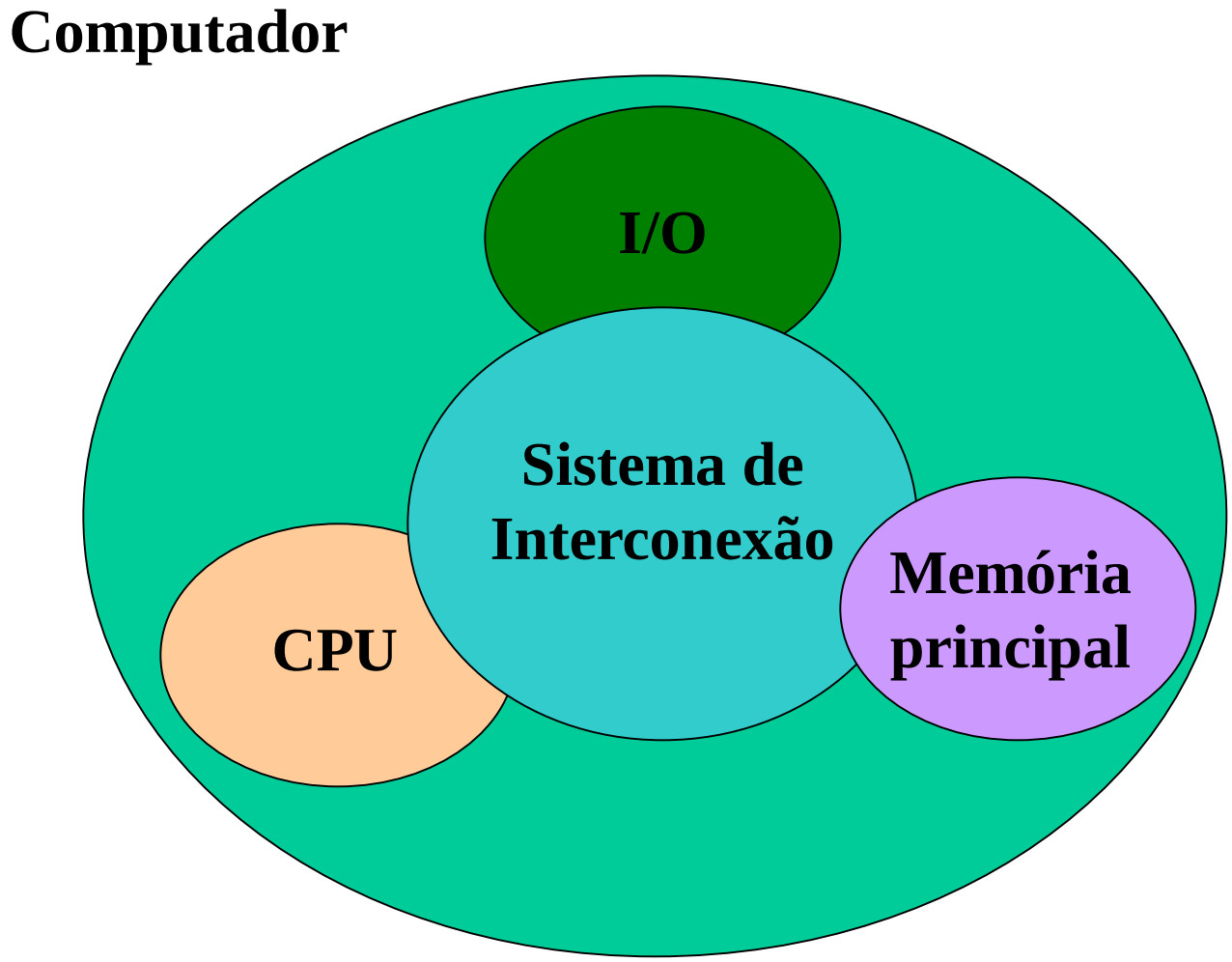
# ALU/ULA

Arithmetic and logic unit/Unidade Lógica e aritmética

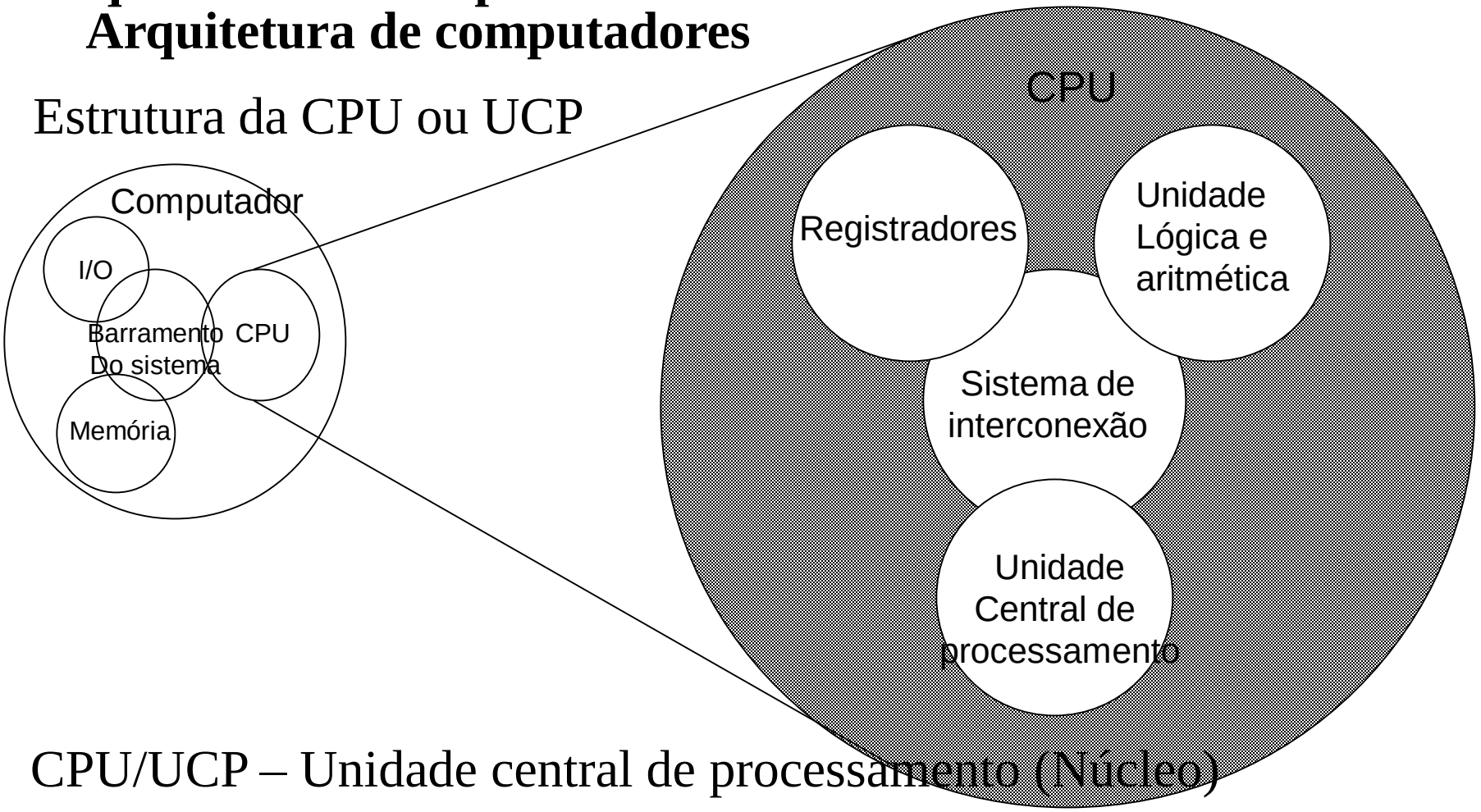
# Arquitetura de computadores



# Arquitetura de computadores

**Arquitetura de computadores**

Estrutura da CPU ou UCP




CPU/UCP – Unidade central de processamento (Núcleo)

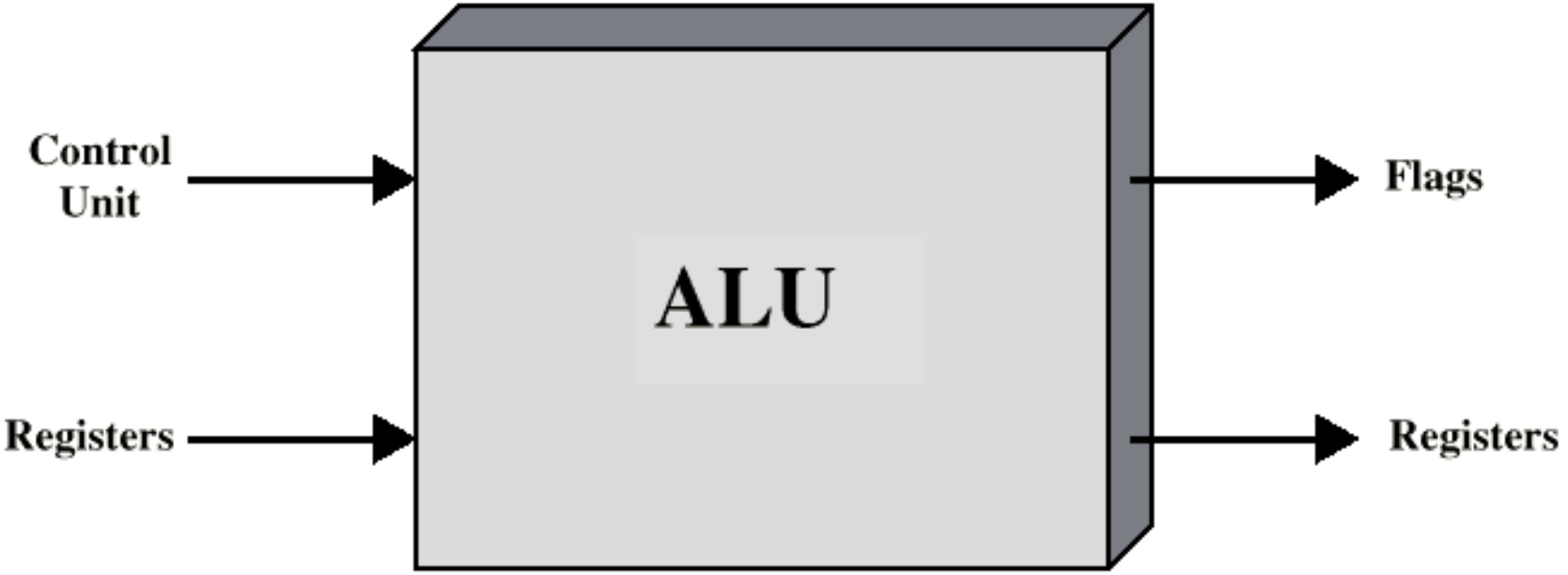
Registradores – Funcionam como “variáveis” para o uso da CPU

Unidade Lógica e aritmética (ALU/ULA) – efetua contas matemáticas e expressões lógicas

Sistema de interconexão – liga os partes os elementos da CPU

# Arquitetura de computadores

Funcionamento básico




A unidade de controle solicita o tipo de operação a ser efetuada Ex: AND,OR,SUB,ADD etc...(operação) e também são passados os registradores para a mesma (operandos).
O mesmo faz a operação e devolve:
A resposta em um/uns registrador(res) e a resposta do que aconteceu com a operação no registrador de flags(ex: carry(vai um da matemática),operação gerou zero,etc...)

**Arquitetura de computadores**

Para entenderemos melhor o funcionamento da ULA/ALU devemos saber como os números são processados dentro do computador.

Isso por vários motivos entre os quais:

a) Ajuda compreender como os circuitos da ALU/ULA fazem as operações
b) A representação de números pode ser implementada de forma diferente entre os computadores, se for necessária a conversa entre computadores que representam esses números de forma diferente é necessário conversão

# Revisão dos Sistemas Numéricos

Decimal

baseado em 10 dígitos (0,1,2,3,4,5,6,7,8.9)

Exemplo:

$5264 = (5x1000) + (2x100) + (6x10) + 4x1$

$5264 = (5x10^{3}) + (2x10^{2}) + (6x10^{1}) + (4x10^{0})$

Base ou raíz

Valores fracionários:

$75{,}32 = (7\text{x}10^{1}) + (5\text{x}10^{0}) + (3\text{x}10^{-1}) + (2\text{x}10^{-2})$

De uma forma geral:

$$X = \sum_{i} x_i 10^i$$

onde: $x_i \in (0,1,2,3,4,5,6,7,8,9)$ e i corresponde a posição do dígito.

Sistema Binário

baseado em dois dígitos: 0 e 1 (base 2)

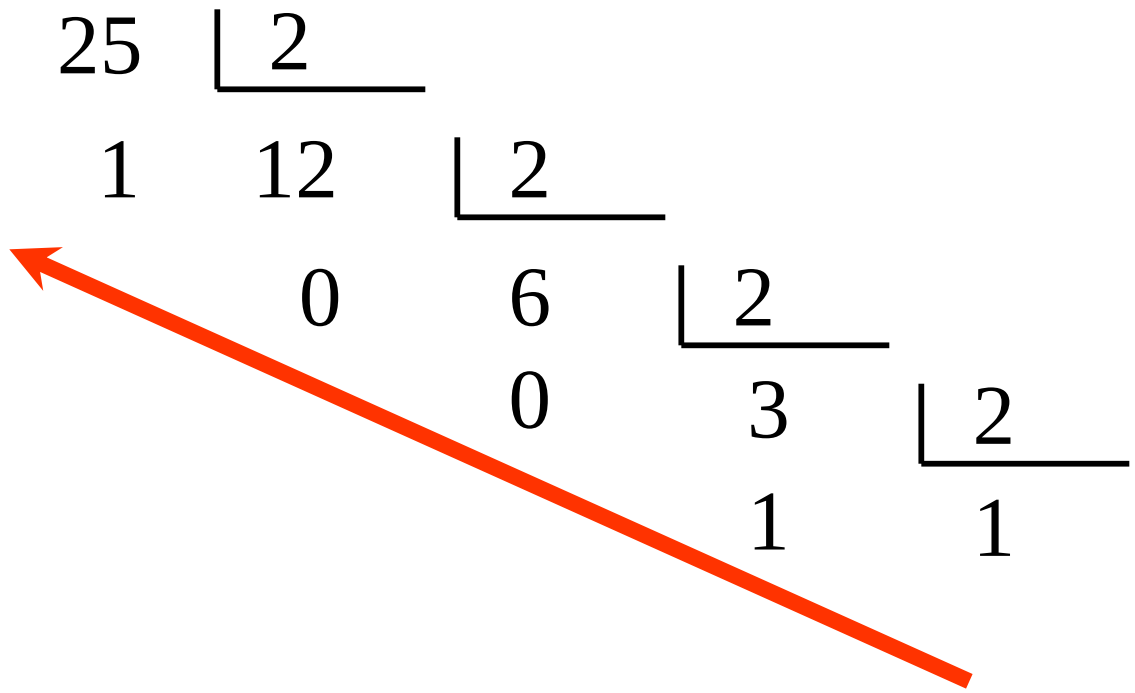
Exemplos:

$11 = (1x2^1) + (1x2^0) \quad = 3$

$110 = (1x2^2) + (1x2^1) + (0x2^0) \quad = 6$

$100.101 = 2^2 + 2^{-1} + 2^{-3} \quad = 4.625$

# Exemplo: conversão de decimal para binário




$$25 = 1x\ 2^4 + 1\ x\ 2^3 + 0\ x\ 2^2 + 0\ x\ 2^1 + 1\ x\ 2^0$$

**1 1 0 0 1**

# Exemplo:conversão de decimal para binário

0.56 x 2 = 1.12
0.12 x 2 = 0.24
0.24 x 2 = 0.48
0.48 x 2 = 0.96
0.96 x 2 = 1.92

$0.56 = 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} \ldots$

**.10001** (aproximadamente)

# Representação de Inteiros

Na representação binária podem ser representados com 0 e 1, o sinal de negativo e ponto:

-101.01010

Ao se trabalhar com números binários no computador não é possível usar o sinal de negativo e o ponto. Com números positivos e inteiros a representação é direta:

00110001 = 49

00010101 = 21

usando-se 8 bits

# Representação de sinal

O bit de maior significância (mais a esquerda) é tratado com o bit de sinal. Se o bit de sinal for 0, o número é positivo, se for 1 é negativo.

# Representação Sinal-Magnitude

A forma mais simples é tal que os n-1 bits representam a magnitude do número. Assim:

$$A = \begin{cases} \sum_{i=0}^{n-2} a_i 2^i & \text{se } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} a_i 2^i & \text{se } a_{n-1} = 1 \end{cases}$$

Exemplo:

+18 = 00010010

- 18 = 10010010

## Desvantagens

- As operações aritméticas se tornam mais complicadas;
- Existem duas representações para o zero:

+0 = 00000000

-0 = 10000000

# Representação de Complemento de Dois

- Números inteiros:

$$A = \sum_{i=0}^{n-2} a_i 2^i \qquad (1)$$

- Números negativos:

$$A = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \qquad (2)$$

Note que (1) está contido em (2) uma vez que para números positivos, $a_{n-1} = 0$

O zero é identificado como sendo positivo assim em sua representação o bit de sinal é 0.

# Arquitetura de computadores

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| $-2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

-128 16 4 1 = -107

| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

-128 64 32 8 = -24

# Conversão do número de bits de representação

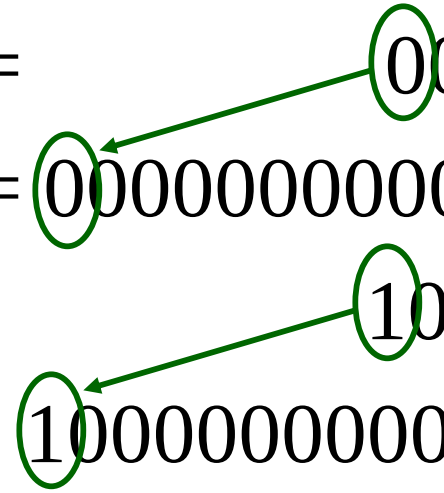
- representação sinal-magnitude:

+18 = 00010010 (8 bits)

+18 = 0000000000010010 (16 bits)

-18 = 10010010 (8 bits)

-18 = 1000000000010010 (16 bits)



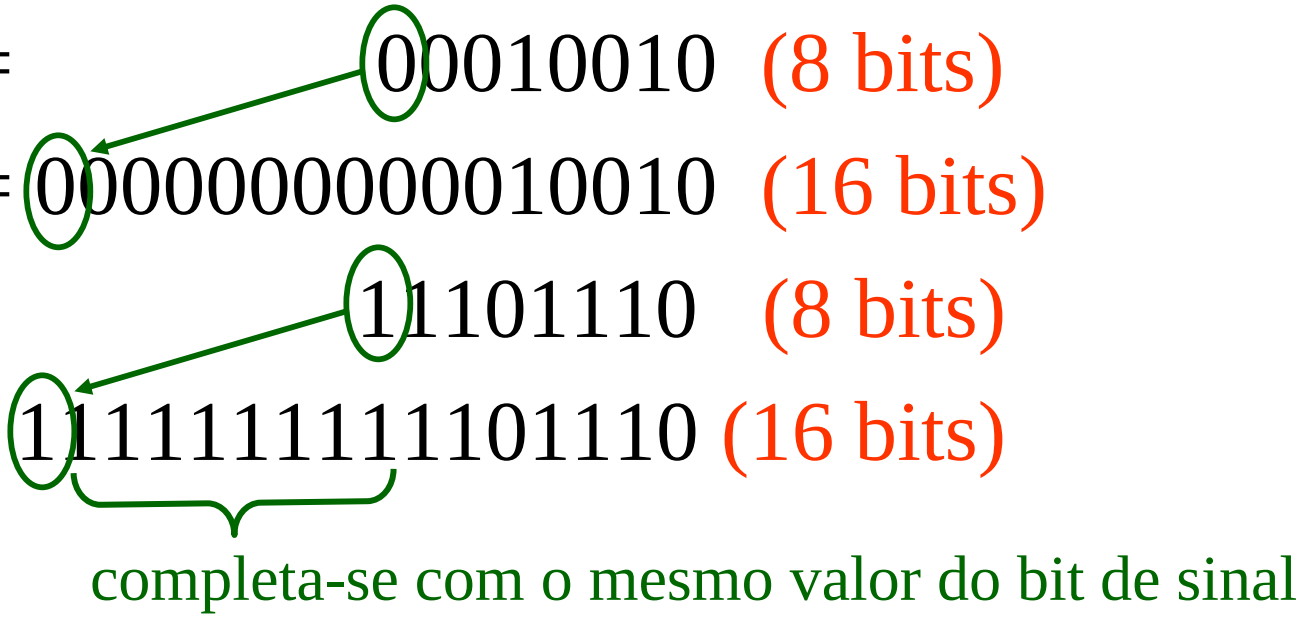
- representação de complemento de 2, forma correta de aumentar o número de bits:

+18 = 00010010 (8 bits)

+18 = 0000000000010010 (16 bits)

-18 = 11101110 (8 bits)

-18 = 1111111111101110 (16 bits)



# Aritmética de Inteiros (Representação de Complemento de Dois)

- Negação

1. Tome o complemento booleano de cada bit (inclusive o bit de sinal).
2. Tratando o resultado como um binário inteiro não sinalizado, acrescente 1

Exemplo

```
          +18 = 00010010
complemento   = 11101101
              +        1
          ----------------
          11101110 = -18
```

***Mostre que: (-18) = +18***

+0 = 00000000
complemento = 11111111
\+ 1
1 00000000 = 0

"carry in" ou "vai um"

128 = 10000000
complemento = 01111111
\+ 1
10000000 = -128

não existe uma representação para $2^n$

**Arquitetura de computadores**

Portanto para 8 bits temos:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| $-2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

O menor número –128 passado por zero e até 127

| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

-128 = -128

| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|

0 64 32 16 8 4 2 1 = 127

# Arquitetura de computadores

| Tipo da ling. C | Número bits | <limits.h> Constant | Valores Faixa |
|---|---|---|---|
| signed char | 8 | SCHAR_MIN | -128 |
| | | SCHAR_MAX | 127 |
| unsigned char | 8 | | 0 |
| | | UCHAR_MAX | 255 |
| signed short | 16 | SHRT_MIN | -32768 |
| | | SHRT_MAX | 32767 |
| unsigned short | 16 | | 0 |
| | | USHRT_MAX | 65535 |
| signed int | 16 | INT_MIN | -32768 |
| | | INT_MAX | 32767 |
| unsigned int | 16 | | 0 |
| | | UNIT_MAX | 65535 |
| signed long | 32 | LONG_MIN | -2147483647 |
| | | LONG_MAX | 2147483647 |
| unsigned long | 32 | | 0 |
| | | ULONG_MAX | 4294967295 |
| signed long long | 64 | LLONG_MIN | -9,22337E+18 |
| | | LLONG_MAX | 9223372036854775807 |
| unsigned long long | 64 | | 0 |
| | | ULLONG_MAX | 1,84467E+19 |

Considere um número inteiro de 8 bits

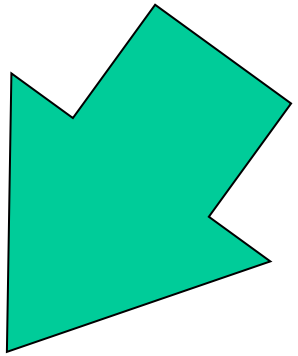
a) Faixa do número para inteiro não sinalizado
   $2^8 = 256$ .: Faixa de número é de 0 .. 255

b) Faixa do número para inteiro sinalizado (perco bit para sinal)
   $2^7 = 128$ .: Faixa de número é de -128 .. 127

# Adição

**Quando ocorre um overflow a ALU deve sinalizar para que este resultado não seja usado.**

**Um overflow pode ocorrer mesmo que não exista um carry.**



**Regra: se dois números são somados, e têm o mesmo sinal, ocorre um overflow se o resultado tiver sinal oposto.1**

```
 1001                1100
+0101               +0100
 1110 = -2          10000 = 0
```
(a) (-7) + (+5) (b) (-4) + (+4)

```
 0011                1100
+0100               +1111
 0111 = 7           11011 = -5
```
(c) (+3) + (+4) (d) (-4) + (-1)

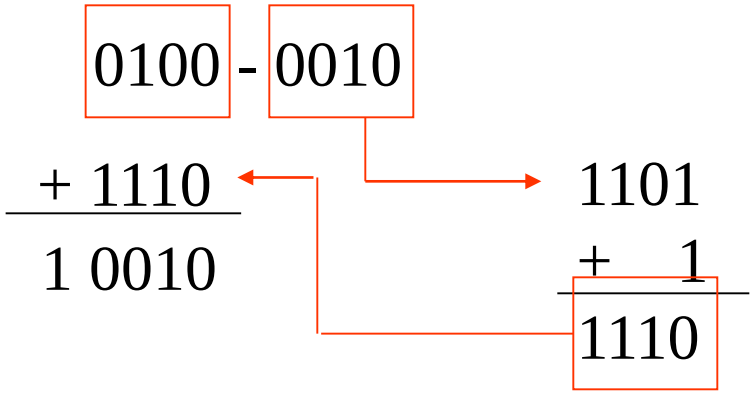
```
 0101                1001
+0100               +1010
 1001 = Overflow    10011 = Overflow
```
(e) (+5) + (+4) (f) (-7) + (-6)

ure 8.4 Addition of Numbers in Twos Comp
Representation

# Subtração

Para subtrair um número (subtraendo) de outro (minuendo), deve-se tomar o complemento de dois (negação) do subtraendo e soma-lo ao minuendo.

Exemplo:



```
  0010
 +1001
  1011 = -5
```

```
(a) M =  2 = 0010
    S =  7 = 0111
   -S =     1001
```

```
  0101
 +1110
 10011 = 3
```

```
(b) M =  5 = 0101
    S =  2 = 0010
   -S =     1110
```

---

```
  1011
 +1110
 11001 = -7
```

```
(c) M = -5 = 1011
    S =  2 = 0010
   -S =     1110
```

```
  0101
 +0010
  0111 = 7
```

```
(d) M =  5 = 0101
    S = -2 = 1110
   -S =     0010
```

---

```
  0111
 +0111
  1110 = Overflow
```

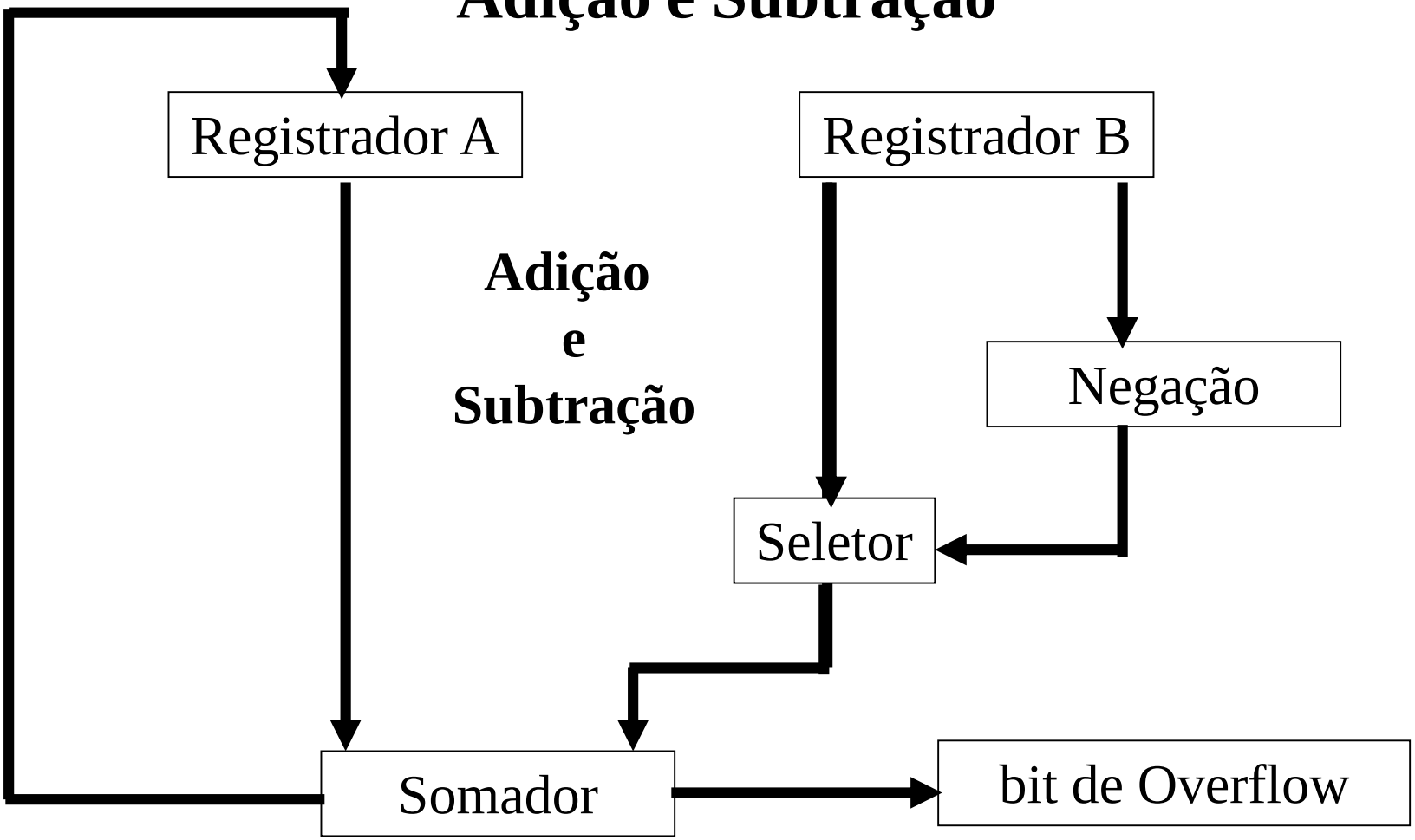
```
(e) M =  7 = 0111
    S = -7 = 1001
   -S =     0111
```

```
  1010
 +1100
 10110 = Overflow
```

```
(f) M = -6 = 1010
    S =  4 = 0100
   -S =     1100
```

Exemplos de subtração em completo de dois (m-s)

# Diagrama de blocos de um Hardware para Adição e Subtração



# Representação de em Ponto Flutuante

Exemplos:

$\pi = 3.141592654$

massa do Sol: $1.99 \times 10^{30}$ Kg

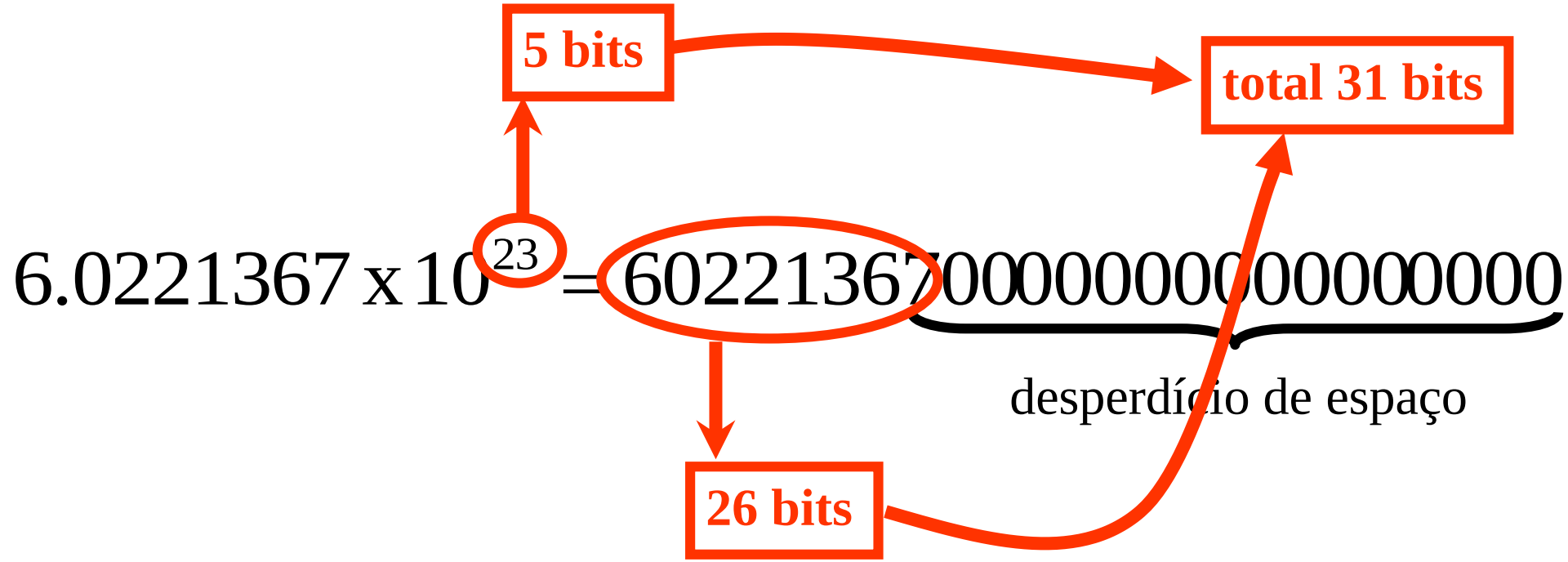
carga elementar do elétron: $1.60217738 \times 10^{-19}$ C

número de Avogrado: $6.0221367 \times 10^{23}$

$$2^{102} = 5.076 \times 10^{30}$$

**palavra com 102 bits !!!!**

**Arquitetura de computadores**




$2^{79} = 6.044629097... \times 10^{23}$

**palavra com 79 bits (no mínimo)!!!!**

.: representar números
Sobre uma base neste exemplo
Base 10 tem vantagens

# Notação Científica

$$602213670000000000000000 = 6.0221367 \text{ x } 10^{23}$$

$$0.000000000000000000160217738$$

$$= 1.60217738 \text{ x } 10^{-19}$$

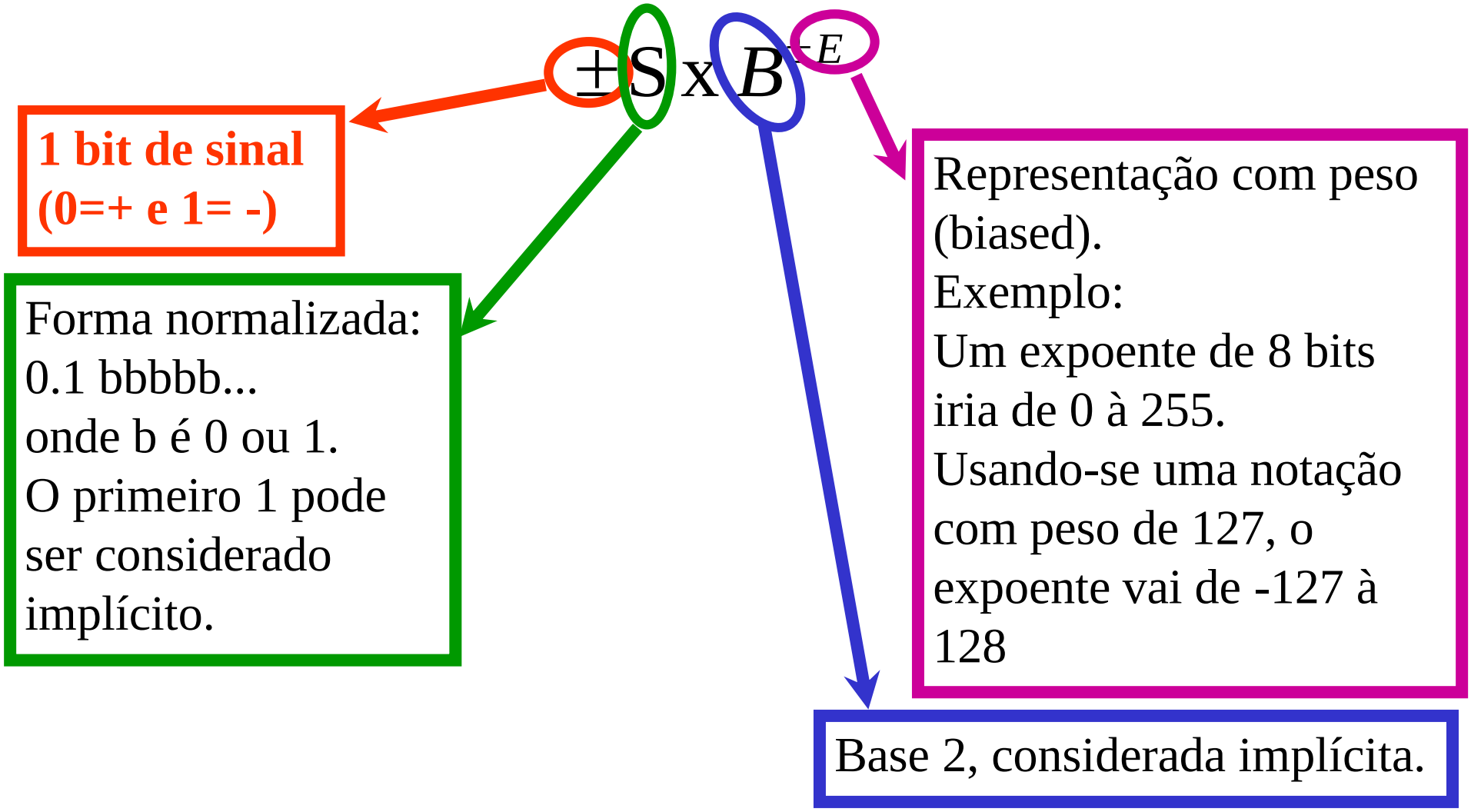
$$\pm S \text{ x } B^{\pm E}$$

**Significante ou mantissa**

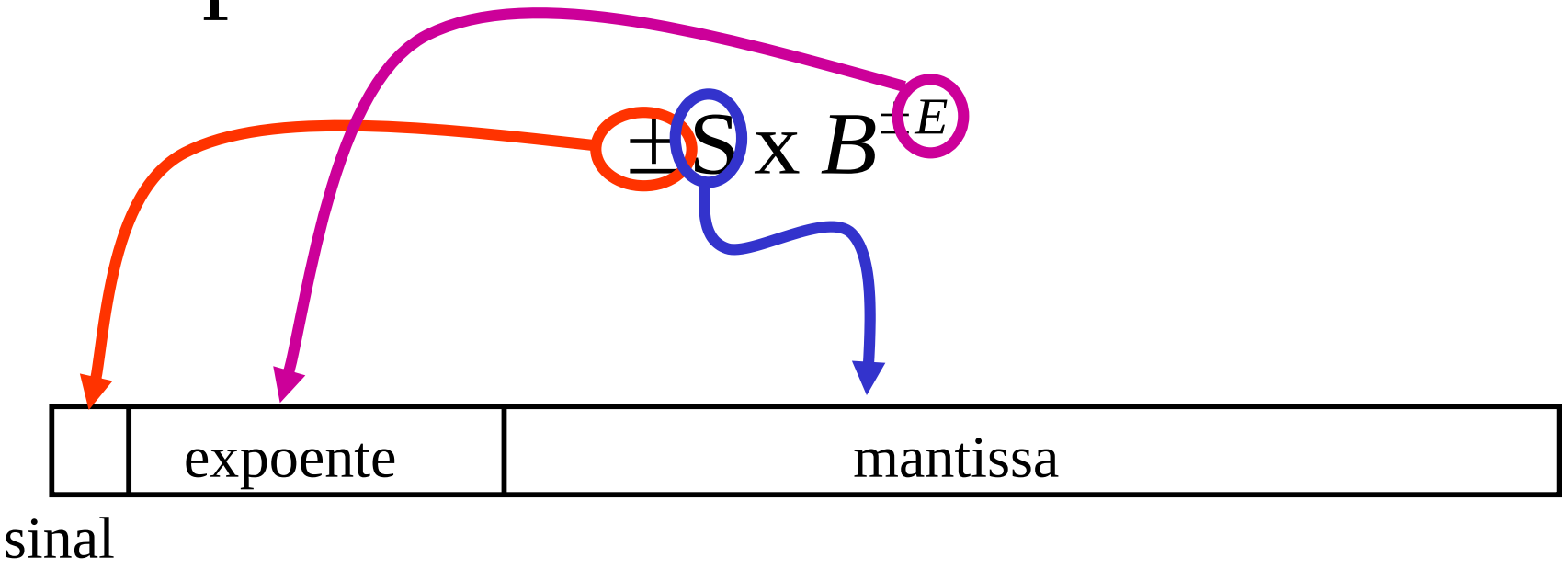
**Base**

**Expoente**

# Representação em binário



# Exemplo de formato:



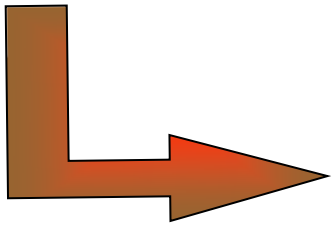

Para uma palavra de tamanho fixo:

- quanto maior o número de bits do expoente, maior (menor) o número a ser representado
- quanto maior o número de bits da mantissa, maior a precisão do número

# Exemplo:

| 1 bit | 8 bits (peso: 127) | 23 bits |
|---|---|---|

0 10010100 10111000000000000000000



0.1Mantissa Expoente - Peso

$$0.110111 \text{x}\, 2^{10010100-01111111}$$

$$0.110111 \text{x}\, 2^{10101}$$

$$0.110111 \text{x}\, 2^{10101} = \left(2^{-1} + 2^{-2} + 2^{-4} + 2^{-5} + 2^{-6}\right) \text{x}\, 2^{21}$$

$$= 0.859375 \text{x}\, 2^{21} = 1802240$$

**Arquitetura de computadores**

| 1 bit | 8 bits (peso: 127) | 23 bits |
|---|---|---|

**- total de números que podem ser representados:** $2^{32}$

**- menor número positivo que pode ser representado:**

$$\underbrace{0\ 00000000\ 00000000000000000000000}$$

$$0.1\text{x}\,2^{10000001} = 0.5\text{x}\,2^{-127} = 2.938735877\text{x}\,10^{-39}$$

**- maior número positivo que pode ser representado:**

$$\underbrace{0\ 11111111\ 11111111111111111111111}$$

$$\left(1-2^{-24}\right)\text{x}\,2^{128} \cong 3.402823668\,\text{x}\,10^{38}$$

| 1 bit | 8 bits (peso: 127) | 23 bits |
|---|---|---|

**- menor número negativo que pode ser representado:**

$$\underbrace{1\ 11111111\ 11111111111111111111111}$$

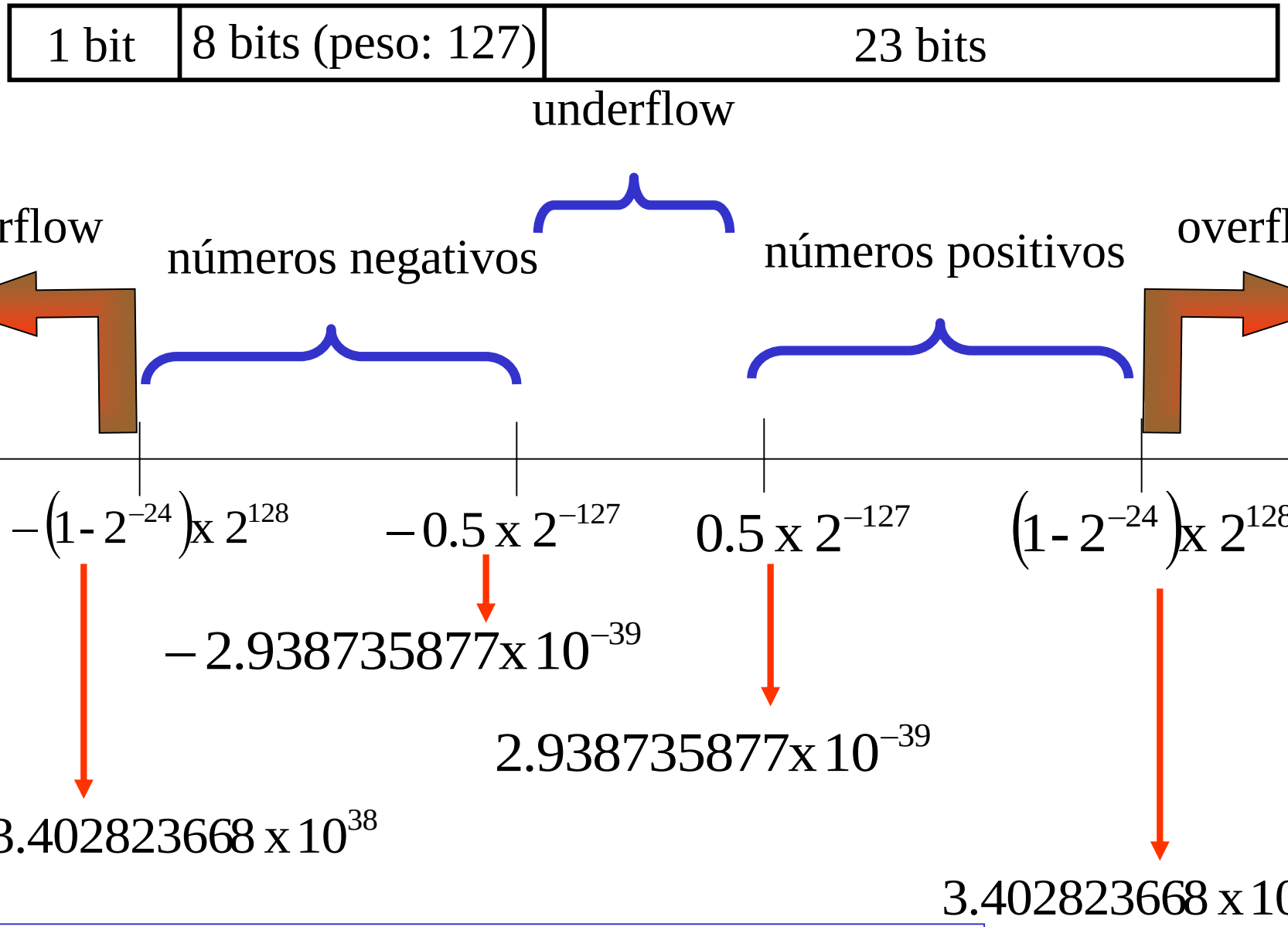
$$-\left(1-2^{-24}\right) \times 2^{128} \cong -3.402823668 \times 10^{38}$$

**- maior número negativo que pode ser representado:**

$$\underbrace{1\ 00000000\ 00000000000000000000000}$$

$$-0.5 \times 2^{-127} = -2.938735877 \times 10^{-39}$$

**Arquitetura de computadores**




**Note que esta representação não acomoda o zero.**

| 1 bit | 8 bits (peso: 127) | 23 bits |
|---|---|---|

**- espaçamento entre dois número positivo próximos de zero:**

$$\underbrace{0\ 0000000\ 00000000000000000000000}$$

$$0.5 \text{x}\ 2^{-127}$$

$$\underbrace{0\ 0000000\ 00000000000000000000001}$$

$$\left(0.5+2^{-24}\right) \text{x}\ 2^{-127}$$

$$\Delta = \left(0.5+2^{-24}-0.5\right) \text{x}\ 2^{-127} = 2^{-151}$$

$$= 3.503246 \text{x}\ 10^{-127}$$

| 1 bit | 8 bits (peso: 127) | 23 bits |
|---|---|---|

**- espaçamento entre dois número positivo próximos do overflow:**

$$\underbrace{0\ 1000000\ 11111111111111111111111}_{\left(1-2^{-24}\right)x\ 2^{128}}$$

$$\underbrace{0\ 1000000\ 11111111111111111111110}_{\left(1-2^{-23}\right)x\ 2^{128}}$$

$$\Delta = \left(1-2^{-24}-\left(1-2^{-23}\right)\right)x\ 2^{128}$$

$$\Delta = \left(1-2^{-24}-1+2^{-23}\right)x\ 2^{128}$$

$$\Delta = \left(-0.1x2^{-23}+2^{-23}\right)x\ 2^{128} = \left(0.9x2^{-23}\right)x\ 2^{128}$$

$$\Delta = 0.9x2^{105} = 3.6508337\ x\ 10^{31}$$

**Arquitetura de computadores**

**- espaçamento entre dois número positivo próximos de zero:**

$$\Delta = 3.503246 \text{x}\, 10^{-127}$$

**- espaçamento entre dois número positivo próximos do overflow:**

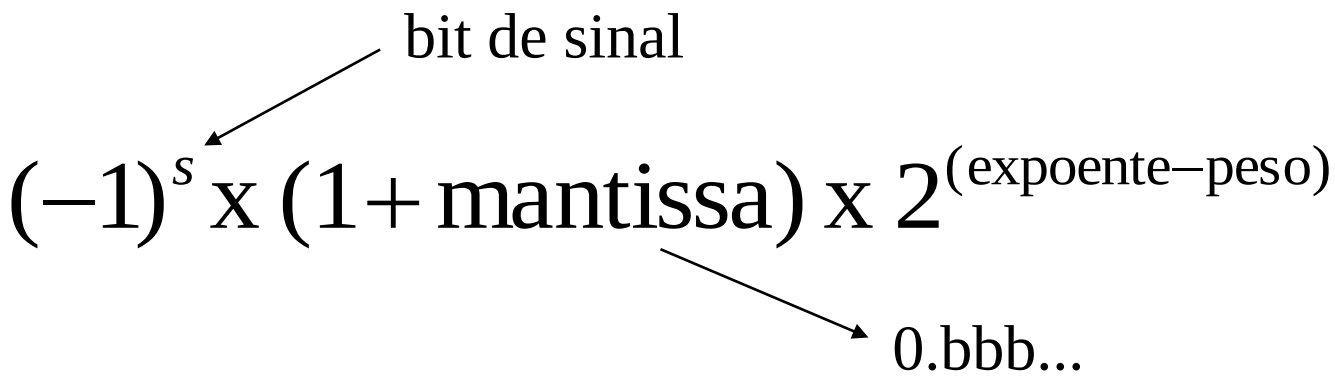
$$\Delta = 3.6508337 \,\text{x}\, 10^{31}$$

Os números não são igualmente espaçados, ficando mais próximos quanto mais próximos a origem

Cálculos que produzem resultados que não podem ser representados devem ser aproximados para o valor mais próximo que a notação possa representar

# Padrão IEEE 754

- Define formatos de 32 bits (simples) e 64 bits (duplo).
- A base implícita usada é 2.
- O expoente usa uma notação com peso, indo de -126 à +127 no formato simples e -1022 à 1023 no formato duplo.
- Um número  normalizado requer um bit 1 a esquerda do ponto binário, tomado com implícito.
- Algumas seqüências de bits são usadas para se representar valores especiais.





formato simples / float da linguagem C


formato duplo / double da linguagem C

# Exemplo

| | | |
|---|---|---|
| 1 | 10000001 | 01000000000000000000000 |

$$(-1)^{s} \text{ x } (1+\text{mantissa}) \text{ x } 2^{(\text{expoente}-127)}$$

$$(-1)^{1} \text{ x } (1+0.01) \text{ x } 2^{(129-127)}$$

$$(-1.01) \text{ x } 2^{2} = -(2^{0}+2^{-2}) \text{x } 2^{2}$$

$$-(2^{0}+2^{-2}) \text{x } 2^{2} = -1.25 \text{x } 4 = -5$$

# Arquitetura de computadores

## Interpretação dos números conforme padrão IEEE 754

| | Precisão simples (32 bits) | | | | Precisão dupla (64 bits) | | | |
|---|---|---|---|---|---|---|---|---|
| | **Sinal** | **Expoente polarizado** | **Fração** | **Valor** | **Sinal** | **Expoente polarizado** | **Fração** | **Valor** |
| Zero positivo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Zero negativo | 1 | 0 | 0 | –0 | 1 | 0 | 0 | –0 |
| Infinito positivo | 0 | 255 (todos 1s) | 0 | $\infty$ | 0 | 2047 (todos 1s) | 0 | $\infty$ |
| Infinito negativo | 1 | 255 (todos 1s) | 0 | $-\infty$ | 1 | 2047 (todos 1s) | 0 | $-\infty$ |
| NaN silencioso | 0 ou 1 | 255 (todos 1s) | $\neq 0$ | NaN | 0 ou 1 | 2047 (todos 1s) | $\neq 0$ | NaN |
| NaN sinalizador | 0 ou 1 | 255 (todos 1s) | $\neq 0$ | NaN | 0 ou 1 | 2047 (todos 1s) | $\neq 0$ | NaN |
| Diferente de zero, normalizado positivo | 0 | $0 < e < 255$ | f | $2^{e-127}(1,f)$ | 0 | $0 < e < 2047$ | f | $2^{e-1023}(1,f)$ |
| Diferente de zero, normalizado negativo | 1 | $0 < e < 255$ | f | $-2^{e-127}(1,f)$ | 1 | $0 < e < 2047$ | f | $-2^{e-1023}(1,f)$ |
| Não-normalizado positivo | 0 | 0 | $f \neq 0$ | $2^{e-126}(0,f)$ | 0 | 0 | $f \neq 0$ | $2^{e-1022}(0,f)$ |
| Não-normalizado negativo | 1 | 0 | $f \neq 0$ | $-2^{e-126}(0,f)$ | 1 | 0 | $f \neq 0$ | $-2^{e-1022}(0,f)$ |

**Arquitetura de computadores**

Exercícios

1) Converta os números de binário para decimal:
   a)10011 b)111011.101 c)111.11001

2) Qual é a melhor forma de representação números inteiros: sinal de magnitude ou complemento de dois.

3) Demonstre a faixa de funcionamento dos números inteiros sendo sinalizado e não sinalizado com:
   a) 12 bits b)20 bits c)24 bits

4) Demonstre os complemento de dois dos números:
   a) 23 (para 8 bits)
   b) 127 (para 8 bits)
   c) 0 (para 8 bits)
   d) 128 (para 8 bits)
   e) 3000(para 16 bits)

**Arquitetura de computadores**

Exercícios

5) Faça contas com os números inteiros (converte em binário) e indique se ocorreu ou não overflow

a) 4 + 2 (8 bits)

b) 120 + 8 (8 bits)

c) 120 – 5 (8 bits)

d) 50 – 50 (8 bits)

d) 50 – 51 (8 bits)

f) 1000 – 500 (12 bits)

6) Converta os números binários em ponto flutuante conforme o padrão IEEE 754

a)

| | | |
|---|---|---|
| 1 | 10111001 | 01011000000000000000000 |

b)

| | | |
|---|---|---|
| 0 | 01111000 | 01100000000000000000000 |