Universidade Federal de Uberlândia
Faculdade de Computação

Sistemas Digitais
Módulo 3
Codificações BCD, Gray e ASCII e Números Fracionários

Graduação em Sistemas de Informação

Prof. Dr. Daniel A. Furtado

Codificação BCD

- **BCD** = ***B**inary-**c**oded-**d**ecimal* (decimal codificado em binário)
- É uma forma de representar números decimais em binário por meio da codificação de cada dígito individualmente;
- BCD não é um sistema de numeração;
- Cada dígito do número decimal é representado por uma **quantidade fixa de bits**;
- Em geral, 4 bits são usados para codificar cada dígito decimal:
  - Permite codificar dois dígitos decimais por byte;
- Exemplo:

8	7	4	(decimal)
↓	↓	↓	
1000	0111	0100	(BCD)

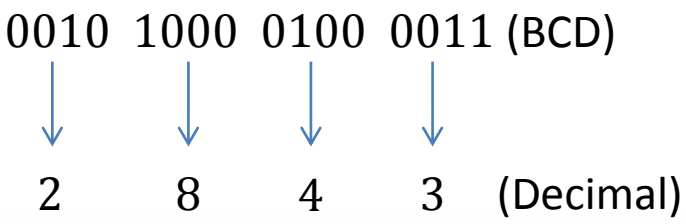
$$874_{10} = 100001110100_{BCD}$$

Codificação BCD

- A decodificação também é direta. Basta agrupa os bits da direita para a esquerda e encontrar o dígito decimal correspondente a cada grupo de bits.

- Exemplo. Encontrar o número decimal codificado na sequência de bits BCD a seguir:

$$0011100001000011$$



$$0011100001000011_{BCD} = 2843_{10}$$

Codificação BCD

- **Vantagens**
  - É uma forma direta de codificar números decimais, pois cada dígito decimal é **sempre** codificado por uma quantidade **fixa** de bits;
  - A decodificação também é direta;
- **Desvantagens**
  - Alguns códigos binários nunca são utilizados na codificação. Com 4 bits, por exemplo, os binários maiores do que 1001 nunca são utilizados (1010, 1011, 1100, 1101, 1110, 1111);
    - Há desperdício de espaço na representação.

Codificação BCD – Exercícios

- Codifique os seguintes números decimais em BCD de 4 bits
  - 45_{10}
  - 196_{10}
- Os códigos binários a seguir representam codificações de números decimais em BCD de 4 bits. Obtenha os números decimais correspondentes.
  - 0100 0001 1001
  - 0011 0010 1000

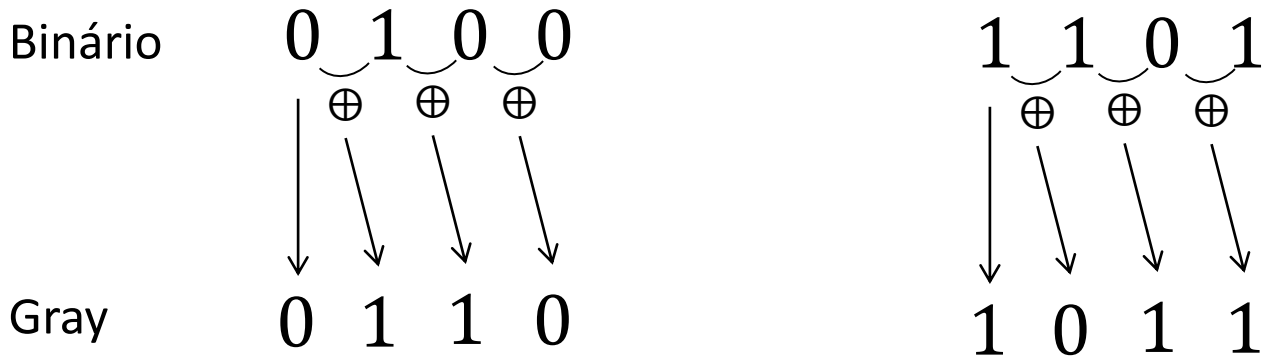
Código Gray

- Forma de codificação em que **apenas 1 bit muda de um número para outro** em sequência;
- Proposto por Frank Gray;
- Utilizado em técnicas de correção de erros, mapas de Karnaugh, algoritmos genéticos, dentre outros.

Binários de 3 bits			Código GRAY equivalente		
B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

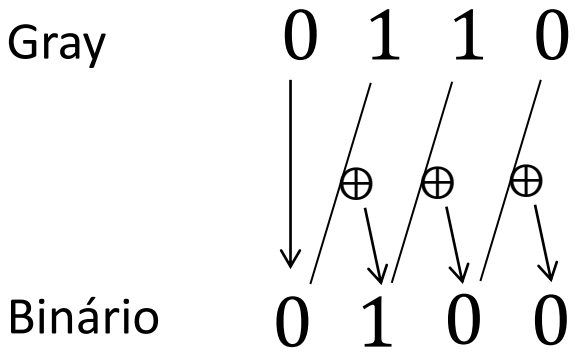
Conversão Binário → Gray

- O MSB do código Gray será igual ao MSB do número binário;
- O próximo bit (da esquerda para a direita) do código Gray é obtido pelo cálculo da operação XOR entre o respectivo bit do número binário e o bit binário anterior;
- Exemplos:



Conversão Gray → Binário

- O MSB do número binário será igual ao MSB do código Gray;
- O próximo bit (da esquerda para a direita) do número binário é obtido pelo cálculo da operação XOR entre o respectivo bit do código Gray e o bit binário anterior;
- Exemplo:



Codificação ASCII

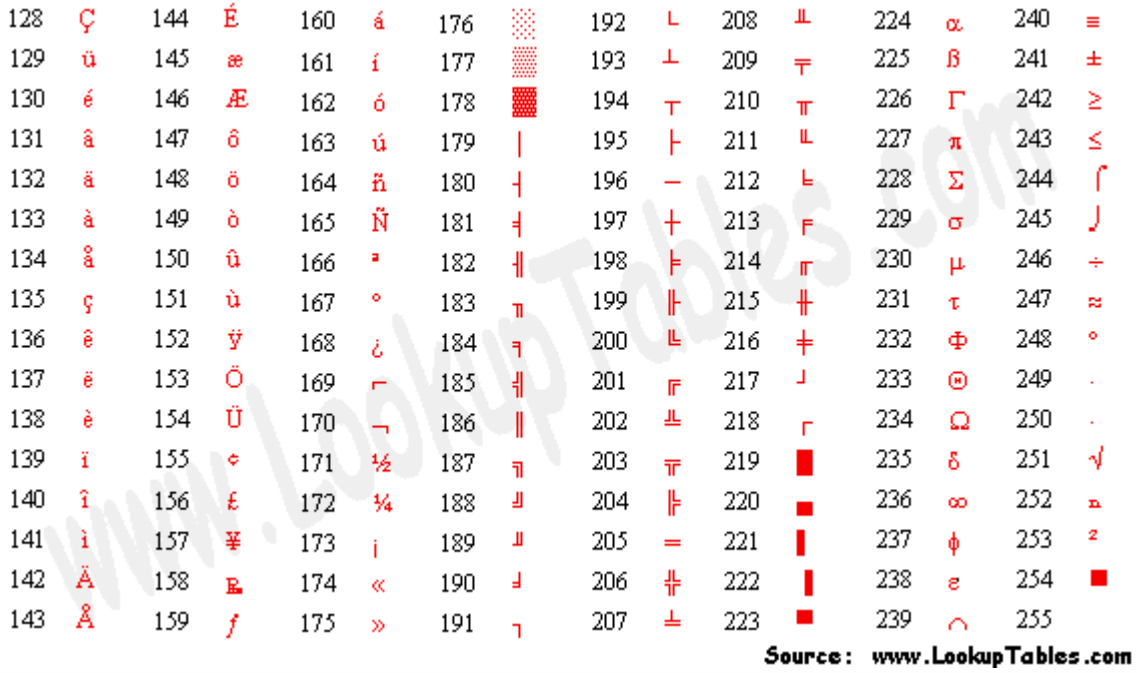
- **ASCII**: ***A****merican* ***S****tandard* ***C****ode for* ***I****nformation* ***I****nterchange*;
- Esquema de codificação utilizado para representar caracteres alfanuméricos e especiais;
- O ASCII original possibilita a codificação de 128 caracteres utilizando um código binário de 7 bits;
- A tabela de codificação é apresentada no próximo slide, em hexadecimal.

Codificação ASCII Original (7 bits)

Character	HEX	Decimal	Character	HEX	Decimal	Character	HEX	Decimal	Character	HEX	Decimal
NUL (null)	0	0	Space	20	32	@	40	64	`	60	96
Start Heading	1	1	!	21	33	A	41	65	a	61	97
Start Text	2	2	"	22	34	B	42	66	b	62	98
End Text	3	3	#	23	35	C	43	67	c	63	99
End Transmit.	4	4	$	24	36	D	44	68	d	64	100
Enquiry	5	5	%	25	37	E	45	69	e	65	101
Acknowlege	6	6	&	26	38	F	46	70	f	66	102
Bell	7	7	`	27	39	G	47	71	g	67	103
Backspace	8	8	(	28	40	H	48	72	h	68	104
Horiz. Tab	9	9	)	29	41	I	49	73	i	69	105
Line Feed	A	10	*	2A	42	J	4A	74	j	6A	106
Vert. Tab	B	11	+	2B	43	K	4B	75	k	6B	107
Form Feed	C	12	,	2C	44	L	4C	76	l	6C	108
Carriage Return	D	13	-	2D	45	M	4D	77	m	6D	109
Shift Out	E	14	.	2E	46	N	4E	78	n	6E	110
Shift In	F	15	/	2F	47	O	4F	79	o	6F	111
Data Link Esc	10	16	0	30	48	P	50	80	p	70	112
Direct Control 1	11	17	1	31	49	Q	51	81	q	71	113
Direct Control 2	12	18	2	32	50	R	52	82	r	72	114
Direct Control 3	13	19	3	33	51	S	53	83	s	73	115
Direct Control 4	14	20	4	34	52	T	54	84	t	74	116
Negative ACK	15	21	5	35	53	U	55	85	u	75	117
Synch Idle	16	22	6	36	54	V	56	86	v	76	118
End Trans Block	17	23	7	37	55	W	57	87	w	77	119
Cancel	18	24	8	38	56	X	58	88	x	78	120
End of Medium	19	25	9	39	57	Y	59	89	y	79	121
Substitue	1A	26	:	3A	58	Z	5A	90	z	7A	122
Escape	1B	27	;	3B	59	[	5B	91	{	7B	123
Form separator	1C	28	<	3C	60	\	5C	92	|	7C	124
Group separator	1D	29	=	3D	61	]	5D	93	}	7D	125
Record Separator	1E	30	>	3E	62	^	5E	94	~	7E	126
Unit Separator	1F	31	?	3F	63	_	5F	95	Delete	7F	127

Códigos ASCII Estendidos (8 bits)

Possibilita a codificação de 256 caracteres: os 128 caracteres da versão original mais os caracteres a seguir:

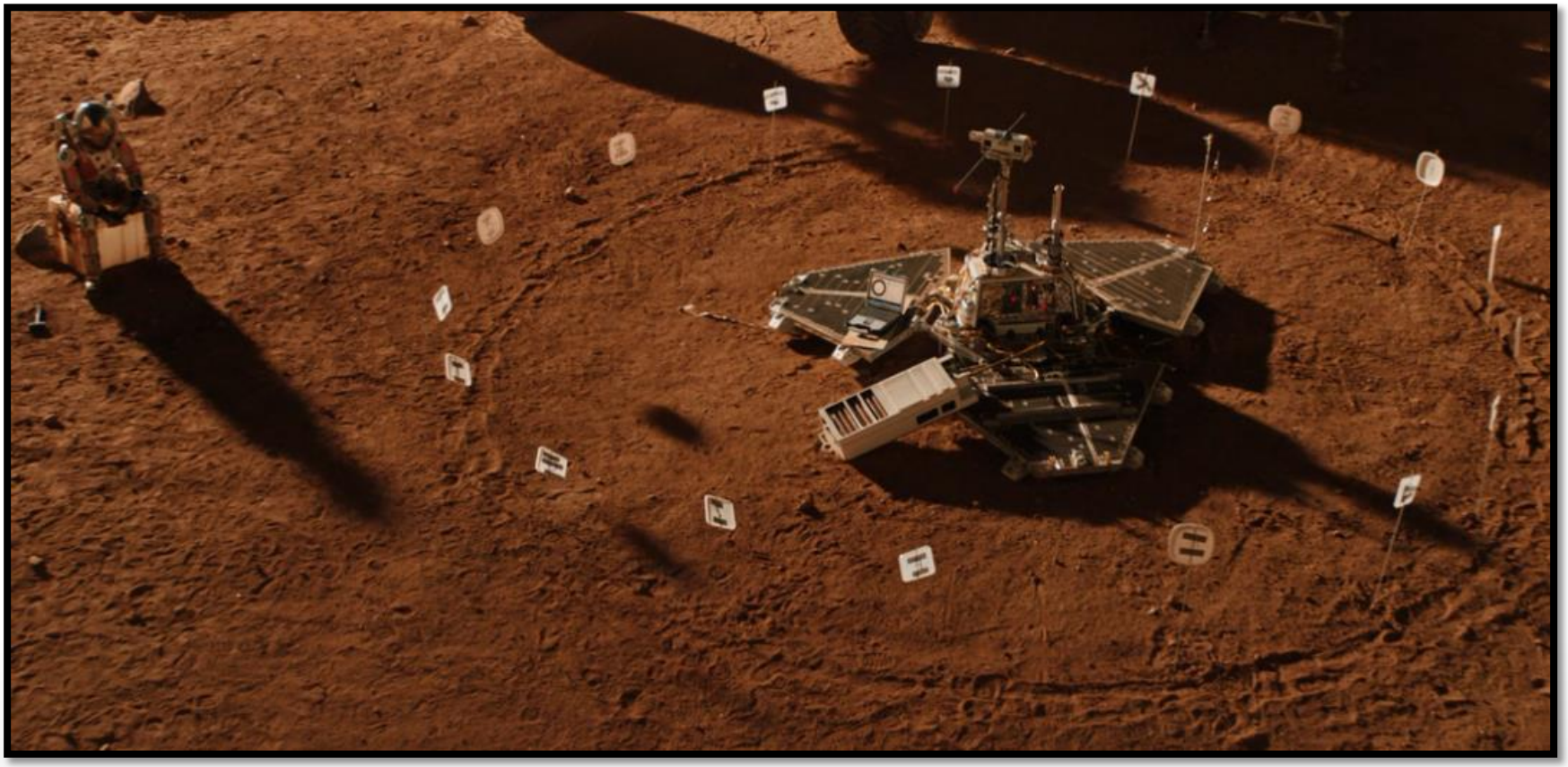


128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

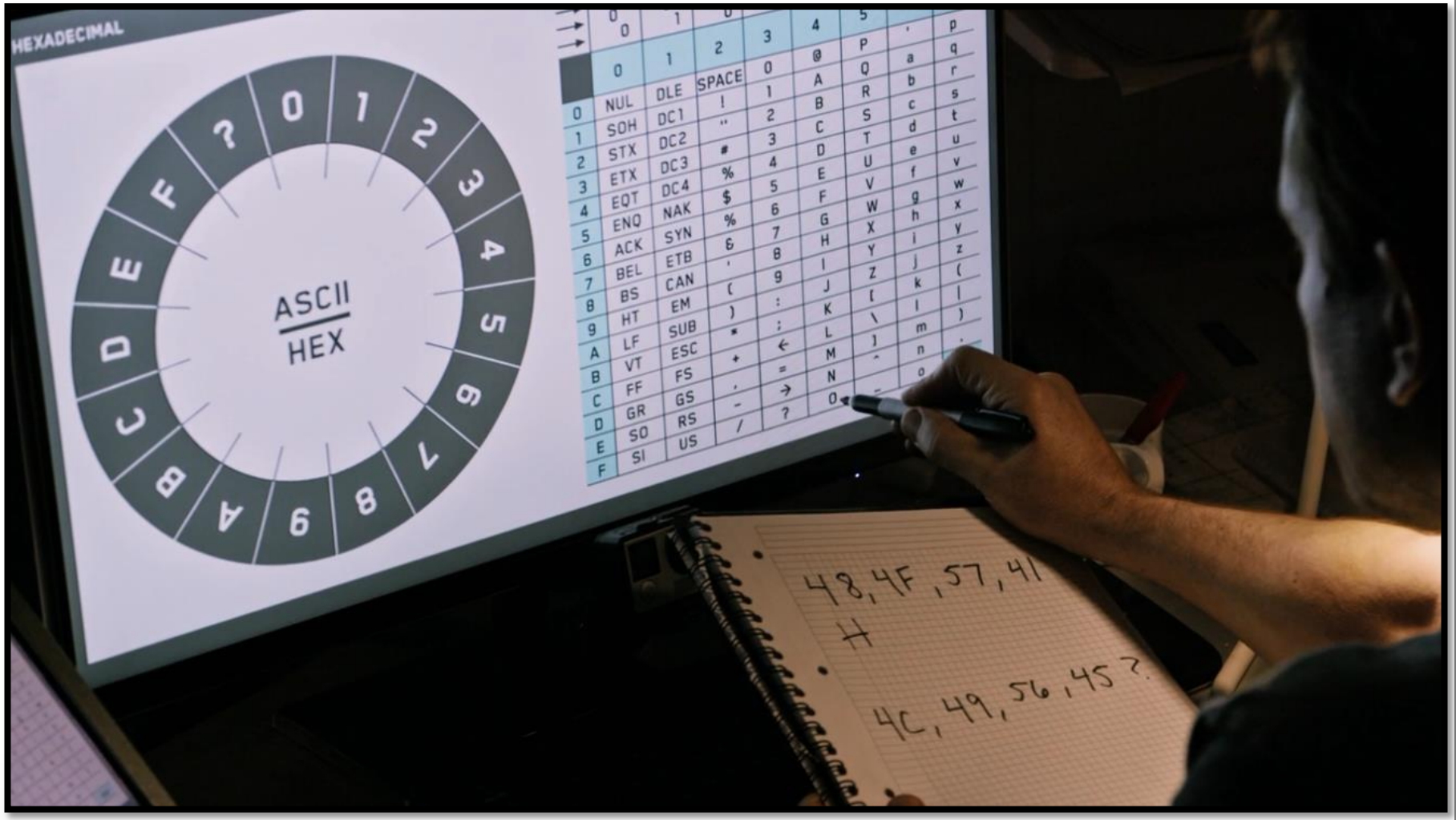
Hexadecimais e ASCII
Curiosidade - Filme *Perdido em Marte*

- Durante uma emergência, a forma mais conveniente encontrada pelo personagem Mark Watney para se comunicar com a terra foi utilizar a movimentação de uma câmera, hexadecimais e o código ASCII ☺

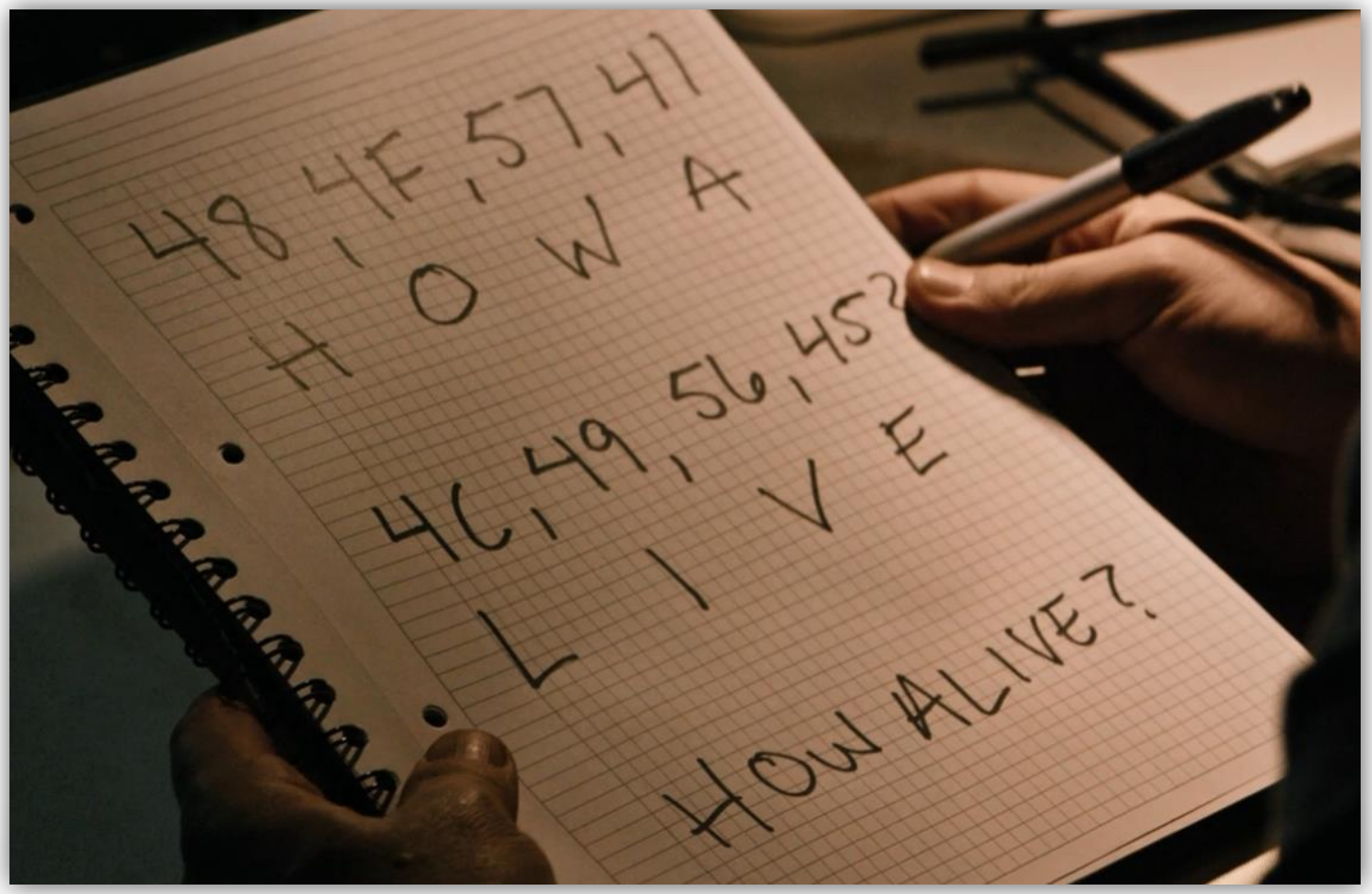


Hexadecimais e ASCII

Curiosidade - Filme *Perdido em Marte*



Hexadecimais e ASCII
Curiosidade - Filme *Perdido em Marte*



Números com Sinal Representados em Excesso-N

Representação Excesso-N

- Forma de representação de números inteiros sinalizados;
- A representação dos números positivos é “deslocada” para frente, de acordo com um *offset* pré-estabelecido, para “ceder espaço” para a representação dos negativos;
- Um valor x é representado pelo número sem sinal que é maior do que x em N unidades;
- Assim, o número zero é representado pelo padrão de bits equivalente ao número N em binário, utilizando uma quantidade de bits pré-determinada;
  - Os padrões que seguem são utilizados para representar os valores +1, +2, +3, etc.;
  - O padrões que antecedem são utilizados para representar os valores -1, -2, -3, etc.;

Representação Excesso-N

- A representação de números sinalizados utilizando a codificação **Excesso-127** com 8 bits é apresentada a seguir;
- Repare que o número 0 é representado pelo binário 01111111 (127)

Número Binário	Interpretação Sem Sinal	**Interpretação em Excesso-127**	Interpretação em Comp. de 2
00000000	0	-127	0
00000001	1	-126	1
…	…	…	…
01111110	126	-1	126
01111111	127	0	127
10000000	128	$+1$	-128
10000001	129	$+2$	-127
…	…	…	…
11111111	255	$+128$	-1

Centro →

- Observe que o **MSB dos negativos é 0**; e o **MSB dos positivos é 1**.

Codificando em Excesso-N

- Considere, como exemplo, a codificação **Excesso-127** com 8 bits;
- Para encontrar a representação de um número x em Excesso-127, basta encontrar o binário puro correspondente a $127 + x$;
- **Exemplo**: codificar $+18$ e -3 utilizando **Excesso-127** com 8 bits;

  - $+18 \Rightarrow 127 + 18 = 145 \Rightarrow 10010001_2$

    $$+18 = 10010001_{\text{exc}_{127}}$$

  - $-3 \Rightarrow 127 - 3 = 124 \Rightarrow 01111100_2$

    $$-3 = 01111100_{\text{exc}_{127}}$$

Decodificando de Excesso-N

- Como na codificação soma-se N, para decodificar basta subtrair N do número correspondente ao binário sem sinal;
- **Exemplo.** Os números a seguir estão codificados em **Excesso-127** com 8 bits. Encontre os valores decimais que tais códigos representam.
  - 00001001
  - 10011001
  - 01100101

$00001001 = 9 \quad \rightarrow \quad 9 - 127 = -118;$

$$00001001_{\text{exc}_{127}} = -118_{10}$$

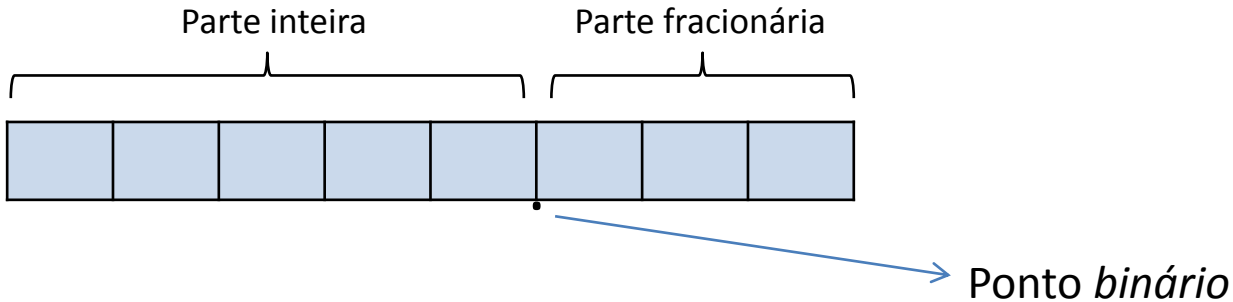
$10011001 = 2^7 + 2^4 + 2^3 + 2^0 = 153 \quad \rightarrow \quad 153 - 127 = +26;$

$$10011001_{\text{exc}_{127}} = +26_{10}$$

Representação de Números Fracionários

Representação em Ponto Fixo *Binário*

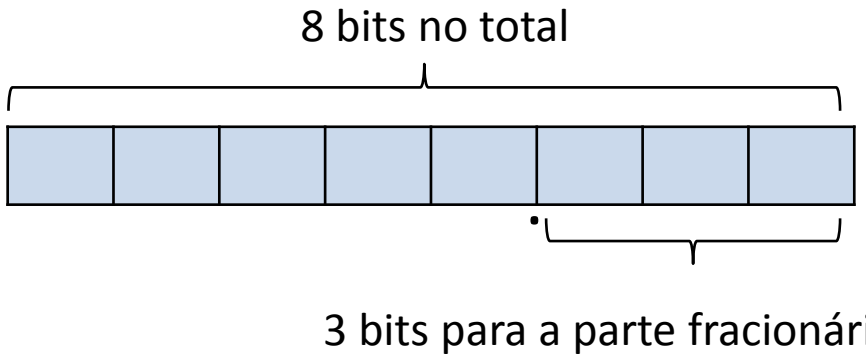
- Uma parte dos bits é utilizada para representar a parte inteira do número; e outra, a parte fracionária (há um número fixo de bits reservado para cada parte);
- Exemplo: número *binário* em ponto fixo (sem sinal) com 8 dígitos, sendo 5 dígitos para a parte inteira e 3 dígitos para a parte fracionária:



  - Maior número que pode ser representado:
    - $11111.111_2 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} = \mathbf{31.875_{10}}$
  - Menor número (exceto o zero):
    - $00000.001 = 0 + 2^{-3} = \mathbf{0.125}$
- Qualquer código que utilizar tal representação precisa ter conhecimento da posição exata do ponto binário.

Representação em Ponto Fixo Binário - Notação

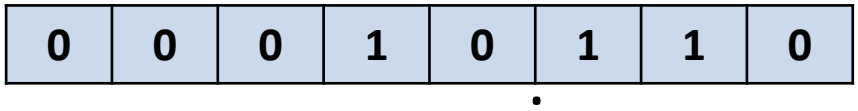
- *Fixed*<n, b>
  - **n**: número total de bits utilizados
  - **b**: posição do **ponto binário**, contando a partir do bit menos significativo
- Exemplo:
  - *Fixed<8, 3>*



OBS: como exemplo, a biblioteca gráfica OpenGL ES disponibiliza o tipo *GLfixed*, representado pela letra *x* e equivalente a ***fixed<32,16>***

Representação em Ponto Fixo Binário - Notação

- Por exemplo, a combinação de bits 10110_2, quando representado como um número em ponto fixo no formato *fixed<8,3>*, denota o número 2.75:



  - 00010.110

    $= 2^1 + 2^{-1} + 2^{-2}$

    $= 2 + 1/2 + 1/4 = 2 + 0.5 + 0.25 = 2.75$

Representação em Ponto Fixo Binário - Notação

- Outras representações para os bits 10110:

Formato	Representação Binária	Valor Decimal
Fixed<8,2>	000101.10	$2^2 + 2^0 + 2^{-1} = 5.5$
Fixed<8,4>	0001.0110	$2^0 + 2^{-2} + 2^{-3} = 1.375$
Fixed<5,1>	1011.0	$2^3 + 2^1 + 2^0 = 11.0$
Fixed<5,0>	10110.	$2^4 + 2^2 + 2^1 = 22$

- $2^{-1} = 1/2 = 0.5$
- $2^{-2} = 1/4 = 0.25$
- $2^{-3} = 1/8 = 0.125$

Representação em Ponto Fixo Binário - Operações

- A adição e a subtração de binários representados em ponto fixo pode ser realizada da mesma forma que em binário puro;
- Exemplo utilizando o formato *fixed(8,2)*

$$
\begin{array}{rl}
 & \overset{1\ 1\ 1\ 1}{} \\
 & 000101.10\ (5.5_{10}) \\
+ & 000011.11\ (3.75_{10}) \\
\hline
 & 001001.01\ (9.25_{10})
\end{array}
$$

Representação em Ponto Fixo Binário

- Vantagens
  - Representação simples;
  - Operações realizadas utilizando a aritmética de inteiros (o hardware desenvolvido para operações com inteiros pode ser reutilizado);
  - Operações mais rápidas (do que as operações em ponto flutuante);
  - Possibilidade de ajustar facilmente o nível de precisão desejado para a parte inteira e para a parte fracionária
- Desvantagens
  - Menor intervalo de valores possíveis (comparado à representação em ponto-flutuante);
  - Impossibilidade de representar certos números com exatidão, como frações de potência de 10 (0.1, 0.2, etc.).

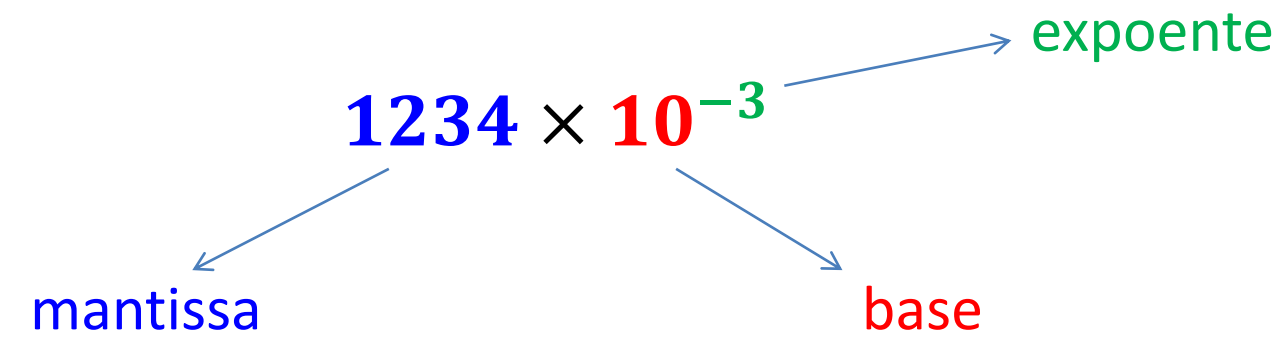
Representação em Ponto Flutuante

Representação em Ponto Flutuante

- Não reserva uma quantidade específica de bits para a parte inteira ou fracionária do número;
- Reserva uma quantidade de bits para a parte principal do número, chamada ***mantissa***, e outra para indicar ***"onde está"*** o ponto binário.

Representação em Ponto Flutuante

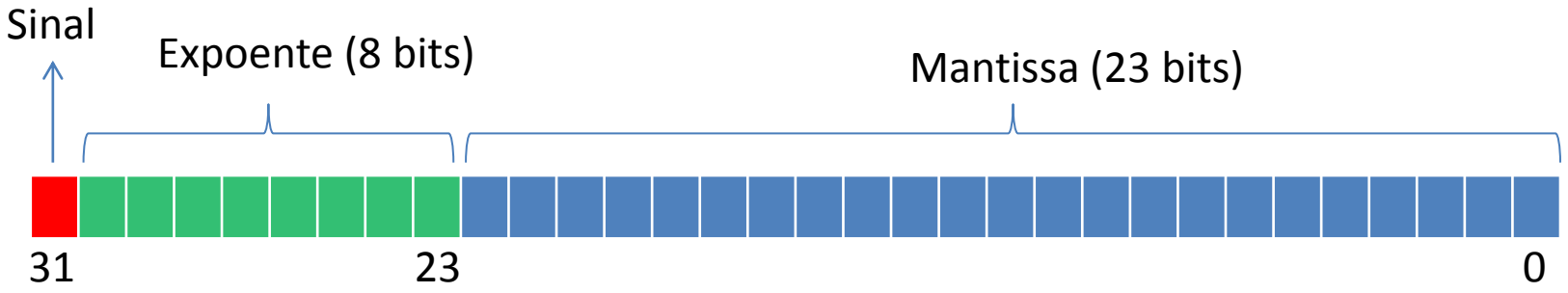
- Baseada na representação de notação científica
- Exemplo de representação de 1,234 em notação científica:



- Outras representações de 1.234 em notação científica
  - 123.4×10^{-2}
  - 12.34×10^{-1}
  - 0.1234×10^{1}

Padrão IEEE 754-1985 (*single, 32 bits*)

- Na representação em ponto flutuante de 32 bits, segundo o padrão IEEE 754-1985 *(float)*, são utilizados:
  - 23 bits para representação da mantissa;
  - 8 bits para representação do expoente (em Excesso-127)
  - 1 bit para representação do sinal.



Padrão IEEE 754-1985 – Passos para Representação de um Número

Os passos as seguir podem ser utilizados para representar um ***número binário fracionário*** em ponto flutuante, segundo o padrão da IEEE (*com 32 bits*):

1. Represente o número binário fracionário em notação científica, deixando 1 bit à esquerda do ponto;
2. Extraia os bits da parte fracionária da mantissa do número obtido no passo anterior. Eles devem ser escritos no espaço de 23 bits da representação em ponto flutuante;
3. Represente o expoente do número obtido no passo 1 em excesso de 127. Os bits encontrados deverão ocupar os 8 bits reservados para o expoente;
4. Defina o bit de sinal: 1 para negativo; 0 para positivo.

Padrão IEEE 754-1985 – Exemplo 1

Exemplo: representar em ponto flutuante, segundo o padrao IEEE 754-1985 (com 32 bits), o número binário fracionário $+\mathbf{1010.01}_2$:

1. Representação em notação científica com 1 bit antes do ponto:

   $\mathbf{+1010.01_2 = 1.01001 \times 2^3}$

   Assim, os bits a serem armazenados para a mantissa são: 01001

   Logo, com 23 bits, temos:

   Mantissa = **01001000000000000000000**

1. Representar o expoente do número 1.01001×2^3 em Excesso-127. Para isso, deve-se somar 127 ao expoente 3 e encontrar o binário correspondente:

   $3 + 127 = 130_{10} = 10000010_2$

   Logo, os 8 bits do expoente são:

   Expoente = **10000010**

2. Definir o bit de sinal (0 para positivo; 1 para negativo).

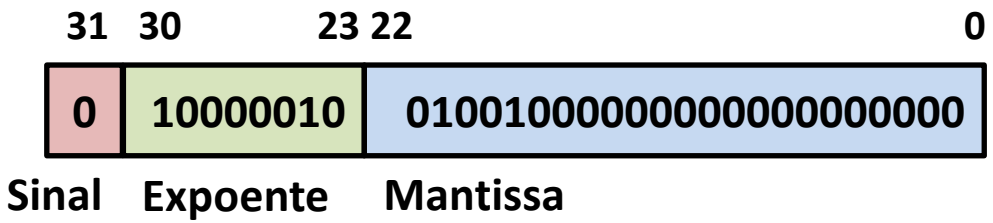
   Sinal = **0**

Padrão IEEE 754-1985 – Exemplo 1

Assim, o número binário **1010.01_2** (10.25_{10}) é representado em ponto flutuante de 32 bits, conforme padrão IEEE 754-1985, como:

$$0 \;\; 10000010 \;\; 01001000000000000000000$$

Ou seja:



Padrão IEEE 754-1985 – Exemplo 2

Em um programa de computador, uma variável do tipo *float* armazena a sequência de bits apresentada a seguir. Sabendo-se que o padrão de representação IEEE 754-1985 foi utilizado, qual é o número decimal efetivamente armazenado na variável?

$$1 \quad 10000011 \quad 00101100000000000000000$$

Número negativo

$10000011_2 = 131_{10}$
Expoente $= 131 - 127 = 4$

Parte fracionária da mantissa
Mantissa $= 1.001011$

$$\begin{aligned} \text{Número} &= -1.001011_2 \times 2^4 \\ &= -10010.11_2 = 2^4 + 2^1 + 2^{-1} + 2^{-2} \\ &= -(16 + 2 + 0.5 + 0.25) \\ &= \mathbf{-18.75_{10}} \end{aligned}$$

Conversão de **Decimal** para ***Float*** *(IEEE 754-1985)*

- Passos para converter um número decimal real para a respectiva representação em ponto flutuante:
  1. Converter a parte inteira para binário;
  2. Converter a parte fracionária para binário seguindo o procedimento descrito a seguir*;
  3. Adicionar as duas partes em binário e seguir o procedimento apresentado anteriormente.

**Multiplique a parte fracionária por 2, resgate o bit da parte inteira do resultado e multiplique novamente a parte fracionária obtida por 2. Repita o procedimento até obter 0 na parte fracionária ou até atingir o limite de precisão desejado (23 bits para o caso de um float IEEE 754)*

Conversão de **Decimal** para ***Float*** *(IEEE 754-1985)*

- **Exemplo**. Representar o número 14.375_{10} em ponto flutuante (*single*).

1. Conversão da parte inteira para binário:

   $14_{10} = 1110_2$

2. Conversão da parte fracionária:

   - 0.375 x 2 = 0.750 → Primeiro bit da parte fracionária será 0
   - 0.750 x 2 = 1.500 → Segundo bit da parte fracionária será 1
   - 0.500 x 2 = 1.000 → Terceiro bit será 1. Nova parte fracionária = .000 indica o término do processo.

   Logo, $0.375_{10} = 0.011_2$

3. Soma das partes inteira e fracionária:

$$1110_2 + 0.011_2 = 1110.011_2$$

(Continuação: seguir o procedimento apresentado nos slides anteriores para representar 1110.011_2 em ponto flutuante)

IEEE 754-1985 – Casos Especiais – Número Zero

- *Sinal* pode ser 0 (zero positivo) ou 1 (zero negativo)
- *Expoente* = 0
- *Mantissa* = 0

+0 (utilizando 32 bits – *single*)

-0 (utilizando 32 bits – *single*)

IEEE 754-1985 – Casos Especiais – Infinito

- *Expoente*: todos os bits iguais a 1
- *Mantissa*: todos os bits iguais a 0
- *Sinal*
  - 0 para +infinito
  - 1 para –infinito

+infinito (utilizando 32 bits – *single*)

-infinito (utilizando 32 bits – *single*)

IEEE 754-1985 – Casos Especiais – NaN

- Representação de NaN *(Not a Number)*
  - *Sinal: 0 ou 1*
  - *Expoente*: todos os bits iguais a 1
  - *Mantissa*: qualquer valor que não seja tudo 0

Representação em Ponto Flutuante

- Vantagens
  - Maior intervalo de valores
- Desvantagens
  - Impossibilidade de representar certos números com exatidão, como frações de potência de 10 (0.1, 0.2, etc.)
  - Problemas com arredondamentos: quanto maior o número, menor a precisão (precisão relativamente pequena para números muito grandes)

Representações em Ponto Binário e Frações de Potência de 10

- Considere o número 1.8_{10}
  - Aproximação utilizando o formato *fixed<8,1>*
    - $0000001.1 = 2^0 + 2^{-1} = 1 + 0.5 = \mathbf{1.5}$
  - Aproximação utilizando o formato *fixed<8,2>*
    - $000001.11 = 2^0 + 2^{-1} + 2^{-2} = 1 + 0.5 + 0.25 = \mathbf{1.75}$
  - Aproximação utilizando o formato *fixed<8,3>*
    - $00001.111 = 2^0 + 2^{-1} + 2^{-2} = 1 + 0.5 + 0.25 + 0.125 = \mathbf{1.875}$
    - $00001.110 = 2^0 + 2^{-1} = 1 + 0.5 + 0.25 = \mathbf{1.75}$
    - $00001.101 = 2^0 + 2^{-1} + 2^{-2} = 1 + 0.5 + 0.125 = \mathbf{1.625}$
- *O número 1.8 nunca será representado com total exatidão utilizando a representação de ponto fixo binário, independentemente da quantidade de bits utilizada na parte fracionária.*
- *Repare que o mesmo problema ocorre nas representações de ponto flutuante (single ou double).*

Referências

- TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas Digitais: princípios e aplicações.** 11.ed. São Paulo: Pearson Prentice Hall, 2011.

- CAPUANO, F. G.; IDOETA, I. V. **Elementos de Eletrônica Digital.** 40.ed. São Paulo: Érica, 2008.

Agradecimentos

- **Prof. Dr. rer. nat. Daniel Duarte Abdala**
- **Prof. Dr. Jamil Salem Barbar**