RELATÓRIO TÉCNICO

UM LEVANTAMENTO DE ALGUNS MÉTODOS E IMPLEMENTAÇÕES DE OPERAÇÕES DE MULTIPLICAÇÃO E DIVISÃO E SUA APLICABILIDADE EM MICROPROCESSADORES

Manuel Lois Anido
NCE/UFRJ

NCE — 10/92
outubro

Núcleo de **C**omputação **E**letrônica
Universidade Federal do Rio de Janeiro

Tel.: 598-3212 - Fax.: (021) 270-8554
Caixa Postal 2324 - CEP 20001-970
Rio de Janeiro - RJ

Um Levantamento de Alguns Métodos e Implementações de Operações de Multiplicação e Divisão e sua Aplicabilidade em Microprocessadores

Manuel Lois Anido

Núcleo de Computação Eletrônica da UFRJ
e-mail : ncd10121@ufrj

Resumo

Este artigo apresenta um levantamento de alguns métodos e implementações de multiplicação e divisão aplicáveis na unidade aritmética inteira ou ponto flutuante de microprocessadores. O artigo também apresenta uma análise qualitativa e discute as principais vantagens e desvantagens de cada um dos métodos em relação aos demais.

Palavras Chave: Multiplicação, Divisão, Aritmética de Computadores

1. Introdução

Apesar de já ter sido mostrado que RISCs de uso geral não demandam a existência de instruções de multiplicação e divisão rápidas, existem muitas aplicações, numericamente intensivas (e.g. computação gráfica, processamento de sinais, CAD), que podem se beneficiar significativamente da existência de um hardware especializado para executar tais instruções eficientemente.

Alguns processadores RISC utilizam a unidade de ponto flutuante para executar as instruções de multiplicação e divisão inteira [1]. Outros implementam um suporte básico na unidade aritmética inteira, que geralmente consiste em passos de instruções de multiplicação e divisão, para executar estas instruções [2,3]. Existem ainda aqueles que recorrem ao compilador para identificar algumas constantes e acelerar alguns casos [4]. Microprocessadores utilizados em processamento de sinais (DSP) apresentam soluções ainda mais radicais, tais como multiplicadores combinacionais, para acelerar a execução de operações aritméticas [5].

Com o aumento da densidade de integração, tem sido possível colocar mais unidades funcionais nos circuitos integrados. Um exemplo disto tem sido os microprocessadores superescalares, que podem executar diversas instruções simultaneamente [6]. Para otimizar o uso das unidades funcionais, alguns destes RISCs superescalares empregam hardware especializado na unidade aritmética inteira para implementar as instruções de multiplicação e divisão. Desta forma o compilador pode escalonar diversas operações de ponto flutuante para serem executadas concorrentemente com operações aritméticas inteiras. Obviamente, isto só é possível graças à disponibilidade de área.

Na implementação de um circuito integrado, existem sempre compromissos entre velocidade, potência, e área. Por esta razão, os métodos aqui discutidos tentam sempre levar em consideração estes aspectos.

2. Multiplicação

2.1. Introdução

No método padrão de somas e deslocamentos, cada bit do multiplicador gera um múltiplo do multiplicando a ser adicionado ao produto parcial. Quando o multiplicador é muito largo, isto significa que um grande número de multiplicandos tem que ser adicionados. O tempo de execução de uma instrução de multiplicação é determinado principalmente pelo número de adiçoes a serem realizadas. Portanto, é desejável reduzir o número de múltiplos do multiplicando, e consequentemente o número de produtos parciais.

Já foram desenvolvidos métodos para reduzir o número de produtos parciais, e estes se enquadram nos esquemas de *recodificação de números com sinal* [7]. O algoritmo de Booth modificado é um destes métodos e codifica os bits do multiplicador de tal forma que o número de produtos parciais necessário é sòmente metade do requerido no método tradicional de somas e deslocamentos.

O algoritmo de Booth aumenta a velocidade da multiplicação, pulando os 1s do multiplicador. O acréscimo de velocidade depende da configuração de bits do multiplicador. Usando a técnica da *recodificação de pares de bits*, é possível acelerar a multiplicação de tal forma que no máximo $n/2$ produtos parciais sejam gerados para um multiplicador com n bits. Esta técnica permite multiplicar, diretamente, dois números em complemento a dois, independentemente dos sinais dos dois números. Em outras palavras, não há necessidade de pré-complementar o multiplicador ou pós-complementar o produto. O produto estará na representação complemento a dois. Isto implica que a velocidade da multiplicação aumentará em quase 2 vezes, em relação ao método tradicional de somas e deslocamentos.

Esta técnica é derivada do método de Booth, o qual considera uma cadeia de 1's como a diferença de dois números e foi originalmente proposta por MacSorley [8]. A propriedade da cadeia de 1's é indicada abaixo, onde k é o número de 1's consecutivos e i é a posição do 1 mais à direita na cadeia de 1's em consideração. Em outras palavras, a multiplicação por uma sequência de 1's requer sòmente uma subtração, ao invés de k adições. No caso geral, o multiplicador pode conter diversas cadeias de 1's de tamanho variável; assim sendo, a propriedade da cadeia de 1's descrita abaixo tem que ser estendida.

$$2^{(i+k)}-2^{i}=2^{(i+k-1)}+2^{(i+k-2)}+\ldots+2^{(i+1)}+2^{i}$$

A técnica da recodificação de pares de bits (ou algoritmo de Booth modificado) estende a propriedade da cadeia acima verificando pares de bits do multiplicador, em conjunto com o bit que está à direita do par de bits em análise. Cada grupo de três bits do multiplicador é decodificado, de acordo com a tabela 1.

Cada passo da multiplicação consiste na decodificação de três bits $(i+1, i, i-1)$, seguido pela adição da versão apropriada do multiplicando, e o deslocamento à direita do multiplicador de duas posições.

É possível reduzir ainda mais o número de produtos parciais, e em consequência acelerar a multiplicação, empregando métodos de multiplicação que analisem mais do que três bits do multiplicador. No entanto, estes métodos requerem a formação de múltiplos do multiplicando não triviais. Consequentemente, tais técnicas não são apresentam boa relação custo-desempenho.

2.2. Circuito de Multiplicação Sequencial Simples

Um dos circuitos mais simples e diretos para implementar a técnica da recodificação de pares de bits discutida acima é ilustrado na figura 1. Essencialmente, é o mesmo hardware que é usado para implementar o método padrão de somas e deslocamentos. A principal diferença é que três bits do multiplicador são decodificados para selecionar os múltiplos do multiplicando a serem adicionados. Uma característica importante deste método é que os múltiplos do multiplicando a serem usados ($0x$, $1x$, $2x$, $-1x$, $-2x$) são fáceis de gerar. O número de ciclos de relógio (ou passos) necessários para efetuar a operação de multiplicação é igual a $n/2$, onde n é o número de bits do multiplicador.

A principal vantagem deste método é que sua implementação requer uma área modesta. Isto se deve ao fato de que são sòmente necessários um somador de $(n+1)$ bits, um multiplexador e um complementador, conforme ilustrado na figura 1.

2.3. Circuito de Multiplicação Sequencial Usando uma Tabela

Neste método é utilizada uma pequena tabela, que pode ser implementada com registros ou memória RAM, e que armazena diversos múltiplos do multiplicando [9]. Este método é uma variante do método tradicional de somas e deslocamentos de 1 bit. Nesta implementação, o produto parcial é deslocado 3 bits para a direita, após a operação de soma. Isto permite reduzir o número de passos da multiplicação para $n/3$ (n = número de bits do multiplicador) e consequentemente acelerar a

multiplicação, em relação ao método anterior. As principais desvantagens consitem no fato de ser necessário gerar os diversos múltiplos do multiplicando, antes de iniciar a multiplicação, e a área adicional necessária para armazená-los.

2.4 Multiplicação Sequencial Usando Diversos Somadores em Cascata

Uma boa solução de compromisso entre área e velocidade, é utilizar diversos somadores em cascata, implementando um multiplicador iterativo, tal como tem sido empregado em alguns processadores RISC e até mesmo em có-processadores de ponto flutuante [10]. Utilizando *pipelining*, é possível utilizar diversos somadores do tipo *Carry Save Adder* (CSA), pois os bits *vai um* gerados em cada iteração são adicionados no passo seguinte. Estes somadores são mais simples do que os somadores completos tradicionais, pois não levam em consideração o *vai um* do estágio anterior, e consequentemente requerem menos área para sua implementação. O diagrama em blocos da figura 2 ilustra este tipo de multiplicador. Em [10] é descrita a implementação de uma unidade de ponto flutuante que implementa um multiplicador de 64 bits por 64 bits, segundo este método.

2.5. Multiplicação Paralela

Quando o principal objetivo é obter velocidade na multiplicação, os circuitos multiplicadores paralelos, ou também chamados combinacionais, são a opção mais utilizada. A literatura descreve a implementação de diversos multiplicadores deste tipo, seja como unidades autônomas, ou incorporados em processadores especializados (principalmente processadores de sinais - DSPs) [5,7,11].

O tempo de multiplicação é proporcional ao número de adições, pois os tempos de geração e de propagação do bit de *soma* e de *vai um* dominam o tempo de operação. Portanto, um dos aspectos mais importantes no projeto de um multiplicador deste tipo é a redução do número de adições. Bàsicamente, existem duas maneiras de atingir esse objetivo. A primeira maneira é reduzir o número de produtos parciais a serem adicionados e a segunda maneira é reduzir o número de estágios somadores. O método de Booth pode ser utilizado para implementar o circuito da primeira maneira e a *árvore de Wallace* da segunda maneira [7,9]. Entretanto, a *árvore de Wallace* apresenta uma regularidade muito pobre, e por essa razão o método tem sido pouco empregado em VLSI.

Em [12] discute-se o projeto de um multiplicador paralelo que utiliza o método de Booth modificado para implementar um multiplicador em forma matricial. O método de Booth modificado permite implementar um multiplicador com aproximadamente metade do número de produtos parciais de um multiplicador matricial standard.

A figura 3 ilustra o esquema básico do multiplicador discutido em [12]. No último estágio da multiplicação é normalmente empregado um somador completo rápido de $2n$ bits, geralmente um somador empregando uma cadeia de vai um do tipo Manchester (*Manchester Carry Chain*), com sinais de *Propagação, Geração e Nulificação.*

3. Divisão

3.1. Introdução

Na maioria das aplicações, a divisao é a instrução menos utilizada dentre as instruções de ponto flutuante básicas e certamente também é a mais difícil de acelerar. A principal consequência destes fatos tem sido que, no passado, o seu tempo de execução tem sido normalmente muito maior do que aquele da multiplicação e adição. No entanto, existem diversas aplicações para as quais um tempo de divisão significativamente mais longo do que as outras instruções é inaceitável. Isto pode ocorrer, por exemplo, em sistemas de tempo real, onde é mandatório minimizar o tempo de resposta das interrupções e onde não seja possível subdividir a instrução de divisão. Em outros casos, como por exemplo na operação de recorte (clipping) tridimensional em computação gráfica, pode ser necessário evitar um gargalo no pipeline gráfico, causado pela lentidão da operação de divisão. Portanto, para as aplicações que são sensíveis à velocidade da instrução mais lenta, é importante acelerar a operação de divisão.

Alguns microprocessadores RISC suportam operações de divisão inteira, provendo instruções que são passos do algoritmo de divisão básico e geralmente geram um bit do quociente em cada iteração [2,3]. Em outros [1], a operação de divisão (inteira ou flutuante) é executada pela unidade de ponto flutuante.

3.2. Divisão Sequencial pelo Método de Subtração e Deslocamento com Pseudo Restauração

Este é o método mais comumemente encontrado em microprocessadores. A divisão, sem sinal, é efetuada em passos de instruções , gerando um bit do quociente de cada vez. Operandos com sinal tem que ser previamente convertidos para positivo e o resultado tem que ser novamente convertido, ou não, dependendo do sinal relativo dos operandos.

A divisão é realizada mediante subtrações sucessivas do divisor do resto parcial deslocado. Se o resultado desta subtração for positivo, ele é deslocado e carregado novamente no registro do resto parcial. Entretanto, se o resultado é negativo, o resto parcial anterior é deslocado à esquerda e carregado no registro do resto parcial (pseudo-restauração). A implementação deste método é ilustrada na figura 4. A detecção de transbordo pode ser implementada facilmente, testando o sinal do quociente da divisão sem sinal. Se for obtido um quociente negativo dessa divisão sem sinal é porque houve um transbordo.

As principais vantagens deste método consistem na simplicidade e no reduzido hardware necessário para sua implementação. Objetivando economizar área de silício, a maioria dos processadores RISC usa a ULA da unidade aritmética inteira para realizar a subtração.

As principais desvantagens do método são o fato de só prover um bit de quociente por iteração e não lidar com números negativos. O número de passos necessários para a conversão de sinal e detecção de transbordo é uma parte significativa do algoritmo. No caso do microprocessador descrito em [2], aproximadamente 43% dos passos necessários para realizar uma divisão (64 por 32 bits) são passos de conversão de sinal e detecção de transbordo, ou seja, não realizam nenhuma tarefa de divisão.

Em [12] discute-se uma variante da implementação descrita acima, que realiza a divisão de números com sinal, a um custo de hardware adicional baixo e torna os passos de conversão de sinal transparentes. O mecanismo básico é realizar a conversão de sinal *on-the-fly*, quando os operandos são carregados e quando o resultado é lido. Este mecanismo é implementado utilizando um multiplexador de n bits de 3:1, n inversores, um meio-somador de n bits e algumas portas lógicas. Desta forma, o tempo gasto na operação de divisão é substancialmente reduzido.

3.3. Divisão por Convergência

Este método de divisão é baseado em multiplicações sucessivas e pressupõe o uso de um multiplicador rápido, tal como um multiplicador paralelo. Basicamente, há dois tipos de métodos de divisão multiplicativos e ambos empregam o hardware da multiplicação para executar a operação de divisão através de convergencia [7,9].

O primeiro método realiza a divisão gerando o inverso do divisor através de um processo iterativo e então multiplicando do dividendo por este inverso para assim obter o quociente. Portanto, para avaliar A/B , é necessário computar $A(1/B)$. O método iterativo de Newton-Raphson é utilizado para calcular $1/B$. Este método converge quadraticamente para o quociente. A principal desvantagem deste método é a necessidade de se ter uma ROM suficientemente grande para obter uma boa primeira aproximação do quociente, de forma a convergir em poucas iterações.

O segundo método multiplicativo consiste em multiplicar tanto numerador como denominador pela mesma sequência de *fatores de convergencia*, até que o denominador tenda a 1. O numerador resultante da operação é o quociente. Da mesma forma que o método anterior, este método também converge quadraticamente em direção ao quociente. De forma análoga ao método anterior, a principal desvantagem é a necessidade de se ter uma ROM grande. A outra desvantagem está no fato de serem necessárias duas multiplicações em cada iteração.

3.4. Divisão Num só Ciclo Usando uma ROM ou uma Matriz de Lógica

É possível realizar a divisão de números pequenos (menos de 18 bits) num só ciclo de relógio, utilizando uma tabela grande (em ROM), contendo o inverso do divisor, e um multiplicador paralelo. Este método foi utilizado em algumas aplicações [14] que não requeriam números muito grandes, mas que demandavam uma operação de divisão extremamente rápida. É possível reduzir o tamanho da tabela para processar os bits menos significativos do divisor,

utilizando interpolação linear. A principal desvantagem do método está no tamanho da ROM necessária. Por exemplo, para determinar o inverso de um divisor de 16 bits com sinal, são necessárias duas ROMs: uma de 8K x 16 bits e outra de 512 x 8 bits [14].

3.5. Divisão SRT

O método de divisão SRT foi desenvolvido independentemente por Sweeney, Robertson e Tocher e daí se origina o seu nome [7,9]. Este método pressupõe que o divisor e o resto parcial são normalizados para o intervalo $[1/2,1)$, antes de iniciar a divisão e gera o quociente com uma representação de digitos com sinal redundante. O método SRT focaliza a atenção na seleção dos dígitos do quociente e a iteração do resto parcial não requer restauração.

Utilizando base 2, os dígitos do quociente são selecionados do conjunto $\{-1,0,1\}$. O divisor pode ser somado ao resto parcial ou subtraído dele, ou então o resto parcial é simplesmente deslocado. Estas operações dependerão do resultado da comparação do resto parcial com o divisor.

Zurawski [14] descreve uma implementação deste método onde utiliza um somador do tipo *carry-save* e um somador do tipo *carry-propagate*. Embora esta implementação forneça somente um bit do quociente por iteração, o ciclo básico do sistema é mais rápido do que numa divisão convencional porque o somador *carry-save* é mais rápido do que um somador completo com propagação de *vai um*. Esta implementação do método é factível de ser utilizada numa unidade de divisão independente, que utilize um relógio separado e mais rápido do que o relógio do microprocessador.

Num artigo recente, Williams e Horowitz [15] descrevem a implementação, em CMOS-1.2um, de um chip de divisão capaz de computar a mantissa (com 54 bits) de uma operação de divisão em 45ns a 160ns. O algoritmo básico empregado é o SRT com base 2 (um bit do quociente por iteração). Os autores conseguem este desempenho expressivo utilizando um anel de cinco estágios com controle auto-temporizado, ou seja, não há relógios na unidade básica que computa as iterações. Isto demonstra que é possível acelerar a operação de divisão através de otimizações de projeto, mesmo utilizando um método que usa base inferior a 4.

Ainda usando base 2, uma outra forma de acelerar a divisão é utilizar diversas unidades em cascata, conforme ilustrado na figura 5. Neste caso, a aceleração é conseguida ao custo do hardware adicional necessário.

3.6. Divisão SRT usando Base 4 ou Base 8

Um artigo clássico de Atkins [16] explica o método de divisão SRT para bases maiores que 2 e estes são os mais prováveis de propiciar ganhos significativos em termos de área e velocidade, uma vez que vão sendo encontrados meios de gerar a lógica de geração do quociente cada vez mais compacta. Outra razão para aprimorar o método é explorar a concorrencia entre diversas partes do algoritmo (formação do resto parcial e seleção do quociente).

Em base 4, os múltiplos do divisor podem ser escolhidos dos conjuntos $\{-2,-1,0,1,2\}$ (redundância mínima) ou $\{-3,-2,-1,0,1,2,3\}$ (redundância máxima). O segundo conjunto é dito ser de redundância máxima, pois permite o número máximo de variações dos dígitos da base para formar o quociente.

Os métodos de divisão SRT produzem um número fixo de bits do quociente em cada iteração (em base 4 são obtidos dois bits do quociente em cada passo). O resto parcial de cada iteração é calculado subtraindo um múltiplo do divisor do produto da base pelo resto parcial anterior (Eq. 1a). O quociente é acumulado numa iteração paralela (Eq. 1b). Os métodos de divisão SRT usam dígitos de quociente *redundantes*, e portanto devem ser convertidos em *irredundantes*. Este método de divisão possui duas variantes básicas: usar resto parcial na forma redundante [17] ou irredundante [18]. Algumas implementações usam um registro para acumular os bits do quociente positivos e outro registro para acumular os bits do quociente negativos. Ao final da divisão, estes dois registros devem ser somados para obter o quociente final.

Esta descrição é formalizada pelas equações abaixo:

$$PR_i = r\,PR_{i-1} - q_i D \quad \text{para } i = 1,\ldots,m \quad \text{(Eq. 1a)}$$

onde

PR_i = resto parcial após i-ésima iteração
rPR_0 = dividendo
r = base
q_i = i-ésimo digito do quociente
D = Divisor.
m = número de iterações

$$Q_i = r\,Q_{i-1} + q_i \quad \text{para } i = 1,\ldots,m \quad \text{(Eq. 1b)}$$

onde

Q_i = quociente acumulado após i-ésima iteração
Q_0 = 0.

A maior parte das implementações do método utiliza base 4 com múltiplos do divisor selecionados do conjunto $\{-2D,-1D,0D,1D,2D\}$. A principal vantagem deste método é que estes múltiplos do divisor são fáceis de gerar, pois para gerar $2D$ basta deslocar D à esquerda. A principal desvantagem do método está no fato de requerer uma tabela maior do que aquela requerida usando o conjunto de redundancia máxima, para determinar os dígitos do quociente seguintes. Em [19], Taylor discute diversas consideraćões para a implementação de um circuito de divisão rápido. Bose, Taylor e Patterson [10] descrevem a implementação deste método utilizando: base 4, resto parcial na forma irredundante, digitos do quociente selecionados do conjunto $\{-2,-1,0,1,2\}$, e diversos estágios em pipeline. O diagrama em blocos da figura 6 ilustra o esquema básico utilizado em [10].

Em [20] é descrito o projeto de um circuito de divisão empregando múltiplos do divisor selecionados do conjunto $\{-3D,-2D,-D,0,D,2D,3D\}$, ou seja, redundância máxima. O múltiplo 3D é formado utilizando o somador/subtrator já existente na unidade de divisão, quando os operandos são carregados e portanto não consome tempo adicional para sua formação. A principal orientação no projeto deste circuito de divisão era ser utilizado na unidade aritmética inteira de um processador RISC orientado para geraçao de imagens em tempo real. Como consequência dessa orientação, o resto parcial foi mantido em forma irredundante para poder utilizar o somador completo da ALU. As figuras 6 e 7 ilustram o circuito necessário para o cálculo do resto parcial e do quociente, respectivamente.

4. Conclusões

Este artigo apresentou um levantamento de alguns métodos e implementações de operações de multiplicação e divisão e discutiu as principais vantagens e desvantagens de cada um em relação aos demais.

Não existe um método ou circuito ideal que proporcione o máximo desempenho e ao mesmo tempo a melhor relação custo benefício. Alguns dos artigos citados demonstram que mesmo métodos conceitualmente piores que outros, podem ter uma implementação mais simples e que proporcione tirar proveito dessa simplicidade obtendo assim um desempenho igual, ou até superior, a métodos mais sofisticados.

Os artigos também demonstram que é possível acelerar a operação de divisão. Tem ocorrido desenvolvimentos significativos em circuitos de multiplicação e divisão nos últimos anos, devido principalmente ao aumento da densidade dos circuitos integrados, pois isto tem permitido colocar mais unidades funcionais dentro de um mesmo chip.

5. Referencias

[1] Grimes, J., Kohn, L., and Bharadhwaj, R., "The Intel i860 64-bit Processor: A General-Purpose CPU with 3D Graphics Capabilities", IEEE Computer Graphics & Applic., pp. 85-94, 1989.

[2] AM29000 32-bit Streamlined Instruction Processor User Manual, AMD, 1988.

[3] Kane, G., "MIPS RISC Architecture", Prentice-Hall Publ., Englewood Cliffs, NJ, 1988.

[4] Magenheimer, B.B., Peters, L., et al, "Integer Multiplication and Division on the HP Precision Architecture", IEEE Transactions on Computers, Vol. C-37, No. 8, August 1988.

[5] Eichen, B., "NEC's uPD77230 Digital Signal Processor", IEEE Micro, pp. 60-69, December, 1986.

[6] Hennessy, J. L. and Patterson, D. A., "Computer

Architecture - a Quantitative Approach", Morgan Kaufmann Publishers Inc., San Mateo, California, 1990.

[7] Hwang, K., "Computer Arithmetic: Principles, Architecture and Design", New York, John Wiley and Sons Inc., 1978.

[8] MacSorley, O.L., "High-Speed Arithmetic in Binary Computers", Proc. IRE, vol. 99, pp.67-91, January 1991.

[9] Cavanagh, J. J. F., "Digital Computer Arithmetic", McGraw Hill, 1985.

[10] Bose,, B.K., Pei, L., Taylor, G.S. and Patterson, D.A., "Fast Multiply and Divide for a VLSI Floating-Point Unit", Proc. 9th IEEE Symposium on Computer Arithmetic, pp. 87-94, 1987.

[11] Henlin, D.A., Fertsch, M.T., Mazin, M. and Lewis, E.T., "A 16x16 Pipelined Multiplier Macrocell", IEEE Journal on Solid State Circuits, pp. 542-547, April, 1985.

[12] Anido, M. L., "Design and Implementation of a RISC Processor for Computer Image Generation Geometric Computations", Ph. D. thesis, Department of Electronics and Computer Science, University of Southampton, 1990.

[13] Lok, Y. F., "A Real_Time Computer Generated Imagery system for Flight Simulation", Ph.D. thesis, University of Sussex, U.K. 1983.

[14] Zurawski, J. H. and Gosling, J. B., "Design of High-speed digital divider units", IEEE Trans. on Computers, Vol. C-30, pp. 691-699, Sept. 1981.

[15] Williams, T.E. and Horowitz, M.A.,"A 160nS 54bit CMOS Division Implementation Using Self-Timing and Symmetrically Overlapped SRT Stages", IEEE Symposium on Computer Arithmetic, pp. 210-217, 1991.

[16] Atkins, D. E.," Higher Radix Division Using Estimates of the Divisor and Partial Remainders", IEEE Trans. on Computers, Vol. C-17, No. 10, pp. 925-934, October 1968.

[17] Taylor, G. S., "Compatible Hardware for Division and Square Root", Proc. of the 5th IEEE Symposium on Computer Arithmetic, pp. 127-134, May, 1981.

[18] Ercegovac, M.D. and Lang, T."A Division Algorithm with Prediction of Quotient Digits", Proc. 7th IEEE Symposium on Computer Arithmetic, pp. 51-56, 1985.

[19] Taylor, G. S., "Radix-16 SRT Dividers with Overlapped Quotient Selection Stages", Proc. of the 7th Symposium on Computer Arithmetic, pp. 64-71, 1985.

[20] Anido, M. L., "Improving the Division Instruction of Application-Specific RISCs", North-Holland Publ., EUROMICRO'91, Microprocessing and Microprogramming, 32, pp. 13-22, 1991.

Tabela 1 - Codificação Booth

Par de bits do Multiplicador		Bit do Meador à direita	Múltiplos do Multiplicando a adicionar
i + 1	i	i - 1	
0	0	0	0 x meando
0	0	1	+1 x meando
0	1	0	+1 x meando
0	1	1	+2 x meando
1	0	0	-2 x meando
1	0	1	-1 x meando
1	1	0	-1 x meando
1	1	1	0 x meando

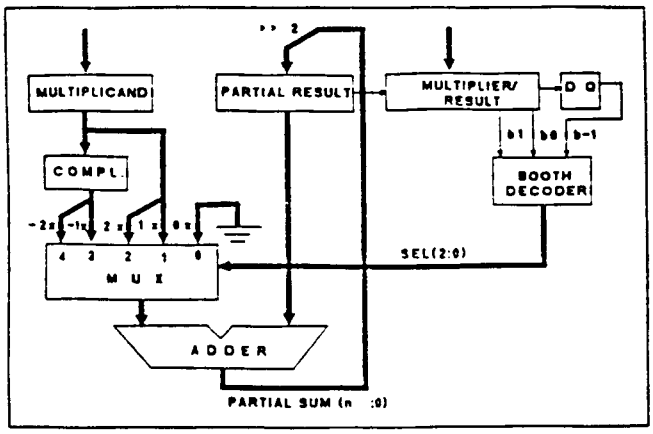

Fig. 1 - Multiplicação Sequencial Usando Recodificação com Pares de Bits

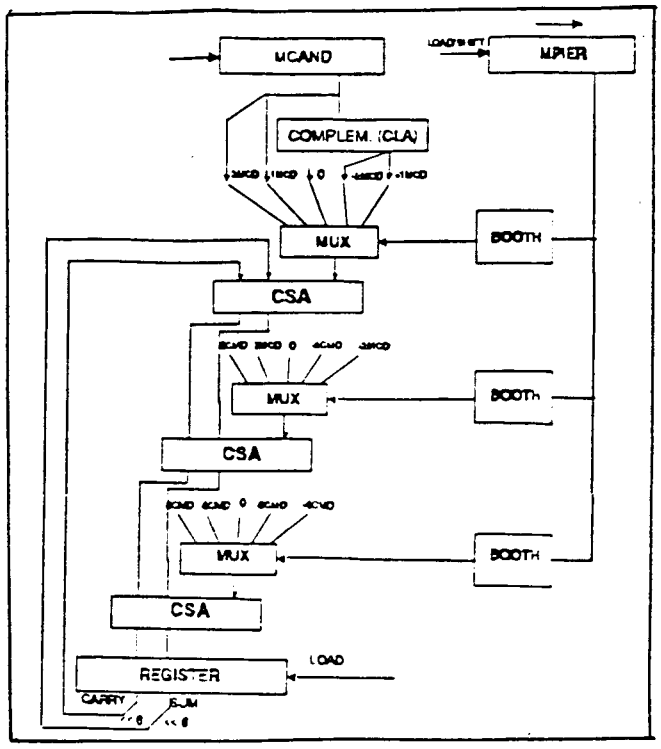

Fig. 2 - Multiplicação Sequencial Usando Diversos Somadores em Cascata

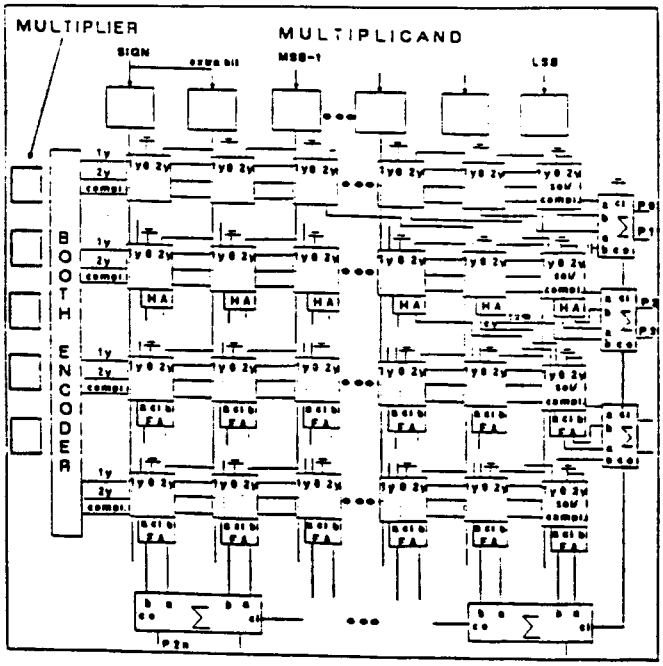

Fig. 3a - Multiplicador Paralelo Usando Método de Booth Modificado

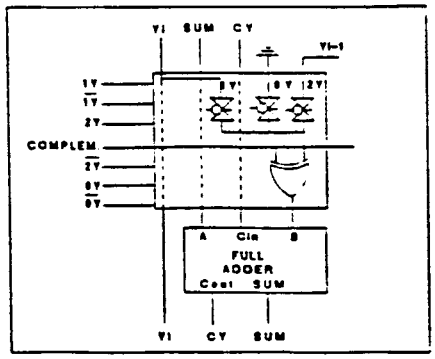

Fig. 3b - Célula Básica do Multiplicador Paralelo ao Lado

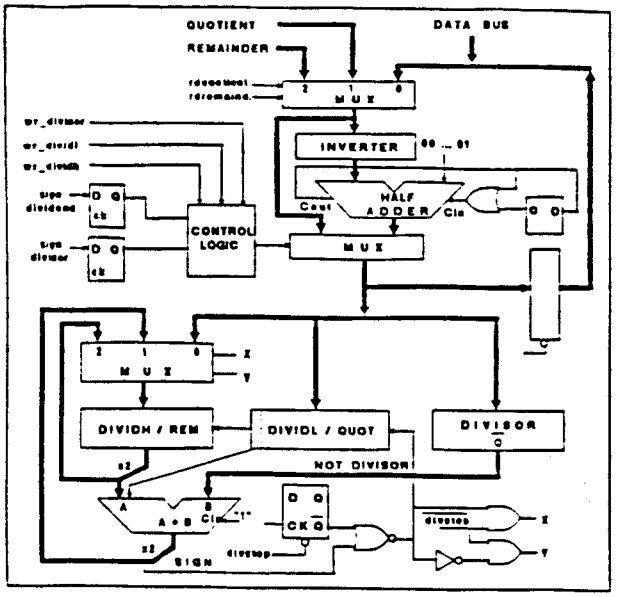

Fig. 4 - Divisão Base 2 com Conversão de Sinal *On-The-Fly*

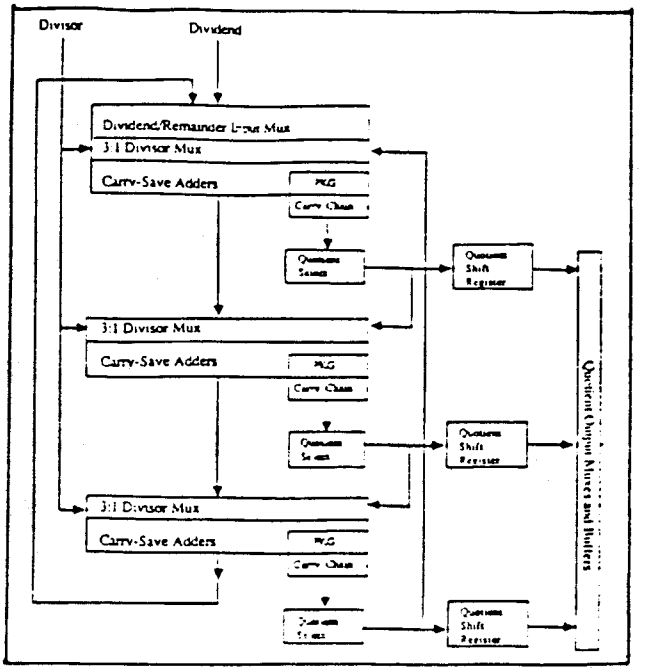

Fig. 5 - Divisão SRT base 2 com Diversas Unidades em Cascata

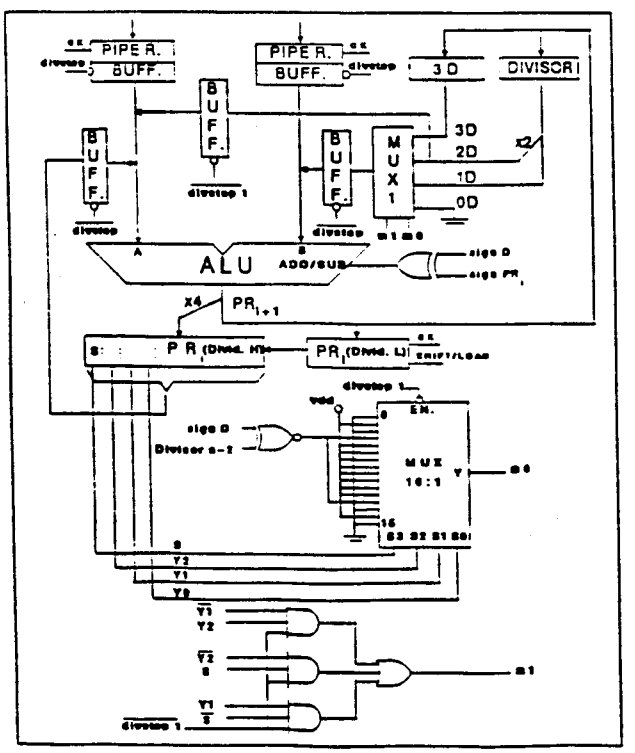

Fig. 6 - Divisão SRT Base 4 - Circuito Para Cálculo do Resto Parcial

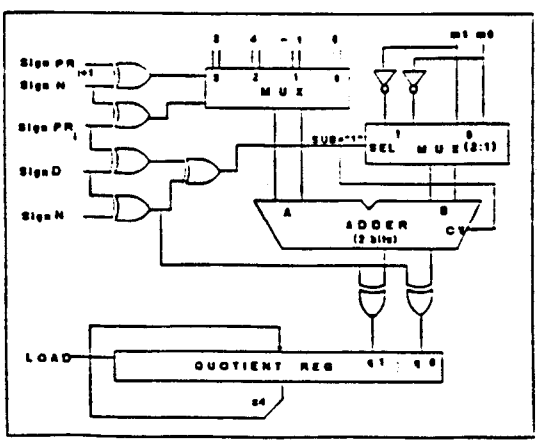

Fig. 7 - Divisão SRT Base 4 - Circuito Para Cálculo dos bits do Quociente