31251-8

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PROJETO DE OPERADORES ARITMETICOS DE PONTO FLUTUANTE EM TECNOLOGIA CMOS

por

Laerte Davi Cleto

Dissertação submetida como requisito parcial
para obtenção do grau de Mestre em
Ciência da Computação

Prof. Tiaraju Vasconcellos Wagner
Orientador

Prof. Philippe Olivier Alexandre Navaux
Co-orientador

SABi

UFRGS 05225332

Porto Alegre, julho de 1990.

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

CATALOGAÇAO NA FONTE

Cleto, Laerte Davi
Projeto de operadores aritméticos ponto flutuante em tecnologia CMOS.
Porto Alegre, CPGCC da UFRGS, 1990.
1v.
Diss. (mestr. ci. comp.) UFRGS-CPGCC, Porto Alegre, BR-RS, 1990.

Dissertação: Microeletrônica: Arquitetura de Computadores: Aritmética Binária: Aritmética Binária de Ponto Flutuante: Projeto VLSI.

Arquitetura de computadores
Microeletrônica - SBU/II
Processadores aritiméticos
Aritmética : Ponto Flutuante

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA
Nº CHAMADA: 621.38-181.4(043) C634p
Nº REG.: 6230
DATA: 10/12/90
ORIGEM: D
DATA: 4/12/90
PREÇO: Cr$ 4000,00
FUNDO: II
FORN.: PGCC

Este trabalho é dedicado a meus pais, Noni e Vica, pelo amor e incentivo que deles sempre tenho recebido.

ESTE LIVRO DEVE SER DEVOLVIDO NA
ÚLTIMA DATA CARIMBADA

21 FEV 1991	23 AGO 1994		
03 MAI 1991	30 AGO 1994		
20 MAI 1991	18 SET 1995		
04 OUT 1991	26 SET 1995		
08 MAI 1992	03 OUT 1995		
13 JAN 1994	19 MAR 1996		
24 MAR 1994	040994		
04 ABR 1994			
18 ABR 1994			
02 MAI 1994			
27 JUN 1994			
21 JUL 1994			
2-8-94			
16 AGO 1994			

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Sistema de Biblioteca da UFRGS

6230 CLETO...

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

MOD. 2.3.2

AGRADECIMENTOS

Este trabalho só foi possível devido a uma série de condições propícias. Entre elas inclui-se todo apoio recebido, de uma forma ou de outra. A todos os que, assim, participaram, sou profundamente grato. No entanto, gostaria de agradecer de uma forma especial:

Ao Sr. Emerson Rogério de Oliveira e à sua esposa, D. Zaira, pela gentileza e boa vontade que tiveram em hospedar-me, provisoriamente, nas primeiras semanas de minha chegada em Porto Alegre. E também, pelo grande apoio em minha instalação definitiva.

A Teilor Keipek (in memorian) e Silvia Dani, bem como seus pais, pelos inestimáveis favores e amizade.

Ao professor Tiaraju Vasconcellos Wagner. Primeiramente por ter aceito orientar esta dissertação; depois, pela disponibilidade e prestatividade com a qual sempre desempenhou tal trabalho e também pelo amigo, que revelou ser, durante o período de orientação.

Ao professor Philippe O. A. Navaux pela sua colaboração na co-orientação desta dissertação.

A Luiz Cláudio Villar dos Santos, colega e companheiro de estudos há muitos, pela grande amizade.

A Gilberto Marchioro (Giba) pela solicitude com que auxiliou-me na implementação de parte do programa simulador.

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

Aos assíduos participantes da Turma do Cafezinho Mais Barato da Cidade: Alcides (patrocinador oficial), Luigi, Santos, Marcelo Walter, Marcos Dossa, Ramon, Rosana, Sílvia entre outros, não tão assíduos, com quem compartilhava - além do cafezinho, evidentemente - momentos de divertida conversa fiada.

Aos professores integrantes do comitê de leitura: Professor Altamiro Amadeu Suzim e Professor Dante Barone pelas leituras cuidadosas e observações, que muito contribuiram para o aprimoramento da versão final deste volume.

Ao coordenador do CPGCC, Professor Ricardo Reis, pela disposição e empenho em conceder-me a oportunidade de continuar desenvolvendo um trabalho de pesquisa em projeto de circuitos integrados, junto ao Grupo de Microeletrônica.

Às funcionárias da secretaria do DI e do CPGCC pela presteza com a qual sempre fui atendido.

Às funcionárias da Biblioteca do CPGCC pela simpatia e eficiência com que sempre exercem seu trabalho.

À querida Taninha pelo valoroso auxílio na pesquisa bibliográfica desta dissertação, pela zeloza correção final deste volume e, sobretudo, pela sua importante presença, paciência, incentivo e carinho.

SUMÁRIO

GLOSSÁRIO 11
LISTA DE ABREVIATURAS 13
LISTA DE FIGURAS 15
LISTA DE TABELAS 17
RESUMO 19
ABSTRACT 21
1 INTRODUÇÃO 23
2 PROCESSADORES ARITMÉTICOS INTEGRADOS 27
2.1 Introdução 27
2.2 Classificação dos Processadores Aritméticos Integrados 28
2.3 Características principais dos Processadores Aritméticos Integrados 31
2.3.1 Formatos de representação numérica suportados .. 32
2.3.2 Operações disponíveis 36
2.3.3 Arquitetura interna 37
2.3.4 Entrada e saída 42
2.3.5 Desempenho 43
2.4 Conclusões e tendências 49
3 ALGORITMOS DAS OPERAÇÕES DE ADIÇÃO, SUBTRAÇÃO E MULTIPLICAÇÃO EM PONTO FLUTUANTE 51
3.1 Formatos de representação numérica em sistemas de computação convencionais 51
3.1.1 Representações numéricas binárias em ponto fixo 52
3.1.2 Representações numéricas binárias em ponto flutuante 57
3.2 Padrão IEEE para aritmética binária de ponto flutuante 59
3.2.1 Definições 59
3.2.2 Formatos 60
3.2.2.1 Conjunto de valores 61
3.2.2.2 Formatos Básicos 62
3.2.2.3 Formatos Extendidos 64
3.2.3 Arredondamento 66

3.2.4 Exceções 67
3.3 Algoritmos das operações de adição, subtração e multiplicação em ponto flutuante 69
3.3.1 Adição e Subtração binária em ponto flutuante .. 70
3.3.1.1 Introdução 70
3.3.1.2 Verificação dos operandos 72
3.3.1.3 Alinhamento das mantissas 72
3.3.1.4 Adição ou Subtração das Mantissas 73
3.3.1.5 Renormalização 74
3.3.1.6 Arredondamento 77
3.3.1.7 Tratamento de exceções 81
3.3.2 Multiplicação binária em ponto flutuante 83
3.3.2.1 Introdução 83
3.3.2.2 Verificação dos operandos 85
3.3.2.3 Adição dos expoentes 85
3.3.2.4 Multiplicação das mantissas 86
3.3.2.5 Renormalização e arredondamento 86
3.3.2.6 Tratamento de exceções 87
3.4 Conclusões 88
4 ARQUITETURA DOS OPERADORES ARITMÉTICOS DE PONTO FLUTUANTE 91
4.1 Introdução 91
4.2 Arquitetura para o operador de adição e subtração em ponto flutuante 91
4.2.1 Aspectos inerentes à implementação em hardware . 92
4.2.1.1 Adição e subtração das mantissas 92
4.2.1.1.1 Operação efetiva 92
4.2.1.1.2 Adição e subtração em sinal-magnitude 94
4.2.1.2 Arredondamento 97
4.2.1.2.1 Bits de arredondamento 98
4.2.1.2.2 Procedimento de arredondamento para o mais próximo a partir dos bits de arredondamento. 101
4.2.2 Descrição da arquitetura do somador/subtrator em ponto flutuante 102
4.2.2.1 Análise e comparação dos expoentes e mantissas 104

4.2.2.2 O subtrator de expoentes 105
4.2.2.3 Lógica de controle de exceção 105
4.2.2.4 Deslocador da mantissa 106
4.2.2.5 Somador/subtrator das mantissas 107
4.2.2.6 Lógica de cálculo dos bits de arredondamento . 109
4.2.2.7 Lógica de renormalização e arredondamento 111
4.2.2.8 Lógica de cálculo de exceção 113
4.2.2.9 Comentários sobre a arquitetura 114
4.3 Arquitetura para o operador de multiplicação em ponto flutuante 116
4.3.1 Aspectos inerentes à implementação em hardware . 116
4.3.2 Descrição da arquitetura do multiplicador ponto flutuante 118
4.3.2.1 Análise das mantissas e expoentes 120
4.3.2.2 Multiplicador das mantissas 120
4.3.2.3 Somador dos expoentes 121
4.3.2.4 Lógica de cálculo dos bits de arredondamento . 121
4.3.2.5 Lógica de renormalização e arredondamento 122
4.3.2.6 Lógica de controle e cálculo de exceção 123
4.4 Comparações e conclusão 124
5 IMPLEMENTAÇAO DE UM MULTIPLICADOR PONTO FLUTUANTE PIPELINE 127
5.1 Introdução 127
5.2 Restrições de ordem prática e econômica 127
5.3 O multiplicador ponto flutuante pipeline 128
5.3.1 Entrada e saída 128
5.3.2 O multiplicador das mantissas 132
5.3.2.1 O Multiplicador proposto 136
5.3.3 A lógica de renormalização e arredondamento 148
5.3.3.1 Renormalização 148
5.3.3.2 Circuito de arredondamento 152
5.3.4 A lógica de exceção 153
5.3.5 O Pipeline 158
5.3.6 A lógica de controle e Temporização 164
5.3.7 Validação 179

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

5.4 Conclusões 192

6 CONCLUSAO 193

ANEXO 1 Listagem do programa simulador do circuito multiplicador ponto flutuante 201

BIBLIOGRAFIA 215

GLOSSÁRIO

Carry : Bit que propaga-se (vai-um) nas operações aritméticas binárias de adição e multiplicação.

Carry-in : Entrada de carry no bit menos significativo de um somador.

Carry-out : Carry que propaga-se além do limite da representação.

Tempo de Latência : Tempo necessário para execução de uma operação especifica.

Throughput : Taxa de saída dos resultados.

LISTA DE ABREVIATURAS

BCD - Binary Code Decimal
CAD - Computer-Aided Design
CMOS - Complementary Metal Oxide Silicon
CSA - Carry-Save Adder
EAC - End Around Carry
Emáx - Expoente máximo
Emin - Expoente mínimo
Exp - Expoente
IEEE - Institute of Electrical and Electronic Engineers
LSB - Least Significant Bit
Mant - Mantissa
máx - máximo
MFLOPS - Millions of Floating-Point Operations per Second
mín - mínimo
MOS - Metal Oxide Silicon
MSB - Most Significant Bit
MSI - Medium Scale Integration
NaN - Not a Number
NMM - Nonadditive Multiply Modulus
PLA - Programable Logic Array
RISC - Reduced Instruction Set Computer
ULA - Unidade Lógica e Aritmética
VLSI - Very Large Scale Integration

LISTA DE FIGURAS

Figura 2.1 - Arquitetura interna do coprocessador aritmético Motorola 68882 [MOT 87] 40
Figura 2.2 - Arquitetura interna do processador aritmético ADSP 3202 [ANA 87] 41
Figura 3.1 - Convenções para posição do ponto binário nas representações em ponto fixo 53
Figura 3.2 - Formato Básico Simples (32 bits) [IEE 87]. 63
Figura 3.3 - Formato Básico Duplo (64 bits) [IEE 87] .. 63
Figura 4.1 - Somador/Subtrator sinal-magnitude com EAC [HWA 79] 95
Figura 4.2 - Somador/Subtrator sinal-magnitude com comparador 97
Figura 4.3 - Bits de arredondamento 99
Figura 4.4 - Arquitetura do somador/subtrator ponto flutuante 103
Figura 4.5 - Bits de arredondamento na multiplicação ponto flutuante 117
Figura 4.6 - Arquitetura do multiplicador ponto flutuante 119
Figura 5.1 - Diagrama de tempos da entrada de operandos - Funcionamento pipeline 131
Figura 5.2 - Diagrama de tempos da saída de resultados - Funcionamento pipeline 132
Figura 5.3 - Multiplicador Array [HWA 79] 136
Figura 5.4 - Algoritmo para multiplicação de operandos de 8 bits utilizando módulos multiplicadores de 4 bits [CAV 85] 138
Figura 5.5 - Adição dos produtos parciais 141
Figura 5.6 - Procedimento de adição dos produtos parciais 142
Figura 5.7 - Componentes do produto final 143
Figura 5.8 - Procedimento de adição dos produtos parciais e cálculo dos bits de arredondamento 145

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

Figura 5.9 - Hardware básico do multiplicador das mantissas 146
Figura 5.10 - Renormalização na multiplicação ponto flutuante após a multiplicação das mantissas 149
Figura 5.11 - Circuito de renormalização com dois deslocadores 150
Figura 5.12 - Circuito de renormalização com um deslocador 151
Figura 5.13 - Circuito de arredondamento 153
Figura 5.14 - Circuito de análise das mantissas e expoentes 155
Figura 5.15 - Arquitetura simplificada do circuito multiplicador ponto flutuante pipeline .. 161
Figura 5.16 - Máquina de estados da entrada de operandos 166
Figura 5.17 - Máquina de estados da multiplicação das mantissas 167
Figura 5.18 - Formas de onda de um clock de duas fases não sobrepostas 168
Figura 5.19 - Um circuito com dois módulos a pré-carga [GLA 85] 170
Figura 5.20 - Estrutura pipeline com módulos combinacionais a pré-carga 171
Figura 5.21 - Estrutura pipeline com módulos combinacionais e sequenciais a pré-carga. 172
Figura 5.22 - O clock e os sinais de controle 173
Figura 5.23 - Estrutura geral do circuito 175
Figura 5.24 - Formato Básico Simples (32 bits) [IEE 87] 183

LISTA DE TABELAS

Tabela 2.1 - Formatos de representação numérica suportados pelos Processadores Aritméticos Integrados 36

Tabela 2.2 - Operações disponíveis nos processadores aritméticos integrados 38

Tabela 2.3 - Desempenho dos processadores aritméticos nas operações de adição e subtração em ponto flutuante 46

Tabela 2.4 - Desempenho dos processadores aritméticos na operação de multiplicação em ponto flutuante 47

Tabela 2.5 - Desempenho dos processadores aritméticos na operação de divisão em ponto flutuante 48

Tabela 2.4 - Desempenho dos processadores aritméticos na operação de raiz quadrada 49

Tabela 3.1 - Comparação dos sistemas de representação numérica em ponto fixo [GOS 80] 56

Tabela 3.2 - Comparação entre intervalos dinâmicos e precisão nas representações ponto fixo e ponto flutuante 58

Tabela 3.3 - Sumário dos parâmetros dos formatos [IEE 87] 61

Tabela 4.1 - Sinal do resultado da Subtração 94

Tabela 4.2 - Análise dos Expoentes e Mantissas 104

Tabela 4.3 - Análise dos Expoentes e Mantissas 120

RESUMO

Este trabalho aborda algumas etapas do projeto de operadores aritméticos de ponto flutuante visando sua implementação integrada.

Inicialmente são estudados os algoritmos das operações de adição, subtração e multiplicação envolvendo operandos representados nos formatos estabelecidos pelo Padrão IEEE para aritmética binária de ponto flutuante [IEE 87].

A partir dos algoritmos são propostas arquiteturas para aqueles operadores, procurando aproveitar características de paralelismo para acelerar a execução.

Detalha-se a proposta arquitetural do operador de multiplicação em ponto flutuante considerando algumas questões de caráter prático. Estabelece-se uma estrutura pipeline, o controle e a temporização para o circuito. A implementação, neste nível, é validada por simulação.

ABSTRACT

This work deals with some design steps of integrated floating-point arithmetic operators.

Firstly, the algorithms of floating-point addition, subtraction and multiplication are studied, based on the IEEE Standard for binary floating-point arithmetic [IEE 87].

After, some architectural solutions are proposed for the above operators, taking in account the parallel caracteristics of the algorithms for gain execution speed.

The architectural level of the floating-point multiplier operator is detailed, emphasizing some practical matters; including a pipeline structure, control and timing of the circuit. Simulation is used to confirm the design proposed.

1 INTRODUÇÃO

O cálculo aritmético constitui uma das atividades mais importantes e freqüentes nos computadores. E, portanto, para uma grande parte das aplicações, o aumento do desempenho no processamento em geral está diretamente relacionado à aceleração das operações aritméticas.

Desde o início, as aplicações ditas científicas revelaram a necessidade de uma representação numérica mais poderosa, que apresentasse intervalo dinâmico e precisão maiores que as oferecidas pelas representações convencionais (ponto fixo). Foi proposta, então, uma representação numérica binária de ponto flutuante caracterizada, principalmente, por constituir-se de dois campos: a mantissa e o expoente, que atendia àqueles requisitos.

Entretanto, operações envolvendo operandos em tal representação mostraram-se sempre muito custosas em tempo de processamento.

Com o objetivo de acelerar a execução destes cálculos desenvolveu-se hardware especializado para implementar os algoritmos das operações de ponto flutuante. Estes subsistemas, inicialmente, devido ao custo excessivamente alto, eram só encontrados em computadores de grande porte.

O desenvolvimento contínuo das tecnologias de implementação de sistemas digitais viabilizou a construção de sistemas cada vez mais rápidos, complexos e baratos.

As tecnologias de alta escala de integração (Very Large Scale Integration - VLSI) possibilitaram a compactação de circuitos que antes ocupavam uma placa ou mais, em

uma pastilha de silício e também a exploração de novas arquiteturas cujas implementações eram inviáveis nas tecnologias anteriores.

Assim, a proliferação de sistemas digitais em outras áreas criou novas necessidades de processamento numérico. Aplicações como CAD (Computer-Aided Design), computação gráfica, simulação, tratamento digital de sinais e imagens, controle numérico, robótica entre outras, vem exigindo capacidade de processamento aritmético de ponto flutuante cada vez maiores. Estas necessidades vêm sendo, em grande parte, atendidas por circuitos integrados dedicados à execução de operações aritméticas.

Este trabalho aborda o projeto de operadores aritméticos de ponto flutuante rápidos (a base de um sistema de processamento numérico de alto desempenho) visando uma implementação integrada, particularmente em tecnologia CMOS (Complementary Metal-Oxide Silicon).

No capítulo 2 apresenta-se um estudo dos circuitos processadores aritméticos comerciais. Tal estudo possibilita o conhecimento de aspectos como estado da arte em termos tecnológicos e algumas necessidades do mercado consumidor (considerando que a oferta ao menos reflete uma demanda potencial) entre outros. A pesquisa é feita procurando extrair as características principais dos processadores aritméticos, comparando-as.

O capítulo 3 apresenta detalhadamente os algoritmos das operações de adição, subtração e multiplicação envolvendo operandos representados em ponto flutuante. Inicia-se apresentando o padrão de representação numérica adotado. Baseado nesta notação, em outros aspectos recomendados pelo mesmo padrão e na aritmética binária

propriamente dita são construídos os algoritmos das operações. É dada certa ênfase a características normalmente pouco abordadas como a renormalização e o arredondamento.

A partir dos algoritmos estabelecidos, são propostas arquiteturas para implementação destes operadores, no capítulo 4. São discutidos aspectos particulares às realizações em hardware, explorando as disponibilidades dos algoritmos objetivando a aceleração das operações.

Finalmente, no capítulo 5, sobre a arquitetura proposta para o operador de multiplicação em ponto flutuante são impostas algumas restrições de caráter prático e econômico que alteram parte desta proposta inicial. Detalha-se o projeto, estabelece-se uma estrutura pipeline, define-se a temporização do circuito e valida-se a implementação por simulação.

2 PROCESSADORES ARITMÉTICOS INTEGRADOS

Neste capítulo faz-se uma breve apresentação dos circuitos processadores aritméticos. Recorda-se, inicialmente, a evolução de tais sistemas devido ao desenvolvimento tecnológico e à necessidade de maior desempenho e velocidade do processamento.

É apresentada uma classificação para os circuitos Processadores Aritméticos Integrados [FAN 85], listando, para cada uma das classes, as características gerais.

Em seguida apresenta-se um pouco mais profundamente as características principais destes circuitos tendo por base uma pesquisa realizada sobre alguns circuitos comerciais de cada categoria [CLE 88].

2.1 Introdução

A crescente necessidade de processamento numérico de alto desempenho em várias áreas, a padronização do sistema de representação numérica de ponto flutuante para pequenos e médios sistemas pelo IEEE (Institute of Electrical and Electronic Engineers), o desenvolvimento das tecnologias de alta integração (VLSI) bem como a evolução das ferramentas de auxílio ao projeto de circuitos integrados motivaram técnica e economicamente o desenvolvimento dos circuitos processadores aritméticos integrados.

Processadores Aritméticos são circuitos integrados que tem por função aumentar o desempenho de sistemas que requerem elevado processamento numérico, principalmente em ponto flutuante e onde não existem unidades embutidas dedicadas a estas funções. Estes sistemas possuem, geralmente, como processador principal microprocessadores

de propósitos gerais (ex.: 68000, 8086, etc.).

Evidentemente as rotinas das operações em ponto flutuante programadas a partir do conjunto de instruções destes microprocessadores de propóstos gerais apresentam desempenho que deixa bastante a desejar. Como poderá ser verificado nos próximos capítulos, os algoritmos das operações aritméticas mais simples tais como adição, subtração, multiplicação, etc. atingem níveis de complexidade consideráveis quando manipulando operandos representados em ponto flutuante.

O aumento da velocidade de processamento numérico é conseguido, em princípio, implementando em hardware as rotinas que realizam as operações arimtéticas, enquanto as transferências de dados (operandos, resultados, instruções e status) são otimizadas para que gargalos de barramentos não anulem o desempenho global.

Os Processadores Aritméticos Integrados encontram-se associados a pequenos e médios sistemas computacionais (computadores pessoais, estações de trabalho, etc.) ou compondo sistemas dedicados como placas para tratamento de sinais ou imagens, etc.

2.2 Classificação dos Processadores Aritméticos Integrados

O mercado de circuitos integrados apresenta uma grande variedade de processadores aritméticos integrados, cada qual com suas particularidades, mas compartilhando o fato de visarem a aceleração de operações, sobretudo quando envolvendo operandos representados em ponto flutuante.

Com o objetivo de sistematizar o estudo utilizaremos a

classificação sugerida em [FAN 85]. Segundo esta classificação, os Processadores Aritméticos Integrados estão divididos em dois grupos: Processadores Aritméticos Integrados de Propósitos Gerais, também chamados de Coprocessadores Aritméticos de Propósitos Gerais e Processadores Aritméticos Integrados Dedicados.

a) Coprocessadores Aritméticos de Propósitos Gerais

Estes circuitos processadores aritméticos integrados são chamados "*de propósitos gerais*" pois executam um grande repertório de operações e funções aritméticas atuando sobre dados representados nos mais variados formatos, tanto em ponto fixo como em ponto flutuante.

Os coprocessadores aritméticos de propósitos gerais aparecem, normalmente, associados ao microprocessador da família correspondente de circuitos integrados. Ex.: o coprocessador aritmético Intel 80387 [INT 87] foi desenvolvido visando integração a sistemas baseados no microprocessador 80386.

Esta característica se evidencia quando observa-se que existe um protocolo especial de comunicação entre os dois circuitos, transparente ao usuário, visando acelerar as transferências de informação e facilitar a programação. Desta forma esta categoria aparece como uma extensão do conjunto de instruções do microprocessador para operações e funções aritméticas e novos formatos de representação numérica.

Por sua flexibilidade e fácil adaptabilidade, constituem uma opção vantajosa para pequenos sistemas. Como representantes dos coprocessadores aritméticos de propósitos gerais podemos citar Intel 80387 [INT 87] e

Motorola 68882 [MOT 87].

b) Processadores Aritméticos Dedicados

São denominados "*Dedicated Building Blocks*" pois constituem os elementos (blocos) principais na elaboração das placas aceleradoras aritméticas de alto desempenho.

A abordagem utilizada nestes circuitos é de se implementar com altíssimo desempenho as operações mais comuns como adição e multiplicação, e, somente sobre poucos formatos de representação numérica. Para tal, valem-se de circuitos dedicados, como os multiplicadores em array, que lhes conferem desempenho elevado. Dispõem, também, de arquiteturas do tipo pipeline que possibilita alto throughput.

Apesar de exigirem o projeto de uma interface adaptativa que realize a conexão entre o processador aritmético e o processador principal, sua implementação é facilitada pois os processadores aritméticos dedicados possuem estruturas de entrada e saída bastante simples.

São empregados em sistemas que exijam alto desempenho nestas operações. A arquitetura pipeline é especialmente interessante em sistemas de tratamento de sinais e imagens.

Como representantes investigados podemos citar os circuitos Weitek WTL 1232/1233 [WEI 86] e WTL 2264/2265 [WEI 86a] e os circuitos Analog Devices ADSP 3201/3202 [ANA] e ADSP 3212/3222 [ANA 87].

No entanto as inovações tecnológicas permitiram que o panorama dos circuitos processadores aritméticos integrados

se alterasse desde a proposta de classificação citada.

A possibilidade de realizar integração de cada vez mais elementos viabilizou a implementação de novas arquiteturas impossibilitadas anteriormente pelo custo excessivo do hardware. Neste caminho podemos vislumbrar os circuitos sistólicos que executam algoritmos extremamente específicos com alto desempenho.

Outra conseqüência direta da viabilidade de integração em alta escala (VLSI) é a sua incorporação aos microprocessadores de propósitos gerais. Os circuitos microprocessadores mais avançados já possuem unidades de processamento aritmético em ponto flutuante embutidas. Os exemplos mais conhecidos são o microprocessador com arquitetura RISC (Reduced Instruction Set Computer) da Motorola 88000 e os últimos descendentes das gerações Intel 8086, o 80486, e Motorola 68000, o 68040.

Uma classificação atual que se pretendesse completa não poderia deixar de incluir os sistemas acima citados. No entanto, foge ao escopo deste trabalho tentar apresentar uma classificação satisfatória para o panorama atual. Adotaremos, portanto, a classificação apresentada tendo em vista que os sistemas acima citados são lançamentos bastante recentes e o estudo que segue baseou-se em sistemas que se enquadram perfeitamente naquela classificação.

2.3 Características principais dos Processadores Aritméticos Integrados

Neste sub-item são apresentadas as características principais dos circuitos processadores aritméticos

integrados, que são:

a) Formatos de representação numérica suportados

b) Operações disponíveis

c) Arquitetura interna

d) Entrada e Saída

e) Desempenho

Em todos os aspectos é realizada uma comparação entre as características dos circuitos coprocessadores de propósitos gerais e dos circuitos processadores aritméticos dedicados.

2.3.1 Formatos de representação numérica suportados

Existem muitos formatos de representação numérica binários, muitos códigos cada qual com suas características e propriedades. Códigos especiais para comunicação de dados, para detecção e/ou recuperação de erros, etc. Em computação numérica convencional os formatos de representação numérica recaem, sobretudo, em duas categorias:

a) representação em ponto fixo
b) representação em ponto flutuante

Na representação em ***ponto fixo*** o sinal separador da parte inteira da parte fracionária do número (o ponto binário) encontra-se subentendidamente definido entre dois dígitos adjacentes no número. A consideração mais comum é localizá-lo imediatamente à direita do bit menos significativo do número, o que equivale a considerá-los todos como inteiros.

A representação em ***ponto flutuante*** é constituída por dois campos: a mantissa e o expoente. O expoente fornece a posição do sinal separador da parte inteira da parte fracionária na mantissa.

A representação em ponto flutuante é necessária onde exige-se grandes intervalos dinâmicos e precisão. No entanto a operacionalidade de tal representação é complexa e portanto lenta. A necessidade de acelerar estas operações motivou o surgimento dos processadores aritméticos.

a) Coprocessadores Aritméticos de Propósitos Gerais

Os que se enquadram nesta categoria operam sobre números representados em vários formatos, tanto em ponto fixo como em ponto flutuante.

Nos coprocessadores aritméticos analisados observa-se que o Intel 80387 [INT 87] executa suas instruções sobre operandos em ponto fixo (complemento de dois) em 16, 32 e 64 bits enquanto seu equivalente, da Motorola, o 68882 [MOT 87], as executa na mesma representação em 8, 16 e 32 bits.

Os formatos de representação em ponto flutuante são os mesmos para ambos (segundo as recomendações do padrão IEEE para aritmética binária de ponto flutuante [IEE 87]).

- precisão simples - 32 bits
- precisão dupla - 64 bits
- precisão extendida - 80 bits

O coprocessador aritmético 80387 ainda aceita operandos em BCD (Binary Coded Decimal) com 18 dígitos decimais em ponto fixo. O 68882 suporta a mesma

representação mas em ponto flutuante tendo 17 dígitos decimais de mantissa e três dígitos decimais de expoente.

Na realidade as operações, em ambos os circuitos, são realizadas sobre um único formato de representação: ponto flutuante precisão dupla extendida - 80 bits. Todos os operandos quando carregados para dentro do circuito são automaticamente convertidos para esta representação. Quando os resultados são enviados para a saída é realizada automaticamente a conversão para o formato desejado.

b) Processadores Aritméticos Dedicados

Os circuitos estudados, desta categoria, se apresentam em chip-sets, ou seja, um conjunto de dois circuitos integrados - um que efetua multiplicações e divisões e outro, uma ULA (Unidade Lógica e Aritmética) que realiza várias operações. Possuem sempre versões para ponto flutuante precisão simples e precisão dupla, sendo que os circuitos que operam em precisão dupla normalmente também operam em precisão simples.

Os circuitos estudados que operam em ponto flutuante-precisão simples são: Weitek WTL 1232 (multiplicador)/ WTL 1233 (ULA) [WEI 86] e Analog Devices ADSP 3201 (multiplicador/divisor)/ ADSP 3202 (ULA) [ANA 87]. Os que operam em precisão dupla: Weitek WTL 2264 (multiplicador/divisor)/ WTL 2265 (ULA) [WEI 86a] e Analog Devices ADSP 3221 (multiplicador/divisor)/ ADSP 3222 (ULA) [ANA 87].

Além dos formatos de representação em ponto flutuante acima citados os circuitos ADSP 3201/3202 e ADSP 3221/3222 também suportam operandos representados em ponto fixo (32

bits) tanto em complemento de dois como em magnitude (sem sinal). Os circuitos WTL 2264/2265 suportam operandos em ponto fixo - complemento de dois - 32 bits.

Existem Processadores Aritméticos Dedicados que executam as mesmas operações que os acima citados mas sobre operandos representados em outros padrões. Os mais comuns, depois do padrão IEEE, são os padrões IBM e DEC.

No caso dos Processadores Aritméticos Dedicados estudados, os circuitos multiplicadores não operam diretamente sobre números denormalizados. (Um número denormalizado é uma representação especial, estabelecida pelo padrão IEEE, para realizar underflow gradual nos números cuja magnitude seja menor que o menor número normalizado representável. Estes conceitos são abordados em maior detalhe no item 3.2.). Os circuitos oferecem duas posturas para este caso: os modos de operação FAST e IEEE. No modo FAST todos os operandos denormalizados são considerados como ZERO. Para aplicações que exigem total conformidade com o padrão IEEE o multiplicador ativa um flag externo quando algum dos operandos é denormalizado. Este flag é recebido pela ULA bem como o(s) operando(s) denormalizado(s) que o(s) converte(m) para um formato intermediário aceito pelo multiplicador. Após a execução da multiplicação (no formato intermediário) o resultado é enviado novamente para a ULA para ser convertido para o formato normal.

Na tabela 2.1 encontra-se um resumo dos formatos de representação de operandos suportados pelos processadores aritméticos integrados estudados.

Tabela 2.1 Formatos de representação numérica suportados pelos Processadores Aritméticos Integrados

PROC. ARITM.	*PONTO FIXO*	*PONTO FLUTUANTE*	*BCD*
INTEL 80387	compl. 2 16,32,64 b.	prec.simpl.(32b) prec.dupla (64b) prec.dpl.ext(80b)	18 dígit. decimais pt.fixo
MOTOROLA 68882	compl. 2 8,16,32 b.	prec.simpl.(32b) prec.dupla (64b) prec.dpl.ext(80b)	17 dígit. dec. mant 3 dígitos dec. exp.
WEITEK WTL 1232 WTL 1233		prec.simpl.(32b)	
ADSP3201 ADSP3202	compl.2 32b. magnit. 32b.	prec.simpl.(32b)	
WTL 2264 WTL 2265	compl.2 32b.	prec.simpl.(32b) prec.dupla (64b)	
ADSP3212 ADSP3222	compl.2 32b. magnit. 32b.	prec.simpl.(32b) prec.dupla (64b)	

2.3.2 Operações disponíveis

Segundo Hwang [HWA 79], as operações aritméticas podem ser classificadas em 3 categorias:

1) **operações aritméticas padrão**: adição, subtração, multiplicação e divisão.

2) **operações elementares**: operações trigonométricas diretas e inversas e operações logarítmicas e exponenciais.

3) **operações pseudoaritméticas**: comparação, operações lógicas, conversões de formato, etc.

De um modo geral os coprocessadores aritméticos de propósitos gerais realizam as operações aritméticas padrão e um conjunto considerável de operações elementares. Realizam também todas as conversões entre formatos de

representação suportados pelo circuito. A partir da análise do conjunto de instruções e dos formatos de representação verifica-se que os coprocessadores aritméticos de propósitos gerais possuem conformidade total com o padrão IEEE.

Já os processadores aritméticos dedicados estudados não apresentam todas as operações requeridas pelo padrão IEEE. Os processadores aritméticos dedicados executam as operações aritméticas padrão (que são as mais comuns em aplicações convencionais) e algumas operações pseudo-aritméticas.

Na tabela 2.2 encontramos listadas o repertório de operações disponíveis nos processadores aritméticos estudados.

2.3.3 Arquitetura interna

As diferentes abordagens que nortearam a implementação dos Processadores Aritméticos Integrados de cada categoria impuseram grandes diferenças arquiteturais entre as duas classe de processadores aritméticos.

Pode-se dizer que os coprocessadores aritméticos de propósitos gerais são bibliotecas de operações e funções aritméticas implementadas em hardware. Para tanto, apresentam unidades operativas mais ou menos genéricas e uma grande unidade de controle que a partir das poucas primitivas disponíveis na parte operativa executa algoritmos tipo CORDIC [VOL 80], [WAL 80]. Estes algoritmos implementam todas as operações aritméticas, trigonométricas e exponenciais através de somas e deslocamentos, possibilitando a implementação de uma parte operativa

relativamente simples e geral.

Por outro lado os processadores aritméticos dedicados, que implementam apenas algumas operações, têm sua arquitetura caracterizada por operadores dedicados àquelas operações visando aumento do desempenho.

Tabela 2.2 Operações disponíveis nos Processadores Aritméticos Integrados

PROC. ARITM.	*OPERAÇÕES ARITMÉTICAS BÁSICAS*	*OPERAÇÕES ELEMENTARES*	*OPERAÇÕES PSEUDO-ARITMET.*
INTEL 80387	adição subtração multipli. divisão	raiz quad, sen, cos, sen e cos, arctg, 2**x-1, x.log(y), x.log(y-1) e resto divisão	conversão entre formatos suportad.
MOTOROLA 68882	adição subtração multipli. divisão	raiz quad, sen, cos, tg, sen e cos arcsen, arccos, arctg, senh, cosh tgh, exp(x), exp(x-1), log(x), 10**x, 2**x.	conversão entre formatos suportad.
WEITEK WTL1232 WTL1233	adição subtração multiplicação		conversão pt fixo(32) <-> pt flut (32)
ADSP3201 ADSP3202	adição subtração multipli.	raiz quad.	conversão fix<->flu comparação op. lóg.
WEITEK WTL 2264 WTL 2265	adição subtração multipli. divisão		conversão fix<->flu flu<->flu comparação op. lóg.
ADSP3221 ADSP3222	adição subtração multipli. divisão	raiz quad.	conversão fix<->flu flu<->flu comparação op. lóg.

a) Coprocessadores Aritméticos de Propósitos Gerais

Esta linha de coprocessadores compartilha aproximadamente a mesma arquitetura interna. São constituídos por uma unidade responsável pela interface de comunicação entre o microprocessador e o coprocessador aritmético. Um banco de registradores (8 registradores de 80 bits em ambos os coprocessadores analisados) para armazenamento de dados e resultados (que aparecem ao programador como se estivessem no microprocessador) e a unidade de processamento aritmético onde são executadas as operações.

Os coprocessadores aritméticos realizam um vasto repertório de operações, que também aparecem ao programador como instruções do microprocessador graças ao protocolo de comunicação embutido em ambos os circuitos e transparente ao usuário. Estas operações são executadas através de algoritmos tipo CORDIC. Estes algoritmos utilizam apenas adições e deslocamentos como primitivas. Logo, os coprocessadores aritméticos possuem unidades operativas relativamente simples mas uma grande unidade de controle para armazenar estes algoritmos.

Na figura 2.1 é mostrada a arquitetura interna do coprocessador aritmético 68882 da Motorola.

b) Processadores Aritméticos Dedicados

Com o objetivo de obter excelente desempenho nas operações aritméticas mais utilizadas, os Processadores Aritméticos Dedicados divergem completamente da abordagem empregada nos coprocessadores aritméticos de propósitos gerais. Sendo implementados com operadores dedicados às

operações (ex.: multiplicadores em array), a parte operativa em si incorpora grande parte dos algoritmos, reduzindo a parte controle.

Figura 2.1 Arquitetura interna do coprocessador aritmético 68882 da Motorola [MOT 87].

Os Processadores Aritméticos Dedicados utilizam extensivamente arquiteturas tipo pipeline. As operações mais comuns (para as quais tais circuitos são mais adaptados) podem ser "*pipelinezadas*". Outras operações tais como divisão e raiz quadrada utilizam durante vários ciclos de relógio a parte operativa não são "*pipelinezáveis*".

A utilização do pipeline confere a tais circuitos desempenho elevadíssimo, atingindo dezenas de MFLOPS (Millions of Floating-Point Operations per Second). Esta

característica é especialmente interessante para aplicações de tratamento de sinais e imagens.

Na figura 2.2 é mostrada a arquitetura interna do circuito processador aritmético dedicado ADSP 3202. Trata-se de uma ULA pipeline de três estágios para operandos de ponto flutuante 32 bits.

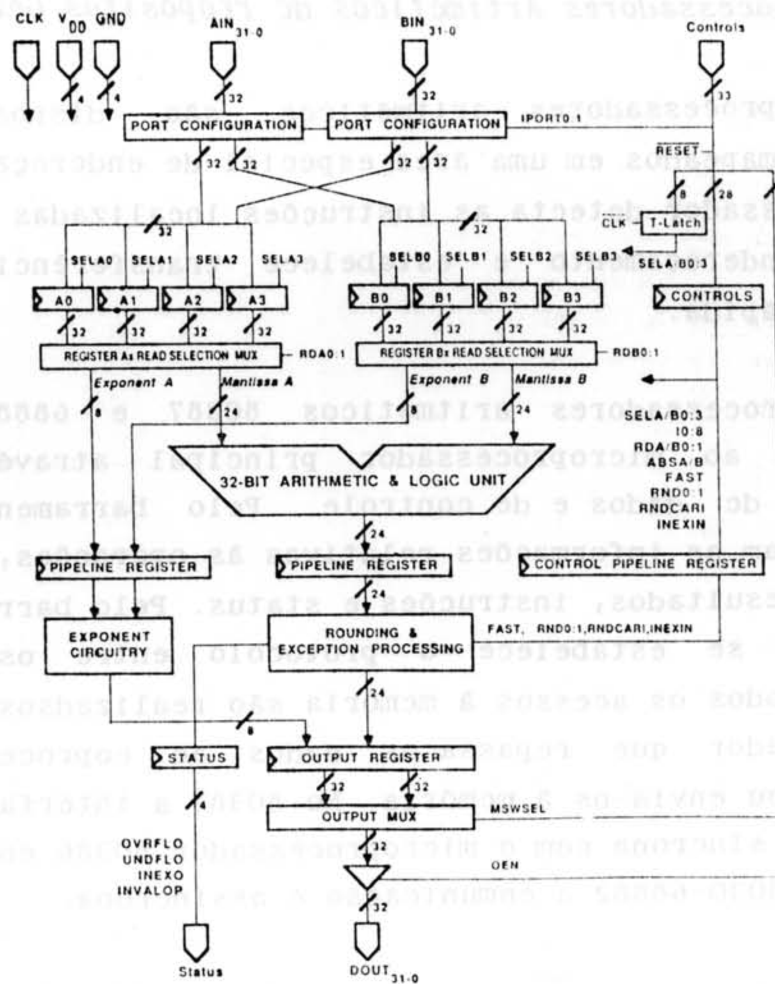


Figura 2.2 Arquitetura interna do Processador Aritmético Dedicado ADSP 3202 [ANA 87].

2.3.4 Entrada e saída

A estratégia de entrada e saída de informações (operandos, resultados, instruções, status, etc.) é de vital importância para o desempenho global do sistema. Uma boa interligação entre microprocessador e o processador aritmético é importante para que os barramentos não tornem-se gargalos, reduzindo o desempenho.

a) Coprocessadores Aritméticos de Propósitos Gerais

Os coprocessadores aritméticos são dispositivos periféricos mapeados em uma área especial de endereçamento. O microprocessador detecta as instruções localizadas nesta área de endereçamento e estabelece transferência de informação rápida.

Os coprocessadores aritméticos 80387 e 68882 são interligados ao microprocessador principal através dos barramentos de dados e de controle. Pelo barramento de dados circulam as informações relativas às operações, i.e., operandos, resultados, instruções e status. Pelo barramento de controle se estabelece o protocolo entre os dois circuitos. Todos os acessos à memória são realizadsos pelo microprocessador que repassa os dados ao coprocessador aritmético ou envia-os à memória. No 80387 a interface de barramento é síncrona com o microprocessador 80386 enquanto no sistema 68030-68882 a comunicação é assíncrona.

b) Processadores Aritméticos Dedicados

Apresentam hardware estremamente dedicado a algumas operações mas não possuem interface adaptada. Apresentando, geralmente, duas portas de entrada (para entrada paralela

de dois operandos), uma porta de saída para os resultados e entrada independente para as instruções. Este esquema é necessário para pleno aproveitamento do pipeline.

O protocolo de comunicação entre o Processador Aritmético Dedicado e o processador principal deve ser particularizado de acordo com o sistema. No entanto, a interface oferece bastante flexibilidade. Uma das abordagens possíveis é o mapeamento em memória (empregada na placa coprocessadora, para o microprocessador 80386, Weitek 1167 [BON 87]). Esta técnica mapeia o processador aritmético em área de endereçamento de memória.

2.3.5 Desempenho

O objetivo principal dos processadores aritméticos integrados é aumentar a capacidade de processamento numérico, sobretudo em ponto flutuante, de sistemas de computação de pequeno e médio porte ou fornecer desempenho compatível com as necessidades de aplicações mais específicas.

A eficiência de um sistema com Processadores Aritméticos recai sobre dois aspectos principais:

a) desempenho do Processador Aritmético propriamente dito
b) eficiência da comunicação entre o processador aritmético e o processador principal.

a) Coprocessadores Aritméticos do Propósitos Gerais

Oferecem, além das operações aritméticas básicas, um repertório de operações transcedentais e total conformidade (em todos os aspectos) com o padrão IEEE [IEE 87]. Isto é conseguido as expensas de uma unidade operativa relativamente genérica que executa através de microprograma algoritmos do tipo CORDIC consumindo vários ciclos de máquina por instrução.

A comunicação entre o microprocessador e o coprocessador aritmético se faz principalmente pelo barramento de dados. Por este barramento circulam operandos, resultados, instruções e palavras de status utilizando-o excessivamente.

Um protocolo dedicado entre o coprocessador aritmético e o microprocessador acelera ao máximo tais transferências evitando degradação acentuada do desempenho global. Um relatório detalhado sobre o desempenho da família 80386/80387 pode ser encontrado em [INT 89a].

b) Processadores Aritméticos Dedicados

Por implementarem poucas operações (sobretudo as aritméticas básicas) atingem desempenhos bastante superiores aos dos Coprocessadores Aritméticos de Propósitos Gerais apresentando tempos de latência de operação da ordem de alguns ciclos de relógio. Deve-se esta característica de desempenho aos operadores dedicados que fazem uso extensivo do paralelismo possível nos algoritmos. Os Processadores Aritméticos Dedicados atingem elevados "trhoughputs" quando a taxa de entrada de operandos permite plena utilização da arquitetura pipeline.

É difícil avaliar o desempenho da estrutura de entrada e saída destes processadores. Originalmente tais circuitos apresentam características de entrada e saída que permitem utilizar ao máximo suas potencialidades explorando várias configurações. No entanto a particularização da interface do circuito com o sistema pode alterar completamente estas características.

Nas tabelas 2.3 a 2.6 encontra-se uma comparação do desempenho das operações de adição/subtração, multiplicação, divisão e raiz quadrada entre os coprocessadores aritméticos de propósitos gerais e os processadores aritméticos dedicados.

Tabela 2.3 Desempenho dos Processadores Aritméticos nas operações de Adição e Subtração em ponto flutuante.

ADIÇÃO E SUBTRAÇÃO PONTO FLUTUANTE

	proces. aritmét.	*no. ciclos latência*	*frequência operação*	*tempo latência*
GENÉRICOS	80387	24-32 (32bits) 29-37 (64bits)	20MHz	1.2-1.6us (32b) 1.5-1.8us (64b)
GENÉRICOS	68882	69 (32bits) 75 (64bits)	25MHz	2.76us (32bits) 3us (64bits)
DEDICADOS	WTL1233	9 (32bits)*	20MHz	450ns (32bits)
DEDICADOS	ADSP3201	3 (32bits)**	25MHz	120ns (32bits)
DEDICADOS	WTL2265	4 (32bits)** 5 (64bits)*	20MHz	200ns (32bits) 250ns (64bits)
DEDICADOS	ADSP3222	3 (32bits)** 3 (64bits)**	20MHz	150ns (32bits) 150ns (64bits)

* - Um resultado a cada dois ciclos (pipeline)

** - Um resultado a cada ciclo (pipeline)

Tabela 2.4 Desempenho dos Processadores Aritméticos na operação de Multiplicação em ponto flutuante.

MULTIPLICAÇAO PONTO FLUTUANTE

	proces. aritmét.	no. ciclos latência	freqüência operação	tempo latência
GENÉRICOS	80387	27-35 (32bits) 32-57 (64bits)	20MHz	1.3-1.8us (32b) 1.6-2.8us (64b)
	68882	89 (32bits) 95 (64bits)	25MHz	3.56us (32bits) 3.8us (64bits)
DEDICADOS	WTL1232	9 (32bits)*	20MHz	450ns (32bits)
	ADSP3202	3 (32bits)**	25MHz	120ns (32bits)
	WTL2264	4 (32bits)** 6 (64bits)*	20MHz	200ns (32bits) 300ns (64bits)
	ADSP3221	3 (32bits)** 3 (64bits)**	20MHz	150ns (32bits) 150ns (64bits)

* - Um resultado a cada dois ciclos (pipeline)

** - Um resultado a cada ciclo (pipeline)

Tabela 2.5 Desempenho dos Processadores Aritméticos na operação de Divisão em ponto flutuante

DIVISAO PONTO FLUTUANTE				
	proces. aritmét.	**no. ciclos latência**	**freqüência operação**	**tempo latência**
GENERICOS	80387	89 (32bits) 94 (64bits)	20MHz	4.45us (32bits) 4.7us (64bits)
	68882	121 (32bits) 127 (64bits)	25MHz	4.84us (32bits) 5.08us (64bits)
DEDICADOS	ADSP3202	16 (32bits)	25MHz	720ns (32bits)
	WTL2264	13 (32bits) 23 (64bits)	20MHz	650ns (32bits) 1.15us (64b)
	ADSP3221	8 (32bits) 14 (64bits)	20MHz	400ns (32bits) 700ns (64bits)
	ADSP3222	18 (32bits) 32 (64bits)	20MHz	850ns (32bits) 1.6us (64bits)

Obs.: O chip-set WTL 1232/1233 não realiza a operação de divisão.

Tabela 2.6 Desempenho dos Processadores Aritméticos na operação de Raiz Quadrada

	RAIZ QUADRADA PONTO FLUTUANTE			
	proces. aritmét.	***no. ciclos latência***	***freqüência operação***	***tempo latência***
GENERICOS	80387	122-129 (32bits) & (64bits)	20MHz	6.1-6.5us (32b) & (64bits)
	68882	123 (32bits) 129 (64bits)	25MHz	4.92us (32bits) 5.16us (64bits)
DEDICADOS	WTL1232/ 1233	-------	------	-------
	ADSP3202	31 (32bits)	25MHz	1.24us (32bits)
	WTL2264/ 2265	-------	------	-------
	ADSP3222	31 (32bits) 60 (64bits)	20MHz	1.550us (32b) 3us (64bits)

2.4 Conclusões e tendências

Os coprocessadores aritméticos de propósitos gerais são circuitos mais genéricos, interessantes em aplicações onde é desejável diversidade de instruções e desempenho moderado.

São mais baratos relativamente aos processadores

aritméticos dedicados sob dois aspectos:

- custo da pastilha em si;
- adaptabilidade ao sistema: podem ser conectados quase diretamente ao sistema; possuem suporte para programação; interface e protocolos transparentes ao usuário e otimizados.

Os processadores aritméticos dedicados são circuitos para aplicações onde o desempenho é a questão principal (sistemas de tratamento de sinais e imagem, estações de trabalho, etc.). São mais caros e exigem desenvolvimento de interface para adaptação ao sistema.

Os mais recentes microprocessadores tais como Motorola RISC 88000 [MOT 88], Motorola 68040 [WIL 89] e Intel 80486 [INT 89], [WIL 89a], [BUR 89] apresentam unidade para processamento aritmético de operandos em ponto flutuante. Isto faz com que as operações aritméticas básicas envolvendo este tipo de operandos sejam realizadas no próprio microprocessador diminiduindo os atrasos com comunicação externa. Observa-se a mesma característica nos mais recentes microprocessadores para DSP [TEX 89] e processadores gráficos [CAS 89].

Observa-se, também, uma forte tendência ao desenvolvimento de circuitos que implementam algoritmos paralelos dedicados [GUN 88] do tipo processadores array e circuitos sistólicos [DRA 88], [BUR 88]. Tais circuitos constituem uma saída alternativa para aplicações onde desempenho elevado é imprescindível.

3 ALGORITMOS DAS OPERAÇÕES DE ADIÇÃO, SUBTRAÇÃO E MULTIPLICAÇÃO EM PONTO FLUTUANTE

Neste capítulo são estudados os algoritmos das operações aritméticas de adição, subtração e multiplicação envolvendo operandos representados em ponto flutuante.

O estudo inicia-se pela apresentação do padrão de representação numérica em ponto flutuante, o Padrão IEEE para aritmética binária de ponto flutuante [IEE 87], haja visto a grande dependência, apresentada pelos algoritmos das operações, com relação ao formato de representação numérica.

Segue-se, então, o estudo dos algoritmos das operações de adição, subtração e multiplicação para operandos representados no formato acima citado. Os algoritmos gerais das operações são seguidos de uma análise detalhada das sub-operações envolvidas em cada etapa.

3.1 Formatos de representação numérica em sistemas de computação convencionais

A implementação de algoritmos aritméticos em hardware depende muito de como os dados - operandos e resultados - estão organizados na memória e nos registradores, i.e., diferentes representações numéricas conduzirão a diferentes implementações em hardware.

Na busca de representações mais eficientes ou mais adaptadas a aplicações específicas, foram sugeridos vários sistemas numéricos com diferentes bases, implicando, evidentemente, em diferentes aritméticas.

Além do sistema binário, o mais difundido, o sistema decimal (de base 10), codificado em binário (BCD), ainda é bastante empregado, principalmente em aplicações comerciais e financeiras.

Outros sistemas de representação, com base diferente da binária, não encontram grande aceitação junto às aplicações convencionais mas podem ser interessantes em casos específicos. Um exemplo seriam os códigos ternários utilizados em modems de banda base.

A aritmética descrita e apresentada ao longo deste trabalho é exclusivamente binária (base 2).

As representações numéricas, independentemente da base, para processamento numérico convencional recaem em duas categorias:

- **representações numéricas em ponto fixo**
- **representações numéricas em ponto flutuante**

3.1.1 Representações numéricas binárias em ponto fixo

Antes de iniciar a apresentação das representações numéricas binárias é importante estabelecer a nomenclatura a ser empregada. Neste estudo o sinal separador da parte inteira da parte fracionária de um número será denominado ponto ou ponto binário, e não vírgula, objetivando manter conformidade com a notação empregada na literatura técnica de língua inglesa, de uso mais corrente.

Os números representados em ponto fixo são normalmente considerados como inteiros. Entretanto este subentendimento não é regra obrigatória. Teoricamente, como o ponto binário é implícito, ele pode ocupar qualquer posição entre dois

dígitos adjacentes na representação.

Existem duas convenções, para posição implícita do ponto binário, mais comuns nas representações numéricas em ponto fixo:

a) considerá-los todos como inteiros; neste caso o ponto binário localiza-se imediatamente à direita do bit menos significativo.

b) considerá-los todos como frações; o ponto binário localiza-se, então, imediatamente à direita do bit mais significativo. Isto porque, geralmente, o bit mais significativo denota o sinal, como veremos adiante.

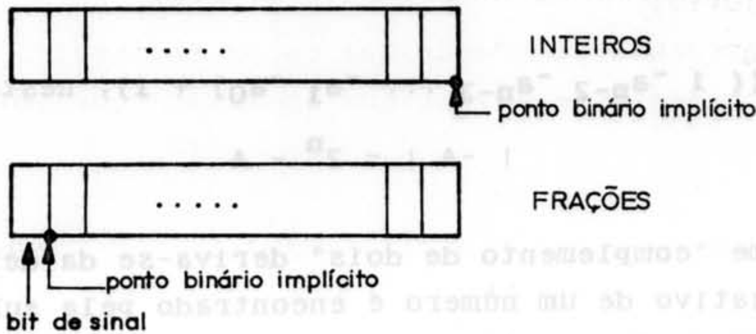


Figura 3.1 Convenções para posição do ponto binário nas representações em ponto fixo

Considere o número binário A de n bits:

$$A = (a_{n-1}\, a_{n-2} \cdots a_1\, a_0),$$

onde o dígito do sinal, a_{n-1}, assume os seguintes valores:

$$a_{n-1} = 0, \text{ se } A >= 0$$
$$a_{n-1} = 1, \text{ se } A < 0$$

Para números positivos em ponto fixo, o dígito de sinal é igual a zero e os demais dígitos representam a

magnitude real do número, que é computada pela seguinte fórmula:

$$A = \sum_{i=0}^{n-2} a_i . 2^i$$

Existem três diferentes notações para números negativos em ponto fixo [HWA 79]: a representação em complemento de dois, em complemento de um e em sinal-magnitude.

Seja, então, -A a versão negativa do número positivo A definido acima. Logo, -A tem o dígito de sinal com valor 1. As três representações distintas de números negativos em ponto fixo são apresentadas abaixo, onde ~ simboliza o operador inversor lógico.

Representação em Complemento de Dois

$$-A = ((1\ \tilde{}a_{n-2}\ \tilde{}a_{n-3} \cdots \tilde{}a_1\ \tilde{}a_0) + 1), \text{ neste caso}$$

$$|-A| = 2^n - A$$

O nome "complemento de dois" deriva-se da definição na qual o negativo de um número é encontrado pela subtração de 2^n onde n é o número de bits da palavra.

Neste sistema de representação há um único formato para representar o zero, que é justamente todos os dígitos iguais a zero. Como todos os números que possuem o MSB (Most Significant Bit) igual a zero são considerados positivos haverá um número negativo a mais do que há de positivos.

As operações de adição e subtração são realizadas diretamente [CAV 85], [GOS 80], [HWA 79], bem como a multiplicação. A divisão apresenta algumas dificuldades

comparativamente às representações seguintes.

Representação em Complemento de Um

$$-A = (1\ \tilde{a}_{n-2}\ \tilde{a}_{n-3}\ \ldots\ \tilde{a}_1\ \tilde{a}_0), \text{ neste caso}$$

$$|-A| = 2^n - 1 - A$$

Logo, para encontrar o negativo de um número em complemento de um basta inverter logicamente todos os bits.

Há, portanto, duas representações de zero, o que implica numa simetria dos números representáveis.

A implementação de operadores de adição e multiplicação é mais complexa que em complemento de dois devido ao end-around-carry (EAC) [GOS 80] e [HWA 79].

A multiplicação é relativamente fácil se forem multiplicados os módulos dos fatores e realizada a correção do sinal ao final da operação; o mesmo vale para a divisão.

Representação em Sinal-Magnitude

$-A = (1\ a_{n-2}\ a_{n-3}\ \ldots\ a_1\ a_0)$, os dígitos apresentam a magnitude real do número. A versão positiva **A** do número difere da negativa **-A** apenas pelo dígito de sinal.

Há, também, duas representações de zero, e, portanto, para todo o valor representável x existe o correspondente valor -x, também representável.

A adição e subtração são de difícil implementação nesta notação. Este estudo é deixado para os próximos itens quando serão abordados em detalhe os algoritmos e implementações das operações em ponto flutuante cuja representação da mantissa é em sinal-magnitude. As operações de multiplicação e divisão são realizadas diretamente.

A tabela 3.1 sintetiza as características principais dos sistemas de representação numéricos em ponto fixo em relação às representações dos números negativos.

Tabela 3.1 Comparação dos sistemas de representação numérica em ponto fixo [GOS 80]

Característica	Sinal-Magnitude	Compl.Um	Compl.Dois
Zero	000...00 100...00	00...00 11...11	00...00
Limites de representação	$-2^{n-1}-1$ a $2^{n-1}-1$	$-2^{n-1}-1$ a $2^{n-1}-1$	-2^{n-1} a $2^{n-1}-1$
Adição e Subtração	Complexa	EAC	Direta
Multiplicação	Direta	alguma dificuldade	pequenas correções
Divisão	Direta	Direta	alguma dificuldade
Negação	Trivial	Trivial	requer adição completa

3.1.2 Representações numéricas binárias em ponto flutuante

Muitas aplicações requerem intervalos numéricos dinâmicos muito maiores do que aqueles que a representação em ponto fixo pode oferecer.

A utilização de fatores de escala, que fazem com que as grandezas envolvidas na computação sejam situadas dentro do intervalo representável, não constitui uma solução satisfatória.

Entre as desvantagens deste expediente está a escolha do fator de escala conveniente, nem sempre simples, e as conversões e desconversões necessárias, antes e após a computação.

Ainda, este método aplica-se somente em casos onde os intervalos dinâmicos não são muito grandes pois as aplicações do fator de escala podem conduzir a grandes perdas de precisão nos valores próximos ao limite inferior da escala, quando então a maioria dos dígitos são nulos.

A necessidade de uma representação que apresentasse características de precisão e intervalo dinâmico, operando segura e eficientemente resultou na proposta de uma aritmética binária de ponto flutuante no início dos anos 40 [HWA 79]. Na tabela 3.2 encontramos uma comparação entre os intervalos dinâmicos correspondentes a cada formato de representação.

Tabela 3.2 Comparação entre intervalos dinâmicos e precisão nas representações ponto fixo e flutuante

	32 bits		64 bits	
Formato	pt.fix.	pt.flut.	pt.fix	pt.flut.
Intervalo Dinâmico	0 a $4.3x10^{9}$	$1.2x10^{-38}$ a $3.4x10^{38}$	0 a $1.8x10^{19}$	$2.2x10^{-308}$ a $1.8x10^{307}$
Precisão	32 bits	24 bits	64 bits	53 bits

A representação em ponto flutuante é constituída de duas partes: a mantissa ou significando (f) e o expoente (e). As duas partes representam um número que é obtido multiplicando a mantissa pela raiz (r) elevada a potência do expoente.

$$A = f \times r^{e}$$

Inicialmente, para representar os números em ponto flutuante cada fabricante de grandes computadores definiu seu próprio padrão de representação, surgindo, assim, padrões como o IBM e DEC.

Visando padronizar os formatos e os aspectos aritméricos da representação em ponto flutuante em pequenos e médios sistemas de computação e estimular o desenvolvimento de sistemas de processamento numérico de alta qualidade e fácil portabilidade, o Instute of Electrical and Electronic Engineers, IEEE, estabeleceu um padrão para esta forma de aritmética binária, parcialmente descrito no item seguinte.

3.2 Padrão IEEE para aritmética binária de ponto flutuante

O padrão IEEE especifica:

(1) Formatos de números ponto flutuante básicos e extendidos.
(2) Operações de adição, subtração, multiplicação, divisão, raiz quadrada, resto da divisão e comparação.
(3) Conversões entre formatos inteiro e ponto flutuante.
(4) Conversões entre diferentes formatos ponto flutuante.
(5) Conversões entre os formatos básicos em ponto flutuante e strings decimais.
(6) Exceções em ponto flutuante e suas manipulações, incluindo "não-números" (NaN).

Transcreve-se, à seguir, somente as informações consideradas relevantes à compreenção deste trabalho. Maiores informações e detalhes podem ser obtidos endereçando-se diretamente ao Padrão IEEE para Aritmética Binária de Ponto Flutuante [IEE 87].

3.2.1 Definições

expoente polarizado. A soma do expoente e uma constante (polarização) escolhida de forma a tornar os valores do expoente polarizado não negativos.

número binário de ponto flutuante (binary floating-point number). Um string de bits caracterizado por três componentes: um sinal, um expoente com sinal e um significando. Seu valor numérico, se houver, é o produto sinalizado entre seu significando e dois elevado ao expoente.

número denormalizado. Um número ponto flutuante não nulo cujo expoente tem um valor reservado, geralmente o valor mínimo do formato, e cujo "leading" bit do significando, explícito ou implícito, é zero.

expoente. O componente de um número binário de ponto flutuante que normalmente significa a potência inteira a qual dois é elevado na determinação do número representado.

fração. O campo do significando que está situado à direita de seu ponto binário implícito.

NaN. (Not a Number) Não número; uma entidade simbólica codificada em um formato de ponto flutuante. Existem dois tipos de NaNs. NaN sinalizados (signaling NaNs) sinalizam a exceção de operação inválida sempre que aparecem como operandos. NaN quietos (quiet NaN) propagam-se através de quase todas as operações aritméticas sem sinalizar exceções.

significando. O componente de um número binário de ponto flutuante que consiste de um "leading" bit explícito ou implícito à esquerda de seu ponto binário implícito e um campo de fração à direita. Neste estudo, significando e mantissa são considerados sinônimos.

3.2.2 FORMATOS

O padrão IEEE define quatro formatos de representação em ponto flutuante:

1) **grupo básico** (simples e duplo) e
2) **grupo extendido** (simples e duplo).

3.2.2.1 CONJUNTO DE VALORES

Os únicos valores representáveis em um dado formato são aqueles especificados por meio dos seguintes três parâmetros inteiros:

p = número de bits significativos (precisão)

$E_{máx}$ = expoente máximo

$E_{mín}$ = expoente mínimo

Os parâmetros de cada formato são dados na tabela 3.3.

Tabela 3.3 Sumário dos Parâmetros dos Formatos [IEE 87].

	Formato			
Parâmetro	**Simples**	**Simples Extendido**	**Duplo**	**Duplo Extendido**
p	24	>=32	53	>=64
$E_{máx}$	+127	>=+1023	1023	>=+16383
$E_{mín}$	-126	<=-1022	-1022	<=-16382
Polariz. Expoente	+127	não especif.	+1023	não espec.
Largura Exp. (bits)	8	>=11	11	>=15
Largura Formato	32	>=43	64	>=79

Dentro do formato somente as seguintes entidades podem ser fornecidas:

Números da forma $(-1)^s 2^E (b_0 . b_1 b_2 ... b_{p-1})$ onde

s = 0 ou 1

E = algum inteiro entre $E_{mín}$ e $E_{máx}$, inclusive

b_i = 0 ou 1

Dois infinitos, +infinito e -infinito

Pelo menos um NaN sinalizador (signaling NaN)

Pelo menos um NaN quieto (quiet NaN)

3.2.2.2 Formatos básicos

Números nos formatos simples e duplo são compostos dos seguintes três campos:

(1) 1 bit de sinal s

(2) Expoente polarizado $e = E$ + *polarização*

(3) Fração $f = .\ b_1 b_2 \ldots b_{p-1}$

O intervalo do expoente não polarizado deve incluir todos os inteiros entre os dois valores E_{min} e E_{max}, inclusive, e também dois outros valores reservados $E_{min} - 1$ para codificar +0, -0 e números **denormalizados**, e $E_{max} + 1$ para codificar **+Infinito**, **-Infinito** e **NaNs**. Os campos são interpretados como segue:

a) FORMATO BASICO SIMPLES

Um número X em formato simples 32 bits é divido como mostrado na figura 3.2. O valor v de X é inferido a partir de seus campos constituintes desta forma:

(1) Se $e = 255$ e $f <> 0$, então v é **NaN** qualquer que seja s.

(2) Se $e = 255$ e $f = 0$, então $v = (-1)^s$**Infinito**

(3) Se $0 < e < 255$, então $v = (-1)^s 2^{e-127}(1.f)$ (números normalizados)

(4) Se $e = 0$ e $f <> 0$, então $v = (-1)^s 2^{-126}(0.f)$ (números denormalizados)

(5) Se $e = 0$ e $f = 0$, então $v = (-1)^s 0$ (zero)

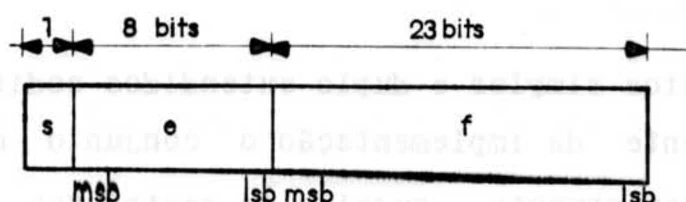


Figura 3.2 Formato Básico Simples (32 bits) [IEE 87]

b) FORMATO BÁSICO DUPLO

Um número *X* em formato duplo 64 bits é divido como mostrado na figura 3.3. O valor *v* de *X* é inferido a partir de seus campos constituintes desta forma:

(1) Se e = 2047 e f<>0, então v é NaN qualquer que seja s.

(2) Se e = 2047 e f=0, então $v = (-1)^s$ Infinito

(3) Se 0<e<2047, então $v = (-1)^s 2^{e-1023}(1.f)$ (números normalizados)

(4) Se e = 0 e f<>0, então $v = (-1)^s 2^{-1022}(0.f)$ (números denormalizados)

(5) Se e = 0 e f = 0, então $v = (-1)^s 0$ (zero)

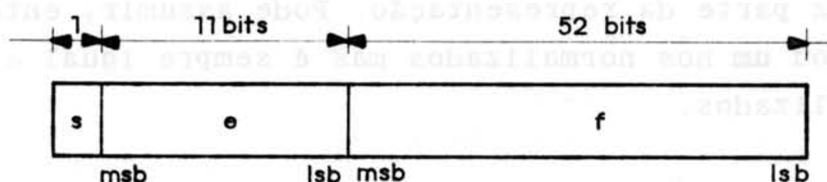


Figura 3.3 Formato Básico Duplo (64 bits) [IEE 87]

3.2.2.3 Formatos extendidos

Os formatos simples e duplo extendidos codificam de um modo dependente da implementação o conjunto de valores descrito anteriormente, sujeito às restrições da tabela 3.1. Este padrão permite a uma implementação codificar alguns valores redundantemente, desde que a redundância seja transparente ao usuário no seguinte sentido: uma implementação ou deve codificar todos os valores diferentes de zero unicamente ou não deve distinguir codificações redundantes de valores diferente de zero. Uma implementação pode também reservar alguns bits para propósitos além do escopo deste padrão.

Observações sobre os formatos de representação do Padrão IEEE

a) o '*leading*' bit

No formato básico simples e duplo, o "*leading*" bit (implícito), ou seja, o MSB do significando, que também corresponde à parte inteira do número representado, é sempre igual a **um** nos números normalizados e igual a **zero** nos números denormalizados.

Nos formatos extendidos o "*leading*" bit é explícito, ou seja, faz parte da representação. Pode assumir, então, o valor **zero** ou **um** nos normalizados mas é sempre igual a zero nos denormalizados.

b) os números denormalizados

Um número denormalizado é um número cuja magnitude é menor que o menor número normalizado representável no formato. Denormalizados possuem o campo do expoente igual a zero e a fração sempre diferente de zero. O "*leading*" bit implícito (ou explícito, no caso dos formatos extendidos) é

sempre igual a zero. O valor real do expoente é (-126) no formato básico simples e (-1022) no formato básico duplo. O padrão IEEE exige suporte para números denormalizados para realizar underflow gradual nos números com magnitude inferior ao menor normalizado. Detalhes sobre números denormalizados e underflow gradual podem ser encontrados em [COO 81].

c) expoente polarizado

Os expoentes no padrão de representação numérica em ponto flutuante do IEEE são representados polarizados, i.e., somados a uma constante dita de polarização (bias), de forma que todos os expoentes representados sejam sempre positivos. Esta técnica é também referida na literatura como representação em excesso [GOS 81].

Existem duas razões principais para usar expoente polarizados [MAN 88]:

1) o fato de que todos os expoentes são positivos fornece simplificação em operações de comparação entre expoentes (necessária na adição e subtração, conforme será visto nos itens posteriores), pois tal operação entre magnitudes é mais simples que entre números com sinal.

2) o outro aspecto diz respeito à representação do valor zero. No formato ponto flutuante com expoentes não polarizados, o menor expoente é o expoente mais negativo, cuja representação é diferente de zero. Com expoentes polarizados, o menor expoente representado é conseguido com todos os bits do expoente iguais a zero, desta forma pode-se representar o zero "limpo", ou seja, a representação do zero corresponde a todos os bits iguais a zero, e ainda corresponde ao menor expoente.

3.2.3 Arredondamento

O Arredondamento toma um número visto como infinitamente preciso e, se necessário, modifica-o para colocá-lo no formato destino sinalizando exceção de resultado inexato.

Exceto para conversões binário<->decimal, toda operação especificada neste padrão deve ser realizada como se primeiramente fosse produzido um resultado intermediário correto em precisão infinita e sem limite e então arredondar este resultado de acordo com um dos modos de arredondamento descritos a seguir. Os modos de arredondamento afetam todas as operações aritméticas com exceção da comparação e do resto da divisão.

a) Arredondamento para o mais próximo (Round to nearest)

Neste modo, o valor representável mais próximo do resultado infinitamente preciso deve ser tomado; se as duas representações mais próximas são igualmente próximas, aquela com o bit menos significativo igual a zero deve ser a escolhida. No entanto, um resultado infinitamente preciso com magnitude de no mínimo $2^{Emáx}(2-2^{-p})$ deve ser arredondado para infinito sem mudança do sinal; aqui $E_{máx}$ e p são determinados pelo formato destino.

b) Arredondamento para Zero (Round to Zero)

Quando arredondando para zero, o resultado deve ser o mais próximo do formato e não maior em magnitude que o resultado infinitamente preciso.

c) Arredondamento para + Infinito (Round toward + infinity)

Quando arredondando para + Infinito, o resultado deve ser o valor no formato representável (possivelmente +Infinito) mais próximo e não menor que o infinitamente preciso resultado.

d) Arredondamento para -Infinito (Round toward - infinity)

Quando arredondando para -Infinito, o resultado deve ser o valor no formato (possivelmente -Infinito) mais próximo e não maior que o infinitamente preciso resultado.

3.2.4 Exceções

a) Operação Inválida.

A exceção de operação inválida é sinalizada se um operando é inválido para a operação que está sendo executada. O resultado deve ser um NaN quieto. As operações inválidas são:

(1) Qualquer operação sobre NaN sinalizados;

(2) Adição ou Subtração; subtração de magnitudes infinitas tais como (+infinito)+(-infinito).

(3) Multiplicação; (0 x infinito)

(4) Divisão; (0/0) ou (Infinito/Infinito)

(5) Resto da divisão; (x REM y), quando y é zero ou x é infinito.

(6) Raiz quadrada quando o operando é menor do que zero.

b) Overflow.

A exceção de overflow deve ser sinalizada sempre que o maior número finito do formato destino é excedido em

magnitude por aquele que teria sido o resultado em ponto flutuante arredondado sendo o expoente ilimitado. O resultado deve ser determinado pelo modo de arredondamento e o sinal do resultado intermediário como segue:

(1) Arredondamento para o mais próximo conduz todos os overflows para infinito com o sinal do resultado intermediário.

(2) Arredondamento para Zero conduz todos os overflows para o maior número finito do formato com o sinal do resultado intermediário.

(3) Arredondamento para -Infinito conduz todos os overflows positivos para o maior número finito do formato e conduz os overflows negativos para -Infinito.

(4) Arredondamento para +Infinito conduz os overflows negativos para o número finito mais negativo do formato e conduz os overflows positivos para +Infinito.

c) Underflow

A exceção de underflow ocorre quando um resultado, diferente de zero, computado como se não houvesse limites para o expoente e para a precisão, localizar-se-ia entre $+2^{Emin}$ e -2^{Emin}.

d) Inexato.

Se o resultado arredondado de uma operação não é exato ou se ocorreu overflow então a exceção de inexato deve ser sinalizada.

e) Divisão por Zero.

Se o divisor é Zero e o dividendo é um número finito diferente de zero, então a divisão por Zero deve ser sinalizada. O resultado deve ser Infinito corretamente sinalizado.

3.3 Algoritmos das operações de adição, subtração e multiplicação em ponto flutuante

As operações de adição, subtração e multiplicação fazem parte do conjunto das operações aritméticas básicas. São as operações mais simples e as mais utilizadas na maioria das computações numéricas. São também a base do produto interno (a*b+c), operação básica dos sistemas DSP e operações matriciais.

Neste item apresenta-se os algoritmos destas operações envolvendo operandos de ponto flutuante representados segundo o padrão IEEE descrito anteriormente.

Antes de iniciar-se o estudo das operações, algumas restrições são feitas em relação à conformidade com o padrão IEEE, visando simplificar o estudo e as implementações, sem, no entanto, perdas significativas de generalidade.

Primeiramente, os números denormalizados, neste estudo, serão considerados aritmeticamente iguais a zero. Terão, no entanto, sua presença na operação detectada e sinalizada. A razão para este procedimento é que o tratamento dos denormalizados implica em grande complicação dos algoritmos e do hardware, sendo dispensável em muitas aplicações onde este nível de refinamento é desnecessário.

Considera-se neste estudo apenas o modo de arredondamento para o mais próximo, pois é o modo mais empregado nas aplicações convencionais (apesar de mais complexo, algoritmicamente).

Neste estudo toma-se como referência o formato básico simples (32 bits). Este procedimento é feito visando

facilitar as referências entre os algoritmos e as implementações nos capítulos seguintes. O estudo, entretanto, é facilmente extrapolado para o formato básico duplo (64 bits). Os formatos extendidos apresentam questões que encontram-se além das pretenções deste trabalho.

3.3.1 Adição e subtração binária em ponto flutuante

3.3.1.1 Introdução

Não há muito sentido em considerar somente adição ou somente subtração de números representados em ponto flutuante pois os operandos podem ser positivos ou negativos, logo a operação que efetivamente se realiza é caracterizada não só pela operação em si, mas também pelos sinais dos operandos envolvidos.

Sendo então:

$$A = f_a x\ r^{ea} \quad e$$

$$B = f_b x\ r^{eb},$$

onde f representa a mantissa (incluindo o sinal), r é a base e e é o expoente (incluindo o sinal), a adição ou subtração de A e B é calculada da seguinte forma:

$$A \pm B = (f_a \times r^{ea}) \pm (f_b \times r^{eb})$$

$$= [f_a \pm (f_b \times r^{-(ea-eb)})] \times r^{ea}, \text{ quando } ea > eb.$$

$$= [(f_a \times r^{-(eb-ea)}) \pm f_b] \times r^{eb}, \text{ quando } eb > ea.$$

$$= [f_a \pm f_b] \times r^{ea}, \text{ quando } ea = eb.$$

Observa-se, na definição acima, que o ajuste sobre os expoentes é feito na mantissa do operando que possui menor expoente. Em princípio, o ajuste pode ser feito em qualquer dos operandos, no entanto mostrar-se-á, mais tarde, ser

conveniente efetuá-lo sobre o operando com menor expoente.

Levando em conta o procedimento de cálculo acima e as características especificadas pelo padrão IEEE, podemos dividir uma implementação da operação de adição/subtração em ponto flutuante em cinco etapas:

a) Verificação dos operandos
b) Alinhamento das mantissas (ou equalização dos expoentes)
c) Adição/Subtração das mantissas
d) Renormalização e arredondamento
e) Tratamento de exceções

Antes de iniciar a operação é necessário verificar se os operandos são normalizados ou se um ou ambos apresentam valores reservados: NaN, Zero, Infinito ou Denormalizado.

Caso ambos os operandos sejam normalizados é realizado o alinhamento das mantissas ou a equalização dos expoentes antes de efetuar-se a operação sobre as mantissas, pois não se pode somar ou subtrair as mantissas diretamente tendo os expoentes valores diferentes.

Na situação em que um ou ambos os operandos não são normalizados o resultado é determinado pelo procedimento de tratamento de exceções.

A operação sobre as mantissas pode conduzir o resultado a um formato não normalizado sendo necessário uma renormalização. Também pode ocorrer do resultado apresentar precisão superior (número de bits) ao disponível no formato destino. É, então, necessário arredondar o resultado.

Um algoritmo preliminar que inclui as cinco etapas

acima citadas é mostrado abaixo. Nos próximos sub-itens analisa-se mais detalhadamente cada etapa.

```
Adição/Subtração_Ponto_Flutuante;
{
Verificação Operandos;
      SE Operandos_Normalizados
      ENTÃO
            {
            Alinhamento_das_Mantissas;
            Adição/Subtração_das_Mantissas;
            Renormalização_Arredondamento;
            }
      SENÃO
            Tratamento_de_Exceção;
}
```

3.3.1.2 Verificação dos operandos

Nesta etapa verifica-se se ambos os operandos são normalizados ou não. Em caso afirmativo, a operação é executada pelo procedimento normal, i.e., alinhando as mantissas, efetuando a adição ou subtração das mantissas, renormalizando e arredondando o resultado. Quando pelo menos um dos operandos for não normalizado a operação é desviada para o procedimento de tratamento de exceção.

Esta etapa pode ser executada verificando apenas se os expoentes dos operandos apresentam valores reservados, que são $(E_{máx}+1)$ e $(E_{min}-1)$ (respectivamente 255 e 0 no formato básico simples).

3.3.1.3 Alinhamento das mantissas

A etapa de alinhamento dos expoentes realiza a equivalência dos expoentes necessária antes da operação de adição ou subtração sobre as mantissas.

O alinhamento é obtido deslocando a mantissa do operando com menor expoente para a direita tantos bits quanto for a diferença entre os dois expoentes, pois cada deslocamento de 1 bit para a direita na mantissa equivale a incrementar de um o expoente. Poder-se-ia optar em deslocar a mantissa do operando com maior expoente para a esquerda, o que seria aritmeticamente equivalente, entretanto os bits mais significativos são mais importantes para a computação e os bits deslocados para fora do limite inferior do formato são os bits que serão posteriormente arredondados.

Antes de realizar a operação de alinhamento deve-se concatenar, a frente dos operandos, o "*leading*" bit de valor um, implícito na notação [IEE 87].

Sabe-se que o tamanho da mantissa é de 24 bits no formato básico simples do Padrão IEEE. No entanto as diferenças entre expoentes podem ser superiores a 24. Deslocamentos superiores a 24 bits fazem com que não haja interação efetiva entre as mantissas sendo o resultado afetado apenas pelo modo de arredondamento. Este fato sugere um tratamento abreviado quando a diferença entre os expoentes for superior ao número de bits da mantissa.

3.3.1.4 Adição ou subtração das mantissas

Após a etapa de alinhamento dos expoentes procede-se a adição ou subtração das mantissas. Observa-se que a mantissa é representada, no padrão IEEE, em sinal magnitude. Entretanto, por facilidade de implementação, os somadores operam geralmente em complemento de dois, o que exige conversão de formato de representação quando os operandos ou resultados são negativos.

3.3.1.5 Renormalização

A Adição/Subtração das mantissas pode resultar em uma representação não normalizada, i.e., em uma representação onde a parte inteira da mantissa seja diferente de um. É necessário, então, reconduzir o resultado novamente ao formato normalizado. A esta sub-operação denomina-se renormalização.

Porém, antes de iniciar-se a descrição da renormalização é necessário estabelecer o conceito de **operação efetiva** que será retomado no capítulo seguinte. Quando os sinais de cada operando associados ao sinal da operação resultar numa adição das magnitudes, independentemente do sinal do resultado, denomina-se adição efetiva. Quando, por outro lado, os sinais dos operandos associados ao sinal da operação resultar numa subtração das magnitudes, denomina-se subtração efetiva.

Podemos dividir a renormalização em dois casos:

a) Overflow da Mantissa

Quando a operação é **adição efetiva**, pode ocorrer um "carry-out" a partir do bit mais significativo do somador das mantissas, fazendo com que exista um bit com valor 1 imediatamente à esquerda do MSB, o que corresponde a uma parte inteira com dois bits.

A correção desta situação é feita deslocando toda a mantissa para a direita de 1 bit, de forma que o bit originado pelo carry-out do MSB ocupe, agora, a posição mais significativa na representação segundo o Padrão IEEE,

a posição do "*leading*" bit. Este deslocamento é compensado pelo incremento do expoente do resultado. Tal operação pode, no entanto, conduzir a um overflow do expoente. Neste caso o modo de arredondamento para o mais próximo estabelece que a magnitude do resultado deve ser infinita.

Exemplo:

```
           Mantissa              Expoente
           1 . 0 1 1 0            1 0 0
     +     1 . 0 1 1 1            0 1 1
     ---------------------------------
           1 . 0 1 1 0            1 0 0
     +     0 . 1 0 1 1  1         1 0 0      Alinhamento
     ---------------------------------
         1 0 . 0 0 0 1  1         1 0 0      Adição Mantis.
     ---------------------------------
           1 . 0 0 0 0  1 1       1 0 1      Renormalização
     ---------------------------------
           1 . 0 0 0 1            1 0 1      Arredondamento
```

b) Underflow da Mantissa

Underflow da mantissa ocorre quando a operação é **subtração efetiva** e a mantissa resultado é menor que a menor mantissa representável segundo o padrão, i.e., o "leading" bit é igual a zero. Neste caso a renormalização é feita deslocando a mantissa para a esquerda e decrementando o expoente até que o MSB seja igual a um. O decremento do expoente resultado pode conduzir a um underflow do expoente. Neste caso o resultado apresentado é zero.

Exemplo:

	Mantissa	Expoente	
	1 . 0 1 1 0	1 0 0	
-	1 . 0 1 0 1	0 1 1	
	1 . 0 1 1 0	1 0 0	*Alinhamento*
-	0 . 1 0 1 0 1	1 0 0	
	1 . 0 1 1 0	1 0 0	
+	1 . 0 1 0 1 1	1 0 0	*Compl. de 2*
	1 0 . 1 0 1 1 1	1 0 0	*Subtr. Mantis.*
	1 . 0 1 1 1	0 1 1	*Renormalização*

Observa-se que o overflow da mantissa, quando ocorre, é sempre de 1 bit, o que implica no deslocamento de 1 bit para a direita para a renormalização. Entretanto o underflofw da mantissa pode exigir deslocamentos de até 24 bits (número de bits da mantissa) para a renormalização, conforme verifica-se no exemplo abaixo:

	Mantissa	Expoente	
	1 . 0 0 0 0	1 1 1	
-	1 . 1 1 1 1	1 1 0	
	1 . 0 0 0 0	1 1 1	*Alinhamento*
-	0 . 1 1 1 1 1	1 1 1	
	1 . 0 0 0 0	1 1 1	
+	1 . 0 0 0 0 1	1 1 1	*Compl. de 2*
	1 0 . 0 0 0 0 1	1 0 0	*Subtr. Mantis.*
	1 . 0 0 0 0	0 1 0	*Renormalização*

É importante deixar clara a diferença entre o overflow e underflow na mantissa e as exceções de overflow e

underflow.

Overflow e Underflow da mantissa são situações que ocorrem durante as operações sobre as mantissas e que conduzem o resultado a um formato não normalizado. A etapa de renormalização corrige esta situação deixando a mantissa no formato normalizado.

As exceções de overflow e underflow são sinalizadas quando os limites de representação do expoente do resultado (tanto superior quanto inferior) são ultrapassados. Esta situação não pode ser corrigida como a anterior (pelo menos quando o formato destino é o mesmo dos operandos). A exceção correspondente é sinalizada e o resultado é definido pelo modo de arredondamento conforme transcrito no item 3.2.4.

3.3.1.6 Arredondamento

O procedimento de Arredondamento é necessário quando o resultado de uma operação apresenta mantissa com número de bits significativos superior ao disponível na representação.

O padrão IEEE especifica 4 modos de arredondamento, conforme transcrito no item 3.2.3.

Dos 4 modos especificados o mais comumente utilizado é o arredondamento para o mais próximo.

Um procedimento de aplicação do arredondamento para o mais próximo é descrito abaixo:

Dado o resultado infinitamente preciso, o bit

imediatamente à direita do bit menos significativo no formato representável, i.e., o bit mais significativo da parte à ser arredondada, tem peso igual à metade do peso do bit menos significativo do formato representável. Então:

a) quando somente este bit é igual a 1 e todos os demais (da parte a ser arredondada) são iguais a zero, este valor infinitamente preciso é igualmente próximo a dois valores:

1. valor representado nos limites do formato, desconsiderando a parte a ser arredondada.

2. valor representado nos limites do formato, incrementando de 1 o bit menos significativo.

A regra para determinação do resultado, conforme o padrão IEEE, é escolher aquele que possuir o bit menos significativo, nos limites do formato representável, igual a zero.

Exemplo:

Considere-se um formato hipotético sendo a mantissa representada com 6 bits. Então, dado o resultado infinitamente preciso abaixo:

LSB da mantissa no formato

1 . 0 1 0 1 1 1 0 . . . 0

MSB da parte a ser arredondada

O resultado infinitamente preciso acima é igualmente próximo aos dois resultados restritos aos limites da representação, mostrados abaixo:

(1) 1 . 0 1 0 1 1

(2) 1 . 0 1 1 0 0

Segundo o critério estabalecido pelo padrão o resultado apresentado é o segundo.

Novamente, sob as mesmas condições, seja o resultado infintamente preciso abaixo:

```
                  LSB da mantissa no formato
                  |
      1 . 0 1 0 1 0 1 0 . . 0
                     |
                     MSB da parte a ser arredondada
```

O resultado infinitamente preciso acima é igualmente próximo aos dois resultados restritos aos limites da representação, mostrados abaixo:

(1) 1 . 0 1 0 1 0

(2) 1 . 0 1 0 1 1

Segundo o critério estabelecido pelo padrão o resultado apresentado é o primeiro.

b) quando o bit mais significativo da parte a ser arredondada é igual a zero, quaisquer que sejam os valores dos outros bits da parte a ser arredondada, o arredondamento é realizado desconsiderando toda a parte a ser arredondada, pois esta será sempre menor que a metade do peso do bit menos significativo nos limites do formato representável.

Exemplo:

Considere-se um formato hipotético sendo a mantissa representada com 6 bits. Então, dado o resultado infinitamente preciso abaixo:

```
                  LSB da mantissa no formato
                  |
      1 . 0 1 0 1 0 0 1 . . . 1
                    |
                    MSB da parte a ser arredondada
```

O valor mais próximo restrito aos limites da reresentação, portanto o resultado apresentado, é:

1 . 0 1 0 1 0

c) quando o bit mais significativo da parte a ser arredondada é igual a 1 e, algum outro bit desta parte é igual a 1, então a parte a ser arredondada é maior que a metade do peso do bit menos significativo no formato representável, e o arredondamento é realizado incrementando de 1 o bit menos significativo da mantissa no formato representável.

Exemplo:

Considere-se um formato hipotético sendo a mantissa representada com 6 bits. Então, dado o resultado infinitamente preciso abaixo:

LSB da mantissa no formato

1 . 0 1 0 1 0 1 0 ... 0 1

MSB da parte a ser arredondada

O valor mais próximo restrito aos limites da reresentação, portanto o resultado apresentado, é:

1 . 0 1 0 1 1

Arredondamento e Renormalização

Em muitos casos a operação sobre as mantissas pode não gerar nem overflow nem underflow na mantissa resultado, permanecendo no formato normalizado. Neste caso o arredondamento é feito diretamente após a operação sobre as mantissas. No entanto, este arredondamento pode conduzir a mantissa a situação de overflow, havendo necessidade, então de renormalizá-la.

De um modo geral podemos distinguir dois casos:

a) adição ou subtração das mantissas produz um resultado não normalizado.

a.1) overflow: renormaliza-se e arredonda-se.

a.2) underflow: renormaliza-se e arredonda-se. Em caso de overflow renormaliza-se novamente.

b) a operação sobre as mantissas não gerou overflow nem underflow, então arredonda-se o resultado. Se ocorrer overflow renormaliza-se.

As informações sobre as etapas de renormalização e arredondamento para as operações de adição e subtração encontram-se sintetizadas no procedimento abaixo:

```
Renormalização_Arredondamento;
{
 SE ( Adição_Efetiva E Overlow_Mantissa)
    ENTÃO
    {
       Renormaliza_Overflow;
       Arredonda;
    }
    SENÃO
      SE (Subtração_Efetiva E Underflow_Mantissa)
      ENTÃO
        {
          Renormaliza_Underflow;
          Arredonda;
          SE Overflow_Mantissa
            ENTÃO
               Renormaliza_Overflow;
        }
      SENÃO
        {
          Arredonda;
          SE Overflow_Mantissa
            ENTÃO
              Renormaliza_Overflow;
        }
}
```

3.3.1.7 Tratamento de exceções

O tratamento de exceções determina o resultado quando um ou ambos os operandos não são normalizados.

Existem quatro representações não normalizadas, conforme o Padrão IEEE:

- três representações com significado aritmético: 0, infinito e números denormalizados e
- uma represesntação não numérica codificada em formato ponto flutuante: NaN (Not a Number).

Os procedimentos adotados em cada caso são mostrados abaixo:

a) Operando NaN

Se pelo menos um dos operandos for NaN, qualquer que seja o outro operando o resultado é NaN e deve ser sinalizada **Operação Inválida**.

b) Operando INFINITO

Se um, e somente um, dos operandos é INFINITO e o outro operando não é NaN, então o resultado é INFINITO (observando-se o sinal).

Se ambos os operandos forem INFINITO então:

- se os sinais de cada operando associados ao sinal da operação resultar numa adição das magnitudes então o resultado é INFINITO (observando-se o sinal).

- se os sinais de cada operando associados ao sinal da operação resultar numa subtração das magnitudes então o resultado é NaN e é sinalizada **Operação Inválida**.

c) Operando DENORMALIZADO

Toda operação que envolver operandos denormalizados será executada considerando-os como ZERO. Ao final da execução é sinalizado **Operando Denormalizado**.

d) Operando ZERO

Quando um dos operandos é igual a ZERO o resultado é igual ao outro operando, observando o sinal associado ao da operação. Se ambos forem iguais a zero o resultado é zero.

3.3.2 Multiplicação binária em ponto flutuante

3.3.2.1 Introdução

A operação de multiplicação em ponto flutuante, em muitos aspectos, é mais simples que a operação de adição e subtração neste mesmo formato. Com exceção da multiplicação das mantissas a operacionalidade da multiplicação é menos complexa.

Seja então:

$$A = f_a \times r^{ea} \quad e$$

$$B = f_b \times r^{eb},$$

onde f representa a mantissa (incluindo o sinal), r é a base e e é o expoente (incluindo o sinal).

$$A \times B = (f_a \times r^{ea}) \times (f_b \times r^{eb})$$

$$= (f_a \times f_b) \times (r^{ea} \times r^{eb})$$

$$A \times B = (f_a \times f_b) \times (r^{(ea + eb)}).$$

Levando em conta o procedimento de cálculo acima e as características especficadas pelo padrão IEEE, podemos dividir uma implementação da operação de multiplicação em cinco etapas:

a) Verificação dos operandos
b) Adição dos expoentes

c) Multiplicação das mantissas
d) Renormalização e Arredondamento
e) Tratamento de exceções

Antes de iniciar a operação é necessário verificar se os operandos são normalizados ou se um ou ambos apresentam valores reservados: NaN, Zero, Infinito ou Denormalizado.

Caso ambos os operandos sejam normalizados é efetuada a adição dos expoentes e a multiplicação das mantissas. Observa-se que as etapas de adição dos expoentes e multiplicação das mantissas não guardam dependência entre si, possibilitando a execução paralela.

No caso de um ou ambos os operandos apresentarem valores não normalizados, o resultado é determinado pelo procedimento de tratamento de exceções.

A multiplicação das mantissas pode gerar um overflow na mantissa resultado que deve, então, ser renormalizada. O resultado pode, também, apresentar precisão superior (número de bits) ao disponível no formato destino. É, então, necessário arredondá-lo.

Abaixo apresenta-se um algoritmo preliminar, referente às cinco etapas acima citadas.

```
Multiplicacao_Ponto_Flutuante;
{
 Verificação_Operandos;
 SE Operandos_Normalizados
    ENTAO
          {
          Adição_Expoentes;
          Multiplicação_Mantissas;
          Renormalização_Arredondamento;
          }
    SENAO
          Tratamento_Exceções;
}
```

3.3.2.2 Verificação dos operandos

Nesta etapa verifica-se se ambos os operandos são normalizados ou não. Quando ambos os operandos são normalizados a operação é executada pelo procedimento normal, i.e., adicionando os expoentes, efetuando a multiplicação das mantissas, renormalizando e arredondando o resultado. Quando pelo menos um dos operandos for não normalizado a operação é desviada para o procedimento de tratamento de exceção.

Esta etapa pode ser executada verificando apenas se os expoentes dos operandos apresentam valores reservados, que são $(E_{máx}+1)$ e $(E_{min}-1)$ (respectivamente 255 e 0 no formato básico simples).

3.3.2.3 Adição dos expoentes

Após a verificação dos operandos, se os mesmos são normalizados, passa-se à execução da operação propriamente dita. Como anteriormente citado, estes dois passos (adição dos expoentes e multiplicação das mantissas) podem ser executados paralelamente.

A etapa de adição dos expoentes compõe-se de duas sub-etapas: Adição dos expoentes e Subtração da polarização (Bias).

Na primeira sub-etapa adicionam-se os expoentes, mas como estes são polarizados (Expoente representado = Expoente Real + Bias) o resultado desta adição resulta num expoente duplamente polarizado. É, então, necessário diminuir do valor obtido pela soma dos expoentes o valor correspondente a uma polarização para que tenha-se o

expoente resultado corretamente polarizado.

3.3.2.4 Multiplicação das mantissas

Paralelamente a adição dos expoentes pode-se efetuar a multiplicação das mantissas.

Como na adição, deve-se concatenar o 1 subentendido na representação à frente dos operandos antes de efetuar-se a multiplicação. Como a mantissa é representada em sinal-magnitude a multiplicação é executada sobre as magnitudes dos operandos, e o sinal é computado independentemente.

3.3.2.5 Renormalização e Arredondamento

A etapa de Renormalização e Arredondamento na multiplicação em ponto flutuante é bem mais simples que a equivalente na adição/subtração. Isto se deve ao fato de que na multiplicação existe apenas uma possibilidade: a de haver ocorrido overflow na multiplicação das mantissas e mesmo assim o overflow é de no máximo 1 bit.

Neste caso desloca-se a mantissa de 1 bit para a direita e incrementa-se o expoente de 1. Após o incremento deve-se verificar se não ocorreu overflow ou underflow do expoente. Em caso afirmativo o resultado a ser apresentado dependerá do modo de arredondamento. É sinalizada a exceção correspondente.

As informações sobre as etapas de renormalização e arredondamento para a operação de multiplicação encontram-se sintetizadas no procedimento abaixo:

```
Renormalização_Arredondamento;
{
    SE Overflow_Mantissa
      ENTAO
        {
          Renormaliza_Overflow;
          Arredonda;
        }
      SENÃO
        {
          Arredonda;
          SE Overflow_Mantissa
           ENTAO
               Renormaliza_Overflow;
        }
}
```

3.3.2.6 Tratamento de exceções

O tratamento de exceções determina o resultado quando um ou ambos os operandos não são normalizados.

Existem quatro representações não normalizadas, conforme o Padrão IEEE:

- três representações com significado aritmético: 0, infinito e números denormalizados e
- uma representação não numérica codificada em formato ponto flutuante: NaN (Not a Number).

Os procedimentos adotados em cada caso são mostrados abaixo:

a) Operando NaN

Se pelo menos um dos operandos for NaN, qualquer que seja o outro operando o resultado é NaN e deve ser sinalizada operação inválida.

b) Operando INFINITO

Se um dos operandos é INFINITO então:

1. se o outro operando é um número normalizado, então o resultado é INFINITO.

2. se o outro operando for zero ou **denormalizado** então o resultado é NaN e deve ser sinalizada operação inválida e operando denormalizado se for o caso.

c) Operando DENORMALIZADO

Se um dos operandos é denormalizado então se o outro operando é um **número** normalizado ou denormalizado o resultado é considerado zero e é sinalizado operando denormalizado.

d) Operando ZERO

Se um dos operando é ZERO então se o outro é um número normalizado ou denormalizado o resultado é zero.

3.4 Conclusões

Neste capítulo estabeleceu-se detalhadamente os procedimentos para as operações de adição, subtração e multiplicação em ponto flutuante, segundo o padrão IEEE de representação numérica apresentado inicialmente, sem contudo envolver aspectos relacionados ao hardware.

Expôs-se os algoritmos gerais e explicitou-se as sub-operações e detalhes inerentes. Deu-se ênfase especial às explicações de etapas normalmente pouco abordadas como a renormalização e arredondamento.

Os algoritmos apresentados não operam sobre números denormalizados. No entanto, detectam e sinalizam sua presença no caso de ocorrerem em alguma operação.

Dos quatro modos de arredondamento recomendados no padrão IEEE, detalha-se o mais empregado, o arredondamento para o mais próximo, apesar de que os procedimentos para os

demais modos de arredondamento são facilmente extraídos do padrão.

O procedimento de tratamento de exceção encarrega-se dos resultados quando os operandos não são normalizados, abreviando a computação.

4 ARQUITETURA DOS OPERADORES ARITMÉTICOS DE PONTO FLUTUANTE

4.1 Introdução

Neste capítulo são apresentadas propostas de arquiteturas para os operadores cujos algoritmos foram expostos no capítulo anterior, decrevendo os recursos e os caminhos de dados necessários.

Como tais operações são convencionalmente lentas as arquiteturas propostas procuram obter o máximo possível do algoritmo. Algumas estruturas similares são duplicadas, outras particionadas tendo em vista uma posterior implementação pipeline.

São discutidas questões particulares às realizações em hardware sem contudo abordar aspectos de temporização, controle ou referências à tecnologia de implementação.

Não são levados em consideração fatores restritivos relativos à interface (entrada e saída) e à área ocupada numa eventual implementação monolítica. Considerações desta ordem serão analisadas no capítulo posterior.

4.2 Arquitetura para o operador de adição e subtração em ponto flutuante

Neste item é descrita uma arquitetura que executa o algoritmo de adição e subtração em ponto flutuante apresentado no capítulo anterior. As premissas são as mesmas: os números denormalizados são considerados como zero mas sua presença é detectada e sinalizada e implementa-se apenas o modo de arredondamento para o mais

próximo.

Os blocos arquiteturais referem-se aproximadamente às etapas do algoritmo. Na medida do possível paralelizou-se estas etapas, de forma particular o procedimento de tratamento de exceções que opera totalmente independente da lógica de cálculo "normal".

Antes da apresentação da arquitetura são discutidas algumas características de hardware particulares a adição e subtração em ponto flutuante tais como a adição em sinal magnitude e o arredondamento.

4.2.1 Aspectos inerentes à implementação em hardware

4.2.1.1 Adição e Subtração das mantissas

4.2.1.1.1 Operação Efetiva

Verifica-se que as mantissas nos números representados segundo o padrão IEEE são representadas em sinal-magnitude.

A possibilidade de representar tanto números positivos como negativos associados as operações de adição e subtração conduz a várias combinações sintetizadas aqui no que denomina-se **operação efetiva**.

A operação que efetivamente realiza-se entre as magnitudes das mantissas depende da operação propriamente dita (adição ou subtração) e dos sinais dos operandos (+ ou -).

Assim, a partir dos sinais dos operandos e da operação, recai-se em duas situações:

a) Adição Efetiva

b) Subtração Efetiva

Adição Efetiva

A Adição Efetiva verifica-se nos seguintes casos:

$$(+A) + (+B) = + (|A| + |B|)$$
$$(-A) + (-B) = - (|A| + |B|)$$
$$(-A) - (+B) = - (|A| + |B|)$$
$$(+A) - (-B) = + (|A| + |B|),$$

ou seja, nestes casos, a operação que é efetivamente realizada entre as magnitudes das mantissas é a adição.

Na representação numérica binária, segundo o padrão IEEE, o sinal positivo corresponde ao valor lógico ZERO no campo do sinal e o sinal negativo ao valor lógico UM. Considerando, ainda, que à operação de adição corresponde o valor lógico ZERO e à subtração o valor lógico UM, conclui-se que a adição efetiva pode ser expressa pela seguinte equação lógica:

Adição Efetiva = ~ [(sinal A) *XOR* (sinal b) *XOR* (operação)]

Nos casos de adição efetiva o sinal do resultado é obtido diretamente a partir dos sinais dos operandos e da operação. Observa-se que quando ocorre adição efetiva o sinal do resultado é sempre o mesmo do operando A.

Subtração Efetiva

A Subtração Efetiva verifica-se nos seguintes casos:

$$(+A) - (+B) = + (|A| - |B|)$$
$$(+A) + (-B) = + (|A| - |B|)$$
$$(-A) - (-B) = - (|A| - |B|)$$
$$(-A) + (+B) = - (|A| - |B|),$$

ou seja, nestes casos, a operação que é efetivamente

realizada entre as magnitudes das mantissas é a subtração.

A subtração efetiva corresponde a situação complementar da adição efetiva, logo, é expressa pelo complemento da equação para a adição efetiva:

Subtração Efetiva =

= [(sinal A) *XOR* (sinal b) *XOR* (operação)]

Nos casos de subtração efetiva o sinal do resultado depende não só dos sinais dos operandos e do sinal da operação mas também da magnitude dos operandos.

Conhecendo-se o operando de maior magnitude obtem-se o sinal do resultado conforme a tabela abaixo.

Tabela 4.1 Sinal do resultado da Subtração

Sinal de A	*Magnitude dos operandos*	*Sinal do Resultado*
+	mag.A > mag.B	+
+	mag.A < mag.B	-
-	mag.A > mag.B	-
-	mag.A < mag.B	+

4.2.1.1.2 A Adição e Subtração em sinal-magnitude

Das três notações para representação de números negativos, a representação em sinal-magnitude apresenta as maiores dificuldades para implementação das operações de adição e subtração.

Quando a operação é adição efetiva o procedimento é direto. As magnitudes são somadas e o sinal determinado conforme mostrado no item anterior.

Na subtração efetiva surge um problema. É necessário saber qual dos operandos é maior, ou seja, qual é o sinal do resultado. Isto exige uma subtração e o único método de automatizar a adição de operandos com sinais opostos é utilizando complementação [GOS 80].

Hwang [HWA 79] e Gosling [GOS 80] apresentam circuitos lógicos para adição e subtração em sinal-magnitude. Na figura 4.1 é mostrado o de Hwang.

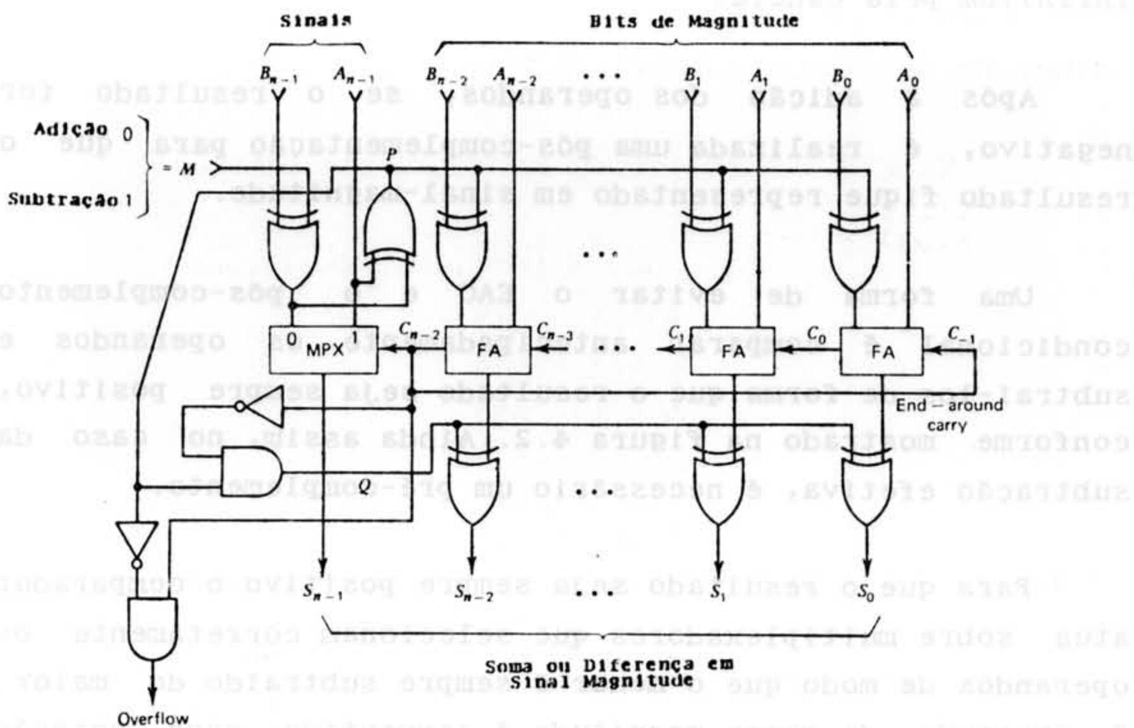


Figura 4.1 Somador/subtrator sinal-magnitude com EAC [HWA 79].

Este circuito tem como base um somador/subtrator em complemento de um. Isto justifica-se pois as representações complemento de um e sinal-magnitude guardam estreita relação entre si: para números negativos basta complementar o número bit a bit (com exceção do bit de sinal) para obtê-

lo na outra representação.

Quando a operação é subtração efetiva o operando B é complementado bit a bit, obtendo assim sua representação negativa em complemento de um. Após este estágio os operandos são adicionados no somador tipo complemento de um. Tal somador exige um End-Around_Carry (EAC): uma linha que permite a propagação do carry-out do bit mais significativo ao carry-in do bit menos significativo. Em [GOS 80] podem ser encontradas explicações detalhadas, bem como uma prova de que o carry não fica circulando *ad infinitum* pela cadeia.

Após a adição dos operandos, se o resultado for negativo, é realizada uma pós-complementação para que o resultado fique representado em sinal-magnitude.

Uma forma de evitar o EAC e o pós-complemento condicional é comparar antecipadamente os operandos e subtraí-los de forma que o resultado seja sempre positivo, conforme mostrado na figura 4.2. Ainda assim, no caso da subtração efetiva, é necessário um pré-complemento.

Para que o resultado seja sempre positivo o comparador atua sobre multiplexadores que selecionam corretamente os operandos de modo que o menor é sempre subtraído do maior. O operando de menor magnitude é convertido para notação complemento de dois (quando a operação é subtração efetiva), ou seja, é complementado bit a bit e é "setado" para 1 a entrada de carry-in do somador, fazendo as vezes do incremento.

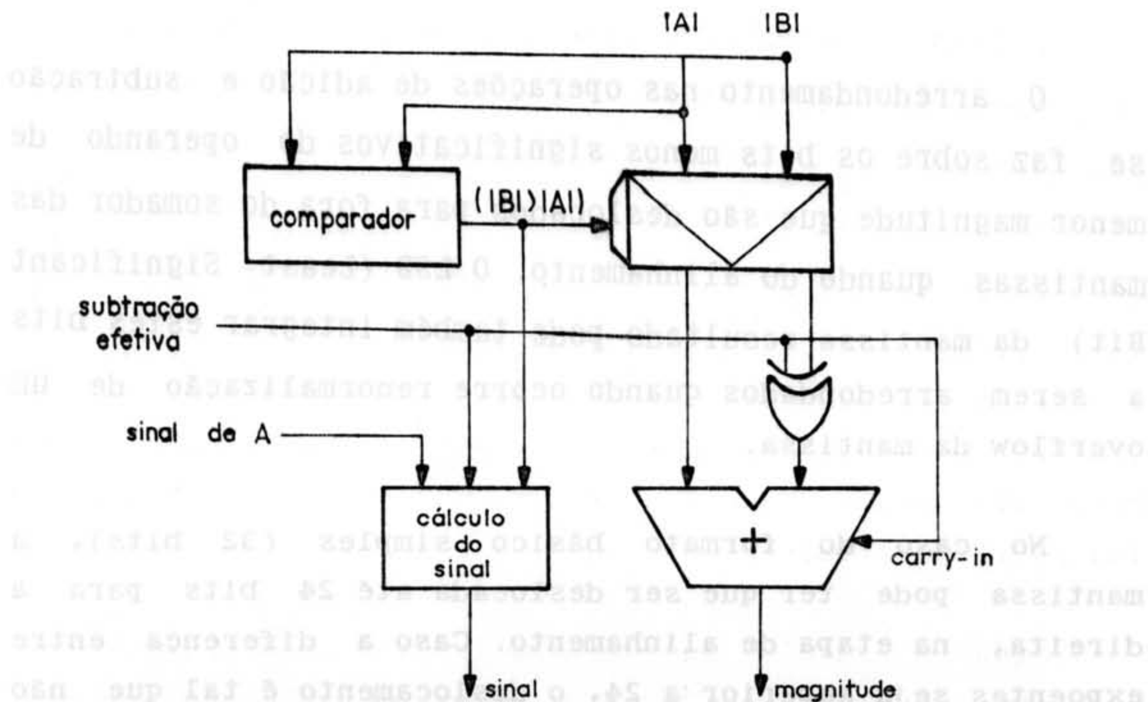


Figura 4.2 Somador/subtrator sinal-magnitude com comparador.

O sinal do resultado é calculado segundo o procedimento apresentado no item anterior.

Finalmente uma outra forma seria converter o operando B para representação negativa em complemento de dois sempre que ocorresse subtração verdadeira, então, após a adição, se o resultado fosse negativo seria realizada a conversão para a representação positiva. A desvantagem deste método é que a pós-conversão requer adição completa.

4.2.1.2 Arredondamento

Conforme visto anteriormente, o arredondamento toma um

resultado infinitamente preciso e modifica-o, se necessário, para colocá-lo nos limites do formato destino.

O arredondamento nas operações de adição e subtração se faz sobre os bits menos significativos do operando de menor magnitude que são deslocados para fora do somador das mantissas quando do alinhamento. O LSB (Least Significant Bit) da mantissa resultado pode também integrar estes bits a serem arredondados quando ocorre renormalização de um overflow da mantissa.

No caso do formato básico simples (32 bits), a mantissa pode ter que ser deslocada até 24 bits para a direita, na etapa de alinhamento. Caso a diferença entre expoentes seja superior a 24, o deslocamento é tal que não há interação efetiva entre as mantissas e o resultado é igual ao outro operando, a menos do arredondamento. Por esta razão limita-se, aqui, o deslocamento em 24 bits.

Com o objetivo de evitar que durante a etapa de arredondamento tenha-se que armazenar e analisar todos os bits deslocados à direita (devido ao alinhamento e à uma eventual renormalização) e facilitar a implementação do hardware de arredondamento, calculam-se ***bits de arredondamento*** que sintetizam as informações da parte a ser arredondada em alguns bits.

4.2.1.2.1 Bits de Arredondamento

Os bits de arredondamento representam uma extensão da mantissa resultado; são três bits à direita do LSB da mantissa no formato, conforme mostra a figura 4.3, que sintetizam informação suficiente para o correto arredondamento do resultado infinitamente preciso.

Os três bits de arredondamento são referenciados na literatura [MOT 87], do MSB para o LSB, respectivamente como: guard bit, round bit e sticky bit.

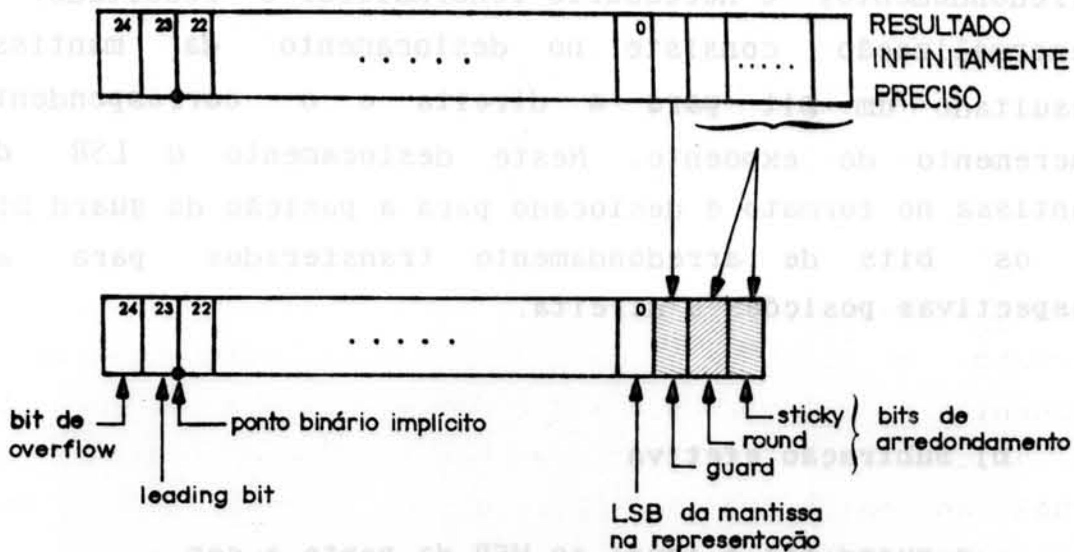


Figura 4.3 Bits de arredondamento

O cálculo dos bits de arredondamento pode ser realizado imediatamente após o deslocamento da mantissa do operandos de menor magnitude, para o alinhamento das mantissas, em paralelo com a adição das mantissas.

Procedimento para cálculo dos bits de arredondamento

Divide-se o procedimento de cálculo em dois, segundo a operação efetiva:

a) Adição efetiva

- o guard bit é igual ao MSB da parte a ser arredondada.

- o round bit é igual ao OU-LÓGICO de todos os demais bits da parte a ser arredondada.
- o sticky bit é igual a zero.

Após a adição das mantissas pode ocorrer um overflow na mantissa resultado. Neste caso, antes de realizar o arredondamento, é necessário renormalizar o resultado. A renormalização consiste no deslocamento da mantissa resultado um bit para a direita e o correspondente incremento do expoente. Neste deslocamento o LSB da mantissa no formato é deslocado para a posição do guard bit e os bits de arredondamento transferidos para as respectivas posições à direita.

b) Subtração efetiva

- o guard bit é igual ao MSB da parte a ser arredondada.
- o round bit é igual ao segundo bit mais significativo da parte a ser arredondada.
- o sticky bit é igual ao OU-LÓGICO de todos os bits da parte a ser arredondada.

A razão para o cálculo de todos os bits de arredondamento na subtração efetiva é de que pode haver necessidade de renormalizar um underflow da mantissa resultado deslocando a mantissa e os bits de arredondamento um bit à esquerda de maneira equivalente ao que acontece na adição efetiva, sobrando, então, dois bits para o arredondamento.

Conforme dito anteriormente, pode ocorrer underflow que exija o deslocamento de até 24 bits à esquerda. No entanto qualquer underflow que exija deslocamentos

superiores a um bit para renormalizar só podem acontecer quando a diferença entre os expoentes dos operandos for igual a um ou zero. Isto garante que os bits de arredondamento são capazes de produzir o correto arredondamento tanto na adição efetiva como na subtração efetiva.

Obs.: Considerando o formato básico simples (32 bits) e o modo de arredondamento para o mais próximo, só há sentido em se calcular os bits de arredondamento quando os deslocamentos forem menores ou iguais a 24 bits. Caso a diferença seja superior a 24 o resultado é igual ao operando de maior magnitude devendo-se apenas sinalizar a exceção de resultado inexato.

4.2.1.2.2 Procedimento de Arredondamento para o mais próximo a partir dos Bits de Arredondamento

Uma vez disponível a mantissa do resultado normalizada e os bits de arredondamento calculados pode-se efetuar o arredondamento. O procedimento para implementação do modo de arredondamento para o mais próximo a partir dos bits de arredondamento é bastante simples, baseado no item 3.3.1.6 e descrito abaixo:

```
Arredondamento;
{
 SE ((guard OU round OU sticky)==1)
  ENTAO
      Inexato=1;

 SE (guard==1)
  ENTAO
      SE ((round OU sticky OU
           LSB da mantissa)==1)
     ENTAO
        { Incrementa LSB_Mantissa;
          Renormaliza;
        }
{
```

4.2.2 Descrição da arquitetura do Somador/Subtrator em ponto flutuante

Na figura 4.4 é mostrada uma arquitetura para implementação do algoritmo de adição/subtração em ponto flutuante descrito anteriormente.

A verificação dos operandos é realizada pelos blocos de análise e comparação dos expoentes e mantissas.

O multiplexador dos expoentes seleciona o expoente maior como expoente resultado e estabelece as entradas do subtrator de expoentes que fornece o valor do deslocamento para alinhar as mantissas.

A adição e subtração é efetuada conforme descrito anteriormente, comparando antes as magnitudes dos operandos e colocando-os de tal forma que o resultado da operação sobre as mantissas seja sempre positivo.

O resultado da adição ou subtração das mantissas entra nos circuitos de renormalização e arredondamento. O circuito de renormalização está conectado ao um circuito de ajuste do expoente resultado para compensar os deslocamentos da renormalização.

Os blocos de controle e cálculo de exceção determinam o resultado nos casos de operandos reservados (NaN e Infinito) e nos casos de overflow e underflow do expoente resultado.

A seguir descreveremos os blocos arquiteturais, suas funções e interconexões.

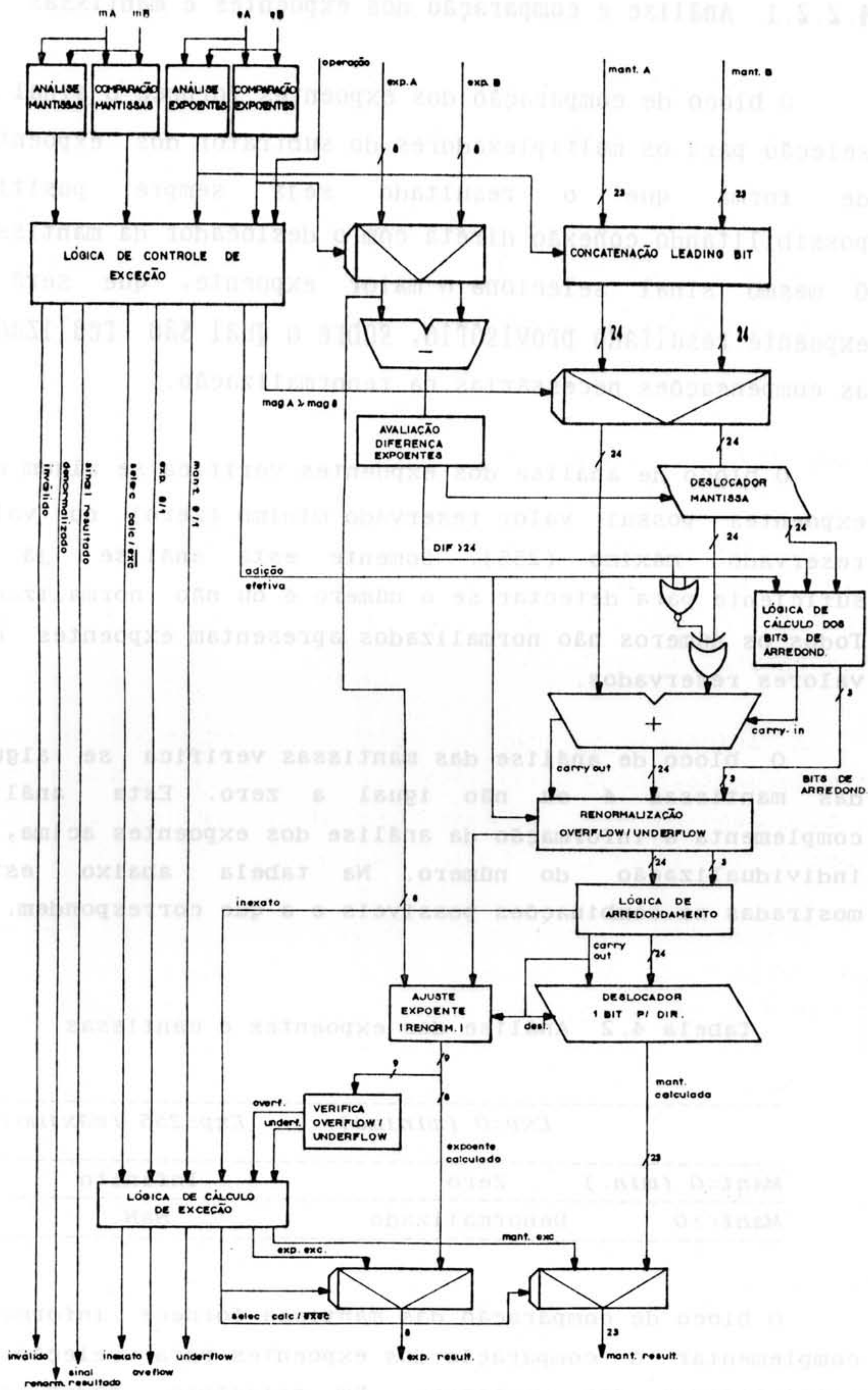


Figura 4.4 Arquitetura do somador/subtrator ponto flutuante

4.2.2.1 Análise e comparação dos expoentes e mantissas

O bloco de comparação dos expoentes fornece o sinal de seleção para os multiplexadores do subtrator dos expoentes de forma que o resultado seja sempre positivo possibilitando conexão direta com o deslocador da mantissa. O mesmo sinal seleciona o maior expoente, que será o expoente resultado provisório, sobre o qual são realizadas as compensações necessárias na renormalização.

O bloco de análise dos expoentes verifica se algum dos expoentes possui valor reservado mínimo (zero) ou valor reservado máximo (255). Somente esta análise já é suficiente para detectar se o número é ou não normalizado. Todos os números não normalizados apresentam expoentes com valores reservados.

O bloco de análise das mantissas verifica se alguma das mantissas é ou não igual a zero. Esta análise complementa a informação da análise dos expoentes acima, na individualização do número. Na tabela abaixo estão mostradas as combinações possíveis e a que correspondem.

Tabela 4.2 Análise dos expoentes e mantissas

	Exp=0 (mínimo)	*Exp=255 (máximo)*
Mant=0 (mín.)	Zero	Infinito
Mant<>0	Denormalizado	NaN

O bloco de comparação das mantissa fornece informação complementar à comparação dos expoentes para seleção do multiplexador das mantissas. Em princípio poder-se-ia utilizar o mesmo sinal de seleção do multiplexador dos

expoentes para a seleção do multiplexador das mantissas, entretanto, pode ocorrer a situação em que os expoentes são iguais. Neste caso se as mantissas forem arbitrariamente subtraídas um eventual resultado negativo desta subtração exigiria a conversão do número negativo para positivo em complemento de dois. Para evitar esta situação, que pode gerar um carry propagando por toda a mantissa compara-se as mantissas antecipadamente, garantindo, assim, o resultado sempre positivo.

4.2.2.2 O Subtrator dos expoentes

As entradas do subtrator dos expoentes são selecionadas através de um multiplexador controlado pelo sinal de comparação dos expoentes, de forma que o resultado da subtração seja sempre positivo, e possa diretamente, sem conversões, fornecer esta informação ao deslocador da mantissa. Esta seleção, conseqüentemente, define o maior expoente, que é o expoente resultado a menos dos ajustes de renormalização ao final da operação.

A diferença entre os expoentes pode ser superior ao tamanho da mantissa no formato (24 bits no formato básico simples) e não haver interação direta entre as mantissas. Nestes casos a lógica de avaliação da diferença entre os expoentes define um deslocamento de 24 bits, o que faz com que toda a mantissa do número de menor magnitude seja deslocada para a lógica de cálculo dos bits de arredondamento e sinaliza a este bloco esta condição.

4.2.2.3 Lógica de controle de exceção

Este bloco arquitetural recebe como entrada os sinais

gerados pelos blocos de análise e comparação dos expoentes e mantissas, os sinais dos operandos A e B e a operação a ser realizada.

A partir dos sinais dos operandos e da operação é calculada a operação efetiva. A operação efetiva associada ao sinal do primeiro operando e a comparação das mangnitudes dos operandos fornece o sinal do resultado conforme descrito no item 4.2.1.1.1.

Com os resultados da análise dos expoentes e das mantissas determina-se se alguma operação ou operando é inválido (operando NaN ou subtração efetiva entre Infinitos) e se existe algum operando denormalizado.

Caso o resultado deva ser calculado pela lógica de exceção, isto é sinalizado para a lógica de cálculo de exceção bem como os valores que terão o expoente e a mantissa. Como todos os resultados gerados pela lógica de exceção são extremos (0, Infinito ou NaN) basta uma linha para definir a mantissa (ou todos os bits iguais a zero ou todos iguais a um) e uma para definir o expoente, da mesma forma.

4.2.2.4 Deslocador da mantissa

O deslocador da mantissa tem dois blocos arquiteturais associados. O primeiro concatena o "leading" bit implícito na representação à frente das mantissa dos dois operandos se estes forem normalizados. Caso algum operando seja zero ou denormalizado (ou seja, sempre que o expoente for igual a zero) toda a mantissa, inclusive o "leading" bit, é zerada.

O multiplexador das mantissas seleciona a do operando de menor magnitude para entrar no deslocador da mantissa enquanto que a outra é direcionada para o somador das mantissas.

O deslocador das mantissa desloca a do operando de menor magnitude de 0 a 24 bits para a direita, preenchendo com zeros as posições mais significativas deslocadas. O número de bits deslocados é fornecido pela lógica de avaliação da diferença entre os expoentes.

A saída do deslocador é de 48 bits. Os 24 mais significativos entram no somador das mantissas enquanto os menos significativos são direcionados para a lógica de cálculo dos bits de arredondamento.

4.2.2.5 Somador/Subtrator das mantissas

Trata-se de um somador convencional (complemento de dois) possivelmente com recursos para acelerar a propagação do carry.

A mantissa do operando de maior magnitude vem diretamente dos multiplexadores das mantissas (entrada esquerda do somador).

A mantissa do operando de menor magnitude, após o deslocamento, passa por um estágio de pré-complementação condicional antes de entrar no somador. Este estágio (portas XOR) é necessário pois quando a operação é subtração efetiva é preciso converter o operando para a representação negativa em complemento de dois.

No item 3.1.1 mostrou-se que a representação negativa

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

de um número é obtida, em complemento de dois, invertendo-se logicamente todos os bits do número e incrementando-se o bit menos significativo. Este estágio realiza a complementação bit a bit do operando. O incremento de 1 é realizado através da entrada de carry-in do somador.

A conversão para a representação negativa em complemento de dois pode produzir um carry que se propaga por toda a mantissa, consumindo o tempo equivalente a uma adição. Aproveita-se, então, a entrada carry-in do somador, evitando, desta forma, dupla propagação do carry.

O sinal de carry-in no somador é função, evidentemente, da operação efetiva (só poderá estar ativo quando a operação for uma subtração efetiva), mas também é função dos bits deslocados para fora do somador pelo deslocador da mantissa.

Sempre que houver diferença entre os expoentes dos operandos, a mantissa do operando de menor magnitude estará, em parte ou em todo, deslocada para fora do somador. Como o número a ser negado (aritmeticamente) deve ser incrementado a partir do bit menos significativo a entrada de carry-in do somador não pode ser simplesmente "setada" para 1 quando a operação for subtração efetiva. Há um procedimento para determinar o valor do carry-in em função dos bits deslocados para fora da entrada do somador. Este procedimento está incluído no procedimento de cálculo dos bits de arredondamento no item 4.2.2.6.

Obs.: Sempre que o deslocamento da mantissa para alinhamento for superior a 24, a lógica de pré-complemento não complementa a mantissa (no caso, toda zerada devido ao deslocamento) qualquer que seja a operação efetiva. Isto porque além de não haver interação entre as mantissas e,

assumindo o modo de arredondamento para o mais próximo, não há modificação do resultado da operação (que é igual ao outro operando). Deve, entretanto, sinalizar-se a exceção de operação inexata.

4.2.2.6 Lógica de cálculo dos Bits de Arredondamento

No item 4.2.1.2.2 apresentou-se um procedimento para o cálculo dos bits de arredondamento. Entretanto, naquele caso, pressupunha-se que o resultado infinitamente preciso estivesse disponível para esta computação, ou seja, que os bits deslocados para fora do somador fossem a continuação, para além do limite inferior do formato, do valor calculado.

Isto é verdade quando a operação é adição efetiva e aquele procedimento pode ser empregado. Mas quando a operação é subtração efetiva, a mantissa do operando de menor magnitude deve ser convertida para a representação negativa em complemento de dois conforme descrito no item anterior.

Para evitar uma cadeia de propagação de carry nos bits deslocados para fora do somador empregam-se os procedimentos abaixo que a partir dos bits deslocados determinam os bits de arredondamento. Nestes procedimentos, conforme dito no item anterior, inclui-se o cálculo do carry-in do somador.

Procedimentos para cálculo dos bits de arredondamento

```
Bits_Arredondamento;
{
 IF (diferença entre expoentes > 24)
  THEN
     {
       carry-in=0;
       guard=0;
       round=0;
       sticky=1;
     }
  ELSE
     IF (Adição Efetiva)
       THEN
           Bits_Arredondamento_Adição_Efetiva;
       ELSE
           Bits_Arredondamento_Subtração_Efetiva;
}
```

```
Bits_Arredondamento_Adição_Efetiva
{
 carry-in=0;
 guard=MSB da parte a ser arredondada;
 round=OU-LÓGICO dos demais bits da parte a ser arred;
 sticky=0;
}
```

```
Bits_Arredondamento_Subtração_Efetiva

{
 IF (deslocamento > 0)
   THEN
     {
      carry-in=1;
      guard=round=sticky=0;
     }
   ELSE
     {
      carry-in=0;
      IF (deslocamento >= 3)
        THEN
          {
           IF (OU-LÓGICO de todos os bits da parte
               a ser arredondada, a partir do terceiro
               MSB inclusive == 0)
             THEN
               {
                guard=NOT(segundo MSB da parte a ser
                      arredondada XOR MSB da parte a
                      ser arredondada);
                round=segundo MSB da parte a ser arre-
                      dondada;
                sticky=0;
               }
             ELSE
               {
                guard=MSB da parte a ser arredondada;
                round=segundo MSB;
                sticky=1;
               }
        ELSE
          IF (deslocamento=2)
            THEN
              {
               sticky=0;
               round=segundo MSB;
               IF (segundo MSB = 1)
                 THEN
                   guard=NOT(MSB);
                 ELSE
                   guard=MSB;
            ELSE
              guard=MSB;
     }
```

4.2.2.7 Lógica de Renormalização e Arredondamento

No resultado do somador das mantissas pode ocorrer overflow, no caso de adição efetiva, ou underflow, no caso de subtração efetiva (vide item 3.3.1.5). Caso ocorra overflow ou underflow da mantissa o primeiro procedimento é renormalizar, ou seja, deixar o número na representação normalizada, na qual a parte inteira da mantissa é composta de um bit apenas (o leading bit) que deve sempre ser igual

a um.

Para esta função a arquitetura prevê um circuito que realiza deslocamentos para a direita e para a esquerda.

O deslocamento para a direita é necessário quando ocorre overflow da mantissa, possível apenas na adição efetiva. Observa-se também que o overflow da mantissa nunca ultrapassa 1 bit. O deslocamento da mantissa para a direita é compensado pelo respectivo incremento do expoente resultado. Neste deslocamento os bits de arredondamento são deslocados juntamente com a mantissa de forma que o LSB da mantissa nos limites do formato ocupa a posição mais significativa dos bits de arredondamento, e estes são deslocados para as respectivas posições um bit à direita.

O deslocamento para a esquerda é necessário quando há underflow da mantissa, possível apenas quando a operação é subtração efetiva. O underflow da mantissa configura-se quando o "leading" bit da mantissa é zero. Neste caso o número deve ser deslocado para a esquerda até que o leading bit seja igual a um. Estes deslocamentos são compensados por decrementos no expoente do resultado.

Podem ser necessários deslocamentos de até 24 bits como mostrado na seção 3.3.1.5. Para acelerar esta operação, um circuito dedicado detecta a posição do "leading" bit 1 mais significativo e envia esta informação ao barrel shifter que executa o deslocamento.

De forma análoga ao caso do overflow, os bits de arredondamento seguem os deslocamentos realizados pela mantissa do resultado como uma extensão do número além dos limites da representação.

Após a renormalização, se necessário, arredonda-se o resultado. O arredondamento não é mais do que um incremento à mantissa do resultado condicionado aos valores dos bits de arrendondamento e do bit menos significativo da mantissa resultado conforme o procedimento descrito no item 4.2.1.2.2. Sempre que algum dos bits de arredondamento for igual a um, a exceção de resultado inexato é sinalizada à lógica de cálculo de exceção. Após o arredondamento pode haver necessidade de uma nova renormalização devido a um overflow na mantissa, para tanto a arquitetura provê um circuito que executa este deslocamento condicional.

Após a etapa de renormalização e arredondamento o expoente resultado é verificado para as situações de overflow e underflow. Caso se apresente uma destas situações é feita uma sinalização à lógica de cálculo de exceção.

4.2.2.8 Lógica de Cálculo de exceção

A lógica de cálculo de exceção recebe da lógica de controle de exceção um sinal que define se o resultado apresentado será o resultado calculado pelo procedimento normal ou definido neste bloco. Também são sinalizados os valores da mantissa e dos expoentes no caso de um resultado de exceção.

Caso a lógica de controle de exceção tenha sinalizado que o resultado deve ser estabelecido pela lógica de cálculo, se ocorrer overflow ou underflow do expoente o resultado será definido pela lógica de exceção.

Assim, a lógica de cálculo de exceção controla o resultado apresentado e também fornece os sinais de

exceção: overflow, underflow e inexato.

A saída do circuito existem dois multiplexadores que comutam o resultado apresentado ao exterior entre o resultado calculado e o resultado de exceção. Este procedimento permite que a lógica de exceção e a lógica de cálculo trabalhem paralela e independentemente sem necessidade de desabilitação de uma ou de outra, selecionando-se apenas ao final da operação o resultado correto, o que simplifica o controle.

4.2.2.9 Comentários sobre a arquitetura

A verificação dos expoentes realizada através dos blocos de análise das mantissas e expoentes é implementada de forma relativamente simples, bem como a comparação entre os expoentes que fornece o sinal para o multiplexador dos expoentes. Entretanto a comparação das mantissas exige a implementação de um comparador de 23 bits. Esta é a principal desvantagem da solução escolhida para a adição e subtração das mantissas. Mas, além de simplificar os demais aspectos, este comparador permite realizar, com o mesmo hardware, uma comparação rápida entre dois operandos ponto flutuante. Outra vantagem é a detecção da situação em que há subtração entre dois números iguais. Neste caso o resultado deve ser zero e caso não houvesse a detecção anterior o hardware de cálculo deveria ser capaz de chegar a este resultado. No caso desta arquitetura o resultado é definido pela lógica de exceção, simplificando o hardware de cálculo.

As lógicas de controle e cálculo de exceção são facilmente implementáveis através de um PLA (Programable Logic Array).

Da mesma forma, a lógica de cálculo dos bits de arredondamento, uma vez estabelecidas e minimizadas as equações lógicas (mais complexas, neste caso), é diretamente implementável em um PLA.

Uma das unidades mais críticas da arquitetura é o deslocador das mantissas. Crítico pois exerce influência decisiva no desempenho do operador. Caso opte-se por implementá-lo através de um barrel-shifter é possível deslocar de 0 a 24 bits em um ciclo de relógio. Se o deslocamento for realizado por um registrador de deslocamento o tempo deverá ser bastante superior. Como sempre, a escolha dependerá do desempenho requerido e da área disponível.

Outra unidade crítica e decisiva é o somador das mantissas (24 bits). Certamente deve ser implementado com alguma estratégia para aceleração do carry tais como carry look-ahead [WES 85], [HWA 79], manchester [WES 85], [GLA 85] ou carry select [WES 85], [HWA 79].

Finalmente o bloco renormalização e arredondamento resume os dois aspectos críticos acima citados. Na renormalização de um underflow da mantissa podem ser necessários deslocamentos da ordem do número de bits da mantissa, ou seja, um deslocador idêntico ao anterior (mas que desloca para a esquerda). Um aspecto complicador é a necessidade de um circuito para detectar a posição do "leading" bit 1. No arredondamento pode haver necessidade de incrementar a mantissa o que exige uma cadeia de propagação de carry bastante semelhante àquela do somador.

A arquitetura proposta possibilita a fácil execução das operações de adição e subtração em ponto flutuante

envolvendo os módulos dos operandos, mediante mínimas alterações na lógica de controle de exceção (que calcula a operação efetiva e o sinal do resultado). Outras operações facilmente implementáveis são: comparação entre números ponto flutuante e algumas operações lógicas (sobre os bits da mantissa).

4.3 Arquitetura para o operador de Multiplicação em ponto flutuante

O mesmo procedimento adotado para a proposta de arquitetura do operador de adição e subtração em ponto flutuante é empregada para o operador de multiplicação em ponto flutuante. As premissas com respeito aos números denormalizados e ao modo de arredondamento são as mesmas.

De forma análoga, apresenta-se inicialmente alguns aspectos das realizações em hardware particulares à operação.

4.3.1 Aspectos inerentes à implementação em hardware

a) A multiplicação em sinal-magnitude

A representação em sinal-magnitude é a mais simples para a operação de multiplicação pois é possível operar independentemente com a magnitude e os sinais.

A operação sobre as magnitudes é efetuada como se fossem ambas sempre positivas, o que simplifica o hardware do multiplicador.

O sinal é calculado à parte: quando os sinais forem iguais o resultado é positivo, quando forem diferentes o

resultado é negativo.

As técnicas de implementação varrem um vasto espectro de alternativas, desde os multiplicadores seriais até os arrays totalmente paralelos, passando por soluções intermediárias. Compromissos de área e desempenho requerido definirão a solução mais adequada.

b) Arredondamento

O procedimento de cálculo dos bits de arredondamento é mais simples que nas operações de adição e subtração. Os bits a partir dos quais são calculados os bits de arredondamento estão disponíveis após a multiplicação das mantissas e constituem a continuação da mantissa para além do limite inferior do formato, como no caso da adição efetiva. Logo, o procedimento de cálculo destes bits é idêntico ao da adição efetiva (item 4.2.1.2.1). Vide a figura 4.5.

O arredondamento é realizado a partir dos bits de arredondamento da mesma forma que na adição e subtração.

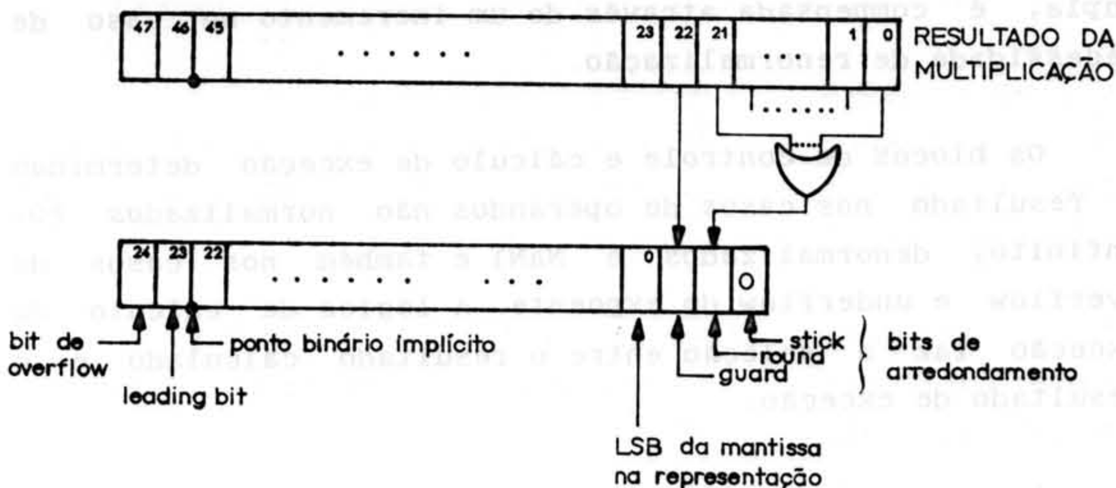


Figura 4.5 Bits de arredondamento na multiplicação ponto flutuante.

4.3.2 Descrição da arquitetura do multiplicador ponto flutuante

Na figura 4.6 é mostrado o esquema geral da arquitetura do multiplicador ponto flutuante. Como na arquitetura do operador de adição e subtração a verificação dos operandos é realizada pelos blocos de análise da mantissas e expoentes. No multiplicador ponto flutuante não existem os circuitos comparadores das mantissas e expoentes pois, ao contrário da operação de adição e subtração, não importa a ordem dos operandos nas sub-operações.

Paralelamente à análise das mantissas e expoentes, são iniciadas as sub-operações sobre o sinal, expoentes e mantissas.

A partir dos bits menos significativos do resultado do produto das mantissas são calculados os bits de arredondamento que, juntamente com a mantissa, entram no circuito de renormalização e arredondamento.

A soma dos expoentes, após a subtração da polarização dupla, é compensada através de um incremento no caso de necessidade de renormalização.

Os blocos de controle e cálculo de exceção determinam o resultado nos casos de operandos não normalizados (0, Infinito, denormalizados e NaN) e também nos casos de overflow e underflow do expoente. A lógica de cálculo de exceção faz a seleção entre o resultado calculado e o resultado de exceção.

A seguir descrevem-se os blocos arquiteturais e suas interconexões.

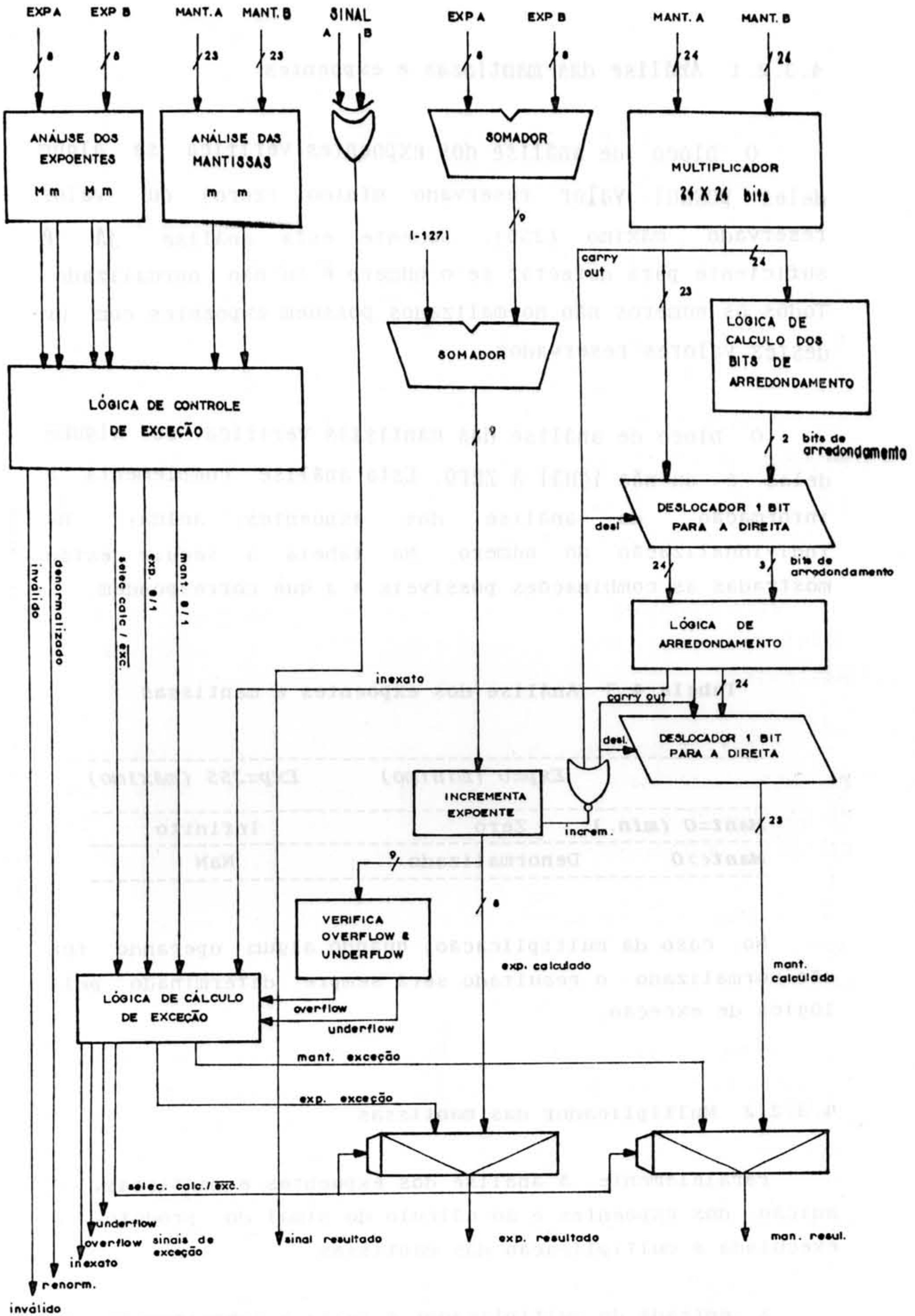


Figura 4.6 Arquitetura do multiplicador ponto flutuante

4.3.2.1 Análise das mantissas e expoentes

O bloco de análise dos expoentes verifica se algum deles possui valor reservado mínimo (zero) ou valor reservado máximo (255). Somente esta análise já é suficiente para detectar se o número é ou não normalizado. Todos os números não normalizados possuem expoentes com um destes valores reservados.

O bloco de análise das mantissas verifica se alguma delas é ou não igual a zero. Esta análise complementa a informação da análise dos expoentes acima, na individualização do número. Na tabela a seguir estão mostradas as combinações possíveis e a que correspondem.

Tabela 4.3 Análise dos expoentes e mantissas

	Exp=0 (mínimo)	*Exp=255 (máximo)*
Mant=0 (mín.)	Zero	Infinito
Mant<>0	Denormalizado	NaN

No caso da multiplicação, quando algum operando for não-normalizado o resultado será sempre determinado pela lógica de exceção.

4.3.2.2 Multiplicador das mantissas

Paralelamente à análise dos expoentes e mantissas, à adição dos expoentes e ao cálculo do sinal do produto, é executada a multiplicação das mantissas.

À entrada do multiplicador é feita a concatenação do

"leading" bit implícito à frente das frações.

O multiplicador de 24 X 24 bits fornece um produto de 48 bits. Os 25 bits mais significativos, que correspondem a parte da mantissa nos limites do formato e ao eventual overflow, são conectados à lógica de renormalização e arredondamento enquanto os restantes 23 bits entram na lógica de cálculo dos bits de arredondamento.

Há várias maneiras de implementar-se o multiplicador. No próximo capítulo serão explorados estes detalhes de projeto.

4.3.2.3 Somador dos expoentes

Paralelamente às operações anteriormente descritas realiza-se a adição dos expoentes. O resultado é dirigido para um circuito que corrige a distorção da polarização dupla, subtraindo um valor igual a uma polarização (127). Este expoente resultado provisório é conectado à lógica de compensação da renormalização.

Também em paralelo com estas operações realiza-se o cálculo do sinal do produto conforme descrito anteriormente.

4.3.2.4 Lógica de cálculo dos Bits de Arredondamento

A lógica de cálculo dos bits de arredondamento no multiplicador é bastante simples comparativamente àquela do somador/subtrator.

No multiplicador esta unidade recebe os 23 bits menos

significativos da saída do multiplicador e fornece à saída os 2 bits de arredondamento.

O mais significativo dos dois é o mais significativo da parte a ser arredondada. O outro corresponde ao OU-LÓGICO de todos os demais 22 bits. Com este procedimento, mais a lógica de arredondamento a precisão de arredondamento requerida pelo padrão está garantida.

4.3.2.5 Lógica de Renormalização e Arredondamento

Novamente este bloco arquitetural é consideravelmente mais simples que o seu correspondente na operação de adição e subtração.

A simplificação decorre do fato de nunca ocorrer underflow da mantissa na multiplicação pois sempre multiplicam-se mantissas maiores ou iguais a um.

No resultado de uma multiplicação pode ocorrer ou não overflow na mantissa. Em caso afirmativo a lógica de renormalização e arredondamento desloca o resultado um bit à direita juntamente com os bits de arredondamento. Este deslocamento é compensado pelo respectivo incremento do expoente resultado. Em seguida a mantissa - já normalizada - bem como os bits de arredondamento são dirigidos ao circuito de arredondamento.

O arredondamento é idêntico ao descrito na operação de adição e subtração. Um incremento condicional da mantissa resultado de acordo com os bits de arredondamento e o bit menos significativo da mantissa. Sempre que um dos bits de arredondamento for igual a um a exceção de operação inexata deve ser sinalizada à lógica de cálculo de exceção.

Após o arredondamento pode ocorrer overflow na mantissa resultado. Um outro circuito deslocador realiza a renormalização enquanto o expoente é incrementado para compensar o deslocamento.

4.3.2.6 Lógica de controle e cálculo de exceção

A lógica de controle de exceção recebe os sinais gerados pelos blocos de análise das mantissas e expoentes. Com estes sinais verifica-se se algum dos operandos é não-normalizado (0, Infinito, denormalizado ou NaN). Em caso afirmativo o resultado será determinado pela lógica de cálculo de exceção que recebe dos blocos de controle de exceção os valores de saída da mantissa e do expoente, da mesma forma que no operador de adição e subtração. Caso contrário, o resultado é determinado pelo procedimento de cálculo normal.

A lógica de cálculo de exceção recebe da lógica de controle de exceção um sinal que define se o resultado apresentado será o resultado calculado pelo procedimento normal ou definido neste bloco. Também são sinalizados os valores da mantissa e dos expoentes no caso de um resultado de exceção.

Caso a lógica de controle de exceção tenha sinalizado que o resultado deve ser estabelecido pela lógica de cálculo, se ocorrer overflow ou underflow do expoente o resultado será definido pela lógica de exceção.

Assim, a lógica de cálculo de exceção controla o resultado apresentado e também fornece os sinais de exceção: overflow, underflow e inexato.

À saída do circuito existem dois multiplexadores que comutam o resultado apresentado ao exterior entre: o resultado calculado e o resultado de exceção. Este procedimento permite que a lógica de exceção e a lógica de cálculo trabalhem paralela e independentemente sem necessidade de desabilitação de uma ou de outra, selecionando-se apenas, ao final da operação, o resultado correto, o que simplifica o controle.

4.4 Comparações e conclusão

No multiplicador os blocos de análise das mantissas e expoentes são iguais aos da arquitetura do somador e subtrator. No entanto, no multiplicador não há necessidade dos circuitos de comparação.

O multiplicador das mantissa é o bloco mais complexo da arquitetura, entretanto, a representação da mantissa no padrão IEEE (sinal-magnitude) favorece a implementação do multiplicador das mantissas ao passo que complica a correspondente do somador/subtrator das mantissas.

Os blocos associados às sub-operações de renormalização e arredondamento apresentam grandes diferenças. Nos referentes ao multiplicador existem dois circuitos deslocadores de 1 bit para a renormalização da mantissa, um incrementador da mantissa e um incrementador para o expoente. Conforme visto, na adição e subtração, tal bloco exige um deslocador de 24 bits para a esquerda e um de 1 bits para a direita, um detector de leading bit, um incrementador para a mantissa e um somador para o expoente.

A lógica de exceção, na multiplicação, trata todos os

casos em que há pelo menos um operando não-normalizado. Na adição e subtração, quando um dos operandos é normalizado e o outro é zero ou denormalizado, por simplicidade, o resultado é fornecido pela lógica de cálculo pois quando um dos operandos é zero ou denormalizado e o outro um normalizado o resultado é o próprio número normalizado. Para evitar que os operandos fossem também dirigidos para a lógica de exceção optou-se por realizar este caso via lógica de cálculo

Neste capítulo propos-se arquiteturas independentes para os operadores estudados no capítulo anterior. Nesta exposição determinou-se os recursos mínimos e a estrutura de interligação necessária a cada um deles, procurando obter o máximo paralelismo dos algoritmos e visualizando uma possível solução pipeline. Isto justifica a duplicação de algumas unidades que executavam funções semelhantes e que, em uma outra abordagem poderiam ser compartilhadas.

Abordou-se algumas questões particulares às implementações em hardware sem, no entanto, levar em consideração alguns aspectos como: estrutura de entrada e saída, área ocupada, potência máxima dissipada e nem mesmo a tecnologia em que se pretende implementar tais operadores.

5 IMPLEMENTAÇAO DE UM MULTIPLICADOR PONTO FLUTUANTE PIPELINE

5.1 Introdução

No capítulo anterior considerou-se as propostas arquiteturais para os operadores visando desempenho elevado e facilidades em estabelecer-se estágios pipeline. Não foram levados em consideração aspectos de caráter restritivo como interface e área, nem controle e temporização de forma explícita.

Neste capítulo parte-se da arquitetura do operador de multiplicação impondo-se as condições acima. Isto conduz a modificações na interface que por sua vez refletem-se na arquitetura.

O circuito é detalhado considerando-se uma implementação final em tecnologia CMOS. Os blocos funcionais são divididos em estágios pipeline.

Uma vez definido o bloco operativo do circuito, define-se a estrutura de controle e a temporização.

Conclui-se o estudo com a validação do circuito no nível de fases por simulação segundo o método HDC [SUZ 89].

5.2 Restrições de ordem prática e econômica

Em tecnologias integradas as funções de entrada e saída são realizadas através dos pinos, constituindo a única interface do circuito com o mundo exterior.

Conforme visto no capítulo 2, vários circuitos

aritméticos comerciais dedicados apresentam duas portas para entrada paralela de operandos e, também, uma porta de saída independente para os resultados, visando obter máximo aproveitamento da arquitetura pipeline. Considerando um caso típico: os circuitos ADSP 3221/3222 [ANA 87] possuem duas portas de entrada com 32 bits cada e uma porta de saída também com 32 bits. Somente a estrutura de entrada e saída de dados consome 96 pinos. Uma abordagem semelhante revela-se inviável em nosso contexto atual.

Além do problema da pinagem, circuitos dotados de semelhantes facilidades de entrada e saída deverão apresentar desempenho interno compatível, não devendo sub-utilizar o elevado investimento na interface. Logo, a implementação de operadores de alto desempenho, conforme infere-se a partir do capítulo anterior, exige soluções amplamente paralelas: multiplicadores arrays, barrel shifters, grandes barramentos, enfim um elevado investimento em área de circuito integrado, apresentando-se, portanto, como um outro fator limitante no projeto.

Neste trabalho busca-se uma solução que compatibilize estes vários fatores. Abrindo mão de um desempenho máximo reduz-se a pinagem a valores aceitáveis, bem como a área de circuito integrado, sem prejuízo excessivo na velocidade de computação.

5.3 O Multiplicador ponto flutuante pipeline

5.3.1 Entrada e Saída

Para que o número de pinos não ficasse muito elevado concedeu-se 32 pinos para realizar entrada e saída.

Uma primeira estratégia seria estabelecer os 32 pinos como uma porta bidirecioanal. Desta forma pode-se realizar uma estrutura pipeline capaz de fornecer um resultado a cada três ciclos: dois ciclos são dedicados à entrada dos operandos e no ciclo seguinte obtém-se a saída de um resultado (considerando o pipeline completo).

Uma outra abordagem para organizar os 32 pinos seria atribuir 16 deles a uma porta de entrada e os demais 16 a outra porta. Neste caso, para entrada de dois operandos de 32 bits, são necessários 4 ciclos. Logo, esta solução apresenta throughput inferior a anterior pois é possível fornecer, no máximo , um resultado a cada 4 ciclos.

Entretanto, a segunda estratégia apresenta algumas vantagens sobre a primeira:

a) Tanto o gerenciamento interno como externo dos barramentos é bastante simples pois ambos são unidirecionais, o que facilita a operação e teste do circuito.

b) Os pads são menores e mais simples: 16 pads de entrada e 16 de saída contra 32 bidirecionais.

c) Menor congestionamento no acesso aos pads. Para cada pad apenas sinais de entrada ou saída.

A única vantagem em optar-se por um barramento bidirecional de 32 bits é a possibilidade de conseguir um throughput de uma multiplicação a cada três ciclos.

Realizando uma contagem preliminar dos pinos necessários obtemos: 32 pinos para entrada e saída de operandos e resultados, 5 para os sinais de exceção (inválido, inexato, denormalizado, overflow e underflow), 2 para alimentação (V_{CC} e GND) e 1 para o clock o que perfaz um total de 40 pinos. Exatamente o número de pinos de um

encapsulamento padrão. Mas serão necessários mais alguns pinos para funções de controle e inicialização tais como Reset e Ready o que obrigará um encapsulamento com mais pinos (48 pinos, no mínimo).

Na segunda estratégia (16 pinos para entrada e 16 para saída) tem-se a possibilidade de realizar a saída dos sinais de exceção pelos mesmos pinos de saída do resultado pois estes são utilizados apenas 2 ciclos em cada 4. Um dos dois ciclos ociosos pode ser usado para a saída dos sinais de exceção, economizado cinco pinos e, portanto, viabilizando um encapsulamento de 40 pinos.

É possível, evidentemente, utilizar o barramento bidirecional da primeira estratégia para realizar, também, a saída dos sinais de exceção economizando, da mesma forma, os cinco pinos. Entretanto, neste caso, é necessário um ciclo extra para esta operação fazendo com que o throughput máximo caia para uma multiplicação a cada 4 ciclos, o mesmo da segunda estratégia, apresentando, portanto, nenhuma vantagem.

Estas razões levam a escolha da segunda estratégia para entrada e saída.

Conseqüências da estratégia de entrada e saída adotada

a) Redução da pinagem para entrada e saída de dados, viabilizando um encapsulamento de 40 pinos para o circuito permitindo pinos dedicados a funções de teste.

b) Limitação do throughput máximo em uma multiplicação a cada 4 ciclos (considerando que a cada ciclo meio operando entra no circuito e que o pipeline está cheio).

A estrutura de entrada dos semi-operandos para uma multiplicação é dada abaixo:

ciclo 1: metade mais significativa do operando A

ciclo 2: metade menos significativa do operando A

ciclo 3: metade mais significativa do operando B

ciclo 4: metade menos significativa do operando B

Esta seqüência é seguida pela metade dos operandos da próxima multiplicação no funcionamento pipeline conforme ilustrado no diagrama de tempos abaixo (Figura 5.1).

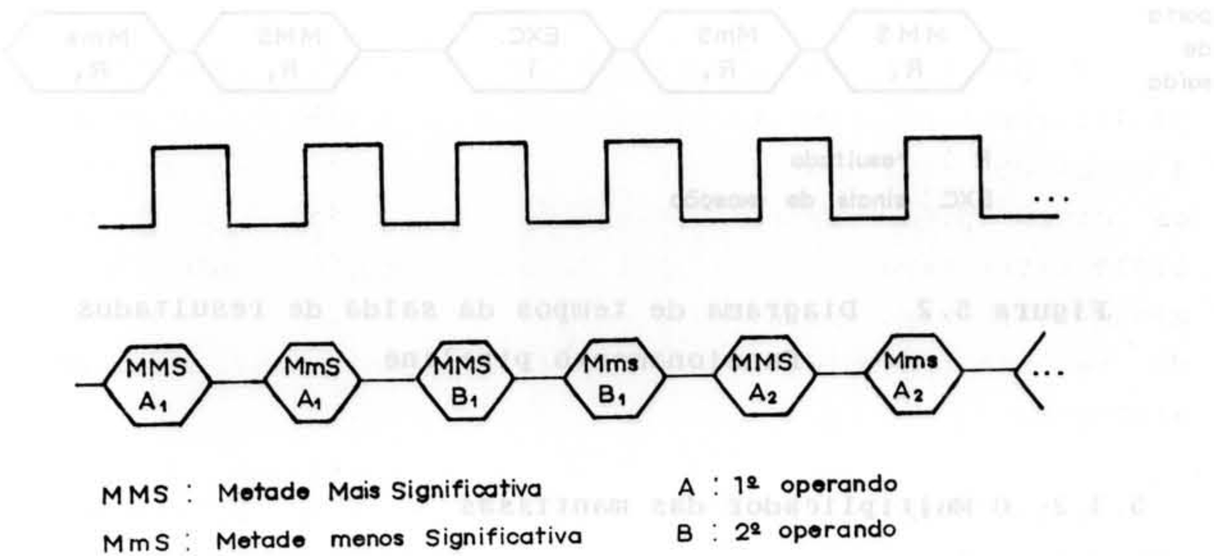


Figura 5.1 Diagrama de tempos da entrada de operandos Funcionamento pipeline

A ordem de saída do resultado e dos sinais de exceção é descrita abaixo:

ciclo 1: metade mais significativa do resultado

ciclo 2: metade menos significativa do resultado

ciclo 3: flags de exceção

ciclo 4: inativo

Da mesma forma que a entrada dos operandos, esta seqüência de sinais é seguida pelos sinais da próxima multiplicação, no funcionamento pipeline, conforme ilustrado no diagrama de tempos abaixo.

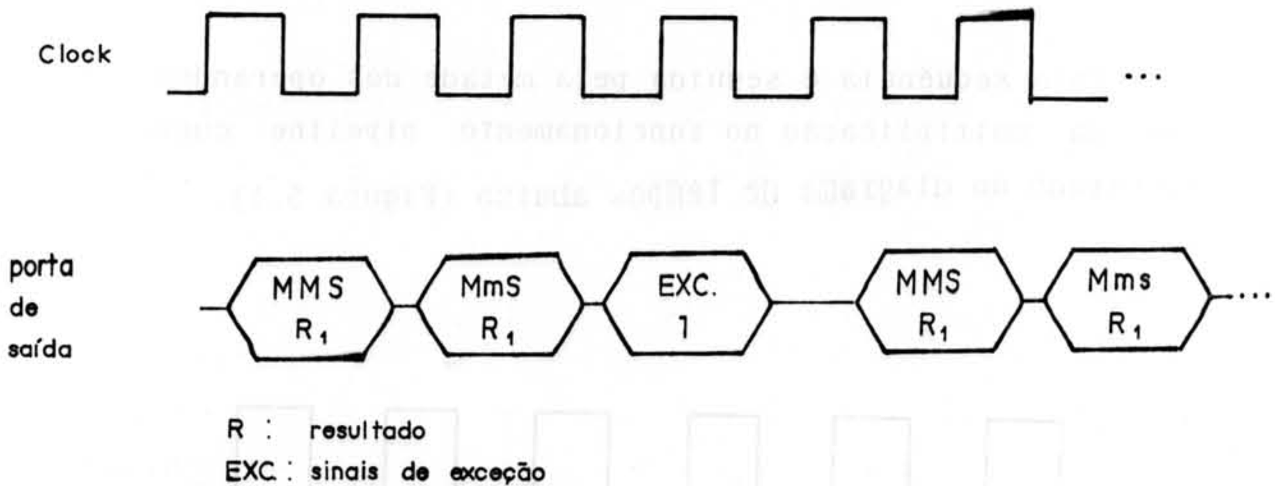


Figura 5.2 Diagrama de tempos da saída de resultados Funcionamento pipeline

5.3.2 O Multiplicador das mantissas

Uma vez definida a estratégia de entrada e saída do circuito e, conseqüentemente, neste caso, seu throughput máximo, procurar-se-á tirar o máximo proveito da situação diante destas restrições.

O multiplicador das mantissas é o bloco mais crítico do circuito sob vários aspectos, entre eles o tempo de latência e a área ocupada.

Uma solução direta seria utilizar um multiplicador array de 24x24 bits, eventuamente dividido ao meio por um

estágio de pipeline visando melhor distribuição nos tempos de latência de cada estágio. Esta escolha conduz, certamente, a um operador com tempo mínimo de latência na execução de uma operação. Entretanto, devido ao estrangulamento da porta de entrada (16 bits), o throughput máximo já está previamente definido.

No funcionamento pipeline (com fatores entrando constantemente no circuito e produtos saindo) a única vantagem em empregar um multiplicador array 24x24 bits seria para obter um menor tempo de latência na primeira multiplicação.

Observa-se, também, que nesta solução o multiplicador das mantissas é utilizado uma vez a cada 4 ciclos, permanecendo ocioso durante 3 ciclos, a espera da entrada dos próximos operandos. É intuitivo que, se há necessidade de esperar-se alguns ciclos para entrada dos operandos, seja possível utilizar este tempo para compartilhar alguns recursos, mantendo o mesmo throughput.

Estes fatos sugerem o estudo de uma solução que tire proveito destas restrições sem acarretar em prejuízos consideráveis ao desempenho do circuito.

Multiplicação no sistema binário

A multiplicação pode ser definida como uma adição repetida. O número a ser somado é o multiplicando, o número de vezes que ele é somado é o multiplicador e o produto, o resultado.

No entanto a adição repetida sugerida pela definição é tão lenta que é geralmente substituída por algoritmos que

fazem uso da representação posicional dos números.

A multiplicação de dois números binários realizada pelo processo de operações sucessivas de soma e deslocamento, idêntico ao empregado na aritmética decimal cotidiana, é ilustrado no exemplo numérico abaixo:

```
              1 0 1 1   Multiplicando
          X   1 1 0 1   Multiplicador
          -----------
              1 0 1 1
            0 0 0 0
                        Produtos Parciais
          1 0 1 1
        1 0 1 1
      ---------------
      1 0 0 0 1 1 1 1   Produto
```

Esquemas de multiplicação automáticos geralmente iniciam de maneira semelhante. Por exemplo, em um multiplicador array a primeira etapa é a geração paralela dos produtos parciais para posterior adição.

Circuitos multiplicadores

As alternativas de circuitos multiplicadores percorrem um vasto espectro de opções que vão desde os multiplicadores seriais até os arrays.

É intuitivo que à medida que se incrementa o hardware o desempenho bem como o custo aumentam.

De uma forma geral pode-se dividir os circuitos multiplicadores em duas categorias:

- multiplicadores de soma e deslocamento (add-shift multipliers)
- matrizes multiplicadoras (array multipliers)

O princípio dos primeiros é determinar seqüencialmente os produtos parciais que, então, vão sendo convenientemente acumulados até o último, quando o resultado é obtido.

Esquemas de recodificação permitem diminuir o número de produtos parciais a serem somados, acelerando a operação ao custo de hardware adicional para realizar tal recodificação [BOO 80] e [HWA 79].

Mesmo utilizando técnicas de recodificação, estes multiplicadores são lentos para a maioria das aplicações ditas científicas e de processamento de sinais.

Em aplicações que exigem desempenho maior são usadas as matrizes multiplicadoras ou multiplicadores array, que são circuitos que efetuam a operação de forma puramente combinacional. Os produtos parciais são gerados paralelamente e então somados conforme ilustrado na figura 5.3.

Existem várias técnicas para a soma dos produtos parciais, as mais conhecidas são Ripple-Carry Multiplier [CAN 86], [DAV 83], [GLA 85], Carry-Save Multiplier [CAN 86], [CAV 85], [DAV 83], [GLA 85] e que tem implementação mais direta e as Wallace trees [WAL 80].

Para desempenho ainda maior e redução da área de circuito são usadas técnicas de recodificação derivadas dos multiplicadores de soma e deslocamento [SHA 87], [PEN 87], [BOS 87].

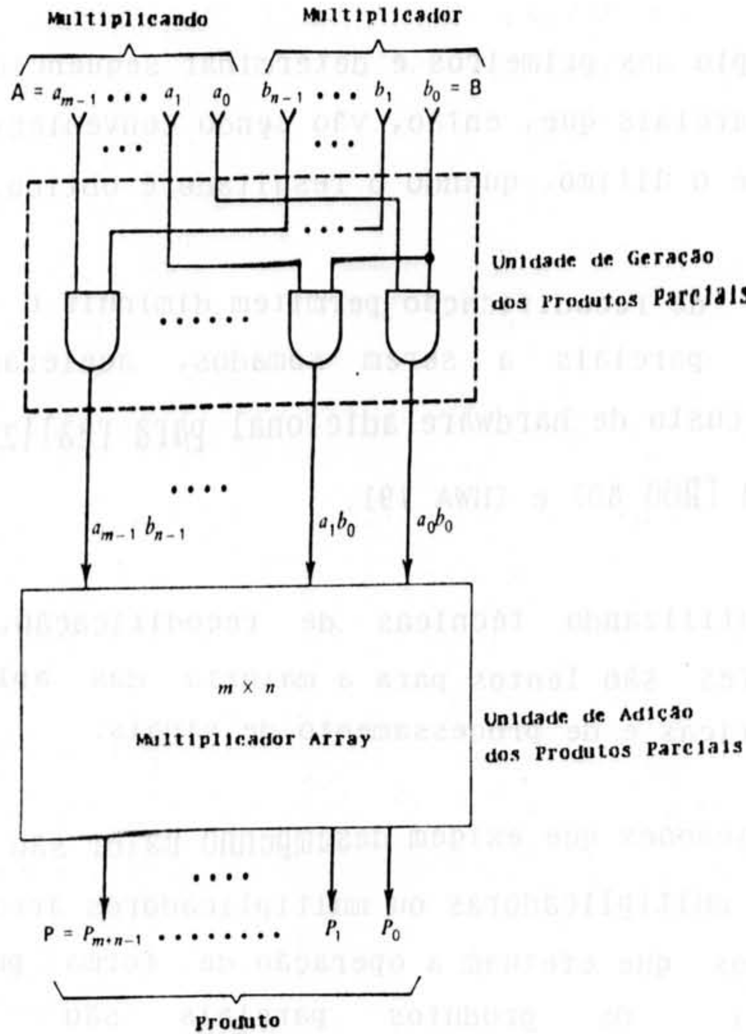


Figura 5.3 Multiplicador Array [HWA 79]

5.3.2.1 O Multiplicador proposto

Multiplicação Modular

No caso deste circuito objetiva-se um desempenho superior ao conseguido pelos multiplicadores de soma e deslocamento, o que induz a uma solução do tipo multiplicador array.

Conforme dito anteriormente, um multiplicador array de 24x24 bits seria sub-utilizado e conseqüentemente o investimento em área não adequadamente compensado.

O método aqui utilizado para diminuir o multiplicador deriva da técnica de ***multiplicação modular*** citada em Hwang [HWA 79], [HWA 84] e Cavanagh [CAV 85].

É assim chamada pois a operação é efetuada a partir de um conjunto de módulos que realizam a multiplicação sobre um número fixo de bits. Dependendo do tamanho da palavra vários módulos são necessários.

Explicitando em um exemplo: sejam A e B dois números binários de 8 bits, então:

$$A = A_h \cdot A_l$$
$$B = B_h \cdot B_l,$$

onde A_h e B_h são os 4 bits mais significativos de A e B e A_l e B_l os 4 menos significativos respectivamente e o ponto significa concatenação. Deste modo, a multiplicação fica:

$$P = A \times B$$
$$P = (A_h \cdot A_l) \times (B_h \cdot B_l)$$
$$P = (A_h \times B_h) + (A_h \times B_l) + (A_l \times B_h) + (A_l \times B_l)$$
$$P = P_{hh} + P_{hl} + P_{lh} + P_{ll}$$

Esta mesma operação é mostrada de forma explícita no figura 5.4.

Os módulos realizam as multiplicações fornecendo os produtos parciais paralelamente. Estes são enviados para módulos bit-slice de Carry Save Adders (CSAs) que adequadamente dispostos somam todos os produtos parciais paralelamente. A estrutura desta árvore de CSAs que soma os produtos parciais é a mesma das Wallace trees, assim, estes módulos de soma levam este nome.

Multiplicando A: a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

Multiplicador B: b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
								a_7b_0	a_6b_0	a_5b_0	a_4b_0	a_3b_0	a_2b_0	a_1b_0	a_0b_0
							a_7b_1	a_6b_1	a_5b_1	a_4b_1	a_3b_1	a_2b_1	a_1b_1	a_0b_1	
						a_7b_2	a_6b_2	a_5b_2	a_4b_2	a_3b_2	a_2b_2	a_1b_2	a_0b_2		
					a_7b_3	a_6b_3	a_5b_3	a_4b_3	a_3b_3	a_2b_3	a_1b_3	a_0b_3			
				a_7b_4	a_6b_4	a_5b_4	a_4b_4	a_3b_4	a_2b_4	a_1b_4	a_0b_4				
			a_7b_5	a_6b_5	a_5b_5	a_4b_5	a_3b_5	a_2b_5	a_1b_5	a_0b_5					
		a_7b_6	a_6b_6	a_5b_6	a_4b_6	a_3b_6	a_2b_6	a_1b_6	a_0b_6						
	a_7b_7	a_6b_7	a_5b_7	a_4b_7	a_3b_7	a_2b_7	a_1b_7	a_0b_7							

$A_H \times B_L$ $\quad A_L \times B_L$ $\quad A_H \times B_H$ $\quad A_L \times B_H$

Figura 5.4 Algoritmo para multiplicação de operandos de 8 bits utilizando módulos multiplicadores de 4 bits [CAV 85]

A técnica de multiplicação modular desenvolveu-se sobretudo na época da integração em média escala (MSI). Multiplicadores de alto desempenho eram implementados a partir de módulos multiplicadores de 4x4 bits chamados de Nonadditive Multiply Modulus (NMM) e de circuitos CSA com três entradas de 4 bits cada para realizar a árvore de soma dos produtos parciais, chamados de Wallace Tree Modulus. Pois a forma paralela como é computada a soma dos produtos parciais é inspirada no método de multiplicação paralela proposto por Wallace [WAL 80]. A partir destes dois módulos básicos, multiplicadores de qualquer dimensão são implementados.

Procedimento

Conforme visto até agora, devido às dimensões da porta de entrada, são necessários 4 ciclos para a entrada de dois operandos e um multiplicador de 24x24 bits puramente

combinacional ficaria ocioso durante 3 ciclos a espera de operandos.

Pelo aproveitamento dos 3 ciclos de ociosidade, sequencializando em parte a multiplicação, pode-se reduzir o hardware. Tal procedimento não diminui em nada o throughput do circuito, apenas o tempo de latência da primeira multiplicação de uma série.

Assim, empregando-se um multiplicador array de 12x12 bits em 4 ciclos são gerados os 4 produtos parciais, da mesma forma que na multiplicação modular com a diferença que aqui os produtos parciais são gerados sequencialmente, pois só existe um módulo, ao passo que lá são gerados paralelamente.

Ainda, os produtos parciais sendo gerados podem ser somados à medida que estejam disponíveis, de forma sequencial, compartilhando o recurso de adição, e não necessariamente de forma paralela como nas Wallace trees da multiplicação modular.

a) Geração dos produtos parciais

Os 4 produtos parciais são gerados sequencialmente por um multiplicador array de 12x12 bits iniciando-se pelas entradas menos significativas dos operandos. Há vantagens, em iniciar-se a geração pelos produtos parciais menos significativos, associadadas ao somador dos produtos parciais e a determinação dos bits de arredondamento.

Então sejam A e B dois operandos de 24 bits cada:

$$A = (a_{23}\ a_{22}\ \ldots\ a_1\ a_0)\ \text{e}$$
$$B = (b_{23}\ b_{22}\ \ldots\ b_1\ b_0)$$

$$A_h = (a_{23}\ a_{22}\ \ldots\ a_{13}\ a_{12}) \qquad A_l = (a_{11}\ a_{10}\ \ldots\ a_1\ a_0)$$
$$B_h = (b_{23}\ b_{22}\ \ldots\ b_{13}\ b_{12}) \qquad B_l = (b_{11}\ b_{10}\ \ldots\ b_1\ b_0)$$

Os produtos parciais são gerados na seguinte ordem:

$$(1)\quad P_{ll} = A_l \times B_l$$
$$(2)\quad P_{lh} = A_l \times B_h$$
$$(3)\quad P_{hl} = A_h \times B_l$$
$$(4)\quad P_{hh} = A_h \times B_h$$

b) Adição dos produtos parciais

Para obter a produto final os produtos parciais devem ser somados. É evidente que não se pode simplesmente somá-los, seu caráter posicional deve ser levado em conta.

O P_{ll} corresponde ao produto dos dois semi-operandos menos significativos. Os produtos parciais P_{lh} e P_{hl}, na adição, devem ser deslocados 12 bits à esquerda, tendo como referência o LSB do P_{ll}, pois correspondem a um produto de um semi-operando menos significativo com um semi-operando mais significativo. Finalmente o P_{hh} é deslocado 24 bits à esquerda em relação ao LSB do P_{ll} pois trata-se do produto parcial de dois semi-operandos mais significativos.

A figura 5.5 ilustra os 4 produtos parciais e os deslocamentos necessários para a correta soma final.

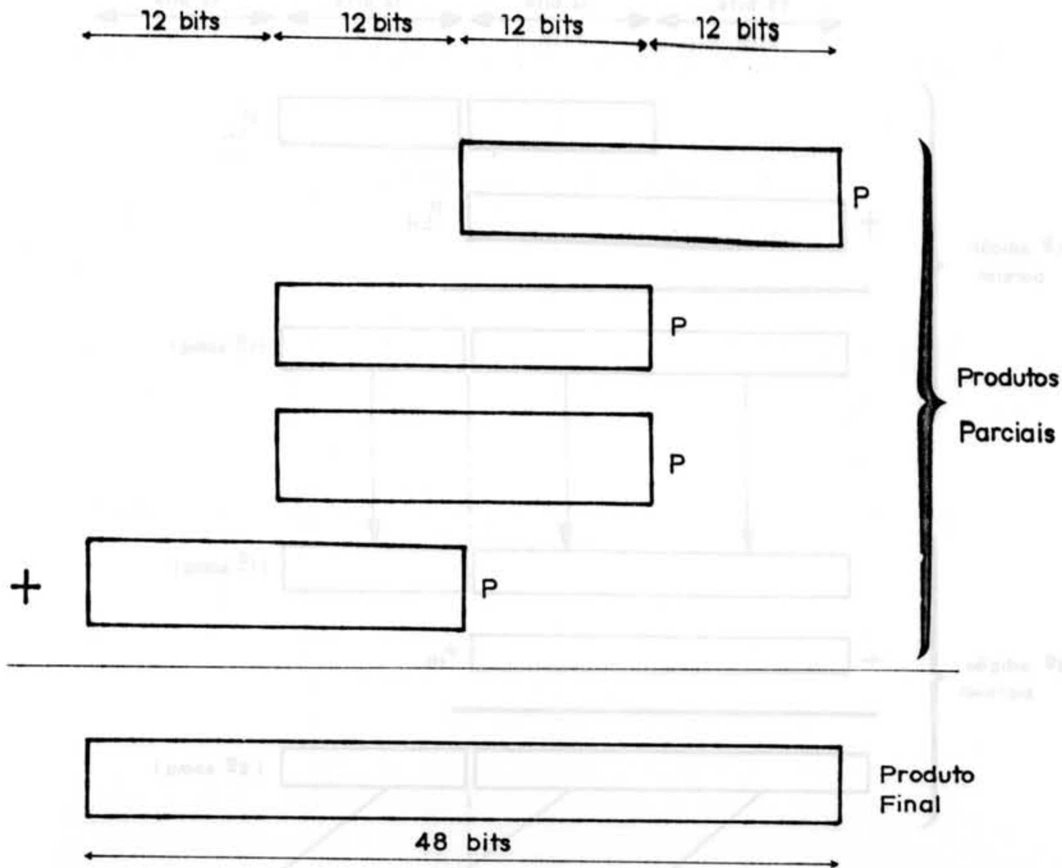


Figura 5.5 Adição dos produtos parciais

Como nesta proposta pretende-se compartilhar o somador adicionando os produtos parciais à medida que estes vão sendo calculados. Utiliza-se um somador de 24 bits e o procedimento encontra-se ilustrado na figura 5.6.

Na primeira adição parcial o produto parcial P_{ll} deslocado 12 bits à direita é adicionado ao P_{lh}. O resultado é adicionado ao produto parcial P_{hl}, na segunda adição parcial. Antes de efetuar-se a terceira e última adição parcial o resultado acumulado da anterior é deslocado 12 bits à direita.

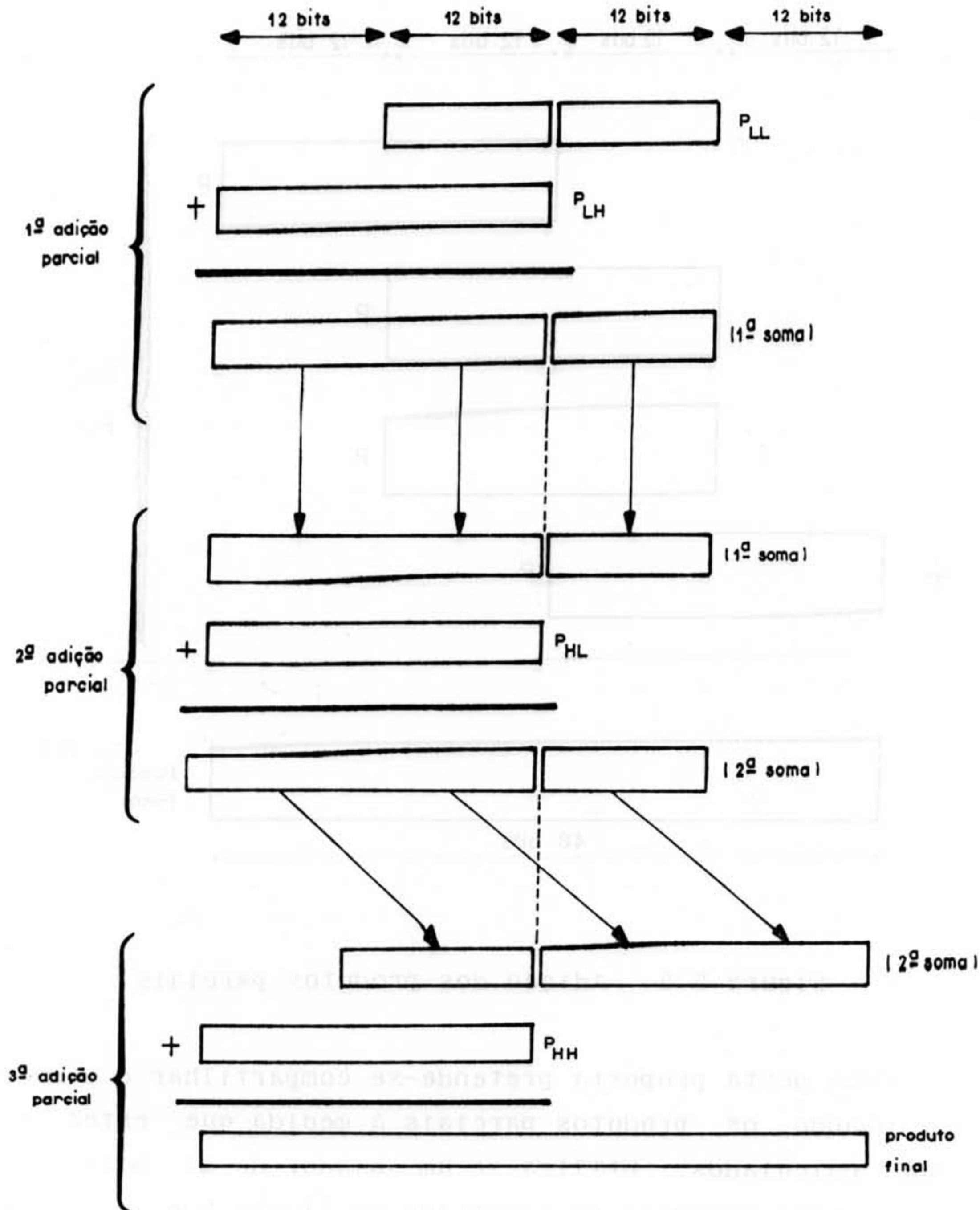


Figura 5.6 Procedimento para adição dos produtos parciais

Assim, deslocando e armazenando convenientemente os resultados intermediários é possível efetuar-se a computação do produto final conforme vão sendo gerados os produtos parciais, utilizando o mesmo recurso de adição.

Observa-se que caso fosse iniciada a geração dos produtos parciais pelo P_{hh} (mais significativo) haveria necessidade de um somador de 36 bits pois ocorreriam

situações onde a adição do último produto parcial (no caso o P_{11}) geraria um carry que se propagaria até os bits mais significativos do produto final.

c) Determinação dos bits de arredondamento

Antes de iniciar a discussão sobre os bits de arredondamento é importante reconhecer os componentes do produto final gerado pelo multiplicador.

A multiplicação de dois números binários de 24 bits resulta num produto com 48 bits. A mantissa resultado no formato, considerando o "leading" bit e um eventual overflow, ocupa 25 bits, os demais são bits que devem ser arredondados, conforme mostra a figura 5.7.

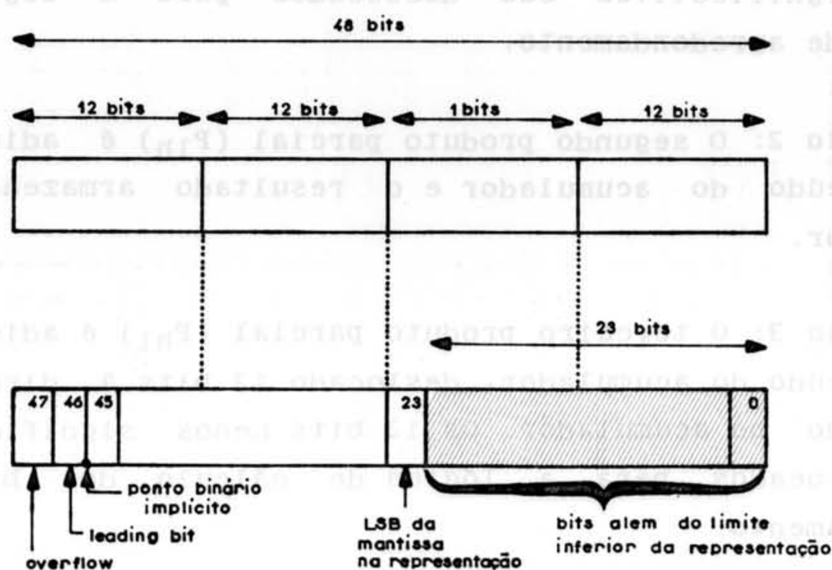


Figura 5.7 Componentes do produto final

Observa-se, a partir do item anterior, que os 12 bits menos significativos estão disponíveis a partir da primeira multiplicação e os demais bits da parte a ser arredondada após a segunda adição parcial. É possível, então, iniciar a computação dos bits de arredondamento antes do fim da

computação do produto final. Conforme descrito anteriormente (item 4.3.1) o procedimento para cálculo dos bits de arredondamento é bastante simples: o guard bit (o mais significativo) é igual ao MSB da parte a ser arredondada e o round bit igual ao OU-LÓGICO de todos os demais.

O procedimento de cálculo dos bits de arredondamento associado às adições parciais é ilustrado na figura 5.8.

Desta forma, o procedimento de multiplicação em 4 ciclos fica distribuído da seguinte forma:

ciclo 1: O primeiro produto parcial (P_{ll}) é deslocado 12 bits à direita e armazenado em um acumulador. Os 12 bits menos significativos são deslocados para a lógica de cálculo de arredondamento.

ciclo 2: O segundo produto parcial (P_{lh}) é adicionado ao conteúdo do acumulador e o resultado armazenado no acumulador.

ciclo 3: O terceiro produto parcial (P_{hl}) é adicionado ao conteúdo do acumulador, deslocado 12 bits à direita e armazenado no acumulador. Os 12 bits menos significativos são deslocados para a lógica de cálculo dos bits de arredondamento.

ciclo 4: O quarto produto parcial (P_{hh}) é adicionado ao conteúdo do acumulador fornecendo os 24 bits mais significativos da mantissa resultado. O bit menos significativo vem da lógica de cálculo dos bits de arredondamento, como mostra a figura 5.8.

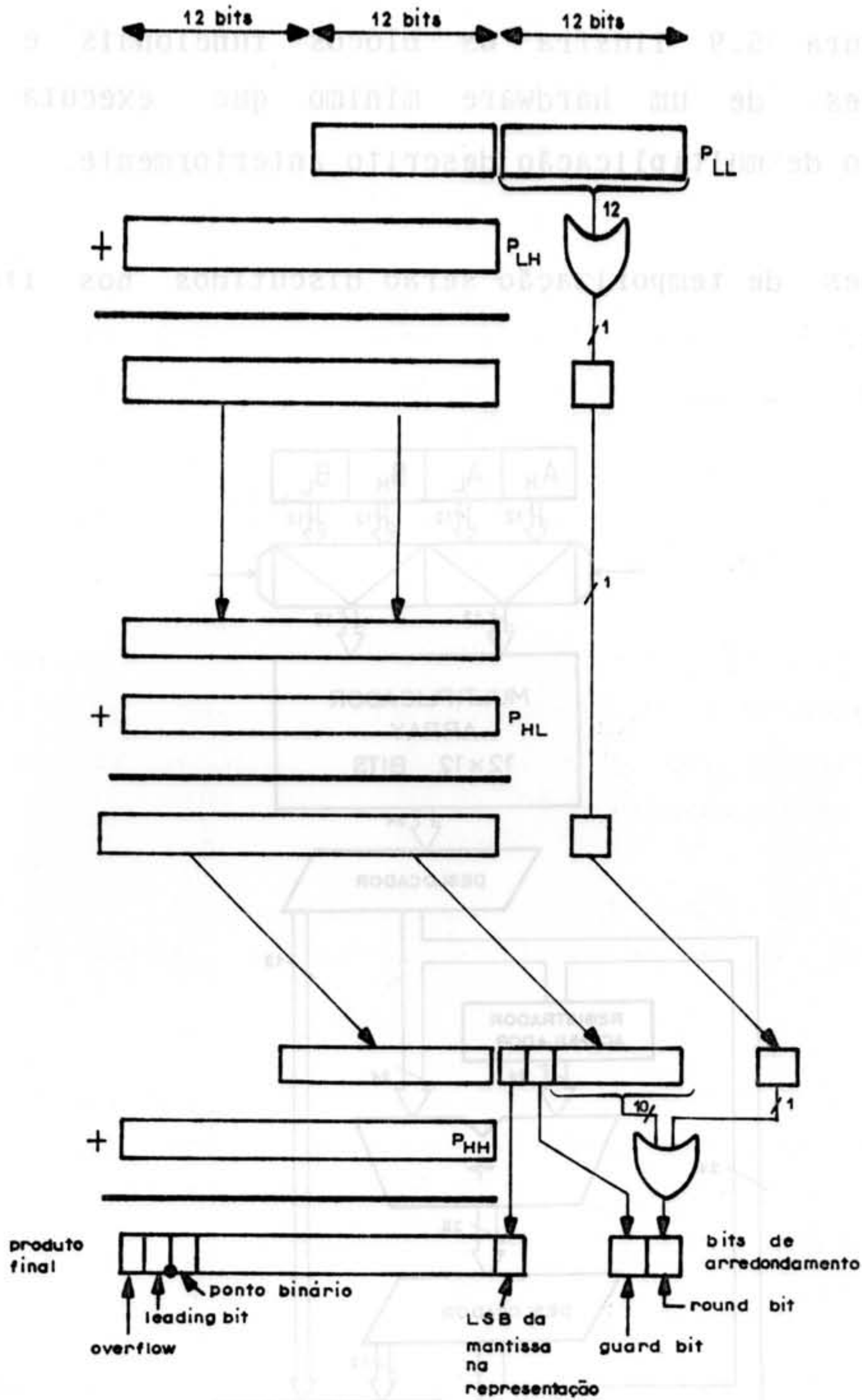


Figura 5.8 Procedimento de adição dos produtos parciais e cálculo dos bits de arredondamento

Hardware do multiplicador

A figura 5.9 ilustra os blocos funcionais e as interligações de um hardware mínimo que executa o procedimento de multiplicação descrito anteriormente.

Detalhes de temporização serão discutidos nos itens posteriores.

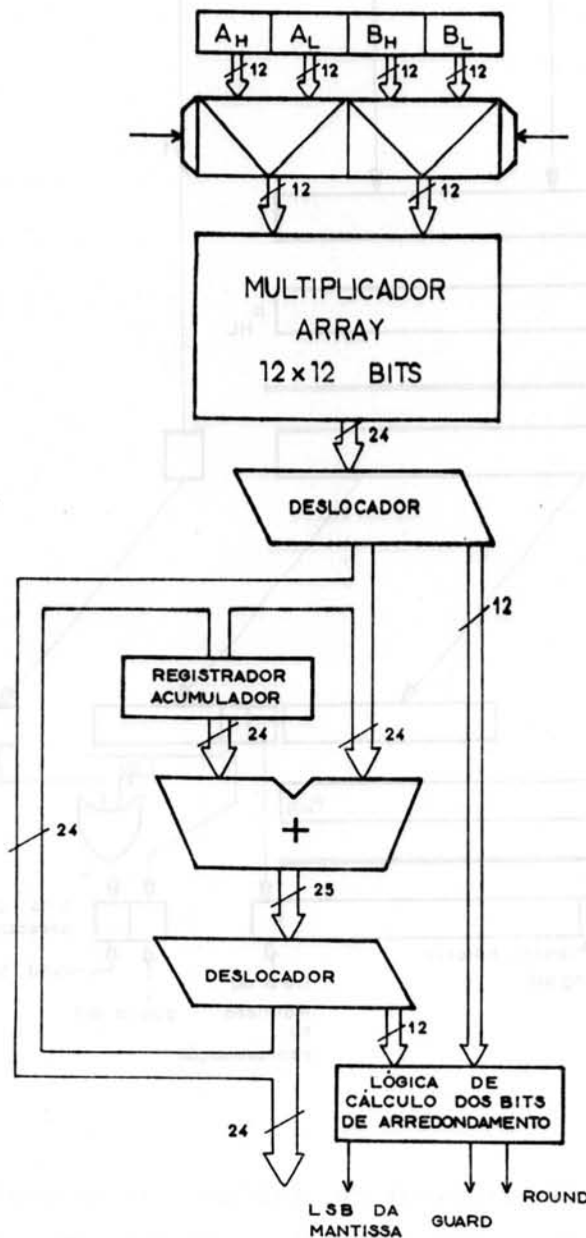


Figura 5.9 Hardware básico do multiplicador das mantissas

Como premissa do procedimento adotado está o multiplicador array de 12x12 bits que fornece os produtos parciais. Os multiplexadores selecionam convenientemente os semi-operandos.

O primeiro produto parcial (P_{11}) é direcionado ao deslocador para o deslocamento de 12 bits à esquerda. Os 12 bits mais significativos, deslocados para as posições menos significativas, são armazenados no acumulador enquanto os menos significativos encaminhados à lógica de cálculo dos bits de arredondamento.

As somas parciais passam pelo segundo circuito deslocador e são armazenadas no acumulador. O resultado da segunda soma parcial é deslocado 12 bits à esquerda e então armazenado. Os 12 bits deslocados para fora do acumulador são também dirigidos para a lógica de cálculo de arredondamento. Os demais resultados intermediários não sofrem deslocamento.

Considerações finais

A abordagem empregada para a realização da multiplicação, a saber: obtenção dos produtos parciais de um multiplicador array menor que o tamanho dos operandos é semelhante a apresentada por Hwang [HWA 79]. A diferença é que lá são empregados vários circuitos integrados que realizam todos os produtos parciais ao mesmo tempo e estes são, em seguida, somados em árvores de Wallace.

No caso do circuito proposto não há sentido em efetuar a soma dos produtos parciais através de uma árvores de Wallace pois tais produtos não são calculados paralelamente como em [HWA 79], e sim de forma seqüencial. Logo, a soma

dos produtos parciais é realizada conforme a sua geração, em um somador convencional de propagação de carry.

A solução adotada foi implementar-se um multiplicador array de 12x12 bits. Com este multiplicador são gerados 4 produtos parciais que são convenientemente somados para formar o produto final, conforme o exposto.

O multiplicador array 12x12 bits ocupa um quarto (1/4) da área de um multiplicador array de 24x24 bits. A lógica necessária para a soma dos produtos parciais não é grande.

Neste projeto o multiplicador está ativo a cada ciclo bem como a lógica para soma dos produtos parciais. A solução exige um controle um pouco mais complexo, mas a área é consideravelmente menor.

Apesar do aumento do tempo de latência da primeira multiplicação, em alguns ciclos, o circuito fornece o mesmo throughput, no funcionamento pipeline, que a solução com o multiplicador array 24x24 bits (considerando a mesma estrutura de entrada e saída), evidenciando a vantagem desta proposta sobre a outra.

5.3.3 A Lógica de Renormalização e Arredondamento

5.3.3.1 Renormalização

Conforme visto nos capítulos anteriores, na multiplicação ponto flutuante a renormalização deve ser realizada sempre que ocorrer overflow na mantissa resultado. Isto pode ocorrer em duas situações:

(1) após a multiplicação das mantissas e

(2) após o arredondamento.

O procedimento para realizar a renormalização é simplesmente deslocar toda a mantissa para a direita, de um bit, de forma que o bit que ocupara a posição de overflow passe a ocupar a posição do *leading* bit e os demais bits as respectivas posições à direita.

Caso a renormalização seja relativa a um overflow causado pela multiplicação das mantissas, os dois bits de arredondamento são deslocados juntamente com a mantissa. O LSB da mantissa passa a ocupar a posição do guard bit e os bits de arredondamento as posições à direita conforme ilustrado na figura 5.10.

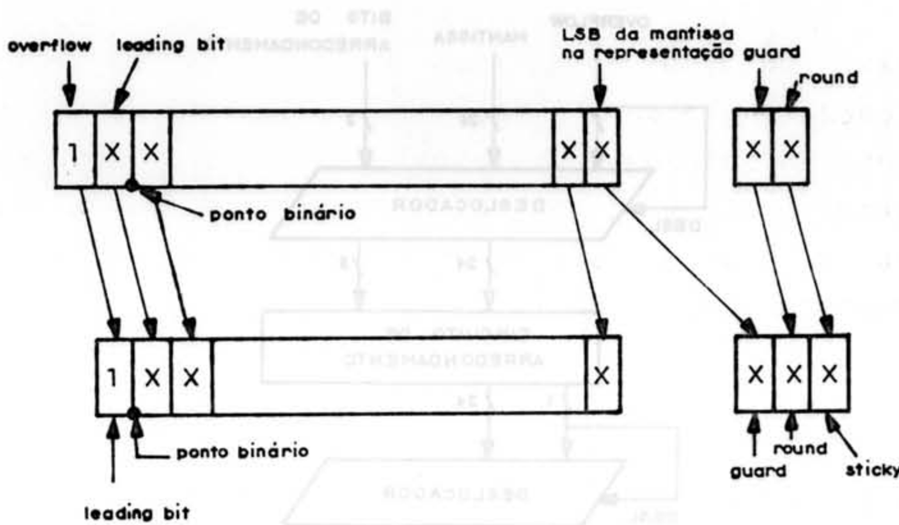


Figura 5.10 Renormalização na multiplicação ponto flutuante após a multiplicação das mantissas.

Para manter o resultado aritmeticamente correto o expoente deve ser incrementado quando renormaliza-se a mantissa.

O circuito de renormalização

O circuito de renormalização é simplesmente um deslocador. O sinal de controle, que é o próprio bit de overflow, seleciona ou não o deslocamento de um bit à direita.

A renormalização nos dois casos (imediatamente antes e imediatamente após o circuito de arredondamento) poderia ser implementada empregando apenas um deslocador. No entanto, a utilização de um deslocador antes e um após o circuito de arredondamento (vide figura 5.11) se revela mais interessante.

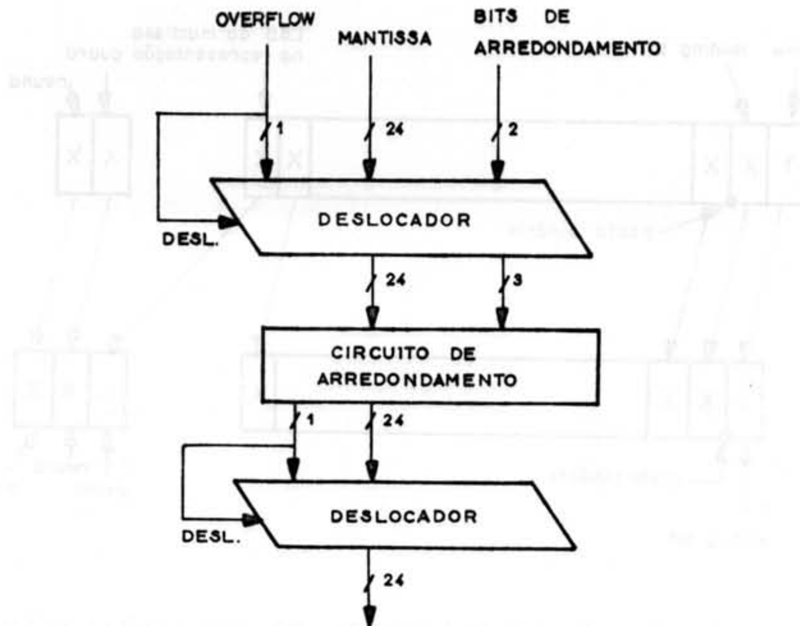


Figura 5.11 Circuito de Renormalização com dois deslocadores.

A razão é que quando se tem apenas um deslocador, é preciso criar uma estrutura que permita a utilização do

deslocador antes e após o circuito de arredondamento, ou seja, caminhos de dados selecionáveis antes e após o deslocador conforme mostra a figura 5.12.

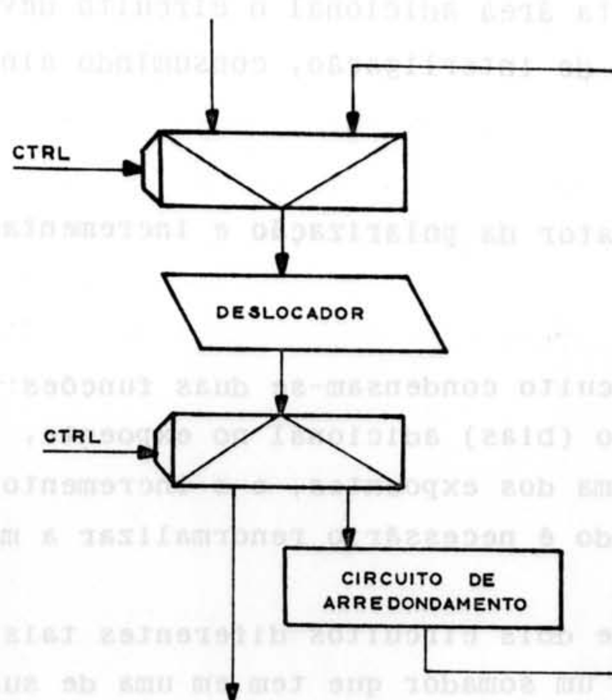


Figura 5.12 Circuito de Renormalização com um deslocador

Ora, o multiplexador e o deslocador, para um número de bits, apresentam praticamente o mesmo número de transistores. Entretanto, o multiplexador ocupa área maior pois é constituído por dois barramentos de saída e o deslocador apenas um. Tendo em conta que trata-se de barramentos de 24 bits, a diferença é considerável.

Outras desvantagens, em relação a alternativa com dois deslocadores, são a necessidade de um controle para seleção dos caminhos de dados e maior dificuldade na adaptação da estrutura à arquitetura pipeline.

Outra constatação é que o circuito com um deslocador faz uso, também, de um demultiplexador que tem a estrutura inversa do multiplexador.

Além desta área adicional o circuito deverá fornecer os barramentos de interligação, consumindo ainda mais área.

Circuito subtrator da polarização e incrementador do expoente

Neste circuito condensam-se duas funções: a subtração da polarização (bias) adicional no expoente, conseqüência da simples soma dos expoentes, e o incremento no expoente resultado quando é necessário renormalizar a mantissa.

Em vez de dois circuitos diferentes tais funções são aglutinadas em um somador que tem em uma de suas entradas o resultado da soma dos expoentes e na outra o valor negativo da polarização. O incremento é realizado através da entrada de carry-in.

Não se utiliza o mesmo somador de expoentes para estas funções pois o controle e o caminho de dados ficam mais diretos e simples, além de apresentar menos dificuldades na adaptação da estrutura à arquitetura pipeline.

5.3.3.2 Circuito de arredondamento

Conforme visto no item 4.2.1.2.2, o procedimento de arredondamento nada mais é do que um incremento na mantissa do resultado, condicionado aos valores dos três bits de arredondamento e do LSB da mantissa.

Assim, o circuito constitui-se de um incrementador e de uma lógica que determina o incremento ou não a partir dos bits de arredondamento e do LSB da mantissa, conforme ilustrado na figura 5.13. Inclui-se também o circuito que verifica a exceção de resultado inexato (OU-LÓGICO dos bits de arredondamento).

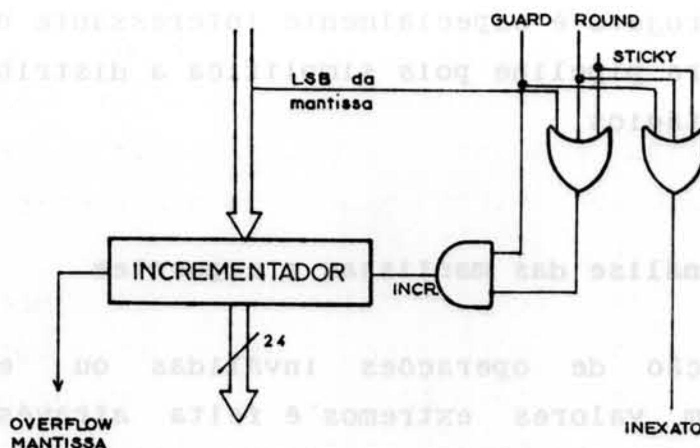


Figura 5.13 Circuito de arredondamento

5.3.4 A Lógica de exceção

Introdução

A lógica de exceção encarrega-se dos casos não tratados pelo procedimento de cálculo normal. São tratados pela lógica de exceção as seguintes situações:

(1) operações inválidas

a) quando algum dos operandos é NaN

b) quando ocorre uma multiplicação de zero ou denormalizado por infinito.

(2) multiplicações envolvendo operandos com valores extremos (0, infinito ou denormalizados).

(3) situações de overflow ou underflow do expoente.

Neste projeto, como descrito no capítulo anterior, as funções da lógica de exceção estão divididas em três circuitos: o circuito de análise das mantissas e expoentes, o circuito de controle de exceção e o circuito de cálculo de exceção. A divisão da lógica de exceção, além de facilitar o projeto é especialmente interessante no caso de uma arquitetura pipeline pois simplifica a distribuição dos blocos nos estágios.

Circuito de análise das mantissas e expoentes

A detecção de operações inválidas ou envolvendo operandos com valores extremos é feita através de uma análise dos operandos. Conforme visto no capítulo anterior todos os operandos "reservados" possuem expoente com valor reservado máximo ou mínimo. A mantissa fornece informação complementar a esta análise.

Todas as operações envolvendo operandos com estes expoentes reservados são tratadas pela lógica de exceção.

Assim, o circuito verifica se os expoentes de ambos os operandos possuem valor reservado máximo ou mínimo e verificam se algum apresenta mantissa nula.

O circuito para verificação do expoente reservado mínimo e da mantissa nula podem ser implementados através

do OU-lógico negado de todos os bits. De forma similar, para a verificação do expoente reservado máximo pode-se realizá-lo através de um E-lógico dos bits do expoente conforme ilustra a figura 5.14.

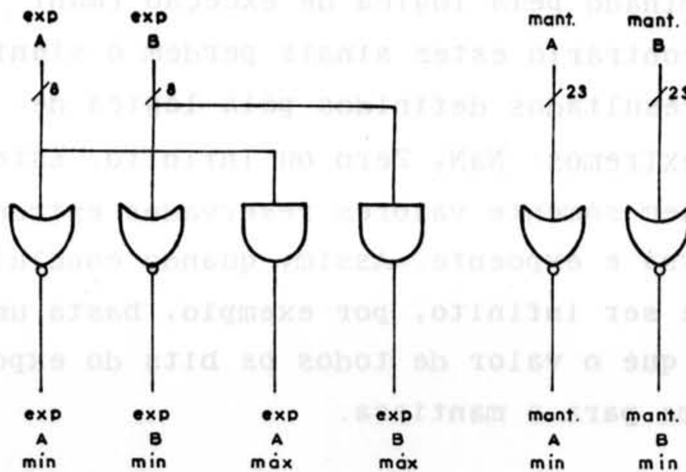


Figura 5.14 Circuito de análise das mantissas e expoentes

Circuito de controle de exceção

Este circuito parte do diagnóstico realizado pelo circuito de análise e determina:

a) se o resultado será o do circuito de cálculo normal ou o da lógica de exceção.

Sempre que houver um expoente com valor reservado máximo ou mínimo, o resultado será determinado pela lógica de cálculo de exceção. O sinal gerado é *selec exceção*.

b) se a operação é inválida.

Sempre que ocorrer uma operação envolvendo operandos NaN ou produto de zero ou denormalizado por infinito é sinalizada a exceção de operação inválida e o resultado

apresentado é um NaN com todos os bits da mantissa iguais a 1.

c) se algum operando é denormalizado.

d) o valor da mantissa e do expoente, caso o resultado deva ser determinado pela lógica de exceção (mant máx ou exp máx), caso contrário estes sinais perdem o significado.

Todos os resultados definidos pela lógica de exceção são resultados extremos: NaN, Zero ou Infinito. Estes três resultados possuem somente valores reservados extremos nos campos da mantissa e expoente. Assim, quando conclui-se que o resultado deve ser infinito, por exemplo, basta uma linha para sinalizar que o valor de todos os bits do expoente é 1, da mesma forma para a mantissa.

Este circuito é facilmente implementável em um pequeno PLA, com 6 entradas e 5 saídas. As equações lógicas referentes a cada sinal de saída, em função dos sinais de entrada (os sinais de saída dos circuitos de análise) encontram-se abaixo:

Equações lógicas do circuito de controle de exceção

```
1)  selec exceção = ( exp min A | exp min B |
                      exp max A | exp max B )

2)  inválido = (exp max A & (~(mant min A) | exp min B)) |
               (exp max B & (~(mant min B) | exp min A))

3)  denorm = (exp min A & ~(mant min A)) |
             (exp min B & ~(mant min B))

4)  mant max = inválido
```

5) exp max = ~((~(exp max A) & exp min B) |
(~(exp max B) & exp min A))

Circuito de cálculo de exceção

Este circuito recebe alguns sinais do circuito de controle de exceção e os flags de ocorrência de overflow ou underflow do expoente (do circuito de verificação) e de resultado inexato (do circuito de arredondamento), do circuito de cálculo normal.

A partir destes sinais o circuito de cálculo de exceção determina se o resultado apresentado será o calculado ou o de exceção (neste caso definindo os valores da mantissa e do expoente de exceção) e sinaliza as exceções de overflow, underflow e inexato, quando for o caso.

Pode-se dividir este procedimento em dois casos:

a) quando o circuito de controle de exceção sinaliza que o resultado deve ser determinado pelo circuito de cálculo de exceção (selec exceção=1).

Neste caso os flags de overflow e underflow do expoente e de resultado inexato são desconsiderados, pois trata-se de uma operação inválida ou de um resultado zero ou infinito. Seleciona-se resultado de exceção e os valores da mantissa e expoente são os determinados pelo circuito de controle de exceção.

b) quando o circuito de controle de exceção não seleciona resultado de exceção (selec exceção=0), então são analisados os sinais de overflow e underflow. Caso algum deles esteja ativado o resultado é determinado pelo

circuito de cálculo de exceção, caso contrário o resultado apresentado é o do circuito de cálculo normal.

De maneira análoga ao circuito de controle de exceção, este circuito é facilmente implementável em um PLA ou mesmo em lógica aleatória. As equações lógicas encontram-se abaixo.

Equações lógicas do circuito de cálculo de exceção

1) result exceção = (selec exceção | overf exp | underf exp)

2) inexato = ((inexato calc | overf exp | underf exp) & ~(selec exceção))

3) overflow = (overf exp & ~(selec exceção))

4) underflow = (underf exp & ~(selec exceção))

5) mant exc max = (selec exceção & mant max)

6) exp exc max = ((selec exceção & exp max) | (overf exp & ~(selec exceção)))

5.3.5 O pipeline

No capítulo anterior mencionou-se a intenção de implementar-se as arquiteturas segundo uma estrutura pipeline, conforme alguns circuitos analisados no capítulo 2. O pipeline oferece uma forma econômica de realizar paralelismo temporal [HWA 84].

O princípio é dividir as tarefas em estágios isolados

por latches que são habilitados segundo um mesmo sinal de clock. À medida que os ciclos vão ocorrendo as informações de cada estágio avançam para o próximo até que a operação se complete. Assim, uma vez cheio o pipeline, ou seja, todos os estágios processando informação válida, os resultados são fornecidos a uma taxa constante do circuito.

Distribuição da parte operativa nos estágios pipeline

A divisão da parte operativa em estágios pipeline deve ser tal que os tempos de propagação em cada estágio sejam aproximadamente os mesmos pois a taxa de avanço do pipe é definida a partir do estágio mais lento.

No circuito multiplicador ponto flutuante proposto encontram-se dois blocos críticos em termos de tempo de propagação: o multiplicador array e o somador de produtos parciais.

O multiplicador array é um circuito combinacional e o tempo de propagação depende da técnica escolhida para implementação.

O somador de produtos parciais tem o tempo de propagação composto pelos tempos de propagação da sequência de sub-operações realizadas em um ciclo, tipicamente: adição de produtos parciais, deslocamento e armazenamento no acumulador.

O somador de produtos parciais não é um bloco puramente combinacional. Este aspecto cria um descompasso entre as sub-operações realizadas sobre as mantissas e as realizadas sobre os expoentes e sinais (executadas em um ciclo em cada estágio). Para manter os resultados parciais

referentes a uma mesma operação sincronizados, os dados referentes às sub-operações sobre expoentes e sinais devem esperar a conclusão da computação do produto final pelo somador de produtos parciais para avançar ao próximo estágio.

A não superposição de dados referentes a operações distintas é garantida pois a técnica de multiplicação por soma de produtos parciais aqui utilizada foi desenvolvida baseando-se nos ciclos necessários para entrada dos operandos. Conforme especificado nos itens anteriores, são necessários 4 ciclos para entrada de um par de operandos referentes a uma multiplicação.

A seguir descreve-se os blocos operacionais de cada estágio pipeline. A figura 5.15 ilustra a arquitetura pipeline simplificada. As barras horizontais simbolizam os latches de pipeline.

Primeiro estágio

Neste estágio realiza-se a entrada, a separação dos campos de sinal, expoente e mantissa e o armazenamento dos operandos. A cada ciclo um semi-operando (16 bits) é carregado através do barramento de entrada permanecendo temporariamente armazenado em um registrador intermediário, até que todos os 4 semi-operandos referentes a esta operação estejam no circuito. Os dois circuitos são, então, transferidos para um outro registrador, quando reinicia o processo com a entrada dos semi-operandos referentes à próxima operação.

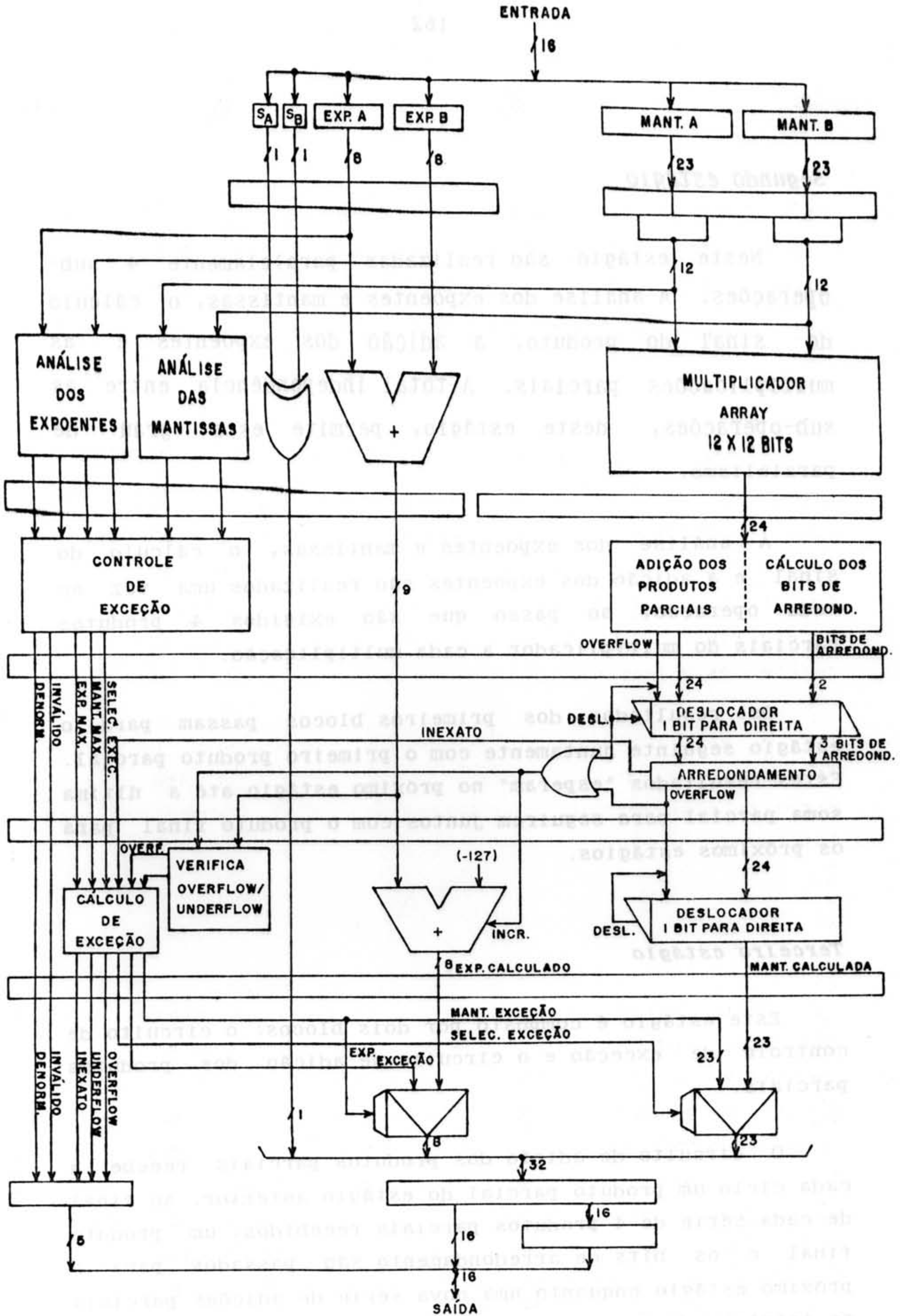


Figura 5.15 Arquitetura simplificada do circuito multiplicador ponto flutuante pipeline

Segundo estágio

Neste estágio são realizadas paralelamente 4 sub-operações. A análise dos expoentes e mantissas, o cálculo do sinal do produto, a adição dos expoentes e as multiplicações parciais. A total independência entre as sub-operações, neste estágio, permite este grau de paralelismo.

A análise dos expoentes e mantissas, o cálculo do sinal e a adição dos expoentes são realizados uma vez em cada operação, ao passo que são exigidos 4 produtos parciais do multiplicador a cada multiplicação.

Os resultados dos primeiros blocos passam para o estágio seguinte juntamente com o primeiro produto parcial. Estes resultados "esperam" no próximo estágio até a última soma parcial para seguirem juntos com o produto final para os próximos estágios.

Terceiro estágio

Este estágio é composto por dois blocos: o circuito de controle de exceção e o circuito de adição dos produtos parciais.

O circuito de adição dos produtos parciais recebe a cada ciclo um produto parcial do estágio anterior. Ao final de cada série de 4 produtos parciais recebidos, um produto final e os bits de arredondamento são passados para o próximo estágio enquanto uma nova série de adições parciais se inicia.

O circuito de controle de exceção que recebe os sinais

gerados pelo circuito de análise dos expoentes e mantissas é capaz de fornecer os sinais de saída em um ciclo. Estes sinais, bem como a soma dos expoentes e o sinal do produto (+/-) "esperam" nos latches até que o produto final esteja totalmente computado.

A partir deste ponto não há mais "espera". A cada ciclo se avança um estágio.

Quarto estágio

Neste estágio executa-se apenas a primeira renormalização e o arredondamento do produto final. Nenhuma outra operação pode ser executada pois todas as demais dependem da ocorrência ou não de overflow no produto final ou após o arredondamento e sua conseqüente necessidade de renormalização.

Quinto estágio

Neste estágio é realizada a subtração da polarização adicional do expoente resultado, bem como o incremento do expoente no caso de renormalização.

O bloco de verificação de overflow e underflow do expoente recebe o valor da soma dos expoentes e o sinal de incremento do expoente determinando a ocorrência de uma das exceções e transmitindo esta informação ao circuito de cálculo de exceção. A partir destes sinais e os outros do circuito de controle de exceção seleciona-se entre o resultado calculado e o de exceção.

Também neste estágio realiza-se a última

renormalização referente a um eventual overflow no arredondamento.

Considerando as operações que devem ser realizadas durante um ciclo no circuito de adição dos produtos parciais é possível acomodar toda a lógica de renormalização e arredondamento no quarto estágio. Entretanto, como não há operação alguma sendo realizada sobre as mantissas neste estágio e o deslocamento desta sub-operação não acarreta em atraso algum para as demais sub-operações, separa-se o segundo circuito de renormalização do circuito de arredondamento pelo latch do pipe facilitando ambos os blocos sem qualquer prejuízo.

Sexto estágio

Neste estágio é realizada a seleção entre o resultado calculado e o resultado de exceção, após é executada a concatenação dos campos do resultado numa só palavra de 32 bits, segundo o formato do padrão IEEE. Ainda neste ciclo é feita a saída da metade mais significativa do resultado, no ciclo seguinte a outra metade e no seguinte a palavra com os sinais de exceção.

5.3.6 A Lógica de Controle e Temporização

Uma vez definidos os recursos de processamento e interligações da parte operativa, conhecidos os requisitos de seqüenciamento para execução das operações com esta unidade operativa é estabelecida a estrutura de controle que fornece os sinais necessários para o correto procedimento.

As restrições nos barramentos de entrada e saída impõem uma seqüencialização de 4 ciclos para entrada dos dois operandos referentes a uma multiplicação.

Diante deste fato decidiu-se seqüencializar a multiplicação das mantissas pois reduziria a área ocupada sem detrimento do throughput.

A seqüencialização da multiplicação das mantissas foi desenvolvida com referências aos ciclos de entrada dos semi-operandos para que se pudesse sincronizar as duas sub-operações utilizando a mesma estrutura de controle e viabilizando uma solução pipeline simples.

A estrutura de controle, comum à seqüência de entrada dos dados e à seqüência de etapas da multiplicação, se estende ao restante do circuito, mesmo à estrutura pipeline. Isto já estava implícito quando mencionou-se, anteriormente, que os resultados intermediários dos expoentes e sinais, referentes a uma multiplicação "esperavam" a conclusão da multiplicação das mantissas.

A Máquina de Estados

Para seqüencializar a entrada de dados e o multiplicador das mantissas estabelece-se uma máquina com 4 estados. Cada um referindo-se a entrada de um semi-operando e também a uma etapa da multiplicação das mantissas.

Tratando-se de uma arquitetura pipeline, após o quarto e último estado reinicia-se um novo ciclo a partir do primeiro estado.

Esta máquina de estados encontra-se ilustrada na figura 5.16. Incorporou-se um sistema de *reset* que permite trazer a máquina ao estado inicial a partir de qualquer estado.

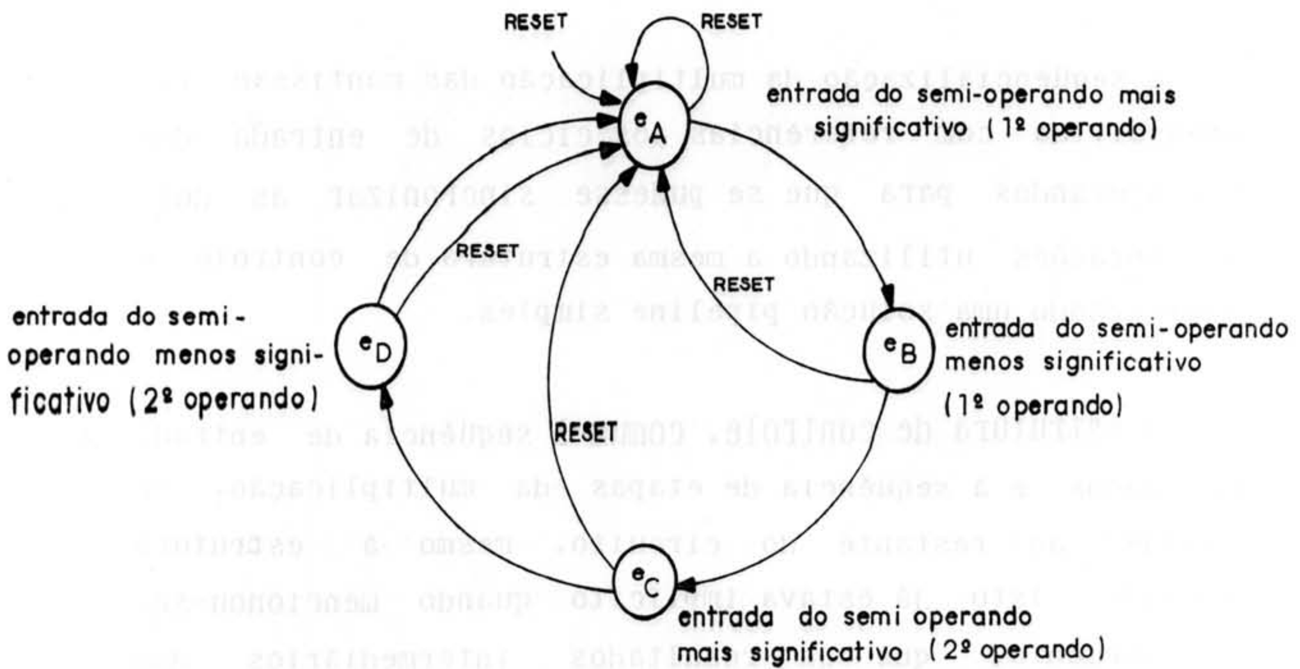


Figura 5.16 Máquina de estados da entrada de operandos

Na figura 5.17 encontramos a mesma máquina de estados para a seqüência de entrada de dados mas agora com as etapas da multiplicação das mantissas.

Assim, cada ciclo referido nos itens 5.3.1 (Entrada e Saída) e 5.3.2 (Multiplicador das Mantissas) refere-se a um estado das máquinas acima.

É necessário, agora, estender esta estrutura de controle para todo o circuito, toda a arquitetura pipeline. Entretanto, para isso, será necessário refinar a temporização, estabelecendo as barreiras temporais de cada estágio.

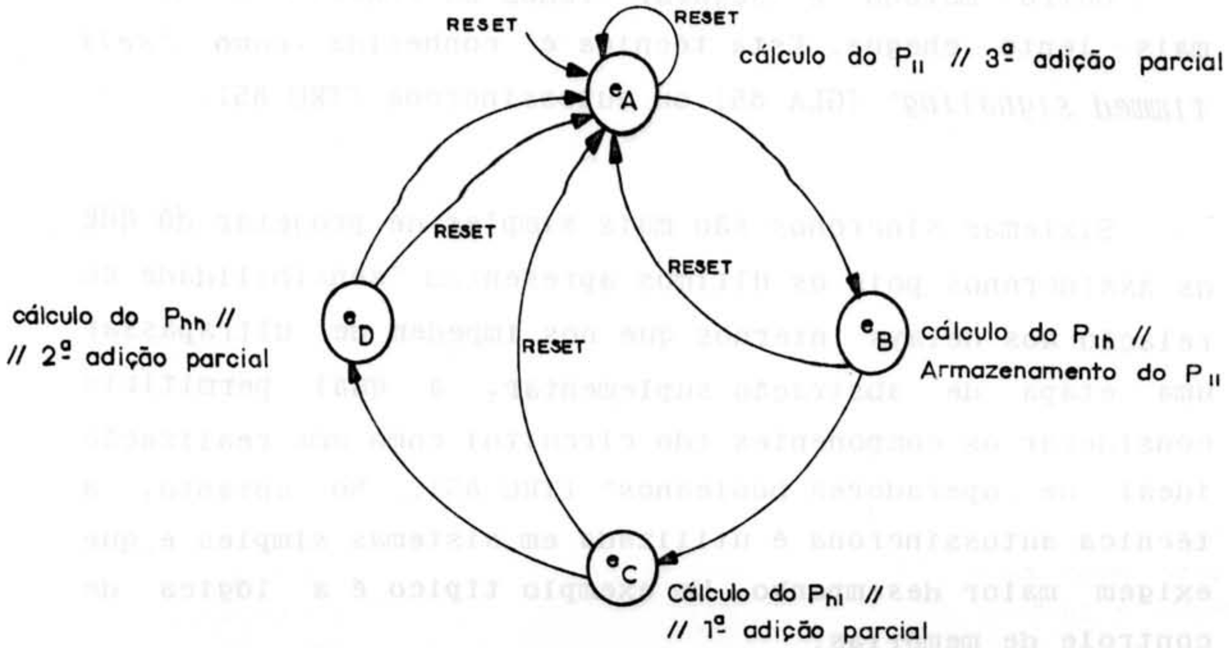


Figura 5.17 Máquina de estados da multiplicação das mantissas

Temporização

Em um sistema digital, para que um sinal influencie outro eles devem intersectar-se não só no espaço mas também no tempo. Logo, é preciso estar certo do exato momento em que se pode interpretar uma forma de onda analógica (como o são, de fato, todos os sinais) como a representação de um nível lógico.

E necessário uma forma de trazer os sinais juntos no tempo, sincronizá-los, i.e., desenvolver uma técnica para tratar com as incertezas de delay.

Um dos métodos, conhecido como "*clocked*" [GLA 85] ou síncrono [TRU 85], iguala todos os delays "segurando" (holding up) periodicamente todos os sinais, ou seja,

iguala-os tornando-os igualmente lentos.

Outro método é "segurar" todos os sinais até que o mais lento chegue. Esta técnica é conhecida como "*self timmed signaling*" [GLA 85] ou autossíncrona [TRU 85].

Sistemas síncronos são mais simples de projetar do que os assíncronos pois os últimos apresentam "sensibilidade em relação aos delays internos que nos impedem de ultrapassar uma etapa de abstração suplementar, a qual permitiria considerar os componentes (do circuito) como uma realização ideal de operadores booleanos" [TRU 85]. No entanto, a técnica autossíncrona é utilizada em sistemas simples e que exigem maior desempenho. Um exemplo típico é a lógica de controle de memórias.

Existem várias opções para temporização com clocks: fase simples, duas fases não sobrepostas, três, quatro ou mais fases. Por simplicidade de projeto e por bem adequar-se aos seqüenciamentos deste trabalho optou-se por um clock de duas fases não sobrepostas. As formas de onda de um clock com as características acima são mostradas na figura 5.18.

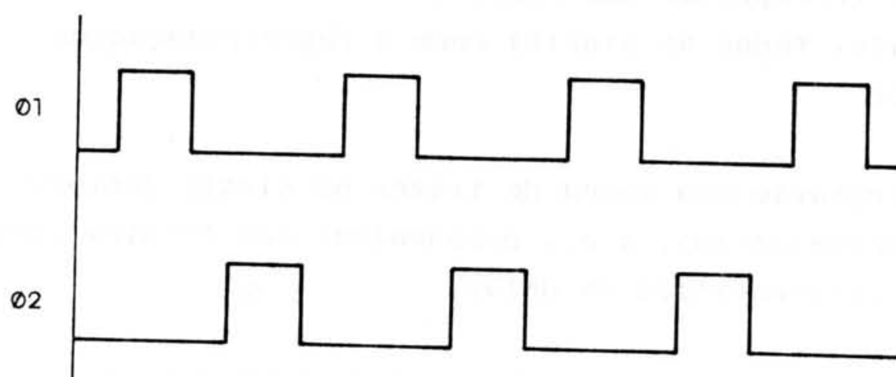


Figura 5.18 Formas de onda de um clock de duas fases não sobrepostas

Pré-carga

Pré-carga é uma técnica comum e especialmente interessante em circuitos MOS. Consiste em levar um nodo do circuito ao nível lógico alto ou baixo, permanecendo este valor armazenado em capacitâncias parasitas inerentes às tecnologias MOS. Um circuito de pull up ou pull down, respectivamente, carrega ou descarrega, respectivamente, o nodo pré-carregado condicionalmente aos sinais de entrada neste circuito.

As duas etapas 1) pré-carga do nodo e 2) avaliação, ou seja, carregamento ou descarregamento condicional do nodo, são normalmente realizadas em fases distintas.

Os circuitos implementados segundo esta técnica são referidos como dinâmicos.

Esta técnica é especialmente interessante para compactação de lay out onde os melhores exemplos são os grandes arrays como PLAs, ROMs e RAMs [GLA 85].

Glasser [GLA 85] mostra várias combinações de interligação entre módulos combinacionais e barramentos a pré-carga utilizando um clock de duas fases não sobrepostas onde em uma fase é feita a pré-carga e na outra a avaliação.

A interligação de dois módulos combinacionais a pré-carga considerando o clock anterior é mostrada na figura 5.19.

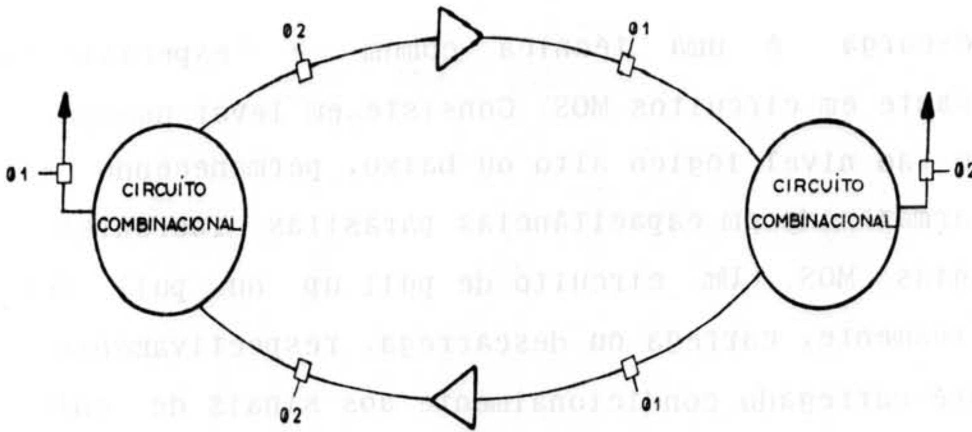


Figura 5.19 Um circuito com dois módulos a pré-carga [GLA 85]

Extrapolando esta contrução para vários módulos a pré-carga interligados e abrindo o laço, obtemos uma estrutura pipeline conforme mostra a figura 5.20 abaixo, onde os buffers e os *transmission gates* são os latches e os módulos combinacionais os estágios.

E interessante viabilizar a implementação de alguns blocos dinâmicos do circuito multiplicador e também, sendo intenção construir uma arquitetura pipeline adota-se esta estratégia de temporização para o circuito.

No entanto, conforme exposto anteriormente, o multiplicador ponto flutuante pipeline que se propõe implementar não é constituído apenas de estágios combinacionais. O circuito somador de produtos parciais é um circuito seqüencial que requer 4 ciclos para somar um produto final.

A interligação anterior, entretanto, permite a inclusão de circuitos seqüenciais, lembrando que a primeira estrutura mostrada (Figura 5.19) é um circuito seqüencial.

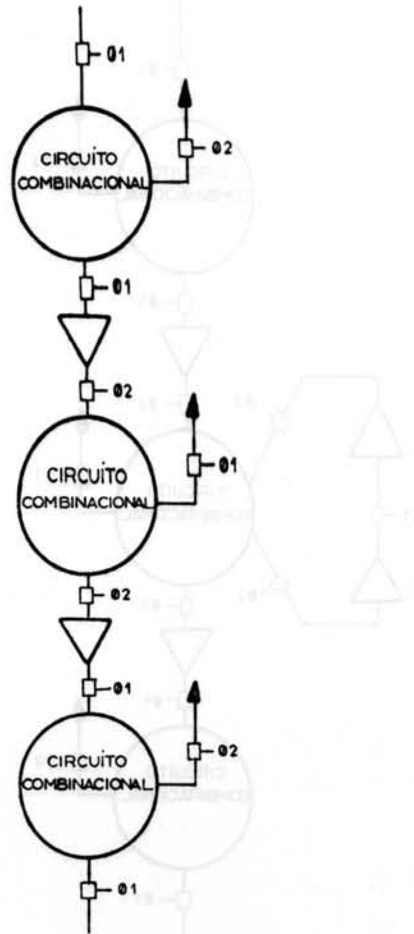


Figura 5.20 Estrutura pipeline com módulos combinacionais a pré-carga

Assim, de uma forma genérica, apresentamos a estrutura geral de temporização adotada no projeto na figura 5.21.

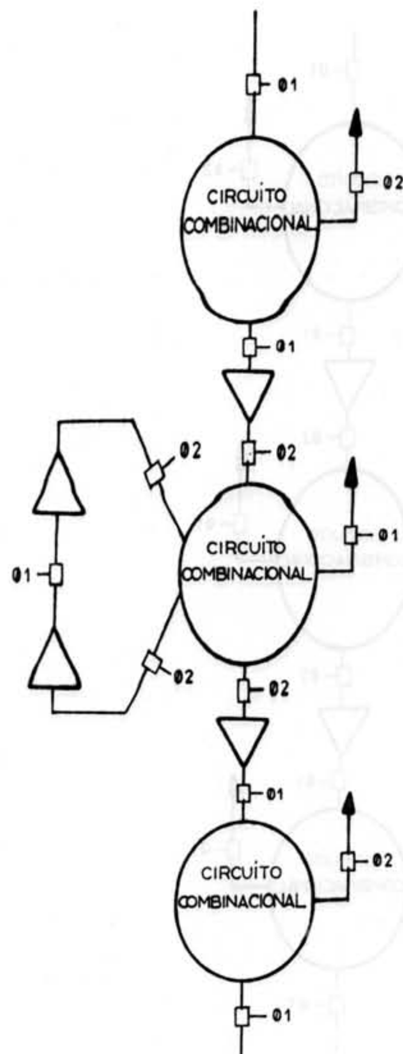


Figura 5.21 Estrutura pipeline com módulos combinacionais e seqüenciais a pré-carga.

A Temporização do circuito

Então, segundo o esquema visto, estabelece-se a temporização com um clock de duas fases não sobrepostas.

Iniciando-se pela entrada de dados: a entrada dos semi-operandos se dará a cada fase FI1. Para distribuir corretamente os semi-operandos referentes a cada metade e a cada fator é necessário um conjunto de sinais cíclicos que identifiquem e controlem o armazenamento correto dos dados de entrada. Desta forma são criados 4 sinais de controle a partir do clock que, em conjunto (clock e sinais de

controle), são capazes, como veremos, de realizar todas as funções de controle necessárias no circuito. Os 4 sinais A, B, C e D referenciados às duas fases de clock são mostrados na figura 5.22 abaixo.

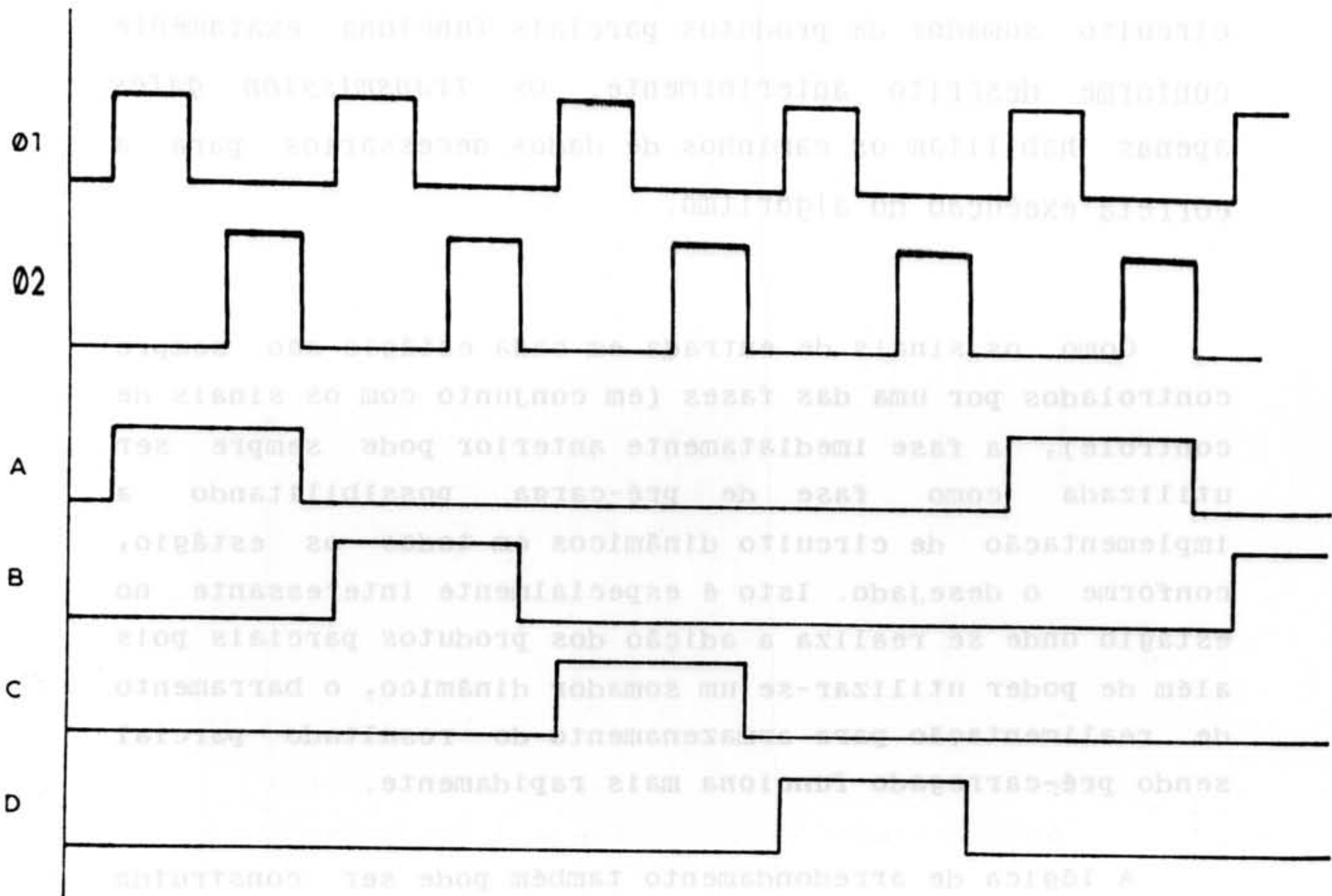


Figura 5.22 O clock e os sinais de controle

Os 4 sinais de controle associados a uma das fases de clock constituem uma máquina de 4 estados como as mostradas anteriormente.

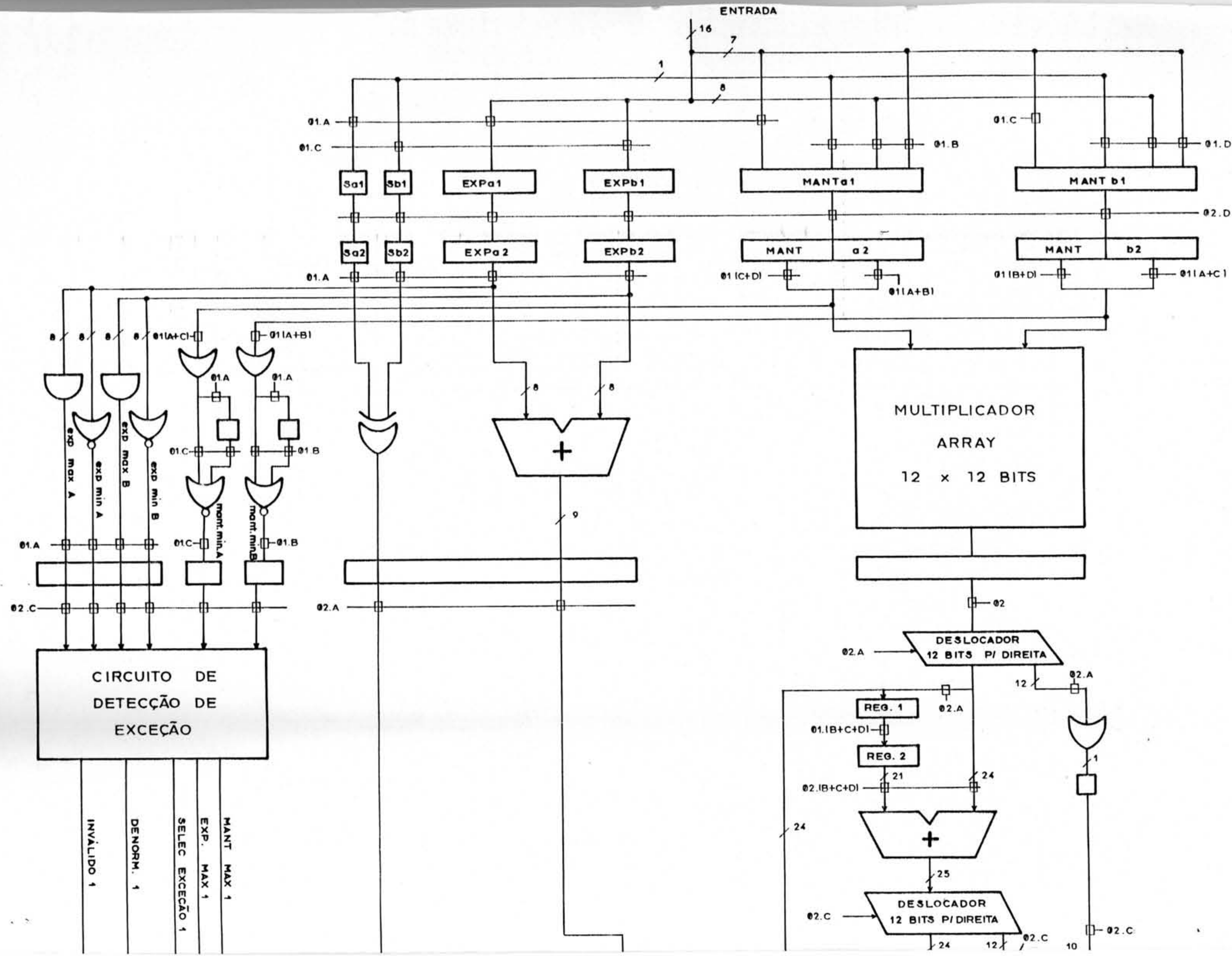
Desta forma, considerando a divisão em estágios pipeline anteriormente estabelecida, a temporização necessária e os sinais de controle, refina-se a descrição do circuito da figura 5.15 incluindo as informações de temporização conforme ilustrado na figura 5.23.

Observa-se que as chaves (*transmission gates*) são ativadas somente por sinais compostos pelas fases e os 4 sinais de controle. Enfim, a estrutura de controle de entrada de operandos e a do cálculo da multiplicação é a mesma utilizada para controlar o avanço do pipeline. O circuito somador de produtos parciais funciona exatamente conforme descrito anteriormente. Os *transmission gates* apenas habilitam os caminhos de dados necessários para a correta execução do algoritmo.

Como os sinais de entrada em cada estágio são sempre controlados por uma das fases (em conjunto com os sinais de controle), a fase imediatamente anterior pode sempre ser utilizada como fase de pré-carga possibilitando a implementação de circuito dinâmicos em todos os estágio, conforme o desejado. Isto é especialmente interessante no estágio onde se realiza a adição dos produtos parciais pois além de poder utilizar-se um somador dinâmico, o barramento de realimentação para armazenamento do resultado parcial sendo pré-carregado funciona mais rapidamente.

A lógica de arredondamento também pode ser construída sob forma de circuito dinâmico diminuindo consideravelmente a área ocupada.

Na figura 5.23 fica claro o descompasso das operações relativas aos expoentes, sinal e sinais de exceção e as sub-operações nas mantissas, nos primeiros estágios. Através dos sinais de controle que ativam os *transmission gates* dos latches verifica-se que os sinais da lógica de controle de exceção, o sinal do produto (+/-) e a soma dos expoentes esperam até o fim das adições parciais relativas a uma multiplicação (FI2.D).


ENTRADA
Sa1
Sb1
EXPa1
EXPb1
MANTa1
MANT b1
Sa2
Sb2
EXPa2
EXPb2
MANT a2
MANT b2
MULTIPLICADOR
ARRAY
12 x 12 BITS
exp max A
exp min A
exp max B
exp min B
mant.min.A
mant.min.B
CIRCUITO DE
DETECÇÃO DE
EXCEÇÃO
INVÁLIDO 1
DENORM. 1
SELEC EXCEÇÃO 1
EXP. MAX 1
MANT MAX 1
DESLOCADOR
12 BITS P/ DIREITA
REG. 1
REG. 2
DESLOCADOR
12 BITS P/ DIREITA

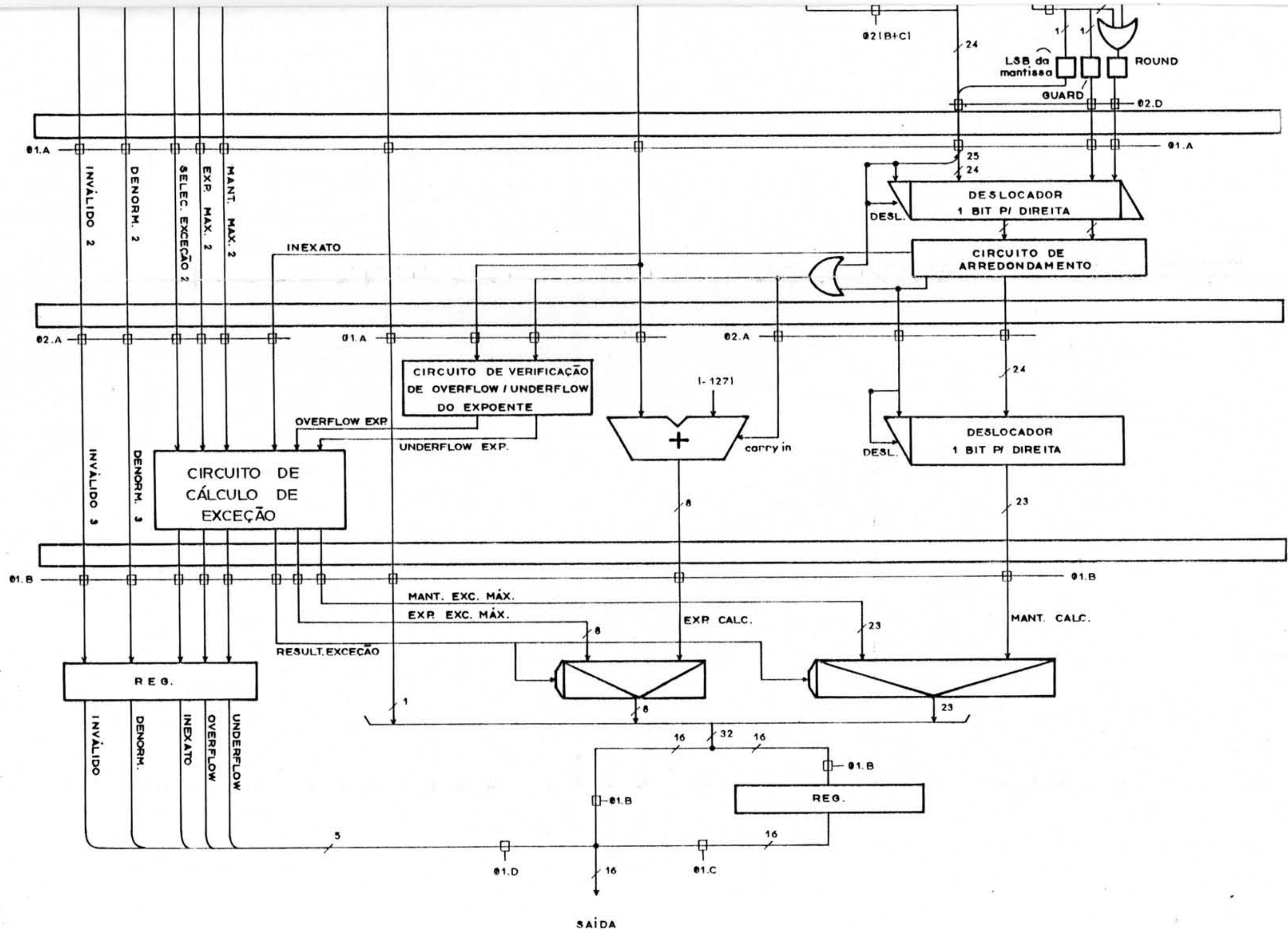


Figura 5.23 Estrutura geral do circuito

A partir deste estágio todos os sinais do caminho de dados avançam sincronamente, acionados pelo mesmo sinal de controle.

A seguir apresenta-se um resumo das sub-operações realizadas em cada estágio. A letra identificadora do estado refere-se a um dos 4 sinais de controle (A, B, C e D).

As sub-operações são referidas a operandos que já se encontram no circuito e a outros que estão entrando no circuito. Os operandos são referenciados com as letras maiúsculas *A*, *B*, *C*, *D*, *E* e *F* em itálico. Supõe-se que os 4 primeiros operandos já se encontram no circuito e entraram nesta ordem.

***Estado* A**

estágio 1: entrada da metade mais significativa do operando *E*

estágio 2: análise dos expoentes (operandos *C* e *D*)
análise das mantissas
cálculo do sinal
adição dos expoentes
primeira multiplicação parcial (P_{11})

estágio 3: deslocamento e armazenamento no acumulador de P_{11} (operandos *C* e *D*)
(acumulador <- (P_{11}>>12))

estágio 4: primeira renormalização e arredondamento (operandos *A* e *B*)

estágio 5: ajuste do expoente (operandos *A* e *B*)
verificação de overflow/underflow expoente
cálculo de exceção
segunda renormalização

estágio 6: inativo

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

Estado B

estágio 1: entrada da metade menos significativa do operando *E*

estágio 2: análise das mantissas (operandos *C* e *D*) segunda multiplicação parcial (P_{lh})

estágio 3: Primeira adição parcial (acumulador <- acumulador + P_{lh})

estágio 4: inativo

estágio 5: inativo

estágio 6: Saída da metade mais significativa do resultado (operandos *A* e *B*).

Estado C

estágio 1: entrada da metade mais significativa do operando *F*

estágio 2: análise das mantissas (operandos *C* e *D*) terceira multiplicação parcial (P_{hl})

estágio 3: Segunda adição parcial (acumulador <- acumulador + P_{hl}) Cálculo do controle de exceção

estágio 4: inativo

estágio 5: inativo

estágio 6: Saída da metade menos significativa do resultado (operandos *A* e *B*).

Estado D

estágio 1: entrada da metade menos significativa do operando *F*

estágio 2: Quarta multiplicação parcial (P_{hh})

estágio 3: Terceira adição parcial (acumulador <- (acumulador + P_{hh})>>12)

estágio 4: inativo

estágio 5: inativo

estágio 6: Saída dos sinais de exceção relativos aos operandos *A* e *B*.

Observa-se que, com o pipe completo, tem-se três multiplicações ponto flutuante em andamento ao mesmo tempo, comprovando o elevado grau de paralelismo temporal conseguido nesta arquitetura pipeline.

5.3.7 Validação

Revela-se extremamente necessário uma maneira de validar esta implementação antes de prosseguir para níveis maiores de refinamento. A propagação de erros deste nível de implementação para níveis de detalhamento maior implica em uma quantidade muito maior de trabalho do que a necessária para realizar esta verificação.

A forma mais simples de validar um sistema como o circuito em questão é através de simulação. Este circuito foi validado através de um simulador funcional de código compilado idealizado por Suzim [SUZ 89].

O simulador utiliza linguagem C de programação para a descrição do hardware e admite que níveis diversos de detalhamento (chaves, operadores aritméticos, blocos funcionais, etc.) compartilhem uma mesma descrição.

Esta técnica mostra-se especialmente interessante no contexto deste projeto pela possibilidade de validar detalhadamente a temporização, um aspecto crucial em qualquer sistema digital, de uma forma especial em arquiteturas pipeline.

O simulador constitui-se de: duas estruturas de dados que armazenam os sinais e os elementos do circuito, o algoritmo de simulação e a descrição do circuito propriamente dito.

As estruturas de dados são iguais e armazenam os sinais e valores associados aos elementos do circuito referentes ao momento atual e ao próximo.

O algoritmo percorre a descrição do circuito avaliando os novos valores a partir dos atuais. O algoritmo resumido é mostrado abaixo.

```
REPITA
{
 Avalia Funções;
 SE estrutura Atual = Estrutura Nova
  ENTAO Troca de fase;
  SENAO Valores atuais = Valores novos;
}
```

As descrições são feitas através de funções ou operações disponíveis diretamente na linguagem, sendo escritos de modo que o valor novo de um sinal ou elemento é uma função dos valores atuais de sinais e elementos:

```
sn->sinal i = f(sa->sinal j, sa->sinal k, ...)
```

onde sn refere-se a estrutura de valores novos e sa refere-se a estrutura de valores atuais.

Para maiores detalhes sobre esta técnica de simulação, os interessados devem endereçar-se a [SUZ 89].

No anexo 1 encontra-se uma listagem de todo o simulador implementado, com a estrutura de dados, o algoritmo de simulação e a descrição do circuito. O nível de detalhamento na descrição refere-se a figura 5.23.

Como ilustração do funcionamento do simulador (e conseqüentemente do multiplicador) apresenta-se em seguida algumas das telas geradas pelo programa, através das quais validou-se o circuito.

Desta forma pôde-se monitorar, a cada fase do relógio, valores de variáveis em todos os estágios, sinais de controle e acompanhar a evolução das operações ao longo de todo o pipeline, validando o funcionamento do hardware proposto.

À cada ciclo de relógio um semioperando (16 bits) deve ser carregado para o circuito. Pode-se notar, conforme afirmado anteriormente, que, com o pipeline cheio, tem-se três multiplicações em andamento simultaneamente.

A entrada de operandos e a saída de resultados é feita segundo o formato de representação estabelecido pelo padrão IEEE. Neste simulador utilizou-se, para a apresentação das grandezas do circuito, a representação no sistema hexadecimal, por questões de compactação de dígitos e por ser diretamente conversível para binário. Os operandos são compostos segundo o padrão IEEE em dígitos binários e a palavra de bits resultante é convertida para o correspondente em dígitos hexadecimais. Para monitoração das variáveis internas também emprega-se a representação hexadecimal.

Inicialmente mostra-se a evolução de uma multiplicação ao longo dos estágios pipeline, desde a entrada dos operandos até a saída dos resultados, apresentando valores associados às sub-operações realizadas em cada estágio.

Em seguida, mostra-se algumas telas relativas a uma seqüência de dados de entrada, possibilitando uma visão global do funcionamento do circuito.

Os números escolhidos para a demonstração são:

A = 987290

B = -967261

Estes valores foram escolhidos pois permitem a observação de duas situações interessantes:

a) Ocorrência de overflow na multiplicação das mantissas, sendo, portanto, necessária uma renormalização.

b) A necessidade de arredondar-se o resultado, o que implicará no incremento do LSB da mantissa do resultado nos limites do formato.

O resultado exato do produto de A por B é:

$$P_{exato} = -954.967.112.690$$

Abaixo apresenta-se a representação binária dos dois operandos e do produto.

A = 1.111 0001 0000 1001 1010 0000 0000 x 2^{19}

B = - 1.110 1100 0010 0101 1101 0000 0000 x 2^{19}

P = - 1.101 1110 0101 1000 0111 1001 ↑ 1011 .. x 2^{39}

limite da mantissa no formato

O valor exato do produto excede o número de bits disponíveis na representação (24 bits), sendo necessário arredondá-lo.

$P_{arred.}$ = - 1.101 1110 0101 1000 0111 1010 x 2^{39}

Obs.: Os dígitos estão separados em conjuntos de 4 para facilitar a visualização na conversão para hexadecimal.

Para realizar a multiplicação e verificar o resultado deve-se compor os operandos e o produto conforme o formato estipulado pelo padrão, ilustrado na figura 5.24.

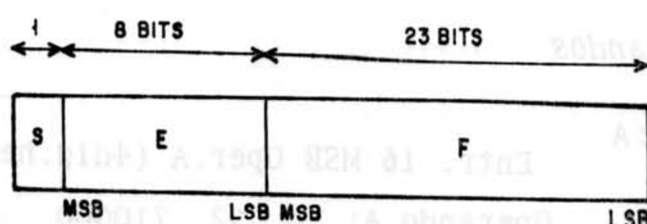


Figura 5.24 Formato Básico Simples

O valor associado (se o operando é um número normalizado, como neste caso) é computado através da fórmula:

$$v = (-1)^{s} \cdot 1.\text{mant} \times 2^{(exp-127)}$$

O primeiro passo é polarizar os expoentes (somar 127).

exp A polarizado = 19 + 127 = 146 = 92H
exp B polarizado = 19 + 127 = 146 = 92H
exp P polarizado = 39 + 127 = 166 = A6H

A seguir, todos os campos são compostos formando uma palavra só de 32 bits. Esta palavra é então representada por 8 dígitos hexadecimais, conforme mostrado abaixo.

	sinal	*expoente*	*mantissa*
A =	0	100 1001 0	111 0001 0000 1001 1010 0000
B =	1	100 1001 0	110 1100 0010 0101 1101 0000
P =	1	101 0011 0	101 1110 0101 1000 0111 1010

A = 4971 09A0 (hex.)
B = C96C 25D0 (hex.)
P = D35E 587A (hex.)

Abaixo apresenta-se a evolução desta operação pelos estágios pipeline.

SIMULADOR MULTIPLICADOR PONTO FLUTUANTE PIPELINE

Entrada dos Operandos

```
-----------------------------------------------------------
ciclo:1  sequenc:A
fase1=1  fase2=0        Entr. 16 MSB Oper.A (4dig.hex):4971

ESTAGIO 1               Operando A:  0  92  710000
                        Operando B:  0  00  000000
-----------------------------------------------------------

-----------------------------------------------------------
ciclo:2  sequenc:B
fase1=1  fase2=0        Entr. 16 LSB Oper.A (4dig.hex):9a0

ESTAGIO 1               Operando A:  0  92  7109A0
                        Operando B:  0  00  000000
-----------------------------------------------------------

-----------------------------------------------------------
ciclo:3  sequenc:C
fase1=1  fase2=0        Entr. 16 MSB Oper.B (4dig.hex):c96c

ESTAGIO 1               Operando A:  0  92  7109A0
                        Operando B:  1  92  6C0000
-----------------------------------------------------------

-----------------------------------------------------------
ciclo:4  sequenc:D
fase1=1  fase2=0        Entr. 16 LSB Oper.B (4dig.hex):25d0

ESTAGIO 1               Operando A:  0  92  7109A0
                        Operando B:  1  92  6C25D0
-----------------------------------------------------------
```

Computação dos Produtos Parciais e Verificação dos Operandos

```
-----------------------------------------------------------
ciclo:5  sequenc:A
fase1=1  fase2=0

ESTAGIO 2   Sinal:1   Exp.Pol.:124   ProdutoParcial:37F200
ExpMaxA:0      ExpMinA:0     MantMinA:1    A:0 92 7109A0
ExpMaxB:0      ExpMinB:0     MantMinB:1    B:1 92 6C25D0
-----------------------------------------------------------

-----------------------------------------------------------
ciclo:6  sequenc:B
fase1=1  fase2=0

ESTAGIO 2   Sinal:1   Exp.Pol.:124   ProdutoParcial:8E0B40
ExpMaxA:0      ExpMinA:0     MantMinA:1    A:0 92 7109A0
ExpMaxB:0      ExpMinB:0     MantMinB:0    B:1 92 6C25D0
-----------------------------------------------------------
```

```
------------------------------------------------------------------
ciclo:7  sequenc:C
fase1=1  fase2=0

ESTAGIO 2   Sinal:1   Exp.Pol.:124   ProdutoParcial:578D00
ExpMaxA:0       ExpMinA:0       MantMinA:0     A:0 92 7109A0
ExpMaxB:0       ExpMinB:0       MantMinB:0     B:1 92 6C25D0
------------------------------------------------------------------

------------------------------------------------------------------
ciclo:8  sequenc:D
fase1=1  fase2=0

ESTAGIO 2   Sinal:1   Exp.Pol.:124   ProdutoParcial:DE4A20
ExpMaxA:0       ExpMinA:0       MantMinA:0     A:0 92 7109A0
ExpMaxB:0       ExpMinB:0       MantMinB:0     B:1 92 6C25D0
------------------------------------------------------------------
```

Computação do Produto Final e dos Bits de Arredondamento

```
------------------------------------------------------------------
ciclo:8  sequenc:D
fase1=0  fase2=1

ESTAGIO 3   Invalido:0  Denorm:0  SelExc:0
MantMax:0  ExpMax:1     Guard:0  Round:1        A:0 92 7109A0
Sinal:1   ExpPol:124  Produto:1BCB0F3           B:1 92 6C25D0
------------------------------------------------------------------
```

Renormalização e Arredondamento

```
------------------------------------------------------------------
ciclo:9  sequenc:A
fase1=1  fase2=0

ESTAGIO 4   Invalido:0  Denorm:0  SelExc:0  Inexato:1
MantMax:0  ExpMax:1  CarryProd:1 CarryArred:0 A:0 92 7109A0
Sinal:1   ExpPol:124  ProdArred:5E587A          B:1 92 6C25D0
------------------------------------------------------------------
```

Verificação de Overflow/Underflow do expoente

e Seleção de Resultado Calculado/Exceção

```
------------------------------------------------------------------
ciclo:9  sequenc:A
fase1=0  fase2=1

ESTAGIO 5     Invalido:0   Denorm:0    Inexato:1
    Overf:0   Underf:0     SelExc:0                A:0 92 7109A0
    Sinal:1   ExpCalc:A6   MantCalc:5E587A         B:1 92 6C25D0
------------------------------------------------------------------
```

Saída do Resultado (Bits Mais Significativos)

```
------------------------------------------------------------------
ciclo:10 sequenc:B
fase1=1  fase2=0

ESTAGIO 6               SAIDA:D35E                 A:0 92 7109A0
Inval:0 Denorm:0 Inexato:1 Overf:0 Underf:0        B:1 92 6C25D0
------------------------------------------------------------------
```

Saída do Resultado (Bits Menos Significativos)

```
------------------------------------------------------------------
ciclo:11 sequenc:C
fase1=1  fase2=0

ESTAGIO 6                  SAIDA:587A                A:0 92 7109A0
Inval:0 Denorm:0 Inexato:1 Overf:0 Underf:0          B:1 92 6C25D0
------------------------------------------------------------------
```

Comentários:

a) Nos primeiros 4 ciclos ocorre a entrada dos operandos.

b) Nos 4 ciclos seguintes é realizado o cálculo dos 4 produtos parciais e a adição destes no estágio 3.

c) No ciclo 8 (fase 2) é computado o produto final e os bits de arredondamento.

d) No ciclo 9 ocorre a Renormalização e o Arredondamento

e) No ciclo 10 a saída da primeira metade do resultado e os flags de exceção.

f) No ciclo 11 a saída da segunda metade do resultado.

A seguir apresenta-se algumas telas relativas a uma série de multiplicações sequenciais que permitem uma visão global do funcionamento pipeline do circuito.

Os pares de operandos que entram no circuito estão listados abaixo, bem como o resultado esperado e sua codificação no formato IEEE (hexadecimal).

a) (-1) x (-2) = (-2) <=> BF80 0000 x C000 0000 = 4000 0000

b) (-Inf.) x (-1) = (+Inf.)

<=> FF80 0000 x 3F80 0000 = FF80 0000

c) (+0) x (-Inf.) = (-NaN)

<=> 0000 0000 x FF80 0000 = FFFF FFFF

d) (+12) x (+25) = (+300)

<=> 4140 0000 x 41C8 0000 = 4396 0000

e) (+2) x (Inf.) = (+Inf.)

<=> 4000 0000 x 7F80 0000 = 7F80 0000

f) (-1) x (+NaN) = (-NaN)

<=> BF80 0000 x 7FF0 F0F0 = FFFF FFFF

As primeiras quatro telas (ciclos 13 a 16) referem-se a entrada do par de operandos *d*. Nas seqüências B e C, apresenta-se, à saída, o resultado da multiplicação relativa ao par de operandos *b*. A multiplicação relativa ao par de operandos *c* está em andamento (computação dos produtos parciais).

Nas duas telas seguintes (ciclos 18 e 19) apresenta-se os resultados referentes aos operandos *c*. Está, simultaneamente, ocorrendo a entrada dos operandos *e* e a computação da multiplicação relativa aos operandos *d*.

Finalmente, nas duas últimas telas (ciclos 22 e 23) é realizada a saída do resultado referente aos operandos *d* (cuja entrada de dados é mostrada nas quatro primeiras telas). Está em andamento a multiplicação *e* e estão entrando os operandos referentes à multiplicação *f*.

```
------------------------------------------------------------------

       SIMULADOR MULTIPLICADOR PONTO FLUT. PIPELINE
ciclo:13 sequenc:A     Digite [s] para Sair
fase1=1  fase2=0       Entr. 16 MSB Oper.A (4dig.hex):4140

ESTAGIO 1              Operando A:  0 82  400000
                       Operando B:  1 FF  000000

ESTAGIO 2    Sinal:1   Exp.Pol.:0FF   ProdutoParcial:000000
ExpMaxA:0       ExpMinA:1      MantMinA:1    A:0 00 000000
ExpMaxB:1       ExpMinB:0      MantMinB:1    B:1 FF 000000

ESTAGIO 3    Invalido:0  Denorm:0  SelExc:1
MantMax:0  ExpMax:1      Guard:0  Round:0    A:1 FF 000000
Sinal:1  ExpPol:17E  Produto:800000          B:0 7F 000000

ESTAGIO 4    Invalido:0  Denorm:0  SelExc:1  Inexato:0
MantMax:0  ExpMax:1  CarryProd:0 CarryArred:0 A:1 FF 000000
Sinal:1  ExpPol:17E  ProdArred:000000        B:0 7F 000000

ESTAGIO 5     Invalido:0    Denorm:0    Inexato:0
    Overf:0  Underf:0      SelExc:0          A:1 7F 000000
    Sinal:0  ExpCalc:80    MantCalc:000000   B:1 80 000000

ESTAGIO 6                 SAIDA:0000         A:1 7F 000000
Inval:0 Denorm:0 Inexato:0 Overf:0 Underf:0  B:1 80 000000

------------------------------------------------------------------
```

```
------------------------------------------------------------------

       SIMULADOR MULTIPLICADOR PONTO FLUT. PIPELINE
ciclo:14 sequenc:B     Digite [s] para Sair
fase1=1  fase2=0       Entr. 16 LSB Oper.A (4dig.hex):0

ESTAGIO 1              Operando A:  0  82  400000
                       Operando B:  1  FF  000000

ESTAGIO 2    Sinal:1   Exp.Pol.:0FF   ProdutoParcial:000000
ExpMaxA:0       ExpMinA:1      MantMinA:1    A:0 00 000000
ExpMaxB:1       ExpMinB:0      MantMinB:1    B:1 FF 000000

ESTAGIO 3    Invalido:0  Denorm:0  SelExc:1
MantMax:0  ExpMax:1      Guard:0  Round:0    A:0 00 000000
Sinal:1  ExpPol:0FF  Produto:800000          B:1 FF 000000

ESTAGIO 4    Invalido:0  Denorm:0  SelExc:1  Inexato:0
MantMax:0  ExpMax:1  CarryProd:0 CarryArred:0 A:1 FF 000000
Sinal:1  ExpPol:17E  ProdArred:000000        B:0 7F 000000

ESTAGIO 5     Invalido:0    Denorm:0    Inexato:0
    Overf:0  Underf:0      SelExc:1          A:1 FF 000000
    Sinal:1  ExpCalc:FF    MantCalc:000000   B:0 7F 000000

ESTAGIO 6                 SAIDA:FF80         A:1 FF 000000
Inval:0 Denorm:0 Inexato:0 Overf:0 Underf:0  B:0 7F 000000

------------------------------------------------------------------
```

```
-----------------------------------------------------------------------
        SIMULADOR MULTIPLICADOR PONTO FLUT. PIPELINE
ciclo:15 sequenc:C      Digite [s] para Sair
fase1=1  fase2=0        Entr. 16 MSB Oper.B (4dig.hex):41c8

ESTAGIO 1               Operando A:  0  82  400000
                        Operando B:  0  83  480000

ESTAGIO 2   Sinal:1    Exp.Pol.:OFF    ProdutoParcial:000000
ExpMaxA:0       ExpMinA:1       MantMinA:1      A:0 00 000000
ExpMaxB:1       ExpMinB:0       MantMinB:1      B:1 FF 000000

ESTAGIO 3   Invalido:0  Denorm:0  SelExc:1
MantMax:0  ExpMax:1     Guard:0  Round:0        A:0 00 000000
Sinal:1  ExpPol:OFF   Produto:800000            B:1 FF 000000

ESTAGIO 4   Invalido:0  Denorm:0  SelExc:1  Inexato:0
MantMax:0  ExpMax:1  CarryProd:0 CarryArred:0 A:1 FF 000000
Sinal:1  ExpPol:17E  ProdArred:000000          B:0 7F 000000

ESTAGIO 5    Invalido:0     Denorm:0      Inexato:0
    Overf:0  Underf:0       SelExc:1               A:1 FF 000000
    Sinal:1  ExpCalc:FF     MantCalc:000000       B:0 7F 000000

ESTAGIO 6              SAIDA:0000                  A:1 FF 000000
Inval:0 Denorm:0 Inexato:0 Overf:0 Underf:0      B:0 7F 000000
-----------------------------------------------------------------------
```

```
-----------------------------------------------------------------------
        SIMULADOR MULTIPLICADOR PONTO FLUT. PIPELINE
ciclo:16 sequenc:D      Digite [s] para Sair
fase1=1  fase2=0        Entr. 16 LSB Oper.B (4dig.hex):0

ESTAGIO 1               Operando A:  0  82  400000
                        Operando B:  0  83  480000

ESTAGIO 2   Sinal:1    Exp.Pol.:OFF    ProdutoParcial:400000
ExpMaxA:0       ExpMinA:1       MantMinA:1      A:0 00 000000
ExpMaxB:1       ExpMinB:0       MantMinB:1      B:1 FF 000000

ESTAGIO 3   Invalido:1  Denorm:0  SelExc:1
MantMax:1  ExpMax:1     Guard:0  Round:0        A:0 00 000000
Sinal:1  ExpPol:OFF   Produto:800000            B:1 FF 000000

ESTAGIO 4   Invalido:0  Denorm:0  SelExc:1  Inexato:0
MantMax:0  ExpMax:1  CarryProd:0 CarryArred:0 A:1 FF 000000
Sinal:1  ExpPol:17E  ProdArred:000000          B:0 7F 000000

ESTAGIO 5    Invalido:0     Denorm:0      Inexato:0
    Overf:0  Underf:0       SelExc:1               A:1 FF 000000
    Sinal:1  ExpCalc:FF     MantCalc:000000       B:0 7F 000000

ESTAGIO 6              SAIDA:0000                  A:1 FF 000000
Inval:0 Denorm:0 Inexato:0 Overf:0 Underf:0      B:0 7F 000000
-----------------------------------------------------------------------
```

```
----------------------------------------------------------------------

        SIMULADOR MULTIPLICADOR PONTO FLUT. PIPELINE
ciclo:18 sequenc:B       Digite [s] para Sair
fase1=1  fase2=0         Entr. 16 LSB Oper.A (4dig.hex):0

ESTAGIO 1               Operando A:  0  80  000000
                        Operando B:  0  83  480000

ESTAGIO 2  Sinal:0   Exp.Pol.:105   ProdutoParcial:000000
ExpMaxA:0    ExpMinA:0     MantMinA:1       A:0 82 400000
ExpMaxB:0    ExpMinB:0     MantMinB:0       B:0 83 480000

ESTAGIO 3   Invalido:1  Denorm:0  SelExc:1
MantMax:1 ExpMax:1      Guard:0  Round:0    A:0 82 400000
Sinal:0  ExpPol:105   Produto:800000        B:0 83 480000

ESTAGIO 4   Invalido:1  Denorm:0  SelExc:1  Inexato:0
MantMax:1  ExpMax:1  CarryProd:0 CarryArred:0 A:0 00 000000
Sinal:1  ExpPol:0FF  ProdArred:000000         B:1 FF 000000

ESTAGIO 5    Invalido:1    Denorm:0     Inexato:0
    Overf:0  Underf:0      SelExc:1             A:0 00 000000
    Sinal:1  ExpCalc:80    MantCalc:000000      B:1 FF 000000

ESTAGIO 6              SAIDA:FFFF               A:0 00 000000
Inval:1 Denorm:0 Inexato:0 Overf:0 Underf:0     B:1 FF 000000
----------------------------------------------------------------------



----------------------------------------------------------------------
        SIMULADOR MULTIPLICADOR PONTO FLUT. PIPELINE
ciclo:19 sequenc:C       Digite [s] para Sair
fase1=1  fase2=0         Entr. 16 MSB Oper.B (4dig.hex):7f80

ESTAGIO 1               Operando A:  0  80  000000
                        Operando B:  0  FF  000000

ESTAGIO 2  Sinal:0   Exp.Pol.:105   ProdutoParcial:000000
ExpMaxA:0        ExpMinA:0     MantMinA:0       A:0 82 400000
ExpMaxB:0        ExpMinB:0     MantMinB:0       B:0 83 480000

ESTAGIO 3   Invalido:1  Denorm:0  SelExc:1
MantMax:1  ExpMax:1     Guard:0  Round:0        A:0 82 400000
Sinal:0  ExpPol:105  Produto:800000             B:0 83 480000

ESTAGIO 4   Invalido:1  Denorm:0  SelExc:1  Inexato:0
MantMax:1  ExpMax:1  CarryProd:0 CarryArred:0 A:0 00 000000
Sinal:1  ExpPol:0FF  ProdArred:000000         B:1 FF 000000

ESTAGIO 5    Invalido:1    Denorm:0     Inexato:0
    Overf:0  Underf:0      SelExc:1             A:0 00 000000
    Sinal:1  ExpCalc:80    MantCalc:000000      B:1 FF 000000

ESTAGIO 6              SAIDA:FFFF               A:0 00 000000
Inval:1 Denorm:0 Inexato:0 Overf:0 Underf:0     B:1 FF 000000
----------------------------------------------------------------------
```

```
-----------------------------------------------------------------
      SIMULADOR MULTIPLICADOR PONTO FLUT. PIPELINE
ciclo:22 sequenc:B    Digite [s] para Sair
fase1=1  fase2=0      Entr. 16 LSB Oper.A (4dig.hex):0

ESTAGIO 1              Operando A:  1  7F  000000
                       Operando B:  0  FF  000000

ESTAGIO 2  Sinal:0  Exp.Pol.:17F  ProdutoParcial:000000
ExpMaxA:0        ExpMinA:0        MantMinA:0      A:0 80 000000
ExpMaxB:1        ExpMinB:0        MantMinB:1      B:0 FF 000000

ESTAGIO 3  Invalido:0  Denorm:0  SelExc:0
MantMax:0  ExpMax:1     Guard:0  Round:0          A:0 80 000000
Sinal:0  ExpPol:17F  Produto:12C0000              B:0 FF 000000

ESTAGIO 4  Invalido:0  Denorm:0  SelExc:0  Inexato:0
MantMax:0  ExpMax:1  CarryProd:1 CarryArred:0 A:0 82 400000
Sinal:0  ExpPol:105  ProdArred:160000             B:0 83 480000

ESTAGIO 5    Invalido:0    Denorm:0    Inexato:0
     Overf:0  Underf:0     SelExc:0               A:0 82 400000
     Sinal:0  ExpCalc:87   MantCalc:160000        B:0 83 480000

ESTAGIO 6                SAIDA:4396               A:0 82 400000
Inval:0 Denorm:0 Inexato:0 Overf:0 Underf:0      B:0 83 480000
-----------------------------------------------------------------


-----------------------------------------------------------------
      SIMULADOR MULTIPLICADOR PONTO FLUT. PIPELINE
ciclo:23 sequenc:C    Digite [s] para Sair
fase1=1  fase2=0      Entr. 16 MSB Oper.B (4dig.hex):7ff0

ESTAGIO 1              Operando A:  1  7F  000000
                       Operando B:  0  FF  700000

ESTAGIO 2  Sinal:0  Exp.Pol.:17F  ProdutoParcial:000000
ExpMaxA:0        ExpMinA:0        MantMinA:1      A:0 80 000000
ExpMaxB:1        ExpMinB:0        MantMinB:1      B:0 FF 000000

ESTAGIO 3  Invalido:0  Denorm:0  SelExc:0
MantMax:0  ExpMax:1     Guard:0  Round:0          A:0 80 000000
Sinal:0  ExpPol:17F  Produto:12C0000              B:0 FF 000000

ESTAGIO 4  Invalido:0  Denorm:0  SelExc:0  Inexato:0
MantMax:0  ExpMax:1  CarryProd:1 CarryArred:0 A:0 82 400000
Sinal:0  ExpPol:105  ProdArred:160000             B:0 83 480000

ESTAGIO 5    Invalido:0    Denorm:0    Inexato:0
     Overf:0  Underf:0     SelExc:0               A:0 82 400000
     Sinal:0  ExpCalc:87   MantCalc:160000        B:0 83 480000

ESTAGIO 6                SAIDA:0000               A:0 82 400000
Inval:0 Denorm:0 Inexato:0 Overf:0 Underf:0      B:0 83 480000
-----------------------------------------------------------------
```

5.4 Conclusões

Com base na arquitetura do multiplicador ponto flutuante no capítulo anterior detalhou-se o projeto.

Inicialmente foram consideradas questões de factibilidade e de viabilidade econômica. Estes aspectos tiveram influência decisiva na estrutura de entrada e saída do circuito e em seu desempenho.

Uma vez estabelecida a estratégia de entrada e saída detalhou-se os blocos constituintes e dividiu-se a arquitetura em estágios pipeline.

Seguiu-se o desenvolvimento da estrutura de controle e a temporização do circuito a nível de fases.

Através de uma simulação funcional pôde-se validar os procedimentos de computação do circuito, a temporização e a estrutura pipeline.

6 CONCLUSAO

Em muitas aplicações computacionais o tempo associado às operações aritméticas é responsável por uma fração considerável do tempo total de processamento. Isto deve-se ao elevado volume de operações requeridas e à complexa operacionalidade apresentada pela representação numérica.

A viabilidade de muitas destas aplicações, como por exemplo Computação Gráfica, Simulação e Processamento Digital de Sinais, é condicionada à disponibilidade de processamento numérico de alto desempenho, que pode ser traduzido sob dois aspectos:

a) uma representação numérica eficiente, ou seja, que forneça um intervalo dinâmico adequado e precisão suficiente. Tais características são apresentadas pelas representações em ponto flutuante.

b) velocidade de computação.

Os algoritmos das operações envolvendo operandos de ponto flutuante são consideravelmente mais complexos que os seus correspondentes em ponto fixo. Apresentam execução tipicamente lenta em hardware convencional, o que sugere, então, sua implementação como circuitos dedicados.

A evolução da tecnologia de implementação de circuitos VLSI tem permitido a integração de vários operadores numa mesma pastilha e desempenhos cada vez maiores, podendo mesmo ser apontada como um dos fatores principais para o acelerado desenvolvimento dos circuitos para processamento aritmético. Como exemplo, observa-se, entre os circuitos comerciais, os Coprocessadores Aritméticos, verdadeiras bibliotecas de operações aritméticas e funções matemáticas para vários formatos de representação. Já os Processadores Aritméticos Dedicados executam poucas operações mas com

altíssimo desempenho. Os microprocessadores de última geração com unidades dedicadas às operações aritméticas básicas em ponto flutuante e circuitos com funções mais específicas como FFT já são realidade.

O impulso no desenvolvimento de Processadores Aritméticos Integrados deve-se, também, em parte, à padronização pelo IEEE, dos formatos de representação para ponto flutuante e de sua aritmética básica. Isto viabilizou a portabilidade tanto do software como do hardware.

Os algoritmos das operações de Adição, Subtração e Multiplicação foram estudados para o formato básico simples (32 bits). Estas operações, apesar de básicas e simples, constituem o núcleo dos sistemas de processamento digital de sinais.

O mapeamento do algoritmo para a arquitetura tem como critérios orientadores gerais - e normalmente antagônicos - custo e desempenho. Nesta dissertação propôs-se, inicialmente, arquiteturas de desempenho máximo pois as operações são normalmente lentas. Visou-se, também, uma possível solução pipeline, por tratar-se de uma forma relativamente barata de conseguir paralelismo temporal e uma alternativa interessante em aplicações DSP.

Observou-se que a conformidade com o padrão IEEE exige um grande investimento na gerência de situações de exceção, refletindo-se, no algoritmo, sob a forma dos procedimentos de Verificação dos Operandos e Tratamento de Exceções. Na arquitetura, por sua vez, como blocos funcionais dedicados.

No detalhamento da arquitetura do multiplicador ponto flutuante pipeline, visando a implementação integrada, o compromisso custo x desempenho foi considerado. Aspectos

relacionados à pinagem e área ocupada foram abordados. O resultado da análise conduziu a uma redução no número de pinos dedicados à entrada e saída. Esta redução, por sua vez, limitou o desempenho do circuito tanto quanto ao tempo de latência como à taxa de saída dos resultados (throughput), levando a uma reestruturação geral da arquitetura.

A técnica de multiplicação modular foi adaptada para realizar o multiplicador das mantissas. A redução da área de circuito foi conseguida aproveitando as restrições na entrada de dados para compartilhar recursos no tempo. Esta solução, entretanto, aumentou o tempo de latência de uma multiplicação, mas não alterou a taxa de saída de resultados.

Para distribuição da estrutura pelos estágios pipeline, efetuou-se uma avaliação preliminar do tempo de latência inerente a cada bloco funcional, procurando reparti-los equivalentemente, otimizando o ciclo de pipeline.

A validação, etapa imprescindível no desenvolvimento de circuitos integrados complexos, foi realizada neste nível de descrição (fases) através da estratégia HDC. Este simulador de código compilado tem o mérito de permitir, à escolha do projetista, que diferentes níveis de detalhamento, referentes a diferentes aspectos do circuito, coexistam na mesma descrição. Isto viabilizou, por exemplo, a descrição, simulação e validação da temporização do circuito a nível de fases de relógio, sem entrar nos detalhes dos blocos operativos como o multiplicador, somadores, deslocadores, etc.

O circuito proposto destina-se, evidentemente, a

aplicações que exijam processamento de alto desempenho em ponto flutuante e que façam uso da estrutura pipeline.

Uma aplicação típica seria como um dos componentes de uma placa aceleradora para operandos em ponto flutuante em sistemas de processamento digital de sinais. O grande número de computações aritméticas em sistemas DSP possibilitam uso extensivo do pipeline e, observa-se também, que somente as versões mais recentes dos microprocessadores DSP incluem ULAs para processamento em ponto flutuante, mesmo assim com desempenho restrito.

Desenvolvimento do projeto

A entrada de operandos no circuito proposto é plenamente utilizada no funcionamento pipeline. O ciclo de entrada de um par de operandos compreende 4 ciclos de relógio. A cada ciclo de relógio um semi-operando é carregado para o circuito e os bits referentes a cada campo componente do número binário ponto flutuante (sinal, expoente e mantissa) são separados para posterior multiplicação.

Segundo esta estrutura, no funcionamento pipeline, caso o próximo par de operandos não esteja disponível para entrar no circuito no primeiro ciclo da seqüência de entrada, deve-se esperar que passe toda a seqüência de entrada (4 ciclos de relógio).

Pode-se eliminar esta restrição definindo uma estrutura de controle que avance juntamente com os operandos pelos estágios pipeline e que gere a seqüência de sinais de controle a partir de qualquer ciclo. Isto permitiria, então, a entrada de um par de operandos a

partir de qualquer fase F1, por exemplo. Finalmente, é necessário, ainda, que sejam estabelecidos pinos e sinais específicos que permitam sincronizar a entrada de operandos e monitorar facilmente a saída de resultados.

Não foram desenvolvidos, e portanto devem ser objeto de projeto, o circuito que gera, a partir do sinal externo de relógio, as fases F1 e F2 e o circuito que gera os sinais de controle.

A implementação do circuito também requer um estudo dos blocos operativos. As várias técnicas de implementação de multiplicadores array, somadores rápidos, etc. devem ser investigadas sob a orientação do compromisso área x desempenho. O estudo e definição detalhada dos blocos operativos, e portanto sua avaliação temporal, podem impor uma redistribuição destes blocos pelos estágios pipeline.

É de fundamental importância a elaboração prévia de um procedimento de teste e a incorporação de estruturas que facilitem a testabilidade do circuito.

Uma vez detalhados os blocos operativos escolhe-se as abordagens de projeto mais adequadas para a implementação de cada um. Esta escolha será norteada por uma relação de compromissos entre o tempo de projeto, a área ocupada e a velocidade de operação do circuito.

Tendo sido definidas as abordagens de projeto para a implementação dos módulos, realiza-se a geração das máscaras, posicionamento dos módulos, o roteamento dos caminhos de dados, sinais de controle e relógio. Segue-se, então, a fabricação e o teste do protótipo.

Evolução do Sistema

Uma verificação mais apurada das aplicações do circuito induzem algumas idéias no sentido de facilitar sua operação, melhorar o desempenho e mesmo extender o leque de aplicações.

Um incremento interessante no circuito seria a inclusão de um banco de registradores para o armazenamento de operandos e resultados intermediários. Isto diminuiria o tempo de latência das operações (evitando carga redundante de operandos freqüentemente usados), facilitaria a multiplicação de vetores por constantes e também o uso do resultado de uma multiplicação nas multiplicações subseqüentes (realimentando o resultado internamente e não externamente ao circuito) entre outras.

Evidentemente isto cria necessidades de implementar-se, além do banco de registradores propriamente dito e dos caminhos de dados necessários, instruções próprias para: carga de operandos nos registradores e realizar a operação envolvendo operandos armazenados ou não em várias combinações para obter-se um bom aproveitamento destes recursos.

Outra extensão, porém de proporções bem mais ambiciosas, seria a incorporação da operação de adição e subtração. Isto viabilizaria a execução, além da própria adição, do produto interno. Esta operação é fundamental no cálculo da multiplicação de matrizes e em sistemas de processamento digital de sinais. Entretanto, a inclusão da adição tem implicações muito mais sérias do que aquelas relativas ao banco de registradores, devido a complexidade inerente à operação e suas diferenças com respeito a multiplicação impedindo um razoável compartilhamento de

recursos.

Olhando em direção a sistemas de processamento aritmético mais completos a evolução deve vir pela subseqüente inclusão das operações de divisão e raiz quadrada.

Este trabalho, portanto, representa uma contribuição à cultura local com respeito ao estudo da representação em ponto flutuante, aos algoritmos das operações aritméticas e às arquiteturas de alto desempenho para estes operadores visando uma implementação monolítica.

O desenvolvimento de algumas etapas do projeto do multiplicador ponto flutuante pipeline em tecnologia CMOS, constitui também uma contribuição por tratar-se da continuidade dos estudos anteriores e também pelas questões inerentes ao projeto de circuitos integrados, cujas análises de compromissos e soluções encontradas podem, eventualmente, servir de base em futuros projetos.

ANEXO 1

Listagem do programa simulador do circuito multiplicador ponto flutuante

```
#include "stdio.h"
#include "stdlib.h"
#include "conio.h"

/* Definicao das constantes */

#define true 0XFF
#define false 0
#define MASC16 0x8000
#define MASC12 0XFFF
#define MASC11 0x7FF
#define MASC23 0X7FFFFFL
#define MASC24 0XFFFFFFL
#define MASC8  0X00FF
#define MASC7  0X007F
#define MSUM   0x800

/* Variaveis do circuito */

typedef struct
 {

                    /* Variaveis do Estagio 1 */

  unsigned long     mantA1, mantB1, mantA2, mantB2;
  unsigned          expA1, expB1, expA2, expB2;
  char              sinalA1, sinalB1, sinalA2, sinalB2;

                    /* Variaveis do Estagio 2 */

  unsigned long     entrmultA, entrmultB, produtoparcial,
                    parcelaA, parcelaB;
  unsigned          somexp1;
  char              sinalresult1, OUmantA1, OUmantA2,
                    OUmantB1, OUmantB2, mantminA, mantminB,
                    expmaxA, expminA, expmaxB, expminB;

                    /* Variaveis do Estagio 3 */

  unsigned long     entrshifter, prod, entrUlaA, entrUlaB,
                    soma_des1, produto, soma, reg1, reg2;
  unsigned          prod_des1, bits_des1, somexp2;
  char              sinalresult2, bit_arredond, round, guard,
                    round_bit, guard_bit, invalido1, denorm1,
                    selec_excecao1, mantmax1, expmax1, LSBprod;
```

```
                   /* Variaveis do Estagio 4 */

unsigned long      prod_arredond, prod_renorm1, prod_renorm2;
unsigned           somexp3;
char               invalido2, denorm2, selec_excecao2,
                   mantmax2, expmax2, sinalresult3,
                   carry_out_prod, bit_arred1, bit_arred2,
                   bit_arred3, carry_out_arred, inexato1;

                   /* Variaveis do Estagio 5 */

unsigned long      mantcalc;
unsigned           expcalc, exp_calc_pol;
char               invalido3, denorm3,sinalresult4, carry_in,
                   overf_exp, underf_exp, inexato2, overflow1,
                   underflow1, selec_calc, mant_exc_max,
                   exp_exc_max, result_excecao;

                   /* Variaveis do Estagio 6 */

unsigned long      result;
unsigned           saida;
char               invalido, denorm, inexato, overflow,
                   underflow;

} circuito;

/*---------------------------------------------------------------*/
/*------AREA DE FUNCOES DO SIMULADOR-----------------------------*/

int cont;

int bool(char x)
{
 return(x ? 1: 0);
}

void carga(circuito *sa, char ch, unsigned n)
{
 char *aux;
 for (aux=(char*)sa; n>0; *aux++=ch, n--);
 return;
}

void cargauno (circuito *sa, char ch, unsigned i)
{
 char *aux;
 aux = (char*) sa;
 *(aux+i)=ch;
}

char valor (circuito *sa, unsigned n)
{
 char *aux;
 aux = (char*) sa;
 return (*(aux+n));
}
```

```
/* Compara e permuta dois blocos de memoria */
int changecomp (circuito **sn, circuito **sa, unsigned nbytes)
{
  circuito *sint;
  char *auxsa, *auxsn;
  int i, v;
  auxsa = (char*) *sa;
  auxsn = (char*) *sn;
  for (i=0; i<nbytes && (*auxsn == *auxsa); i++,auxsn++,auxsa++);
  if (i==nbytes)
    {
     v=1;
    }
  else
    {
     v=0;
     if(cont>=90) printf("Nao convergencia na variavel # %d",i);
    }
  sint = *sa;
  *sa = *sn;
  *sn = sint;
  return(v);
}

char *binar (unsigned long int num, int bits)
{
 int i;
 static char VET[33];
 VET[32]='\0';
 for (i=0; i<bits; ++i)
 {
  VET[31-i] = (num % 2)+'0';
  num/=2;
 }
 return (VET+(32-bits));
}


void desenhatela(void)
{
 clrscr();
 gotoxy(7,1);
 printf("SIMULADOR MULTIPLICADOR PONTO FLUT. PIPELINE");
 gotoxy(23,2); printf("Digite [s] para Sair");
 gotoxy(1,2); printf("ciclo:");
 gotoxy(10,2); printf("sequenc:");
 gotoxy(23,3); printf("Entr. 16");
 gotoxy(33,3); printf("SB Oper.");
 gotoxy(43,3); printf("(4dig.hex):");
 gotoxy(1,3); printf("fase1=");
 gotoxy(10,3); printf("fase2=");

 gotoxy(1,5); printf("ESTAGIO 1");
 gotoxy(22,5); printf("Operando A:");
 gotoxy(22,6); printf("Operando B:");

 gotoxy(1,8); printf("ESTAGIO 2");
 gotoxy(13,8); printf("Sinal:");
 gotoxy(23,8); printf("Exp.Pol.:");
 gotoxy(38,8); printf("ProdutoParcial:");
 gotoxy(1,9); printf("ExpMaxA:");
 gotoxy(17,9); printf("ExpMinA:");
 gotoxy(32,9); printf("MantMinA:");
 gotoxy(47,9); printf("A:");
 gotoxy(1,10); printf("ExpMaxB:");
 gotoxy(17,10); printf("ExpMinB:");
 gotoxy(32,10); printf("MantMinB:");
 gotoxy(47,10); printf("B:");
```

```
gotoxy(1,12); printf("ESTAGIO 3");
gotoxy(13,12); printf("Invalido:");
gotoxy(25,12); printf("Denorm:");
gotoxy(35,12); printf("SelExc:");
gotoxy(1,13); printf("MantMax:");
gotoxy(12,13); printf("ExpMax:");
gotoxy(1,14); printf("Sinal:");
gotoxy(47,13); printf("A:");
gotoxy(10,14); printf("ExpPol:");
gotoxy(22,14); printf("Produto:");
gotoxy(25,13); printf("Guard:");
gotoxy(34,13); printf("Round:");
gotoxy(47,14); printf("B:");

gotoxy(1,16); printf("ESTAGIO 4");
gotoxy(13,16); printf("Invalido:");
gotoxy(25,16); printf("Denorm:");
gotoxy(35,16); printf("SelExc:");
gotoxy(1,17); printf("MantMax:");
gotoxy(12,17); printf("ExpMax:");
gotoxy(1,18); printf("Sinal:");
gotoxy(47,17); printf("A:");
gotoxy(10,18); printf("ExpPol:");
gotoxy(22,17); printf("CarryProd:");
gotoxy(34,17); printf("CarryArred:");
gotoxy(22,18); printf("ProdArred:");
gotoxy(45,16); printf("Inexato:");
gotoxy(47,18); printf("B:");

gotoxy(1,20); printf("ESTAGIO 5");
gotoxy(14,20); printf("Invalido:");
gotoxy(27,20); printf("Denorm:");
gotoxy(38,20); printf("Inexato:");
gotoxy(5,21); printf("Overf:");
gotoxy(14,21); printf("Underf:");
gotoxy(27,21); printf("SelExc:");
gotoxy(5,22); printf("Sinal:");
gotoxy(14,22); printf("ExpCalc:");
gotoxy(27,22); printf("MantCalc:");
gotoxy(47,21); printf("A:");
gotoxy(47,22); printf("B:");

gotoxy(1,24); printf("ESTAGIO 6");
gotoxy(1,25); printf("Inval:");
gotoxy(9,25); printf("Denorm:");
gotoxy(18,25); printf("Inexato:");
gotoxy(28,25); printf("Overf:");
gotoxy(36,25); printf("Underf:");
gotoxy(47,24); printf("A:");
gotoxy(23,24); printf("SAIDA:");
gotoxy(47,25); printf("B:");

gotoxy(43,2);

}

/* AREA DE FUNCOES DEFINIDAS PELO USUARIO PARA DESCRICAO */

char OU_bit_a_bit (unsigned long int x)
{
  char y;
  y = (x!=0) ? true : false;
  return (y);
}
```

```
char E_bit_a_bit (unsigned x)
{
 char y;
 y = (x==0xFF) ? true : false;
 return (y);
}

unsigned long separa (unsigned x)
{
 return(((unsigned long)x & MASC7)<<16);
}


unsigned long concatenacao (char sinal, unsigned expoente,
                            unsigned long mantissa)
{
 unsigned long auxexp;

 auxexp = (unsigned long)expoente;
 auxexp = auxexp<<23;
 auxexp = sinal ? (auxexp | (1l<<31)) : auxexp;
 return(auxexp | mantissa);
}


char func(int i)
{ char a;
  void *pnt;
  pnt= _DS;
   _DS= 0xB800;
  a=(char) *((char *)i);
   _DS = pnt;
  return(a);
}

/*------------------------------------------------------*/


main()

{

 static circuito *sa, *sn;
 char f1, f2, fase, ch;
 char A, B, C, D;
 int contador, estado;
 unsigned int entrada;
 int ind, cc;

 char sest2A, sest3A, sest4A, sest5A,
      sest2B, sest3B, sest4B, sest5B;
 unsigned eest2A, eest3A, eest4A, eest5A,
          eest2B, eest3B, eest4B, eest5B;
 unsigned long mest2A, mest3A, mest4A, mest5A,
               mest2B, mest3B, mest4B, mest5B;


 sa = (circuito*) calloc(1,sizeof(circuito));
 sn = (circuito*) calloc(1,sizeof(circuito));

 desenhatela();

 if (sa == NULL || sn == NULL)
   {
    printf("Nao foi possivel alocar memoria");
    exit(1);
   }
```

```
/* -------------------- inicializacao ----------------- */

f1=f2=false;
fase=true;
A=B=C=D=false;
contador=estado=0;
ch='Z';

/*----------------------------------------------------------*/

while ((ch=getch())!='s')
 {
    /* --entrada de dados e
          mudanca de valor variaveis controle -- */
  static int prim;

  if (ch=='@') {
  FILE *lixo;
  int i;

  if (prim==0) lixo=fopen("lixo","w");
  else lixo=fopen("lixo","a");

  prim=1;

  for (i=0;i<80*25*2;i+=2)
  {
     { if (((i+2) % 160)==0)
        { putc('',lixo); }
        else
         { putc((int)func(i),lixo); }
     }
  }
  putc('',lixo);
  putc('',lixo);
  fclose(lixo);

  }

  if (f1)  {gotoxy(54,3); printf("     ");}
  if (f1 & (A|C)) {gotoxy(32,3); printf("M");}
  if (f1 & (B|D)) {gotoxy(32,3); printf("L");}
  if (f1 & (A|B)) {gotoxy(41,3); printf("A");}
  if (f1 & (C|D)) {gotoxy(41,3); printf("B");}

  if (f1|f2)
   {
    gotoxy(7,3); printf("%d",bool(f1));
    gotoxy(7,2); printf("%d",contador);
    gotoxy(16,3); printf("%d",bool(f2));
    gotoxy(18,2);
    if (A) printf("A");
    if (B) printf("B");
    if (C) printf("C");
    if (D) printf("D");
   }

  if (f1)
   {
    gotoxy(54,3);
    scanf ("%x", &entrada);
   }
```

```
if (ch=='w')
 {
  printf("para terminar troca, digite -1");
  printf("troca do valor da variavel numero  (de 0 a %d):)",
         sizeof(circuito));
  ind = 1;
  while (ind>=0)
   {
    scanf("%i", &ind);
    if (ind>=0)
     {
      cargauno(sa,~valor(sa,ind),ind);
      cargauno(sn,~valor(sn,ind),ind);
      printf ("novo valor da variavel %d = %d ",
              ind, bool(valor(sa,ind)));
     }
    else
     {
      ch = 's'; break;
     }
   }
 }
/*---------------------------------------------------------*/
 cc=0; cont=0;
 while((!cc)||(bool(~f1)&&bool(~f2)))
  {
   if (++cont >= 100)
   {
    printf("Valores nao estabilizaram apos %d iteracoes",
           cont);
    break;
   }
/* ------------ DESCRICAO DO CIRCUITO ----------------*/
/*------------- ESTAGIO 1 ----------------------------*/
/*------------- Entrada de Dados ---------------------*/
   if (f1 & A)
    {
     sn -> sinalA1 = (entrada & MASC16) ? true : false;
     sn -> expA1 = ((entrada >> 7) & MASC8);
     sn -> mantA1 = separa(entrada);
    }

   if (f1 & B)
     (sn -> mantA1 = sa -> mantA1 | entrada);

   if (f1 & C)
    {
     sn -> sinalB1 = (entrada & MASC16) ? true : false;
     sn -> expB1 = ((entrada >> 7) & MASC8);
     sn -> mantB1 = separa(entrada);
    }

   if (f1 & D)
     (sn -> mantB1 = sa -> mantB1 | entrada);
   if (f2 & D)
    {
     sn -> sinalA2=sa -> sinalA1;
     sn -> sinalB2=sa -> sinalB1;
     sn -> expA2=sa -> expA1;
     sn -> expB2=sa -> expB1;
     sn -> mantA2=sa -> mantA1;
     sn -> mantB2=sa -> mantB1;
    }
```

```
/*------------ ESTAGIO 2 -----------------------------*/
/*------------ Multiplicador -------------------------*/

    if ((f1 & C)|(f1 & D))
     (sn->entrmultA=(((sa->mantA2) >> 12) & MASC12));
    if ((f1 & A)|(f1 & B))
     (sn->entrmultA=(sa->mantA2 & MASC12));
    if ((f1 & B)|(f1 & D))
     (sn->entrmultB=(((sa->mantB2) >> 12) & MASC12));
    if ((f1 & A)|(f1 & C))
     (sn->entrmultB=(sa->mantB2 & MASC12));

    if ((f1 & A)|(f1 & B))
     sn->parcelaA = sa->entrmultA;
    if ((f1 & C)|(f1 & D))
     sn->parcelaA = sa->entrmultA | MSUM;
    if ((f1 & A)|(f1 & C))
     sn->parcelaB = sa->entrmultB;
    if ((f1 & B)|(f1 & D))
     sn->parcelaB = sa->entrmultB | MSUM;

    if (f1)
     sn->produtoparcial=(sa->parcelaA * sa->parcelaB);

/*------------ Somador de Expoentes ------------------*/

    if (f1 & A) (sn->somexp1 = (sa->expA2 + sa->expB2));

/*------------ Calculo Sinal do Produto --------------*/

    if (f1 & A) (sn->sinalresult1 =
                          (sa->sinalA2 ^ sa->sinalB2));

/*------------ Analise das Mantissas -----------------*/

    if ((f1 & A)|(f1 & C))
     (sn->OUmantA1 = OU_bit_a_bit(sa->entrmultA));
    if (f1 & A)  (sn->OUmantA2 = sa->OUmantA1);
    if (f1 & C)
     sn->mantminA = (sa->OUmantA1 | sa->OUmantA2) ?
                    false : true;

    if ((f1 & A)|(f1 & B))
     (sn->OUmantB1 = OU_bit_a_bit(sa->entrmultB));
    if (f1 & A)  (sn->OUmantB2 = sa->OUmantB1);
    if (f1 & B)
     sn->mantminB = (sa->OUmantB1 | sa->OUmantB2) ?
                    false : true;

/*------------ Analise dos Expoentes -----------------*/

    if (f1 & A)
     {
      (sn->expmaxA = E_bit_a_bit(sa->expA2));
      (sn->expminA = ~OU_bit_a_bit((unsigned long)sa->expA2));
      (sn->expmaxB = E_bit_a_bit(sa->expB2));
      (sn->expminB = ~OU_bit_a_bit((unsigned long)sa->expB2));
     }
```

```
/*----------- ESTAGIO 3 ----------------------------*/
    if (f2 & A)
     {
      sn->sinalresult2 = sa->sinalresult1;
      sn->somexp2 = sa->somexp1;
     }

/*----------- Adicao dos Produtos Parciais ------------*/

    if (f2)  (sn->entrshifter = sa->produtoparcial);

    if (f2 & A)
     {
      sn->prod = ((sa->entrshifter)>>12);
      sn->prod_desl = ((unsigned)(sa->entrshifter) & MASC12);
     }
    if (f2 & (B|C|D))  (sn->prod = sa->entrshifter);

    if (f2 & A)
     {
      sn->bit_arredond =
            OU_bit_a_bit((unsigned long)sa->prod_desl);
      sn->reg1 = sa->prod;
     }

    if (f1 & (B|C|D))  (sn->reg2 = sa->reg1);

    if (f2 & (B|C|D))
     {
      sn->entrUlaA = sa->reg2;
      sn->entrUlaB = sa->prod;
      sn->soma = (sa->entrUlaA + sa->entrUlaB);
     }

    if (f2 & C)
     {
      sn->soma_desl = (sa->soma)>>12;
      sn->bits_desl = ((unsigned)(sa->soma) & MASC12);
     }
    if (f2 & (B|D))  (sn->soma_desl = sa->soma);

    if (f2 & (B|C))  (sn->reg1 = (sa->soma_desl & MASC24));

    if (f2 & C)
     {
      sn->LSBprod = ((sa->bits_desl >> 11) == 1) ? true : false;
      sn->round =
      (OU_bit_a_bit((unsigned long)(sa->bits_desl & MASC11)) |
       sa->bit_arredond);
      sn->guard = ((sa->bits_desl >> 10) == 1) ? true : false;
     }

    if (f2 & D)
     {
      sn->produto = sa->LSBprod ? ((sa->soma_desl<<1)+1) :
                                   (sa->soma_desl<<1);
      sn->guard_bit = sa->guard;
      sn->round_bit = sa->round;
     }
```

```
/*------------- Controle de excecao ---------------------*/

     if (f2 & C)
      {
       sn->selec_excecao1 = (sa->expminA | sa->expminB |
                             sa->expmaxA | sa->expmaxB);
       sn->invalido1 = (sa->expmaxA & (~sa->mantminA)) |
                       (sa->expmaxB & (~sa->mantminB)) |
                       (sa->expmaxA & sa->mantminA &
                        sa->expminB) |
                       (sa->expmaxB & sa->mantminB &
                        sa->expminA);
       sn->denorm1 = (sa->expminA & (~sa->mantminA))|
                     (sa->expminB & (~sa->mantminB));
       sn->mantmax1 = sa->invalido1;
       sn->expmax1 = ~(((~sa->expminA) & sa->expminB)|
                       ((~sa->expmaxB) & sa->expminA));
      }
/*------------- ESTAGIO 4 -------------------------------*/

     if (f1 & A)
      {
       sn->invalido2 = sa->invalido1;
       sn->denorm2 = sa->denorm1;
       sn->selec_excecao2 = sa->selec_excecao1;
       sn->mantmax2 = sa->mantmax1;
       sn->expmax2 = sa->expmax1;
       sn->sinalresult3 = sa->sinalresult2;
       sn->somexp3 = sa->somexp2;
      }

/*------------- Renormalizacao e Arredondamento ---------*/

     if (f1 & A)
      {
       sn->carry_out_prod = ((sa->produto>>24)==1) ?
                            true : false;
       sn->prod_renorm1 = (sa->carry_out_prod) ?
                          (sa->produto>>1) : (sa->produto);
       sn->bit_arred1 = (sa->carry_out_prod) ?
                        (sa->produto % 2) : (sa->guard_bit);
       sn->bit_arred2 = (sa->carry_out_prod) ?
                        (sa->guard_bit) : (sa->round_bit);
       sn->bit_arred3 = (sa->carry_out_prod) ?
                         sa->round_bit : false;

       sn->prod_arredond = (((sa->bit_arred1) &
           (sa->bit_arred2 | sa->bit_arred3)) |
           ((sa->bit_arred1) & (((sa->produto % 2)==1) ?
           true : false))) ? (sa->prod_renorm1+1) :
           sa->prod_renorm1;

       sn->carry_out_arred = ((sa->prod_arredond>>24)==1) ?
                              true : false;

       sn->prod_renorm2 = (sa->carry_out_arred) ?
                          ((sa->prod_arredond>>1) & MASC23) :
                          ((sa->prod_arredond) & MASC23);

       sn->inexato1 = ((sa->bit_arred1 | sa->bit_arred2 |
                      sa->bit_arred3)) ? true : false;

      }
```

```
/*------------ ESTAGIO 5 ------------------------------*/
    if (f2 & A)
     {
      sn->invalido3 = sa->invalido2;
      sn->denorm3 = sa->denorm2;
      sn->sinalresult4 = sa->sinalresult3;
      sn->mantcalc = sa->prod_renorm2;

/*------------- Ajuste Bias Expoente -------------------*/

      sn->carry_in = sa->carry_out_prod | sa->carry_out_arred;
      sn->expcalc = (sa->carry_in)?
                    ((sa->somexp3 - 127 +1) & MASC8) :
                    ((sa->somexp3 - 127) & MASC8);

/*------------ Verifica Overflow/Underflow Expoente ----*/

      sn->exp_calc_pol = (sa->carry_in) ?
                    (sa->somexp3 + 1) : (sa->somexp3);
      sn->overf_exp = (sa->exp_calc_pol > 382) ? true : false;
      sn->underf_exp = (sa->exp_calc_pol < 128) ?
                    true : false;

/*------------- Calculo Excecao ------------------------*/

      sn->result_excecao = (sa->selec_excecao2 |
                    sa->overf_exp | sa->underf_exp);
      sn->inexato2 = ((sa->inexato1 | sa->overf_exp |
                    sa->underf_exp) &
                    (~sa->selec_excecao2));
      sn->overflow1 = sa->overf_exp & (~sa->selec_excecao2);
      sn->underflow1 = sa->underf_exp & (~sa->selec_excecao2);
      sn->mant_exc_max = sa->selec_excecao2 & sa->mantmax2;
      sn->exp_exc_max = ((sa->selec_excecao2 & sa->expmax2) |
                    (sa->overf_exp & (~sa->selec_excecao2)));

     }
/*------------ ESTAGIO 6 ------------------------------*/
/*------------- Saida Status ---------------------------*/

    if (f1 & B)
     {
      sn->invalido = sa->invalido3;
      sn->denorm = sa->denorm3;
      sn->inexato = sa->inexato2;
      sn->overflow = sa->overflow1;
      sn->underflow = sa->underflow1;

/*------------- Saida Resultado ------------------------*/

      sn->result = (sa->result_excecao) ?
       concatenacao(sa->sinalresult4, sa->exp_exc_max ? 0xFF : 0,
                    sa->mant_exc_max ? 0x7FFFFFL : 0) :
       concatenacao(sa->sinalresult4, sa->expcalc, sa->mantcalc);

      sn->saida = (unsigned)(sa->result>>16);
     }

    if (f1 & C)  (sn->saida = (unsigned)(sa->result & 0xFFFF));

/*---------------------------------------------------*/
```

```
/*------------- c o n t r o l e -------------------------*/

  if ( ( cc=changecomp(&sn,&sa,sizeof(circuito)) ) == 1 )
    { if (f1 || f2)
        {

        /*--------- ESTAGIO 1 ------------*/
        gotoxy(35,5); printf("%d",bool(sa->sinalA1));
        gotoxy(38,5); printf("%02X",sa->expA1);
        gotoxy(42,5); printf("%061X",sa->mantA1);
        gotoxy(35,6); printf("%d",bool(sa->sinalB1));
        gotoxy(38,6); printf("%02X",sa->expB1);
        gotoxy(42,6); printf("%061X",sa->mantB1);

        /*---------- ESTAGIO 2 ------------*/
        gotoxy(19,8); printf("%d",bool(sa->sinalresult1));
        gotoxy(32,8); printf("%03X",sa->somexp1);
        gotoxy(53,8); printf("%061X",sa->produtoparcial);
        gotoxy(9,9); printf("%d",bool(sa->expmaxA));
        gotoxy(25,9); printf("%d",bool(sa->expminA));
        gotoxy(41,9); printf("%d",bool(sa->mantminA));
        gotoxy(9,10); printf("%d",bool(sa->expmaxB));
        gotoxy(25,10); printf("%d",bool(sa->expminB));
        gotoxy(41,10); printf("%d",bool(sa->mantminB));

        /*---------- ESTAGIO 3 ---------------*/
        gotoxy(22,12); printf("%d",bool(sa->invalido1));
        gotoxy(32,12); printf("%d",bool(sa->denorm1));
        gotoxy(42,12); printf("%d",bool(sa->selec_excecao1));
        gotoxy(9,13); printf("%d",bool(sa->mantmax1));
        gotoxy(19,13); printf("%d",bool(sa->expmax1));
        gotoxy(7,14); printf("%d",bool(sa->sinalresult2));
        gotoxy(17,14); printf("%03X",sa->somexp2);
        gotoxy(30,14); printf("%061X",sa->produto);
        gotoxy(31,13); printf("%d",bool(sa->guard_bit));
        gotoxy(40,13); printf("%d",bool(sa->round_bit));

        /*----------- ESTAGIO 4 -----------------*/
        gotoxy(22,16); printf("%d",bool(sa->invalido2));
        gotoxy(32,16); printf("%d",bool(sa->denorm2));
        gotoxy(42,16); printf("%d",bool(sa->selec_excecao2));
        gotoxy(9,17); printf("%d",bool(sa->mantmax2));
        gotoxy(19,17); printf("%d",bool(sa->expmax2));
        gotoxy(7,18); printf("%d",bool(sa->sinalresult3));
        gotoxy(17,18); printf("%03X",sa->somexp3);
        gotoxy(32,17); printf("%d",bool(sa->carry_out_prod));
        gotoxy(45,17); printf("%d",bool(sa->carry_out_arred));
        gotoxy(32,18); printf("%061X",sa->prod_renorm2);
        gotoxy(53,16); printf("%d",bool(sa->inexato1));

        /*----------- ESTAGIO 5 -----------------*/
        gotoxy(23,20); printf("%d",bool(sa->invalido3));
        gotoxy(34,20); printf("%d",bool(sa->denorm3));
        gotoxy(46,20); printf("%d",bool(sa->inexato2));
        gotoxy(11,21); printf("%d",bool(sa->overflow1));
        gotoxy(21,21); printf("%d",bool(sa->underflow1));
        gotoxy(34,21); printf("%d",bool(sa->result_excecao));
        gotoxy(11,22); printf("%d",bool(sa->sinalresult4));
        gotoxy(22,22); printf("%02X",sa->expcalc);
        gotoxy(36,22); printf("%061X",sa->mantcalc);

        /*------------ ESTAGIO 6 -----------------*/
        gotoxy(7,25); printf("%d",bool(sa->invalido));
        gotoxy(16,25); printf("%d",bool(sa->denorm));
        gotoxy(26,25); printf("%d",bool(sa->inexato));
        gotoxy(34,25); printf("%d",bool(sa->overflow));
        gotoxy(43,25); printf("%d",bool(sa->underflow));
        gotoxy(29,24); printf("%04X",sa->saida);
```

```
/*------------------------------------*/
if (contador > 4)
{
 if (f1 & A)
 {
  gotoxy(49,9); printf("%d",bool((sest2A=sa->sinalA2)));
  gotoxy(51,9); printf("%02X",(eest2A=sa->expA2));
  gotoxy(54,9); printf("%06lX",(mest2A=sa->mantA2));
  gotoxy(49,10); printf("%d",bool((sest2B=sa->sinalB2)));
  gotoxy(51,10); printf("%02X",(eest2B=sa->expB2));
  gotoxy(54,10); printf("%06lX",(mest2B=sa->mantB2));
 }
 if (f2 & A)
 {
  gotoxy(49,13); printf("%d",bool((sest3A=sest2A)));
  gotoxy(51,13); printf("%02X",(eest3A=eest2A));
  gotoxy(54,13); printf("%06lX",(mest3A=mest2A));
  gotoxy(49,14); printf("%d",bool((sest3B=sest2B)));
  gotoxy(51,14); printf("%02X",(eest3B=eest2B));
  gotoxy(54,14); printf("%06lX",(mest3B=mest2B));
 }
}

if (contador > 8)
{
 if (f1 & A)
 {
  gotoxy(49,17); printf("%d",bool((sest4A=sest3A)));
  gotoxy(51,17); printf("%02X",(eest4A=eest3A));
  gotoxy(54,17); printf("%06lX",(mest4A=mest3A));
  gotoxy(49,18); printf("%d",bool((sest4B=sest3B)));
  gotoxy(51,18); printf("%02X",(eest4B=eest3B));
  gotoxy(54,18); printf("%06lX",(mest4B=mest3B));
 }
 if (f2 & A)
 {
  gotoxy(49,21); printf("%d",bool((sest5A=sest4A)));
  gotoxy(51,21); printf("%02X",(eest5A=eest4A));
  gotoxy(54,21); printf("%06lX",(mest5A=mest4A));
  gotoxy(49,22); printf("%d",bool((sest5B=sest4B)));
  gotoxy(51,22); printf("%02X",(eest5B=eest4B));
  gotoxy(54,22); printf("%06lX",(mest5B=mest4B));
 }
 if (f1 & B)
 {
  gotoxy(49,24); printf("%d",bool((sest5A)));
  gotoxy(51,24); printf("%02X",(eest5A));
  gotoxy(54,24); printf("%06lX",(mest5A));
  gotoxy(49,25); printf("%d",bool((sest5B)));
  gotoxy(51,25); printf("%02X",(eest5B));
  gotoxy(54,25); printf("%06lX",(mest5B));
 }
}
/*----------------------*/

gotoxy(43,2);

f1=f2=false;

 }
else
 {
  f1=fase;
  fase=~fase;
  f2=fase;
```

```
                if (f1)
                 {
                   estado = (contador++ % 4);
                   if (estado==0) { A=true; D=false; }
                   if (estado==1) { B=true; A=false; }
                   if (estado==2) { C=true; B=false; }
                   if (estado==3) { D=true; C=false; }
                 }
               }
             }
         }
       }
clrscr();
return;
}
```

BIBLIOGRAFIA

[ANA 8?] ANALOG DEVICES. 32-Bit IEEE Floating-Point **Chipset**. Norwood, s.d. 34p. (ADSP-3201 / ADSP-3202)

[ANA 87] ANALOG DEVICES. 64-Bit IEEE Floating-Point **Chipset**. Norwood, May 1987. 58p. (ADSP-3212 / ADSP-3222)

[BOO 80] BOOTH, A. A Signed binary multiplication technique. In: SWARTZLANDER, E. E., ed. **Computer Arithmetic**. Stroudsburg, Dowden, Hutchinson & Ross. 1980. part 3:multiplication, p.100-104. (Benchmark Papers in Electrical Engineering and Computer Science v.21).

[BON 88] BONONI, Mauro. Avoiding coprocessors bottlenecks. **Byte**, Peterborough, 13(3):197-204, Mar. 1988.

[BOS 87] BOSE, B. K. et al. Fast multiply and divide for a VLSI floating-point Unit. In: SYMPOSIUM ON COMPUTER ARITHMETIC, 8., Como, Italy, May 19-21, 1987. **Proceedings**. New York, IEEE, 1987. p.87-94.

[BUR 88] BURSKI, D. Speedy arrays accelerate systems. **Electronic Design**, Rochelle Park, 36(26):61-74, Nov. 23, 1988.

[BUR 89] BURSKI, D. Advanced CISC processor catch up to RISC speeds. **Electronic Design**, Rochelle Park, 37(16):41-8, July 27, 1989.

[CAN 86] CAND, M. et al. **Conception des circuits integrés MOS**: elements de base - perspectives. Paris, Eyrolles, 1986. 437p.

[CAS 89] CASE, B. RISC hits its first million. **ESD : The Electronic System Design Magazine**, Boston, 19(6):45-50, June 1989.

[CAV 85] CAVANAGH, Joseph J. F. **Digital Computer Arithmetic**: Design and Implementation. s.l., McGraw-Hill, 1984. 459p.

[CLE 88] CLETO, Laerte Davi. **Processadores Aritméticos Integrados**. Porto Alegre, CPGCC da UFRGS, 1988. TI n.71.

[COO 81] COONEN, Jerome T. Underflow and Denormalized Numbers. **Computer**, Los Alamitos, 14(3):75-87, Mar 1981.

[DAV 83] DAVIO, M.; DESCHAMPS, J.-P.; THAYSE, A. **Digital systems**: with algorithm implementation. Chichester, John Wiley & Sons, 1983. 505p.

[DRA 88] DRAFZ, R. Turn a PC into a super computer with plug in boards. **Electronic Design**, Rochelle Park, 36(26):89-93, Nov 23, 1988.

[FAN 85] FANDRIANTO, Jan & WOO, B. Y. VLSI Floating-Point Processors. In: SYMPOSIUM ON COMPUTER ARITHMETIC, 7., Urbana, June 4-6, 1985. **Proceedings**. New York, IEEE, 1985. p.93-100.

[GLA 85] GLASSER, L. A. & DOBBERPUHL, D. W. **The design and analysis of VLSI circuits**. Reading, Addison-Wesley, 1985. 473p.

[GOS 80] GOSLING, John B. **Design of Arithmetic Units for Digital Computers**. London, Macmillan. 1980. 139p.

[GUN 88] GUNN, L. At 100 MFLOPS, the fastest DSP chip ever!. **Electronic Design**, Rochelle Park, 36(23):73-6, Oct 13, 1988.

[HWA 79] HWANG, Kai. **Computer Arithmetic: principles, architecture, and design**. New York, John Wiley, 1979. 423p.

[HWA 84] HWANG, K. & BRIGGS, F. A. **Computer Architecture and parallel processing**. New York, McGraw-Hill, 1984. 846p.

[IEE 87] IEEE. IEEE Standard for Binary Floating-Point Arithmetic. New York. 1985. ANSI/IEEE Std 754-1985. **SIGPLAN Notices**, New York, 22(2):9-25, Feb. 1987.

[INT 87] INTEL CORPORATION. **80387 80-Bit CHMOS III Numeric Processor Extension**. Oct. 1987. 37p.

[INT 89] INTEL CORPORATION. **i486™ MICROPROCESSOR**. Apr. 1989. 175p.

[INT 89a] INTEL CORPORATION. **A performance report on the 386 Family of high performance, 32-bits microprocessor**. Apr. 1989. 38p.

[MOT 87] MOTOROLA. **MC68881 / MC68882 Floating-Point Coprocessor User's Manual**. Englewood Cliffs, Prentice-Hall, 1987.

[MOT 88] MOTOROLA. **MC88100 RISC Microprocessor User's Manual**. 1988.

[PEN 87] PENG, V. et al. On the implementation of shifters, multipliers, and dividers in VLSI floating-point units. In: SYMPOSIUM ON COMPUTER ARITHMETIC, 8., Como, Italy, May 19-21, 1987. **Proceedings**. New York, IEEE, 1987. p.95-105.

[SHA 87] SHARMA, R. Area-time efficient arithmetic elements for VLSI System. In: SYMPOSIUM ON COMPUTER ARITHMETIC, 8., Como, Italy, May 19-21, 1987. **Proceedings**. New York, IEEE, 1987. p.57-62.

[SUZ 89] SUZIM, Altamiro Amadeu & SANTOS, Luiz Cláudio Villar dos. HDC - Uma técnica de descrição de hardware para simulação funcional. In: SEMINÁRIO INTERNO DE MICROELETRÔNICA, 5., Tramandaí, 17-18 nov. 1989. **Anais**. Porto Alegre, UFRGS/CPGCC/GME. 1989. p.81-84. Comunicação técnica.

[TEX 83] TEXAS INSTRUMENTS. **TM 32010 user's guide. 16/32 bit digital signal processors.** 1983.

[VOL 80] VOLDER, J. E. The CORDIC trigonometric computing technique. In: SWARTZLANDER, E. E., ed. **Computer Arithmetic.** Stroudsburg, Dowden, Hutchinson & Ross. 1980. part 6: elementary functions, p. 226-30. (Benchmark Papers in Electrical Engineering and Computer Science v.21).

[WAL 80] WALLACE. C. S. A sugestion for a fast multiplier. In: SWARTZLANDER, E. E., ed. **Computer Arithmetic.** Stroudsburg, Dowden, Hutchinson & Ross. 1980. part 3: multiplication, p.114-118 (Benchmark Papers in Electrical Engineering and Computer Science v.21).

[WAL 80a] WALTHER, J. S. A Unified Algorithm for Elementary Functions. In: SWARTZLANDER, E. E., ed. **Computer Arithmetic.** Stroudsburg, Dowden, Hutchinson & Ross. 1980. part 6: elementary functions, p.272-8. (Benchmark Papers in Electrical Engineering and Computer Science v.21).

[WEI 86] WEITEK. **WTL 1232 / 1233 32 bit floating point multiplier and ALU.** July 1986. 26p.

[WEI 86a] WEITEK. **WTL 2264 / 2265 64 bit floating point multiplier/divider and ALU.** July 1986. 47p.

[WES 85] WESTE, N. & ESHRAGHIAN, K. **Principles of CMOS VLSI Design - A System Perspective**. Reading, Addison-Wesley, 1985. 531p.

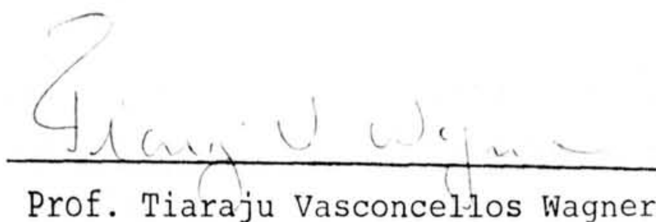
[WIL 89] WILSON, R. 68040 moves toward RISC camp with redesigned pipelines, caches. **Computer Design**, Littleton, 28(9):22, May 1, 1989.

[WIL 89a] WILSON, R. Intel 80486 carries complex instruction set to RISC speeds. **Computer Design**, Littleton, 28(9):18-20, May 1, 1989.

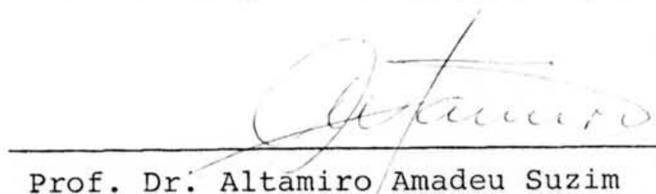
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Projeto de Operadores Aritméticos
de ponto flutuante em tecnologia CMOS.

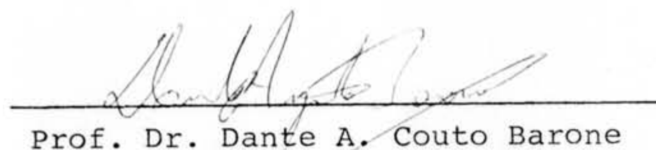
Dissertação apresentada aos Srs.



Prof. Tiaraju Vasconcellos Wagner

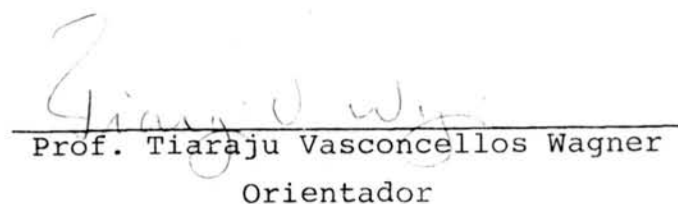


Prof. Dr. Altamiro Amadeu Suzim

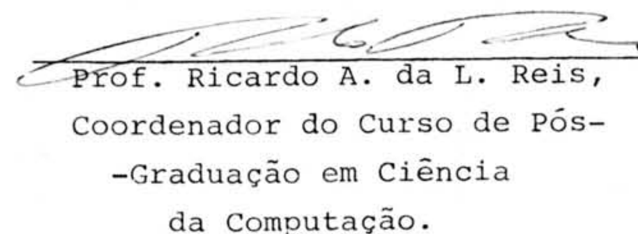


Prof. Dr. Dante A. Couto Barone

Visto e permitida a impressão.
Porto Alegre, 10 / 10 / 1990



Prof. Tiaraju Vasconcellos Wagner
Orientador



Prof. Ricardo A. da L. Reis,
Coordenador do Curso de Pós-Graduação em Ciência da Computação.