que apresenta menor número de frequência e maior dimensão. Este processo continua até que a tabela fique reduzida à metade. Quando a tabela está com a metade desocupada será feita uma relocação semelhante a das variáveis.

BIBLIOGRAFIA

1 - BOUHOT, Jean-Pierre , "Cobol Efficace", L'Informatique, Nov/Dez 72, Jan/73 .

2 - GILES, P. , "Mini-Cobol", Computer Journal, Agosto/69.

3 - GRIES, David , "Compiler Construction for Digital Computers", John Wiley & Sons, 1971.

4 - IBM : "OS Full American National Standard COBOL, GC 28-6396-3

5 - KATZAN Jr., Harry, "Advanced Programming, Van Nostrand Reinhold Company, 1970.

6 - KNUTH, D.E. , "The Art of Computer Programming", Vol.1, Addison-Wesley Publ. Company, 1968.

7 - DONAVAN, John, "Systems Programming", McGraw-Hill Book Co.,1972.

8 - KATZAN, Jr. Harry, "Operational Systems", Van Nostrand Reinhold Company, 1973.