Math library for PIC18 processors

version 0.05

The library was prepared as a replacement of the official Math library (__Lib_Math.mcl) in mP PRO (or mB PRO). It is an improved version of the fixed-point library, free of its quirks and code-optimized to give twice smaller overall size. (Most code savings were obtained on division routines for signed numbers). In some cases it's also noticeably faster. The library comes in two versions – one of them introduces math error exception mechanism.

Problems that were corrected:

- numbers of type short:
  - division by 0 gives -128, whatever the dividend's sign,
  - division of -128 by -1 gives -128,
  - division of positive odd and negative even numbers by -128 gives 1, not 0,
- division of integer by 0 gives maximum positive value, whatever the dividend's sign,
- division of longint by 0 gives maximum positive value, whatever the dividend's sign,
- modulo operation on signed types gives results with wrong sign if second operand is negative ('signed' division routines produce wrong sign of remainder if divisor is negative),
- modulo operation on minimal longint number produces results smaller by one in magnitude than valid ones.

Note, that problems with modulo operations, besides leading to wrong results, lead also to discrepancies between compilation-time calculations and runtime ones.

Unresolved problems:

- compiler uses 'unsigned' routines for multiplication of signed numbers (integers and longints), though there are specialized routines in the library.

Differences and introduced features:

- var mod 0 gives var in mE lib, 0 in new lib (and status FP_FOV bit set),
- exception mechanism for math errors,
- math operations status available with Get_FPstatus function,
- constants corresponding to types' ranges were added.

There are some public routines in Math library (click on their names to jump to their prototypes):

abs_i

isqrt

WordxByte

dWordxByte

Div_dWordByWord

DivRem

First four routines allow to exploit the exception mechanism (or just use the status byte of fixed-point calculations). The abs function may be used with any signed integral value, but smaller code may be obtained when using specific functions, abs_s (for arguments of type short), and abs_i (for arguments of type integer). The isqrt routine calculates square root of (positive) integral numbers. Next two routines allow to save execution time when multiplying byte variables by, respectively, word and dword ones. Function Div_dWordByWord as its name suggests, performs division of a dWord variable by a word one. DivRem function allows to retrieve remainder of any division.

Status of fixed-point operations may be cleared with Clear_FPstatus procedure and read with function Get_FPstatus (these are the same routines that the floating-point replacement library uses). Available bits of the status byte are declared as constants:

FP_IOV - integer types out-of-range flag
FP_FOV – overflow (infinity) exception flag
FP_FDZ - divide by zero exception flag
FP_NAN - not-a-number flag

as well as values of the status byte in specified situations:

FPS_IOV - integer types out-of-range
FPS_FOV – overflow (infinite result)
FPS_FDZ - divide by zero (and overflow)
FPS_NAN - not-a-number

Boolean function FPerror is used for error testing in the exception mechanism. When the latter is activated, exception is raised automatically in all cases listed above (but for overflow caused by multiplication, which will be explained later). One may also raise an exception using FPraise procedure with mask=0.

Integer types' overflow (out of type range error) is caused by dividing maximum negative value by -1, or trying to get absolute value of a maximum negative value for chosen type, as range of signed types is asymmetric (for example, -128 fits within short type range, but maximum positive value is 127). Not-a-number error will arise when one divides zero by zero. Overflow to infinity is caused by dividing any non-zero value by zero.

Naturally, multiplication may also produce results out of range of used operands, but then multiplication of byte variables is coded inline in mE compilers for PIC18s (no way to raise exception there) and rising an exception in other cases would prevent one from optimizing code. For example, one may reach ten times higher execution speed by replacing

```
dword_var := word_var1 * word_var2;
```

with

```
dword_var := word(word_var1 * word_var2);
Mul16hi(dword_var);
```

(Mul16hi fills upper word of dword-type variable with values that are produced by 16-bit multiplication routine but are not used by the compiler).

Introduced to the library, execution-time effective multiplication routines, WordxByte and dWordxByte, do raise an exception in case of type range overflow.

Exception mechanism may be used in a following way:

Code:

```
var status: byte;
    st: string[20];
    x,y: integer;

  if not FPerror then
   begin
    y:=x/y;
    if y>1000 then FPraise(0);     // raise user-defined exception
    st:='success';
   end
  else
   begin
    stat:=Get_FPstatus;
    case stat of                   // determine exception cause
     0:       st:='too big';       // user-defined exception
     FPS_FOV: st:='overflow';
     FPS_FUN: st:='underflow';
     FPS_FDZ: st:='division by zero';
    end;
   end;
  Clear_FPstatus;                  // deactivate exceptions
```

Another example:

Code:

```
procedure handle_FPerrors;
 begin
  // any necessary steps in case of math error
  ...
  calc_error:=true;
```

```
     Clear_FPstatus;      // deactivate exceptions & clear status
    End;{handle_FPerror}

   begin
    ...
    calc_error:=false;
    if FPerror then handle_FPerrors
     else
      begin
       int_var3:=int_var1/int_var2;
       // other calculations
      end
```

The exception mechanism ensures that the conditional **if** FPerror **then** ... will be executed when an error occurs, even though it precedes the calculations.

As mentioned previously, the Math lib comes in two versions. When using the version with exception mechanism, remember to call the Clear_FPstatus procedure at program beginning. Naturally, one does not have to always use exception mechanism with the version equipped with it – without activating the mechanism, calculations will be performed just like with the other version.

The library defines some constants that should be useful:

maxByte
minShort
maxShort
maxWord
minInt
maxInt
minLongInt
maxLongInt
maxdWord

There is also a version string declared, called Lib_Math_ver (constant string [4]) that may be called from user code for verification.

Manual library installation:

- find mP PRO installation directory and subdirectory **.../Uses/P18**,
- find original library file **__Lib_Math.mcl** there and rename it,
- unpack the replacement lib – there'll be two versions:

  __Lib_ Math_exc.mcl

  __Lib_ Math_no_exc.mcl

  choose one of them, change it's name to **__Lib_ Math.mcl** and move the file to the **.../Uses/P18** directory,.

Have fun,

janni

Math library for PIC18 processors

version 0.05
date 21.07.13
Revision history:
0.01 – exception mechanism added
0.02 – DivRem added, compiled with mP v 5.0
0.03, 0.04 – recompiled with mP v 5.01 & 5.80
0.05 – added Div_dWordByWord, recompiled with mP v 6.01

Added declarations:

```
const Lib_Math_ver: string[4]
      maxByte: byte = 255;
      minShort: short = -128;
      maxShort: short = 127;
      maxWord: word = 65535;
      minInt: integer = -32768;
      maxInt: integer = 32767;
      minLongInt: longint = -2147483648;
      maxLongInt: longint = 2147483647;
      maxdWord: dWord = 4294967295
```

Public library routines:

procedure Clear_FPstatus;
clears status of math operations

function Get_FPstatus: byte;
reads status of previous math operations

function FPerror: boolean;
used for fp-math error test; present only in the version with fp-math errors exceptions

procedure FPraise(mask:byte);
raises exception if error declared in mask took place before, or if mask=0 (user exception); present only in the version with fp-math errors exceptions

function abs(arg:longint): longint;
produces absolute value of any integral argument

function abs_s(arg:short): short;
produces absolute value of argument of type short

function abs_i(arg:integer): integer;
produces absolute value of argument of type integer

function isqrt(arg:dword): dword;
calculates square root of integral number

procedure WordxByte(var arg16:word; arg8:byte);
returns 16-bit result of multiplication, arg16=arg16*arg8

procedure dWordxByte(var arg32:dword; arg8:byte);
returns 32-bit result of multiplication, arg32=arg32*arg8

procedure Mul16hi(var arg:dword);
loads higher word of 16x16 multiplication (preceding the call) to upper word of arg

function Div_dWordByWord(divid:dWord; divis:word): dWord;
returns 32-bit result of unsigned division of 32-bit variable by 16-bit one, divid/divis

function DivRem: dWord;
returns remainder from 8-, 16- and 32-bit division, preceeding function call;
may be used to retrieve higher 4 bytes of 32-bit multiplication to form 64-bit result