Unidade 5: Sistemas de Representação

Números de Ponto Flutuante IEEE 754/2008 e Caracteres ASCII

Prof. Daniel Caetano

Objetivo: Compreender a representação numérica em ponto flutuante.

INTRODUÇÃO

Como foi visto em aulas anteriores, é possível armazernar na memória um valor que pode ser considerado fracionário usando a representação de **ponto fixo**, isto é, considerando a vírgula fixa em uma determinada posição. Assim, o número 110,101b, considerando ponto fixo com 4 dígitos depois da vírgula, este número poderia ser armazenado assim:

Parte Inteira				Parte Fracionária			
0	1	1	0	1	0	1	0

Essa representação é conveniente, porque permite representar e realizar operações com números reais; entretanto, ela tem o inconveniente de limitar a representação; o menor valor fracionário representável por este caso acima é $1/2^4 = 1/16 = 0,0625$. Não há como representar, por exemplo, o número 0,01. Isso é indesejável porque, na prática, alguns números que trabalhamos são de ordens de grandeza muito baixas, isto é, da ordem de 10^{-50}, que precisariam de um número binário muito grande para ser representado.

A solução para isso está em usar números de **ponto flutuante**.

1. REPRESENTAÇÃO EM PONTO FLUTUANTE

Quando os engenheiros se depararam com o problema de representação acima, tiveram que parar para pensar um pouco. Não demorou muito e a sugestão para resolver este problema veio, tendo como base a representação numérica científica - muito usada por engenheiros e físicos. Na base decimal, esta representação é a seguinte:

Tradicional	**Científica**	**Científica Reduzida**
125	$1,25 * 10^2$	1,25E2

A idéia é representar um número com apenas um dígito inteiro e ajustar a posição correta da vírgula com uma potência de 10; desta forma, um número com qualquer número de casas decimais fica representado por três números: a parte inteira (denominada **característica**), a parte fracionária (denominada **mantissa**) e o expoente da potência de 10 (denominado **expoente**). Obseve:

Tradicional	Científica	Característica	Mantissa	Expoente
125	1,25E2	1	25	2

Um número que seria muito difícil de escrever na forma decimal, por ser muito pequeno, pode ser facilmente escrito na notação científica e, portanto, representado como um número de ponto flutuante.

Científica	Característica	Mantissa	Expoente
1,25E-56	1	25	-56

O nome "ponto flutuante" vem da existência do expoente, que indica o número de dígitos que a vírgula deve ser deslocada; um expoente negativo significa que a vírgula deve ser deslocada à esquerda e um exponte positivo significa que a vírgula deve ser deslocada a direita.

2. PONTO FLUTUANTE COM NÚMEROS BINÁRIOS

Da mesma forma que representamos números decimais na forma de ponto flutuante, podemos representar números binários. Por exemplo:

Tradicional	Científica	Característica	Mantissa	Expoente
100b	$1,00b * 2^{2}$	1b	00b	2
101b	$1,01b * 2^{2}$	1b	01b	2
11,101b	$1,1101b * 2^{1}$	1b	1101b	1
0,1001b	$1,001b * 2^{-1}$	1b	001b	-1

Observe que a lógica é a mesma... e, inclusive, existe uma regra curiosa: a característica vale sempre 1! **Como a característica é sempre 1, ela não precisa ser representada!**

Assim, um número binário de ponto flutuante é representado apenas por sua mantissa e seu expoente, admitindo-se que sua característica é sempre o bit 1. No caso de números com sinal, jogamos o sinal na mantissa!

Tradicional	Científica	Mantissa	Expoente
100b	$1,00b * 2^{2}$	00b	2
101b	$1,01b * 2^{2}$	01b	2
11,101b	$1,1101b * 2^{1}$	1101b	1
0,1001b	$1,001b * 2^{-1}$	001b	-1

Mas, como representar isso na memória?

3. PONTO FLUTUANTE BINÁRIO NA MEMÓRIA

Anteriormente vimos uma forma simplificada de representar os números negativos, em que reservávamos um bit para indicar o sinal (0 = positivo e 1 = negativo), e usáamos os outros para representar o valor numérico, normalmente.

Sinal	Número						
0	1	1	0	1	0	1	0

Bem, a idéia é a mesma, mas agora teremos que reservar bits para:

a) Sinal do número
b) Sinal do expoente
c) Expoente
d) Mantissa

Para entender a idéia, vamos considerar primeiramente números de 8 bits. Em teoria, podemos reservar um bit para o sinal do número, um bit para o sinal do expoente, dois bits para o expoente e quatro bits para a mantissa:

Sinal	Sinal do Expoente	Expoente		Mantissa			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Vejamos como o número 2,25 fica representado em ponto flutuante:

a) Primeiramente vamos convertê-lo em binário.
   Parte inteira: 2 = 10b
   Parte Fracionária: 0,25 = 0,01b
   Assim: 2,25 = 10,01b
b) Agora vamos reescrevê-lo em notação científica:
   $10,01b = 1,001b * 2^1$
c) Agora dividimos as partes:
   Sinal: 0 (positivo)
   Característica: 1b
   Mantissa: 001b
   Sinal Expoente: 0 (positivo)
   Expoente: 1
d) Agora representamos na memória

Sinal	Sinal do Expoente	Expoente		Mantissa			
0	0	0	1	0	0	1	0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Observe que o último dígito à direita da mantissa - um zero - foi adicionado para completar a representação. **Nunca** remova zeros à esquerda na mantissa: **eles são importantes**.

4. REPRESENTAÇÃO IEEE 754/2008

A representação anterior com 8 bits é adequada didaticamente, mas como é possível observar, os números que podem ser representados são muito limitados. Em especial, o número de bits da mantissa limita o número de dígitos que podem ser representados (dígitos significativos) e o número de bits do expoente limita a variação de magnitude do número.

Assim, na prática, são usadas representações que usam vários bytes de memória para indicar um único número; como a organização dos bits que representam sinal, expoente e mantissa é arbitrária, é preciso adotar uma padronização.

O IEEE é um órgão composto por engenheiros que define uma série de normas de engenharia. Uma destas normas, a 754 de 2008, define a representação de números mais usada em computadores modernos, definindo números de ponto flutuante de **precisão simples**, com 32 bits, e números de ponto flutuante de **precisão dupla**, com 64 bits.

A representação para precisão simples (32 bits) é a seguinte:

Sinal	Sinal do Expoente	Expoente	Mantissa
Bit 31	Bit 30	Bit 29 ~ Bit 23	Bit 22 ~ Bit 0

São, portanto, 1 bit para sinal, 8 bits para o expoente (incluindo o sinal) e 23 bits para a mantissa.

A representação para precisão dupla (64 bits), por sua vez, é a seguinte:

Sinal	Sinal do Expoente	Expoente	Mantissa
Bit 63	Bit 62	Bit 61 ~ Bit 52	Bit 51 ~ Bit 0

São, portanto, 1 bit para sinal, 11 bits para o expoente (incluindo o sinal) e 52 bits para a mantissa.

O detalhe nestas duas representações do IEEE é que o sinal do número (bit 31 e 63, respectivamente para simples e dupla precisão) é o tradicional, isto é, bit 0 é positivo e bit 1 é negativo; o **sinal do expoente**, entretanto, é **invertido**, isto é, bit 1 é positivo e bit 0 é

negativo. A razão foge ao escopo do curso, mas tem a ver com facilitar as operações matemáticas com números de ponto flutuante.

5. REPRESENTAÇÃO DO ZERO

O aluno, neste instante, pode estar se perguntando: se nas representações de ponto flututante binário o valor da característica é considerado fixo em 1 - e por essa razão nem é indicado -, como representar o valor zero?

A primeira alternativa seria a "gambiarra", isto é, representar uma mantissa vazia com o maior expoente negativo possível. Isso seria adequado na suposição de que 0,000000000000000001 = 0,0, por exemplo. Como essa é uma aproximação grosseira, o IEEE definiu uma maneira diferente de representar o número zero.

Sempre que o valor do EXPOENTE for igual a -0 (isto é, todos os bits zero), o valor da característica é considerado ZERO. Assim, se todos os bits do expoente forem zero, assim como todos os bits da mantissa forem zero, o valor representado será considerado exatamente igual a zero (independente do bit de sinal do número). Exemplo (simulando IEEE com 8 bits):

Sinal	Sinal do Expoente	Expoente		Mantissa			
0	0	0	0	0	0	0	0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Se o número representado no expoente for -0 e a mantissa for um outro valor, será considerado um valor de característica igual a 0. Por exemplo:

Sinal	Sinal do Expoente	Expoente		Mantissa			
0	0	0	0	1	1	0	1
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Esse valor representa o número binário 0,1101b. Repare que isso é totalmente diferente disso (expoente +0):

Sinal	Sinal do Expoente	Expoente		Mantissa			
0	1	0	0	1	1	0	1
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Cujo valor é o número binário 1,1101b

Assim, quando o expoente é -0, consideradmos que o valor está em notação científica **não-normalizada**, isto é, o número representado pela **característica+mantissa** gera um valor entre 0,0 e 1,0. Quando o expoente é diferente de -0, considera-se uma notação

científica **normalizada**, isto é, o número representado pela **característica+mantissa** gera um valor entre 1,0 e 2,0.

O IEEE também criou uma representação para o valor "infinito": quando todos os bits do expoente valerem 1 e todos os bits da mantissa valerem 0.

Assim, +infinito pode ser representado assim (simulando em 8 bits):

Sinal	**Sinal do Expoente**	**Expoente**		**Mantissa**			
0	1	1	1	0	0	0	0
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**

E o -inifinito, por sua vez, pode ser representado assim (simulando em 8 bits):

Sinal	**Sinal do Expoente**	**Expoente**		**Mantissa**			
1	1	1	1	0	0	0	0
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**

Observe que os valores da mantissa **devem** ser zero. Se todos os bits do expoente forem 1 e pelo menos um dos bits da mantissa for diferente de zero, o valor será considerado não-numérico (+NaN e -NaN: *Not A Number*).

6. OPERAÇÕES EM PONTO FLUTUANTE (OPCIONAL)

As operações com números em ponto flutuante são realizadas mediante muitas conversões e deslocamentos, já que para muitas delas asmbos os números precisam ter o mesmo expoente. Em algumas operações, a soma ocorre entre os expoentes e, em outras, ocorre apenas com a mantissa, após ajustes.

Visto que as operações não podem ser feitas de maneira direta, exigem um processamento diferenciado. Por essa razão, **nem todos os processadores são capazes de realizar aritmética de ponto flutuante**. Aqueles que as fazem, além da Unidade de Controle (UC) e Unidade Lógica Aritmética (ULA) possuem também uma UPF (Unidade de Ponto Flutuante) para realizar esses cálculos mais rapidamente.

A realização de cálculos de ponto flutuante em equipamentos que não possuem uma UPF exige que o cálculo seja feito "por software", isto é, exigem que um pequeno programa realize essas operações. Isso torna o processamento muito mais lento, sendo uma das razões pelas quais o uso de ponto flutuante é evitado em uma série de razões.

Adicionalmente, como o número de dígitos é limitado ao número de bits e, para realizar operações é frequente que ambos os números sendo operados precisem ser convertidos para o mesmo expoente, pode haver perda significativa de bits durante as

operações. Por exemplo, considere a soma de 2,25 com 0,5625, usando a representação simulada IEEE em 8 bits:

$2{,}25 = 10{,}01b = 1{,}001b * 2^1$

Sinal	Sinal do Expoente	Expoente		Mantissa			
0	1	0	1	0	0	1	0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

$0{,}5625 = 0{,}1001b = 1{,}001b * 2^{-1}$

Sinal	Sinal do Expoente	Expoente		Mantissa			
0	0	0	1	0	0	1	0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Bem, o resultado da soma deveria ser 2,25+0,5625 = 2,8125, certo? Vejamos o que ocorre!

Para realizar a soma das mantissas, é preciso que ambos os números estejam com o mesmo expoente. Sempre iremos converter o de menor expoente para se equiparar ao de maior expoente. Assim:

$1{,}001b * 2^{-1} = 0{,}1001b * 2^0 = 0{,}01001b * 2^1$

Se simplesmente somássemos os dois binários, teríamos:

$1{,}00100b * 2^1$
$\underline{0{,}01001b * 2^1}$
$1{,}01101b * 2^1 \quad = \quad 10{,}1101b \quad = \quad 2{,}8125$

Ora, antes, vamos representar este número 0,01001b na notação IEEE simulada com 8 bits (ignoremos a característica, já que estamos trabalhando com a soma das mantissas, fixando a característica em 1):

Sinal	Sinal do Expoente	Expoente		Mantissa				
0	1	0	1	0	1	0	0	1
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

O último bit se perdeu! Assim, a soma efetivamente realizada será:

$1{,}0010b * 2^1$
$\underline{0{,}0100b * 2^1}$
$1{,}0110b * 2^1 \quad = \quad 10{,}110b \quad = \quad 2{,}75$

Repare que 2,75 é **diferente** de 2,8125. Essa diferença, chamada erro, se reduz bastante à medida que se trabalha com números de maior precisão (32, 64, 128 bits... e assim por diante). Entretanto, esse erro sempre existe, consistindo em uma **limitação** da representação em ponto flutuante. Adicionalmente, diversos números que possuem representação finita na base decimal (como 0,1) se tornam "dízimas periódicas" na base dois (0,1 = 0,00011001100110011...b). Isso faz com que executar 1000 somas do número 0,1, por exemplo, não levem ao valor 100, consistindo uma **limitação** da representação binária em ponto flutuante para realizar operações com números decimais!

7. REPRESENTAÇÃO DE CARACTERES

Até o momento vimos como armazenar números de diferentes tipos na memória. Mas como armazenar letras? Bem, este foi um problema que surgiu nos primórdios da computação e, por esta razão, existe uma solução padrão, que é a chamada Tabela ASCII (ASCII significa *American Standard for Computer Information Interchange*). A tabela ASCII relaciona cada valor numérico de um byte a cada um dos códigos visuais usados por nós na atividade da escrita. A tabela de conversão é apresentada na página seguinte (fonte: Wikipédia).

Observe, porém, que nem todos os caracteres são definidos por essa tabela: em especial, os caracteres acentuados estão faltando. Mas não são apenas estes: também não estão presentes os caracteres japoneses, chineses, russos... dentre tantos outros.

Por essa razão, atualmente existem diversas outras "tabelas de código de caracteres" ou "páginas de código de caracteres" (do inglês *codepage*), que extendem a tabela abaixo indicando os símbolos faltantes aos códigos livres (não especificados pela tabela ASCII). Entretanto, com a grande troca de arquivos entre pessoas de países diferentes, isso começou a causar alguma confusão.

Foi assim que surgiram então os códigos Unicode, que são versões alternativas e universais à tabela ASCII. O padrão UTF (Unicode Transformation Format) define várias tabelas, sendo as mais conhecidas e usadas as tabelas UTF-8 e UTF16. A tabela UTF-8 define 256 caracteres, como a tabela ASCII, mas com um padrão que tenta alocar a grande maioria dos símbolos usados pela maioria das línguas. Já o UTF-16 define 65.536 caracteres, englobando a grande maioria dos caracteres de todas as línguas. Existe ainda o padrão UTF-32, com capacidade para definir até 4 bilhões de caracteres, mas que é muito pouco usado.

Binário	Decimal	Hexa	Glifo
0010 0000	32	20	
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Binário	Decimal	Hexa	Glifo
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

Binário	Decimal	Hexa	Glifo
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~