

Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática – ICEI
Arquitetura de Computadores I

ARQ1 _ Aula_01 - Revisão

Tema: Sistemas de Numeração e representações de dados

Preparação

Como preparação para o início das atividades, recomendam-se

- a.) leitura prévia do resumo teórico, do detalhamento na apostila e referências recomendadas
- b.) estudo e testes dos exemplos
- c.) assistir aos seguintes vídeos:

<https://www.youtube.com/user/henriquencunha/vídeos>

<https://www.youtube.com/watch?v=DJYIndxhcKc>

<https://www.youtube.com/watch?v=Ojd770C2GTk>

Orientação geral:

Atividades previstas como parte da avaliação

Apresentar todas as soluções em apenas um arquivo com formato e extensão para texto (.txt). Sugere-se usar como nome Guia_xx.txt, onde xx indicará o guia, exemplo Guia_01.txt.

Todos os arquivos deverão conter identificações iniciais com o nome e matrícula, no caso de programas, usar comentários.

As implementações e testes dos exemplos em Verilog (.v) fornecidos como pontos de partida, também fazem parte da atividade e deverão ter os códigos fontes entregues **separadamente**, com o código fonte e módulo de testes, a fim de que possam ser compilados e verificados. Sugere-se usar como nomes Guia_01yy.v, onde yy indicará a questão, exemplo Guia_0101.v. As saídas de resultados, opcionalmente, poderão ser copiadas ao final do código, em comentários.

Quaisquer outras anotações, observações ou comentários poderão ser colocadas em arquivo texto (README.txt) acompanhando a entrega.

Atividades extras e opcionais

Outras formas de solução serão **opcionais**; não servirão para substituir as atividades a serem avaliadas. Caso entregues, poderão contar apenas como atividades extras.

Os programas com funções desenvolvidas em C, Java ou Python (c, .java, py), como os modelos usados para verificação automática de testes das respostas; caso entregues, também deverão estar em arquivos **separados**, com o código fonte, a fim de serem compilados e testados.

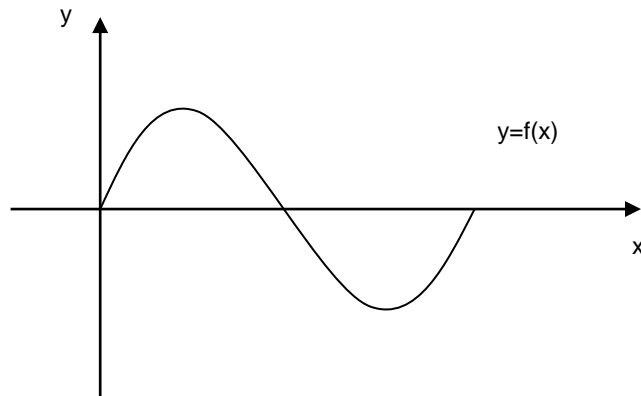
As execuções deverão, preferencialmente, serão testadas mediante uso de redirecionamento de entradas e saídas padrões, cujos dados/resultados deverão ser armazenados em arquivos textos. Os resultados poderão ser anexados ao código, ao final, como comentários.

Planilhas, caso venham a ser solicitadas, deverão ser **programadas** e/ou usar funções nativas. Serão suplementares e opcionais, e deverão ser entregues em formato texto, preferencialmente, com colunas separadas por tabulações ou no formato (**.csv**), acompanhando a solução em texto.

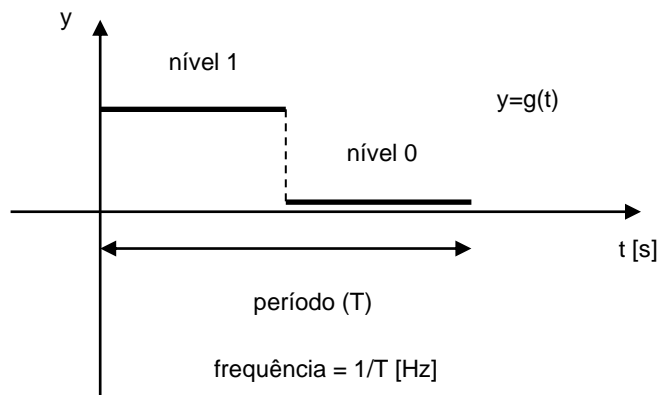
Arquivos em formato (**.pdf**), fotos, cópias de tela ou soluções manuscritas também poderão ser aceitos como recursos suplementares para visualização, mas não servirão como substitutos e **não** terão validade para fins de avaliação.

Representação de dados

Função contínua em um intervalo.



Função discreta em um intervalo.



Computadores Analógicos x Digitais

Analógicos - trabalham com elementos representados por grandezas físicas com estados contínuos (corrente, tensão, pressão, vazão etc.)

Digitais - trabalham com elementos representados por valores numéricos com estados discretos (ou valores distintos distribuídos ao longo de determinado intervalo de tempo)

Sistemas de Numeração

Exemplo:

Sistema decimal

$1x2^7 + 0x2^6 + 1x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 1x2^0$ - forma canônica

$$128 + 0 + 32 + 0 + 0 + 0 + 2 + 1 = 163_{(10)}$$

Sistema binário

1010 0011₍₂₎

- número na base 2

representado apenas com algarismos {0,1}

2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	potências da base 2
128	64	32	16	8	4	2	1	valor equivalente da potência
1	0	1	0	0	0	1	1	coeficientes

Equivalentes em sistemas com potências de 2

1010 0011₍₂₎ = [1010] [0011]₍₁₆₎ = A3₍₁₆₎ e A₍₁₆₎=10 em hexadecimal (grupos de 4)
= 10x16¹ + 3x16⁰ = 163₍₁₀₎ com algarismos {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

1010 0011₍₂₎ = [10] [10] [00] [11]₍₄₎ = 2203₍₄₎ em quaternário (grupos de 2)
= 2x4³ + 2x4² + 0x4¹ + 3x4⁰ = 163₍₁₀₎ com algarismos {0,1,2,3}

1010 0011₍₂₎ = [010] [100] [011]₍₈₎ = 243₍₈₎ em octal (grupos de 3)
= 2x8² + 4x8¹ + 3x8⁰ = 163₍₁₀₎ com algarismos {0,1,2,3,4,5,6,7}

OBS: Caso necessário, completar com zeros (0) para formar grupos de mesmo tamanho.

Conversões entre bases

1.) Converter decimal para binário

Sistema decimal

$$163_{(10)} = 1 \times 10^2 + 6 \times 10^1 + 3 \times 10^0 \quad - \text{na forma canônica}$$

Para converter um valor decimal (base=10) para binário (base=2),
usar divisões sucessivas por 2 e tomar os restos na **ordem inversa**
em que forem calculados:

operação	quociente	resto
163 / 2	= 81	+ 1 (último)
81 / 2	= 40	+ 1
40 / 2	= 20	+ 0
20 / 2	= 10	+ 0
10 / 2	= 5	+ 0
5 / 2	= 2	+ 1
2 / 2	= 1	+ 0
1 / 2	= 0	+ 1 (primeiro)

Sistema binário

$$1010\ 0011_{(2)} \quad - \text{número na base 2}$$

ou

2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	potências da base 2
128	64	32	16	8	4	2	1	valor equivalente da potência
1	0	1	0	0	0	1	1	coeficientes

2.) Converter decimal para binário

Para converter um valor binário (base=2) para decimal (base=10),
usar a soma dos produtos de cada algarismo pela potência da base
equivalente à posição:

Sistema binário

$$1010\ 0011_{(2)} \quad - \text{número na base 2}$$

Sistema decimal

$$1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad - \text{forma canônica}$$

$$128 + 0 + 32 + 0 + 0 + 0 + 2 + 1 = 163_{(10)}$$

3.) Converter decimal para base 4 (quaternário)

Para converter um valor decimal para a base 4 (quaternário):

operação	quociente	resto
$163 / 4 =$	40	+ 3 (último)
$40 / 4 =$	10	+ 0
$10 / 4 =$	2	+ 2
$2 / 4 =$	0	+ 2 (primeiro)

Sistema quaternário

$2203_{(4)}$

- número na base 4

4.) Converter decimal para base 8 (octal)

Para converter um valor decimal para a base 8 (octal):

operação	quociente	resto
$163 / 8 =$	20	+ 3 (último)
$20 / 8 =$	2	+ 4
$2 / 8 =$	0	+ 2 (primeiro)

Sistema octal

$243_{(8)}$

- número na base 8

5.) Converter decimal para base 16 (hexadecimal)

Para converter um valor decimal para a base 16 (hexadecimal):

operação	quociente	resto
$163 / 16 =$	10	+ 3 (último)
$10 / 16 =$	0	+ 10 (primeiro, substituindo pelo algarismo A=10)

Sistema hexadecimal

$A3_{(16)}$

- número na base 16

6.) Converter da base 4 para decimal

Sistema quaternário

$$\begin{aligned} 2203_{(2)} &= 2 \times 4^3 + 2 \times 4^2 + 0 \times 4^1 + 3 \times 4^0 \\ &= 128 + 32 + 0 + 3 = 163_{(10)} \end{aligned}$$

- número na base 4 na forma canônica

7.) Converter da base 8 para decimal

Sistema octal

$$\begin{aligned} 243_{(8)} &= 2 \times 8^2 + 4 \times 8^1 + 3 \times 8^0 \\ &= 128 + 32 + 3 = 163_{(10)} \end{aligned}$$

- número na base 8 na forma canônica

8.) Converter da base 16 para decimal

Sistema hexadecimal

$$\begin{aligned} A3_{(16)} &= (A=10) \times 16^1 + 3 \times 16^0 && \text{- número na base 16 forma canônica} \\ &= 160 + 3 = 163_{(10)} \end{aligned}$$

9.) Converter entre bases potências múltiplas sem passar para decimal

As bases que são potências múltiplas de outra compartilham propriedades especiais, como a possibilidade de conversões entre elas, sem passar pela base decimal:

Sistema binário (base=2) para quaternário (base=4=2²):

$$1010\ 0011_{(2)} = [10][10][00][11]_{(4)} = 2203_{(4)} \quad \begin{array}{l} \text{agrupar de 2 em 2} \\ \text{e substituir pelos dígitos equivalentes} \end{array}$$

Sistema binário (base=2) para quaternário (base=8=2³):

$$1010\ 0011_{(2)} = [\underline{0}10][100][011]_{(8)} = 243_{(8)} \quad \begin{array}{l} \text{agrupar de 3 em 3} \\ \text{e substituir pelos dígitos equivalentes} \end{array}$$

OBS: Caso necessário, completar com zeros para formar os grupos.

Sistema binário (base=2) para quaternário (base=16=2⁴):

$$1010\ 0011_{(2)} = [1010][0011]_{(16)} = A3_{(16)} \text{ e } A_{(16)}=10 \quad \begin{array}{l} \text{agrupar de 4 em 4} \\ \text{e substituir pelos dígitos equivalentes} \end{array}$$

ou usar uma tabela com as principais equivalências entre essas bases de numeração.

X ₍₁₀₎ decimal	X ₍₂₎ binário	X ₍₄₎ quaternário	X ₍₈₎ octal	X ₍₁₆₎ hexadecimal
00	0000 0000	00 00	000	00
01	0000 0001	00 01	001	01
02	0000 0010	00 02	002	02
03	0000 0011	00 03	003	03
04	0000 0100	00 10	004	04
05	0000 0101	00 11	005	05
06	0000 0110	00 12	006	06
07	0000 0111	00 13	007	07
08	0000 1000	00 20	000	08
09	0000 1001	00 21	011	09
10	0000 1010	00 22	012	0A
11	0000 1011	00 23	013	0B
12	0000 1100	00 30	014	0C
13	0000 1101	00 31	015	0D
14	0000 1110	00 32	016	0E
15	0000 1111	00 33	017	0F

Representações de potências de 2.

x	2 ^x	X ₍₁₀₎	X ₍₂₎	X ₍₄₎	X ₍₈₎	X ₍₁₆₎
0	2 ⁰	1	1	1	1	1
1	2 ¹	2	10	2	2	2
2	2 ²	4	100	10	4	4
3	2 ³	8	1000	20	10	8
4	2 ⁴	16	1 0000	100	20	10
5	2 ⁵	32	10 0000	200	40	20
6	2 ⁶	64	100 0000	1000	100	40
7	2 ⁷	128	1000 0000	2000	200	80
8	2 ⁸	256	1 0000 0000	10000	400	100
9	2 ⁹	512	10 0000 0000	20000	1000	200
10	2 ¹⁰	1024	100 0000 0000	100000	2000	400

Termos associados à representação de dados em binário

Termo	Quantidade	Observação
<i>bit</i>	1	" <i>binary digit</i> " – dígito binário (0 ou 1)
<i>nibble</i>	4 bits	dígito hexadecimal equivalente (semiocteto)
<i>byte</i>	8 bits	octeto (Werner Buchholz, 1956) – unidade de armazenamento
<i>word</i>	xx bits	dependente do sistema (ex.: 14, 16, 32, 54, 64 etc.)
kiloBytes (kB)	1024 Bytes	(ex.: arquivo texto)
MegaBytes (MB)	1024 kiloBytes (kB)	1 048 576 bytes (ex.: arquivo mp3)
GigaBytes (GB)	1024 MegaBytes (MB)	1 073 741 824 bytes (ex.: filme)
TeraBytes (TB)	1024 GigaBytes (GB)	1 099 511 627 776 bytes (ex.: 800 filmes)
PetaBytes (PB)	1024 TeraBytes (TB)	1 125 899 906 842 624 bytes (ex.: acervo do Google)
ExaBytes (EB)	1024 PetaBytes (PB)	(ex.: acervo da Internet)
ZetaBytes (ZB)	1024 ExaBytes (EB)	
YottaBytes (YB)	1024 ZetaBytes (ZB)	

Representação de símbolos por códigos equivalentes (Tabela ASCII - 8 bits)

(2) 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

(8) 0_000 0_001 0_010 0_011 0_100 0_101 0_110 0_111 1_000 1_001 1_010 1_011 1_100 1_101 1_110 1_111

		(10)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
		(16)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0_00_	0	0	^@	^A	^B	^C	^D	^E	^F	^G	^H	^I	^J	^K	^L	^M	^N	^O
0001	0_01_	16	1	^P	^Q	^R	^S	^T	^U	^V	^W	^X	^Y	^Z	^[^\	^]	^^	^
0010	0_10_	32	2		!	“	#	\$	%	&	‘	()	*	+	,	-	.	/
0011	0_11_	48	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	1_00_	64	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	1_01_	80	5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0110	1_10_	96	6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	1_11_	112	7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	←

Exemplos:

'0' = 0011_0000₍₂₎ = 30₍₁₆₎ = 00_110_000₍₂₎ = 060₍₈₎ = 48₍₁₀₎

'A' = 0100_0001₍₂₎ = 41₍₁₆₎ = 01_000_001₍₂₎ = 101₍₈₎ = 65₍₁₀₎

'a' = 0110_0001₍₂₎ = 61₍₁₆₎ = 01_100_001₍₂₎ = 141₍₈₎ = 97₍₁₀₎

"Computador" = 43 6F 6D 70 75 74 61 64 6F 72₍₁₆₎

Sistemas de Numeração – Operações aritméticas

Exemplos:

1.) Adição

Sistema binário

Relações fundamentais:

$$0_{(2)} + 0_{(2)} = 0_{(2)}$$

$$0_{(2)} + 1_{(2)} = 1_{(2)}$$

$$1_{(2)} + 0_{(2)} = 1_{(2)}$$

$$1_{(2)} + 1_{(2)} = 10_{(2)} \quad (\text{zero e "vai-um" para a próxima potência})$$

Aplicação:

$$1111 \leftarrow \text{"vai-um"}$$

$$101101_{(2)} \leftarrow \text{operando 1}$$

$$+ 111_{(2)} \leftarrow \text{operando 2}$$

$$\hline 110100_{(2)} \leftarrow \text{resultado}$$

Sistema quaternário

Aplicação:

$$1111$$

$$11 \leftarrow \text{"vai-um" (excessos de 4)}$$

$$101101_{(2)}$$

$$231_{(4)} \leftarrow \text{operando 1}$$

$$+ 111_{(2)}$$

$$+ 13_{(4)} \leftarrow \text{operando 2}$$

$$\hline 110100_{(2)}$$

$$\hline 310_{(4)} \leftarrow \text{resultado}$$

Sistema octal

Aplicação:

$$1111$$

$$1 \leftarrow \text{"vai-um" (excessos de 8)}$$

$$101101_{(2)}$$

$$55_{(8)} \leftarrow \text{operando 1}$$

$$+ 111_{(2)}$$

$$+ 7_{(8)} \leftarrow \text{operando 2}$$

$$\hline 110100_{(2)}$$

$$\hline 64_{(8)} \leftarrow \text{resultado}$$

Sistema hexadecimal

Aplicação:

$$\begin{array}{r}
 \begin{array}{r}
 1111 \\
 101101_{(2)} \\
 + \quad 111_{(2)} \\
 \hline
 110100_{(2)}
 \end{array}
 \qquad
 \begin{array}{l}
 1 \quad \leftarrow \text{"vai-um"} \text{ (excessos de 16)} \\
 2D_{(16)} \leftarrow \text{operando 1} \\
 + \quad 7_{(16)} \leftarrow \text{operando 2} \\
 \hline
 34_{(16)} \leftarrow \text{resultado}
 \end{array}
 \end{array}$$

2.) Subtração

Relações fundamentais:

$$\begin{aligned}
 0_{(2)} - 0_{(2)} &= 0_{(2)} \\
 0_{(2)} - 1_{(2)} &= ??? \\
 1_{(2)} - 0_{(2)} &= 1_{(2)} \\
 1_{(2)} - 1_{(2)} &= 0_{(2)} \\
 10_{(2)} - 1_{(2)} &= 01_{(2)} \text{ (zero e "vem-um" para a potência considerada)} \\
 100_{(2)} - 1_{(2)} &= 011_{(2)} \text{ (zero e "vem-um" para as potências necessitadas)}
 \end{aligned}$$

Aplicação:

$$\begin{array}{r}
 \begin{array}{r}
 \begin{array}{r}
 1 \\
 101101_{(2)} \\
 - \quad 111_{(2)} \\
 \hline
 0_{(2)}
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 101\textcolor{red}{0}01_{(2)} \\
 - \quad 111_{(2)} \\
 \hline
 0_{(2)}
 \end{array}
 \qquad
 \begin{array}{r}
 (10) \\
 101\textcolor{red}{0}(\textcolor{red}{0})1_{(2)} \\
 - \quad 111_{(2)} \\
 \hline
 10_{(2)}
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 100(\textcolor{red}{10})01_{(2)} \\
 - \quad 111_{(2)} \\
 \hline
 100110_{(2)}
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 101101_{(2)} \\
 - \quad 111_{(2)} \\
 \hline
 100110_{(2)}
 \end{array}
 \end{array}
 \qquad
 \begin{array}{l}
 \leftarrow \text{"vem-um"} \\
 \leftarrow \text{operando 1} \\
 \leftarrow \text{operando 2} \\
 \leftarrow \text{resultado}
 \end{array}
 \end{array}$$

OBS:

Quando se "toma emprestado" na potência seguinte, um valor unitário é debitado na potência que "empresta", e "creditado" na potência que o recebe, compensada a diferença entre essas potências.

3.) Multiplicação

Sistema binário

Relações fundamentais:

$$0_{(2)} * 0_{(2)} = 0_{(2)}$$

$$0_{(2)} * 1_{(2)} = 0_{(2)}$$

$$1_{(2)} * 0_{(2)} = 0_{(2)}$$

$$1_{(2)} * 1_{(2)} = 1_{(2)}$$

Aplicação:

$$\begin{array}{r} 101101_{(2)} \quad \leftarrow \text{operando 1} \\ * \quad 101_{(2)} \quad \leftarrow \text{operando 2} \\ \hline \begin{array}{r} 1111 \\ 101101 \\ + 000000- \\ 101101-- \end{array} \\ \hline 11100001_{(2)} \quad \leftarrow \text{resultado} \end{array}$$

4.) Divisão

Sistema binário

Aplicação:

$$\begin{array}{r} 11100001_{(2)} \mid \underline{101_{(2)}} \\ - 101 \quad 1_{(2)} \\ \hline 010 \end{array}$$

$$\begin{array}{r} 11100001_{(2)} \mid \underline{101_{(2)}} \\ - 101 \quad 101_{(2)} \\ \hline 01000 \\ - 101 \\ \hline 00011 \end{array}$$

$$\begin{array}{r} 11100001_{(2)} \mid \underline{101_{(2)}} \\ - 101 \quad 10110_{(2)} \\ \hline 01000 \\ - 101 \\ \hline 000110 \\ - 101 \\ \hline 00000101 \end{array}$$

$$\begin{array}{r} 11100001_{(2)} \mid \underline{101_{(2)}} \\ - 101 \quad 10_{(2)} \\ \hline 0100 \end{array}$$








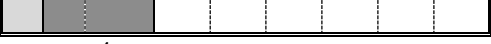

$$\begin{array}{r} 11100001_{(2)} \mid \underline{101_{(2)}} \\ - 101 \quad 1011_{(2)} \\ \hline 01000 \\ - 101 \\ \hline 000110 \\ - 101 \\ \hline 0000010 \end{array}$$

$$\begin{array}{r} 11100001_{(2)} \mid \underline{101_{(2)}} \\ - 101 \quad 101101_{(2)} \\ \hline 01000 \\ - 101 \\ \hline 000110 \\ - 101 \\ \hline 00000101 \\ - 101 \\ \hline 00000000 \end{array}$$

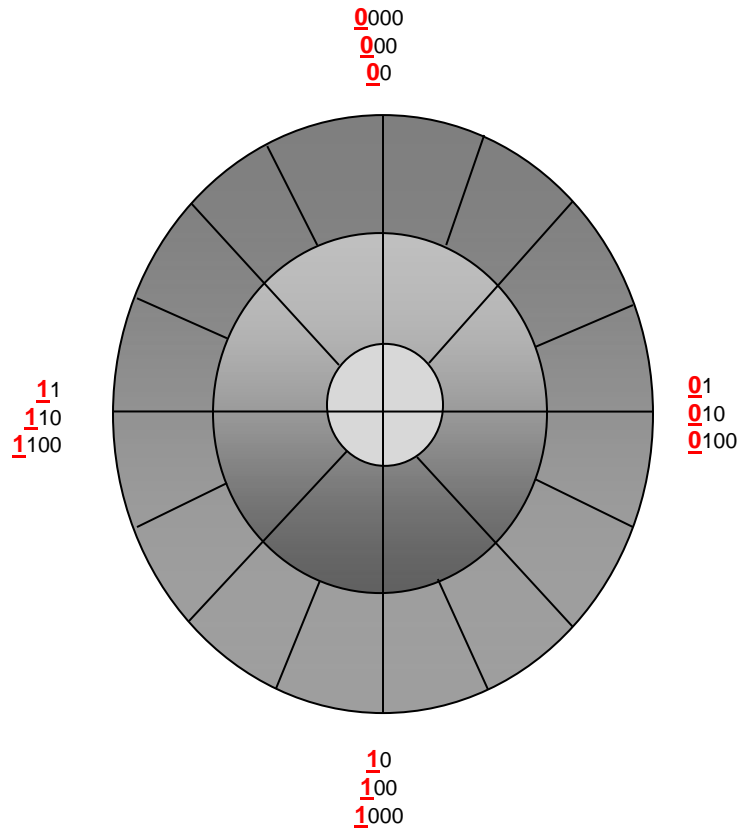
Sistemas de Numeração – Representações de dados

A representação de dados numéricos necessita, por vezes, utilizar uma indicação especial para sinal (positivo e negativo). Para isso, é comum reservar o primeiro bit (o mais à direita para isso), em valores inteiros ou reais. Entretanto, a representação de valores negativos necessitará de ajustes a fim de que as operações aritméticas produzam resultados coerentes.

Representações para tipos de dados comuns (em Java)

Tipos		Intervalo	Tamanho
boolean <i>false, true</i>		[false:true]	1 byte
byte 0, 0x00		[-128 : 127] [0 : 255] (sem sinal)	1 byte
char '0', '\u0000'		[0 : 65535]	2 bytes (Unicode)
short 0 ± a		[-32768 : 32767] (sinal+amplitude)	2 bytes
int 0 ± a		$[-2^{31} : 2^{31}-1]$ (sinal+amplitude)	4 bytes
long 0L ± a		$[-2^{63} : 2^{63}-1]$ (sinal+amplitude)	8 bytes
float 0.0f ± e 1.m IEEE754		$[-3.4e^{-38} : 3.4e^{38}]$ (sinal+amplitude+1)	4 bytes .mantissa)
double 0.0, 0.0e0 ± e 1.m IEEE754		$[-1.7e^{-308} : 1.7e^{308}]$ (sinal+amplitude+1.	8 bytes .mantissa)
String "", "0", <i>null</i>			n bytes

Representação binária dependente do número de bits.



A representação binária depende da quantidade de bits disponíveis e dos formatos escolhidos.

Para os valores inteiros, por exemplo, pode-se utilizar o formato em que o primeiro bit, à esquerda, para o sinal e o restante para a amplitude, responsável pela magnitude (grandeza) do valor representado.

Exemplo:

$$5_{(10)} = 101_{(2)}$$

$$+5_{(10)} = \underline{0}101_{(2)}$$

$$-5_{(10)} = \underline{1}101_{(2)}$$

Essa representação, contudo, não é conveniente para realizar operações, pois ao adicionar ambos, obtém-se:

$$+5_{(10)} = \underline{0}101_{(2)}$$

$$-5_{(10)} = \underline{1}101_{(2)}$$

$$\hline 0_{(10)} = (\underline{1})0010_{(2)}$$

o que ultrapassa a quantidade de bits originalmente escolhida e, obviamente, não é igual a zero em sua amplitude.

Complemento de 1

Uma das possíveis representações para valores negativos pode ser aquela onde se invertem os valores individuais de cada bit.

Exemplo:

$$5_{(10)} = 101_{(2)}$$

$$+5_{(10)} = \underline{0}101_{(2)}$$

$$-5_{(10)} = \underline{1}010_{(2)} \text{ (complemento de 1)}$$

Essa representação, contudo, também não é conveniente para realizar operações, pois ao adicionar ambos, obtém-se:

$$+5_{(10)} = \underline{0}101_{(2)}$$

$$-5_{(10)} = \underline{1}010_{(2)}$$

$$-0_{(10)} = \underline{1}111_{(2)} \rightarrow +0_{(10)} = \underline{0}000_{(2)}$$

o que mantém a quantidade de bits originalmente escolhida, mas gera duas representações para zero (-0) e (+0), o que requer ajustes adicionais nas operações.

Complemento de 2

Outra das possíveis representações para valores negativos pode ser aquela onde se invertem os valores individuais de cada bit, e acrescenta-se mais uma unidade ao valor encontrado, buscando completar o que falta para atingir a próxima potência da base.

Exemplo:

$$5_{(10)} = 101_{(2)}$$

$$+5_{(10)} = \underline{0}101_{(2)}$$

$$-5_{(10)} = \underline{1}010_{(2)} \text{ (complemento de 1, ou } C_1(5))$$

$$-5_{(10)} = \underline{1}011_{(2)} \text{ (complemento de 2, ou } C_2(5))$$

Essa representação é bem mais conveniente para realizar operações, pois ao adicionar ambos, obtém-se:

$$+5_{(10)} = \underline{0}101_{(2)}$$

$$-5_{(10)} = \underline{1}011_{(2)}$$

$$0_{(10)} = (\underline{1})\underline{0}000_{(2)}$$

com uma única representação para zero, mas com um excesso (1) que não é comportado pela quantidade de bits originalmente escolhida. Porém, se desprezado esse excesso, o valor poderá ser considerado correto, com a ressalva de que a quantidade de bits deverá ser rigorosamente observada (ou haverá risco de transbordamento – OVERFLOW).

Para efeitos práticos, o tamanho da representação deverá ser sempre indicado, e as operações deverão ajustar os operandos para a mesma quantidade de bits (de preferência, a maior possível).

Exemplo:

$$5_{(10)} = 101_{(2)}$$

$$+5_{(10)} = \underline{0}101_{(2)}$$

$$-5_{(10)} = \underline{1}010_{(2)} \text{ (complemento de 1, com 4 bits ou } C_{14} \text{ (5))}$$

$$-5_{(10)} = \underline{1}011_{(2)} \text{ (complemento de 2, com 4 bits ou } C_{24} \text{ (5))}$$

logo,

$$C_{15} (5) = C_1 (\underline{0}0101_{(2)}) = \underline{1}1010_{(2)}$$

$$C_{25} (5) = C_2 (\underline{0}0101_{(2)}) = \underline{1}1011_{(2)}$$

$$C_{18} (5) = C_1 (\underline{0}0000101_{(2)}) = \underline{1}1111010_{(2)}$$

$$C_{28} (5) = C_2 (\underline{0}0000101_{(2)}) = \underline{1}1111011_{(2)}$$

De modo inverso, dado um valor em complemento de 2, se desejado conhecer o equivalente positivo, basta retirar uma unidade e substituir os valores individuais de cada dígito binário.

Exemplo:

$$\underline{1}011_{(2)} \text{ (complemento de 2, com 4 bits)}$$

$$\underline{1}011_{(2)} - 1 = \underline{1}010_{(2)} \text{ e invertendo } \underline{0}101_{(2)} = +5_{(10)}$$

$$\text{logo, } \underline{1}011_{(2)} = -5_{(10)}$$

Portanto, para diferentes quantidades de bits:

$$\underline{1}1011_{(2)} = \underline{1}1010_{(2)} = \underline{0}0101_{(2)} = 5_{(10)}$$

$$\underline{1}1111011_{(2)} = \underline{1}1111010_{(2)} = \underline{0}0000101_{(2)} = 5_{(10)}$$

Subtração mediante uso de complemento

Operar a subtração mediante uso de complemento pode ser mais simples do que realizar a operação diretamente, como visto anteriormente.

Aplicação:

	1		(10)		1	← "vem-um"
101101 ₍₂₎	101 <u>0</u> 01 ₍₂₎	101 <u>0</u> (<u>0</u>)1 ₍₂₎	1 <u>00</u> (<u>10</u>)01 ₍₂₎	101101 ₍₂₎		← operando 1
- 111 ₍₂₎	- 111 ₍₂₎	- 1 1 1 ₍₂₎	- 1 11 ₍₂₎	- 111 ₍₂₎		← operando 2
0 ₍₂₎	0 ₍₂₎	1 0 ₍₂₎	100 1 10 ₍₂₎	100110 ₍₂₎		← resultado

OBS:

Quando se "toma emprestado" na potência seguinte, um valor unitário é debitado na potência que "empresta", e "creditado" na potência que o recebe, compensada a diferença entre essas potências.

Aplicação do complemento:

Para aplicar o complemento, a primeira providência é normalizar os operandos na mesma quantidade de bits, reservado o bit de sinal.

101101 ₍₂₎	→ 0 101001 ₍₂₎
- 111 ₍₂₎	→ - 0 000111 ₍₂₎

Em seguida, calcular e substituir o subtraendo pelo complemento:

$$C2 (0 000111_{(2)}) = C1 (0 000111_{(2)}) + 1_{(2)} = 1 111000_{(2)} + 1_{(2)} = 1 111001_{(2)}$$

101101 ₍₂₎	→ 0 101001 ₍₂₎
- 111 ₍₂₎	→ - 1 111001 ₍₂₎

Para finalizar, operar a **soma** dos operandos, respeitando a quantidade de bits:

	(1) 1 1 1 1	← "vai-um"
101101 ₍₂₎	→ 0 101101 ₍₂₎	
- 111 ₍₂₎	+ 1 111001 ₍₂₎	
	0 100110	

Observar que o bit que exceder a representação deverá ser desconsiderado, por não haver onde acomodá-lo. Ainda poderá haver erro por transbordamento (OVERFLOW).

Atividades:

01.) Fazer as conversões de decimal para binário:

01a.) manualmente (em arquivo texto)

- a.) $29_{(10)} = X_{(2)}$
- b.) $53_{(10)} = X_{(2)}$
- c.) $751_{(10)} = X_{(2)}$
- d.) $312_{(10)} = X_{(2)}$
- e.) $365_{(10)} = X_{(2)}$

01b.) mediante uso de um programa em Verilog (em arquivo fonte com extensão (.v))

```
/*
  Guia_0101.v
  999999 - Xxx Yyy Zzz
*/
module Guia_0101;
// define data
  integer x = 13; // decimal
  reg [7:0] b = 0; // binary (bits - little endian)

// actions
  initial
    begin : main
      $display ( "Guia_0101 - Tests" );
      $display ( "x = %d" , x );
      $display ( "b = %8b", b );
      b = x;
      $display ( "b = %8b", b );
    end // main

endmodule // Guia_0101
```

Notas:

Para compilar:	iverilog -o Guia_0101.vvp Guia_0101.v
Para executar:	vvp Guia_0101.vvp

Extras / Opcionais:

01c.) mediante uso de uma função dec2bin(x) (em linguagem de programação: Python, Java, ...)

01d.) mediante uso de uma planilha (APENAS se usar programação com funções nativas)

Exemplo:

[illegible]

02.) Fazer as conversões de binário para decimal:

02a.) manualmente

a.) $10001_{(2)} = X_{(10)}$

b.) $10111_{(2)} = X_{(10)}$

c.) $10100_{(2)} = X_{(10)}$

d.) $101010_{(2)} = X_{(10)}$

e.) $110011_{(2)} = X_{(10)}$

02b.) mediante uso de um programa em Verilog

```
/*
  Guia_0102.v
  999999 - Xxx Yyy Zzz
*/
module Guia_0102;
// define data
  integer x = 0; // decimal
  reg [7:0] b = 8'b0001101; // binary (bits - little endian)

// actions
  initial
    begin : main
      $display ( "Guia_0102 - Tests" );
      $display ( "x = %d" , x );
      $display ( "b = %8b", b );
      x = b;
      $display ( "b = %d", x );
    end // main

endmodule // Guia_0102
```

Extras / Opcionais:

02c.) mediante uso de uma função bin2dec(x) (em linguagem de programação: Python, Java)

02d.) mediante uso de uma planilha (APENAS se usar programação com funções nativas)

Exemplo:

[illegible]

03.) Fazer as conversões de decimal para a base indicada:

03a.) manualmente

- a.) $45_{(10)} = X_{(4)}$
- b.) $66_{(10)} = X_{(8)}$
- c.) $79_{(10)} = X_{(16)}$
- d.) $151_{(10)} = X_{(16)}$
- e.) $781_{(10)} = X_{(16)}$

03b.) mediante uso de um programa em Verilog

```
/*
  Guia_0103.v
  999999 - Xxx Yyy Zzz
*/
module Guia_0103;
// define data
  integer x = 13; // decimal
  reg [7:0] b = 0; // binary

// actions
  initial
    begin : main
      $display ( "Guia_0103 - Tests" );
      $display ( "x = %d" , x );
      $display ( "b = %8b", b );
      b = x;
      $display ( "b = %B (2) = %o (8) = %x (16) = %X (16)", b, b, b, b );
    end // main

endmodule // Guia_0103
```

Extras / Opcionais:

03c.) mediante uso de uma função dec2bin(x) (em linguagem de programação: Python, Java)

03d.) mediante uso de uma planilha (APENAS se usar programação com funções nativas)

Exemplo:

Usar divisões sucessivas pela base (2) e juntar os restos na ordem inversa:

$X_{(10)}$	$X_{(10)}/2$ (quociente)	$X_{(10)}\%2$ (resto)	$X_{(2)}$
163	81	1	1
81	40	1	1 1
40	20	0	0 11
20	10	0	0 011
10	5	0	0 0011
5	2	1	1 00011
2	1	0	0 100011
1	0	1	1 0100011
0	(parar)		<u>10100011</u>

04.) Fazer as conversões de base entre as bases indicadas por agrupamento:

04a.) manualmente

- a.) $10100_{(2)} = X_{(4)}$
- b.) $11001_{(2)} = X_{(8)}$
- c.) $100101_{(2)} = X_{(16)}$
- d.) $101001_{(2)} = X_{(8)}$
- e.) $100101_{(2)} = X_{(4)}$

04b.) mediante uso de um programa em Verilog

```
/*
  Guia_0104.v
  999999 - Xxx Yyy Zzz
*/
module Guia_0104;
// define data
  integer x = 13; // decimal
  reg [7:0] b = 0; // binary

// actions
  initial
    begin : main
      $display ( "Guia_0104 - Tests" );
      $display ( "x = %d" , x );
      $display ( "b = %8b", b );
      b = x;
      $display ( "b = [%4b] [%4b] = %x %x", b[7:4], b[3:0], b[7:4], b[3:0] ); // agrupamento
    end // main

endmodule // Guia_0104
```

OBS.: O uso de agrupamento será melhor apresentado no próximo guia.

Também poderá ser usado para as bases 4 e 8,
com outras quantidades (2 e 3, respectivamente).

Extras / Opcionais

DICAS: Para conferir, comparar os valores decimais equivalentes.

04c.) mediante uso de uma função bin2dec(x) (em linguagem de programação: Python, Java)

04d.) mediante uso de uma planilha (APENAS se usar programação com funções nativas)

Exemplos:

$X_{(10)}$	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1	Σ	nova base
163	1	0	1	0	0	0	1	1	128+32+2+1	10100011₍₂₎
	1 2^1	0 2^0	1 2^1	0 2^0	0 2^1	0 2^0	1 2^1	1 2^0	(grupos de 2) (em evidência)	$X_{(4)}$ (usar dígitos)
	4^3 64		4^2 16		4^1 4		4^0 1			
163	$(1*2+ 0)=2$		$(1*2+ 0)=2$		$(0+ 0)=0$		$(1*2+ 1)=3$		$2*64+2*16+0*4+3$	2203₍₄₎
	1 2^3	0 2^2	1 2^1	0 2^0	0 2^3	0 2^2	1 2^1	1 2^0	(grupos de 4) (em evidência)	$X_{(16)}$ (usar dígitos)
	16^1 16				16^0 1					
163	$(1*8+ 0+ 1*2+ 0)=10$				$(0+ +0+ 1*2+ 1)=3$				$(10=A)*16+3$	A3₍₁₆₎
	1 2^1	0 2^0	1 2^2	0 2^1	0 2^0	0 2^2	1 2^1	1 2^0	(grupos de 3) (em evidência)	$X_{(8)}$ (usar dígitos)
	8^2 64		8^1 8		8^1 1					
163	$(1*2+ 0)=2$		$(1*4+ 0+ 0)=4$		$(0+ 1*2+ 1)=3$				$2*64+4*8+3$	243₍₈₎

05.) Converter entre símbolos e códigos de representação alfanumérico (ASCII):

05a.) manualmente

- a.) "PUC-MG" = $X_{(16_ASCII)}$
- b.) "2025-02" = $X_{(16_ASCII)}$
- c.) "Belo Horizonte" = $X_{(2_ASCII)}$
- d.) 124 141 162 144 145 ₍₈₎ = $X_{(ASCII)}$
- e.) 4E 6F 69 784 65 ₍₁₆₎ = $X_{(ASCII)}$

05b.) mediante uso de um programa em Verilog

```
/*
  Guia_0105.v
  999999 - Xxx Yyy Zzz
*/
module Guia_0105;
  // define data
  integer x = 13; // decimal
  reg [7:0] b ; // binary
  reg [0:2][7:0] s = "PUC"; // char array[3] (3x8 bits - little Endian)

  // actions
  initial
  begin : main
    $display ( "Guia_0105 - Tests" );
    $display ( "x = %d" , x );
    $display ( "b = %8b", b );
    $display ( "s = %s" , s );
    b = x;
    $display ( "b = [%4b] [%4b] = %h %h", b[7:4], b[3:0], b[7:4], b[3:0] );
    s[0] = "-";
    s[1] = 8'b01001101; // 'M'
    s[2] = 71; // 'G'
    $display ( "s = %s" , s );
  end // main

endmodule // Guia_0105
```

Extras / Opcionais

05c.) mediante uso de funções ASCII2hex(x) e hex2ASCII(xx)
(em linguagem de programação: Python, Java)

05d.) mediante uso de uma planilha
(APENAS se usar programação com funções intrínsecas para buscar códigos)

Modelos de programas para servir como unidades de testes

Modelo em Java

```
/**
    Arquitetura de Computadores I - Guia_01.java
    999999 - Xxx Yyy Zzz
*/
public class Guia_01
{
    /**
        Contador de erros.
    */
    private static int errors = 0;

    /**
        Testar se dois valores sao iguais.
        @param x - primeiro valor
        @param y - segundo valor
    */
    public static void test_equals ( Object x, Object y )
    {
        if ( (""+x).compareTo(""+y) != 0 )
            errors = errors + 1;
    } // end test_equals ( )

    /**
        Exibir o total de erros.
        @return mensagem com o total de erros
    */
    public static String test_report ( )
    {
        return ( ""+errors );
    } // end test_report ( )

    /**
        Converter valor decimal para binario.
        @return binario equivalente
        @param value - valor decimal
    */
    public static String dec2bin ( int value )
    {
        return ( "0" );
    } // end dec2bin ( )
}
```

```

/*
    Converter valor binario para decimal.
    @return decimal equivalente
    @param value - valor binario
*/
public static int bin2dec ( String value )
{
    return ( -1 );
} // end bin2dec ( )

/*
    Converter valor decimal para base indicada.
    @return base para a conversao
    @param value - valor decimal
*/
public static String dec2base ( int value, int base )
{
    return ( "0" );
} // end dec2base ( )

/*
    Converter valor binario para base indicada.
    @return valor equivalente na base indicada
    @param value - valor binario
    @param base - para a conversao
*/
public static String bin2base ( String value, int base )
{
    return ( "0" );
} // end bin2base ( )

/*
    Converter valor em ASCII para hexadecimal.
    @return hexadecimal equivalente
    @param value - caractere(s) em codigo ASCII
*/
public static String ASCII2hex ( String value )
{
    return ( "0" );
} // end ASCII2hex ( )

```

```
/*  
    Converter valor em hexadecimal para ASCII.  
    @return caractere(s) em codigo ASCII  
    @param value - hexadecimal equivalente(s)  
*/  
public static String hex2ASCII ( String value )  
{  
    return ( "0" );  
} // end hex2ASCII ( )
```

```

/*
Acao principal.
*/
public static void main ( String [ ] args )
{
    System.out.println ( "Guia_01 - Java Tests " );
    System.out.println ( "999999 - Xxx Yyy Zzz " );
    System.out.println ( );

    test_equals ( dec2bin ( 29 ), "10101" );
    test_equals ( dec2bin ( 53 ), "10101" );
    test_equals ( dec2bin ( 751 ), "10101" );
    test_equals ( dec2bin ( 312 ), "10101" );
    test_equals ( dec2bin ( 365 ), "10101" );
    System.out.println ( "1. errorTotalReport = "+test_report ( ) );

    test_equals ( bin2dec ( "10001" ), 0 );
    test_equals ( bin2dec ( "10111" ), 0 );
    test_equals ( bin2dec ( "10100" ), 0 );
    test_equals ( bin2dec ( "101010" ), 0 );
    test_equals ( bin2dec ( "110011" ), 0 );
    System.out.println ( "2. errorTotalReport = "+test_report ( ) );

    test_equals ( dec2base ( 45, 4 ), "10101" );
    test_equals ( dec2base ( 66, 8 ), "10101" );
    test_equals ( dec2base ( 79, 16 ), "10101" );
    test_equals ( dec2base ( 151, 16 ), "10101" );
    test_equals ( dec2base ( 781, 16 ), "10101" );
    System.out.println ( "3. errorTotalReport = "+test_report ( ) );

    test_equals ( bin2base ( "10100", 4 ), "10101" );
    test_equals ( bin2base ( "11001", 8 ), "10101" );
    test_equals ( bin2base ( "100101", 16 ), "10101" );
    test_equals ( bin2base ( "101001", 8 ), "10101" );
    test_equals ( bin2base ( "100101", 4 ), "10101" );
    System.out.println ( "4. errorTotalReport = "+test_report ( ) );

    test_equals ( ASCII2hex ( "PUC-MG" ), "10101" );
    test_equals ( ASCII2hex ( "2025-02" ), "10101" );
    test_equals ( ASCII2hex ( "Belo Horizonte" ), "10101" );
    // OBS.: A seguir, exemplos apenas para os primeiros, acrescentar todos os outros códigos propostos!
    test_equals ( hex2ASCII ( "124 ..." ), "10101" ); // OBS.: 124 e' o primeiro octal (0o164)!
    test_equals ( hex2ASCII ( "4E ..." ), "10101" ); // OBS.: 4E e' o primeiro hexadecimal (0x4E)!
    System.out.println ( "5. errorTotalReport = "+test_report( ) );

    System.out.print ( "\n\nApertar ENTER para terminar." );
    System.console ( ).readLine ( );
} // end main
} // end class

```

Modelo em Python

```
'''
    Arquitetura de Computadores I - Guia_01.py
    999999 - Xxx Yyy Zzz
'''
'''
    Contador de erros.
'''
errors = 0;

'''
    Testar se dois valores sao iguais.
    @param x - primeiro valor
    @param y - segundo valor
'''
def test_equals ( x, y ):
    global errors;
    if ( str(x) != str(y) ):
        errors = errors + 1;
# end test_equals ( )

'''
    Exibir o total de erros.
    @return mensagem com o total de erros
'''
def test_report ( ):
    return ( ""+str(errors) );
# end test_report ( )

'''
    Converter valor decimal para binario.
    @return binario equivalente
    @param value - valor decimal
'''
def dec2bin ( value ):
    return ( "0" );
# end dec2bin ( )

'''
    Converter valor binario para decimal.
    @return decimal equivalente
    @param value - valor binario
'''
def bin2dec ( value ):
    return ( -1 );
# end bin2dec ( )
```

```

'''
    Converter valor decimal para base indicada.
    @return base para a conversao
    @param value - valor decimal
'''

def dec2base ( value, base ):
    return ( "0" );
# end dec2base ( )

'''
    Converter valor binario para base indicada.
    @return valor equivalente na base indicada
    @param value - valor binario
    @param base - para a conversao
'''

def bin2base ( value, base ):
    return ( "0" );
# end bin2base ( )

'''
    Converter valor em ASCII para hexadecimal.
    @return hexadecimal equivalente
    @param value - caractere(s) em codigo ASCII
'''

def ASCII2hex ( value ):
    return ( "0" );
# end ASCII2hex ( )

'''
    Converter valor em hexadecimal para ASCII.
    @return caractere(s) em codigo ASCII
    @param value - hexadecimal equivalente(s)
'''

def hex2ASCII ( value ):
    return ( "0" );
# end hex2ASCII ( )

```



```

'''
    Acao principal.
'''

def main ( ):
    print ( "Guia_01 - Python Tests" );
    print ( "999999 - Xxx Yyy Zzz" );
    print ( );

    test_equals ( dec2bin ( 29 ), "10101" );
    test_equals ( dec2bin ( 53 ), "10101" );
    test_equals ( dec2bin ( 751 ), "10101" );
    test_equals ( dec2bin ( 312 ), "10101" );
    test_equals ( dec2bin ( 365 ), "10101" );
    print ( "1. errorTotalReport = "+test_report ( ) );

    test_equals ( bin2dec ( "10001" ), 0 );
    test_equals ( bin2dec ( "10111" ), 0 );
    test_equals ( bin2dec ( "10100" ), 0 );
    test_equals ( bin2dec ( "101010" ), 0 );
    test_equals ( bin2dec ( "110011" ), 0 );
    print ( "2. errorTotalReport = "+test_report ( ) );

    test_equals ( dec2base ( 45, 4 ), "10101" );
    test_equals ( dec2base ( 66, 8 ), "10101" );
    test_equals ( dec2base ( 79, 16 ), "10101" );
    test_equals ( dec2base ( 151, 16 ), "10101" );
    test_equals ( dec2base ( 781, 16 ), "10101" );
    print ( "3. errorTotalReport = "+test_report ( ) );

    test_equals ( bin2base ( "10100", 4 ), "10101" );
    test_equals ( bin2base ( "11001", 8 ), "10101" );
    test_equals ( bin2base ( "100101", 16 ), "10101" );
    test_equals ( bin2base ( "101001", 8 ), "10101" );
    test_equals ( bin2base ( "100101", 4 ), "10101" );
    print ( "4. errorTotalReport = "+test_report ( ) );

    test_equals ( ASCII2hex ( "PUC-MG" ), "10101" );
    test_equals ( ASCII2hex ( "2025-02" ), "10101" );
    test_equals ( ASCII2hex ( "Belo Horizonte" ), "10101" );
    // OBS.: A seguir, exemplos apenas para os primeiros, acrescentar todos os outros códigos propostos!
    test_equals ( hex2ASCII ( "124 ..." ), "10101" ); // OBS.: 124 e' o primeiro octal (0o164)!
    test_equals ( hex2ASCII ( "4E ..." ), "10101" ); // OBS.: 4E e' o primeiro hexadecimal (0x4E)!
    print ( "5. errorTotalReport = "+test_report ( ) );

    print ( "\n\nApertar ENTER para terminar." );
    input ( );
# end main ( )
if __name__ == "__main__":
    main( );

```