Hexadecimal and binary conversions and functions

Hexadecimal or binary numbers are represented as strings, decimal numbers as numbers. The following functions work with unsigned integers only. The intention is to have a possibility to handle register values of microcontrollers or other integrated circuits.

```
hex2dec(Hex) :=
    HEXC ← "0123456789ABCDEF .hx"
    len ← strlen(Hex)
    Dec ← 0
    j ← 0
    idx ← 0
    while idx < len
        digit ← substr(Hex, len − idx − 1, 1)
        cpos ← search(HEXC, digit, 0)
        if cpos < 0
            error("wrong hex digit")
        if (cpos < 16)
            Dec ← Dec + cpos · 16^j
            j ← j + 1
        idx ← idx + 1
    return Dec
```

Hexadecimals have a prefix "0x" or a suffix "h". They consist of the digits 0..9,A..F and are case sensitive. It is allowed to insert blanks or dots to separate the digits. Maximum size is 49 bits ~ 5.6 10^14. Bigger values will not be represented exactly.

$hex2dec("0x1234") = 4660$ $hex2dec("0x1\ FFFF\ FFFF\ FFFF") = 562949953421311$

$hex2dec("1234h") = 4660$

$hex2dec("12\ 34") = 4660$ $hex2dec("12.34") = 4660$

$hex2dec("0AFFE\ DEADh") = 2952715949$

```
bin2dec(Bin) :=
    BINC ← "01 .b"
    len ← strlen(Bin)
    Dec ← 0
    j ← 0
    idx ← 0
    while idx < len
        digit ← substr(Bin, len − idx − 1, 1)
        cpos ← search(BINC, digit, 0)
        if cpos < 0
            error("wrong binary digit")
        if (cpos < 2)
            Dec ← Dec + cpos · 2^j
            j ← j + 1
        idx ← idx + 1
    return Dec
```

Binary numbers have a prefix "0b" or a suffix "b". They consist of the digits 0,1. It is allowed to insert blanks or dots to separate the digits. Maximum size is 49 bits ~ 5.6 10^14. Bigger values will not be represented exactly.

$bin2dec("011101110101111b") = 7639$

$bin2dec("01\ 1101\ 1101\ 0111b") = 7639$ $hex2dec("1DD7") = 7639$

$bin2dec("0b\ 0001\ 1101\ 1101\ 0111") = 7639$

Conversion of a decimal to hexadecimal or binary:

```
dec2hex(Dec) :=
    HEXC ← "0123456789ABCDEF"
    j ← 0
    Hex ← "h"
    rem ← Dec − 16 · floor(Dec / 16)
    digit ← substr(HEXC, rem, 1)
    Hex ← concat(digit, Hex)
    j ← j + 1
    div ← floor(Dec / 16)
    while div > 0
        rem ← div − 16 · floor(div / 16)
        digit ← substr(HEXC, rem, 1)
        Hex ← concat(digit, Hex)
        j ← j + 1
        div ← floor(div / 16)
        if (j = 4) ∧ (div > 0)
            Hex ← concat(" ", Hex)
            j ← 0
    if hex2dec(substr(Hex, 0, 1)) > 9
        Hex ← concat("0", Hex)
    return Hex
```

```
dec2bin(Dec) :=
    BINC ← "01"
    j ← 0
    Bin ← "b"
    rem ← Dec − 2 · floor(Dec / 2)
    digit ← substr(BINC, rem, 1)
    Bin ← concat(digit, Bin)
    j ← j + 1
    div ← floor(Dec / 2)
    while div > 0
        rem ← div − 2 · floor(div / 2)
        digit ← substr(BINC, rem, 1)
        Bin ← concat(digit, Bin)
        j ← j + 1
        div ← floor(div / 2)
        if (j = 4) ∧ (div > 0)
            Bin ← concat(" ", Bin)
            j ← 0
    return Bin
```

dec2hex(2952715949) = "0AFFE DEADh"

dec2hex(257) = "101h"

dec2bin(2952715949) = "1010 1111 1111 1110 1101 1110 1010 1101b"

dec2bin(257) = "1 0000 0001b"

Helper function to detect the base

```
base(x) := if IsString(x)
             lenx ← strlen(x)
             if (substr(x, lenx − 1, 1) = "h") ∨ (substr(x, 1, 1) = "x")
               base ← 16
             else if (substr(x, lenx − 1, 1) = "b") ∨ (substr(x, 1, 1) = "b")
               base ← 2
             else
               error("wrong base")
           else
             base ← 10
```

$base(\text{“0x1234”}) = 16$ $base(\text{“0 1 2 3 4h”}) = 16$

$base(\text{“1234h”}) = 16$ $base(1234) = 10$

$base(\text{“0b0110101”}) = 2$

bitfield extracts a sequence of bits from a starting position with a defined length. The result is a decimal

```
bitfield(x, start, len) := bas ← base(x)
                           if bas = 16
                             dec ← hex2dec(x)
                           else if bas = 2
                             dec ← bin2dec(x)
                           else if bas = 10
                             dec ← x
                           else
                             error("Unknown base")
                           b1 ← 2^start
                           tmp1 ← floor(dec / b1)
                           b2 ← 2^len
                           field ← tmp1 − b2 · floor(tmp1 / b2)
                           return field
```

$REGISTER := \text{“AFFEDEADh”}$ $base(REGISTER) = 16$

$bitfield(REGISTER, 0, 8) = 173$ $bitfield(REGISTER, 4, 8) = 234$

$hex2dec(\text{“AD”}) = 173$ $hex2dec(\text{“EA”}) = 234$

$REGISTER := \text{“1010 1111 1111 1110 1101 1110 1010 1101b”}$ $base(REGISTER) = 2$

$bitfield(REGISTER, 0, 8) = 173$

$REGISTER := \text{“1 FFFF FFFF FFFFh”}$

$hex2dec(REGISTER) = 562949953421311$

$bitfield(REGISTER, 8, 8) = 255$

Logic functions with hexadecimal, decimal or binary parameters, the result is a binary. The size of the result is determined by the biggest size of the parameters.

```
AND(x, y) :=
    bas ← base(x)
    if bas = 16
        binx ← dec2bin(hex2dec(x))
    else if bas = 2
        binx ← dec2bin(bin2dec(x))
    else if bas = 10
        binx ← dec2bin(x)
    else
        error("Unknown base")
    bas ← base(y)
    if bas = 16
        biny ← dec2bin(hex2dec(y))
    else if bas = 2
        biny ← dec2bin(bin2dec(y))
    else if bas = 10
        biny ← dec2bin(y)
    else
        error("Unknown base")
    i ← strlen(binx) − 2
    j ← strlen(biny) − 2
    res ← "b"
    while (i ≥ 0) ∧ (j ≥ 0)
        bx ← substr(binx, i, 1)
        by ← substr(biny, j, 1)
        if bx ≠ " "
            bit ← str2num(bx) ∧ str2num(by)
            res ← concat(num2str(bit), res)
        else
            res ← concat(" ", res)
        i ← i − 1
        j ← j − 1
    return res
```

```
OR(x, y) :=
    bas ← base(x)
    if bas = 16
        binx ← dec2bin(hex2dec(x))
    else if bas = 2
        binx ← dec2bin(bin2dec(x))
    else if bas = 10
        binx ← dec2bin(x)
    else
        error("Unknown base")
    bas ← base(y)
    if bas = 16
        biny ← dec2bin(hex2dec(y))
    else if bas = 2
        biny ← dec2bin(bin2dec(y))
    else if bas = 10
        biny ← dec2bin(y)
    else
        error("Unknown base")
    i ← strlen(binx) − 2
    j ← strlen(biny) − 2
    res ← "b"
    while (i ≥ 0) ∨ (j ≥ 0)
        bx ← if(i < 0, "0", substr(binx, i, 1))
        by ← if(j < 0, "0", substr(biny, j, 1))
        if (bx ≠ " ") ∧ (by ≠ " ")
            bit ← str2num(bx) ∨ str2num(by)
            res ← concat(num2str(bit), res)
        else
            res ← concat(" ", res)
        i ← i − 1
        j ← j − 1
    return res
```

```
XOR(x, y) := bas ← base(x)
             if bas = 16
                binx ← dec2bin(hex2dec(x))
             else if bas = 2
                binx ← dec2bin(bin2dec(x))
             else if bas = 10
                binx ← dec2bin(x)
             else
                error("Unknown base")
             bas ← base(y)
             if bas = 16
                biny ← dec2bin(hex2dec(y))
             else if bas = 2
                biny ← dec2bin(bin2dec(y))
             else if bas = 10
                biny ← dec2bin(y)
             else
                error("Unknown base")
             i ← strlen(binx) − 2
             j ← strlen(biny) − 2
             res ← "b"
             while (i ≥ 0) ∨ (j ≥ 0)
                bx ← if(i < 0, "0", substr(binx, i, 1))
                by ← if(j < 0, "0", substr(biny, j, 1))
                if (bx ≠ " ") ∧ (by ≠ " ")
                   bit ← str2num(bx) ⊕ str2num(by)
                   res ← concat(num2str(bit), res)
                else
                   res ← concat(" ", res)
                i ← i − 1
                j ← j − 1
             return res
```

```
NOT(x) := bas ← base(x)
          if bas = 16
             binx ← dec2bin(hex2dec(x))
          else if bas = 2
             binx ← dec2bin(bin2dec(x))
          else if bas = 10
             binx ← dec2bin(x)
          else
             error("Unknown base")
          i ← strlen(binx) − 2
          res ← "b"
          while (i ≥ 0)
             bx ← substr(binx, i, 1)
             if bx ≠ " "
                bit ← str2num(bx) ⊕ 1
                res ← concat(num2str(bit), res)
             else
                res ← concat(" ", res)
             i ← i − 1
          return res
```

```
ID(x, y) := y
```

helper function for set_bitfield

$XOR(\text{“1011 0100b”}, \text{“0xFF”}) = \text{“0100 1011b”}$

$NOT(\text{“1011 0100b”}) = \text{“0100 1011b”}$

$AND(\text{“0b101”}, 15) = \text{“101b”}$

$AND(5, \text{“1111b”}) = \text{“101b”}$

$OR(5, 2) = \text{“111b”}$

$OR(\text{“0xA5”}, \text{“0b0101 1010”}) = \text{“1111 1111b”}$

Shift shifts the binary representation of the parameter left (pos > 0) or right (pos < 0), filling up with zeroes. The result is an integer:

$$
\begin{aligned}
&SHIFT(x, pos) := \\
&\quad bas \leftarrow base(x) \\
&\quad \text{if } bas = 16 \\
&\quad\quad dec \leftarrow hex2dec(x) \\
&\quad \text{else if } bas = 2 \\
&\quad\quad dec \leftarrow bin2dec(x) \\
&\quad \text{else if } bas = 10 \\
&\quad\quad dec \leftarrow x \\
&\quad \text{else} \\
&\quad\quad \text{error}(\text{“Unknown base”}) \\
&\quad dec \leftarrow \text{floor}\left(dec \cdot 2^{pos}\right)
\end{aligned}
$$

$dec2bin(SHIFT(\text{“0101 0010b”}, 2)) = \text{“1 0100 1000b”}$

$dec2bin(SHIFT(\text{“0101 0010b”}, 0)) = \text{“101 0010b”}$

$dec2bin\langle SHIFT\langle \text{“0101 0010b”}, -1\rangle\rangle = \text{“10 1001b”}$

$dec2bin\langle SHIFT\langle \text{“0101 0010b”}, -2\rangle\rangle = \text{“1 0100b”}$

set_bitfield inserts a bitfield at a start position with defined length into a value using a function like AND, OR or XOR. This allows to set, reset or invert individual bits. The function ID allows to completely replace the bitfield.

$$
\begin{aligned}
&set\_bitfield(reg, bitf, start, len, func) := \\
&\quad mask \leftarrow hex2dec(\text{“1 FFFF FFFF FFFFh”}) \\
&\quad mask \leftarrow mask - \left(2^{len} - 1\right) \cdot 2^{start} \\
&\quad res \leftarrow AND(reg, mask) \\
&\quad regfield \leftarrow bitfield(reg, start, len) \\
&\quad regfield \leftarrow func(regfield, bitf) \\
&\quad regfield \leftarrow SHIFT(regfield, start) \\
&\quad res \leftarrow OR(res, regfield)
\end{aligned}
$$

$REGISTER := \text{“1010 1101b”}$

$set\_bitfield(REGISTER, \text{“111b”}, 2, 3, XOR) = \text{“1011 0001b”}$ invert bits 2,3,4

$set\_bitfield(REGISTER, \text{“110b”}, 2, 3, ID) = \text{“1011 1001b”}$ replace bits 2-4

$set\_bitfield(REGISTER, \text{“0b”}, 5, 1, AND) = \text{“1000 1101b”}$ clear bit 5

$set\_bitfield(REGISTER, \text{“1b”}, 6, 1, OR) = \text{“1110 1101b”}$ set bit 6

clear$(REGISTER)$