

## Arithmetic operators

<b>+</b>	<b>addition</b>
<b>-</b>	<b>subtraction</b>
<b>*</b>	<b>multiplication</b>
<b>**</b>	<b>exponentiation</b>
<b>/</b>	<b>division</b>
<b>mod</b>	<b>modulo</b>
<b>abs</b>	<b>absolute value</b>
<b>rem</b>	<b>remainder</b>

**Operands of the same type**

**Predefined for**

- **integer**
- **real (except mod and rem)**
- **physical types (e.g. time)**

**Not defined for bit\_vector**

**'+' and '-' may also be used as unary operators**

**Operands for arithmetic operations are presented in 2's complement form and the result is presented in the same form as well.**

2's COMPLEMENT ADDITION RULES		
Augend	Addend	Rule
+	+	Add magnitudes, ignore end-around carry. The sign of the result is the sign found by the result of addition operation.
-	-	
+	-	
-	+	

1.

+	+13	=	+	001101	← (+13) in ordinary form
	+11			001011	← (+11) in ordinary form
<hr/>				011000	← (+24) in ordinary form
	+24				

'0' means (+) sign

2.

+	-13	=	+	110011	← (-13) in 2's complement
	-11			110101	← (-11) in 2's complement
<hr/>				101000	← (-24) in 2's complement
	-24				

'1' means (-) sign

3.

+	-11	=	+	110101	← (-11) in 2's complement
	+13			001101	← (+13) in ordinary form
<hr/>				000010	← (+2) in ordinary form
	+2				

'0' means (+) sign

4.

+	+11	=	+	001011	← (+11) in ordinary form
	-13			110011	← (-13) in 2's complement
<hr/>				111110	← (-2) in 2's complement
	-2				

'1' means (-) sign

2's COMPLEMENT SUBTRACTION RULES		
Minuend	Subtrahend	Rule
+	+	Complement all bits of the subtrahend to 2's complement, add to minuend, ignore end-around carry. The sign of the result is the sign found by the result of addition operation.
-	-	
+	-	
-	+	

1. 
$$\begin{array}{r} - \quad +13 \\ +11 \\ \hline +2 \end{array} = \begin{array}{r} + \quad +13 \\ -11 \\ \hline +2 \end{array} = \begin{array}{r} + \quad 001101 \\ 110101 \\ \hline 000010 \end{array}$$
  
 ← (+13) in ordinary form  
 ← (-11) in 2's complement  
 ← (+2) in ordinary form  
 '0' means (+) sign

2. 
$$\begin{array}{r} - \quad -13 \\ -11 \\ \hline +2 \end{array} = \begin{array}{r} + \quad -13 \\ +11 \\ \hline -2 \end{array} = \begin{array}{r} + \quad 110011 \\ 001011 \\ \hline 111110 \end{array}$$
  
 ← (-13) in 2's complement  
 ← (+11) in ordinary form  
 ← (-2) in 2's complement  
 '1' means (-) sign

3. 
$$\begin{array}{r} - \quad -11 \\ +13 \\ \hline -24 \end{array} = \begin{array}{r} + \quad -11 \\ -13 \\ \hline -24 \end{array} = \begin{array}{r} + \quad 110101 \\ 110011 \\ \hline 101000 \end{array}$$
  
 ← (-11) in 2's complement  
 ← (-13) in 2's complement  
 ← (-24) in 2's complement  
 '1' means (-) sign

4. 
$$\begin{array}{r} - \quad +11 \\ -13 \\ \hline +24 \end{array} = \begin{array}{r} + \quad +11 \\ +13 \\ \hline +24 \end{array} = \begin{array}{r} + \quad 001011 \\ 001101 \\ \hline 011000 \end{array}$$
  
 ← (+11) in ordinary form  
 ← (+13) in ordinary form  
 ← (+24) in ordinary form  
 '0' means (+) sign

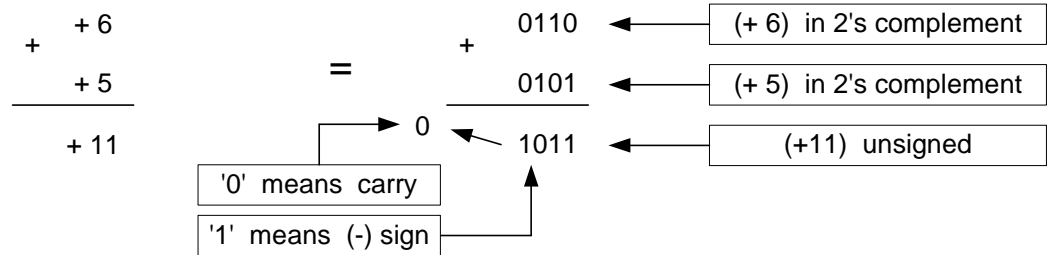
## Overflow

### Overflow Rules

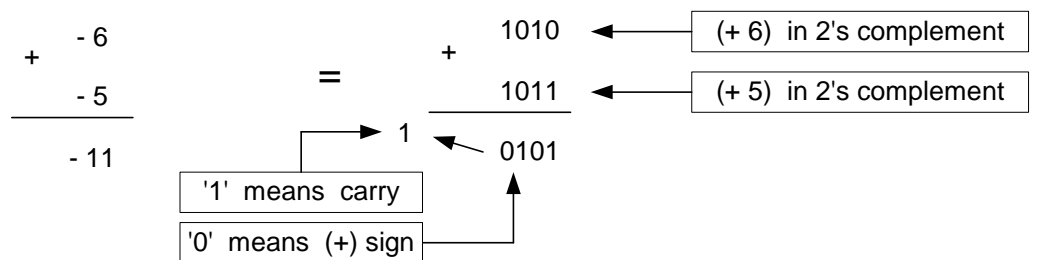
- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs:
  1. when adding two positives yields a negative;
  2. adding two negatives gives a positive
  3. subtract a negative from a positive and get a negative
  4. subtract a positive from a negative and get a positive

## Addition

### 1. Both operands are positive

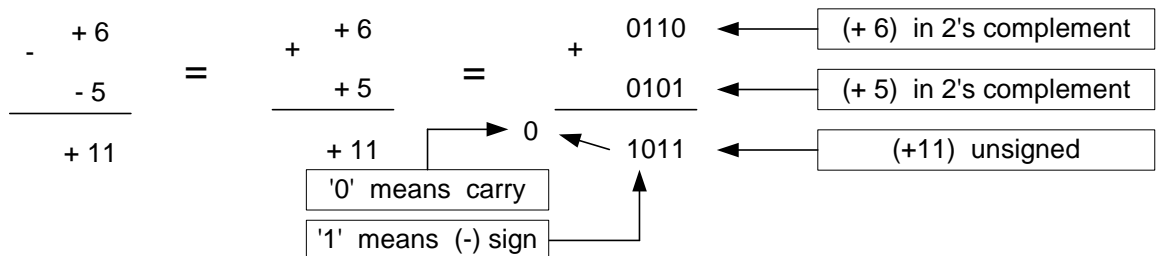


### 2. Both operands are negative



## Subtraction

### 1. Minuend is positive but subtrahend is negative. Result negative



### 2. Minuend is negative but subtrahend is positive. Result positive

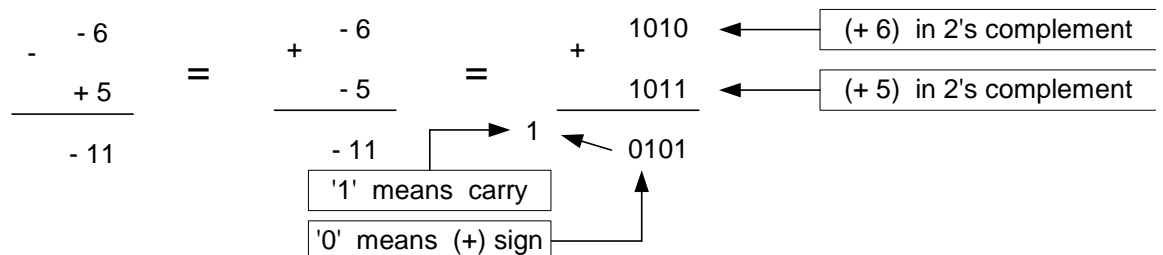
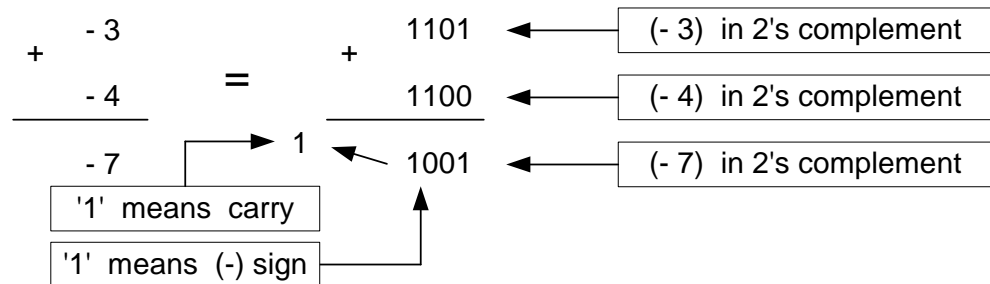


Figure 3. Examples of overflow for addition and subtraction

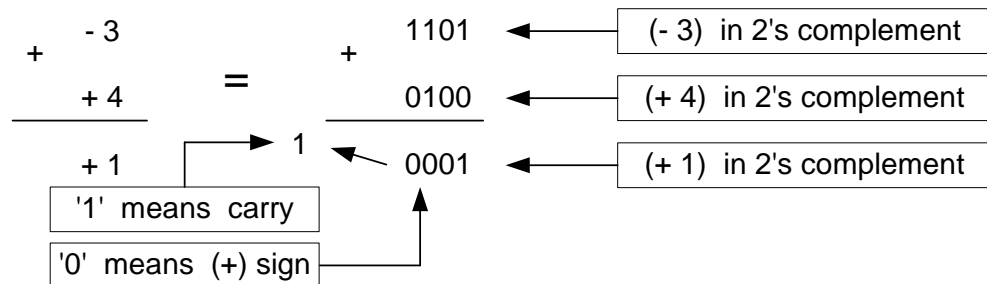
## Carry

### Addition

#### 1. Both operands are negative



#### 2. First operand is negative but result is positive



#### 3. Second operand is negative but result is positive

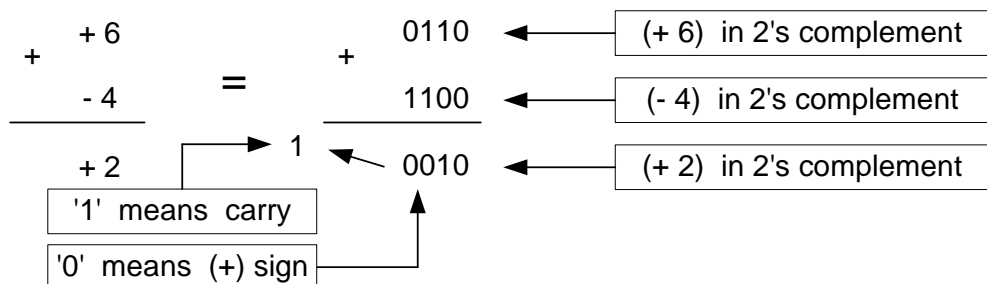


Figure 4. Illustration carry for addition

## Subtraction

### 1. Minuend is negative but subtrahend is positive

$$\begin{array}{r} - \quad -3 \\ + \quad +4 \\ \hline -7 \end{array} = \begin{array}{r} + \quad -3 \\ - \quad -4 \\ \hline -7 \end{array} = \begin{array}{r} + \quad 1101 \\ - \quad 1100 \\ \hline 1001 \end{array}$$

Annotations for the binary result 1001:

- 1101 ← (- 3) in 2's complement
- 1100 ← (- 4) in 2's complement
- 1001 ← (- 7) in 2's complement
- '1' means carry (from the 4th bit to the 5th bit)
- '1' means (-) sign (the 5th bit)

### 2. Minuend is negative but result is positive

$$\begin{array}{r} - \quad -3 \\ - \quad -4 \\ \hline +1 \end{array} = \begin{array}{r} + \quad -3 \\ + \quad +4 \\ \hline +1 \end{array} = \begin{array}{r} + \quad 1101 \\ + \quad 0100 \\ \hline 0001 \end{array}$$

Annotations for the binary result 0001:

- 1101 ← (- 3) in 2's complement
- 0100 ← (+ 4) in 2's complement
- 0001 ← (+ 1) in 2's complement
- '1' means carry (from the 4th bit to the 5th bit)
- '0' means (-) sign (the 5th bit)

### 3. Subtrahend is positive and result is positive

$$\begin{array}{r} - \quad +6 \\ + \quad +4 \\ \hline +2 \end{array} = \begin{array}{r} + \quad +6 \\ - \quad -4 \\ \hline +2 \end{array} = \begin{array}{r} + \quad 0110 \\ - \quad 1100 \\ \hline 0010 \end{array}$$

Annotations for the binary result 0010:

- 0110 ← (+ 6) in 2's complement
- 1100 ← (- 4) in 2's complement
- 0010 ← (+ 2) in 2's complement
- '1' means carry (from the 4th bit to the 5th bit)
- '0' means (+) sign (the 5th bit)

Figure 5. Illustration carry for subtraction

## Operations with std\_logic\_vectors

In a *numeric package*, three types of arrays of std\_logic can be used:

*std\_logic\_vector*;  
*signed*;  
*unsigned*.

```
signal x, y, z : std_logic_vector (3 downto 0);  
signal w : Boolean;
```

x <= "1011" – It is not assignment of  $(11)_{10}$  or  $(-3)_{10}$ .  
It is assignment of a vector with four components,  
each of them has a type std\_logic.

z <= x and y;      correct

w <= x > y;      correct

z <= x + y;      is not defined.

### *std\_logic\_vector*

We can make logical operations and comparison operations with std\_logic array types, but arithmetic operations are not defined with such arrays.

```
signal x, y, z : signed (3 downto 0);
```

x <= 1011;  $(-5)_{10}$  in 2's complement form

y <= 0011;  $(+3)_{10}$  in 2's complement form

z <= x + y;  $1110 = (-2)_{10}$  in 2's complement form

if x > y then ...      correct (false)

### *signed*

2's complement 4-bit vector presents integers from  $(-8)_{10}$  to  $(7)_{10}$

```
signal x, y, z : unsigned (3 downto 0);
```

x <= 1011;  $(11)_{10}$

y <= 0011;  $(3)_{10}$

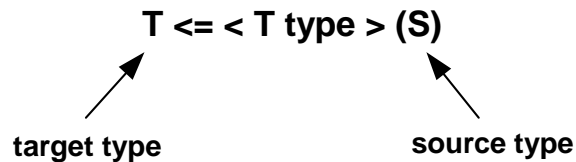
z <= x + y;  $1110 = (14)_{10}$

if x > y then ...      correct (true)

### *unsigned*

4-bit vector presents positive integers from  $(0)_{10}$  to  $(15)_{10}$

## Conversion between types *std\_logic\_vector*, *signed*, *unsigned* and *integer*



It is a representation of conversion between compatible types *std\_logic\_vector*, *signed*, *unsigned*. These types are called compatible because signals or variables of these types are vectors of *std\_logic*.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity number is
  port (
    a : in std_logic_vector(0 to 3);
    b : in signed(0 to 3);
    c : in unsigned(0 to 3);
    d1, d2 : in natural 0 to 15;
    e1, e2 : in integer -7 to 7;
    s1 : out unsigned(0 to 3);
    t1 : out signed(0 to 3);
    w1, w2 : out integer range -15 to 15;
    x1, x2 : out std_logic_vector(0 to 3);
    y1, y2 : out signed(0 to 3);
    z1, z2 : out unsigned(0 to 3)
  );
end number;

architecture arc_number of number is
begin
  x1 <= std_logic_vector(b);
  x2 <= std_logic_vector(c);
  y1 <= signed(a);
  y2 <= signed(c);
  z1 <= unsigned(a);
  z2 <= unsigned(b);
end;

configuration cfg_number of number is
  for arc_number
  end for;
end cfg_number;
```



## Conversion between types *signed*, *unsigned* and *integer*

**INT** <= to\_integer (UNS)

**INT** <= to\_integer (SIG)

**UNS** <= to\_unsigned (INT, Length)

**SIG** <= to\_signed (INT, Length)

```
w1 <= to_integer (b);  
w2 <= to_integer (c);  
  
s1 <= to_unsigned (d1, 4) + to_unsigned (d2, 4);  
t1 <= to_signed (e1, 4) + to_signed (e2, 4);
```

## Some operators in *numeric\_std* package

### Comparison operators

Operands: signed, signed; result – Boolean;  
signed, integer; result – Boolean;  
unsigned, unsigned; result – Boolean;  
unsigned, natural; result – Boolean.

### Addition and Subtraction

Operands: signed, signed; result – signed;  
signed, integer; result – signed;  
unsigned, unsigned; result – unsigned;  
unsigned, natural; result – unsigned.

### Logical operators

Only between operands of the same type:  
signed, signed; result – signed;  
unsigned, unsigned; result – unsigned.

```

-----
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
-----

entity pc_counter is
    port
        (
            rsta  : in std_logic;
            clk   : in std_logic;
            en     : in std_logic;
            count  : in std_logic;
            din    : in std_logic_vector (0 to 15);
            dout   : out std_logic_vector (0 to 15)
        );
end pc_counter;
-----

architecture arch_pc_counter of pc_counter is
begin

    process (rsta, clk)
        variable tmp : unsigned(0 to 15);
    begin
        if rsta = '1' then
            tmp := x"0000";
        elsif clk'event and clk = '1' then
            if en = '1' then
                tmp := unsigned(din);
            elsif count = '1' then
                if tmp = x"ffff" then
                    tmp := x"0000";
                else
                    tmp := tmp + 1;
                end if;
            end if;
        end if;
        dout <= std_logic_vector(tmp);
    end process;
end arch_pc_counter;
-----

configuration cfg_pc_counter of pc_counter is
    for arch_pc_counter
        end for;
end cfg_pc_counter;
-----

```

### **Exercise: Check operators in *numeric\_std***

```

--type conversion example
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity numeric is
    port (
        a1, a2 : in std_logic_vector(0 to 3);
        w1, w2 : out std_logic_vector(0 to 3);
        x1, x2 : out signed(0 to 3);
        y1, y2 : out unsigned(0 to 3);
        z1, z2 : out integer range -15 to 15
    );
end numeric;

```

```

architecture arc_numeric of numeric is
begin
    process (a1, a2)
        variable xx1, xx2 : signed(0 to 3);
        variable yy1, yy2 : unsigned(0 to 3);
    begin
        xx1 := signed (a1) + signed (a2);
        xx2 := signed (a1) - signed (a2);

        z1 <= to_integer (xx1);
        z2 <= to_integer (xx2);

        yy1 := unsigned (a1) + unsigned (a2);
        yy2 := unsigned (a1) - unsigned (a2);

        w1 <= std_logic_vector(yy1);
        w2 <= std_logic_vector(yy2);

        y1 <= yy1;
        y2 <= yy2;
        x1 <= xx1;
        x2 <= xx2;
    end process;
end arc_numeric;

configuration cfg_numeric of numeric is
    for arc_numeric
    end for;
end cfg_numeric;

```

### Test bench

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test_numeric is
end test_numeric;

architecture arc_test_numeric of test_numeric is
    component numeric
        port (
            a1, a2 : in std_logic_vector(0 to 3);
            w1, w2 : out std_logic_vector(0 to 3);
            x1, x2 : inout signed(0 to 3);
            y1, y2 : out unsigned(0 to 3);
            z1, z2 : out integer range -15 to 15
        );
    end component;

    signal a1, a2, w1, w2 : std_logic_vector(0 to 3);
    signal x1, x2 : signed(0 to 3);
    signal y1, y2 : unsigned(0 to 3);
    signal z1, z2 : integer range -15 to 15;

begin
    uut : numeric port map (a1, a2, w1, w2, x1, x2, y1, y2, z1, z2);

```

```

process
begin
    a1 <= "1011"; a2 <= "0010"; -- a1 = -5; a2 = 3
    wait for 10 ns;

    assert x1 = "1101"
        report "output x1 is wrong!"
        severity error;
    assert x2 = "1001"
        report "output x2 is wrong!"
        severity error;

    assert z1 = -3
        report "output z1 is wrong!"
        severity error;
    assert z2 = -7
        report "output z2 is wrong!"
        severity error;

    assert y1 = "1101"
        report "output y1 is wrong!"
        severity error;
    assert y2 = "1001"
        report "output y2 is wrong!"
        severity error;

    assert w1 = "1101"
        report "output w1 is wrong!"
        severity error;
    assert w2 = "1001"
        report "output w2 is wrong!"
        severity error;

    wait;
end process;
end arc_test_numeric;

configuration cnf_test_numeric of test_numeric is
    for arc_test_numeric
    end for;
end cnf_test_numeric;

```