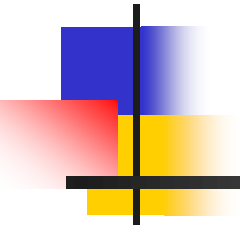
Cálculo Numérico



Conceitos Básicos

Profs.: Bruno C. N. Queiroz
J. Antão B. Moura
Ulrich Schiel
Maria Izabel C. Cabral

DSC/CCT/UFCG

Princípios usados em CN

- Comuns à análise matemática, C&T

1. Iteração ou aproximação sucessiva

- **Partindo-se de solução aproximada, inicial, repetem-se mesmas ações/processos para refinar solução inicial**
- OBS: para evitar trabalho sem fim (e de graça), deve-se determinar se a iteração **converge** (nem sempre é o caso...) e **condições de parada**

Princípios usados em CN

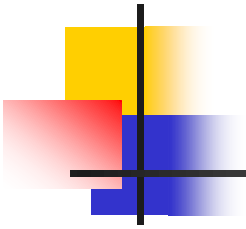
2. Discretização

- **Na resolução de problemas contínuos (aqueles definidos matematicamente com uma passagem ao limite), inverte-se a passagem ao limite, discretizando o problema**
- Ex: $\int e^{x^2} dx$ ~ **Σ...**

Princípios usados em CN

3. Aproximação

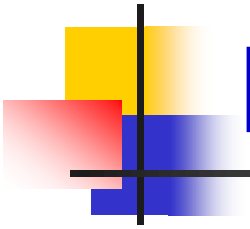
- **Substituir uma função ou modelo por outro que ofereça comportamento (de interesse) semelhante, mais simples de manipular**
  - f(x) ⇨ g(x)
- Ex: assíntotas ilustram comportamento "no limite" de uma função (complexa) de interesse



Princípios usados em CN

4. Transformação

- Dado um problema P, desmembra-se P em dois problemas mais simples de resolver, P1 e P2
  - **Área de um trapézio por retângulo (P1) e triângulos (P2)**



Princípios usados em CN

5. Divisão e Conquista

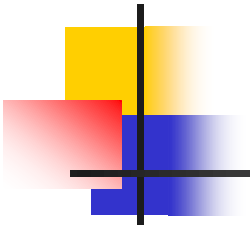
- Resolver um problema P, por partes ou etapas
  - **Exemplo anterior (área do trapézio)**
  - **Aulas nesta disciplina de CN**

Sistemas de numeração

- Representação não posicional
  - *romanos*
    - MDCCCXLIX e MMCXXIV
    - Como seria MDCCCXLIX + MMCXXIV ?
- Representação semi-posicional
  - *hebraicos*
    - 1= א (aleph), 2= ב (beth), 10= י (yod),
      100= ק(kuph), 11= יא, 101= קא 15=טו (9+6)

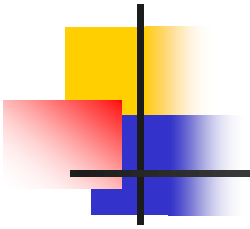
Sistemas de numeração

- *alemão*
  - *Vinte e um = ein-und-zwanzig*
- *francês*
  - Noventa = quatre-vingt-dix



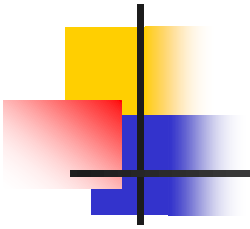
Sistemas de numeração

- Representação posicional
  - Base decimal (10)
    - 10 dígitos disponíveis [0,1,2, ... ,9]
    - "Posição" indica potência positiva de 10
    - $5432 = 5x10^3 + 4x10^2 + 3x10^1 + 2x10^0$



Sistemas de numeração

- Representação de inteiros
  - Base binária (2)
    - 2 "**bits**" disponíveis [0,1]
    - "Posição" indica potência positiva de 2
    - 1011 na base 2 = $1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$ = 8+0+2+1 = 11 na base decimal
    - Ou, melhor $1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$ = 1 + 2(1+2(0+2(1))) = 11



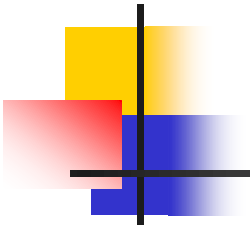
Sistemas de numeração

- Representação de números fracionários
  - Base decimal (10)
    - "Posição" da parte inteira indica potência positiva de 10
    - Potência negativa de 10 para parte fracionária
    - $54{,}32 = 5\text{x}10^{1} + 4\text{x}10^{0} + \mathbf{3\text{x}10^{-1} + 2\text{x}10^{-2}}$

Sistemas de numeração

- Representação de números fracionários
  - Base binária (2)
    - "Posição" da parte inteira indica potência positiva de 2
    - Potência negativa de 2 para parte fracionária
    - 10,11 na base 2 = $1x2^1 + 0x2^0 + \mathbf{1x2^{-1} + 1x2^{-2}}$ = 2+0+1/2+1/4 = 2,75 na base decimal

Outros sistemas de numeração



- Maior interesse em decimal (10)
  - Anatomia e cultura humanas

  e binário (2)
  - Uso em sistemas computacionais
- Outros sistemas
  - Octal (8), {0,1,2, ... , 7}
  - Hexadecimal (16), {0,1,2, ... , 9, A,B,C,D,E,F}
  - Duodecimal (relógio, calendário)

Alguns sistemas numéricos

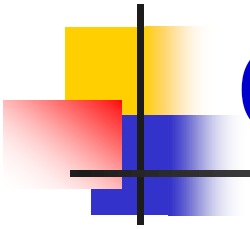
Decimal	Binário	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
⋮	⋮	⋮	⋮

Conversão de sistema ou base

- Uma caixa alienígena com o número **25** gravado na tampa foi entregue a um grupo de cientistas. Ao abrirem a caixa, encontraram **17** objetos. Considerando que o alienígena tem um formato humanóide, quantos dedos ele tem nas duas mãos?

Conversão de base

- $17_{10} = 25_b$
- $17 = 2 \times b^1 + 5 \times b^0$
- $17 = 2b + 5$
- $b = (17-5)/2 = 6$

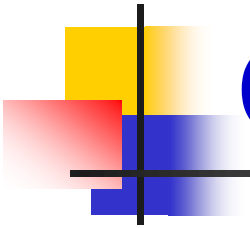


Conversão de base

- Um *sistema ternário* tem 3 "trits", cada trit assumindo o valor 0, 1 ou 2. Quantos "trits" são necessários para representar um número de seis bits?

bits para *trits*

- $2^6 = 3^y$
- $64 = 3^y$
- $y =$ **maior inteiro** $\{6 \times \log_2 2 / \log_2 3\}$
- $y = 4$
  - $(3^3 = 27 < 64 < 3^4 = 81)$



Conversão de Inteiro

- Binário para decimal
  - Já visto
- Inteiro decimal para binário
  - Divisão inteira (do quociente) sucessiva por 2, até que resto seja = 0 ou 1
  - Binário = composição do **último quociente** (Bit Mais Significativo – BMS) com **restos** (primeiro resto é bit menos significativo – bms)

Em inglês, *Most Significant Bit – MSB e least significat bit – lsb,* respectivamente.

Conversão de inteiro

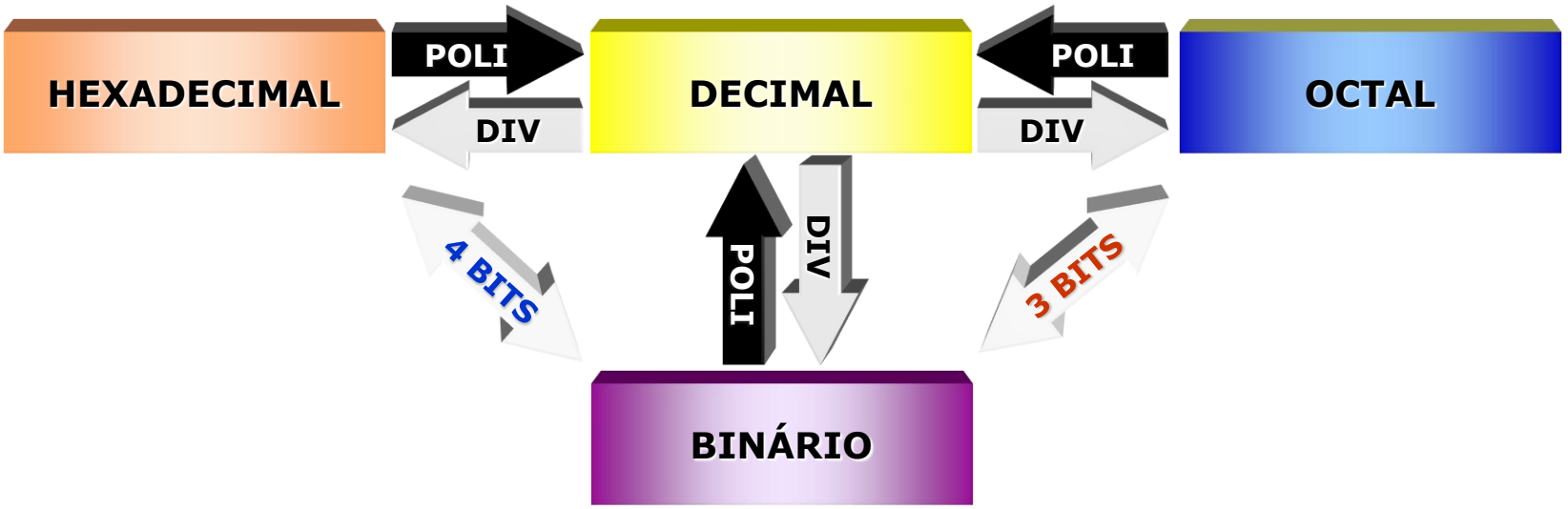
- Exemplo: Converter 25 decimal para binário
- 25 / 2 = 12 (quociente) e resto **1**=bms
- 12 / 2 = 6 (quociente) e resto **0**
- 6 / 2 = 3 (quociente) e resto **0**
- 3 / 2 = **1** (último quociente=BMS) e resto **1**
- Binário = BMS ... bms = **1 1 0 0 1**

  = $1x2^4 + 1x2^4 + 0x2^2 + 0x2^1 + 1x2^0$

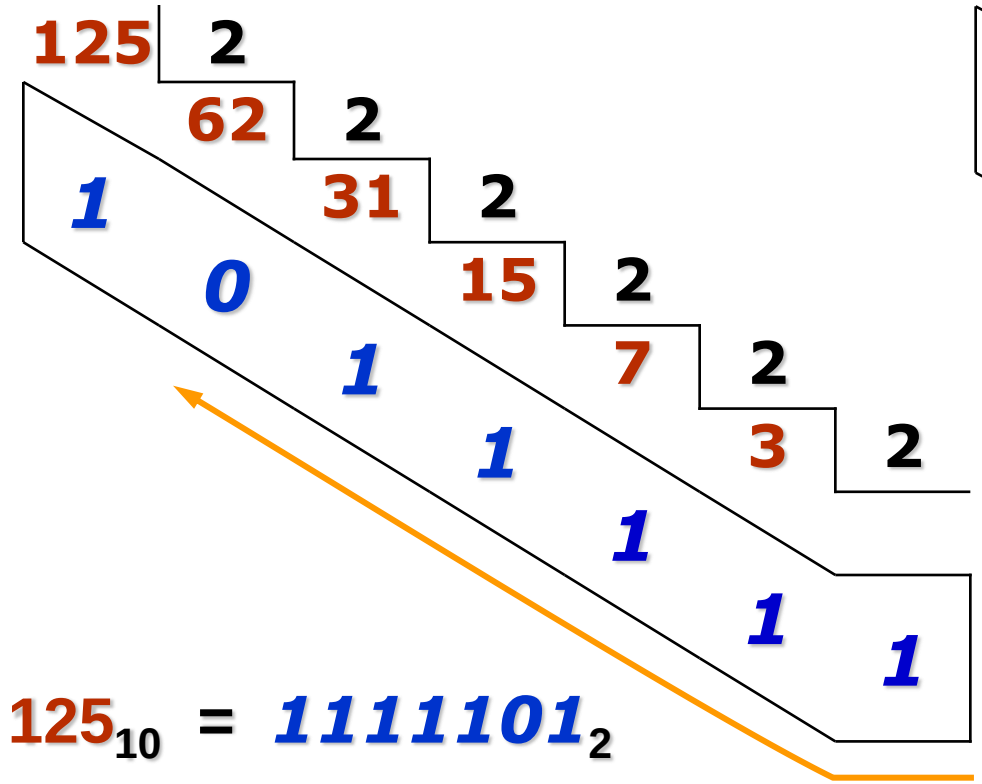
  = 16 + 8 + 0 + 0 + 1 = 25 decimal

Conversão de Inteiros entre Sistemas

- **Procedimentos básicos**:
  - Divisões sucessivas pela base do sistema para o qual se deseja converter o número
  - Decomposição polinomial do número a ser convertido
  - Agrupamento de bits

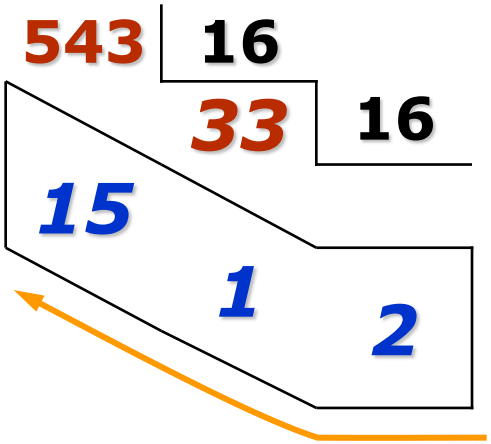


Conversão (Inteiros) entre sistemas



$125_{10} = 1111101_2$

Sentido da leitura



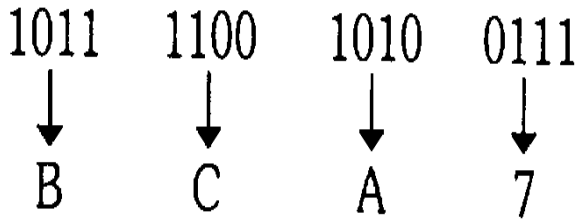
Sentido da leitura

O resto *15* é representado pela letra *F*

$538_{10} = 21F_{16}$

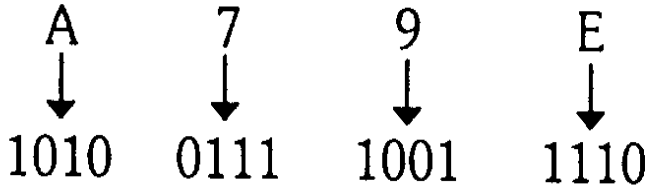
Conversão (Inteiros) entre sistemas

a) $(1011110010100111)_2 = (\ ?\ )_{16}$



$(1011110010100111)_2 = (BCA7)_{16}$

b) $(A79E)_{16} = (\ ?\ )_2$



$(A79E)_{16} = (1010011110011110)_2$

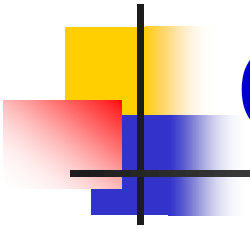
Conversão (Inteiros) entre sistemas

Conversão octal → hexadecimal

- Não é realizada diretamente → não há relação de potências entre as bases oito e dezesseis.
- Semelhante à conversão entre duas bases quaisquer → **base intermediária** (base binária)
- Conversão em duas etapas:

1 - número: base octal (hexadecimal) → binária.

2 - resultado intermediário: binária → hexadecimal (octal).



Conversão de fração

- Operação inversa: multiplicar parte fracionária por 2 até que parte fracionária do resultado seja 0 (zero)
- Bits da parte fracionária derivados das partes inteiras das multiplicações
- Bit imediatamente à direita da vírgula = Parte inteira da primeira multiplicação

Conversão de fração

- Exemplo: converter 0,625 decimal para binário
- 0,625 x 2 = 1,25 logo a primeira casa fracionária é **1** ; nova fração (resto) é 0,25 (1,25-1=0,25)
- 0,25 x 2 = 0,5 segunda casa é **0** ; resto é 0,5
- 0,5 x 2 = 1,0 terceira casa é **1** ; resto é zero.
- Resultado: $0{,}625_{10} = 0{,}\mathbf{101}_{2}$

Conversão partes inteira,fracionária juntas

- Para converter um número com parte inteira e parte fracionária, fazer a conversão de cada parte, separadamente.

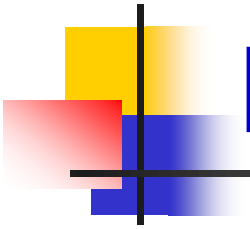
$$\underbrace{a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-2} + \ldots + a_0 b^0}_{\text{Parte } \textit{Inteira}} + \underbrace{a_{-1} b^{-1} + a_{-2} b^{-2} + \ldots + a_{-m} b^{-m}}_{\text{Parte } \textit{Fracionária}}$$

Conversão partes inteira,fracionária juntas

$(8{,}375)_{10} = (\ ?\ )_2$

0,375	0,750	0,500	0,000
x 2	x 2	x 2	
0,750	1,500	1,000	
↓	↓	↓	
0	*1*	*1*	

$8{,}375_{10} = 1000{,}011_2$



Exercícios

- Mostre que:
  - 5,8 = 101,11001100... , uma dízima.
  - 11,6 = 1011,10011001100...
    - a vírgula foi deslocada uma casa para a direita, pois 11,6 = 2 x 5,8 .

Representação em ponto (vírgula) flutuante - *float*

- Representação pode variar (“flutuar”) a posição da vírgula, ajustando potência da base.
  - $54,32 = 54,32 \times 10^0 = 5,432 \times 10^1 = 0,5432 \times 10^2 = 5432,0 \times 10^{-2}$
  - Forma normalizada usa um único dígito antes da vírgula, diferente de zero
    - Exemplo: $5,432 \times 10^1$

Representação em ponto (vírgula) flutuante - *float*

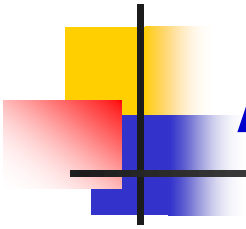
- No sistema binário:
  - $110101 = 110{,}101 \times 2^3 = 1{,}10101 \times 2^5 = 0{,}0110101 \times 2^7$
  - No caso dos números serem armazenados em um computador, os expoentes serão também gravados na base dois
    - Como $3_{10} = 11_2$ e $7 = 111_2$
    - $110{,}101 \times (10)^{11} = 1{,}10101 \times (10)^{101} = 0{,}0110101 \times (10)^{111}$
  - Na representação normalizada, há apenas um "1" antes da vírgula
    - Exemplo: $1{,}10101 \times (10)^{101}$

Representação em ponto (vírgula) flutuante - *float*

- **Algumas definições**
  - No número **1,10101x**$(10)^{101}$, tomado como referência:
    - **1,10101 = significando** (ou "mantissa")
    - **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").

Representação em ponto (vírgula) flutuante - *float*

- **Representação genérica**
  - $\pm d_0,d_1d_2...d_t \mathbf{x} (b)^{exp}$,
    - **t** é o número de dígitos da mantissa
    - $d_1d_2...d_t$ **=** mantissa, com $0 \leq d_i \leq (b-1)$
    - **exp =** expoente (inteiro com sinal)
- **OBS:**
  - a base não precisa ser explicitada



Armazenamento de *floats*

- Na organização/arquitetura do computador, definir:
  - Número de bits da mantissa (precisão, ***p***)
  - Número de bits do expoente
  - Um bit de sinal ("0" para + e "1" para -) para o número (geralmente o primeiro, da esquerda)

Armazenamento de *floats*

- Ilustração

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
  - 000 e 111 – especiais (ver adiante)
  - 011 (3_{10}) = expoente zero
  - 001 e 010 = expoente –2 e –1 (abaixo de zero)
  - 100, 101 e 110 = expoentes 1, 2 e 3 (acima zero)
  - OBS: Não podem seguir aritmética normal!

Armazenamento de *floats*

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

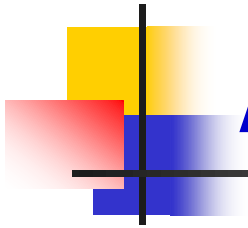
0 = +
1 = -

000 (especial)
001 (2^{-2})
010 (2^{-1})
011 (2^{0})
100 (2^{1})
101 (2^{2})
110 (2^{3})
111 (especial)

1,0000
1,0001
....
....
1,1111

1 = bit escondido

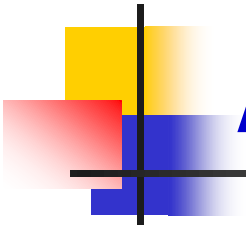
Armazenamento de *floats*



- Ainda os expoentes na ilustração...
  - **Maior número positivo é (lembre do bit escondido)**
    - 0 110 1111 = + 2^3 x 1,1111 = 2^3 x (2- 2^{-4}) = 1111,1 = 15,5 decimal
  - Menor número positivo é (lembre do bit escondido)
    - 0 001 0000 = + 2^{-2} x 1,0000 = 2^{-2} x 2^0 = 0,01 ou 0,25 decimal

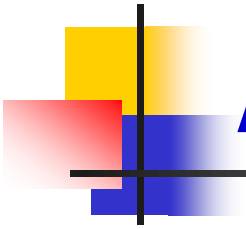
Armazenamento de *floats*

- 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



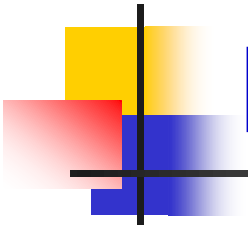
Armazenamento de *floats*

- Ainda as combinações especiais...
  - Normalização não permite representar zero!
  - 000 – representação NÃO normalizada
    - 00000000 = + 0 decimal
    - 10000000 = - 0 decimal
    - São iguais em comparações



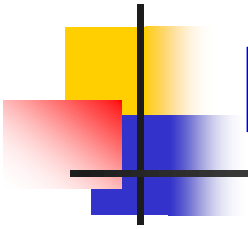
Armazenamento de *floats*

- Ainda as combinações especiais...
  - 111 – representações de infinito
    - 01110000 = + infinito
    - 11110000 = - infinito
    - 11111000 = indeterminação
    - Outras combinações 11111_ _ _ = Not A Number (NANs)



Padrão IEEE para *floats*

- 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.
  - OBS: Padrão 854 (base = 10 ou 2, nem especifica layout dos bits)



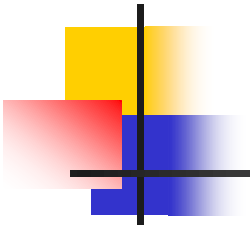
Padrão IEEE para *floats*

- O padrão (ou norma) IEEE 754 define dois formatos básicos para os números em ponto flutuante:
  - o formato ou precisão simples, com 32 bits; e,
  - o duplo com 64 bits.

Padrão IEEE 754 para *floats*

	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



IEEE 754 com precisão simples

- Expoentes na precisão simples c/256 combinações
  - 1111 1111
    - 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

IEEE 754 com precisão simples

- Expoentes na precisão simples c/256 combinações
  - 0111 1111 (127_{10}) = expoente zero (*bias* = polarização)
  - 0000 0001 = menor expoente = –126 (abaixo de um)
  - 1111 1110 = maior expoente = +127 (acima de um)

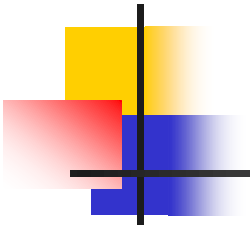
    OBS: Expoente vale (Número em binário MENOS 127)
  - 0000 0000
    - sinal=1 e significando = 0...0 : -zero
    - sinal=0 e significando = 0...0 : +zero

IEEE 754 com precisão simples

- Expoentes na precisão simples c/256 combinações

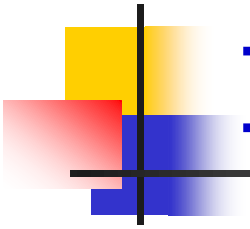
(0) 0000 0000 (especial)
(1) 0000 0001 (2^{-126}) menor expoente
.....
0111 1100
(125) 0111 1101 (2^{-2})
(126) 0111 1110 (2^{-1})
(127) 0111 1111 (2^{0})
(128) 1000 0000 (2^{1})
(129) 1000 0001 (2^{2})
1000 0010
.....
(254) 1111 1110 (2^{127}) maior expoente
(255) 1111 1111 (especial)

IEEE 754 com precisão simples



- 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 1111110 11…11 = $2^{127} \times (2-2^{-23})$
- A faixa de números negativos é:
  - de $-(2-2^{-23}) \times 2^{127}$ a -2^{-149}

IEEE 754 com precisão dupla



- No formato (precisão) duplo, o menor expoente é representado por 00000000001, valendo -1022, e o maior expoente é representado por 11111111110, valendo +1023. Em ambos os casos, o expoente vale o número representado em binário menos 1023 (este é o valor da *bias* = zero).

IEEE 754 com precisão dupla

Verifique:

- 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 1111110 11…11 = $2^{1023} \times (2-2^{-52})$
- A faixa de números negativos é:
  - de $-(2-2^{-52}) \times 2^{1023}$ a -2^{-1074}

IEEE 754 com precisão simples

- Expoentes na precisão dupla c/2048 combinações

(0) 00000000000 (especial)
(1) 00000000001 (2^{-1022}) menor expoente
.....
01111111100
01111111101 (2^{-2})
(1022) 01111111110 (2^{-1})
(1023) 01111111111 (2^{0})
(1024 10000000000 (2^{1})
10000000001 (2^{2})
10000000010
.....
(2046) 11111111110 (2^{1023}) maior expoente
(2047) 11111111111 (especial)

Quadro resumo IEEE 754

	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 $\sim 10^{308.3}$

Erro na representação de *floats*

- Número finito de bits na representação (número é apenas "maior" na precisão dupla), implica em "truncamento" (ou arredondamento) do número real a ser representado. Truncamento introduz erro na representação. Casos especiais:
  - *Overflow*: número a representar é maior que maior número possível de ser representado
  - *Underflow*: número a representar é menor que menor número possível de ser representado

Limite no erro na representação de um *float*

- A forma normalizada do número **N** é 1,**n** x $2^{\mathbf{e}}$
  - Supõe-se que **e** esteja dentro dos limites dessa representação (ou ocorreria *overflow*).
  - Se **n** não couber no número de bits da representação (precisão) do significando, **p**, haverá truncamento, introduzindo erro.

Limite no erro na representação de um *float*

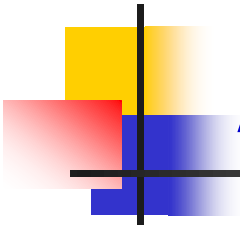
- A forma normalizada do número **N** é 1,**n** x 2^{e}
- **Ex: N** = 1,101011110100101... x 2^{e} e que **p** número de bits (precisão) do significando seja 4.
  - A representação de **N** seria 1,1010 x 2^{e} gerando um $Erro_{N}$ = 0,11110100101... x 2^{c-4}
  - O erro relativo é definido como $E_{N} = Erro_{N} / N$, ou:

    0,11110100101... x 2^{c-4} / 1,101011110100101... x 2^{e}

    = 0,11110100101... x 2^{-4} / 1,101011110100101...

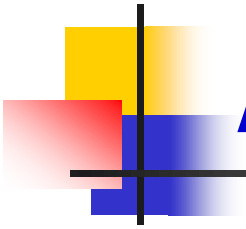
Limite no erro na representação de um *float*

- Note que E_N será máximo quando o numerador for máximo e o denominador for mínimo, ou seja:
  - E_N (max) = 0,1111111….. x 2^{-4} / 1,0000000…
  - Lembrando que 0,11111…. < 1 , tem-se:
  - E_N (max) < 2^{-4} , onde 4 está representando **p**, número de bits (precisão) do significando.
- Portanto, E_N (max) < 2^{-p}, para representações normalizadas.



Aritmética com *floats*

- Conhecidos os erros em dois números, é possível determinar o erro de uma operação entre eles, como adição, subtração, multiplicação e divisão.
  - Erro depende de método / procedimentos empregados



Aritmética com *floats*

- Padrão IEEE 754 define algoritmo para adição, subtração, multiplicação, divisão e raiz quadrada e exige que implementações produzam o(s) mesmo(s) resultado(s).
  - Igualdade dos bits (resultados) em várias processadores
  - Portabilidade de *software*
  - Vide próximo módulo

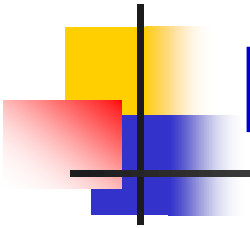
Exercício Nr. 1

Seja a seguinte representação de números **positivos** em ponto flutuante:

Bit 7.	**Bit 6. Bit 5 Bit 4**	**Bit 3 Bit 2 Bit 1 Bit 0**
Sinal do expoente	EXPOENTE	MANTISSA

Sendo que o expoente é representado diretamente pelo respectivo número binário e os números são normalizados pela primeira casa decimal, ou seja
4.5 é representado como $0.45\ 10^{1}$ ou, em binário,
100.1 é representado por $0.1001\ 2^{11}$ o que daria 00111001 na representação acima.

1) Qual o maior e o menor número positivo que podem ser representados neste formato? Mostre o resultado em decimal, binário e na representação interna.
2) Com fica a situação do número 0? Sugira uma solução.
3) Represente, neste formato os números (decimais) 13, 0.12 e 3.501.
Em quais números ocorreram erros de representação?
4) Seja a representação 00101000. Ela representa qual número? Se eu subtrair 0.1_2 deste número, como seria representado o número resultante?



Exercício Nr. 2

1) Repita os itens 3 e 4 do exercício anterior, agora usando a representação em ponto flutuante de 8 bits vista anteriormente neste documento.

2) Determine, para ambas as representações, a densidade dos números maiores que 1, ou seja, a distância entre dois números subseqüentes. *SUGESTÃO: tome a representação de um número qualquer some 0.0001 à mantissa e calcule a diferença entre estes dois números*.