










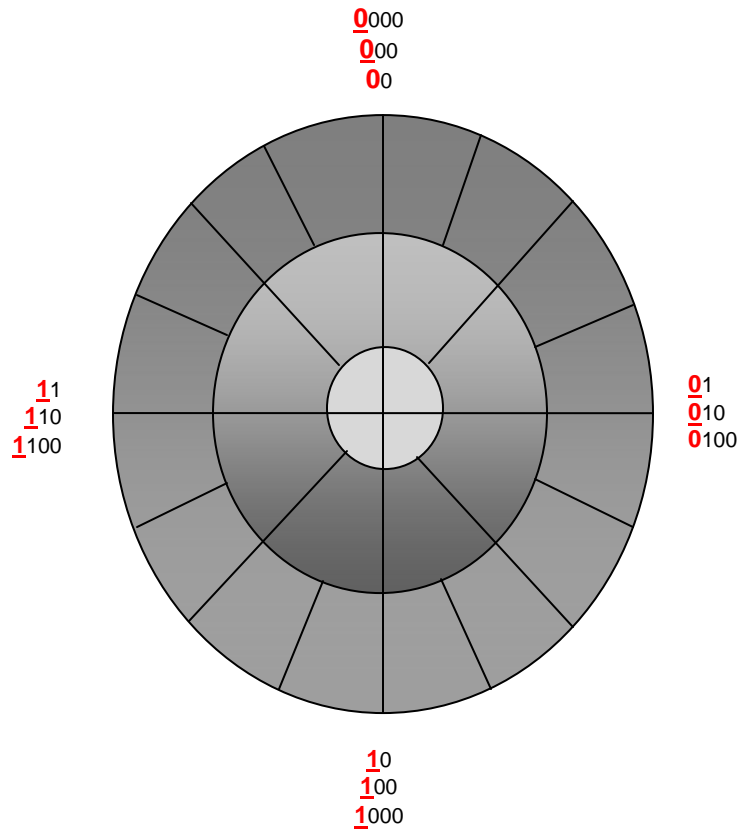
Sistemas de Numeração – Representações de dados com sinal

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

Tipos	Intervalo	Tamanho
boolean false 	[false:true]	1 byte
byte 0, 0x00 	[-128 : 127]	1 byte
char '0', '\u0000' 	[0 : 65535] Unicode	2 bytes
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 .mantissa)	4 bytes
double 0.0, 0.0e0 ± e=11 1.m IEEE754 	$[-1.0e^{-308} : 1.0e^{308}]$ (sinal+amplitude+1 .mantissa)	8 bytes
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:

$$\begin{array}{rclclcl}
 & & 1 & & (10) & & 1 & \leftarrow \text{"vem-um"} \\
 101101_{(2)} & - & 101\textcolor{red}{0}01_{(2)} & - & 101\textcolor{red}{0}(0)1_{(2)} & - & 100\textcolor{red}{(10)}01_{(2)} & \leftarrow \text{operando 1} \\
 - & 111_{(2)} & - & 111_{(2)} & - & 1\ 1\ 1_{(2)} & - & 1\ 11_{(2)} & \leftarrow \text{operando 2} \\
 \hline
 0_{(2)} & 0_{(2)} & 1\ 0_{(2)} & 100\ 1\ 10_{(2)} & 100110_{(2)} & \leftarrow \text{resultado}
 \end{array}$$

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.

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.

$$\begin{array}{rcl}
 101101_{(2)} & \rightarrow & \textcolor{red}{0}\ 101001_{(2)} \\
 - & 111_{(2)} & \rightarrow - \textcolor{red}{0}\ 000111_{(2)} \\
 \hline
 \end{array}$$

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

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

$$\begin{array}{rcl}
 101101_{(2)} & \rightarrow & \textcolor{red}{0}\ 101001_{(2)} \\
 - & 111_{(2)} & \rightarrow - \textcolor{red}{1}\ 111001_{(2)} \\
 \hline
 \end{array}$$

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

$$\begin{array}{rclcl}
 & & (1) & 1 & 1 & 1 & 1 & \leftarrow \text{"vai-um"} \\
 101101_{(2)} & \rightarrow & \textcolor{red}{0}\ 101001_{(2)} & & & & & \\
 - & 111_{(2)} & + & \textcolor{red}{1}\ 111001_{(2)} & & & & \\
 \hline
 & & \textcolor{red}{0}\ 100110 & & & & &
 \end{array}$$

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

Vídeos recomendados

Como preparação para o início das atividades, recomenda-se assistir os seguintes vídeos:

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

<http://www.youtube.com/watch?v=XGAl7irIEtc>

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

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

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

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

Exercícios:

Orientação geral:

Basta apresentar uma das formas de soluções propostas como resposta.

Mais de um tipo de resposta será avaliado como extra.

As funções poderão ser desenvolvidas em C ou Java (ver modelo Guia_03.java).

Exemplos em Verilog serão fornecidos como ponto de partida.

01.) Determinar os complementos para os valores e as quantidades de bits indicadas:

a.) $C_{1,6} (1011_{(2)}) = X_{(2)}$

b.) $C_{1,8} (1100_{(2)}) = X_{(2)}$

c.) $C_{2,6} (101001_{(2)}) = X_{(2)}$

d.) $C_{2,7} (10101_{(2)}) = X_{(2)}$

e.) $C_{2,8} (10001_{(2)}) = X_{(2)}$

DICA: Ajustar primeiro o tamanho, antes de calcular o complemento ($C_{1,n}$ ou $C_{2,n}$).

01a.) mediante uso da função $C1(nbits, x)$ ou da função $C2(nbits, x)$

01b.) mediante uso de uma planilha

01c.) mediante uso de um programa em Verilog

```
/*
  Guia_0301
*/
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
```

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

- a.) $C_{1,6} (321_{(4)}) = X_{(2)}$
- b.) $C_{1,8} (4E_{(16)}) = X_{(2)}$
- c.) $C_{2,6} (13_{(4)}) = X_{(2)}$
- d.) $C_{2,7} (161_{(8)}) = X_{(2)}$
- e.) $C_{2,8} (6D_{(16)}) = X_{(2)}$

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

02a.) mediante uso de funções $C1(nbits, x, basex)$ e $C2(nbits, x, basex)$.

02b.) mediante uso de uma planilha

02c.) mediante uso de um programa em Verilog

```
/*
  Guia_0302
*/
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
```

03.) Determinar os valores positivos equivalentes aos complementos indicados:

a.) $\underline{1}001_{(2)} = X_{(10)}$

b.) $\underline{1}01101_{(2)} = X_{(10)}$

c.) $\underline{1}1001_{(2)} = X_{(2)}$

d.) $\underline{1}010101_{(2)} = X_{(2)}$

e.) $\underline{1}0101100_{(2)} = X_{(16)}$

DICA: Subtrair uma unidade, antes de inverter cada bit.

03a.) mediante uso de uma função sbin2dec(x)

03b.) mediante uso de uma planilha

03c.) mediante uso de um programa em Verilog

```
/*
  Guia_0303
*/
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
```

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

- a.) $11111_{(2)} - 1101_{(2)} = X_{(2)}$
- b.) $100,1001_{(2)} - 10,11_{(2)} = X_{(2)}$ (OBS.: Alinhar as vírgulas, primeiro, antes de operar)
- c.) $321_{(4)} - 123_{(4)} = X_{(4)}$
- d.) $214_{(8)} - 132_{(8)} = X_{(8)}$
- e.) $3B5_{(16)} - A3E_{(16)} = X_{(16)}$

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

Substituir apenas os subtraendos pelos complementos equivalentes e somar.

Voltar os resultados às bases originais.

04a.) mediante uso de uma função baseEval(x, "-", y, base)

04b.) mediante uso de uma planilha

04c.) mediante uso de um programa em Verilog

```
/*
  Guia_0304
*/
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
```

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

- a.) $101001_{(2)} - 1101_{(2)}$
- b.) $11101_{(2)} - 17_{(8)}$
- c.) $34_{(8)} - C_{(16)}$
- d.) $AC_{(16)} - 1011101_{(2)}$
- e.) $36 - 2B_{(16)}$

05c.) mediante uso de um programa em Verilog

```
/*
  Guia_0305
*/
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
    b = 10 - 5 + 25 + 3 + 1; // OVERFLOW
    c = 2 - 35;         // OVERFLOW

    $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.
- 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.

Modelo em Java

```
import IO.*;

/**
 * Arquitetura de Computadores I - Guia_03.
 */
public class Guia_03
{
    /**
     * 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 )
{
    IO.println ( "Guia_03 - Java Tests" );
    IO.println ( "Nome: _____ Matricula: _____ " );
    IO.println ( );

    IO.equals ( C1 ( 6, "01001" ), "0" );
    IO.equals ( C1 ( 8, "01110" ), "0" );
    IO.equals ( C2 ( 6, "101101" ), "0" );
    IO.equals ( C2 ( 7, "10111" ), "0" );
    IO.equals ( C2 ( 8, "10011" ), "0" );
    IO.println ( "1. errorTotalReportMsg = "+IO.errorTotalReportMsg( ) );

    IO.equals ( C1 ( 6, "231", 4 ), "0" );
    IO.equals ( C1 ( 8, "4D", 16 ), "0" );
    IO.equals ( C2 ( 6, "12", 4 ), "0" );
    IO.equals ( C2 ( 7, "171", 8 ), "0" );
    IO.equals ( C2 ( 8, "6E", 16 ), "0" );
    IO.println ( "2. errorTotalReportMsg = "+IO.errorTotalReportMsg( ) );

    IO.equals ( sbin2dec ( "001011" ), 0 );
    IO.equals ( sbin2dec ( "101001" ), 0 );
    IO.equals ( sbin2dec ( "11011" ), 0 );
    IO.equals ( sbin2dec ( "1010001" ), 0 );
    IO.equals ( sbin2dec ( "10100100" ), 0 );
    IO.println ( "3. errorTotalReportMsg = "+IO.errorTotalReportMsg( ) );

    IO.equals ( eval ( "11101", "-", "1101", 2 ), "0" );
    IO.equals ( eval ( "100.1101", "-", "10.11", 2 ), "0" );
    IO.equals ( eval ( "321", "-", "132", 4 ), "0" );
    IO.equals ( eval ( "214", "-", "123", 8 ), "0" );
    IO.equals ( eval ( "3B5", "-", "A3E", 16 ), "0" );
    IO.println ( "4. errorTotalReportMsg = "+IO.errorTotalReportMsg( ) );

    IO.equals ( evalB1B2 ( "101101", 2, "-", "1101", 2 ), "0" );
    IO.equals ( evalB1B2 ( "11001", 2, "-", "17", 8 ), "0" );
    IO.equals ( evalB1B2 ( "34", 8, "-", "D", 16 ), "0" );
    IO.equals ( evalB1B2 ( "AC", 16, "-", "1011100", 2 ), "0" );
    IO.equals ( evalB1B2 ( "36", 2, "-", "2D", 16 ), "0" );
    IO.println ( "5. errorTotalReportMsg = "+IO.errorTotalReportMsg( ) );

    IO.println ( );
    IO.pause ( "Apertar ENTER para terminar." );
} // end main

} // end class

```