z/OS

IBM

Language Environment Vendor Interfaces Supplement

Contents

__to_xx() – C/C++ Compiler Casting Support

Standards

Standards / Extensions	C or C++	Dependencies
		z/OS V1R8

Format

The following prototypes are not supplied in any header file, so they must be defined before these functions can be used.

```
#ifdef __cplusplus
extern "C" {
#endif

float       __to_b1(unsigned int conv, void *value_p);
double      __to_b2(unsigned int conv, void *value_p);
long double __to_b4(unsigned int conv, void *value_p);
_Decimal32  __to_d1(unsigned int conv, void *value_p);
_Decimal64  __to_d2(unsigned int conv, void *value_p);
_Decimal128 __to_d4(unsigned int conv, void *value_p);
float       __to_h1(unsigned int conv, void *value_p);
double      __to_h2(unsigned int conv, void *value_p);
long double __to_h4(unsigned int conv, void *value_p);

#ifdef __cplusplus
}
#endif

#pragma map(__to_b1, "\174\174TO\174B1")
#pragma map(__to_b2, "\174\174TO\174B2")
#pragma map(__to_b4, "\174\174TO\174B4")
#pragma map(__to_d1, "\174\174TO\174D1")
#pragma map(__to_d2, "\174\174TO\174D2")
#pragma map(__to_d4, "\174\174TO\174D4")
#pragma map(__to_h1, "\174\174TO\174H1")
#pragma map(__to_h2, "\174\174TO\174H2")
#pragma map(__to_h4, "\174\174TO\174H4")
```

Note: The only names that can be called are the #pragma mapped names beginning with @@. These names are also the default compiler short names.

General Description

These functions convert an input floating-point number pointed to by *value_p* to an output floating-point number of the return type shown in the prototypes in the previous format section. The *conv* parameter specifies the type of the input floating-point number, as well as the rounding mode to use. The return values of __to_b1(), __to_b2(), and __to_b4() are always binary floating-point numbers of the indicated length. The return values of __to_h1(), __to_h2(), and __to_h4() are always hexadecimal floating-point numbers of the indicated length.

Table 1. Arguments for __to_xx()

Argument	Description
conv	Conversion descriptor: Bits 0-18 – must be 0 Bit 19 – allow/suppress exceptions • 0 - do not suppress any hardware exceptions during the conversion. • 1 - suppress any hardware exceptions using the FPC register or program check shunting in Language Environment. **Note:** This program check shunting does not suppress HFP exponent overflow exceptions when all of the following conditions are met: • TRAP(ON,NOSPIE) is in effect. • Using __to_h1() to convert input hexadecimal floating-point double or long double values, or using __to_h2() to convert input hexadecimal long double values. • The input value is too large to convert to the shorter format, causing the hardware to report an HFP exponent overflow. Bits 20-23 – type of input value: • 0 - Hexadecimal float • 1 - Hexadecimal double • 2 - Hexadecimal long double • 5 - Binary float • 6 - Binary double • 7 - Binary long double • 8 - _Decimal32 • 9 - _Decimal64 • A - _Decimal128 • Other values are not valid Bits 24-27 – exception control flags (These bits must all be zero when converting from DFP input to DFP output, BFP input to BFP output, or HFP input to HFP output. They are honored only when converting between any two of the following: DFP, BFP, HFP.) Bit 24 – inexact suppression control • 0 - IEEE-inexact exceptions are recognized and reported in the normal manner. • 1 - IEEE-inexact exceptions are not recognized. Bit 25 – must be zero Bit 26 – HFP-overflow control • 0 - HFP-overflow exceptions are reported as IEEE-invalid-operation exceptions and are subject to the IEEE-invalid-operation mask. • 1 - HFP-overflow exceptions are reported as IEEE-overflow exceptions and are subject to the IEEE-overflow mask. This bit must be zero unless the output value is HFP.

Table 1. Arguments for __to_xx() (continued)

Argument	Description
	Bit 27 - HFP-underflow control/DFP quantum control If the output number is HFP (underflow control) : • 0 - HFP underflow causes the result to be set to a true zero with the same sign as the input, and the underflow is not reported. The result in this case is inexact and is subject to the inexact-suppression control (Bit 24) • 1 - HFP underflow is reported as an IEEE-underflow exception, and is subject to the IEEE-underflow mask. If the output number is DFP (quantum control) : • 0 - The preferred quantum for exact DFP results is the maximum possible, which means that trailing zeroes are removed and the exponent is adjusted upward, if possible. For example, _Decimal32 100.0 becomes +0000001.E+02. • 1 - The preferred quantum for exact DFP results is 1, which means that the exponent is 0, when possible, and the output number is an integer that may have trailing zeroes. For example: _Decimal32 100.0 becomes +0000100.E+00, and _Decimal32 1000000000.0 becomes +1000000.E+03. Bits 28-31 – rounding mode: • 0 - according to DFP rounding mode in FPC • 1 - according to BFP rounding mode in FPC • 8 - round to nearest, ties to even • 9 - round towards 0 • A - round toward +infinity • B - round toward –infinity • C - round to nearest, ties away from 0 • D - round to nearest, ties toward 0 • E - round away from 0 • F - round to prepare for shorter precision • Other values are not valid **Notes:** 1. When converting a binary floating-point value to a shorter binary floating-point value, rounding mode must be 1. 2. When converting a decimal floating-point value to a shorter decimal floating-point value, the rounding mode must be 0, 8, 9, A, B, C, D, E, F. 3. When converting a hexadecimal floating-point value to another hexadecimal floating-point value, the rounding mode must be valid, but does not affect the result, which is rounded by the hardware. 4. When converting a decimal floating-point value to a longer decimal floating-point value, or a binary floating-point value to a longer binary floating-point value, the rounding mode must be valid, but is otherwise ignored (there is no rounding). 5. When the input and output type is the same (no conversion), the rounding mode must be valid, but is otherwise ignored.
value_p	Pointer to the input floating-point value to be converted to the return type for this function. The type of the floating-point value depends on the *conv* parameter.

Returned Value

These functions return floating-point values as shown in the prototypes. When the *conv* parameter is not valid, the floating-point return value is 0.0. The return value is undefined when the input floating-point number cannot be converted to a return value of the requested type.

These functions do not set errno.

Note: When either the input value or output value are not hexadecimal floating-point, the raising of IEEE exceptions is allowed or suppressed by the combination of the five exception control flags and the current value of the exception mask bits in the floating-point control register (FPC). If both the input and output values are HFP, the raising of exceptions is controlled by the program mask in the PSW and Bit 19 in the exception control flags.

Related Information

There are no prototypes provided for these functions. These functions are called by the compiler to support casting operations.

| When executing on hardware that does not have the PFPO facility installed, the IEEE Interruption-Simulation (IIS) facility reports some of the exceptions that can occur when the conversion between numbers is in any two of the following formats:

- BFP
- DFP
- HFP

| IIS might cause the contents of the floating-point control (FPC) register to be different from regular IEEE exceptions. In particular, the FPC flags and DXC bytes are different. See *z/Architecture Principles of Operation* for more information about the FPC register contents after IIS events.