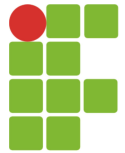
Representação Numérica

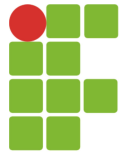
Algoritmos e Programação de Computadores – ABI/LFI/TAI

Prof. Daniel Saad Nogueira Nunes

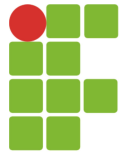
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



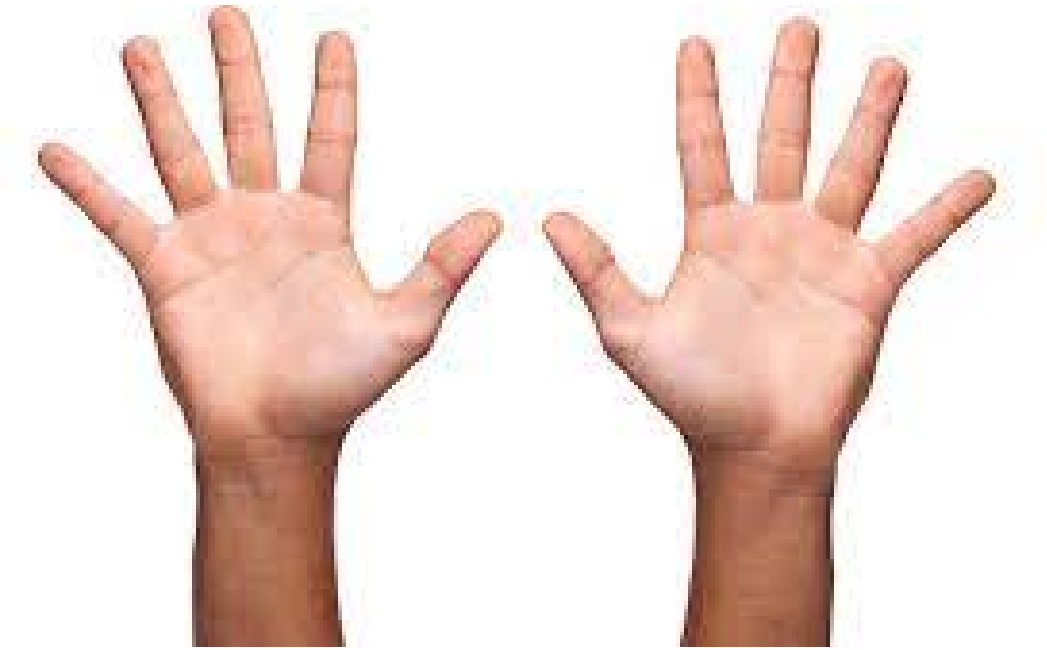
Sumário

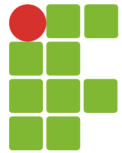


Sistemas de Numeração

- Utilizamos números para expressar quantidades.
- Um sistema de numeração consiste nos símbolos utilizados para representar os números.
- O mais utilizado no dia a dia é o sistema decimal de numeração, apesar de várias civilizações terem adotado outros sistemas no passado.

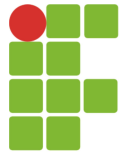
Sistemas de Numeração





Sistemas de Numeração

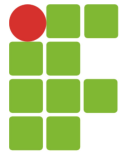
- No sistema decimal, temos 10 algarismos: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- Portanto dizemos que a **base** dele é 10.
- Além disso, ele é um sistema **posicional**, em que a posição dos algarismos indica o peso a ser aplicado àquele algarismo.
- Os algarismos são colocados em sequência para compor um número.
- $9543 = 3 \cdot 10^0 + 4 \cdot 10^1 + 5 \cdot 10^2 + 9 \cdot 10^3 = 3 + 40 + 500 + 9000$.



Sistemas de Numeração

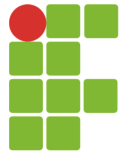
- Sendo um pouco mais formal, um número x, de n dígitos, composto da sequência de dígitos $x_{n-1}x_{n-2}\ldots x_0$ pode ser interpretado da seguinte maneira:

$$x = \sum_{i=0}^{n-1} x_i \cdot 10^i$$

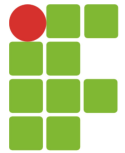


Sistemas de Numeração

- Podemos ter sistemas de numeração com diferentes bases.
- Como exemplos importantes para computação, podemos destacar os sistemas posicionais: binário, octal e hexadecimal.

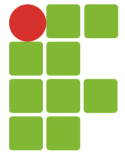


Sumário



Sistema Binário

- O sistema de numeração binário é muito relevante para a Computação, uma vez que a lógica digital é booleana.
- Este sistema possui apenas dois algarismos (bits): 0 ou 1, que representam ligado ou desligado; com tensão ou sem tensão; ...
- Ele também é um sistema posicional.
- Sua base é 2.



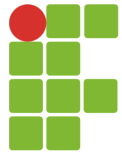
Sistema Binário

Notação

De agora em diante, para evitar ambiguidades, usaremos a notação de subscrito para indicar a base do sistema no qual o número foi escrito.

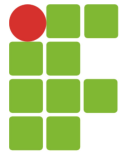
Seja x um número e d uma base. Diremos que x está escrito de acordo com o sistema de base d com a seguinte notação:

$$x_d$$



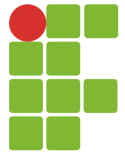
Notação

- 1010010_2 significa que estamos falando do número 1010010_2 em binário.
- 10_{10} significa que estamos falando do número 10 (dez) em decimal.
- 712_8 significa que estamos falando de 712 na base 8.



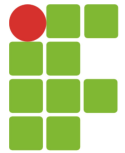
Sistema Binário

- Exemplos de números em binário: 1001001_2, 1101_2, 11001100_2.
- Como interpretar um número em binário no seu equivalente em decimal?



Sistema Binário

- Exemplos de números em binário: 1001001_2, 1101_2, 11001100_2.
- Como interpretar um número em binário no seu equivalente em decimal?
- Adaptamos a fórmula utilizada para a base 10.

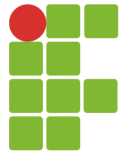


Sistema Binário

Conversão Binário-Decimal

- Um número binário $x = x_{n-1}x_{n-2}\ldots x_0$ pode ser convertido para o seu equivalente em decimal realizando a seguinte soma:

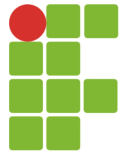
$$\sum_{i=0}^{n-1} x_i \cdot 2^i$$



Conversão Binário-Decimal

Exemplos

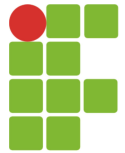
- $10_2 = 0 \cdot 2^0 + 1 \cdot 2^1 = 2_{10}$
- $1101_2 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 = 13_{10}$.
- $111_2 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 = 7_{10}$.



Conversão de Bases

Conversão Decimal-Binário

- Também é possível realizar o processo inverso: converter um número decimal em seu equivalente em binário.
- Para isto utilizamos o processo inverso: em vez de multiplicações, usamos divisões.
- Algoritmo: dividir o número por 2 e guardar o resto da divisão a cada etapa. Quando o quociente chegar a 0, compor o número binário utilizando o resultado dos restos de trás para frente.

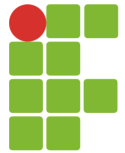


Conversão de Bases

Exemplo

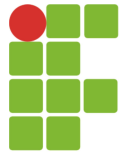
Número	Quociente	Resto
225	112	1
112	56	0
56	28	0
28	14	0
14	7	0
7	3	1
3	1	1
1	0	1

- $225_{10} = 11100001_2$.

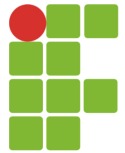


Sistemas Alternativos

- Outros sistemas bem relevantes para computação são os sistemas octal e hexadecimal (base 8 e base 16), que também são posicionais.
- Eles permitem compactar números em binários de maneira a simplificar a sua conversão.

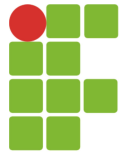


Sumário



Sistema Octal

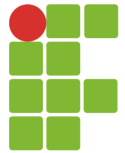
- O sistema octal, base 8, é baseado nos algarismos $\{0, 1, 2, 3, 4, 5, 6, 7\}$.
- Um número binário pode ser convertido facilmente para um número octal ao separá-lo em triplas (da esquerda para a direita) e interpretar os valores destas triplas.
- Caso o número de bits não seja múltiplo de 3, o último agrupamento de bits (mais à esquerda) terá 1 ou 2 bits.



Sistema Octal

Conversão Binário-Octal

- $\underbrace{101}_{5}\underbrace{011}_{3} = 53_8$
- $\underbrace{11}_{3}\underbrace{010}_{2} = 32_8.$
- $\underbrace{1}_{1}\underbrace{110}_{6} = 16_8.$

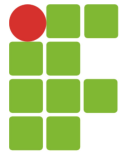


Sistema Octal

Conversão Octal-Binário

- Para converter um número octal para um número em binário fazemos o processo inverso.
- Cada algarismo em octal representará três em binário.
- Podemos utilizar a seguinte tabela:

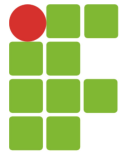
Octal	**Binário**
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111



Sistema Octal

Conversão Octal-Binário

- $42_8 = 100010_2$
- $71_8 = 111001_2$
- $15_8 = 001101_2 = 1101_2$ (zeros à esquerda são omitidos).



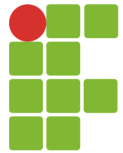
Conversão de Bases

Conversão Octal para Decimal

- Para converter um número octal $x = x_{n-1}x_{n-2}\ldots x_0$ para decimal, basta aplicar o somatório:

$$\sum_{i=0}^{n-1} = x_i \cdot 8^i$$

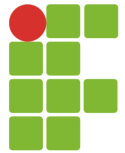
  - $7234_8 = 4 \cdot 8^0 + 3 \cdot 8^1 + 2 \cdot 8^2 + 7 \cdot 8^3 = 3740_{10}$



Conversão de Bases

Conversão Decimal para Octal

- Algoritmo: dividir o número por 8 e guardar o resto da divisão a cada etapa. Quando o quocientePP chegar a 0, compor o número octal utilizando o resultado dos restos de trás para frente.

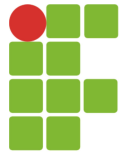


Conversão de Bases

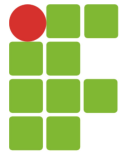
Exemplo

Número	Quociente	Resto
3740	467	4
467	58	3
58	7	2
7	0	7

- $3740_{10} = 7234_8$.

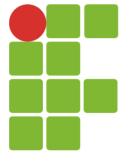


Sumário



Sistema Hexadecimal

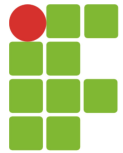
- O sistema hexadecimal é composto pelos algarismos $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$.
- Permite compactar ainda mais um número binário, enquanto mantém sua conversão simples.



Conversão de Bases

Conversão Binário-Hexadecimal

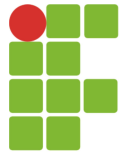
- Para converter um número binário em hexadecimal, seguimos uma estratégia muito parecida com a conversão de binário para octal.
- Mas em vez de triplas, utilizamos quádruplas!
- Caso o número de bits não seja divisível por 4, a quádrupla mais à esquerda terá de 1 a 3 bits.



Conversão de Bases

Exemplo

- $\underbrace{1010}_{\text{A}}\underbrace{0101_2}_{5} = \text{A}5_{16}$
- $\underbrace{110}_{6}\underbrace{1111_2}_{\text{F}} = 6\text{F}_{16}$
- $\underbrace{10}_{2}\underbrace{0111_2}_{7} = 27_{16}$
- $\underbrace{1}_{1}\underbrace{0011_2}_{3} = 13_{16}$



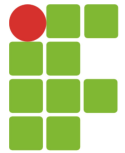
Conversão de Bases

Conversão Hexadecimal-Binário

- A conversão de hexadecimal para binário é bem simples, cada número hexadecimal produz 4 bits. Utilizamos a seguinte tabela:

Hexadecimal	Binário
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

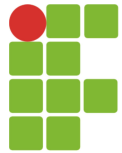
Hexadecimal	Binário
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111



Conversão de Bases

Exemplo

- $\mathrm{FC}_{16} = 11111100_2$
- $79_{16} = 01111001_2 = 1111001_2$
- $2\mathrm{D}_{16} = 00101101_2 = 101101_2$
- $1\mathrm{B}_{16} = 00011011_2 = 11011_2$



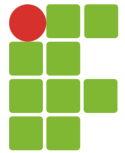
Conversão de Bases

Conversão Hexadecimal para Decimal

- Para converter um número hexadecimal $x = x_{n-1}x_{n-2}\ldots x_0$ para decimal, basta aplicar o somatório:

$$\sum_{i=0}^{n-1} = x_i \cdot 16^i$$

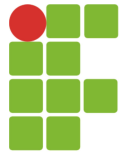
  - $\mathrm{F53A}_{16} = 10 \cdot 16^0 + 3 \cdot 16^1 + 5 \cdot 16^2 + 15 \cdot 16^3 = 62778_{10}$



Conversão de Bases

Conversão Decimal para Hexadecimal

- Algoritmo: dividir o número por 16 e guardar o resto da divisão a cada etapa. Quando o quociente chegar a 0, compor o número hexadecimal utilizando o resultado dos restos de trás para frente.

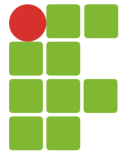


Conversão de Bases

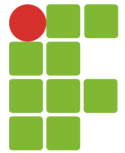
Exemplo

Número	Quociente	Resto
62778	3923	10 (A)
3923	245	3
245	15	5
15	0	15 (F)

- $62778_{10} = \mathrm{F53A}_{16}$.

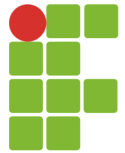


Sumário

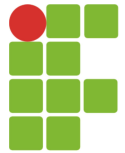


Números Inteiros

- Até o momento vimos como representar números naturais em binário e em outras bases.
- Mas como representamos números negativos?
- Como representar os números inteiros computacionalmente?
- Algumas estratégias:
  - Sinal-magnitude;
  - Complemento de um;
  - Complemento de dois.

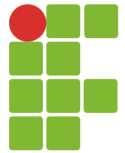


Sumário



Sinal-magnitude

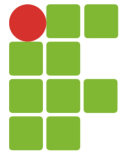
- Na abordagem de sinal magnitude, reservamos o bit mais significativo (mais à esquerda) para descrever o sinal:
  - 0: indica que o número é positivo.
  - 1: indica que o número é negativo.
- Os demais bits descrevem o número da forma como vimos anteriormente.



Sinal-magnitude

- Formalmente temos que se $x = x_{n-1}x_{n-2}\ldots x_0$ é um número em binário no formato sinal-magnitude, podemos obter o decimal correspondente da seguinte maneira:

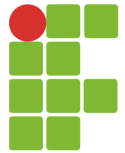
$$-1^{x_{n-1}} \cdot \sum_{i=0}^{n-2} x_i \cdot 2^i$$



Sinal-magnitude

Exemplo

- $0100101_2 = -1^0 \cdot (1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5) = 37_{10}$
- $111001_2 = -1^1 \cdot (1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4) = -25_{10}$

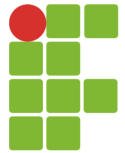


Sinal-Magnitude

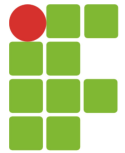
Vantagens/Desvantagens

Assumindo que estamos utilizando n bits para representar números binários:

- Vantagens:
  - Representação familiar.
- Desvantagens:
  - o valor 0_{10} possui duas representações em binário: $1\underbrace{00\ldots0}_{n-1}$ e $0\underbrace{0\ldots0}_{n-1}$.
  - Aritmética mais complicada, sempre temos que examinar o bit de sinal.
- Intervalo representado é simétrico $[-(2^{n-1}-1), 2^{n-1}-1]$.

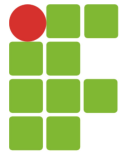


Sumário



Complemento de Um

- A abordagem *complemento de um* representa os números negativos simplesmente invertendo os bits do número correspondente positivo.
- O bit mais significativo é utilizado para indicar o sinal, como na abordagem sinal-magnitude.

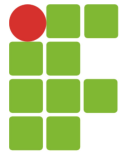


Complemento de Um

Exemplo

Considerando que estamos representando os números binários com 8 bits:

- $43_{10} = 00101011_2$.
- $-43_{10} = 11010100_2$

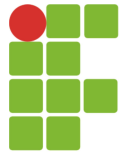


Complemento de Um

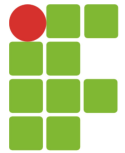
Vantagens/Desvantagens

Assumindo que estamos utilizando n bits para representar números binários:

- Vantagens:
  - Aritmética mais direta.
- Desvantagens:
  - o valor 0_{10} possui duas representações em binário: $1\underbrace{00\ldots0}_{n-1}$ e $0\underbrace{0\ldots0}_{n-1}$.
- Intervalo representado é simétrico $[-(2^{n-1}-1), 2^{n-1}-1]$.



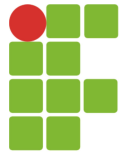
Sumário



Complemento de Dois

- A estratégia de *complemento de dois* é obtida a partir da representação em complemento de um somada com 1_2.
- Supondo que os números binários estejam sendo representados com n bits em complemento de dois, então o número binário $x = x_{n-1} \ldots x_0$ pode ser convertido para decimal da seguinte forma:

$$-1 \cdot x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i$$

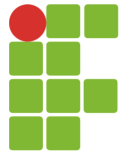


Complemento de Dois

Exemplo

Considerando que estamos representando os números binários com 8 bits:

- $43_{10} = 00101011_2$.
- Complemento de um: $-43_{10} = 11010100_2$
- Complemento de dois: 11010101_2
- $11010101_2 = -1 \cdot 1 \cdot 2^7 + 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 + 1 \cdot 2^6 = -128 + 1 + 4 + 16 + 64 = -43_{10}$

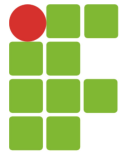


Complemento de Dois

Exemplo

Considerando que estamos representando os números binários com 8 bits:

- $1_{10} = 00000001_2$.
- Complemento de um: $-1_{10} = 11111110_2$
- Complemento de dois: 11111111_2
- $11111111_2 = -1 \cdot 1 \cdot 2^7 + 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6 = -128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = -1_{10}$

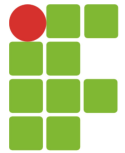


Complemento de Dois

Exemplo

Considerando que estamos representando os números binários com 8 bits:

- $0_{10} = 00000000_2$.
- Complemento de um: $-0_{10} = 11111111_2$
- Complemento de dois: 00000000_2

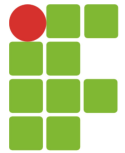


Complemento de Dois

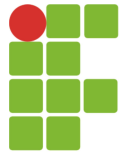
Vantagens/Desvantagens

Assumindo que estamos utilizando n bits para representar números binários:

- Vantagens:
  - Aritmética mais direta ainda.
  - O valor 0_{10} possui uma única representação em binário: $\underbrace{0\ldots0}_{n}$.
- Desvantagens:
  - Um pouco mais difícil de compreender.
- Intervalo representado é assimétrico $[-2^{n-1}, 2^{n-1}-1]$.
- **Estratégia padrão para representar inteiros!**

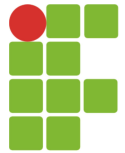


Sumário



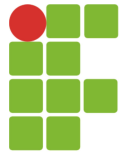
Reais

- Diferentemente dos números inteiros, não é possível representar todos os números reais dentro de um intervalo em um computador usando uma quantidade fixa de bits.
- Isso dá margem à arredondamentos, o que leva à imprecisões.
- Erros são introduzidos durante os cálculos.

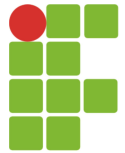


Reais

- Apesar das limitações, precisamos trabalhar com números reais.
- Duas principais estratégias são:
  - Representação em ponto fixo.
  - Representação em ponto flutuante.

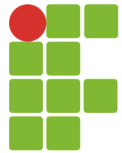


Sumário



Ponto fixo

- A representação em ponto fixo se parece muito com a representação de inteiros em complemento de dois.
- A diferença é que temos a presença de um **ponto binário**.
- A localização deste ponto binário nos indica que à direita deste ponto, os expoentes passam a ser negativos.
- Equivalentemente podemos ver a representação em ponto fixo como a representação inteira em complemento de dois dividido por alguma potência de dois.

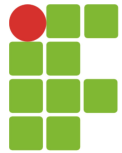


Ponto Fixo

Exemplo

Supondo que o ponto binário se encontra antes do segundo bit menos significativo, e assumindo números binários de 8 bits em complemento de dois temos:

- $010011.10_2 = 0 \cdot 2^{-2} + 1 \cdot 2^{-1} + 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = 19.5$
- $010011.10_2 = 01001110_2 / 4_{10} = 78/4 = 19.5$

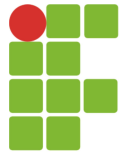


Ponto Fixo

Exemplo

Supondo que o ponto binário se encontra antes do segundo bit menos significativo, e assumindo números binários de 8 bits em complemento de dois temos:

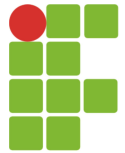
- $110011.11_2 = -1 \cdot 2^5 + 1 \cdot 2^{-2} + 1 \cdot 2^{-1} + 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = -12.25_{10}$
- $110011.11_2 = 11001111_2/4_{10} = -49/4 = -12.25_{10}$



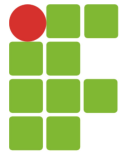
Ponto Fixo

Vantagens/Desvantagens

- Vantagens:
  - Aritmética extremamente simples, podemos usar a mesma lógica de hardware da representação dos inteiros.
  - Alto desempenho.
- Desvantagens:
  - Baixa precisão.
- Encontrado em hardwares mais simples e sistemas embarcados.

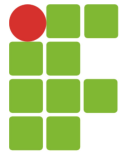


Sumário

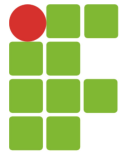


Ponto Flutuante

- A representação em ponto flutuante, como o nome diz, permite que o ponto binário seja regulado conforme um valor de expoente.
- A aritmética é mais complexa, mas permite representar mais valores que a estratégia de ponto fixo.
- Padrão universalmente adotado: IEEE 754:
  - Precisão simples (32-bits);
  - Precisão dupla (64-bits);

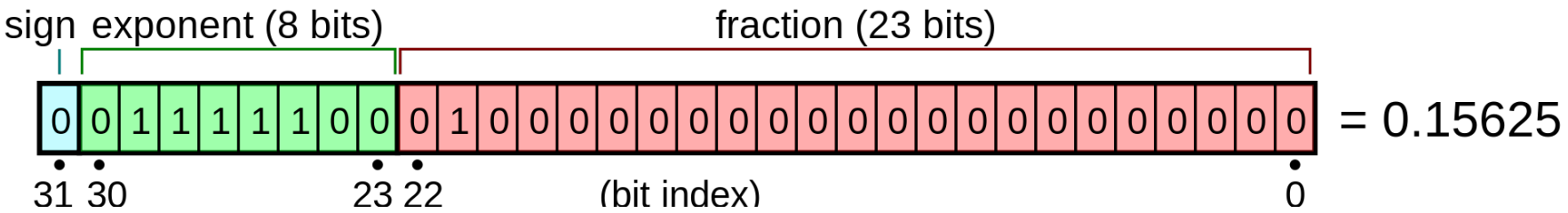


Sumário



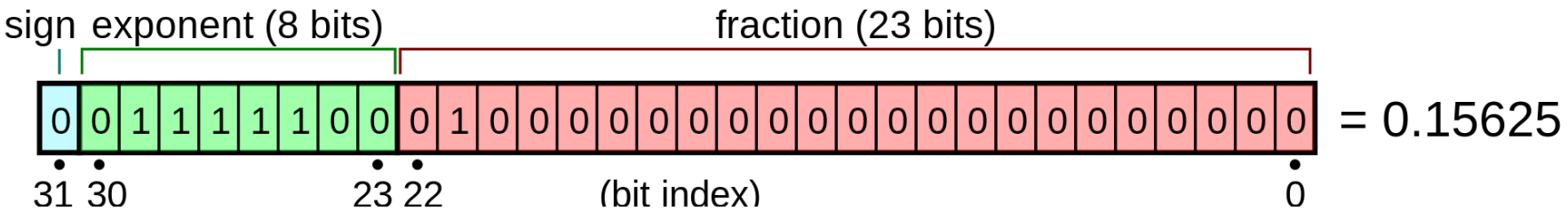
IEEE 754: Precisão Simples

- O formato IEEE 754 de precisão simples ocupa exatamente 32-bits:
  - 1 bit representa o sinal (s).
  - 8 bits para o expoente (e).
  - 23 bits para a mantissa, parte fracionária, (f), considerando representação de ponto-fixo).
- Forma geral: $(-1)^s \cdot 1.f \cdot 2^{e-127}$

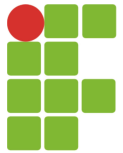


IEEE 754: Precisão Simples

Exemplo

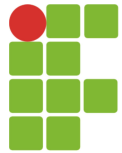


- $s = 0.$
- $e = 124$
- $f = 0.25$
- Resultado: $(-1)^0 \cdot 1.25 \cdot 2^{(124-127)} = 1.25 \cdot 2^{-3} = 0.15625$



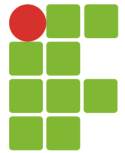
IEEE 754: Precisão Simples

- Considerando a forma geral, existem 7 casos do padrão IEEE para precisão simples:
  1. $e = 255$ e $f \neq 0$ temos NaN (not a number).
  2. $s = 0$, $e = 255$ e $f = 0$: temos ∞
  3. $s = 1$, $e = 255$ e $f = 0$: temos $-\infty$
  4. $0 < e < 255$: temos $(-1)^s \cdot 1.f \cdot 2^{e-127}$
  5. $e = 0$ e $f \neq 0$: temos $(-1)^s \cdot (0.f) \cdot 2^{-126}$
  6. $s = 0$, $e = 0$ e $f = 0$: temos 0.
  7. $s = 1$, $e = 0$ e $f = 0$: temos -0.

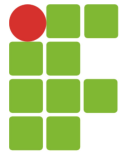


IEEE 754: Precisão Simples

- NaN: usado para indicar um valor irrepresentável, como uma raiz de número negativo.
- ∞ e $-\infty$: utilizados para representador *overflow*.

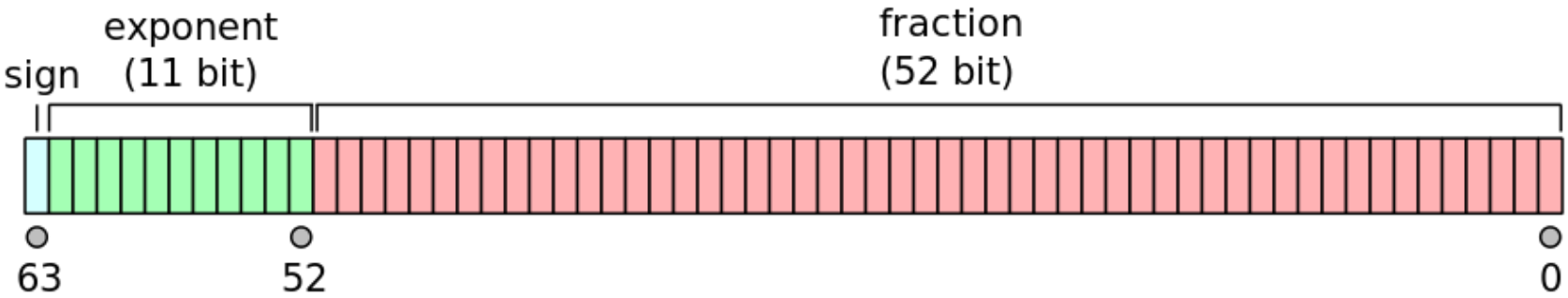


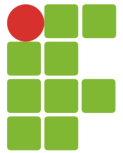
Sumário



IEEE 754: Precisão Dupla

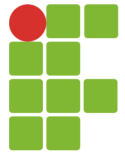
- O formato IEEE 754 de precisão dupla ocupa exatamente 64-bits:
  - 1 bit representa o sinal (s).
  - 11 bits para o expoente (e).
  - 52 bits para a mantissa, parte fracionária, (f), considerando representação de ponto-fixo.
- Forma geral: $(-1)^s \cdot 1.f \cdot 2^{e-1023}$



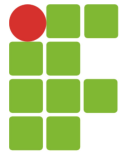


IEEE 754: Precisão Dupla

- Considerando a forma geral, existem 7 casos do padrão IEEE para precisão dupla:
  1. $e = 2047$ e $f \neq 0$ temos NaN (not a number).
  2. $s = 0$, $e = 2047$ e $f = 0$: temos ∞
  3. $s = 1$, $e = 2047$ e $f = 0$: temos $-\infty$
  4. $0 < e < 2047$: temos $(-1)^s \cdot 1.f \cdot 2^{e-1023}$
  5. $e = 0$ e $f \neq 0$: temos $(-1)^s \cdot (0.f) \cdot 2^{-1022}$
  6. $s = 0$, $e = 0$ e $f = 0$: temos 0.
  7. $s = 1$, $e = 0$ e $f = 0$: temos -0.

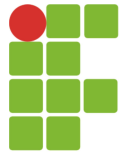


Sumário



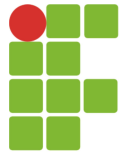
Representação IEEE 754

- Permitem maior precisão para representação de números reais.
- Aritmética mais complexa e mais lenta que a de ponto fixo.
- Conseguem representar resultados como NaN, ∞ e $-\infty$.
- 0 e -0 são representados.

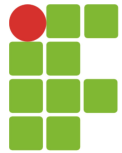


Representação IEEE754

- Além dos dois formatos compreendidos temos o formato de **precisão estendida**, que considera mais bits.
- O padrão IEEE 754 também especifica quais são as opções no caso de arredondamentos.

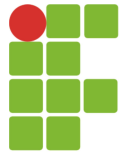


Sumário



ASCII

- A codificação ASCII consegue representar caracteres utilizando inteiros sem sinal de 7 bits.
- Cada inteiro corresponde a um caractere, seja ele de controle ou não.
- Caracteres de controle possuem várias finalidades, como: quebra de linha, tabulação horizontal e o chamado *carriage return*.
- A maioria das codificações de caracteres atuais é baseado na ASCII, considerando ela como um subconjunto.
  - ASCII estendida.
  - UTF-8.
  - UTF-16.
  - ...



ASCII

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]