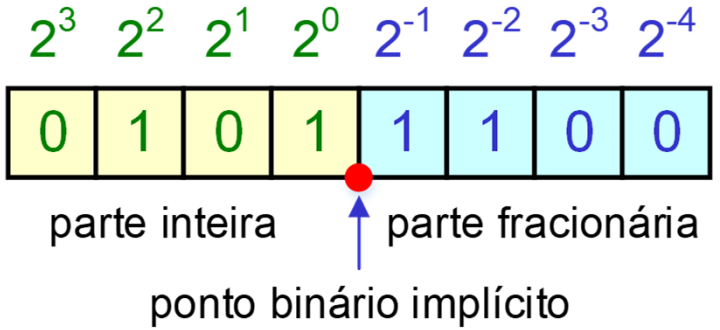
Aula 7

- Representação de números em vírgula flutuante
- A norma IEEE 754
  - Operações aritméticas em vírgula flutuante
  - Precisão simples e precisão dupla
  - Casos particulares; representação desnormalizada
  - Técnicas de arredondamento

Bernardo Cunha, José Luís Azevedo

Representação de quantidades fracionárias

- A codificação de quantidades numéricas com que trabalhámos até agora esteve sempre associada à representação de números inteiros
- A representação posicional de inteiros pode também ser usada para representar números racionais considerando-se potências negativas da base
- Por exemplo a representação da quantidade 5.75 em base 2 com 4 bits para a parte inteira e 4 bits para a parte fracionária poderia ser:



- Esta representação designa-se por "**representação em vírgula fixa**"

Representação de quantidades fracionárias

- A representação de quantidades fracionárias em vírgula fixa coloca de imediato a questão da divisão do espaço de armazenamento para as partes inteira e fracionária
- Quantos bits devem ser reservados para a **parte inteira** e quantos para a **parte fracionária**, sabendo nós que o espaço de armazenamento é limitado?
- O número de bits da parte inteira determina a **gama de valores representáveis** (2^4, no exemplo anterior)
- O número de bits da parte fracionária, determina a **precisão** da representação (passos de $2^{-4} = 0.0625$, no exemplo anterior)

Representação de números em Vírgula Flutuante

- **Exemplo**: **-23.45129** (vírgula fixa). A mesma quantidade pode também ser representada recorrendo à notação científica:

-2.345129×10^{1}	$-(2\times10^{0}+3\times10^{-1}+4\times10^{-2}+5\times10^{-3}+\ldots+9\times10^{-6}) \times 10^{1}$
-0.2345129×10^{2}	$-(0\times10^{0}+2\times10^{-1}+3\times10^{-2}+4\times10^{-3}+\ldots+9\times10^{-7}) \times 10^{2}$

- São representações do mesmo valor em que a posição da vírgula tem de ser ponderada, na interpretação numérica da quantidade, pelo valor do expoente de base 10
- Esta técnica, em que a vírgula pode ser deslocada sem alterar o valor representado, designa-se também por **representação em vírgula flutuante (VF)**
- A representação em VF tem a vantagem de não desperdiçar espaço de armazenamento com os zeros à esquerda da quantidade representada
- No primeiro exemplo, o número de dígitos diferentes de zero à esquerda da vírgula é igual a um: diz-se que a **representação está normalizada**

Representação de números em Vírgula Flutuante

- A representação de quantidades em vírgula flutuante, em sistemas computacionais digitais, faz-se recorrendo à estratégia descrita no slide anterior, mas usando agora a base dois:

$$N = (+/-)\ 1.f \times 2^{Exp}$$

(representação em binário de uma quantidade real, no formato de **vírgula flutuante normalizada**)

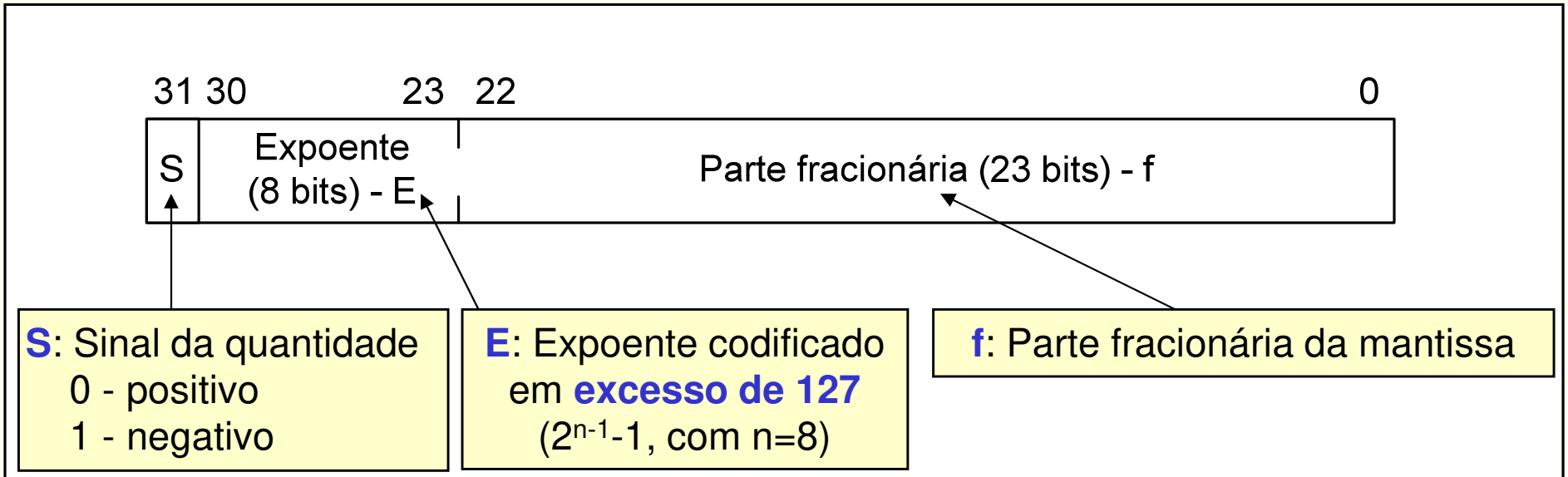
- Em que:

  f – parte **fracionária** representada por ***n*** bits
  1.f – **mantissa** (também designada por significando)
  Exp – **expoente** da potência de base 2 representado por ***m*** bits

Representação de números em Vírgula Flutuante

- O problema da divisão do espaço de armazenamento coloca-se também neste caso, mas agora na determinação do **número de bits** ocupados pela **parte fracionária** e pelo **expoente**
- Essa divisão é um **compromisso** entre **gama de representação** e **precisão**:
  - Aumento do número de bits da parte fracionária $\Rightarrow$ maior precisão na representação
  - Aumento do número de bits do expoente $\Rightarrow$ maior gama de representação
- Um bom *design* implica compromissos adequados!

Norma IEEE 754 (precisão simples)



- A representação é **normalizada**: o bit à esquerda do ponto binário é **sempre 1**. Como é sempre 1, esse bit não é explicitamente representado (***hidden bit***)
- A parte fracionária (23 bits) pode então tomar valores compreendidos entre:

  .00000000000000000000000 e .11111111111111111111111
- E os limites de representação da mantissa (**1.f**) são:

  1.00000000000000000000000 e 1.11111111111111111111111

Norma IEEE 754 (precisão simples)

- O expoente é codificado em **excesso de 127** ($2^{n-1}-1$, n=8 bits). Ou seja, é somado ao expoente verdadeiro (**Exp**) o valor 127 para obter o código de representação
  (i.e. ***E = Exp + 127***, em que E é o expoente codificado)

$$N = (-1)^S \; 1.f \times 2^{Exp} = (-1)^S \; 1.f \times 2^{E-127}$$

- O código 127 representa, assim, o expoente zero; códigos maiores do que 127 representam expoentes positivos e códigos menores que 127 representam expoentes negativos
- **Os códigos 0 e 255 são reservados**. O expoente pode, desta forma, tomar valores entre **-126** e **+127** [códigos 1 a 254].

Norma IEEE 754 – Exemplo (precisão simples)

Exemplo: Qual o valor, em decimal, representado em **0x41580000**?

0 10000010 10110000000000000000000

Sinal = 0 (quantidade positiva)

Expoente = 130 – *offset* = 130 – 127 = 3 ⇔ (Exp = E – *offset*)

Mantissa = (1 + parte fracionária) = **1** + **.1011** = **1.1011**

A quantidade representada (R) será então: $+1.1011 \times 2^3$

$R = +1.1011 \times 2^3 = (1 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 1\times2^{-4}) \times 2^3$

$= +1.6875 \times 8 = +13.5$ $(+1.1011 \times 2^3 = +1101.1_2 = +13.5)$

Norma IEEE 754 – Exemplo (precisão simples)

- **Exemplo**: codificar no formato vírgula flutuante IEEE 754 precisão simples, o valor **$-12593.75_{10} \times 10^{-3}$**

$-12593.75 \times 10^{-3} = -12.59375$

Parte inteira: $12_{10} = 1100_2$

Parte fracionária: $0.59375_{10} = 0.10011_2$

$12.59375_{10} = 1100.10011_2 \times 2^0$

Normalização: $1100.10011_2 \times 2^0 = 1.10010011_2 \times 2^3$

Expoente codificado: $+3 + 127 = 130_{10} = 10000010_2$

1	10000010	10010011000000000000000

0xC1498000

```
                0.59375
              × 2
MSb   1.18750
                0.18750
              × 2
          0.37500
                0.37500
              × 2
          0.75000
                0.75000
              × 2
          1.50000
                0.50000
              × 2
LSb       1.00000
```

Norma IEEE 754 (precisão simples)

- A gama de representação suportada por este formato será portanto:

  $\pm$ [**1.00000000000000000000000**$\times$**2**$^{-126}$, **1.11111111111111111111111**$\times$**2**$^{+127}$]

  $\pm$ [**1.175494** $\times$ **10**$^{-38}$, **3.402824** $\times$ **10**$^{+38}$]

- Qual o número de dígitos à direita da vírgula na representação em decimal (casas decimais)?
- Partindo de uma representação com "**n**" dígitos fracionários na base "**r**", o número máximo de dígitos na base "**s**" que garante que a mudança de base não acrescenta precisão à representação original é:

  $$m = \left\lfloor n \frac{\log r}{\log s} \right\rfloor$$

  $\lfloor . \rfloor$ é o operador *floor*

- Assim, de modo a não exceder a precisão da representação original, a **representação em decimal** deve ter, no máximo, 6 casas decimais:

  $$m = \left\lfloor n \frac{\log r}{\log s} \right\rfloor = \left\lfloor 23 \frac{\log 2}{\log 10} \right\rfloor = 6$$

- Ou, sabendo que o nº de bits por casa decimal = $\log_2(10) \cong 3.3$), o número de casas decimais é $\lfloor$ 23 / 3.3 $\rfloor$ = **6 casas decimais**

Norma IEEE 754 (precisão simples)

- Nas operações com quantidades representadas neste formato podem ocorrer situações de ***overflow*** e de ***underflow***:
  - **Overflow**: quando o expoente do resultado não cabe no espaço que lhe está destinado → **E > 254**)

    $$N_{resultado} > 1.11111111111111111111111 \times 2^{+127}$$

  - **Underflow**: caso em que o expoente é tão pequeno que também não é representável → **E < 1**)

    $$0 < N_{resultado} < 1.00000000000000000000000 \times 2^{-126}$$

Norma IEEE 754 – Adição / Subtração

Exemplo: **N = 1.1101 × 2^0 + 1.0010 × 2^{-2}**

1º Passo: **Igualar os expoentes** ao **maior** dos expoentes

a = 1.1101 × 2^0 b = 0.010010 × 2^0

2º Passo: **Somar / subtrair** as mantissas mantendo os expoentes

N = 1.1101 × 2^0 + 0.010010 × 2^0 = 10.000110 × 2^0

3º Passo: **Normalizar** o resultado

N = 10.000110 × 2^0 = **1.0000110 × 2^1**

4º Passo: **Arredondar** o resultado e renormalizar (se necessário)

N = 1.0000|110 × 2^1 = **1.0001 × 2^1**

```
  1.0000|11
+ 0.0000|10
  1.0001|01
```

Exemplo com 4 bits fracionários

Norma IEEE 754 – Multiplicação

Exemplo: **N = (1.1100 $\times$ 2^{0}) $\times$ (1.1001 $\times$ 2^{-2})**

1º Passo: **Somar** os expoentes

Exp. Resultado = 0 + (-2) = -2

2º Passo: **Multiplicar** as mantissas

Mr = 1.1100 $\times$ 1.1001 = 10.101111

3º Passo: **Normalizar** o resultado

N = 10.101111 $\times$ 2^{-2} = **1.0101111 $\times$ 2^{-1}**

4º Passo: **Arredondar** o resultado e renormalizar (se necessário)

N = 1.0101|111 $\times$ 2^{-1} = **1.0110 $\times$ 2^{-1}**

```
  1.0101|111
+ 0.0000|100
  1.0110|011
```

Exemplo com 4 bits fracionários

Norma IEEE 754 – Divisão

Exemplo: **N = (1.0010 $\times 2^{0}$) / (1.1000 $\times 2^{-2}$)**

1º Passo: **Subtrair** os expoentes

Exp. Resultado = 0 - (-2) = 2

2º Passo: **Dividir** as mantissas

Mr = 1.0010 / 1.1000 = 0.11

3º Passo: **Normalizar** o resultado

N = 0.11 $\times 2^{2}$ = **1.1** $\times$ **2**^{1}

4º Passo: **Arredondar** o resultado

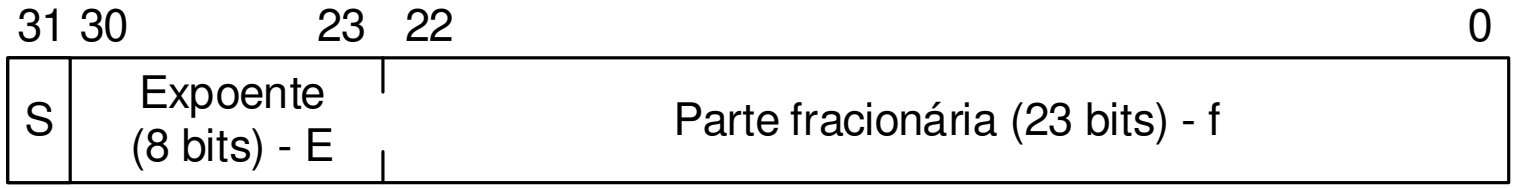
N = 1.1 $\times 2^{1}$ = **1.1000** $\times$ **2**^{1}

1.1000	**0**
+ 0.0000	**1**
1.1000	1

Exemplo com 4 bits fracionários

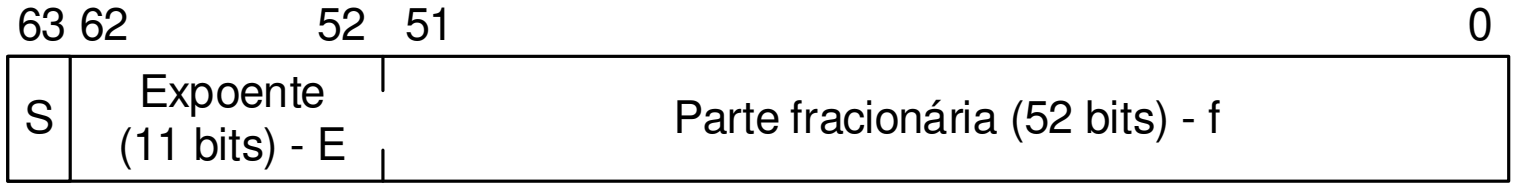
Norma IEEE 754 (precisão dupla)

- A norma IEEE 754 suporta a representação de quantidades em **precisão simples (32 bits)**



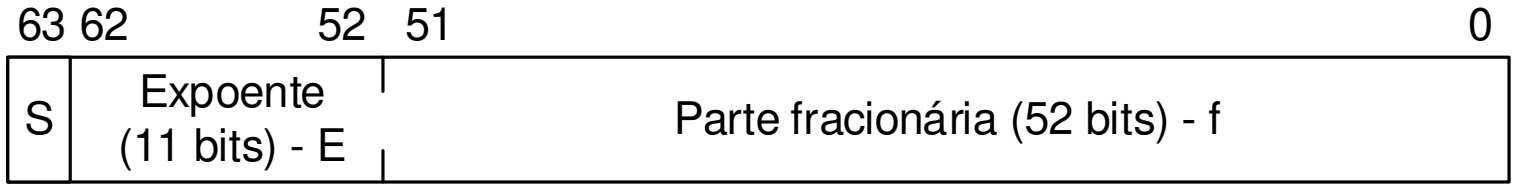
$N = (-1)^S \ 1.f \times 2^{(E - 127)}$ (Precisão simples - tipo **float**)

- e em **precisão dupla (64 bits)**



$N = (-1)^S \ 1.f \times 2^{(E - 1023)}$ (Precisão dupla - tipo **double**)

Norma IEEE 754 (precisão dupla)



$$N = (-1)^S \, 1.f \times 2^{Exp} = (-1)^S \, 1.f \times 2^{E-1023}$$

- Na codificação do expoente, os códigos 0 e 2047 são reservados. O expoente pode então tomar valores entre -1022 e +1023 [códigos 1 a 2046]
- A gama de representação suportada pelo formato de precisão dupla será:

$$\pm [1.0000000000000000\ldots000 \times 2^{-1022} , 1.1111111111111111\ldots111 \times 2^{+1023}]$$

$$\pm [2.225073858507201 \times 10^{-308} , 1.797693134862316 \times 10^{+308}]$$

- De modo a não exceder a precisão da representação original, a **representação em decimal** deve ter, no máximo, $\lfloor 52 / \log_2(10) \rfloor =$ **15 casas decimais**

Norma IEEE 754 – casos particulares

- A norma IEEE 754 suporta ainda a representação de alguns casos particulares:
  - A **quantidade zero**; essa quantidade não seria representável de acordo com o formato descrito até aqui
  - **+/-infinito (inf)**. Gama de representação excedida; divisão por 0. Exemplos: 1.0 / 0.0, -1.0 / 0.0
  - Resultados não numéricos (**NaN – Not a Number**). Exemplo: 0.0 / 0.0, inf / inf, nan * 2
  - Afim de aumentar a resolução (menor quantidade representável) é ainda possível usar um formato de **mantissa desnormalizada** no qual o bit à esquerda do ponto binário é zero

Norma IEEE 754 – casos particulares

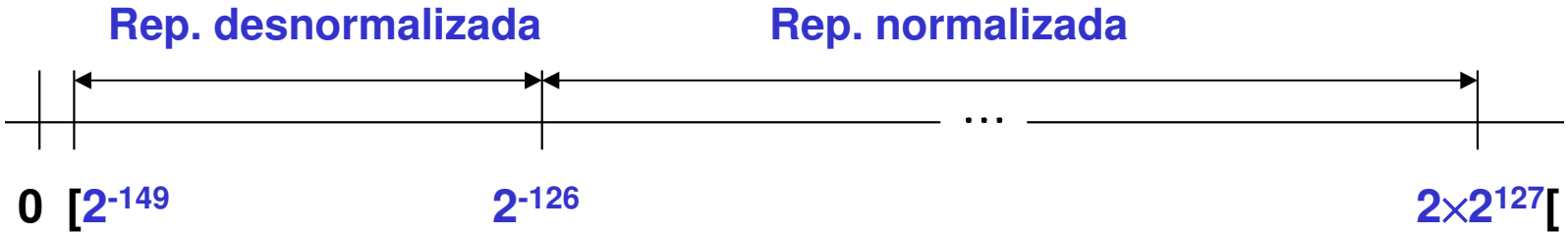
Precisão Simples		Precisão Dupla		Representa
Expoente	Parte Frac.	Expoente	Parte Frac.	
0	0	0	0	0
0	≠ 0	0	≠ 0	Quantidade desnormalizada
1 a 254	*qualquer*	*1 a 2046*	*qualquer*	*Nº em vírgula flutuante normalizado*
255	0	2047	0	Infinito
255	≠ 0	2047	≠ 0	NaN (Not a Number)

Norma IEEE 754 – representação desnormalizada

- Representação com mantissa desnormalizada: assume-se que o bit à esquerda do ponto binário é 0
- O expoente codificado "E" é 0; **o expoente verdadeiro é -126 (precisão simples) ou -1022 (precisão dupla)**
- Permite a representação de quantidades cada vez mais pequenas (*underflow* gradual)
- Gama de representação com mantissa desnormalizada, em precisão simples:

$$\pm [0.00000000000000000000001 \times 2^{-126},\ 0.11111111111111111111111 \times 2^{-126}]$$

$$\pm\ [1 \times 2^{-23} \times 2^{-126},\ 1.0 \times 2^{-126}[$$



$$\pm [1.401299 \times 10^{-45},\ 1.175494 \times 10^{-38}[$$

Técnicas de arredondamento do resultado

- As operações aritméticas são efetuadas com um número de bits da parte fracionária superior ao disponível no espaço de armazenamento
- Desta forma, na conclusão de qualquer operação aritmética é necessário proceder ao arredondamento do resultado por forma a assegurar a sua adequação ao espaço que lhe está destinado
- As técnicas mais comuns no processo de **arredondamento do resultado** (o qual introduz um erro) são:
  - Truncatura
  - Arredondamento simples
  - Arredondamento para o par (ímpar) mais próximo

Técnicas de arredondamento do resultado

- **Truncatura** (exemplo com 2 bits na parte fracionária: d=2)

val	Trunc(val)	Erro
x.00	x	0
x.01	x	-1/4
x.10	x	-1/2
x.11	x	-3/4

Erro médio = (0 – 1/4 – 1/2 – 3/4) / 4

= **–3/8**

- Mantém-se a parte inteira, desprezando qualquer informação que exista à direita do ponto binário

Técnicas de arredondamento do resultado

- **Arredondamento simples** (exemplo com 2 bits na parte fracionária: d=2)

val	Arred(val)	Erro
x.00	x	0
x.01	x	x - x.25 = -1/4
x.10	x + 1	(x+1) - x.5 = +1/2
x.11	x + 1	(x+1) - x.75 = +1/4

Erro médio

= (0 – 1/4 + 1/2 + 1/4) / 4

= **+1/8**

- Soma-se 1 ao 1º bit à direita do ponto binário e trunca-se o resultado (**`arred(val) = trunc(val + 0.5)`**)

x.00	x.01	x.10	x.11
+ 0.1	+ 0.1	+ 0.1	+ 0.1
x.10	x.11	x+1.00	x+1.01

- O erro médio é mais próximo de zero do que no caso da truncatura, mas ligeiramente polarizado do lado positivo

Técnicas de arredondamento do resultado

- **Arredondamento para o par mais próximo** (exemplo com 2 bits na parte fracionária: d=2)

val	Arred(val)	Erro	val	Arred(val)	Erro
x0.00	x0	0	x1.00	x1	0
x0.01	x0	-1/4	x1.01	x1	-1/4
x0.10	**x0**	**-1/2**	**x1.10**	**x1 + 1**	**+1/2**
x0.11	x1	+1/4	x1.11	x1 + 1	+1/4

- Semelhante à técnica de arredondamento simples, mas decidindo, para o caso "**xx.10**", em função do primeiro bit à esquerda do ponto binário

- **Erro médio** = (0 - 1/4 - 1/2 + 1/4) / 4 + (0 - 1/4 + 1/2 + 1/4) / 4
  = -1/8 + 1/8 = 0

Técnicas de arredondamento do resultado

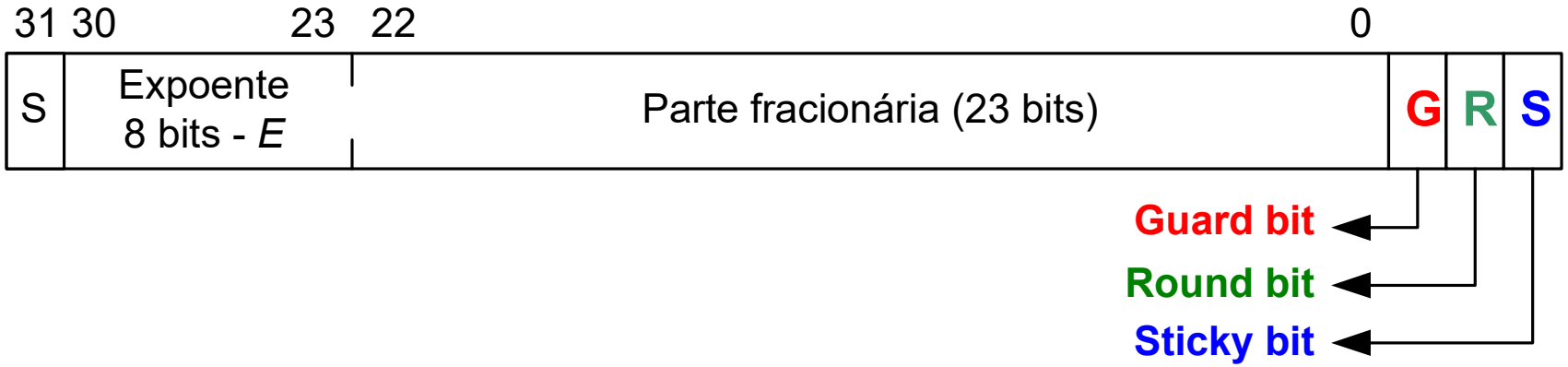
O que fica à direita de b_{23}	Exemplo	Resultado
< 0.5	$1.b_1b_2 \ldots b_{22}b_{23}$ 011	***Round down***: bits à direita de b_{23} são descartados
> 0.5	$1.b_1b_2 \ldots b_{22}b_{23}$ 101	***Round up***: soma-se 1 a b_{23} (propagando o *carry*)
= 0.5	$1.b_1b_2 \ldots b_{22}1$ 100	***Round up***: soma-se 1 a b_{23} (propagando o *carry*) (*)
= 0.5	$1.b_1b_2 \ldots B_{22}0$ 100	***Round down***: bits à direita de b_{23} são descartados (*)
= 0.5	$1.b_1b_2 \ldots B_{22}1$ 100	***Round down***: bits à direita de b_{23} são descartados (**)
= 0.5	$1.b_1b_2 \ldots b_{22}0$ 100	***Round up***: soma-se 1 a b_{23} (propagando o *carry*) (**)

(*) Arredondamento para o **par mais próximo**.

(**) Arredondamento para o **ímpar mais próximo**.

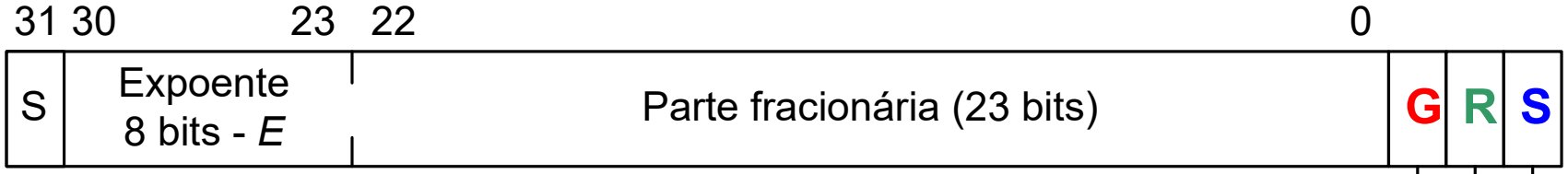
Norma IEEE 754 – arredondamentos

- Os valores resultantes de cada fase intermédia do cálculo de uma operação aritmética são armazenados com três bits adicionais, à direita do bit menos significativo da mantissa (i.e., para o caso de precisão simples, com pesos 2^{-24}, 2^{-25} e 2^{-26})



- Objetivos: 1) ter bits suplementares para a pós-normalização e 2) minimizar o erro introduzido pelo processo de arredondamento
  - **G – Guard Bit;**
  - **R – Round bit**
  - **S – Sticky bit** – Resultado da soma lógica de todos os bits à direita do bit R (i.e., se houver à direita de R pelo menos 1 bit a '1', então S='1')

Norma IEEE 754 – arredondamentos



Guard bit

Round bit

Sticky bit

Exemplo 1 (com f de 5 bits, A + B)

$A = 1.11010 \times 2^{0}$ **$B = 1.00100 \times 2^{-2}$**

$B = 0.0100100 \times 2^{0}$ (igualar ao maior dos expoentes)

Mant(A+B) = 1.11010 + 0.0100100 Expoente(A+B) = 0

= **10.00011** **000** **G = 0, R = 0, S = 0**

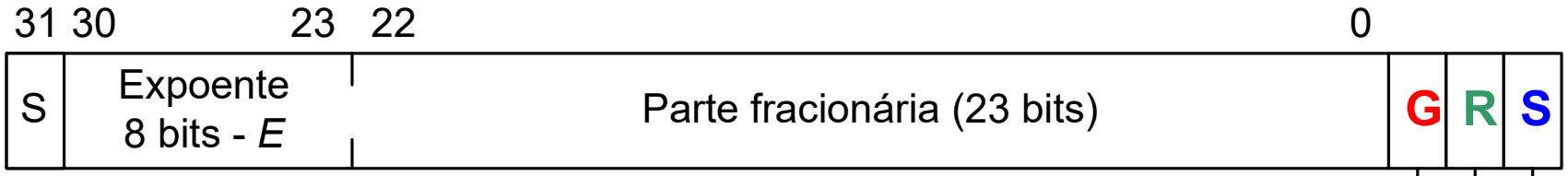
Mant(A+B) $_{norm}$= **1.00001** **100** **G = 1, R = 0, S = 0** Expoente(A+B) = 1

Arredondamento:

Mant(A + B) = 1.00010, se arred. para o par mais próximo (R=1.00010×2^{1})

Mant(A + B) = 1.00001, se arred. para o ímpar mais próximo (R=1.00001×2^{1})

Norma IEEE 754 – arredondamentos



Guard bit

Round bit

Sticky bit

Exemplo 2 (com f de 5 bits, A / B)

A = 1.00001 × 2^2 **B = 1.11111 × 2^{-1}**

Mant(A/B) = 1.00001 / 1.11111 Expoente(A/B) = 2 – (-1) = 3

= **0.10000 1100001** **G = 1, R = 1, S = OR(00001) = 1**

= **0.10000 111**

Mant(A/B) $_{norm}$ = **1.00001 110**

Arred($1,11_2$) = 10_2

Expoente(A/B) = 2

Arredondamento ⇒ Mant(A/B) = 1.00010

A/B = 1.00010 × **2^2**

Exercícios

- Na conversão de uma quantidade codificada em formato IEEE754 precisão simples para decimal, qual o número máximo de casas decimais com que o resultado deve ser apresentado? E se o valor original estiver representado em formato IEEE754 precisão dupla?
- Determine a representação em formato IEEE754 precisão simples da quantidade real **`19,1875`**. Determine a representação da mesma quantidade em precisão dupla
- Determine o valor em decimal da quantidade representada em formato IEEE754, precisão simples, como **`0xC19AB000`**
- Determine o valor em decimal da quantidade representada em formato IEEE754, precisão simples, como **`0x80580000`**