

A photograph of a server room with rows of black server racks. A semi-transparent blue rectangle is overlaid on the right side of the image, containing white text. The text reads "Representação e operações sobre números" in a large, bold font, and "Cap. 3: 'Arithmetic for Computers' (parcial)" in a smaller font below it.

Representação e operações sobre números

Cap. 3: "Arithmetic for Computers" (parcial)



Representação de números

Revisão de Sistemas Digitais, mas não só...

1. Operandos em arquitetura de computadores

- Todas as operações são realizadas em base 2 (binário)
- Para efeitos de visualização os operandos podem ser representados em base 16 (hexadecimal) ou 10 (decimal)

Conversão de bases:

- Considere o número x representado na base b
 - Como converter para uma base c ?
1. Separar o número x em parte inteira (x_i) e parte fracionária (x_f)
 2. Converter as duas partes em separado e somar no final
 - A. Converter a parte inteira através do método das divisões sucessivas pela base de destino
 - B. Converter a parte fracionária através do método das multiplicações sucessivas pela base de destino

Ex. conversão do número 19,27 para base 2:

$19 \rightarrow 19 \begin{array}{r l} 2 & \\ \hline 1 & 9 \\ 1 & 4 \\ 0 & 2 \\ 0 & 1 \\ 1 & 0 \end{array}$	$0.27 \times 2 = \underline{0}.54$
	$0.54 \times 2 = \underline{1}.08$
	$0.08 \times 2 = \underline{0}.16$
	$0.16 \times 2 = \underline{0}.32$
	$0.32 \times 2 = \underline{0}.64$
	...

$$19,27 = 10011,0100010100011101011100001010...$$

Uma dízima finita numa base b por ser infinita numa base c !

1. Operandos em arquitetura de computadores

- Todas as operações são realizadas em base 2 (binário)
- Para efeitos de visualização os operandos podem ser representados em base 16 (hexadecimal) ou 10 (decimal)

Conversão de bases:

- Considere o número x representado na base b
 - Como converter para uma base c ?
1. Separar o número x em parte inteira (x_i) e parte fracionária (x_f)
 2. Converter as duas partes em separado e somar no final
 - A. Converter a parte inteira através do método das divisões sucessivas pela base de destino
 - B. Converter a parte fracionária através do método das multiplicações sucessivas pela base de destino

Ex. conversão do número 11101.101_2 para base 10:

$$\begin{aligned} 11101.101_2 &= 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + \\ &\quad 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = \\ &= 16 + 8 + 4 + 1 + 0.5 + 0.125 \\ &= 29.625 \end{aligned}$$

1. Operandos em arquitetura de computadores

- Todas as operações são realizadas em base 2 (binário)
- Para efeitos de visualização os operandos podem ser representados em base 16 (hexadecimal) ou 10 (decimal)

Conversão de bases:

- Considere o número x representado na base b
- Como converter para uma base c ?

Se $b = p^m$ e $c = p^n$, então a conversão pode ser feita diretamente!

Cada conjunto de m símbolos na base c corresponde a um conjunto de n símbolos na base b

Muito fácil de aplicar se $m=1$ ou $n=1$!

Base binária (2^1) para hexadecimal (2^4)

Cada 4 símbolos na base 2 equivale a 1 símbolo na base 16

$$00100111011_2 = \underbrace{0001}_1 \underbrace{0011}_3 \underbrace{1011}_B_2 = 13B_{16}$$

Base octal (2^3) para base binária (2^1)

Cada 1 símbolo na base 8 equivale a 3 símbolos na base 2

$$13421_8 = \underbrace{001}_1 \underbrace{011}_3 \underbrace{100}_4 \underbrace{010}_2 \underbrace{001}_1_2$$

1. Operandos em arquitetura de computadores

- Todas as operações são realizadas em base 2 (binário)
- Para efeitos de visualização os operandos podem ser representados em base 16 (hexadecimal) ou 10 (decimal)

2. Operandos com sinal

- O bit mais significativo representa o sinal: 0 positivo; 1 negativo
- Três representações:

Sinal e módulo:

- Considere um número x representado com 8 bits: $x_7x_6x_5x_4x_3x_2x_1x_0$
- Sinal → bit mais significativo (neste caso x_7)
- Módulo → restantes bits ($x_6x_5x_4x_3x_2x_1x_0$)
- A representação do número $y = -x$ é realizada complementando o bit de sinal
- Existem dois zeros, +0 e -0.

1. Operandos em arquitetura de computadores

- Todas as operações são realizadas em base 2 (binário)
- Para efeitos de visualização os operandos podem ser representados em base 16 (hexadecimal) ou 10 (decimal)

2. Operandos com sinal

- O bit mais significativo representa o sinal: 0 positivo; 1 negativo
- Três representações:

Complemento para um:

- Considere um número x representado com 8 bits: $x_7x_6x_5x_4x_3x_2x_1x_0$
- Sinal \rightarrow bit mais significativo (neste caso x_7)
- Se o sinal for positivo, os restantes bits representam o módulo, caso contrário representam o complemento do módulo
- A representação do número $-x$ é realizada complementando (operação lógica *not*) todos os bits do número x , i.e., $\overline{x_7} \overline{x_6} \overline{x_5} \overline{x_4} \overline{x_3} \overline{x_2} \overline{x_1} \overline{x_0}$
- Existem dois zeros, +0 e -0.

1. Operandos em arquitetura de computadores

- Todas as operações são realizadas em base 2 (binário)
- Para efeitos de visualização os operandos podem ser representados em base 16 (hexadecimal) ou 10 (decimal)

2. Operandos com sinal

- O bit mais significativo representa o sinal: 0 positivo; 1 negativo
- Três representações:

Complemento para dois:

- Considere um número x representado com 8 bits: $x_7x_6x_5x_4x_3x_2x_1x_0$
- Sinal → bit mais significativo (neste caso x_7)
- Se o sinal for positivo, os restantes bits representam o módulo, caso contrário representam o complemento do módulo, somado de um
- A representação do número $-x$ é realizada complementando (operação lógica *not*) todos os bits do número x , i.e., $\bar{x}_7 \bar{x}_6 \bar{x}_5 \bar{x}_4 \bar{x}_3 \bar{x}_2 \bar{x}_1 \bar{x}_0$, a que se soma o valor 1
- Só existe um zero, mas a gama dos números negativos é maior que a gama dos números positivos

$$-x = \bar{x} + 1$$



Independentemente de x
ser positivo ou negativo

3. Tipos de operandos em arquitetura de computadores

- Nomenclatura baseada na implementação

Programador (ex: C)	Arquiteto de computadores (ex: Assembly)	Dimensão
char / byte	byte	1B = 8 bits
short	half word	2B = 16 bits
int / long	word	4B = 32 bits
long long	double word	8B = 64 bits
	quad word	16B = 128 bits



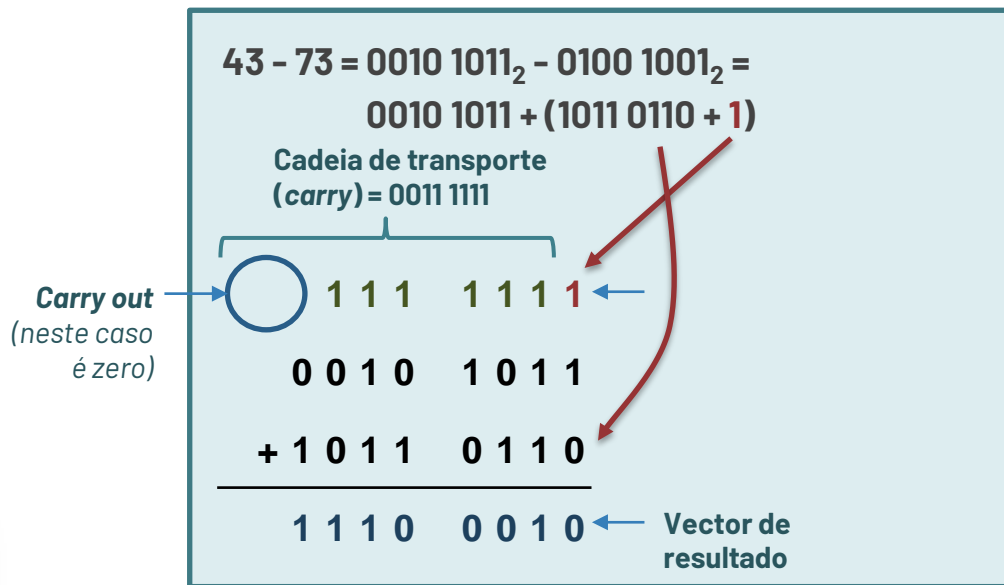
Podem ser *unsigned*



Mesma designação → a diferença está na operação! Ex: DIV (signed) ou DIVU (unsigned)

1. Soma/subtração

- Geralmente os números inteiros com sinal são representados em formato complemento para dois → facilita a implementação do somador!
- A operação é realizada como se tratasse de um número binário convencional.



Flags:

- Carry (C) = 0
- Zero (Z) = 0
- Negative (N) = 1
- Overflow (V) = 0

1. Soma/subtração

- Geralmente os números inteiros com sinal são representados em formato complemento para dois → facilita a implementação do somador!
- A operação é realizada como se tratasse de um número binário convencional.

114 + 67 = 0111 0010₂ + 0100 0011₂

Carry out (neste caso é zero) → 0 1 1 1 0 0 1 0

+ 0 1 0 0 0 0 1 1

1 0 1 1 0 1 0 1 ← Vector de resultado

Flags:

- **Carry (C) = 0** (o resultado está correto se os operandos não tiverem sinal)
- Zero (Z) = 0
- Negative (N) = 1
- **Overflow (V) = 1** (O resultado está errado se os operandos tiverem sinal, i.e., não é representável em 8 bits com sinal)

2. Multiplicação sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$\begin{array}{r} 114 \times 67 = 0111\,0010_2 \times 0100\,0011_2 \\ 01110010 \\ \times 01000011 \\ \hline 01110010 \\ 01110010 \\ + 01110010 \\ \hline 01110111010110 \end{array}$$

O produto de 2 números com n e m bits, respetivamente, resulta num número contido em $n+m$ bits

8 bits x 8 bits → 16 bits

3. Multiplicação com sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$\begin{array}{r} -114 \times 67 = 1000\ 1110_2 \times 0100\ 0011_2 \\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0 \\ \times 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1 \\ \hline \end{array}$$

O produto de 2 números com n e m bits, respetivamente, resulta num número contido em $n+m$ bits

8 bits x 8 bits → 16 bits

3. Multiplicação com sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$-114 \times 67 = 1000\ 1110_2 \times 0100\ 0011_2$$

	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	
x	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	
<hr/>																
	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	
	1	1	1	1	1	1	1	0	0	0	1	1	1	0		
+	1	1	1	0	0	0	1	1	1	0						
<hr/>																
	1	1	1	0	0	0	1	0	0	0	1	0	1	0	1	0

Para realizar um produto em números com sinal, primeiro deve-se estender ambos os operandos de forma a perfazer o número de bits usado no resultado.

A operação de multiplicação de números (no pior dos casos) usa uma cadeia de somadores igual ao número de bits do resultado!

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$100 \overline{) 10101101}$$

$$1 > 100?$$

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 0 \\ 100 \overline{) 10101101} \end{array}$$

10 > 100?

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 00 \\ 100 \overline{) 10101101} \end{array}$$

101 > 100?

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 001 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 1 \end{array}$$

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 0010 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 10 \end{array}$$

10 > 100?

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 0010 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 101 \end{array}$$

$$101 > 100?$$

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 00101 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 1 \end{array}$$

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 00101 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 11 \end{array}$$

11 > 100?

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 001010 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 110 \end{array}$$

$$110 > 100?$$

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 0010101 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 110 \\ \underline{-100} \\ 10 \end{array}$$

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 0010101 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 110 \\ \underline{-100} \\ 101 \end{array}$$

101 > 100?

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 00101011 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 110 \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 1 \end{array}$$

Resultado: $101011_2 = 43$

Resto: 1

A operação de divisão é bastante mais complexa, envolvendo várias operações encadeadas!



Embora existam vários algoritmos para a implementação eficiente da divisão, como se verá mais à frente, o tempo de cálculo é sempre bastante superior ao de uma soma ou mesmo de uma multiplicação!

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.
- Se o divisor for uma potência de dois, existe uma forma mais simples:



$$z = \frac{x}{2^y} = \begin{cases} x \text{ shr } y & \text{se } y > 0 \\ x \text{ shl } y & \text{se } y < 0 \end{cases}$$

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$173 / 4 = 173 / 2^2 =$$

$$173 \text{ shr } 2 =$$

$$1010\ 1101_2 \gg 2$$

$$0010\ 1011$$

$$\text{Resultado: } 101011_2 = 43$$

Boas práticas de programação:

sempre que possível, evitar a utilização da operação de divisão!

Nota: também se deve evitar a multiplicação, embora a complexidade da multiplicação seja bastante inferior

4. Divisão sem sinal

- No caso da multiplicação (ou da divisão) a operação tem de ser realizada tendo em vista o sinal dos operandos.
- Se estivermos a dividir várias vezes pelo mesmo valor (x), podemos calcular o recíproco ($1/x$) e depois multiplicar os valores. Ex:

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$173 / 4 = 173 / 2^2 =$$

$$173 \text{ shr } 2 =$$

$$1010\ 1101_2 \gg 2$$

$$0010\ 1011$$

$$\text{Resultado: } 101011_2 = 43$$

Boas práticas de programação:

sempre que possível, evitar a utilização da operação de divisão!

Nota: também se deve evitar a multiplicação, embora a complexidade da multiplicação seja bastante inferior

```
for (i=0; i<N; i++)  
    A[i] = A[i]/x;
```



```
tmp=1/x;  
for (i=0; i<N; i++)  
    A[i] = A[i]*tmp;
```

- **Como representar um número com virgula?**

- Exemplo 110.010_2



Não podemos representar a virgula nem com 1 nem com 0!

Em ambos os casos seria visto simplesmente como um número inteiro de valor elevado.

Representação de números com virgula

1. Virgula fixa

- A vírgula está numa posição fixa, mas apenas o programador sabe onde ela está.
- Para realizar uma operação, é necessário realizar todas as normalizações e desnormalizações à mão

2. Virgula flutuante

- Representação em modo científico, ex: $1427.3 \rightarrow 1.427 \times 10^3$ (com 3 dígitos de parte fracionária)
- Geralmente faz uso da norma IEEE-754
- A representação adapta-se de acordo com o número de bits do sinal.
- A desvantagem é uma potencial redução na precisão do número

1. Virgula fixa

- A vírgula está numa posição fixa, mas apenas o programador sabe onde ela está.
- Para realizar uma operação, é necessário realizar todas as normalizações e desnormalizações à mão
- Utiliza-se frequentemente a nomenclatura da Texas Instruments, a qual assume números em complemento para dois.
- Formato $Q_n \rightarrow n$ bits para parte fracionária
- Formato $Q_m.n \rightarrow m$ bits para a parte inteira, n bits para parte fracionária

O número $0110\ 11.01_2 = +27,25$ está representado no formato $Q2$ ou $Q6.2$

O formato $Q6.2$ permite representar números na gama $[-2^5, \dots, 2^5 - 2^{-2}]$

1. Virgula fixa

- A vírgula está numa posição fixa, mas apenas o programador sabe onde ela está.
- Para realizar uma operação, é necessário realizar todas as normalizações e desnormalizações à mão
- Utiliza-se frequentemente a nomenclatura da Texas Instruments, a qual assume números em complemento para dois.
- Formato $Q_n \rightarrow n$ bits para parte fracionária
- Formato $Q_{m.n} \rightarrow m$ bits para a parte inteira, n bits para parte fracionária

O número $0110\ 11.01_2 = +27,25$ está representado no formato $Q2$ ou $Q6.2$

O formato $Q6.2$ permite representar números na gama $[-2^5, \dots, 2^5 - 2^{-2}]$

- Na prática a representação indica que o operando pode ser representado como inteiro (para o exemplo em cima, $0110\ 1101_2$) à parte de um fator de escala de 2^{-n} (que neste caso corresponde a 2^{-2})


Representação de números com virgula

1. Virgula fixa

- Para realizar operações em virgula fixa é necessário ter em consideração os fatores de escala de cada um dos operandos, normalizando-os (converter para o mesmo fator de escala) antes de efetuar a operação (quando necessário).

A. Soma/subtração de operandos com formato Qx e Qy (assumindo que o número total de bits é o mesmo)

- Verificar qual dos números tem **maior** número de bits na parte inteira, i.e., qual tem menor número de bits na parte fracionária
- Ao operando escolhido em 1) fazer shift left do módulo da diferença $|x-y|$
- Somar os operandos

$$\begin{array}{ccc} 1101\ 01.10 & + & 1100.1110 \rightarrow (1101\ 01.10\ \text{sl } 2) + 1100.1110 \rightarrow 0101.1000 + 1100.1110 \\ \text{(Q2)} & \text{(Q4)} & \text{Q4} \end{array}$$


Esta solução maximiza a precisão, mas pode levar a **overflow** e portanto deve ser evitada

1. Virgula fixa

- Para realizar operações em virgula fixa é necessário ter em consideração os fatores de escala de cada um dos operandos, normalizando-os (converter para o mesmo fator de escala) antes de efetuar a operação (quando necessário).

A. Soma/subtração de operandos com formato Qx e Qy (assumindo que o número total de bits é o mesmo)

1. Verificar qual dos números tem menor número de bits na parte inteira, i.e., qual tem maior número de bits na parte fracionária
2. Ao operando escolhido em 1) fazer shift right aritmético (complemento para dois) do módulo da diferença $|x-y|$
3. Somar os operandos

$1101\ 01.10 + 1100.1110 \rightarrow 1101\ 01.10 + (1100.1110 \text{ asr } 2) \rightarrow 1101\ 01.10 + 111100.11$
(Q2) (Q4)

Representação de números com virgula

1. Virgula fixa

- Para realizar operações em virgula fixa é necessário ter em consideração os fatores de escala de cada um dos operandos, normalizando-os (converter para o mesmo fator de escala) antes de efetuar a operação (quando necessário).

A. Soma/subtração de operandos com formato Qx e Qy (assumindo que o número total de bits é o mesmo)

1. Verificar qual dos números tem **menor** número de bits na parte inteira, i.e., qual tem maior número de bits na parte fracionária
2. Ao operando escolhido em 1) fazer shift right aritmético (complemento para dois) do módulo da diferença $|x-y|$
3. Somar os operandos

$1101\ 01.10 + 1100.1110 \rightarrow 1101\ 01.10 + (1100.1110 \text{ asr } 2) \rightarrow 1101\ 01.10 + 111100.11$
(Q2) (Q4)

Esta solução minimiza a existência de overflows, mas leva a perda de precisão

Ainda assim, é a solução preferencial (a não ser que saibamos à partida que o número de bits é suficiente)

1. Virgula fixa

- Para realizar operações em virgula fixa é necessário ter em consideração os fatores de escala de cada um dos operandos, normalizando-os (converter para o mesmo fator de escala) antes de efetuar a operação (quando necessário).

A. Soma/subtração de operandos com formato Q_x e Q_y (assumindo que o número total de bits é o mesmo)

- Em geral, quando somamos dois operandos no formato $Q_z.y$, só poderemos garantir a inexistência de overflow se a operação for realizada com 1 bit extra na parte inteira, i.e., fazendo:

$$\begin{array}{ll} a + b \rightarrow (a \text{ asr } 1) + (b \text{ asr } 1) \\ (Q_z.y) & (Q_{z+1}.y-1) \end{array}$$

1. Virgula fixa

- Para realizar operações em virgula fixa é necessário ter em consideração os fatores de escala de cada um dos operandos, normalizando-os (converter para o mesmo fator de escala) antes de efetuar a operação (quando necessário).

B. Produto de operandos com formato Qx e Qy (assumindo que o número total de bits é o mesmo)

1. Não é necessária qualquer normalização à entrada, i.e., pode-se multiplicar os operandos normalmente
2. O resultado estará no formato Qx+y

$$\begin{array}{ll} 1101\ 01.10_2 \times 1100.1110_2 & = (1101\ 0110_2 \times 2^{-2}) \times (1100\ 1110_2 \times 2^{-4}) \rightarrow 1101\ 0110_2 \times 1100\ 1110_2 \times 2^{-6} \\ \text{(Q2)} \quad \quad \text{(Q4)} & \qquad \qquad \qquad 0000\ 1000\ 0011\ 0100_2 \times 2^{-6} \\ & \qquad \qquad \qquad 0000\ 1000\ 00.11\ 0100 \quad (\text{formato Q6}) \end{array}$$

1. Virgula fixa

- Para realizar operações em virgula fixa é necessário ter em consideração os fatores de escala de cada um dos operandos, normalizando-os (converter para o mesmo fator de escala) antes de efetuar a operação (quando necessário).

B. Produto de operandos com formato Qx e Qy (assumindo que o número total de bits é o mesmo)

- Se o realizarmos as operações com número de bits limitado (ex: 8), para garantir a inexistência de overflow, deve-se preservar sempre a parte alta

$$11.01\ 0110_2 \times 11.001110_2 = (1101\ 0110_2 \times 2^{-6}) \times (1100\ 1110_2 \times 2^{-6}) \rightarrow 0000\ 1000\ 0011\ 0100_2 \times 2^{-12}$$

(Q6) **(Q6)**

0000 . 1000 0011 0100 (formato Q12)

0000 . 1000 (formato Q4)

2. Virgula flutuante

- Representação em modo científico, ex: $1427 \rightarrow 1.427 \times 10^3$
- **Norma IEEE-754**
 - Base b , 2 ou 10
 - Mantissa m , a qual é sempre representada no formato 1.xxx
 - Expoente e
 - Sinal s
 - Representa o número: $(-1)^s \times m \times b^e$

$$\begin{aligned} 1427 &= 0000\ 0101\ 1001\ 0011_2 = \\ &= 1.011\ 0010\ 0110_2 \times 2^{10} \end{aligned}$$

(a virgula desloca-se 10 casas)

Representação de números com virgula

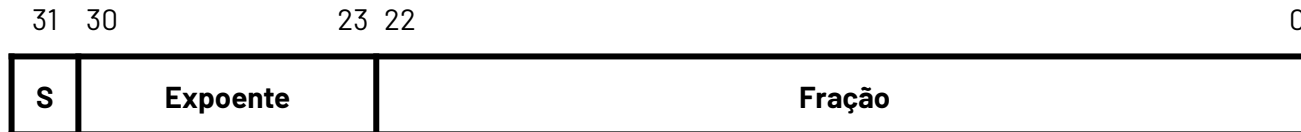
2. Virgula flutuante

- Representação em modo científico, ex: $1427 \rightarrow 1.427 \times 10^3$
- Norma IEEE 754**
 - Base b , 2 ou 10
 - Mantissa m , a qual é sempre representada no formato 1.xxx
 - Expoente e
 - Sinal s
 - Representa o número: $(-1)^s \times m \times b^e$

$$\begin{aligned} 1427 &= 0000\ 0101\ 1001\ 0011_2 = \\ &= 1.011\ 0010\ 0110_2 \times 2^{10} \end{aligned}$$

(a virgula desloca-se 10 casas)

Representação no formato de 32 bits (single-precision)



Representação de números com virgula

2. Virgula flutuante

- Representação em modo científico, ex: $1427 \rightarrow 1.427 \times 10^3$
- Norma IEEE 754**
 - Como a mantissa começa com 1.xxxx, não precisamos de escrever o 1. no campo de fração!
 - Assim existe sempre um 1 “escondido” que não é representado na fração

$$1427 = 0000\ 0101\ 1001\ 0011_2 = \\ = 1.\underline{011\ 0010\ 0110}_2 \times 2^{10}$$

Fração:

0110 0100 1100 0000 0000 000

Representação no formato de 32 bits (single-precision)

31 30

23 22

0



2. Virgula flutuante

- Representação em modo científico, ex: $1427 \rightarrow 0.1427 \times 10^4$
- **Norma IEEE 754**
 - O expoente é representado com um offset: 127.

$$1427 = 0000\ 0101\ 1001\ 0011_2 =$$
$$= 1.011\ 0010\ 0110_2 \times 2^{10}$$

Expoente:

$$10 + 127 = 137 = 1000\ 1001$$

Representação no formato de 32 bits (single-precision)

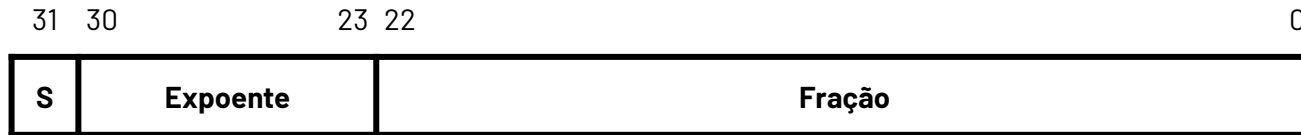
0

S=0	Expoente= 1000 1001	Fração = 0110 0100 1100 0000 0000 000
-----	---------------------	---------------------------------------

2. Virgula flutuante

- Representação em modo científico, ex: $1427 \rightarrow 0.1427 \times 10^4$
- Norma IEEE 754**

Representação no formato de 32 bits (single-precision)



Expoente	Tipo	Representação
01h ... FEh	Formato normalizado	$(-1)^s \times 1.fracção \times b^{exp-127}$
00h	Formato não normalizado	$(-1)^s \times 0.fracção \times b^{exp-126}$
FFh	Valores especiais	Se $fracção=0 \rightarrow +/- \infty$ Se $fracção \neq 0 \rightarrow NaN$

Representação de números com virgula

2. Virgula flutuante

- Representação em modo científico, ex: $1427 \rightarrow 0.1427 \times 10^4$
- Norma IEEE-754**

Nome	Nome comum	Base	Fração (#bits)	Expoente (#bits)	Expoente (Bias)	Casas decimais da fração	Expoente decimal máximo
Binary16	Half precision	2	11	5	15	3.31	4.51
Binary32	Single precision (float)	2	23	8	127	7.22	38.23
Binary64	Double precision (double)	2	53	11	1023	15.95	307.95
Binary128	Quadruple precision	2	113	15	16383	34.02	4931.77
Binary256	Octuple precision	2	237	19	262143	71.34	78913.2