LINGUAGENS DE IMPLEMENTAÇÃO DE SISTEMAS E A LINGUAGEM "LAPA"

GRAÇA BRESSAN

DISSERTAÇÃO APRESENTADA AO

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA

DA

UNIVERSIDADE DE SÃO PAULO

PARA OBTENÇÃO DO GRAU DE MESTRE

EM

MATEMÁTICA APLICADA

ORIENTADOR:

Prof.Dr.VALDEMAR W. SETZER

São Paulo, novembro de 1977

A meus pais

Meus sinceros agradecimentos

ao Prof. Dr. Valdemar W. Setzer, que me incentivou desde os primeiros cursos de pós-graduação e que além de excelente orientador, revelou-se um amigo incomparável,

aos amigos do Departamento de Matemática Aplicada do IME e do Setor de Matemática Aplicada do IFUSP cujo apoio e incentivo muito contribuiram para a conclusão deste trabalho,

às srtas. Regina, Sueli e Luzia, pela excelente datilografia.

São Paulo, novembro de 1977.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

O objetivo deste trabalho é apresentar um estudo de linguagens utilizadas para programação de sistemas bem como a descrição da linguagem LAPA definida com este fim.

Verificamos que para a descrição das diversas linguagens e comparação consistente entre as suas características seria necessário o estabelecimento de uma metodologia própria para esse estudo. Para isto, desenvolvemos no capítulo II uma lista de itens que consideramos relevantes na diversas linguagehs, e que fornece um esquema para descrição sucinta das mesmas e possibilita a sua comparação.

No capítulo III descrevemos diversas linguagens utilizadas para programação de sistemas de acordo com os itens definidos no capítulo II. Foram escolhidas para descrição as linguagens PL 360, BLISS, PL/I, XPL, PASCAL sequencial,PASCAL concorrente, ALGOL B6700, ESPOL e SPL.

No capítulo IV relacionamos os principais critérios levados em consideração no desenvolvimento de linguagens de programação de sistemas e as características das linguagens estudadas no capítulo III que visam satisfazer a estes critérios.

Apresentamos no capitulo V uma descrição da linguagem LAPA, desenvolvida para programação de sistemas e programação-científica do computador PADE,juntamente com as justificativas dos diversos recursos introduzidos na LAPA.

Como conclusão deste trabalho, apresentamos no capítulo VI um resumo esquemático das principais características das linguagens descritas no capítulo III, juntamente com as características da LAPA.

Uma descrição da linguagem LAPA sequencial através de diagramas sintáticos é apresentado no apêndice.

Usaremos, como notação, o símbolo "λ" para indicar a sequência nula de caracteres e todas as palavras reservadas das diversas linguagens aparecerão sublinhadas.

CAPÍTULO II

PROGRAMAÇÃO DE SISTEMAS E LINGUAGENS UTILIZADAS E UM ESQUEMA DE SUAS CARACTERÍSTICAS

1. PROGRAMAÇÃO DE SISTEMAS

Denominamos "Programação de Sistemas" à atividade de produção do "software" básico de um computador. Este "software" básico inclui os sistemas operacionais, interpretadores, compiladores, rotinas para processamento de entrada e saída, bibliotecas de rotinas, enfim, todo o conjunto de programas que constitui uma conexão entre a máquina física ("hardware") e os programas de aplicação.

2. ESCOLHA DE UMA LINGUAGEM PARA PROGRAMAÇÃO DE SISTEMAS

Uma das decisões na programação de sistemas é quanto à utilização ou da linguagem "assembler" ou da linguagem de alto-nível, considerando-se que a eficiência do código gerado, nesta modalidade de programação, é de fundamental importância.

As principais vantagens das linguagens de alto-nível são:

- Rapidez de programação e deteção de erros,
- Facilidade de manutenção e documentação de programas,
- Facilidade de introdução de alterações,
- Relativa portabilidade.

A maior desvantagem é a ineficiência do código gerado em relação a programas codificados em "assembler". Entretanto, esta ineficiência pode ser atenuada através de otimizações efetuadas pelos compiladores, como ocorre nos compiladores FORTRAN IV (H) e PL/I otimizador da série IBM/360, /370, e no uso de comandos ou procedimentos específicos para as características de cada máquina.

O programador de sistemas tem uma terceira opção nas linguagens de nível intermediário, como o PL 360, que procuram

reunir às vantagens das linguagens de máquina algumas das vantagens das linguagens de alto-nível. Elas diferenciam-se das linguagens de montagem pelo formato livre, utilização de estruturas de controle, blocos e declarações semelhantes aos do ALGOL 60 . Contudo, são introduzidos nestas linguagens somente os recursos que possam ser traduzidos para linguagem de máquina de forma simples e eficiente.

Outra possibilidade é a produção de programas híbridos, escrevendo-os em linguagem de alto-nível, reescrevendo em linguagem de máquina as partes cujas ineficiências forem críticas.

Todas linguagens estudadas neste trabalho são ou de nível intermediário ou de alto-nível.

3. LINGUAGENS UTILIZADAS PARA PROGRAMAÇÃO DE SISTEMAS

Podemos enumerar uma grande quantidade de linguagens que já foram utilizadas em programação de sistemas [41]. Nos limitaremos a citar algumas: ALGOL 60, ALGOL 68, ALGOL B6700, APL, BLISS, BCPL, ESPOL, PASCAL sequencial, PASCAL concorrente, PL/I, PL360, SPL (HP 3000) e XPL.

A maioria das linguagens desta lista não tem na programação de sistemas seu principal objetivo. Foram definidas especificamente para programação de sistemas as linguagens: BLISS , BCPL, ESPOL, PASCAL concorrente, PL360, SPL e XPL. Por outro lado o ALGOL 60, COGOL, FORTRAN e LISP têm poucos recursos apropriados para programação de sistemas.

Em programação de sistemas, as principais aplicações destas linguagens concentram-se no desenvolvimento de compiladores, interpretadores, sistemas operacionais e sistemas em "time-sharing" ou em tempo real.

a) Linguagens utilizadas para implementação de compiladores ou interpretadores: ALGOL 60, ALGOL 68, ALGOL B6700, APL, BLISS, BCPL, COBOL, FORTRAN, LISP, PASCAL sequencial, PL/I, PL360 , SPL e XPL.

b) Linguagens utilizadas para programação de sistemas operacionais, "time-sharing" ou partes destes sistemas: ALGOL B6700, ESPOL (um dialeto), PASCAL concorrente, PL/I e SPL.

É surpreendente o fato do FORTRAN ser uma das linguagens largamente utilizadas em programação de sistemas, apesar de não possuir as características desejáveis para este fim. As razões de seu sucesso está relacionada à portabilidade de seus programas e a facilidade de treinamento de pessoal desde que é uma das linguagens de programação de uso mais difundido.

Selecionamos para estudo as linguagens PL360, BLISS , PL/I, XPL, PASCAL sequencial, PASCAL concorrente, ALGOL B6700, ESPOL e SPL. Na seleção destas linguagens procuramos reunir as mais conhecidas e utilizadas em programação de sistemas ou que apresentaram, em uma primeira análise, características de interesse. Através de comparações entre estas linguagens, do ponto de vista dos recursos oferecidos, determinaremos as características mais relevantes e desejáveis às linguagens de programação de sistemas. A determinação destas características tem interesse tanto para a definição de novas linguagens quanto para a escolha, entre as linguagens disponíveis, da mais adequada para a produção do "software" desejado.

4. TÓPICOS PRINCIPAIS DAS LINGUAGENS DE PROGRAMAÇÃO DE SISTEMAS

Em nosso trabalho de comparação entre linguagens de programação de sistemas defrontamo-nos inicialmente com o problema de relacionar os ítens relevantes que nos dessem possibilidade de descrever e analisar os pontos mais importantes das diversas linguagens. Para isto, desenvolvemos a lista de ítens que expomos a seguir:

1- Tipos
   1.1 Tipos simples
   1.2 Tipos estruturados
   1.3 Definição de novos tipos

2- Declarações

2.1 Formas de declaração - declarações de tipos, constantes, variáveis, etc. Excluímos declaração de funções e procedimentos que se encontra no ítem 7.

2.2 Localização da declaração - início de bloco, por exemplo.

2.3 Inicialização.

2.4 Redefinição - associação de outro identificador e outro tipo a uma variável já declarada.

2.5 Domínio da declaração - bloco, por exemplo.

2.6 Pré-declarações - identificadores considerados declarados em um contexto externo ao programa e válidos dentro deste.

3- Alocação

3.1 Tipos de alocação

(a) Estática - áreas permanentes.

(b) Dinâmica - áreas alocadas durante a execução, automàticamente, no início de blocos ou procedimentos.

(c) Controlada - áreas alocadas durante a execução explícitamente através de comandos ou funções especiais de alocação e liberação de áreas.

3.2 Meio: memória e registradores.

3.3 Validade de apontadores - se existe verificação se a área apontada ainda está ativa.

4- Constantes e variáveis

5- Expressões

5.1 Formas de expressões

5.1.1 Expressões simples

(a) Operandos: constantes, variáveis e chamadas de funções.

(b) Operadores e precedência dos mesmos.

5.1.2 Expressões condicionais e outras.

5.2 Mistura de tipos e conversão.

6- Comandos

6.1 Rótulos de comandos

6.2 Bloco e comando composto

6.3 Atribuição

6.4 Estruturas de controle

6.4.1 Desvio incondicional ou escape de comandos

6.4.2 Comandos seletivos

6.4.3 Comandos repetitivos

6.5 Chamada de procedimento

6.6 Comando nulo

6.7 Outros

7- Declaração de função ou procedimento

7.1 Forma da declaração

7.2 Parâmetros

(a) Tipos de parâmetros formais

(b) Formas de passagem de parâmetros

(c) Associação dos parâmetros formais e atuais e verificação da correspondência.

7.3 Retorno de valor em funções

7.4 Dados acessíveis a procedimentos e funções:

(a) Locais - dados declarados no corpo da função.

(b) Globais - dados declarados em um meio que envolve a declaração da função.

(c) Parâmetros formais

7.5 Recursão

7.6 Procedimentos e funções externas

7.7 Efeitos colaterais em funções

7.8 Procedimentos e funções pré-declaradas

8- Programa

9- Recursos de entrada e saída

10- Recursos para manipulação de bits e caracteres

11- Processamento concorrente

11.1 Processos

11.2 Criação e destruição de processos

11.3 Exclusão mútua

11.4 Sincronização e comunicação entre processos concorrentes

11.5 Processamento de interrupção

12- Acesso a características de máquina

13- Comunicação com procedimentos em outras linguagens

14- Recursos em tempo de compilação - macros, por exemplo.

15- Recursos para "depuração" de programas - tal como "trace".

As linguagens PL360, BLISS, PL/I, XPL, PASCAL sequencial, PASCAL concorrente, ALGOL B6700, ESPOL e SPL foram descritas e analisadas segundo estes ítens.

CAPÍTULO III

DESCRIÇÕES DE LINGUAGENS DE PROGRAMAÇÃO DE SISTEMAS

Neste capítulo apresentaremos descrições detalhadas dos aspectos mais relevantes das seguintes linguagens utilizadas na programação de sistemas. PL360, BLISS, PL/I, XPL, PASCAL sequencial, PASCAL concorrente, ALGOL B6700, ESPOL e SPL. Tais descrições serão efetuadas segundo a lista de itens desenvolvida no capítulo II deste trabalho.

1. PL360

Foi definida por N. Wirth [48] e utiliza para implementação do compilador ALGOL W. Esta linguagem destina-se aos computadores IBM 360/370 sendo de nível intermediário entre o ASSEMBLER e as linguagens de alto nível.

1.1 - Tipos

1.1.1 Tipos simples - byte (8 bits), short integer e logical (32 bits), real (32 bits, 7 para o expoente e 24 para a mantissa) e long real (64 bits, 7 para o expoente e 56 para a mantissa).

1.1.2 Tipo estruturado - array

Características: (i) uma única dimensão
(ii) limites fixos

1.1.3 Definição de novos tipos - não há.

1.2 - Declarações

1.2.1 Formas de declaração

(a) Variáveis

t v_1, v_2,...,v_n

<u>array</u> n t v_1, v_2,...,v_n

onde t ε{<u>byte</u>, <u>short integer</u>, <u>integer</u>, <u>logical</u>, <u>real</u>, <u>long real</u> },

n - número inteiro sem sinal,

v_i - das formas: id
id = c
id = (c_1,...,c_m)

sendo id - um identificador;
c, c_1,...,c_m - constantes que constituem valores iniciais.

(b) Base de segmento - ver seção 1.12

<u>segment base</u> r_i

onde r_i é um dos registradores inteiros.

(c) Redefinição

t S_1, S_2,..., S_m

<u>array</u> n t S_1, S_2,...,S_m

t <u>register</u> r_1, r_2,...,r_m

onde tε{<u>byte</u>, <u>short integer</u>, <u>integer</u>, <u>logical</u>, <u>real</u>, <u>long real</u>},

n - número inteiro sem sinal,

S_i - das formas:

id <u>syn</u> v ou id <u>syn</u> k

r_i - da forma:

id <u>syn</u> reg

id - identificador,
v - variável,
k - número inteiro positivo,
reg- nome de um registrador.

(d) Procedimentos e funções - ver seção 1.7.

1.2.2 Localização - início de blocos.

1.2.3 Inicialização - permitida.

1.2.4 Redefinição - ver seção 1.2.1, ítem (c).

1.2.5 Domínio - bloco em que ocorreu a declaração.

1.2.6 Pré-declarações

(a) R0, R1,...,R15 - correspondentes aos 16 registradores inteiros sendo de tipo integer ou logical.

(b) F0, F2, F4 e F6 - correspondentes aos 4 registradores de ponto flutuante, sendo de tipo real.

(c) F01, F23, F45, F67 - correspondentes aos 4 registradores de ponto flutuante estendidos para dupla precisão, com tipo long real.

1.3 - Alocação

1.3.1 Tipos de alocação

. Estática - todas as variáveis declaradas dentro do programa.
. Controlada - através da instrução de máquina de chamada ao supervisor para reservar e liberar áreas de memória. O acesso a estas áreas é feito por meio de apontadores.

1.3.2 Meio - Todas as variáveis que não são explicitamente declaradas como registradores são alocadas na memória, sequencialmente, respeitada a fronteira requerida pelo seu tipo. O acesso é permitido a todas as células de

memória do programa, por meio de cálculo de endereços.

1.4 - Constantes e variáveis

(a) Constantes

. Numéricas - representadas na forma c t
onde t ε {X, S, R, L, λ} indicando os tipos byte, short integer, real, long real e integer.

c um número na base hexadecimal, precedido de " #/ ", ou na base decimal, na notação usual.

Na representação decimal, o sinal negativo é indicado pelo símbolo "_" e o expoente é separado da mantissa por um apóstrofo.

. Sequências de caracteres - representação entre aspas.

(b) Variável simples

id
id (x)

sendo id um identificador
x um índice de uma das formas

r + c, r - c , r ou c

onde r é um registrador inteiro, diferente do R0,

c é uma constante inteira.

O valor de x representa o deslocamento em bytes em relação ao início do array.

1.5 - Expressões - ver comando de atribuição.

1.6 - Comandos

1.6.1 Rótulos - múltiplos, representados por um identificador.

. Definição:

$r_1 : r_2 : \ldots r_n : C$.

onde r_i é um identificador de rótulo,

c é um comando.

. Utilização: go to r onde r é um rótulo.

. Domínio: bloco em que o rótulo foi definido.

1.6.2 Bloco e comando composto

(a) Bloco

. Representação:

begin d_1; d_2;...;d_n;c_1;c_2...;c_m end

ou

begin c_1; c_2;...;c_m end

onde $d_1,\ldots,d_n$ - são declarações,

$c_1,\ldots c_m$ - são comandos.

. Alocação das variáveis declaradas nos blocos: estática.

(b) Comando composto - não há.

1.6.3 Atribuição

. Atribuição de valor a uma célula de memória

v := r

onde v é uma variável,

r é um registrador.

. Atribuição de valor a um registrador.

(a) Formas:

r := a

r := uop a

r_i := s

r_i := ∂v

atr op a

sendo: r - registrador inteiro ou ponto flutuante;

r_i - registrador inteiro;

a - constante, variável ou registrador;

uop - operador unário;

s - sequência de caracteres;

∂ - referência, isto é, operador que obtém o endereço de uma variável;

v - variável;

atr - atribuição a registrador;

op - operador binário.

(b) Operadores:

- unários: abs, neg, negabs

- binários:

  . aritméticos: +, -, *, /, ++, --

  . lógicos sobre 32 bits: and, or, xor

  . de deslocamento: shll, shla, shrl, shra

- precedência dos operadores: todos os operadores têm a mesma precedência sendo que o cálculo das operações é feito da esquerda para a direita, após atribuição.

(c) Mistura de tipos de operandos: somente algumas

combinações são válidas.

(d) Conversão: não há.

(e) Efeitos colaterais:

- As operações de multiplicação e divisão com operandos inteiros, que especificam o registrador R_n, alteram, também, o registrador R_{n-1}.
- Além do resultado de uma operação, fica armazenado no registrador de código de condição do processador, uma das condições: maior, menor ou igual a zero, conforme o valor do resultado. Este código de condição pode ser testado por comandos posteriores.

Observações:

(i) A atribuição R1:=R2 + R1 tem resultado diferente de R1:=R1+R2 e o mesmo resultado que R1:=R2; R1:=R1+R1.

(ii) Os símbolos "_", neg, "-", "--" indicam respectivamente, sinal negativo de constante, operador unário de inversão de sinal, operador binário de subtração aritmética e operador binário de subtração lógica.

1.6.4 Estruturas de controle

1.6.4.1 Desvio incondicional

go to r

onde r é o identificador de rótulo.

1.6.4.2 Comandos seletivos

(a) if c then S_1 else S_2

onde c é uma condição de uma das formas:

c_1

c_1 or c_2 or c_3 ... or c_n

c_2 and c_2 and c_3...and c_n

sendo que $c_i \varepsilon \{r\, o_{rel}\, a,\ v_b, \neg v_b, o_{rel}, \underline{overflow}\}$,

r é um registrador,

$o_{rel} \varepsilon \{ =, \neg =, >, > =, <, < = \}$,

a é uma constante, variável ou registrador,

v_b variável de tipo byte.

S_1 é um comando simples, isto é,

diferente de if, while e for,

S_2 é um comando.

Quando um operador relacional (o_{rel}) aparece sem operandos ou a condição é overflow, ocorre uma verificação do código de condição.

(b) case r_i of begin s_1; s_2,...,s_n end

onde r_i é um registrador inteiro.

s_1,...,s_n são comandos.

1.6.4.3 Comandos repetitivos

(a) for x step c_1 until c_2 do s

onde x é uma atribuição a um registrador inteiro,

c_1 é uma constante inteira,

c_2 é uma constante, variável ou registrador do tipo integer ou short integer,

s é um comando.

(b) while c do s

onde c é uma constante como a do comando if;

s é um comando.

1.6.5 Chamada de procedimento

Forma: p

onde p - é o identificador do procedimento.

Não existe passagem de parâmetros na chamada de procedimento.

1.6.6 Comando nulo - indicado por null.

1.6.7 Outros - chamada de função: ver seção 1.12.

1.7 Declaração de função ou procedimento.

1.7.1 Forma da declaração

(i) Função - utilizada para introdução de instruções de máquina entre os comandos do programa (ver seção 1.12).

(ii) Procedimento - a declaração é da forma:

procedure p (r_i) S ou

Segment procedure p (r_i)S

onde: p é o identificador associado ao procedimento,

S é um comando,

r_i é um registrador inteiro. Na chamada

deste procedimento será colocado em r_i o endereço da instrução seguinte à de chamada. O retorno é feito através de uma instrução de desvio para o endereço contido em r_i.

Através da segunda forma de declaração, o procedimento irá constituir um segmento (ver seção 1.12) com registrador base de instruções diferente do utilizado pelo programa.

1.7.2 Parâmetros - não há.

1.7.3 Retorno de valores em funções - não se aplica.

1.7.4 Dados acessíveis a procedimentos - locais e globais.

1.7.5 Recursão - não há.

1.7.6 Procedimentos externos - não existem na linguagem definida por Wirth [48] . Foram introduzidos na versão de Malcolm [32].

1.7.7 Efeitos colaterais em funções - não se aplica.

1.7.8 Procedimentos e funções pré-declaradas - nenhum,na versão de Wirth.

1.8 Programa - é um comando.

1.9 Recursos de entrada e saída - as operações de entrada e saída devem ser efetuadas através de procedimentos intrínsecos (read, write, punch, etc.), codificados de acordo com o sistema operacional da máquina (DOS, OS, etc.).

1.10 Recursos para manipulação de caracteres - sequências de caracteres podem ser armazenadas como vectores de tipo byte e manipulados através de indexação e comandos convencionais da linguagem.

1.11 Processamento concorrente

Não existem, na linguagem, recursos para criação e controle de processos concorrentes. Contudo, através da fun-

ção SVC (chamada ao supervisor), o programador tem acesso às rotinas existentes no sistema operacional para tais fins.

1.12 Acesso às características de máquina

(a) Registradores - todos os registradores são acessíveis.

(b) Código de condição - pode ser testado nos comandos if e while.

(c) Instruções - todas as instruções de máquina do computador IBM/360 [25] são acessíveis, na forma de funções.

. Declaração de funções

function $f_1(t_1,c_1),\ldots,f_n(t_n,c_n)$

onde $f_1,\ldots,f_n$ são identificadores de funções,

$t_1,\ldots,t_n$ são constantes inteiras entre 0 e 10 que indicam os formatos das instruções,

$c_1,\ldots,c_n$ são os códigos de operação das instruções.

. Comando de chamada de função - é indicado por:

f

ou

$f(p_1,p_2,\ldots,p_n)$

onde f é o identificador da função,

$p_1,\ldots,p_n$ - podem ser constantes, variáveis, sequências de caracteres ou registradores; seus valores irão completar os campos não preenchidos da instrução de máquina.

(d) Segmentação e registradores base.

Dados e programa são endereçados em relação a registradores base diferentes. O registrador 15 foi escolhido como base de programa e o registrador 14 é implicitamente considerado como base de dados.

Outro registrador pode ser indicado como base de dados através da declaração de base de segmento (ver seção 1.2.1, ítem (b)). Para todas as declarações de variáveis que sucederem a declaração de base de segmento, o registrador especificado na declaração será utilizado como registrador base. A carga do endereço base neste registrador é feita na entrada do bloco em que ocorreu a declaração.

1.13 Comunicação com procedimentos em outras linguagens.

Na versão de Malcolm [32] , é permitida a comunicação com subprogramas em FORTRAN e ASSEMBLER.

1.14 Recursos em tempo de compilação - nenhum.

1.15 Recurso para depuração de programas.

Na versão de Malcolm existe uma opção de compilação que causa a impressão de informações na forma de "trace" durante a execução.

2. BLISS - BASIC LANGUAGE FOR IMPLEMENTATION OF SYSTEM SOFTWARE

Esta linguagem foi definida por Wulf, Russel e Habermann [51, 52, 53] e implementada no computador PDP-10 na Carnegie - Mellon University em 1970.

Vários compiladores, entre os quais os compiladores BLISS e WATFOR, foram codificados na linguagem BLISS, além de um sistema conversacional APL, rotinas de entrada e saída e partes de um sistema operacional.

2.1. Tipos

2.1.1 Tipos simples - palavra de 36 bits.

2.1.2 Tipo estruturado - "array" com as características:

- uma dimensão,
- número de elementos constante durante a execução,
- através de uma declaração pode ser associado um algoritmo de acesso aos seus elementos.

2.1.3 Definição de novos tipos - não há.

2.2. Declarações

2.2.1 Formas de declaração

(a) Áreas da memória

a $p_1, p_2, \ldots, p_n$

onde a ε {global, own, local, external} é o tipo de alocação,

$p_1, p_2, \ldots, p_n$ - são da forma:

s $ar_1 : ar_2 : \ldots : ar_m$

sendo s - identificador de uma estrutura já declarada ou quando omitido é assumida a estrutura vector (ver seção 2.2.6).

$ar_1, ar_2, \ldots, ar_m$ são das formas :

id

ou id $[e_1, \ldots, e_\ell]$

id - identificador associado à área declarada,

$e_1, \ldots, e_\ell$ - são expressões que devem ser calculadas em tempo de compilação, permitindo o cálculo do tamanho da área quando associadas à especificação de tamanho, da declaração da estrutura s.

(b) Registradores

<u>register</u> $r_1, r_2, \ldots, r_n$

ou

<u>register</u> $p_1, p_2, \ldots, p_n$

onde $r_1, \ldots, r_n$ são das formas:

s id $[e_1, \ldots, e_\ell] = e$

ou s id = e

sendo s - identificador de uma estrutura já declarada ou, se for omitida, será assumida a estrutura <u>vector</u> (seção 2.2.6).

id - identificador declarado como registrador,

$e, e_1, \ldots, e_\ell$ - expressões calculadas em tempo de compilação. O valor de e indica o número de um registrador da máquina.

$p_1, \ldots, p_n$ - como no ítem (a).

(c) Estruturas

structure s ℓ = t e_1

onde s - identificador da estrutura,

ℓ - lista de nomes formais que pode ser omitida e, quando presente, é da forma:

[$nome_1$, $nome_2$, ..., $nome_n$]

sendo que $nome_i$ é um nome formal;

t - especificação de tamanho da estrutura - que pode ser omitida e, quando presente, é da forma: [e_2]

sendo e_2 uma expressão envolvendo os nomes formais e cujo cálculo é efetuado durante a compilação da declaração de área; para isto, os nomes formais serão substituídos pelas expressões da declaração de área.

e_1 - expressão envolvendo os nomes formais e que constitui um algoritmo para o cálculo da posição de um elemento dentro da estrutura.

Exemplo:

```
begin
   structure ary2[i,j]=[i*j](.ary2+(.i-1)*j+(.j+1));
   own  ary2  x:y:z [10,20];
    :
   x[.a,.b] ← .y[.a,.b]
    :
end
```

Na compilação da declaração dos identificadores x, y e z, a estrutura ary2 é associada a estas áreas O tamanho de cada uma é calculada pela substituição dos nomes formais i e j da especificação de tamanho na declaração da estrutura, pelos valores 10 e 20 respectivamente, da declaração de área. Desta forma, o tamanho de cada área x, y e z será 200 palavras.

Na execução da atribuição x[.a,.b] ← .y[.a,.b], os valores de a e b irão substituir .i e .j e os endereços das áreas x e y irão substituir .ary2 da declaração da estrutura ary2, para o cálculo da posição dos elementos de índices .a e .b, dentro das áreas x e y.

(d) Associação de um nome a uma expressão

bind eq_1, eq_2, ..., eq_n

onde eq_1, ..., eq_n são da forma: id=e, sendo que

id - identificador,

e - expressão calculada na entrada do bloco em que ocorreu a declaração

Esta declaração torna id equivalente ao valor da expressão e.

(e) Macro - ver seção 2.14.

(f) Associação de uma outra estrutura a uma área já declarada

map p_1, p_2, ..., p_n

onde p_1, ..., p_n tem a mesma forma introduzida no item (a) desta seção.

(g) Mnemônicos de instruções de máquina - ver seção 2.12, item (b).

(h) Função - ver seção 2.7.

2.2.2 Localização da declaração - início de blocos.

2.2.3 Inicialização - não há.

2.2.4 Redefinição - a uma área já declarada pode ser associada uma outra estrutura, através da declaração map (ítem (f) da seção 2.2.1).

2.2.5 Domínio da declaração - bloco em que a mesma ocorreu.

2.2.6 Pré-declarações

vector - estrutura pré-definida como:

struture vector [i] = [i](.vector+.i) <0,36>

2.3. Alocação

2.3.1 Tipos de alocação

(a) Estática - ocorre nas declarações own, global e external, sendo que na declaração external não existe alocação de área e sim, equivalência com uma área declarada com o mesmo nome como global, em um módulo (ver seção 2.8) compilado separadamente.

(b) Dinâmica - em declarações local e register a alocação ocorre na entrada dos blocos.

2.3.2 Meio: registradores e memória

(a) Memória - identificadores definidos em declarações de áreas são alocados na memória. Esta área é alocada em múltiplos de palavras.

(b) Registradores - identificadores definidos em declarações register são associados a alguns dos 16 registradores da máquina.

2.3.3 Validade de apontadores - não é feita verificação.

2.4. Constantes e variáveis

(a) Constantes

. número inteiro sem sinal

. número em ponto flutuante

n.m , n. , n.mE+ ℓ ,
n.mE-ℓ ou n.m E ℓ

onde n,m e ℓ são números inteiros decimais.

. número octal - até 12 dígitos octais precedidos de "#".

. sequência de caracteres.

e , d ,
c e ou cd

onde c ε {ASCII, ASCIZ, SIXBIT, RADIX 50}

e - sequência de caracteres entre apóstrofos que internamente será ajustada à esquerda,

d - sequência de caracteres entre aspas que internamente será ajustada à direita.

. apontador de literal - esta constante é um apontador de uma palavra cujo conteúdo (literal) é especificado em tempo de compilação.
Representação: plit a

onde a é uma literal de uma das formas:

(i) expressão calculada em tempo de compilação;

(ii) sequência de caracteres eventualmente maior que uma palavra;

(iii) $(\ell_1, \ell_2, \ldots, \ell_n)$

sendo ℓ_i das formas (i) ou (ii) ou:

$y_i : z_i$

onde y_i é da forma (i),

z_i é uma literal.

(b) Variável simples

(i) Apontador - representado nas formas:

id_1

$id_2\ [e_1,e_2,\ldots,e_m]$

ou

e <p,t,idx,ind>

onde id_1 - identificador que não foi declarado com estrutura

id_2 - identificador declarado com estrutura,

$e_1,\ldots,e_m$ - expressões;

e,p,t,idx,ind - expressões cujos valores indicam:

e - endereço de uma palavra;

p - posição de um campo dentro da palavra, em termos do número de bits à direita do mesmo; se for omitido será considerado o valor 0;

t - número de bits no campo; se omitido será assumido como 36;

idx - indexador; se omitido será assumido o valor 0 indicando que não há indexação.

ind - endereçamento indireto se ind mod 2=1; se omitido será assumido o valor 0.

(ii) Conteúdo

Sendo p um apontador de um campo de uma palavra, os operadores ., @ e / obtém o conteúdo do campo apontado (ver seção 2.5.2.1, item (c)).

2.5. Expressões

2.5.1 Formas de expressões

Nas descrições que se seguem, e, e_o, e_1, ... indicam expressões.

2.5.1.1 Expressões simples

(a) Bloco e expressão composta

Bloco: begin $d_1;d_2;...;d_n;e_1;e_2;...;e_m$ end

ou $(d_1;d_2;...;d_n;e_1;e_2;...;e_m)$

onde d_1, d_2,..., d_n são declarações.

Expressão composta: begin $e_1;e_2;...;e_m$ end

ou $(e_1;e_2;...;e_m)$

O valor do bloco ou expressão composta é o resultado da expressão e_m, a menos que em seu interior seja executada uma expressão de escape.

Em blocos, a alocação de dados é como no ALGOL-60.

(b) Expressões constituídas de operandos e operadores.

(i) Operandos

- Apontadores
- Constantes,
- Chamada de função, da forma:

$$e_o(e_1,e_2,...,e_n)$$

onde e_o - expressão cujo valor indica o nome de uma função ou rotina,

$e_1,...,e_n$ - expressões que constituem os parâmetros atuais, sendo que n pode ser zero.

(ii) Operadores e precedência dos mesmos

Os níveis de precedência são 6:

1º Nível:

.e, @e, /e - operadores que obtém o conteúdo do campo apontado por e, sendo que @e equivade a .e<0,36, 0,0>
/e equivale a .(t←e)<0,36,.t<18,4>, .t<22,1>>

2º Nível:

e_1 ↑ e_2 - deslocamento lógico de e_1, sendo e_2 o número de bits que serão deslocados para a direita ou esquerda conforme o valor de e_2 seja positivo ou negativo.

3º Nível:

e_1 * e_2, e_1/e_2, e_1 mod e_2 - multiplicação, divisão e resto em aritmética inteira.

e_1 fmpr e_2, e_1 fdvr e_2 - multiplicação e divisão em aritmética de ponto flutuante.

4º Nível:

$-e_1$, e_1+e_2, e_1-e_2 - mudança de sinal, soma e subtração em aritmética inteira.

fneg e_1, e_1 fadr e_2, e_1 fsbr e_2 - mudança de sinal, soma e subtração em aritmética flutuante.

5º Nível:

e_1 eql e_2, e_1 neq e_2, e_1 lss e_2
e_1 leq e_2, e_1 gtr e_2, e_1 geq e_2 } operadores relacionais

6º Nível:

not e_1, e_1 and e_2, e_1 or e_2
e_1 xor e_2, e_1 eqv e_2 } operadores lógicos

$e_1 \leftarrow e_2$ - atribuição cujo resultado é o valor de e_2, sendo que $e_1 \leftarrow e_2 \leftarrow e_3$ equivale a $e_1 \leftarrow (e_2 \leftarrow e_3)$.

2.5.1.2 Expressões de controle

(a) Expressões condicionais

(i) if e_1 then e_2 else e_3

Efeito: e_2 se e_1 é impar,
e_3 se e_1 é par.

(ii) if e_1 then e_2

É equivalente a: if e_1 then e_2 else 0.

(b) Expressões repetitivas - o resultado é -1, exceto quando em seu interior for executada uma expressão de escape.

(i) while e_1 do e_2

Efeito: repete o cálculo de e_2 enquanto e_1 for ímpar.

(ii) do e_1 while e_2

É equivalente a (e_1; while e_2 do e_1)

(iii) until e_1 do e_2

É equivalente a while not e_1 do e_2

(iv) do e_1 until e_2

É equivalente a do e_1 while not e_2

(v) incr id from e_1 to e_2 by e_3 do e_4

(vi) decr id from e_1 to e_2 by e_3 do e_4

sendo que, nos dois comandos anteriores:

id- identificador considerado como register ou local e utilizado para controle da repetição,

from e_1 - pode ser omitido sendo assumido o valor $e_1=0$,

to e_2 - pode ser omitido, sendo assumido o valor $e_2=2^{35}-1$ no comando incr e $e_2=-2^{35}$ no comando decr,

by e_3 - pode ser omitido, sendo assumido o valor $e_3=1$

e_1, e_2 e e_3 são calculados apenas uma vez no início da repetição.

(c) Expressões de escape - são das formas:

c nível v

ou return v

onde c ε {exit, exitcompound, exitloop, exitblock, exitcond, exitcase, exitset, exitselect},

nível - da forma [e] ou, quando omitido, é considerado como [1],

v - expressão que pode ser omitida.

A expressão return v causa o término da função ou rotina que a contém. O resultado da função ou rotina será a expressão v.

As várias formas de exit permitem que um ou mais níveis de expressões sejam terminados resultando no valor v. O valor entre colchetes

indica o número de níveis que serão termina dos, a partir do mais interno.

O efeito dos vários comandos exit são:

exit - termina vários níveis de expressões, que podem ser blocos, expressões com postas, condicionais ou repetitivas;

exitblock - termina vários níveis de blocos;

exitcompound - termina vários níveis de ex pressões compostas;

exitloop - termina vários níveis de expres- sões de repetição;

exitcond - termina vários níveis de expres- sões condicionais;

exitcase - termina vários níveis de espres- sões case;

exitset - termina vários nívies de espres- sões contidas na parte set (ver item (d));

exitselect - termina vários níveis de expres sões select.

(d) Expressões seletivas

(i) case $e_1,e_2,\ldots,e_n$ of set $p_o;p_1;\ldots;p_m$ tes

onde $e_1,\ldots,e_n,p_o,\ldots,p_n$ são expressões

Efeito: Execução das expressões $p_{e_1};p_{e_2};$ p_{e_n} nesta ordem, resultando no valor de expressão p_{e_n}.

(ii) select $e_1,e_2,\ldots,e_n$ of nset $s_1:p_1;s_2:p_2;$ $;\ldots;s_m:p_m$ tesn

onde $e_1,\ldots,e_n,p_1,\ldots,p_m$ são expressões.

$s_1,\ldots,s_m$ são expressões ou as pa lavras reservadas

"always" ou "otherwise"

Efeito: As expressões $e_1,\ldots,e_n$ são calculadas, e para cada $i \mid 1 \leq i \leq m$:

se $[s_i = e_j$ para algum $j \mid 1 \leq j \leq n]$

ou $[s_i = \underline{always}]$

ou $[s_i = \underline{otherwise}$ e $S_k \neq e_j$ ou $S_k \neq$ always, p/ todos valores de j e k $\mid 1 \leq j \leq m$ e $1 \leq k \leq i-1]$

então p_i é executada.

O resultado da expressão select é o valor da última expressão p_i executada ou se nenhuma foi executada então o resultado é -1.

(e) Expressões de ativação de corotina

(i) create $e_1(p_1,p_2,\ldots,p_n)$ at e_2 lenght e_3 then e_4

onde e_1 - expressão cujo valor deve ser o nome de uma função ou rotina;

$p_1,\ldots,p_n$ - expressões que constituem os parâmetros atuais da chamada da rotina e_1;

e_2 e e_3 - expressões que indicam, respectivamente, o endereço base e o tamanho de uma área que constitui o contexto (ou pilha) da corotina e_1.

A expressão create não passa o controle para a corotina e_1. Após a execução de um retorno pela corotina e_1, será executada a expressão e_4 no contexto de e_1. O valor da expressão create é o nome da corotina e_1.

(ii) exchj (e_1,e_2)

onde e_1 - nome de uma corotina cujo con

texto já foi criado por uma expressão create;

e_2 - expressão.

A expressão exchj causa a passagem de controle para a corotina e_1. O valor e_2 será transmitido à corotina e_1, tornando-se o valor da mais recente expressão exchj executada por e_1 e que lhe tenha tirado o controle. A execução de e_1 continuará a partir desta última expressão exchj executada por e_1 e caso nenhuma tenha sido executada, começará do início de e_1.

2.5.2 Mistura de tipos e conversão - não se aplica.

2.6. Comandos - nenhum, a linguagem é orientada para expressões.

2.7. Declaração de procedimentos

2.7.1 Formas de declaração

function $f(id_1,\ldots,id_n) = e$

function $f = e$

routine $r(id_1,\ldots,id_n) = e$

routine $r = e$

global routine $r(id_1,\ldots,id_n) = e$

global routine $r = e$

forward $fr_1(e_1), fr_2(e_2),\ldots,fr_m(e_m)$

onde f - identificador de função;

r - identificador de rotina;

$id_1,\ldots,id_n$ - identificadores que constituem os parâmetros formais;

e - expressão que constitui o corpo da função ou rotina.

$fr_1,\ldots,fr_m$ - identificadores de funções ou rotinas que serão declaradas mais adiante no bloco atual;

$e_1,\ldots,e_m$ - expressões que indicam o número de parâmetros das funções ou rotinas $fr_1,\ldots,fr_m$.

A rotina distingue-se da função por não poder fazer referências a variáveis globais (isto é, declaradas fora das mesmas) nem efetuar chamadas de funções. Uma rotina declarada como global é conhecida em todos os módulos que constituem o programa.

2.7.2 Parâmetros

(a) Tipos de parâmetros formais - identificadores de variáveis ou funções.

(b) Formas de passagem de parâmetros - por valor, sendo que no caso do parâmetro atual ser um apontador, o efeito será de chamada por referência.

(c) Associação dos parâmetros atuais e formais e verificação da correspondência.
Pode haver mais parâmetros atuais que formais, o contrário poderá causar resultados incorretos durante a execução.

2.7.3 Retorno de valor de funções e rotinas - o valor retornado é definido pela expressão que constitui o corpo da função ou rotina.

2.7.4 Dados acessíveis a funções e rotinas

Funções: têm acesso a variáveis locais e globais.

Rotinas: não podem utilizar diretamente as variáveis globais.

2.7.5 Recursão - funções e rotinas podem ser recursivas.

2.7.6 Procedimentos externos - são rotinas declaradas como global em módulos (ver seção 2.8) compilados separadamente.

2.7.7 Efeitos colaterais em funções e rotinas - não existem restrições.

2.7.8 Funções pré-declaradas

Sendo p,p_1 e p_2 expressões cujos valores são apontadores para sequências de caracteres; e expressão qualquer.

scann (p) e scani (p) - retornam como valor o caráter apontado por .p.

replacen (p,e) e replacei (p,e)- substituem o valor apontado por .p pelo valor e.

copynn (p_1,p_2), copyni (p_1,p_2), copyin (p_1,p_2) e copyii(p_1,p_2) efetuam cópia do caráter apontado por .p_1 em .p_2.

incp (p) - avança o ponteiro .p para o próximo caráter.

sign (e) = -1 se e<0, 0 se e=0 e 1 se e>0.

abs (e) - valor absoluto de e

firstone (e) - se e≠0 então o resultado é o número de bits iguais a zero à esquerda do primeiro bit 1 do valor e; se e=0 então o resultado é -1.

2.8. Programa

Um programa é constituído de vários módulos compilados separadamente e reunidos através de um programa carregador.

Definição de módulo:

module m(id_1,...,id_n) = e eludom

onde m - identificador do módulo;

id_1,...,id_n - parâmetros;

e - expressão

Um módulo permite que outros módulos tenham acesso às suas rotinas e variáveis declaradas como global, e pode ter acesso à variáveis e rotinas de outros módulos, declarando -as como external.

2.9. Recursos de entrada e saída - nenhum.

2.10. Recursos para manipulação de caracteres - funções pré- declaradas (7.7).

2.11. Processamento concorrente - nenhum recurso.

Segundo Wulf, as expressões create e exchj podem ser modificadas para permitir a criação de processos concorrentes.

2.12. Acesso a características de máquina.

(a) Os operadores da linguagem correspondem a operadores de máquina (PDP-10).

(b) Uma instrução de máquina pode ser introduzida no código do programa na forma de chamada de função:

$$op(e_1, e_2, e_3, e_4)$$

onde op - mnemônico de instrução de máquina,

e_1 - expressão calculada em tempo de compilação e cujos 4 bits mais à direita indicam o registrador que será utilizado como acumulador,

e_2, e_3, e_4 - expressões que correspondem, respectivamente, aos campos de endereço relativo, indexador e indireto.

A declaração

machop $op_1=e_1$, $op_2=e_2$, ..., $op_n=e_n$ define os identificadores $op_1,\ldots,op_n$ como mnemônicos das instruções de máquina de códigos $e_1,\ldots,e_n$. As expressões $e_1,\ldots,e_n$ são calculadas em tempo de compilação.

(c) A especificação de apontador na forma e<p, t, idx, ind> da seção 2.4, item (b) corresponde à forma de especificação de palavra de endereço do computador PDP-10.

2.13. Comunicação com outras linguagens.

É possível mas, segundo observações dos autores (Wulf et alli [53]), tal comunicação será dificultada devido às diferentes convenções na utilização de registradores, estrutura de pilha de variáveis locais e passagem de parâmetros nas várias linguagens.

2.14. Recursos em tempo de compilação - macros.

(a) Definição

macro $d_1,d_2,\ldots,d_n$

onde $d_1,\ldots,d_n$ - são das formas:

$m(id_1,\ldots mid_k)$ = c $

ou m = c $

sendo m - identificador da macro;

$id_1,\ldots,id_k$ - identificadores

c - sequência de caracteres que não contenha o caráter "$".

(b) Chamada

$m(c_1,\ldots,c_k)$

ou m

onde m - identificador da macro;

$c_1,\ldots,c_k$ - sequência de caracteres com os sinais "(", ")", "[", "]", "<" e ">" balanceados.

As cadeias $c_1,\ldots,c_k$ irão substituir as ocorrências dos nomes $id_1,\ldots,id_k$ da definição da macro e as ocorrências de m no texto do programa serão substituídas pela sequência de caracteres da definição da macro, após a substituição dos nomes $id_1,\ldots,id_k$.

As chamadas da macro podem ser encaixadas.

2.15. Recursos para depuração de programas - nenhum.

3. PL/I

Esta linguagem foi definida pela IBM [24] com o objetivo de cobrir uma ampla faixa de aplicações. Um dos exemplos mais notáveis de sistema programado em linguagem de alto-nível é o MULTICS [15], desenvolvido no MIT. Este é um sistema operacional em "time-sharing" para o computador GE 645 e foi programado em uma versão do PL/I. Somente uma parte reduzida do sistema foi programada em "assembler".

Além do exemplo citado acima, muitos outros sistemas foram programados em PL/I ou versões do PL/I. Juntamente com o FORTRAN é uma das linguagens mais utilizadas em programação de sistemas.

3.1 - Tipos

3.1.1 Tipos simples

aritméticos: decimal fixed, binary fixed, decimal float, binary float, complex;

character - sequência de caracteres;

bit - sequência de bits;

picture - especificações de edição;

label - rótulos de comandos;

area - conjunto de localizações da memória;

event - para sincronização de processos concorrentes;

pointer - apontador para posições da memória;

offset - endereço relativo ao início de uma área.

3.1.2 Tipos estruturados

(a) "array" - como no ALGOL 60, especificado na forma:

$$(\ell_1:u_1,\ \ell_2:u_2,\ \ldots,\ell_n:\ u_n)\ t$$

onde t é um tipo simples ou a descrição de um tipo estruturado.

(b) Estrutura - com vários níveis de encaixamento e componentes de tipos diferentes, organizadas em forma de árvore. As componentes podem ser "array's, bem como os elementos de um "array" podem ser estruturas.

3.1.3 Definição de novos tipos - não há.

3.2 - Declarações

3.2.1 Formas de declarações

declare d_1, d_2,..., d_n;

onde d_1, ..., d_n são das formas:

$n(id_1,...,id_m)\ at_1,...,at_k$

ou n id at_1, at_2,..., at_k

sendo id, id_1, ..., id_m - identificadores,

at_1, ..., at_k - atributos associados aos identificadores podendo especificar: tipo, alinhamento, tamanho do campo, alocação, valores iniciais ou redefinição,

n - constante inteira positiva, utilizada em declaração de estrutura para indicar o nível de encaixamento da componente, e omitida nos demais casos.

Os atributos são especificados da seguinte forma:

(a) Tipos de dados - ver seção 3.1

(b) Alinhamento em fronteira de palavra: aligned ou unaligned

(c) Tamanho do campo

. dado aritmético - especificado na forma (p,q) onde p é o número total de

bits ou dígitos) e q é o número de bits ou dígitos na parte fracionária.

. sequências - especificado nas formas:
(d) onde d é o número total de bits ou caracteres na sequência;
varying quando a sequência terá tamanho variável durante a execução.

(d) alocação - ver seção 3.3.1

based, static, automatic, controlled ou external.

(e) inicialização - especificada nas formas:

initial (lista de valores iniciais)
ou
initial call $p_1,\ldots,p_n$

onde $p_1,\ldots, p_n$ são procedimentos que serão chamados na ativação do bloco em que ocorre a declaração, para inicialização de variáveis.

(f) redefinição - especificada na forma:

defined id pos

onde id é um identificador já declarado,
pos é das formas:
(lista de subscritos) - para especificar uma localização dentro do "array" id (se id foi declarado com este atributo).

position (p) - onde p é uma constante inteira que indica um deslocamento em bits ou caracteres em relação ao início da área de nome id.

3.2.2 Localização das declarações - entre os comandos de um bloco ou de um procedimento, e antes da utilização dos

nomes declarados.

3.2.3 Redefinição - ver seção 3.2.1.

3.2.4 Domínio da declaração - bloco ou procedimento em que ocorre a declaração.

3.2.5 Pré-declarações

Um identificador que não foi declarado terá seus atributos deduzidos pelo contexto em que foi utilizado, tal como file, label , etc. Se este contexto indicar que o identificador é uma variável aritmética, então se começar com uma das letras I,J,K,L,M ou N será do tipo binary fixed, caso contrário será considerado binary float.

Os identificadores sysin e sysprint são pré-declarados com o tipo file .

3.3 - Alocação

3.3.1 Tipos de alocação

(a) Estática - atributos static ou external;

(b) Dinâmica - atributo automatic, associado também a variáveis sem especificação do tipo de alocação.

(c) Controlada - atributos controlled ou based (p) sendo p um ponteiro. A área é alocada e liberada através dos comandos allocate e free (ver seção 3.6.7).

3.3.2 Meio: todas as variáveis são alocadas em áreas de memória. Não existe acesso aos registradores da máquina.

3.3.3 Validade dos apontadores - não é feita verificação da validade dos apontadores.

3.4 - Constantes e variáveis

(a) Constantes

Numéricas: binária ponto fixo, binária ponto flutuante, decimal ponto fixo, decimal ponto flutuante, complexa e esterlina (com especificação de libras, "Shillings" e "pounds").

Sequências: de bits e de caracteres, representadas entre apóstrofos.

Rótulo : representada por um identificador (ver seção 3.6.1).

(b) Variáveis

Uma variável de um tipo t, sento t um dos tipos simples ou estruturado da seção 1.1, pode ser representada das formas:

id_1 - id_1 - identificador de tipo t;

$v_1(e_1,\ldots,e_n)$ - v_1 - variável de tipo "array" cujos elementos são de tipo t;

$e_1,\ldots,e_n$ - expressões;

$v_2 \cdot id_2$ - v_2 - variável de tipo estrutura;

id_2 - identificador declarado com tipo t, como componente da estrutura v_2;

$p_2 \rightarrow id_3$ - p - variável de tipo pointer;

id_3 - identificador de tipo t, alocado em uma área apontada por p.

Algumas restrições são colocadas sobre as formas acima, tal como no caso da variável p, que não pode ser subscritada, nem declarada como based.

3.5 - Expressões

3.5.1 Formas de expressões

3.5.1.1 Expressões simples

(a) Operandos

. Variáveis - de tipos simples ou estruturados;

. Constantes- com exclusão de rótulos;

. Chamadas de procedimentos, nas formas:

$$p(a_1,a_2,\ldots,a_n)$$

ou

p

sendo p - identificador do procedimento

$a_1,\ldots,a_n$ - lista de parâmetros atuais, sendo cada um dos quais uma expressão.

(b) Operadores

. aritméticos: +, - (unário e binário), *, /, **

. relacionais: <, ¬<, <=, =, ¬=, >=, >, ¬>

. booleanos (entre sequências de bits): ¬(not), &(and), |(or)

. sequências de bits e caracteres: || (concatenação)

Precedência:

1ª. ** , -(unário), (not)

2ª. * , /

3ª. + , -

4ª. ||

5ª. <, ¬<, <=, =, ¬=, >=, >, ¬>

6ª. &

3.5.1.2 Expressões condicionais - não existem na linguagem.

3.5.2 Mistura de tipos e conversão

(a) São feitas poucas restrições quanto à mistura de tipos de operandos em expressões.

(b) Em caso de mistura de tipos são efetuadas conversões, segundo um conjunto de regras.

3.6 - Comandos

3.6.1 Rótulos de comandos:

(a) Definição

(i) Rótulo variável

. É definido através de uma declaração de identificador com o tipo label podendo ser simples ou "array".

. Pode ser inicializado com um rótulo constante.

(ii) Rótulo constante - $r_1:r_2:\ldots:r_n:S$

onde S é um comando.

$r_1,\ldots,r_n$ são os rótulos constantes, das formas id ou id(c)

sendo id - identificador;
c - constante decimal inteira.

No caso em que o rótulo constante é id(c), então id deverá ter sido declarado como "array" de tipo label, ocorrendo automaticamente a inicialização do elemento de índice c do rótulo variável com o valor do rótulo constante id(c).

(b) Utilização -

(i) go to r; onde r é um rótulo constante ou va

(ii) comando de atribuição:

$r_1 = r_2$;

onde r_1 é um rótulo variável;

r_2 é um rótulo constante ou variável.

(iii) Como parâmetro formal ou atual em procedimentos.

(c) Domínio - bloco em que ocorreu a definição do rótulo.

3.6.2 Bloco e comando composto

(a) Bloco - como no ALGOL-60

begin c_1 c_2 ... c_n end

onde $c_1,\ldots,c_n$ são declarações ou comandos; cada declaração é válida apenas nos comandos seguintes, até o fim do bloco.

(b) Comando composto - do; c_1 c_2 ... c_n end;

onde $c_1,\ldots,c_n$ são comandos.

3.6.3 Atribuição - múltipla, da forma

$var_1, var_2,\ldots,var_n = e$;

onde $var_1,\ldots,var_n$ são variáveis simples, "array's", estruturas ou pseudo-variáveis (ver seção 3.7.8).

e - expressão de tipo simples, "array" ou estrutura .

3.6.4 Estruturas de controle

3.6.4.1 Desvio incondicional ou de escape

(a) go to r;

onde r é um rótulo, constante ou variável, de

(b) return;

ou

return (expressão);

. As duas formas são utilizadas para retorno de procedimento.

. A segunda forma retorna o valor da expressão como resultado da execução do procedimento (ver seção 3.7).

(c) stop;

Causa o término da execução do programa.

3.6.4.2 Comandos seletivos

if e then c_1 else c_2;

ou

if e then c_1;

onde e é uma expressão que será convertida, se necessário, para uma sequência de bits;

c_1 e c_2 - comandos.

c_1 é executado se ao menos um bit de e for '1'.

3.6.4.3 Comandos repetitivos

(a) do while (e); $c_1 c_2 \ldots c_n$ end;

e é uma expressão cujo valor se necessário, é convertido para uma sequência de bits;

$c_1, \ldots, c_n$ são comandos.

Os comandos $c_1, \ldots, c_n$ são repetidos enquanto houver algum bit de e igual a '1'.

(b) do var = esp_1,esp_2,...,esp_m; c_1 c_2...c_n end;

onde var é uma variável simples ou pseudo - variável;

esp_1,...,esp_m são especificações das formas:

e_1 to e_2 by e_3 while (e_4)

ou

e_1 by e_3 to e_2 while (e_4)

sendo que e_1,e_2,e_3 e e_4 são expressões, e cada uma das partes to e_2, by e_3 e while (e_4) pode ser omitida ;

c_1,c_2,...,c_n são comandos.

3.6.5 Chamada de procedimento

call p(a_1,...,a_n);

ou

call p;

onde p - identificador do procedimento;

a_1,...,a_n - lista de parâmetros atuais, onde cada um dos quais é uma expressão.

3.6 Comando nulo - ;

3.7 Outros

(a) Alocação:

(i) allocate id_1, id_2,...,id_n;

onde id_1,...,id_n são identificadores declarados como controlled.

A execução deste comando causa a alocação de área em estrutura de pilha.

(ii) allocate a_1,...,a_n;

onde $a_1,\ldots,a_n$ são das formas:

id set (p) in (área)

id set (p)

id in (área)

ou id

sendo id - identificador declarado como based (p);

p - variável de tipo pointer;

área - variável de tipo area que especifica a área da memória em que será feita a alocação.

O endereço da área alocada será colocado na variável p.

(iii) free $id_1,id_2,\ldots,id_n$;

onde $id_1,\ldots,id_n$ - identificadores declarados como controlled.

Este comando libera a área alocada pela última execução de um comando da forma (i) para os identificadores $id_1,\ldots,id_n$.

(iv) free $a_1,a_2,\ldots,a_n$;

onde $a_1,\ldots,a_n$ são das formas:

p → id in (área)

p → id

id in (área)

id

sendo id, p e área como no item (ii) desta seção.

Este comando libera a área alocada através de um comando da forma definida no item (iii) desta seção.

(b) Comandos de entrada e saída - open, close, delete, display, format, get, locate, put, read, rewrite, unlock, write (ver seção 3.9).

(c) Comandos para processamento concorrente - delay, wait, on, signal (ver seção 3.11).

(d) Comandos para pré-processamento - ver seção 3.14.

(e) Comando entry - ver 3.7.1

3.7 Declaração de procedimento

3.7.1 Forma da declaração

$p_1:p_2:\ldots p_m$:procedure ($id_1,\ldots,id_k$) opc rec ret otim;
ou $c_1\ c_2\ \ldots\ c_n$ end;

$p_1:p_2:\ldots p_m$: procedure opc rec ret otim;
$c_1\ c_2\ \ldots\ c_n$ end;

onde $p_1,\ldots,p_m$ - identificadores do procedimento;

$id_1,\ldots,id_k$ - identificadores dos parâmetros formais;

opc - na forma options (lista de opções);

rec - recursive, quando o procedimento será chamado recursivamente;

ret - returns (atributos), especifica os atributos do valor que será retornado pelo procedimento quando este for utilizado como função;

otim-order ou reorder, para controlar a reordenação dos blocos no caso de otimizações feitas pelo compilador.

$c_1,\ldots,c_n$ - comandos ou declarações que constituem o corpo do procedimento, e incluem comandos return ou return (e).

Qualquer uma das partes opc, rec, ret e otim pode ser omitida.

No corpo do procedimento podemos ter o comando entry para definição de um ponto de entrada no procedimento. As formas deste comando são:

$p_1:p_2:\ldots\ p_m$: entry $(id_1,\ldots,id_n)$ ret;

ou

$p_1:p_2:\ldots\ p_m$: entry ret;

onde $p_1,\ldots,p_m$, $id_1,\ldots,id_n$ e ret são definidos da mesma forma que na declaração de procedimentos.

3.7.2 Parâmetros

(a) Tipos de parâmetros - nomes de procedimentos e variáveis de qualquer um dos tipos mencionados na seção 3.1. A declaração destes parâmetros deve ser feita no corpo do procedimento.

(b) Formas de passagem de parâmetros - por referência.

(c) Associação entre os parâmetros formais e atuais e verificação da correspondência.
Um parâmetro atual deve ser do mesmo tipo do parâmetro formal correspondente. Não é feita verificação da correspondência, exceto quando o nome do procedimento chamado for declarado com o atributo entry, na forma:

entry $(at_1,\ldots,at_n)$

onde $at_1,\ldots,at_n$ são listas de atributos dos parâmetros formais. Esta declaração deverá aparecer no bloco ou no corpo do procedimento que contém a chamada.

3.7.3 Retorno de valor em funções

Quando o procedimento for utilizado como função, deve

rá conter ao menos um comando return (ver seção 3.6.4. 1) para especificação do valor que será retornado.

3.7.4 Dados acessíveis a procedimentos - dados locais, globais e parâmetros.

3.7.5 Recursão - é permitida.

3.7.6 Procedimentos externos

Um procedimento pode ser compilado separadamente. Neste caso terá acesso aos dados do programa principal passados como parâmetro ou declarados no programa principal e no procedimento externo com o atributo external.

3.7.7 Procedimentos pré-declarados

(a) Aritméticos - abs, add, binary, ceil, complex, conjg, decimal, divide, fixed, float, floor, imag, max, min, mod, multiply, precision, real, round, sign, trunc.

(b) Matemáticos - atan, atand, atanh, cos, cosd, cosh, erf, erfc, exp, log, log10, log2, sin, sind, sinh, sqrt, tan, tand, tanh.

(c) Manipulação de sequências de caracteres - bit, bool, char, high, index, length, low, repeat, string, substr, translate, unspec, verify.

(d) Manipulação de "array" - all, any, dim, hbound, lbound, poly, prod, sum.

(e) Manipulação de áreas de tipo based - addr, empty, null, nullo.

(f) Verificação de interrupções ocorridas: datafield, onchar, oncode, oncount, onfile, onkey, onloc, onsource.

(g) Verificação do estado de um processo assíncrono - completion, priority, status.

(i) Pseudo-variables - são funções que podem aparecer como lado esquerdo de uma atribuição - completion, complex, imag, onchar, onsouree , priority, real, status, string, substr, unspec. Um exemplo:

substr (título,2,4)= 'ABCD'

Neste comando, os caracteres "ABCD" são inseridos na variável de tipo character título, nas posições 2 até 5.

(j) Outras - allocation, count, date, lineno, time.

3.8 - Programa

Um programa é definido como um procedimento que tem a opção main em sua lista de opções. Este programa poderá utilizar procedimentos externos e, para isto, os módulos compilados deverão ser reunidos em um único módulo (no sistema operacional O.S. - IBM/360 - esta reunião é feita pelo LINKAGE EDITOR).

3.9 - Recursos de entrada e saída

A transmissão de dados pode ser feita de dois modos:

. stream - um caráter de cada vez, com conversão para a forma interna.

. record - um registro lógico de cada vez, sem conversão.

(a) Arquivos

. Declaração - como na seção 3.2.1, com os atributos: file, backwards, buffered, unbuffered, direct, environment, exclusive, input, output, update, keyed, print, record , sequential, stream, transient. No atri

buto environment podem ser especificadas várias informações sobre as características do arquivo.

. Comandos para abrir e fechar arquivos:

open file(id_1)op_1, file(id_2)op_2,...,file(id_n)op_n ;

close file (id_1), file(id_2),..., file(id_n);

onde

id_1,...,id_n - identificadores de arquivos;

op_1,...,op_n - listas de atributos de arquivos, e que podem ser omitidas.

(b) Entrada e saída da forma stream - através dos comandos:

get file(id_1) copy skip(e_1) especificações;

put file(id_1) page skip(e_1) line(e_2) especificações;

get string (id_2) especificações;

put string (id_2) especificações;

onde id_1 - identificador de arquivo;

e_1,e_2 - expressões;

id_2 - identificador de tipo character;

especificações - de uma das formas:

list(d_1,...,d_n) - lista de entrada ou saída em formato livre;

data(d_1,...,d_n) - lista de entrada ou saída em que os dados são precedidos do nome da variável (como no namelist do FORTRAN);

edit(d_1,...,d_n)(f_1,...,f_m) - lista de entrada e saída com formatos.

sendo que $d_1,\ldots,d_n$ - lista de dados que serão transferidos (expressões, veriáveis ou grupos de repetições do).

$f_1,\ldots,f_m$ - lista de formatos de adição (F, picture, A, B, E , etc.).

As partes file(id_1), copy, skip(e_1), line(e_2), page e especificações podem aparecer em qualquer ordem no comando e qualquer uma delas pode ser omitida.

Nas formas get string e put string a transferência não é feita sobre um arquivo e sim da memória para a memória. No comando get a transferência é feita da sequência de caracteres id_2 para as variáveis da parte de especificações e no put as expressões da parte de especificações serão inseridas em id_2, como se esta sequência fosse um registro de um arquivo.

(c) Entrada e saída de forma record - através dos comandos:

read file(id) into(var_1) chave event(var_2);

read file(id) set (var_3) chave;

read file(id) ignore (e);

rewrite file(id) from(var_4) key(e) event(var_2);

write file (id) from(var_4) keyfrom (e) event(var_2);

locate var_1 file (id) set(var_3) keyfrom (e);

delete file (id) key (e) event(var_2);

unlock file (id) key (e);

onde id - nome do arquivo;

var_1 - variável não subscritada e não qualificada na qual será colocado o registro lido;

e - expressão;

var_2 - variável de tipo event, para entrada e saída assíncrona.

var_3 - variável de tipo pointer que irá conter o endereço de um "buffer" no qual o registro foi lido.

var_4 - variável não subscritada e não qualificada cujo conteúdo será impresso.

chave - especificação de chaves para manipulação dos registros em arquivos de acesso direto, nas formas:

key (e)

ou

keyto (var)

sendo var uma variável de tipo character.

As pontes event(var_2), key(e), keyfrom(e) e keyto(var) podem ser omitidas.

O comando com a opção ignore(e) permite que e registros sejam ignorados tornando acessível o (e+1)-ésimo registro a partir do atual.

O comando unlock torna o arquivo especificado acessível a outro processo (task).

(d) Mensagem ao operador

display (e) reply (var_1) event (var_2);

onde e - expressão que será exibida ao operador;

var_1 - variável de tipo character que irá receber a resposta do operador;

var_2 - variável de tipo event para processamento assíncrono do comando.

Em todos comandos de entrada e saída, quanto é especificada a opção event(v) o programa continuará a execução enquanto a operação de entrada ou saída es

tiver sendo efetuada. Quando esta operação terminar, o evento associado à variável v será considerado como ocorrido. O programa deverá verificar esta ocorrência através do comando wait (ver seção 3.11).

3.10 - Recursos para manipulação de bits e caracteres.

Os recursos disponíveis para manipulação de caracteres - são:

. Declaração de variáveis de tipo character e bit,

. Funções de manipulação de caracteres e bits - ver seção 3.7.8 - entre as quais tem-se:

char - converte um valor para uma sequência de caracteres.

substr - extrai uma subsequência de uma sequência de caracteres (ou bits) ou, se usada como pseudo-variável, efetua inserção de uma subsequência em uma sequência de caracteres.

translate - efetua uma tradução de um caráter para outro, de acordo com uma tabela.

index - busca uma subsequência de caracteres em uma sequência de caracteres, retornado a posição em que a subsequência se encontra.

3.11 - Processamento concorrente

3.11.1 Processos concorrentes

Um procedimento pode ser ativado como um processo concorrente se em sua declaração foi especificada a opção task em sua lista de opções.

3.11.2 Criação de processos concorrentes - através do comando call nas formas:

call $p(a_1,a_2,\ldots,a_n)$ task(id) event(var) priority (e)

onde

p - identificador do procedimento;

$a_1,\ldots,a_n$ - lista de parâmetros atuais que podem ser expressões, nomes de procedimentos ou identificadores de tipo simples ou estruturado;

id - identificador que indica o nome do processo criado;

var - variável declarada implícitamente como de tipo event;

e - expressão que indica a prioridade do processo criado, em relação ao processo criador.

Cada uma das partes $(a_1,\ldots,a_n)$, task(id_1), event(id_2) e priority(e) pode ser omitida. Ao menos uma das partes task, event e priority deve estar presente para indicar processamento concorrente.

3.11.3 Exclusão mútua

(a) Arquivos - o acesso a registros de arquivos com os atributos exclusive direct update é restrito a um único processo por vez, exceto se for executado o comando unlock (ver seção 3.9).

(b) Dados - A exclusão mútua em relação ao acesso a dados por processos concorrentes pode ser conseguida através da utilização do comando wait nas formas:

wait $(var_1,var_2,\ldots,var_n)$;

ou

wait $(var_1,var_2,\ldots,var_n)(e)$

onde

$var_1,\ldots,var_n$ - variáveis de tipo event;

e - expressões.

Uma variável de tipo event está associada a um valor '1'B ou '0'B correspondentes aos estado completado ou não completado. Seu valor se torna '1'B quando:

(i) Termina uma operação de entrada ou saída ao qual o evento foi associado;

(ii) Termina a execução de um processo ao qual o evento foi associado, através do comando call;

(iii) É atribuído o valor '1'B através da pseudo- variável completion na forma:

completion (var) = '1'B

onde

var é a variável de tipo event.

O comando wait coloca o processo que o executou, em espera até que os eventos $var_1,\ldots,var_n$ sejam completados ou, na segunda forma de comando, até que o número de eventos $var_1,\ldots,var_n$ completados seja igual ao valor de e. Sob o sistema operacional OS, um evento não pode ser esperado por dois ou mais processos.

A exclusão mútua entre dois processos poderia ser conseguida da forma:

processo 1:	processo 2:
completion(e_1)='0'B;	completion(e_2)='1'B;
⋮	⋮
wait (e_1);	wait (e_2);
"região crítica 1"	"região crítica 2"
completion(e_2)='1'B;	completion(e_1)='1'B;

3.11.4 Sincronização e comunicação entre processos concorrentes.

(a) Através do comando wait (ver seção 3.11.3) um processo pode ficar à espera de que um evento v seja completado, isto é, que termine a execução

de outro processo ou de uma entrada ou saída assíncrona associados ao evento v ou então, que seja executada a atribuição completion(v)='1'B.

Por exemplo:

```
p1: procedure;
      .
      .
call p2 event(v);
      .
      .
wait (v);
      .
      .
end p1;
```

O comando seguinte ao wait será executado somente após o término do procedimento concorrente p_2.

(b) Comando delay

delay(e);

onde e é uma expressão.

O processo que executou o comando será colocado em espera e será ativado depois de decorridos e milisegundos.

3.11.5 Processamento de interrupções

(a) Condições que causam a interrupção da sequência de execução :

(i) Condições de computação - conversion, fixedoverflow, overflow, size, underflow , zerodivide.

(ii) Condições de entrada e saída - endfile(arq), endpage(arq), key(arq), name(arq), pending(arq), record(arq), transmit(arq) , undefinedfile(arq)

onde arq é o identificador do arquivo.

(iii) Condições para depuração de programas -

check, subscriptrange, stringrange.

(iv) Outras - area, error, finish

(v) Condições definidas pelo programador - condition (id) onde id é o identificador da condição.

Uma condição pode estar autorizada ("enabled") ou desautorizada ("disabled"). Se a condição estiver desautorizada e ocorrer, não resultará em uma interrupção.

(b) Comandos.

on cond snap c;

on cond snap system;

revert cond;

signal cond;

onde cond - uma das condições do item (a) desta seção.

c - comando que será executado quando ocorrer a interrupção;

system - quando ocorrer a interrupção será executada a ação padrão do sistema;

snap - pode ser omitido e se estiver presente causará a impressão de um "trace" dos procedimentos ativos no instante da interrupção.

O comando on especifica a ação que será executada em caso de interrupção, sendo válido até o próximo comando on sobre a mesma condição ou até o fim do bloco em que se encontra.

O comando revert anula o efeito dos comandos on sobre a condição especificada, que tenham sido executados no bloco em que o revert se encontra.

O comando signal simula a ocorrência da condição, forçando a interrupção se esta estiver autorizada.

(c) Autorização e desautorização de condições.

- Autorização: cond: proc
  cond: bl

- Desautorização: no cond: proc
  no cond: bl

onde cond é uma das condições do item(a) desta seção;

proc declaração de procedimento;

bl bloco

As formas acima tornam a condição autorizada/desautorizada no interior do procedimento ou bloco.

3.12 - Acesso a características de máquina - nenhum.

3.13 - Comunicação com outras linguagens - é possível, desde que sejam respeitadas várias convenções na transmissão de parâmetros.

3.14 - Recursos em tempo de compilação

(a) Declarações de identificadores

% declare id_1 at_1, id_2 at_2,...,id_n at_n;

onde id_1,...,id_n - identificadores

at_1,...,at_n - atributos de variáveis ou de nome de procedimento.

.atributo de variável: fixed ou character.

.atributo de nome de procedimento: entry (t_1,...,t_n) returns (t)

onde $t, t_1, \ldots, t_n$ são atributos de variáveis.

(b) Declaração de procedimentos

% p_1: p_2: ... p_n: procedure($id_1, \ldots, id_m$) returns(t);

c_1 $c_2 \ldots c_k$ % end;

onde $p_1, \ldots, p_n$ - identificadores do procedimento;

$id_1, \ldots, id_m$ - identificadores dos parâmetros;

t - tipo do valor retornado: fixed ou character;

$c_1 c_2 \ldots c_k$ - comandos ou declarações de pré-processamento sem o sinal "%", sendo que, os únicos comandos permitidos são atribuição, do, go to, if, nulo e return.

O procedimento é chamado como função, devendo aparecer entre os seus comandos, ao menos um comando return. O nome do procedimento deve ser declarado como entry.

(c) Comandos

Nos comandos a seguir $r_1, \ldots, r_m$ são rótulos de comandos de pré-processamento.

(i) atribuição

% r_1:...r_m: var = expressão;

onde var - é uma variável de pré-processamento;

expressão - envolve constantes inteiras, sequências de bits e de caracteres, de pré-processamento e a função substr.

(ii) activate

% r_1:...r_m: activate id_1,... id_n;

. torna ativos os identificadores $id_1, \ldots, id_n$.

(iii) deactivate

% r_1:...r_m: deactivate id_1,...,id_n;

. torna inativos os identificadores id_1, ...,id_n, impedindo a sua substituição no texto.

(iv) go to

% r_1:...;r_m go to r;

sendo r - rótulo de um comando de pré-processamento.

(v) do

% r_1:... r_m: do c_1... c_n % end;

% r_1:... r_m: do i = e_1 to e_2 by e_3;
c_1 c_2 ... c_n % end

% r_1:... r_m: do i = e_1 by e_3 to e_2;
c_1 c_2... c_n % end

c_1...c_n são comandos de pré-processamento ou texto do programa;

e_1...e_3 são expressões;

i - variável de pré-processamento.

Cada uma das partes to e_2 e by e_3 pode ser omitida .

(vi) if

% r_1:... r_m: if e then c_1

% r_1:... r_m: if e then c_1 else c_2

onde e - expressão de pré-processamento;
c_1 e c_2 - comandos de pré-processamento.

(vii) include

% r_1:... r_m: include mb_1,mb_2,...,mb_n end;

mb_1,...,mb_n - são nomes de membros de arquivos em que se encontram textos a serem incluídos no programa e sujeitos a pré-processamento.

(viii) nulo

% r_1:... r_m: ;

(ix) return

r_1:... r_n: return (e)

onde e - expressão de pré-processamento.

Este comando pode aparecer apenas no interior de procedimentos.

3.15 - Recursos para depuração de programas

Comando on (ver seção 3.11.5) com as condições check, subscriptrange, stringrange.

(a) Condição check (n_1, n_2,..., n_m)

A ação especificada no comando on com esta condição será executado quando:

(i) For atingido um comando cujo rótulo está na lista n_1,...,n_m;

(ii) Uma variável cujo nome está na lista n_1,...,n_m, tiver seu valor alterado.

(iii) For chamado um procedimento cujo nome está na lista n_1,...,n_m.

(b) Condição stringrange

A ação especificada no comando on com esta condição será executada se em uma chamada da função substr os

parâmetros forem incorretos.

(c) Condição subscriptrange

A ação especificada no comando on será executada quando os subscritos de um "array" estiverem fora dos limites da declaração.

Estas condições são desautorizadas, isto é, a interrupção ocorrerá somente no bloco ou procedimento que tiver o prefixo cond: .

4. XPL

Esta linguagem foi definida por McKeeman, Horning e Wortman [31], tendo por base o PL/I. Constitui uma parte da "Translator Writing System" - TWS - sendo destinada à programação de compiladores. O compilador da linguagem SPL, denominado XCOM, produz programa em código objeto para o computador IBM/360. A conexão entre este programa objeto e o sistema operacional da máquina é feita pelo submonitor XPLSM.

Entre os compiladores escritos em XPL, podemos citar o próprio compilador XCOM [31] e um compilador PASCAL.

4.1 - Tipos

4.1.1 Tipos simples

fixed - número inteiro com representação interna binária, em palavra de 32 bits;

character - sequência de caracteres EBCDIC de comprimento variável durante a execução;

bit (n) - sequência de bits de comprimento n (n inteiro);

label - rótulo de comando.

4.1.2 Tipos estruturados

array - de uma dimensão, limite inferior zero e limite superior constante.

4.1.3 Definição de novos tipos - não há.

4.2 - Declarações

4.2.1 Formas de declaração

declare $d_1, d_2, \ldots, d_n$;

onde $d_1, \ldots, d_n$ são declarações de variáveis ou de macros.

(a) Variáveis - declaradas de uma das formas:

. Variável simples $\quad \ell$ t inic

. Variável do tipo "array" $\quad \ell$(c) t inic

onde ℓ - lista de identificadores de uma das formas:

id

ou $(id_1, \ldots, id_n)$

sendo id, $id_1, \ldots, id_n$ identificadores

c - constante inteira que indica o limite superior do "array";

t - tipo simples;

inic - especificações de inicialização, que pode ser omitida, sendo da forma:

initial $(c_1,c_2,\ldots,c_n)$

onde $c_1,\ldots,c_n$ são constantes.

(b) Macros

id - literally c

sendo id - identificador da macro;
c - sequência de caracteres.

4.2.2 Localização da declaração - entre os comandos do procedimento.

4.2.3 Inicialização - permitida, de acordo com o ítem (a) da seção 4.2.1.

4.2.4 Redefinição - não há.

4.2.5 Domínio da declaração - programa ou procedimento em que foi declarado.

4.2.6 Pré-declarações- as seguintes variáveis são consideradas pré-declaradas:

corebyte - "array" de tipo byte (bit (8)) correspondente à memória. O índice de um elemento é o endereço absoluto de um "byte" da memória.

coreword - "array" de palavras (bit (32)) correspondente à memória. O índice de um elemento é o endereço de uma palavra da memória (endereço em "byte's" dividido por 4).

date_of_generation - variável inteira inicializada com a data em que o programa foi compilado.

descriptor - "array" do tipo fixed de descritores de sequências de bits ou de caracteres (ver seção 4.4, ítem (a)).

ndescript - variável inteira que contém o limite superior do "array" descriptor.

freebase, freelimit e freepoint - variáveis inteiras que contêm apontadores para a área de sequências de bits ou de caracteres (- ver seção 4.4, ítem (a)).

time_of_generation - variável inteira cujo valor é a hora em que o programa foi compilado.

4.3 - Alocação

4.3.1 Tipos de alocação - A alocação de todas as variáveis é estática, inclusive de variáveis declaradas em procedimentos.

4.3.2 Meio - todas variáveis são alocadas em memória.

4.3.3 Validade de apontadores - não se aplica.

4.4 - Constantes e variáveis

(a) Constantes

(i) Numéricas - números inteiros na notação decimal, representada na forma indicada no ítem (ii) desta seção, ocupando internamente um byte, dois bytes ou uma palavra, conforme o seu comprimento.

(ii) Sequência de bits - de comprimento maior que 32

"$g_1 g_2 \cdots g_n$"

onde $g_1,\ldots,g_n$ - são das formas:

(1) b
(2) q
(3) o
(4) h

sendo b - número binário;
q - número quaternário (base 4);
o - número octal;
h - número hexadecimal.

g_1 pode ser das formas acima ou apenas um número hexadecimal não precedido de (4).

(iii) Sequência de caracteres EBCDIC

$'c_1c_2\ldots c_n'$ ou

ou '' (sequência vazia)

onde $c_1,\ldots,c_n$ - são caracteres EBCDIC ou '' (dois apóstrofos) para representação de um apóstrofo.

Sequências de bits de comprimento maior que 32 e sequências de caracteres são mantidas em uma área de memória livre que é automaticamente compactada quando necessário. A manipulação de uma sequência é feita através de uma palavra de 32 bits denominada descritor. Este descritor contém o tamanho da sequência em bytes e o seu endereço dentro da área livre.

(b) Variáveis - representadas nas formas:

id_1

ou

id_2 (e)

onde id_1 - identificador de variável declarada com tipo simples;

id_2 - identificador de variável declarada com tipo "array";

e - expressão.

4.5 - Expressões

4.5.1 Formas

4.5.1.1 Expressões simples

(a) Operandos - constantes, variáveis e chamadas de funções. Formas de chamada de função:

id

ou id $(e_1,\ldots,e_n)$

onde id - identificador de procedimento;

$e_1,\ldots,e_n$ - expressões.

Os operandos podem ser numéricos, descritores de sequências e valores condicionais, sendo que estes resultam de operações relacionais (< =, > =, etc.).

(b) Operadores - os mesmos do PL/I (ver seção 3.5.2.1) com a inclusão do operador mod - (resto de divisão).

4.5.1.2 Expressões condicionais - não há.

4.5.2 Mistura de tipos e conversão

São permitidas misturas de operandos numéricos e condicionais, havendo conversão automática. Descritores de sequências de caracteres ou bits não podem ser misturados com valores numéricos ou condicionais. Expressões que devem resultar em um valor condicional ,

como as utilizadas nos comandos if e do while, se tiverem valor numérico, este valor será convertido para true ou false, conforme seu bit menos significativo - seja 1 ou 0 respectivamente.

4.6 Comandos

4.6.1 Rótulos de comandos

(a) Definição

$$r_1 : r_2 : \ldots r_n : c$$

onde $r_1,\ldots,r_n$ - são identificadores;

c - comando

Os rótulos podem ser declarados da forma introduzida na seção 4.2.1.

(b) Utilização - comando go to

4.6.2 Bloco e comando composto

(a) Bloco - não existe.

(b) Comando composto:

do; c_1 c_2 ... c_n end;

onde $c_1,\ldots,c_n$ são comandos ou declarações.

4.6.3 Atribuição - nas formas:

v = e;

$v_1, v_2,\ldots, v_n$ = e;

e - é uma expressão.

A atribuição de uma sequência de caracteres ou bits a uma variável declarada como sequência de caracteres ou bits é feita através da atribuição do

descritor.

4.6.4 Estruturas de controle

4.6.4.1 Desvio incondicional e escape.

(a) go to r;

onde r é um identificador de rótulo.

(b) comandos para retorno de procedimentos

return ;

ou

return (e);

onde e é uma expressão cujo valor será retornado como resultado da execução de um procedimento.

4.6.4.2 Comandos seletivos

(a) if e then c_1 else c_2;

if e then c_1;

sendo e - expressão condicional;
c_1, c_2 - comandos.

(b) do case e; c_0 c_1 ... c_n end;

sendo e - expressão;
$c_0 c_1 \ldots c_n$ - comandos.

Se $0 \leqslant e \leqslant n$ será executado o comando c_e, caso contrário o resultado será imprevisível.

4.6.4.3 Comandos repetitivos

(a) do while e; c_1 c_2 ... c_n end;

sendo e - expressão condicional;

$c_1,...c_n$ - comandos que serão repetidos enquanto e for true.

(b) do v = e_1 to e_2; $c_1c_2...c_n$ end;

do v = e_1 to e_2 by e_3; $c_1c_2...c_n$ end;

sendo v - variável;

e_1,e_2,e_3 - expressões calculadas somente no início da execução do comando, sendo que e_3 deve ser maior que zero;

$c_1,c_2,...,c_n$ - comandos que serão repetidos, sendo que se inicialmente $e_2 < e_1$, os comandos não serão executados.

4.6.5 Chamada de procedimento

call p;

ou call p($e_1,...,e_n$);

onde p - identificador do procedimento;

$e_1,...,e_n$ - expressões.

4.6.6 Comando nulo - ; .

4.6.7 Outros - não existem.

4.7 Declaração de função ou procedimento

4.7.1 Forma da declaração

p: procedure (id_1, $id_2,...,id_n$) t; $c_1c_2...c_m$ end;

ou p: procedure t; $c_1c_2...c_m$ end;

onde p - identificador do procedimento;

$id_1,...id_n$ - identificadores dos parâmetros for

mais;

t - tipo simples do valor retornado quando o procedimento for chamado como função. Pode ser omitido.

$c_1,\ldots,c_m$ - comandos ou declarações que constituem o corpo do procedimento.

4.7.2 Parâmetros

Devem ser declarados no interior do procedimento.

(a) Tipos de parâmetros formais - o tipo de um parâmetro formal pode ser simples ou estruturado.

(b) Forma de passagem de parâmetros - por valor.

(c) Associação dos parâmetros formais e atuais e verificação da correspondência. Não existe verificação da correspondência entre os tipos dos parâmetros formais e atuais.

4.7.3 Retorno de valor em funções

Quando o procedimento é chamado como função, entre os seus comandos deve haver ao menos um return (e), onde e é uma expressão cujo valor será retornado.

4.7.4 Dados acessíveis a procedimentos: locais, globais e valores passados como parâmetros.

4.7.5 Recursão - não há.

4.7.6 Procedimentos e funções externas - não há.

4.7.7 Efeitos colaterais em funções - nenhuma restrição.

4.7.8 Procedimentos pré-declarados

Sendo e - expressão envolvendo sequências de bits ou caracteres;

n, n_1 e n_2 - expressões com valores inteiros;

v - variável

temos as seguintes funções:

(a) Manipulação de sequências de bits ou caracteres

byte (e,n) - seleciona o n-ésimo byte de sequência e.

substr (e,n_1,n_2) - seleciona a subsequência de comprimento n_2 a partir da n_1-ésima posição da sequência e. Pode ser utilizada como pseudo-variável (ver seção 3.7.8).

length (e) - obtém o comprimento da sequência e.

(b) Entrada e saída - file, input e output (ver seção 4.9).

(c) Acesso a instruções de máquina - inline,shl,shr (ver seção 4.12).

(d) Depuração de programas-trace, untrace (ver seção 4.14).

(e) Outras:

addr (v) - obtém o endereço absoluto da variável v.

compatify - compacta a área livre das sequências.

date - obtém a data no instante da execução.

time - obtém a hora no instante da execução.

clock_trap, interrupt_trap, monitor - procedimentos previstos, mas não implementados, para comunicação com o submonitor do compilador.

4.8 Programa - definido na forma:

$c_1 c_2 \ldots c_n$ eof

onde $c_1,\ldots,c_n$ são comandos ou declarações.

4.9 Recursos de entrada e saída

Toda entrada e saída é feita através de procedimentos.

Cada arquivo é identificado por um número inteiro. Assim, seja f uma expressão cujo valor identifica o arquivo.

(a) file (f,r) - pseudo-variável para manipulação de arquivos de acesso direto.

Utilização:

var = file (f,r) - transfere o r-ésimo registro do arquivo f para a variável var.

file (f,r) = var - transfere o conteúdo da variável var para o r-ésimo - registro do arquivo.

A transferência é feita sem conversão.

(b) input (f) - função que efetua leitura - de um registro do arquivo - sequencial f. O registro lido é considerado como uma sequência de caracteres.

Utilização:

var = input (f) - sendo var uma variável declarada como sequência de bits ou caracteres. Seu valor se torna a sequência de caracteres lidos.

(c) output (f) - pseudo-variável que transfere uma sequência de caracteres para o arquivo f.

Utilização:

output (f) = e - sendo e uma expressão resultando em uma sequência de

caracteres que será transferida para o arquivo f.

4.10 Recursos para manipulação de bits e caracteres.

(a) Declaração de variáveis de tipo character e bit.

(b) Funções e pseudo-variáveis - byte, substr e length.

4.11 Processamento concorrente - nenhum recurso.

4.12 Acesso a características de máquina

(a) Acesso à memória e manipulação de endereços -através das variáveis corebyte e coreword e da função addr.

(b) Operações de deslocamento - através das funções:

shl(e_1,e_2) e shr(e_1,e_2) - deslocamento lógico para a esquerda e direita, respectivamente, sendo que a expressão e_1 indica o valor que será deslocado e a expressão e_2 indica o número de posições deslocadas.

(c) Acesso a instruções de máquina - função ou procedimento

inline (e_1,e_2,...,e_n)

sendo e_1 - expressão que indica o código de uma instrução de máquina do IBM/360 [25];

e_2,...,e_n - expressões que irão preencher os demais campos da instrução, indicando registradores, endereços, etc..

4.13 Comunicação com procedimentos em outras linguagens - sem informação; provavelmente possível como nas outras linguagens do IBM/360.

4.14 Recursos em tempo de compilação - definição de macros.

(a) Declaração - ver seção 4.2.1, ítem (b).

(b) Utilização - a ocorrência do identificador da macro no texto do programa, causa a sua substituição pela sequência de caracteres associada ao mesmo, durante a compilação.

4.15 Recursos para depuração de programas - procedimentos :

(a) trace - causa a impressão de um "trace" de cada instrução de máquina executada.

(b) untrace - cessa a impressão do "trace".

5. PASCAL sequencial

Esta linguagem foi definida por Wirth [26, 49], tendo sido utilizada na codificação do próprio compilador. Uma versão desta linguagem foi utilizada por Brinch Hansen [9] para a programação do compilador PASCAL concorrente [10, 11] .

A definição do PASCAL teve como objetivos principais o ensino de programação segundo os conceitos de programação estruturada e também a implementação de compiladores que gerassem código eficiente e confiável, mas computadores convencionais.

5.1 - Tipos

5.1.1 Tipos simples

(a) real

(b) tipos escalares (ou enumeráveis)

integer

boolean - conjunto dos valores true e false;

char

(id_1, id_2,...,id_n) - conjunto ordenado de valores; os identificadores id_1,...,id_n constituem as constantes do tipo;

c_1 ..c_2 - subconjunto de um tipo escalar ao qual as constantes c_1 e c_2 pertencem; estas constantes indicam os valores extremos do novo tipo.

(c) id - onde id é um identificador associado a um tipo simples através de uma definição de tipo (ver seção 5.2.1).

(d) ↑id - conjunto de endereços (apontadores); os elementos apontados são do tipo id.

5.1.2 Tipos estruturados

(a) array $[t_1, t_2, \ldots, t_n]$ of t

onde $t_1, t_2, \ldots, t_n$ são tipos escalares e definem o conjunto de valores que os índices podem assumir;

t é o tipo, simples ou estruturado, dos elementos do "array".

Como decorrência desta definição temos:

(a) pode haver um ou mais índices;

(b) o número de elementos no "array" é fixo.

(b) record $f_1; f_2; \ldots; f_n$ end

onde f_i é da forma:

$id_1, id_2 \ldots, id_n$: t

onde $id_1, \ldots, id_n$ são identificadores das componentes do record;

t é do tipo das componentes, simples ou estruturado.

O último campo f_n, além da forma anterior, pode ser:

case s : t of $v_1; v_2; \ldots; v_n$

onde s é identificador de componente e também é utilizado como seletor de uma das variantes $v_1, \ldots, v_n$;

t é o tipo da componente s;

v_i são campos "variantes" da forma:

$c_1^i, c_2^i, \ldots, c_k^i : (f_1^i; f_2^i; \ldots; f_\ell^i)$

c_j^i é uma constante do tipo t;

f_j^i é uma componente sendo que f_ℓ^i também pode ser da forma case.

Se o valor da componente s for c e c aparecer na lista de constantes c_j^i, c_2^i,...c_k^i, para algum i, então o campo f_n é equivalente a:

$$s : t ; f_1^i; f_2^i;...;f_\ell^i$$

(c) set of t - conjunto potência (conjunto de todos subconjuntos) do tipo simples t; cada elemento do tipo set é um subconjunto do tipo t.

(d) file of t - sequência de componentes de tipo t que constitui um arquivo sequencial.
Em cada instante somente um elemento da sequência é acessível.

(e) id - onde id é um identificador associado a um tipo estruturado através de uma definição de tipo (ver seção 5.2.1).

(f) packed t

onde t é um tipo estruturado.

O tipo packed t só se distingue do tipo t quanto à forma de implementação. No primeiro, obtém-se economia de memória, através da compactação dos campos.

5.1.3 Definição de novos tipos - através de uma definição de tipo (ver seção 5.2.1) é possível associar um identificador a um tipo simples ou estruturado. Este identificador poderá ser utilizado como o nome do novo tipo.

5.2 - Declarações

5.2.1 Formas de declaração

(a) Definição de tipo

type $id_1 = t_1; id_2 = t_2;...;id_n = t_n$

onde $id_1, \ldots, id_n$ são identificadores;

$t_1, \ldots, t_n$ são tipos simples ou estruturados.

Esta definição torna o identificador id_i equivalente ao tipo t_i.

(b) Definição de constante

const $id_1 = c_1$; $id_2 = c_2$; ...; $id_n = c_n$;

onde $id_1, \ldots, id_n$ são identificadores;
$c_1, \ldots, c_n$ são constantes.

Esta definição torna o identificador id_i equivalente à constante c_i.

(c) Declaração de variável

var d_1; d_2; ...; d_n;

onde $d_1, \ldots, d_n$ são da forma
$id_1, id_2, \ldots, id_m : t$

sendo $id_1, \ldots, id_m$ - identificadores;
t - tipo simples ou estruturado.

(d) Declaração de rótulo de comando

label $c_1, c_2, \ldots, c_n$;

onde $c_1, \ldots, c_n$ são números inteiros sem sinal.

Declaração de procedimento ou função: ver seção 5.7.1.

5.2.2 Localização das declarações: início de blocos, na seguinte ordem:

1º declaração de rótulos;
2º definição de constantes;
3º definição de tipos;
4º declaração de variáveis;
5º declaração de procedimentos e funções.

5.2.3 Inicialização: não há.

5.2.4 Redefinição: não há.

5.2.5 Domínio da declaração: bloco em que ocorreu a declaração.

5.2.6 Pré-declarações:

. tipos: integer, real, boolean e char são tipos pré-definidos.

. arquivos - input e output são pré-declarados como de tipo file of char.

5.3 - Alocação

5.3.1 Tipos de alocação

(a) Estática - variáveis declaradas no bloco mais externo do programa.

(b) Dinâmica - variáveis declaradas em procedimentos e funções.

(c) Controlada - através dos procedimentos new (p), para alocar uma área e dispose(p), para liberar a área alocada, sendo p um apontador para o início da área e seu tamanho é fornecido pelo tipo especificado na declaração de p.

5.3.2 Meio: memória e registradores

Todos os dados declarados no programa são alocados na memória.

5.3.3 Validade de apontadores: uma área alocada pelo comando new (p) é válida até que seja executado o procedimento dispose (p). Após a execução deste, a validade do apontador p não é verificada.

5.4 - Constantes e variáveis

(a) Constantes

(i) id - onde id é um identificador declarado como constante.

(ii) k, +k ou -k
onde k é um número sem sinal em uma das formas:
inteira - sequência de dígitos decimais.
real - m . n
mE + e, mE - e, mE e
m . nE + e, m . nE - e, m . nE e
sendo m, n, e - constantes inteiras sem sinal.

(iii) sequência de caracteres

' c_1 c_2... c_m '

onde c_1,..., c_m são caracteres alfanuméricos sendo que um apóstrofo dentro de sequência é representado por dois apóstrofos.

(iv) nil - única constante de tipo apontador.

(v) true e false - constantes de tipo Boolean.

(b) Variáveis - uma variável de um tipo t, simples ou estruturado, é representada nas formas:

(i) id - onde id é um identificador declarado com tipo t;

(ii) var_a [exp_1, exp_2,..., exp_n]
onde var_a é uma variável de tipo "array", cujos elementos são de tipo t;

exp_1,..., exp_n são expressões cujos valores são do mesmo tipo dos índices da declaração do "array".

(iii) var_r . id
onde var_r é uma variável de tipo record;
id - identificador declarado como componente do tipo record, sendo de tipo t.

(iv) $var_f\uparrow$
onde var_f é uma variável de tipo file cujos elementos são do tipo t.

O valor da variável $var_f\uparrow$ é um elemento de sequência sendo que o valor da variável é mudado através dos procedimentos de entrada e saída (ver seção 5.8).

(v) $var_p\uparrow$
onde var_p é uma variável de tipo apontador, cujos valores apontados são de tipo t.

5.5 - Expressões

5.5.1 Formas

5.5.1.1 Expressões simples

(a) Operados

- variáveis
- constantes sem sinal
- chamada de função
  $f(a_1, a_2, \ldots, a_n)$
  ou
  f
  onde f - identificador de função:
  $a_1, \ldots, a_n$ - parâmetros atuais que podem ser expressões, variáveis e identificadores de procedimentos ou funções.
- conjuntos - representados por:
  [ ] - conjunto vazio
  $[e_1, \ldots, e_n]$
  onde $e_1, \ldots, e_n$ são das formas:
  exp
  ou $exp_1 \ldots exp_2$
  sendo exp, exp_1, exp_2 - expressões de um mesmo tipo base.

(b) Operadores e precedência dos mesmos

Os níveis de precedência são 4.

1º nível: Operador not - aplicado sobre operando de tipo Boolean.

2º nível: Operadores de multiplicação

* multiplicação - operandos: real ou integer;
resultado: real ou integer;

intersecção - operandos: conjuntos de tipo T;
resultado: conjunto de tipo T.
(onde T é um tipo simples).

/ divisão - operandos: real ou integer;
resultado: real.

div divisão com truncamento -
operandos: integer;
resultado: integer.

mod resto da divisão -
operandos: integer;
resultado: integer.

and "e" lógico - operandos: Boolean;
resultado: Boolean.

3º nível: Operadores de adição

\+ adição - operandos: integer ou real;
resultado: integer ou real.

união - operandos: conjuntos de tipo base T;
resultado: conjunto de tipo base T.

\- subtração - operandos: integer ou real;
resultado: integer ou real.

diferença - operandos: conjuntos de tipo base T;

resultado:conjunto de tipo base T.

troca de sinal- operandos:integer ou real;
resultado:integer ou real.

or "ou" lógico- operandos:Boolean;
resultado:Boolean.

4º nível:Operadores relacionais

=, <>, <,<=,>,>= operandos:tipos escalares;
resultado:Boolean.

in - inclusão - operandos:o 1º de tipo escalar;
o 2º de tipo set;
resultado:Boolean.

Note-se a necessidade do uso de parênteses, como no exemplo: (a=b) ou (c=d) onde a, b, c e d são inteiros ou reais.

5.5.1.2 Expressões condicionais - não existem na linguagem

5.5.3 Mistura de tipos e conversão

Misturas de tipos não são permitidas e, portanto, não existe conversão.

5.6 - Comandos

5.6.1 Rótulo: um único rótulo por comando, representado por um número inteiro sem sinal.

. Declaração - ver seção 5.2.1
. Definição - r:c
onde r - rótulo;
c - comando sem rótulo.
. Utilização - no comando go to (ver seção 5.6.4.1)
. Domínio - bloco em que o rótulo está definido.

5.6.2 Bloco e comando composto

(a) Bloco - não existe como comando, sendo utilizado apenas para definição do corpo de procedimentos , corpo de funções e programa. Sua representação é na forma:

$d_r\ d_c\ d_T\ d_v\ d_{pf}\ c$

onde d_r - declaração de rótulos;
d_c - declaração de constantes;
d_T - declaração de tipos;
d_v - declaração de variáveis;
d_{pf} - declarações de funções e procedimentos;
c - comando composto.

(b) Comando composto

begin c_1; c_2;...; c_n end

onde c_1, c_2,..., c_n são comandos com ou sem rótulos.

5.6.3 Atribuição - formas

var: = exp
fid: = exp
onde var é uma variável;
fid é um identificador de função;
exp é uma expressão

A variável (ou função) deve ser do mesmo tipo da expressão, exceto se:

(a) A variável é do tipo real e a expressão é de tipo integer ou um subconjunto de integer.

(b) O tipo da expressão é um subconjunto do tipo da variável.

5.6.4 Estruturas de controle

5.6.4.1 Desvio incondicional ou escape

go to r

onde r é um rótulo.

5.6.4.2 Comandos seletivos

(a) Comando if

if exp then c_1

if exp then c_1 else c_2

onde exp é uma expressão de tipo Boolean;

c_1 e c_2 são comandos.

(b) Comando case

case exp of p_1; p_2;...; p_n end

onde exp é uma expressão;

p_i é da forma:

k_1^i: k_2^i:...k_m^i: c_i

k_j^i é uma constante do tipo de exp;

c_i é um comando.

Deve existir um i e um j tal que o valor de exp seja igual a k_j^i. Neste caso será executado o comando c_i.

5.6.4.3 Comandos seletivos

(a) while exp do c

onde exp é uma expressão de tipo Boolean;

c é o comando a ser repetido.

(b) repeat c_1; c_2;...; c_n until exp

onde exp é uma expressão de tipo Boolean;

c_1, c_2,..., c_n são comandos a serem repetidos.

(c) for var: = exp_1 to exp_2 do c

ou

for var: = exp_1 downto exp_2 do c

onde var é uma variável de controle;
exp_1 e exp_2 são expressões do mesmo tipo de var;
c é o comando a ser repetido.

Na primeira forma o incremento é 1 e na segunda é -1.

5.6.5 Chamada de procedimento

$p(a_1, a_2, \ldots, a_n)$

ou p

onde p - é o identificador associado ao procedimento ;
$a_1, a_2, \ldots, a_n$ - parâmetros atuais que podem ser expressões, variáveis e identificadores de procedimentos ou funções.

5.6.6 Comando nulo - cadeia vazia de caracteres

5.6.7 Outros:

Comando with na forma

with $var_1, var_2, \ldots, var_n$ do c

onde $var_1, \ldots, var_n$ são variáveis de tipo record;
c é um comando.

Efeito: no interior do comando c, sempre que for feita uma referência a um record da lista $var_1, \ldots, var_n$, podemos utilizar a forma: comp ao invés de var_i.comp, onde comp é uma componente do record var_i.

5.7 - Declaração de função ou procedimento

5.7.1 Forma da declaração

procedure p; bℓ

procedure p (par_1; par_2;...; par_n); bℓ

function f : t; bℓ

function f (par_1; par_2;...; par_n) : t; bℓ

onde p - identificador do procedimento;
f - identificador de função;
t - tipo simples;
bℓ- bloco;
$par_1,...,par_n$ - declarações de parâmetros formais das formas:

(a) parâmetros passados por valor

$id_1, id_2,..., id_m : t$

(b) parâmetros passados por referência

var $id_1, id_2,...,id_m$: t

(c) funções e procedimentos passados como parâmetros:

function $id_1, id_2,..., id_m$: t

procedure $id_1, id_2,..., id_m$

sendo que $id_1,..., id_m$ são identificadores dos parâmetros;

t é o tipo do parâmetro ou, no caso de função, o tipo do valor retornado.

5.7.2 Parâmetros

(a) Tipos de parâmetros formais - constantes e variáveis de tipos simples ou estruturados, funções e procedimentos.

(b) Formas de passagem de parâmetros

. Por valor - quando o parâmetro formal é declarado como no ítem (a) da seção 5.7.1.

. Por referência - quando é utilizada a forma de declaração (b) da seção 5.7.1.

(c) Associação dos parâmetros atuais com os parâmetros formais e verificação da correspondência.

. Expressões - devem ser passadas por valor.

- Variáveis podem ser passadas por valor e por referência.
- Funções e procedimentos passados como parâmetros devem corresponder respectivamente a parâmetros formais declarados como funções e procedimentos.
  Não é feita verificação da correspondência entre os parâmetros.

5.7.3 Retorno de valor em funções - os comandos do corpo da função devem incluir ao menos uma atribuição cujo lado esquerdo do sinal ":=" seja o identificador da função.

5.7.4 Dados acessíveis aos procedimentos e funções: globais, locais e valores passados como parâmetros.

5.7.5 Recursão - permitida

5.7.6 Procedimentos e funções externas - não são permitidas na definição de Wirth [26, 49] tendo sido introduzidas na versão implementada no computador CDC-6000("User Manual" [26]).

5.7.7 Efeitos colaterais em funções - não existem restrições; funções distinguem-se de procedimentos apenas por retornarem um valor.

5.7.8 Procedimentos e funções pré-declaradas:

(a) Funções aritméticas:

função	cálculo efetuado	tipo do parâmetro	tipo do resultado
abs(x)	$\|x\|$	real ou integer	tipo do parâmetro
sqr(x)	x^2	real ou integer	tipo do parâmetro
sin(x)	sen x	real ou integer	real
cos(x)	cos x	real ou integer	real
exp(x)	e^x	real ou integer	real
ℓn(x)	ℓn x	real ou integer	real
sqrt(x)	$\sqrt{x}$	real ou integer	real
arctan(x)	arctg x	real ou integer	real

(b) Funções - predicados

função	efeito	tipo do parâmetro	tipo do resultado
odd(x)	true se x é impar false se x é par	integer	boolean
eof(f)	true se é fim do arquivo f	file	boolean
eoln(f)	true se ocorre fim de registro (linha) no arquivo f	file	boolean

(c) Funções de transferência

função	efeito	tipo do parâmetro	tipo do resultado
trunc(x)	parte inteira do valor de x	real	integer
round(x)	arredonda x para o inteiro mais próximo	real	integer
ord(x)	ordem em que o valor de x aparece no conjunto definido pelo tipo de x	escalar	integer
chr(x)	obtém x - ésimo caracter do conjunto de caracteres	integer	char

(d) Outras funções

função	efeito	tipo do parâmetro	tipo do resultado
succ(x)	obtém o valor sucessor de x no tipo escalar	escalar	é o mesmo do parâmetro
pred(x)	obtém o valor anterior a x no tipo escalar	escalar	é o mesmo do parâmetro

(e) Procedimentos para alocação dinâmica

nome	efeito	tipo(s) do(s) parâmetro(s)
new(p) e new(p,t_1,...t_n)	aloca uma área e coloca o endereço inicial em p. O tamanho da área é indicado pelo tipo introduzido na declaração de p. A segunda forma é utilizada quando o tipo associado a p admite variantes.	apontador
dispose(p) e dispose(p,t_1,...,t_n)	libera a área referenciada através do ponteiro p.	apontador

(f) Procedimentos para manipulação de arquivos -ver seção 5.9.

5.8 - Programa

program id (id_1,...,id_n); bℓ.

onde id é o identificador do programa;

id_1,...,id_n - identificadores externos ao programa mas que devem ser declarados no bloco que constitue o programa;

bℓ - bloco.

5.9 - Recursos para entrada e saída

(a) Arquivos - devem ser declarados como do tipo file. Existem pré-declarados os arquivos input e output como de tipo "text" sendo que "text" tem a seguinte definição:

type text = file of char

Se f é o identificador de um arquivo, f↑ é

uma variável cujo valor é um elemento de f, podendo ser imaginado como um "buffer".

(b) get(f) - torna acessível em f↑ o próximo elemento do arquivo f. Se este elemento não existe, então eof(f) se torna true.

(c) put(f) - Se eof(f) é true, o conteúdo de f↑ é acrescentado ao arquivo f. Após a execução, eof(f) continua true e f↑ fica indefinido.

(d) reset(f)- posiciona f no início, atribuindo à variável f↑ o primeiro elemento da sequência. eof(f) se torna false se o arquivo está vazio.

(e) rewrit (f)- elimina os elementos do arquivo f de forma que outro arquivo possa ser gerado e eof(f) se torna true.

Os procedimentos a seguir manipulam arquivos de tipo text e se as variáveis v, v_1, v_2,..., v_n forem de tipos diferentes do tipo do arquivo f a transferência é acompanhada de conversão automática de tipo.

(f) read(f,v)- será lida uma sequência de caracteres do arquivo f até que a mesma constitua uma constante de acordo com a sintaxe do PASCAL. Este número será convertido para a representação interna e atribuído a v. Se v é de tipo char, então esta chamada do procedimento será equivalente a v:=f↑; get(f).

read(f,v_1,...,v_n) - é equivalente a read(f,v_1); ... ; read(f,v_n)

read(v_1,...,v_n) - é equivalente a read(input,v_1,...,v_n)

Ocorrendo a condição eoln(f)=true correspondente a fim de registro físico a leitura continuará no registro seguinte.

(g) readln(f)- equivale a

while not eoln(f) do get(f)

readln(f,v_1,...,v_n) - equivale a

read(f,v_1,...,v_n); readln(f)

Isto é, após a leitura de v_1,...,v_n, o arquivo é posicionado no início de uma nova "linha" (ou registro).

Nos procedimentos a seguir, p, p_1, p_2,..., p_n são de uma das formas:

e

e:m

e:m:n

onde e - expressão cujo valor será escrito no arquivo f.

m,n - expressões inteiras que constituem especificação do número de caracteres ocupados pelo número e número de dígitos da parte fracionária, respectivamente.Nos casos de omissão de m e n, serão utilizados valores definidos na implementação.

(h) write(f,p)- o valor da expressão especificada em p é convertida para a forma externa segundo a sintaxe do PASCAL e impresso como uma sequência de caracteres no arquivo f.

write(f,p_1,p_2,...,p_n) é equivalente a

write(f,p_1),write(f,p_2),...,write(f,p_n).

write(p_1,p_2,...,p_n) é equivalente a

write(output, p_1, p_2,...,p_n).

(i) writeln(f)- acrescenta um marcador de fim de linha ao arquivo f, isto é, efetua uma mudança de registro.

writeln(f,p_1,p_2,...,p_n) é equivalente a write(f,p_1,p_2,...,p_n) ; writeln(f).

writeln(p_1,...,p_n) é equivalente a writeln(output,p_1,...,p_n).

(j) page(f) - causa um salto para o início de uma nova página quando o arquivo f é uma impressora.

5.10 - Recursos para manipulação de caracteres

Sequências de caracteres são consideradas como sendo do tipo packed array [1..n] of char. O conteúdo de um vetor com esta definição pode ser manipulado através de indexação ou sem indexação como uma sequência de caracteres, sendo que o seu valor poderá ser atribuído a outro vetor de mesmas características.

5.11 - Processamento concorrente - não há

5.12 - Acesso a características de máquina - não há.

5.13 - Comunicação com procedimentos em outras linguagens - não existe na definição de Wirth [49] , sendo que na versão implementada no CDC-6000 ("User Manual" - [26]) é possível comunicação com procedimentos em FORTRAN.

5.14 - Recursos em tempo de compilação - nenhum.

5.15 - Recursos de "debug" - nenhum na definição de Wirth [49], existindo na versão implementada no CDC-6000 [26] uma opção de compilação que gera código para emissão de"dump" após a ocorrência de erros de execução.

6. PASCAL concorrente

A linguagem PASCAL concorrente foi definida por Brinch Hansen [10,11] para programação estruturada de sistemas operacionais, tendo sido baseada no PASCAL de Wirth [26,49].

Foram implementadas no computador PDP 11/45 do California Institute of Tecnology, uma versão do PASCAL [9], que chamaremos PASCAL S e o PASCAL concorrente [10,11] ou PASCAL C. Ambas linguagens foram utilizadas para a programação do sistema operacional SOLO [12].

O PASCAL S, de Brinch Hansen, difere do PASCAL de Wirth nos seguintes aspectos:

(a) Não tem o comando go to.

(b) Na declaração de funções e procedimentos, o corpo pode ser substituído por forward.

(c) Os parâmetros de funções devem ser declarados como constantes.

(d) O tipo do parâmetro pode ser precedido da especificação univ que suprime a verificação de correspondência de tipo dos parâmetros formais e atuais.

(e) Um programa é precedido de um prefixo (ver seção 6.8).

(f) Os símbolos "[" e "{" são substituídos por "(." e "] " e "{" substituídos por ".)".

Apresentamos a seguir uma descrição das características do PASCAL concorrente, acrescentadas à definição do PASCAL S.

6.1 - Tipos

6.1.1 Tipos simples - os que existem no PASCAL S, com exclusão de apontadores e com a inclusão do tipo queue, utilizado para manter processos colocados em espera através do procedimento delay (ver seção 6.11).

6.1.1 Tipos estruturados

(a) array - como no PASCAL S.

(b) record - como no PASCAL S, com exclusão da especificação case para seleção de variantes.

(c) set - como no PASCAL S.

(d) sistemas - definidos como:

process (p_1; p_2;...;p_n);b ou process b

monitor (p_1; p_2;...;p_n);b ou monitor b

class (p_1; p_2;...;p_n);b ou class b

onde p_1;...;p_n - são parâmetros (ver ítem (b), seção 6.7.2) declarados como constantes de tipo simples, set ou monitor, sendo que um tipo class também pode ser parâmetro de outro tipo class;

b - bloco (ver seção 6.8).

6.2 - Declarações

6.2.1 Formas de declaração

(a) constantes e tipos - como no PASCAL S, considerando-se os novos tipos introduzidos na seção 6.1.

(b) Variáveis

var d_1; d_2;...; d_n

ou

var entry d_1; d_2;...; d_n

onde d_1, d_2,... d_n são da forma:

id_1,...id_m : t

sendo id_1,...,id_m identificadores das variáveis;

t - tipo (simples ou estruturado)

A declaração var entry pode aparecer somente dentro de classes e o tipo destas variáveis - não pode ser queue. Variáveis declaradas com tipos monitor, class ou process são denominadas - "componentes de sistemas".

(c) Rotina - procedimentos, funções e programas sequenciais denominam-se "rotinas" (seção 6.7)

Rotinas são declaradas somente dentro de sistemas, não sendo permitidas declarações de rotinas dentro de rotinas.

6.2.2 Localização da declaração - início de blocos, como no PASCAL S.

6.2.3 Inicialização - não existe inicialização na declaração mas esta pode ser feita através do comando init (ver seção 6.6.7).

6.2.4 Redefinição - não há.

6.2.5 Domínio das declarações - é definido da forma:

(a) Variáveis declaradas em rotinas têm como domínio o bloco que constitui o corpo da rotina.

(b) Variáveis declaradas em sistemas são conhecidas - no bloco do sistema e no bloco de rotinas declaradas no sistema, mas não são conhecidas nos blocos de outros sistemas declarados no interior deste.

(c) Variáveis declaradas como var entry dentro de uma classe podem ser utilizadas pelo sistema na qual a classe foi declarada (seção 6.4).

(d) Constantes e tipos têm seus domínios definidos pelo bloco em que foram declaradas, como no ALGOL 60.

2.2.6 Pré-declarações

Tipos: integer, real, Boolean, char e queue

2.3 - Alocação

6.3.1 Tipos de alocação

(a) Permanente - após a inicialização de um sistema através do comando init seus parâmetros e variáveis tornam-se permanentes isto é, suas áreas não são liberadas após o término da execução do sistema.

(b) Dinâmica - parâmetros e variáveis declaradas em rotinas são alocadas dinamicamente.

(c) Controlada - não há.

6.3.2 Meio

Todas as variáveis são alocadas na memória não existindo acesso aos registradores.

6.3.3 Validade de apontadores - não se aplica

6.4 - Constantes e variáveis

(a) Constantes - como no PASCAL S, sendo que um valor de tipo set é representado na forma:
(. exp_1, ..., exp_n .)

(b) Variáveis - como no PASCAL S, com exclusão de apontadores e incluíndo a especificação:

c.v

sendo c - identificador de uma variável de tipo class;
v - variável declarada na forma var entry dentro da componente de sistema c, de tipo class.

A forma c.v permite a um sistema utilizar uma variável declarada com entry em uma classe definida dentro do sistema.

6.5 - Expressões - como no PASCAL S, devendo haver compatibilidade entre os tipos dos operandos em uma operação.
A chamada de rotinas como função é das seguintes formas;

f (a_1,..., a_n)

f

v.f(a_1,..., a_n)
ou
v.f

sendo f - identificador da rotina;

a_1,..., a_n - expressões que constituem a lista de parâmetros atuais;

v - identificador da componente de sistema na qual a rotina f está declarada com entry.

6.6 - Comandos - os mesmos que existem no PASCAL S, com exclusão do go to e de rótulos (exceto nos comandos internos a um case), com os seguintes comandos adicionais:

(a) cycle c_1; c_2;...; c_n end

onde c_1,..., c_n são comandos.

Este comando efetua a repetição da sequência c_1,..., c_n um número indefinido de vezes, equivalendo a um do forever.

(b) init S_1, S_2,..., S_n

onde S_1,..., S_n são da forma:

c

ou

c(a_1,..., a_n)

sendo c - identificador de uma componente de sistema (v. 6.2.1. b).

$a_1, \ldots, a_n$ - expressões que constituem a lista de parâmetros atuais.

Este comando inicializa as componentes de sistema especificadas, alocando espaço para suas variáveis, efetuando a transmissão dos parâmetros e, finalmente, executando os comandos do bloco que constitui o corpo do sistema. Parâmetros e variáveis se tornam permanentes. Uma componente de sistema deve ser inicializada apenas uma vez.
Quando o sistema inicializado é um processo, sua execução será feita concorrentemente com a execução dos demais processos já inicializados.

(c) Chamada de rotinas - corresponde à chamada de procedimentos existente no PASCAL S. A chamada é especificada na forma:

p

$p(a_1, \ldots, a_n)$

c.p

ou $c.p(a_1, \ldots, a_n)$

sendo p - identificador de rotina;

$a_1, \ldots, a_n$ - lista de parâmetros atuais, constituída de expressões;

c - identificador de uma componente de sistema na qual a rotina p foi declarada com entry.

6.7 - Declaração de rotinas

6.7.1 Formas da declaração

procedure p parâmetro; b

procedure entry p parâmetros; b

function f parâmetros: t; b

function entry f parâmetros: t; b

program pr parâmetros

program pr parâmetros; entry r_1, r_2,..., r_n

onde pr - identificador de programa;

p - identificador de procedimento;
f - identificador de função;

t - identificador de tipo;

b - bloco;

r_1, r_2,..., r_n - identificadores de rotinas declaradas como entry no processo em que ocorre a declaração program;

parâmetros - esta parte pode ser omitida ou então especificada na forma:

$(p_1;...; p_n)$

sendo p_1,..., p_n lista de declarações de parâmetros formais (ver seção 6.7.2).

- Não são permitidas declarações de rotinas dentro de outras rotinas.

- Rotinas declaradas como entry em processos, monitores e classes são denominadas, respectivamente, "process entry", "monitor entry" e "class entry", não podendo ser chamadas do interior do sistema em que foram declaradas. Um "process entry" pode ser chamado apenas por processos sequenciais (ver seção 6.8).
  As rotinas declaradas como "monitor entry" e "class entry" podem ser chamadas por componentes de sistema, sendo que somente uma componente de sistema de cada vez tem acesso às rotinas de uma determinada classe ou monitor.

- A rotina program é utilizada para declarar um programa sequencial, compilado através do PASCAL S, e cuja

execução será controlada por um processo especial, denominado job process. A declaração program deverá aparecer no interior do job process (ver seção 6.8).

6.7.2 Parâmetros

(a) Tipos de parâmetros formais

Parâmetros de rotinas podem ser de tipo simples ou estruturado; sendo que parâmetros de rotinas declaradas como entry não podem ser de tipo queue.

(b) Formas de passagem de parâmetros

(i) "Por valor" - ou parâmetros constantes - são declarados das formas:

$id_1, id_2, \ldots, id_n$: t

ou

$id_1, id_2, \ldots, id_n$: univ t

(ii) "Por referência" - ou parâmetros variáveis - são declarados das formas:

var $id_1, id_2, \ldots, id_n$: t

ou

var $id_1, id_2, \ldots, id_n$: univ t

sendo $id_1, \ldots, id_n$ - identificadores dos parâmetros;

t - tipo dos parâmetros.

A especificação univ é utilizada para suprimir verificação de compatibilidade entre os tipos dos parâmetros formais e atuais.
Parâmetros de funções devem ser passados "por valor".

(c) Associação dos parâmetros formais e atuais e verificação da correspondência.

(i) Parâmetros formais e atuais devem corresponder quanto ao número; a correspondência de ti

po deve existir se não for especificado univ na declaração do parâmetro.

(ii) Uma expressão como parâmetro atual deve corresponder a um parâmetro formal declarado como constante.

6.7.3 Retôrno de valor em funções - No bloco que constitui o corpo da função deve haver uma atribuição da forma:

f:= e

onde f - identificador da função;
e - expressão.

6.7.4 Dados acessíveis a procedimentos e funções

(a) Parâmetros

(b) Dados locais - declarados no bloco que constitui o corpo do procedimento ou função.

(c) Variáveis declaradas no sistema em que o procedimento ou função foi declarado.

(d) Constantes e tipos globais ao procedimento ou função.

6.7.5 Recursão - não há.

6.7.6 Procedimentos e funções externas - não há.

6.7.7 Efeitos colaterais em funções - são restritos desde que os parâmetros devem ser passados "por valor".

6.7.8 Procedimentos e funções pré-declaradas

(a) Funções existentes no PASCAL S: succ, pred, abs , trunc, chr, conv e ord.

(b) Funções e procedimentos adicionais, chamados dentro de monitores:

empty (x) - sendo x uma variável de tipo queue , esta função retorna o valor booleano true se a fila x está vazia - nenhum processo em espera - ou false se existe um processo em espera na fila x.

delay (x) - o processo que está executando uma rotina do monitor é colocado em espera na fila x, perdendo o controle do monitor, que poderá ser utilizado por outro processo.

continue(x)-a execução deste procedimento causa um retorno da rotina do monitor que efetuou a chamada da operação "continue". Se existir um processo em espera na fila x, este processo sairá da fila continuando a ser executada a rotina que havia efetuado a chamada do procedimento delay que o colocou em espera. O novo processo terá acesso exclusivo às variáveis do monitor.

6.8 - Programa

(a) Programa concorrente - é definido como um bloco.

Este programa concorrente (um sistema operacional,por exemplo) é considerado como um processo sem parâmetros denominado initial process que é inicializado automaticamente após a carga do programa.

(b) Programa sequencial - é um programa em PASCAL S compilado e mantido em uma memória secundária.
Sua definição (na linguagem PASCAL S) é da forma:

d Program p parâmetros; b.

sendo p - identificador do programa;

d - prefixo constituído de declarações de constantes, tipos e rotinas (sem o bloco que

constitui seu corpo) e utilizado para permitir ao programa sequencial ter acesso a rotinas do programa concorrente.

b - bloco.

(c) Relação entre um programa sequencial e um programa concorrente.

Um processo do programa concorrente (job process) deverá carregar o programa sequencial na memória e iniciar sua execução. Para isto, o processo deverá conter uma declaração program (ver seção 6.7.1) e as declarações das rotinas na forma entry, que serão utilizadas pelo programa sequencial. Os nomes destas rotinas deverão aparecer na lista de rotinas da declaração program e também deverão ser declaradas no prefixo do programa sequencial. Desta forma, estas rotinas (por exemplo: read, write) definidas no programa concorrente serão acessíveis ao programa sequencial.

O parâmetro mais à direita da declaração program representa uma variável do programa concorrente na qual o programa sequencial será armazenado; este parâmetro não irá aparecer na lista de parâmetros do programa sequencial. Os demais parâmetros da declaração program e do programa sequencial devem corresponder - quanto a tipo e número (a menos da especificação univ)

Após efetuar a carga do programa sequencial na variável especificada na declaração program, o processo (job process) deverá iniciar a sua execução através - do comando de chamada de rotina (ver ítem (c), seção 6.b) efetuando a chamada da rotina program.

6.9 - Recursos de entrada e saída - não existem no PASCAL concorrente. Toda entrada e saída deverá ser feita através - de rotinas pré-declaradas adicionais, dependentes da particular implementação do PASCAL concorrente.

6.10 - Recursos para a manipulação de bits e caracteres.

Não existe manipulação explícita de bits. Existe o tipo char, sendo que sequências de caracteres são consideradas como vetores de tipo char. Tais vetores podem ser manipulados através de indexação.

6.11 - Processamento concorrente

6.11.1 Processo - componente de sistema de tipo process.

6.11.2 Criação e destruição de processos - o número de processos em um programa concorrente é fixo durante a compilação. O comando init é utilizado para alocar à rea para as variáveis e iniciar a execução concorrente de processos.

6.11.3 Exclusão mútua

A única forma de processos concorrentes compartilharem dados é através da definição dos mesmos no interior de monitores, devido às seguintes restrições de linguagem:

(a) Processo, monitores e classes podem utilizar apenas seus parâmetros e variáveis locais.

(b) Processos e classes não podem ser passados como parâmetros a sistemas, com excessão de classes que podem ser parâmetros de classes.

(c) Os parâmetros de sistemas devem ser definidos como constantes.

O acesso aos dados compartilhados é feito através de de chamadas de rotinas declaradas como entry no monitor. A implementação de monitores garante a exclusão mútua na execução destas rotinas.

Classes são utilizadas de forma semelhante a monitores, sendo que somente um processo tem acesso a uma

classe e esta condição pode ser verificada durante a compilação devido às restrições especificadas nos itens (a), (b) e (c).

6.11.4 Sincronização e comunicação entre processos concorrentes.

Juntamente com monitores, os procedimentos "delay", "continue" e "empty" (seção 6.7.8) podem ser utilizados para sincronização entre processos concorrentes.

6.11.5 Processamento de interrupções - não há. As interrupções são processadas pelo núcleo do sistema, codificado em linguagem de máquina.

6.12 - Acesso a características de máquina - não há.

6.13 - Comunicação com procedimentos em outras linguagens -sem informações.

6.14 - Recursos em tempo de compilação - nenhum.

6.15 - Recursos para "depuração" de programas - nenhum.

7. ALGOL B6700 - Extended B6700 ALGOL

A linguagem ALGOL B6700 [13,45] foi desenvolvida pela Burroughs para o computador B6700 e contém o ALGOL 60, a menos de declarações de rótulos. Todos compiladores do B6700, inclusive o ESPOL (seção 8), foram codificados em ALGOL B6700. A linguagem permite acesso à todas as facilidades da máquina e do sistema, na forma de comandos de alto-nível, cuja utilização não afete o processamento do sistema e de outros programas.

7.1 - Tipos

7.1.1 Tipos simples - além dos tipos integer, real e boolean do ALGOL 60, existem: alpha (8 bits), double (real dupla precisão; no B6700 este último tipo tem também o expoente extendido), pointer (utilizado unicamente para referências a linhas de "array's" de caracteres), task e event. Chamamos "linha de array" ao conjunto de elementos que se obtém variando o índice mais à direita e mantendo os demais índices constantes.

7.1.2 Tipos estruturados

array - como no ALGOL 60, de um dos tipos simples ou então: hex, bcl (codificação de bits da Burroughs), ascii ou ebcdic

7.1.3 Definição de novos tipos - não há.

7.2 - Declaração

7.2.1 Formas de declarações

(a) Variáveis

(i) Simples - como no ALGOL 60.

(ii) Array - além da forma existente no ALGOL 60,

a declaração array permite as seguintes especificações adicionais:

long - utilizado em declarações de "array's"de uma única dimensão, para evitar a segmentação feita pelo compilador ao detectar um "array" maior que 1023 palavras.

direct - define um "array" que será utilizado em entrada e saída direta , isto é, sem utilizar as facilidades de entrada e saída existentes na linguagem.

reference- define, apenas, um "dope vector" de "array" com informações sobre o número de dimensões e seus limites inferiores. As demais informações, tais como origem e número de elementos em cada dimensão, bem como a área do "array " são obtidas na primeira vez que for executada uma atribuição de um "array" àquele definido como reference. A declaração array reference permite a associação de tipos e limites diferentes a uma mesma área.

(b) Constante

A declaração value array permite a definição de um array com uma dimensão e inicializado com valores constantes que não podem ser modificados durante a execução.

(c) Rótulos - declarados na forma:

label $id_1,\ldots,id_n$

onde $id_1,\ldots,id_n$ são identificadores de rótulos. Um rótulo não pode ocorrer sem que tenha sido declarado dessa maneira.

(d) switch - como no ALGOL 60.

(e) translatetable - declaração que define uma tabela de tradução constituída de duas sequências de caracteres, através da qual o i-ésimo caráter de uma sequência poderá ser substituído pelo i-ésimo caráter da outra sequência. Identificadores declarados como translatetable podem ser utilizados no comando replace (seção 7.6.7).

(f) truthset - esta declaração associa um identificador a um conjunto de caracteres. Identificadores declarados como truthset são utilizados em testes nos comandos scan e replace (seção 7.6.7).

(g) picture $p_1, p_2,\ldots,p_n$

onde $p_1,\ldots,p_n$ são da forma:

$$id(S_1\ldots S_m)$$

sendo id - identificadores de picture;
$S_1,\ldots,S_m$ - símbolos de especificação de picture (X,Z,9, etc.)

Identificadores declarados como picture são utilizados no comando replace para edição de sequências de caracteres.

(h) declarações adicionais

define (seção 7.14);
file, format, switch file, switch format, list e switch list (seção 7.9);
procedure (seção 7.7);

de apontadores, isto é, um apontador global pode fazer referências a uma linha de "array" local.

7.4 Constantes e variáveis

(a) Constantes

. Booleanas: true e false.

. Numéricas: representadas na base decimal,com ou sem expoente (o expoente é indicado por @), em precisão simples ou dupla.

. Sequências de caracteres: $c"c_1,...,c_n"$

c - indica a codificação EBCDIC ou ASCII;

$c_1,...,c_n$ - sequências de caracteres EBCDIC.

. Sequências numéricas - nas bases 2, 4, 8 e 16, com alinhamento à esquerda ou direita opcional.

(b) Variáveis - simples e subscritadas como no ALGOL 60.

7.5 Expressões

7.5.1 Formas de expressões

7.5.1.1 Expressões simples

(a) Aritméticas e booleanas - como no ALGOL 60 , existindo operadores adicionais e outras representações de operandos.

(i) Operandos

. constantes - sendo permitidas cadeias de até 6 caracteres (uma palavra);

. chamada de funções - como no ALGOL 60;

. variáveis;

. designação de atributo de arquivo,processo (7.11.1, ítem (d)) ou de " array" direto (7.2.1. (a)).

interrupt (seção 7.11);
dump e monitor (seção 7.15).

7.2.2 Localização das declarações - início de blocos,como no ALGOL 60.

7.2.3 Inicialização - não há.

7.2.4 Redefinição - pode ocorrer nas declarações array, direct array e array reference, em área de outra array previamente declarado.

7.2.5 Domínio da declaração - bloco em que a declaração ocorreu.

7.2.6 Pré-declarações - nenhuma.

7.3 Alocação

7.3.1 Tipos de alocação

(a) Estática - variáveis declaradas como own ou no bloco mais externo do programa.

(b) Dinamica - todas as demais declarações.

(c) Controlada - uma linha de array pode ter sua área liberada ou redimencionada através dos comandos - deallocate e resize (seção 7.6.7).

7.3.2 Meio

Todas variáveis são alocadas na memória, não sendo permitido o acesso aos registradores da máquina.

7.3.3 Validade de apontadores - nas versões iniciais do compilador ALGOL B6700 não existe verificação da validade

. especificação de palavra parcial, da forma:

x .[ℓ: n]

sendo x - variável, chamada de função designação de atributo, expressão entre parênteses - ou expressão case (7.5.1.2)

ℓ e n - expressões aritméticas que indicam respectivamente a posição do bit mais à esquerda na palavra e o comprimento em bits da palavra parcial.

. identificador de truthset (7.2.1, ítem (f)).

(ii) Operadores - os que existem no ALGOL 60, com a inclusão dos operadores diádicos:

mod - resto da divisão;

mux - multiplicação com resultado em dupla precisão;

:= - atribuição;

& - concatenação de campos, utilizado - nas formas:

op & (e) c ou op & v c

onde op, e , v - são respectivamente operando, expressão e variável, devendo ser do - mesmo tipo, aritmético ou booleano;

c - especificação de palavras parciais nas formas:

$[\ell_1 : \ell_2 : n]$

ou

$[\ell_1 : \ell_2 : n]$

sendo ℓ_1, ℓ_2 e n - expressões aritméticas.

O efeito é a concatenação da palavra parcial à esquerda do bit ℓ_1 de op, com os n bits mais à esquerda do bit ℓ_2 de e ou de v.

in - operador lógico que efetua busca em tabela, utilizado na forma: in id

sendo - expressão aritmética ou de tipo pointer;

id- identificador declarado como truthset

Os operadores eql (ou =) e neq podem ser utilizados para comparações entre expressões de tipo pointer, resultando em um valor booleano.

Precedência - é a mesma que a do ALGOL 60, sendo que mod e mux têm mesma precedência que * e /; :=, in e & _ têm precedência maior que os demais operadores.

(b) Designação de rótulo - como no ALGOL 60.

(c) Expressão de tipo pointer - nas formas:

op , op + a , op - a ou ar

onde op - operando de tipo pointer, que pode ser

- . variável de tipo pointer;
- . expressão de tipo simples ou case , de tipo pointer e entre parênteses
- . chamada de função - como no ALGOL 60.

a - operando aritmético ou comando de atribuição aritmético.

ar - identificador de array de caracteres, com especificação de índices opcional.

7.5.2 Expressões condicionais

(a) if e_1 then e_2 else e_3 - como no ALGOL 60.

(b) case e $(e_0,\ldots,e_n)$

onde e - expressão aritmética com valor entre 0 e n;

$e_0,\ldots, e_n$ - expressões de mesmo tipo.

7.5.3 Mistura de tipos e conversão - como no ALGOL 60, com a inclusão do tipo double em expressões aritméticas.

7.6 Comandos

7.6.1 Rótulos - como no ALGOL 60.

7.6.2 Bloco e comando composto como no ALGOL 60.

7.6.3 Atribuição

Nas formas:

(a) Aritmética e booleana - como no ALGOL 60, sendo que a variável à esquerda do sinal ":=" pode ser uma especificação de palavra parcial ou designador de atributo de task, como em expressões (seção 7.5.1). v := v $\pm$ c também pode ser escrita - como v := * $\pm$ c onde v é uma variável e c uma constante.

(b) Pointer - atribuição, com forma análoga à atribuição aritmética, sendo que a expressão à direita -

do ":=" e a variável à esquerda do mesmo são do tipo pointer.

(c) Referência a array - este comando é utilizado para a inicialização do "dope vector" de um "array" declarado como reference, sendo especificada da forma:

$$v := a[\, e_1,\ldots,e_n,\ s_1,\ldots,s_m \,]$$

ou $v := a$

onde v - variável declarada como array reference;

a - identificador de "array" de n dimensões;

$e_1,\ldots,e_n$ - expressões aritméticas;

s_i - "*" (i=1,...,m e m < n)

A primeira forma de atribuição seleciona uma parte do array a, permitindo uma implementação mais eficiente das operações sobre partes de "array's".

(d) task - (ver seção 7.11)

$$dt_1 \,.\, at := dt_2$$

sendo dt_1, dt_2 - designadores de task's (seção 7.11.1, ítem (c))

at ε {exceptiontask, partner}, atributos - de task (seção 7.11.1, ítem (b))

7.6.4 Estruturas de controle

7.6.4.1 Desvio incondicional ou escape.

go to - como no ALGOL 60

exit - utilizado, apenas, para abandonar um comando "vector mode" (Ver 7.6.7).

7.6.4.2 Comandos seletivos

(a) if - como no ALGOL 60 sendo ainda que entre o then e else é permitido um comando - qualquer; a ambiguidade é resolvida pela "regra do mais próximo then".

(b) case

case e of begin c_0; c_1;...;c_n end

case e of begin n_1:ℓ_1; n_2:ℓ_2;..;n_m: ℓ_m end

onde e - expressão aritmética;

$c_0,\ldots,c_n$ - comandos;

$n_1,\ldots,n_m$ - números inteiros sem sinal;

$\ell_1,\ldots,\ell_m$ - lista de comandos.

Na primeira forma, se $0 < e < n$, será executado o comando c_e; caso contrário ocorre uma interrupção de "índice inválido".

Na segunda forma do comando case, se existir k, (com $1 < k < n$) tal que $e = n_k$ então será executada a lista de comandos ℓ_k. Caso constrário pode ocorrer as interrupções de "índice inválido" ou "operador inválido" conforme e seja maior que $n_1,\ldots,n_m$ ou $e < n_m$ mas diferente de $n_1,\ldots,n_{m-1}$.

7.6.4.3 Comandos repetitivos

(a) do c until e_1

(b) for - como no ALGOL 60.

(c) thru e_2 do c

(d) <u>while</u> e_1 <u>do</u> c

sendo e_1 - expressão booleana;

e_2 - expressão aritmética;

c - comando.

7.6.5 Chamada de procedimento - como no ALGOL 60, sendo permitidos outros tipos de parâmetros atuais (ver 7.7.2).

7.6.6 Comando nulo - como em ALGOL 60.

7.6.7 Outros

(a) Manipulação de "array's"

<u>deallocate</u> - libera uma linha de um "array".

<u>resize</u> - redefine o tamanho de uma linha de um "array", opcionalmente mantendo os valores anteriores dos elementos da linha.

<u>swap</u> - permuta o conteúdo de dois "array's" desde que ambos tenham o mesmo número de elementos, tamanho de caráter e número de dimensões e não sejam - "diretos".

<u>vectormode</u> - reúne um grupo de comandos que efetuam operações sobre "array's", nos quais será usado um dispositivo especial opcional, para cálculo mais rápido de endereços.

(b) Manipulação de sequências de caracteres

<u>fill</u> - insere em uma linha de um <u>array</u> de caracteres uma lista de números ou de sequências de caracteres.

replace - substitui partes de uma sequência de caracteres por caracteres de outra sequência. Permite especificações for, while, until whilein, untilin que controlam o término de substituição.

scan - efetua uma busca em uma sequência de caracteres sendo que o término da busca é determinado por especificações for, while, until, whilein ou untilin.

(c) sort e merge - comandos para classificar e intercalar arquivos.

(d) programdump - ver seção 7.15.

(e) zip - este comando ativa o compilador de WFL ("WORK FLOW LANGUAGE") que irá considerar como cartões de controle de programas o conteúdo de uma linha de "array" ou de um arquivo especificados no comando zip. Com isso tem-se acesso, por programa, a ações do Sistema Operacional, podendo- se iniciar processos paralelos.

(f) comandos de entrada e saída - ver seção 7.9.

(g) comandos para processamento concorrente - ver seção 7.11.

7.7 Declaração de função ou procedimento

7.7.1 Forma da declaração: como no ALGOL 60, sendo que o corpo do procedimento pode ser substituido por external

ou forward.

7.7.2 Parâmetros

(a) Tipos de parâmetros formais: variáveis simples , array, direct array, file, switch, picture, procedure, event, format, task, list, switch list , label, switch format , direct file, event array , task array.

(b) Formas de passagem de parâmetros: por nome e por valor como no ALGOL 60.

(c) Associação dos parâmetros formais e atuais e verificação da correspondência.

Um parâmetro atual deve ser do mesmo tipo do parâmetro formal correspondente, havendo verificação desta correspondência.

7.7.3 Retorno de valor em funções - como no ALGOL 60,através de comando de atribuição.

7.7.4 Dados acessíveis a funções e procedimentos:parâmetros, variáveis locais e globais, como no ALGOL 60.

7.7.5 Recursão - como no ALGOL 60.

7.7.6 Procedimentos e funções externas - procedimentos compilados separadamente e declarados no programa que os utiliza como external.

7.7.7 Efeitos colaterais em funções - nenhuma restrição.

7.7.8 Procedimentos e funções pré-declaradas - denominados intrínsecos do sistema:

(a) Funções aritméticas - abs, exp, cos, gamma, etc.

(b) Funções especiais:

listlookup - efetua uma busca do valor de uma expressão em uma lista ligada, a partir de uma palavra cujo índice é especificado por uma expressão aritmética. A lista é constituida de palavras de uma linha de "array", sendo que cada palavra contém um valor, representado por 28 bits e um apontador para a palavra seguinte da lista, representado por 20 bits. A função retorna o índice do elemento que aponta para a palavra que contém o valor procurado, ou -1 se o valor não foi encontrado na lista.

time - fornece o tempo decorrido desde o início da execução do programa ou a data atual.

size - fornece o número de elementos em uma linha de um determinado "array " ou então, se o argumento da função é um apontador, fornece o tamanho (em bits) do caráter apontado (4,6, 8).

delta - tem como argumentos dois apontadores, fornecendo como resultado o número de caracteres entre os valores apontados.

masksearch - efetua uma busca de uma configuração de bits em um "array" de uma dimensão, sendo que uma máscara indicará os bits da configuração que serão testados. O resultado é o índice da

palavra que coincidiu, segundo a máscara, com a configuração de bits apresentada.

random - gera um número aleatório.

readlock - armazena o valor de uma expressão em uma variável da memória e retorna o valor anterior desta variável em um único ciclo de memória.

available - testa se um evento está no estado disponível, retornando, neste caso, o valor true.

happened - verifica se um evento ocorreu, retornando neste caso o valor true.

toggle - retorna como valor o conteúdo de um registrador que mantém os valores true ou false conforme a condição verificada na execução mais recente de um comando scan ou replace.

compile time - obtém o tempo de compilação do programa.

7.8 Programa - como em ALGOL 60 e também somente uma declaração de procedimento.

7.9 Recursos de entrada e saída

(a) Declarações

(i) Arquivos

file $f_1, f_2, \ldots, f_n$

direct file $f_1, f_2, \ldots, f_n$

onde $f_1, \ldots, f_n$ são especificações da forma:

i_d $(at_1=v_1, at_2=v_2,\ldots,at_m=v_m)$

sendo id - identificador de arquivo;

$at_1,\ldots,at_m$ - atributos tais como: blocksize, density, eof, etc.

$v_1,\ldots,v_m$ - valores iniciais que serão atribuidos aos atributos.

A declaração direct file define um arquivo que será utilizado em entrada e saída direta, isto é, sem utilizar as facilidades existentes na linguagem.

(ii) Formatos para entrada e saída com edição.

format m $f_1,\ldots,f_n$

onde m ε { λ, in, out} tipo de transferência;

$f_1,\ldots,f_n$ são das formas:

id (especificações de edição)

ou

id <especificações de edição>

sendo id - identificador de formato;

especificações de edição - semelhantes às que existem no FORTRAN com recursos adicionais, tais como fatores de repetição variáveis durante a execução.

(iii) list $\ell_1,\ldots,\ell_n$

onde $\ell_1,\ldots,\ell_n$ são da forma:

id $(e\ell_1,\ldots,e\ell_n)$

sendo id - identificador de lista;

$e\ell_1,\ldots, e\ell_n$ - listas de variáveis,

com especificações de repetição (for, thru, while) ou de seleção (if e case).

Essas listas de variáveis são usadas nos comandos de entrada e saída.

(iv) switch file, list e format

direct switch file id_1 := $a_1,\ldots,a_n$

switch file id_1 := $a_1,\ldots,a_n$

switch list id_2 := $\ell_1,\ldots,\ell_n$

switch format id_3 := $f_1,\ldots,f_1$

onde id_1 - identificadór de switch file;

id_2 - identificador de switch list;

id_3 - identificador de switch format;

$a_1,\ldots,a_n$ - identificadores de arquivos;

$\ell_1,\ldots,\ell_n$ - identificadores de listas;

$f_1,\ldots,f_n$ - identificadores de formatos;

As declarações switch file, list e format permitem, designação de um arquivo,lista ou formato de uma lista, através de indexação, de forma análoga à switch do ALGOL 60.

(b) Comandos

(i) Comunicação com o operador

accept (p)

display (p)

sendo p - expressão de tipo pointer.

Estes comandos exibem ao operador do compu-

tador a sequência de caracteres apontada por p. No comando accept, após a exibição da mensagem, o programa fica em espera da resposta do operador que é colocada na linha do array de caracteres apontada por p.

(ii) "Fechamento" de arquivos

close (f)

close (f,op_1)

lock (f)

lock (f,op_2)

onde f - designação de arquivo;

$op_{1\varepsilon}$ { *, purge, reel, crunch}

$op_{2\varepsilon}$ { *, crunch}

op_1 e op_2 especificam o estado do arquivo f após a execução do comando.

O comando lock torna o arquivo permanente.

(iii) Troca do conteúdo de áreas de dois arquivos

exchange (f_1, f_2)

sendo f_1 e f_2 especificações de arquivos e áreas dos mesmos que serão trocadas.

(iv) Atribuição de valores a atributos de arquivos

$f(at_1 = c_1, at_2 = c_2, \ldots, at_n = c_n)$

sendo f - identificador de arquivo;

$at_1, \ldots, at_n$ - atributos de arquivos;
$c_1, \ldots, c_n$ - constantes.

(v) Entrada e saída

read (f, form, ℓ) ações

write (f, form, ℓ) ações

onde f - designação de arquivo ou linha de "array" no caso de entrada e saída direta.

form - especificação de formato de uma das formas:

< especificações de formato > - como em declaração de formato;

* - para entrada e saída sem conversão;

/ - entrada e saída com formato livre;

id_1 - identificador de formato (4.7.9.a)

$id_2[e]$ - sendo id_2 declarado como switch format e e uma expressão.

ℓ - lista de variáveis, especificadas como na declaração list, identificadores declarados como list ou identificadores declarados como switch seguidos da especificação de um índice de uma lista.

ações - especificações de rótulos de comandos para o qual o controle será transferido quando ocorrer um erro de entrada ou saída, e especificação de evento , para entrada e saída assíncrona. O e-

vento especificado será "causado" quando ocorrer o término da operação de entrada ou saída. Estas especificações são opcionais.

Esses comandos podem ser utilizados como funções retornando o valor true se a transferência foi completada sem erro e false em caso contrário. Neste caso não devem ser especificadas a parte "ações".

(vi) Comandos de controle

rewind (f)

seek (f, [e_1])

space (f, e_2) ações

sendo f - designação de um arquivo;

e_1- expressão aritmética para selecionar um registro de um arquivo que será transferido para um "buffer" através do comando seek;

e_2- expressão aritmética que indica o número de registros que serão ignorados pelo comando space.

ações-especificações de evento e rótulos, como nos comandos read e write.

7.10 - Recursos para manipulação de bits e caracteres

(a) Tipos alpha, pointer e array de caracteres;

(b) Manipulação de sequências de caracteres e "array's" de caracteres através de apontadores, comandos fill

scan e replace e indexação.

(c) Acesso a grupos de bits de uma palavra através da especificação de palavra parcial (seção 7.5.1.1, ítem (a)).

7.11 - Processamento concorrente

7.11.1 Processos

(a) Um processo (task) pode ser definido como um programa ou procedimento (sem especificação de tipo para utilização como função) externo ou interno ao programa.

(b) A cada processo estão associadas diversas informações denominadas atributos. Os atributos de processos são os seguintes:

name - de tipo pointer, indica o nome de um arquivo no qual se encontra o código do processo.

taskvalue - de tipo real, utilizado para transmitir valores entre processos.

status - de tipo real, indica o estado atual do processo.

1 - pronto para ser ativado;
2 - ativo;
3 - suspenso;
-1 - terminado ou descontinuado;
-2 - não foi possível inicializar o processo;

exceptiontask - de tipo task, especifica um processo que será notificado quando o atributo status sofrer uma mudança de valor;

partner - de tipo task, indica o processo que deverá receber o controle quando o processo ao qual o atributo está associado executar o comando continue.

Os valores dos atributos exceptiontask e partner podem ser alterados através do comando de atribuição de task (seção 7.6.3); task value e status pelo comando de atribuição aritmético e name pelo comando replace.

(c) Designadores de task's - são das formas:

t_1

t_2 $e_1, \ldots, e_n$

myself

dt . exceptiontask

dt . partner

onde t_1 e t_2 são identificadores declarados como task, sendo t_2 um array;

$e_1, \ldots, e_2$ - expressões aritméticas;

dt - designador de task

Usa-se myself para designar um processo dentro do próprio processo.

(d) Designador de atributos - é da forma:

dt . atr

sendo dt - designador de task;

atr - atributo de task (ver ítem (b) desta seção).

(e) Bloco crítico de um processo - é o bloco mais interno que contém uma das seguintes declarações:

(1) Procedimento que constitui o código do processo.

(2) Identificador de tipo task, associado ao processo quando este foi criado.

(3) Argumento passado por nome na criação do processo.

7.11.2 Criação de processos

(a) Co-rotina - processo não concorrente

call p [dt]

ou

call p (lista de parâmetros atuais) [dt]

(b) Processo concorrente, dependente do processo que o criou.

process p [dt]

ou

process p (lista de parâmetros atuais) [dt]

(c) Processo concorrente, independente do processo que o criou.

run p [dt]

ou

run p (lista de parâmetros atuais) [dt]

Nos comandos acima:

p - identificador de procedimento que constitui o código do processo; p deverá ser compilado separadamente para ativação como processo concorrente através do comando run;

lista de parâmetros atuais - como em chamadas de procedimentos;

dt - designador de task que é associado ao processo criado.

Co-rotinas e processos criados através do coman-

do process devem terminar antes que o processo que os criou desative seus blocos críticos.

Um procedimento de um processo criado através do comando run não pode ter parâmetros passados"por nome" nem pode utilizar variáveis globais ao processo que o criou.

7.11.3 Exclusão mútua - pode ser obtida de duas formas:

(a) Função readlock - permite a implementação de exclusão mútua de uma forma primitiva. A execução desta função é indivisível, isto é, em um único ciclo de máquina o valor de uma expressão é atribuído a uma variável, retornando como resultado o valor anterior da mesma.

(b) Comandos especiais

Associando um evento e às regiões críticas de processos concorrentes, nas quais a execução destes processos se excluem mutuamente, o estado available (disponível) do evento e indica que nenhum processo está em sua região crítica relacionada ao evento, e o estado not available (não disponível), indica que existe um processo em sua região crítica associada ao evento e.Inicialmente, após a declaração, o evento está no estado available. Os estados available e not available são considerados "propriedades de recurso".

(i) procure (e)

Se o evento e está no estado available , a execução do processo continua normalmente; se e está no estado not available, o processo é bloqueado em uma fila até que e se torne available.

(ii) liberate (e)

O efeito deste comando é o seguinte:

. Se a fila de processos a espera que o evento e se torne available é vazia, o estado do evento e continua available.

. Se existem processos esperando o evento e se tornar available, o processo de maior prioridade será ativado e o evento se torna not available.

. Todos os processos que esperam o evento e ser causado (ver seção 7.11.4) são ativados.

. Se existe algum processo colocado em espera pela execução do comando waitandreset (ver seção 7.11.4), o estado do evento , quanto à "propriedade condição", será not happened (não ocorrido), em caso contrário será happened (ocorrido).

(iii) fix (e)

O evento e é colocado no estado not available. Este comando pode ser utilizado como função de tipo boolean, retornando o valor true, se o estado anterior de e era available, e false em caso contrário.

(iv) free (e)

O evento e torna-se available mas nenhum processo em espera é ativado. Pode ser utilizado como função de tipo boolean, retornando o valor true se o evento estava no estado available e false em caso contrário.

A exclusão mútua entre dois processo, por exemplo pode ser garantida da seguinte forma:

```
procedure p1;                    procedure p2;
   begin                            begin
     :                                :
   procure (e);                     procure (e);
     :                                :
   liberate (e);                    liberate (e);
     :                                :
   end                              end
```

Sendo e declarado como event, as regiões entre os comandos procure e liberate constituem regiões críticas, sendo executadas em exclusão mútua com as regiões críticas associadas a e em outros processos.

7.11.4 Sincronização e comunicação entre processos concorrentes

Juntamente com os estados considerados "propriedades de recursos" (available e not available), um evento pode estar em um dos estados happened (ocorrido) e not happened (não ocorrido) considerados "propriedades de condição". Existe uma fila de espera associada a estes dois últimos estados, além de fila associada aos estados available e not available. Inicialmente, após a declaração, um evento está no estado not happened (e available).

A sincronização e comunicação entre processos pode ser feita através dos seguintes comandos.

(a) set (e)

coloca o evento e no estado happened, sem ativar os processos em espera.

(b) reset (e)

coloca o evento e no estado not happened.

(c) cause (e)

Ativa todos os processos que esperam o evento e ser causado. Se algum processo foi colocado em espera pelo comando waitandreset, o evento voltará ao estado not happened após a ativação dos processos, em caso contrário ficará no estado happened.

(d) causeandreset (e)

Tem o mesmo efeito do comando cause, exceto quanto ao estado do evento que ficará not happened após a ativação dos processos.

(e) wait (e_1,..., e_n)

wait (t)

when (t) - equivalente a wait (t).

wait (t, e_1,..., e_n)

onde t - expressão aritmética com valor real que indica um período de tempo em segundos e frações de segundos;

e_1,..., e_n - designadores de eventos.

Na primeira forma do comando wait , se algum dos eventos e_1,..., e_n estiver not happened, o processo será colocado em espera até que um dos eventos seja causado; em caso contrário a execução continuará normalmente.

Com a especificação do intervalo de tempo t, o processo ficará em espera até que decorra o tempo t ou um dos eventos e_1,..., e_n seja causado.

(f) waitandreset (e_1,..., e_n)

waitandreset (t)

ou

waitandreset (t, e_1,..., e_n)

Têm o mesmo efeito do comando wait, exceto quanto ao estado do evento que causar a ativação dos processos em espera, que ficará no estado not happened.

Os comandos wait e waitandreset podem ser utilizados como funções de tipo integer cujo valor retornado é o índice do evento da lista $e_1,\ldots, e_n$ responsável pela ativação do processo.

A transferência de controle entre processos ativados como co-rotinas é feita pelo comando:

continue

ou

continue (dt)

sendo dt - designador de task

A primeira forma do comando causa a transferência do controle para o processo associado ao atributo partner. A segunda forma do comando causa a transferência do controle para o processo dt. A execução do processo que recebeu o controle continua após o comando continue ou call que havia causado a transferência de controle para outro processo.

7.11.5 Processamento de interrupção

(a) Interrupções definidas pelo programador

(i) Declaração

interrupt id:c

onde id - identificador da interrupção;
c - comando associado à interrupção.

Uma interrupção pode ser associada a um evento, através do comando attach, e pode estar em um dos estados.

enabled - quando o evento for causado , ocorrerá a interrupção, sendo executado o comando associado à mesma através da declaração.

disabled - quando o evento for causado, a interrupção será colocada em uma fila, ficando pendente.

(ii) Comandos

attach id to e

detach id

onde id - identificador declarado como interrupção;

e - evento que é associado à interrupção.

Em cada instante uma interrupção pode estar associada a apenas um evento, mas diferentes interrupções podem estar associadas a um mesmo evento.

O comando detach desfaz a associação entre a interrupção e o evento. Se houver alguma interrupção pendente, esta será perdida.

(b) Interrupções decorrentes de falhas de programa.

Estas interrupções são processadas através dos comandos:

on ℓ inf, c

on ℓ inf: c

ou

on ℓ

onde ℓ - lista de falhas de programa, da forma:

f ou

f_1 or f_2 or ... or f_n

sendo que

f, f_1,..., f_n ε {zerodivide, scanparity, integeroverflow, invalidop, memoryprotect, ..., stackoverflow}.

As falhas de programa, inativequeue, botton of stack, sequence e stackunderflow não podem ser processadas em ALGOL B6700.

inf - especifica variáveis nas quais serão guardadas as informações relacionadas à interrupção.

c - comando que será executado quando ocorrer a interrupção.

Na segunda forma do comando on , após a execução do comando c o processamento continuará a partir do ponto em que foi interrompido. Na primeira forma do comando on o retorno não é automático, devendo ser especificado no comando c, através de go to, o comando que será executado após o processamento da interrupção.

A terceira forma é utilizada apenas para colocar a interrupção especificada no estado disabled,enquanto que as duas primeiras deixam a interrupção no estado enabled.

7.12 - Acesso a características de máquina - não existe acesso a registradores, instruções ou localizações absolutas da memória.

7.13 - Comunicação com procedimentos em outras linguagens - é permitida, desde que sejam respeitadas as convenções de passagem de parâmetros.

7.14 - Recursos em tempo de compilação

(a) Declaração de macro

define $d_1, d_2, \ldots, d_n$

onde $d_1, \ldots, d_n$ são de uma das formas:

id ($id_1, \ldots, id_m$) = texto #

id = texto #

sendo id - identificador de macro;

$id_1, \ldots, id_m$ - identificadores que constituem os parâmetros da macro.

texto - sequência de caracteres na qual podem aparecer os identificadores $id_1, \ldots id_m$; o texto não deve conter o caráter"#".

(b) Chamada de macro - nas formas:

id

id ($texto_1, \ldots, texto_m$) ou

id [$texto_1, \ldots, texto_m$]

sendo id - identificador declarado como macro;

$texto_1, \ldots, texto_m$ - sequências de caracteres que não contenham vírgulas e "bem formada" em relação a parênteses e colchetes.

Durante a compilação, cada chamada de macro do programa fonte será substituída pelo texto de macro. Nesse texto, cada ocorrência do parâmetro id_i será substituída por $texto_i$ (i=1,..., m).

7.15 - Recursos para "depuração" de programas

(a) Declarações

(i) Dump $arq_1(\ell_1)\ c_1,\ldots,arq_n(\ell_n)\ c_n$

sendo $arq_1,\ldots, arq_n$ - identificadores de arquivos;

$\ell_1,\ldots, \ell_n$ - lista de variáveis simples, identificadores de "array" e de rótulos;

$c_1,\ldots, c_n$ - informações de controle.

Este comando registra no arquivo arq_i os valores das variáveis e do número de execuções dos comandos associados a rótulos que aparecem na lista ℓ_i. A frequência com que estes valores são registrados depende da informação de controle c_i.

(ii) monitor $id_1(\ell_1),\ id_2(\ell_2),\ldots,\ id_n(\ell_n)$

onde $id_1,\ldots,id_n$ - identificadores de arquivos ou de procedimentos;

$\ell_1,\ldots,\ \ell_n$ - lista de variáveis simples, identificadores de arra's e de rótulos.

Cada vez que for efetuada uma atribuição a uma variável simples ou "array" ou então executado um rótulo da lista ℓ_i, a declaração monitor causa a impressão de informações no arquivo id_i ou se id_i é um procedimento, causa a chamada de id_i.

(b) Comando programdump

programdump $(op_1,\ldots,op_n)$

ou programdump

onde $op_1,\ldots,op_n \in$ {array, arrays, base, code, file, files, all, expressão aritmética }.

Este comando efetua a impressão das informações selecionadas por $op_1,\ldots,op_n$, contidas na pilha do progra

ma. Se for especificada uma expressão aritmética, os bits de seu valor indicam as informações desejadas (por exemplo, se o sétimo bit for 1, equivale à especificação base).

8. ESPOL - Executive System Problem Oriented Language

A linguagem ESPOL [14] foi desenvolvida pela Burroughs especificamente para a programação do sistema operacional MCP-Master Control Program - do computador B6700. Tem a mesma estrutura do ALGOL B6700 e possui muitos recursos dependentes da máquina.

8.1 - Tipos

8.1.1 Tipos simples - todos do ALGOL B6700 exceto task e alpha, com a inclusão de:

word - 48 bits sem interpretação de tipo;

reference - apontador para localizações da memória .

8.1.2 Tipos estruturados

queue, queue array - filas e "array" de filas.

array, direct array, event array - como no ALGOL B6700.

8.1.3 Definição de novos tipos - não há.

8.2 - Declarações

8.2.1 Formas de declaração

(a) Variáveis simples - como no ALGOL 60 sendo que opcionalmente podem ser especificados valores iniciais e um endereço para alocação.

(b) Variáveis de tipo array, direct array e array event

- várias dimensões;
- limites inferiores não são especificados (tomados como zero);
- inicialização opcional;
- especificação opcional de um endereço para aloca

ção do "dope vector" do "array".

- quando a declaração se inicia com o atributo save, o "array" não estará sujeito a ser removido da memória durante a execução.
- "array" de tipo event não pode ter na declaração a opção save ou own, nem especificações de valores iniciais.

(c) field - associa um identificador a uma especificação de posição e de comprimento (em bits) de um campo de uma palavra. Por exemplo , field a = 3:2, associa o identificador a ao campo de 2 bits de comprimento cujo bit mais à esquerda ocupa a posição 3 de uma palavra.

(d) layout - associa um identificador a uma lista de especificações de campos ("field's")de uma palavra.

(e) queue e queue array - declaração de fila e "array" de filas, nas formas:

queue q p_1(e_1, e_2,..., e_n: i_1, i_2,..., i_m); v; d ;
using alg_1: alg_2: ...: alg_k

ou

queue array q p_1 [exp] (e_1, e_2,..., e_n: i_1, i_2, ..., i_m); v; d; using alg_1: alg_2:...: :alg_k

onde q - identificador de fila;
p_1- especificação, que pode ser omitida, de um segundo nome associado à fila e referido no interior da declaração queue.
Forma:id end

sendo id - identificador da fila, local à declaração queue;

end - especificação de um endereço.

exp - expressão aritmética que indica o limite superior do índice do "array" de filas.

$e_1,\ldots, e_n, i_1,\ldots, i_n$ - lista de itens que constituem uma entrada da fila q; $e_1,\ldots, e_n$ são locais ao bloco em que houve a declaração queue e $i_1,\ldots, i_m$ são locais à declaração queue.

v - especificações de itens passados por valor , análoga à "< value part >" das declarações de procedimentos do ALGOL 60.

d - declarações de tipos dos itens.

$alg_1,\ldots, alg_k$ - especificações de algoritmos, locais ao bloco da declaração queue, das seguintes formas:

b if exp_1

r is exp_2

to a, c

lock ou lock gn

population,c

sendo b ε { empty, full }, identificadores de algoritmos booleanos;

r ε { allocate, next, last, first, prior } identificadores de algoritmos de tipo reference;

a ε { insert, remove, delink, identificador} identificadores de algoritmos;

exp_1, exp_2 - expressões respectivamente bo-

oleana e de tipo reference;

c - comando

Uma fila é manipulada através de chamadas aos seus algoritmos e de comandos de atribuição de tipo queue.

(f) array constante - como no ALGOL 60.

(g) file, interrupt, label, monitor, picture e define como no ALGOL B6700.

8.2.2 Localização da declaração - início de blocos.

8.2.3 Inicialização - variáveis de tipo simples ou "array" podem ser inicializadas na declaração.

8.2.4 Redefinição - através de especificação de um endereço em declarações de variáveis simples e de "array".

8.2.5 Domínio da declaração - como no ALGOL 60.

8.2.6 Pré-declarações - além dos procedimentos intrínsecos (seção 8.7.8) existem pré-declarados os identificadores.

topofstack - de tipo word, corresponde ao topo da pilha de execução.

tag - pré-declarado como field tag = 50:3

entry - de tipo reference, local e uma declaração queue e utilizado para referências à entrada mais recente da fila.

index - local a uma declaração queue array, indica o índice de uma fila do "array".

m [i] ou memory [i]- array de tipo word de uma dimensão, para endereçamento de localizações absolutas de memória.

register [i] - array de uma dimensão para acesso aos re

gistradores da máquina.

stack [x,y] - array de tipo real, bidimencional, sendo x o número da pilha e y o índice de uma palavra dentro da pilha.

stackvector [n] - array de uma dimensão e de tipo word, sendo n o índice de uma linha da pilha.

wordstack [x,y] - equivale a "stack", sendo de tipo word.

8.3- Alocação

8.3.1 Memória e registradores - as variáveis declaradas são alocadas na memória, existindo pré-declarado o vetor "register" que permite acesso aos registradores da máquina. Através de especificação de um endereço na declaração de variável, o programador pode escolher a área em que uma variável será alocada. A linguagem permite a manipulação do "dope vector" de "array's" e o acesso ao "tag" de palavras da memória.

8.3.2 Tipos de alocação - como no ALGOL B6700, sendo que a especificação own causa a alocação das variáveis em nível zero (nível inferior de endereçamento, utilizado apenas pelo sistema operacional).

8.3.3 Validade de apontadores - não existem restrições

8.4 - Constantes e variáveis - como no ALGOL B6700, com a inclusão da constante null (de tipo reference) e as formas de variáveis:

(a) Designação de ítem -

ítem @ e_r

ou

id ∂ e_r

onde Ítem - designação de ítem;
id - identificador de um ítem de uma fila;
e_r - expressão de tipo reference que indica uma entrada de uma fila;

(b) Especificação de palavra parcial - em lugar da forma existente no ALGOL B6700, tem-se:

op.f

sendo op - operando aritmético ou booleano (seção 7.5.1.1);
f - identificador declarado como field.

8.5 - Expressões

8.5.1 Formas

8.5.1.1 Expressões simples - de tipo aritmético, booleano, pointer e designação de rótulo, como no ALGOL B6700 ou ainda:

(a) Expressão de tipo word - consistindo de uma atribuição, designação do ítem ou chamada de função de tipo word, expressão de tipo word entre parênteses ou ainda a forma word (exp), onde exp é uma expressão aritmética, booleana, reference ou array

(b) Expressão de tipo array - para manipulação de "array's" ou partes de "array's" (denominadas "sub-array's"). A única operação permitida é a atribuição que inicializa ou altera informações do "dope vector" de um "array". Um "sub-array" é representado na forma de uma variável indexada na qual os índices mais à direita são

indefinidos, indicador por "*".

(c) Expressão reference - resulta em um apontador para uma localização, consistindo de uma variável, constante, atribuição ou chamada de função de tipo reference, um designador de fila (representado como variável simples ou subscritada), expressão de tipo entry ou a forma; reference (v) onde v é uma variável, gerando um apontador para a variável. A expressão de tipo entry é da forma:

$$q\ (a_1, a_2, \ldots, a_n)$$

sendo q - designador de fila;

$a_1, \ldots, a_n$ - lista de itens atuais de forma análoga aos parâmetros atuais do ALGOL 60, que correspondem aos itens $e_1, \ldots, e_n$ da declaração queue (seção 8.2.1, ítem (e)).

A expressão entry causa a criação de uma n-upla com as informações dos itens atuais e com o formato de uma entrada da fila q. O resultado da expressão é o apontador para a n-upla criada.

8.5.1.2 Expressões condicionais e seletivas - como no ALGOL B6700.

8.5.2 Mistura de tipos e conversão - como no ALGOL B6700, não se aplicando a expressões de tipos word.

8.6 - Comandos

8.6.1 Rótulo - como no ALGOL B6700.

8.6.2 Bloco e comando composto - como no ALGOL B6700.

8.6.3 Atribuição - como no ALGOL B6700, sendo que:

(a) O operador de atribuição pode ser: ←, :=, ←*, :=*.

(b) Pode ser de tipo aritmético, booleano, reference, word (8.5.1.1 ítem (a)), array (8.5.1.1 ítem (b))e queue.

A forma da atribuição queue é a seguinte:

q atr e_r

onde q - designador de fila (8.5.1., ítem (c));

atr - operador de atribuição (ítem (a), nesta seção);

e_r - expressão de tipo reference que aponta para uma entrada de fila.

A atribuição queue torna a entrada apontada por e_r acessível ao algoritmo de inserção definido na declaração da fila q.

8.6.4 Estruturas de controle - iguais às do ALGOL B6700.

8.6.5 Chamada de procedimento - como no ALGOL 60.

8.6.6 Comando nulo - como em ALGOL 60

8.6.7 Outros

(a) Processamento concorrente - seção 8.11.

(b) Entrada e saída - seção 8.9.

(c) Manipulação de caracteres - scan e replace, como no ALGOL B6700 (não tem o comando fill), (seção 7.6.7).

8.7 - Declaração de função ou procedimento

8.7.1 Forma de declaração - como no ALGOL 60, sendo que:

(a) A declaração pode ser precedida de um dos atributos:

save - indica ao compilador que o código do procedimento deverá ficar no mesmo segmento que o bloco em que foi declarado.

save 1-indica ao compilador que o procedimento será utilizado para inicialização, devendo o seu código ser gerado após o bloco inicial de informações carregado pelo "hardware".

(b) O corpo de procedimentos, além de um comando, poderá ser external, forward ou null.

8.7.2 Parâmetros

(a) Tipos de parâmetros formais: simples, queue, array, picture, direct array e procedure.

(b) Formas de passagem de parâmetros - por nome e por valor, como no ALGOL 60 e por referência (somente no caso de parâmetros de tipo event).

(c) Associação dos parâmetros atuais e formais e verificação de correspondência - como no ALGOL 60.

8.7.3 Retorno de valor em funções - como no ALGOL 60.

8.7.4 Dados acessíveis a procedimentos e funções - como no ALGOL 60.

8.7.5 Recursão - é permitida.

8.7.6 Procedimentos e funções externas - são permitidas.

8.7.7 Efeitos colaterais em funções - não existem restrições.

8.7.8 Procedimentos pré-declarados - denominados intrínsecos:

(a) Foram incluídos no ESPOL, todos procedimentos do ALGOL B6700 que correspondem a uma instrução de máquina. Além destes, também são comuns ao ALGOL B6700, os seguintes procedimentos:
dmax, dmin, max, min, available, happened.

(b) Existem no ESPOL outros procedimentos especiais, que correspondem, em geral, a uma ou duas operações da máquina e que atuam sobre a pilha de execução, estados dos processadores, dispositivos de entrada e saída e controle da execução do programa, tais como: allow, disallow, exit, heyou, iio, pause, halt, etc.

8.8 - Programa - como no ALGOL 60.

8.9 - Recursos de entrada e saída.

(a) Existentes no ALGOL B6700.

Declarações: file.

Comandos: close, lock, rewind, seek e space.

(b) Restritos em relação ao ALGOL B6700.

Comandos read e write sem especificação de formatos:

read (f, e, a) ev

write (f,e,a) ev

sendo f - das formas

arq
arq [e_1]
arq [line e_1]
arq [skip e_1]
arq [space e_1]
arq [stacker e_1]

ou

no

onde arq - identificador de arquivo;

e_1 - expressão aritmética;

e - expressão aritmética que indica o número de palavras a serem lidas ou impressas;

a - especificação de linha de array, expressão de tipo pointer ou variável subscritada, representando a área que será utilizada para entrada ou saída.

ev - designação de evento (que pode ser omitido) que será causado quando ocorrer o término da operação de entrada ou saída.

8.10 - Recursos para manipulação de bits e caracteres

(a) Tipo word, pointer e array de tipo word ;

(b) Manipulação através de indexação, especificação de palavras parciais e os comandos scan e replace.

8.11 - Processamento concorrente

8.11.1 Processos concorrentes - constituídos de procedimentos.

8.11.2 Criação de processos concorrentes - através do comando

fork p [e_1, e_2]

ou

fork p [e_1, e_2, e_3]

onde p é uma chamada de procedimento, como no ALGOL 60;

e_1, e_2, e_3 - expressões aritméticas, sendo e_1 o tamanho da pilha, e_2 a prioridade do processo e e_3 um índice para uma tabela de nomes de processos.

Este comando cria um processo independente dentro do sistema operacional.

8.11.3 Exclusão mútua - como no ALGOL B6700 (7.11.3), através dos comandos procure, liberate, fix e free, ou de forma mais elementar, através das funções "buzy" e "unlock", indivisíveis.

8.11.4 Sincronização e comunicação entre processos concorrentes.

Como no ALGOL B6700, através dos comandos set, reset, cause, wait, waitandreset e causeandreset.

8.11.5 Processamento de interrupções - tanto as interrupções definidas pelo programa, como as interrupções de erros de programa são processadas de mesma forma que no ALGOL B6700, com a inclusão das interrupções: inative queue, bottomofstack, sequence e stackunderflow que não eram processadas no ALGOL B6700.

8.12 - Acesso a características de máquina

Registradores e memória - através de vetores pré-declarados e especificação de endereços nas declarações de variáveis.

Operações de máquina - através de procedimentos intrínsecos.

8.13 - Comunicação com procedimentos em outras linguagens - permitida.

8.14 - Recursos em tempo de compilação - macros definidos através da declaração define, como no ALGOL B6700 (7.14).

8.15 - Recursos para "depuração" de programas - nenhum.

9. SPL - Systems Programming Language

A linguagem SPL foi desenvolvida pela Hewlett-Packard para o computador HP3000 [21, 22]. Destina-se ao desenvolvimento de compiladores, sistemas operacionais e subsistemas deste, tendo sido utilizada para a programação do "software" básico do HP3000, incluindo seu sistema operacional.

O HP3000 [23] é um computador com estrutura de pilha que é utilizada não só para chamada de procedimentos, mas também para o armazenamento de todas variáveis e temporários, bem como operandos durante a execução. As rotinas são reentrantes e os dados são alocados em área independente da rotina que os declarou. Existem os registradores DB (base de pilha), S (topo de pilha), Q (base local de procedimentos), Z (fim da área alocada para a pilha), PB (base do programa), PL (limite do programa), P (endereço da instrução atual) e X (indexador).

9.1 - Tipos

9.1.1 Tipos simples

t onde t ε{integer, real, logical (inteiro sem sinal), double (inteiro com sinal em palavra dupla), long (ponto flutuante de três palavras), byte}

ou

pointer t

9.1.2 Tipos estruturados

"array" - de uma dimensão;
- limites variáveis somente dentro de procedimentos.

9.1.3 Definição de novos tipos - não há.

9.2 - Declarações

9.2.1 Formas de declaração

(a) Variáveis

a t ℓ_1

ou

a t <u>array</u> ℓ_2

onde a ε { λ, <u>global</u>, <u>external</u> };

t - tipo simples;

ℓ_1 - lista de indentificadores, com eventuais inicializações e endereços relativos a registradores base ou a outras variáveis;

ℓ_2 - lista de identificadores, com especificação dos limites do índice e eventualmente valores iniciais e endereços relativos a registradores base ou a outras variáveis.

(b) Constantes - identificador associado a uma constante inteira ou "array" declarado com relação à base do código, devendo ter inicialização obrigatória.

(c) Registradores - identificadores, declarados com qualquer tipo simples, podem ser associados ao indexador X.

(d) <u>Define</u> - ver seção 9.14.

(e) Rótulo - identificadores podem ser declarados como rótulos.

(f) <u>Switch</u> - associação de um identificador a um conjunto de rótulos; mais restrito do que no ALGOL 60.

(g) Procedimentos, pontos de entrada e procedimentos intrínsecos - ver seção 9.7.

9.2.2 Localização de declaração: início do bloco que constitui o programa principal ou o corpo de um procedimento.

9.2.3 Inicialização - qualquer variável pode ser inicializada, excluídos "array's" com limites variáveis (locais a procedimentos).

9.2.4 Redefinição - através de especificação, na declaração, de endereço relativo a um registrador base, ou referência a uma variável já declarada.

9.2.5 Domínio da declaração - bloco (que só existe como nível mais externo de programas principais e procedimentos).

9.2.6 Pré-declarações.

TOS - variável associada ao topo da pilha.

9.3 - Alocação

9.3.1 Tipos de alocação - estática e dinâmica, como no ALGOL 60.

9.3.2 Memória e registradores.

Os "array's" são normalmente endereçados através de apontador guardado na área inicial da pilha correspondente ao bloco ou procedimento em que foram declarados.Pode-se declarar "array direto", isto é, endereçado sem apontador, perdendo-se então parte desse espaço inicial, que é limitado devido ao endereçamento.
Identificadores podem ser associados no indexador X.

9.3.3 Validade de apontadores - não é verificada.

9.4 - Constantes e variáveis

(a) Constantes

Constantes de tipos integer, real, double e long -re-

presentados nas base= 2 a 16.

Constantes lógicas - true e false;

Sequências de caracteres - entre aspas.

(b) Variáveis - representadas nas formas:

id_1 sendo id_1 um identificador declarado com tipo simples;

$id_2(e)$ sendo id_2 um identificador declarado como "array" ou pointer;

e expressão;

v indica o endereço da variável v, relativo ao início da base de dados;

absolute (e) indica a localização da memória de endereço e, sendo e uma expressão; esta forma pode ser utilizada somente no modo de execução privilegiado.

9.5 - Expressões

9.5.1 Formas

9.5.1.1 Expressões simples

Podem ser aritméticas, lógicas, operações sobre bits e condições (utilizadas somente nos comandos if, while e do until)

(a) Operandos - constante, variável, operação sobre bits, expressão (entre parênteses), comando de atribuição (entre parênteses) e chamada de função.
A chamada de função é de uma das formas:

f

ou

f(s, p)

onde f - identificador de procedimento ou subro-

tina;

s - lista da forma: *, *,..., * que indica que os parâmetros correspondentes estão na pilha de dados;

p - lista de parâmetros atuais, cada um dos quais podendo ser expressão ou rótulo.

(b) Operandos

(i) De expressões aritméticas:

+, -, *, /, ∧ (exponenciação)

/e/ - indica o valor absoluto da expressão e

(ii) De expressões lógicas

. Relacionais: <,> , =,<> , <=, >=

É permitida a forma $e_1 < e_2 < e_3$ onde e_1, e_2 e e_3 são expressões aritméticas.

. Lógicos: +, -, *, /, mod - operações lógicas de soma, subtração, multiplicação e resto em precisão simples;

**, //, modd - operações lógicas de multiplicação, divisão e resto em precisão estendida.

land, lor, xor (ou exclusivo) e not.

(iii) De operações sobre bits - deslocamentos ("shift"), extração de palavras parciais e concatenação.

(iv) Condições

. Relacionais - podem ser utilizados com operandos ou sem operandos, sendo que neste caso é feita verificação do código de condição da máquina.

. and e or-geram desvios, ao passo que land, lor e xor geram instruções lógicas.

. Testes de condições - carry, nocarry, overflow, noverflow, iabz, dabz, ixbz, dxbz. Estes operadores são utilizados sem operandos, para verificação de condições relacionadas a indicadores da máquina.

Precedências:

1a.) not ,
2a.) *, /, **, //, mod, modd
3a.) +, -
4a.) operadores relacionais
5a.) land
6a.) xor
7a.) lor e teste de intervalo da forma $e_1 < e_2 < e_3$

Nas condições, and tem precedência maior que or, sendo permitido apenas mais um nível com o operador or através da utilização de parênteses. Por exemplo, op_1 or op_2 and (op_3 or op_4) and op_5, sendo que $op_1, \ldots, op_5$ são expressões lógicas ou operadores de teste de condição . Não há restrições de nível para land e lor.

9.5.1.2 Expressões condicionais

if c then e_1 else e_2

onde c - condição

e_1, e_2 - expressões aritméticas ou lógicas.

9.5.2 Mistura de tipos e conversão

(a) não é permitida mistura de tipos em expressões aritméticas ou lógicas, exceto na exponenciação na qual

um valor real pode ser elevado a expoente inteiro.

(b) Em operações sobre bits os operandos podem ser de qualquer tipo aritmético ou lógico. Todos valores são considerados como sequências de bits.

9.6 - Comandos

9.6.1 Rótulo - múltiplos, representados por identificadores.

. Declaração - opcional.

. Definição - $c_1: c_2: \ldots c_n: S$

onde $c_1, \ldots, c_n$ são os rótulos;

S é um comando.

9.6.2 Bloco e comando composto

Bloco - não existe como comando (existe apenas no nível mais externo de programas principais e procedimentos).

Comando composto - como no ALGOL 60.

9.6.3 Atribuição - múltipla;

- o valor atribuído pode ser de tipo diferente, mas do mesmo comprimento das variáveis à esquerda do sinal de atribuição;
- pode haver atribuição a palavras parciais.

9.6.4 Estruturas de controle

9.6.4.1 Desvio incondicional ou escape

(a) go to r

ou

go r

onde r é um rótulo, ou das formas:

s(e) ou * s(e)

sendo s um identificador declarado como switch; e expressão que constitui o índice de um rótulo.
Na segunda forma não é efetuado teste para verificar se o índice é válido.

(b) return

ou

return n

Comando para retorno de procedimento, sendo n o número de níveis de procedimentos que serão abandonados.

9.6.4.2 Comandos seletivos

(a) if c then s_1
ou

if c then s_1 else s_2

onde c é uma condição;

s_1, s_2 são comandos.

(b) case e of c

ou

case * e of c

onde e é uma expressão de tipo integer, logical ou byte;

c é um comando composto.
Os comandos do comando composto c são enumerados a partir de 0. Na segunda forma do comando case não há teste para verificar se o valor da expressão é um índice válido.

9.6.4.3 Comandos repetitivos

(a) do s until c

(b) while c do S

(c) for id: = e_1 step e_2 until e_3 do S
ou
for* id: = e_1 step e_2 until e_3 do S

Nos comandos repetitivos acima:

s - é um comando;
c - é uma condição;
e_1, e_2, e_3 - expressões com valores inteiros;
id - identificador.

A parte step e_2 pode ser omitida, sendo considerada como step 1.

Na segunda forma do comando for, o incremento e teste é feito após a execução do comando S.

9.6.5 Chamada de procedimento e de subrotina

p
ou
p (s,a)
onde p - nome do procedimento ou subrotina;
s - lista da forma *,*,..., *, significando que os valores ou endereço dos parâmetros atuais estão na pilha;
a - lista de parâmetros atuais, onde cada um pode ser uma expressão ou rótulo.

9.6.6 Comando vazio - cadeia vazia de caracteres.

9.6.7 Outros

(a) Assemble (c_1; c_2;...; c_n)

onde c_1,..., c_n são instruções de máquina em linguagem simbólica.

(b) Delete: comandos correspondentes às instruções de máquina del, delb, ddel, do HP3000 [23], utilizados para retirar elementos do topo da pilha de dados.

(c) Move - move grupos de palavras ou bytes de uma área para outra. A forma move while permite a transferência de bytes enquanto estes forem numéricos ou alfanuméricos.

(d) Scan - efetua uma busca em uma cadeia de caracteres até que seja encontrado (ou enquanto for encontrado) um determinado caráter.

(e) Push (r_s,...,r_n) - empilha o conteúdo dos registradores r_1,..., r_n;
ou

Set (r_1,..., r_n) - carrega os registradores r_1,..., r_n com os valores contidos nas localizações do topo da pilha de dados;
sendo que r_1,..., $r_n \in$ {S, Q, X, STATUS, Z, DL,DB}.
O comando set com os registradores DB, DL, STATUS ou Z deve ser executado no modo privilegiado.

9.7 - Declaração de procedimentos e subrotinas

9.7.1 Formas da declaração

t procedure proc (p_1, p_2,..., p_n); v del op bc

t procedure proc; op bc
t subroutine s (p_1, p_2,...,p_n); v del c
t subroutine s; v c
intrinsic (f) $proc_1$, $proc_2$,..., $proc_n$
intrinsic $proc_1$, $proc_2$,..., $proc_n$
entry r_1, r_2,... r_n

onde t - tipo do valor retornado pelo procedimento, quando este é utilizado como função.

t $\in$ {λ, integer, logical, byte, real, long , double};

proc - identificador do procedimento ;

s - identificador de subrotina;

p_1,...,p_n - identificadores dos parâmetros formais;

v - lista dos parâmetros com modo de passagem por valor. Pode ser omitida, ou da forma;

value $id_1, \ldots, id_m$;

sendo $id_1, \ldots, id_m$ - identificadores de parâmetros formais.

dcl - lista de declarações de tipo dos parâmetros formais;

op - lista de opções, que pode ser omitida, ou especificada da forma;

option $o_1, o_2, \ldots, o_k$;

sendo que $o_1, \ldots, o_n \in$ {uncallable, privileged, external, check, variable, forward, interrupt, internal} ;

c - comando;

bc - comando ou bloco (ver seção 9.8) que são omitidos - quando o procedimento é declarado com a opção external ou forward.

f - nome de um arquivo que constitui uma biblioteca de procedimentos intrínsecos (pré-declarados);

$proc_1, \ldots, proc_n$ - identificadores de procedimentos intrínseco;

$r_1, \ldots, r_n$ - rótulos associados a comandos do procedimento e que através da declaração entry irão constituir pontos de entrada do procedimento.

Dentro de um procedimento não são permitidas declarações de outros procedimentos, exceto aqueles declarados como intrinsic ou external. Note-se que, assim, um único registrador base (Q) é suficiente para endereçamento das variáveis locais a procedimentos, pois no máximo dois níveis podem estar ativos: o programa principal e o procedimento que está sendo executado no momento.

Subrotinas distinguem-se de procedimentos por não possuirem declarações de dados, procedimentos ou su-

brotinas locais. Devido ao fato de não conterem variáveis locais, declarações de subrotinas podem ocorrer dentro de procedimentos. O endereçamento dos parâmetros de subrotinas é feito em relação ao topo da pilha.
A declaração intrinsic tem a vantagem de permitir ao compilador obter procedimentos do arquivo de intrínsecos, não havendo necessidade de declaração do valor retornado e dos parâmetros.

9.7.2 Parâmetros

(a) Tipos de parâmetros formais - variáveis simples, identificadores de array (eventualmente com dimensões variáveis), label e procedimentos.

(b) Formas de passagem de parâmetros: por valor e por referência. Expressões devem ser passadas por valor.

(c) Associação entre parâmetros formais e atuais.
Os parâmetros formais e atuais não precisam ser dos mesmos tipos, havendo verificação da correspondência de tipos somente quando for especificada a opção check na declaração do procedimento.

9.7.3 Retorno de valor em funções o valor retornado é definido através de um comando de atribuição cujo lado esquerdo seja o nome do procedimento ou subrotina.

9.7.4 Dados acessíveis a procedimentos

(a) Subrotinas - têm acesso a dados globais (somente do programa principal) e valores passados como parâmetros, não possuindo declarações de dados locais.

(b) Procedimentos - têm acesso a dados locais, globais (somente do programa principal) e parâmetros.

9.7.5 Recursão - subrotinas e procedimentos podem ser recursivos

9.7.6 Procedimentos externos - aqueles declarados com a opção external, com corpo de procedimento vazio,e os que constarem em arquivos de procedimentos intrínsecos, introduzidos através da declaração intrinsic.
Além dos valores passados como parâmetros, um procedimento externo pode utilizar variáveis declaradas como global no programa principal. Para isto, tais variáveis deverão ser declaradas como external no procedimento (ver seção 9.2.1).

9.7.7 Efeitos colaterais em funções - podem ocorrer por alteração de parâmetros e de variáveis globais.

9.7.8 Procedimentos pré-declarados-não existem na definição apresentada no manual de referência do SPL [21]. A linguagem permite acesso a procedimentos pré-declarados desde que seus nomes apareçam em uma declaração intrinsic ou que sejam declarados no programa com a opção external.

9.8 - Programa - é um bloco definido da forma:

begin dcl proc c_1; c_2;...; c_n end

onde c_1,..., c_n são comandos;

dcl são declarações de dados;
proc são declarações de procedimentos;

9.9 - Recursos de entrada e saída nenhum, essas operações são feitas por meio de chamadas a procedimentos intrínsecos.

9.10- Recursos para manipulação de bits e caracteres.

(a) Declarações de variáveis simples ou array com o tipo byte.

(b) Constantes nas bases 2 a 16 e sequências de caracteres.

(c) Comando de atribuição que permite acesso a palavras

parciais (grupos de bits de uma palavra) e operações de "shift" e concatenação.

(d) Comandos move e scan.

9.11 - Processamento concorrente - pode ser feito através de procedimentos intrínsecos do sistema operacional.

9.12 - Acesso a características de máquina

(a) Acesso a qualquer posição da memória através da variável absolute (i), onde i é um endereço absoluto permitido somente a programas com processamento no modo privilegiado).

(b) Dados alocados em relação a um dos registradores base, DB, Q ou S, segundo escolha do programador.

(c) Acesso às instruções de máquina através do comando assemble.

(d) Operadores correspondendo a instruções de máquina, tais como: de deslocamento ("shift"), extração de palavras parciais e operações lógicas.

(e) Comandos que correspondem a instruções de máquina:
Push e set - empilham ou atribuem valores aos registradores base.
move e scan - para manipulação de caracteres.
del, delb e ddel - retiram valores do topo da pilha de dados.

9.13 - Comunicação com procedimentos em outras linguagens - é permitida.

9.14 - Recursos em tempo de compilação.

(a) Declaração de macros:

define m = S #

onde

m - identificador da macro;

S - sequência de caracteres não contendo"#".

(b) Utilização - no texto do programa fonte, cada ocorrência do identificador m será substituída pela sequência S, após o qual o texto substituído será compilado. Não há parâmetros na macro.

9.15 - Recursos para depuração de programas - nenhum.

CAPÍTULO IV

CRITÉRIOS GERAIS NO DESENVOLVIMENTO DE LINGUAGENS DE PROGRAMAÇÃO DE SISTEMAS

Diversos autores, entre os quais Bergeron [1,2], Horning e Clark [20], Sammet [41] e Seegmüller [42] mencionam critérios a serem considerados no desenvolvimento de linguagens de programação de sistemas. Apresentamos neste capítulo uma compilação desses critérios juntamente com as características das linguagens que, em nossa opinião, visam satisfaze-los e quais, entre as linguagens estudadas, as que apresentam tais características.

1. A estrutura geral da linguagem deve ter por base a do ALGOL 60 ou outra linguagem de alto-nível de ampla utilização.

   Todas as linguagens estudadas no capítulo III têm o aspecto geral do ALGOL 60, com restrições e recursos adicionais.

2. Modularidade - as características das linguagens que em nossa opinião contribuem para a modularidade de programas são :

   (a) Blocos - existem nas linguagens PL/I, ALGOL B6700, ESPOL e PL360, sendo que nesta última a alocação dos dados de blocos é estática.

   (b) Comandos: if, case, for, repeat, while e comando composto- são comuns à todas as linguagens, com pequenas variações e exceções.

   (c) Procedimentos - existem em todas as linguagens, com muitas variações. De um ponto de vista prático, é desejável que seja permitida a codificação e chamada de procedimentos externos. Na fase de codificação e testes de grandes sistemas, as compilações de procedimentos separadamente significam uma grande economia de tempo de utilização do computador desde que cada vez que um procedimento externo é alterado não é necessário recompilar todo o sistema. Além disso, um procedimento externo po-

de ser chamado por vários usuários, sem que se necessite incluir o código fonte no programa e se este código for reentrante, uma única cópia do mesmo poderá ser mantida na memória para ser executada por mais de um usuário ao mesmo tempo.

(d) Ausência de formas indesejáveis de desvios, tal como go to para fora de um bloco (denominado "bad go to").Somente as linguagens BLISS e PASCAL C não possuem o comando go to. Nas demais linguagens nenhuma restrição foi introduzida no go to para evitar essas formas indesejáveis de desvios.

Na linguagem BLISS como foi visto na seção 2 do capítulo III, a ausência do comando go to é compensada pela inclusão de diversas formas de comando exit para escape de cada uma das classes de comandos (repetitivos, seletivos, blocos, etc) e de procedimentos, e pela inclusão de muitas variações de comandos repetitivos e seletivos. Em nossa opinião, é suficiente e mais conveniente uma única forma de comando exit, como existe na linguagem LAPA (ver capítulo V, seção 2.6.4.1), que utiliza um rótulo para indicar o comando que será abandonado, o que evita a especificação do número de níveis de blocos que serão abandonados, como ocorre no BLISS.

3. Segurança - a linguagem deve possibilitar a verificação durante a compilação do maior número possível de erros sintáticos e semânticos. A seguir, enumeramos aqueles que consideramos como principais problemas que afetam a segurança de um programa e que podem ser verificados durante a compilação.

(a) Correspondência de tipos entre parâmetros atuais e formais em chamadas de procedimentos - existe verificação nas linguagens PL/I (opcional), PASCAL C, ALGOL B6700 , ESPOL e SPL (opcional). Notamos que nas linguagens ALGOL B6700, ESPOL e SPL, quando o procedimento chamado é parâmetro formal de outro procedimento, não é possível e-

fetuar esta verificação durante a compilação pois não são declarados os tipos de seus parâmetros formais; como é o caso de, por exemplo, o ALGOL 68 [28].

(b) Efeitos colaterais na execução de funções - são restringidos apenas no PASCAL C. Nesta linguagem os parâmetros de funções são passados "por valor" para evitar sua alteração.

(c) "Dangling pointers"- este problema ocorre quando o endereço de uma área local a um bloco é atribuído a uma variável de tipo pointer global a este bloco. Após a saída do bloco o apontador poderá fazer referência a uma área já liberada ou realocada a outro bloco. Este problema não é verificado em nenhuma das linguagens estudadas, a menos de uma versão mais recente do compilador ALGOL B6700, existindo esta verificação no ALGOL 68 [28].

(d) Exclusão mútua no acesso a variáveis compartilhadas por processos concorrentes. Existem várias propostas de soluções ao problema de exclusão mútua [6,7,10,17,18,19].As linguagens PL/I, ALGOL B6700 e ESPOL possuem operações primitivas (ver seções 3.11.3, 7.11.3 e 8.11.3 do capítulo III) que quando utilizadas de forma conveniente garantem a exclusão mútua no acesso a variáveis compartilhadas por processos concorrentes. Contudo não é possível ao compilador verificar se essas operações primitivas foram utilizadas da forma correta para garantir a exclusão mútua. Uma solução melhor deste problema foi adotada pelo PASCAL C, através da construção monitor. Um monitor permite acesso às variáveis declaradas no mesmo por processos concorrentes, garantindo a exclusão mútua nestes acessos, sendo esta a única forma de processos concorrentes compartilharem dados.

(e) "Deadlock" - ocorre em processamento concorrente quando dois ou mais processos se bloqueiam mutuamente quando cada um dos processos envolvidos detem o controle de recursos do sistema (dispositivos de entrada e saída, informa

ções, etc) e está em espera de recursos que estão com os outros processos. Uma situação de "deadlock" pode ocorrer no ALGOL B6700, por exemplo, pelo uso incorreto das primitivas procure e liberate (ver seção 7.11.3 do capítulo III). Somente no PASCAL C é possível uma prevenção parcial de "deadlock" (ver capítulo III, seção 6.11.3). Lister [29] menciona situações em que podem ocorrer "deadlock" quando são efetuadas chamadas encaixadas de monitores em PASCAL C.

4. Eficiência do código gerado -podemos citar os seguintes aspectos no desenvolvimento de linguagens de programação de sistemas que afetam a eficiência do código:

   (a) A linguagem não deve conter construções poderosas demais cuja implementação seja ineficiente na máquina à qual a linguagem se destina. Entre tais construções, as seguintes constituem pontos críticos em relação à eficiência:

      (i) Matrizes com limites variáveis durante a execução. A alocação destas matrizes envolve um certo gasto em tempo de execução, em cada ativação do bloco em que a matriz foi declarada. À exceção de PL/I, ALGOL B6700 e ESPOL, as demais linguagens admitem apenas matrizes com limites fixos para permitir uma implementação eficiente da entrada de blocos.

      (ii) Procedimentos encaixados - No ALGOL 60 são permitidas declarações de procedimentos dentro de procedimentos. A cada um destes procedimentos está associada uma área de dados sendo que em um determinado instante, dependendo da sequência de chamada, várias destas áreas podem estar ativas e serem acessíveis ao último dos procedimentos chamados. O endereçamento dessas áreas pode ser feito de forma eficiente se a máquina possuir um número "razoável" de registradores base ("displays" no B6700), utilizando-se um registrador base para endereçamento da

área de dados de cada nível de procedimentos , Se houver apenas um registrador base para endereçamento da área de dados de procedimentos, o acesso a variáveis dos procedimentos de níveis inferiores em relação ao último procedimento chamado é possível mas ineficiente, como ocorre no BLISS implementado no PDP 10.

Nas linguagens PASCAL C, SPL e XPL não são permitidas declarações encaixadas de procedimentos, sendo que no BLISS existe a declaração de rotina no qual não se permite acesso a variáveis globais e chamadas de procedimentos. Nessas condições é suficiente um único registrador para base de área de dados de procedimentos pois em cada instante somente o último nível é acessível.

Com a introdução das restrições indicadas no itens (i) e (ii) é possível a implementação de blocos encaixados e de procedimentos recursivos de forma eficiente com um único registrador para base de dados de procedimentos, além do registrador base de dados do programa.

(b) A linguagem deve incluir construções que explorem as instruções especiais eventualmente existentes na máquina e que correspondam a operações frequentemente utilizadas em programação de sistemas. Desta forma, ao mesmo tempo que atingimos o critério de eficiência do código objeto, obtemos também a simplicidade do código fonte. Como exemplo citamos os comandos scan e move/replace, existente no SPL, ALGOL B6700 e ESPOL, que efetuam busca e movimento de caracteres de forma mais eficiente do que através dos demais comandos da linguagem,explorando instruções de máquina que executam essas operações.

(c) Os testes para detecção de erros durante a execução devem ser opcionais desde que impliquem na inserção de código extra, segundo escolha do programador durante a compilação, a menos dos testes efetuados automaticamente pe

la máquina.

5. Extensibilidade

Entre os recursos das linguagens estudadas consideremos que os seguintes relacionam-se à extensibilidade das linguagens:

(a) Recursos em tempo de compilação - macros existentes no PL/I, ALGOL B6700, ESPOL e SPL. Na primeira os recursos de pré-processamento são mais poderosos, permitindo declarações de variáveis e de procedimentos e comandos de controle, constituindo praticamente uma linguagem de pré-processamento que exige um passo extra para pré-compilação. Notamos que nessa situação o pré-processador poderia ser um programa independente, servindo ao pré-processamento de programas em outras linguagens, como ocorre no MAPPA [46] - Macro Processador do PADE.

(b) Definição de novos tipos - existe no PASCAL S e PASCAL C.

Um recurso que incluiríamos no critério de extensibilidade e que não verificamos em nenhuma das linguagens estudadas é a definição de operadores, existente no ALGOL 68 [28].

Os recursos mencionados nesta seção aumentam o poder de programação da linguagem sem afetar a eficiência do código gerado.

6. Aplicabilidade - as aplicações particulares da linguagem irão exigir características adicionais.

(a) A maioria das aplicações em programação de sistemas exigem da linguagem recursos flexíveis para definição e manipulação de estruturas de dados. São necessários os tipos inteiro, booleano, caráter (ou byte), palavra (sem interpretação de tipo) e apontadores para manipulação de listas e áreas de dados.

São necessários ainda tipos estruturados que facilitem a definição de tabelas, pilhas, filas e listas ligadas. Nes

te sentido o mínimo necessários é a declaração de vetor ("array" de uma única dimensão) sendo desejável tipos análogos ao record, existente no PASCAL (ambos) e PL/I . Verificamos que os tipos caráter e apontador estão presentes na maioria das linguagens estudadas, com exceção do XPL e PASCAL C. No PL360 e BLISS não existe declaração de apontador mas é possível a manipulação de endereços existindo a operação de referência (obtenção do endereço de uma variável).

Para manipulação desses tipos encontramos nas diversas linguagens além das operações básicas(aritméticas,booleanas e relacionais), operações de concatenação de sequências de bits ou caracteres, deslocamentos ("shift"), extração de palavras parciais (grupos de bits de uma palavra) e comandos especiais tais como scan e move/replace.

(b) Desenvolvimento de sistemas operacionais - seus principais componentes efetuam as seguintes funções:

(i) Manipulação de interrupções;
(ii) Administração de processador;
(iii) Administração de memória;
(iv) Administração de dispositivos de entrada e saída ;
(v) Administração de informações (arquivos).

Para a programação destas componentes é necessário que a linguagem permita acesso aos recursos da máquina tais como registradores, localizações absolutas de memória e instruções de máquina. Verificamos que as linguagens PL360, BLISS, XPL, ESPOL e SPL permitem acesso a estes recursos. Os demais componentes do sistema operacional utilizam as funções mencionadas nos itens (i) a (v), para efetuar criação e manipulação de processos, requisição e devolução de memória, operações de entrada e saída, etc.

Das linguagens estudadas, o PL/I, PASCAL C, ALGOL B6700 e ESPOL possuem recursos para manipular processos; PL/I , XPL, PASCAL S, ALGOL B6700 e ESPOL possuem recursos para entrada e saída e PL/I, PASCAL S, PASCAL C e ALGOL B6700 possuem comandos ou procedimentos para alocação de áreas

de dados. No SPL estas funções são efetuadas por procedimentos intrínsecos do sistema (ver capítulo III, seção 9.7.1) que podem ser chamados pelo programa em SPL.
Observamos que a linguagem PASCAL C definida para programação de sistemas operacionais não permite acesso aos recursos da máquina, sendo dependente de um núcleo, escrito em linguagem de máquina, que efetua as funções dos itens (i) a (v).

(c) Desenvolvimento de compiladores e interpretadores - além das facilidades para definição e manipulação de estruturas de dados são convenientes procedimentos recursivos , para implementação de analisadores sintáticos,que utilizam métodos recursivos (como por exemplo o método Recursivo Descendente) e o comando go to para implementação direta de analisador sintático através de produções de Floyd-Evans traduzidas para comandos da linguagem de programação de sistemas.
Podemos verificar que a maioria das linguagens estudadas possuem estas características, sendo adequadas para a programação de compiladores e interpretadores. A linguagem XPL, desenvolvida especificamente para a programação de compiladores , não possui procedimentos recursivos o que limita sua utilização com métodos de análise sintática não recursivos. Na realidade a principal utilização da linguagem XPL tem sido com o "Translator Writing System" [31] cujo método de análise sintática é de "estratégia de precedência mista", usualmente implementado em algoritmo não recursivo.

(d) Programação das bibliotecas de funções matemáticas utilizadas por outras linguagens - exige da linguagem de programação de sistemas o tipo real, além do tipo integer , bem como as operações entre reais.
Verificamos que todas as linguagens estudadas, com exceção do XPL possuem o tipo real e as operações entre reais.

7. Independência do suporte de rotinas em tempo de execução.

A linguagem não deve requerer o suporte de rotinas em tempo de execução.
As linguagens PL/I e XPL são as que mais fazem uso de rotinas em tempo de execução e o BLISS e PL360 não necessitam o suporte de tais rotinas.

8. Testes e depuração de programas

A linguagem deve possuir recursos para testes e depuração de programas para facilitar a localização de erros que não foram detectados durante a compilação. Nas linguagens estudadas verificamos que enquanto algumas possuem recursos para depuração de programas (PL/I, XPL e ALGOL B6700) em outras tais recursos são introduzidos como opção de compilação não existindo construções especiais na linguagem para este fim.

9. Clareza - através de uma simples leitura do programa fonte deve ser possível extrair o máximo de informações sobre as propriedades do programa. Em geral, todas as linguagens estudadas, tendo as estruturas básicas do ALGOL 60 são de leitura relativamente fácil. Em nossa opinião, as linguagens PASCAL S e PASCAL C são as que mais satisfazem a este critério, o que torna mais simples o seu ensino.

10. Portabilidade - a principal exigência para que os programas escritos em uma linguagem sejam portáteis é que esta não tenha recursos dependentes de máquina. Das linguagens estudadas, o PL 360, ESPOL e SPL são as que apresentam maior dependência da máquina para a qual foram definidas, refletindo em suas estruturas muitas características de sua arquitetura.
Este critério entra em conflito com as características necessárias para o desenvolvimento de sistemas operacionais (seção 6, ítem (6)) qual seja, o acesso a características de máquina. Nesta situação podemos deixar de lado a portabilidade mesmo porque no caso de sistemas operacionais esta não é essencial.

CAPÍTULO V

APRESENTAÇÃO DA LINGUAGEM LAPA DE PROGRAMAÇÃO DE SISTEMAS

1. INTRODUÇÃO

A LAPA - Linguagem Algorítmica do PADE - foi definida para ser a única linguagem de alto-nível do computador PADE [33, 34]. Esta linguagem tem como objetivos a programação científica e a programação de sistemas, destinando-se à produção de todo o "software" do computador.

Uma primeira descrição, que iremos denominar LAPA sequencial, foi apresentada em [43] e que será descrita neste trabalho foi projetada por V.W. Setzer com a colaboração de C.Z. Mammana, W.P. Paulo Filho, L.M.M. Lago e G. Bressan. Esta versão sofreu acréscimos de construções propostas por W.P. Paula Filho para processamento concorrente resultando na versão que denominamos LAPA concorrente. Tais construções acompanham as idéias de P. Brinch Hansen [7,8,10] introduzidas na linguagem PASCAL C.

Na definição da LAPA sequencial existiram duas principais orientações. A primeira, visando o provimento de recursos para a programação de sistemas, exigia que a linguagem fosse aderente à arquitetura, o que permitiria a implementação de um compilador que produzisse código eficiente e compacto, e que permitisse acesso aos recursos da máquina. A segunda orientação pretendia suprir a linguagem com recursos que permitissem a plena utilização dos conceitos de programação estruturada e que não exigissem um esforço exagerado para seu aprendizado ; por outro lado, a tentativa de ser esta a única linguagem do computador forçava uma amplitude que cobrisse todo o espectro da computação desejável.

Na seção 2 apresentamos uma descrição da LAPA sequencial através das mesmos itens mencionados no capítulo II sendo que um diagrama sintático da LAPA encontra-se como apêndice des

te trabalho.

A seguir apresentamos um resumo das principais características do mini-computador PADE:

(a) Palavra - de 24 bits, podendo ser subdividida em grupos de bits de comprimento 1,2,3,4,6,8 e 12. Cada uma destas subdivisões denomina-se "ion" e o número de bits em um ion é denominado "valência".

(b) Registradores

8 registradores que podem ser utilizados para endereçamento, operações aritméticas e como apontadores de pilha.

2 registradores especiais para base da área de dados e da área de código do programa, denominados β e β' respectivamente.

1 registrador base local, denominado γ, utilizado para endereçamento da área de dados de procedimentos, sendo relativo a β.

(c) Instruções - além de instruções convencionais , o PADE contém: instruções especiais de pilha que podem ser utilizadas para transmissão de parâmetros e ligação entre procedimentos; instruções denominadas "ionicas" que efetuam operações entre ions; instruções de controle de sequência - que repetem um grupo de instruções o número de vezes que for especificado em um contador e instruções denominadas "polakas" que executam expressões em notação polonesa com até cinco símbolos, incluindo operando, operadores unários e binários, compactados em uma só palavra.

2. Descrição da LAPA sequencial

2.1 - Tipos

2.1.1 Tipos simples

integer, real, alpha - palavra de 24 bits.

ion n - grupo de bits de comprimento n, onde n é um divisor de 24.

unpacked ion n - ion de comprimento n alinhado à direita em uma palavra.

Boolean - equivalente a ion 1.

char - equivalente a ion 6 ou ion 8, dependendo da implementação (ASCII ou EBCDIC).

pointer ou pointer(t) - sendo t um tipo simples ou estruturado correspondendo ao tipo dos valores apontados.

2.1.2 Tipos estruturados

array - uma ou mais dimensões,
limites constantes,
elementos de tipo simples ou record.

record - constituídos de várias componentes de tipos diferentes, simples ou estruturados.

2.1.3 Definição de novos tipos - um identificador pode ser associado a uma descrição de record, constituindo um novo tipo.

2.2 - Declarações

2.2.1 Formas de declaração

(a) Variável

(i) t id_1, id_2,..., id_n

(ii) [i_1: s_1, i_2: s_2,..., i_m: s_m] t id_1, id_2,..., id_n

(iii) record (ℓ) id_1, id_2,..., id_n

(iv) register id_1, id_2,..., id_n

sendo id_1, id_2,..., id_n - identificadores definidos pela declaração.

t - tipo simples ou estruturado

i_1,..., i_m, s_1,..., s_m - constantes indicando os limites inferiores e superiores dos índices de "array's".

ℓ - lista de declarações de componentes, da forma f_1; f_2;...; f_k

sendo que cada uma das declarações f_i pode ser das formas (i), (ii) ou (iii) ou então ($\ell_1/\ell_2/.../\ell_r$) onde ℓ_j é uma lista de declarações de componentes (ver 2.2.4).

(b) Tipo

record id (ℓ)

sendo id - identificador associado à descrição do record

ℓ - lista de declarações de componentes, como a do ítem (a).

2.2.2 Localização da declaração - início de bloco.

2.2.3 Inicialização de variáveis - não há.

2.2.4 Redefinição - ocorre em declarações de record através da descrição de componentes da forma (ℓ_1/ℓ_2 .../ ℓ_r).

Nesta declaração, $\ell_2, \ldots, \ell_r$ são redefinições da componente ℓ_1.

2.2.5 Domínio da declaração - bloco em que ocorreu a declaração.

2.2.6 Pré-declarações - nenhuma.

2.3 - Alocação

2.3.1 Tipos de alocação

(a) Estática - variáveis declaradas no bloco mais externo do programa principal.

(b) Dinâmica - variáveis declaradas nos blocos, excluído o mais externo do programa principal e incluíndo quaisquer blocos de procedimentos.

2.3.2 Meio

(a) Registrador - variáveis declaradas sem tipo, como register são alocadas em um dos 8 registradores de uso geral.

(b) Memória - demais variáveis não declaradas como register.

2.3.3 Validade de apontadores - não há verificação se um valor de apontador é válido, isto é, se a área apontada está ativa.

2.4 - Constantes e variáveis

(a) Constantes

- números decimais inteiros ou reais, com ou sem expoente.

- sequências de dígitos nas bases: binária, octal e hexadecimal.
- sequências lógicas - true e false (representadas internamente pelos valores -1 e 0).
- constante de tipo pointer - nil.

(b) Variáveis

As seguintes formas constituem variáveis de um tipo t:

id

v_1 [e_1,..., e_n]

v_2 ↑ ou v_3↑ t

v_4 . id

sendo t - tipo simples ou estruturado;

id - identificador declarado como de tipo t;

e_1,..., e_n - expressões com valores inteiros;

v_1 - variável de tipo "array" cujos elementos são de tipo t;

v_2 - variável de tipo pointer , com especificação do tipo t na declaração;

v_3 - variável de tipo pointer;

v_4 - variável de tipo record.

2.5 - Expressões

2.5.1 Formas

2.5.1.1 Expressões simples

(a) Operandos

. constantes,
. variáveis
. chamada de função

$f(a_1, \ldots, a_n)$ ou f

onde f - identificador da função

$a_1, \ldots, a_n$ - expressões.

(b) Operadores e precedência dos mesmos

. 1º nível (precedência maior) - operadore unários

fabr - valor absoluto em ponto flutuante

not - complemento para 1

incr - incrementa 1

decr - decrementa 1

halve- divide por 2

dup - dobra

neg - complemento para 2 do valor absoluto

abs - valor absoluto

\- - complemento para 2

fneg - menos o valor absoluto em ponto flutuante

sign - testa sinal

loadtrue - carrega -1

minus- complemento em ponto flutuante

norm - normalização

. 2º nível

** - exponenciação

cat - concatenação de sequências de caracteres

. 3º nível - operadores multiplicativos

* - multiplicação inteira

/ - divisão inteira

fmul - multiplicação em ponto flutuante

fdiv - divisão em ponto flutuante

mul - multiplicação fracionária

mod - resto de divisão

and - e

shl - deslocamento ("shift") lógico

sha - deslocamento ("shift") aritmético

. 4º nível - operadores aditivos

\+ - adição

\- - subtração

fadd- adição em ponto flutuante

fsub- subtração em ponto flutuante

ladd- adição lógica

lsub- subtração lógica

or - ou

xor - ou exclusivo

. 5º nível - operadores relacionais

>, >=, <, <=, =, neq

2.5.1.2 Expressões condicionais e seletivas

(a) if e_1 then e_2 else e_3

Efeito: se $e_1 \neq$ zero então e_2 senão e_3

(b) case e of $S_1 \rightarrow e_1$, $S_2 \rightarrow e_2$, ..., $S_n \rightarrow e_n$;
out e_{n+1}

Efeito: se existe k ($1 \leq k \leq n$) tal que

$S_k = e$ e

$S_i \neq e$ para $1 \leq i < k$

então e_k senão e_{n+1}

(c) index e of $d_1 \rightarrow e_1$, $d_2 \rightarrow e_2$,..., $d_n \rightarrow e_n$,
out e_{n+1}

Efeito: se existe k($1 \leq k \leq n$) tal que

$d_k = e$

$d_i \neq e$ para $1 \leq i < k$

então e_k

senão e_{n+1}

Nas expressões condicionais e seletivas acima, e, S_1,..., S_n - são expressões de tipo simples,

e_1, e_2,..., e_{n+1} - são expressões,

d_1, d_2,..., d_n - são constantes inteiras.

2.5.2 Mistura de tipos e conversão

Os operandos de uma expressão devem ser do mesmo tipo. As conversões devem ser feitas explicitamente através de funções a serem definidas com este fim.

2.6 - Comandos

2.6.1 Rótulos

Definição: r:c
sendo r - o identificador do rótulo;
c - um comando sem rótulo.

Utilização: comando exit r (2.6.4.1)
onde r foi definido como rótulo.

Domínio: comando ao qual o rótulo está associado.

2.6.2 Bloco e comando composto

(a) Bloco -

Begin d_1; d_2;...; d_n; c_1;...; c_m end

onde d_1,..., d_n - é uma lista, possivelmente vazia de declarações de variáveis, tipos, procedimentos ou funções.

c_1,..., c_n - lista de comandos, com ou sem rótulos.

(b) Comando composto - não há.

2.6.3 Atribuição -

v:= e ou v_1:= ... v_n:= e

sendo v, v_1,..., v_n - variáveis de tipos simples ou estruturados.

Se a variável e a expressão forem de tipos diferentes, a atribuição será efetuada sem conversão.

2.6.4 Estruturas de controle

2.6.4.1 Desvio incondicional ou escape -

(a) exit r onde r - é o rótulo de um comando que contém o comando exit.

Efeito - término do comando cujo rótulo é r.

(b) exit p onde p - é um identificador declarado como procedimento.

Efeito - término do procedimento p.

(c) exit f with e

onde f - é um identificador declarado como função.

e - é uma expressão.

Efeito - término da função f devolvendo a expressão e como resultado.

2.6.4.2 Comandos seletivos

(a) if e then c_1 else c_2.

Efeito: se e ≠ zero então c_1, senão c_2.

(b) when e then c_1

Efeito: se e ≠ zero então c_1.

(c) case e of $S_1 \rightarrow c_1$, $S_2 \rightarrow c_2$, ..., $S_n \rightarrow c_n$, out c_{n+1}

Efeito: se existe k ($1 \leq k \leq n$) tal que

S_k = e e

$S_i \neq$ e para $1 \leq i < k$

então c_k senão c_{n+1}

(d) index e of $d_1 \rightarrow c_1$, $d_2 \rightarrow c_2$, ..., $d_n \rightarrow c_n$, out c_{n+1}

Efeito: se existe k ($1 \leq k \leq n$) tal que

d_k = e e

$d_i \neq$ e para $1 \leq i < k$

então c_k senão c_{n+1}

Nos comandos acima,

e, S_1, ..., S_n são expressões de tipo simples.

d_1, ..., d_n são constantes inteiras.

c_1, ..., c_n são comandos.

2.6.4.3 Comandos repetitivos

(a) repeat c until e

equivale a: c; when e then repeat c until e

(b) repeat c forever

equivale a: repeat c until false.

(c) while e do c

equivale a: when e then repeat c until e

(d) for id from e_1 to e_2 by e_3 do c

. Qualquer uma ou mais das partes for id, from e_1, to e_2, by e_3 pode ser omitida sendo assumidos os valores: $e_1 = 1$, $e_2 = \infty$, $e_3 = 1$ respectivamente.

. Os valores de e_1, e_2 e e_3 são calculados apenas uma vez em cada execução do comando for.

. O identificador id, quando a parte for id estiver presente, é considerado declarado com tipo integer.

Nos comandos acima,

e, e_1, e_2 e e_3 são expressões;
c é um comando;
id é um identificador.

2.6.5 Chamada de procedimentos

$p(a_1, a_2, \ldots, a_n)$ ou p

onde p é o nome do procedimento
$a_1, \ldots, a_n$ - expressões.

2.6.6 Comando nulo - cadeia vazia.

2.6.7 Outros.

(a) comandos de entrada e saída - ver seção 2.9.

(b) comando assemble.

assemble $(a_1; a_2; \ldots; a_n)$

onde $a_1, \ldots, a_n$ são instruções da linguagem de montagem do PADE (ref. [36]).

2.7 - Declaração de função ou procedimento

2.7.1 Forma da declaração

procedure p:c

procedure (p_1, p_2,..., p_n) p:c

ts function f:c

ts function (p_1, p_2,..., p_n) f:c

sendo p - identificador que constitui o nome do procedimento;

f - identificador que constitui o nome da função,

ts - tipo simples;

c - corpo do procedimento ou função consistindo de um comando ou das palavras reservadas external ou forward;

p_1,..., p_n - lista de parâmetros formais em que p; é da forma modo t id_1,... id_m.

onde modo ε {value, result, value result, reference, λ};

t - tipo simples ou estruturado,

id_1,..., id_m - identificadores dos parâmetros formais.

Não são permitidas declarações de funções ou procedimentos no corpo de funções e de procedimentos.

2.7.2 Parâmetros

(a) Tipos dos parâmetros formais.

Parâmetros formais podem ser variáveis simples ou estruturadas.

(b) Formas de passagem de parâmetros.

Por valor, por resultado, por valor e resultado e por referência. Os parâmetros de funções devem ser

passados por valor.

(c) Associação de parâmetros atuais com os formais e verificação da correspondência.
O tipo do parâmetro atual deve ser o mesmo do parâmetro formal correspondente. Se um parâmetro atual é constante ou expressão envolvendo operadores ou funções, o parâmetro formal correspondente deverá especificar o modo de passagem value (por valor). A verificação da correspondência é feita durante a compilação.

2.7.3 Retôrno de valor em funções - através do comando exit ... with (ver 2.6.4.1, ítem (c)) que deve aparecer, ao menos uma vez, no bloco que constitui o corpo da função.

2.7.4 Dados acessíveis a procedimentos e funções: globais e locais.

2.7.5 Recursão - permitida.

2.7.6 Procedimentos e funções externas - Procedimentos e funções externas devem ser declaradas como external, no programa que os utiliza.

2.7.7 Efeitos colaterais em funções - não existem restrições.

2.7.8 Procedimentos e funções pré-declarados - nenhum.

2.8 - Programa

É constituído de um comando (com ou sem rótulo) ou de uma declaração de procedimento ou função.

2.9 - Recursos de entrada e saída

Comandos:

read arq (i_1,..., i_n) form

read arq form

write arq(o_1,..., o_n) form

write arq form

sendo que arq - especificação de arquivo que pode ser omitida ou, quando presente, é da forma: file f, onde f é o nome do arquivo.

form - especificação de formato que pode ser omitida ou, quando presente, é da forma: form exp, onde exp é uma expressão resultando uma sequência de caracteres interpretados como formatos de edição (ver ref [5]).

$i_1,...,i_n$ - lista de entrada constituída de variáveis ou especificações de repetição com sintaxe análoga à do comando for (2.6.4.3) no qual o comando repetido é substituído por uma lista de entrada.

$o_1,...,o_n$ - lista de saída constituída de expressões ou especificações de repetição com sintaxe análoga à do comando for (2.6.4.3) no qual o comando repetido é substituído por uma lista de saída.

2.10 - Recursos para manipulação de caracteres

Sequências de caracteres podem ser definidas como vetores de tipo char. A manipulação de caracteres individuais pode ser feita através de indexação. A sequência pode ser manipulada como um todo através da especificação do nome do vetor sem indexação.

2.11 - Processamento concorrente - a Lapa sequencial não tem ne-

nhum recurso para processamento concorrente. Tais recursos são encontrados na versão concorrente definida por W. P. Paula Filho [38].

2.12 - Acesso a características de máquina.

(a) registradores - através da declaração register.

(b) instruções da máquina - através do comando assemble.

2.13 - Comunicação com outras linguagens - permitida.

2.14 - Recursos em tempo de compilação - nenhum.

2.15 - Recursos para depuração de programas - nenhum.

3. JUSTIFICATIVAS

Muitos dos critérios apresentados no capítulo IV foram levados em consideração na definição da LAPA. De acordo com o primeiro critério, a estrutura geral da linguagem foi baseada no ALGOL 60 que constitui o padrão de muitas das linguagens atualmente utilizadas. Entre os demais critérios, a modularidade, eficiência e aplicabilidade assumiram maior relevância no desenvolvimento da LAPA. Apresentamos, a seguir as características introduzidas para satisfazer a esses critérios e alguns aspectos adicionais considerados.

3.1 Modularidade

Em razão da modularidade, foram mantidos os blocos como no ALGOL 60 e incluídos os comandos case, repeat e while, semelhantes aos que existem no PASCAL, com a substituição do goto pelo exit [4] para abandono de comandos.

No comando case foi incluída a especificação out , obrigatória, na qual é indicado um comando (possivelmente nulo) para ser executado quando nenhum dos comandos anteriores do case foram selecionados para execução. Desta forma fica definido de forma clara o efeito do case quando nenhum dos seletores satisfazem às comparações.

Quanto ao comando exit, em lugar das diversas formas particulares para escape de comandos existentes no BLISS, existe uma única forma na qual o comando que será abandonado é indicado através de um rótulo no exit, sendo possível o escape de vários níveis de comandos diferentes, como no seguinte exemplo:

```
A: for v from 1 to 5 do
       begin
         :
       when b then exit A;
         :
       end
b:=false;
```

Pelas razões mencionadas no capítulo IV (seção 2), a linguagem LAPA admite declarações e chamadas de procedimentos externos.

3.2 Eficiência do código gerado

Este critério exigiu a introdução de algumas restrições em relação ao ALGOL 60 e a inclusão de comandos especiais. Comentamos a seguir as características que sofreram a influência deste critério.

(a) Matrizes com limites constantes para evitar o gasto em tempo de execução na ativação dos blocos quando são declaradas matrizes com limites variáveis (cf. IV.3).

(b) Procedimentos não podem ser declarados dentro de procedimentos (cf. IV.3) devido à existência de um único registrador base (γ) para endereçamento de dados locais. Dessa maneira, não é necessário varrer-se a cadeia dinâmica para se endereçar dados de um nível superior.

(c) Além do comando case, foi incluído o comando index (análogo ao case do PL360, XPL, ALGOL B6700 e SPL) que permite uma implementação mais eficiente [4]. Enquanto que no case da LAPA os seletores de comandos são expressões calculadas cada vez que o comando é executado, no index os seletores são constantes inteiras o que torna possível a implementação do index através de desvio indexado indireto sobre uma tabela de endereços de comandos.

(d) O comando for é semelhante ao do ALGOL 68 [28] sem a especificação while, na qual as expressões que constituem o limite e incremento são calculadas apenas uma vez em cada execução do for. Estas restrições permitiram uma implementação efi-

ciente pela utilização da instrução itera [34] do PADE que repete um grupo de instruções utilizando um registrador como contador. Quando a especificação da variável de controle é omitida, o código é mais eficiente pois é eliminada a atualização da variável.

3.3 Aplicabilidade

Para programação de sistemas, conforme mencionamos no capítulo IV, são necessários recursos para manipulação de estruturas de dados e acesso a características da máquina.

Para manipulação de estruturas de dados foi incluído o tipo record, que permite a definição de tabelas e de estruturas com organização de árvore, e o tipo pointer para manipulação de listas e áreas de dados. Na declaração pointer ou na especificação de uma variável apontada pelo pointer é obrigatória a especificação do tipo apontado para que o compilador possa efetuar as verificações de correspondência de tipos em expressões aritméticas, e evitar ambiguidades na identificação de componentes de "records" manipulados através de apontador, como a que ocorre no exemplo:

```
begin
    record (integer a, real b) r1;
    record (char c, real a) r2 ;
    pointer p;
     :
        P↑.a
     :
end
```

a declaração de record admite variantes que permitem a interpretação de uma mesma parte de um record com tipos diferentes, tal como no seguinte exemplo:

```
record (integer i, (alpha a / pointer p))r
  :
  r.a   #  variável de tipo alpha  #
  r.p   #  variável de tipo pointer #
  :
```

Para permitir o acesso a características da máquina foram introduzidos os seguintes recursos:

(a) Tipos register e ion, sendo que ion permite a manipulação de todas as subdivisões válidas de uma palavra do PADE.

(b) Todos operadores existentes no PADE foram introduzidos como operadores na LAPA.

(c) Comando assemble que permite o acesso a todas as instruções da máquina.

3.4 Segurança

Existe verificação de correspondência de tipos entre os parâmetros formais e atuais em chamadas de funções e procedimentos.

3.5 Clareza

Este critério esteve presente em toda a definição da LAPA, revelando-se particularmente nos seguintes aspectos da linguagem:

(a) O comando if ... then (sem a parte else) do ALGOL 60 foi substituído na LAPA pelo comando when ... then. Esta modificação aumenta a clareza do programa ficando determinado de forma inequívoca a qual comando if um else pertence, tal como no seguinte exemplo:

if...then if...then...else when...then...else..

(b) Não são permitidas misturas de tipos em expressões para evitar as conversões implícitas cujos efeitos muitas vezes não são claros ao programador e que prejudicam a consideração do programa como sua própria documentação.

(c) No comando index, a constante que causa a seleção de um comando deve preceder este comando,cons

tituindo uma forma mais clara do que a do case existente nas linguagens PL360, XPL, ALGOL B6700, ESPOL e SPL.

Exemplo:

```
index x+y of
  3 → Z:= a+1,
  1 → Z:= a+2,
  2 → Z:= a,
  5 → Z:= a+3,
out → Z:= 0;
```

3.6 Aspectos adicionais considerados

(a) Na LAPA a atribuição ocorre sem conversão, isto é, o valor é atribuído sem interpretação de tipo. Esta forma de atribuição foi introduzida para permitir ao programador de sistemas a facilidade de manipulação de uma mesma configuração de bits sob tipos diferentes.

(b) Foi introduzida a expressão seletiva case para manter a ortogonalidade da linguagem desde que foi decidido manter a expressão seletiva if.

(c) A definição de tipo record, além de simplificar a codificação do programa pois evita declarações repetidas de uma mesma estrutura, permite ainda uma forma de declaração pseudo-recursiva como a do seguinte exemplo:

```
record t (..., pointer (t) p,...)
```

(d) O comando repeat forever foi introduzido para evitar a utilização da forma while true do ... menos eficiente.

4. CONSIDERAÇÕES GERAIS

O início da definição da linguagem LAPA coincidiu com a fase final do projeto da arquitetura do PADE o que possibilitou a introdução de algumas alterações em sua arquitetura visando facilitar a implementação da linguagem.

No projeto da LAPA existiu a preocupação de não introduzir recursos demasiados para não sobrecarregar o compilador tornando-o excessivamente grande e difícil de ser implementado. Desta forma evitou-se acrescentar alguns recursos considerados desejáveis mas não imprescindíveis.

Sugerimos a seguir algumas alterações que consideramos desejáveis e que poderão ser incluídas em futuras versões da linguagem.

(a) Definição de constantes - contribui para a clareza e documentação de programas, além do que, através da alteração dos valores das constantes declaradas e recompilação do programa torna-se fácil a modificação de seus parâmetros tais como limites de matrizes, como no seguinte exemplo:

```
const liminf=1, limisup=10, # declaração de constante #

[liminf : limisup] integer a,b;
```

(b) Inicialização de variáveis simples e estruturadas na declaração. Pode-se notar pelas descrições das diversas linguagens no capítulo III que a maioria permite inicialização na declaração. Pudemos notar também, durante a programação do compilador da LAPA efetuada em PL/I, a grande utilidade deste recurso.

(c) A atribuição existente no PADE, sem conversão de tipo, não é cômoda nem segura ao programador científico, podendo dar origem a erros. Desta forma sugerimos uma forma adicional de atribuição na qual não seja permitida a atribuição de uma expres

são a uma variável de tipo diferente.

(d) A declaração register deveria permitir ao programador a escolha de um dos oito registradores da máquina, com as seguintes vantagens:

(i) Facilidade de comunicação entre procedimentos externos através da declaração dos mesmos registradores.

(ii) Facilidade de comunicação de um trecho de código em linguagem de montagem, introduzido através de um comando assemble, com os demais comandos do programa.

(iii) Possibilidade de utilização de registradores como apontadores de pilha de acordo com o permitido pela arquitetura do PADE que exige neste caso que o registrador seja de número par.

(e) Inclusão de comandos scan e move, análogos aos que existem nas linguagens ALGOL B6700, ESPOL e SPL, que podem ser implementados de forma eficiente através da utilização da instrução repete do PADE [34].

(f) Declaração de "array's" e "record's" opcionalmente na área de alcance através de endereçamento direto.

CAPÍTULO VI

CONCLUSÃO

O estudo das diversas linguagens de programação d e sistemas permite constatar que a linguagem LAPA está bem situada em relação às demais linguagens. Pela tabela anexa ("Resumo esquemático das características das linguagens") pode-se verificar que a LAPA possui a maioria dos recursos necessários à programação de sistemas, a menos dos recursos para programação concorrente incluídos na LAPA concorrente. Um aspecto que ressaltamos é a simplicidade e clareza das construções da linguagem que facilitam o seu ensino e contribuem para que o programa constitua sua própria documentação.

Um ponto que poderia ser questionado é a razão de se desenvolver uma nova linguagem em lugar de se implementar uma linguagem já definida, escolhida por exemplo entre as descritas no capítulo III. Podemos apresentar os seguintes motivos - que levaram à definição da LAPA.

a) Todo o projeto do computador foi desenvolvido em um meio acadêmico sendo por isso plenamente justificável deixar de lado as soluções cômodas como a de implementar uma linguagem já existente. A definição de uma nova linguagem possibitou a ampliação e aprofundamento de nossos conhecimentos nesta área bem como apresentar boas soluções em relação às existentes nas demais linguagens.

b) Muitas das linguagens de programação de sistemas, como pode-se verificar pelas descrições do capítulo III, são orientadas para uma máquina em particular. Desta forma, em lugar de se escolher uma linguagem já existente e tentar adaptá-la às característica do PADE, optou-se pela definição de

uma linguagem levando em consideração algumas características particulares da máquina.

c) Entre as linguagens estudadas, muitas possuiam restrições desnecessárias (tal como, a inexistência de blocos como comando), apresentavam características que poderiam ser melhoradas (tal como a substituição do go to pelo exit como o existente na LAPA) ou possuiam recursos que não eram adequados para a implementação de forma eficiente no PADE - (como declaração de procedimentos encaixados). Desta forma foi definida uma linguagem com os recursos considerados necessários e que pudesse ser implementada de forma eficiente no PADE.

A linguagem LAPA sequencial, embora sujeita a algumas alterações como as que sugerimos no capítulo V para assumir seu aspecto definitivo, pode ser efetivamente utilizada em sua forma atual para programação científica e de sistemas.

RESUMO ESQUEMÁTICO DAS CARACTERÍSTICAS DAS LINGUAGENS

características					LAPA	PL360	BLISS	PL/I	XPL	PASCAL S	PASCAL C	ALGOL B	ESPOL	SPL
Tipos	Simples	inteiro			X	X		X	X	X	X	X	X	X
		real			X	X		X		X	X	X	X	X
		lógico			X	X		X		X	X	X	X	X
		apontador			X			X		X		X	X	X
		bit			X			X	X					
		byte ou caráter			X	X		X	X	X	X	X		X
		palavra			X		X	X					X	
		evento						X				X	X	
		outros						X						
	Estruturados	array	dimensões	uma		X	X		X					X
				uma ou mais	X			X		X	X	X	X	
			limites	fixos	X	X	X		X	X	X			X
				variáveis				X				X	X	(1)
		record			X			X		X	X			
		set								X	X			
		queue									X		X	
		outros								X	X			
	Definição de novos tipos				X					X	X			
Declarações	de	variáveis			X	X	X	X	X	X	X	X	X	X
		constantes					X			X	X	X		X
		registradores			X		X							
		macros					X	X	X			X		X
	Localização	início de blocos			X	X	X			X	X	X	X	X
		entre os comandos						X	X					
	Inicialização					X		X	X				X	X
	Redefinição					X	X	X				X	X	X
	Domínio definido por um bloco				X	X	X	X	X	X		X	X	X
	Pré-declarações	identificadores associados a registradores				X							X	
		vetor associado à memória				X			X				X	
Alocação	Tipos de alocação	estática			X	X	X	X	X	X	X	X	X	X
		dinâmica	em nível de blocos		X		X	X				X	X	
			em nível de procedimentos							X	X			X
		controlada						X		X	X		X	
	Meio	memória			X	X	X	X	X	X	X	X	X	X
		registradores			X	X	X						X	X
	Verificação da validade de apontadores					n.a.			n.a.		n.a.	(2)		
Expressões	Simples	operandos	constantes	base 2	X			X	X			X	X	X
				base 8	X		X		X			X	X	X
				base 10	X	X	X	X	X	X	X	X	X	X
				base 16	X	X			X			X	X	X
			variáveis		X	X	X	X	X	X	X	X	X	X
			chamada de funções		X		X	X	X	X	X	X	X	X
			especificação de palavra parcial			X	X	X	X			X	X	X

características				LISPA	PL360	BLISS	PL/I	XPL	PASCAL S	PASCAL C	ALGOL B	ESPOL	SPL
Expressões (continuação)	Simples	operadores	aritméticos	X	X	X	X	X	X	X	X	X	X
			relacionais	X	X	X	X	X	X	X	X	X	X
			lógicos	X	X	X	X	X	X	X	X	X	X
			deslocamentos("shift")	X	X	X		X					X
			atribuição			X					X	X	X
			concatenação	X			X	X			X	X	X
		precedência dos operadores	sem precedência		X								
			semelhante à do ALGOL60			X	X	X			X	X	X
			outra	X					X	X			
	Seletivas	if then else		X		X					X	X	X
		case		X		X					X	X	
	Outras					(3)							
	Mistura de tipos	não são permitidas		X					X	X			X
		permitidas	com conversão			na	X	X			X	X	
			sem conversão		X							(4)	(4)
Comandos	Rótulos de comandos			X	X		X	X	X		X	X	X
	Bloco	com alocação estática			X								
		com alocação dinâmica		X		X	X		X	X	X	X	X
	Comando composto			X	X	X	X	X	X	X	X	X	X
	Atribuição	simples			X	X			X	X			
		múltipla		X			X	X			X	X	X
		a palavras parciais			X	X	X	X				X	X
	Estruturas de controle	Desvio ou escape	go to		X		X	X	X		X	X	X
			return			X	X	X					
			exit	X		X							
		Comandos seletivos	if then else	X	X	X	X	X	X	X	X	X	X
			if then	X	X	X	X	X	X	X	X	X	X
			case	X	X	X		X	X	X	X	X	X
		Comandos repetitivos	for	X	X	X	X	X	X	X	X	X	X
			while do	X	X	X	X	X	X	X	X	X	X
			repeat	X		X			X	X		X	X
			thru do (contagem)	(5)							X	X	
	Chamada de procedimentos	semelhante à do ALGOL 60		X			X	X	X	X	X	X	X
		outra forma			X								
	Comando nulo	cadeia vazia		X			X	X	X	X	X	X	X
		null			X								
	scan										X	X	X
	replace ou move										X	X	X
Procedimentos	Formas de declaração	procedimentos semelhantes aos do ALGOL 60		X			X	X	X	X	X	X	X
		procedimentos sem parâmetros			X								
		funções		X		X			X	X			
		subrotinas	com dados locais			X							
			sem dados locais										X
	Declarações encaixadas	de funções ou procedimentos			X	X	X	X	X		X	X	
		de subrotinas		n.a.	n.a.	X	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	

características				LAPA	PL/360	BLISS	PL/I	XPL	PASCAL S	PASCAL C	ALGOL B	ESPOL	SPL
Procedimentos (continuação)	Parâmetros	tipos	variáveis	X		X	X	X	X	X	X	X	X
			procedimentos		n.a.	X	X		X	X	X	X	X
			outros (label, event, etc.)			X	X		X	X	X	X	X
		modos de passagem	por nome								X	X	
			por valor	X	n.a.	X		X	X	X	X	X	X
			por referência	X			X		X	X		X	X
		verificação da correspondência entre os parâmetros formais e atuais		X	n.a.					X	X	X	X
	Retorno de valor por comando	de atribuição							X	X	X	X	
		return ou exit		X		X	X	X					X
	Acesso a dados globais	em procedimentos		X	X	X	X	X	X	X	X	X	X
		em subrotinas		n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	X
	Recursão			X		X	X		X		X	X	X
	Procedimentos externos- declaração e chamada			X		X	X		X	X	X	X	X
	Efeitos colaterais em funções		com restrições							X			
			sem restrições		X	X	X	X	X		X	X	X
	Procedimentos pré-declarados	manipulação de bits e caracteres				X	X	X					
		entrada e saída						X	X				
		acesso a operadores/instruções de máquina						X					
		conversão de tipo					X						
Programa	Bloco												X
	Comando			X	X								
	Procedimento			X			X				X	X	
	Outra forma					X		X	X	X			
Entrada e saída	nenhum recurso				X	X				X			X
	através de funções e procedimentos							X	X				
	através de comandos			X			X				X	X	
	com especificação de formatos			X			X				X		
Processamento concorrente	nenhum recurso								X				
	através de instruções de máquina ou procedimentos do sistema			X	X	X		X				X	X
	recursos especiais	wait, cause ou formas equivalentes					X			X	X	X	
		monitor								X			
		processamento de interrupções					X				X	X	
		outros									X	X	
Acesso a características da máquina	instruções			X	X	X		X				X	X
	registradores			X	X	X		(6)				X	X
	memória (através de endereçamento absoluto				X	X		X				X	X
Recursos em tempo de compilação	nenhum			X	X				X	X			
	macros					X	X	X			X	X	X
	comandos executados por pré-processador						X						
Recursos para depuração de programas					X		X	X			X		

Notações e Observações:

X - A característica está presente na linguagem;

n.a. - A característica não se aplica à linguagem;

PASCAL S - PASCAL sequencial;

PASCAL C - PASCAL concorrente;

ALGOL B - ALGOL B6700;

(1) - Limites variáveis somente no interior de procedimentos;

(2) - Existe verificação da validade de apontadores em uma versão recente do ALGOL B6700;

(3) - No BLISS, toda construção executável da linguagem é uma expressão;

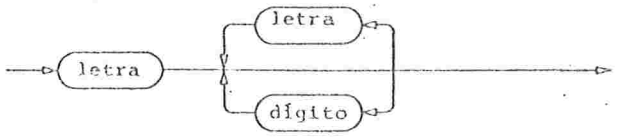
(4) - Em formas especiais de expressões são permitidas as misturas de tipos sem conversão, sendo que os operandos são interpretados como sequências de bits;

(5) - Caso particular da construção for;

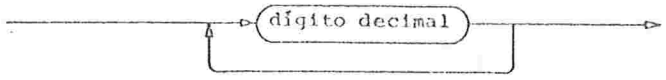
(6) - Através de acesso a instruções de máquina.

APÊNDICE - DIAGRAMA SINTÁTICO DA LAPA

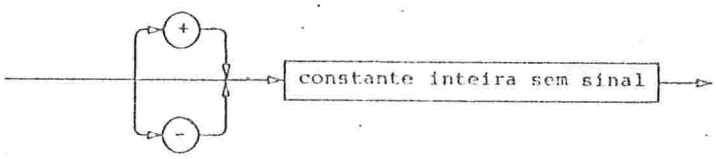
identificador



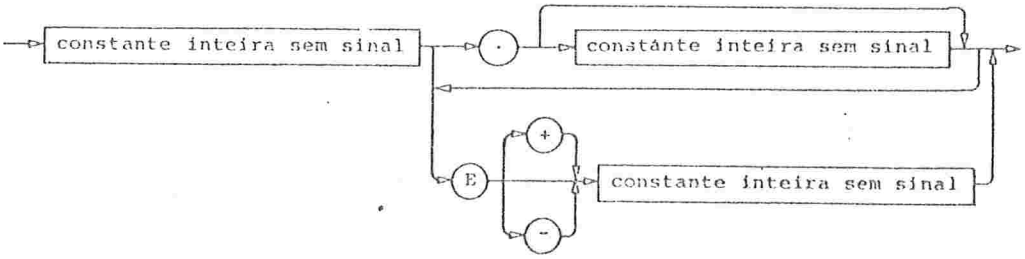
constante inteira sem sinal



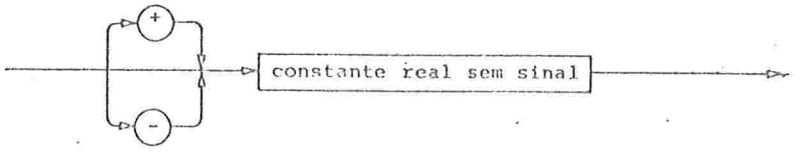
constante inteira



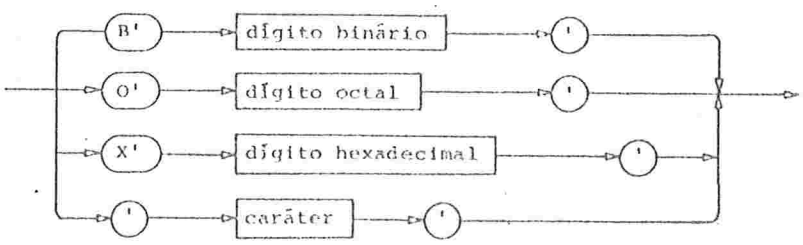
constante real sem sinal



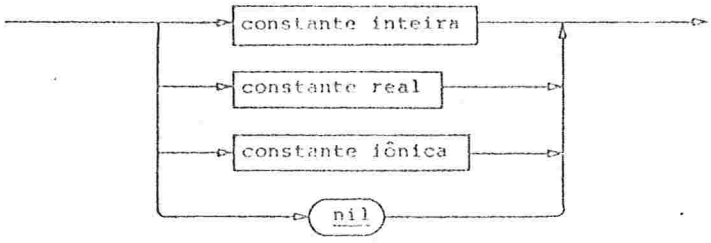
constante real



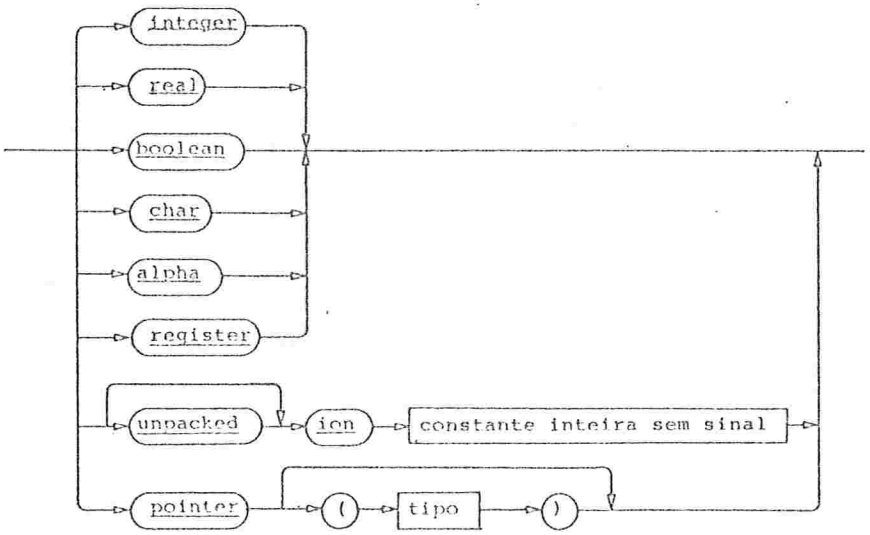
constante lônica



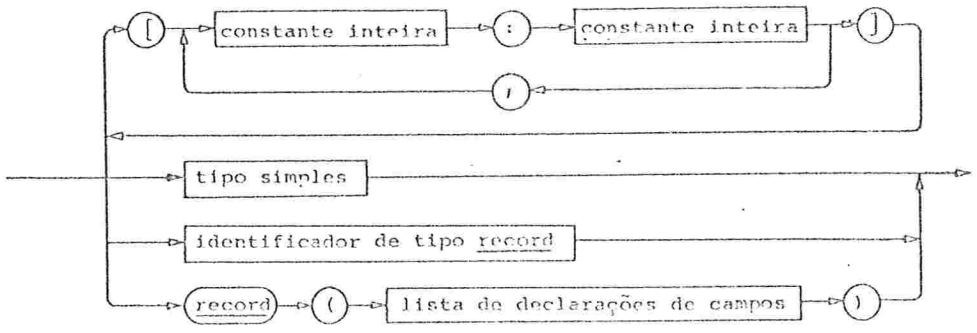
constante



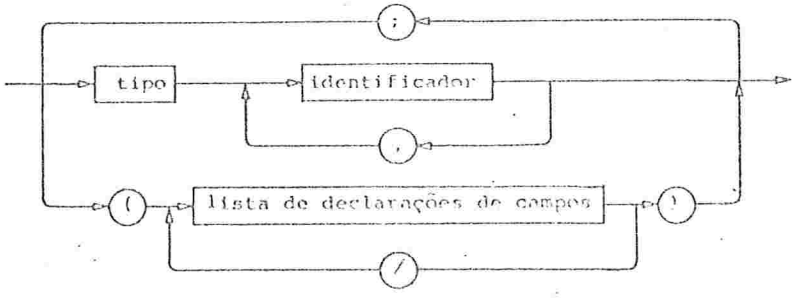
tipo simples



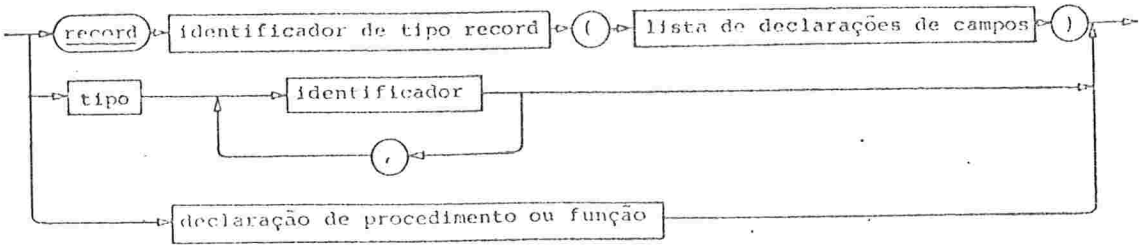
tipo



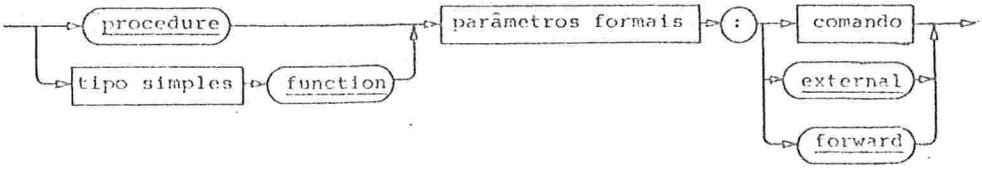
lista de declarações de campos



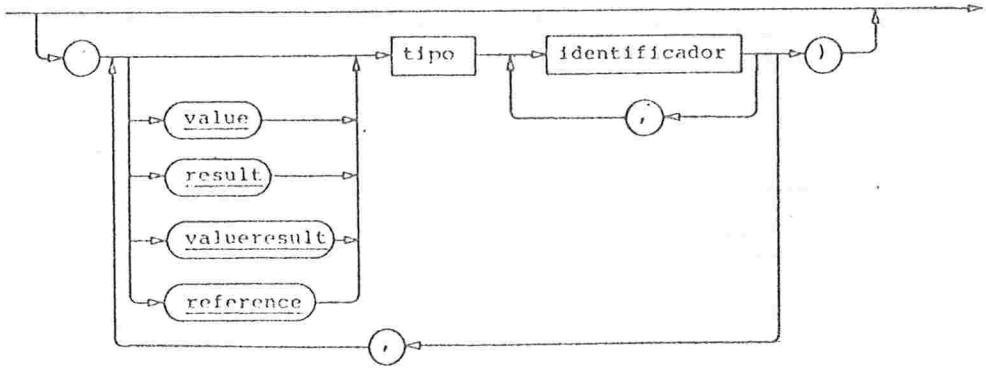
declaração



declaração de procedimento ou função

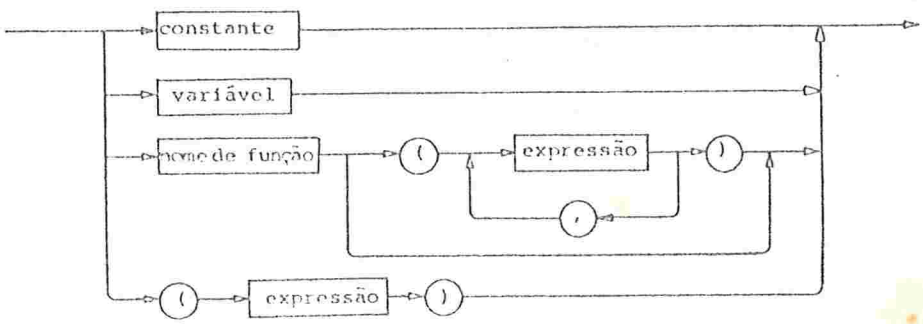


parâmetros formais

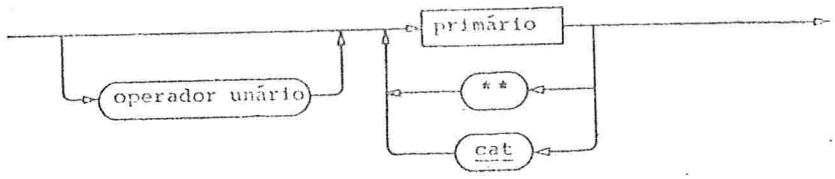


variável

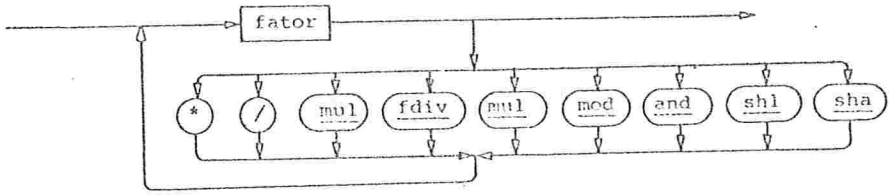
primário



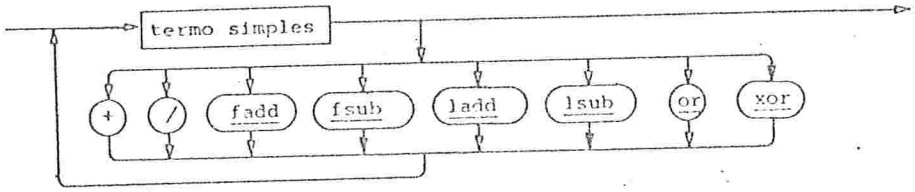
fator



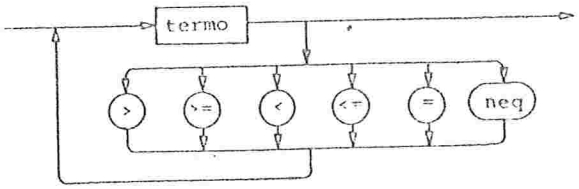
termo simples



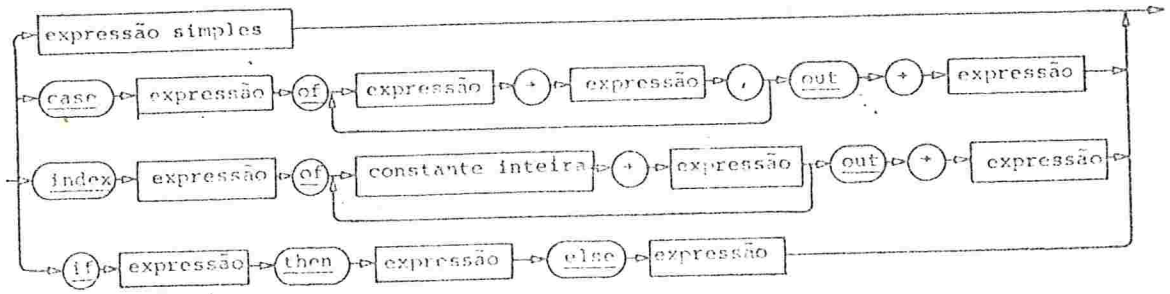
termo



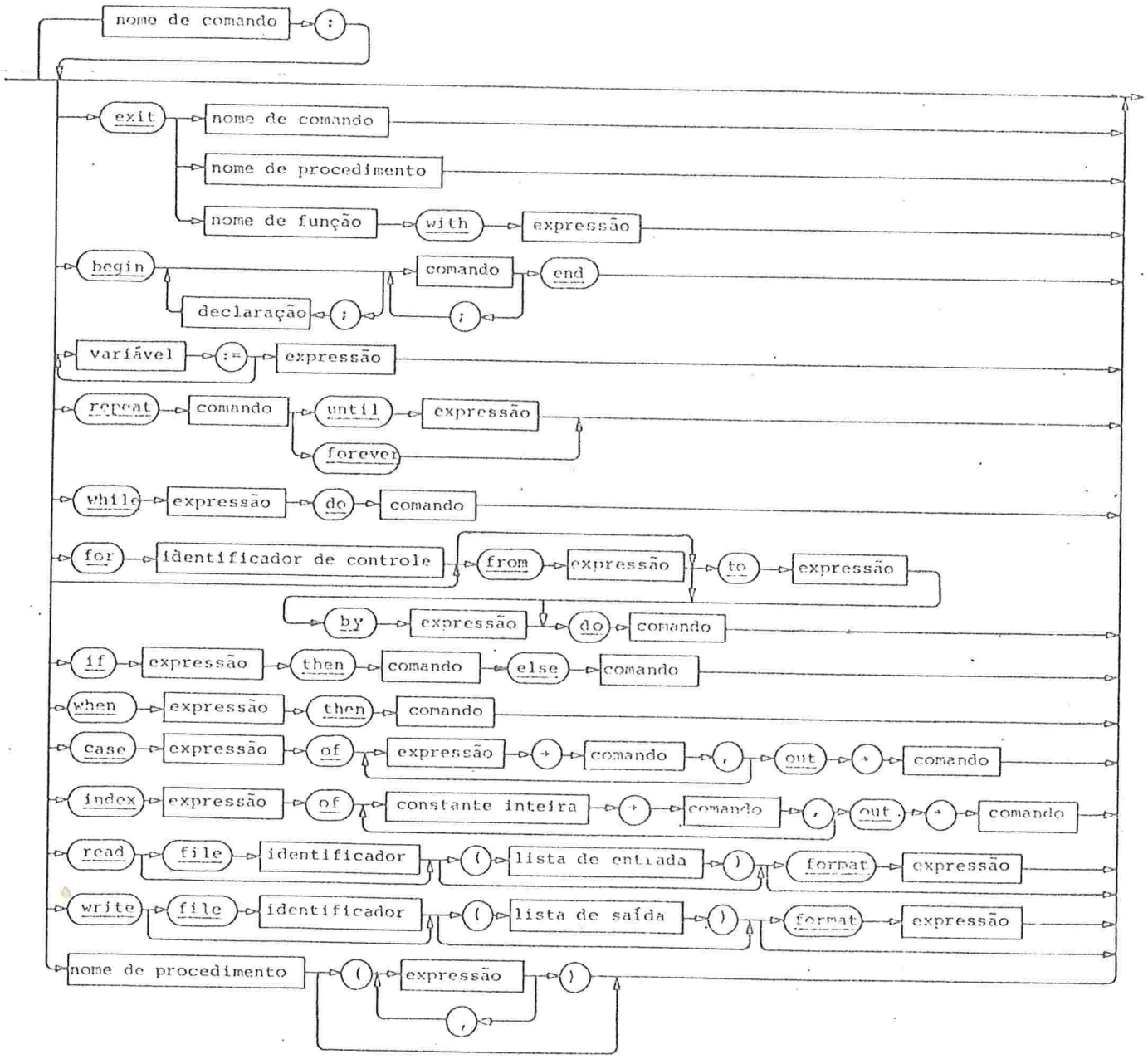
expressão simples



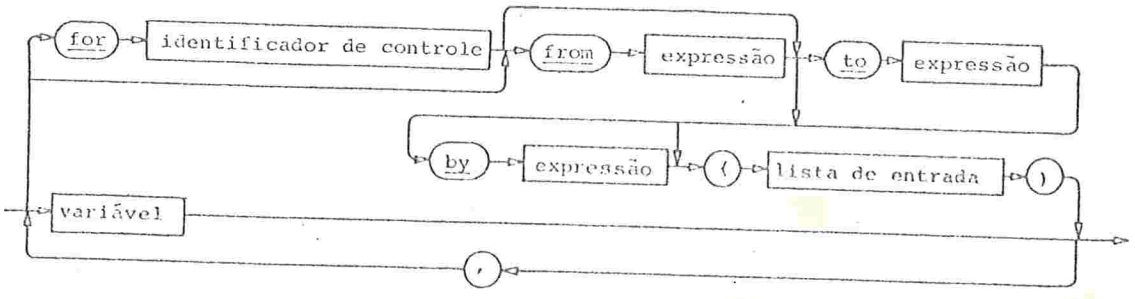
expressão



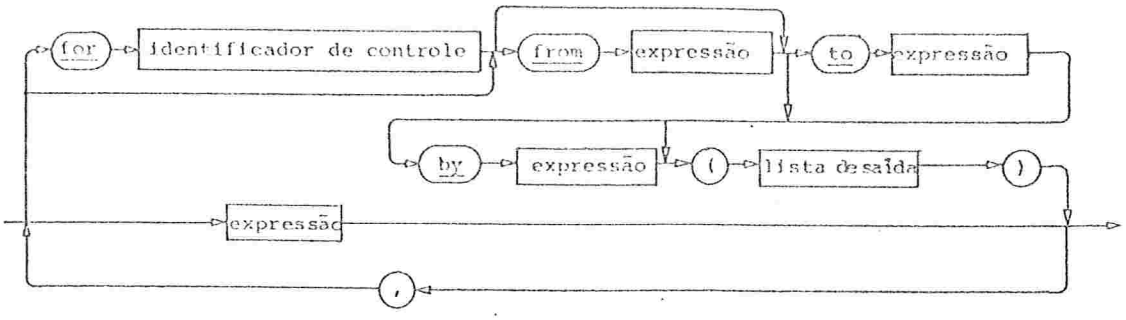
comando



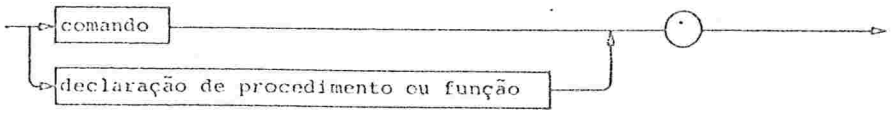
lista de entrada



lista de saída



programa



REFERÊNCIAS BIBLIOGRÁFICAS

[1] Bergeron, R. and Van Dam, A. - *A Language for System Development* - Proc. ACM SIGPLAN, Symposium on Languages for Systems Implementation, SIGPLAN Notices, Vol. 6, Nº 9 , Oct. 1971, 50-72.

[2] Bergeron, R.D. et alli - *Systems Programming Languages* - Advances in Computers, Vol. 11, Academic Press,New York, 1971, 175-284.

[3] Bigonha, R.S. e Diniz, M.T.G. - *FOPA - Linguagem de Formatos do PADE* - IV Seminário sobre Desenvolvimento Integrado de Software e Hardware, UFMG, Belo Horizonte, Julho 1977.

[4] Bressan, G. e Homem de Melo, I.S. - *Uma Linguagem de Alto Nível Aderente a uma Arquitetura e sua Implementação*- IV Seminário sobre Desenvolvimento Integrado de Software e Hardware, UFMG, Belo Horizonte, Julho 1977.

[5] Bressan, G. - *Uma Avaliação Comparativa de Linguagens de Programação de Sistemas* - IV Seminário sobre Desenvolvimento Integrado de Software e Hardware, UFMG, Belo Horizonte, Julho 1977.

[6] Brinch Hansen, P. - *Concurrent Programming Concepts* - ACM Computing Surveys, Vol. 5, Nº 4, Dec. 1973, 223-245.

[7] Brinch Hansen, P. - *A Programming Methodology for Operating System Design* - in Rosenfeld, J.L. (Ed.), Information Processing 74, Amsterdam, North-Holland, 1974 , 394-397.

[8] Brinch Hansen, P. - *Concurrent PASCAL Machine* - Information Science, California Institute of Technology, June 1975.

[9] Brinch Hansen, P. and Hartman, A.C. - *Sequential PASCAL Report*-Information Science, California Institute of Technology, July 1975.

[10] Brinch Hansen, P. - *Concurrent PASCAL Introduction* - Information Science, California Institute of Technology , July 1975.

[11] Brinch Hansen, P. - *Concurrent PASCAL Report* - Information Science, California Institute of Technology, Oct. 1975.

[12] Brinch Hansen, P. - *The SOLO Operating System: A concurrent PASCAL Program* - Software - Practice and Experience, Vol. 6, Nº 2, 1976, 141-148.

[13] Burroughs B6700 - *ALGOL Language Reference Manual* - Nº 5000128, June 1972.

[14] Burroughs B6700/B7700 - *ESPOL - Executive System Programming Language* - Information Manual - Nº 5000094, June 1972.

[15] Corbató, F.J. - *PL/I As a Tool for System Programming* - Datamation, Vol. 15, Nº 5, May 1969, 68-76.

[16] Dahl, O.J. and Nygaard, K. - *SIMULA 67: Common Base Language* - Norwegian Computing Center, Oslo.

[17] Dijkstra, E.W. - *Cooperating Sequential Process* - in Programming Languages, Genuys, F. (Ed.), Academic Press,New York, 1968, 43-112.

[18] Dijkstra, E.W. - *The Structure of THE Multiprogramming System*, Comm. ACM, Vol. 11, nº 5, May 1968, 341-346.

[19] Hoare, C.A.R. - *Monitors: An Operating System Structuring Concept* - Comm. ACM, Vol. 17, Nº 10, Oct. 1974 , 549-557.

[20] Horning, J.J., and Clark, B.L. - *The System Language for Project SUE* - Proc. ACM SIGPLAN, Symposium on Languages for Systems Implementation, SIGPLAN Notices, Vol. 6,Nº 9 Oct. 1971, 79-88.

[21] HP-3000 - SPL - *System Programming Language* -03000- 9000 2A, Nov. 1972.

[22] HP-3000 - SPL - *System Programming Language Textbook* - 03000-90003, Nov. 1972.

[23] HP-3000 - *Reference Manual* - 03000-90019, Sept. 1973.

[24] IBM System /360 Operating System - *PL/I(F) Language Reference Manual* - C28 - 8201, White Plains, N.Y., 10601.

[25] IBM System /360 - *Principles of Operation* - GA22-6821-8.

[26] Jensen, K. and Wirth, N. - *PASCAL - User Manual and Report* - Lectures Notes in Computer Science Nº 18, Springer - Verlag, 1974.

[27] Knuth, D.E. - *Structured Programming with GO TO Statements* - Computing Surveys, Vol. 6, 261-301, 1974.

[28] Lindsey, C.H. and van der Meulen, S.G. - *Informal Introduction to ALGOL 68* - North Holland, Amsterdam, 1971.

[29] Lister, A. - *The Problem of Nested Monitor Calls* - ACM

SIGOPS, Operating System Review, Vol. 11, Nº 3,July 1977, 5-7.

[30] Lyle, D.M. - *A Hierarchy of High Order Languages for System Programming* - Proc. ACM SIGPLAN, Symposium on Languages for Systems Implementation, SIGPLAN Notices, Vol. 6, Nº 9, Oct. 1971, 73-78.

[31] Mackeeman, W.M., Horning, J.J. and Wortman, D.B. - *A Compiler Generator* - Prentice-Hall, 1970.

[32] Malcoln, M.A. - *PL360 (Revised) - A Programming Language for the IBM 360* - Computer Science Department, Stanford University, Stanford, California, STAN-CS-71-215.

[33] Mammana, C.Z. et alli - *Digital Processor for Data Acquisition (PADE)* - Annual Report, Nuclear Physics Department, USP, 1975, 123-127.

[34] Mammana, C.Z. - *Arquitetura de um Processador para Aquisição de Dados Estocásticos* - IV Seminário sobre Desenvolvimento Integrado de Software e Hardware, UFMG, Belo Horizonte, Julho 1977.

[35] Mammana, C.Z. e Ferraretto, M.D. - *LORPA - Linguagem Orientada para o PADE* - IV Seminário sobre Desenvolvimento Integrado de Software e Hardware, UFMG, Belo Horizonte, Julho 1977.

[36] Mammana, C.Z. - *LIMPA - Linguagem de Montagem do PADE* - Publicação interna do SEMA, GEPS, Departamento de Física Nuclear, IFUSP, 1977.

[37] Naur, P. et. alli - *Revised Report on the Algorithmic Language ALGOL 60*, Comm. ACM, Vol. 6, Nº 1, Jan. 1963, 1-17.

[38] Paula Filho, W.P. e Bigonha, R.S. - *SAPPA - Um Sistema para Construção de "Software" de Sistemas* - III Seminário sobre Desenvolvimento Integrado de Software e Hardware, UFRS, Porto Alegre, Julho 1976.

[39] Richards, M. - *BCPL: A Tool for Compiler Writing and System Programming* - Proc. AFIPS, 1969, SJCC, Vol. 34,AFIPS Press, Montvale, N.J., 557-566.

[40] Russel, R.D. - *Intermediate - level Programming Languages for On-line Data Acquisition and Control* - Computer Physics Communications, Vol. 5, 1973, 89-97.

[41] Sammet, J.E. - *A Brief Survey of Languages Used in Systems Implementation* - Proc. ACM SIGPLAN, Symposium on Languages for Systems Implementation, SIGPLAN Notices,Vol.6, Nº 9, Oct. 1971, 1-19.

[42] Seegmüller, G. - *Some Design Considerations for a Universal System Programming Language* - Working Notes of the Language Group of the Leibniz - Rechenzentrum der Bayerischen Akademie der Wissenschaften, NR. 7303,Oct. 1973.

[43] Setzer, V.W. and Bressan, G. - *A Programming Language for a Dedicated-language Mini-computer System* - International Symposium on Metodologies for the Design and Construction of Software and Hardware Systems, PUC, Rio de Janeiro, June 1976.

[44] Stolfi, J., Zwicker, R., Kowaltowski, T. and Setzer,V.W. *Relatório Preliminar sobre ALGOLUSP* - Publicação interna do IME-USP-DMA, 1974.

[45] Toscani, S.S. - *Processamento Assíncrono na Linguagem ALGOL B6700* - UFRS, Pós- Graduação em Ciência da Computação, Set. 1976.

[46] Vieira, N.J. e Streitwieser, W. - MAPPA - *Um Macro Processador Sintático de Uso Múltiplo* - III Seminário sobre Desenvolvimento Integrado de Software e Hardware, UFRS , Porto Alegre, Julho 1976.

[47] Wirth, N. and Hoare, C.A.R. - *A Contribution to the Development of ALGOL* - Comm. ACM, Vol. 9, Nº 6, June 1966 , 413-441.

[48] Wirth, N. - *PL360, A Programming Language for the 360 Computers* - JACM, Vol. 15, Nº 1, Jan. 1968, 37-74.

[49] Wirth, N. - *The Programming Language PASCAL* - Acta Informatica, Vol. 1, 1971, 35-63.

[50] Wirth, N. - *Systematic Programming - An Introduction* -Englewoods Cliffs (N.J.) - Prentice Hall, 1973.

[51] Wulf, W.A. et alli - *BLISS Reference Manual* - Carnegie - Mellon University, Department of Computer Science, Pittsburgh, Pa. 15213, Jan. 1970.

[52] Wulf, W.A. et alli - *Reflections on a System Programming Language* - Proc. ACM SIGPLAN, Symposium on Languages for Systems Implementation, SIGPLAN Notices, Vol. 6, Nº 9 , Oct. 1971.

[53] Wulf, W.A., Russel, D.B. and Habermann, A.N. - *BLISS: A Language for Systems Programming* - Comm. ACM, Vol. 14 , Nº 12, Dec. 1971, 780-798.