1 Circuitos Aritméticos

1.1 Somador de Ponto Flutuante

Uma representação usada é o formato IEEE 754 onde

sinal S	expoente E	mantissa M
1 bit	8 bits	23 bits

onde são usados 32 bits no total. A codificação tem o seguinte significado:

$$N = (-1)^s 2^{(e-127)} * 1.M$$

Por exemplo:

0	00000000	00000000000000000000000	$+0.0$
1	00000000	00000000000000000000000	-0.0
0	01111111	00000000000000000000000	$+1.0$
1	01111111	00000000000000000000000	-1.0
0	10000000	00000000000000000000000	$+2.0$
0	10000001	11100000000000000000000	$+7.5$
0	11111111	01001010100010000000000	NaN

Onde NaN =Not a Number, ou infinito. Para entender por exemplo como o número 2 foi gerado basta usar a fórmula:

$$\begin{aligned} N &= (-1)^s 2^{(e-127)} * 1.0 && (1)\\ &= (-1)^0 2^{(10000000_2-127)} * 1.0 && (2)\\ &= 2^{(128-127)} * 1.0 && (3)\\ &= 2^{(1)} * 1.0 && (4)\\ &= 2.0 && (5) \end{aligned}$$

Para a mantissa podemos usar o somatório de potência negativas, já que depois da vírgula (ou ponto na notação americana, de onde vem o nome ponto flutuante), os dígitos tem peso negativo $2^{-1}, 2^{-2}, \ldots, 2^{-23}$. Ou seja $M = \sum_{i=-1}^{-23} D_i * 2^i$. Para calcular, podemos usar frações pois $2^{-1}, 2^{-2}, \ldots, 2^{-23} = 1/2, 1/4, \ldots, 1/2^{23}$. Considere o número 7.5. PAra calculá-lo temos:

$$\begin{aligned} N &= (-1)^s 2^{(e-127)} * 1.M && (6)\\ &= (-1)^0 2^{(10000001_2-127)} * (1 + D_1 * 1/2 + \ldots + D_{23} * 1/2^{23}) && (7)\\ &= 2^{(129-127)} * (1 + 1/2 + 1/4 + 1/8) && (8)\\ &= 2^{(2)} * (\frac{8+4+2+1}{8}) && (9)\\ &= 4 * (\frac{15}{8}) && (10)\\ &= \frac{15}{2} && (11)\\ &= 7.5 && (12) \end{aligned}$$

Abaixo ilustramos o algoritmo básico da soma (extraído da página http://venus.rdc.puc-rio.br/rmano/rd6aritr.html, Prof Rui Mano - PUC-RIO)

1. Verifica-se se uma das mantissas a operar é zero;

   (a) caso afirmativo: se for uma soma e uma das parcelas for zero —¿ o resultado é igual à outra parcela

   (b) Se não houver zeros: - reduzir ao mesmo expoente (o maior, suponha e_1), ajustar as mantissas (suponha $m_2 -> m_3$); - somar as mantissas; - normalizar o resultado, ou seja, $e_1 * m_1 + e_2 * m_2 = e_1 * m_1 + e_1 * m_3 = e_1 * (m_1 + m_2)$.

Por exemplo, $x_1 = 2^2 * (1 + 2^{-1})$ e $x_2 = 2^1 * (1 + 2^{-3})$. Como o expoente $e_1$de x_1é maior, devemos ajustar a mantissa de x_2, ou seja $x_2 = 2^1 * (2^0 + 2^{-3}) = 2^2 * (2^{-1} + 2^{-4})$, o que em binário, significa deslocar a mantissa de x_2, na representação

$x_1 = 10\ldots01 \quad 1000\ldots0$

$x_2 = 10\ldots00 \quad 0010\ldots0$

ao normalizar pelo expoente de x_1 temos

$x_1 = 10\ldots01 \quad 1000\ldots0$

$x_2 = 10\ldots01 \quad 10010\ldots0$

ou seja, e_2 foi incrementado de 1 e m_2 foi deslocado para direita de 1 bit. Vericamos que o terceiro bit que era 1 virou o quarto bit após o deslocamento, masis porque apareceu um 1 no primeiro bit que era 0 ? Para ficar mais claro, é interessante representar o 1 do 1.M da mantissa, o termo 2^0 que é implicíto e não aparece na codificação, mas deve ser levado em conta.

número	*expoente*	1	M
x_1	$10\ldots01$	1	$100000\ldots00$
x_2	$10\ldots00$	1	$001000\ldots00$
$x_2(ajustado)$	$10\ldots01$	0	$100010\ldots00$

Como se pode observar, o 1 implícito (2^0) que foi deslocado para dentro da mantissa e virou 2^{-1}. E como o número pode existir que o 1 implitico foi substituído por 0 ? Para resolver este problema iremos normalizar o número após a operação.

Podemos refinar o algoritmo, supondo 3 casos

- Expoentes iguais $e_1 = e_2$
- $e_1 > e_2$
- $e_1 < e_2$

O terceiro caso é equivalente ao segundo se trocarmos x_1e x_2. Suponha que $x_1$seja o número com maior expoente. Assim, temos que tratar 2 casos, expoentes iguais ou x_1 é maior.

1.1.1 Expoentes iguais

Se $e_1 = e_2$, teremos

	expoente	1	$Mantissa$
x_1	e_1	1	M_1
x_2	e_1	1	M_2
$x_1 + x_2$	e_1	$1C_M$	$M_1 + M_2$
	$ajusta\,expoente$		
	$e_1 + 1$	1	$C_M \cdot M1 + M_2$

Ao somar, teremos um vai-um do termo constante 1, e o termo C_M é o vai-um da soma das mantissas, que pode ser 0 ou 1, depende dos valores. Como a constante tem que ser 1, temos que reajustar o expoente, decrementando e_1 e deslocando a mantissa, onde o valor $M_1 + M_2$ é deslocado para direita e o vai-um C_m passa a ser o bit mais significativo.

Para simplificar as operações iremos considerar uma nova representação com objetivos didáticos:

$$N = 2^{(e-3)} * 1.M$$

Onde serão usados 3 bits de expoente e 4 de mantissa. Suponha $x_1 = 3$ e $x_2 = 3.25$.

	expoente	*constante*	*mantissa*
x_1	100	1	1000
x_2	100	1	1010
$s = x_1 + x_2$	100	11	0010

Ou seja, $s = 2^{4-3} * (1 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3})$ que é igual à $2 * (2 + 1 + 0.125) = 6.25$. Porém o resultado está violando a condição de ter apenas um único bit 1 implicíto. Ao normalizar para $N = 2^{5-3} * (1 + M)$, temos a soma $x_1 + x_2 = 2^2 * (1 + 1/2 + 1/16) = 6.25$. Portanto a soma correta deve fazer o ajuste:

	expoente	*constante*	*mantissa*
x_1	100	1	1000
x_2	100	1	1010
$s = x_1 + x_2$	100	11	0010
$ajustes =$	101	1	1001

Resumindo, para somar com expoentes iguais, basta somar as mantissas, deslocar o resultado para direita e incrementar o expoente em 1.

1.1.2 Expoente Maior

No caso de expoente maior, temos 2 casos:

- diferença dos expoentes é maior que o número de bits da mantissa
- diferença dos expoentes é menor ou igual ao número de bits da mantissa.

Considere nosso exemplo, com 3 bits de expoente e 4 de mantissa. Se $e_1 = 7$ e $e_2 = 2$, ou seja, $e_1 \rightarrow 2^{7-3} = 2^4, e_2 \rightarrow 2^{2-3} = 2^{-1}$, teremos

	expoente	*constante*	*mantissa*	*continua...*
x_2	010	1	$b_1b_2b_3b_4$	
x_1	111	1	$a_1a_2a_3a_4$	
$ajuste\, x_2$			$5\, vezes \rightarrow$	
x_2	111	0	0000	$1a_1a_2a_3a_4$

Como a mantissa de x_2, se consideramos apenas 4 bits, fica igual a zero, já que os bits foram deslocados para "fora" da mantissa, não é necessário somar, o resultado será x_1. Em decimal, seria analógo a soma$10^4 * 23 + 10^{-1} * 34 = 10^4 * 23 + 10^4 * 0.0034 = 10^4 * 23,0034$, sem consideramos somente 2 dígitos de mantissa, o número continua o mesmo $x_1 + x_2 = x_1 = 10^4 * 23$.

No outro caso, que a diferença dos expoente é menor que o número de bits da mantissa, algo de $x_2$será somado a x_1, se isso gerar um vai-um, o expoente e_1 do resultado deve ser ajustado (incrementado), e a mantissa também (desloca 1 bit para direita), senão tiver vai-um, nada o resultado e a soma de $m_1 + m_2$.

2 Multiplicação

Para executar a multiplicação, o tratamento é mais simples, segundo alguns autores, pois não é necessário alinhar os bits da mantissa. Seja $x_1 = 2^{e_1-3} \cdot (1 + M_1)$ e $x_2 = 2^{e_2-3} \cdot (1 + M_2)$. O produto p será $p = x_1 \cdot x_2 = 2^{e_1-3+e_2-3} \cdot (1.M_1 \cdot 1.M_2)$. Como o número deve ser normalizado e

o expoente deve seguir o padrão 2^{e_p-3}, o novo expoente $e_p = e_1 + e_2 - 3$ e as mantissa devem ser multiplicadas. Depois o resultado deve ser normalizado e ajustado.

Vamos fazer um exemplo supondo $x_1 = 3$ e $x_2 = 3.25$.

	expoente	*constante*	*mantissa*
x_1	100	1	1000
x_2	100	1	1010
m	110	1	0011

Logo $m = 2^{6-3} \cdot (1 + 2^{-3} + 2^{-4}) = 8 \cdot (\frac{16+2+1}{16}) = 9,5$. Porém, $3 \cdot 3.25 = 9.75$! Vamos fazer a multiplicação passo a passo, para entender o resultado.

Multiplicando as mantissas, considerando o 1 implícito.

					1	1	0	0	0
					1	1	0	1	0
–	–	–	–	–	–	–	–	–	–
					0	0	0	0	0
				1	1	0	0	0	
			0	0	0	0	0		
		1	1	0	0	0			
	1	1	0	0	0	0			
–	–	–	–	–	–	–	–	–	–
1	0	0	1	1	1	0	0	0	0

Qual é o peso do bit menos significativo do resultado ? Basta lembrar que o bit menos significativo do multiplicador e multiplicando é igual a 2^{-4}, portanto, no resultado teremos $2^{-4} \cdot 2^{-4} = 2^{-8}$. Alinhando o número temos

	expoente	*constante*	*mantissa*
x_1	100	1	1000
x_2	100	1	1010
m	?	10	01110000

Agora iremos calcular o expoente, lembre-se que $e_p = e_1 + e_2 - 3$, portanto teremos $e_p = 4 + 4 - 3 = 5$, ou seja, 2^5. Mas o valor do expoente será $2^{e_p-3} = 2^{5-3} = 2^2$.

Vamos conferir o resultado, $2^2 \cdot (2^2 + 2^{-2} + 2^{-3} + 2^{-4}) = 4 \cdot (\frac{32+4+2+1}{16}) = 9,75$.

Mas temos que normalizar o número, portanto, teremos uma perda de precisão e um ajuste no expoente (ao alinhar o 1 implícito).

	expoente	*constante*	*mantissa*	*perda*
x_1	100	1	1000	
x_2	100	1	1010	
m	101	10	0111	0000
Normalizado	110	1	0011	10000

Vamos conferir o resultado, $2^6 - 3 \cdot (1 + 2^{-3} + 2^{-4}) = 8 \cdot (\frac{16+2+1}{16}) = 9,5$.

3 Projeto por etapas

Nesta seção será mostrado um possibilidade de projeto por etapas. O objetivo é minimizar o custo, compartilhando recursos. A representação IEEE 784 Para sincronizar as etapas, uma máquina de estados será empregada. Primeiro, o problema será decomposto em etapas. Para o expoente pode ser necessário comparar, subtrair e deslocar de 1 dígito. Para a mantissa, pode ser necessário somar, deslocar 0 à 23 bits. Para simplificar, considere a operação A+B onde A é maior que B. O algoritmo seria:

1. Subtrai os expoentes (soma/subtrator)

2. ajusta a Mantissa pela diferença dos expoentes

3. soma mantissa

4. ajusta mantissa (deslocar ou não)

5. ajusta expoente (soma 1 ou não)

Pode-se notar que a operação de soma/subtração ocorre duas vezes (expoente e mantissa), a operação de ajuste 3 vezes. Pode-se propor um circuito que usa um deslocador e um soma-subtrator com o controle de uma máquina de estados. O algoritmo abaixo mostra um refinamento onde E representa expoente de A,B e C, respectivamente, assim como M representa a mantissa. O carry é o vai-um do somador. Note que Mc e Ec são usados como temporários.

```
Ec = Ea - Eb;
Mc = desloca(Mb,Ec);
Mc = Ma + Mc;
Mc = desloca(Mc,Carry);
Ea = Ea + Carry;
```