










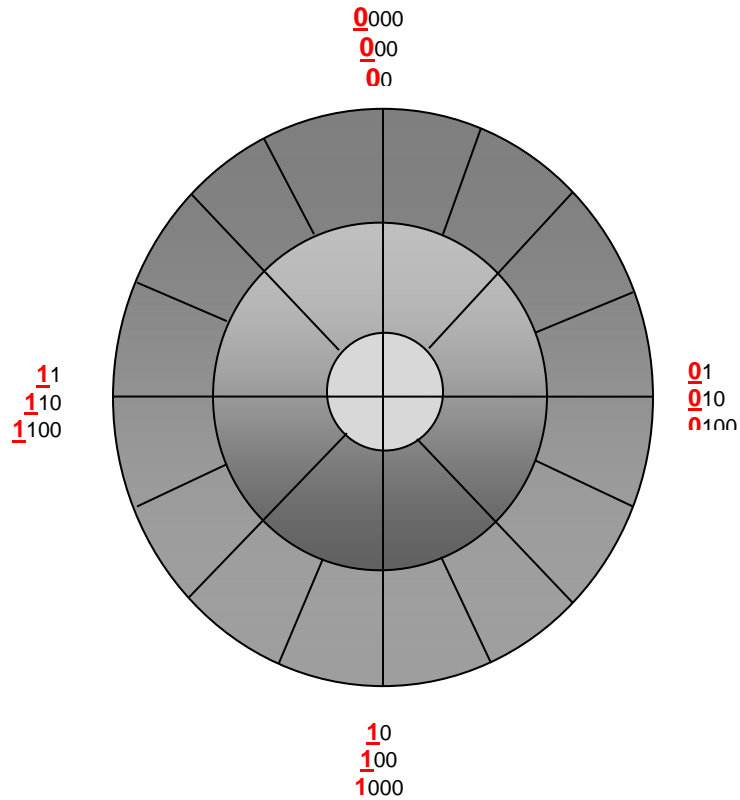
Sistemas de Numeração – Representações de dados com sinal

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

Representações para tipos de dados comuns (em Java)

Tipos	Intervalo	Tamanho
boolean  false	[false:true]	1 byte
byte  0, 0x00	[-128 : 127]	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=8 1.m IEEE754	$[-1.0e^{-38} : 1.0e^{38}]$ (sinal+amplitude+1)	4 bytes .mantissa)
double  0.0, 0.0e0 ± e=11 1.m IEEE754	$[-1.0e^{-308} : 1.0e^{308}]$ (sinal+amplitude+1)	8 bytes .mantissa)
String  "", "0", null		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 binários 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 binários 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:

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

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

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

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

logo,

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

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

$$C_{1,8} (+5) = C_1 (\underline{0}0000101_{(2)}) = \underline{1}1111010_{(2)}$$

$$C_{2,8} (+5) = C_2 (\underline{0}0000101_{(2)}) = \underline{1}1111011_{(2)}$$

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:

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

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

OBS.: A representação do sinal dependerá sempre da quantidade de bits.

Portanto, recomenda-se usar, pelo menos, o maior tamanho dos operandos; se possível, usar representação com um tamanho ainda maior que esse.

Subtração mediante uso de complemento

Operar a subtração mediante uso de complemento pode ser mais simples do que realizar a operação direta com empréstimos ("vem-um"), 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₎	10 <u>0</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 usando a mesma quantidade de bits (ou superior), reservado o bit de sinal.

101101 ₍₂₎	→ <u>0</u> 101101 ₍₂₎
- 111 ₍₂₎	→ - <u>0</u> 000111 ₍₂₎

Em seguida, calcular e substituir apenas o subtraendo pelo seu complemento de 2:

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

101101 ₍₂₎	→ <u>0</u> 101101 ₍₂₎
- 111 ₍₂₎	→ - <u>1</u> 111001 ₍₂₎

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

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

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

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:

<http://www.youtube.com/watch?v=ZwRfnmXY7VY>

<http://www.youtube.com/watch?v=XGA17irIEtc>

http://www.youtube.com/watch?v=Zi3Bg6_ihjg

<https://www.youtube.com/watch?v=q1QwC3YIHG0>

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

<https://www.youtube.com/watch?v=wd7g7a7AjBM>

<https://www.youtube.com/watch?v=1VSFyremNeY>

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

Orientação geral:

Atividades previstas como parte da avaliação

Apresentar todas as soluções em apenas um arquivo com formato 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**, a fim de que possam ser compilados e testados.

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, como comentários.

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 utilizadas, 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 serão aceitos como recursos suplementares para visualização, e **não** terão validade para fins de avaliação.

01.) Determinar os complementos para os valores e as quantidades de bits indicadas:
DICA: Ajustar primeiro o tamanho, antes de calcular o complemento ($C_{1,n}$ ou $C_{2,n}$).

Exemplos:

$$C_{1,5}(1101_{(2)}) = X_{(2)}$$

$$1101_{(2)} = 0\ 1101 = 1\ 0010_{(2)}$$

$$C_{2,5}(1101_{(2)}) = X_{(2)}$$

$$1101_{(2)} = 0\ 1101 = 1\ 0010 + 1 = 1\ 0011_{(2)}$$

01a.) manualmente

$$\text{a.) } C_{1,6}(1100_{(2)}) = X_{(2)}$$

$$\text{b.) } C_{1,8}(1110_{(2)}) = X_{(2)}$$

$$\text{c.) } C_{2,6}(100101_{(2)}) = X_{(2)}$$

$$\text{d.) } C_{2,7}(100101_{(2)}) = X_{(2)}$$

$$\text{e.) } C_{2,8}(110101_{(2)}) = X_{(2)}$$

01b.) mediante uso de um programa em Verilog

```
/*
  Guia_0301.v
  999999 - Xxx Yyy Zzz
*/
module Guia_0301;
// define data
  reg [7:0] a = 8'b000_1010 ; // binary
  reg [6:0] b = 8'b000_101  ; // binary
  reg [5:0] c = 8'b001_01   ; // binary
  reg [7:0] d = 0           ; // binary
  reg [6:0] e = 0           ; // binary
  reg [5:0] f = 0           ; // binary
// actions
  initial
  begin : main
    $display ( "Guia_0301 - Tests" );
    d = ~a+1;
    $display ( "a = %8b -> C1(a) = %8b -> C2(a) = %8b", a, ~a, d );
    e = ~b+1;
    $display ( "b = %7b -> C1(b) = %7b -> C2(b) = %7b", b, ~b, e );
    f = ~c+1;
    $display ( "c = %6b -> C1(c) = %6b -> C2(c) = %6b", c, ~c, f );
  end // main

endmodule // Guia_0301
```

Extras / Opcionais:

01c.) mediante uso de funções C1(nbits, x) e C2(nbits, x)
(em linguagem de programação: Python, Java)

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

02.) Determinar os complementos para os valores e as quantidades indicadas:

DICA: Para valores em outras bases, converter para binário, primeiro;
encontrar a representação em complemento, e retornar à base,
caso necessário.

Exemplos:

$$C_{1,5}(E_{(16)}) = X_{(2)}$$

$$E_{(16)} = 1110_{(2)} = 01110 = 10001_{(2)}$$

$$C_{2,5}(E_{(16)}) = X_{(2)}$$

$$E_{(16)} = 1110_{(2)} = 01110 = 10001 + 1 = 10010_{(2)}$$

02a.) manualmente

$$a.) C_{1,6}(132_{(4)}) = X_{(2)}$$

$$b.) C_{1,8}(4B_{(16)}) = X_{(2)}$$

$$c.) C_{2,6}(213_{(4)}) = X_{(2)}$$

$$d.) C_{2,10}(154_{(8)}) = X_{(2)}$$

$$e.) C_{2,8}(B8_{(16)}) = X_{(2)}$$

02b.) mediante uso de um programa em Verilog

```
/*
  Guia_0302.v
  999999 - Xxx Yyy Zzz

*/
module Guia_0302;
// define data
  reg [7:0] a = 8'h0a ; // hexadecimal
  reg [6:0] b = 8'o15 ; // octal
  reg [5:0] c = 13    ; // decimal
  reg [7:0] d = 0     ; // binary
  reg [6:0] e = 0     ; // binary
  reg [5:0] f = 0     ; // binary
// actions
  initial
  begin : main
    $display ( "Guia_0302 - Tests" );
    d = ~a+1;
    $display ( "a = %8b -> C1(a) = %8b -> C2(a) = %8b", a, ~a, d );
    e = ~b+1;
    $display ( "b = %7b -> C1(b) = %7b -> C2(b) = %7b", b, ~b, e );
    f = ~c+1;
    $display ( "c = %6b -> C1(c) = %6b -> C2(c) = %6b", c, ~c, f );
  end // main

endmodule // Guia_0302
```

Extras / Opcionais:

02c.) mediante uso de funções C1(nbits, x, basex) e C2(nbits, x, basex).
(em linguagem de programação: Python, Java)

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

03.) Determinar os valores positivos equivalentes aos complementos de dois indicados:
DICA: Subtrair uma unidade, antes de inverter cada bit.

Exemplo:

$$\underline{1}001_{(2)} = X_{(2)}$$

$$C_{2,5}(\underline{1}001_{(2)}) = \underline{0}111_{(2)} = +7_{(10)}$$

03a.) manualmente

$$a.) \underline{1}0111_{(2)} = X_{(10)}$$

$$b.) \underline{1}10001_{(2)} = X_{(10)}$$

$$c.) \underline{1}00101_{(2)} = X_{(2)}$$

$$d.) \underline{1}011101_{(2)} = X_{(2)}$$

$$e.) \underline{1}010011_{(2)} = X_{(16)}$$

03b.) mediante uso de um programa em Verilog

```
/*
  Guia_0303.v
  999999 - Xxx Yyy Zzz
*/
module Guia_0303;
// define data
  reg signed [7:0] a = 8'b1111_1010; // binary
  reg signed [6:0] b = 8'b1111_101  ; // binary
  reg signed [5:0] c = 8'b1111_10   ; // binary
  reg signed [7:0] d = 0              ; // binary
  reg signed [6:0] e = 0              ; // binary
// actions
  initial
  begin : main
    $display ( "Guia_0303 - Tests" );
    d = ~a+1;
    e = ~(a-1);
    $display ( "a = %8b -> C1(a) = %8b -> C2(a) = %8b = %d = %d", a, ~a, d, d, e );
    d = ~b+1;
    e = ~(b-1);
    $display ( "b = %7b -> C1(b) = %7b -> C2(b) = %7b = %d = %d", b, ~b, d, d, e );
    d = ~c+1;
    e = ~(c-1);
    $display ( "c = %6b -> C1(c) = %6b -> C2(c) = %6b = %d = %d", c, ~c, d, d, e );
  end // main  end // main

endmodule // Guia_0303
```

Extras / Opcionais:

03c.) mediante uso de uma função sbin2dec(x)
(em linguagem de programação: Python, Java)

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

04.) Fazer as operações indicadas mediante uso de complemento de dois:

DICAS: Levar todas as representações para binário, com a mesma quantidade de bits, a menor necessária para acomodar a parte significativa e incluir o sinal.

Substituir apenas os subtraendos pelos complementos equivalentes e somar.

Voltar os resultados às bases originais.

Exemplo:

$$32_{(4)} - 1001_{(2)} = X_{(8)}$$

$$1110_{(2)} - 1001_{(2)} = 1110 + C_{2,4}(1001) = 1110 + 0111 = (1) 0101_{(2)} = 5_{(8)}$$

04a.) manualmente

- a.) $11101_{(2)} - 1011_{(2)} = X_{(2)}$ (OBS: Colocar do mesmo tamanho, primeiro)
- b.) $101,0101_{(2)} - 10,11_{(2)} = X_{(2)}$ (OBS.: Alinhar as vírgulas, primeiro, antes de operar)
- c.) $321_{(4)} - 213_{(4)} = X_{(4)}$
- d.) $461_{(8)} - 247_{(8)} = X_{(8)}$
- e.) $7C4_{(16)} - B1D_{(16)} = X_{(16)}$

04b.) mediante uso de um programa em Verilog

```
/*
  Guia_0304.v
  999999 - Xxx Yyy Zzz
*/
module Guia_0304;
// define data
  reg signed [7:0] a = 8'b1111_1010; // binary
  reg signed [6:0] b = 8'b1111_101  ; // binary
  reg signed [5:0] c = 8'b0001_10   ; // binary
  reg signed [7:0] d = 0              ; // binary
  reg signed [6:0] e = 0              ; // binary
  reg signed [5:0] f = 0              ; // binary
// actions
  initial
  begin : main
    $display ( "Guia_0304 - Tests" );
    $display ( "a = %8b = %d", a, a );
    $display ( "b = %8b = %d", b, b );
    $display ( "c = %8b = %d", c, c );
    d = a-b;
    $display ( "d = a-b = %8b-%8b = %8b = %d", a, b, d, d );
    d = b-a;
    $display ( "d = b-a = %8b-%8b = %8b = %d", b, a, d, d );
    d = a-c;
    $display ( "d = a-c = %8b-%8b = %8b = %d", a, c, d, d );
    d = c-a;
    $display ( "d = c-a = %8b-%8b = %8b = %d", c, a, d, d );
  end // main

endmodule // Guia_0304
```

Extras / Opcionais:

04c.) mediante uso de uma função baseEval(x, "-", y, base)
(em linguagem de programação: Python, Java)

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

05.) Executar as operações abaixo,
armazenar seus dados e resultados em registradores de 8 bits,
usar complemento de 2 nas subtrações,
e mostrar os valores resultantes em binário:

Exemplo:

$$3,2_{(4)} - 1,001_{(2)} = X_{(8)}$$

$$11,10_{(2)} - 1,001_{(2)} = 0\ 11,100 + C_{2,4}(0\ 01,001) = \underline{0}\ 11,100 + \underline{1}\ 01,111 = \underline{0}\ 10,011_{(2)} = +\ 2.375_{(8)}$$

05a.) manualmente

a.) $110011_{(2)} - 1101_{(2)}$

b.) $101,1101_{(2)} - 3,3_{(8)}$

c.) $231_{(4)} - E_{(16)}$

d.) $D4_{(16)} - 1011101_{(2)}$

e.) $67_{(10)} - 3B_{(16)}$

05b.) mediante uso de um programa em Verilog

```
/*
  Guia_0305.v
  999999 - Xxx Yyy Zzz
*/
module Guia_0305;
// define data
  reg [2:0] a = 0 ; // binary
  reg [3:0] b = 0 ; // binary
  reg [4:0] c = 0 ; // binary
  reg [4:0] d = 0 ; // binary
  reg [6:0] e = 0 ; // binary
// actions
  initial
  begin : main
    $display ( "Guia_0305 - Tests" );
    a = 5 + 3;           // OVERFLOW (transbordamento)
    b = 10 - 5 + 25 + 3 + 1; // OVERFLOW (transbordamento)
    c = 2 - 35;          // OVERFLOW (transbordamento)
    $display("\nOverflow");
    $display("a = %d = %3b = %d", (5+3) , a, a);
    $display("b = %d = %4b = %d", (10 - 5 + 25 + 3 + 1), b, b);
    $display("c = %d = %5b = %d", (2-35), c, c);
    $display("\nComplements");
    $display("0= %d = %3b = %3b", ~1 , (1-1), ~(1*1) );
    $display("1= %d = %3b = %3b", ~0 , (2-1), ~(0*1) );
    $display("2= %d = %3b = %3b", (1+1), (3-1), ~0+~0 );
  end // main

endmodule // Guia_0305
```

Extras

- 06.) Definir e testar um módulo para calcular o complemento de 1, de um valor qualquer contido em um byte.
É recomendável simular o mesmo em Logisim (Gates - NOT)
- 07.) Definir e testar um módulo para calcular o complemento de 2, de um valor qualquer contido em um byte.
É recomendável simular o mesmo em Logisim (Arithmetic - Negator).

Modelo em Java

```
/**
    Arquitetura de Computadores I - Guia_03.java
    999999 - Xxx Yyy Zzz
 */
public class Guia_03
{
    /**
        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 binario para o complemento de 1.
        @return complemento de 1 equivalente
        @param length - tamanho
        @param value - valor binario
    */
    public static String C1 ( int length, String value )
    {
        return ( "0" );
    } // end C1 ( )
}
```

```

/*
    Converter valor binario para o complemento de 2.
    @return complemento de 2 equivalente
    @param length - tamanho
    @param value - valor binario
*/
public static String C2 ( int length, String value )
{
    return ( "0" );
} // end C2 ( )

/*
    Converter valor em certa base para binario em complemento de 1.
    @return complemento de 1 equivalente
    @param length - tamanho
    @param value - valor em outra base
    @param base - base desse valor
*/
public static String C1 ( int length, String value, int base )
{
    return ( "0" );
} // end C1 ( )

/*
    Converter valor em certa base para binario em complemento de 2.
    @return complemento de 2 equivalente
    @param length - tamanho
    @param value - valor em outra base
    @param base - base desse valor
*/
public static String C2 ( int length, String value, int base )
{
    return ( "0" );
} // end C2 ( )

/*
    Converter valor binario com sinal para decimal.
    @return decimal equivalente
    @param value - valor binario
*/
public static String sbin2dec ( String value )
{
    return ( "0" );
} // end sbin2dec ( )

```

```

/*
  Operar (subtrair) valores em certa base.
  @return valor resultante da operacao
  @param value1 - primeiro valor na base dada
  @param op      - operacao ("-")
  @param value2 - segundo valor na base dada
  @param base    - base para a conversao
*/
public static String eval ( String value1, String op, String value2, int base )
{
    return ( "0" );
} // end eval ( )

```

```

/*
  Operar valores em certas bases.
  @return valor resultante da operacao, se valida
  @param value1 - primeiro valor
  @param base1  - primeira base
  @param op     - operacao
  @param value2 - segundo valor
  @param base2  - segunda base
*/
public static String evalB1B2 ( String value1, int base1, String op, String value2, int base2 )
{
    return ( "0" );
} // end dbinEval ( )

```

```

/*
Acao principal.
*/
public static void main ( String [ ] args )
{
    System.out.println ( "Guia_03 - Java Tests  " );
    System.out.println ( " 999999 - Xxx Yyy Zzz" );
    System.out.println ( );

    test_equals ( C1      ( 6,      "1100" ), "0" );
    test_equals ( C1      ( 8,      "1110" ), "0" );
    test_equals ( C2      ( 6,      "100101" ), "0" );
    test_equals ( C2      ( 7,      "100101" ), "0" );
    test_equals ( C2      ( 8,      "110101" ), "0" );
    System.out.println    ( "1. errorTotalReportMsg = "+test_report ( ) );

    test_equals ( C1      ( 6, "132",  4 ), "0" );
    test_equals ( C1      ( 8,  "4B", 16 ), "0" );
    test_equals ( C2      ( 6, "213",  4 ), "0" );
    test_equals ( C2      ( 7, "154",  8 ), "0" );
    test_equals ( C2      ( 8,  "B8", 16 ), "0" );
    System.out.println    ( "2. errorTotalReportMsg = "+test_report ( ) );

    test_equals ( sbin2dec (      "10111" ), 0 );
    test_equals ( sbin2dec (      "110001" ), 0 );
    test_equals ( sbin2dec (      "100101" ), 0 );
    test_equals ( sbin2dec (      "1011101" ), 0 );
    test_equals ( sbin2dec (      "1010011" ), 0 );
    System.out.println    ( "3. errorTotalReportMsg = "+test_report ( ) );

    test_equals ( eval      (      "11101", "-", "1011", 2 ), "0" );
    test_equals ( eval      ( "101.0101", "-", "10.11", 2 ), "0" );
    test_equals ( eval      (      "321", "-",  "213", 4 ), "0" );
    test_equals ( eval      (      "461", "-",  "247", 8 ), "0" );
    test_equals ( eval      (      "7C4", "-",  "B1D", 16 ), "0" );
    System.out.println    ( "4. errorTotalReportMsg = "+test_report ( ) );

    test_equals ( evalB1B2 (      "110011", 2, "-",      "1101", 2 ), "0" );
    test_equals ( evalB1B2 ( "101,1101", 2, "-",      "3,3", 8 ), "0" );
    test_equals ( evalB1B2 (      "231", 4, "-",      "E", 16 ), "0" );
    test_equals ( evalB1B2 (      "D4", 16, "-", "1011101", 2 ), "0" );
    test_equals ( evalB1B2 (      "67", 16, "-",      "3B", 16 ), "0" );
    System.out.println    ( "5. errorTotalReportMsg = "+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_03.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 ( )

'''
    Converter valor binario para decimal com parte fracionaria.
    @return decimal equivalente
    @param value - valor binario
'''

def bin2double ( value ):
    return ( -1.0 );
# end bin2double ( )

```

```

'''
    Converter valor decimal para binario com parte fracionaria.
    @return valor binario equivalente
    @param value - decimal
'''

def double2bin ( value ):
    return ( "0" );
# end double2bin ( )

'''
    Converter valor binario com parte fracionaria para base indicada.
    @return base para a conversao
    @param value - valor binario
'''

def dbin2base ( value, base ):
    return ( "0" );
# end dbin2base ( )

'''
    Converter valor com parte fracionaria de uma base para outra base indicada.
    @return valor equivalente na segunda base
    @param value - valor na base1
    @param base1 - primeira base
    @param base2 - base para a conversao
'''

def dbase2base ( value, base1, base2 ):
    return ( "0" );
# end dbase2base ( )

'''
    Operar valores em binartest_
    @return valor resultante da operacao, se valida
    @param value1 - primeiro valor binario
    @param op      - operacao
    @param value2 - segundo valor binario
'''

def dbinEval ( value1, op, value2 ):
    return ( "0" );
# end dbinEval ( )

'''
    Converter valor binario para o complemento de 1.
    @return complemento de 1 equivalente
    @param length - tamanho
    @param value   - valor binario
'''

def C1a ( length, value ):
    return ( "0" );
# end C1a ( )

```

```

'''
    Converter valor binario para o complemento de 2.
    @return complemento de 2 equivalente
    @param length - tamanho
    @param value - valor binario
'''

def C2a ( length, value ):
    return ( "0" );
# end C2a ( )

'''
    Converter valor em certa base para binario em complemento de 1.
    @return complemento de 1 equivalente
    @param length - tamanho
    @param value - valor em outra base
    @param base - base desse valor
'''

def C1b ( length, value, base ):
    return ( "0" );
# end C1b ( )

'''
    Converter valor em certa base para binario em complemento de 2.
    @return complemento de 2 equivalente
    @param length - tamanho
    @param value - valor em outra base
    @param base - base desse valor
'''

def C2b ( length, value, base ):
    return ( "0" );
# end C2b ( )

'''
    Converter valor binario com sinal para decimal.
    @return decimal equivalente
    @param value - valor binario
'''

def sbin2dec ( value ):
    return ( "0" );
# end sbin2dec ( )

```



```
'''
Operar (subtrair) valores em certa base.
@return valor resultante da operacao
@param value1 - primeiro valor na base dada
@param op      - operacao ("-")
@param value2 - segundo valor na base dada
@param base    - base para a conversao
'''
```

```
def eval ( value1, op, value2, base ):
    return ( "0" );
# end eval ( )
```

```
'''
Operar valores em certas bases.
@return valor resultante da operacao, se valida
@param value1 - primeiro valor
@param base1   - primeira base
@param op      - operacao
@param value2  - segundo valor
@param base2   - segunda base
'''
```

```
def evalB1B2 ( value1, base1, op, value2, base2 ):
    return ( "0" );
# end dbinEval ( )
```

```

'''
    Acao principal.
'''

def main ( ):
    print ( "Guia_03 - Python Tests" );
    print ( " 999999 - Xxx Yyy Zzz " );
    print ( );

    test_equals ( C1      ( 6,      "1100" ), "0" );
    test_equals ( C1      ( 8,      "1110" ), "0" );
    test_equals ( C2      ( 6,      "100101" ), "0" );
    test_equals ( C2      ( 7,      "100101" ), "0" );
    test_equals ( C2      ( 8,      "110101" ), "0" );
    print      ( "1. errorTotalReportMsg = "+test_report ( ) );

    test_equals ( C1      ( 6, "132",  4 ), "0" );
    test_equals ( C1      ( 8,  "4B", 16 ), "0" );
    test_equals ( C2      ( 6, "213",  4 ), "0" );
    test_equals ( C2      ( 7, "154",  8 ), "0" );
    test_equals ( C2      ( 8,  "B8", 16 ), "0" );
    print      ( "2. errorTotalReportMsg = "+test_report ( ) );

    test_equals ( sbin2dec (      "10111" ), 0 );
    test_equals ( sbin2dec (      "110001" ), 0 );
    test_equals ( sbin2dec (      "100101" ), 0 );
    test_equals ( sbin2dec (      "1011101" ), 0 );
    test_equals ( sbin2dec (      "1010011" ), 0 );
    print      ( "3. errorTotalReportMsg = "+test_report ( ) );

    test_equals ( eval      (      "11101", "-", "1011", 2 ), "0" );
    test_equals ( eval      ( "101.0101", "-", "10.11", 2 ), "0" );
    test_equals ( eval      (      "321", "-",  "213", 4 ), "0" );
    test_equals ( eval      (      "461", "-",  "247", 8 ), "0" );
    test_equals ( eval      (      "7C4", "-",  "B1D", 16 ), "0" );
    print      ( "4. errorTotalReportMsg = "+test_report ( ) );

    test_equals ( evalB1B2 ( "110011", 2, "-",      "1101", 2 ), "0" );
    test_equals ( evalB1B2 ( "101,1101", 2, "-",      "3,3", 8 ), "0" );
    test_equals ( evalB1B2 (      "231", 4, "-",      "E", 16 ), "0" );
    test_equals ( evalB1B2 (      "D4", 16, "-", "1011101", 2 ), "0" );
    test_equals ( evalB1B2 (      "67", 16, "-",      "3B", 16 ), "0" );
    print      ( "5. errorTotalReportMsg = "+test_report ( ) );

    print ( "\n\nApertar ENTER para terminar." );
    input ( );
# end main ( )

if __name__ == "__main__":
    main( );

```