1. Publicação nº	2. Versão	3. Data	5. Distribuição
INPE- 3910 -RPE/509		*Junho, 86*	☐ Interna ☒ Externa ☐ Restrita

4. Origem	Programa
DIN	*DESOFT*

6. Palavras chaves - selecionadas pelo(s) autor(es)
COMPILADOR
LINGUAGEM C
GERADOR DE CÓDIGO

7. C.D.U.: *681.3.068*

8. Título *INPE-3910-RPE/509*

PROJETO BÁSICO DE UM GERADOR DE CÓDIGO PARA UM COMPILADOR C, TENDO COMO MÁQUINA ALVO O MICROPROCESSADOR 8086 DA INTEL CORPORATION

10. Páginas: *95*

11. Última página: *A.23*

12. Revisada por

Heloisa G.V.S. Borges

9. Autoria *Vânia Aparecida Dinardo Oleinki*
Sergio Roberto Matiello Pellegrino

13. Autorizada por

Marco Antonio Raupp
Diretor Geral

Assinatura responsável

14. Resumo/Notas

É apresentado o projeto básico do gerador de código do compilador para a linguagem C implementada no microprocessador 8086 da INTEL CORPORATION. A arquitetura básica do referido microprocessador é comentada de modo a facilitar o entendimento dos passos necessários para desenvolver este projeto. Foram desenvolvidos os algoritmos básicos para a tradução de expressões e definidas as estruturas de dados para o gerenciamento da memória e registradores.

15. Observações

ABSTRACT

The basic project of the compiler's code generation for the C language implemented in INTEL CORPORATION 8086 microprocessor is presented. The basic architecture of the microprocessor is commented in order to facilitate the understanding of the necessary steps taken to develop this project. Basic algorithms for the translation of expressions were developed and data structures defined to manage the memory and the registers.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

Este trabalho apresenta o projeto básico do gerador de código para o microprocessador 8086 da INTEL CORPORATION, do Compilador para a linguagem C em desenvolvimento no Instituto de Tecnologia/Centro Tecnológico para Informática/Secretaria Especial de Informática (IT/CTI/SEI) de Campinas. Este projeto foi elaborado em conjunto por pesquisadores do INPE e do IT/CTI/SEI.

O compilador em questão está sendo escrito na própria linguagem C, para ser posteriormente autocompilado. Basicamente o compilador é composto por duas partes: a primeira (passo 1) é responsável pela entrada do programa fonte na linguagem C, análise léxica, análise sintática, análise semântica e geração da linguagem intermediária; a segunda parte (passo 2), que corresponde ao gerador de código, faz a transformação da linguagem intermediária em código Assembler da máquina alvo.

O Capítulo 2 deste documento apresenta a arquitetura do microprocessador alvo, 8086 da INTEL, com a finalidade de auxiliar o leitor no entendimento das estratégias adotadas no projeto básico do gerador de código.

O Capítulo 3 apresenta a definição do sistema de alocação de dados em tempo de execução (RUN TIME), bem como as ações tomadas nas entradas e saídas de procedimentos.

No Capítulo 4 é apresentada a estrutura geral para a tradução dos comandos da linguagem C, bem como considerações sobre a linguagem intermediária.

O Capítulo 5 apresenta o mecanismo para a tradução de expressões, as estratégias para a identificação da melhor sequência de código, as estruturas de dados necessárias ao gerenciamento dos registradores e memória, e os algoritmos básicos para a geração de código para expressões.

Finalmente o Capítulo 6 informa o estágio atual do projeto e apresenta algumas considerações sobre a sua futura implementação.

CAPÍTULO 2

ARQUITETURA DO MICROPROCESSADOR 8086 DA INTEL CORPORATION

Este Capítulo descreve o conjunto de registradores, os modos de endereçamento e os grupos de instruções do 8086 da INTEL.

2.1 - CONJUNTO DE REGISTRADORES

O micropocessador INTEL 8086 (Rector and Alexy, 1980), contém 3 grupos de 4 registradores de 16 bits e um grupo de 9 "flags" de 1 bit. Nestes 3 grupos estão os registradores gerais, os registradores de índice, os registradores apontadores e os registradores de segmento. A Figura 2.1 descreve esta estrutura.

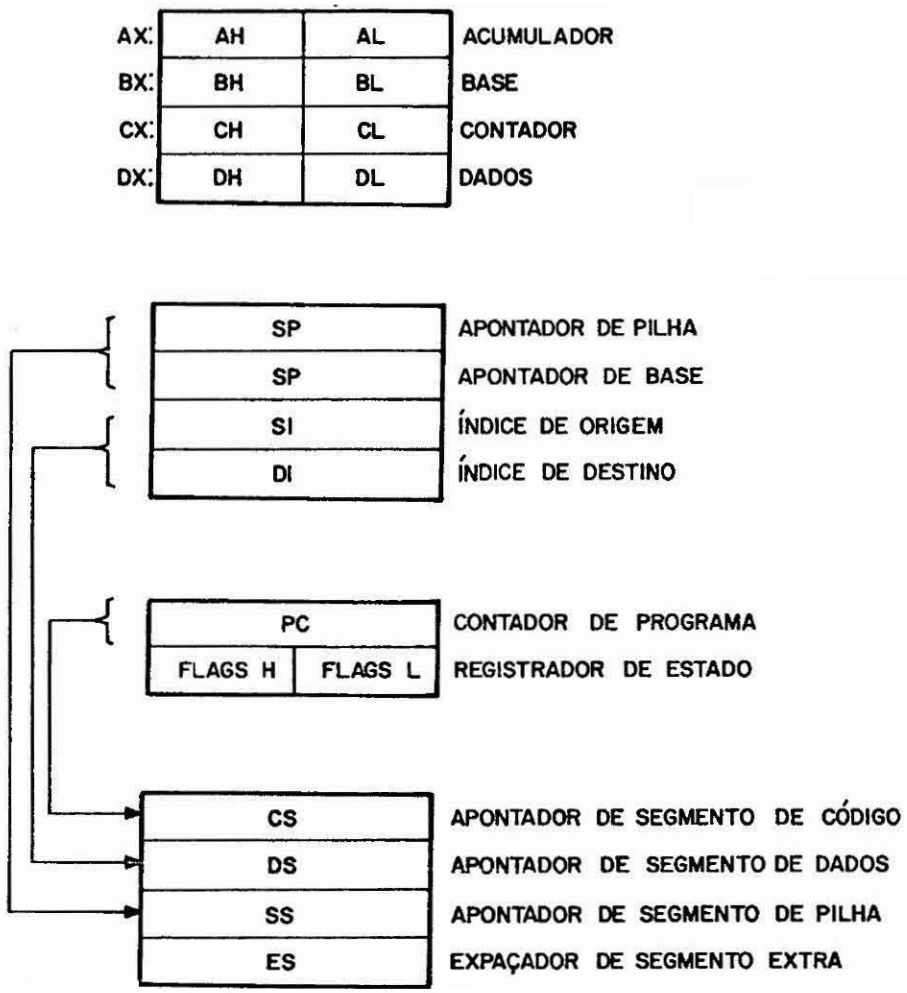


Fig. 2.1 - Estrutura do conjunto de registradores.

2.1.1 - REGISTRADORES GERAIS

Os registradores de propósitos gerais também podem ser referenciados separadamente como dois registradores de 8 bits, isto é, o registrador AX é de 16 bits e os registradores AL e AH são os registradores de 8 bits que compõem AX. A Figura 2.2 mostra esta estrutura.

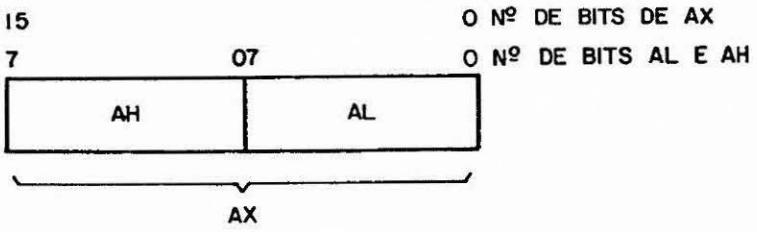


Fig. 2.2 - Divisão de Registrador de 16 bits.

As operações sobre objetos de 8 bits tornam-se mais eficientes usando esses registradores, pois ocupam menos espaço de memória e têm menor tempo de execução.

A seguir são relacionadas as particularidades de cada registrador desse grupo.

a) Registrador AX

- serve como acumulador primário;
- efetua operações de entrada e saída;
- as operações com dados imediatos têm melhor desempenho nesse registrador;
- as operações sobre cadeias e algumas instruções aritméticas são efetuadas exclusivamente em AX.

b) Registrador BX

- único registrador de propósito geral que pode ser usado no cálculo de endereço de memória.

c) Registrador CX

- utilizado na manipulação de cadeias e operações de LOOP.

d) Registrador DX

- usado no cálculo de endereço para operações de entrada e saída.

2.1.2 - REGISTRADORES APONTADORES

São utilizados no acesso de dados ao segmento de pilha e podem ser operandos de qualquer operação lógica ou aritmética. Os registradores apontadores são:

a) SP - apontador para o topo da pilha;

b) BP - apontador para a base da pilha.

2.1.3 - REGISTRADORES DE ÍNDICE

São utilizados no acesso à memória de dados e em operações de manipulação de cadeia; podem ser operandos de qualquer operação aritmética ou lógica. Os registradores de índice são SI e DI.

2.1.4 - REGISTRADORES DE SEGMENTO

Todo endereço de memória é calculado através da soma de um registrador de segmento, deslocado à esquerda de 4 bits, mais o endereço efetivo de memória. Desta forma, obtém-se um endereço de memória de 20 bits a partir de registradores de 16 bits, permitindo o endereçamento direto de 1 Mbytes de memória. Os registradores de segmento atuam como registradores de base e apontam sempre para uma posição de memória com endereço múltiplo de 16 bytes (fronteira de parágrafo). A partir desta posição de memória pode-se acessar qualquer endereço num intervalo de 64 Kbytes. Os registradores de segmento são:

a) *Registrador de segmento de código (CS)* que é usado juntamente com o contador de programa (PC) no cálculo do endereço de memória e na busca da próxima instrução.

b) *Registrador de segmento de dados (DS)* que é usado em todos os cálculos de endereço da memória de dados, com exceção aos endereços referentes à memória de pilha e aos endereços usados em operações com cadeias que usam os registradores DI e ES.

c) *Registrador de segmento de pilha (SS)* que é usado no cálculo de endereços de memória de pilha e em todas as instruções que manipulam estrutura de pilha, como PUSH, POP, CALL.

d) *Registrador de segmento extra (ES)* que é usado em operações sobre cadeias de caracteres no cálculo de endereço.

2.1.5 - REGISTRADOR DE ESTADO

Identifica o estado de operação do processador num dado instante; a Figura 2.3 ilustra sua estrutura.

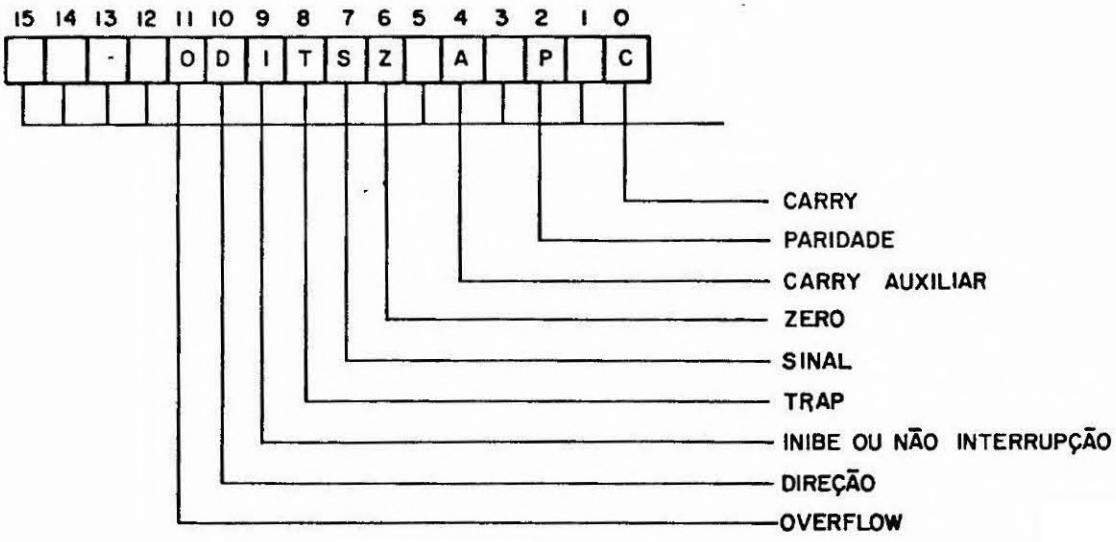


Fig. 2.3 - Registrador de estado.

As informações significativas deste registrador são:

a) *Bit de carry* que é uma saída "vai um" relativa ao bit de mais alta ordem, nas operações aritméticas.

b) *Bit de paridade* que se refere ao número de bits com valor 1 do byte menos significativo da última operação:

1 - número par de bits com valor 1;

0 - número ímpar de bits com valor 1.

c) *Bit de carry auxiliar* que é a saída "vai um" do bit de número 3 nas operações aritméticas em dados de 8 bits.

d) *Bit de zero* que se refere ao resultado da última operação de dados:

1 - resultado é zero;

0 - resultado é diferente de zero.

e) *Bit de sinal:*

0 - resultado positivo;

1 - resultado negativo.

f) *Bit de "trap"* que auxilia a depuração, coloca o processador no modo de execução passo a passo ("single-step").

g) *Bit de interrupção:*

0 - permite interrupções;

1 - inibe interrupções.

h) *Bit de direção* que determina, nas operações sobre cadeias de caracteres, se o conteúdo dos registradores SI e DI será incrementado ou decrementado:

0 - registrador será decrementado;

1 - registrador será incrementado.

i) *Bit de "overflow"* OU-exclusivo das saídas CARRY dos bits de mais alta ordem e do penúltimo bit, nas operações aritméticas.

2.2 - MODOS DE ENDEREÇAMENTO

Como já citado, todo cálculo de endereço neste microprocessador implica uma adição do endereço efetivo ao conteúdo de um registrador de segmento. Tendo em conta este fato, serão apresentadas as possibilidades existentes no cálculo do endereço efetivo.

a) Endereçamento direto

Os dois bytes seguintes ao código da operação no código objeto dão o deslocamento relativo ao segmento em questão.

b) Endereçamento direto indexado

Os registradores SI ou DI são usados como índice; pode haver a soma opcional de um deslocamento de 8 ou 16 bits.

c) Endereçamento implícito

É forma degenerada de endereçamento direto indexado, onde não é especificado nenhum deslocamento.

d) Endereçamento relativo à base

Usa o registrador BX como base no cálculo do endereço efetivo. Podem-se combinar todas as formas de endereçamento citadas até agora.

e) Endereçamento relativo a pilha

É uma variação do endereçamento anterior onde o segmento em questão é o de pilha e o registrador de base é BP.

A Tabela 2.1 ilustra os modos de endereçamento disponíveis, onde "desl" pode representar um deslocamento de 1 ou 2 bytes.

TABELA 2.1

MODOS DE ENDEREÇAMENTO - INTEL 8086

Relativo à base, indexado BX + SI	Relativo à base, direto, indexado BX + SI + desl.
Relativo à base, indexado BX + DI	Relativo à base, direto, indexado BX + DI + desl.
Relativo à base de pilha, indexado BP + SI	Relativo à base de pilha, direto, indexado BP + SI + desl.
Relativo à base de pilha, indexado BP + DI	Relativo à base de pilha, direto, indexado BP + DI + desl.
Implícito SI	Direto, indexado SI + desl.
Implícito DI	Direto, indexado DI + desl.
Direto "Endereço direto"	Direto, relativo à base de pilha BP + desl.
Relativo à base BX	Direto, relativo à base BX + desl.

2.3 - CONJUNTO DE INSTRUÇÕES

As instruções do Intel 8086 podem ser divididas em 5 grandes grupos.

- Transferência de dados;
- Aritmética inteira;
- Lógicas, deslocamento - rotação e manipulação de cadeias de caracteres;
- Transferência de controle;
- Controle do estado do processador.

a) Transferência de Dados

As instruções podem ser subdivididas por sua vez em 4 grupos: propósito geral, transferência específica para acumulador, transferência de endereços e transferência de "flags" (Rector and Alexy, 1980).

b) Aritmética Inteira

Estas instruções agem sobre um operando de 8 ou 16 bits em registrador; em alguns casos esse registrador é predeterminado pela instrução.

c) Lógica, Rotação e Manipulação de Caracteres

Os operandos são de 8 ou 16 bits e, no caso das operações lógicas, rotação e deslocamento, um dos operandos deve estar em um registrador.

d) Transferência de Controle

- CALL chamada de procedimento,
- RETURN retorno de procedimento,
- JMP desvio

e) Instruções de Controle de Processador

São as instruções que operam com o registrador de estado e controlam dispositivos externos.

CAPÍTULO 3

SISTEMA DE ALOCAÇÃO DE DADOS EM TEMPO DE EXECUÇÃO "RUN TIME"

Neste Capítulo apresentam-se a alocação de memória para variáveis e o tratamento efetuado nas entradas e saídas de funções.

3.1 - ALOCAÇÃO DE MEMÓRIA PARA VARIÁVEIS GLOBAIS

O microprocessador em questão possibilita o endereçamento de 64 kbytes a partir de um registrador de segmento. Por isto, a área de dados é limitada a este tamanho. De modo a permitir ao usuário acesso a uma área maior de memória, duas soluções foram analisadas. A primeira, mapear o registrador de segmento para cada objeto de dado alocado, mostra-se ineficiente quando aplicada a todos os dados do programa, pois este tipo de endereçamento é custoso em termos de tempo de processamento, causando com isto uma penalização ao usuário que não necessite de área de memória maior que 64 kbytes. A segunda solução, que foi a adotada neste projeto, é a de identificar os elementos que necessitam mais de 64 kbytes de memória (ditos longos) e endereçá-los através de mapeamento de um registrador de segmento diferente do usado para os demais dados globais (ditos estreitos). A identificação dos elementos ditos longos é feita através da declaração do usuário de uma nova classe de armazenamento, aqui especificada pela palavra chave "FAR".

3.1.1 - ÁREA DE DADOS GLOBAIS, DITOS "ESTREITOS" (NÃO FAR)

É uma área que ocupa menos de 64 kbytes, com variáveis endereçáveis via registrador de segmento DS. Este registrador mantém seu conteúdo inalterado durante toda a execução do programa, tendo então uma única área endereçável de 64 kbytes. Para acessar um elemento nesta área, o registrador de segmento DS já armazena o endereço base adequado.

3.1.2 - ÁREA DE DADOS GLOBAIS, DITOS "LONGOS" (FAR)

É uma área que pode ter mais de 64 kbytes de memória. Objetos nesta área são endereçados através do registrador de segmento ES.

Para acesso a um elemento de um dado objeto, o valor de ES é posicionado de forma a apontar a base do objeto; este conteúdo é ajustado durante o cálculo da expressão referente a cada índice.

A Figura 3.1 ilustra o posicionamento do registrador ES no acesso ao elemento A[i][j][k].

A Figura 3.2 mostra o acesso ao elemento em questão dado por um deslocamento d a partir da posição endereçada pelo registrador ES.

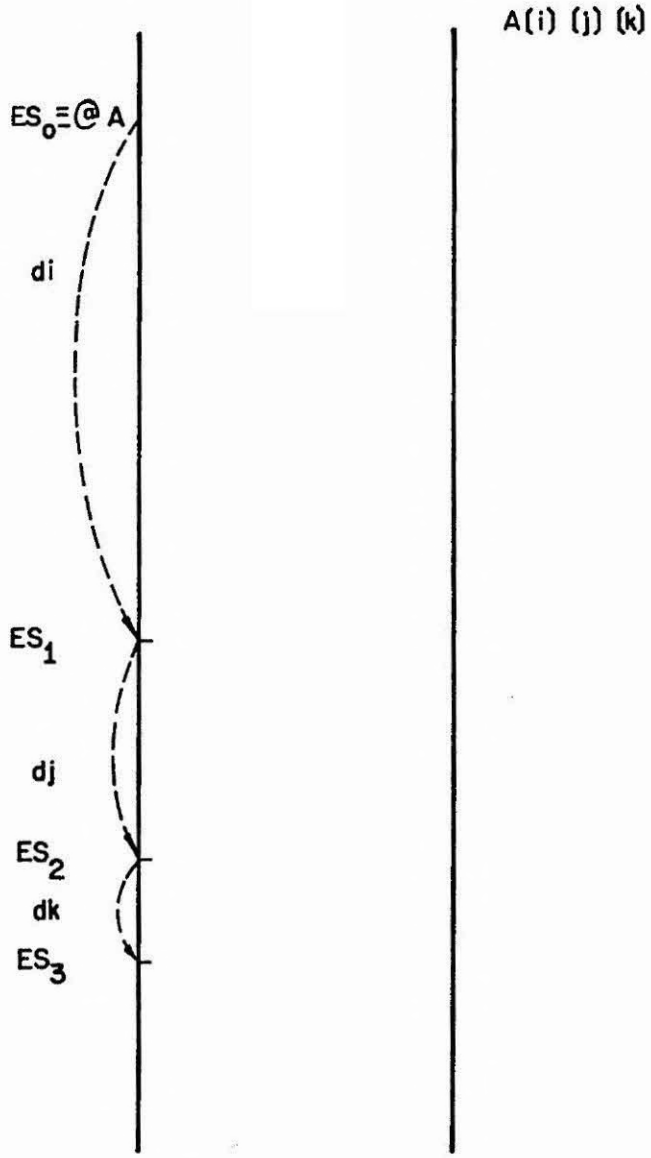


Fig. 3.1 - Esquema de posicionamento do registrador ES.

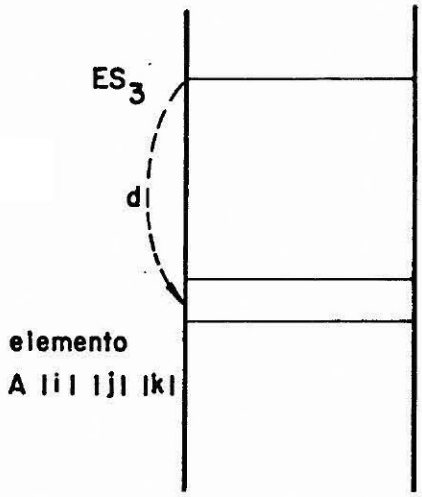


Fig. 3.2 - Acesso a um elemento qualquer.

3.2 - ALOCAÇÃO DE MEMÓRIA PARA DADOS AUTOMÁTICOS

Dados automáticos são aqueles alocados dinamicamente em tempo de execução, por ocasião da ativação de uma função (parâmetros + variáveis locais).

Os objetos com a classe de armazenamento tipo "register" são tratados como variáveis locais.

Pelo esquema de endereçamento disponível no INTEL 8086, têm-se duas alternativas para alocação dos tipos de dados descritos no subitem anterior:

1) O registrador de segmento mantém-se fixo durante toda a execução do programa, o que impõe uma limitação de espaço máximo de 64 kbytes para o espaço ocupado pelos dados decorrentes de todas as chamadas aninhadas de funções.

2) O registrador de segmento é atualizado, tendo o seu conteúdo alterado em cada entrada de uma função. Deste modo é possível utilizar até 64 kbytes para as variáveis locais e parâmetros (registro de ativação) de uma função. O chaveamento do registrador de segmento aumenta o "overhead" na chamada e retorno de funções.

Foi adotada a segunda alternativa, esquematizada na Figura 3.3.

Os dados automáticos alocados dinamicamente serão armazenados em uma estrutura de pilha, usando como base o registrador de segmentos SS. Todo endereçamento dentro de um registro de ativação da função será feito através desse registrador, com deslocamentos via o apontador de pilha SP e o ponteiro de Base BP.

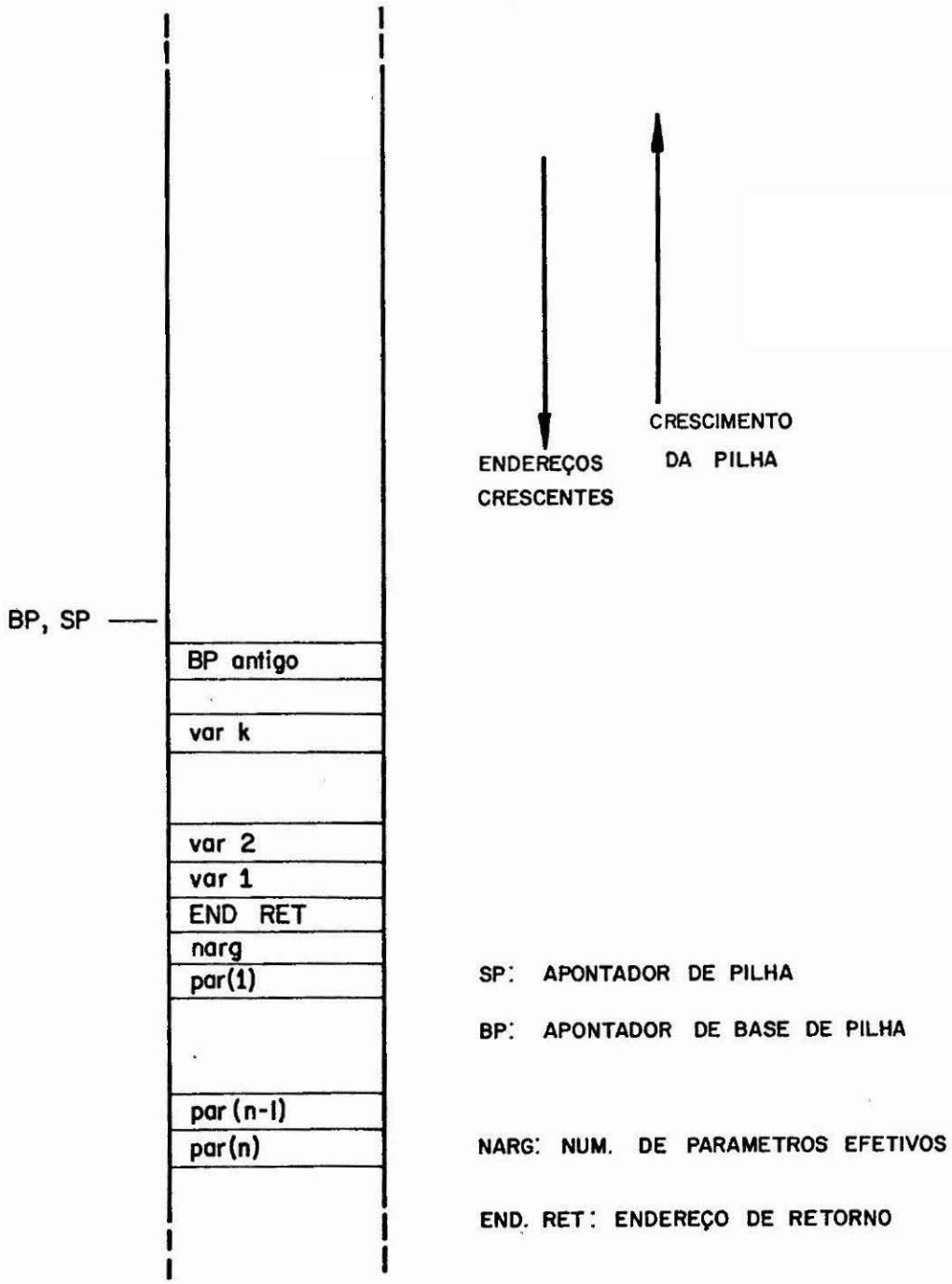


Fig. 3.3 - Registro de ativação de uma função.

3.2.1 - ENDEREÇAMENTO DE PARÂMETROS E VARIÁVEIS LOCAIS

O endereçamento de parâmetros e de variáveis locais será feito sempre através de um deslocamento a partir do apontador de base BP.

As variáveis locais estão deslocadas de um valor delta positivo em relação a BP, onde delta representa, na verdade, dois deslocamentos:

a) d1 - positivo, que é o número de bytes ocupados pelos parágrafos reservados às variáveis locais mais 2; esses dois bytes são relativos ao espaço ocupado pelo armazenamento do valor anterior de BP;

b) d2 - negativo, que é o deslocamento de cada variável na área de variáveis locais; equivale ao espaço ocupado pelas variáveis antecessoras.

O formato de endereçamento é:

BP + delta, onde delta = d1 + (d2)

Os parâmetros, da mesma forma, são endereçados a partir do mesmo deslocamento d1 e um certo d3 (positivo), dado pelo número de bytes ocupados pelos parâmetros anteriores acrescidos de 6 bytes (4 relativos ao armazenamento do endereço de retorno e 2 relativos ao armazenamento do número de parâmetros efetivos da função).

O formato de endereçamento é:

BP + delta, onde delta = d1 + d3

3.3 - AÇÕES TOMADAS NAS CHAMADAS DE FUNÇÃO

Na chamada de função, os parâmetros são calculados e empilhados na ordem inversa de sua declaração de maneira a fazer com que o primeiro parâmetro ocupe a posição mais próxima do topo da pilha; exemplo:

Func (param(1), param(2) ..., param(n))

ordem de cálculo e empilhamento dos parâmetros.

Após o empilhamento dos parâmetros, é empilhado o número de parâmetros efetivos e efetuada a chamada da função que causa o empilhamento do endereço de retorno. A sequência de código gerada para a chamada de uma função é mostrada a seguir:

```
PUSH param (n)     ;  empilha os parâmetros
PUSH param (n-1)   ;
      .
      .
      .
PUSH param (1)     ;
PUSH narg          ;  empilha o número de parâmetros
CALL ROT           ;  chama a função
ADD SP, # W        ;  restaura o topo da pilha
```

onde W é o espaço (em bytes) ocupado pelos parâmetros + 2 (número de parâmetros). A Figura 3.4 mostra o esquema de empilhamento.

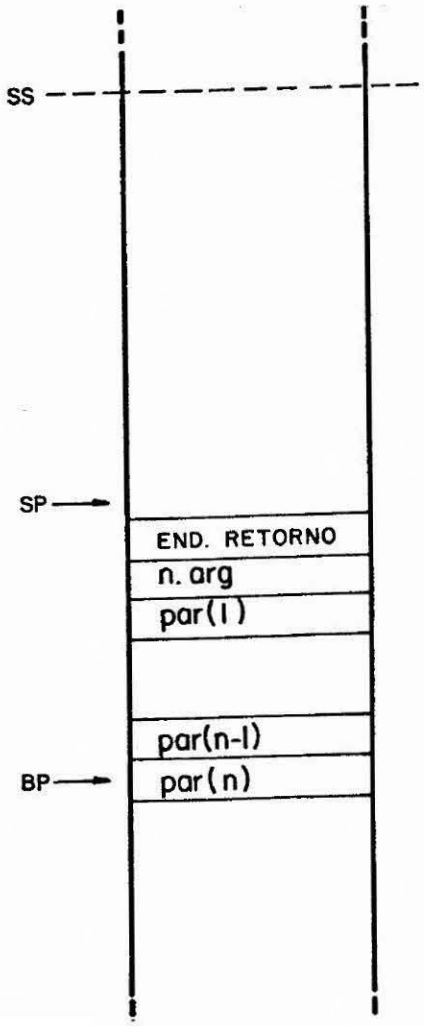


Fig. 3.4 - Situação da pilha após a chamada de função.

3.4 - AÇÕES TOMADAS NAS ENTRADAS DE FUNÇÃO

Nas entradas de função o registrador de segmentos, SS, é deslocado de modo a permitir a alocação dos dados locais. Este deslocamento é expresso em termos de número de parágrafos ocupados pela área de variáveis locais mais o número de parágrafos ocupados pelos parâmetros que constam na declaração da função.

O apontador de pilha (SP) representa um deslocamento em relação ao registrador de segmento (SS). Portanto, o deslocamento dado

a SS causa o mesmo deslocamento a SP. O apontador de pilha (SP) é então ajustado de modo a apontar para o fim dos parágrafos ocupados pelas variáveis locais.

A próxima ação é empilhar o valor antigo do apontador de base (BP), para a restauração na saída da função, e atribuir o valor de SP a BP. Todo procedimento é ilustrado na Figura 3.5. A seguir apresenta-se a sequência de código gerada na entrada de uma função:

```
MOV    AX, SS       ;
SUB    AX, #L       ;
MOV    SS, AX       ;   altera o valor de SS
ADD    SP, #M       ;   ajusta SP
PUSH   BP           ;   salva o valor de BP
MOV    BP, SP       ;   faz BP apontar para o topo da pilha
```

onde:

L = (área de locais + 15)/16) +(número de parâmetros declarados + 15))

M = ((número de parâmetros declarados + 15)/16) * 16

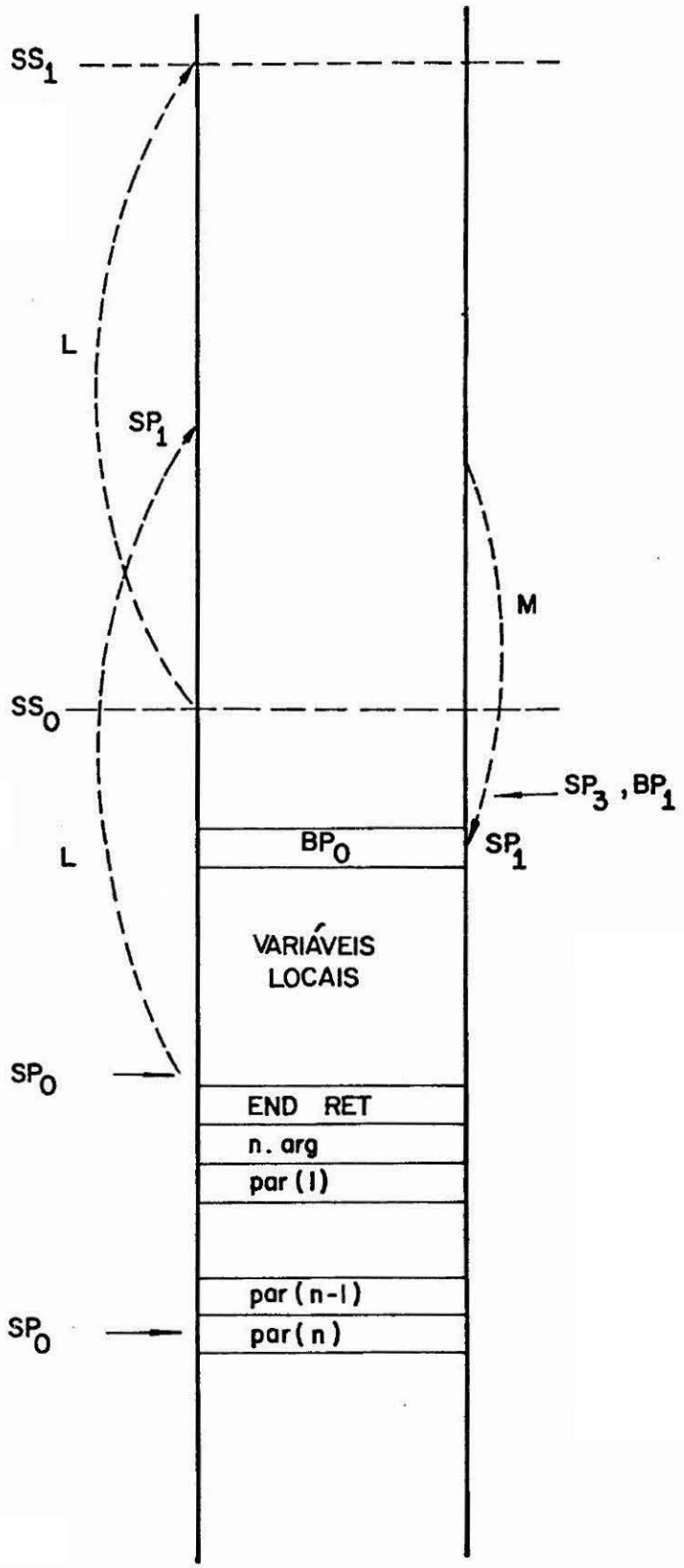


Fig. 3.5 - Situação da pilha depois da entrada em uma função.

3.5 - AÇÕES TOMADAS NAS SAÍDAS DE FUNÇÕES

A Figura 3.6 mostra a situação da pilha durante a saída da função, onde o ponteiro de base BP e o registrador de segmento SS devem ser restaurados.

O registrador SP é deslocado em virtude do deslocamento de SS, devendo ser restaurado de forma a apontar para a posição de memória onde se encontra o endereço de retorno. Esta situação está esquematizada na Figura 3.7.

A Figura 3.8 apresenta a situação final da pilha após a execução da instrução ADD SP,#W.

No caso da função retornar um valor, este é devolvido em um registrador. A sequência de código gerada na saída de uma função é apresentada a seguir:

```
POP BP      ;  restaura o valor anterior de BP
MOV AX, SS  ;
ADD AX,#L   ;
MOV SS, AX  ;  ajusta SS
SUB SP,#M   ;  ajusta SP
RET
```

onde:

L = (área de variáveis locais + 15)/16 +
(número parâmetros declarados + 15)/16

M = ((número de parâmetros declarados + 15)/16)* 16.

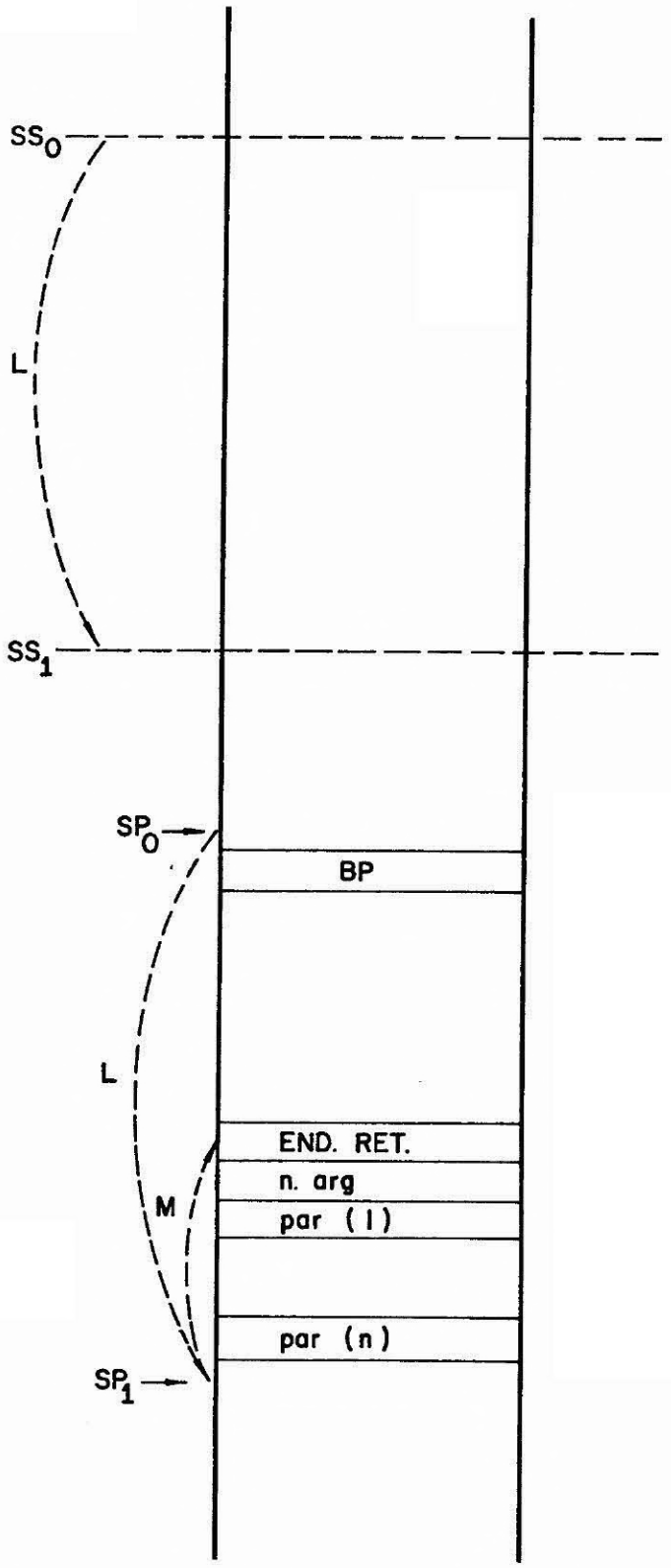


Fig. 3.6 - Pilha durante a saída de funções.

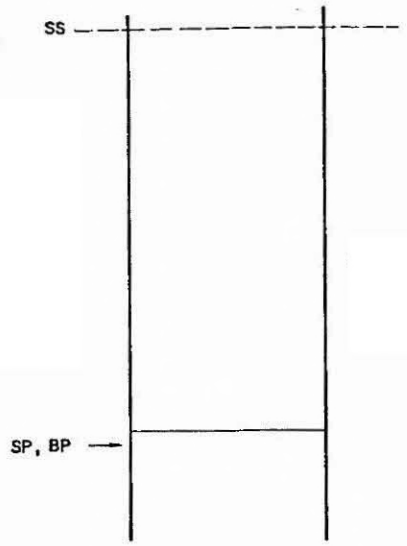


Fig. 3.7 - Pilha após a saída de funções.

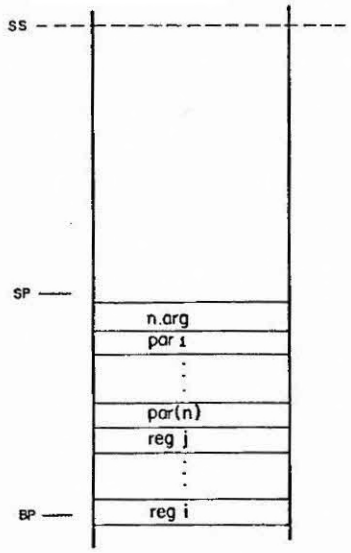


Fig. 3.8 - Situação da pilha após a execução da instrução: ADD SP, #W.

- W = nº de bytes ocupados pelos parâmetros e seu número.

3.6 - ATRIBUIÇÃO DE ENDEREÇOS PARA AS VARIÁVEIS

No início da geração de código serão atribuídos deslocamentos relativos à base do segmento de acordo com a classe de armazenamento dos objetos. Para isso são percorridos as árvores de declarações:

Dados Globais - terão um deslocamento em relação ao registrador de segmento "DS".

Dados Globais do Tipo FAR - o deslocamento (d) corresponde ao início de um parágrafo.

Variáveis Locais e Parâmetros - deslocamento em relação a "BP".

São alocados espaços ao objeto, de acordo com o tipo deste:

Char	:	DB .... (reserva 1 byte)
Short	:	DW .... (reserva 2 bytes)
Long, Float	:	DD .... (reserva 4 bytes)
Double	:	DD .... (reserva 8 bytes)
		DD ....

As árvores de declarações são percorridas também para inicializar as variáveis necessárias.

CAPÍTULO 4

GERAÇÃO DE CÓDIGO PARA COMANDOS

O gerador de código traduz cada procedimento para a linguagem ASSEMBLER analisando uma forma intermediária do tipo árvore. Essa tradução é feita em duas etapas:

- definição e alocação de endereços para variáveis,
- tradução de comandos e expressões.

4.1 - ESTRUTURA DA FORMA INTERMEDIÁRIA

A forma intermediária é gerada a medida que o programa fonte é analisado, correspondendo à árvore sintática do programa em C. Todas as informações semânticas necessárias ao gerador de código estão contidas nesta linguagem intermediária (CTI, 1985).

Esta linguagem intermediária é representada por uma árvore binária, onde o ponteiro esquerdo identifica o nó mais à esquerda (filho mais velho) e o ponteiro direito identifica o nó direito (irmão); caso não exista o filho direito, o ponteiro direito aponta para a raiz da subárvore na forma de costura para o pai.

A Figura 4.1 ilustra a árvore para a expressão: (a+b)*(c+d). A Tabela 4.1 apresenta o vetor de nós onde está armazenada esta árvore.

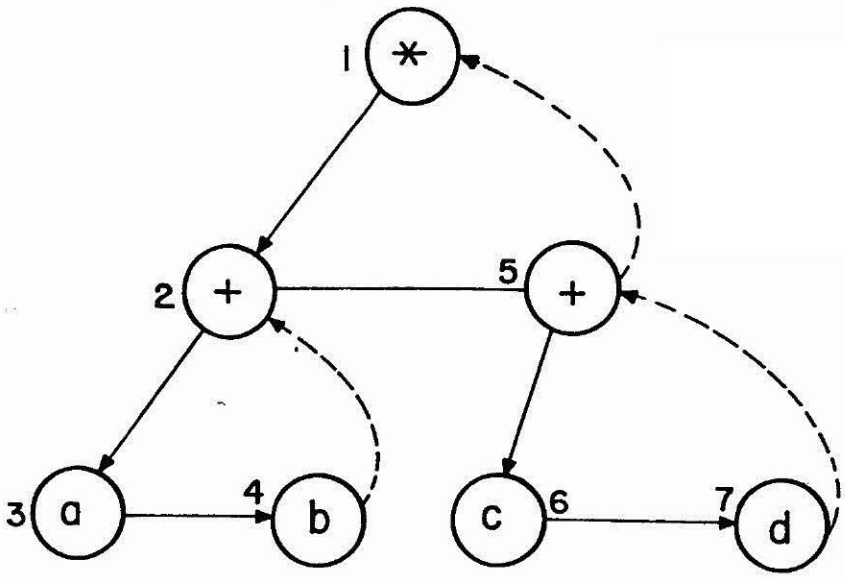


Fig. 4.1 - Exemplo da forma intermediária.

TABELA 4.1

REPRESENTAÇÃO INTERNA DA FORMA INTERMEDIÁRIA

	tipo	link-esq.	link-dir.
1	*	2	-
2	+	3	5
3	id	10	4
4	id	11	2
5	+	6	1
6	id	12	7
7	id	13	5

vetor de nós

	id		
.			
.			
.			
10	a	...	...
11	b		
12	c		
13	d		
.			
.			
.			

tabela de identificadores

Obs.: Se a folha da árvore for um identificador, o ponteiro esquerdo aponta para o identificador na Tabela de Identificadores.

4.2 - TRADUÇÃO DE COMANDOS

A geração de código para os comandos da linguagem é feita por procedimentos recursivos, sendo um procedimento por comando da linguagem C (CTI, 1985).

4.2.1 - PROCEDIMENTO GERAL PARA TRADUÇÃO DE COMANDOS

Apresenta-se a seguir tal procedimento:

```
procedimento gera-comando(no);
inicio
   caso no seja:
      comando-composto:gera-cmd-composto(no);
      do:              gera-do(no);
      while:           gera-while(no);
      if:              gera-if(no);
      goto:            gera-goto(no);
      for:             gera-for(no);
      case:            gera-case(no);
      comando-label:   gera-comando-label(no);
      break:           gera-break(no);
      comando-vazio:   gera-comando-vazio(no);
      switch:          gera-switch(no);
      continue:        gera-continue(no);
      default:         gera-default(no);
   fim do caso
fim /* gera-comando */
```

São apresentados procedimentos que traduzem alguns comandos típicos da linguagem C.

4.2.2 - TRADUÇÃO DOS COMANDOS "IF" E "WHILE"

a) Formato do comando "IF"

IF <expressão> THEN <comando-1>
/ELSE <comando-2>

a.1) Forma intermediária para o comando IF (Figura 4.2).

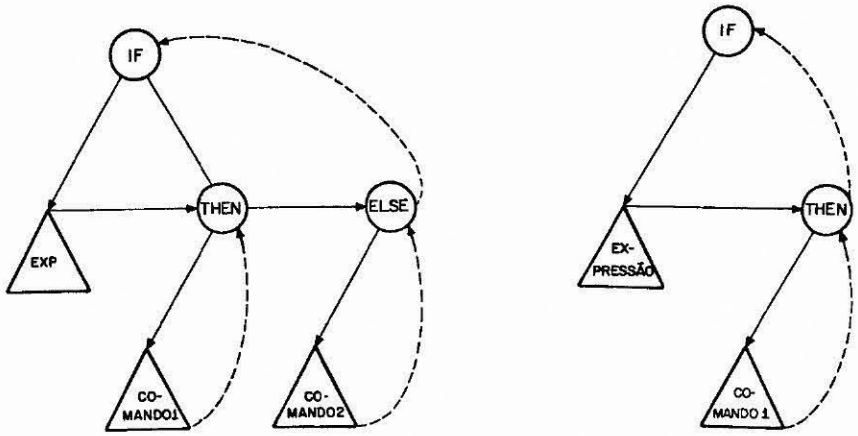


Fig. 4.2 - Árvore intermediária do comando IF.

a.2) Procedimento para a tradução do comando IF.

Este procedimento consiste em:

```
procedimento gera-if(no);
inicio
   noi := no;
   no2 := no-esq;
   rotula(no2,filho);
   gera-expressao(teste,no2,resultado);
   no3 := no2-dir;
   no4 := no3-esq;
   se flag-teste = falso entao gera-desvio(else);
   gera-comando(no4);
   gera-desvio(fimif);
else:  no5 := no4-dir;
       no6 := no5-esq;
       gera-comando(no6);
fimif:
fim /* gera-if */
```

b) Formato do comando "WHILE" :

WHILE <expressão> <comando>

b.1) Forma intermediária para o comando WHILE (Figura 4.3)

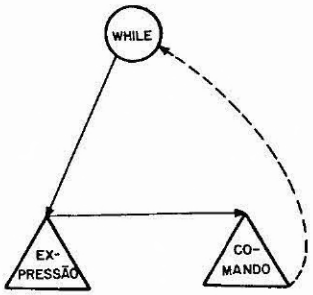


Fig. 4.3 - Árvore intermediária para o comando WHILE.

b.2) Procedimento para a tradução do comando "WHILE"

Este procedimento consiste em:

```
procedimento gera-while(no);
inicio
  no := no-esq;
  noi := no-dir;
  rotula(no,filho);
 l2: gera-expressao(teste,no,resultado);
    se flag-teste = falso entao gera-desvio(l1)
       senao inicio
                gera-comando(noi);
                gera-desvio(l2);
             fim;
 l1:
fim /* gera-while */
```

CAPÍTULO 5

GERAÇÃO DE CÓDIGO PARA EXPRESSÕES

A geração de código para expressões é feita a partir da linguagem intermediária, descrita no Capítulo 4, Seção 4.1. A tradução das expressões utiliza a árvore intermediária depois que é feita uma rotulação (Aho, 1979). O processo de rotulação consiste em percorrer a subárvore correspondente, em pós-ordem, associando a cada nó um número interno que denota quantos registradores são necessários para a tradução da expressão. Com esse mecanismo pode-se otimizar o uso de registradores utilizados na tradução (CTI, 1985).

5.1 - DETERMINAÇÃO DA SEQÜÊNCIA DE CÓDIGO

A seqüência de código gerado é função dos contextos propagados e da disponibilidade dos operandos (Akin, 1982).

5.1.1 - CONSIDERAÇÕES SOBRE OS CONTEXTOS PROPAGADOS

A melhor seqüência de código para uma expressão depende do contexto onde ocorre a expressão e do operador analisado. Por exemplo, considerando a variável "a" nos seguintes casos:

a) c:=a+b; neste caso deve ser obtido o valor da variável "a", para que somado ao conteúdo da variável "b" seja atribuído a variável "c". Logo o contexto de avaliação da variável "a" é determinado pelo operador "+".

b) IF a THEN...; neste caso, apenas um "flag" de condição deve ser ativado de modo a ser testado pelo comando "IF". Logo, o contexto de avaliação da variável "a" é determinado pelo comando "IF".

5.1.1.1 - CONTEXTOS ANALISADOS PARA UMA EXPRESSÃO

São eles:

1) Vazio - Identificado na raiz da árvore quando a expressão for usada como comando.

2) Teste - Quando a expressão estiver sendo usada para ativar um bit do registrador de estado. Este contexto é propagado pelos comandos condicionais if, while, do-until e operador condicional "?".

3) Parâmetro - Quando a expressão estiver sendo usada para obter um parâmetro para uma função. Este contexto é propagado pelo comando de chamada de função, onde o valor do objeto pode ser empilhado sem a necessidade de carregá-lo em registrador.

4) Valor - Quando o operador estiver sendo usado para obter um valor em registrador. É subdividido em 4 casos, dependendo dos registradores necessários para a execução do operador; são eles:

   a) valor1 - força que o valor do operando esteja nos registradores DX-AX.

   b) valor2 - força que o valor do operando esteja no registrador AX.

   c) valor3 - força que o valor do operando esteja no registrador do CL.

   d) valor4 - não impõe restrição ao registrador onde deve ser carregado o operando.

5) Endereço - Quando se deseja obter o endereço de um objeto.

6) Endereço1 - Quando se deseja obter o endereço de um objeto e inicializar alguns registradores.

5.1.1.2 - CONTEXTOS PROPAGADOS PELOS OPERADORES

São eles:

1) Os operadores "/" (divisão) e "%" (resto) propagam contexto VALOR1 para o operando esquerdo e contexto ENDEREÇO para o operando direito.

2) O operador "*" (multiplicação) propaga contexto VALOR2 para o operando esquerdo e contexto ENDEREÇO para o operando direito.

3) O operador "[]" (indexação) propaga contexto ENDEREÇO1 para o operando esquerdo e contexto VALOR2 para o operando direito.

4) Os operadores "<<" (deslocamento à esquerda) e ">>" (deslocamento à direita) propagam contexto ENDEREÇO para o operando esquerdo e contexto VALOR3 para o operando direito.

5) O operador "." (elemento de estrutura) propaga o contexto ENDEREÇO para os operandos esquerdo e direito.

6) Os demais operadores binários propagam o contexto VALOR4 para o operando esquerdo e ENDEREÇO para o operando direito.

7) Os operadores unários "*" e "&" propagam o contexto ENDEREÇO para o operando.

8) Os demais operadores unários propagam o contexto VALOR4 para o operando.

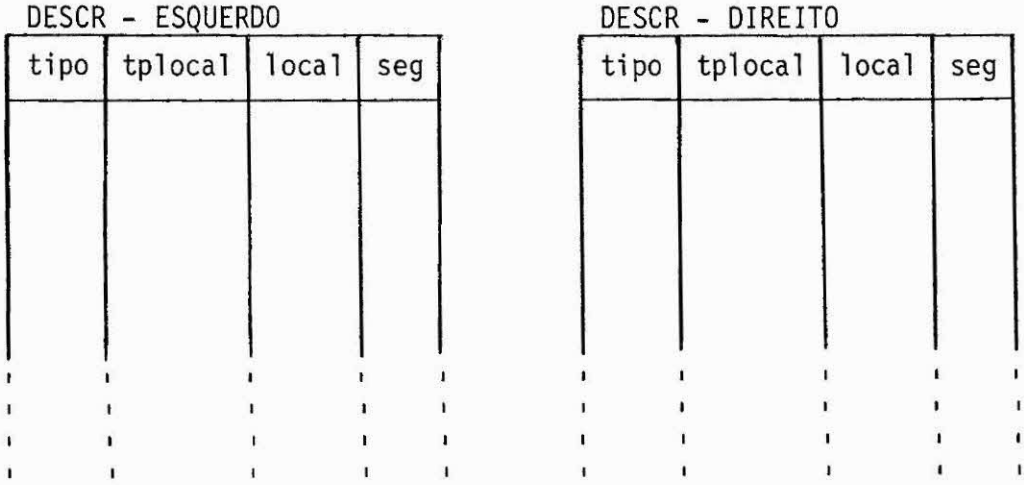
9) Os operadores de atribuição propagam o contexto ENDEREÇO para o operando esquerdo e o contexto VALOR4 para o operando direito.

5.1.2 - DISPONIBILIDADE DOS OPERANDOS

A disponibilidade dos operandos é conhecida nas "folhas" e propaga para "raiz" através de dois descritores. A Tabela 5.1 mostra o esquema dos descritores dos operandos esquerdo e direito, bem como o conteúdo de cada campo.

TABELA 5.1

REPRESENTAÇÃO DOS DESCRITORES, ESQUERDO E DIREITO



Descrição dos campos da Tabela 5.1:

a) tipo - denota o tipo do operando: CHAR, SHORT, LONG, FLOAT, DOUBLE.

b) tplocal - denota o tipo de armazenamento em que se encontra o operando e o número de bytes por ele ocupado. Pode ser:

1) R0, ocupa registrador de 8 bits;

2) R1, ocupa 1 registrador de 16 bits;

3) R2, ocupa 2 registradores de 16 bits;

4) R4, ocupa 4 registradores de 16 bits;

5) memória global, operando endereçado a partir de DS;

6) memória Obj-longos, operando endereçado a partir de ES;

7) memória local, operando endereçado a partir de BP;

8) parâmetro, operando endereçado a partir de BP;

9) memória temporária, operando endereçado a partir de SP+delta;

10) constante, valor literal da constante no próprio descritor.

c) local - indica a localização do objeto. Pode ser:

1) operando em memória - tem-se o deslocamento a partir da base considerada.

2) operando em registrador - tem-se um ponteiro para o descritor de registradores.

3) constante no descritor - tem-se o próprio valor literal da constante.

d) Seg - indica o registrador de segmento relacionado ao endereço do operando DS, ES ou SS.

5.2 - GERENCIAMENTO DOS REGISTRADORES

Os registradores são gerenciados através de um descritor de registradores, (Tabela 5.2), que mantêm informações sobre o estado

atual dos registradores. São considerados não só os registradores de 16 bits, mas também os de 8 bits que os compõem.

TABELA 5.2

ESTRUTURA DO DESCRITOR DE REGISTRADORES

	conteúdo	ocupação	ligação	referência
AL				
AH				
AX				
BL				
BH				
BX				
CL				
CH				
CX				
DL				
DH				
DX				
SI				
DI				

Descrição dos campos da Tabela 5.2

1) Conteúdo - é um apontador para o nó da linguagem intermediária.

2) Ocupação - a Tabela 5.3 mostra a relação de comprometimento entre registradores. O campo "ocupação" = 0 denota, para qualquer registrador, que ele pode ser usado; para "ocupação" = 1 tem-se duas interpretações:

a) Quando X está ocupado, ou seja, os 16 bits contêm dados significativos, implica que ocupação(L)=1 e ocupação(H)=1.

b) Quando apenas 8 bits são significativos, isto é, os 8 bits mais baixos X(L) ou os 8 bits mais altos X(H). Neste caso, não podem ser usados os 16 bits como um único significado, portanto o registrador X deve ser considerado ocupado. Se "ocupação" = 2, tem-se que a parte X(L) e X(H) estão sendo ocupadas por informações distintas, mas X como um todo não é significativo.

TABELA 5.3

ESQUEMA DE COMPROMETIMENTO ENTRE PARES DE REGISTRADORES

H/L X	L	H
0	livre	livre
1	livre ocupado não signif.	ocupado livre não signif.
2	ocupado	ocupado

3) Ligação - no caso de um objeto ocupar mais de um registrador de 16 bits, esse campo faz a ligação entre esses registradores. Quando esse valor for zero, denota que esse é o último (ou único) registrador da seqüência.

4) Referência - ponteiro para o descritor esquerdo ou direito que referencia esse registrador.

Obs.: Todos os registradores são liberados na "raiz" de uma árvore correspondente a um comando.

5.3 - ARMAZENAMENTO DE VALORES EM ÁREA DE MEMÓRIA TEMPORÁRIA

Quando o número de registradores torna-se insuficiente, faz-se necessário o salvamento de valores intermediários em área de memória temporária. A área reservada para esse salvamento será a área de memória do "RUN TIME" após a área reservada às variáveis locais. Durante a compilação, à medida que for surgindo a necessidade de temporários, esses serão alocados nesse trecho de memória, atualizando o número de bytes usados para temporários, como ilustra a Figura 5.1.

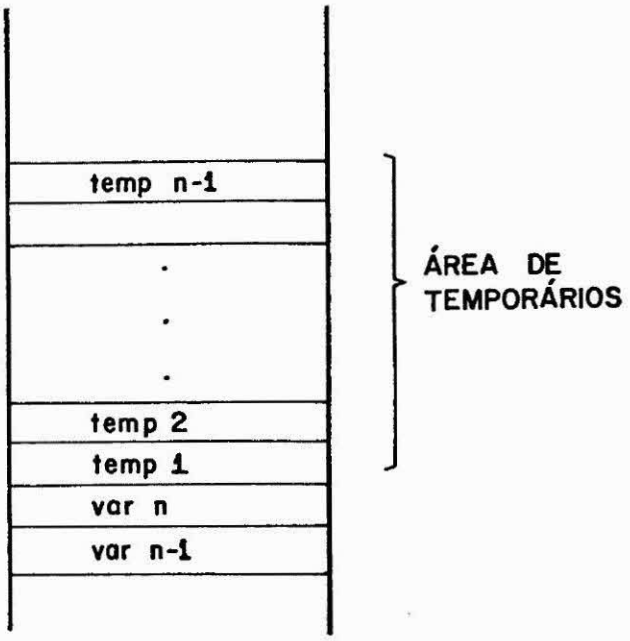


Fig. 5.1 - Alocação de memória para temporários.

5.4 - REAPROVEITAMENTO DO CONTEÚDO DE REGISTRADORES

Para alocar registradores para os nós "folhas" variáveis ou constantes) e feita uma consulta para verificar se a variável ou constante em questão já está carregada em algum registrador.

No caso de variáveis, a tabela de identificadores criada no Passo 1 do compilador, será acrescida de um campo, "campo-reg", para indicar se a variável está em registrador, através de um ponteiro para o descritor de registradores.

No caso de constantes, o descritor de registradores é percorrido e o campo "conteúdo" é consultado para verificar se aponta para um nó cujo conteúdo seja a constante a ser carregada.

5.5 - CLASSIFICAÇÃO DOS OPERADORES

A rotina que executa a geração de código para expressões considera os operadores classificados nos seguintes tipos:

1) Operadores binários aritméticos, relacionais e deslocamento - "binop":

   +, -, *, /, %, <>, >=, <=, ==, !=, >>, <<, >, <, ^, |

2) Operadores binários lógicos - "binop":

   && , ||

3) Operador binário de indexação:

   [ ]

4) Operador binário de seleção:

5) Operadores de atribuição - "asgnop":

=, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=

6) Operadores unários não-aritméticos:

*, &

7) Operadores unários aritméticos:

-, !, ~, ++, --

5.6 - PROCEDIMENTOS ENVOLVIDOS NA GERAÇÃO DE CÓDIGO PARA EXPRESSÕES

O gerador de código considera isoladamente os operadores e variáveis/constantes através de dois procedimentos.

5.6.1 - PROCEDIMENTO PARA TRATAMENTO DE OPERADORES

A seguir apresenta-se a forma geral do procedimento.

Gera-exp-op (contexto, endereço-no, resultado-no),

onde:

CONTEXTO - contexto transmitido ao nó pelo "pai";

ENDEREÇO-NO - endereço na linguagem intermediária do nó em questão;

RESULTADO-NO - posição do descritor de lado esquerdo ou direito, retornado ao "pai" como resultado.

Efetivamente os operadores são traduzidos pelos seguintes procedimentos:

a) Procedimento para a geração de código de um nó tipo binário aritmético, relacional, deslocamento ou lógico:

Traduz-binop (contexto-no, op, desc-esq, desc-dir, resultado, no);

onde:

op: +, -, *, /, %, ^, |, (aritméticos)
<>, =, ==, !=, (relacionais)
>>, <<, (deslocamento)
&&, ||, (lógicos).

b) Procedimento para a geração de código de um nó do tipo binário de indexação:

Traduz-indice (contexto-no, op, desc-esq, desc-dir, resultado, no);

onde:

op: [ ]

c) Procedimento para a geração de código de um nó do tipo binário de seleção:

Traduz-estr (contexto-no, op, desc-esq, desc-dir, resultado, no);

onde:

op: .

d) Procedimento para a geração de código de um nó do tipo atribuição:

Traduz-asgnop (contexto-no, op, desc-esq, desc-dir, resultado, no);

onde:

op: =, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=

e) Procedimento para a geração de código de um nó do tipo unário.

Traduz-unario (contexto-no, desc-esq, resultado, no);

onde:

op: *, &, -, !, ~, ++, --

f) Procedimento para a geração de código de um nó do tipo "folha", variável ou constante.

Gera-var-cte (contexto-no, no, resultado).

No intuito de apresentar ao leitor a forma geral dos algoritmos necessários à geração de código para os operadores, encontram-se no Apêndice A alguns procedimentos que mostram a complexidade no atendimento a todos os aspectos que favorecem à escolha da melhor seqüência de código.

O que contribui para o grande número de linhas de código nesses procedimentos é a variedade de tipos de operandos considerados pela linguagem C.

A seguir é apresentado o pssudocódigo do procedimento gera-exp-op.

```
procedimento gera_exp_op (contexto,endereco,resultado);
inicio
  se operador e do tipo BINOP
  entao inicio
         se operador e logico
         entao inicio
                gera_exp_op (valor4,no_esq,desc_esq);
                gera_exp_op (endereco,no_dir,desc_dir);
               fim;
         senao
         se rotulo (no_esq) >= rotulo (no_dir)  /* avalia-se primeiro o lado esquerdo */
         entao inicio
                caso operador seja : /* gera expressao no devido contexto */
                  *       : gera_exp_op (valor2,no_esq,desc_esq);
                  %,/     : gera_exp_op (valor1,no_esq,desc_esq);
                  <<,>>   : gera_exp_op (valor3,no_esq,desc_esq);
                  senao   : gera_exp_op (valor4,no_esq,desc_esq);
                fim de caso;
                gera_exp_op (endereco,noi_dir,desc_dir);
               fim
         senao inicio
                gera_exp_op (endereco,no_dir,desc_dir);
                caso operador seja : /* gera expressao no devido contexto */
                  *       : gera_exp_op (valor2,no_esq,desc_esq);
                  %,/     : gera_exp_op (valor1,no_esq,desc_esq);
                  <<,>>   : gera_exp_op (valor3,no_esq,desc_esq);
                  senao   : gera_exp_op (valor4,no_esq,desc_esq);
                fim de caso;
               fim;
                   /* e traduzido o operador do tipo BINOP */
               traduz_binop (contexto,OP,desc_esq,desc_dir,resultado,no);
       fim
  senao
    se operador e "[]"
    entao inicio
           gera_exp_op (endereco1,no_esq,desc_esq);
           muda_seg; /* objetos longos sao enderecados via ES */
           gera_exp_op (valor2,no_dir,desc_dir);
           traduz_indice (contexto,desc_esq,desc_dir,reesultado,no);
           se mudou_seg entao seg:= "DS";
          fim
    senao
      se operador e "."
      entao inicio
             gera_exp_op (endereco,no_esq,desc_esq);
             gera_exp_op (endereco1,no_dir,desc_dir);
             traduz_estr (contexto,OP,desc_esq,desc_dir,resultado,no);
            fim
      senao
        se operador e ASGNOP
        entao inicio
               gera_exp_op (endereco,no_esq,desc_esq);
               gera_exp_op (valor4,no_dir,desc_dir);
               traduz_asgnop (contexto,desc_esq,desc_dir,resultado,no);
              fim
```

```
        senao
          se operador e UNARIO
          entao inicio
                  se operador e ARITMETICO
                  entao gera_exp_op (valor4,no_esq,desc_esq)
                  senao gera_exp_op (endereco,no_esq,desc_esq);
                  traduz_unario (contexto,OP,desc_esq,resultado,no);
                fim
          senao /* no e folha */
            gera_var_cte (contexto,no,resultado);
fim; /* gera_exp_op */
```

5.6.2 - PROCEDIMENTO PARA TRATAMENTO DE VARIÁVEIS E CONSTANTES

Os nós "folhas", variáveis e constantes, são tratados pelo procedimento: gera-var-cte (contexto-no, endereço-no, resultado).

No caso de o nó analisado ser do tipo variável, é calculado o endereço, e o contexto é então analisado:

a) Se contexto = "VALOR", então carregue o conteúdo do endereço calculado em registrador.

b) Se contexto = "TESTE", então carregue o conteúdo do endereço calculado em registrador e gere uma operação "OR" desse registrador com ele mesmo para setar o "flag".

c) Se contexto = "PARÂMETRO", então empilhe o conteúdo do endereço calculado.

d) Se contexto = "ENDEREÇO", então devolva no descritor [resultado] o endereço calculado.

No caso de o nó analisado ser do tipo constante, as seguintes ações são tomadas:

a) Se contexto = "VALOR", antão carregue o literal da constante em registrador.

b) Se contexto = "PARÂMETRO", então empilhe o literal da constante.

c) Se contexto = "TESTE", então carregue o literal da constante em registrador e gere uma operação "OR" desse registrador com ele mesmo.

d) Se contexto = "ENDEREÇO", então devolva no descritor [resultado] o próprio literal da constante.

É apresentado a seguir o pseudocódigo do procedimento gera-var-cte.

```
procedimento gera_var_cte (contexto,no,resultado);
inicio
se contexto <> (parametro,endereco)
  entao inicio
         verifica_tipo (no,tipo_no);
         se contexto = (valor4,teste,vazio)
         entao aloca_reg (reg,cond,tipo_no,no,resultado);
         senao inicio
                caso tipo_no seja :
                  char  : caso contexto seja
                           valor1 : forca_reg (AX DX,2,no,cond,resultado);
                           valor2 : forca_reg (AL,1,no,cond,resultado);
                           valor3 : forca_reg (CL,1,no,cond,resultado);
                          fim de caso;
                  short : caso contexto seja
                           valor1 : forca_reg (AX DX,2,no,cond,resultado);
                           valor2 : forca_reg (AL,1,no,cond,resultado);
                           valor3 : inicio
                                     forca_reg (CL,1,no,cond,resultado);
                                     calcula_end (no,base,desl,seg);
                                     escreva ( MOV    CL,base+desl+1);
                                     cond := verdadeiro;
                                     descritor [resultado].seg := seg;
                                    fim;
                                 fim de caso;
                  long,float : caso contexto seja
                                valor1,valor2 : forca_reg (DX AX,2,no,cond,resultado);
                                valor3        : inicio
                                                   forca_reg (CL,1,no,cond,resultado);
                                                   calcula_end (no,base,desl,seg);
                                                   escreva ( MOV    CL,base+desl+4);
                                                   cond := verdadeiro;
                                                   descritor [resultado].seg:=seg;
                                                  fim;
                                 fim de caso;
                  double : inicio
                            se contexto = valor3
                            entao inicio
                                   forca_reg (CL,1,no,cond,resultado);
                                   calcula_end (no,base,desl,seg);
                                   escreva ( MOV    CL,base,desl+8);
                                   descritor [resultado].seg := seg;
                                  fim
                            senao aloca_reg (reg,cond,"double",no,resultado);
                           fim
                  fim de caso;
                  descritor [resultado].tipo := tipo_no;
                  se nao cond
                  entao inicio
                         se no e var
                         entao calcula_end (no,base,desl,seg);
                         carrega_valor (reg,nreg,base,desl,seg);
                        fim;
               fim;
        fim;
```

```
senao
  se no e var
  entao inicio
         verifica_tipo (no,tipo_no);
         calcula_end (no,base,desl,seg);
         descritor [resultado].seg := seg;
         descritor [resultado].tipo := tipo_no;
         se contexto = endereco!
         entao inicio
                se seg = "ES" ou "DS"
                entao inicio
                       forca_reg (BX,1,no,cond,var_auxiliar)
                       escreva ( MOV    BX,#0);
                      fim;
               fim;
         senao
           se contexto = parametro
           entao inicio
                  escreva ( PUSH    base+desl);
                  descritor [resultado].tplocal := "BP";
                  descritor [resultado].local := desl;
                  descritor [resultado].seg := "SS";
                 fim;
           senao inicio
                  descritor [resultado].tplocal := base;
                  descritor [resultado].local:= desl;
                 fim;
        fim
   senao inicio
          verifica_tipo (no,tipo_no);
          se contexto =  parametro
          entao inicio
                 aloca_reg (reg,cond,tipo_no,no,resultado);
                 carrega_valor (reg,nreg,no,base,desl,seg);
                 escreva ( PUSH    reg);
                 descritor [resultado].seg := "SS";
                 descritor [resultado].tplocal := "BP" ;
                 descritor [resultado].local := desl;
                fim
          senao inicio
                 descritor [resultado].local := valor_literal;
                 descritor [resultado].tplocal := "const";
                 descritor [resultado].seg := 0;
                fim;
          descritor [resultado].tipo := tipo_no;
         fim;
fim  /* gera_var_cte */
```

5.7 - PROCEDIMENTOS PARA GERENCIAMENTO DE REGISTRADORES E MEMÓRIA

1) Procedimento - aloca_Reg (reg, cond, tipo_no, no resultado);

Este procedimento aloca registrador(es) de acordo com o tipo de objeto a ser carregado. Se o objeto a ser carregado (nó) já está em registrador, o procedimento retorna em "reg" a identificação desse registrador e em "cond" o valor verdadeiro. Se o objeto não está em registrador e existe registrador livre, o procedimento retorna a identificação do registrador livre em "reg" e em "cond" o valor falso. Caso não exista registrador livre, o procedimento salva registrador na memória temporária; retorna em "reg" a identificação do registrador salvo e em "cond" o valor falso.

A seguir é apresentado o pseudocódigo do procedimento aloca-reg.

```
procedimento aloca_reg (reg,cond,tipo_no,no,resultado);
inicio
  caso tipo_no seja:
    char  : inicio
              seq := AL,AH,BL,BH,CL,CH,DL,DH;
              nreg := 1;
              x = -1;
            fim;
    short : inicio
              seq := AX,BX,CX,DX;
              nreg := 1;
              x := 1;
            fim ;
    long,float : inicio
                   seq := AX,BX,CX,DX;
                   nreg := 2;
                   x := 1;
                 fim;
    double : inicio
               seq := AX,BX,CX,DX;
               nreg := 4;
               x := 1;
             fim;
  fim de caso;
```

```
se busca (seq,nreg,no,reg1)
entao inicio  /* variavel ja esta em registrador */
       cond := verdadeiro;
       reg := reg1;
       se ((x = -1) e (reg mod 3) <> 0)
       entao descritor [resultado].tplocal := "registrador pequeno ";
       senao descritor [resultado].tplocal := "registrador nreg";
       descritor [resultado].local := reg;
      fim;
senao
  se existe_registrador_livre (seq,nreg,reg1)
  entao inicio
         reg := reg1;
         cond := falso;
         se ((x = -1) e (reg mod 3) <> 0)
         entao descritor [resultado].tplocal := " registrador pequeno ";
         senao descritor [resultado].tplocal := " registrador nreg ";
         descritor [resultado].local := reg;
        fim ;
  senao inicio
         seq := AX,BX,CX,DX;
         salva_registrador (nreg,seq);
         cond := falso;
         se reg = -1
         entao inicio
                reg := seq [1]-2;
                descritor [resultado].tplocal := " registrador pequeno ";
               fim;
         senao inicio
                reg := seq [1];
                descritor [resultado].tplocal := " registrador nreg ";
               fim;
         fim;
  ocupa registradores;
  descritor [resultado].tipo := tipo_no;
fim; /* aloca_reg */
```

2) Procedimento - busca (seq, nreg, no, regl): boolean

Esta função verifica se a variável ou constante já tem seu conteúdo em registrador: em caso afirmativo, ela retorna o valor verdadeiro e informa qual registrador o elemento ocupa (o primeiro no caso de o objeto ocupar mais de um registrador), caso contrário, retorna o valor falso.

A seguir é apresentado o pseudocódigo do procedimento busca.

```
procedimento busca (seq,nreg,no,reg1);
inicio
  se LI [no].tipo = " lit ";
  entao inicio
         j:=1;
         i := seq [j];
         enquanto (j <= nreg) e (LI[descritor_reg[i].conteudo].tipo <> LI[no].tipo
         faca inicio
                j := j+1;
                i := seq [j];
              fim;
         se j <= nreg
         entao inicio
                busca := verdadeiro;
                reg := seq [j];
               fim
         senao busca := falso;
        fim
  senao
    se tabela_de_identificadores [LI[no].tipo].campo_reg <> 0
    entao inicio
           busca := verdadeiro;
           reg := tabela_de_identificadores [LI [no].tipo].campo_reg;
          fim
    senao busca := falso
fim; /* busca */
```

3) Procedimento - existe-registrador-livre (seq, nreg, reg1): boolean

É uma função que verifica se existem "nreg" registradores da sequência "seq" que estejam livres ou cujo conteúdo seja variável ou constante. Se existir, ele retorna com valor verdadeiro e devolve em "reg" a identificação do primeiro registrador.

É apresentado a seguir o pseudocódigo do procedimento existe-registrador-livre.

```
procedimento existe_registrador_livre (seq,nreg,reg1) :booleana;
inicio
  j := 1;
  i:= seq [j];
  cont := 0;
  enquanto j <= maxreg e descritor_reg [i].ocup = 0
  faca inicio
        cont := cont+1;
        vet [cont] := i;
        j := j +1;
        i := seq [j];
      fim;
  se cont <> nreg
  entao inicio
         j := 1;
         i := seq [j];
         enquanto (j <= maxreg  e ((LI[descritor_reg [i].conteudo].tipo = "id")
                     ou (LI[descritor_reg [i].conteudo].tipo = "lit")) e
                     (cont < nreg))
         faca inicio
                cont := cont +1;
                muda descritor de lado esquerdo/direito
                vet [cont] := i;
                j:=j+1;
                i := seq [j];
               fim;
       fim;
  se cont = nreg
  entao inicio
         reg := vet [1];
         i := 1;
         enquanto i <= (cont -1)
         faca inicio
                descritor_reg [vet [i].ligacao]:= vet [i+1];
                i := i+1;
               fim;
         descritor_reg [vet [cont]].ligacao := 0;
         existe_registrador_livre := verdadeiro;
       fim
  senao existe_registrador_livre := falso;
  fim; /* existe_registrador_livre */
```

4) Procedimento - salva-registrador (nreg, seq);

Salva em memória temporária os "nreg" registradores pedidos. A seguir é apresentado o pseudocódigo deste procedimento.

```
procedimento salva_registrador (nreg,seq);
  inicio
    j:=1;
    i:=seq [j];
    enquanto j <= nreg
    faca inicio
          escreva ( MOV    [SP+d],descritor_reg [i].conteudo);
          atualiza descritor de lado esquerdo/direito
          vet [j] := i;
          se descritor_reg [i].ligacao <> 0
          entao inicio
                  j:= j+1;
                  i:=descritor_reg [i].ligacao
                fim;
          senao inicio
                  j:= j+1;
                  i:=seq [j];
                fim;
          d:=d+2;
        fim;
    i:=1;
    enquanto i <= (nreg - 1)
    faca inicio
          descritor_reg [vet [i]].ligacao:=vet[i+1];
          i:=i+1;
        fim;
    descritor_reg[vet[nreg]].ligacao :=0
    fim /* salva_registrador */
```

5) Procedimento - força-reg (seq, nreg, no, cond, resultado);

Força a alocação dos registradores indicados na sequência preferencial "seq". O pseudocódigo deste procedimento é apresentado a seguir.

```
procedimento forca_reg(seq,nreg,no,cond,resultado);
  inicio
    se busca (seq,nreg,no,regi)
    entao cond:= verdadeiro
    senao se existe_registrador_livre (seq,nreg,regi)
          entao cond := falso
          senao inicio
                  salva_registrador (nreg,seq);
                  cond := falso;
                fim;
    i:=1;
    enquanto i <= (nreg - 1)
    faca inicio
           descritor_reg [seq[i]].ligacao := seq [i+1];
           i := i+1;
         fim;
    descritor_reg [seq[nreg]].ligacao := 0;
    ocupa os registradores;
    descritor [resultado].tplocal := "registrador";
    descritor [resultado].local := seq [i];
  fim; /* forca_reg */
```

6) Procedimento - Calcula-end(no, base, desl, seq).

Calcula o deslocamento do objeto e determina o registrador de segmento a ser usado no endereçamento deste objeto. O pseudocódigo é apresentado a seguir.

```
procedimento calcula_end (no,base,desl,seg);
  inicio
    se no e variavel global
    entao se no e FAR
          entao inicio
                  seg := "ES";
                  escreva ( MOV    ES,endereco base do objeto);
                  base := 0;
                  desl := 0;
                fim
          senao inicio
                  seg := "DX";
                  base := 0;
                  desl := "deslocamento do objeto";
                fim
     senao inicio
            /* d1 (+) - numero de bytes da area de variaveis locais + 2 */
            /* d2 (-) - deslocamento da variavel na area de variaveis locais */
            /* d3 (+) - numero de bytes de parametros anteriores +6 */
            seg := "SS";
            base := "BP";
            salva_registrador (SI,1);
            se no e parametro
            entao inicio
                    d:=d1+d2;
                    escreva ( MOV    SI,d);
                  fim
            senao inicio
                    d:=d1+d3;
                    escreva ( MOV    SI,d);
                  fim;
          fim;
   fim. /* calcula_end */
```

7) Procedimento - carrega-valor (reg, nreg, no, base, desl, seq);

Gera as instruções necessárias para carregar o objeto em questão (Base + desl) no registrador reg. O pseudocódigo é apresentado a seguir.

```
procedimento carrega_valor (reg,nreg,no,base,desl,seg);
  inicio
    se reg mod 3 = 0
    entao nbyte := 2
    senao nbyte := 1;
    se no e variavel
    entao enquanto nreg <> 0
          faca inicio
                escreva ( MOV    reg,[base+des]);
                reg := descritor_reg [reg].ligacao;
                desl := desl - nbyte;
                nreg := nreg - 1
               fim
    senao inicio /* e constante */
            verifica_tipo (no,tipo_no);
            se tipo = (char,short)
            entao escreva ( MOV    reg,constante)
            senao ajusta_cte (reg,nreg,LI[no].tipo)
          fim
  fim; /* carrega_valor */
```

CAPÍTULO 6

CONCLUSÕES

Como já salientado este documento descreve o projeto básico do gerador de código de um compilador da linguagem C para o microprocessador 8086 da INTEL CORPORATION . Como trabalhos futuros destacam-se as fases de implementação, depuração, e teste. O gerador de código será programado na linguagem C utilizando apenas os comandos e as estruturas que estão sendo implementados neste compilador.

Todo o projeto visa tornar disponível um compilador C que se mostre eficiente e que se compare ao "software" não-nacional já disponível no País.

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, A.V.; ULLMAN, J.D. *Principles of compiler design.* 3. ed. Reading, M.A., Addison-Wesley, 1979.

AKIN, T.A.; LEBLANC, R.J. The design and implementation of a code generation tool. *Software-Practice and Experience*, 12(3):1027-1041, 1982.

CENTRO TECNOLÓGICO PARA INFORMÁTICA (CTI) *Conjunto de programas para sistemas de desenvolvimento de microprocessadores;* Contrato Embratel-CTI, Segunda fase - projeto básico. Campinas, Jul. 1985, v.1. (RF2-EB-01-001/00).

RECTOR, R.; ALEXY, G. *The 8086 book.* Berkeley, CA, Osborne-Mcgraw Hill, 1980.

APÊNDICE A

PROCEDIMENTOS DE PROPÓSITO GERAL

A seguir apresentam-se os procedimentos para a tradução de alguns operadores da linguagem C.

```
procedimento traduz_binop (contexto,OP,desc_esq,desc_dir,resultado,no);
  inicio
    caso OP seja:
      *   : traduz_mult (contexto,desc_esq,desc_dir,resultado,no);
      /   : traduz_div (contexto,desc_esq,desc_dir,resultado,no);
      %   : traduz_mod (contexto,desc_esq,desc_dir,resultado,no);
      +   : traduz_soma (contexto,desc_esq,desc_dir,resultado,no);
      -   : traduz_sub (contexto,desc_esq,desc_dir,resultado,no);
      >>  : traduz_desldir (contexto,desc_esq,desc_dir,resultado,no);
      <<  : traduz_deslesq (contexto,desc_esq,desc_dir,resultado,no);
      <   : traduz_menor (contexto,desc_esq,desc_dir,resultado,no);
      >   : traduz_maior (contexto,desc_esq,desc_dir,resultado,no);
      <=  : traduz_menorigual (contexto,desc_esq,desc_dir,resultado,no);
      >=  : traduz_maiorigual (contexto,desc_esq,desc_dir,resultado,no);
      ==  : traduz_igual (contexto,desc_esq,desc_dir,resultado,no);
      !=  : traduz_diferente (contexto,desc_esq,desc_dir,resultado,no);
      &   : traduz_earit (contexto,desc_esq,desc_dir,resultado,no);
      ^   : traduz_ou_excl_arit (contexto,desc_esq,desc_dir,resultado,no);
      |   : traduz_ou_arit (contexto,desc_esq,desc_dir,resultado,no);
      &&  : traduz_elog (contexto,desc_esq,desc_dir,resultado,no);
      ||  : traduz_oulog (contexto,desc_esq,desc_dir,resultado,no);
    fim de caso
  fim; /* traduz_binop */
```

```
procedimento traduz_asgnop(contexto,OP,desc_esq,desc_dir,resultado,no);
 inicio
   caso OP seja:
     =   : traduz_atrib (contexto,desc_esq,desc_dir,resultado,no);
     +=  : traduz_atrib_incr (contexto,desc_esq,desc_dir,resultado,no);
     -=  : traduz_atrib_dec (contexto,desc_esq,desc_dir,resultado,no);
     *=  : traduz_atrib_mult (contexto,desc_esq,desc_dir,resultado,no);
     /=  : traduz_atrib_div (contexto,desc_esq,desc_dir,resultado,no);
     %=  : traduz_atrib_mod (contexto,desc_esq,desc_dir,resultado,no);
     >>= : traduz_atrib_desldir (contexto,desc_esq,desc_dir,resultado,no);
     <<= : traduz_atrib_deslesq (contexto,desc_esq,desc_dir,resultado,no);
     &=  : traduz_atrib_e (contexto,desc_esq,desc_dir,resultado,no);
     ^=  : traduz_atrib_ouexcl (contexto,desc_esq,desc_dir,resultado,no);
     !=  : traduz_atrib_ou (contexto,desc_esq,desc_dir,resultado,no);
   fim de caso
 fim; /* traduz_asgnop */
```

```
procedimento traduz_unario(contexto,OP,desc_esq,resultado,no);
 inicio
   caso OP seja:
     -  : traduz_menos (contexto,desc_esq,resultado,no);
     *  : traduz_apont (contexto,desc_esq,resultado,no);
     &  : traduz_ end (contexto,desc_esq,resultado,no);
     !  : traduz_neg (contexto,desc_esq,resultado,no);
     ~  : traduz_compl (contexto,desc_esq,resultado,no);
     ++ : traduz_incr (contexto,desc_esq,resultado,no);
     -- : traduz_decr (contexto,desc_esq,resultado,no);
   fim de caso
 fim; /* traduz_unario */
```

A.1 - ESQUEMATIZAÇÃO DOS PROCEDIMENTOS DE GERAÇÃO DE CÓDIGO PARA ALGUNS OPERADORES

```
procedimento traduz_soma (contexto,desc_esq,desc_dir,resultado,no);
  inicio
    tipo_esq := descritor [desc_esq].tipo;
    tipo_dir := descritor [desc_dir] .tipo;
    se tipo_esq <> tipo_dir
    entao compatibiliza_tipo (desc_esq,desc_dir,tipooperacao)
    senao caso tipo_esq seja :
           char  : tipooperacao := 1;
           short : tipooperacao := 2;
           long  : tipooperacao := 3;
           float : tipooperacao := 4;
           double: tipooperacao := 5;
         fim de caso;
   caso tipooperacao seja :
     1 : inicio  /* prepara operandos para soma do tipo "CHAR" */
          caso descritor [desc_esq].tplocal seja :
            men : inicio
                  toma_end (desc_esq,endoperesq);
                  se descritor[desc_dir].tplocal ="registrador"
                  entao inicio
                          oper1 := descritor[desc_dir].local;
                          oper2 := endoperesq;
                        fim;
                  senao se descritor[desc_dir].tplocal ="const"
                        entao inicio
                                aloca_reg (reg,cond,char,no_dir,var_auxiliar);
                                se nao cond
                                entao escreva(  MOV reg,desc_dir[local]);
                                oper1 := reg,
                                oper2 := endoperesq;
                              fim
                        senao inicio
                                toma_end (desc_dir,endoperdir);
                                aloca_reg (reg,cond,char,no_dir,var_auxiliar);
                                oper1 :=reg;
                                oper2 := endoperesq
                              fim;
                fim;
```

```
const : inicio
          se descritor [desc_dir].tplocal = "registrador"
          entao inicio
                  oper1 := descritor[desc_dir].local;
                  oper2 := descritor [desc_esq].local;
                fim
          senao inicio
                  aloca_reg (reg,cond,char,no-esq,var-auxiliar);
                  se nao cond
                  entao escreva (  MOV     reg,descritor[desc_esq].tplocal);
                  oper1 := reg;
                  se descritor [desc_dir].tplocal = "const"
                  entao oper2 := descritor [desc_dir].local;
                  senao inicio
                          toma_end(desc_dir,endoperdir);
                          oper2 := endoperdir
                          fim
                  fim
          fim ;

   registrador : inicio
                  oper1 := descritor[desc_esq].local;
                  se descritor[desc_dir].tplocal = "registrador"
                  entao oper2 := descritor[desc_dir].local
                  senao se descritor[desc_dir].tplocal ="const"
                        entao oper2 := descritor[desc_dir].local
                        senao inicio
                                toma_end (desc_dir,endoperdir);
                                oper2 := endopeerdir
                              fim
                fim;
   fim de caso;
   escreva (   ADD     oper1,oper2);
   testa_contexto (contexto,oper1,no,tipooperacao,resultado);
  fim;

2 : inicio /* prepara operandos para soma do tipo "SHORT' */
    caso descritor [desc_esq].tplocal seja :
     men : inicio
            toma_end (desc_esq,endoperesq);
            se descritor[desc_dir].tplocal ="registrador"
            entao inicio
                    oper1 := descritor[desc_dir].local;
                    oper2 := endoperesq;
                  fim;
            senao se descritor[desc_dir].tplocal ="const"
                  entao inicio
                          aloca_reg (reg,cond,short,no_dir,var_auxiliar);
                          se nao cond
                          entao escreva(      MOV     reg,descritor[desc_dir].local);
                          oper1 := reg,
                          oper2 := endoperesq;
                        fim
                  senao inicio
                          toma_end (desc_dir,endoperdir);
                          aloca_reg (reg,cond,char,no_dir,var_auxiliar);
                          oper1 :=reg;
                          oper2 := endoperesq
                        fim;
```

```
      const : inicio
               se descritor [desc_dir].tplocal = "registrador"
               entao inicio
                      oper1 := descritor[desc_esq].local;
                      oper2 := descritor [desc_dir].local;
                     fim
               senao inicio
                      aloca_reg (reg,cond,short,no-esq,var-auxiliar);
                      se nao cond
                      entao escreva (  MOV     reg,descritor[desc_esq].local);
                      oper1 := reg;
                      se descritor [desc_dir].tplocal = "const"
                      entao oper2 := descritor [desc_dir].local
                      senao inicio
                             toma_end(desc_dir,endoperdir);
                             oper2 := endoperdir
                            fim
                     fim
              fim ;

      registrador : inicio
                     oper1 := descritor[desc_esq].local;
                     se descritor[desc_dir].tplocal = "registrador"
                     entao oper2 := descritor[desc_dir].local
                     senao se descritor[desc_dir].tplocal ="const"
                           entao oper2 := descritor[desc_dir].local
                           senao inicio
                                  toma_end (desc_dir,endoperdir);
                                  oper2 := endopeerdir
                                 fim
                    fim;
      fim de caso;
      escreva (   ADD     oper1,oper2);
      testa_contexto (contexto,oper1,no,tipooperacao,resultado);
     fim;

3 : inicio /* prepara operandos para soma do tipo "LONG" */
      caso descritor[desc_esq].tplocal seja :
        men : inicio
               toma_end (desc_esq,endoperesq);
               se descritor [desc_dir].tplocal = "registrador"
               entao inicio
                      oper1 := descritor [desc_dir].local;
                      oper2 := descritor_reg[oper1].ligacao;
                     fim
               senao se descritor [desc_dir].tplocal = ("const","men")
                     entao inicio /* aloca r1 e r2 */
                            aloca_reg (reg,cond,long,no_dir,var_auxiliar);
                            se descritor [desc_dir].tplocal ="const"
                            entao ajusta_cte(reg,descritor[desc_dir].local);
                            senao inicio
                                   toma_end (desc_dir,endoperdir);
                                   escreva (   MOV     r1,endoperdir);
                                   escreva (   MOV     r2,endoperdir+2);
                                  fim;
                            oper1 := r1;
                            oper2 := r2;
                           fim;
               oper3 := endoperesq;
               oper4 := endoperesq + 2;
              fim;
```

```
registrador : inicio
                oper1 := descritor[desc_esq].local;
                oper2 := descritor_reg[oper1].ligacao;
                se descritor[desc_dir].tplocal ="registrador"
                entao inicio
                        oper3 := descritor[desc_dir].local;
                        oper4 := descritor_reg[oper3].ligacao;
                      fim;
                senao se descritor [desc_dir].tplocal = "const"
                      entao inicio
                              aloca_reg(reg,cond,long,no_dir,var_auxiliar);
                              ajusta_cte (reg,2,descritor[desc_dir].local);
                              oper3 := reg;
                              oper4 := descritor_reg[oper3].ligacao;
                            fim
                      senao inicio
                              toma_end (desc_dir,endoperdir);
                              oper3 := endoperdir;
                              oper4 := endoperdir + 2;
                            fim
              fim;
 const : inicio
          se descritor[desc_dir].tplocal = "registrador"
          entao inicio
                  oper3 := descritor[desc_dir].local;
                  oper4 := descritor_reg [oper3].ligacao;
                fim
          senao se descritor[desc_dir].tplocal ="men"
                entao inicio
                        toma_end (desc_dir,endoperdir);
                        oper3 := endoperdir;
                        oper4 := endoperdir +2;
                      fim
                senao inicio
                        aloca_reg (reg,cond,long,no_dir,var_auxiliar);
                        ajusta_cte (reg,2,descritor[desc_dir].local);
                        oper3 := reg;
                        oper4 := descritor_reg[oper3].ligacao;
                      fim
          aloca_reg (reg,cond,long,no_esq,var_auxiliar);
          ajusta_cte (reg,2,descritor[desc_esq].local);
          oper1 := reg,
          oper2 := descritor_reg[oper1].ligacao;
        fim;
fim;
escreva (   CLC     );
escreva (   ADC     oper2,oper4);
escreva (   ADC     oper1,oper3);
testa_contexto (contexto,oper1,no,tipooperacao,resultado);
fim;
```

```
   4 : inicio /* prepara os operandos para soma do tipo "FLOAT" */
         empilha operando esquerdo;
         empilha operando direito;
         chama rotina somafloat;  /* resultado em r1,r2 */
         testa_contexto (contexto,r1,no,tipooperacao,resultado);
       fim;

    5 : inicio
          empilha operando esquerdo;
          empilha operando direito;
          chama rotina somadouble ; /* resultado em r1,r2,r3,r4 */
          testa_contexto (contexto,r1,no,tipoperacao,resultado);
        fim;
  fim de caso
fim; /* traduz_soma */
```

```
procedimento compatibiza_tipo (desc_esq,desc_dir,tipooperacao);
  inicio
    tipoesq := descritor[desc_esq].tipo;
    tipodir := descritor[desc_dir].tipo;
    se (tipoesq = "char" e dipodir = "short") ou (tipoesq = "short" e tipodir = "char")
    entao inicio
            extende sinal do operando char;
            tipooperacao := 2;
          fim
    senao se tipoesq = "char" ou tipodir = "char"
          entao inicio
                  empilha_operando_char;
                  caso outrooperando seja :
                    long : inicio
                             chama rotina de conversao de "char" em "long";
                             /* resultado retorna em r1,r2 */
                             tipooperacao := 3;
                             fim ;
                    float : inicio
                              chama rotina de conversao de "char" em "float";
                              /* resultado retorna em r1,r2 */
                              tipooperacao := 4;
                            fim;
                    double : inicio
                               chama rotina de conversao de "char" em "double";
                               /* resultado retorna em r1,,r2,r3,r4 */
                               tipooperacao :=5;
                             fim;
                    fim de caso;
                 fim
          senao se tipoesq = "short" ou tipodir = "short"
                entao inicio
                       caso outrooperando seja :
                         long : inicio
                                  converte operando "short" em "long";
                                  tipooeracao := 3;
                                fim;
                         float : inicio
                                   empilha_operando_short;
                                   chama rotina que converte "short" em "float";
                                   /* resultado retorna em r1,r2 */
                                   tipooeracao := 4;
                                 fim;
                         double : inicio
                                    empilha_opeerando_short;
                                    chama rotina de conversao de "short" em "double";
                                    tipooperacao :=5;
                                  fim;
                         fim de caso;
                       fim
                senao se (tipoesq = "long" ou tipodir = "long")
                       entao inicio
                              empilha_operando_long;
                              caso outrooperando seja:
                                float : inicio
```

```
                                chama rotina de conversao de "long" em "float";
                                tipooperacao := 4;
                               fim;
                          double : inicio
                                    chama rotina de conversao de "long" em "double";
                                    tipooperacao := 5;
                                   fim
                        fim de caso
                       fim
                senao se tipoesq = "float" ou tipodir = "float"
                       entao inicio
                              empilha_operando_float
                              se outrooperando e double
                              entao inicio
                                      chama rotina de conversao de "float" em "double";
                                      tipooperacao := 5;
                                     fim
                             fim
fim; /* compatibiliza_tipo */
```

```
procedimento testa_contexto (contexto,oper1,no,tipoperacao,resultado);
 inicio
   caso tipooperacao seja :
     1 : inicio
          caso contexto seja :
            teste : libera oper1;
            parametro : inicio
                         escreva ( PUSH    oper1);
                         descritor [resultado].tplocal := "BP";
                         descritor [resultado].local := d;
                         descritor [resultado].seg := "SS";
                         libera oper1;
                        fim;
            valor1 : inicio
                      forca_reg (AX,1,no.cond,var_auxiliar);
                      salva_registrador (1,DX);
                      carrega oper1 em DX,AX;
                      descritor [resultado].local := "DX";
                      descritor [rresultado].tplocal := "registrador2";
                     fim;
            valor2 : inicio
                      se oper1 nao e AL
                      entao inicio
                             forca_reg (AL,1,no,cond,resultado)
                             carrega oper1 em AL;
                            fim;
                      descritor [resultado].tplocal := "regpequeno";
                     fim;
            valor3 : inicio
                      se oper1 nao e CL
                      entao inicio
                             forca_reg (C1,1,no,cond,resultado)
                             carrega oper1 em CL;
                            fim;
                      descritor [resultado].tplocal := "regpequeno";
                     fim;
            senao : inicio
                     descritor [resultado].tplocal := "regpequeno";
                     descritor [resultado].local := oper1
                    fim
          fim
         fim;
     2 : inicio
          caso contexto seja :
            teste : libera oper1;
            parametro : inicio
                         escreva ( PUSH    oper1);
                         descritor [resultado].tplocal := "BP";
                         descritor [resultado].local := d;
                         descritor [resultado].seg := "SS";
                         libera oper1;
                        fim;
            valor1 : inicio
                      se oper1 <> "AX"
                      entao forca_reg (AX,1,no.cond,var_auxiliar);
```

```
                salva_registrador (1,DX);
                carrega oper1 em DX,AX;
                descritor [resultado].local := "DX";
                descritor [rresultado].tplocal := "registrador2";
               fim;
        valor2 : inicio
                se oper1 nao e AX
                entao inicio
                       forca_reg (AX,1,no,cond,resultado)
                       carrega oper1 em AX;
                      fim;
                descritor [resultado].tplocal := "registrador";
               fim;
        valor3 : inicio
                escreva ( MOV    CL,oper1(H));
                descritor [resultado].tplocal := "regpequeno";
                descritor [resultado].local := "CL"
               fim;
        senao : inicio
                 descritor [resultado].tplocal := "registrador";
                 descritor [resultado].local := oper1
                fim
      fim
    fim;
3,4 : inicio
        caso contexto seja :
              teste : libera oper1 e descritor_reg[oper1].ligacao;
              parametro : inicio
                           oper2 := descritor_reg[oper1].ligacao;
                           escreva ( PUSH    oper2);
                           escreva ( PUSH    oper1);
                           descritor [resultado].tplocal := "BP";
                           descritor [resultado].local := d;
                           descritor [resultado].seg := "SS";
                           libera oper1 e oper2;
                          fim;
              valor1 : inicio
                         se oper1 <> "AX"
                         entao inicio
                                forca_reg (AX,1,no,cond,resultado);
                                carrega oper1 em "AX";
                               fim;
                         se oper1.ligacao <> "DX"
                         entao inicio
                                forca_reg (DX,1,no,cond,resultado);
                                carrega oper1 em "DX";
                               fim;
                         descritor [resultado].tplocal := "registrador2";
                        fim
              senao : inicio
                        descritor [resultado].tplocal := "registrador2";
                        descritor [resultado].local := oper1;
                       fim
        fim de caso
      fim;
```

```
    5 : inicio
          oper2 := descritor_reg [oper1].ligacao;
          oper3 := descritor_reg [oper2].ligacao;
          oper4 := descritor_reg [oper3].ligacao;
          caso contexto seja :
            teste : libera oper1,oper2,oper3,oper4;
            parametro : inicio
                          escreva ( PUSH    oper4 );
                          escreva ( PUSH    oper3 );
                          escreva ( PUSH    oper2 );
                          escreva ( PUSH    oper1 );
                          libera oper1,oper2,oper3,oper4;
                          descritor [resultado].tplocal := "BP";
                          descritor [resultado].local := d;
                          descritor [resultado].seg := "SS";
                        fim
            senao : inicio
                      descritor [resultado].tplocal := "registrador4";
                      descritor [resultado].local := oper1;
                    fim;
          fim de caso
        fim;
  fim de caso;
  descritor [resultado].tipo := tipooperacao;
fim; /* testa_contexto */
```

```
procedimento traduz_menos (contexto,desc_esq,resultado,no);
  inicio
    tipoesq := descritor [desc_esq].tipo;
    caso tipoesq seja:
      char : inicio
               caso descritor [desc_esq].tplocal seja :
                 men : inicio
                         aloca_reg (reg,cond,char,no_esq,var_auxiliar);
                         se nao cond
                         entao inicio
                                 toma_end (desc_esq,endoperesq);
                                 escreva (  MOV    reg,[endoperesq]);
                               fim;
                         oper1 := reg;
                       fim;

                 registrador : oper1 := descritor [desc_esq].local;

                 const : inicio
                           aloca_reg (reg,cond,char,descritor [desc_esq].local,var_auxiliar);
                           se nao cond
                           entao escreva ( MOV    reg,descritor [desc_esq].local);
                           oper1 := reg;
                         fim
               fim de caso;
               escreva ( NEG    oper1);
               testa_contexto (contexto,oper1,no,1,resultado);
             fim;

      short : inicio
               caso descritor [desc_esq].tplocal seja :
                 men : inicio
                         aloca_reg (reg,cond,short,no_esq,var_auxiliar);
                         se nao cond
                         entao inicio
                                 toma_end (desc_esq,endoperesq);
                                 escreva ( MOV    reg,[endoperesq]);
                               fim;
                         oper1 := reg;
                       fim;

                 registrador : oper1 := descritor [desc_esq].local;

                 const : inicio
                           aloca_reg (reg,cond,short,descritor [desc_esq].local,var_auxiliar);
                           se nao cond
                           entao escreva ( MOV    reg,descritor [desc_esq].local);
                           oper1 := reg;
                         fim
               fim de caso;
               escreva ( NEG    oper1);
               testa_contexto (contexto,oper1,no,2,resultado);
             fim;
```

```
long : inicio
        caso descritor [desc-esq].tplocal seja :
          men : inicio
                  aloca_reg (reg,cond,long,no_esq,var_auxiliar);
                  r1 := reg;
                  r2 := descritor_reg [reg].ligacao;
                  toma_end (desc_esq,endoperesq);
                  oper1 := endoperesq;
                  oper2 := endoperesq + 2;
                fim;
          registrador : inicio
                         oper1 := descritor [desc_esq].local;
                         oper2 := descritor_reg [oper1].ligacao;
                        fim;
          const : inicio
                    aloca_reg (reg,cond,long,descritor [desc_esq].local,var_auxiliar);
                    ajusta_cte (reg,2,descritor [desc_esq].local);
                    oper1 := reg;
                    oper2 := descritor_reg [oper1].ligacao;
                  fim;
        fim de caso;
        aloca_reg (reg,cond,long,var_auxiliar1,var_auxiliar);
        r1 := reg;
        r2 := descritor_reg [r1].ligacao;
        escreva (       MOV     r1,#0);
        escreva (       MOV     r2,#0);
        escreva (       CLC);
        escreva (       SUB     r2,oper2);
        escreva (       SBB     r1,oper1);
        oper1 := r1;
        oper2 := r2;
        testa_contexto (contexto,oper1,no,3,resultado);
       fim;

float : inicio
         caso descritor [desc_esq].tplocal seja :
           men : inicio
                   aloca_reg (reg,cond,float,no_esq,var_auxiliar);
                   se nao cond
                   entao inicio
                           toma_end (desc_esq,endoperesq);
                           escreva (     MOV     descritor_reg [reg].ligacao,endoperesq +2);
                           escreva (     MOV     reg,endoperesq);
                         fim;
                   oper1 := reg;
                   oper2 := descritor [oper1].ligacao;
                 fim;
           registrador : inicio
                          oper1 := descritor[desc_esq].local;
                          oper2 := descritor_reg [oper1].ligacao;
                         fim;
           const : inicio
                     aloca_reg (reg,cond,float,no_esq,var_auxiliar);
                     se nao cond
                     entao ajusta_cte (reg,2,descritor [desc_esq].local);
```

```
                        oper1 := reg;
                        oper2 := descritor_reg [oper1].ligacao;
                       fim;
              fim de caso;
              troca sinal de oper1;
              testa_contexto (contexto,oper1,no,4,resultado );
             fim;

      double : inicio
               caso descritor [desc_esq].tplocal seja :
                 men : inicio
                        aloca_reg (reg,cond,double,no_esq,var_auxiliar);
                        oper2 := descritor_reg [oper1].ligacao;
                        oper3 := descritor_reg [oper2].ligacao;
                        oper4 := descritor_reg [oper3].ligacao;
                        se nao cond
                        entao inicio
                               toma_end (desc_esq,endoperesq);
                               escreva ( MOV    oper4,endoperesq + 6)
                               escreva ( MOV    oper3,endoperesq + 4)
                               escreva ( MOV    oper2,endoperesq + 2)
                               escreva ( MOV    oper1,endoperesq )
                              fim;
                       fim;
                 registrador : inicio
                                oper1 := descritor [desc_esq].local;
                                oper2 := descritor_reg [oper1].ligacao;
                                oper3 := descritor_reg [oper2].ligacao;
                                oper4 := descritor_reg [oper3].ligacao;
                               fim;
                 const : inicio
                          aloca_reg (reg,cond,double,no_esq,var_auxiliar);
                          oper1 := reg;
                          oper2 := descritor_reg [oper1].ligacao;
                          oper3 := descritor_reg [oper2].ligacao;
                          oper4 := descritor_reg [oper3].ligacao;
                          se nao cond
                          entao ajusta_cte (reg,4,descritor [desc_esq].local);
                         fim;
               fim de caso;
               troca sinal de operando1
               testa_contexto (contexto,oper1,no,4,resultado);
    fim de caso
fim; /* traduz_menos */
```

```
procedimento traduz_atrib (contexto,desc_esq,desc_dir,resultado,no);
inicio
  toma_end (desc_esq,endoperesq);
  tipoesq := descritor [desc_esq].tipo;
  tipodir := descritor [desc_dir].tipo;
  r1 := descritor [desc_dir].local;
  caso tipoesq seja :
    char : inicio
            caso tipodir seja :
              char : escreva ( MOV    endoperesq,r1);
              short : escreva ( MOV    endoperesq,r1(H));
              long : inicio
                      r2 := descritor_reg [r1].ligacao;
                      escreva ( MOV    endoperesq,r2(H));
                     fim;
              float : inicio
                       r2 := descritor_reg [r1].ligacao;
                       escreva ( PUSH    r2);
                       escreva ( PUSH    r1);
                       escreva ( PUSH    #2);
                       escreva ( CALL    convflsh);
                       escreva ( ADD    SP,#6);
                       /* resultado retorna em r1 */
                       escreva ( MOV    endoperesq,r1(H));
                      fim;
              double : inicio
                        r2 := descritor_reg [r1].ligacao;
                        r3 := descritor_reg [r2].ligacao;
                        r4 := descritor_reg [r3].ligacao;
                        escreva ( PUSH    r4);
                        escreva ( PUSH    r3);
                        escreva ( PUSH    r2);
                        escreva ( PUSH    r1);
                        escreva ( PUSH    #4);
                        escreva ( CALL    convdsh);
                        escreva ( ADD    SP,#10);
                        escreva ( MOV    endoperesq,r1(H));
                       fim;
             fim de caso;
             libera registradores
             testa_contexto_atrib (contexto,desc_esq,1,no,resultado);
            fim;

    short : inicio
             caso tipodir seja :
               char : inicio
                       compatibiliza_tipo (short,char,tipooperacao);
                       escreva ( MOV    endoperesq,r1);
                      fim;
               short : escreva ( MOV    endoperesq,r1);
               long : inicio
                       r2 := descritor_reg [r1].ligacao
                       escreva ( MOV    endoperesq,r2);
                      fim;
```

```
                float : inicio
                         r2 := descritor_reg [r1].ligacao;
                         escreva ( PUSH    r2);
                         escreva ( PUSH    r1);
                         escreva ( PUSH    #2);
                         escreva ( CALL    convflsh);
                         escreva ( ADD     SP,#6);
                         /* resultado retorna em r1 */
                         escreva ( MOV     endoperesq,r1);
                        fim;
                double : inicio
                          r2 := descritor_reg [r1].ligacao;
                          r3 := descritor_reg [r2].ligacao;
                          r4 := descritor_reg [r3].ligacao;
                          escreva ( PUSH    r4);
                          escreva ( PUSH    r3);
                          escreva ( PUSH    r2);
                          escreva ( PUSH    r1);
                          escreva ( PUSH    #4);
                          escreva ( CALL    convdsh);
                          escreva ( ADD     SP,#10);
                          escreva ( MOV     endoperesq,r1);
                         fim;
               fim de caso;
               libera registradores;
               testa_contexto_atrib (contexto,desc_esq,2,no,resultado);
              fim;
long : inicio
        r2 := descritor_reg [r1].ligacao;
        caso tipodir seja :
          char : inicio
                   compatibiliza_tipo (desc_esq,desc_dir,tipooperacao);
                   /* resultado retorna em r1,r2 */
                   escreva ( MOV     endoperesq + 2,r2);
                   escreva ( MOV     endoperesq ,r1);
                  fim;
          short : inicio
                   compatibiliza_tipo (desc_esq,desc_dir,tipooperacao);
                   /* resultado retorna em r1,r2 */
                   escreva ( MOV     endoperesq + 2,r2);
                   escreva ( MOV     endoperesq ,r1);
                  fim;
          long : inicio
                  escreva ( MOV     endoperesq + 2,r2);
                  escreva ( MOV     endoperesq ,r1);
                 fim;
          float : inicio
                   escreva ( PUSH    r2);
                   escreva ( PUSH    r1);
                   escreva ( PUSH    #2);
                   escreva ( CALL    convfll);
                   escreva ( ADD     SP,#6);
                   escreva ( MOV     endoperesq + 2,r2);
                   escreva ( MOV     endoperesq,r1);
                  fim;
```

```
          double : inicio
                    r3 := descritor_reg [r2].ligacao;
                    r4 := descritor_reg [r3].ligacao;
                    escreva ( PUSH    r4);
                    escreva ( PUSH    r3);
                    escreva ( PUSH    r2);
                    escreva ( PUSH    r1);
                    escreva ( PUSH    #4);
                    escreva ( CALL    convdl);
                    escreva ( ADD     SP,#10);
                    escreva ( MOV     endoperesq,r1);
                   fim;
         fim de caso;
         libera registradores
         testa_contexto_atrib (contexto,desc_esq,3,no,resultado);
       fim;
float : inicio
         r2 := descritor_reg [r1].ligacao;
         caso tipodir seja :
           char,short,long : inicio
                               compatibiliza_tipo (desc_esq,desc_dir,tipooperacao);
                               escreva ( MOV     endoperesq +2,r2);
                               escreva ( MOV     endoperesq,r1);
                              fim;
           float : inicio
                    escreva ( MOV     endoperesq +2,r2);
                    escreva ( MOV     endoperesq,r1)
                   fim;
           double : inicio
                     r3 := descritor_reg [r2].ligacao;
                     r4 := descritor_reg [r3].ligacao;
                     escreva ( PUSH    r4);
                     escreva ( PUSH    r3);
                     escreva ( PUSH    r2);
                     escreva ( PUSH    r1);
                     escreva ( PUSH    #4);
                     escreva ( CALL    convdfl);
                     escreva ( ADD     SP,#10);
                     escreva ( MOV     endopereq +2,r2);
                     escreva ( MOV     endoperesq,r1);
                    fim;
         fim de caso;
         testa_contexto_atrib (contexto,desc_esq,4,no,resultado);
       fim;
```

```
    double : inicio
               r2 := descritor_reg [r1].ligacao;
               r3 := descritor_reg [r2].ligacao;
               r4 := descritor_reg [r3].ligacao;
               se tipodir = (char,long,short,float)
               entao compatibiliza_tipo (desc_esq,desc_dir,tipooperacao);
               escreva ( MOV    endoperesq+6,r4);
               escreva ( MOV    endoperesq+4,r3);
               escreva ( MOV    endoperesq+2,r2);
               escreva ( MOV    endoperesq,r1);
               libera registradores;
               libera registradores;
               testa_contexto_atrib (contexto,desc_esq,5,no,resultado);
             fim;
  fim de caso;
fim; /* traduz_atrib */
```

```
procedimento testa_contexto_atrib (contexto,desc_esq,tipo,no,resultado);
inicio
  toma_end (desc_esq,endoperesq);
  se contexto = valor4
  entao inicio
         caso tipo seja:
           1 : inicio
                 aloca_reg (reg,cond,char,no,var_auxiliar);
                 escreva ( MOV    reg,endoperesq);
                 descritor [resultado].tplocal := "regpequeno";
               fim;
           2 : inicio
                 aloca_reg (reg,cond,short,no,var_auxiliar);
                 escreva ( MOV    reg,endoperesq);
                 descritor [resultado].tplocal := "registrador";
               fim;
           3,4 : inicio
                   aloca_reg (reg,cond,long,no,var_auxiliar);
                   r2 := descritor_reg [reg].ligacao;
                   cccreva ( MOV    r2,endoperesq + 2);
                   escreva ( MOV    reg,endoperesq);
                   descritor [resultado].tplocal := "registrador2";
                 fim;
           5   : inicio
                   aloca_reg (reg,cond,double,no,var_auxiliar);
                   r2 := descritor_reg [reg].ligacao;
                   r3 := descritor_reg [r2].ligacao;
                   r4 := descritor_reg [r3].ligacao;
                   escreva ( MOV    r4,endoperesq + 6);
                   escreva ( MOV    r3,endoperesq + 4);
                   escreva ( MOV    r2,endoperesq + 2);
                   escreva ( MOV    reg,endoperesq);
                   descritor [resultado].tplocal := "registrador4";
                 fim;
         fim de caso;
         descritor [resultado].local := reg
  senao inicio
         descritor [resultado].tplocal := "memoria";
         descritor [resultado].local := descritor [desc_esq].local;
         descritor [resultado].seg := descritor [desc_esq].seg;
       fim;
  descritor [resultado].tipo := tipo;
fim; /* testa_contexto_atrib */
```

```
procedimento traduz_indice (contexto,desc_esq,desc_dir,resultado,no);
inicio
  se descritor [desc_dir].tipo = "long"
  entao chama rotina que multiplica par (DX AX) por fator_indice
         /* resultado retorna em DX AX */
  senao multiplica AX por fator_indice;
       /* resultado em DX AX */
  se descritor [desc_esq].seg = "ES" /* objeto e FAR */
  entao inicio
          forca_reg (CX,1,no,cond,resultado);
          CX := 4 bits menos significativos de AX;
          BX := BX + CX;
          CX := #16;
          AX := DX,AX / CX;
          CX := ES;
          AX := AX + CX;
          ES := AX;
          libera CX,AX,DX
          descritor [resultado].base := "BX";
          descritor [resultado].desl := 0;
        fim
  senao se descritor [desc_esq].seg = "DS"
        entao inicio
                BX := BX + AX;
                descritor [resultado].base := "BX";
                descritor [resultado].desl := 0;
                libera AX,DX;
              fim
        senao inicio  /* seg = "SS" e variavel e local */
                SI := SI + AX;
                libera AX,DX;
                descritor [resultado].tplocal := descritor [desc_esq].tplocal; /*  BP */
                descritor [resultado].local := descritor [desc_esq].local;     /*  SI */
              fim;
  descritor [resultado].seg := descritor [desc-esq].seg;
  descritor [resultado].tipo := descritor [desc_esq].tipo;
  testa_contexto_indice;
fim; /* traduz_indice */
```

```
procedimento testa_contexto_indice (contexto,desc_esq,tipo,no,resultado);
inicio
  caso contexto seja :
   valor1 : inicio
              forca_reg (DX AX,no_atual,cond);
              se nao cond
              entao se descritor [resultado].tipo = long
                    entao carrega_valor (DX,2,descritor [resultado].tplocal,descritor [resultado].local);
                    senao inicio
                            escreva ( MOV    AX,descritor [resultado].tplocal + descritor [resultado].local);
                            escreva ( MOV    DX,#0);
                          fim;
             fim;
  valor2 : inicio
             forca _reg (Ax,1 no_atual,cond);
             se nao cond
             entao carrega_valor (AX,no_atual,descritor [resultado].tplocal,descritor [resultado].local);
           fim;
  valor3 : inicio
             forca_reg (CL,1,cond);
             se nao cond
             entao carrega_valor (CL,1,no_atual.descritor [resultado].tplocal,descritor [resultado].local);
           fim;
  valor4 : inicio
             aloca_reg (reg,cond,descritor [resultado].tipo,no);
             caso descritor [resultado].tipo seja :
               char,short : carrega_valor (reg,1,no_atual,descritor [resultado].tplocal,descritor [resultado].local);
               long : carrega_valor (reg,2,no_atual,descritor [resultado].tplocal,descritor [resultado].local);
               float,double : carrega_valor (reg,4,no_atual,descritor [resultado].tplocal,descritor [resultado].local);
             fim de caso;
           fim;
   teste : inicio
             escreva ( CLC );
             escreva ( ADC    descritor [resultado].tplocal + descritor [resultado].local,#0);
           fim;
```

```
parametro : inicio
             caso descritor [resultado].tipo seja :
               char,short : escreva ( PUSH    descritor [resultado].tplocal + descritor [resultado].local);
               long : inicio
                        escreva ( PUSH    descritor [resultado].tplocal + descritor [resultado].local);
                        local := local + 2;
                        escreva ( PUSH    descritor [resultado].tplocal + descritor [resultado].local);
                      fim;
               float,double : inicio
                                nreg := 4;
                                enquanto nreg > 0
                                faca inicio
                                       escreva ( PUSH    descritor [resultado].tplocal +descritor [resultado].local);
                                       local := local + 2;
                                       nreg := nreg-1;
                                     fim;
                              fim;
             fim de caso;
             descritor [resultado].seg := "SS";
            fim;

             fim de caso;
            fim; / *testa_contexto_indice */
```