1. Publicação nº	2. Versão	3. Data	5. Distribuição
INPE-3880-NTE/257		*Abril, 1986*	☐ Interna ☒ Externa ☐ Restrita
4. Origem *DIN/DCS*	Programa *DENUME*		

6. Palavras chaves - selecionadas pelo(s) autor(es)
COMPILADOR
MICROCOMPUTADOR
PONTO FLUTUANTE

7. C.D.U.: *681.322:621.38*

8. Título *INPE-3880-NTE/257*

OPERAÇÕES EM PONTO FLUTUANTE PARA MICROCOMPUTADORES MC68000 E IT8086

10. Páginas: *46*

11. Última página: *41*

12. Revisada por

Edson L.F. Senne

9. Autoria *Guilherme Bittencourt*
Nandamudi Lankalapalli Vijaykumar
Santiago Alves Tavares

13. Autorizada por

Marco Antonio Raupp
Diretor Geral

Assinatura responsável N.L. Vijaykumar

14. Resumo/Notas

O relatório descreve algoritmos de operações de ponto flutuante. As operações descritas são: adição, subtração, multiplicação, divisão, resto, raiz quadrada, seno, co-seno, tangente, arco-seno, arco co-seno, arco-tangente, exponencial, logaritmo, comparação e conversão de tipos. Alguns detalhes relacionados à implementação das operações nos microcomputadores INTEL 8086 e MOTOROLA 68000 são comentados.

15. Observações *Este é o relatório final do projeto conjunto INPE/CTI, a ser divulgado pelo CTI (Centro Tecnológico para Informática) junto à EMBRATEL (Empresa Brasileira de Telecomunicações).*

ABSTRACT

This report describes algorithms for the floating point operations which, are: addition, subtraction, multiplication, division, remainder, square root, sine, cosine, tangent, arc sine, arc cosine, arc tangent, exponential, natural logarithm, compare, and type conversions. Some implementation details of the operations in the microcomputers INTEL 8086 and operations in the MOTOROLA 68000 are commented.

SUMÁRIO

Pág.

1. OPERAÇÕES DE PONTO FLUTUANTE

Neste relatório descreve-se o módulo de operações em ponto-flutuante. Inicialmente apresentam-se os formatos dos dados e comentam-se os procedimentos de arredondamento e de tratamento de exceções. A seguir, apresentam-se os algoritmos para cada operação aritmética no módulo. Descreve-se também a forma de comunicação deste módulo com o compilador da linguagem C e, finalmente, comentam-se detalhes visando a implementação destes algoritmos em microcomputadores MC 68000 e IT 8086.

1.1 - DESCRIÇÃO DO FORMATO

Nesta seção discute-se o formato dos operandos a serem utilizados pelas rotinas do módulo e o formato dos resultados intermediários. Comentam-se as formas de arredondamento, as condições de exceção e como tratá-las.

1.1.1 - FORMATO DOS DADOS

Será adotado o formato de dados do padrão IEEE, descrito por Coonen (1980).

Todo número real pode ser expresso em forma normalizada de ponto flutuante como:

$$(-1)^S * 2^{(E-B)} * (1.F),$$

onde S é o bit de sinal (0 ou 1).

E é o valor do expoente polarizado pelo valor B.

B é o valor da polarização,

F é o valor da parte fracionária do número, sendo considerado implicitamente um bit à esquerda do ponto decimal.

Propõe-se a implementação de três tipos de formatos SIMPLES, DUPLO e DUPLO-ESTENDIDO, sendo todas as operações realizadas no formato DUPLO-ESTENDIDO, e o resultado posteriormente normalizado e arredondado para um dos outros dois formatos, se necessário. O formato DUPLO-ESTENDIDO é utilizado apenas no interior das rotinas, não sendo acessível aos usuários destas. Por simplicidade, chamar-se-á, daqui em diante, o formato DUPLO-ESTENDIDO apenas ESTENDIDO.

Os três formatos podem ser armazenados na cadeia de bits mostrada na Figura 1, onde S, E e F são cadeias de bits com comprimentos determinados pelo tipo de formato, de acordo com a Tabela 1.

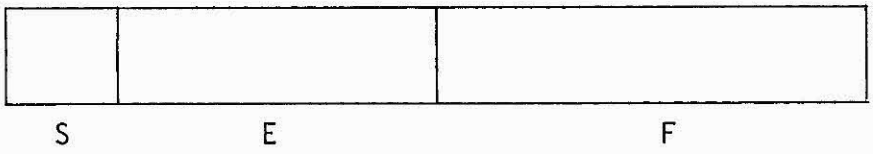

Fig. 1 - Formato de número de ponto flutuante.

TABELA 1

COMPRIMENTO DOS CAMPOS (EM BITS)

	SIMPLES	DUPLO	ESTENDIDO
S	1	1	1
E	8	11	15
F	23	52	64
TOTAL	32	64	80

O expoente é representado na forma sinal-magnitude e seus valores máximo e mínimo são reservados para designar operandos especiais dos quais um é o ZERO NORMAL, representado por E=F=0. Com os tamanhos de campos descritos acima, os formatos podem armazenar o seguinte número de dígitos decimais significativos: SIMPLES, 7 dígitos, e DUPLO, 16 dígitos. O valor da polarização B também varia de acordo com o formato utilizado, para o formato SIMPLES B=127 e para o formato DUPLO B=1023.

Assim, números normalizados não-nulos em ponto-flutuante podem variar em magnitude entre:

$$2^{(-126)} * 1.000...0_2 \leq X \leq 2^{127} * 1.111...1_2 \text{ (SIMPLES)},$$

$$2^{(-1022)} * 1.000...0_2 \leq X \leq 2^{1023} * 1.111...1_2 \text{ (DUPLO)}.$$

Esta maneira de codificar os números em ponto-flutuante tem a propriedade de fazer com que a ordem dos números coincida com sua ordem lexicográfica, o que permite que se possa, para comparações, considerá-los como números binários em formato sinal-magnitude.

1.1.2 - ARREDONDAMENTO

Existem quatro tipos de arredondamento. Seja um número real Z, não-representável exatamente em um dado formato, e sejam Z1 e Z2 dois números em ponto-flutuante representáveis no formato, de modo que Z1 < Z < Z2. Os quatro arredondamentos possíveis são:

- Para o mais próximo (RN): aproxima-se Z para o número em ponto-flutuante mais próximo. Este tipo de arredondamento é o mais usual.

- Para zero (RZ): aproxima-se Z para o menor em magnitude entre Z1 e Z2.

- Para mais infinito (RP): aproxima-se Z para Z2.

- Para menos infinito (RM): aproxima-se Z para Z1.

Os dois últimos modos são arredondamentos dirigidos e são utilizados nas implementações de aritmética de intervalo. Um sistema de ponto-flutuante pode conter apenas o primeiro modo (RN), e o segundo (RZ), para ser utilizado nos arredondamentos para inteiro, ou todos os quatro modos. No presente sistema serão implementados apenas os tipos RN e RZ.

1.1.3 - EXCEÇÕES

São considerados cinco tipos de exceções: operação inválida, "underflow", "overflow", divisão por zero e resultado inexato. A cada tipo de exceção está associado um determinado bit de ocorrência que é tornado 1 sempre que a exceção acontece, ficando a cargo do usuário recolocar o valor 0. A cada exceção o sistema responde com um resultado determinado, liga o bit de ocorrência e prossegue. Sendo possível, no entanto, a implementação de desvios para determinadas exceções, através de bits de desvio, nestes casos o usuário deverá determinar quais as operações a serem realizadas no ponto de desvio. A seguir descreve-se cada um dos tipos de exceção considerados:

a) Operação inválida: Este tipo de exceção corresponde a erros que ocorrem em uma variedade de operações aritméticas e que não são suficientemente importantes ou freqüentes a ponto de merecer uma condição de erro particular. Alguns exemplos de operação inválida são:

   SQRT(-2),

   (+infinito)-(+infinito),

   zero x infinito.

O resultado de operações onde ocorre este tipo de exceção faz parte de uma classe de operandos chamados NÃO-NUMÉRICOS (NaN), que

são caracterizados pelos seguintes valores de seus campos de bits (de acordo com a Figura 1):

S - bit de sinal, não importa,

E - $1111...1_2$,

F - diferente de zero.

O campo F pode conter informação dependente do sistema, como por exemplo: ponteiro para outras informações, ponteiro para a linha de código que causou a exceção, etc.

b) "Underflow": Ocorre "underflow" quando o expoente de um resultado é menor que o expoente mínimo do formato de destino, ou no caso de uma multiplicação ou divisão entre números não-nulos e finitos resultar em zero normal.

Caso não exista desvio definido pelo usuário, o resultado de um "underflow" é denormalizado deslocando seus bits significantes para a direita, enquanto o expoente é incrementado até que atinja o valor mínimo do formato de destino, sendo então arredondado o resultado de modo a caber no formato.

c) "Overflow": Ocorre "overflow" quando o expoente de um resultado arredondado de uma operação aritmética é maior que o máximo expoente do formato de destino. O resultado de uma operação onde ocorreu "overflow" é "infinito", que é representado pelos seguintes campos de bits (ver Figura 1):

S - bit de sinal,

E - $1111...1_2$,

F - 0.

d) "Divisão por zero": Ocorre divisão por zero quando um número finito não-nulo é dividido por um zero normal. O resultado de uma operação onde ocorre esta exceção é "infinito", com o sinal de acordo com a convenção.

e) "Resultado inexato": Esta exceção ocorre sempre que um resultado obtido necessita ser arredondado para caber no formato de destino. O resultado obtido é o número devidamente arredondado, de acordo com a opção do usuário.

2. OPERAÇÕES

Nesta seção serão descritos algoritmos para realizar as seguintes operações em ponto flutuante: adição, subtração, multiplicação, divisão, resto, raiz quadrada, seno, co-seno, tangente, arco seno, arco co-seno, arco tangente, exponencial, logaritmo neperiano, comparações e conversão de tipos.

Os algoritmos das rotinas serão descritos utilizando uma metalinguagem baseada na linguagem LP, a qual é descrita com detalhes por Luchesi et alii (1979). A seguir, apresenta-se uma descrição da metaliguagem utilizada na "Forma Normal de Backus":

```
<comando> ::=
      <variável> <-- <expressão> [, <expressão>]
    | DEVOLVA <expressão> [,  <expressão>]
    | SE <expressão> ENTÃO <comando> SENÃO <comando>
    | ENQUANTO <expressão> FAÇA <comando>
    | REPITA <comando>  [,<comando>]  ATÉ QUE <expressão>
    | PARA <variável> <-- <expressão> ATÉ <expressão> FAÇA
    | INÍCIO <comando> [, <comando>]  FIM
    | <texto informal> .
    | <comentário>
```

```
<declarações> ::=
      VAR <nome> [ , <nome> ] : <tipo> ;
      [ <declarações> ]

<procedimento> ::=
   PROCEDIMENTO <nome> () [ : <tipo> ] ;
     [ <declarações> ]
     <comando> ;
   PROCEDIMENTO <nome> (<nome> [, <nome> ] : <tipo>) [ : <tipo> ];
      [ <declarações> ]
      <comando> ;

<tipo> ::= <tipo inteiro> | <tipo ponto flutuante>
<tipo inteiro> ::= CURTO | LONGO
<tipo ponto flutuante> ::= SIMPLES | DUPLO | ESTENDIDO
<comentário> ::= (* <texto informal> *)
<texto informal> ::= Texto em língua portuguesa
```

A notação [x] indica a repetição do elemento x, zero ou mais vezes. Esta é a descrição da sintaxe da linguagem utilizada. Para que a descrição fosse completa seria necessário descrever também o significado de cada construção, isto é, a semântica da linguagem. Como a descrição dos algoritmos será de certo modo informal, não se considerou necessário entrar em detalhes semânticos.

Todas as rotinas serão calculadas utilizando internamente o formato estendido, tanto nos parâmetros de entrada como nos de saída, sendo o resultado posteriormente arredondado para o formato desejado.

Assim, todas as rotinas, com exceção das rotinas de conversão de tipos, além de seguir o algoritmo da função específica, seguem o seguinte algoritmo:

```
PROCEDIMENTO operação-X (P1, P2: <tipo p.f.>): <tipo p.f.>
VAR PE1, PE2: ESTENDIDO
INÍCIO
   SE (P1 = NaN) ou
      (P2 = NaN)
     ENTÃO ligar bit de operação inválida.
   PE1 <-- converte P1 para ESTENDIDO
   PE2 <-- converte P2 para ESTENDIDO
   RESULTADO <-- operação-X(PE1, PE2)
   RESULTADO <-- converte RESULTADO para <tipo p.f.>
   DEVOLVA RESULTADO
FIM do PROCEDIMENTO operação-X;
```

onde os parâmetros P1 e P2 podem ser dos tipos SIMPLES ou DUPLO e a operação-X é uma das operações citadas anteriormente.

Além disso, existe uma seqüencia de testes de exceção que deve ser realizada em quase todas as rotinas. Reuniram-se estes testes em um procedimento que pode ser invocado quando necessário. Seu algoritmo é o seguinte:

```
PROCEDIMENTO TESTE(Z : INTERMEDIÁRIO);
INÍCIO
   SE houve underflow
     ENTÃO Ligar bit de underflow
   SENÃO
   SE houve arredondamento
     ENTÃO Ligar bit de arredondamento
   SENÃO
   SE houve operação inválida
     ENTÃO Ligar bit de operação inválida
   SENÃO
   SE houve overflow
     ENTÃO Ligar bit de overflow.
FIM do PROCEDIMENTO TESTE;
```

Os algoritmos que descrevem as operações básicas (adição, subtração, multiplicação e divisão) em ponto-flutuante são descritos em termos de operações aritméticas inteiras entre campos de bits de representação em ponto-flutuante. Assim, é interessante definir os seguintes TIPOS utilizando, de maneira informal, a notação da linguagem PASCAL, de modo a ser possível a referência a uma parte de um determinado operando. Sejam as seguintes definições de tipos:

```
TIPO <tipo inteiro> : REGISTRO
                          S : sinal (1 bit)
                          M : magnitude (Nm bits)

TIPO <tipo p.f.> : REGISTRO
                     S : sinal (1 bit)
                     E : expoente (Ne bits)
                     F : parte significante (Nf bits)
                   FIM;

TIPO INTERMEDIÁRIO :
     REGISTRO
        S : sinal (1 bit)
        E : expoente (15 bits)
        F : REGISTRO
               V : overflow
               N : bit mais significante (1 bit)
               NL : bits significantes (62 bits)
               L : bit menos significante (1 bit)
            FIM;
        G : bit de arredondamento (1 bit)
        R : bit reserva (1 bit)
        O : "OU" lógico dos bits restantes no registrador
     FIM;
```

Assim, por exemplo, o sinal de um operador X do tipo DUPLO pode ser referido como X.S, e o bit de "overflow" de uma variável intermediária Z pode ser referido como Z.F.V. O tipo INTERMEDIÁRIO é utilizado somente no interior dos algoritmos, sendo sua estrutura inacessível ao usuário de uma dada rotina. Na verdade, o tipo INTERMEDIÁRIO apenas reflete o armazenamento em registradores de uma variável do tipo ESTENDIDO. Os campos S, E e F correspondem aos campos do mesmo nome do tipo ESTENDIDO, ocupando o mesmo número de bits.

Cada tipo de processamento é considerado associado a um determinado bit; assim nos algoritmos consideram-se os bits RN e RZ inicializados pelo usuário das rotinas.

2.1 - ADIÇÃO/SUBTRAÇÃO

Tanto a adição quanto a subtração serão realizados pelo mesmo algoritmo, precedido, na rotina de subtração, pela troca de sinal do operando adequado. O algoritmo de soma de dois operandos é o seguinte:

```
PROCEDIMENTO soma(X, Y : ESTENDIDO) : ESTENDIDO ;
VAR Z : INTERMEDIÁRIO;
INÍCIO
   SE (X = +0 ou -0) e
      (Y = +0 ou -0)
   ENTÃO   (* operandos zero *)
     INÍCIO
        Z.S <-- 0
        Z.E <-- 0
        Z.F <-- 0
     FIM
   SENÃO
   SE X = infinito ENTÃO Z <-- X
   SENÃO
   SE Y = infinito ENTÃO Z <-- Y
```

```
SENÃO
INÍCIO
   Alinhar os pontos binarios de X e Y
       desnormalizando o operando de menor
       expoente até que os expoentes sejam
       iguais.
   Z.F <-- X.F + Y.F
   SE X.S = Y.S   (* soma de magnitudes *)
     ENTÃO
       SE Z.F.V = 1   (* overflow na magnitude *)
         ENTÃO
           INÍCIO
              Deslocar Z.F um bit para a direita e
              incrementar o expoente Z.E.
              Z.0 <-- Z.0 ou Z.R   (* bit a bit *)
           FIM
   SENÃO (* subtração de magnitudes *)
     SE Z.F = 0   (* todos os bits significantes são 0 *)
       ENTÃO
         INÍCIO
            SE RN = 1 ou
               RZ = 1 ou
               RP = 1
               ENTÃO Z.S <-- 0
            SENÃO
            SE RM = 1 ENTÃO Z.S  <-- 1
            Z.F <-- 0 (caso os operandos estivessem
                       normalizados depois do alinhamento
                       do ponto decimal)
         FIM
       SENÃO
         Caso os operandos não estejam normalizados,
         normalizá-los deslocando os bits
         significativos (Z.F, Z.G, Z.R) para a esquerda
```

```
            decrementando o expoente (Z.E) até obter um 1 à
            esquerda.
            Ao deslocar, devem ser introduzidos 0's
            na parte direita do campo de bits
            significativos (Z.R).
         TESTE(Z)
      FIM
      DEVOLVA Z.S, Z.E, Z.F
FIM do PROCEDIMENTO soma;
```

2.2 - MULTIPLICAÇÃO/DIVISÃO

A multiplicação e a divisão serão realizadas pelos algoritmos apresentados a seguir.

Algoritmo para multiplicação:

```
PROCEDIMENTO multiplicação(X, Y : ESTENDIDO) : ESTENDIDO ;
VAR Z : INTERMEDIÁRIO;
INÍCIO
   SE (X = +0 ou -0)  ou
      (Y = +0 ou -0)
   ENTÃO   (* pelo menos um operando zero *)
     INÍCIO
        Z.F <-- 0
        SE X.S = Y.S
          ENTÃO Z.S <-- 0
          SENÃO Z.S <-- 1
     FIM
   SENÃO
   SE (X = infinito e Y = 0) ou
      (Y = infinito e X = 0) ou
      (X = zero não-normalizado) ou
      (Y = zero não-normalizado)
```

```
    ENTÃO
      INÍCIO
         Z <-- NaN
         Ligar bit de operação inválida.
      FIM
  SENÃO
  SE (X = infinito) ou
     (Y = infinito)
    ENTÃO
      INÍCIO
         Z <-- infinito
         Z.S <-- X.S ou-exclusivo Y.S  (*  bit a bit  *)
      FIM
  SENÃO
    INÍCIO
       Z.S <-- X.S ou-exclusivo Y.S  (*  bit a bit  *)
       Z.E <-- X.E + Y.E
       Z.F <-- X.F * Y.F
       SE Z.F.V = 1   (* overflow na magnitude *)
         ENTÃO Deslocar Z.F um bit para a direita
               e incrementar Z.E.
       TESTE(Z)
    FIM
  DEVOLVA Z.S, Z.E, Z.F
FIM do PROCEDIMENTO multiplicação;
```

Algoritmo para divisão:

```
PROCEDIMENTO divisão(X,Y : ESTENDIDO) : ESTENDIDO ;
VAR Z : INTERMEDIÁRIO;
INÍCIO
   SE (X = 0 e Y = 0)  ou
      (X = infinito e Y = infinito)  ou
      (X = zero não-normalizado)
```

```
  ENTÃO
    INÍCIO
       Z <-- NaN
       Ligar bit de operação inválida.
    FIM
SENÃO
SE (X = 0)  ou
   (Y = infinito)
  ENTÃO
    INÍCIO
       Z <-- zero
       Z.S <-- X.S ou-excluisvo Y.S  (* bit a bit *)
    FIM
SENÃO
SE (X = infinito)
  ENTÃO
    INÍCIO
       Z <-- infinito
       Z.S <-- X.S ou-exclusivo Y.S  (* bit a bit*)
    FIM
SENÃO
SE (Y = 0)
  ENTÃO
    INÍCIO
       Z <-- infinito
       Z.S <-- X.S. ou-exclusivo Y.S  (* bit a bit *)
       Ligar bit de divisão por zero.
    FIM
SENÃO
  INÍCIO
     Z.S <-- X.S ou-exclusivo Y.S  (* bit a bit *)
     Z.E <-- X.E - Y.E
     Z.F <-- X.F / Y.F
     ENQUANTO Z.F.N = 0 FAÇA
```

```
          INICIO
             Deslocar Z.F. para a direita
             decrementando o expoente.
             Durante o deslocamento Z.F.R pode receber
             ou 0 ou Z.F.O, alternativamente.
          FIM
        TESTE(Z)
     FIM
   DEVOLVA Z.S, Z.E, Z.F
FIM do PROCEDIMENTO divisão;
```

2.3 - RESTO

O resto será implementado através do seguinte algoritmo:

```
PROCEDIMENTO resto(X,Y : ESTENDIDO) : ESTENDIDO ;
VAR Q : <tipo inteiro>;
VAR Z : INTERMEDIÁRIO;
INÍCIO
   SE (Y = 0)  ou
      (X = inifinito) ou
      (Y = zero não-normalizado)
     ENTÃO
       INÍCIO
          Z <-- NaN
          Ligar bit de operação inválida.
       FIM
   SENÃO
   SE (X = 0) ou
      (Y = infinito)
     ENTÃO
       Z <-- X
   SENÃO
     INÍCIO
```

```
            Z <-- X/Y
            Q <-- Converter Z para inteiro.
                  (* Inteiro mais próximo de X/Y,
                     Se X/Y ficar entre dois inteiros
                     utilize o inteiro par.
                     Se Q contiver mais bits significantes
                     do que o formato de destino, descartar
                     os bits de mais alta ordem *)
            Z <-- X - (Q*Y)
          FIM
        DEVOLVA Z.S, Z.E, Z.F
FIM do PROCEDIMENTO resto;
```

2.4 - RAIZ QUADRADA

A raiz quadrada de um número N ($N \geq 0$), no formato em ponto-flutuante, é calculada usando o método de NEWTON-RAPHSON. O palpite inicial será 2**(N.E/2). Para a precisão simples, quatro iterações são suficientes, enquanto para a precisão dupla, cinco iterações são necessárias. A raiz quadrada será implementada através do seguinte algoritmo:

```
PROCEDIMENTO raiz-quadrada(X : ESTENDIDO) : ESTENDIDO;
VAR Z : INTERMEDIÁRIO;
VAR ITERAÇÕES : CURTO;
INÍCIO
   ITERAÇÕES=5  (*número de iterações *)
   SE (X = 0) ou
      (X = +infinito) ou
      (X = NaN)
     ENTÃO
       Z <-- X
   SENÃO
   SE (X.S = 1)
     ENTÃO
```

```
      INÍCIO
         Ligar bit de operação inválida.
         Z <-- NaN
      FIM
   SENÃO
     INÍCIO
        Z <--- 2**(X.E)/2
        PARA I <--- 1 ATÉ ITERAÇÕES FAÇA
          Z <-- ( (Z*Z) + X)/(2 * Z)
        TESTE(Z)
     FIM
   DEVOLVA Z.S, Z.E, Z.F
FIM do PROCEDIMENTO raiz-quadrada;
```

2.5 - FUNÇÕES TRIGONOMÉTRICAS

As funções trigonométricas podem ser calculadas por uma série de algoritmos: para cada função é sugerido um algoritmo cujas características de tempo e eficiência são aceitáveis. Muitos dos compiladores implementados em microprocessadores usam estes algoritmos. São os seguintes os algoritmos para cada uma das funções:

2.5.1 - FUNÇÃO SENO

Esta rotina calcula o valor do seno de um número x. O valor de x deverá ser dado em radianos. O seno de um ângulo X é calculado usando aproximação polinomial. Para precisão simples, seis coeficientes são suficientes, enquanto na precisão dupla são necessários oito coeficientes. O algoritmo para seno é o seguinte:

```
PROCEDIMENTO seno(X : ESTENDIDO) : ESTENDIDO;
VAR
   Y, Y2, Y4, A1, A2, A3, A4, T1, T2, Z : ESTENDIDO;
INÍCIO
```

```
Y <--- ABS(X)
SE ( Y > 2*PI)
  ENTÃO Y <--- Y MOD (2*PI)
Reduz Y para que fique no intervalo 0 ≦ Y ≦ (PI/2).
Y3 <--- Y*Y*Y
Y4 <--- Y3*Y
A1 <--- 1/FATORIAL(5)
A2 <--- 1/FATORIAL(9)
A3 <--- 1/FATORIAL(13)
A4 <--- 1/FATORIAL(17)

T1 <--- Y*(1 + Y4*(A1 + Y4*(A2 + Y4*(A3 + Y4*A4))))
A1 <--- 1/FATORIAL(3)
A2 <--- 1/FATORIAL(7)
A3 <--- 1/FATORIAL(11)
A4 <--- 1/FATORIAL(13)

T2 <--- Y3*(A1 + Y3*(A2 + Y3*(A3 + Y3*A4)))
Z <--- T1 - T2
```

Ajusta o sinal de acordo com o quadrante a qual pertence X.

```
SE ( X < 0)
  ENTÃO Z <--- -Z
  DEVOLVA Z
FIM do PROCEDIMENTO seno
```

2.5.2 - FUNÇÃO CO-SENO

Esta rotina calcula o valor do co-seno de um número X. O valor de X deverá ser dado em radianos. O co-seno de um ângulo X é calculado usando o próprio algoritmo do seno, pois, co-seno(X) = seno (X + PI/2).

Por este algoritmo o co-seno é calculado utilizando o mesmo algoritmo polinomial (método de Krammer) utilizado para o seno, sendo apenas deslocado o valor do ângulo.

```
PROCEDIMENTO co-seno(X : ESTENDIDO) : ESTENDIDO;
VAR
   Y, Z : ESTENDIDO;
INÍCIO
    Y <--- ABS(X)
    Y <--- Y + (PI/2)
    Z = ABS( seno(Y) )
    Ajusta o sinal de acordo com o quadrante a qual pertence X.
    DEVOLVE Z
FIM do PROCEDIMENTO co-seno
```

2.5.3 - FUNÇÃO TANGENTE

A tangente é calculada primeiro reduzindo o seu ângulo para [0, PI]. Esta região é dividida em quatro octantes do círculo nos pontos PI/4, PI/2 e 3*PI/4. Estes octantes são numerados de 1 a 4, respectivamente. A seguinte redução é feita de acordo com o octante a que pertence o ângulo x:

octante	Y (redução para [0, PI/4])
1	x
2	PI/2 - x
3	x - PI/2
4	PI - x

A faixa da redução é $0 < y \leq PI/4$. Uma aproximação para a tangente T, baseada na fração contínua racionalizada e truncada, é usada e, depois de calculado T, o valor da tangente é o que se segue de acordo com o octante a que pertence o ângulo x:

octante	tangente(x)
1	Y/T
2	T/Y
3	-T/Y
4	-Y/T

A função tangente será calculada utilizando o seguinte algoritmo polinomial:

```
PROCEDIMENTO tangente(X : ESTENDIDO) : ESTENDIDO;
VAR
   Y, Y2, C1, C2, C3, C4, C5, C6, T, OCTANTE, Z : ESTENDIDO;
INÍCIO
   Y <--- ABS(X)
   SE (Y > 2*PI)
     ENTÃO Y <--- Y MOD (2*PI)
   SE (Y <> PI/2 e Y <> 3*PI/2)
     ENTÃO
       INÍCIO
         Reduz Y para que fique no intervalo [0, PI].
         SE Y <> 0 e Y <> PI
           ENTÃO
             INÍCIO
               (* nesta região [0, PI], determinar a que octante *)
               (* pertence o ângulo Y *)
               SE (Y > 3*PI/4 e Y ≤ PI)
                 ENTÃO INÍCIO (* quarto octante *)
                         OCTANTE <--- 4
                         Y <--- PI - Y
                       FIM
                 SENÃO SE (Y > PI/2 e Y ≤ 3*PI/4)
                         ENTÃO INÍCIO (* terceiro quadrante *)
                                 OCTANTE <--- 3
                                 Y <--- Y - PI/2
                               FIM
```

```
            ENTÃO Z <-- -Z
            DEVOLVE Z.
          FIM
    SENÃO INÍCIO
            Liga bit de operação inválida.
            Z <--- NaN
          FIM
FIM do PROCEDIMENTO arco-seno.
```

2.5.5 - FUNÇÃO ARCO CO-SENO

O arco co-seno de um valor X (que pertence a [-1, 1]) é calculado usando o próprio algoritmo do arco-seno, pois arco co-seno (X) = PI/2 - arco-seno(X). O arco co-seno será calculado utilizando o seguinte algoritmo:

```
PROCEDIMENTO arco co-seno(X : ESTENDIDO) : ESTENDIDO;
VAR Z : ESTENDIDO;
INÍCIO
  (SE (X estiver dentro da faixa -1..1)
     ENTÃO
       Z <--- PI/2 - arco-seno(X)
     SENÃO
       INÍCIO
          Liga bit de operação inválida
          Z <-- NaN
       FIM
     DEVOLVE Z
FIM do PROCEDIMENTO arco co-seno.
```

2.5.6 - FUNÇÃO ARCO TANGENTE

O arco tangente será calculado utilizando o seguinte algoritmo:

```
PROCEDIMENTO arco-tangente(X:ESTENDIDO): ESTENDIDO;
VAR
   I, ICASO, IFLAG, ITERAÇÕES: CURTO;
   Y, T, Z: ESTENDIDO;
INICIO
  Y <-- ABS(X)
  ITERAÇÕES <-- 11
  (* redução se Y > 1 *)
  SE ( Y > 1 )
    ENTÃO INÍCIO
            Y <-- 1/Y
            IFLAG <-- 1
          FIM

  (* redução se tangente(PI/20) < Y ≦ tangente(3*PI/20) *)
  SE ( tangente(PI/20) < Y ≦ tangente(3*PI/20) )
    ENTÃO INÍCIO
            Y <-- (Y - tangente(PI/10))/(1 + Y * tangente(PI/10))
            ICASO <-- 1
          FIM

  (* redução se tangente(3*PI/20) < Y ≦ 1 *)
  SE ( tangente(3*PI/20) < Y ≦ 1)
    ENTÃO INÍCIO
            Y <-- (Y - tangente(PI/5) )/(1 + Y * tangente(PI/5))
            ICASO <-- 2
          FIM
  (* calcule o arco tangente pela série interpolada de Chebyshev *)
  T <-- 0
  PARA I <-- 0 ATÉ ITERAÇÕES FAÇA
    T <-- T * (-1)**I * ( (Y ** (2*I+1)/(2*I+1) )
    SE ( ICASO = 1 )
      ENTÃO (* tangente(PI/20) < Y ≦ tangente(3*PI/20) *)
            T <-- PI/10 + T
```

```
      SENÃO (* tangente(3*PI/20) < Y ≤ 1 *)
            T <-- PI/5 + T
    SE ( IFLAG = 1 )
      ENTÃO T <PI/2 - T
    Z <-- T
    SE ( X < 0 )
      ENTÃO Z <-- -Z
    DEVOLVE Z
FIM do PROCEDIMENTO arco-tangente.
```

2.6 - EXPONENCIAIS

A função exponencial calcula e**X, onde e = 2.71828... O cálculo é feito em base dois, como se segue:

$$e^{**}X = (X * \log2(e))$$

$$= 2^{**}Y \quad (Y = X * \log2(e)),$$

onde log2(e) é uma constante. Para calcular o valor de (2 ** Y), o número Y é dividido em sua parte inteira e em sua parte fracionária. No caso, têm-se:

$$Y = (2^{**}Y.E)*(1.Y.F)$$

$$= (2^{**}Y.E)*(1 + 0.Y.F)$$

$$= (2^{**}Y.E) + (2^{**}Y.E)*(0.Y.F),$$

$$(2^{**}Y) = 2^{**}(2^{**}Y.E + (2^{**}Y.E)*(0.Y.F))$$

$$= (2^{**}Y.E) * ((2^{**}Y.E)^{**}(0.Y.F)).$$

Então:

Y2 * (2 ** Y.E) - calculado por deslocamento

Y3 = (2 ** Y2) - calculado por deslocamento

Y4 = (Y3 ** 0.Y.F) - calculado por aproximação racional

usando a seguinte fórmula:

Y4 = (P + Q)/(P - Q), onde

P = C1 + C2 * (Y.F)**2 + C3 * (Y.F)**4,

Q = C4 * (Y.F) + C5 * (Y.F)**3 + C6 * (Y.F)**5

e os valores C1, C2, C3, C4, C5 e C6 são as constantes listadas no algoritmo:

e ** X = 2 ** Y = Y3 * Y4.

O algoritmo é o seguinte:

```
PROCEDIMENTO exponencial(X: ESTENDIDO): ESTENDIDO;
VAR
   Y1, Y2, Y3, Y4, C1, C2, C3, C4, C5, C6, P, Q, Z: ESTENDIDO;
INÍCIO
   Y1 <--- X ** log2(e)
   Y2 <--- 2 ** Y1.E  (* calcula usando a operação deslocamento *)
   Y3 <--- 2 ** Y2    (* calcula usando a operação deslocamento *)
   (* cálculo do Y3 ** 0.Y1.F *)
   F2 <--- Y1.F * Y1.F
   (* coeficientes para usar o método de aproximação racional *)
   C1 <--- 189561.9508
   C2 <--- 10118.4316448
   C3 <--- 43.3583754205
   C4 <--- 65697.1658690
```

```
   C5 <--- 876.41440418
   C6 <--- 1.0

   P <--- C1 + F2*(C2 + F2*C3)

   Q <--- Y1.F*(C4 + F2*(C5 + F2*C6)

   Y4 <--- (P + Q)/(P - Q)

   (* cálculo do e** X *)

   Z <-- Y3 * Y4

   DEVOLVE Z
FIM do PROCEDIMENTO exponencial.
```

2.7 - LOGARITMOS

Os logaritmos serão calculados na base "e" e será utilizado o seguinte algoritmo:

```
PROCEDIMENTO logaritmo(X : ESTENDIDO):ESTENDIDO;
VAR
   Y, N, M, TERMO1, Z, T: ESTENDIDO;
   I, ITERAÇÕES: CURTO;
   INÍCIO
     SE ( X > 0 )
       ENTÃO INÍCIO
               ITERAÇÕES <-- 12
               Y <-- X
               (* se Y < 1 fazer Y <-- 1/Y *)
               SE ( Y < 1 )
                 ENTÃO Y <-- 1/Y
               (* achar o primeiro N tal que (2**N) > Y *)
```

```
            N <-- 0
            ENQUANTO ( Y ≥ 2**N) FAÇA
              N <-- N + 1
            (* calcula o primeiro termo da expressão *)
          TERMO1 <-- N * logaritmo(2)
                     (* logaritmo(2) é constante *)
          M <-- Y/2**N   (* 0.5 ≤ 1 *)
          (* cálculo do termo logaritmo(M) por Taylor *)
          T <-- (M - 1)/(M + 1)
          TERMO2 <-- 0
          PARA I <-- 0 ATÉ ITERAÇÕES FAÇA
            TERMO2 <-- TERMO2 + (T ** (2*I+1)/(2*I+1)
          Z <-- TERMO1 + 2 * TERMO2
          (* acerta sinal se X < 1 *)
          SE ( X < 1 )
            ENTÃO Z <-- -Z
        FIM
SENÃO INÍCIO
        Liga o bit de operação inválida.
        Z <-- Código de valor inválido.
      FIM
DEVOLVE Z
FIM do PROCEDIMENTO logaritmo.
```

2.8 - COMPARAÇÕES

As comparações entre números de ponto-flutuante serão feitas utilizando as mesmas rotinas de comparação de números inteiros, pois devido ao tipo de formato utilizado a ordem dos números em ponto-flutuante é a mesma de números inteiros representados em sinal magnitude numa cadeia de bits de igual comprimento.

2.9 - CONVERSÃO DE TIPOS

As operações de conversão dos tipos em ponto-flutuante (SIMPLES e DUPLO) para os tipos inteiros (CURTOS e LONGOS) utilizam o seguinte algoritmo:

```
PROCEDIMENTO pflut-inteiro(X:<tipo p.f.>):<tipo inteiro>;
VAR Z : INTERMEDIÁRIO;
INÍCIO
   SE X = 0
     ENTÃO Z <-- 0
   SENÃO
   SE X = infinito ou
      X = NaN
     ENTÃO
       INÍCIO
          Ligar bit de operação inválida.
          Z <-- X
       FIM
   SENÃO
     INÍCIO
        Z.F <-- 1.X.F
        Z.E <-- X.E
        Z.S <-- X.S
        SE Z.E > tamanho de Z.F (* não há dígitos
                                   significantes *)
          ENTÃO Z <-- X
        SENÃO
          INÍCIO
             Deslocar Z.F para a direita enquanto
               incrementa Z.E até nenhum bit de Z.F
               estar fora da precisão de arredondamento
               utilizada.
```

```
            Arredondar Z.F (* devem estar disponíveis
               arredondamentos para o inteiro mais
               próximo e por corte *).
            SE Z.F = 0
              ENTÃO Z <-- 0
            SENÃO Normalizar Z.
          FIM
      FIM
   DEVOLVE Z.F (* truncado para a precisão de <tipo inteiro> *)
FIM;
```

As rotinas de conversão dos tipos inteiros para os tipos em ponto-flutuante são mais simples, uma vez que a representação inteira é mais simples. O algoritmo utilizado é o seguinte:

```
PROCEDIMENTO inteiro-pflut(X:<tipo inteiro> ) : <tipo p.f.>;
VAR Z : INTERMEDIÁRIO;
INÍCIO
   SE número de bits de X.M >
      número de bits de Z.F
     ENTÃO
       INÍCIO
          Z.F <-- X.M (* arredondado *)
          Ligar bit de resultado inexato.
       FIM
   SENÃO Z.F <-- X.M
   Normalizar Z.
   DEVOLVE Z.S, Z.E, Z.F   (* truncado para a precisão  <tipo p.f.> *)
FIM;
```

As conversões entre os tipos em ponto-flutuante e o tipo ESTENDIDO (e INTERMEDIÁRIO) não são realizadas através de rotinas, pois, uma vez que as variáveis do tipo estendido são sempre residentes em registradores, ou posições da pilha, as conversões que envolvem este tipo são dependentes do microprocessador a ser utilizado e são

comentadas na Seção 4. Estas conversões são utilizadas apenas internamente às rotinas, não sendo acessíveis ao usuário.

3. ESQUEMA DE COMUNICAÇÃO COM O COMPILADOR

Para o microprocessador MC 68000, os parâmetros para as operações de ponto-flutuante serão passados em registradores determinados, sendo o resultado armazenado também em registradores dados.

Já para o microprocessador IT 8086, os operadores para as rotinas serão passados no topo da pilha, sendo o resultado colocado em registradores a serem determinados.

4. COMENTÁRIOS ESPECÍFICOS PARA IMPLEMENTAÇÃO

Nesta seção comentam-se problemas que podem surgir durante a implementação das rotinas descritas na Seção 2 nos microprocessadores específicos MC 68000 e IT 8086.

4.1 - MICROPROCESSADOR MOTOROLA 68000

Este microprocessador apresenta uma arquitetura homogênea, o que facilita a implementação dos algoritmos aritméticos. São comentados diversos aspectos da implementação. Inicialmente, descreve-se a localização dos dados no tipo ESTENDIDO, comentam-se os procedimentos de arredondamento e o tratamento de exceções. A seguir, descreve-se a maneira de implementar os procedimentos da Seção 2 como sub-rotinas na linguagem de montagem do MC 68000, segue-se a descrição de como implementar cada comando da metalinguagem utilizada na descrição das rotinas e, finalmente, comentam-se mais alguns detalhes úteis na implementação das operações entre campos de bits.

4.1.1 - O TIPO ESTENDIDO

Todas as operações deste pacote, exceto as conversões de tipo discutidas na Seção 4.1.6, recebem dois parâmetros de tipo SIMPLES ou DUPLO e são efetuadas utilizando operandos do tipo ESTENDIDO. Assim, é necessária a existência de procedimentos para a conversão entre os tipos SIMPLES e DUPLO para o tipo ESTENDIDO e vice-versa. Neste tipo de microprocessador as rotinas recebem parâmetros, devolvem resultados e realizam operações utilizando os registradores de dados (D0 e D7).

Sendo assim, para a conversão de operandos dos tipos SIMPLES e DUPLO para o tipo ESTENDIDO, é necessário deslocar os campos S, E e F (ver Figura 1) de um registrador Di (para o tipo SIMPLES) ou de dois registradores Di e Di+1 (para o tipo DUPLO) para uma seqüência de registradores Dj, Dj+1 e Dj+2. Para tal, é necessária a existência de "máscaras", de modo a ser possível separar cada campo a ser deslocado. A declaração destas "máscaras" na linguagem de montagem do MC 68000 é a seguinte:

```
EQU SS "10000000000000000000000000000000" ; SINAL SIMPLES OU DUPLO
EQU SE "01111111100000000000000000000000" ; EXPOENTE SIMPLES
EQU SF "00000000011111111111111111111111" ; FRAÇÃO SIMPLES

EQU DE "01111111111100000000000000000000" ; EXPOENTE DUPLO
EQU DF "00000000000011111111111111111111" ; FRAÇÃO DUPLA
```

Utilizando estas "mascaras" é possível fazer a conversão do tipo SIMPLES para o tipo ESTENDIDO através do seguinte algoritmo:

```
Dj <-- Di
Dj <-- Dj and SE
Deslocar Dj 7 posições para a direita
Dj+1 <-- Di
Dj+1 <-- Dj+1 and SS
Dj <-- Dj+1 or Dj
```

```
Dj+2 <-- Di
Dj+2 <-- Dj+2 and SF
Deslocar DJ+2 16 posições para a esquerda
Dj+1 <-- Di
Dj+1 <-- Dj+1 and SF
Deslocar Dj+1 16 posições para a direita
```

A idéia do algoritmo anterior pode ser resumida nos seguintes passos, os quais devem ser realizados para cada campo do formato de origem:

1. mover o registrador origem para o registrador destino (instrução MOVE);

2. separar a parte desejada utilizando as "máscaras" (instruções AND e OR);

3. deslocar de modo que os limites do campo sejam adequados ao novo formato (instruções ASL e ASR).

Assim, para realizar a conversão do tipo DUPLO para o tipo ESTENDIDO deve-se escrever um algoritmo análogo ao descrito anteriormente, utilizando os três passos mencionados de maneira adequada. Devido ao número de registradores necessários para armazenar operandos destes tipos, a ordem de conversão e quais os registradores usados não são quaisquer. As conversões do tipo ESTENDIDO para os tipos em ponto-flutuante são análogas, no entanto envolvem arredondamento, o que será comentado na seção seguinte.

4.1.2 - ARREDONDAMENTO

Existem três tipos de arredondamento: um implícito na normalização de um número do tipo ESTENDIDO, um necessário na conversão de um número ESTENDIDO em um número nos formatos SIMPLES ou DUPLO e um na conversão de um número em ponto-flutuante em um número inteiro.

O primeiro tipo é feito diretamente pelas rotinas das funções básicas. Já os dois últimos são referidos nos algoritmos apenas pelo verbo "arredondar". Estes tratam do arredondamento de um número fracionário de n bits para um outro número fracionário com m < n bits. Devem estar disponíveis dois tipos de arredondamento: para o mais próximo (RN) e para zero (RZ). Devem ser declarados, utilizando a instrução DC.B que inicializa uma área de memória, os bits de controle de qual dos tipos de arredondamento devem ser utilizados.

Ambos os tipos de arredondamento podem ser facilmente implementados utilizando as instruções de deslocamento (ASL e ASR) e as instruções lógicas (OR e AND).

4.1.3 - TRATAMENTO DE EXCEÇÕES

Deve ser definida uma posição de cinco bits na memória, utilizando a instrução DS.B que reserva uma determinada área para representar os bits das cinco exceções possíveis. A alteração e verificação do estado destes bits podem ser realizadas utilizando as instruções de manipulação de bits (BTST, BSET, BCLR e BCHG).

O microprocessador MC 68000 contém um registrador de estado (SR) de 16 bits, dos quais oito são acessíveis ao usuário e três são disponíveis para utilização pelo usuário. Caso se deseje implementar "armadilhas" para algumas condições de exceção, estes bits podem ser utilizados através de instruções que geram interrupções por "software".

4.1.4 - SUB-ROTINAS

As chamadas de sub-rotinas na linguagem de montagem do MC 68000 são feitas pela instrução JSR e o retorno através da instrução privilegiada RTS. Todas as rotinas do pacote de ponto-flutuante para este microprocessador receberão os parâmetros em registradores de dados (D0 e D7). No caso de rotinas do tipo SIMPLES (parâmetros de 32 bits),

serão utilizados os registradores D0 e D1, sendo o resultado retornado no registrador D0. Para rotinas do tipo DUPLO (parâmetros de 64 bits) serão utilizados os registradores D0 e D1 para o primeiro parâmetro, e os registradores D2 e D3 para o segundo, sendo o resultado retornado nos registradores D0 e D1. Para rotinas do tipo ESTENDIDO (parâmetros de 80 bits), serão utilizados os registradores D0 e D1, sendo o resultado retornado no registrador D0. Para rotinas do tipo DUPLO (parâmetros de 64 bits) serão utilizados os registradores D0 e D1 para o primeiro parâmetro, e os registradores D2 e D3 para o segundo, sendo o resultado retornado nos registradores D0 e D1. Para rotinas do tipo ESTENDIDO (parâmetros de 80 bits), e todas as rotinas aritméticas do pacote são deste tipo, o primeiro parâmetro será recebido nos registradores D2, D3 e parte superior de D4, enquanto o segundo parâmetro será recebido nos registradores D5, D6 e parte superior do D7. As partes inferiores dos registradores D4 e D7 serão utilizadas pelos procedimentos de arredondamento. O resultado será retornado nos registradores D5, D6 e parte superior de D7.

Esta conversão leva em conta o fato de que as conversões ocorrem sempre entre o tipo ESTENDIDO e os tipos SIMPLES ou DUPLO, sendo por isto interessante colocar esta distribuição de registradores.

As variáveis que aparecem nas declarações VAR devem ser posições da memória delcaradas através da instrução DS.W que reserva um certo número de palavras endereçáveis por um símbolo. Para acessar estas variáveis utiliza-se a instrução MOVEM.

4.1.5 - IMPLEMENTAÇÃO DOS COMANDOS DA METALINGUAGEM

Nesta seção comenta-se a implementação dos comandos da metalinguagem utilizada na descrição dos algoritmos da Seção 2.

a) Atribuição: A atribuição, como já foi comentado, é realizada utilizando a instrução MOVEM, uma vez uqe as variáveis são posições na memória.

b) DEVOLVA: este comando foi comentado na seção anterior.

c) SE ENTÃO: este comando é implementado utilizando "labels" e as instruções de controle de programa condicionais (Bcc, DBcc e Scc).

d) ENQUANTO: idem ao item c.

e) REPITA: idem ao item c.

f) PARA ATÉ FAÇA: neste caso, além de "labels" e instruções de controle de programa, é necessário um contador, que será um dos registradores de endereço disponíveis (A0 e A7). O controle do contador pode ser feito utilizando as instruções aritméticas existentes.

g) INÍCIO/FIM: a implementação deste comando está implícita na colocação dos "labels" necessários aos comandos anteriores.

4.1.6 - OUTROS COMENTÁRIOS

Algo que ainda resta a comentar é a maneira de implementar as operações inteiras entre os campos dos registros definidos na Seção 2. Como estas operações são feitas sempre entre campos de operadores ESTENDIDOS, têm-se, no início da operação, os registradores D2 a D7 ocupados, restando D0 e D1 disponíveis.

O tamanho dos campos do formato ESTENDIDO foi escolhido de modo que o sinal e o expoente ocupem os primeiros 16 bits, ou seja, uma palavra da cadeia de 80 bits reservada para um número, restando assim exatamente quatro palavras para os dígitos significantes. Assim, utilizando apenas as instruções EXG (que troca os valores de dois registradores) e SWAP (que troca a palavra inferior com a superior de um dado registrador), é possível separar os campos deste formato, dispensando o uso de "máscaras" e deslocamentos. Uma vez que as instruções

aritméticas do MC 68000 permitem operandos de 8, 16 ou 32 bits, as operações entre campos são facilmente implementadas. Deve-se apenas atentar para as operações entre campos de mais de 32 bits, para as quais é necessário levar em conta a ordem de operação dos registradores e os bits de "carry" e "overflow".

Outro ponto a ser comentado é o caso das rotinas de conversão. As rotinas que tratam das operações aritméticas recebem sempre parâmetros do tipo SIMPLES, DUPLO ou ESTENDIDO e devolvem um resultado do mesmo tipo. As rotinas de conversão, no entanto, recebem apenas um parâmetro e devolvem um resultado de tipo diferente deste. A entrada e saída das rotinas de conversão devem seguir a Tabela 2.

TABELA 2

ENTRADAS E SAÍDAS DAS ROTINAS DE CONVERSÃO

CONVERSÃO	ENTRADA	SAÍDA
SIMPLES → inteiro	D7	D0
DUPLO → inteiro	D6+D7	D0
inteiro → SIMPLES	D7	D0
inteiro → DUPLO	D7	D0+D1

4.2 - MICROPROCESSADOR INTEL 8086

Este microprocessador apresenta um número bem menor de registradores, além destes contarem com apenas 16 bits. Devido a esta limitação a memória deve ser utilizada tornando os cálculos mais lentos. Nesta seção comentam-se os mesmos aspectos de implementação tratados na Seção 4.1.

4.2.1 - CONVERSÕES ENTRE TIPOS

Neste tipo de microprocessador as rotinas recebem os parâmetros na pilha e devolvem o resultado nos acumuladores (AX, BX, CX e DX).

Devido a limitações de espaço nos registradores, os operandos do tipo ESTENDIDO devem ser armazenados em memória. Assim, devem ser alocadas duas áreas de cinco palavras (de 16 bits) cada, chamadas daqui por diante E1 e E2, para armazenar estes operandos.

Deste modo, a conversão dos tipos SIMPLES e DUPLO para o tipo ESTENDIDO é um procedimento que retira os parâmetros da pilha e os armazena adequadamente nas posições E1 e E2 da memória. Para separar os campos dos formatos SIMPLES e DUPLO é necessária a utilização de "máscaras", cuja declaração é a seguinte:

```
SS "1000000000000000"  ; SINAL
SE "0111111110000000"  ; EXPOENTE SIMPLES
SF "0000000001111111"  ; FRAÇÃO SIMPLES

DE "0111111111110000"  ; EXPOENTE DUPLO
DF "0000000000001111"  ; FRAÇÃO DUPLO
```

O algoritmo para conversão de um parâmetro SIMPLES para um operando no formato ESTENDIDO, armazenado a partir da posição Ei da memória, é dado por:

```
BX  <-- POP Pilha
CX  <-- BX
AX  <-- BX
AX  <-- AX and SE
AX  <-- Deslocar AX 7 posições para a direita
BX  <-- BX and SS
AX  <-- AX or BX
```

```
Ei    <-- AX
CX    <-- CX and SF
Ei+3  <-- CX
AX    <-- POP Pilha
Ei+4  <-- AX
```

A idéia do algoritmo anteriormente citado pode ser resumida nos seguintes passos, que devem ser repetidos para cada palavra do parâmetro a ser convertido:

1. desempilhar uma palavra do parâmetro (instrução POP);

2. salvá-la em outros registradores, se for o caso (instrução MOV);

3. separar a parte desejada utilizando "máscaras" (instruções AND e OR);

4. deslocar, se necessário, de modo a ajustar ao novo formato (instruções SHR e SHL);

5. armazenar na posição de memória adequada (instrução MOV).

O algoritmo para o tipo DUPLO é análogo, bastando seguir os passos já apresentados. Como os resultados são devolvidos nos registradores, a passagem do tipo ESTENDIDO para os tipos SIMPLES e DUPLO é feita de maneira inversa, não esquecendo que neste caso os campos devem ser arredondados.

4.2.2 - ARREDONDAMENTO

Os arredondamentos são efetuados do mesmo modo como no MC 68000, utilizando instruções de deslocamento e lógicas. O tipo de armazenamento a ser usado deve ser fixado pela inicialização de 2 bits (RN, RZ) na memória.

4.2.3 - SUB-ROTINAS

As chamadas de sub-rotinas na linguagem de montagem do IT 8086 sao feitas através da instrução CALL e o retorno é feito através da instrução RET.

Quanto ao recebimento de parâmetros, neste caso, existem dois tipos de rotinas: as rotinas externas ou do usuário, que recebem parâmetros na pilha e devolvem o resultado em registradores, e as rotinas internas ou de funções, que recebem os parâmetros nas posições E1 e E2 da memória e devolvem o resultado na posição E1.

A Tabela 3 apresenta esta situação de maneira mais clara:

TABELA 3

LOCAL DE ARMAZENAMENTO DE PARÂMETROS DE SUB-ROTINA

TIPO DE ROTINA	ENTRADA	SAÍDA
SIMPLES	pilha	AX, BX
DUPLO	pilha	AX, BX, CX, DX
ESTENDIDO	E1 e E2	E1

As variáveis das declarações VAR, do mesmo modo que para o MC 68000, devem ser posições na memória reservadas anteriormente e acessadas através da instrução MOV.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAMOWITZ, M.; STEGUN, I.A. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. New York, N.Y. Dept. of Commerce, Dec. 1972.

BURROUGHS. *B6700 mathematical intrinsics*. Information manual. Detroit, MI, 1971.

COONEN, J.T. An implementation guide to a proposed standard for floating-point arithmetic. *Computer*, V(13):68-79, Jan. 1980.

HAYES, J.P. *Computer architecture and organization*. Auckland, McGraw-Hill Kogakusha, 1980.

KNUTH, D.E. *The art of computer programming*. Reading, MA, Addison-Wesley, 1973. v.2.

LUCHESI, C.L.; SIMON, I.; SIMON, J.; KOWALTOWSKI, T. *Aspectos teóricos de Computação*, Brasília, CNPq, 1979.

MOTOROLA. *MC 68000 microprocessor user's manual*. 2.ed. California, 1980.

RECTOR, R.; ALEXY, G. *The 8086 book*. Berkeley, CA, OSBORNE/McGraw-Hill, 1980.

TANENBAUM, A.S. *Structured computer organization*. Englewood Cliffs, NJ, Prentice-Hall, 1976.