Universidade Federal de Campina Grande

Unidade Acadêmica de Sistemas e Computação

Introdução à Computação

A Informação e sua Representação (Parte II)

Prof.ª Joseana Macêdo Fechine Régis de Araújo

joseana@computacao.ufcg.edu.br

Carga Horária: 60 horas

A Informação e sua Representação

- Em um computador são armazenados e processados apenas dados e instruções.
- Um computador executa operações sobre dados numéricos (os números) ou alfabéticos (letras e símbolos).
- É preciso definir uma forma de representar os dados, codificados em uns e zeros, que possam ser interpretados pelo computador, de forma correta e eficiente (com bom desempenho e pouco consumo de memória).

A Informação e sua Representação

Os dados podem ser:

- **Alfabéticos**
  - letras, números e símbolos (codificados em ASCII e EBCDIC)
- **Numéricos**
  - ponto fixo, números inteiros
  - ponto flutuante (números reais ou fracionários)
  - BCD (representação decimal codificada em binário)
- **Lógicos**
  - Variáveis que possuem apenas dois valores para representação (FALSO e VERDADEIRO).

Representação de Números Inteiros

- Todos os dados numéricos são representados em um computador como uma seqüência de 0s e 1s.
- Os números podem ser positivos ou negativos. As operações aritméticas, em particular a subtração, podem originar resultados negativos.
- Um aspecto primordial a ser definido seria então como representar o sinal.
- **Como é que um computador sabe que um dado número é negativo?**

Representação de Números Inteiros

- A resposta a esta pergunta é que isso depende da convenção usada na representação de números.

- As convenções mais usuais são as seguintes :
  - **Representação de grandeza com sinal (sinal e magnitude)**
  - **Representação em complemento de 2**

Outras formas de representação:

Complemento de 1: para negar o valor de um número deve-se inverter os bits do sinal (obsoleta) e **Excesso de 2^{m-1}**: representação do número é dada pela soma de seu valor absoluto com 2^{m-1}. Exemplo: Um sistema de 8 bits é chamado de excesso de 128 e um número é armazenado com seu valor real somado a 128. Ex.:-3=01111101_2 (-3+128=125)

Representação de grandeza com sinal

- O bit mais significativo representa o sinal:
  - **0** (indica um **número positivo**)
  - **1** (indica um **número negativo**)
- Os demais bits representam a **grandeza (magnitude)**.

sinal	magnitude

- O valor dos bits usados para representar a magnitude independe do sinal (sendo o número positivo ou negativo, a representação binária da magnitude será a mesma).

Exemplos: (8 bits)

- $00101001_2 = +41_{10}$
- $10101001_2 = -41_{10}$

Representação de grandeza com sinal

- **Exemplos: (8 bits)**

Valor decimal	Valor binário com 8 bits (7 + bit de sinal)
+9	00001001
-9	10001001
+127	01111111
-127	11111111

Assim, uma representação em binário com **n** bits teria disponível para a representação do número ***n-1*** bits (o bit mais significativo representa o sinal).

Representação de grandeza com sinal

- Apresenta uma grande **desvantagem**: ela exige um grande número de testes para se realizar uma simples soma de dois números inteiros.

- Requer que na UAL existam dois circuitos distintos para a adição e a subtração.

- Existem **duas representações** para o zero.

Representação em complemento de 2

- **Representação de números inteiros positivos**
  - igual à representação de grandeza com sinal.
- **Representação de números inteiros negativos**
  - mantém-se os bits menos significativos da direita para a esquerda até à ocorrência do primeiro bit igual a 1 (inclusive), sendo os bits restantes complementados de 1.
  - Esta operação equivale a realizar: complemento de 1 + 1.

Exemplo : (8 bits)

$00001100_2 = +12_{10}$

$11110100_{c2} = -12_{10}$

Exemplo : (8 bits)

$00101001_2 = +41_{10}$

$11010111_{c2} = -41_{10}$

Representação em complemento de 2

- **Exemplo:** Números inteiros codificados em **binário de 8 bits** em um sistema que utiliza complemento de 2:

(-128, -127, ..., -2. -1, 0, +1, +2,..., +127)

{10000000, 10000001, ..., 11111110, 11111111, 00000000, 00000001, 00000010, ..., 01111111}

- Bit mais significativo ➡ informação de sinal

(0 = positivo e 1 = negativo)

Representação em complemento de 2

- Requer **um só circuito** (somador) para fazer a adição e a subtração.
- Há apenas uma representação para o valor **0** (disponibilidade para **mais uma representação)** - mais um número negativo pode ser representado (para 8 bits, pode-se representar o número $-128_{10} \Rightarrow 10000000_2$) .
- A quantidade de números positivos é **diferente** da quantidade de números negativos.

Representação de Números Inteiros

Exemplo:

Escreva os números decimais abaixo nas seguintes representações: sinal e magnitude; representação em complemento de 1; representação em complemento de 2 e excesso de 128 (utilizando 8 bits, se existir representação).

a) -1

b) –20

c) –127

d) –128

Representação de Números Inteiros

Números negativos de 8 bits expressos em 4 sistemas diferentes

N (decimal)	N (binário)	-N (sinal-magnitude)	-N (comple-mento de 1)	-N (comple-mento de 2)	-N (excesso de 128)
1	00000001	10000001	11111110	11111111	01111111
2	00000010	10000010	11111101	11111110	01111110
3	00000011	10000011	11111100	11111101	01111101
4	00000100	10000100	11111011	11111100	01111100
10	00001010	10001010	11110101	11110110	01110110
20	00010100	10010100	11101011	11101100	01101100
100	01100100	11100100	10011011	10011100	00011100
127	01111111	11111111	10000000	10000001	00000001
128		Não existe represen-tação	Não existe represen-tação	10000000	00000000

Representação de Números Inteiros

- Haverá sempre um padrão de bits a mais ou a menos, não importa qual a representação escolhida.
- O padrão de bits extra pode ser usado como –0, como o menor número negativo da representação, ou algo assim, mas, independentemente de como esse padrão de bits for usado, ele poderá ser um estorvo.

Representação de Números Reais

- Em alguns tipos de cálculo, a faixa de variação dos números envolvidos é muito grande.

- **Exemplo**:
  - 1) Massa do elétron - da ordem de 9×10^{-28} gramas
  - 2) Massa do Sol - aproximadamente igual a 2×10^{33} gramas
  - Faixa de variação: $> 10^{60}$
  - Exemplo de representação (34 dígitos à esquerda do ponto decimal e 28 dígitos à direita do mesmo)

1) 0000000000000000000000000000000000.000000000000000000000000009

2) 2000000000000000000000000000000000.000000000000000000000000000

- **Como representar esses números no computador?**

Representação de Números Reais

- Forma usual de representação de números reais: **parte inteira, vírgula (ou ponto), parte fracionária.**
- Esta representação, embora cômoda para cálculos no papel, não é adequada para processamento no computador.
- **Exemplo: 45,724**

Representação de Números Reais

- O número **45,724** pode ser expresso como:
  - **$45{,}724 \times 10^{0}$**
  - **45724×10^{-3}**
  - **$0{,}45724 \times 10^{2}$**

- É necessário o uso de um sistema de representação de números no qual a faixa de variação dos números seja independente do número de dígitos significativos dos números representados.

Representação em Ponto Flutuante

- Uma maneira de separar a faixa de variação dos números de sua precisão consiste em representá-lo na notação científica.

$$n = f \times 10^{e}$$

  - ***f*** - fração ou significando (ou mantissa)
  - ***e*** - expoente (inteiro positivo ou negativo)

- Qualquer número (inteiro ou fracionário) pode ser expresso no formato **número x base$^{\text{expoente}}$**, podendo-se variar a posição da vírgula e o expoente.

- Denominação (computacional): **representação em ponto flutuante** (o ponto varia sua posição, modificando, em conseqüência, o valor representado).

Representação em Ponto Flutuante

- Representação pode variar ("**flutuar**") a posição da vírgula, ajustando a potência da base.

- **Exemplos**:
  - $3{,}14 = 0{,}314 \times 10^{-1} = 3{,}14 \times 10^{0}$
  - $0{,}000001 = 0{,}1 \times 10^{-5} = 1{,}0 \times 10^{-6}$
  - $1941 = 0{,}1941 \times 10^{4} = 1{,}941 \times 10^{3}$

- A **faixa de variação** dos números é determinada pela quantidade de dígitos do expoente e a **precisão** é determinada pela quantidade de dígitos do significando.

Representação em Ponto Flutuante

- **Forma normalizada:** usa um único dígito antes da vírgula, diferente de zero (*).
- Na representação computacional de números em ponto flutuante, a representação normalizada é, em geral, melhor que a não-normalizada.
  - **Forma normalizada**: só existe uma forma de representar um número.
  - **Forma não normalizada**: um mesmo número pode ser representado de diversas maneiras.

(*) Padrão IEEE 754 para números em ponto flutuante – **significando normalizado** – começa com um bit 1, seguido de um ponto (vírgula) binário e pelo resto do significando (número = ± 1,_ _ ... x 2^{exp})

Mantissa normalizada - começa com o ponto (vírgula) binário seguido por um bit 1 e pelo resto da mantissa (bit antes da vírgula igual a zero).

Representação em Ponto Flutuante

Ilustração:

- No sistema binário:
  - **110101** = **110,101x2^3** = **1,10101x2^5** = **0,0110101x2^7**
- Números armazenados em um computador - os expoentes serão também gravados na base dois
  - Como $3_{10} = 11_2$ e $7=111_2$
  - $110,101 \times (10)^{11} = 1,10101 \times (10)^{101} = 0,0110101 \times (10)^{111}$
- Representação normalizada - há apenas um "1" antes da vírgula
  - **Exemplo**: **1,10101x(10)^{101}**

Representação em Ponto Flutuante

Algumas definições:

- No número **1,10101x**$(10)^{\mathbf{101}}$:
  - **1,10101 = significando**
  - **101 = expoente**

- **OBS:**
  - a base binária não precisa ser explicitada (o computador usa sempre esta)
  - O "1" antes da vírgula, na representação normalizada – se esta for adotada, também pode ficar implícito, economizando um bit ("**bit escondido**")

Armazenamento de *Floats*

- **Na organização/arquitetura do computador, deve-se definir**:
  - Número de bits do significando (precisão, ***p*** ou ***f***)
  - Número de bits do expoente (***e***)
  - Um bit ("**0**" para + e "**1**" para -) de sinal (tipicamente o primeiro, da esquerda)

Armazenamento de *Floats*

- **Ilustração (8 bits)**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sinal	Expoente (+/-)			Significando			

- Sinal do número: 0 = + e 1 = -
- Expoentes: 8 combinações possíveis
  - **OBS**: Não seguem aritmética normal
    (p.ex.: Utiliza notação em excesso)

Armazenamento de *Floats*

000	Caso especial
001	Expoente –2
010	Expoente -1
011	Expoente 0
100	Expoente 1
101	Expoente 2
110	Expoente 3
111	Caso especial

Abaixo de zero

(***bias*** **= polarização**)

Acima de zero

Armazenamento de *Floats*

Exemplo: Realize as conversões abaixo:

- $6{,}75_{10} = (\ ?\ )_2$ (ponto flutuante, com 8 bits)
- $1110\ 1001_2$ (ponto flutuante, com 8 bits) = $(\ ?\ )_{10}$

Armazenamento de *Floats*

Solução:

- $6{,}75_{10} = 110{,}11_2 = 1{,}1011 \times 2^2$

  sinal: 0

  expoente: $2_{10} + 3_{10} = x_{10}$, $x_{10} = 5_{10} = 101_2$

  significando: 1011

  Número (ponto flutuante, com 8 bits):

  01011011_2

Armazenamento de *Floats*

Solução:

- **1110 1001$_2$** (ponto flutuante, 8 bits)

  sinal: 1

  expoente: $110_2 = 6_{10}$, $x_{10} + 3_{10} = 6_{10}$,

  $x_{10} = 3_{10}$

  significando: 1001_2 =

  Número:(negativo) $1{,}1001_2 \times 2^3 = 1100{,}1_2 = \mathbf{-12{,}5_{10}}$

Armazenamento de *Floats*

Ainda os expoentes na ilustração (8 bits) ...

- **Maior número positivo** (lembre do bit escondido):
  0 110 1111 = + 2^3 x 1,1111 = 1111,1 = **15,5 decimal**

- **Menor número positivo** (lembre do bit escondido):
  0 001 0000 = + 2^{-2} x 1,0000 = 0,01 ou **0,25 decimal**

Armazenamento de *Floats*

Combinações especiais dos expoentes na ilustração...

- **000** – representação NÃO normalizada
  - Significando passa a ser 0,_ _ _ ...
  - Expoente (000) = -2
  - **Menor número positivo passa a ser**
  - 0 000 0001 = 2^{-2} x 0,0001 = 2^{-2} x 2^{-4} = 2^{-6} = **0,015625 decimal**

A norma IEEE prevê o *underflow* gradual (a mantissa deixa de ser normalizada), permitindo obter números bem mais próximos de zero.

Armazenamento de *Floats*

Ainda as combinações especiais...

- **Normalização não permite representar zero!**
- **000** – representação NÃO normalizada
  - 00000000 = + 0 decimal
  - 10000000 = - 0 decimal (iguais em comparações)
- **111** - representações de infinito
  - 01110000 = + infinito
  - 11110000 = - infinito
  - 11111000 = indeterminação
  - Outras combinações 11111_ _ _ = *Not A Number* (**NANs**)

O Padrão IEEE 754 para Números em Ponto Flutuante

- Até meados dos anos 1980, cada fabricante de computador tinha seu próprio formato para representar números em ponto flutuante.
- **Solução**: criação do **Padrão 754** (IEEE 1985).
- O Padrão IEEE 754 procurou uniformizar a maneira como as diferentes máquinas representam os números em ponto flutuante, bem como devem operá-los.
- O padrão IEEE 754 para ponto (vírgula) flutuante é a representação mais comum para **números reais** em computadores de hoje, incluindo PC's compatíveis com Intel, Macintosh, e a maioria das plataformas Unix/Linux.

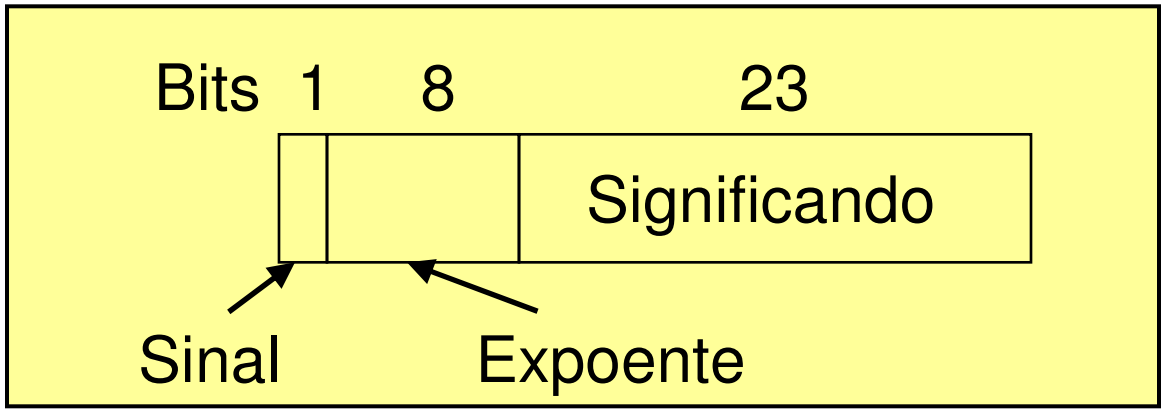
O Padrão IEEE 754 para Números em Ponto Flutuante

O padrão IEEE 754 define três formatos:

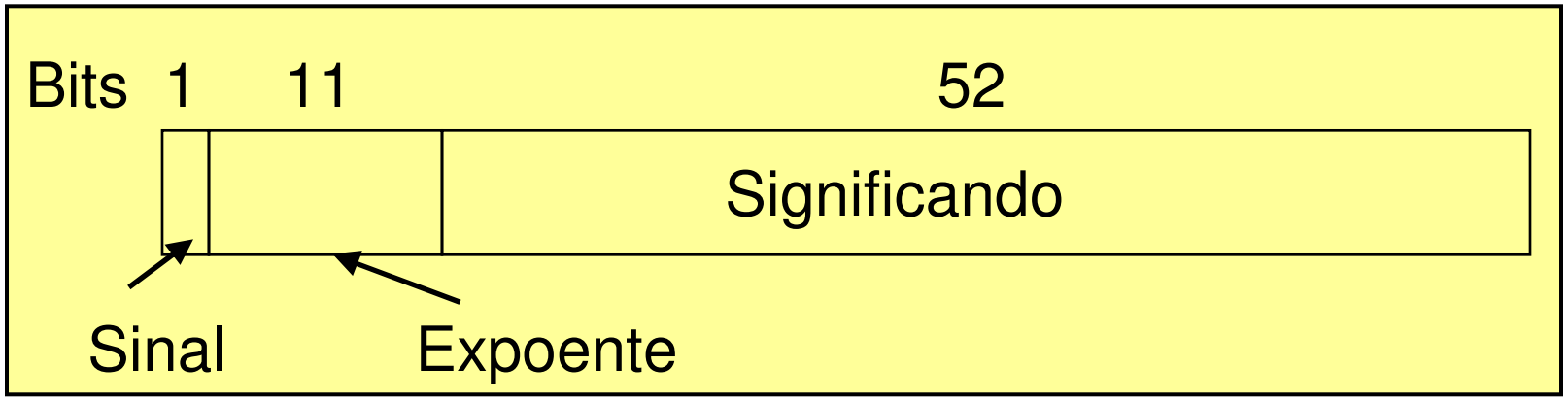
- **Precisão simples** (32 bits)
- **Precisão dupla** (64 bits)
- Precisão estendida (80 bits)

- Os formatos de precisão simples e precisão dupla usam a base 2 para o significando e a notação em excesso para o expoente.

O Padrão IEEE 754 para Números em Ponto Flutuante



Precisão simples



Precisão dupla

O Padrão IEEE 754 para Números em Ponto Flutuante

Precisão	Sinal	Expoente(+/-)	Significando
Simples (32bits)	**1** [bit31]	**8** [bits30-23]	**23** [bits22-00]
Dupla (64 bits)	**1** [bit63]	**11** [bits62-52]	**52** [bits51-00]

- **Sinal**: 0 = + e 1 = -
- **Combinações**: Sinal + Expoente + Significando
- **Notação em excesso de 127** (bit de polarização): precisão simples.
- **Notação em excesso de 1023** (bit de polarização): precisão dupla.

O Padrão IEEE 754 para Números em Ponto Flutuante

Ilustração

- Expoentes na **precisão simples** c/256 combinações
  - **01111111** (127_{10}) = expoente zero (*bias* = polarização)
  - **00000001** = menor expoente = –126 (abaixo de zero)
  - **11111110** = maior expoente = +127 (acima de zero)

O Padrão IEEE 754 para Números em Ponto Flutuante

Exemplo: Realize as conversões abaixo:

- $10{,}875_{10}$ = (?)$_2$ (IEEE 754, com 32 bits)
- $11000001110100000000000000000000_2$ (IEEE 754, com 32 bits) = (?)$_{10}$

O Padrão IEEE 754 para Números em Ponto Flutuante

Solução:

- $10{,}875_{10} = 1010{,}111_2 = 1{,}010111 \times 2^3$

  sinal: 0

  expoente: $3_{10} + 127_{10} = x_{10}$, $x_{10} = 130_{10} = 10000010_2$

  significando: $01011100000000000000000_2$

  Número (IEEE 754, com 32 bits):

  $$01000001001011100000000000000000_2$$

O Padrão IEEE 754 para Números em Ponto Flutuante

Solução:

- **$11000001110100000000000000000000_2$**

  (IEEE 754, com 32 bits)

  sinal: 1

  expoente: $10000011_2 = 131_{10}$, $x_{10} + 127_{10} = 131_{10}$,

  $x_{10} = 4_{10}$

  significando: $10100000000000000000000_2$ =

  Número:(negativo) $1{,}101_2 \times 2^4 = 11010_2 = \mathbf{-26_{10}}$

O Padrão IEEE 754 para Números em Ponto Flutuante

Tipos numéricos IEEE 754

Normalizado	±	**0 < Exp < Max**	**Qualquer configuração de bits**
Não-Normalizado	±	**000 ... 0**	**Qualquer configuração de bits diferente de zero**
Zero	±	**000 ... 0**	**000 ...0**
Infinito	±	**111 ... 1**	**000 ..0**
Not a Number	±	**111 ... 1**	**Qualquer configuração de bits diferente de zero**

Bit de sinal

O Padrão IEEE 754 para Números em Ponto Flutuante

Ilustração

- Expoentes na **precisão simples** c/256 combinações
  - **00000000**
    - sinal=1 e significando = 0...0 : -zero
    - sinal=0 e significando = 0...0 : +zero
  - **11111111**
    - sinal=1 e significando = 0...0 : -infinito
    - sinal=0 e significando = 0...0 : +infinito
    - sinal=1 e significando =10...0: indeterminado
    - c/outras combinações: NAN

O Padrão IEEE 754 para Números em Ponto Flutuante

Ilustração

- **Menor número positivo** (lembre do bit escondido e não normalizada)
  - **0 00000000 00....01** = $2^{-126} \times 2^{-23} = 2^{-149}$
- **Maior número positivo** (lembre do bit escondido)
  - **0 11111110 11...11** = $2^{127} \times (2-2^{-23})$

O Padrão IEEE 754 para Números em Ponto Flutuante

Ilustração

- Expoentes na **precisão dupla**
  - **01111111111** (1023_{10}) = expoente zero (*bias* = polarização)
  - **00000000001** = menor expoente = –1022 (abaixo de zero)
  - **11111111110** = maior expoente = +1023 (acima de zero)
- **Menor número positivo** (lembre do bit escondido e não normalizada)
  - **0 00000000000 00...01** = $2^{-1022} \times 2^{-52} = 2^{-1074}$
- **Maior número positivo** (lembre do bit escondido)
  - **0 11111111110 11...11** = $2^{1023} \times (2-2^{-52})$

O Padrão IEEE 754 para Números em Ponto Flutuante

Quadro Resumo - IEEE 754

Precisão	Não normalizado	Normalizado	Decimal
Simples	$\pm 2^{-149}$ a $(1-2^{-23}) \times 2^{126}$	$\pm 2^{-126}$ a $(2-2^{-23}) \times 2^{127}$	$\pm \sim 10^{-44.85}$ a $\sim 10^{38.53}$
Dupla	$\pm 2^{-1074}$ a $(1-2^{-52}) \times 2^{1022}$	$\pm 2^{-1022}$ a $(2-2^{-52}) \times 2^{1023}$	$\pm \sim 10^{-323.3}$ a $\sim 10^{308.3}$

Representação em Ponto Flutuante

- A representação em ponto flutuante tem limites de **alcance** e de **precisão**.
- O **alcance** é limitado pelo número de bits do **expoente**.
- A **precisão** é determinada pelo número de bits do **significando**.

Representação em Ponto Flutuante

- Ocorre ***overflow*** quando o valor absoluto do dado a ser representado excede a capacidade de representação, porque o número de bits do expoente (neste caso, positivo) é insuficiente para representar o dado.

- Ocorre ***underflow*** quando o valor absoluto do dado a ser representado é tão pequeno que fica menor que o menor valor absoluto representável.

- No caso de **imprecisão**, a normalização permite que o dado seja representado, porém com perda de precisão.

Representação de Números Decimais Codificados em Binário (BCD)

- A representação de números reais em ponto flutuante é perfeitamente adequada para fazer cálculos matemáticos, científicos, etc.
- Na representação em ponto flutuante pode-se ter perda de precisão do número representado ou mesmo haverá números que não podem ser representados por *overflow*.
- Para representação de números em que é necessário manter precisão até o último algarismo, não é admissível erro por aproximação.
- **Solução**: usar a representação **BCD** ou *Binary Coded Decimal* (Decimal Representado em Binário).

Representação de Números Decimais Codificados em Binário (BCD)

- A idéia do BCD é representar, em binário, cada algarismo de forma que o número original seja integralmente preservado.

- A codificação BCD não possui extensão fixa, possibilitando representar números com precisão variável - quanto maior o número de bits, maior será a precisão.

Representação de Números Decimais Codificados em Binário (BCD)

Tabela de Representação dos Números Decimais em BCD

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Continua ...

Representação de Números Decimais Codificados em Binário (BCD)

Tabela de Representação dos Números Decimais em BCD

Decimal	BCD
10	Inválido
11	Inválido
12	Inválido
13	Inválido
14	Inválido
15	Inválido

- **Exemplo: 239_{10} = (?) BCD**
  - $2 = 0010_2$
  - $3 = 0011_2$ e
  - $9 = 1001_2$, logo: **239 = 001000111001 (BCD).**

Representação de Números Decimais Codificados em Binário (BCD)

- A codificação de um dígito em BCD requer 4 bits.
- Como a utilização de apenas 4 bits por byte não é eficiente, normalmente são armazenados 2 dígitos BCD em um só byte. Esta representação é chamada **BCD comprimido ou compactado** ("***packed BCD***").
- Exemplo: **14239_{10} = (?) BCD**

1	42	39	número decimal
xxxx0001	**01000010**	**00111001**	representação BCD comprimido
a+2	a+1	a	endereço

Representação de Números Decimais Codificados em Binário (BCD)

- Entre os algarismos sem código válido em decimal (códigos representativos dos valores decimais de 10 a 15), é comum utilizar alguns deles para indicar o sinal do número.

- Há sistemas que adotam a seguinte convenção para o sinal dos números representados em BCD:
  - 1100 → representa o sinal positivo (“+”)
  - 1101 → representa o sinal negativo (“-”)

Representação de Números Decimais Codificados em Binário (BCD)

- Com nesta representação ainda há um desperdício de códigos; como BCD usa 4 bits (16 representações possíveis) para representar 10 algarismos, 6 (ou 4) códigos não são utilizados.

- Portanto, essa representação é menos eficiente em relação à utilização dos recursos do computador que a **representação em ponto flutuante**.

A informação e sua Representação

Observações – Representação em Ponto Fixo

- **Esse método consiste na determinação de uma posição fixa para a vírgula** (ou ponto).
- Todos os valores representados em ponto fixo para uma determinada operação possuem a mesma quantidade de algarismos inteiros, bem como a mesma quantidade de algarismos fracionários.
  - **Exemplo**: 1101,101 1110,001 0011,110

A informação e sua Representação

Observações – Representação em Ponto Fixo

- **As posições mais adotadas para a vírgula são**:
  - Na extremidade esquerda do número – nesse caso, o número é totalmente fracionário;
  - Na extremidade mais a direita do número – nesse caso, o número é inteiro.

- Em qualquer desses casos, no entanto, a vírgula fracionária não estará fisicamente representada na memória; sua posição é determinada na definição da variável, realizada pelo programador (ou pelo compilador), e o sistema memoriza essa posição, mas não a representa fisicamente.

A informação e sua Representação

Observações – Representação em Ponto Fixo

- Na maioria das linguagens de programação e nos sistemas de computação (e os compiladores da maior parte das linguagens de programação) emprega-se a representação de números em ponto fixo para indicar apenas valores inteiros (a vírgula fracionária é assumida na posição mais à direita do número); **números fracionários são, nesses casos, representados apenas em ponto flutuante**.

- Exemplos de tipos de dados na linguagem Java:

Tipo de dado	Representação interna
int	Ponto fixo (inteiro)
float	Ponto flutuante (real)