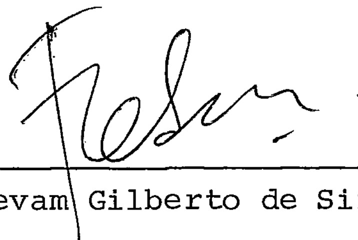
"PLMIX - UM MODELO DE LINGUAGEM DE MÉDIO NÍVEL E SEU COMPILADOR

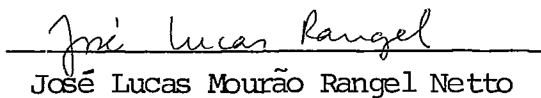
Lígia Barros Caúla

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.).

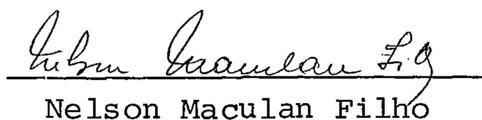
Aprovada por:



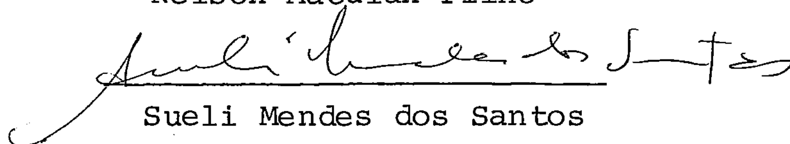
Estevam Gilberto de Simone



José Lucas Mourão Rangel Netto



Nelson Maculan Filho



Sueli Mendes dos Santos

RIO DE JANEIRO, RJ - BRASIL

NOVEMBRO DE 1978

CAÚLA, LÍGIA BARROS

PLMIX - Um Modelo de Linguagem de Médio Nível e seu Compilador|Rio de Janeiro| 1978

189 , IX 29,7 cm (COPPE-UFRJ, M.Sc, Engenharia de Sistemas e Computação, 1978).

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

1. Um Modelo de Linguagem de Médio Nível e seu Compilador. I.COPEE/UFRJ. II. PLMIX - Um Modelo de Linguagem de Médio Nível e seu Compilador.

À Meus Pais,

Minha Avó Arlinda,

e Eduardo.

AGRADECIMENTOS

Ao Professor Estevam Gilberto de Simone pelas idéias e paciente orientação ministrada.

Aos Professores José Lucas Mourão Rangel Netto e Jano Moreira de Souza pelas valiosas sugestões.

Ao Professor Gerhard Schwarz e à colega Lilian Markenzon, pelo interesse e colaboração.

Aos colegas do Programa de Engenharia de Sistemas e Computação pelo apoio que recebi.

A meus amigos, em especial meu marido, pelo incentivo.

À COPPE, CAPES e FINEP pelos recursos oferecidos durante a execução deste trabalho.

RESUMO

Para permitir especificar os critérios de projeto para linguagens de programação de médio nível, foi criada a linguagem PLMIX para o computador hipotético MIX , definido por D.E. Knuth.

É apresentada uma conceituação de linguagens de médio nível e propostos critérios que justificam as decisões tomadas na elaboração da sintaxe, sobre os tipos de variáveis, ações semânticas e geração de código.

Paralelamente são apresentadas a estrutura do compilador para a linguagem-usando na análise sintática o método de matriz de transição - e a descrição de algumas rotinas semânticas.

Finalmente é feita uma avaliação da linguagem comparando-se algoritmos escritos em MIXAL, por Knuth, com as traduções para MIXAL dos mesmos algoritmos escritos em PLMIX, e julgando as diferenças encontradas no código.

A B S T R A C T

A language PLMIX for the hypothetical computer MIX created by D.E. Knuth, was defined just only as a means to specify the criteria for designing medium level programming languages.

Besides presenting a definition of medium level programming languages we propose criteria that justify the specific form taken by the syntax of PLMIX, its variable types, semantic actions and code generation.

We present also the structure of the compiler and the description of some semantic routines. The compiler uses the transition matrix method for the parsing.

Finaly to evaluate the performance of PLMIX we compare the length of the code generated by MIXAL with the code generated by PLMIX for exactly the same algorithms.

I N D I C E

CAPÍTULO I - Introdução 1

CAPÍTULO II - Descrição do MIX 5

II.1. A Máquina 5

II.1.1. Memória 5

II.1.2. Registros 6

II.1.3. Indicadores 7

II.1.4. Dispositivos de Entrada e Saída ... 8

II.2. Linguagem de Máquina 9

II.2.1. Formato da Instrução 9

II.2.2. Notação Utilizada 12

II.2.3. Instruções 12

II.2.3.1. Instruções de Carga 12

II.2.3.2. Instruções de Armazenamento 13

II.2.3.3. Instruções Aritméticas .. 14

II.2.3.4. Instruções Imediatas 15

II.2.3.5. Instruções de Comparação. 15

II.2.3.6. Instruções de Desvio 16

II.2.3.7. Instruções de Entrada e Saída 17

II.2.3.8. Instruções de Converção.. 19

II.2.3.9. Outras Instruções 20

CAPÍTULO III - Decrição da Linguagem PLMIX 23

III.1. Estrutura da Linguagem 23

III.2. Notação para a Sintaxe 25

III.3. Especificação da Linguagem 26

CAPÍTULO IV - Estrutura do Compilador 87

IV.1. Aspectos Gerais 87

IV.2. Analisador Léxico 88

IV.3. Analisador Sintático 95

IV.3.1. Método de Compilação 95

IV.3.2. Gramáticas de Matriz de Transição ... 96

IV.3.2.1. Definição 96

IV.3.3. Determinação da Matriz de Transição.. 99

IV.4. Código Objeto e Algumas Rotinas Semânticas.. 99

IV.4.1. Estrutura dos Dados no Compilador ... 100

VI.4.2. Algoritmo do Compilador 109

IV.4.2.1. Algoritmo de "Parsing" 110

IV.4.2.2. Esquema do Programa 110

IV.4.2.3. Algumas Rotinas Semânticas. 113

IV.5. Tratamento de Erros 130

CAPÍTULO V - Avaliação da Linguagem 135

V.1. Exemplo Número 1 135

V.1.1. Programa T 137

V.1.2. Programa em PLMIX para T 139

V.1.3. Programa em PLMIX Estruturando o Algoritmo 141

V.2. Exemplo Número 2 143

V.2.1. Programa S 143

V.2.2. Programa em PLMIX para S 144

V.2.3. Programa em PLMIX Estruturando o Algoritmo 147

V.3. Exemplo Número 3 149

V.3.1. Programa B 149

V.3.2. Programa em PLMIX para B 150

V.3.3. Programa em PLMIX Estruturando o Algoritmo 153

V.4. Exemplo Número 4 155

V.4.1. Programa A 156

V.4.2. Programa em PLMIX para A 158

V.4.3. Programa em PLMIX Estruturando o Algoritmo 162

CAPÍTULO VI - Conclusões 167

APÊNDICE Nº 1 170

APÊNDICE Nº 2 173

BIBLIOGRAFIA 187

C A P Í T U L O I

I.1. INTRODUÇÃO

Projetos de novas linguagens de programação surgem com a evolução da utilização do computador tanto no aspecto quantitativo como qualitativo. Além disso, há uma variedade de características de arquitetura que fazem com que certas linguagens sejam adequadas ou mais facilmente traduzidas em determinadas máquinas, enquanto que para outras arquiteturas seria desejável uma linguagem com estrutura diferente.

Para o usuário final as linguagens tendem a evoluir para o mais alto nível possível, voltadas para as aplicações desejadas e o mais independentes da máquina.

A programação de "software" básico e de suporte tem sido normalmente feita com linguagem dependente da máquina.

Durante muito tempo a linguagem dependente da máquina utilizada era a linguagem montada. Ela permite ao programador uma gerência sobre memória e registros e código eficiente. Porém a sintaxe usual faz com que se produzam textos longos, de difícil leitura e acompanhamento da lógica e depuração demorada.

A linguagem de alto nível impede este gerenciamento pois o "software" necessário tanto para compilação como para execução dispõe da memória e dos registros de maneira variável, imprevisível ou incontrolável para o programador. Além disso estas linguagens não fornecem, em geral, mecanismos para que o programador atue diretamente sobre a máquina, por exemplo, impedindo ou tornando complexa a introdução de código de linguagem montada no texto do programa.

Com os conceitos de programação estruturada e a definição dos tipos básicos de estruturas necessárias e suficientes para se escrever programas estruturados (Wulf [1]) surgiram projetos de linguagens, que visam permitir a criação destas estruturas básicas de modo simples sem perderem as características da linguagem montada em gerência de memória e registros e eficiência de código gerado.

Chamaremos esta classe de linguagens de médio nível apresentando as seguintes características:

- permitem ao programador a gerência de memória e registros, tal como as linguagens montadas;

- deverão procurar englobar todo o potencial das instruções de máquina, seja em comandos estruturados, funções ou introdução direta do código em linguagem montada;

- sua sintaxe é do tipo da sintaxe das linguagens de alto nível, mais difundidas, com blocos, procedimentos, expressões, instruções condicionais, iterativas, de chaveamento e declaração de tipos;

- os tipos permitidos são os previstos pelo

hardware ou estruturados com simplicidade (sem necessidade de descritores) a partir dos tipos simples;

- o código gerado deve ser o mais eficiente e o mais próximo possível da linguagem montada;

- a tradução dos comandos tem uma sequência de códigos fixa e deverá ser apresentada ao programador.

Cumpre ressaltar que a terminologia linguagem de médio nível não é restrita à linguagens com as características acima. Ela é usada para linguagens cuja utilização da memória e registros é feita sem conhecimento do programador , com a tradução variável dos comandos, etc. Entretanto as características da linguagem não permitem que sejam consideradas de alto nível.

O objetivo deste trabalho é estabelecer critérios para o desenvolvimento de projetos de linguagens de médio nível, levando-se em consideração as notas sobre projetos de linguagens escritas por Hoare [2].

Em vez de se fazer uma enumeração desses critérios foi projetada uma linguagem onde a apresentação das instruções, dos tipos permitidos, as justificativas das decisões tomadas e as explicações complementares caracterizam estes critérios.

A escolha da máquina recaiu sobre o MIX (Knuth [3]) por possuir um jogo de instruções que permitem um projeto amplo, por ser utilizado em cursos de graduação e pós-graduação em várias instituições de ensino e por existir um simulador desenvolvido por Markenzon [6] e implementado no

computador MITRA 15 do Laboratório de Automação de Sistemas e Simulação da COPPE-UFRJ, como parte do projeto de um Laborató rio de Ensino de Computação.

CAPÍTULO II

DESCRIÇÃO DO MIX

O MIX é um computador hipotético criado com fins didáticos por Donald E. Knuth [3] e apresentado no livro "Fundamental Algorithms" no capítulo I, seções 1.3.1 e 1.3.2 .

Devido a finalidade proposta o projeto apresenta uma estrutura e uma linguagem simples e de fácil aprendizagem e no entanto poderosa o suficiente para permitir que sejam escritos pequenos programas para a maioria dos algoritmos propostos por Knuth [3~4~5].

II.1. A MÁQUINA

II.1.1. MEMÓRIA

A memória MIX é composta de 4000 palavras, cada uma com 5 bytes mais sinal "+" ou "-".

O byte é a unidade básica de informação sendo capaz de armazenar no mínimo 64 valores e no máximo 100 valores distintos. Esta variação é decorrente da possibilidade do MIX poder ser binário ou decimal. É importante ressaltar que

o programador deve escrever programas que possam ser executados em qualquer implementação, isto é, assumindo que o byte não pode representar mais que 64 valores.

Na implementação do simulador existente no MITRA 15 do Laboratório de Sistemas da COPPE-UFRJ, feita por Markenzon [6] assumiu-se o máximo de 64 valores e assim o byte é composto de 6 bits.

II.1.2. REGISTROS

O MIX possui 9 registros sendo que dois com cinco bytes mais sinal (registro A e registro X) e sete com dois bytes mais sinal (registros I1, I2, I3, I4, I5, I6 e registro J).

- Registro A (rA)

É o registro acumulador no qual são efetuadas das operações aritméticas, de deslocamento e conversão.

- Registro X (rX)

Extensão à direita do acumulador. É usado junto com o registro A para formar dez bytes necessários às operações de multiplicação, divisão e conversão, ou para armazenar informações produzidas por deslocamento à direita do registro A.

- Registros I (rI1, rI2, rI3, rI4, rI5 e rI6)

São registros de índice usados geralmente como contadores e para modificar endereços de acesso à memória.

- Registro J (rJ)

Contém o endereço da instrução seguinte a uma instrução de desvio e é usado basicamente para retorno de subrotina.

II.1.3. INDICADORES

São dois os indicadores:

- bit de overflow (OVF)

Modificado pelas instruções aritméticas, "jump on overflow" (JOV) e "jump on no overflow" (JNOV).

- indicador de comparação

Assume valores menor, igual ou maior e é controlado pelas operações de comparação (CMPA, CMPX e CMPi).

II.1.4. DISPOSITIVOS DE ENTRADA E SAÍDA

Os dispositivos dependem dos existentes na máquina onde será feita a simulação. Para a implementação feita por Markenzon [6] temos as seguintes especificações.

- Disco MIX

As especificações do disco são fornecidas por Knuth na página 362 do livro "Sorting and Searching" [5].

Sua capacidade de armazenamento é de vinte milhões de caracteres, distribuídos em duzentos cilindros de vinte trilhas, cada uma com 5.000 caracteres.

As informações transmitidas em operações de entrada e saída são armazenadas em blocos de cem palavras. A referência a um particular bloco é especificada pelo conteúdo dos dois bytes menos significativos de rX.

- Fita MIX

As características da unidade de fita se encontram na página 320 do livro "Sorting and Searching" [5].

A unidade lê e escreve 800 caracteres por polegada de fita, com velocidade de 75 polegadas por segundo. Isto significa que um carater é lido ou escrito cada 1/60 ms. Cada carretel contém 2.400 pés de fita.

As informações são armazenadas por blocos na fita, sendo que cada instrução de leitura ou impressão acarreta a transmissão de um único bloco. Cada bloco tem 100

palavras e o intervalo entre os blocos é de 480 caracteres.

- Teletipo

Definida conforme as características do modelo ASR33 [7] que apresenta velocidade de impressão de 10 caracteres por segundo. Note-se que cada registro possui 70 caracteres, equivalente a 14 palavras MIX.

A transmissão é feita caracter a caracter, onde cada byte representa um caracter.

- Leitora de Cartões

Definida conforme as especificações de modelo LC300 [7] que lê cartões standard de 80 colunas (16 palavras MIX) à velocidade de 300 cartões por minuto.

A transmissão é feita caracter a caracter.

II.2. LINGUAGEM DE MÁQUINA

II.2.1. FORMATO DA INSTRUÇÃO

A maioria das instruções MIX permite ao programador um acesso direto à partes (bytes) da palavra e para permitir identificar os bytes a serem acessados eles são

numerados da seguinte forma:

0	1	2	3	4	5
±	byte	byte	byte	byte	byte

As palavras que contêm instruções são codificadas segundo o seguinte formato:

0	1	2	3	4	5
±	AA		I	F	C

onde:

- ± AA : endereço.

A instrução MIX utiliza endereçamento direto. O sinal da palavra pertence ao endereço.

- I : especificação de índice

É usado para alterar o valor do endereço . Se I = 0, ± AA é usado sem modificação. Os outros valores permitidos variam de 1 a 6 correspondendo aos registros de índice existentes na máquina. Neste caso o valor do registro de índice indicado é somado algebricamente ao valor de ± AA sendo o resultado usado como endereço. Este processo de indexa

ção ocorre para qualquer instrução.

- F : modificação do código de instrução

Em geral F representa uma especificação de campo (L:R), onde L é o número do byte mais à esquerda e R do byte mais à direita. Para permitir a representação desta especificação F é calculado através da fórmula 8*L + R.

Algumas instruções utilizam o campo F com outro significado tal como número do dispositivo de entrada e saída, modificador do código de desvio incondicional, tornando-o condicional, tipo do deslocamento a ser efetuado, etc.

- C : código de operação

Na descrição da linguagem PLMIX algumas vezes serão referenciadas as instruções de máquina correspondentes às instruções PLMIX. A representação da instrução será feita da forma

OP ± AA, I (F)

onde OP é o mneumônico do código de instrução (c). Admite-se opcionalmente as seguintes alterações:

- se I = 0, pode ser omitido
- se F é a especificação padrão da instrução, pode ser omitido.

II.2.2. NOTAÇÃO UTILIZADA

rR : registros quaisquer

rAX: registros rA e rX formando 10 bytes

i : número de 1 a 6

M : endereço resultante após indexação

C(M): conteúdo da posição de memória M

campo: especificação de bytes de uma palavra

V : valor de um campo específico de C(M)

PC : endereço da próxima instrução

A ← B: A recebe o valor de B.

II.2.3. INSTRUÇÕES

A tabela com os códigos e as especificações padrão de F está no Apêndice número 1.

II.2.3.1. INSTRUÇÕES DE CARGA

"load rR": LDA, LDX, LDi

ação : rR ← C(M)

"load rR negative": LDAN, LDXN, LDiN

ação : rR ← - C(M)

Em todas as operações onde um campo parcial é utilizado o sinal é usado se ele faz parte do campo (especificação (0:k), $0 \leq k \leq 5$), caso contrário assume-se sinal +. O campo é deslocado para a direita do registro onde será carregado.

Na atribuição aos registros Ii como este só tem 2 bytes, se o campo englobar mais de dois bytes, os outros deverão ser necessariamente iguais a zero. Caso contrário a instrução será considerada indefinida.

II.2.3.2. INSTRUÇÕES DE ARMAZENAMENTO

"store rR": STA, STX, STi, STJ

ação: M ← rR

"store zero": STZ

ação: M ← 0

Nas instruções de armazenamento o número de bytes do campo desejado é tomado do lado direito do registro e inserido na posição especificada pelo campo.

Os registros rIi se comportam como se tivessem 5 bytes mais sinal, porém com os bytes 1, 2 e 3 iguais a zero. Para rJ o sinal é sempre positivo e só são alterados os bytes 1 e 2, pois eles representam um endereço.

II.2.3.3. INSTRUÇÕES ARITMÉTICAS

ADD (adição)

ação: rA ← rA + V

Se o resultado precisar de mais de 5 bytes para sua representação, OVF é ligado e rA armazena os cinco bytes à direita do resultado e seu sinal.

SUB (subtração)

ação: rA ← rA - V

Poderá ocorrer transbordo e será tratado como na adição.

MUL (multiplicação)

ação: rAX ← rA * V

Os sinais de rA e rX recebem o sinal algébrico do produto, isto é, "+" se rA e rV são do mesmo sinal e "-" caso contrário.

DIV (divisão)

ação: rA ← rA/V ; rX ← (rAX) MOD V

O sinal de rAX é o de rA. Se V = 0 ou o resultado é maior que 5 bytes rA e rX ficam com valores indefinidos e OVF é ligado. Caso contrário rA recebe o quociente

e o sinal algébrico da operação, rX é substituído pelo resto e o sinal de A (de antes da operação).

II.2.3.4. INSTRUÇÕES IMEDIATAS

"enter rR": ENTA, ENTX, ENTi

ação: $rR \leftarrow M$

Se $M = 0$ o sinal da instrução é carregado.

"enter negative rR": ENNA, ENNX, ENNi

ação: $rR \leftarrow -M$

"increment rR": INCA, INCX, INCi

ação: $rR \leftarrow rR + M$

"decrement rR": DECA, DECX, DECi

ação: $rR \leftarrow rR - M$

II.2.3.5. INSTRUÇÕES DE COMPARAÇÃO

"compare rR": CMPA, CMPX, CMPi

ação: comparar um campo especificado de rR com o mesmo campo de C(M).

Se $rR > C(M)$ então $CI \leftarrow$ MAIOR

Se $rR < C(M)$ então $CI \leftarrow$ MENOR

Se rR = C(M) então CI ← IGUAL

Uma comparação com campo (0:0) sempre fornece re sultado IGUAL pois + 0 é igual a - 0.

II.2.3.6. INSTRUÇÕES DE DESVIO

O registro J é alterado toda vez que ocor rer um desvio (exceto na instrução JSJ), recebendo o endereço da instrução seguinte aquela onde se deu o desvio.

JMP (jump)

ação: rJ ← PC;

PC ← M;

JSJ (jump save J)

ação: rJ inalterado

PC ← M

JOV (jump on overflow)

ação: se OVF = true então rJ ← PC

PC ← M

senão;

JNOV (jump on no overflow)

ação: se OVF = false então rJ ← PC

PC ← M

senão OVF ← false

JL, JE, JG, JGE, JNE, JLE (jump on less, equal, greater, non-less, non-equal, non-greater).

ação: o desvio ocorre se o indicador de comparação apresenta a condição indicada.

JAN, JAZ, JAP, JANN, JANZ, JANP (jump A negative, zero, positive, nonnegative , nonzero, nonpositive).

JXN, JXZ, JXP, JXNN, JXNZ, JXNP (jump X negative, zero, positive, nonnegative, nonzero, nonpositive).

JiN, JiZ, JiP, JiNN, JiNZ, JiNP (jump Ii negative, zero, positive, nonnegative, nonzero, nonpositive).

ação: o desvio ocorre se o conteúdo de rR satisfizer a condição, caso contrário nada ocorre.

II.2.3.7. INSTRUÇÕES DE ENTRADA E SAÍDA

IN (entrada)

ação: é iniciada a transferência de informação da unidade especificada para posições consecutivas da memória a partir de M. O número de células da memória afetada é igual ao tamanho do bloco desta unidade.

OUT (saída)

ação: são transferidas informações a partir da posição M da memória para o dispositivo indicado.

Para IN e para OUT a máquina aguarda a liberação do dispositivo se ele estiver executando uma operação. A transferência de informação dura um intervalo de tempo que depende da velocidade da unidade que está sendo utilizada. Assim não é aconselhável fazer referências, sem precauções, às posições de memória que estão sendo alteradas, até que a operação seja completada.

IOC (controle de entrada e saída)

ação: uma operação de controle é executada, dependendo do dispositivo:

Fita magnética: Se M = 0 a fita é reenrolada.

Se M < 0 a fita volta M registros ou volta para o início da fita, o que ocorrer primeiro.

Se M > 0 a fita avança (a operação será ignorada e a unidade suspensa se avançar mais do que o último registro escrito na fita).

Disco: M deve ser zero. Posiciona o dispositivo de acordo com rX para economizar tempo da próxima instrução IN ou OUT.

JRED (jump ready)

ação: o desvio ocorre se a unidade especificada terminou a operação iniciada por IN, OUT ou IOC.

JBUS (jump busy)

ação: o desvio ocorre se a unidade requerida não estiver liberada.

II.2.3.8. INSTRUÇÕES DE CONVERSÃO

NUM (conversão a numérico)

ação: converte código de caracteres para código numérico. rAX é considerado um número de 10 bytes escrito em caracteres que são convertidos a um número decimal armazenado em rA. O valor de rX e o sinal de rA permanecem inalterados. É possível ocorrer tansbordo e neste caso o resto é armazenado.

CHAR (conversão a caracteres)

ação: converte código numérico a código de caracteres, necessário para a saída em cartões ou teletipo. O valor contido em A é convertido a um número decimal de 10 bytes em rAX. Os sinais de rA e rX não sofrem alterações.

II.2.3.9. OUTRAS INSTRUÇÕES

MOVE

ação: transfere o número de palavras especificadas por F começando da posição de memória M para a posição de memória dada pelo conteúdo de rI1. É transferida uma palavra de cada vez e rI1 é incrementado do valor de F ao fim da operação. Se F = 0 nada acontece.

SLA, SRA, SLAX, SRAX, SLC, SRC (shift left A, shift right A, shift left AX, shift right AX, shift left AX circular, shift right AX circular).

ação: rR é movimentado o número de bytes especificado por M.

Os sinais de rA e rX não são afetados nas operações. SLA e SRA não afetam rX, os outros operadores alteram rX. Nas instruções SLA, SRA, SLAX e SRAX bytes "desaparecem" de um lado enquanto zeros são colocados do outro lado . SLC e SRC efetuam deslocamentos circulares, isto é, os bytes que "desaparecem" de um lado, "surgem" do outro.

NOP (no operation)

ação: nada ocorre

HLT (halt)

ação: a máquina para

As instruções abaixo não constam da definição original. Foram acrescentadas no simulador projetado por Markenzon [6] visando facilitar a depuração e a medida do desempenho dos programas. Para estas instruções o relógio do simulador não é incrementado, não afetando assim a medida do tempo de processamento.

Para maiores detalhes sobre estas instruções ver páginas 20, 64 e 65 da referência [6].

CLOC

ação: o relógio MIX é impresso

LOOP

ação: o campo (18 bits) formado pelas partes AA e I da instrução é incrementado de 1 e o novo valor guardado no mesmo campo desta instrução.

OUTL

ação: imprime o campo formado pelas partes AA e I do endereço mencionado.

TRCE (trace)

ação: imprime a partir deste ponto, em cada instrução: a instrução corrente, e os conteúdos dos registros rA, rX, rI1, rI2, rI3, rI4, rI5, rI6.

NTRC (no trace)

ação: desligar o rastreamento, se ligado. Caso contrário nada ocorre.

C A P Í T U L O III

DESCRIÇÃO DA LINGUAGEM PLMIX

III.1. ESTRUTURA DA LINGUAGEM

PLMIX é uma linguagem de médio nível com as características apresentadas na introdução (páginas 2 e 3) e com uma sintaxe semelhante a do ALGOL 60 [21].

Um programa PLMIX é um bloco, isto é, é um conjunto de instruções, precedidas por declarações e encerradas entre 'begin' e 'end'.

As declarações fornecem informações sobre as variáveis e estabelecem a alocação de memória. A alocação é estática e a semântica de cada declaração especifica como é feita a ocupação de memória para aquela variável, permitindo assim ao programador uma gerência sobre os endereços.

Os tipos de variáveis são os existentes no 'hardware' e alguns facilmente estruturados a partir destes. Esta exigência é para evitar que seja necesário o uso de descritores e instruções extras introduzidas no código para cálculo de acesso a endereço. Deste modo é evitado o uso de posições de memória pelo compilador sem o conhecimento do

programador.

Os procedimentos podem ou não ter parâmetros e o corpo é um bloco ou um comando composto ou apenas uma instrução. Quando houver declarações de variáveis estas são locais ao procedimento, havendo verificação de escopo. Não é permitida a declaração de procedimento no corpo de outra, porém é permitida a chamada desde que esta já tenha sido declarada. A passagem dos parâmetros é por valor-resultado ou por valor [24]. Na chamada os parâmetros formais são inicializados com os valores dos parâmetros reais correspondentes. No retorno os parâmetros reais do tipo valor-resultado, recebem o valor do parâmetro formal correspondente. Os demais permanecem inalterados.

As instruções podem ser compostas, encerradas entre 'begin' e 'end' ou '[' e ']'. Algumas refletem diretamente instruções da máquina, como por exemplo, as instruções de deslocamento, aritméticas, transferência, conversão. Outras são compostas por várias instruções, como os comandos iterativos, condicionais, de chaveamento (CASE), porém é definida na semântica uma tradução fixa de modo que o programador possa saber exatamente o código gerado por determinada instrução.

Os procedimentos podem ser chamados de qualquer ponto do programa.

A atribuição de expressões às variáveis de memória é feita indicando-se na sintaxe em que registro será efetuado o cálculo, para evitar que sejam tomadas deci

sões pelo compilador. É feita exceção apenas para cálculo de expressão em uma condição. Porém a ausência da especificação do registro indica que será utilizado o registro acumulador (rA).

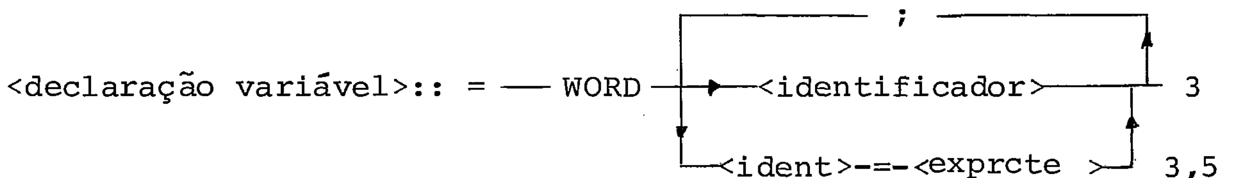
As instruções introduzidas por Markenzon [6] no simulador também tem declarações e instruções correspondentes em PLMIX.

Não há necessidade da introdução de código montado no programa pois todas as instruções de máquina do MIX tem, direta ou indiretamente, equivalente em PLMIX.

III.2. NOTAÇÃO PARA A SINTAXE

A descrição das categorias sintáticas da linguagem utiliza uma variação da forma normalizada de Backus. Sua principal característica é diminuir o número de categorias sintáticas.

O exemplo abaixo utiliza todas as regras da representação.



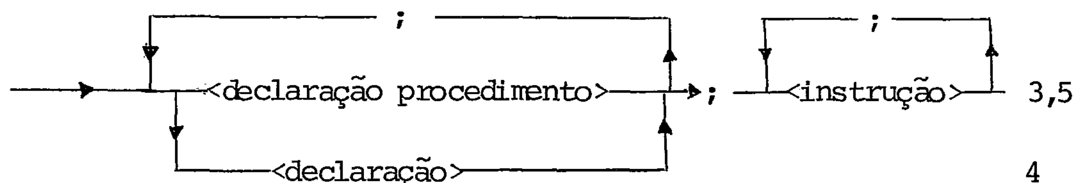
- as categorias sintáticas são as palavras delimitadas por "<" e ">".
- as palavras reservadas são escritas com letras maiúsculas.
- o sinal :: = é lido como "é definido por".
- o sinal → é lido como "é seguido por".
- o sinal —↑— é lido como "ou seguido por".
- o(s) número(s) ao lado de cada linha indica(m) o(s) número (s) da(s) definição (ões) da(s) categoria(s) sintática (s) referenciada(s) naquela linha.

III.3. ESPECIFICAÇÃO DA LINGUAGEM

1 <programa>:: =

——BEGIN ————<lista>————END 2

2 <lista>:: =



O corpo do programa é constituído de declarações separadas por ";" e seguidas por instruções também separadas por ";".

As declarações dão informações sobre as características das variáveis que serão utilizadas no programa - tipo, área de memória associada, valor inicial, etc. É feita uma associação entre o identificador e uma ou mais posições de memória, um registro, um grupo de instruções, uma tabela de endereços, etc.

Um identificador só pode aparecer uma vez nas declarações "globais". Dentro da declaração de procedimento é permitido declarar variáveis com um nome já declarado , pois elas são locais ao procedimento.

Todas as variáveis devem ser declaradas.

A instrução é a unidade de operação da linguagem. O conjunto das instruções é a representação do algoritmo proposto para resolução de um problema.

Normalmente as instruções são executadas sequencialmente, podendo esta ordem ser modificada por um comando explícito de desvio ou uma alteração do resultado de um teste de condição em um comando condicional. Um desvio incondicional é feito indicando-se o rótulo da instrução a ser executada após o desvio.

3 <declaração procedimento>:: =

→ PROCEDURE → <identificador> → ; <corpoproc> 6,7

└→ (-<lista param. formais>-) ┘ 8

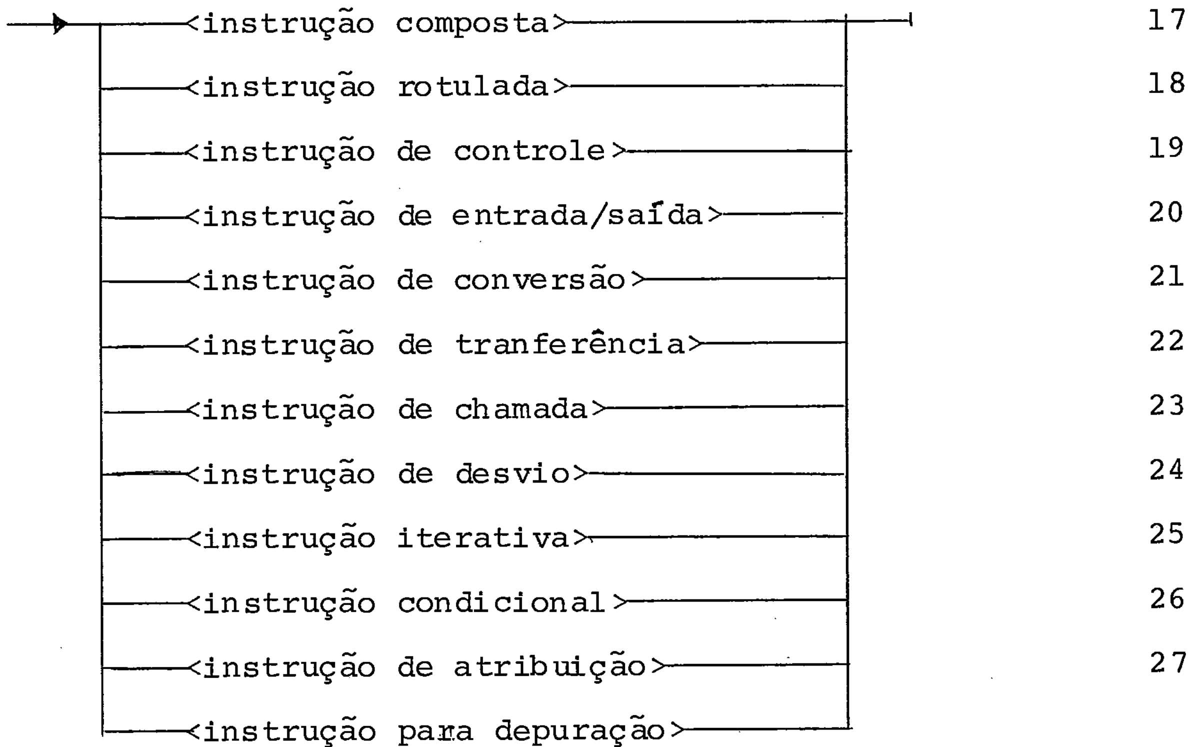
A declaração de procedimento associa um identificador a uma instrução ou a uma sequência de instruções. O identificador pelo qual o procedimento é referenciado é o que aparece na declaração.

Não existe o procedimento tipificado (função). Isto é devido ao fato de, para o cálculo da função que está sendo chamada em uma expressão, haver a possibilidade de estarem sendo usados e eventualmente alterados, sem estar visível para o programador, os mesmos registros que aparecem nesta expressão. Para evitar estes problemas seria necessário guardar os valores destes registros em variáveis temporárias antes de executar a função e restaurá-los após o término do cálculo. Este tipo de solução não se enquadra nas características propostas para linguagem de médio nível, por não ser permitido o uso de temporárias pelo compilador.

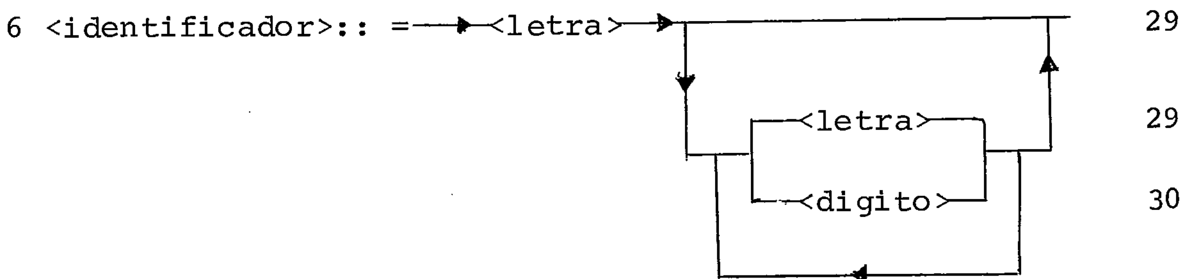
4 <declaração>: : =

- <declaração de constante> — 9
- <declaração de variável simples> — 10
- <declaração de record> — 11
- <declaração de variável equate> — 12
- <declaração de vetor> — 13
- <declaração de rótulo> — 14
- <declaração de tabela de desvios> — 15
- <declaração de contadores> — 16

5 <instrução>:: =

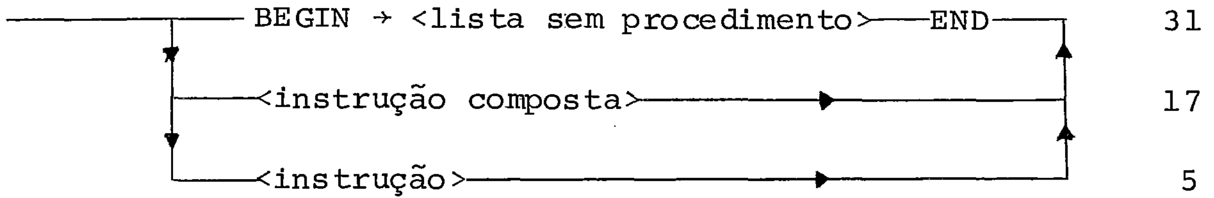


6 <identificador>:: =



O identificador deve começar por uma letra e pode ter até 10 caracteres, entre letras e digitos.

7 <corpo do procedimento>: : =



Um procedimento pode ou não conter declaração de variáveis. Se houver declarações estas serão locais ao procedimento e cada referência à variável é testada para verificação do escopo. A alocação destas variáveis é estática.

A alocação de memória por um procedimento é feita do seguinte modo:

1. uma palavra para cada parâmetro formal
2. área para as variáveis locais (se houver)
3. uma instrução para aguardar o endereço de retorno
4. instruções do procedimento
5. instrução de desvio para a instrução após à chamada ao procedimento.

A necessidade da reserva de palavras para os parâmetros é devido ao método de transmissão de parâmetros: a) na chamada ao procedimento o parâmetro real é calculado e seu valor é o valor inicial do parâmetro formal correspondente; b) após a execução os parâmetros reais correspondentes aos parâmetros formais de tipo VALUE não são alterados, os demais

recebem o valor final dos parâmetros formais correspondentes.

Deste modo a transmissão dos parâmetros pode ser "by value" ou "by value-result". O formalismo das definições destes métodos se encontra em Pratt [24] nas seções 6.9 e 6.10.

Estes métodos foram adotados devido ao fato do MIX não possuir registros de indireção. Neste caso o uso de passagem de parâmetros "by reference" se tornaria excessivamente oneroso, pois a cada acesso ao parâmetro teríamos necessariamente uma instrução de carga. A transmissão de parâmetros "by name" é claramente incompatível com os objetivos propostos para linguagens de médio nível, além de apresentar problema semelhante.

De qualquer forma, o uso de subprogramas com parâmetros é claramente desaconselhável, devendo o programador utilizar os registros da máquina para esse fim.

Não é permitida a declaração de procedimento dentro do corpo de outro procedimento. Porém é permitida a chamada a um outro procedimento previamente declarado, não se permitindo recursão. A responsabilidade por evitar recursão fica a cargo do programador.

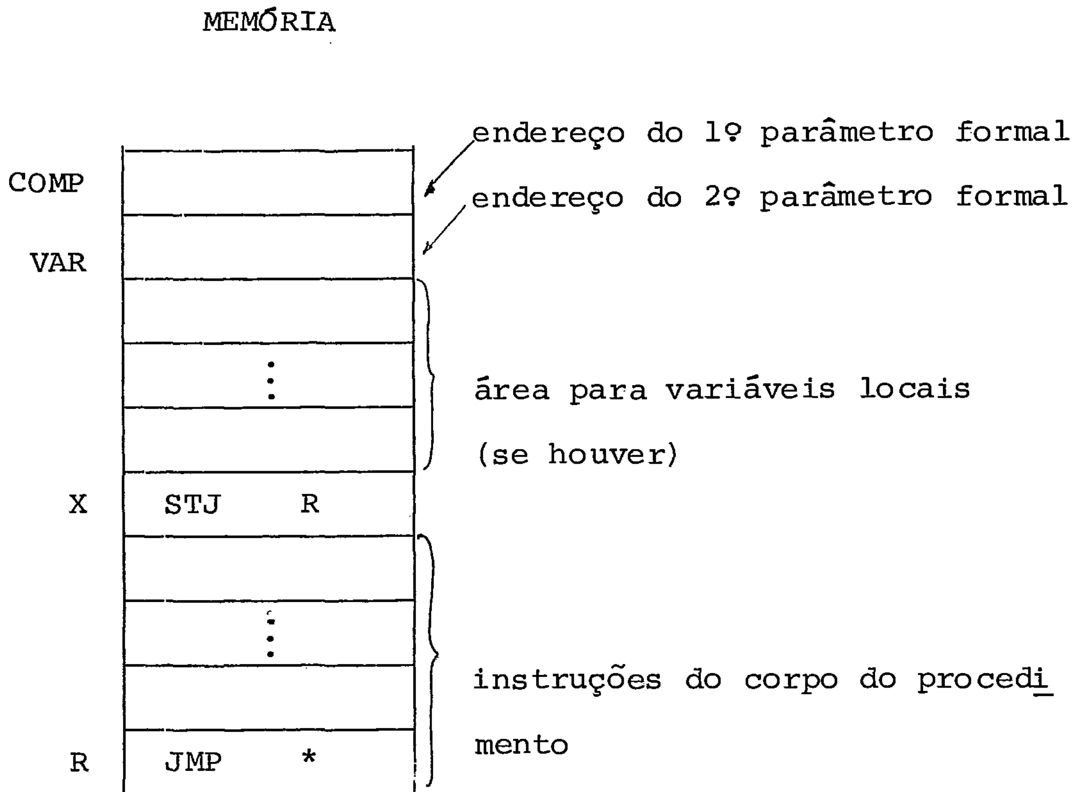
Exemplo:

```
PROCEDURE TESTE (VALUE WORD COMP; WORD VAR);
     BEGIN
      <lista sem procedimento>
     END
```

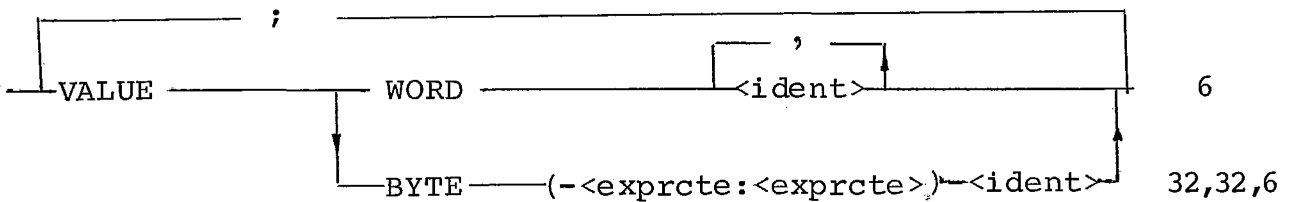
Após a compilação teremos:

TESTE na tabela de símbolos aponta para X.

MEMÓRIA



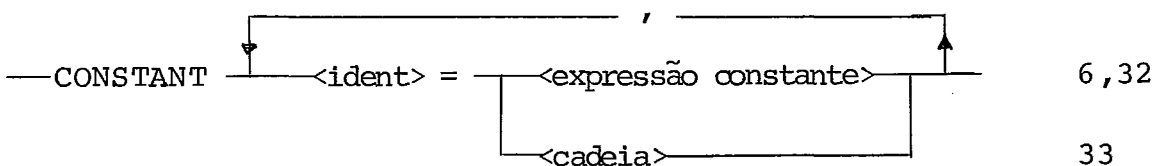
8 <lista dos parâmetros formais>:: =



A palavra VALUE indica transmissão "by value". O parâmetro sem esta especificação é transmitido "by value-result".

Não foi permitido o uso de vetores como parâmetros pois, com o método de transmissão adotado, significaria incentivar desperdício de memória em casos que poderiam ser solucionados de forma mais economica através da utilização adequada de registros, o que acontece com grande frequência.

9 <declaração de constante>:: =



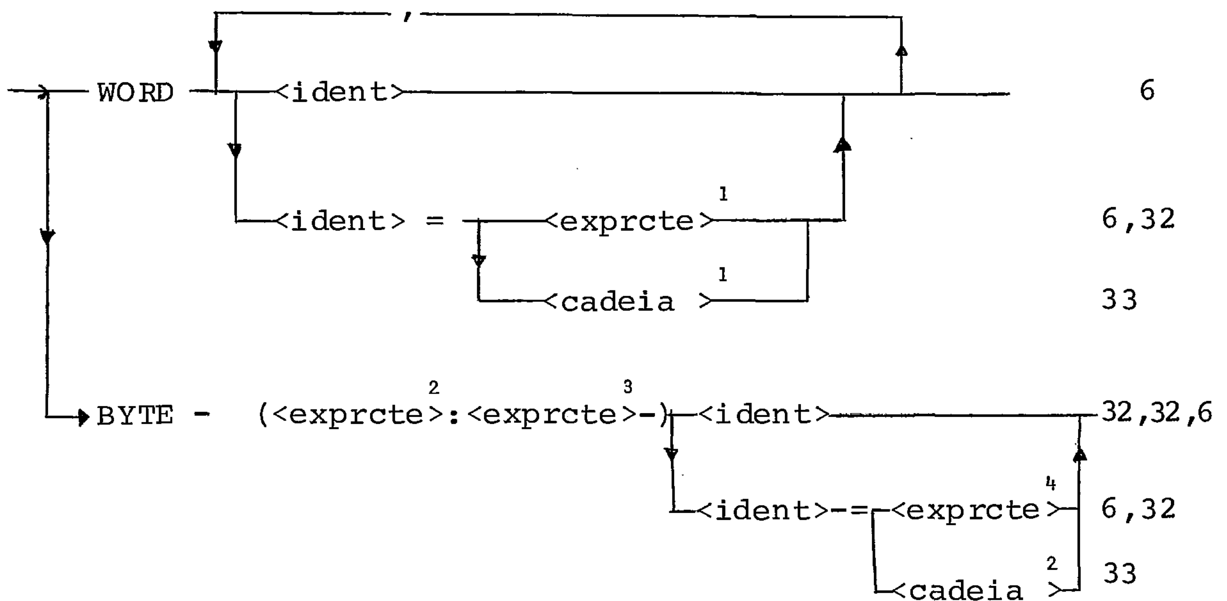
A declaração de constante associa um identificador a um valor, sem alocação de memória. A inicialização de constante obedece às mesmas regras de inicialização de variável tipo WORD que serão vistas em (10).

Exemplo

CONSTANT DEZ = 10, PRINTER = 18;

A compilação desta declaração atribui valor ao identificador na tabela de símbolos e equivale à pseudo-instrução "CON" do assembler MIX.

10 <declaração de variável simples>:: =



A declaração de variável associa um identificador a uma posição de memória podendo ou não ser inicializada.

A variável tipo WORD ocupa os 5 bytes e mais o byte de sinal.

Se a <exprcte>[1] necessitar de mais de 5 by-

tes para sua representação o compilador envia uma mensagem de erro. A inicialização com uma cadeia é feita alocando-se cada caracter em um byte da esquerda para a direita. Se a cadeia tiver mais de 5 caracteres só serão utilizados os 5 primeiros. Se tiver menos de 5, os bytes não inicializados ficam com valores indefinidos.

A variável simples tipo BYTE ocupa uma posição de memória mas somente um ou alguns bytes contíguos da palavra estão associados ao identificador. A <exprcte>$_2$ indica o byte mais à esquerda e <exprcte>$_3$, o byte mais à direita. Assim devemos ter a relação <exprcte>$_2$ $\leq$ <exprcte>$_3$ conforme a numeração dos bytes definida em III.2.1.

Na inicialização de uma variável BYTE somente os bytes especificados recebem o valor a ser atribuído ficando os outros bytes com valores indefinidos.

O compilador envia mensagem de erro para os seguintes casos: a) tentativa de atribuir um número com sinal a uma variável BYTE onde o byte de sinal (byte Ø) não faz parte da especificação do campo; b) inicializar somente o byte Ø; c) inicializar com um número cuja representação exige um número de bytes maior que o campo.

O byte Ø da palavra de memória ficará com valor indefinido se a inicialização for com uma cadeia.

O conteúdo de uma posição de memória não inicializada é indefinido.

Exemplo:

BYTE (0:3) AB;

0	1	2	3	4	5
?	?	?	?	?	?

AB

BYTE (3:5) COD = 'A';

?	?	?	A	?	?

COD

BYTE (4:4) INIC = 1;

?	?	?	?	1	?

INIC

WORD A1, B = 5, NOME='ABCDEFG';

?	?	?	?	?	?	A1
+	0	0	0	0	5	B
?	A	B	C	D	E	NOME

? = valor indefinido

11 <declaração de record>:: =

```
→RECORD ———<ident>¹——— : ——┬——<corpo do record>————┬—6,34
                           └——STRUCTURE —<ident>²———┘  6
```

Um record associa uma ou mais palavras a um identificador. Cada uma dessas palavras terá um identificador que permite seu acesso individual. Cada palavra poderá ainda ter bytes contíguos referenciados por outros identificadores. Esta hierarquização será explicada em (34).

A cláusula STRUCTURE permite a criação de um novo record com as mesmas características do record de nome <ident>[2]. Assim para haver uma distinção dos subcampos do record, cada um terá seu nome composto: <ident.record>.<ident nível>* (* será visto em (34)).

Exemplos:

```
RECORD      NO:
   .1       WORD  INFO
   .1       WORD  PONT
            .2 BYTE (1:2) ESQ
            .2 BYTE (3:4) DIR
RECORD      NO1: STRUCTURE NO
```

12 <declaração de variável equate>: : =

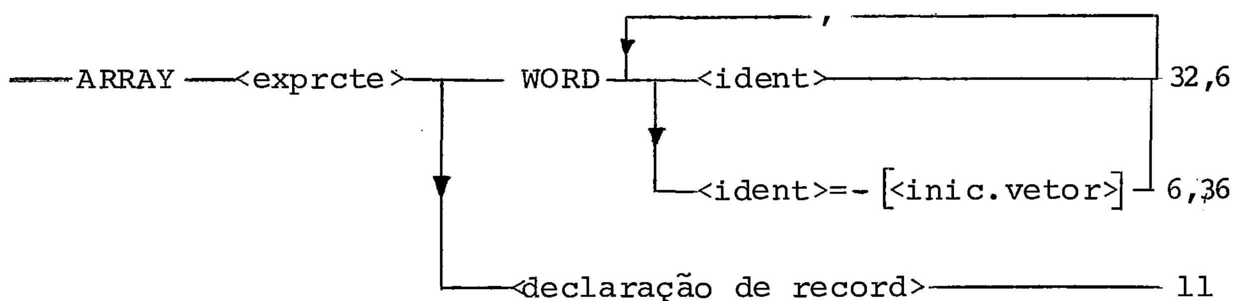
```
                  ┌────────────── , ──────────────────────────┐
                  ▼                                           ▲
──▶EQUATE ────────┴── <ident> ────── = ─┬──── RAC ──────┬─────┴──      6
                                       ▼                ▲
                                       └──── <regix> ───┘              35
```

A declaração de equate associa um identificador a um registro. Este identificador não aloca memória . Na compilação qualquer referência a uma variável deste tipo é tratada como uma referência ao registro associado.

Exemplo:

EQUATE INDICE = RI1, ACUMULADOR = RAC

13 <declaração de vetor>:: =



Os arranjos são unidimensionais, com limi te inferior igual a zero (Ø) e limite superior igual ao valor da <exprcte> menos 1.

A restrição a arranjos unidimensionais obe dece à filosofia da linguagem de não utilizar palavras da memó ria sem expresso conhecimento do programador.

O uso de arranjos multidimensionais impli caria, necessariamente, em utilizar memória para descritores e /ou gerar instruções adicionais para acesso, além de implicar obviamente em problemas de otimização de código.

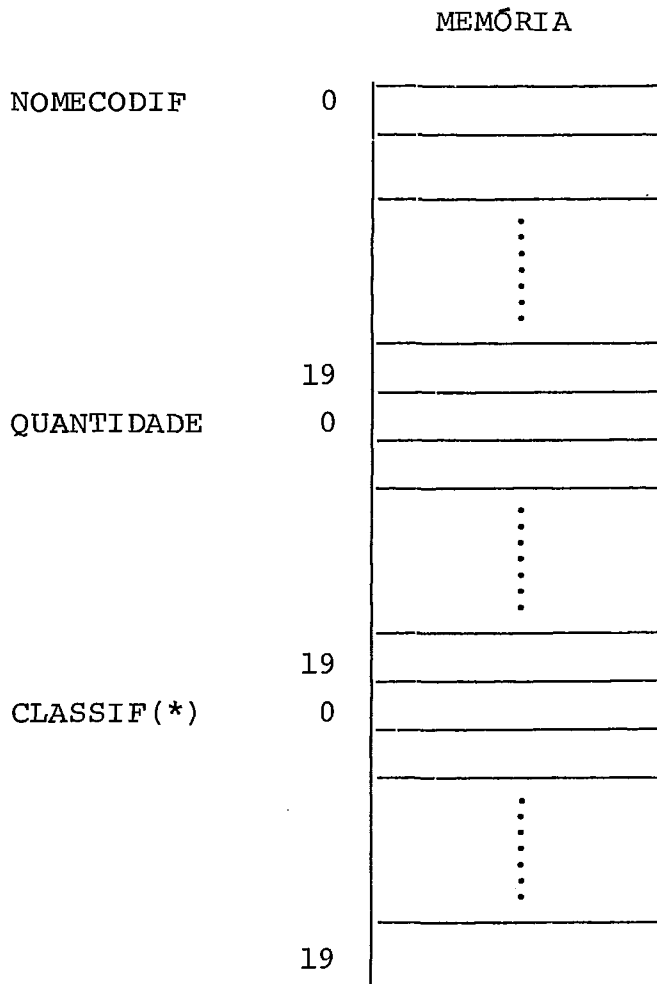
Se o número de palavras a serem inicializa das for menor que o alocado pelo vetor, as palavras não inicia lizadas tem valor indefinido. Se o número for maior será envia da uma mensagem de erro.

Os arranjos de tipo record são armazenados

de forma a evitar o uso de descritores e geração de instruções para cálculo de acesso. Cada nível é considerado como um vetor em separado. Deste modo o acesso à palavra de índice I de um determinado nível é feito atribuindo-se I a um registro de índice e usando como endereço base o endereço do nível. Temos assim tantos arranjos quanto o número de palavras da estrutura.

Exemplos:

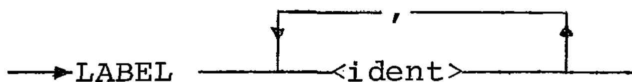
```
CONSTANT     VINTE = 20
ARRAY        VINTE    WORD  ITENS
ARRAY        8        WORD  COD = [1,*2[10],-16,'A']

ARRAY        20      RECORD   MATERIAL:
                    .1 WORD      NOMECODIF
                    .1 WORD      QUANTIDADE
                    .1 WORD      CLASSIF
                       .2  BYTE (1:2) TIPO
                       .2  BYTE (3:5) ESPECIE
```



(*) o endereço de CLASSIF é o mesmo de TIPO e ESPECIE.

Alocação de memória para a declaração do vetor do record.

14 <declaração de rótulo>:: =



6

Os rótulos associam um identificador a uma posição de memória que contém uma instrução. Eles são usados em instruções de desvio incondicional (GOTO).

Exemplo:

LABEL   AQUI, FIM.

15 <declaração de tabela de desvios>:: =

——▸CASE ————<ident>——:——<exprcte>————   6,32

Para utilizarmos a instrução de CASE-OF é necessário dispormos de uma tabela com os endereços de cada uma das instruções correspondentes a cada um dos valores do desvio.

Como toda utilização de memória deve ser alocada pelo programador é necessário que ele indique ao compilador a reserva de área necessária para a tabela dos endereços.

A <exprcte> indica o número de opções dentro do CASE-OF porém é utilizada uma palavra a mais. Sendo EØ, E1,... E_{n-1} os endereços de desvio para a primeira, segunda , ..., enésima instrução, E_n é o endereço da primeira instrução após o CASE-OF.

Cada instrução CASE-OF deve ter uma declaração associada.

Exemplo:

CASE   CASO1 : 10

16 <declaração de contadores>:: =

6

Um contador é usado como um rótulo e pode ser referenciado como tal.

Sua função é computar quantas vezes foi executada a instrução à qual ele está associado.

Ao encontrar um contador o compilador gera a instrução LOOP antes da instrução a ser controlada. A instrução tem um formato diferente das instruções do "assembler" MIX. O campo ± AA é integrado ao campo I, formando um campo com 3 bytes, que é inicializado com zero. Durante a execução do programa a cada passagem pela instrução é incrementado o campo do contador de 1. Esta instrução não influencia no tempo do relógio simulado.

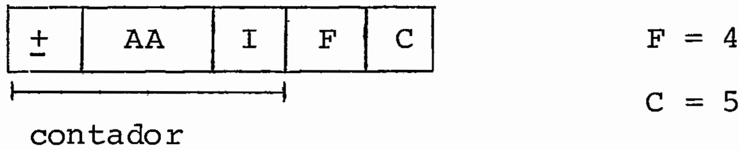
A impressão deste contador através da instrução OUTCOUNTER permite descobrir, por exemplo, quantas vezes foi executado determinado laço de iteração.

Esta facilidade não faz parte da definição do MIX feita por Knuth [3]. Ela foi implementada por Markenzon [6] no simulador do MIX.

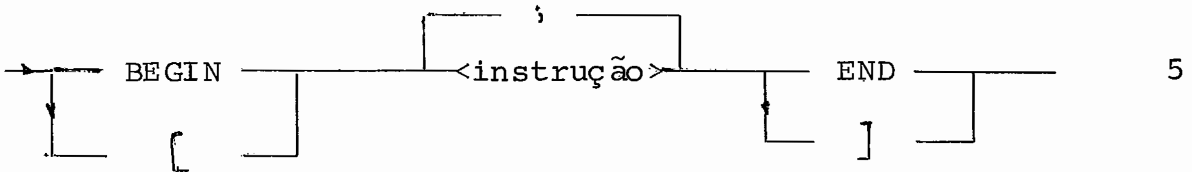
Exemplo:

```
COUNTER    LOOP1, LOOP2
```

formato da pseudo instrução

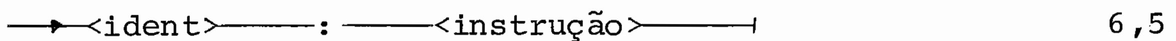


17 <instrução composta>: : =



A instrução composta reúne uma ou mais instruções entre delimitadores BEGIN-END ou [-], de modo que elas possam ser tratadas como uma única instrução.

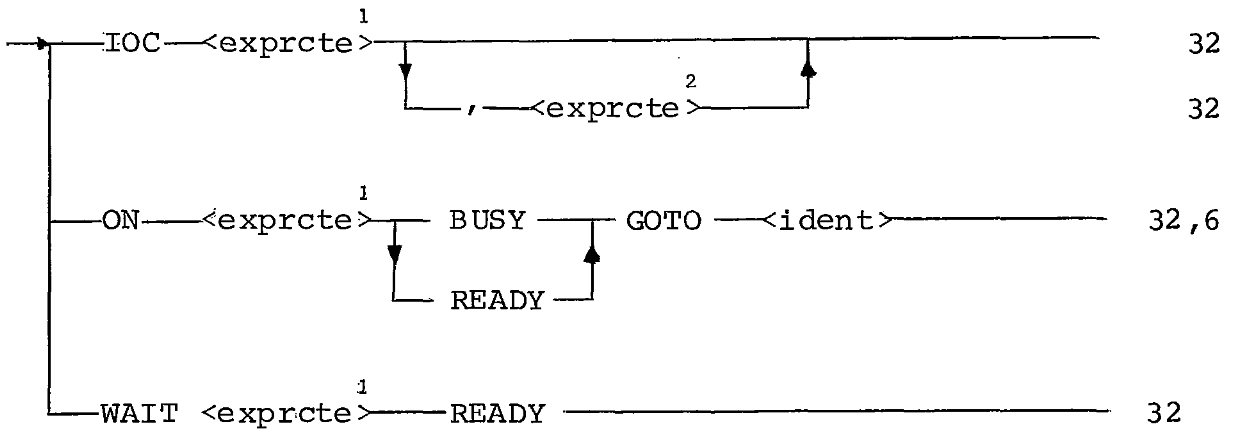
18 <instrução rotulada>: : =



A instrução rotulada permite que uma instrução possa ser referenciada e que ocorra um desvio da sequência de execução das instruções através de um desvio incondicional.

Se <ident> for um contador, a cada passagem por <instrução>, durante a execução do programa, o contador será incrementado de 1.

19 <instrução de controle>: : =



A instrução IOC indica algumas operações de controle sobre determinado periférico. <exprcte[1] > indica o periférico a ser tratado e <exprcte>[2] informa sobre a operação a ser efetuada. A tradução deste comando é a instrução assembler IOC com ± AA = <exprcte>[2] e F = <exprcte>[1].

A opção ON faz com que ocorra um desvio para a instrução de rótulo <ident> dependendo do estado do periférico: livre ou ocupado.

A opção WAIT representa uma espera já que é feito um desvio para a mesma posição do teste. Logo a condição do desvio deve ser trocada.

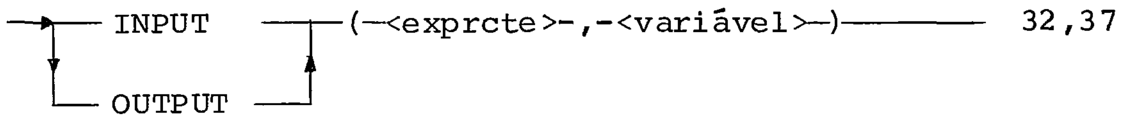
ON e WAIT geram as mesmas instruções JBUS e JRED, diferindo apenas os endereços dos desvios.

Para maiores detalhes ler a seção II.2.3.7.

Exemplos:

PLMIX	MIXAL
IOC 3, -1	IOC -1 (3)
IOC 18	IOC 0 (18)
ON READER READY GOTO VOLTA	JRED 1000(16)
(1000 é o endereço de volta)	
WAIT PRINTER READY	JBUS 500 (18)
(500 é o endereço da instrução JBUS)	

20 <instrução de entrada/saída>:: =



<exprcte> é o periférico onde ocorrerá a operação de entrada/saída. <variável> é nome da área que funciona como "buffer".

Maiores detalhes sobre as operações de entrada/saída estão na seção II.2.3.7.

Exemplos:

INPUT (6, DADOS [RI1]) IN 1000 (6)

(1000 endereço de DADOS [RI1])

OUTPUT (PRINTER,RESULT [0,RI3]) OUT 2000 (18)

(2000 endereço de RESULT [0])

21 <instrução de conversão>:: =

```
───┬── CHAR ──┬──
   │          │
   └── NUM ───┘
```

CHAR converte código numérico (em rA) para código caracter com o resultado em rA e rX.

NUM converte 10 bytes (rA e rX) de código caracter para código numérico ficando o valor resultante em rA.

Os detalhes destas operações se encontram na seção II.2.3.8.

22 <instrução de transferência>:: =

```
                       1                  2
── MOVE ──<variável>── TO ─┬─<variável>─┬─FOR ─<exprcte>──┤ 37,37,32
                           │            │
                           └── ENDRI1 ──┘
```

A instrução de transferência permite que sejam copiadas informações de uma posição de memória para outra.

Se <variável> for um identificador de variável simples ou indexada são transferidas <exprcte> palavras a partir do endereço de <variável>^{1} para o endereço de <variável>^{2} ou o endereço que está em rI1. Para indicar a segunda opção usa-se a palavra reservada ENDRI1.

Após a execução da instrução, rI1 = endereço de <variável>^{2} + <exprcte> ou rI1 = rI1 + <exprcte>.

Se <variável>^{1} for um identificador de record, MOVE transfere todas as palavras do record. Neste caso <exprcte> deverá ser igual a 1 para evitar erro. No final teremos rI1 = endereço de <variável>^{2} + número de palavras do record.

Se <variável>^{1} for um identificador de vetor de records são transferidos <exprcte> records que correspondem a <exprcte> x (número de palavras do record) posições de memória. Neste caso <variável>^{2} também deverá ser identificador de vetor de record. A opção ENDRI1 não poderá ser usada pois neste caso para gerar código para a instrução é necessário termos o endereço da entrada na tabela de símbolos do nome do record para localizarmos os endereços dos diversos níveis do record.

Exemplos:

PLMIX	MIXAL
MOVE DOADOR[RI3] TO RECEP[RI4] FOR 2	ST1 RECEP, 4
	MOVE DOADOR, 3 (2)
MOVE FONTE[0,RI5] TO ENDR1 FOR 8	MOVE FONTE, 5 (8)

Supondo as seguintes declarações:

```
ARRAY 4 RECORD  ARVORE:
   .1  WORD  INFO
   .1  WORD  PONT
       .2 BYTE (1:2) ESQ
       .2 BYTE (4:5) DIR
ARRAY 4 RECORD  NOVAARV: STRUCTURE ARVORE
```

e a instrução

MOVE ARVORE[RI5] TO NOVAARV[RI6] FOR 4	ST1 N1,6
	MOVE A1,6 (4)
	ST1 N2,6
	MOVE A2,6(4)

Supondo:

A1 = endereço ARVORE.INFO

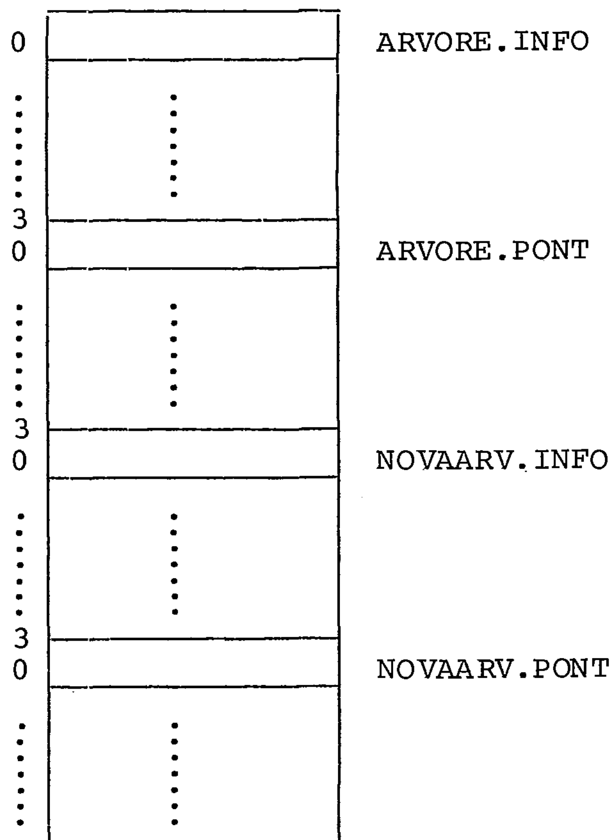
A2 = endereço ARVORE.PONT

N1 = endereço NOVAARV.INFO

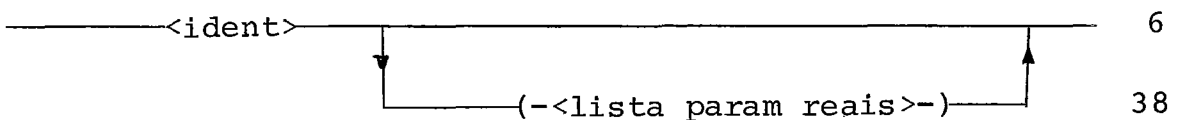
N2 = endereço NOVAARV.PONT

Para melhor compreensão deste último código gerado será mostrada a memória para as declarações dos records acima.

MEMÓRIA



23 <instrução de chamada>:: =



A chamada a um procedimento pode ocorrer em qualquer ponto do programa, inclusive numa declaração de procedimento. Não é permitida uma chamada recursiva, porém não é feito teste para deteção deste erro.

Se não há passagem de parâmetros a chamada se reduz a uma instrução de desvio incondicional para o início do corpo do procedimento.

Havendo passagem de parâmetros a inicialização dos parâmetros formais, e as atribuições de retorno dos parâmetros de tipo "value-result", se faz através de rA. Neste caso a instrução de chamada se traduz por: a) atribuição a cada um dos parâmetros formais; b) instrução de desvio para o início do corpo do procedimento; c) atribuição aos parâmetros transmitidos "by value-result".

Exemplos:

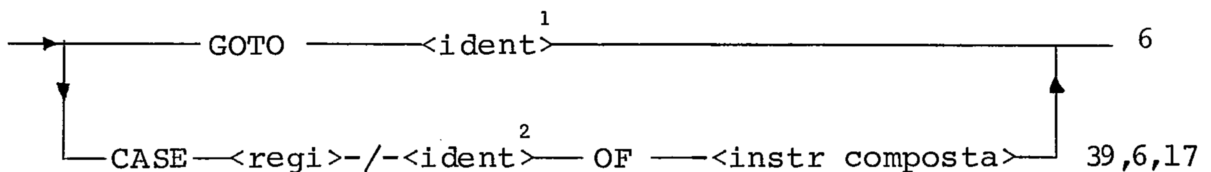
na declaração:

```
PROCEDURE IMP;
<corpo procedimento>

PROCEDURE SOMA (VALUE WORD CONST; WORD RESULT);
<corpo do procedimento>
```

na chamada:

IMP;	JMP	IMP
SOMA(10,VAR)	ENTA	10
	STA	CONST
	LDA	VAR
	STA	RESULT
	JMP	SOMA
	LDA	RESULT
	STA	VAR

24 <instrução de desvio>:: =



A instrução GOTO gera um desvio incondicional para a instrução de rótulo <ident>[1]. Ela pode ocorrer em qualquer ponto do programa, inclusive em um corpo de procedimento, estando o rótulo dentro ou fora dele.

A instrução CASE-OF trata dos desvios através de uma tabela de endereços. <ident>[2] é o nome da tabela e deve estar na declaração de CASE associada.

É feito um teste antes de ser executada a instrução para evitar que seja dado um desvio para uma instru

ção fora do CASE-OF introduzindo erros de maneira incontrolável e às vezes imperceptível para o programador. Deste modo são geradas as seguintes instruções antes do desvio do CASE-OF:

suponha a seguinte declaração:

CASE <ident^1> : <numero>

e a instrução

CASE <regi>/<ident^1> OF <instr composta>

teremos:

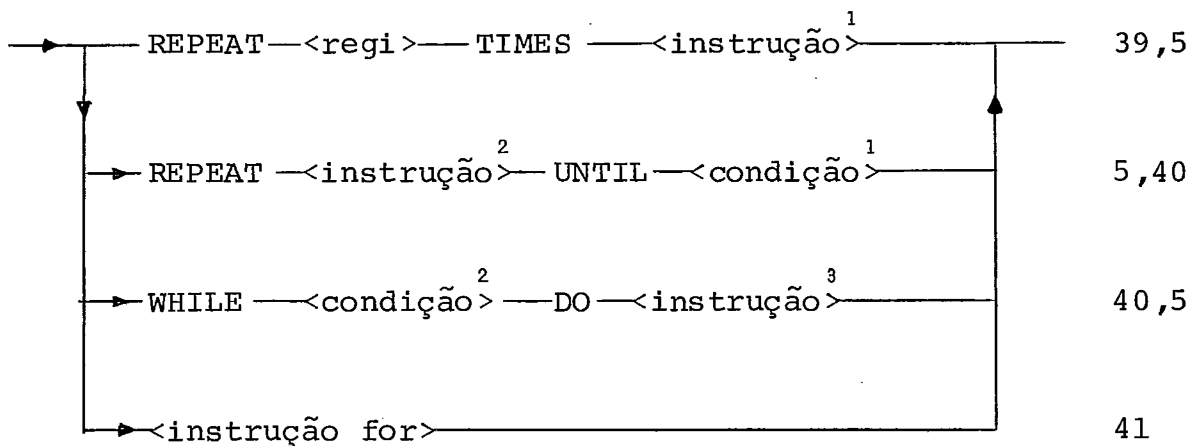
```
     J<regi> NN        * + 2           % testa se ≥ 0
     JMP               ERRO
     DEC<regi>         <numero>        % testa se c(rI:)<numero>
     J<regi> NN        ERRO
     INC<regi>         <numero>        % restaura <regi>
     JMP               * + 2
ERRO ENT<regi>         <numero>        % desvia la. intr após CASE-OF
     JMP           <ident>,<regi>      % desvio para tabela
```

Os valores permitidos para rIi variam de 0 a <numero>-1. Após cada instrução correspondente a uma opção é gerado um desvio para primeira instrução após o CASE-OF.

Exemplos

```
GOTO   SAI                              JMP    SAI

CASE RI2/CASO1 OF                       J2NN * + 2
     BEGIN                              JMP  5F
       VAL,RAC:=3; % RI2=0              DEC2 3
       BEGIN       % RI2=1              J2NN 5F
         RI5: = VAL + MAX;              INC2 3
         VAL: = 0                       JMP  * + 2
     END;                            5H ENT2 3
     VAL,RAC:=27;%RI2=2                 JMP  CASO1,2
                                        <instr 0>
                                        JMP  6F
                                        <instr 1>
                                        JMP  6F
                                        <instr 2>
                                     6H
```

25 <instrução iterativa>:: =



A instrução iterativa permite a execução de uma instrução, em geral composta, um determinado número de vezes sem que estas instruções sejam instruções com contadores ou testes de saída (explícitos). A instrução gera um comando de desvio condicional que é utilizado de modos diferentes dependendo da instrução.

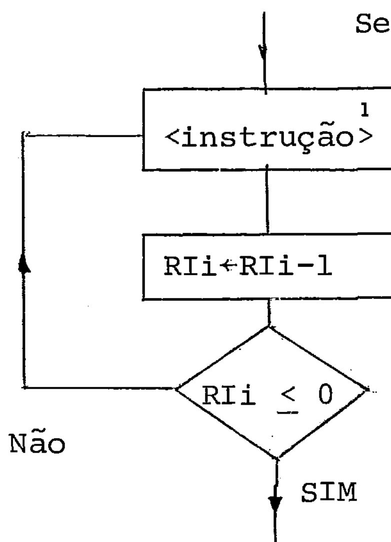
A instrução REPEAT <regi> TIMES <instrução>^{1} faz com que a <instrução>^{1} seja executada um número de vezes igual ao conteúdo do registro Ii associado à instrução.

O registro deve ser carregado antes da instrução. Se o conteúdo do registro for zero ou negativo, <instrução>^{1} será executada uma vez.

Para que o número de iterações seja igual ao valor carregado inicialmente em rIi este não deve ser alterado em <instrução>^{1}. Porém não é feito nenhum teste para verificar a alteração do conteúdo do registro.

A cada execução de <instrução>^{1} o valor do registro é decrementado de 1, interrompendo a iteração quando este valor for zero ou negativo.

Segundo o esquema abaixo a compilação será:

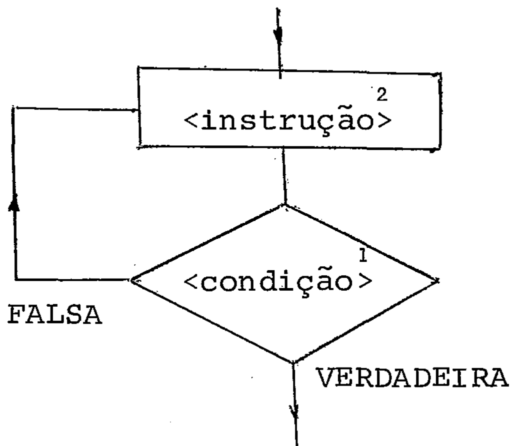



```
XXX <instrução>1
    DEC<regi>1
    J<regi>P  XXXX
```

A instrução REPEAT -<instr>^{2} UNTIL <condição>^{1} faz com que <instrução>^{2} seja executada enquanto a <condição> for falsa, sendo que a instrução é executada pelo menos uma vez.

Para que a iteração termine é necessário que pelo menos um dos elementos que fazem parte de <condição>^{1} tenha seu valor alterado no corpo de <instrução>^{2}.

Para o esquema da instrução o código gerado será:



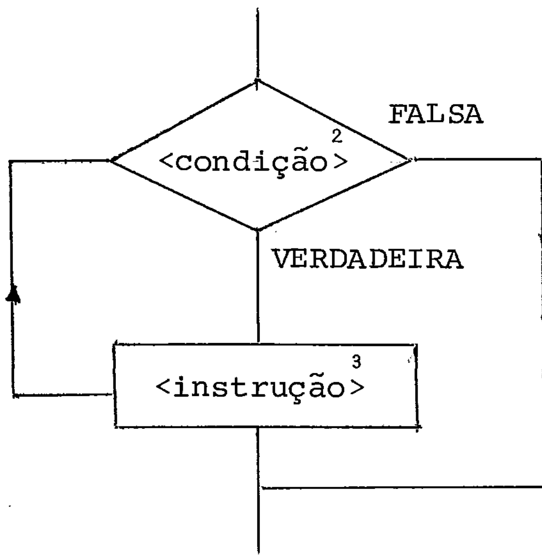

```
XXX <instrução>²
    <instr p/teste condição>*
    J ¬<condição>¹        XXX
```

* as instruções que são geradas para o teste de condição serão vistas em (40).

A instrução WHILE<condição>^{2} DO <instrução>^{3} permite a iteração enquanto a condição for verdadeira. Se no primeiro teste a condição for falsa, <instrução>^{3} não é executada nenhuma vez.

Para que a iteração processe um número finito de vezes é necessário que pelo menos um dos elementos que fazem parte da condição seja alterado.

Com o esquema teremos:




```
XXX <instr p/teste condição>
    J ¬ <condição>[2]    ZZZ
    <instrução>[3]
    JMP    XXX
ZZZ
```

26 <instrução condicional>:: =

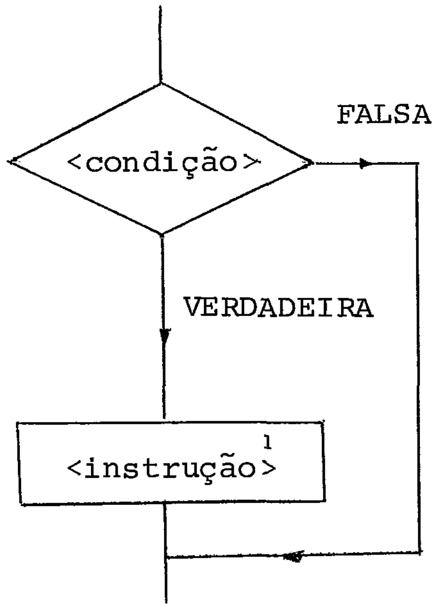
```
—►—IF —<condição>— THEN —<instrução>[1]—┬————————————————————┬—40,5
                                        └—ELSE-<instrução>[2]—┘  5
```

A instrução condicional permite que após um teste sejam executadas ou saltadas uma ou mais instruções.

Se a condição testada for verdadeira a instrução após o THEN é executada, sendo saltada a seguinte ao ELSE, se houver. Para condição falsa é executada a instrução após o ELSE, se houver.

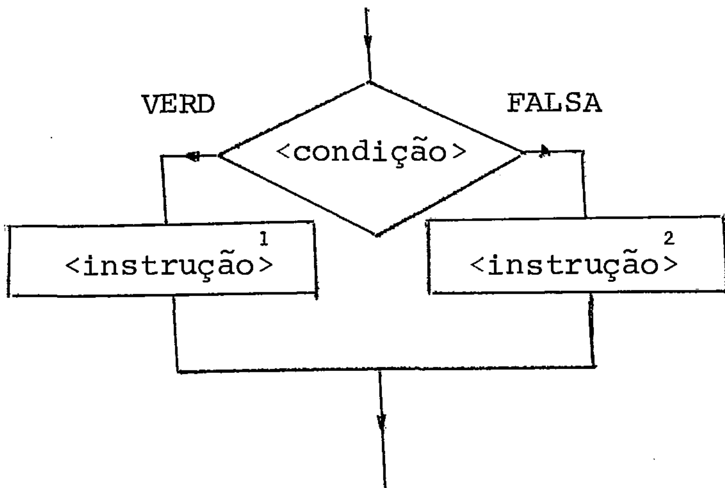
Se houver várias instruções IF aninhadas, é obedecida a regra que um ELSE se relaciona sempre com o THEN mais próximo. A alteração desta regra é feita usando-se instrução composta.

Para a compilação de IF <cond> THEN <instr>[1] temos:



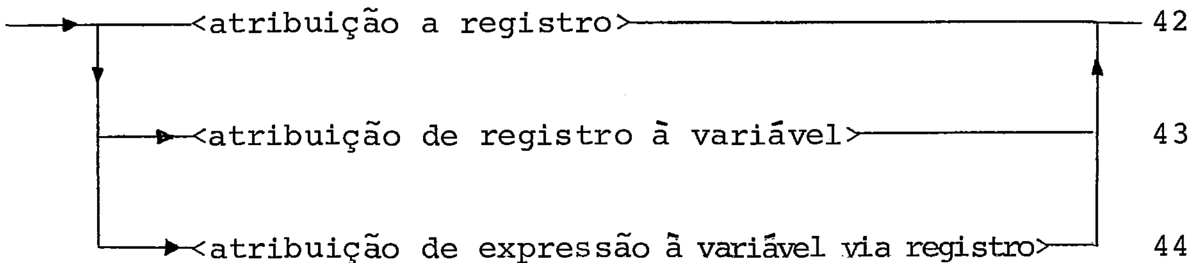

```
<instr p/teste condição>
 J ¬<condição>        XXX
                 1
 <instrução>
XXX
```

Para IF <cond> THEN <instr>[1] ELSE <instr>[2] teremos:




```
<instr p/teste condição>
 J ¬<condição>  XXX
              1
 <instrução>
 JMP      ZZZ
              2
XXX <instrução>
ZZZ
```

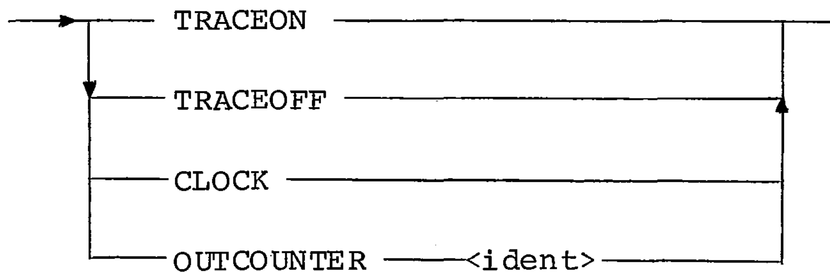
27 <instrução de atribuição>:: =



A instrução de atribuição permite, após o cálculo da expressão à direita do sinal de atribuição (:=) , alterar o conteúdo de um registro, uma ou mais posições de memória associadas a variáveis simples, indexadas ou identificador de nível de record substituindo-o pelo resultado do cálculo efetuado.

Na atribuição à variável indexada não é feito teste para verificação se o acesso esta fora dos limites da declaração.

28 <instruções para depuração>:: =



6

A instrução TRACEON avisa que a partir deste ponto antes de cada instrução serão impressos o código da instrução, e os conteúdos de rA, rX e rIi, $1 \leq i \leq 6$.

A instrução TRACEOFF desativa a instrução TRACEON.

A instrução CLOCK faz com que seja impresso o conteúdo do relógio do simulador.

OUTCOUNTER <ident> imprime o conteúdo des-

te contador.

Nenhuma destas instruções afeta o conteúdo do relógio.

Estas instruções não existem na definição feita por Knuth [3] e foram introduzidas no simulador do MIX por Markenzon [6], cujo trabalho deve ser procurado para maiores detalhes.

Exemplo: PLMIX   MIXAL

```
Exemplo: PLMIX                              MIXAL
:
COUNTER  LOOP1;                   PC
RI2: = 2;                         0       ENT2      2
RAC: = 10;                        1       ENTA     10
RI1: = 20;                        2       ENT1     20
TRACEON;                          3       TRCE
REPEAT  RI2 TIMES                 4       LOOP
    BEGIN                         5       INCA     0,1
    LOOP1:RAC:=RAC + RI1          6       ENT3     0,1
          RI3:=RI1 - 3            7       DEC3     3
    END                           8       DEC2     1
TRACEOFF;                         9       J2P      4
OUTCOUNTER LOOP1;                10       NTRC
                                 11       OUTL     3
```

Na execução teremos na impressora:

INST = 5   A = 10   I1 = 20   I2 = 2   I3 = ?
X = ?   I4 = ?   I5 = ?   I6 = ?

INST = 5   A = 10   I1 = 20   I2 = 2   I3 = ?
X = ?   I4 = ?   I5 = ?   I6 = ?

INST = 48   A = 30   I1 = 20   I2 = 2   I3 = ?
X = ?   I4 = ?   I5 = ?   I6 = ?

INST = 51   A = 30   I1 = 20   I2 = 2   I3 = 20
X = ?   I4 = ?   I5 = ?   I6 = ?

INST = 51   A = 30   I1 = 20   I2 = 2   I3 = 17
X = ?   I4 = ?   I5 = ?   I6 = ?

INST = 50   A = 30   I1 = 20   I2 = 1   I3 = 17
X = ?   I4 = ?   I5 = ?   I6 = ?

INST = 42   A = 30   I1 = 20   I2 = 1   I3 = 17
X = ?   I4 = ?   I5 = ?   I6 = ?

INST = 5   A = 30   I1 = 20   I2 = 1   I3 = 17
X = ?   I4 = ?   I5 = ?   I6 = ?

INST = 48	A = 50	I1 = 20	I2 = 1	I3 = 17
	X = ?	I4 = ?	I5 = ?	I6 = ?
INST = 51	A = 50	I1 = 20	I2 = 1	I3 = 20
	X = ?	I4 = ?	I5 = ?	I6 = ?
INST = 51	A = 50	I1 = 20	I2 = 1	I3 = 17
	X = ?	I4 = ?	I5 = ?	I6 = ?
INST = 50	A = 50	I1 = 20	I2 = 0	I3 = 17
	X = ?	I4 = ?	I5 = ?	I6 = ?

** LOOP : 2

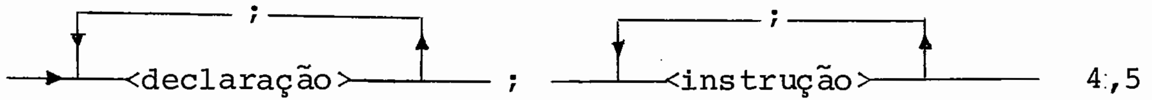
29 <letra>:: =

A
B
⋮
Z

30 <digito>:: =

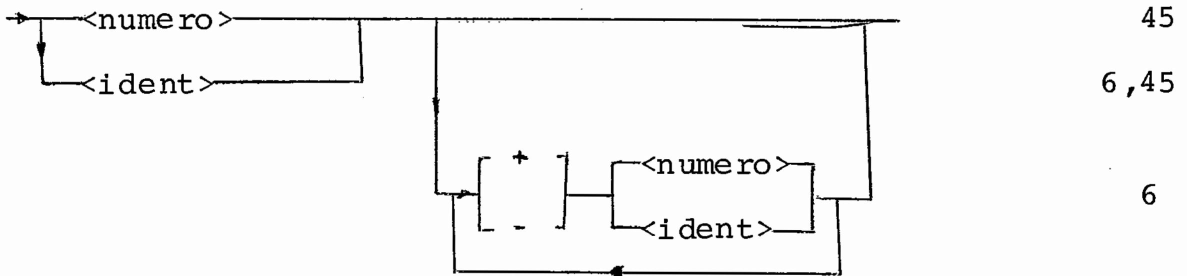
0
1
⋮
9

31 <lista sem procedimento>:: =



A <lista sem procedimento> aparece na definição do <corpo do procedimento> para indicar na sintaxe, a impossibilidade de declaração de subprograma no corpo de outro.

32 <expressão constante>:: =



Uma expressão constante é uma expressão que pode ser computada em tempo de compilação. Deste modo os componentes da expressão são números ou identificadores de constantes já definidas.

O compilador sempre trabalha com o resultado da expressão.

Exemplo:

PLMIX	MIXAL	
CONSTANT VINTE = 20		
RAC: = VINTE + 17 + RI2	ENTA	37
	INCA	0,2
RAC: = RI2 + VINTE + 17	ENTA	0,2
	INCA	20
	INCA	17

No segundo exemplo, como as expressões são calculadas da esquerda para a direita sem precedência, a presença de RI2 indica não estarmos tratando de expressões constantes, logo o compilador tratará cada constante em separado, o que justifica o código.

33 <cadeia>:: =

46

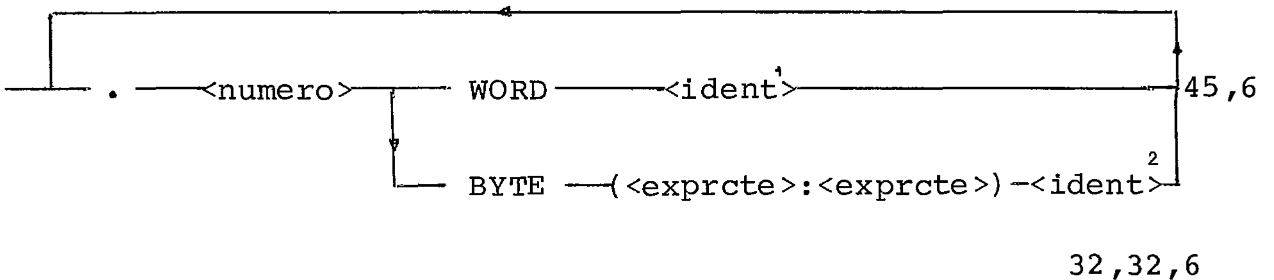
Uma cadeia é uma sequência de caracteres, inclusive branco, encerrada entre plics ('). É permitido o caracter plic dentro da cadeia mas deverão ser escritos dois plics juntos, sem espaço entre eles, e um só será considerado o plic da cadeia, o outro servirá apenas como informação para o analisador léxico.

Exemplo:

'ISTO E" UM EXEMPLO'

a cadeia será ISTO E"UM EXEMPLO.

34 <corpo do record>:: =



O nível superior é aquele da definição tipo WORD. Ela associa <ident>[1] a uma posição de memória.

As declarações de tipo BYTE que seguem a declaração WORD se referem à palavra associada a <ident>[1]. Estas declarações de BYTE dão nomes a bytes contíguos desta palavra. Poderá haver declarações que se superponham, isto é, dar um nome aos bytes (0:3), dar outro nome ao campo (2:3) e um outro a (2:2).

O valor de <numero> é irrelevante para o compilador.

A referência a um nível é feita por <identificador record>.<identificador nível>.

Exemplo:

```
RECORD MATRICULA :
.01 WORD CODIGO
.01 WORD NASCIM
    .02 BYTE (1:4) DATA
        .03 BYTE (1:2) ANO
        .03 BYTE (3:4) MES
    .02 BYTE (5:5) SEXO
```

35 <regix>:: =

```
──┬──<regi>──┬──                                        39
  └────RX────┘
```

A entidade <regix> engloba referências a um registro de índice ou ao registro RX, que é a extensão à direita do acumulador (rA).

36 <inicialização de vetor>:: =

```
  ┌─────────────── , ───────────────┐
──┴──<cadeia>───────────────────────┴──                 33
   ├─<exprcte>──────────────────────┤                   32
   └─ * ──<exprcte>¹[┬<cadeia>─┬]───┘                   32, 33
                     └<exprcte>┘                        32
```

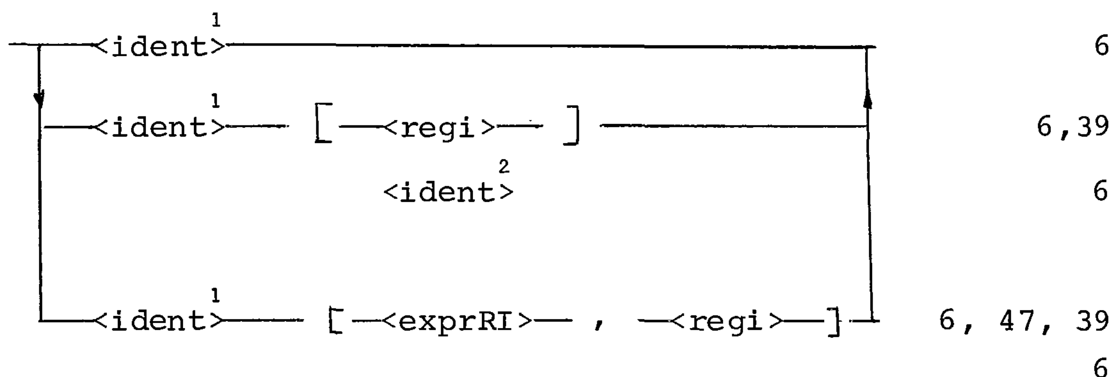
A inicialização de um vetor obedece às mesmas regras que a inicialização de variável simples tipo WORD (10).

Se várias palavras vão ser inicializadas com um mesmo valor, então usa-se o fator de repetição *<exprcte>[1] e o valor a ser repetido entre colchetes ([e]).

Exemplo:

ARRAY 10 WORD VAL = [20,30,-17,*5[31], 'AB', 'CD']

37 <variável>:: =



Uma <variável> é uma variável simples ou indexada.

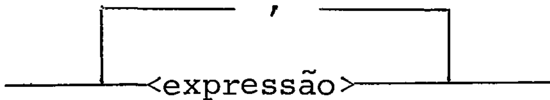
O índice deve ser um registro de índice , um identificador de equate (associado a um registro de índice), ou uma expressão RI. Uma expressão RI é calculada em um registro de índice e neste caso é necessário indicar em qual registro será efetuado o cálculo. Uma expressão constante também é uma expressão RI. Neste caso após o compilador avaliar

o valor da expressão, ela é atribuida ao registro de índice indicado.

Exemplo:

```
DADO
VALOR [RI1]
TABELA [RI2 + VINTE - 10, RI4]
```

38 <lista de parâmetros reais>:: =



48

Os parâmetros reais devem corresponder em tipo e número aos parâmetros formais.

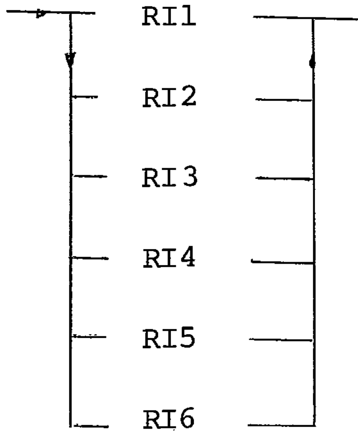
Para os parâmetros de tipo BYTE não é feita verificação se o campo do parâmetro real é igual ao do parâmetro formal.

Na execução do procedimento só é utilizado o campo especificado na declaração dos parâmetros formais.

Exemplo:

```
TESTE (MAX, 20 + RI3, ITEM [RI4])
```

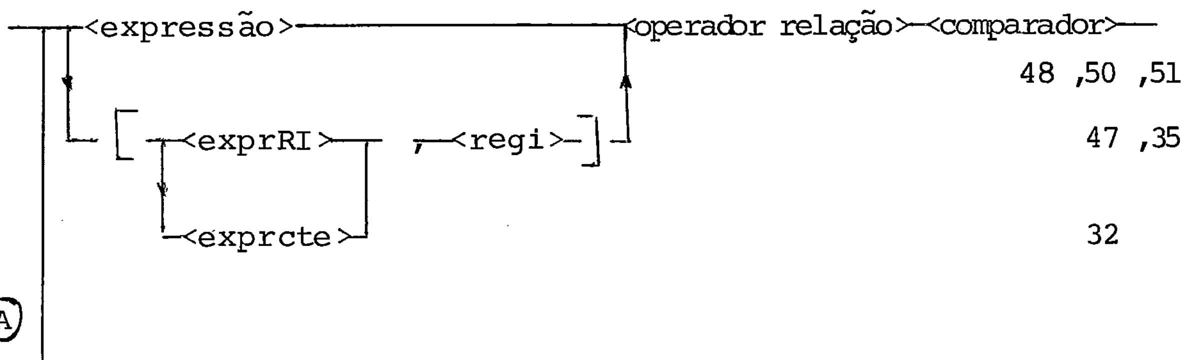
39 <regi>:: =

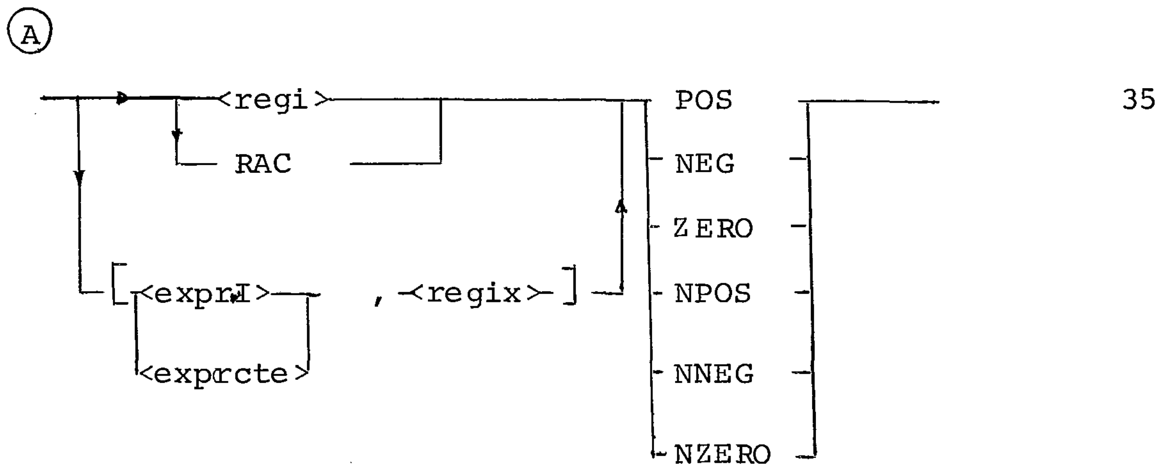


Os registros de índice tem dois bytes mais o sinal e são usados basicamente como contadores e índices nas variáveis indexadas.

A operação de comparação pode ser feita com um dos elementos do teste no registro de índice. Neste caso supõe-se que rIi tem cinco bytes, sendo que os bytes 1, 2 e 3 são iguais a zero.

40 <condição>:: =





Para o teste de condição <expressão> é calculada no registro A e depois executada a instrução de comparação - CMPA - entre o rA e uma posição de memória. Dependendo do resultado é ligado o indicador de MENOR, IGUAL ou MAIOR.

Se <expressão> for somente um registro de índice ou rX ou uma variável equate associada a um desses registros, a comparação é feita com este registro.

A comparação também pode ser feita com expressões RI calculadas em um registro, evitando assim utilizar rA. É necessário então indicar em qual registro será efetuado o cálculo da expressão RI e posterior comparação.

A instrução de desvio não altera o indicador de comparação.

Uma comparação de + 0 e - 0 resulta em IGUAL.

Para desvio usando o resultado no indicador de comparação a compilação gera a negação da relação testada. Assim teremos:

EQ	JNE
NE	JE
GT	JLE
GE	JL
LT	JGE
LE	JG

Os registros podem ser testados para condições: positivo, negativo, zero, não positivo, não negativo e não zero. Para tais condições teremos na compilação:

POS	J <reg> NP
NEG	J <reg> NN
ZERO	J <reg> NZ
NPOS	J <reg> P
NNEG	J <reg> N
NZERO	J <reg> Z

onde <reg> pode ser A, X ou Ii, $1 \leq i \leq 6$.

Exemplos:

PLMIX	MIXAL
IF INDICE GT MAX	LDA INDICE
THEN <instrução>	CMPA MAX
	JLE XXX
	<instrução>
	XXX

```
            PLMIX                                          MIXAL

WHILE  [FIM + 16 + RI6, RI4] LT  FINAL           LD4         FIM
DO <instrução>                                   INC4        16
                                                 INC4        0,6
                                                 CMP4        FINAL
                                                 J4GE        ZZZ
                                                 <instrução>
                                                 JMP         XXX
                                             ZZZ
REPEAT <instrução>                           XXX <instrução>
UNTIL   RAC   ZERO                               LDA         TESTE
                                                 JANZ        XXX
```

41 <instrução for>

```
           1              1                  1                  2
— FOR —< ident>— := —<expressão>— STEP —<exprcte>— UNTIL —<expressão>(A)

                                                        6, 48, 32, 48

             (A) — DO ——<instrução>——                   5

                         1                1                2
— FOR —<regix>—  := ┬<exprRI>┬ STEP —<exprcte>— UNTIL┬<exprRI>┬(B) 35,47,32,47
        2           │        │                       │        │
     <ident>        └<exprcte>┘                      └<exprcte>┘    6,32,32

             (B) — DO ——<instrução>——
```

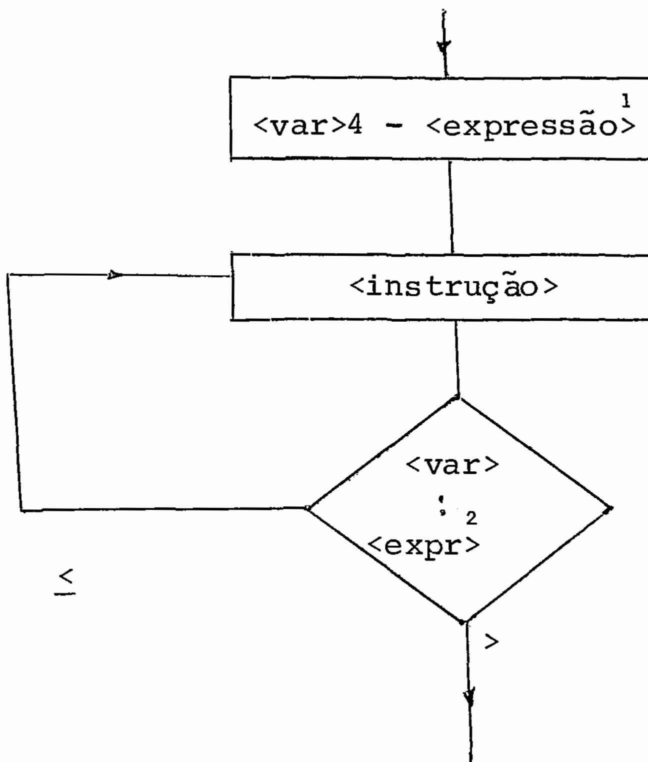
A <instrução for> faz com que <instrução> seja executada uma ou mais vezes, dependendo de uma variável ou registro de controle.

Esta variável ou registro recebe um valor inicial <expressão>^{1} ou <exprRI>^{1} (<exprcte>^{1}) antes do início da iteração. Após cada execução de <instrução>, a variável ou registro de controle é alterada de um valor constante , chamado passo. (<exprcte>^{1}). A iteração é repetida até que a variável (ou registro) seja inferior ao valor final <expressão>^{2}, ou superior ao valor final, para passo positivo.

A variável (ou registro) de controle pode ser usada em <instrução>, porém a alteração explicita (pelo programador) de seu valor, acarretará na modificação no número de iterações.

O esquema abaixo é geral e serão mostrados códigos gerados para variável de controle sendo uma variável simples e sendo um registro

supondo passo positivo



Código gerado para <ident>, expressão no rA e passo positivo

```
      <instr para cálculo <expressão>^1 >
      STA        XXX
      <instr para cálculo <expressão>^2 >
      STA        YYY
      JMP        ZZZ
YYY
ZZZ   <instrução>
      LDA        XXX
      INCA       <passo>
      CMPA       YYY
      JLE        ZZZ
```

XXX endereço da variável de controle

YYY posição de memória que contém o valor limite. É necessário guardar esta informação na memória, pois só é possível fazermos comparação entre registro e posição de memória.

Código gerado para <regi> (ou <ident> é uma variável equate), passo negativo.

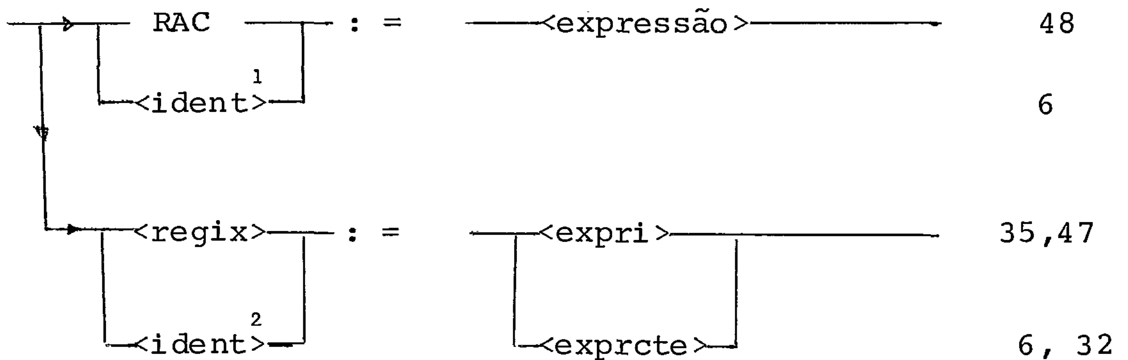
```
      <instr para cálculo <exprRI[1]> >          (1)
      <instr para cálculo <exprRI[2]> >          (2)
       STA          YYY
       JMP          ZZZ
YYY
ZZZ   <instrução>
       DECIi        <passo>
       CMPIi
       JGE          ZZZ
```

(1) a <exprrI[1]> será calculada no <regix> usado como variável de controle. Deste modo o valor inicial já esta nele e não é preciso instrução de carga.

(2) <exprI[2]> será calculada no rA para evitar destruir o conteúdo do registro de controle. Deste modo qualquer instrução "for" altera o conteúdo de rA.

O segundo exemplo de código gerado é igual para o rX.

42 <atribuição a registro>:: =



A operação de atribuição a registro, é uma indicação do registro sobre o qual será calculada a expressão à direita do sinal de atribuição, e onde ficará o resultado . Quando <expressão> for uma <expressão constante>, esta será calculada pelo compilador e depois carregada no registro através da instrução do tipo ENT <reg>.

<ident>^{1} e <ident>^{2} são variáveis do tipo equate, sendo <ident>^{1} associada ao rA e <ident>^{2} associada a rX ou rIi.

Só é feita indicação no "bit de overflow " se este ocorrer no registro rA. Para os registros rX e rIi não é fornecida nenhuma informação pela máquina.

Na atribuição de uma variável tipo BYTE o conteúdo do campo especificado é atribuido ao registro (rA , rX ou rIi) ajustado à direita e os outros bytes do registro são zerados. Se o campo de uma variável BYTE não inclui o byte de sinal, após a atribuição o registro fica com sinal positivo.

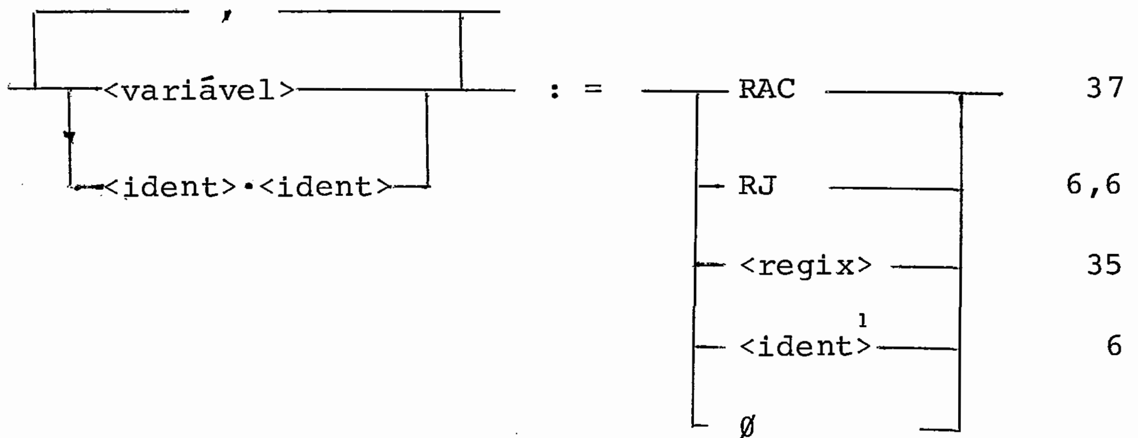
Exemplos:

PLMIX	MIXAL	
RAC: = VAR * COD [RI1] - 13 SLA 2	LDA	VAR
	MUL	COD,1
	DECA	13
	SLA	2

Suponha DELTA e MAX constantes e CONT variá vel

RI1: = CONT + DELTA - MAX + RI3	LD1	CONT
	INC1	DELTA
	DEC1	MAX
	INC1	0,3

43 <atribuição de registro a variável>: : =



A variável tipo WORD tem todos os seus bytes alterados quando recebe rA, rX, rIi ou uma variável declarada EQUATE (<ident>^{1}) com um desses registros. Na atribuição do registro rJ somente são alterados os bytes 1 e 2 e o sinal, que é sempre positivo. Atribuindo-se um registro de índice os

bytes 1,2 e 3 serão zerados e só recebem valores os bytes 4 e 5.

Se a variável tipo BYTE só ocupar alguns bytes da palavra, somente estes serão alterados, permanecendo os demais com o conteúdo anterior à operação. A atribuição é feita tomando-se os bytes do registro à partir da direita e efetuando um deslocamento para à esquerda, se necessário, para inserir no campo especificado.

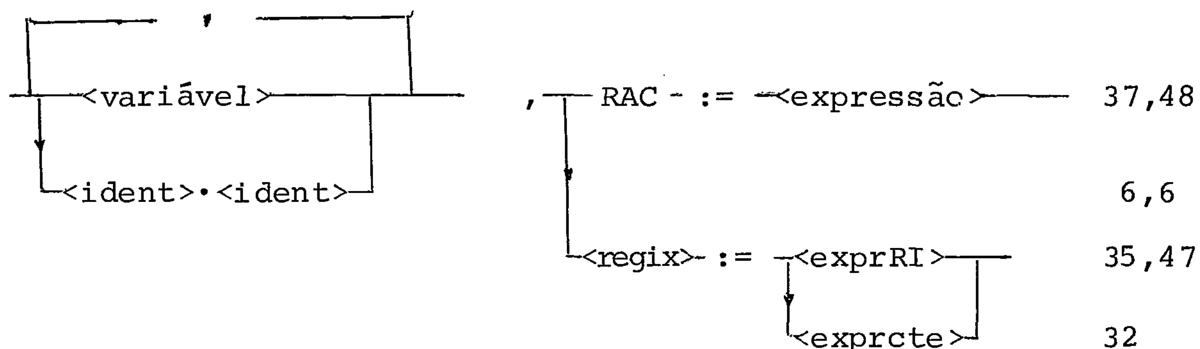
Além dos registros poderá ser feita a atribuição do valor zero, que corresponde à instrução STZ (store zero). Na inicialização de uma variável com este valor, é aconselhável esta atribuição pois ela gasta apenas uma instrução, enquanto que uma atribuição via registro demandará duas instruções - uma para carregar o registro e outra para transferir o conteúdo do registro para a memória.

O mesmo registro pode ser atribuido a mais de uma variável e neste caso se fará da esquerda para a direita.

Exemplos:

VALTR: = RAC	STA	VALOR
A, B, D [RI1]: = RI2	ST2	A
	ST2	B
	ST2	D,1
NO.LINK,INICIO[RI4]: = 0	STZ	NO(LINK)
	STZ	INICIO,4

44 <atribuição de expressão a variável via registro>:: =



Esta atribuição permite que uma ou mais variáveis recebam uma expressão. Como a atribuição à memória só pode ser feita através de registro, é necessário indicar qual será usado. A expressão deve poder ser calculada no registro referenciado.

A atribuição na lista será feita da esquerda para a direita, sendo válidas as observações feitas em (43) sobre atribuição de registro à variável.

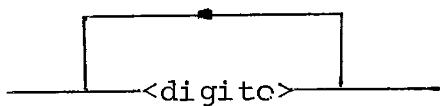
Exemplos:

PLMIX	MIXAL	
VAL,DADO[RI4], RAC:= MATRIZ[RI4]* DEZ + 5	LDA	MATRIZ,4
	MUL	DEZ
	INCA	5
	STA	VAL
	STA	DADO,RI4

TESTE[10 + MAX,RI4], RI1: = CONT + 5

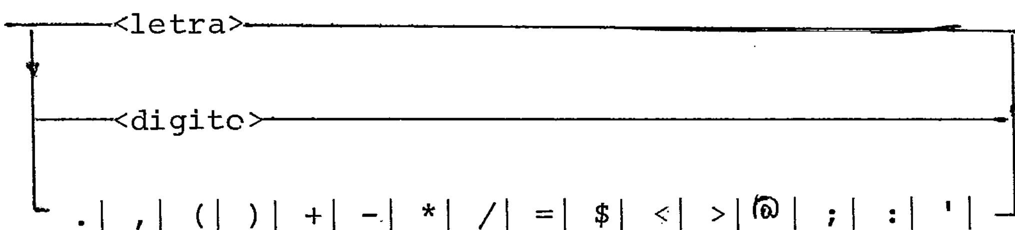
TESTE[10 + MAX,RI4], RI1: = CONT + 5	LD1	CONT
	INC1	5
	ENT4	10
	INC4	MAX
	ST1	TESTE,4

45 <número>:: =



30

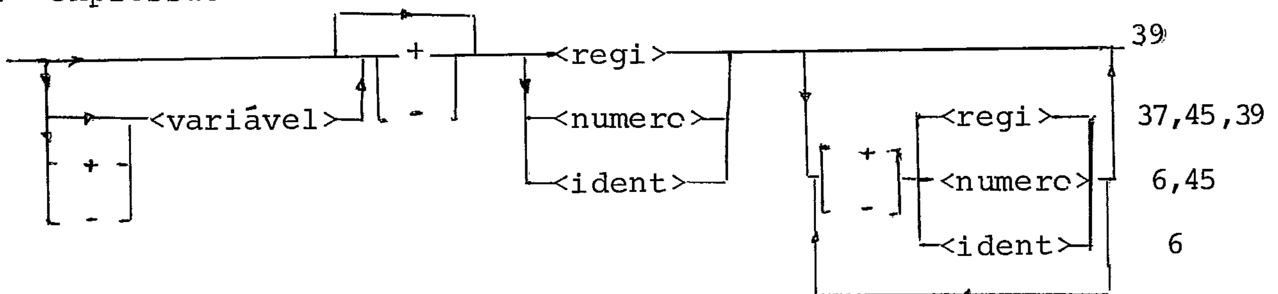
46 <caracter>:: =



29

30

47 <expressão RI>:: =



A expressão de registro I só admite os operadores de soma e subtração que correspondem às instruções INC e DEC. Os fatores só podem ser registros, números ou identificadores de constante ou equate.

Uma variável só pode aparecer como primeiro elemento da expressão.

A expressão é avaliada da esquerda para a direita.

O registro que está à esquerda do sinal de atribuição só poderá aparecer na expressão se for o primeiro elemento desta, caso contrário o compilador enviará uma mensagem de erro e não calculará a expressão.

Se o resultado da operação não puder ser colocado em 2 bytes o registro fica com valor indefinido e não ocorrerá ação sobre o indicador de "overflow".

Exemplos:

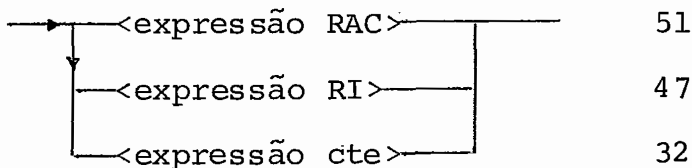
Supondo as declarações:

PLMIX	MIXAL

```
WORD VAR;
BYTE (3:4) ABC;
CONSTANT MAX = 100;
ARRAY 5 WORD A;
RECORD NO:
     .1 WORD INFO
     .1 WORD PONT
```

PLMIX	MIXAL	
.2 BYTE (1:2) ESQ		
.2 BYTE (3:4) DIR;		
ARRAY 3 RECORD TIPO: STRUCTURE NO;		
RI1 : = MAX	ENT1	100
RI6 : = NO.ESQ	LD6	NO (ESQ)
RI5 : = RI5 - 39	DC5	39
RI3 : = - RI2	ENN3	0,2
RI4 : = A[RI1] + RI2 - 15	LD4	A,1
	INC4	0,2
	DEC4	15
RI2: =TIPO.DIR[MAX-RI1,RI3]+ RI5	ENT3	MAX
	DEC3	0,1
	LD2	TIPO,3(DIR)
	INC2	0,5
VAR, RI6: = VAR + MAX	LD6	VAR
	INC6	MAX
	ST6	VAR

48 <expressão>



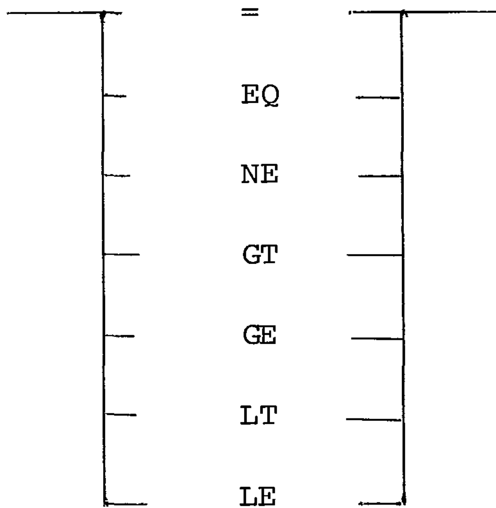
51

47

32

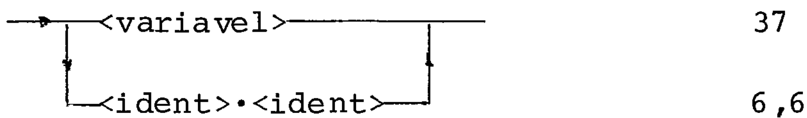
Uma expressão deverá ser calculada no acumulador.

49 <operador de relação>:: =



Na compilação das instruções com condição a relação testada será a negação da relação escrita no programa (vide (40)).

50 <comparador>:: =



37

6,6

O segundo elemento de uma comparação deverá ser sempre referenciável por um endereço de memória.

Esta restrição deriva dos testes serem sempre entre registro e variável, e da impossibilidade de serem usadas variáveis temporárias pelo compilador. Se fossem permitidas expressões no segundo membro da comparação, a utilização

de variáveis temporárias seria inevitável, conforme o exemplo:

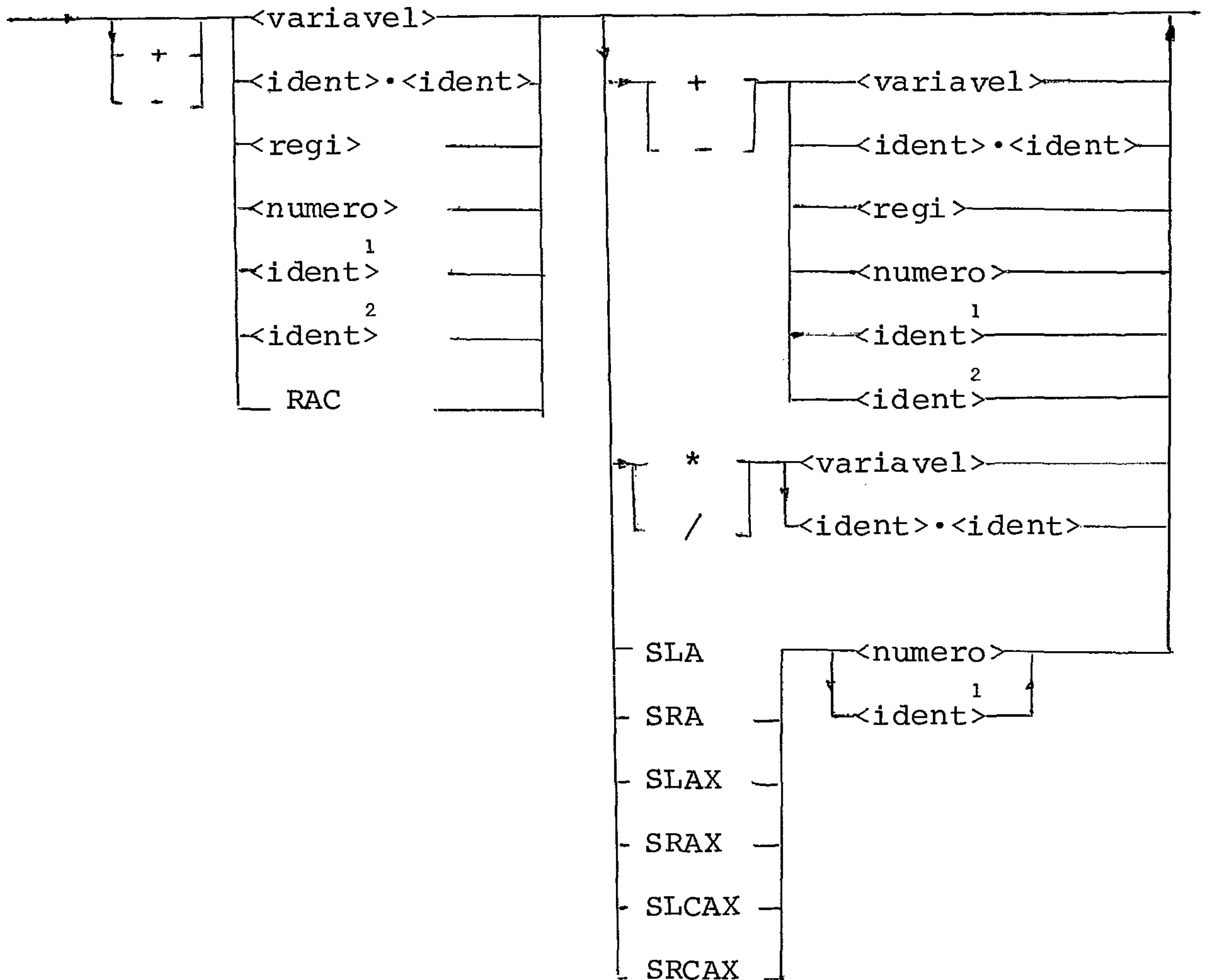
A * B > C * D

exigiria sua transformação em

A * B - C * D > 0

com uso obrigatório de temporária.

51 <expressão RAC>:: =



<ident>^{1} - identificador de constante

<ident>^{2} - identificador de variável equate

A expressão é calculada da esquerda para a direita sem prioridade entre os operadores. Cada uma das operações tem um tratamento sendo que o resultado fica em rA. A operação seguinte trabalha com o novo conteúdo de rA.

Atuação dos operadores:

adição: se a magnitude do resultado for maior que o permitido o indicador de "overflow" é ligado e permanece no acumulador o valor remanescente do resultado que couber em 5 bytes. A parte mais significativa ficará em um registro à esquerda de A.
Se o resultado for nulo o sinal do acumulador não é alterado.

subtração: mesmo procedimento da adição.

multiplicação: a multiplicação é fetuada sempre o rA e uma variável. O resultado obtido é armazenado em 10 bytes (rA e rX) e os bytes menos significativos ficam em rX. Os bytes de sinal recebem o sinal algébrico do produto.

divisão: o valor em rA e rX, tratado como um número de 10 bytes com o sinal de rA, é dividido

pelo valor da variável. Se a variável for igual a zero ou o quociente precisar de mais de 5 bytes para armazenar seu valor, rA e rX ficam com valores indefinidos e é ligado o indicador de "overflow". Caso contrário o quociente fica em rA e o resto em rX. O sinal de rA é o sinal algebrico da operação e rX recebe o sinal de rA, anterior à divisão.

deslocamento: em PLMIX os deslocamentos são operadores binários. Maiores detalhes sobre atuação dos operadores estão em II.2.3.9.

Exemplos

Suponha as declarações:

```
WORD VAR;
BYTE (3:4) ABC;
CONSTANT MAX = 100;
ARRAY 5 WORD A
RECORD NO:
      .1 WORD INFO
      .1 WORD PONT
            .2 BYTE (1:2) ESQ
            .2 BYTE (3:4) DIR;
ARRAY 3 RECORD TIPO: STRUCTURE NO;
EQUATE I = RI1;
```

```
RAC : = - ABC                              LDN     ABC (3:4)
RAC : = RAC * A[RI4]                       MUL     A,4
RAC : = RAC SLA 2 + NO.DIR                 SLA     2
                                           ADD     NO (DIR)
RAC : = TIPO.DIR[3,RI1]/A[MAX+5,RI2]       ENT1    3
                                           LDA     TIPO,1(DIR)
                                           ENT2    MAX
                                           INC2    5
                                           DIV     A,2
RAC : = VAR+MAX*ABC-A[I] + I               LDA     VAR
                                           INCA    MAX
                                           MUL     ABC(3:4)
                                           SUB     A,1
                                           INCA    0,1
RAC : = TIPO.PONT[A[I]-MAX-RI3,RI2]        LD2     A,1
                                           DEC2    MAX
                                           DEC2    0,3
                                           LDA     TIPO,2 (PONT)
RAC : = - 350 + MAX / VAR                  ENNA    250
                                           DIV     VAR
RAC : = -I + RI4 - A[RI5]                  ENNA    0,1
                                           INCA    0,4
                                           SUB     A,5
```

CAPÍTULO IV

ESTRUTURA DO COMPILADOR

IV.1. ASPECTOS GERAIS

O compilador PLMIX é um tradutor de um passo. Em uma passagem pelo programa a ser compilado reconhece os símbolos, realiza análise sintática, constroi tabela de símbolos, resolve referências, assinala erros e gera código-objeto.

A compilação em um passo foi possível devido às características da linguagem de ter todas as variáveis declaradas e dispensar otimização de código. Para cada comando da linguagem existe uma tradução definida, de modo que a otimização de código deverá ser feita pelo próprio programador. O uso previsto de linguagens de médio nível, tipo PLMIX, exige que a otimização de subexpressões e alterações da estrutura do programa visando otimização de código sejam tão radicais que não é possível efetuá-los automaticamente durante a compilação, a custos razoáveis (Knuth [25]).

A exigência de declaração de todas as variáveis e rótulos a serem usados no programa, facilitou a escrita do compilador permitindo saber, a cada momento, o códi

go a ser gerado.

IV.2. ANALISADOR LÉXICO

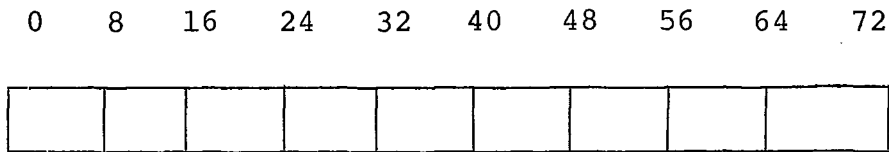
A função básica do analisador léxico é examinar o texto fonte e devolver o próximo símbolo terminal ("token") ao analisador sintático. Os comentários e os brancos são reconhecidos pelo analisador léxico e não são submetidos ao analisador sintático.

Entidades sintáticas como identificadores, constantes numéricas e cadeias são reconhecidas e devolvidas ao analisador sintático com uma codificação numérica, assim como as palavras reservadas e os símbolos especiais que também são definidos como símbolos terminais.

Para determinar se uma sequência de caracteres representa um identificador, uma palavra reservada ou um identificador de constante, o analisador léxico consulta a tabela de símbolos, que guarda informações sobre uma determinada sequência de caracteres.

No que concerne ao analisador léxico a tabela de símbolos é uma tabela tipo "hashing" com a estrutura definida em IV.4.1. e que utiliza a função abaixo para calcular o endereço de entrada na tabela e o incremento para caso de colisão (endereço aberto com hash duplo). Os 10 bytes reservados para o identificador são numerados a seguir, para fa

cilitar a descrição do cálculo.



NOME[0:16] = NOME[0:16] ⊕ NOME[16:16] ⊕ NOME[32:16] ⊕ NOME[48:16] +
⊕ NOME[64:16]

NOME[0:16] = NOME[2:6] * NOME[10: 6]

ENDEREÇO = NOME[4:8]

INCREMENTO = NOME[12:4]

No início da compilação a tabela de símbolos contém todas as palavras reservadas, guardadas em endereços calculados pelo método de Brent [23].

O analisador léxico reconhece os elementos gramaticais da linguagem definidos pelas seguintes regras

	regra	código	informação adicional
1	<letra> { <letra> | <digito> }	identificador ident.constante	entrada tabela símbolos valor constante
2	<digito> { <digito> }	numero	valor
3	' <carater> { <carater> } '	caracter em cadeia	carater

<table>
<tr><th></th><th><u>regra</u></th><th><u>código</u></th><th><u>informação adicional</u></th></tr>
<tr><td>4</td><td>$</td><td>$</td><td></td></tr>
<tr><td>5</td><td>;</td><td>;</td><td></td></tr>
<tr><td>6</td><td>:</td><td>:</td><td></td></tr>
<tr><td>7</td><td>|</td><td>|</td><td></td></tr>
<tr><td>8</td><td>|</td><td>|</td><td></td></tr>
<tr><td>9</td><td>(</td><td>(</td><td></td></tr>
<tr><td>10</td><td>,</td><td>,</td><td></td></tr>
<tr><td>11</td><td>)</td><td>)</td><td></td></tr>
<tr><td>12</td><td>/</td><td>/</td><td></td></tr>
<tr><td>13</td><td>: =</td><td>: =</td><td></td></tr>
<tr><td>14</td><td>=</td><td>=</td><td></td></tr>
<tr><td>15</td><td>+</td><td>+</td><td></td></tr>
<tr><td>16</td><td>-</td><td>-</td><td></td></tr>
<tr><td>17</td><td>.</td><td>.</td><td></td></tr>
<tr><td>18</td><td>*</td><td>*</td><td></td></tr>
<tr><td>19</td><td>'</td><td>'</td><td></td></tr>
<tr><td>20</td><td>BEGIN</td><td>palavra reservada</td><td></td></tr>
<tr><td>21</td><td>END</td><td></td><td></td></tr>
</table>

	regra	código	informação adicional
22	ARRAY	palavra reservada	
23	RECORD		
24	STRUCTURE		
25	CASE		
26	ON		
27	GOTO		
28	WAIT		
29	INPUT		
30	OUTPUT		
31	CHAR		
32	NUM		
33	OF		
34	MOVE		
35	TO		
36	FOR		
37	ENDRI1		
38	REPEAT		
39	UNTIL		

	regra	código	informação adicio nal
40	TIMES	palavra reservada	
41	WHILE		
42	DO		
43	STEP		
44	IF		
45	THEN		
46	TRACEON		
47	TRACEOFF		
48	CLOCK		
49	OUTCOUNTER		
50	RJ		
51	PROCEDURE		
52	BYTE		
53	WORD		
54	LABEL		
55	EQUATE		
56	CONSTANT		
57	COUNTER		

	regra	código	informação adicional
58	READY	palavra reservada	
59	BUSY		
60	IOC		
61	RI1		
62	RI2		
63	RI3		
64	RI4		
65	RI5		
66	RI6		
67	OVFL		
68	NOTOVF		
69	EQ		
70	NE		
71	GT		
72	GE		
73	LT		
74	LE		
75	ELSE		
76	RX		

	regra	código	informação adicional
77	SLA	palavra reservada	
78	SRA		
79	SLAX		
80	SRAX		
81	SLCAX		
82	SRCAX		
83	RAC		
84	VALUE		
85	POS		
86	NEG		
87	ZERO		
88	NPOS		
89	NNEG		
90	NZERO		

Para as regras número 1, 2, 3, 6, 13 e 14 são utilizados automatos finitos. Para as regras número 4 , 5, de 7 a 12 e de 15 a 19 é utilizada a tradução direta, visto serem símbolos com um só elemento e que não dão origem a dúvidas.