UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

PONTO FLUTUANTE: ANÁLISE DE ERROS, PROCESSADORES E PROPOSIÇÃO DE UMA LINGUAGEM PARA ESTUDO DE CASOS

LIKISO HATTORI

CAMPINA GRANDE - PARAÍBA

OUTUBRO/1980

H366p Hattori, Likiso
Ponto flutuante : analise de erros, processadores e proposicao de uma linguagem para estudo de casos / Likiso Hattori. - Campina Grande, 1980.
149 f. : il.

Dissertacao (Mestrado em Ciencias) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Matematica Computacional 2. Calculos Computacionais - 3. Teoria dos Erros - 4. Dissertacao I. Queiroz, Bruno Correia da Nobrega, M.Sc. II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 519.6(043)

PONTO FLUTUANTE:

ANÁLISE DE ERROS, PROCESSADORES E PROPOSIÇÃO DE UMA LINGUAGEM PARA ESTUDO DE CASOS

LIKISO HATTORI

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS CURSOS DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DA PARAÍBA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

Aprovada por:

BRUNO CORREIA DA NÓBREGA QUEIROZ - M.Sc
- Presidente -

GIUSEPPE MONGIOVI - M.Sc
- Examinador -

IVAN ROCHA NETO - Ph.D
- Examinador -

CAMPINA GRANDE
ESTADO DA PARAÍBA - BRASIL
OUTUBRO/1980

A minha esposa *Lígia* pela participação e estímulo constante.

AGRADECIMENTOS

Ao Professor BRUNO CORREIA DA NÓBREGA QUEIROZ pela orientação, incentivo e acompanhamento atencioso em todo o desenvolvimento desta tese.

Ao Professor GIUSEPPE MONGIOVI, pela co-orientação, de parte deste trabalho e também pelas sugestões e críticas que muito contribuiram para o desenvolvimento de todo o trabalho.

Ao Departamento de Sistemas e Computação da Universidade Federal da Paraíba, pelo incentivo.

À UFBa, UFRN e UNICAP, que permitiram o uso de seus computadores sem ônus.

À MARIA DA GUIA M. GUIMARÃES, pelo esforço e paciência na datilografia deste trabalho.

RESUMO

O objetivo deste trabalho, é apresentar as idiossincrasias da aritmética de ponto flutuante, as restrições de implementação em *hardware* e *software*, a origem dos erros nos cálculos computacionais, o problema da instabilidade numérica dos métodos e finalmente uma linguagem interativa para ensino da teoria dos erros.

ABSTRACT

The object of this work, is to present the floating-point arithmetic idiosincrasies, the implementation restrictions in *hardware* and *software*, the errors origin in the computational calculus, the numeric instability problem of the methods and finally an interactive language for teaching errors theory.

INDICE

CAPÍTULO I

INTRODUÇÃO

Os métodos numéricos são indispensáveis para resolver determinados problemas da área tecnológica. Acontece que a maioria dos usuários tem pouco conhecimento de computadores digitais e também de análise numérica. Infelizmente, as publicações disponíveis em termos de métodos numéricos aplicados, não fazem um estudo detalhado das causas dos possíveis erros computacionais.

Segundo FORSYTHE [*FORSYTHE, 1970*] , há quatro propriedades do computador que são relevantes para o seu uso na solução numérica de problemas da álgebra e análise, e são causadoras de muito perigo. Essas propriedades são:

(a) O computador não usa o sistema de números reais, mas sim uma simulação denominada de *"sistema de números de ponto flutuante"*. Isto introduz o problema de

"round-off errors" (erros de arredondamentos);

(b) A velocidade de processamento do computador permite a solução de problemas muito grandes (que envolvem muitos cálculos), e frequentemente, mas não sempre, os problemas deste tipo tem respostas muito mais sensíveis as perturbações de dados do que problemas que envolvem poucos cálculos;

(c) Devido a alta velocidade de processamento do computador, muito mais operações podem ser realizadas por um preço bem mais razoável do que na era pré-computador. Em consequência, a instabilidade de muitos processos é notoriamente patenteada;

(d) Normalmente os resultados intermediários de uma computação são transparentes ao programador. Isto exige que o mesmo seja capaz de prever os possíveis erros no processo e, garantir que o processo não falhará antes de executá-lo.

Os problemas inerentes ao sistema de números de ponto flutuante serão explicados no Capítulo II. Considerando que muitos minicomputadores e a maioria dos microcomputadores não tem a capacidade de manipular com números de ponto flutuante, no Capítulo III apresentamos um método de como poderia ser adicionado esta facilidade. No Capítulo IV apresentamos algumas metodologias para fazer testes de validade dos resultados obtidos através de cálculos computacionais. No Capítulo V apresentamos um breve estudo sobre instabilidade numérica de dois métodos e, verificamos comparativamente a precisão de quatro computadores bastantes uti

lizados no Brasil. No Capítulo VI desenvolvemos uma linguagem interativa que permite ao programdor observar os resultados intermediários de uma computação, mais especificamente, como as operações aritméticas são realizadas internamente. Finalmente, no Capítulo VII apresentamos as conclusões do trabalho e algumas sugestões de continuidade.

CAPÍTULO II

FUNDAMENTOS DA ARITMÉTICA DE PONTO FLUTUANTE

2.1 - INTRODUÇÃO

O computador não usa o sistema de números reais e sim uma simulação denominada de *"sistema de números de ponto flutuante"* [*FORSYTHE, 1970*] .

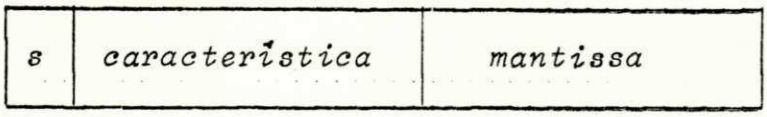
Um número de ponto flutuante pode ser representado da seguinte forma:

$$x = r^e m \qquad (2.1)$$

onde r - é a raiz ou base do sistema de números
(valores típicos são 2, 8, 10, 16),
e - um inteiro qualquer,

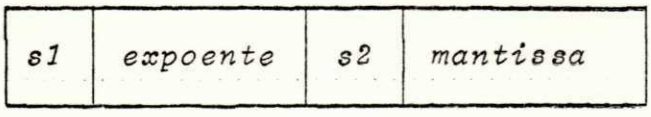
m - é uma fração positiva ou negativa

A representação interna de um número de ponto flutuante numa máquina, normalmente é dada da seguinte forma:



onde s denota o sinal do número armazenado, a característica é dada por $e + \gamma$ onde γ é um número inteiro positivo, sendo conhecido por excesso, variando de acordo com a estrutura do *hardware* e finalmente a mantissa é a fração m de (2.1) sem sinal, de ponto fixo, estando o ponto quase sempre a esquerda do dígito de mais baixa ordem.

A outra opção bastante difundida é a seguinte:



onde $s1$ e $s2$ são sinais do expoente e da mantissa, respectivamente.

O uso do expoente por excesso ou característica tem larga vantagem sobre o uso, puro e simples do expoente, pois facilita sobremaneira a execução de operações aritméticas e lógicas [*GSCHWIND, 1967*] .

Um número de ponto flutuante está normalizado se o dígito mais significativo damantissa é diferente de zero. O número

zero não pode ser normalizado porque não possui um dígito diferente de zero.

O uso da representação de ponto flutuante, aumenta o intervalo dos números que podem ser acomodados num dado registrador [*MANO, 1976*] . Por esta razão na maioria dos computadores as operações de ponto flutuante são realizadas em *hardware*, e quando não o são a nível de *hardware* o são a nível de *software*. Muitas linguagens de alto nível tem a facilidade de especificação de números de ponto flutuante. O mais comum é especificá-los por meio de uma declaração *REAL* diferenciando da especificação *INTEGER* para ponto fixo.

Devido a grande facilidade de uso dos computadores e das necessidades cada vez maiores nos meios científicos, a maioria dos seus usuários desconhecem os problemas pertinentes ao sistema de números de ponto flutuante, pois sua preocupação não vai além de como usar o magnífico instrumento capaz de fazer verdadeiros milagres.

2.2 - Sistemas de Números de Ponto Flutuante

O sistema de números reais é um dos triunfos da mente humana [*FORSYTHE, 1970*] . Sob este sistema podem ser efetuados cálculos e altas análises que em outros sistemas seriam impossíveis. De todas as maneiras de simular o sistema de números reais no computador, uma classe é largamente usada atualmente - o sistema de números de ponto flutuante.

2.2.1 - Representação dos Sistemas de Números de Ponto Flutuante

Representaremos genericamente por:

$$PF(r,p,a)$$

onde,

- r - é a raiz ou base do sistema de números sendo sempre um inteiro maior ou igual a 2;
- p - mostra a precisão, e designa o número de dígitos na base r contidos na mantissa;
- a - especifica os detalhes de como a aritmética será executada, podendo ser substituido por vários símbolos, tais como c para aritmética de corte, R para a aritmética de arredondamento, etc.

$PF(r,p,a)$ é composto de um conjunto S de números, que chamaremos de números de ponto flutuante e da definição das operações aritméticas de adição, subtração, multiplicação e divisão de números de ponto flutuante definidas por elementos de S. O conjunto S depende de r e p, mas não de a. Quando r e p são fixados pelo contexto, escreveremos $S(r,p)$ em lugar de S. O conjunto $S(r,p)$ contém zero e todos os números da forma

$$x = r^e m,$$

com a condição

$$r^{-1} \leq |m| < 1 \qquad (2.2)$$

cujo valor absoluto pode ser expresso na base r, usando no máximo p dígitos. Isto é,

$$|m| = r^{-p}M$$

onde M é um inteiro contido no intervalo $r^{p-1} \leq M < r^{p}$. Os números m e e são chamados de mantissa e expoente, respectivamente.

Considerando que todo número de ponto flutuante é um número real, podemos executar operações aritméticas de adição, subtração, multiplicação e divisão sobre os elementos de S visto como números reais. Entretanto, é bastante possível que estas operações produzam números que não estejam em S. Uma vez que os resultados da aritmética de ponto flutuante devem estar contidos em S, as operações de ponto flutuante produzem resultados que podem diferir dos resultados produzidos pela aritmética no campo dos números reais. Portanto, para a aritmética de ponto flutuante, utilizaremos os operadores $\oplus$, $\ominus$, $*$, $\div$ que representam a adição, subtração, multiplicação e divisão respectivamente. Então temos:

$$\begin{aligned} x * y &\approx xy \\ x \div y &\approx x/y \\ x \oplus y &\approx x + y \\ x \ominus y &\approx x - y. \end{aligned}$$

Na representação $PF(r,p,a)$ foram omitidos os seguintes detalhes da representação de números de ponto flutuante:

(a) Limites do intervalo de definição do expoente, que sempre existem, mas são particulares a cada máquina;

(b) Variações na forma como os números negativos são armazenados, assumindo o sinal negativo e o valor correto da magnitude da mantissa;

(c) Não permitimos variações onde o ponto da raiz deve ficar na mantissa de um número de ponto flutuante, assumindo que o mesmo reside a esquerda da mantissa.

2.2.2 - Aritmética de Corte para Números de Ponto Flutuante

Representaremos por $PF(r,p,c)$ o sistema de números de ponto flutuante onde as operações $\oplus$, $\ominus$, $*$, $\div$, são efetuadas em aritmética de corte (com truncamento) [*McCRAKEN, 1978*].

Seja x um número real, então $\overline{x}$ denota x cortado para p dígitos na base r. Para qualquer número real x, seja T o conjunto de todos os números y em $S(r,p)$ com $|y| \leq |x|$. Então $\overline{x}$ é o elemento de T que é fechado para x. Para efetuar operações aritméticas em $PF(r,p,c)$, primeiro executamos no sistema de números reais, e então cortamos o resultado para p dígitos na base r. Portanto,

$$x \oplus y = \overline{x + y}$$
$$x \ominus y = \overline{x - y}$$
$$x * y = \overline{xy}$$
$$x \div y = \overline{x/y}.$$

2.2.3 - Aritmética de Arredondamento para Números de Ponto Flutuante

Representaremos por $PF(r,p,R)$ o sistema de números de ponto flutuante onde as operações $\oplus$, $\ominus$, $*$, $\div$ são efetuadas em aritmética de arredondamento. O arredondamento quase sempre adotado é o denominado de simétrico. Seja x um número qualquer, e $\overline{x}^{\circ}$ o número x arredondado para p dígitos na base r. Isto é, $\overline{x}^{\circ}$ é um número em $S(r,p)$ que é fechado para x. Para executar operações aritméticas em $PF(r,p,R)$, primeiro executamos no sistema de números reais, e então arredondamos os resultados para p dígitos na base r. Portanto em $PF(r,p,R)$ temos:

$$x \oplus y = \overline{x + y}^{\circ}$$
$$x \ominus y = \overline{x - y}^{\circ}$$
$$x * y = \overline{xy}^{\circ}$$
$$x \div y = \overline{x/y}^{\circ}$$

2.2.4 - Leis da Álgebra

As manipulações algébricas de fórmulas estão baseadas na validade de poucas leis fundamentais. Especificamente, nós apegamos ao fato de que os número reais formam um campo. Isto significa que a soma e o produto de números reais estão definidos e que as leis: Do fechamento, comutativa, associativa, distributiva, existência do elemento neutro e do elemento simétrico são válidos para quaisquer números reais a, b e c.

Definiremos a subtração por:

$$a - b = a + (-b),$$

e se $b \neq 0$, definiremos a divisão por

$$\frac{a}{b} = ab^{-1}.$$

Duas consequências destas definições, com a lei comutativa e a lei associativa são:

$$(a + b) - b = a \tag{2.3}$$

e

$$b\left(\frac{a}{b}\right) = a. \tag{2.4}$$

O problema agora é saber se estas leis são válidas para o sistema de números de ponto flutuante. A validade destas leis dependem dos detalhes de como a aritmética é executada, assim estudaremos apenas o sistema específico $PF(r,p,c)$.

O teorema seguinte é uma consequência imediata das definições e validade da lei comutativa nos números reais.

Teorema 2.1 - Em $PF(r,p,c)$, a soma $a \oplus b$ e o produto $a*b$ de dois números de ponto flutuante a e b são números de ponto flutuante. Assim, para quaisquer a e b em $S(r,p)$ temos,

$$a \oplus b = b \oplus a$$

$$a * b = b * a$$

$$a \oplus 0 = 0 \oplus a = a$$

$$a * 1 = 1 * a = a$$

$$a \oplus (-a) = (-a) \oplus a = 0.$$

Mostraremos agora que as leis da associatividade e da distributividade além de *(2.3)* e *(2.4)* falham em $PF(r,p,c)$ para combinações não triviais de r e p.

2.2.4(a) - Falha da Lei Associativa da Adição

Mostraremos que

$$(a \oplus b) \oplus c = a \oplus (b \oplus c),$$

falha em $PF(r,p,c)$. Seja $a = r^{-2p}$, $b = 1$, $c = -1$. Então,

$$a \oplus b = \overline{1 + r^{-2p}} = 1,$$

assim $$(a \oplus b) \oplus c = 1 + (-1) = 0$$

Mas $$a \oplus (b \oplus c) = r^{-2p} \oplus 0 = r^{-2p}$$

então a lei falha, visto que $r^{-2p} \neq 0$.

Se a lei associativa da adição falha, consequentemente (2.3) também falhará.

2.2.4(b) - Falha da Lei Associativa da Multiplicação

Mostraremos que

$$(a * b) * c = a * (b * c)$$

falha em $PF(r,p,c)$, exceto para combinações triviais de r e p. Seja $s = p - 1$, assumimos que

$$4r^{-2s} \leq r^{-p}$$

é verdadeiro para todas as combinações de r e p. Seja $a = b = 1 + r^{-s}$ e $c = 1 - 2r^{-s}$. Então,

$$\begin{aligned}(a * b) * c &= \overline{1 + 2r^{-s} + r^{-2s}} * c \\ &= (1 + 2r^{-s}) * (1 - 2r^{-s}) \\ &= \overline{1 - 4r^{-2s}} \\ &= 1 - r^{-p}.\end{aligned}$$

Mas

$$\begin{aligned}a * (b * c) &= a * \overline{1 - r^{-s} - 2r^{-2s}} \\ &= (1 + 2r^{-s}) * (2r^{-s}) \\ &= \overline{1 - r^{-p} - r^{-2s} - r^{-(s+p)}} \\ &= 1 - 2r^{-p}.\end{aligned}$$

Portanto, a lei associativa da multiplicação falha em $PF(r,p,c)$ se $4r^{-2s} \leq r^{-p}$, isto é, se $4 \leq r^{p-2}$.

2.2.4(c) - Falha da Lei do Cancelamento

Mostraremos que a lei do cancelamento falha em $PF(r,p,c)$, exceto para combinações triviais de r e p, encontraremos valores de a, b e c em $S(r,p)$ tais que $a \neq 0$, $b \neq c$ e $a*b = a*c$. Para este fim, primeiro consideraremos o caso em que $r > 2$ $p \geq 2$. Seja $a = 2$, $b = r - 1$, e $c = r - 1 + r^{-(p-1)}$. Então $b \neq c$, mas

$$a * b = r + r - 2$$

e

$$a * c = \overline{r + r - 2 + 2r^{-(p-1)}}$$

$$= r + r - 2$$

Portanto, $a * b = a * c$, de forma que a lei do cancelamento falha.

2.2.4(d) - Falha da Lei Distributiva

Mostraremos que a lei distributiva

$$a * (b \oplus c) = (a * b) \oplus (a * c)$$

falha em $PF(r,p,c)$ para certos valores de r e p. Primeiro, suponha que $r > 2$ e $p \geq 2$. Seja $a = 1$, $b = r-1+r^{-(p-1)}$, e

$c = (r - 1)r^{-(p - 1)}$. Então

$$a * b = (r - 2)r + 1$$

e

$$a * c = \left[(r - 2)r + 1\right] r^{-(p - 1)}$$

assim

$$(a * b) \oplus (a * b) = \overline{(r-2)r + 1 + (r-2)r^{-(p-2)} + r^{-(p-1)}}$$

$$= (r-2)r + 1 + (r-2)r^{-(p-2)}.$$

Mas

$$a * (b \oplus c) = (r-1)\ (r-1 + r^{-(p-2)})$$

$$= (r-2)r + 1 + (r-1)r^{-(p-2)}$$

comprovando que a lei distributiva falha.

2.2.4(e) - Falha da Relação $a * (b \div a) = b$

Sejam a e b positivos e $c = b \div a$, em $PF(r,p,c)$. Então $c = \overline{b/a}$, assim a menos que b/a possa ser expresso em p dígitos na base r, temos $c < b/a$. Então $\overline{ac} < b$, assim

$$a * (b \div a) = \overline{ac} < b.$$

Portanto, a relação

$$a * (b \div a) = b \qquad (2.5)$$

será válida se e somente se $b \div a = b/a$.

2.2.5 - Desigualdades em $PF(r,p,c)$

Leis fundamentais para manipulação de desigualdades no sistema de números reais:

1. Se $a < b$, então para todo c
   $a + c < b + c$
2. Se $a < b$ e $c < d$, então
   $a + c < b + d$
3. Se $b < c$ e a é positivo, então
   $ab < ac$

Teorema 2.2 - Em $PF(r,p,c)$ temos

1. Se $a < b$, então $a \oplus c \leq b \oplus c$ válido para todo c
2. Se $a < b$ e $c < d$, então $a \oplus c \leq b \oplus d$
3. Se $b < c$ e a é positivo, então $a * b \leq a * c$

Infelizmente, estas relações, que são desigualdades restritas no sistema de números reais, foram enfraquecidas em $PF(r,p,c)$ por $\leq$. Mostraremos que as desigualdades estritas falham em $PF(r,p,c)$.

Para *(1)*, seja $a = r^{-2p}$, $b = 2r^{-2p}$, $c = 1$. Então $a < b$ mas $a \oplus c = b \oplus c = 1$.

Para *(2)*, assuma que $p \geq 2$ e seja

$$a = 1 - r^{-p}$$
$$b = 1$$
$$c = r^{-p}$$
$$d = r^{-p} + r^{-(p+1)}$$

Então $a < b$ $c < d$, mas $a \oplus c = b \oplus d = 1$.

Para *(3)*, note que em $PF(r,p,c)$ a lei do cancelamento falha, e temos números positivos a, b e c, $b < c$ e $a * b = a * c$.

2.2.6 - Aritmética de Corte com q Dígitos Adicionais para Números de Ponto Flutuante.

Representaremos por $PF(r,p,clq)$ uma ligeira modificação do sistema $PF(r,p,c)$ que descreve exatamente a aritmética que tem sido implementado em muitas máquinas (por exemplo, IBM/360 e /370).

O símbolo clq significa que executaremos a aritmética de corte usando um registrador de baixa ordem de q dígitos de comprimento, onde q pode ser qualquer inteiro maior ou igual a zero. Este registrador será usado nas operações $\oplus$, $\ominus$, e $*$ para manter os dígitos de baixa ordem dos resultados intermediários

que tenham mais do que p dígitos. Mais especificamente, na operação * guardará os próximos q dígitos do produto, e nas operação + e - guardará os próximos q dígitos do operando que é deslocado. Se este registrador é de comprimento *1* é denominado de *"guard digit"* (aritmética de precisão simples no IBM/370).

2.2.6(a) - Divisão de Ponto Flutuante

Nas máquinas IBM 7090 e IBM/370 a operação de ponto flutuante produz resultado corretamente cortado. Portanto, em $PF(r,p,clq)$ definimos $a \div b$ igual a $\overline{a/b}$.

2.2.6(b) - Multiplicação de Ponto Flutuante

Definimos o produto $a * b$ igual a zero se um dos fatores é zero. Se $ab \neq 0$, o sinal do produto é + ou -, conforme os sinais de a e b sejam iguais ou não. Seja

$$a = r^e m, \quad r^{-1} \leq m < 1$$

e

$$b = r^f n, \quad r^{-1} \leq n < 1.$$

Seja $\mu' = nm$, assim

$$ab = r^{e+f}\mu'.$$

Desde que

$$r^{-2} \leq \mu' < 1$$

μ' é um número de $2p$ dígitos com o ponto da raiz a esquerda e no máximo um zero após. Assumimos que podemos reter apenas $p + q$ dígitos do resultado, e seja μ'' os primeiros $p + q$ dígitos a direita do ponto em μ'. Seja

$$a * b = r^g \mu$$

onde g e μ são definidos como se segue: Se $r^{-1} \leq \mu < 1$, então $g = e + f$ e $\mu = \overline{\mu''}$. Por outro lado $\mu'' < r^{-1}$, então deslocamos uma posição para a esquerda para normalizá-lo e compensá-lo pelo decréscimo de 1 do expoente (pós-normalização). Neste caso temos $g = e + f - 1$ e $\mu = \overline{r\mu''}$. Sumarizamos em

$$g = e + f - k$$
$$\mu = \overline{r^k \mu''}$$

onde k é 1 ou 0 dependendo se haverá pós-normalização ou não.

Teorema 2.3 - O produto de ponto flutuante produz o mesmo resultado em $PF(r,p,c|q)$ para $q \geq 1$ como em $PF(r,p,c)$. Para $q = 0$ o produto produz em $PF(r,p,c|0)$ o mesmo resultado em $PF(r,p,c)$ se não for requerido pós-normalização. Se a pós-normalização é requerida, o produto $a*b$ em $PF(r,p,c|0)$ é obtido do produto $a * b$ em $PF(r,p,c)$ pela substituição do $p^{ésimo}$ dígito da mantissa por zero.

2.2.6(c) - Adição e Subtração de Ponto Flutuante

Definimos

$$a \ominus b = a \oplus (-b) \tag{2.6}$$

assim podemos nos preocupar apenas com adição de números com sinais. Sejam

$$a = r^e m \quad \text{e} \quad b = r^f n$$

normalizados e $e \geq f$ (se $e < f$ permutamos a e b). Então

$$b = r^e n'$$

onde $n' = r^{-(e-f)}$ obtido pelo deslocamento de n, $e-f$ posições a direita. Assim n' não é normalizado a menos que $e = f$. Temos ainda um registrador de q dígitos para reter os dígitos de baixa ordem deslocados para fora do registrador de b, assim seja n'' os $p + q$ bits de alta ordem do número n' de $\lceil p + (e-f) \rceil$ dígitos. Se $e-f \leq q$, temos $n'' = n'$, e se $e-f > q$ então n'' é obtido pelo abandono dos $e - f - q$ dígitos de baixa ordem. Temos então

$$\mu' = m + n'$$
$$a \oplus b = \overline{r^e \mu'}.$$

Se $e \neq f$, e um dos operandos é zero e que é deslocado para a direita, assim $n'' = n' = n$. Então nossa definição produz

$$a \oplus 0 = a \ominus 0 = a$$
$$0 \oplus b = b \qquad (2.7)$$
$$0 \ominus b = -b$$

como era esperado.

2.2.7 - Divisão

Definimos tanto para $PF(r,p,c)$ como para $PF(r,p,clq)$ por

$$a \div b = \overline{a/b}.$$

Se b é zero, o quociente é indefinido, e para qualquer $b \neq 0$, temos $0 \div b = 0$. Então podemos assumir que a e b são diferentes de zero e normalizados. Sejam

$$a = r^e m \ , \quad r^{-1} \leq |m| < 1$$
$$b = r^f n \ , \quad r^{-1} \leq |n| < 1.$$

Então

$$\frac{a}{b} = r^{e-f} \frac{m}{n}$$

e

$$r^{-1} \leq |\frac{m}{n}| < r.$$

Escrevemos $a \div b = r^g \mu$. Se $|m/n| < 1$, fazemos $g = e-f$ e $\mu = \overline{m/n}$. por outro lado, se $|m/n| \geq 1$, temos que

$$\frac{a}{b} = r^{e-f+1}(r^{-1}\ \frac{m}{n})$$

e

$$r^{-1} \leq |r^{-1}\ \frac{m}{n}| < 1,$$

assim $g = e-f+1$ e $\mu = \overline{r^{-1}\ m/n}$. Seja $k = 0$ se $|m| < |n|$ e $k = 1$ se $|m| \geq |n|$. Então temos

$$g = e-f+k$$
$$\mu = \overline{r^{-k}\ m/n}.$$

2.3 - *"OVERFLOW"* E *"UNDERFLOW"* DE PONTO FLUTUANTE

2.3.1 - Limites para Expoentes

Até este ponto, não nos preocupamos com a magnitude do expoente, escrevendo apenas $r^e m$ para representar um número de ponto flutuante. Em geral, o expoente está restrito ao intervalo

$$e_* \leq e \leq e^*. \tag{2.8}$$

Para uma máquina que armazena o expoente como uma característica, usualmente temos

$$e_* = -(e^* + 1). \tag{2.9}$$

Mas se a máquina mantêm expoentes negativos tanto como complemento de um ou como sinal e magnitude verdadeira, temos $e_* = -e^*$.

Restringindo o intervalo do expoente, restringimos o in

tervalo dos números de ponto flutuante que podemos representar. Usaremos Ω para representar o maior número de ponto flutuante que podemos representar sujeitos a (2.8). De modo similar, ω designará o menor número positivo que pode ser normalmente representado, sujeito a (2.8). Então

$$\Omega = r^{e*}(1 - r^{-p}) < r^{e*}$$

e (2.10)

$$\omega = r^{e}{}_{*}r^{-1} = r^{e}{}_{*} - 1.$$

Por exemplo, para o sistema IBM/360, teríamos

$$\Omega = 16^{63}\ (1 - 16^{-6}) < 16^{63}$$
$$\omega = 16^{-65} \qquad (2.11)$$

Há uma ligeira assimetria no nosso sistema de números de ponto flutuante. Isto sugere que no lugar de $S(r,p)$ deveríamos ter $S(r,p,e*,\ e*)$ que contém zero e todos os números em $S(r,p)$ que podem ser escritos na forma $r^{e}m$, onde e é um inteiro satisfazendo (2.8) e $r^{-1} \leq |m| < 1$. Mas continuaremos com a abordagem anterior porque trataremos o problema de *"overflow"* e *"underflow"* separado dos problemas relacionados com erros de arredondamentos e outras anomalias pertinentes a aritmética de ponto flutuante. Ocorrerá *"overflow"* se tentarmos produzir um número com valor absoluto maior do que Ω, e se o valor absoluto do número produzido for menor do que ω ocorrerá *"underflow"*. Em ambos os casos ocorre transbordamento de expoente.

2.3.2 - Arranjos Ω-*Zero*

Uma das primeiras abordagens para o problema de transbordamento de expoente foi chamada de arranjo Ω-*zero*. Este arranjo funciona da seguinte maneira:

(a) Se houver *"underflow"* de ponto flutuante o resultado se torna zero;

(b) Após ocorrer um *"overflow"* de ponto flutuante o número terá o sinal correto e o valor absoluto igual a Ω. Este arranjo é conhecido em maior escala sob a denominação de arranjo padrão.

Existe uma abordagem mais elaborada (usada pelo *Control Data Computer (CDC) 6600)*'' denominada de arranjo ∞-*zero* que funciona da seguinte maneira:

(a) Se ocorrer *"underflow"* de ponto flutuante o resultado produzido será zero;

(b) Se ocorrer *"overflow"* de ponto flutuante o resultado será um genuíno infinito.

Se o *hardware* provê uma interrupção quando ocorre transbordamento de expoente, o arranjo terá de ser executado por *Software*. Isto nem sempre é possível porque o *hardware* pode perder o sinal do número quando ocorrer um *"overflow"*.

2.3.3 - Interrupção

Frequentemente, o programador deseja saber se ocorreu ou não transbordamento de expoente. Isto é, a menos que o *hardware* produza um arranjo padrão, deve haver algum monitoramento da aritmética de ponto flutuante, de modo que o *software* possa prover um arranjo após ocorrer transbordamento de expoente.

Em muitas implementações da aritmética de ponto flutuante, o resultado após transbordamento de expoente tem uma característica de contorno. Para definir este resultado mais precisamente, assumiremos que a característica é definida por expoente mais γ e que pode ser qualquer inteiro de 0 a $c-1$. Então $e^* = c - 1 - \gamma$, assim

$$c = e^* - e_* + 1 \qquad (2.12)$$

Dizemos que o resultado após ocorrer transbordamento de expoente tem uma característica de contorno, se tem sinal correto, a mantissa correta, e uma característica no intervalo $[0, c-1]$ que difere da característica correta por c. Portanto, no lugar de $c + 1$ a característica será 1; no lugar de -3 será $c - 3$. Isto significa que se executarmos operações em que o resultado é tão grande que cause *"overflow"*, o *hardware* pode retornar um valor muito pequeno como resposta. De modo semelhante, o resultado após ocorrer *"underflow"* pode ser um número muito grande.

Um tratamento do transbordamento de expoente muito usado nas máquinas é o baseado no enfoque de interrupção. Após transbordamento de expoente, o registrador manterá o resultado com a

característica de contorno como tinha antes. Entrentanto o fluxo do programa será interrompido, e o programa desviará para uma rotina de manipulação de transbordamento.

Se o *hardware* provê uma interrupção após transbordamento de expoente, o arranjo é frequentemente produzido por rotinas de manipulação de *"overflow"*. Além disso, as rotinas de *"overflow"* imprimem mensagens para indicar que ocorreu transbordamento de expoente. A rotina de *"overflow"* é normalmente suprida pelo compilador, e determina quais as opções que estão disponíveis ao programador. Por exemplo, o compilador PL/I provê as declarações *ON OVERFLOW* e *ON UNDERFLOW* que permite ao programador tratar o transbordamento dentro de seu próprio programa. *O FORTRAN IV G* e *H* do sistema IBM / 360 e /370 provê o *"extended error-handling facility"* para tratar os transbordamentos de expoentes. Existem diversas técnicas de tratamento de transbordamento de expoentes que apenas citaremos neste texto. Podemos citar o modo da contagem desenvolvido por *W. Kahan* para o IBM 7094 na Universidade de Toronto e um outro método denominado de *"underflow"* gradual também do mesmo autor [*STERBENZ, 1974*] .

CAPÍTULO III

PROCESSADORES DE PONTO FLUTUANTE

3.1 - Introdução

A maioria das primeiras máquinas foram projetadas com a unidade aritmética de ponto fixo e frequentemente, eram usadas com representações fracionárias. Portanto, era de responsabilidade do programador fazer a escala de seus números para a forma fracionária, tornando o desenvolvimento de programas de aplicações científicas muito dispendioso [*STERBENZ, 1974*] . Para que este trabalho, realizado até então pelo programador, fosse transferido ao computador, começou-se o desenvolvimento de processadores de números de ponto flutuante [*MANO, 1976*] . Muitos computadores têm sido construídos com a capacidade de operar com aritmética de ponto flutuante. Em ponto flutuante, o ponto da raiz para cada número varia automáticamente. Esta característica

é considerada como um dos maiores desenvolvimentos nos computadores digitais, desde que permite grande flexibilidade na manipulação de quantidades numéricas.

A inclusão da capacidade da aritmética de ponto flutuante resultou em aumento da complexidade no projeto da unidade aritmética.

Computadores que não incluem o *hardware* para operações em ponto flutuante, normalmente executa-as através de programação adicional, aumentando o tempo de execução. Assim concluímos que o processador de ponto flutuante pode ser dispositivo de *hardware* ou mesmo *software* desenvolvido para tal fim.

3.2 - Conceitos Básicos

Apresentaremos algumas notações, conceitos e algorítmos imprescindíveis para a compreensão do que será apresentado no restante do capítulo.

3.2.1 - Notações

As operações básicas da álgebra para circuítos serão assim representados:

OPERAÇÕES	*NOTAÇÕES USADAS*
AND	$X.Y$ ou XY
OR	$X + Y$
NOT (ou complemento)	X' ou $\overline{X}$
OR EXCLUSIVO	$X \oplus Y$

onde X e Y são operandos.

Transferência paralela de registrador a registrador terá a seguinte notação:

$$A \rightarrow B$$

onde A e B são registradores de n bits.

A transferência de subregistradores terá a seguinte forma:

$$B \rightarrow F(A)$$
$$G(A) \rightarrow G(A)$$
$$F(A) \rightarrow F(A)$$
$$C \rightarrow G(A)$$

onde A, B e C são registradores de n bits, m bits e $n-m$ bits respectivamente, com n maior ou igual a m, $F(A)$ é subregistrador dos primeiros m bits de A e $G(A)$ o subregistrador dos $n-m$ últimos bits de A.

Seja $A = (A_1, A_2, \ldots, A_n)$ um registrador com capacidade de deslocamentos, e D um registrador de 1 bit. Considere os subregistradores $L(A) = (A_1, A_2, \ldots, A_{n-1})$ e $R(A) = (A_2, A_3, \ldots, A_n)$, então o deslocamento de um bit do registrador A para a esquerda

terá a seguinte notação:

$$R(A) \rightarrow L(A),\ D \rightarrow A_n$$

e para a direita

$$L(A) \rightarrow R(A),\ D \rightarrow A_1 .$$

A notação de transferência condicional será

$$A_o A \ + \overline{A}_o A \rightarrow A$$

onde A_o é o bit de sinal do registrador A.

3.2.2 - Algorítmos de Operações Aritméticas em Binário de Numéros de Ponto Fixo

Apresentaremos os algorítmos de [*ABD-ALLA, 1976*] adição/subtração, mutiplicação e divisão de binários inteiros, com os números negativos representados em complemento de *2*.

3.2.2(a) - Adição/Subtração

PASSO 1. Compare os bits de sinais dos números. Se são iguais, vá ao passo 2; caso contrário vá ao passo 3.

PASSO 2. Adicione as magnitudes para formar a soma. O sinal é igual ao dos dois números. Fim.

PASSO 3. Adicione o complemento de 2 do número negativo. Se o bit de sinal da soma é 1 atribuímos sinal 0; caso contrário sinal 1. Fim.

3.2.2(b) - Multiplicação

PASSO 1. Inicia com o produto acumulado igual a zero, determina o sinal do produto e complementa os fatores que estiverem na forma complementar.

PASSO 2. Inspeciona os bits do multiplicador individualmente, iniciando pelo bit menos significativo.

PASSO 3. Adiciona o multiplicando ao produto acumulado se o bit do multiplicador for 1; caso contrário adiciona *0's*.

PASSO 4. Desloca o multiplicando de 1 bit para a esquerda.

PASSO 5. Vá ao passo 2 e repita a iteração até que todos os produtos parciais sejam adicionados. No final colocar o sinal.

3.2.2(c) - Divisão

PASSO 1. Determinar o sinal do quociente e converter os termos que estiveram na forma complementar.

PASSO 2. Subtrair o divisor do dividendo, somando o complemento de 2 do divisor. Se o resto é um número positivo, o primeiro dígito do quociente é 1. Caso contrário fazer o primeiro dígito igual a zero e somar o divisor.

PASSO 3. Deslocar o resto para a esquerda de um bit e subtrair o divisor do resto. Se o resultado for positivo, atribuir 1 ao quociente e ir ao passo seguinte; caso contrário atribuir 0 ao quociente e somar o divisor.

PASSO 4. Repetir o passo 3 até que o número de deslocamentos seja igual a diferença do número de bits do dividendo e do divisor. No final colocar o sinal.

3.3 - Processador de Ponto Flutuante em *HARDWARE*

Existem várias opções de estrutura para o processador. Por exemplo:

- Unidade aritmética de ponto flutuante independente da

unidade aritmética de ponto fixo, permitindo processamento paralelo;

- Unidade aritmética de ponto flutuante contendo a de ponto fixo;

- Unidade aritmética de ponto flutuante separada, mas que utiliza a de ponto fixo para operar com o expoente e a mantissa separadamente.

Para o projeto que apresentaremos, escolhemos a segunda opção porque os computadores já possuem unidade aritmética de ponto fixo e para expandí-la para ponto flutuante basta acrescentar um pequeno processador de ponto fixo para operar com o expoente, com um custo não muito elevado e uma performance aceitável.

Agora podemos escolher a base que deverá ser utilizada pelo processador. O critério de escolha normalmente leva em consideração velocidade, custo e facilidade de uso [*ABD-ALLA, 1976*]. As bases mais frequentemente utilizadas são binária, octal, hexadecimal e decimal. Entre estes escolhemos a base binária, porque o custo de memória utilizada é menor que o das outras e também pela facilidade de implementação. Então a aritmética de ponto flutuante será $PF(2,p,clq)$, onde as operações serão executadas paralelamente em todos os bits. A representação de números negativos será na forma de complemento de 2.

3.3.1 - Unidade Aritmética de Ponto Fixo

Apresentaremos primeiro a unidade aritmética de ponto fixo e depois construiremos a de ponto flutuante acrescentando apenas um outro adicionador, outros registradores e dispositivos necessários.

3.3.1(a) - Adição e Subtração

Considere o algorítmo de adição e subtração para números binários em complemento de 2 apresentado em 3.2.2(a).

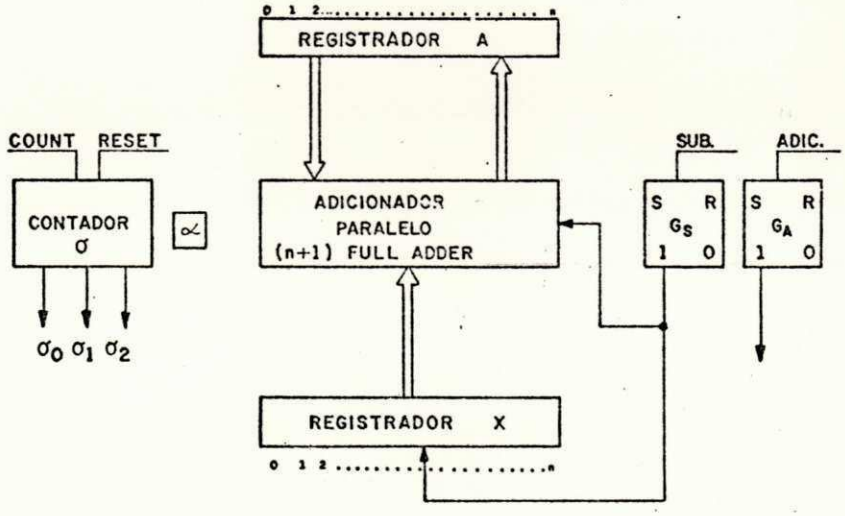


Figura 3.1 - Layout da adição e subtração em uma unidade aritmética de ponto fixo paralela.

Na figura 3.1, durante a operação, o registrador A (acumulador) mantém a primeira parcela (ou minuendo), enquanto o registrador X mantém a segunda parcela (ou subtraendo). Os *flip-flops* G_S e G_A indicam se a operação é adição ou subtração. No caso da

subtração a saída do *flip-flop* G_S gera um sinal para complementar X e adiciona 1 ao registrador A para formar o complemento de 2. A sequência dos sinais de controle e as transferências correspondentes são descritas a seguir:

INICIO | $1 \rightarrow G_A$ para adição,

$1 \rightarrow G_S$ para subtração,

$0 \rightarrow \sigma$

$(G_S+G_A)\sigma_0$ | $G_S\overline{X} + G_AX \rightarrow X$, complementa X se $G_S = 1$

$A_0 \oplus [G_AX_0 + G_S\overline{X}_0] \rightarrow \alpha$, armazena o sinal do resultado em α

$(\sigma + 1) \rightarrow \sigma$

$(G_S+G_A)\sigma_1$ | $S(A,X,G_S) \rightarrow A$, efetua a adição de A com X e se $G_S = 1$ adiciona 1 para obter o complemento de 2.

$(\sigma + 1) \rightarrow \sigma$

$(G_S+G_A)\sigma_2$ | $\alpha(X_0 \oplus A_0) \rightarrow V$, liga o *flip-flop* caso ocorra *overflow*

$0 \rightarrow G_S$,

$0 \rightarrow G_A$

Tanto a adição como subtração requerem três ciclos *(time slice)* para sua execução $(\sigma_0, \sigma_1, \sigma_2)$.

$S(A,X,G_S) = A + X + G_S$ - adição binária bit a bit

3.3.1(b) - Multiplicação

Consideramos a multiplicação binária como uma série de adições repetidas conforme o algorítmo apresentado em 3.2.2.(b). A operação de multiplicação usa os registradores *A*, *Q* e *X* mostrados na figura 3.2. O multiplicador e o multiplicando são colocados nos registradores *A* e *X*, respectivamente e o resultado final ficará em *(A,Q)*.

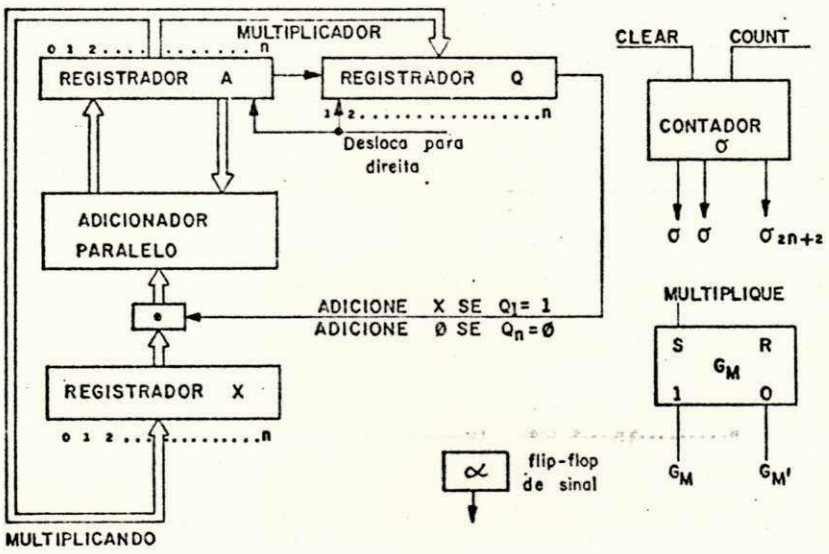


Figura 3.2 - Layout de multiplicação em uma unidade aritmética de ponto fixo paralela.

Os sinais de controle e transferência correspondentes são:

INICIO | $1 \rightarrow G_M$, $0 \rightarrow \sigma$

$G_M\ \sigma_0$ | $A_0 \oplus X_0 \rightarrow \propto$, armazena o sinal do produto

$A_0 . S(\overline{A}, 1) + \overline{A}_0 A \rightarrow A$, Faz A positivo

$(\sigma + 1) \rightarrow \sigma$

$G_M\ \sigma_1$ | $R(A) \rightarrow Q$, Transfere A para Q

$X_0 . S(\overline{X}, 1) + \overline{X}_0 . X \rightarrow A$, Faz X positivo e coloca em A

$(\sigma + 1) \rightarrow \sigma$

$G_M\ \sigma_2$ | $A \rightarrow X$, $0 \rightarrow A$, Transfere A para X e zera A

$(\sigma + 1) \rightarrow \sigma$

$G_M(\sigma_3+\sigma_5+\ldots+\sigma_{2n+1})$ | $S(A, Q_n . X) \rightarrow A$, $(\sigma + 1) \rightarrow \sigma$

$G_M(\sigma_4+\sigma_6+\ldots+\sigma_{2n+2})$ | $L(A,Q) \rightarrow R(A,Q)$

$0 \rightarrow A_0$, $(\sigma + 1) \rightarrow \sigma$

} processo repetido de adição e deslocamento

$G_M \sigma_{2n+3}$ | $\propto . S(\overline{A}, 1) + \overline{\propto} A \rightarrow A$, coloca o resultado na forma de complemento de 2.

$0 \rightarrow G_M$

3.3.1(c) - Divisão

A operação inicia após o carregamento do dividendo e do divisor nos registradores A e X. O quociente é colocado no registrador Q, o resto no registrador A e o sinal no *flip-flop* $\propto$. O esquema dos registradores usados na divisão é mostrado na figura 3.3.

Os sinais de controle e as transferências associadas são mostradas abaixo:

INICIO | $1 \rightarrow G_D$, $0 \rightarrow \sigma$

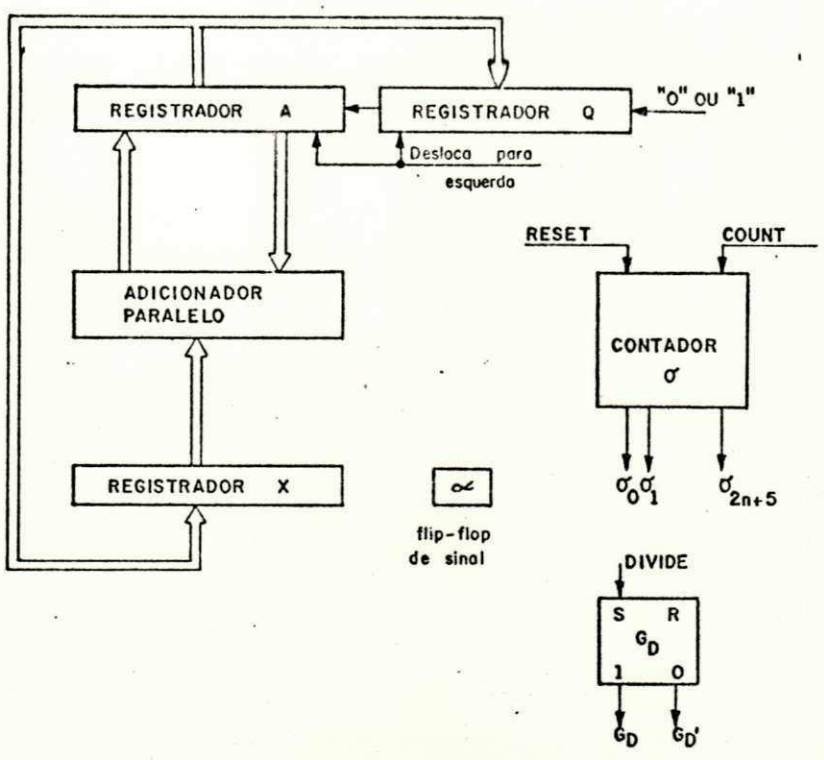


Figura 3.3 - Layout de divisão numa unidade aritmética de ponto fixo paralelo.

$G_D\ \sigma_0$ | $A_0 \oplus X_0 \rightarrow \alpha$, armazena o sinal do quociente

$A_0 S(\overline{A}, 1) + \overline{A}_0 A \rightarrow A$, Faz A positivo

$(\sigma + 1) \rightarrow \sigma$

$G_D\ \sigma_1$ | $A \rightarrow Q$, coloca A em Q

$X_0 S(\overline{X}, 1) + \overline{X}_0 X \rightarrow A$, Faz X positivo e transfere para A

$(\sigma + 1) \rightarrow \sigma$

$G_D\ \sigma_2$ | $A \rightarrow X$, coloca A em X

$Q \rightarrow A$, coloca Q em A

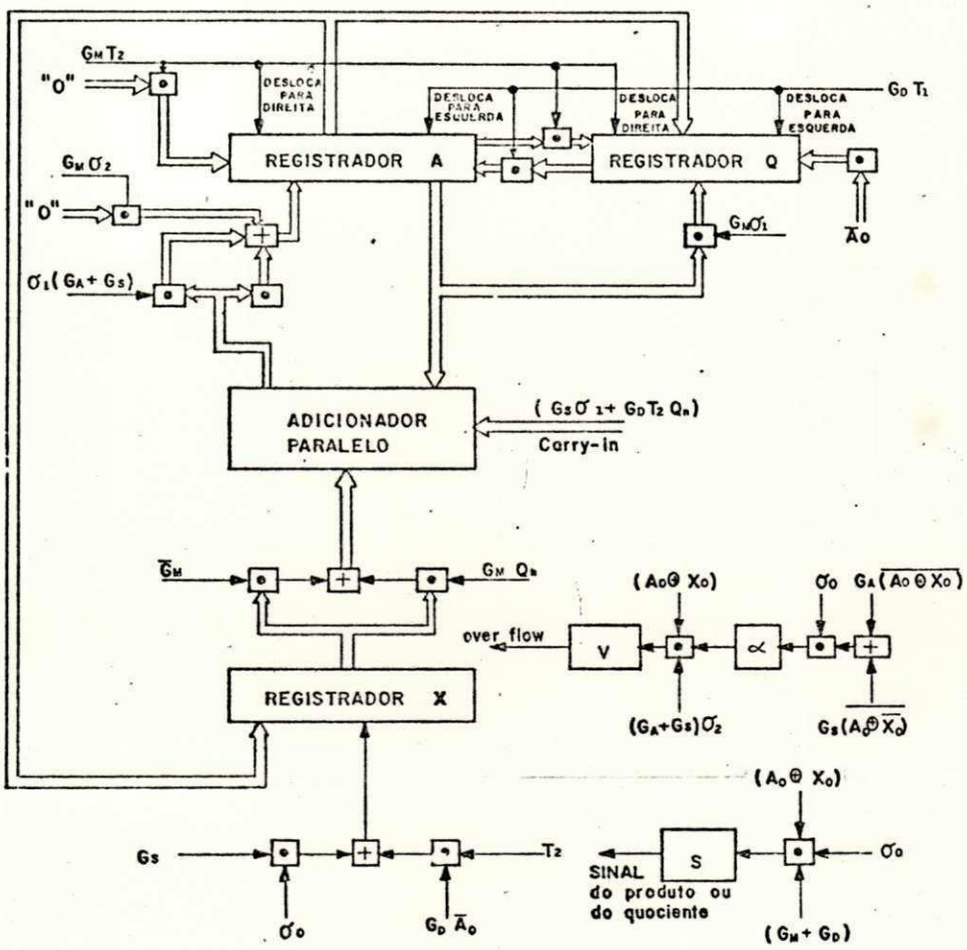
$0 \rightarrow Q$, zera Q

$(\sigma + 1) \rightarrow \sigma$

$G_D\ \sigma_3$	$R(A,Q) \rightarrow L(A,Q)$,	desloca A e Q 1 posição para esquerda
	$(\sigma + 1) \rightarrow \sigma$	
$G_D\ \sigma_4$	$S(\overline{A},X,1) \rightarrow A$,	subtrai X de A
	$(\sigma + 1) \rightarrow \sigma$	
$G_D(\sigma_5+\sigma_7+\ldots+\sigma_{2n+3})$	$R(A,Q) \rightarrow L(A,Q)$,	se A é positivo gera
	$\overline{A}_0 \rightarrow Q_n$	bit 1 para quociente, caso contrário bit 0
	$(\sigma + 1) \rightarrow \sigma$	
$G_D(\sigma_6+\sigma_8+\ldots+\sigma_{2n+4})$	$Q_n S(A,\overline{X},1) + \overline{Q}_n S(A,X) \rightarrow A$	adição ou subtração de X de A se $Q_n=0$ ou $Q_n = 1$, respectivamente
	$(\sigma + 1) \rightarrow \sigma$	
$G_D(\sigma_{2m+5})$	$\propto S(\overline{Q},1) + \overline{\propto} Q \rightarrow Q$	complemento de 2 se Q é negativo
	$0 \rightarrow G_D$	

3.3.1(d)- Diagrama Lógico da Unidade Aritmética de Ponto Fixo Paralela.

Agora, após a apresentação do modo como as operações aritméticas de ponto fixo serão realizadas, podemos construir a unidade aritmética de ponto fixo. Utilizaremos 3 registradores A, X, e Q, sendo que A e X devem ser do tipo capaz de realizar deslocamentos *(shift registers)*. Na figura 3.4 apresentamos o diagrama lógico e na figura 3.5 os sinais de controle da unidade aritmética de ponto fixo.



$T_3 = \sigma_3 + \sigma_5 + \ldots\ldots + \sigma_{2n+3}$

$T_2 = \sigma_4 + \sigma_6 + \ldots\ldots + \sigma_{2n+4}$

figura 3.4 - Unidade Aritmética de ponto fixo paralela.

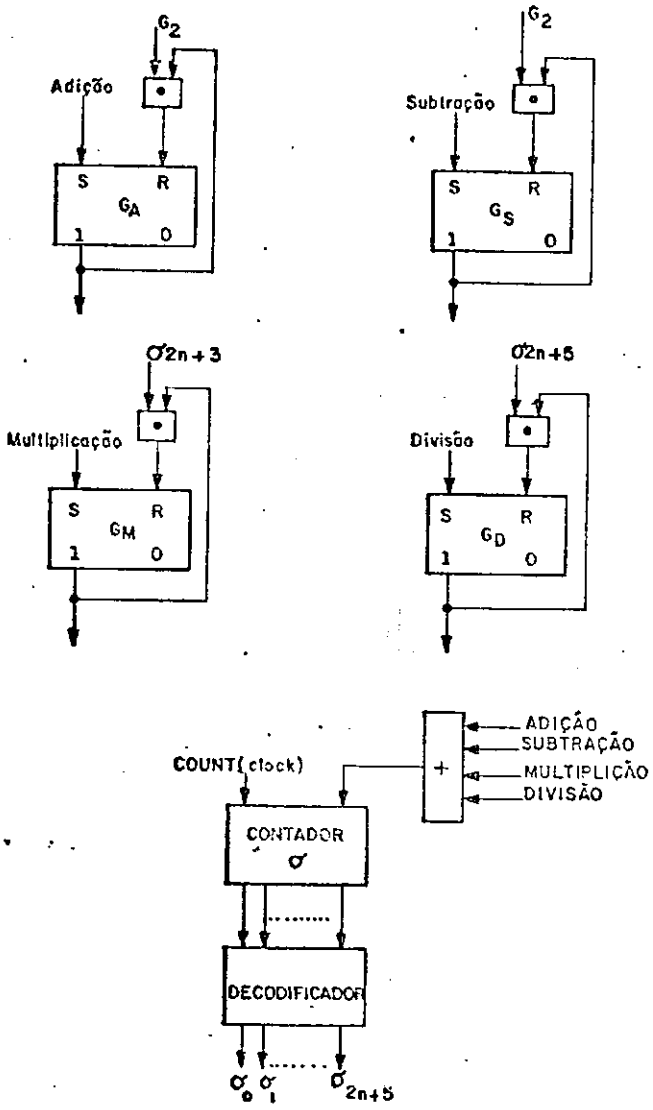


Figura 3.5 - Sinais de controle da unidade aritmética de ponto fixo paralela.

O contador σ é um contador binário com uma contagem máxima de $2n + 5$, onde $n + 1$ é o comprimento dos operandos. O contador é re-inicializado com zero (0) no início de cada operação. A adição e subtração necessita apenas de três saídas $(\sigma_0, \sigma_1, \sigma_2)$, enquanto a multiplicação utiliza até $2n + 3$ e a divisão até $2n + 5$.

3.3.2 - Unidade Aritmética de Ponto Flutuante Paralela

Dividiremos um registrador logicamente da seguinte forma:

EXPOENTE	±	*MANTISSA*

O primeiro passo será a discussão individual da implementação de cada operação aritmética de ponto flutuante.

3.3.2(a) - Adição e Subtração em Ponto Flutuante

Afim de simplificar a lógica de controle na execução, assumiremos a disponibilidade dos registradores C e D para manter os expoentes e um adicionador paralelo para operar com os expoentes.

O primeiro passo após o início da operação é a transferência dos expoentes para C e D.

O segundo passo é o alinhamento inicial das mantissas. Consideramos que o número de menor expoente está em A; caso contrário, será necessário uma permuta.

O terceiro passo será a operação de adição ou subtração das mantissas.

O quarto passo será a normalização do resultado e o quinto e último passo será a união do expoente com a mantissa do resultado.

Na figura 3.6 mostramos como fica o esquema dos registradores e dos adicionadores.

A seguir mostraremos os sinais de controle e as transferências necessárias.

INICIO	$1 \rightarrow Q_A$,	para adição
	$1 \rightarrow Q_S$,	para subtração
	$0 \rightarrow P$	zera o contador
$(Q_A + Q_S)P_0$	$A_e \rightarrow C$,	transfere expoente de A para C
	$X_e \rightarrow D$,	idem de X para D
	$(P+1) \rightarrow P$	
$(Q_A + Q_S)P_1$	$S(C,\overline{\overline{D}}) \rightarrow C$,	faz $C-D$ e coloca em C
	$(P+1) \rightarrow P$	
$(Q_A + Q_S)P_2$	$C_0A_m + \overline{C}_0X_m \rightarrow A_m$,	Se $C-D>0$ efetua a troca de A_m e X_m
	$C_0X_m + C_0A_m \rightarrow X_m$	
	$C_0(P+4) + \overline{C}_0(P+1) \rightarrow P$	se houve troca P_3; caso contrário P_5

$\overline{\overline{D}}$ - complemento de 2 do registrador D.

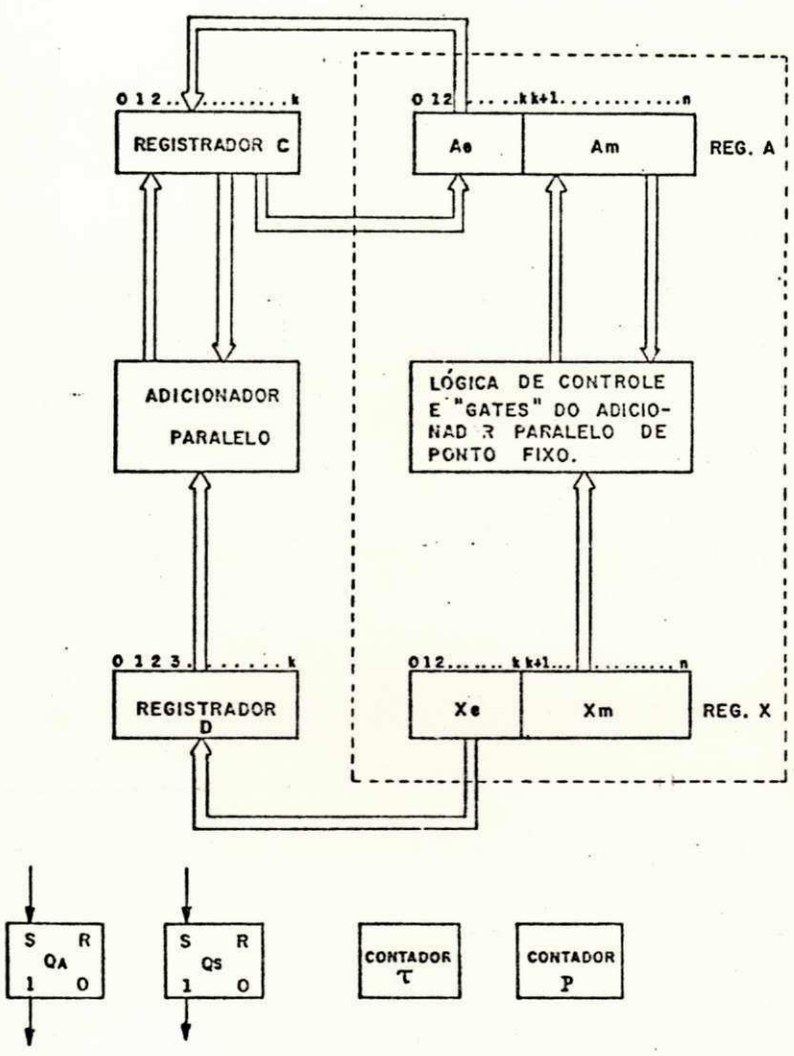


FIGURA 3.6 - Layout de registradores para adição e subtração de números de ponto flutuante.

$(Q_A+Q_S)P_3$ | $S(C,D) \rightarrow C$, soma C com D colocando o resultado em C

$(P+1) \rightarrow P$

$(Q_A+Q_S)P_4$ | $C \rightarrow D$, $D \rightarrow C$, troca C por D

$(P+1) \rightarrow P$

$(Q_A+Q_S)P_5$ | $S(C,\overline{D}) \rightarrow C$, faz $C-D$ e coloca em C

$(P+1) \rightarrow P$

$(Q_A+Q_S)P_6$ | $C_0L(A) + \overline{C}_0R(A) \rightarrow R(A)$,
$C_0(C+1) + \overline{C}_0C \rightarrow C$,
$C_0P + \overline{C}_0(P+1) \rightarrow P$
efetua o alinhamento necessário de $D-C$ posições

$(Q_A+Q_S)P_7(\tau_0+\tau_1+\ldots+\tau_k)$ | $R(A) \rightarrow L(A)$, desloca A, 1 posição p/esquerda
$R(X) \rightarrow L(X)$ desloca X, 1 posição p/esquerda
$0 \rightarrow A_n$,
$0 \rightarrow X_n$,
$(\tau+1) \rightarrow \tau$

$(Q_A+Q_S)P_7\ \tau_k$ | $(P+1) \rightarrow P$,
$0 \rightarrow \tau$

$(Q_A+Q_S)P_8$ | $Q_A \rightarrow G_A$, $Q_S \rightarrow G_S$, transfere o controle para a unidade aritmética de ponto fixo
$(P+1) \rightarrow P$

$(Q_A+Q_S)P_9$ | $(Q_A\overline{G}_A+Q_S\overline{G}_S)(P+1) + (Q_AG_A + Q_SG_A)P \rightarrow P$
verifica se a adição ou subtração de ponto fixo já terminou

$(Q_A+Q_S)P_{10}$ | $V.L(A) + \overline{V}.R(A) \rightarrow R(A)$,
$V(C+1) + \overline{V}C \rightarrow C$,
$V(P+2) + \overline{V}(P+1) \rightarrow P$
se houver *overflow*, desloca 1 para a direita, adiciona 1 ao expoente e faz $P=P_{12}$

$(Q_A+Q_S)P_{11}$ | $(\overline{A_0 \oplus A_1})R(A) + (A_0 \oplus A_1)L(A) \rightarrow L(A)$,

$(\overline{A_0 \oplus A_1})(C-1) + (A_0 \oplus A_1)\ C \rightarrow C$,

$(A_0 \oplus A_1)P + (A_0 \oplus A_1)(P+1) \rightarrow P$

normaliza o registrador A

$(Q_A+Q_S)P_{12}(\tau_0+\tau_1+\ldots+\tau_k)$ | $L(A) \rightarrow R(A)$, $(\tau+1) \rightarrow \tau$ — desloca A, 1 posição p/direita

$(Q_A+Q_S)P_{12}\ \tau_k$ | $(P+1) \rightarrow P$, $0 \rightarrow \tau$

$(Q_A+Q_S)P_{13}$ | $C \rightarrow A_e$, $0 \rightarrow Q_A$, $0 \rightarrow Q_S$ — une o expoente a mantissa A

3.3.2(b) - Multiplicação de Ponto Flutuante

A multiplicação de números de ponto flutuante requer a adição de seus expoentes e a multiplicação das mantissas. O multiplicador ficará no registrador X e o multiplicando no registrador A. Mostraremos o esquema dos registradores necessários para a multiplicação na figura 3.7.

Os sinais de controle e as sequências de transferência associadas são mostrados a seguir:

INÍCIO | $1 \rightarrow Q_M$, $0 \rightarrow P$, $0 \rightarrow \tau$

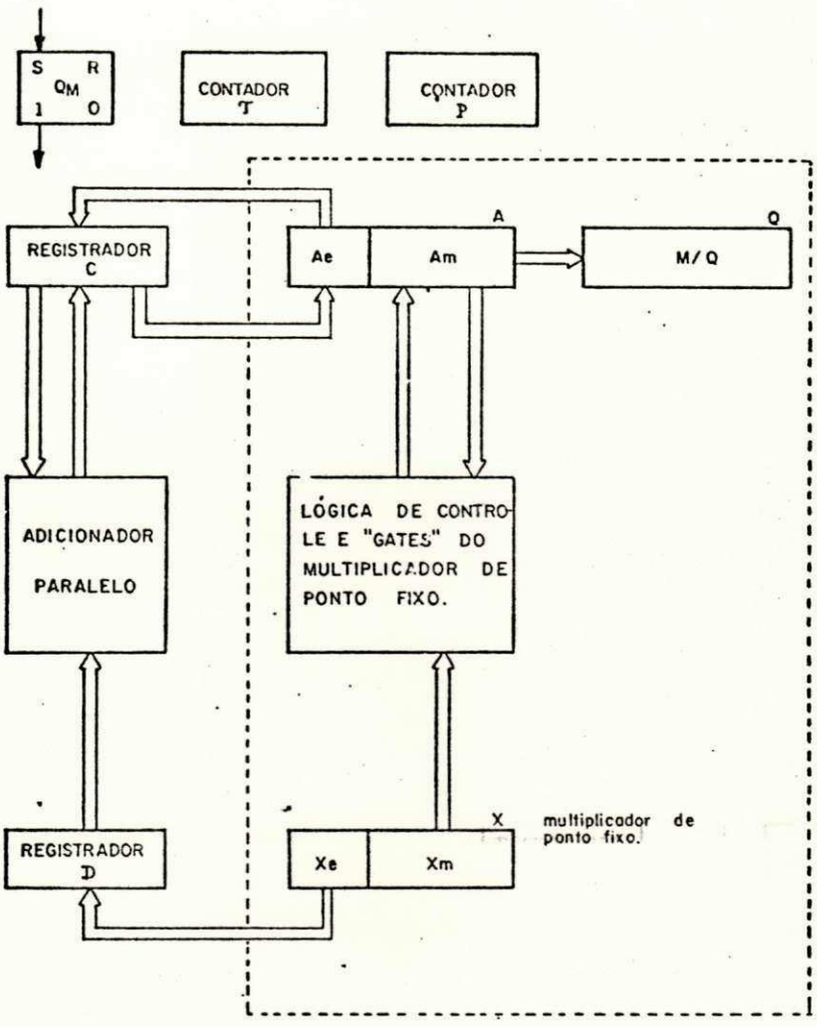


Figura 3.7 - Layout de registradores para multiplicação de números de ponto flutuante.

$Q_M P_0$ | $A_e \rightarrow C$, transfere o expoente de A e X para $X_e \rightarrow D$, C e D, respectivamente
$(P+1) \rightarrow P$

$Q_M P_1$ | $S(C,D) \rightarrow C$, adiciona C e D em C
$(\overline{C_0 \oplus D_0}) \rightarrow \alpha_1$ armazena o sinal do expoente
$(P+1) \rightarrow P$

$Q_M P_2$ | $\alpha_1 (C_0 \oplus D_0) \rightarrow V$ *overflow* de expoente

$Q_M P_3 \ \tau_0$ | $R(A) \rightarrow L(A)$, desloca A e X uma posição para
$R(X) \rightarrow L(X)$, a esquerda
$0 \rightarrow A_n$, coloca zero nas últimas posi
$0 \rightarrow X_n$, ções de A e de X
$(\tau+1) \rightarrow \tau$

$Q_M P_3 (\tau_1+\tau_2+\ldots\tau_k)$ | As mesmas transferências de τ_0

$Q_M P_3 \tau_k$ | $(P+1) \rightarrow P$,
$0 \rightarrow \tau$, ativa o circuíto de multiplica
$1 \rightarrow G_M$ ção de ponto fixo

$Q_M P_4$ | $\overline{G}_M (P+1) + G_M P \rightarrow P$ verifica se a multiplica ção de ponto fixo já en cerrou

$Q_M P_5 (\tau_0+\tau_1+\ldots+\tau_k)$ | $L(A,Q) \rightarrow R(A,Q)$, desloca uma posição para
$(\tau + 1) \rightarrow \tau$, a direita $k+1$ vezes

$Q_M P_5 \tau_k$ | $(P+1) \rightarrow P$

$Q_M P_6$ | $(\overline{A_k \oplus A_{k+1}})\ R(A_m) + (A_k \oplus A_{k+1})\ L(A_m) \rightarrow L(A_m)$,
efetua a pós-normalização
$(\overline{A_k \oplus A_{k+1}})\ (C-1) + (A_k \oplus A_{k+1}) C \rightarrow C$,
se houve a pós-normalização
subtrai 1 do expoente
$(\Gamma+1) \rightarrow P$

$Q_M P_7$ | $C \rightarrow A_e$, une o expoente a mantissa em A
$0 \rightarrow Q_M$

· Na multiplicação pode ocorrer *overflow* de expoente, mas de mantissa não poderá ocorrer, porque a magnitude do produto de duas frações normalizadas está contida no intervalo $1/4 \leq x \leq 1$, onde x representa o produto.

3.3.2(c) - Divisão de Ponto Flutuante

A divisão de dois números de ponto flutuante é realizada pela subtração dos expoentes e divisão das mantissas. A divisão começa com o armazenamento do dividendo e do divisor nos registradores A e X, respectivamente. Os expoentes A_e e X_e são transferidos para os registradores C e D. O adicionador paralelo, como mostra a figura 3.8, efetua a diferença dos expoentes e as mantissas após serem deslocadas sofrem divisão de ponto fixo.

A seguir, apresentaremos os sinais de controle e as respectivas transferências:

INICIO | $1 \rightarrow Q_D$,
$0 \rightarrow P$,
$0 \rightarrow \tau$

$Q_D P_0$ | $A_e \rightarrow C$, transfere os expoentes de A e de X
$X_e \rightarrow D$, para C e D, respectivamente
$(P+1) \rightarrow P$

$Q_D P_1$ | $S(C,\overline{\overline{D}}) \rightarrow C$, faz a diferença $C-D$ em C armaze
$(\overline{C_0 \oplus D_0}) \rightarrow \alpha$, na o sinal do expoente
$(P+1) \rightarrow P$

$Q_D P_2$ | $\alpha(C_0 \oplus D_0) \rightarrow V$, *overflow* de expoente
$(P+1) \rightarrow P$

$Q_D P_3$ | $R(A) \rightarrow L(A)$, desloca A e X 1 posição para a
$R(X) \rightarrow L(X)$, esquerda
$0 \rightarrow A_n$, coloca zero na posição n de A
$0 \rightarrow X_n$, e de X
$(\tau+1) \rightarrow \tau$

$Q_D P_3(\tau_1+\tau_2+\ldots+\tau_k$ | similar a τ_0, desloca A e X 1 posição para a esquerda k vezes

$Q_D P_3 \tau_k$ | $(P+1) \rightarrow P$, ativa o circuito de divisão de
$0 \rightarrow \tau$, ponto fixo
$1 \rightarrow G_D$

$Q_D P_4$ | $\overline{G}_D(P+1) + G_D P \rightarrow P$, verifica se a operação de divisão de ponto fixo já terminou
| $\overline{G}_D Q + G_D A \rightarrow A$ transfere o quociente Q para A.

$Q_D P_5(\tau_0+\tau_1+\ldots+\tau_k)$ | $L(A,Q) \rightarrow R(A,Q)$, desloca (A,Q) $k+1$ posi
$(\tau+1) \rightarrow \tau$, ções para a direita

$Q_D P_5 \tau_k$ | $(P+1) \rightarrow P$

$Q_D P_6$ | $C \rightarrow A_e$, une o expoente com a mantissa em A
$0 \rightarrow Q_D$

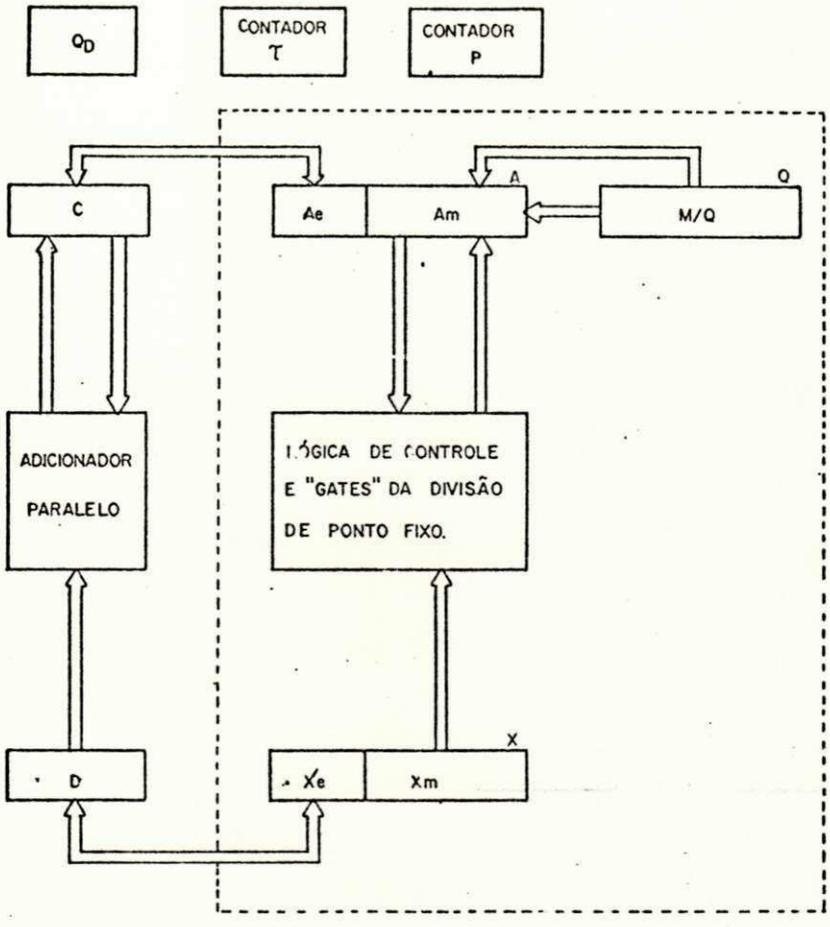
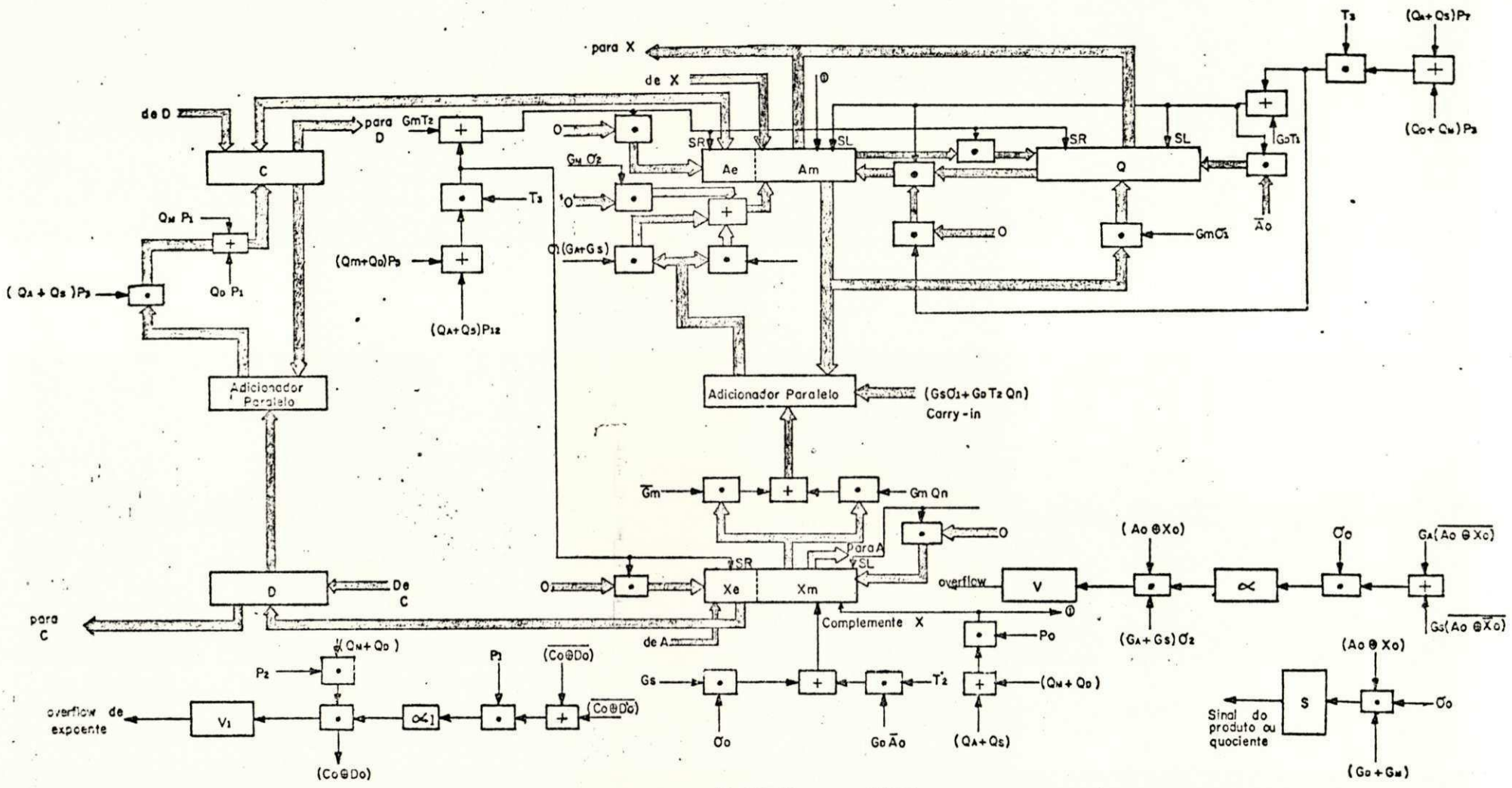


Figura 3.8 - Layout de registradores para divisão de ponto flutuante paralela.

3.3.2(d) - Diagrama Lógico da Unidade Aritmética

Finalmente, podemos esquematizar a Unidade Aritmética capaz de operar com números de ponto fixo e números de ponto flutuante.

Aproveitaremos a unidade aritmética de ponto fixo, apresentada em 3.3.1(d) na figura 3.4, e adicionaremos as novas funções através de uma divisão lógica dos registradores *A* e *X* além da inclusão dos registradores *C* e *D* e de um adicionador paralelo de ponto fixo. A unidade aritmética de ponto flutuante paralela é mostrada na figura 3.9 e os sinais de controle na figura 3.10.



$T_1 = O_3 + O_5 + \ldots\ldots\ldots + O_{2n+3}$

$T_2 = O_4 + O_6 + \ldots\ldots\ldots + O_{2n+3}$

$T_3 = T_0 + T_1 + \ldots\ldots\ldots + T_k$

Figura 3.9 – Unidade aritmética de ponto flutuante paralela.

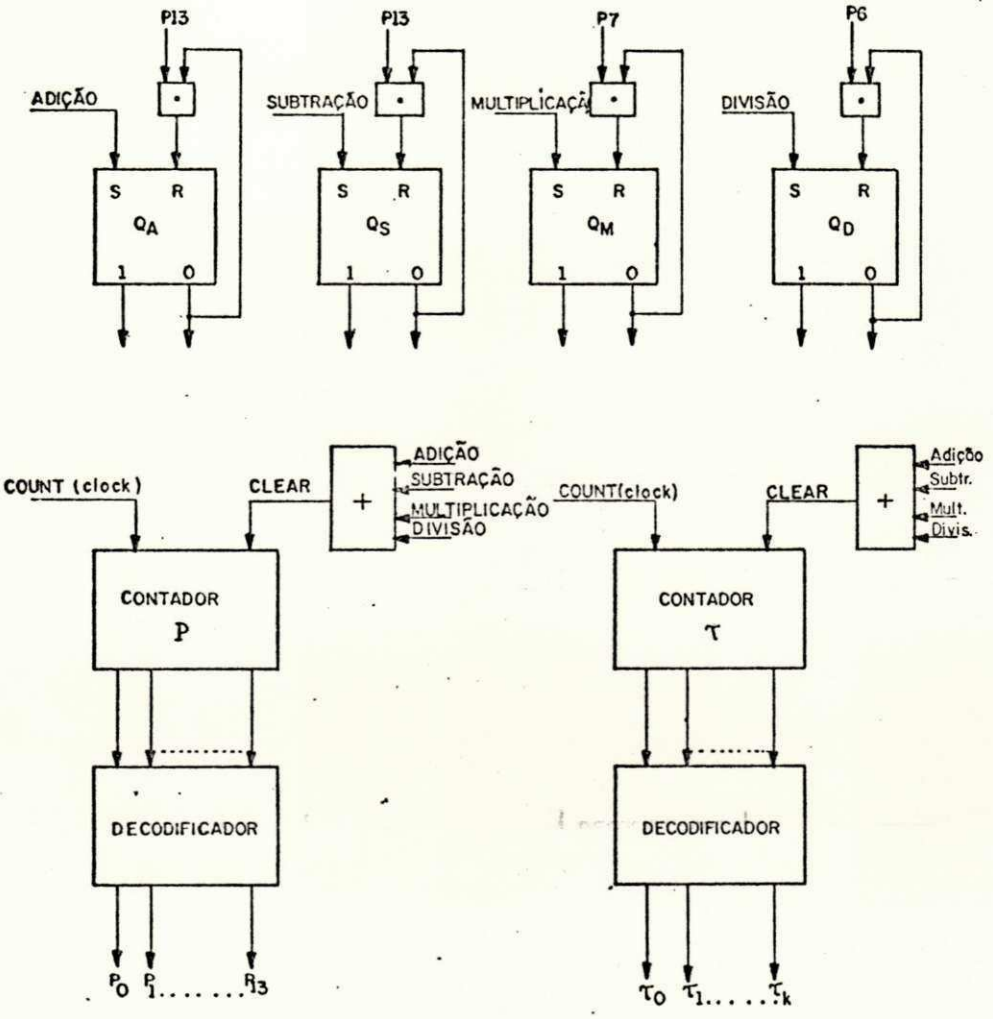


Figura 3.10 - Sinais de controle para a unidade aritmética de ponto flutuante.

3.4. - Processadores de Ponto Flutuante em *SOFTWARE*

Os microcomputadores e certos minicomputadores não possuem processadores de ponto flutuante em *hardware*. Para suprir esta lacuna, existem soluções econômicas, e uma delas é o desenvolvimento de rotinas que simulem as funções de um processador.

As rotinas são transparentes ao usuário, pois na maioria dos casos, o compilador gera desvios para as rotinas de Ponto Flutuante, na fase de tradução do programa escrito em linguagem de alto nível para a linguagem de máquina.

Antes de apresentarmos uma possível estrutura de um *software* processador de números de ponto flutuante, discutiremos rapidamente os problemas dos compiladores com a finalidade de executar cálculos científicos para máquinas desprovidas de *hardware* para ponto flutuante.

3.4.1 - Ligação das Rotinas de Ponto Flutuante

Em geral as rotinas de ponto flutuante são ligadas aos programas fontes, conforme mostra a figura 3.11.

O primeiro passo será desenvolver subrotinas que façam os processamentos que envolvem números de ponto flutuante e em seguida essas subrotinas são utilizadas pelos compiladores durante a tradução das linguagens de alto nível. Temos os compiladores *FORTRAN*, *SL/1* e outros desenvolvidos para o computador IBM

1130 que pertencem a categoria descrita acima.

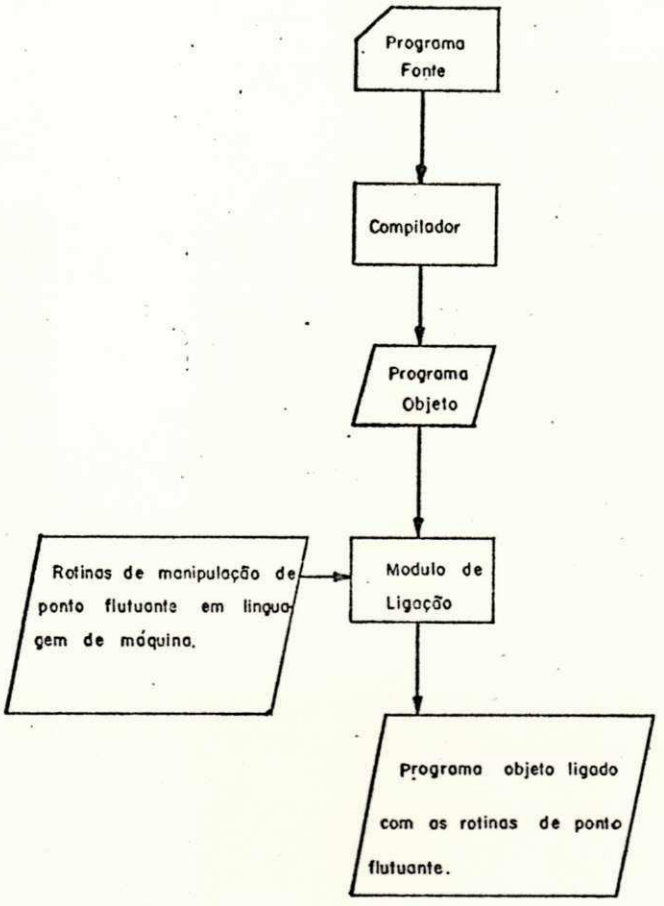


Figura 3.11 - Compilação de programa fonte e ligação das rotinas de ponto flutuante.

A principal desvantagem deste tipo de compilador, é a impossibilidade de transportá-lo para outro computador diferente daquele para o qual fora desenvolvido.

Uma das possibilidades para tornar um compilador transportável de uma para outra máquina diferente, é escrever o compilador e as rotinas de ponto flutuante numa linguagem de alto nível comum a várias máquinas. O compilador deverá gerar um código intermediário (independente de máquina), que necessita ser traduzido para o código de uma máquina específica para ser executado, conforme mostra a figura 3.12

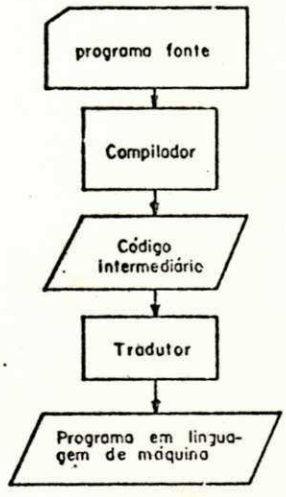


Figura 3.12 - Compilacao e geração de código de maquina de um programa fonte em linguagem de alto nível.

3.4.2 - Rotinas para Processamento de Números de Ponto Flutuante

Existem várias maneiras de resolver o problema de processar números de ponto flutuante quando a máquina não dispõe de um processador em *hardware*.

Temos a opção mais tradicional de desenvolver subrotinas, abertas ou fechadas, em linguagem de baixo nível e colocá-las numa biblioteca do sistema. Existe a possibilidade de gravá-las numa *ROM (Ready Only Memory)* deixando as operações microprogramadas, conseguindo incluir macro-instruções ao conjunto básico de instruções.

A nossa tarefa aquí, será descrever os requisitos necessários para compor o processador em *software*. O trabalho inicial será o mesmo que efetuamos para o *hardware*, ou seja, escolha da base, escolha dos algorítmos, definição da aritmética e finalmente a estrutura global. Basicamente, o processador deverá ter os

seguintes tipos de rotinas [*PRICE, 1968*] :

- Rotinas para as operações aritméticas básicas

  . Adição/Subtração
  . Multiplicação
  . Divisão

- Operações de transferência

  . Carregue pseudo-registradores
  . Armazene conteúdo dos pseudo-registradores

- Operações de conversões

  . Inteiro para ponto flutuante
  . Ponto flutuante binário para decimal
  . Decimal de ponto flutuante para binário
  . Ponto flutuante para inteiro
  . Mudança de sinal de um número de ponto flutuante
  . Valor absoluto de um número de ponto flutuante

- Outras funções (opcional)

  . Seno
  . Cosseno
  . Raiz quadrada
  . Exponencial
  . Logaritmo
  . etc...

- Rotinas para análise de erros

A função básica do *software* processador, será dar um tratamento inicial aos números de ponto flutuante, de forma que os expoentes e as mantissas possam ser tratados separadamente pela unidade aritmética de ponto fixo, como ilustra a figura 3.13, e depois dar o tratamento final para unir o expoente e a mantissa do número.

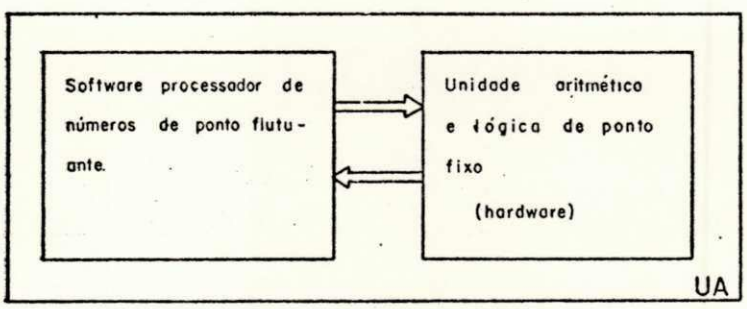


Figura 3.13 - Layout da pseudo unidade aritmética de ponto flutuante.

Não apresentaremos um projeto de um processador de ponto flutuante em *software*, porque seria basicamente simular o que foi apresentado para *hardware*. A implementação do processador é simples, desde que a aritmética esteja bem definida.

UA - Unidade aritmética

CAPÍTULO IV

ANALISE DE ERROS

4.1 - Introdução

A análise de erros em um resultado numérico é fundamental para qualquer computação inteligente, seja ela feita manualmente ou com um computador [*McCRAKEN, 1978*] . Respostas Numéricas para problemas, geralmente contém erros que proveêm de duas áreas - aquelas inerentes a formulação matemática do problema e aqueles sujeitos durante os cálculos da solução numérica [*RALSTON, 1965*]. A primeira categoria inclui os erros introduzidos na formulação matemática de um problema que espelha somente uma aproximação para a situação física. Tais erros são muitas vezes desprezíveis, como no caso dos efeitos relativísticos, que são desprezados nos problemas da mecânica clássica. Se eles não são desprezíveis, então, não importa quão correta seja a computação numérica, sem

pre haverá um erro significativo na resposta. Uma outra fonte de erros inerentes é a incerteza nos dados físicos. Geralmente são desprezíveis quando causados pelos erros nas constantes físicas (por exemplo, na constante gravitacional), mas quando são resultados de erros nos dados empíricos, dignos de soluções computadas, devem ser cuidadosamente ponderadas contra estes erros.

Há três fontes principais dos erros computacionais:

(a) O erro grosseiro, ou erro causado por negligência;

(b) Solução de uma aproximação do problema formulado. Um caso típico é a substituição de um processo infinito por uma aproximação finita [*KNUTH, 1971*] ;

(c) Corte ou arredondamento dos números causados pelas limitações da máquina.

Observemos agora, em poucos exemplos, a importância da análise de erros.

O primeiro exemplo, refere-se ao cálculo de uma das raízes da equação $x^2 + 0.4002x + 0.00008 = 0$, usando a aritmética $PF(10, 4, c)$ e a fórmula

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

a resposta é -0.00015. A aritmética introduz um erro de 25%. A

raiz correta obtida com o uso da aritmética $PF(10,8,c)$ é -0.0002. Isto não significa que a aritmética de oito dígitos resolverá todos os problemas.

Considere agora a série de *Taylor* para o seno

$$Sen\ x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \ldots$$

Esta série é teoricamente válida para qualquer ângulo finito, onde o erro de truncamento cometido pelo corte dos termos menores do que um determinado limite, digamos 10^{-16}, é menor em valor absoluto que o primeiro termo desprezado. Esta afirmação seria verdadeira se houvesse alguma forma de conservar um número infinito de dígitos em cada resultado aritmético.

A avaliação do seno pela série de *Taylor* numa máquina que utiliza a aritmética $PF(16,14,c11)$ para ângulo 1110^0 o erro em valor absoluto é da ordem de 10^{-11}, já para ângulos maiores, por exemplo, 2550^0 o resultado é totalmente sem significado.

Sem nos estendermos em novos exemplos, fica claro que, sem uma análise dos erros nos cálculos, não sabemos muito sobre os resultados.

4.2 - Erro Relativo

Se $\tilde{x}$ é uma aproximação de x e $x \neq 0$, o erro relativo ρ é definido por

$$\rho = \frac{\tilde{x} - x}{x} \tag{4.1}$$

isto é equivalente a

$$\tilde{x} = (1 + \rho)x. \tag{4.2}$$

Note que ρ é um número com sinal. O valor absoluto de $\tilde{x}$ é maior que $|x|$ quando ρ é positivo, e é menor quando $-1 < \rho < 0$. Se $\rho < -1$, $\tilde{x}$ tem o sinal incorreto. Se $\tilde{x} = x = 0$, então $\rho = 0$, se $x = 0$ e $\tilde{x} \neq 0$, $\rho = \infty$

Suponha que x é escrito na forma:

$$x = r^e m, \quad r^{-1} \leq |m| < 1. \tag{4.3}$$

Não é necessário que x esteja em $S(r,p)$, pois desta forma m poderá ter um número ilimitado de dígitos.

Seja $\tilde{x} = r^e \tilde{m}$. O expoente e foi usado para x e $\tilde{x}$, logo não podemos exigir que $\tilde{m}$ seja normalizado, assim

$$\rho = \frac{\tilde{x} - x}{x} = \frac{\tilde{m} - m}{m} . \tag{4.4}$$

Portanto, o erro relativo na aproximação $\tilde{x} \approx x$ é também o erro relativo nas mantissas, quando os números têm o mesmo expoente.

Um caso importante surge quando x satisfaz (4.3) $\tilde{x} = \overline{x}$ onde $\overline{x}$ é a aproximação pelo corte de p dígitos de x na base r. Então $|\tilde{m}| \leq |m|$, assim $\rho \leq 0$. Uma vez que

$$|\tilde{m} - m| < r^{-p} \quad \text{e} \quad r^{-1} \leq |m| \leq 1.$$

(4.4) produz

$$0 \geq \rho > -r^{-(p-1)}. \tag{4.5}$$

Frequentemente re-escrevemos (4.5) como

$$\bar{x} = (1-\rho)x, \quad 0 \leq \rho < r^{-(p-1)}. \tag{4.6}$$

Este limite de erro relativo proveniente do corte de um número em $S(r,p)$ é bastante conveniente, entretanto podemos melhorá-lo, para uso ocasional. Suponhamos que x satisfaz (4.3) e escrevemos $\bar{x} = (1-\rho)x$. Agora $|\bar{x}| = (1-\rho)|x|$, assumindo que $x > 0$. Seja $x - \bar{x} = \varepsilon$, então

$$\rho = \frac{\varepsilon}{\bar{x}+\varepsilon}$$

onde $0 \leq \varepsilon < r^{e-p}$ e $r^{e-1} \leq \bar{x} < r^{e}$. Seja $f(t) = \frac{t}{\bar{x}+t}$ então

$$f'(t) = \frac{\bar{x}}{(\bar{x}+t)^2} > 0,$$

o valor máximo de $f(t)$ no intervalo $0 \leq t \leq r^{e}-p$ e $f(r^{e}-p)$. Portanto

$$0 \leq \rho < f(r^{e}-p) = \frac{r^{e-p}}{\bar{x}+r^{e-p}} \tag{4.7}$$

Agora $\bar{x} \geq r^{e}-1$, assim (4.7) produz

$$\bar{x} = (1-\rho)x, \quad 0 \leq \rho < \frac{r^{-(p-1)}}{1+r^{-(p-1)}}. \tag{4.8}$$

4.3 - Erro Relativo em $PF(r,p,c\ell q)$

Estudaremos o erro relativo introduzido pela operação aritmética em $PF(r,p,c\ell q)$ e o modo como o erro se propaga pelas operações aritméticas. Assuminos $q>0$ para o estudo que realizaremos.

4.3.1 - Erro Relativo na Multiplicação e Divisão

Em $PF(r,p,c\ell q)$ com $q > 0$, temos $a*b = \overline{ab}$ e $a\div b = \overline{a/b}$, logo

$$a*b = (1 - \rho)ab, \qquad 0 \leq \rho < r^{-(p-1)};$$
$$a\div b = (1 - \rho)\frac{a}{b}, \qquad 0 \leq \rho < r^{-(p-1)}. \tag{4.12}$$

Suponha que $\tilde{x}$ e $\tilde{y}$ são aproximações de x e y respectivamente, com

$$\tilde{x} = (1 + \sigma)x$$
$$\tilde{y} = (1 + \tau)y. \tag{4.13}$$

Se $\tilde{x}$ e $\tilde{y}$ são resultados de cálculos aritméticos, normalmente não conhecemos os sinais de σ e τ, e os seus limites podem ser tão grandes quanto $r^{-(p-1)}$. Agora se $\tilde{x}$ e $\tilde{y}$ são números armazenados na máquina, podemos ter $\tilde{x} * \tilde{y}$.

De (4.1) podemos escrever

$$\tilde{x} * \tilde{y} = (1 - \rho)\tilde{x}\tilde{y}\ , \quad 0 \leq \rho < r^{-(p-1)}$$

e assim

$$\tilde{x} * \tilde{y} = (1 - \rho)(1 + \sigma)(1 + \tau)xy.$$

substituindo o erro relativo por ψ, temos

$$\tilde{x} * \tilde{y} = (1 + \psi)xy$$

onde

$$\psi = -\rho + \sigma + \tau - \rho\sigma - \rho\tau + \sigma\tau - \rho\sigma\tau \ . \qquad (4.14)$$

Suponha que ρ, σ e τ representam valores relativamente pequenos, digamos menores do que 10^{-5}. Então os últimos termos do produto em (4.14) são bem insignificantes em relação a ρ, σ e τ, assim

$$\psi = -\rho + \sigma + \tau \ . \qquad (4.15)$$

Uma vez que não conhecemos os sinais de σ e τ o sinal em (4.15) não tem significado, pois não sabemos se σ e τ serão adicionados ou subtraídos.

A divisão se comporta de forma similar. Se $\tilde{x}$ e $\tilde{y}$ satisfazem (4.13), escrevemos

$$\tilde{x} \div \tilde{y} = (1 - \rho)\frac{\tilde{x}}{\tilde{y}}, \quad 0 \leq \rho < r^{-(p-1)}$$

assim

$$\tilde{x} \div \tilde{y} = \frac{(1 - \rho)(1 + \sigma)}{(1 + \tau)} \frac{x}{y} \ .$$

Se escrevermos

$$\tilde{x} \div \tilde{y} = (1 + \delta)\,\frac{x}{y}$$

temos

$$1 + \delta = \frac{(1 + \rho)(1 + \sigma)}{(1 + \tau)} .$$

Lembrando que $|\tau| < 1$, obtemos uma aproximação útil para $1 + \delta$,

$$\frac{1}{1 + \tau} = 1 - \tau + \tau^2 + \tau^3 + \ldots$$

Se $|\tau|$ é muito próximo de zero, isto dá

$$\frac{1}{1 + \tau} \approx 1 - \tau$$

então

$$\delta = - \rho + \sigma - \tau$$

que tem pouca significância pelo mesmo motivo de (4.15).

4.3.2 - Erro Relativo na Adição e Subtração

No Capítulo II vimos que no caso da adição de magnitudes temos,

$a \oplus b = \overline{a + b}$, e

$a \ominus b = \overline{a - b}$, desta forma

$$a \oplus b = (1 - \rho)(a + b),\ 0 \leq \rho < r^{-(p-1)}$$

$$a \ominus b = (1 - \rho)(a + b),\ 0 \leq \rho < r^{-(p-1)}. \qquad (4.17)$$

A subtração pode ser resuzida para a adição $a \oplus b$ onde $a > 0$ e $b < 0$ com $a \geq |b|$. A operação $a \oplus b$ é exata se $a = -b$, desta forma assumiremos que $a > 0$, $b < 0$ a, $b > -a$. Escrevendo

$$a + b = r^e m\ , \qquad r^{-1} \leq m < 1$$

e $$a \oplus b = r^e \tilde{m}\ , \qquad |\tilde{m} - m| < r^{-p}$$

assim

$$a \oplus b = (1 + \rho)(a + b),\ |\rho| < r^{-(p-1)} \qquad (4.18)$$

Podemos melhorar ligeiramente (4.18) se $a \oplus b > a + b$, então se $a \oplus b \neq \overline{a + b}$, logo $a \oplus b$ é maior do que $a + b$ pelo menos na posição $p + q - 1$. Temos que

$$a \oplus b = (1 + \rho)(a + b),\ -r^{-(p-1)} < \rho < r^{-(p+q-2)} \qquad (4.19)$$

Passaremos a estudar o problema da propagação de erros, embora seja mais vantajoso estudá-lo em termos de erro absoluto.

Suponhamos que $\tilde{x}$ e $\tilde{y}$ são aproximações de x e y respectivamente, satisfazendo (4.13), e com σ e τ maiores do que -1, portanto $\tilde{x}$ e $\tilde{y}$ têm os sinais corretos.

Considerando que x e y são positivos, temos

$$\tilde{x} + \tilde{y} = x + y + \sigma x + \tau y,$$

assim $\tilde{x} + \tilde{y} = (1 + \psi)(x + y)$

onde $\psi\ (x + y) = \sigma x + \tau y.$

Agora

$$\left[min(\sigma,\tau)\right]\ (x + y) \leq \sigma x + \tau y \leq \left[max(\sigma,\tau)\right]\ (x + y)$$

então $$min\ (\sigma,\tau) \leq \psi \leq max(\sigma,\tau) \qquad (4.20)$$

Isto produz

$$|\psi| \leq max\ (|\sigma|,\ |\tau|)$$

que no caso da adição, o erro relativo é no máximo igual ao maior erro relativo dos termos.

Na subtração a aritmética de ponto flutuante não introduz erro, os erros são provenientes dos próprios operandos.

4.4 - Propagação de Erros de Arredondamentos

Para mostrar o crescimento do erro de arredondamento, consideraremos o problema da computação da seguinte expressão:

$$x = \prod_{i=0}^{n} x_i. \qquad (4.21)$$

Assumimos que todos os x_i's são números de ponto flutuante exatos. O procedimento da computação é o seguinte

$$P_o = x_o$$
$$P_k = P_{k-1} * x_k, \quad k = 1, 2, \ldots, n$$

assim $x = P_n$. Uma vez que estamos utilizando $PF(r,p,a)$, temos

$$\tilde{P}_o = P_o$$
$$\tilde{P}_k = \tilde{P}_{k-1} * x_k, \quad k = 1, 2, \ldots, n$$

então,

$$\tilde{P}_k = (1 + \rho_k)\, \tilde{P}_k\, x_k,$$

onde ρ_k depende da aritmética usada. Para $PF(r,p,clq)$ com $q \geq 1$, temos

$$0 \leq \rho_k \leq r^{-(p-1)}$$

Por outro lado, se utilizarmos aritmética com arredondamento, teremos

$$|\rho_k| \leq \frac{1}{2}\, r^{-(p-1)}$$

onde

$$\tilde{P}_1 = (1 + \rho_1)\, P_1 \tag{4.22}$$

e por indução

$$\tilde{P}_k = \left[\prod_{i=1}^{k} (1 + \rho_i) \right] P_k \,. \tag{4.23}$$

Observemos isoladamente cada ρ. Daremos os limites ρ^* e ρ_*, tal que

$$-\rho_* < \rho_i < \rho^* \tag{4.24}$$

Assumiremos que $\rho_* \geq 0$ e $\rho^* \geq 0$ e que $\rho_* < 1$, $1 - \rho_* > 0$. Então

$$-1 + (1 - \rho_*)^n \leq \sigma \leq (1 + \rho_*)^n - 1$$

$$-1 + (1 - \rho_*)^n = -\sum_{k=1}^{n} \binom{n}{k} \rho_*^k (-1)^{k+1} > \sum_{k=1}^{n} \binom{n}{k} \rho_*^k$$

$$= \left[(1 + \rho_*)^n - 1\right]$$

Seja $\bar{\rho}$ maior do que ρ_* e ρ^*. Então

$$|\sigma| \leq (1 + \bar{\rho})^n - 1.$$

Consideremos $(1 + \rho)^n - 1$:

$$(1 + \rho)^n - 1 = \sum_{k=1}^{n} \binom{n}{k} \rho^k = n\rho + \frac{n(n+1)}{2} \rho^2 + \ldots$$

assim se $n\rho$ é pequeno, temos

$$(1 + \rho)^n - 1 \approx n\rho \tag{4.25}$$

Entretanto, desejamos quase sempre um limite para $|\sigma|$ ao invés de uma aproximação. Para $\rho > 0$,

$$(1 + \rho)^n - 1 = -1 + \sum_{k=0}^{n} \binom{n}{k} \rho^k < -1 + \sum_{k=0}^{n} \frac{(n\rho)^k}{k!} = e^{n\rho} - 1$$

então temos

$$|\sigma| < e^{n\rho} - 1. \tag{4.26}$$

4.4.1 - Condição

No estudo de propagação de erros, frequentemente confrontamos com a questão de como um erro nos dados de entrada afeta o resultado de uma função. Em geral, podemos dizer que o problema da computação de

$$y = f(x) \tag{4.27}$$

está bem condicionado se pequenas mudanças em x produzem pequenas mudanças em $f(x)$, enquanto que está mal condicionado ou pobremente condicionado se pequenas alterações em x causam grandes mudanças em $f(x)$. Isto é muito vago, porque não podemos especificar o significado de pequeno e grande. Em alguns casos interessa-nos mudanças absolutas, enquanto em outras mudanças relativas.

A condição do problema (4.27) pode depender de x bem como da função $f(x)$. Portanto, para uma dada função $f(x)$, o problema de sua computação pode estar bem condicionado para alguns valores de x e mal condicionado para outros.

4.5 - Análise de Erros de um Programa

É muito difícil analisar os erros provenientes de programas, principalmente se esses programas não foram desenvolvidos pelo usuário. Isto geralmente é devido ao desconhecimento da implementação do algorítmo utilizado, e dos resultados parciais.

Suponha que temos uma subrotina para calcular $y = ax + b$ e a aritmética é $PF(10,8,c)$ considerando que:

$$a = .56785679$$
$$b = -.30849066$$
$$x = .54325433$$

temos

$$y = -(0.00125993).\ 10^{-8}$$

e

$$\tilde{y} = -10^{-8}$$

Se $\rho = \frac{\tilde{y} - y}{y}$ é o erro relativo, temos então que $\rho = 792.64$. Entretanto o erro absoluto $|\tilde{y} - y|$ é inferior a 10^{-8}. Agora se

$$a = (.56785679).\ 10^{40}$$
$$x = .54325433$$
$$b = -(.30849066).\ 10^{40}$$

teremos $|\tilde{y} - y| = -(.99874007).\ 10^{32}$. Desta maneira concluímos que são irrelevantes os erros absolutos e relativos na análise de erros de um programa.

A análise de erro *"backward"* permitirá uma análise um pouco mais confiável. Esta análise consiste em pesquisar um número $\tilde{x}$ onde $\tilde{y} = f(\tilde{x})$ e tentar limitar tanto a diferença absoluta $\tilde{x} - x$ como a diferença relativa $(\tilde{x} - x)/x$ ao invés de limitar $\tilde{y} - y$ ou $(\tilde{y} - y)/y$. Sabemos que ao tentarmos avaliar $y = f(x)$ normalmente temos $\tilde{y}$ e não y como era de se esperar, onde $\tilde{y} \approx y$. Portanto, ao invés de perguntar quão bem foi avaliada a função, procuramos encontrar qual é o valor $\tilde{x}$ que produz $\tilde{y}$.

Pode haver vários valores de $\tilde{x}$ com $\tilde{y} = f(\tilde{x})$, no caso devemos escolher um fechado em x, ou seja, que $|\tilde{x} - x| < \sigma$ ou que $|(\tilde{x} - x)/x| < \rho$.

Com esta abordagem, observamos os erros na computação como sendo equivalente a uma perturbação dos dados, e podemos dar limites a esta perturbação.

Uma outra possibilidade é a análise estatística dos erros. Muitas vezes é mais apropriado considerar a média dos erros ao invés do erro máximo [*HENRICI, 1964*] . O enfoque probabilístico para erros de arredondamentos tem fundamento, porque os erros individuais de cada operação podem ter sinais diferentes com muitos deles se cancelando e outros jamais atingindo o valor máximo calculado.

É claro que, sobre uma dada máquina para um dado problema, os erros de arredondamentos locais, não são de fato variáveis aleatórias. Podemos adotar um modelo estocástico de propagação de erros, onde os erros locais são tratados como se fossem variáveis aleatórias.

Assuminos que os erros locais estão uniformemente distribuidos entre seus valores extremos $-\varepsilon$ e ε. A densidade de probabilidade $p(x)$ desta distribuição é dada por.

$$p(x) = \begin{cases} 0, & x < -\varepsilon \\ c, & -\varepsilon \leq x \leq \varepsilon \\ 0, & x > \varepsilon \ . \end{cases} \qquad (4.28)$$

A constante c é determinada pela condição

$$\int_{-\infty}^{\infty} p(x)\ dx = 1.$$

Para fins teóricos é mais conveniente assumir que os erros de arredondamentos são normalmente distribuidos. A distribuição normal é definida pela densidade de probabilidade

$$p(x) = \frac{1}{2\pi\sigma}\ e^{-(x^2/2\sigma^2)} \qquad (4.29)$$

onde σ é o desvio padrão da distribuição, que mede a extensão da distribuição.

4.6 - Análise Automática de Erros

Mostraremos diversos enfoques que têm sido usados na tentativa de fazer com que o computador nos assista na análise de erros. Com estes enfoques exigimos que o computador produza a

resposta e o grau de confiabilidade da mesma. Nenhum deles é uma panacéia, tanto é que não são indicados para os processamentos normais em substituição a aritmética de ponto flutuante, mas apenas para dar uma estimativa razoável dos erros.

4.6.1 - Aritmética de Significância

A aritmética de significância é uma técnica para análise automática de erros de arredondamentos. Com esta aritmética, utilizamos números não normalizados durante os cálculos. O objetivo é representar cada número com zeros a esquerda, de modo que os dígitos restantes sejam significativos. As operações aritméticas devem ser modificadas para que produzam resultados não normalizados com o número correto de zeros a esquerda. A adição e subtração são fáceis de implementar, basta que se omita a pós-normalização. Para a multiplicação e a divisão, a modificação é baseada na idéia de que a resposta deve ter tantos zeros a esquerda quanto o operando significativo. A outra modificação necessária é na biblioteca de programas.

É altamente desapontador saber que a aritmética de significância, cujo objetivo é mostrar quanto a resposta é confiável, não possa garantir que os dígitos produzidos são corretos. Pelo menos, resta o consolo de que fornece uma boa indicação do grau de certeza da resposta.

4.6.2 - Modo *"Noisy"*

Algumas máquinas provêm o modo *"noisy"* além da aritmética de ponto flutuante. A execução da aritmética de ponto flutuante com o modo *"noisy"* modifica os dígitos que serão deslocados na resposta quando a pós-normalização é necessária. Para utilizar o modo *"noisy"*, executamos os cálculos duas vezes, com a aritmética normal e com o modo *"noisy"*. A indicação de quanto as duas respostas se aproximam fornece o grau de confiabilidade do resultado. A idéia básica deste método, consiste em inserir *"ruídos"* quando estamos incertos de qual dígito deve ser inserido. Na pós-normalização, quando da execução de uma subtração na base r, os dígitos deslocados são os complementos de $(r - 1)$ dos dígitos que deveriam ser deslocados pela aritmética de ponto flutuante. Isto também é válido para a multiplicação e adição. Para a divisão, pode-se estender o dividendo pela anexação de vários dígitos de $(r - 1)s$ a direita antes da divisão.

Como ilustração de uma implementação típica do modo *"noisy"*, considere um exemplo em $PF(10,8,a)$. Seja

$$x = 1234.5678$$
$$y = 1234.4321$$
$$z = x - y.$$

Tanto em $PF(10,8,c)$ como em $PF(10,8,r)$ o valor de z deve ser $.13570000$. O resultado foi deslocado quatro dígitos para a esquerda na pós-normalização, e inserimos quatro zeros a direita. No modo *"noisy"*, em lugar de inserir zeros a direita, inserimos noves, então $z = .13579999$. Assim, podemos afirmar preciptada

mente que o resultado correto de z deve estar no intervalo. $.13570000 \leq z \leq .13579999$. Essa afirmação não é sempre válida, porque é possível inserir os ruídos na direção errada e o resultado do modo *"noisy"* ser menor do que o processamento normal.

4.6.3 - Aritmética de Intervalo

A sua implementação tem sido feita por chamadas de subrotinas ao invés de códigos de operação de *hardware*. A idéia básica é que cada número x, nos cálculos, seja representado por um intervalo (x_1, x_2) onde x_1 e x_2 são escolhidos de tal modo que $x_1 \leq x \leq x_2$. Portanto, se temos uma aproximação $\tilde{x}$ para x com $|\tilde{x} - x| \leq \varepsilon$, devemos representar x pelo intervalo $(\tilde{x} - \varepsilon, \tilde{x} + \varepsilon)$. É necessário que $x_2 < x_1$, mas permitiremos o uso do intervalo degenerado (x, x) para representar um número que conhecemos ser exato.

Através dos cálculos, representaremos os números por meio de intervalos. O objetivo é representar a resposta y por um intervalo (y_1, y_2) com $y_1 \leq y \leq y_2$. Se o intervalo (y_1, y_2) é bastante pequeno, digamos $|y_2 - y_1| \leq 10^{-6}$, o ponto médio do intervalo provê uma boa aproximação para y. Agora se o intervalo for grande, obtemos pouca informação sobre y, mas podemos afirmar que não temos uma boa aproximação para y.

As operações de adição e subtração são definidas como:

$$(x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2)$$
$$(x_1, x_2) - (y_1, y_2) = (x_1 - y_2, x_2 - y_1) \qquad (4.30)$$

De maneira similar, a multiplicação é dada por:

$$(x_1, x_2).(y_1, y_2) = (x_1, x_2)$$
$$z_1 = min\ (x_1y_1\ ,\ x_1y_2\ ,\ x_2y_1\ ,\ x_2y_2)$$
$$z_2 = max\ (x_1y_1\ ,\ x_1y_2\ ,\ x_2y_1\ ,\ x_2y_2) \qquad (4.31)$$

Definiremos a divisão por:

$$\frac{(x_1, x_2)}{(y_1, y_2)} = (z_1, z_2)$$
$$z_1 = min\ (\frac{x_1}{y_1}\ ,\ \frac{x_1}{y_2}\ ,\ \frac{x_2}{y_1}\ ,\ \frac{x_2}{y_2}\ )$$
$$z_2 = max\ (\frac{x_1}{y_1}\ ,\ \frac{x_1}{y_2}\ ,\ \frac{x_2}{y_1}\ ,\ \frac{x_2}{y_2}\ ) \qquad (4.32)$$

não sendo válida a definição se o zero pertencer ao intervalo (y_1, y_2).

Entretanto, não podemos utilizar (4.31) e (4.32) diretamente, porque o número de dígitos necessários para representar os pontos finais do intervalo deve crescer rapidamente. A solução é usar ao invés de (y_1, y_2) o intervalo (ω_1, ω_2) onde ω_1 e ω_2 pertencem a $S(r,p)$ com $\omega_1 \leq z_1$ e $\omega_2 \geq z_2$.

No caso das bibliotecas de programas que computem funções, quando usamos a aritmética de intervalo, o argumento será um intervalo (x_1, x_2) e a resposta um outro intervalo (y_1, y_2) .

Para a função $f(x)$, temos

$$y = \min_{x_1 \leq x \leq x_2} f(x)$$

$$y = \max_{x_1 \leq x \leq x_2} f(x) \qquad (4.33)$$

Para funções monotônicas, os números em (4.33) são facilmente computados.

4.6.4 - Re-execução do Programa numa Precisão Maior

A maneira mais comum de se utilizar o computador para estudar a exatidão de nossas respostas, é simplemente re-executar o programa numa precisão maior. Suponha que a resposta original foi y_1 e na re-execução numa precisão maior, a resposta seja y_2. Podemos estimar os erros relativo e absoluto como sendo $(y_1 - y_2)/y_2$ e $y_1 - y_2$, respectivamente, considerando que a maior precisão produz resposta mais exata. Geralmente essa consideração é verdadeira, mas falha se os erros forem provenientes dos dados.

A justificativa para o seu uso é a facilidade de conversão dos programas para a re-execução, pois normalmente basta modificar a declaração das variáveis, alterar as constantes utilizadas no corpo do programa e recompilar. Apesar de que com isto incorremos em outros erros, como imprecisão das constantes ou mesmo imprecisão nos dados que estávamos utilizando até então.

CAPÍTULO V

ESTUDO DE CASOS

5.1 - Introdução

No Capítulo anterior apresentamos um estudo para análise de erros computacionais onde descrevemos diversos métodos para análise de erros com o auxílio do próprio computador. Neste Capítulo, estudaremos dois casos típicos de instabilidade numérica causado pelo próprio algorítmo e também pelas restrições das máquinas. Em todos os casos atenuamos o problema da instabilidade modificando os algorítmos e também mudando a precisão da aritmética utilizada.

Além disso, para cada algorítmo, implementamos em quatro computadores diferentes para verificar até onde a aritmética utilizada influi nos resultados finais. Em 5.2 apresentaremos as

aritméticas utilizadas pelos computadores testados, para que possamos realizar a análise dos resultados.

Finalmente, em 5.3 e 5.4 apresentaremos os algorítmos com o estudo das causas das suas instabilidades e tentaremos efetuar a sua recuperação por meio do aumento de precisão ou modificando-os. Em 5.3 tiramos nossas conclusões quanto as máquinas utilizadas e ainda sugerimos alguns ítens que devem ser levados em consideração para se obter maior exatidão nos cálculos computacionais.

5.2 - Apresentação da Aritmética dos Computadores Utilizados

5.2.1 - Computador IBM/370-145

As operações aritméticas de ponto flutuante são executadas por *hardware especial* que o usuário pode ativar ou desativar, dependendo das finalidades a que se destina o computador.

A aritmética de ponto flutuante utilizada para a precisão simples é *PF(16,6,c11)* e para a precisão dupla é *PF(16,14,c11)*.

O formato dos dados de precisão simples é dado pela figura 5.1.

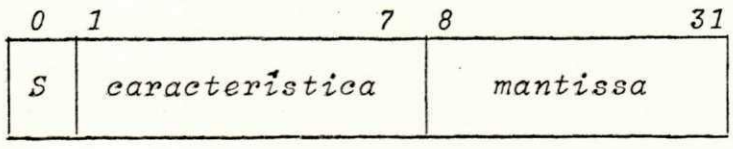


Figura 5.1 - Formato dos dados de precisão simples do IBM/370-145.

Um número consiste de um bit de sinal *(0 = positivo, 1 = negativo)*, de uma característica dada por um expoente hexadecimal mais excesso 64 e uma mantissa de 24 bits normalizada à esquerda (ponto decimal a esquerda). Os números negativos serão representados da mesma forma dos números positivos, com excessão do bit de sinal.

Os dados de precisão dupla são armazenados conforme mostra a figura 5.2

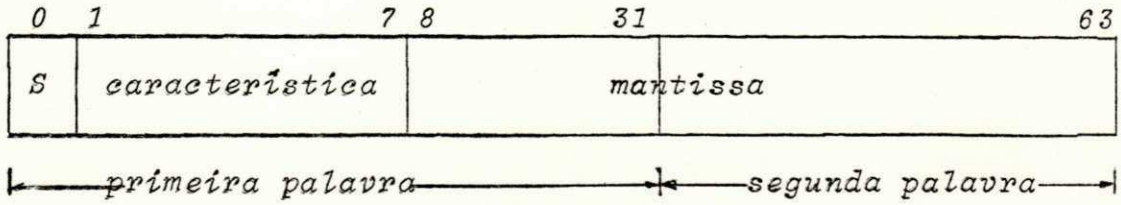


Figura 5.2 - Formato dos dados de precisão dupla do IBM /370-145.

Tudo funciona como em precisão simples, apenas que é dado mais uma palavra estendendo a mantissa mais 32 bits.

5.2.2 - Computador *DEC SYSTEM 10 (PDP10-90)*

As operações aritméticas de ponto flutuante são executadas por *hardware específico*. A aritmética utilizada para precisão simples é *PF(2,27,c)* e para a precisão dupla é *PF(2,62,R)*

Os dados de precisão simples são armazenados conforme o formato apresentado na figura 5.3.

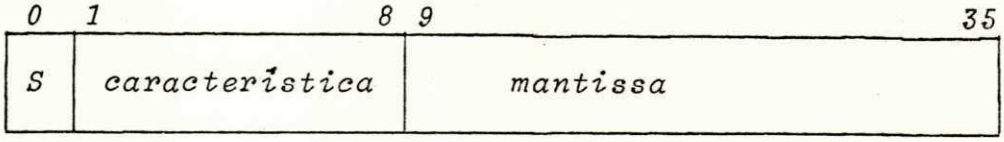


Figura 5.3 - Formato dos dados de precisão simples do PDP10-90.

Cada número consiste de um bit de sinal *(0 = positivo, 1 = negativo)*, oito bits de característica *(expoente + excesso 128)* e 27 bits de mantissa com o ponto decimal mais a esquerda. Um número negativo é representado pelo complemento de 2 de toda a palavra, onde o *hardware* compensa automaticamente, colocando a característica em complemento de 1 e a mantissa em complemento de 2.

Os dados de precisão dupla são armazenadas de acordo com o formato da figura 5.4.

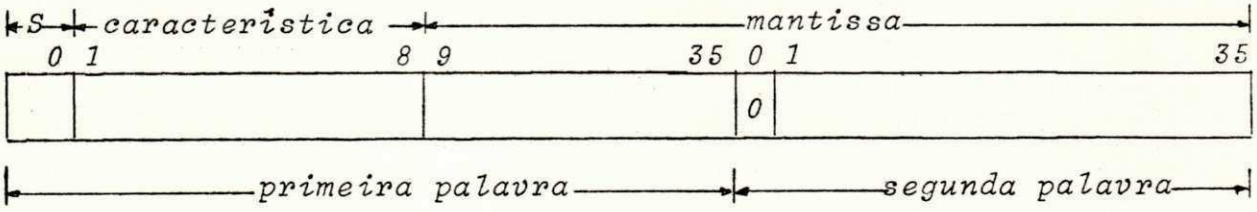


Figura 5.4 - Formato dos dados de precisão dupla do PDP10-90.

O bit *0* da palavra de mais baixa ordem é sempre *0*, sendo desprezado em todos os operandos. A primeira palavra funciona como na precisão simples e a segunda funciona como extensão da mantissa. Em precisão dupla, todos os operandos e resultados são de tamanho duplo e todas as instruções calculam um tamanho extra de resposta, que é arredondado para o comprimento de precisão

dupla.

5.2.3 - Computador *Burroughs 1700-26*

É um computador cujas operações aritméticas de ponto flutuante são realizadas por *software*.

A aritmética de precisão simples é *PF(2,24,R)* e para precisão dupla é *PF(2,60,R)*.

Os dados de precisão simples são armazenados segundo o formato da figura 5.5.

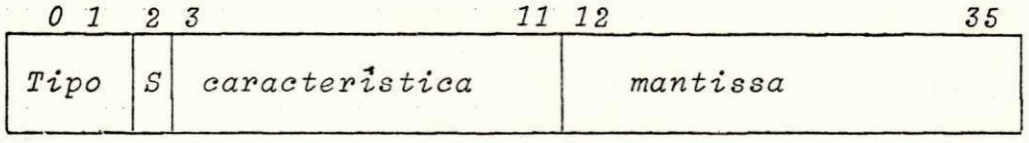


Figura 5.5 - Formato dos dados de precisão simples do Burroughs 1700-26.

Cada número consiste de 2 bits de tipo *(01 = ponto flutuante de precisão simples)* 1 bit de sinal *(1 = negativo e 0 = positivo)*, de uma característica de 9 bits *(expoente + excesso 256)* e uma mantissa de 24 bits *(ponto decimal mais a esquerda)*. O número negativo difere do positivo apenas pelo bit de sinal.

Os dados de precisão dupla são armazenados conforme mostra a figura 5.6.

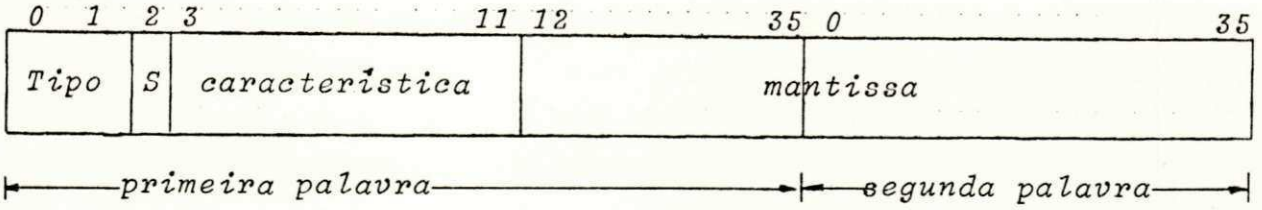


Figura 5.6 - Forma dos dados de precisão dupla do Burroughs 1700-26.

Os bits de tipo 11, significa ponto flutuante de precisão dupla. O resto é similar a precisão simples, apenas a mantissa é maior, ou seja, 60 bits.

5.2.4 - Computador IBM 1130

As operações aritméticas de ponto flutuante neste computador são efetuadas por rotinas programadas e armazenadas numa biblioteca. A aritmética de ponto flutuante de precisão simples é assim definida *PF(2,23,c)* e para precisão estendida, a aritmética utilizada é *PF(2,31,c)*.

Os dados de ponto flutuante de precisão simples são armazenados conforme mostra a figura 5.7.

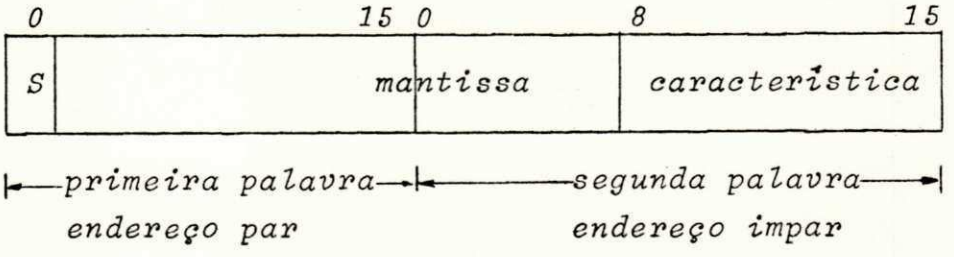


Figura 5.7 - Formato dos dados de precisão simples do IBM 1130.

Cada número consiste de um bit de sinal, 23 bits de precisão *(mantissa)*, e um expoente em forma de característica *(expoente + excesso 128)*. O número positivo é armazenado na forma natural com o bit de sinal igual a *0* e o negativo na forma de complemento de 2 *(bit de sinal igual a 1)*.

Os dados de precisão estendida são armazenados de acordo com a figura 5.8.

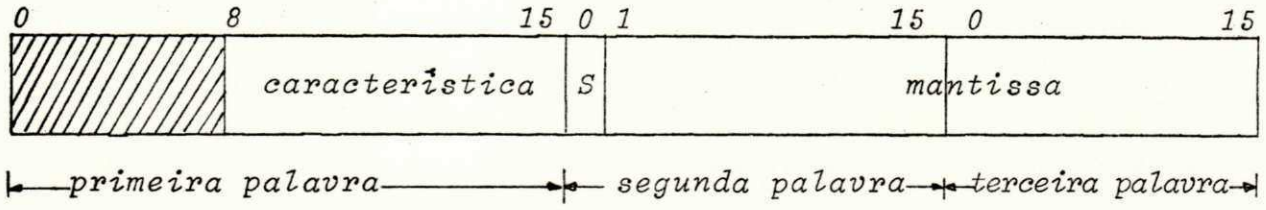


Figura 5.8 - Formato dos dados de precisão estendida do IBM 1130.

A diferença básica para a precisão simples é que o número de bits na mantissa é igual a 31, e que a primeira palavra pode estar num endereço par ou impar. Para ambos os casos o ponto decimal é considerado como estando mais a esquerda da mantissa.

5.3 - Erros no Cálculo Direto da Série do SENO

A dificuldade numérica no caso, surge porque o argumento da função é tão grande que muitos dígitos significativos são perdidos antes do critério de convergência ser satisfeito.

A série de Taylor para seno

$$sen\ x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

é teoricamente válida para todos os valores de x. Na prática, entretanto, é quase inútil para valores grandes de x, por exemplo na precisão simples do IBM/370-145, para $k = 4$ em $x = n + 2k\Pi$. Para evitarmos os termos intermediários muito grandes para uma variável de ponto flutuante, que ultrapassariam os tamanhos permitidos em quase todos os computadores, utilizaremos uma relação de recorrência. Tendo o primeiro termo x, podemos obter o próximo termo multiplicando por $-x^2$ e dividindo pelo produto dos dois denominadores próximos.

No Apêndice, apresentamos o algorítmo *A.1*, para o cálculo direto da série do seno.

O algorítmo *A.1* foi implementado na linguagem de programação *FORTRAN* em precisão simples, onde utilizamos a tolerância de 1.0×10^{-8}. Executamos nos quatro computadores apresentados em 5.2 e os resultados estão nas tabelas 5.1 a 5.4.

Os resultados apresentados são para 30^0 e para os seus múltiplos de 360^0. Os valores deveriam ser todos iguais a $1/2$. O seno de 30^0 deu aproximação para o IBM 1130 ($\rho = 1 \times 10^{-5}$) e exato para os demais computadores. Para 360^0 o resultado foi aproximado para todos, $\rho = 1.3 \times 10^{-5}$ para o IBM/370, $\rho = 1.0 \times 10^{-6}$ para

---

ρ = erro absoluto

GRAUS	X	SÉRIE DO SENO	FUNÇÃO SENO
30.	0.523599	0.500000	0.500000
390.	6.806784	0.500013	0.500000
750.	13.089960	0.499872	0.499999
1110.	19.373150	-11.297890	0.499999
1470.	25.656320	891.799300	0.499988
1830.	31.939510	639408.000000	0.499990
2190.	38.222700	**********	0.499993
2550.	44.505880	**********	0.499995
2910.	50.789070	**********	0.499997
-30.	-0.523599	-0.500000	-0.500000
-390.	-6.806784	-0.500013	-0.500000

TABELA 5.1 - Cálculo do seno pela série de Taylor, precisão simples do sistema IBM/370-145.

GRAUS	X	SÉRIE DO SENO	FUNÇÃO SENO
30.	0.523599	0.500000	0.500000
390.	6.806784	0.500001	0.500000
750.	13.089969	0.500089	0.500000
1110.	19.373155	0.531730	0.500000
1470.	25.656340	43.891564	0.500000
1830.	31.939525	-17285.886000	0.500000
2190.	38.222711	**********	0.500000
2550.	44.505896	**********	0.500000
2910.	50.789091	**********	0.500000
-30.	-0.523599	-0.500000	-0.500000
-390.	-6.806784	-0.500001	-0.500000

TABELA 5.2 - Cálculo do seno pela série de Taylor, precisão simples do sistema DEC 10 (PDP10-90)

GRAUS	X	SÉRIE DO SENO	FUNÇÃO SENO
30.	0.523599	0.500000	0.500000
390.	6.806784	0.500009	0.500000
750.	13.089970	0.498062	0.500001
1110.	19.373160	-0.396267	0.499999
1470.	25.656340	87.088230	0.500001
1830.	31.939530	-9152.599000	0.500001
2190.	38.222710	**********	0.499999
2550.	44.505900	**********	0.499999
2910.	50.789080	**********	0.500002
-30.	-0.523599	-0.500000	-0.500000
-390.	-6.806784	-0.500009	-0.500000

TABELA 5.3 - Cálculo do seno pela série de Taylor, precisão simples do Burroughs 1700 - 26.

GRAUS	X	SÉRIE DO SENO	FUNÇÃO SENO
30.	0.523598	0.499999	0.500000
390.	6.806784	0.499994	0.499999
750.	13.089971	0.498126	0.500000
1110.	19.373157	-2.789121	0.499999
1470.	25.656341	-934.850465	0.499997
1830.	31.939529	-401621.12[illegible]354	0.500000
2190.	38.222717	**********	0.499999
2550.	44.505905	**********	0.500001
2910.	50.789085	**********	0.499996
-30.	-0.523598	-0.499999	-0.500000
-390.	-6.806784	-0.499994	-0.499999

TABELA 5.4 - Cálculo do seno pela série de Taylor, precisão simples do IBM 1130.

o *PDP10-90*, $\rho = 9\times10^{-6}$ para *Burroughs 1700-26 (B1700-26)*, e $\rho = 6\times10^{-6}$ para o IBM 1130. Os resultados deixaram a ter significado a partir de 1110^0, exceto para o *PDP10-90* que isto aconteceu apenas a partir de 1470^0.

Considerando que o algorítmo é teoricamente correto, deve existir uma causa para que ocorram estas distorções. Pela estrutura de cada máquina, era de se esperar que o *PDP10-90* tivesse um pouco mais de precisão, pois mantém 27 bits na mantissa, enquanto o IBM/370 mantém 24 o *B1700-26 24* bits e o IBM 1130 23 bits. A razão porque o /370 divergiu mais rapidamente foi devido a sua estrutura hexadecimal, porque na normalização o deslocamento é feito de 4 em 4 bits. Os demais computadores utilizam a aritmética binária onde a normalização é feita deslocando-se bit a bit.

Com referência ao IBM /370, podemos assegurar apenas que sua precisão simples está entre 21 a 24 bits. Quando o pior caso ocorrer, terá 2 bits a menos do que o IBM 1130, 3 bits a menos que o *B1700-26* e 6 bits a menos do que o *PDP10-90*.

Devido a natureza discreta dos números de ponto flutuante, o método falha, porque em cada termo calculado, dígitos significativos são perdidos a ponto dos últimos termos calculados não terem mais nenhum dígito significativo, pois foram todos perdidos por corte ou arredondamento dos números para armazená-los em palavras de comprimentos limitados.

Um outro motivo pelo qual o método falha, é porque as adições dos termos estão sendo realizadas numa sequência longe de ser a ideal, a melhor seria começar com os termos menores para

conservar as somas parciais tão pequenas quanto possíveis. Pode mos ainda indicar o problema de conversão de base dos dados de entrada e saída.

No intuito de confirmar as afirmações acima, re-executamos os programas em precisão dupla, com tolerância 1.0×10^{-16}, cujos resultados estão nas tabelas 5.5 a 5.8.

Se os computadores trabalhassem com aritmética infinite simal, todos os resultados obtidos seriam válidos, mas apenas com o uso da precisão dupla não é suficiente o número de dígitos disponíveis, como mostram as tabelas 5.5 a 5.8. Os computadores IBM/370, *PDP10-90*, *B1700-26* e *IBM 1130* apresentaram resultados sem nenhum significado a partir de 2550^0, 2910^0, 2550^0 e 1470^0 respectivamente. A explicação para essas divergências, é a mesma dada para a precisão simples, ou seja, de 5.2 temos que o IBM 1130 mantém 31 bits na mantissa, o IBM/370-145 mantém 50 bits, *B1700-26* 60 bits e o *PDP-10-90* mantém 62 bits na mantissa.

A precisão dupla não é suficiente para análise apropriada do algorítmo. Para este caso, é aconselhável encontrar outra ma neira melhor de computar os ângulos.

A solução mais apropriada para o caso é reduzir o tamanho do argumento, utilizando a periodicidade da função. Desta manei ra obteríamos um resultado significativo para as quatro máquinas utilizadas, como pode ser observado nas tabela 5.1 a 5.8, onde para 30^0 os senos calculados foram bastante significativos. Se reduzirmos todos os ângulos para o intervalo $0^0 \leq \hat{a}ngulo \leq 360^0$, garantiremos a confiabilidade dos resultados.

GRAUS	X	SÉRIE DO SENO	FUNÇÃO SENO
30.	0.52359877571785	0.50000000010354	0.50000000010354
390.	6.80678408433208	0.50000000134598	0.50000000134597
750.	13.08996939294630	0.50000000258934	0.50000000258840
1110.	19.37315470156052	0.50000000391152	0.50000000383083
1470.	25.65634001017475	0.49999998255804	0.50000000507326
1830.	31.93952531878897	0.49956835566751	0.50000000631570
2190.	38.22271062740319	0.50588801083643	0.50000000755813
2550.	44.50589593601742	122.38835081627710	0.50000000880056
2910.	50.78908124463164	134057.81467031740000	0.50000001004299
-30.	-0.52359877571785	-0.50000000010354	-0.50000000010354
-390.	-6.80678408433208	-0.50000000134598	-0.50000000134597

TABELA 5.5 - Cálculo do seno pela série de Taylor, precisão dupla do sistema IBM /370 - 145.

GRAUS	X	SÉRIE DO SENO	FUNÇÃO SENO
30.	0.52359877526760	0.49999999971361	0.49999999971361
390.	6.80678403377533	0.49999995756254	0.49999995756254
750.	13.08996939659119	0.50000000574496	0.50000000574497
1110.	19.37315464019775	0.49999995063696	0.49999995068912
1470.	25.65633988380432	0.49999989541987	0.49999989563326
1830.	31.93952512741089	0.49999866626168	0.49999984057741
2190.	38.22271060943604	0.49930462572077	0.49999999199811
2550.	44.50589561462402	0.61402808406343	0.49999973046569
2910.	50.78908109664917	-71.40012177734988	0.49999988188641
-30.	-0.52359877526760	-0.49999999971361	-0.49999999971361
-390.	-6.80678403377533	-0.49993995756254	-0.49999995756254

TABELA 5.6 - Cálculo do seno pela série de Taylor, precisão dupla sistema DEC 10 (PDP10 - 90).

GRAUS	X	SÉRIE DO SENO	FUNÇÃO SENO
30.	0.52359879016876	0.50000001261839	0.50000001261839
390.	6.80678415298462	0.50000006080081	0.50000006080081
750.	13.08996963500977	0.50000021222153	0.50000021222150
1110.	19.37315368652344	0.49999912479699	0.49999912478268
1470.	25.65633964538574	0.49999968541077	0.49999968915669
1830.	31.93952560424805	0.50000006118364	0.50000025353049
2190.	38.22270965576172	0.49850392988821	0.49999916609170
2550.	44.50589370727539	1.20612797850728	0.49999807865212
2910.	50.78908157348633	-285.75783002459079	0.50000029483948
-30.	-0.52359879016876	-0.50000001261839	-0.50000001261839
-390.	-6.80678415298462	-0.50000006080081	-0.50000006080081

TABELA 5.7 - Cálculo do seno pela série de Taylor, precisão dupla do Burroughs 1700 - 26.

GRAUS	X	SÉRIE DO SENO	FUNÇÃO SENO
30.	0.52359877643175	0.50000000046566	0.50000000093132
390.	6.80678408965468	0.49999994423706	0.50000000372529
750.	13.08996940776705	0.49998914531897	0.50000000372529
1110.	19.37315472960472	0.50170545442961	0.50000001396983
1470.	25.65634003281593	-14.62316007912158	0.50000001396983
1830.	31.93952533602714	-987.41100692749023	0.50000001396983
2190.	38.22271066904067	1859212.83691406250731	0.50000001396983
2550.	44.50589597225189	**********	0.50000001396893
2910.	50.78908127546310	**********	0.50000001396983
-30.	-0.52359877643175	-0.50000000093132	-0.50000000093132
-390.	-6.80678408965468	-0.49999994516838	-0.50000000372529

TABELA 5.8 - Cálculo do seno pela série de Taylor, precisão dupla IBM 1130

5.4. - Cancelamento Subtrativo em um Cálculo de π

O número π, é definido como a relação da circunferência de um círculo para o seu diâmetro. Segundo Arquimedes, pode-se determinar limites inferior e superior para π, da seguinte forma:

$$\lim_{n\to\infty} \frac{p_i(n)}{2} \leq \pi \leq \lim_{n\to\infty} \frac{p_c(n)}{2}$$

com $p_i(n)$ = o perímetro do polígno de n lados inscrito num círculo cuja circunferência tem raio *1;*

$p_c(n)$ = o perímetro do polígno de n lados circunscrito a um círculo cuja circunferência tem raio *1*.

No Apêndice, apresentamos o algorítmo *A.2* para calcular os limites infeiror e superior de π pelo método descrito acima.

O algorítmo *A.2* foi implementado em *FORTRAN* e executado em precisão simples com $N_{max} = 15$. Os resultados estão nas tabelas 5.9 a 5.12.

Os resultados da execução do programa nos computadores escolhidos para estudos foram os seguintes:

(a) IBM/370-145 - nenhum resultado satisfatório, a melhor aproximação foi obtida para $n = 6$, com

$inf\pi = 3.136580$ $\quad sup\pi = 3.151756;$

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.765366	3.061465	3.313707
4	16.	0.390182	3.121452	3.182607
5	32.	0.196036	3.136580	3.151756
6	64.	0.098138	3.140430	3.144218
7	128.	0.049091	4.141829	3.142776
8	256.	0.024550	3.142450	3.142689
9	512.	0.012314	3.152380	3.152440
10	1024.	0.006176	3.162277	3.162293
11	2048.	0.003239	3.316625	3.316628
12	4096.	0.001691	3.464101	3.464103
13	8192.	0.000977	4.000000	4.000002
14	16384.	0.000977	8.000000	8.000004
15	32768.	0.000977	16.000000	16.000000

TABELA 5.9 - Limites de π - versão precisão simples do sistema IBM /370 - 145

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.765367	3.061467	3.313709
4	16.	0.390181	3.121445	3.182598
5	32.	0.196034	3.136548	3.151725
6	64.	0.098135	3.140331	3.144118
7	128.	0.049082	3.141276	3.142222
8	256.	0.024543	3.141519	3.141755
9	512.	0.012272	3.141519	3.141578
10	1024.	0.006135	3.141208	3.141223
11	2048.	0.003066	3.139964	3.139968
12	4096.	0.001530	3.132491	3.132492
13	8192.	0.000762	3.122499	3.122499
14	16384.	0.000366	3.000000	3.000000
15	32768.	0.000173	2.828427	2.828427

TABELA 5.10 - Limites de π - versão precisão simples do sistema DEC 10 (PDP10 - 90).

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.765367	3.061467	3.313708
4	16.	0.390181	3.121445	3.182597
5	32.	0.196034	3.136546	3.151723
6	64.	0.098135	3.140334	3.144121
7	128.	0.049081	3.141208	3.142154
8	256.	0.024543	3.141519	3.141755
9	512.	0.012270	3.141208	3.141267
10	1024.	0.006128	3.137475	3.137490
11	2048.	0.003069	3.142451	3.142455
12	4096.	0.001505	3.082207	3.082208
13	8192.	0.000691	2.828427	2.828427
14	16384.	0.000000	0.000000	0.000000
15	32768.	0.000000	0.000000	0.000000

TABELA 5.11 - Limites de π - versão precisão simples do Burroughs 1700-26.

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.765366	3.061467	3.313708
4	16.	0.390180	3.121445	3.182598
5	32.	0.196034	3.136551	3.151728
6	64.	0.098135	3.140343	3.124130
7	128.	0.049082	3.141299	3.142245
8	256.	0.024544	3.141757	3.141994
9	512.	0.012275	3.142509	3.142569
10	1024.	0.006147	3.147419	3.147435
11	2048.	0.003088	3.162586	3.162590
12	4096.	0.001544	3.163512	3.163513
13	8192.	0.000773	3.167214	3.167215
14	16384.	0.000490	4.015595	4.015595
15	32768.	0.000345	5.656854	5.656854

TABELA 5.12 - Limites de π - versão precisão simples do IBM 1130.

(b) *PDP10-90* - um resultado satisfatório para $n = 8$, com $inf\pi = 3.141519$ e $sup\pi = 3.141755$;

(c) *B1700-26* - resultado similar ao do *PDP10-90*;

(d) IBM 1130 - nenhum resultado satisfatório, a melhor aproximação foi obtida para $n = 7$, com $inf\pi = 3.141299$ e $sup\pi = 3.142245$.

Estes resultados mostraram claramente que o problema numérico está ligado com o número finito de dígitos, mas mesmo assim re-executamos o programa em precisão dupla com $N_{max} = 30$, cujos resultados podem ser vistos nas tabelas 5.13 a 5.16.

Sabendo-se que $\pi = 3.14159265359$, passemos ao estudo dos resultados obtidos. No IBM/370-145 obtivemos resultado significativo para $n = 15$ e outros razoáveis para $n = 12$, 13 e 14; no *PDP10-90* para $n = 17$ tivemos um bom resultado e para $n = 12$, 13, 14, 15 e 16 somente aproveitáveis; no *B1700-26* para $n = 16$ bom e $n = 12$, 13, 14 e 15 aceitáveis e finalmente no IBM 1130, apenas aceitável para $n = 9$.

Os resultados foram como esperávamos, ou seja, mais uma vez o *PDP10-90* deu melhor aproximação por ter mais bits na mastissa, seguido pelo *B1700-16*, IBM/370-145 e finalmente IBM 1130.

Houve divergência em todas as máquinas, pois o problema numérico apresentado é o cancelamento subtrativo que ocorre no cálculo do comprimento do lado (S) do polígono e no cálculo do limite superior de π, porque $4 - S^2$ quando n cresce, S^2 tende a zero e a expressão $4 - S^2$ tende a 4. Assim quando fazemos

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.76536686473018	3.06146745892072	3.31370849898476
4	16.	0.39018064403226	3.12144515225805	3.18259787807453
5	32.	0.19603428065912	3.13654849054594	3.15172490742926
6	64.	0.09813534865484	3.14033115695474	3.14411838524589
7	128.	0.04908245704582	3.14127725093276	3.14222362994244
8	256.	0.02454307657144	3.14151380114415	3.14175036916881
9	512.	0.01227176929831	3.14157294036788	3.14163208070397
10	1024.	0.00613591352594	3.14158772527996	3.14160251025961
11	2048.	0.00306796037256	3.14159142150464	3.14159511774302
12	4096.	0.00153398063751	3.14159234561108	3.14159326967027
13	8192.	0.00076699037513	3.14159257654500	3.14159280755978
14	16384.	0.00038349519451	3.14159263346325	3.14159269121694
15	32768.	0.00019174759798	3.14159264532122	3.14159265975964
16	65536.	0.00009587379899	3.14159264532122	3.14159264893082
17	131072.	0.00004793689892	3.14159260737572	3.14159260827812
18	262144.	0.00002396845177	3.14159291093967	3.14159291116527
19	524288.	0.00001198422126	3.14159169668369	3.14159169674008
20	1048576.	0.00000599210136	3.14158683965504	3.14158683969614
21	2097152.	0.00000299605995	3.14159655370482	3.14159655370834
22	4194304.	0.00000149799292	3.14151884046555	3.14151884046643
23	8388608.	0.00000074892234	3.14120796828227	3.14120796828249
24	16777216.	0.00000037431290	3.13996417177012	3.13996417177017
25	33554432.	0.00000018730469	3.14245127249413	3.14245127249415
26	67108864.	0.00000009305772	3.12249899919920	3.12249899919920
27	134217728.	0.00000004712161	3.16227766016838	3.16227766016838
28	268435456.	0.00000002107342	2.82842712474619	2.82842712474619
29	536870912.	-0.00000000000000	0.00000000000000	0.00000000000000
30	1073741824.	-0.00000000000000	0.00000000000000	0.00000000000000

TABELA 5.13 - Limites de π - versão precisão dupla do sistema IBM / 370 - 145.

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.76536686473018	3.06146745892072	3.31370849898476
4	16.	0.39018064403226	3.12144515225805	3.18249787807453
5	32.	0.19603428065912	3.13654849054594	3.15172490742926
6	64.	0.09813534865484	3.14033115695475	3.14411838524590
7	128.	0.04908245704582	3.14127725093277	3.14222362994246
8	256.	0.02454307657144	3.14151380114430	3.14175036916897
9	512.	0.01227176929831	3.14157294036709	3.14163208070318
10	1024.	0.00613591352593	3.14158772527716	3.14160251025681
11	2048.	0.00306796037257	3.14159142151122	3.14159511774961
12	4096.	0.00153398063749	3.14159234557026	3.14159326962945
13	9192.	0.00076699037514	3.14159257658438	3.14159280759915
14	16384.	0.00038349519462	3.14159263433870	3.14159269209239
15	32768.	0.00019174759819	3.14159264876744	3.14159266320586
16	65536.	0.00009587379921	3.14159265236188	3.14159265597149
17	131076.	0.00004793689962	3.14159265332534	3.14159265422774
18	262144.	0.00002396844981	3.14159265362179	3.14159265384739
19	524288.	0.00001198422491	3.14159265480759	3.14159265486399
20	1048576.	0.00000599211244	3.14159264532122	3.14159264533532
21	2097152.	0.00000299605618	3.14159260737572	3.14159260737924
22	4194304.	0.00000149802809	3.14159260737572	3.14159260737660
23	8388608.	0.00000074901412	3.14159291093967	3.14159291093989
24	16777216.	0.00000037450691	3.14159169668369	3.14159169668374
25	33554432.	0.00000018725259	3.14157712557523	3.14157712557524
26	67108864.	0.00000009362456	3.14151884046555	3.14151884046555
27	134217728.	0.00000004681228	3.14151884046555	3.14151884046555
28	268435456.	0.00000002340382	3.14120796828227	3.14150796828227
29	536870912.	0.00000001170654	3.14245127249313	3.14245127249413
30	1073741824.	0.00000000585327	3.14245127249413	3.14245127249413

TABELA 5.14 - Limites de π - versão de precisão dupla do sistema DEC 10 (PDP10 - 90).

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.76536686473018	3.06146745892072	3.31370849898476
4	16.	0.39018064403226	3.12144515225805	3.18259787807453
5	32.	0.19603428065912	3.13654849054594	3.15172490742926
6	64.	0.09813534865484	3.14033115695475	3.14411838524590
7	128.	0.04908245704582	3.14127725093277	3.14222362994246
8	256.	0.02454307657144	3.14151380114430	3.14175036916896
9	512.	0.01227176929831	3.14157294036709	3.14163208070318
10	1024.	0.00613591352593	3.14158772527714	3.14160251025679
11	2048.	0.00306796037257	3.14159142151100	3.14159511774939
12	4096.	0.00153398063749	3.14159234556939	3.14159326962858
13	8192.	0.00076699037514	3.14159257658206	3.14159280759683
14	16384.	0.00038349519462	3.14159263433407	3.14159269208776
15	32768.	0.00019174759819	3.14159264873038	3.14159266316880
16	65536.	0.00009587379920	3.14159265213955	3.14159265574915
17	131072.	0.00004793689962	3.14159265362179	3.14159265452419
18	262144.	0.00002396844982	3.14159265480759	3.14159265503319
19	524288.	0.00001198422487	3.14159264532122	3.14159264537762
20	1048576.	0.00000599211236	3.14159260737572	3.14159260738982
21	2097152.	0.00000299605618	3.14159260737572	3.14159260737924
22	4194304.	0.00000149802766	3.14159169668369	3.14159169668457
23	8388608.	0.00000074901383	3.14159169668369	3.14159169668391
24	16777216.	0.00000037450518	3.14157712557523	3.14157712557528
25	33554432.	0.00000018725375	3.14159655370482	3.14159655370483
26	67108864.	0.00000009362456	3.14151884046555	3.14151884046555
27	134217728.	0.00000004680765	3.14120796828227	3.14120796828227
28	268435456.	0.00000002341309	3.14245127249413	3.14245127249413
29	536870912.	0.00000001170654	3.14245127249413	3.14245127249413
30	1073741824.	0.00000000574106	3.08220700148449	3.08220700148449

TABELA 5.15 - Limites de π - versão de precisão dupla do Burroughs 1700 - 26.

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.76536686532199	3.06146746315062	3.31370850559324
4	16.	0.39018064539413	3.12144516315311	3.18259788956493
5	32.	0.19603428523987	3.13654856570065	3.15172498393803
6	64.	0.09813535469584	3.14033135212957	3.14411858096718
7	128.	0.04908245611295	3.14127719309180	3.14222357049584
8	256.	0.02454307802690	3.14151398744434	3.14175055362284
9	512.	0.01227178565386	3.14157712738960	3.14163626823574
10	1024.	0.00613593077468	3.14159655850380	3.14161134138703
11	2048.	0.00306788949546	3.14151884522289	3.14152254071086
12	4096.	0.00153379295488	3.14120797347277	3.14120889734476
13	8192.	0.00076720001811	3.14245127420872	3.14245150517672
14	16384.	0.00038360000905	3.14245127420872	3.14245133195072
15	32768.	0.00019301011121	3.16227766405791	3.16227767895907
16	65536.	0.00009650505560	3.16227766405791	3.16227766778320
17	131072.	0.00004315837291	2.82842712756246	2.82842712756246
18	262144.	0.00000000000000	0.00000000000000	0.00000000000000
19	524288.	0.00000000000000	0.00000000000000	0.00000000000000
20	1048576.	0.00000000000000	0.00000000000000	0.00000000000000
21	2097152.	0.00000000000000	0.00000000000000	0.00000000000000
22	4194304.	0.00000000000000	0.00000000000000	0.00000000000000
23	8388608.	0.00000000000000	0.00000000000000	0.00000000000000
24	16777216.	0.00000000000000	0.00000000000000	0.00000000000000
25	33554432.	0.00000000000000	0.00000000000000	0.00000000000000
26	67108864.	0.00000000000000	0.00000000000000	0.00000000000000
27	134217728.	0.00000000000000	0.00000000000000	0.00000000000000
28	268435456.	0.00000000000000	0.00000000000000	0.00000000000000
29	536870912.	0.00000000000000	0.00000000000000	0.00000000000000
30	1073741725.	0.00000000000000	0.00000000000000	0.00000000000000

TABELA 5.16 - Limites de π - versão precisão dupla do IBM 1130.

$S = \sqrt{2 - \sqrt{4 - S^2}}\ \sqrt{4 - S^2}$ tende a 2, logo estaremos subtraindo números positivos cujas grandezas são aproximadamente iguais.

Como se pode calcular os limites de π, com mais exatidão, utilizando o mesmo método? A solução é evitar as operações que causam o cancelamento subtrativo, que no caso, é a operação $\sqrt{4 - S^2}$.

Podemos re-escrever a operação utilizando o teorema do binômio. Primeiro faz-se $n = 1/2$, $a = 4$ e $b = S^2$ em $(a + b)^n$, então obtemos,

$$\sqrt{4 - S^2} = 2 - d_1 - d_2 - \ldots - d_k - \ldots$$

onde

$$d_{k+1} = \frac{2k - 1}{k + 1} \cdot \frac{S^2}{2^3}\ d_k$$

sabendo-se que $d_1 = \frac{S^2}{4}$, temos

$$\sqrt{2 - \sqrt{4 - S^2}} = d_1 + d_2 + \ldots + d_k + \ldots$$

Fizemos esta modificação no programa *FORTRAN* e re-executamos em precisão simples (tabelas 5.17 a 5.20) e em precisão dupla (tabelas 5.21 a 5.24). Note que agora os resultados são bem estáveis e com maior precisão.

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.765366	3.061465	4.959311
4	16.	0.390180	3.121440	3.878000
5	32.	0.196034	3.136543	3.477386
6	64.	0.098135	3.140326	3.302365
7	128.	0.049082	3.141271	3.220302
8	256.	0.024543	3.141507	3.180538
9	512.	0.012272	3.141566	3.160961
10	1024.	0.006136	3.141581	3.151248
11	2048.	0.003068	3.141584	3.146411
12	4096.	0.001534	3.141584	3.143996
13	8192.	0.000767	3.141584	3.142790
14	16384.	0.000383	3.141584	3.142188
15	32768.	0.000192	3.141584	3.141886

TABELA 5.17 - Limites de π - versão Teorema do Binômio precisão simples do sistema IBM/370-145

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE	LIMITE SUPERIOR DE
3	8.	0.765367	3.061467	4.959315
4	16.	0.390181	3.121445	3.878007
5	32.	0.196034	3.136548	3.477393
6	64.	0.098135	3.140331	3.302371
7	128.	0.049082	3.141277	3.220307
8	256.	0.024543	3.141514	3.180544
9	512.	0.012272	3.141573	3.160968
10	1024.	0.006136	3.141588	3.151256
11	2048.	0.003068	3.141591	3.146418
12	4096.	0.001534	3.141592	3.144004
13	8192.	0.000767	3.141593	3.142798
14	16384.	0.000383	3.141593	3.142195
15	32768.	0.000192	3.141593	3.141894

TABELA 5.18 - Limite de π - versão Teorema do Binômio precisão simples do sistema DEC 10 (PDP10-90).

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.765367	3.061467	4.959315
4	16.	0.390181	3.121445	3.878006
5	32.	0.196034	3.136548	3.477392
6	64.	0.098135	3.140331	3.302370
7	128.	0.049082	3.141277	3.220307
8	256.	0.024543	3.141513	3.180543
9	512.	0.012272	3.141572	3.160968
10	1024.	0.006136	3.141587	3.151255
11	2048.	0.003068	3.141591	3.146417
12	4096.	0.001534	3.141592	3.144003
13	8192.	0.000767	3.141592	4.142797
14	16384.	0.000383	3.141592	3.142194
15	32768.	0.000192	3.141592	3.141893

TABELA 5.19 - Limites de π - versão Teorema do Binômio precisão simples de Burroughs 1700-26.

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.765366	3.061464	4.959307
4	16.	0.390180	3.121440	3.877999
5	32.	0.196033	3.136543	3.477386
6	64.	0.098135	3.140326	3.302364
7	128.	0.049082	3.141272	3.220302
8	256.	0.024543	3.141508	3.180537
9	512.	0.012271	3.141567	3.160962
10	1024.	0.006135	3.141581	3.151249
11	2048.	0.003067	3.141585	3.146411
12	4096.	0.001533	3.141586	3.143997
13	8192.	0.000766	3.141586	3.142791
14	16384.	0.000383	3.141586	3.142188
15	32768.	0.000191	3.141586	3.141887

TABELA 5.20 - Limites de π - versão Teorema do Binômio precisão simples do IBM 1130.

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.76536686473018	3.06146745892072	4.95931523537420
4	16.	0.39018064403226	3.12144515225805	3.87800673496262
5	32.	0.19603428065912	3.13654849054594	3.47739256563251
6	64.	0.09813534865484	3.14033115695475	3.30237081249040
7	128.	0.04908245704582	3.14127725093277	3.22030755454287
8	256.	0.02454307657144	3.14151380114430	3.18054396821972
9	512.	0.01227176929831	3.14157294036709	3.16096827709498
10	1024.	0.00613591352593	3.14158772527716	3.15125564133382
11	2048.	0.00306796037257	3.14159142151120	3.14641796432624
12	4096.	0.00153398063749	3.14159234557011	3.14400376602074
13	8192.	0.00076699037514	3.14159257658487	3.14279782442605
14	16384.	0.00038349519462	3.14159263433856	3.14219514270745
15	32768.	0.00019174759819	3.14159264877698	3.14189387407905
16	65536.	0.00009587379921	3.14159265238659	3.14174325781772
17	131072.	0.00004793689962	3.14159265328899	3.14166795419962
18	262144.	0.00002396844981	3.14159265350459	3.14163030351871
19	524288.	0.00001198422491	3.14159265357099	3.14161147846025
20	1048576.	0.00000599211245	3.14159265358509	3.14160206600152
21	2097152.	0.00000299605623	3.14159265358861	3.14159735978978
22	4194304.	0.00000149802811	3.14159265358949	3.14159500668831
23	8388608.	0.00000074901406	3.14159265358971	3.14159383013868
24	16777216.	0.00000037450703	3.14159265358977	3.14159324186414
25	33554432.	0.00000018725351	3.14159265358978	3.14159294772694
26	67108864.	0.00000009362676	3.14159265358979	3.14159280065836
27	134217728.	0.00000004681338	3.14159265358979	3.14159272712407
28	268435456.	0.00000002340669	3.14159265358979	3.14159269035693
29	536870912.	0.00000001170334	3.14159265358979	3.14159267197336
30	1073741824.	0.00000000585167	3.14159265358979	3.14159266278157

TABELA 5.21 - Limites de π - versão Teorema do Binômio precisão dupla do sistema IBM/370-145.

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.76536686473018	3.06146745892072	4.95931523537420
4	16.	0.39018064403226	3.12144515225805	3.87800673496262
5	32.	0.19603428065912	3.13654849054594	3.47739256553251
6	64.	0.09813534865484	3.14033115695475	3.30237081249040
7	128.	0.04908245704582	3.14127725093277	3.22030755454287
8	256.	0.02454307657144	3.14151380114430	3.18054396821973
9	512.	0.01227176929831	3.14157294036709	3.16096827709498
10	1024.	0.00613591352593	3.14158772527716	3.15125564133382
11	2048.	0.00306796037257	3.14159142151120	3.14641796432625
12	4096.	0.00153398063749	3.14159234557012	3.14400376602075
13	8192.	0.00076699037514	3.14159257658487	3.14279782442605
14	16384.	0.00038349519462	3.14159263433856	3.14219514270746
15	32768.	0.00019174759819	3.14159264877699	3.1418938740790[illegible]
16	65536.	0.00009587379921	3.14159265238659	3.14174325781772
17	131072.	0.00004793689962	3.14159265328899	3.14166795419967
18	262144.	0.00002396844981	3.14159265351459	3.14163030351872
19	524288.	0.00001198422491	3.14159265357099	3.14161147846025
20	1048576.	0.00000599211245	3.14159265358509	3.14160206600152
21	2097152.	0.00000299605623	3.14159265358862	3.14159735978978
22	4194304.	0.00000149802811	3.14159265358950	3.14159500668832
23	8388608.	0.00000074901406	3.14159265358972	3.14159383013869
24	16777216.	0.00000037450703	3.14159265358977	3.14159324186415
25	33554432.	0.00000018725351	3.14159265358979	3.14159294772695
26	67108864.	0.00000009362676	3.14159265358979	3.14159280065837
27	134217728.	0.00000004681338	3.14159265358979	3.14159272712408
28	268435456.	0.000000002340669	3.14159265358979	3.14159269035694
29	536870912.	0.000000001170334	3.14159265358979	3.14159267197336
30	1073741824.	0.000000000585167	3.14159265358979	3.14159266278158

TABELA 5.22 - Limites de π - versão Teorema do Binômio precisão dupla do sistema DEC 10 (PDP 10-90).

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.76536686473018	3.06146745892072	4.95931523537420
4	16.	0.39018064403226	3.12144515225805	3.87800673496262
5	32.	0.19603428065912	3.13654849054594	3.47739256563251
6	64.	0.09813534865484	3.14033115695475	3.30237081249040
7	128.	0.04908245704582	3.14127725093277	3.22030755454287
8	256.	0.02454307657144	3.14151380114430	3.18054396821973
9	512.	0.01227176929831	3.14157294036709	3.16096827709498
10	1024.	0.00613591352593	3.14158772527716	3.15125564133382
11	2048.	0.00306796037257	3.14159142151120	3.14641796432625
12	4096.	0.00153398063749	3.14159234557012	3.14400376602075
13	8192.	0.00076699037514	3.14159257658487	3.14279782442605
14	16384.	0.00038349519462	3.14159263433856	3.14219514270746
15	32768.	0.00019174759819	3.14159264877699	3.14189387407905
16	65536.	0.00009587379921	3.14159265238659	3.14174325781772
17	131072.	0.00004793689962	3.14159265328899	3.14166795419967
18	262144.	0.00002396844981	3.14159265351459	3.14163030351872
19	524288.	0.00001198422491	3.14159265357099	3.14161147846025
20	1048576.	0.00000599211245	3.14159265358509	3.14160206600152
21	2097152.	0.00000299605623	3.14159265358862	3.14159735978978
22	4194304.	0.00000149802811	3.14159265358950	3.14159500668832
23	8388608.	0.00000074901406	3.14159265358972	3.14159383013869
24	16777216.	0.00000037450703	3.14159265358977	3.14159324186415
25	33554432.	0.00000018725351	3.14159265358979	3.14159294772695
26	67108864.	0.00000009362676	3.14159265358979	3.14159280065837
27	134217728.	0.00000004681338	3.14159265358979	3.14159272712408
28	268435456.	0.00000002340669	3.14159265358979	3.14159269035694
29	536870912.	0.00000001170334	3.14159265358979	3.14159267197336
30	1073741824.	0.00000000585167	3.14159265358979	3.14159266278158

TABELA 5.23 - Limites de π - versão Teorema do Binômio precisão dupla do Burroughs 1700-26.

N	LADOS	COMPRIMENTO DO LADO	LIMITE INFERIOR DE π	LIMITE SUPERIOR DE π
3	8.	0.76536686299368	3.06146745197474	4.95931521244347
4	16.	0.39018064213450	3.12144513707607	3.87800670973956
5	32.	0.19603427953552	3.13654847256839	3.47739254403859
6	64.	0.09813534800196	3.14033113606274	3.30237078666687
7	128.	0.04908245675323	3.14127723407000	3.22030753549188
8	256.	0.02454307641164	3.14151378255337	3.18054394610226
9	512.	0.01227176921747	3.14157292153686	3.16096825525164
10	1024.	0.00613591348701	3.14158770721405	3.15125562064349
11	2048.	0.00306796035238	3.14159140270203	3.14641794282943
12	4096.	0.00153398062820	3.14159232657402	3.14400374516844
13	8192.	0.00076699037049	3.14159255754202	3.14279780257493
14	16384.	0.00038349519229	3.14159261714667	3.14219512511044
15	32768.	0.00019174759694	3.14159262832254	3.14189385157078
16	65536.	0.00009587379864	3.14159263577312	3.14174323715269
17	131072.	0.00004793689932	3.14159263577312	3.14166793227195
18	262144.	0.00002396844966	3.14159263577312	3.14163028262555
19	524288.	0.00001198422483	3.14159263577312	3.14161145687103
20	1048576.	0.00000599211241	3.14159263577312	3.14160204492509
21	2097152.	0.00000299605620	3.14159263577312	3.14159734081476
22	4194304.	0.00000149802810	3.14159263577312	3.14159498829394
23	8388608.	0.00000074901405	3.14159263577312	3.14159381110221
24	16777216.	0.00000037450702	3.14159263577312	3.14159322436898
25	33554432.	0.00000018725351	3.14159263577312	3.14159292820841
26	67108864.	0.00000009362675	3.14159263577312	3.14159278292208
27	134217728.	0.00000004681337	3.14159263577312	3.14159270841628
28	268435456.	0.00000002340668	3.14159263577312	3.14159267116338
29	536870912.	0.00000001170334	3.14159263577312	3.14159265253692
30	1073741825.	0.00000000585167	3.14159263577312	3.14159264508634

TABELA 5.24 - Limites de π - versão Teorema do Binômio precisão dupla do IBM 1130.

5.5 - Conclusão

Apresentamos apenas dois casos estudados, apesar de outros programas terem sidos processados. A razão de restringirmos a dois casos, foi porque atingimos o objetivo de análise das aritméticas das máquinas e ainda mostramos algumas causas porque os algorítmos numéricos podem apresentar alta instabilidade numérica.

Em primeiro lugar, sugerimos algumas regras para se obter resultados melhores em computações numéricas, tais como:

(a) Quando os números devem ser somados e/ou subtraídos, deve-se trabalhar com os números menores primeiramente;

(b) Se possível, evitar subtrações de dois números aproximadamente iguais;

(c) Se existir uma expressão como $a(b - c)$ poderá ser re-escrita como $ab - ac$ o mesmo acontecendo com $(a - b)/c$ por $a/c - b/c$. Se envolver números aproximadamente iguais, efetuar primeiro a subtração;

(d) Evitar operações aritméticas com operandos cujas grandezas sejam suficientemente diferentes;

(e) Evitar operações aritméticas com operandos cujos expoentes podem causar *overflow* ou *underflow*. Quando for

inevitável, a primeira medida é verificar se as operações não podem ser re-arranjadas, se não for possível, procurar armazenar os expoentes separadamente e operá-los como se fossem inteiros;

(f) Evitar divisão onde o divisor é uma quantidade muito pequena;

(g) Evitar impressão de resultados imprecisos ou duvidosos, especificando formatos suficientemente grandes, permitindo a impressão, pelo menos, do número de dígitos mantidos nas mantissas dos números;

(h) Evitar testes de igualdade, dando sempre um fator de precisão;

(i) Especificar corretamente as constantes, identificando-as se são de precisão simples ou dupla para evitar erros de conversão (erros inerentes);

(j) Evitar expressões aritméticas mistas;

(k) Um programa em precisão simples quando for re-executado em precisão dupla deverá ter suas constantes alteradas para o número de dígitos aceitos pela precisão dupla, o mesmo acontecendo com os dados para evitar erros inerentes;

(l) Quando nada do exposto acima se aplicar, sugerimos minimizar o número de operações aritméticas.

As regras acima podem ser facilmente comprovadas pelo que foi apresentado nos capítulos anteriores, razão pela qual não estenderemos a dissertação com trechos de programas *FORTRAN* para ilustrar e justificar as regras.

Fazendo uma análise da aritmética de cada um dos computadores, sem levar em consideração o tempo de processamento, custo de processamento e magnitude do expoente, considerando apenas os casos estudados e a estrutura dos computadores utilizados, chegamos a conclusão de que as melhores precisões, simples e dupla, para a maioria dos casos, possívelmente, obedecerão a seguinte ordem: PDP10-90, B1700-26, IBM/370-145 e IBM 1130.

Comparando como as operações aritméticas são executadas pelas máquinas, como o processo de corte ou arredondamento, o processo de normalização e ainda observando o tamanho da palavra e a forma de armazenamento, é quase que suficiente para se fazer uma análise com a finalidade de escolha de uma aritmética mais conveniente para aplicações que envolvem números de ponto flutuante.

Para um estudo mais aprofundado dos problemas inerentes a uma máquina é necessário estudar os seguintes aspectos:

(a) Como são feitas as conversões de base dos dados de entrada e saída;

(b) Como são avaliadas as funções embutidas;

(c) Qual o comportamento do sistema quando ocorre *overflow* ou *underflow* de expoente;

(d) Como são feitas as operações aritméticas (corte ou arredondamento, normalização, algorítmos para execução de operações, tamanho da mantissa, do expoente,.);

(e) Estrutura de funcionamento da unidade aritmética de ponto flutuante, se a unidade é constituida por *hardware* ou *software*;

(f) Até que ponto o usuário poderá interferir no fluxo de processamento quando condições anormais ocorrerem, e a que nível.

Dos computadores que estudamos, considerando alguns dos itens acima, podemos afirmar que, provavelmente, o mais indicado para processamento de números de ponto flutuante em condições normais, sem problema de *overflow* ou *underflow*, com o problema bem condicionado, sem problemas de instabilidade, é o PDP-10-90.

Finalmente, observamos que na maioria dos casos práticos, apenas o aumento de precisão dos números de ponto flutuante não resolve, é necessário uma análise detalhada do algorítmo utilizado e principalmente dos dados para evitar erros inerentes. Isto significa que é possível obter bons resultados em qualquer uma das máquinas estudadas, desde que, o algorítmo não apresente instabilidade numérica e o problema esteja bem condicionado.

CAPÍTULO VI

LINGUAGEM PARA ANÁLISE DE ERROS COMPUTACIONAIS

6.1 - Introdução

Qualquer usuário que não tenha noções sobre a arquitetura do computador, normalmente, acredita nos resultados obtidos como se fossem corretos, ou pelo menos, aproximadamente corretos. Quando o usuário é um iniciante na área de análise numérica, o seu maior problema é entender a teoria dos erros, pois é difícil acreditar que uma máquina não possa manipular com números reais e sim com um subconjunto discreto desses números. Difícil ainda é aceitar que o computador, na execução de um algorítmo numérico simples, possa apresentar soluções totalmente desprovidas de significância.

Como ensinar a teoria dos erros? Mesmo o melhor método de

ensino sem o auxílio de uma ferramenta eficaz não obteria resultados satisfatórios.

Considerando a falta de recursos para auxíliar a aprendizagem do aluno, passamos a descrever uma linguagem interativa que possa, não só motivar, mas mostrar aos iniciantes, como as operações aritméticas são executadas interneamente, afim de que o aluno consiga avaliar os erros cometidos por uma máquina.

6.2 - Elementos Sintáticos da Linguagem

Apresentaremos os elementos sintáticos que compõem a linguagem.

6.2.1 - Conjunto de Caracteres

A linguagem identificará como caracter todas as letras do alfabeto *(A - Z)*, todos os dígitos decimais *(0 - 9)* e os seguintes caracteres especiais:

*+, -, *, /, =, (,), ., >, <, , ",", ";"*

6.2.2 - Identificadores

Um identificador é uma cadeia de caracteres alfanuméricos

começando com uma letra. A linguagem aceitará identificadores de até 10 caracteres, de comprimento.

6.2.3 - Símbolos Operadores

Os operadores binários serão os seguintes:

(a) Aritméticos

SÍMBOLOS	SIGNIFICADOS
+	adição
-	subtração
*	multiplicação
/	divisão
* *	exponenciação

(b) Lógicos

SÍMBOLOS	SIGNIFICADOS
=	igual
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual
¬=	não igual

(c) Unários

SÍMBOLOS	SIGNIFICADOS
+	mais
-	menos (hífem)
¬	negação

6.2.4 - Palavras Reservadas

São identificadores que não podem receber atributos no programa. A linguagem contém as seguintes palavras reservadas:

INICIO	*DECLARE*	*APAGUE*	*RAIZQUADR*
FIM	*REAL*	*DETALHE*	*VALORABS*
ARITPF	*INTEIRO*	*EXECUTE*	*SENO*
ARIFIX	*VETOR*	*ATE*	*COS*
EXPOENTE	*MATRIZ*	*FIM*	*EXP*
PRECISAO	*BLOCO*	***E***	*POTENCIA*
NUMDIG	*FIMBLOCO*	*OU*	*LOG*
MOD			

6.2.5 - Comentários

Os comentários terão o seguinte formato:

COM .... cadeia de caracteres ....;

serão tratados como instrução sem efeito, podendo aparecer em qualquer parte do programa.

6.2.6 - Brancos

O carater branco terá a função de delimitador além de ser

vir para dar estética visual aos programas.

6.2.7 - Delimitadores e Parênteses

Os delimitadores de programa são *INICIO* e *FIM*, indicando o início e fim respectivamente.

O delimitador de instrução é ";", para marcar o seu fim.

Os delimitadores de blocos são: No início a palavra reservada *BLOCO* e no fim a palavra reservada *FIMBLOCO*.

Os parênteses têm a finalidade de eliminar ambiguidades de sintaxe, como por exemplo:

DECLARE (A, B, C, D) REAL;

e ainda ditar a precedência de operações.

6.2.8 - Formato dos Campos

As instruções terão formato livre de entrada, podendo iniciar em qualquer posição de uma linha da tela do terminal de vídeo, e terminar em uma outra linha qualquer.

6.2.9 - Expressões

A linguagem é composta de dois tipos de expressões: (a) aritmética e (b) lógica.

A expressão aritmética é constituida de um operando, precedido ou não, de um operador unário aritmético, e opcionalmente seguido por um operador aritmético e um segundo operando. O operando poderá ser uma variável, uma constante numérica, ou referência de função.

Uma expressão lógica, poderá ser simples ou composta. A simples será constituida por duas expressões aritméticas quaisquer, relacionadas por um dos operandos lógicos e podendo, o conjunto, ser precedido ou não por um operador unário lógico. A composta será formada por mais de uma expressão lógica simples, relacionadas por um dos operadores *Booleanos ("E"* ou *"OU")* e o conjunto, poderá ser precedido ou não por um operador unário lógico.

6.2.10 - Instruções

A linguagem admitirá os seguintes tipos de instruções:

(a) instrução de declaração,
(b) instrução de atribuição,
(c) instrução de definição de bloco,
(d) instrução *EXECUTE*,
(e) outras instruções.

6.2.10(a) - Instrução de Declaração

Temos duas classes distintas: (1) declaração da aritmética e (2) declaração de variáveis.

A primeira é utilizada para definir a magnitude e a precisão dos números de ponto flutuante através da declaração

ARITPF EXPOENTE = inteiro PRECISÃO = nº de dígitos,

e a magnitude dos números de ponto fixo por meio da declaração

ARITFIX NUMDIG = nº de dígitos.

Quando qualquer uma das duas opções da declaração da aritmética aparecerem no corpo do programa, a ação a ser tomada deixamos para a etapa de implementação da linguagem.

A segunda, poderá estar presente em qualquer ponto do corpo do programa, exceto dentro de blocos.

Há dois tipos básicos de variáveis: *simples* e *indexada*.

O formato da instrução de declaração de variáveis é o seguinte:

- variáveis simples

DECLARE (lista de variáveis) tipo;

- variáveis indexadas

DECLARE (lista de variáveis com atributos) tipos;

onde: tipo deve ser inteiro ou real, e os atributos devem indicar o número de elementos de um vetor ou o número de linhas e colunas para uma matriz.

6.2.10(b) - Instrução de Atribuição

O formato geral será:

Variável = expressão aritmética

onde a variável, previamente declarada, poderá ser simples ou indexada.

6.2.10(c) - Instrução de Definição de Bloco

Um bloco será constituído por dois delimitadores e um corpo. Os delimitadores serão as palavras reservadas *BLOCO* seguido de um nome, no início, e *FIMBLOCO* acompanhado opcionalmente, de um nome, no fim. O corpo conterá instruções, com excessão de instruções de declarações.

Por Exemplo:

```
BLOCO  A;

    X = C + D;
    Y = X * X;

FIMBLOCO  A;
```

6.2.10(d) - Instrução Execute

Servirá para executar um bloco previamente definido e, a presenta o seguinte formato:

EXECUTE "nome bloco" [*ATE condição*]

O bloco identificado por *nome bloco* será executado, exceto quando for assinalado *"ATE condição"* e a condição for satisfeita.

6.2.10(e) - Outras Instruções

Incluímos aqui, algumas instruções de controle, que são:

- *DETALHE*
- *APAGUE "identificadores"*

A primeira servirá para detalhar a próxima instrução, exibindo na tela do terminal de vídeo como as operações aritméticas estão sendo executadas internamente. A sua utilização só terá efei

to antes de uma instrução de atribuição que não esteja dentro de um bloco.

A segunda terá o seguinte formato:

APAGUE (lista de identificadores);

onde os identificadores poderão ser variáveis e nome de blocos. A função dessa instrução é tornar disponível o espaço ocupado pela lista de identificadores.

6.2.11 - Exemplo de Programa para Determinar a Solução de uma Equação do 2º Grau.

```
INICIO
     ARITPF EXPOENTE = 99  PRECISÃO = 10;
     ARITFIX  NUNDIG=10;
     DECLARE(A,B,C, DELTA, X1,X2, TEMP1)REAL;
     COM ... ENTRADA DE A,B e C;
     A = 2.EO;
     B = 4.OEO;
     C = -4.0;
     DETALHE
     TEMP1 = B*B;
     *- - - - - -
     DECLARE (TEMP2)REAL;
     DETALHE;
     TEMP2 = 4.*A;
     *- - - - - -
     DETALHE;
     TEMP2= TEMP2 *C;
     *- - - - - -
     DELTA   = TEMP1 - TEMP2;
     *- - - - - -
     DECLARE (AUX1)REAL;
     COM ... CALCULO DA PRIMEIRA RAIZ
     BLOCO RAIZ1;
         AUX1 = RAIZQUAD(DELTA);
         TEMP1 = -B + AUX1
         TEMP2 = 2.0 *A;
         X1 = TEMP1/TEMP2;
     FIMBLOCO RAIZ1;
     EXECUTE RAIZ1;
     TEMP1 = -B - AUX1;
     TEMP2 = 2. *A;
     X2 = TEMP1/TEMP2;
FIM.
```

---

* - - - - - - - onde serão impressos os detalhes da operação aritmética.

6.3 - Definição Formal da Linguagem

Definiremos a sintaxe da linguagem interativa, utilizando a notação *BNF (Backus - Naur Form)*

6.3.1 - Gramática da Linguagem de Programação

<programa> :: = INICIO <lista de instruções> FIM

<lista de instruções> :: = <instrução>

| <lista de instruções>; <instrução>

<instrução> :: = <declaração>

| <atribuição>

| <definição de bloco>

| <execute>

| <outros>

<declaração> :: = <declaração da aritmética>

| <declaração de variáveis>

<declaração da aritmética> :: = <aritmética de ponto fixo>

| <aritmética de ponto flutuante>

<aritmética de ponto fixo> :: = ARITFIX NUMDIG = <constante inteira>

<constante inteira> :: = <números inteiros>

<número inteiro> :: = <dígito decimal>

| <número inteiro> <dígito decimal>

<dígito decimal> :: = 0|1|2|3|4|5|6|7|8|9

<aritmética de ponto flutuante> :: = ARITPF EXPOENTE = <constante inteira>

PRECISÃO = <constante inteira>

<declaração de variável> :: = DECLARE (<lista de identificadores>)<tipo>

| DECLARE (<lista de identificadores com

atributos>) <tipo>

<tipo> : = INTEIRO

| REAL

<lista de identificadores> :: = <identificador>

| <lista de identificadores> , <identificador>

<identificador> :: = <letra>

| <identificador> <letra>

| <identificador> <dígito decimal>

<lista de identificadores com atributos> :: = <identificador com atributos>

| <lista de identificadores com atributos> , <identificador com atributos>

<identificador com atributos> :: = <identificador> (<dimensão>)

<dimensão> :: = <constante inteira>

| <constante inteira> , <constante inteira>

<atribuição> :: = <variável> = <expressão aritmética>

<variável> :: = <variável simples>

| <variável indexada>

<variável simples> :: = <identificador>

<variável indexada> :: = <identificador> (<índice>)

| <identificador> (<índice> , <índice>)

<índice> :: = <constante inteira>

| <variável simples>

< *expressão aritmética* > :: = < *operando1* > |- < *operando1* > |+ < *operando1* >

| < *operando1* > < *operador aritmético* > < *operando2* >

| +< *operando1* > < *operador aritmético* > < *operando2* >

| -< *operando1* > < *operador aritmético* > < *operando2* >

|< *operando1* > <*operador aritmético* > (- < *operando2* >)

|< *operando1* > <*operador aritmético* > (+ < *operando2* >)

|+ < *operando1*><*operador aritmético* > (- < *operando2* >)

|+ < *operando1*><*operador aritmético* > (+ < *operando2* >)

|- < *operando1*><*operador aritmético* > (- < *operando2* >)

|- < *operando1*><*operador aritmético* > (+ < *operando2* >)

< *operando1* > :: = < *variável* >

| < *constante numérica* >

| < *invocação de função* >

< *constante numérica* > :: = < *constante inteira* >

| < *constante real* >

< *constante real* > :: = < *constante inteira* >

| .< *constante inteira* >

| < *constante inteira* > . < *constante inteira* >

| < *constante inteira* >.E < *expoente* >

| .< *constante inteira* > E < *expoente* >

| < *constante inteira* >.< *constante inteira* > E < *expoente* >

< *expoente* > :: = < *constante inteira* >

| +< *constante inteira* >

| -< *constante inteira* >

< invocação de função > :: = < identificador > (< parâmetros >)

< parâmetros > :: = < constante numérica >
| < variável >
| - <constante numerica >
| + <consntante numérica >
| - <variável >
| + <variável >
| <parâmetros > , <constante numérica >
| <parâmetros > ,- <constante numérica >
| <parâmetros > ,+ <constante numérica >
| <parâmetros > , <variável >
| <parâmetros > ,- <variável >
| <parâmetros > ,+ <variável >

< operador aritmético > :: = +|-|*|/|**

< definição de bloco > :: = <cabeça de bloco ><corpo do bloco ><fim do bloco>

< cabeça do bloco > :: = BLOCO <identificador > ;

< corpo do bloco > :: = <lista de instruções >

< fim do bloco > :: = FIMBLOCO
| FIMBLOCO <identificador >

< execute > :: = EXECUTE <identificador >
| EXECUTE <identificador > ATE < condição >

< condição > :: = <expressão lógica simples >
| <expressão lógica composta >
| ¬(< condição >)

< *expressão lógica simples* > :: = < *expressão aritmética* > < *relação* >
< *expressão aritmética* >

< *relação* > :: = =|< | > |< =|> =| =

< *expressão lógica composta* > :: =< *expressão lógica simples* > < *booleano* >
< *expressão lógica simples* >

< *booleano* > :: = E|OU

< *outros* > :: = < *apague* >
| < *detalhe* >
| < *comentário* >

< *apague* > :: = APAGUE < *lista de identificadores* >

< *detalhe* > :: = DETALHE

< *comentário* > : = COM < *cadeia de caracteres* >

< *cadeia de caracteres* > :: = < *letra* >
| < *digito decimal* >
| < *caracter especial* >
| < *cadeia de caracteres* >< *letra* >
| < *cadeia de caracteres* >< *digito decimal* >
| < *cadeia de caracteres* >< *caracter especial*>

< *caracter especial* > :: = +| - |-|*|/|=|(|)|.|<|>|¬|,|:|"|';
| -| $ |%| |→|≠

6.4 - Sugestões para Implementação da Linguagem de Programação

Apresentaremos apenas as sugestões referentes a estrutura, que achamos viáveis, para a implementação da linguagem.

A tabela de símbolos poderá ser composta com as informações apresentadas na figura 6.1

IDENTIFICADOR	CLASSE	TIPO DO DADO	APONTADOR
⋮	⋮	⋮	⋮

Figura 6.1 – tabela de símbolos

A classe poderá ser uma das seguintes:

0 - variável simples

1 - variável indexada (vetor)

2 - variável indexada (matriz)

3 - identificação de bloco.

Teremos os seguintes tipos de dados

0 - sem efeito (usado para identificador de bloco)

1 - ponto fixo

2 - ponto flutuante

Conforme a classe o apontador indicará:

- variável simples *(classe ∅)* - o endereço da *"memória"* onde o dado está armazenado;

- variável indexada *(classe 1)* - o endereço da tabela *"descritor de vetor"*;

- variável indexada *(classe 2)* - o endereço da tabela *"descritor de matriz"*;

- identificador de bloco *(classe 3)* - o endereço da tabela *"descritor de bloco"*.

O descritor de vetor deverá conter o endereço da *"memória"* onde começa o armazenamento dos dados e o número de elementos do vetor, conforme mostra a figura 6.2.

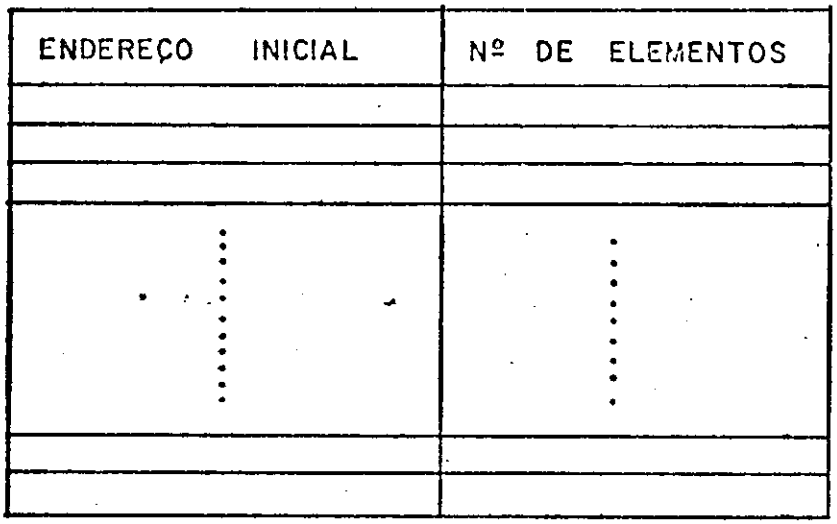


Figura 6.2 - Descritor de vetor.

O descritor de matriz, mostrado na figura 6.3, conterá uma entrada para assinalar o endereço de *"memória"* onde começa o armazenamento dos dados, uma entrada para guardar o número de linhas da matriz e uma outra entrada para reter o número de colunas da matriz.

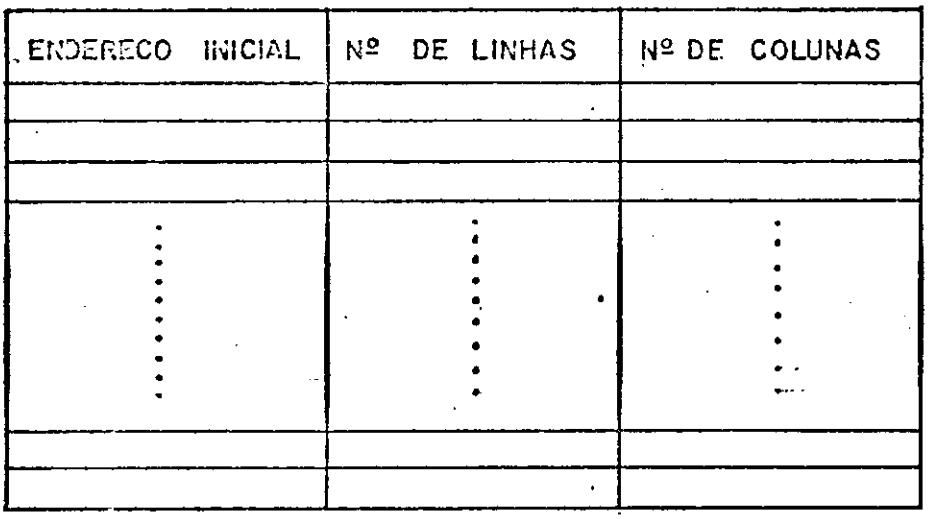


Figura 6.3 - Descritor de matriz.

O descritor de bloco, terá apenas duas entradas, uma para

apontar para o endereço de memória onde começa o armazenamento das instruções do bloco e outra para apontar para o endereço onde for armazenada a última instrução do bloco, como mostra a figura 6.4.

ENDEREÇO DA PRIMEIRA INSTRUÇÃO DO BLOCO	ENDEREÇO DA ÚLTIMA INSTRUÇÃO DO BLOCO
⋮	⋮

Figura 6.4 - Descritor de bloco.

Como guia para implementação da linguagem, apresentaremos algumas ilustrações.

(a) Variável simples, dado de ponto fixo armazenado num endereço de memória α, conforme ilustrado na figura 6.5.

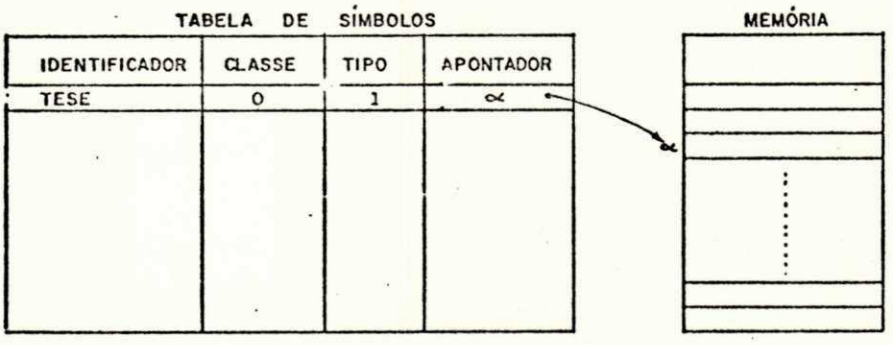


Figura 6.5 - Estrutura dos dados para variável simples.

(b) Variável indexada do tipo vetor e variável indexada do tipo matriz, serão ilustrados na figura 6.6, assim declarados:

DECLARE (A(10)) INTEIRO;

DECLARE (B(5,5)) REAL;

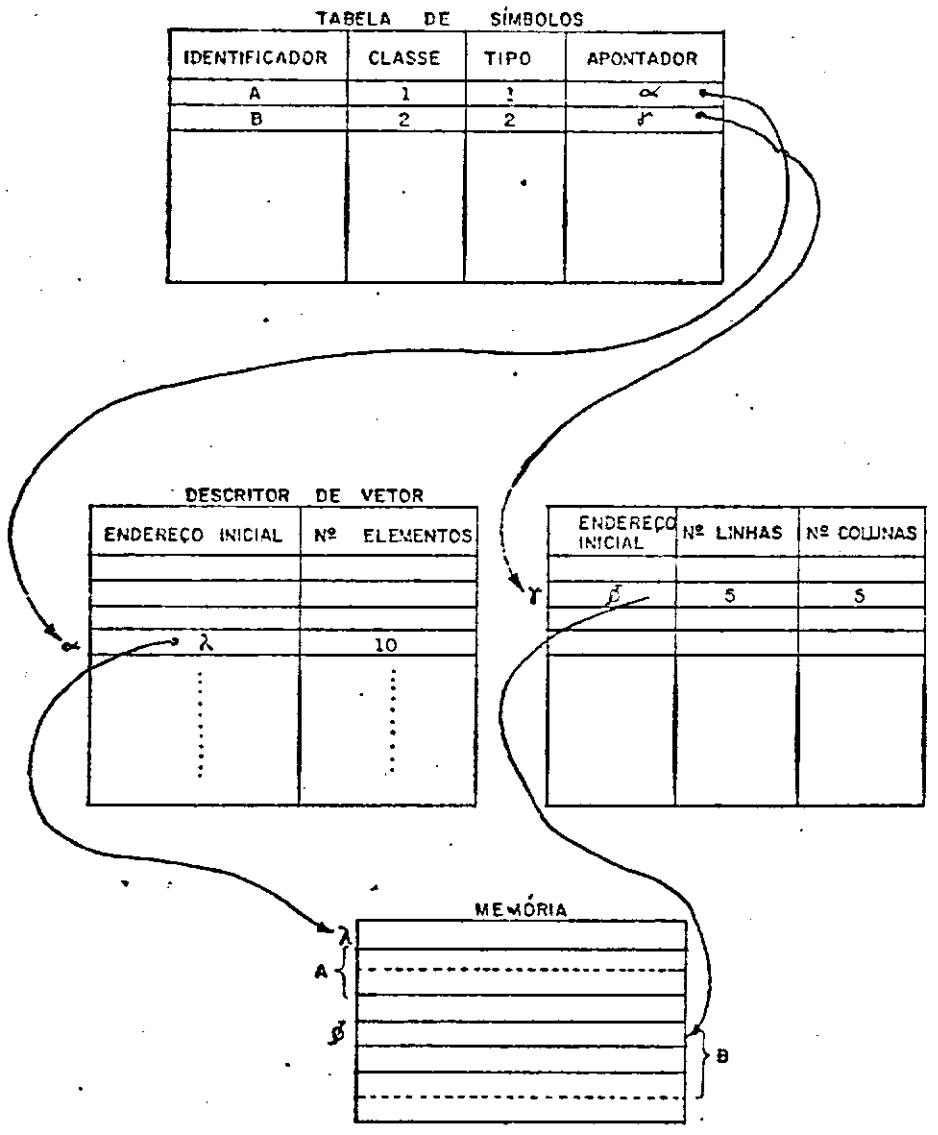


Figura 6.6 - Estrutura de dados para vetor e matriz.

CAPÍTULO VII

CONCLUSÕES E SUGESTÕES

Neste Capítulo, apresentaremos as conclusões referentes ao trabalho desenvolvido e como sugestão citaremos trabalhos que futuramente poderão ser feitos, com o objetivo de dar continuidade ao nosso trabalho.

7.1 - Conclusões

Com a visível redução nos custos de fabricação de microprocessadores devido ao avanço tecnólogico alcançado pela micro-eletrônica, estima-se que para um futuro próximo, a grande maioria dos cientistas terão a disposição, não mais calculadoras eletrônicas, mas sim microcomputadores com grande capacidade de processamento e linguagens de alto nível que facilitem o desen

volvimento de *softwares* de aplicações. Desta forma, teremos num futuro próximo, muito mais pessoas envolvidas com as idiossincrasias da aritmética de ponto flutuante.

Desenvolvemos o nosso trabalho com o objetivo de mostrar e analisar os problemas inerentes ao processamento de números de ponto flutuante.

No Capítulo II, determinamos uma forma geral de representação dos sistemas de números de ponto flutuante. A seguir passamos a analisar os sistemas numéricos utilizados pelos computadores, ou seja, aritmética de corte, aritmética com arredondamento, e uma ligeira modificação da aritmética de corte. Esta análise foi feita com o objetivo de mostrar que as leis da álgebra, quando aplicadas aos sistemas de números de ponto flutuante falham em consequência das restrições desses sistemas, ditadas pelas restrições do *hardware* das máquinas.

Considerando que a magnitude dos números de ponto flutuante é limitada pelo tamanho do expoente do número, procuramos apresentar o que acontece nos diversos computadores, quando uma operação aritmética gera um resultado fora do intervalo permitido.

No Capítulo III, estudamos os processadores de números de ponto flutuante. Procuramos mostrar que o desenvolvimento desses processadores é bastante viável, uma vez que, os computadores são projetados com a unidade aritmética de ponto fixo. Isto nos permite desenvolver processadores através de rotinas de manipulação de números de ponto flutuante, ou até mesmo por meio de *hardware* adicional que pode funcionar independente ou não do processador

de ponto fixo. Apresentamos, a título de ilustração, uma unidade aritmética de ponto flutuante, que pode ser implementada num microcomputador ou qualquer outra máquina desprovida de tal unidade. Relacionamos ainda, os requisitos necessários para implementar um processador de números de ponto flutuante em *software*.

No Capítulo IV, procuramos mostrar algumas formas sistemáticas de análise de erros, uma vez que os erros são provenientes de diversas fontes. Este capítulo poderia ser complementado com mais exemplos reais, ou mesmo através da implementação dos métodos de análise automática de erros. Uma outra forma de enriquecer este estudo, seria por meio de ilustrações que mostrassem como as operações aritméticas são feitas pelos processadores. Isto iria enriquecer a parte de geração e propagação de erros.

Intuitivamente, a maioria dos usuários, tem conhecimento que a estrutura de *hardware* influi na precisão dos resultados. Com o objetivo de analisar esta influência, no Capítulo V, fizemos o estudo de dois casos típicos de algorítmos instáveis, em quatro computadores diferentes. Poderíamos ter apresentado mais casos específicos para mostrar o comportamento de cada máquina em condições desfavoráveis, como por exemplo, em situações de transbordamento, ou mesmo com problemas mal condicionados. Achamos desnecessário estes tipos de testes, pois no seu lugar apresentamos uma série de regras para evitar certos tipos de erros, principalmente à nível de programação.

Devido a grande dificuldade dos alunos compreenderem os problemas causados pela aritmética de ponto flutuante, projetamos uma linguagem interativa com a finalidade de auxiliar a aprendizagem da teoria dos erros. Esta linguagem foi apresentada

no Capítulo VI, apenas em projeto, onde a descrevemos, formalizamos através da gramática e ainda apresentamos algumas sugestões acerca da estrutura dos dados para auxiliar na sua implementação. A linguagem de programação poderia ser mais geral, mas tendo em vista o seu objetivo, procuramos simplificar ao máximo para facilitar a sua implementação.

7.2 - Sugestões

Afim de dar continuidade ao trabalho por nós iniciado, sugerimos alguns novos trabalhos que viriam complementá-lo. A seguir passamos a enumerá-los:

1. Tomando por base as regras para evitar certos tipos de erros computacionais, sugerimos que sejam testadas no computador, acrescentando outras regras que por ventura tenhamos esquecido. Após vencer esta etapa, escrever um guia prático para desenvolvimento de programas que envolvam cálculos com números de ponto flutuante.

2. A linguagem interativa apresentada no Capítulo VI, poderá ser implementada no computador PDP11-34, instalado no Núcleo Setorial de Computação de Campina Grande. O objetivo principal, seria o assessoramento aos professores de cálculo numérico. Poderia ainda implementá-la no computador IBM 5100, ou mesmo no microcomputador MOTOROLA instalado no laboratório de sistemas digitais.

3. Implementação de alguns métodos de análise automática de erros, apresentados no Capítulo IV, que servirá para realizar análise do tipo *"benchmark"* dos programas de aplicações desenvolvidos no âmbito da UFPb.

4. Implementação de rotinas que permita ao usuário operar com a aritmética infinitesimal, ou até mesmo com múltipla-precisão. O objetivo deste trabalho é semelhante ao do ítem 3, com a diferença de que os resultados obtidos com esta aritmética seriam bem mais confiáveis.

5. Desenvolvimento e implementação de processadores de números de ponto flutuante, tanto a nível de *hardware* como de *software*, em microcomputadores ou qualquer outra máquina desprovida desses processadores.

6. A Linguagem apresentada no Capítulo VI, pode ser ampliada por meio de instruções mais gerais de construção de iterações, por acréscimo de instruções de entrada/saída, pela generalização da aritmética de ponto flutuante e por comandos auxiliares para facilitar a análise dos resultados computacionais.

A P Ê N D I C E

A.1 - Algorítmo para computação do seno de um ângulo pelo cálculo da série de potência de Taylor

```
Begin
   Leitura do grau G
   Repeat
         x = G/(180/π)
         soma = x
         termo = x
         denom = 3.0
         xq = x²
         Repeat
               termo = (-termo.xq)/(denom.(denom - 1))
               soma = soma + termo
               denom = denom + 2.0
         Until |termo| < tolerância
         Imprima  G, soma, sin(G)
         Leitura do grau G
   Until G = 0

End.
```

---

soma = *sen x* pela série de Taylor

Sin(x) = *sen G* calculado pela função embutida

A.2 - Algorítmo para calcular os limites de π pelo método do polígono inscrito e circunscrito

Begin

$s = 2$ s é o comprimento do lado de um quadrado inscrito

$n = 3$ determinará o número de lados do próximo polígono inscrito

Repeat

$$s = \sqrt{2 - \sqrt{4 - s^2}}$$

$$\ell = 2^n$$

$$inf\pi = \frac{s.\ell}{2}$$

$$sup\pi = \frac{s.\ell}{\sqrt{4 - s^2}}$$

imprima n, ℓ, $inf\pi$, $sup\pi$

$n = n + 1$

Until $n > n_{max}$

End.

$inf\pi$ = limite inferior de π

$sup\pi$ = limite superior de π

n_{max} = determina o número de lados do maior polígono inscrito

B I B L I O G R A F I A

1. [ABD-ALLA, 1976] - Adb-elfattah ABD-ALLA, Arnold G MELTZER
Principles of digital Computer Design, Vol.I
Prentice-Hall, Inc., Englewood Cliffs, N. J.

2. [BRENT, 1978] - Richard P. BRENT
A FORTRAN Multiple-Precision Arithmetic Package ACM Trans. on Math. Software, 1978
páginas 57 - 81.

3. [FORSYTHE, 1970] - George E. FORSYTHE
Pitfalls in Computation, or Why a Math isn't Enough
American Mathematics Monthly, Nov. 1970
páginas 931 - 956.

4. [FRÖBERG, 1970] - Carl-Erik FRÖBERG
Introduction To Numerical Analysis
Addison-Wesley Publishing Company

5. [GALLER, 1970] - B.A. GALLER, A. J. PERLIS
A View of Programming Languages
addison-Wesley Publishing Company

6. [GRIES, 1971] - David GRIES
Compiler Construction for Digital Computers
John Wiley & Sons, Inc

7. [GSCHIND, 1967] - Hans W. GSCHIND
Design of Digital Computers
Springer-Verlag, N. Y.

8. [HAMACHER, 1976] - V.C. HAMACHER, Z.G. VRANESIC, S.G. ZAKY
Computer Organization
Faculty of Applied Science and Engineering
University of Toronto, Department of Electrical
Engineering, Lecture Notes

9. [HAYES, 1978] - John P. HAYES
Computer Architecture and Organization
McGraw-Hill Book Company, N. Y.

10. [HEHL, 1974] - Maximilian Emil HEHL
FORTRAN Técnicas Práticas e Eficientes em Programação
Editora da Universidade de São Paulo, Livros Técnicos e Científicos Editora S.A. - R.J.

11. [HENRICI, 1964] - P. HENRICI
Elements of Numerical Analysis
John Wiley & Sons, Inc. N. Y.

12. [KNUTH, 1971] - Donald E. KNUTH
The Art of Computer Programming,
Vol.2 Seminumerical Algorithms
Addison-Wesley Publishing Company

13. [LEE, 1972] - John A. N. LEE
Computer Semantics - Studies of
Algorithms, Processors and
Languages
Van Nostrand Reinhold Company

14. [LEE, 1967] - John A. N. LEE
The Anotomy of a Compiler
Van Nostrand Reinhold Company

16. [MANO, 1976] - M. Morris MANO
Computer System Architecture
Prentice-Hall, Inc., Englewood
Cliffs, N. J.

17. [MARCUS, 1967] - Mitchell P. MARCUS
Switching Circuits for Engineers
Prentice-Hall, Inc., Englewood
Cliffs, N.J.

18. [MCCRAKEN, 1978] - Willian S. DORN, Daniel D. MCCRAKEN
Cálculo Numérico com Estudos de Casos em FORTRAN IV
Editora Campus Ltda., Editora da Universidade de São Paulo, R.J.

19. [MCKEEMAN, 1970] - William M. MCKEEMAN, Janes J. HORNING, David B. WORTMAN
A Compiler Generator
Prentice-Hall, Inc., Englewood Cliffs, N.J.

20. [MILLER, 1975] - Webb MILLER
Software for Roundoff Analysis
ACM Trans. on Math. Software,
June, 1975
páginas 108 - 128

21. [MILLER, 1978] - Webb MILLER, David Spooner
Software for Roundoff Analysis, II
ACM Trans. on Math. Software,
dec. 1978
páginas 369 - 390

22. [ORTEGA, 1972] - Janes M. ORTEGA
Numerical Analysis - A Second Course
Academic Press, N. Y.

23. [PRATT, 1975] - Terrence W. PRAT
Programming Languages: Design and Implementation
Prentice-Hall, Inc., Englewood Cliffs, N.J.

24. [PENNINGTON, 1965] - Ralph H. PENNINGTON
Introductory Computer Methods and Numerical Analysis
The Macmillan Company, N.Y.

25. [PRICE, 1968] - Wilson T. PRICE
Elements of IBM 1130 Programming
Holt, Rinehart and Winston, Inc.

26. [RALSTON, 1965] - Anthony RALSTON
A First Course in Numerical Analysis
McGraw-Hill Kogakusha, Ltda.

27. [STERBENZ, 1974] - Pat H. STERBENZ
Floating-Point Computation
Prentice-Hall, Inc., Englewood Cliffs, N. J.

28. [YOUNG, 1972] - David M. YOUNG, Robert Todd GREGORY
A Survey of Numerical Mathematics, Vol. I
Addison-Wesley Publishing Company, Inc.