Quick Start

This document is a quick introduction, for using the SmplMath-macros with MASM and jWasm. For a full description see the main documentation. At the end of this document you will find a complete example program with comments.

Introduction

The SmplMath-system is a collection of macros for using mathematic expression as known from HLLs like c/c++, c#, … .There are a whole bunch of macros for expression evaluation, comparison and FPU related topics. The following examples show some of them:

Evaluation

```
fSlv y = a*x^2 + b*x + _c
mov eax,@fSlvI( 100*sin(2*pi*f*t+p) )
```

Comparison

```
.if fEQ(a,b) || fGT(x,1.E-7) || fLE(1,y)
    …
```

FPU Settings

```
fpuSetPrecision ,REAL8
```

Installation

Copy the folder ***macros*** to the root directory of your working drive (mainly C:\). If you won't use this location, you must adjust the relative paths in \SmplMath***math.inc***.

Requirements

The macros are tested with masm version 6-10 and jWasm. However, using MASM version 10 and the latest release of jWasm is recommended. The following examples use the MASM32 SDK version 10.

Syntax

Here a quick list of the most important macros:

macro syntax	description	default precision
fSlv [dest] = *expression*	procedural macro	REAL8, SDWORD
fSlv4/8/10 [dest] = *expression*	procedural macro	REAL4/8/10, SDWORD
@fSlv4/8([dest=] *expression*)	returns a REAL4/8	REAL4/8, SDWORD
@fSlvI([dest=] *expression*)	returns a SDWORD	REAL4, SDWORD
fEQ/NE/LT/GT/LE/GE(a , b)	returns a Boolean byte (0,1)	REAL8, SDWORD
faEQ(a , b , *f*)	approximately equal. *f* is a tolerance factor	REAL8, SDWORD
[] = optional		

The term 'precision' refers to, how to created constants and code. This means that the macros don't affect FPU's precision control.

Expressions can consist of sums (***+ -***), products (****/***) exponents (***x^y***) and function calls. Also Brackets can use – there is generally no nesting limit, but extreme usage may cause the synthesis to fail (e.g. no more free registers). The priority of operators from high to low is:

signs, brackets, exponents, products, sums

Operands can be variables of any FPU supported data type (except BCD). If arithmetic for Addressing is need, it must place inside square brackets:

```
fSlv x = SDWORD ptr [esi+4*ecx] + y
fSlv x = myreal4[edi*8] + 2.3
```

Immediately values (constants) can be written either as integer (decimal or hex decimal) or as floating point values (syntax=masm specific).

For integer constants the numeric range is checked, if the current precision is set to SWORD or SDWORD (SQWORD-checks can be enabled by flags). The GPRs (eax,ax,al,ah,…) can also be used. For accessing FPU registers see *Chapter 3.7.3.2 "FPU registers"* in the documentation.

Functions can called by their name followed by a bracket enclosed, comma separated list of arguments:

```
sin(x)   or    logbx(b,x)
```

For all current available function see top of the file *Macros**SmplMath**math_functions.inc*.

The fSlv-macros won't change any FPU setting, thus it is programmers job to setup the FPU. The precision-setting by the SmplMath-macros affects only code and constants creation. However, there are some helper-macros like *fpuSetPrecision*, which change the FPU settings.

Generally each of the code producing macros has its own default precision setting for integer and floating point constants. This setting can either be changed by adjusting the global settings or by local usage of Attributes. These are passed in a comma separated list, enclosed by braces:

{attr1,attr2,[,...]}

Each Expression can have one randomly placed Attribute-list - it is recommended to place it at the end or begin of an expression. The following example shows how to change the local precision to SWORD and REAL4:

```
fSlv x = 1*2+123.0^0.5     {i2,r4}
```

Currently reserved Attributes for local precision control are:

Attribute	data type
i2	SWORD
i4	SDWORD
i8	SQWORD
r4	REAL4
r8	REAL8
r10	REAL10

Runtime behavior

The macros assume that the FPU stack is unused, so that all 8 FPU register can be used. Also they doesn't change the FPU settings – this must done explicit (e.g. by using fpuSetPrecision).

The GPRs are generally not violated, except if they are specified as destination operands. The Flags may be violated.

If no local variables are supplied through fslvTLS(), global variables are created in the _BSS Segment (.data?) if needed. This cause the macros to throw warnings that the produced code is not thread save. However, this only affects the function-like macros. The syntax for fSvlTLS() is:

```
fSvlTLS(name,cb)    ; name is optional. cb specific the number of bytes to allocate (default=16).
```

Also the fSlv-macros assume to be called from the ***.code***-section.

Example program (MASM32 SDK, console application)

```
include masm32rt.inc
.686p   ;\
.mmx    ; } needed!
.xmm    ;/

include \macros\smplmath\math.inc       ; include the macros
.code
main proc
LOCAL x:REAL8,y:REAL4,a:REAL4,b:REAL4,_c:REAL4
LOCAL sz[128]:CHAR
LOCAL fSlvTLS() ; <= not really needed in this example, but it is generally a good attitude to share some
                ; locals with the fSlv-macros (thread save). This call allocates 16 Bytes on the stack.

    ;/*-------------------------------------*/
    ;/* init FPU and set precision to REAL8 */
    ;/*-------------------------------------*/
    finit
    fpuSetPrecision ,REAL8

    ;/*-----------------------------------------------------*/
    ;/* force fSlv to create REAL4 and SWORD constants    */
    ;/* (the default is REAL8 and SDWORD)                 */
    ;/*-----------------------------------------------------*/
    fSlvSetPrecision <fSlv>,REAL4,SWORD

    ;/* load locals */
    ldl x=3.5, a=1, b=5, _c=1

    ;/*--------------------------------------*/
    ;/* let's do some basic calculations ;-) */
    ;/*--------------------------------------*/
    fSlv y = a*x^2+b*x +_c
    print "a*x^2+b*x +_c = "
    print real4$(y),13,10

    ;/* use SIB addressing */
    lea edx,y
    xor ecx,ecx
    fSlv x = REAL4 ptr [edx+4*ecx] * 123

    ;/*--------------------------------------*/
    ;/* @fSlvI() returns a SDWORD           */
    ;/* In this case it is a local variable */
    ;/*--------------------------------------*/
    ldl y=12.34
    print str$( @fSlvI(y^2+2+pi) ),13,10    ; 'pi' is a known constant

    ;/*-------------------------------------------------------------*/
    ;/* In the following @fSlv8() returns the local x, because it's */
    ;/* type is the same as the return-type of the macro.           */
    ;/*-------------------------------------------------------------*/
    print real8$( @fSlv8(x = x^2) ),13,10

    mov sz[0],0
    print cat$(ADDR sz,"the logarithm of 12 to base 3 is: ",real8$( @fSlv8(x= logbx(3,12) ) )),13,10
    mov sz[0],0
    print cat$(ADDR sz,"3^",real8$(x)," = ",real8$( @fSlv8(x= 3^x))),13,10

    ;/*-------------------------------------------------------------------------------------------------------------*/
    ;/* The next line show the usage of comparison macros. Each of them returns a Boolean BYTE variable (0,1).  */
    ;/* This allows to call more than one macros per .IF-line. The returned byte is either a local, or if no    */
    ;/* local storage available, a global variable (_BSS).                                                        */
    ;/*-------------------------------------------------------------------------------------------------------------*/
    .if fEQ( 1 , 2.E-7 ) || fGT( @fSlv4(x^2) , y )
            print "whatever",13,10
    .endif

    ;/*---------------------------------------------*/
    ;/* This example shows how to use Attributes   */
    ;/* to force new (local) precision settings    */
    ;/* {i2,r4} -> SWORD and REAL4                 */
    ;/*---------------------------------------------*/
    fSlv y = 1*2+3*4.0 +_c {i2,r4}

    .if fEQ( 1.2 , x {r8})  ; 1.2 will be a REAL8 constant
          ;...
    .endif

    inkey
    exit
main endp
end main
```