FP_IEEE_DENORM_GET_ Procedure

Summary

The FP_IEEE_DENORM_GET_ procedure reads the IEEE floating-point denormalization mode.

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
fp_ieee_denorm FP_IEEE_DENORM_GET_ (void);
```

Parameters

DeNorm output

INT

The denormalization control mode.

DeNorm can have the following values:

FP_IEEE_DENORMALIZATION_ENABLE	Denormalization in IEEE floating point allows for greater precision in the representation of numbers that are very close to zero. This is the standard mode.
FP_IEEE_DENORMALIZATION_DISABLE	The nonstandard mode. When denormalization is disabled, fractions that are too small to be represented in standard IEEE form are represented as zero, causing a loss of precision.

FP_IEEE_DENORM_SET_ Procedure

Summary

The FP_IEEE_DENORM_SET_ procedure sets the IEEE floating-point denormalization mode.

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
void FP_IEEE_DENORM_SET_( fp_ieee_denorm new_mode );
```

Parameters

NewMode input

INT

The denormalization control mode.

NewMode can have the following values:

FP_IEEE_DENORMALIZATION_ENABLE	Denormalization in IEEE floating point allows for greater precision in the representation of numbers that are very close to zero. This is the standard mode.
FP_IEEE_DENORMALIZATION_DISABLE	The nonstandard mode. When denormalization is disabled, fractions that are too small to be represented in standard IEEE form are represented as zero, causing a loss of precision.

Consideration

Operations with denormalization disabled can cause problems by causing a gap around zero in the distribution of values that can be represented. With denormalization disabled, the results will not comply with the IEEE standard and might not match results on any other system.

FP_IEEE_ENABLES_GET_ Procedure

Summary

The FP_IEEE_ENABLES_GET_ procedure reads the IEEE floating-point trap enable mask. A set bit (value of one) means that the trap for that particular exception is enabled. A zero bit means that it is disabled.

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
fp_ieee_enables FP_IEEE_ENABLES_GET_(void);
```

Parameters

Traps input

INT

The 32-bit trap enable mask.

Mask bit values of *Traps* are:

FP_IEEE_ENABLE_INVALID	Trap on FP_IEEE_INVALID exception.
FP_IEEE_ENABLE_DIVBYZERO	Trap on FP_IEEE_DIVBYZERO exception.
FP_IEEE_ENABLE_OVERFLOW	Trap on FP_IEEE_OVERFLOW exception.
FP_IEEE_ENABLE_UNDERFLOW	Trap on FP_IEEE_UNDERFLOW exception.
FP_IEEE_ENABLE_INEXACT	Trap on FP_IEEE_INEXACT exception.

Considerations

- A constant named FP_IEEE_ALL_ENABLES is equivalent to a combination of the mask bits to enable traps for all the exceptions.
- In some cases, the conditions that cause a trap are slightly different from the conditions that cause the corresponding exception flag to be set.
- When a trap happens, a `SIGFPE` signal is raised, and the corresponding signal handler is called. The `SIGFPE` signal handler typically does a function frame trace showing the point of failure, and then abends the process. The `SIGFPE` signal is not allowed to return to the point where the trap happened.
- Trap handling is an optional part of the IEEE floating-point standard. See FP_IEEE_EXCEPTIONS_GET_ Procedure on page -10 and FP_IEEE_EXCEPTIONS_SET_ Procedure on page -12 for an alternative to using traps.
- The compiler optimizer might reorder operations within a local routine and cause different results from the FP_ IEEE status procedures than intended. To work around this, place arithmetic operations in a separate function. The compiler cannot optimize across function boundaries, so the FP_IEEE status procedure will be called in the intended order.

FP_IEEE_ENABLES_SET_ Procedure

Summary

The FP_IEEE_ENABLES_SET_ procedure sets the IEEE floating-point trap enable mask. A set bit (value of one) enables a trap for the particular exception. A zero bit (the normal value) disables that trap.

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
void FP_IEEE_ENABLES_SET_( fp_ieee_enables new_mask );
```

Parameters

NewMask input

INT

The 32-bit traps flag.

Traps flag values of *Traps* are:

FP_IEEE_ENABLE_INVALID	Trap on FP_IEEE_INVALID exception.
FP_IEEE_ENABLE_DIVBYZERO	Trap on FP_IEEE_DIVBYZERO exception.
FP_IEEE_ENABLE_OVERFLOW	Trap on FP_IEEE_OVERFLOW exception.
FP_IEEE_ENABLE_UNDERFLOW	Trap on FP_IEEE_UNDERFLOW exception.
FP_IEEE_ENABLE_INEXACT	Trap on FP_IEEE_INEXACT exception.

Considerations

- When you enable traps, you will not get a trap from a left-over status; you will trap only from operations that happen after you enable the traps.
- See Considerations on page -5 for more considerations for this procedure.

Example

```
#include <kfpieee.h>

void TrapsEnableExample(void) {

   FP_IEEE_ENABLES_SET_
         ( FP_IEEE_ENABLE_INVALID   |
           FP_IEEE_ENABLE_DIVBYZERO|
           FP_IEEE_ENABLE_OVERFLOW
         );
}
```

This sets traps on the FP_IEEE_INVALID, FP_IEEE_DIVBYZERO, and FP_IEEE_OVERFLOW exceptions.

FP_IEEE_ENV_CLEAR_ Procedure

Summary

The FP_IEEE_ENV_CLEAR_ procedure sets the floating-point environment (consisting of the rounding mode, the exception flags, the trap enables, and the denormalization mode) back to its initial values. The initial values are as follows:

Rounding mode	Round to nearest or nearest even value
Exception flags	No exceptions encountered (zeroes)
Trap enables	All floating-point traps disabled
Denormalization	Denormalized enabled

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
fp_ieee_env FP_IEEE_ENV_CLEAR_(void);
```

Parameters

SavedEnv input

INT

The current floating-point environment is saved here before it is set to its initial values.

Consideration

FP_IEEE_ENV_CLEAR_ and FP_IEEE_ENV_RESUME_ are for use by a process, such as a signal handler, a clean-up routine, or a procedure that needs to tolerate being called with any possible values in the floating-point status and control. They are not for use by interrupt handlers.

Example

```
#include <kfpieee.h>

void TotalEnvExample(void) {
   fp_ieee_env previousEnv;
   previousEnv = FP_IEEE_ENV_CLEAR_(); /*restore initial env*/
   Do_Computation();
   FP_IEEE_ENV_RESUME_( previousEnv )  /*restore previous env*/
}
```

FP_IEEE_ENV_RESUME_ Procedure

Summary

The FP_IEEE_ENV_RESUME_ procedure restores the floating-point environment (the rounding mode, the exception flags, the trap enables, and the denormalization mode) to the values it had before calling FP_IEEE_ENV_CLEAR_.

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
void FP_IEEE_ENV_RESUME_( fp_ieee_env savedEnv );
```

Parameters

SavedEnv input

INT

The previous floating-point environment that was saved by the last call to FP_IEEE_ENV_CLEAR_.

Considerations

See Consideration on page -8 for a description of considerations for this procedure.

Examples

See Example on page -8 for an example of the use of this procedure.

FP_IEEE_EXCEPTIONS_GET_ Procedure

Summary

The FP_IEEE_EXCEPTIONS_GET_ procedure reads the IEEE floating-point exception mask.

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
fp_ieee_exceptions FP_IEEE_EXCEPTIONS_GET_(void);
```

Parameters

Exceptions input

INT

The 32-bit exception flags.

Exception flag values of *Exceptions* are:

Value	Cause
FP_IEEE_INVALID	Arithmetic calculations using either positive or negative infinity as an operand, zero divided by zero, the square root of -1, the rem function with zero as a divisor (which causes divide by zero), comparisons with invalid numbers, or impossible binary-decimal conversions.
FP_IEEE_DIVBYZERO	Computing x/0, where x is finite and nonzero.
FP_IEEE_OVERFLOW	Result too large to represent as a normalized number.
FP_IEEE_UNDERFLOW	Result both inexact and too small to represent as a normalized number.
FP_IEEE_INEXACT	Result less accurate than it could have been with a larger exponent range or more fraction bits. Most commonly set when rounding off a repeating fraction such as 1.0/3.0. Also set for underflow cases and some overflow cases, but not for division by zero.

Considerations

- In addition to the above enumerated constants, a constant named FP_IEEE_ALL_EXCEPTS is equivalent to a combination of all the exception bits.
- Once exception flags are set, they stay set until explicitly reset.
- More than one exception flag can result from a single floating-point operation.

Example

```
#include <kfpieee.h>

void Example(void) {

   FP_IEEE_EXCEPTIONS_SET_( 0 );   /* clear exceptions */
   DoComputation();   /* floating-point computation */
   if( FP_IEEE_EXCEPTIONS_GET_() &
      (FP_IEEE_INVALID|FP_IEEE_OVERFLOW|FP_IEEE_DIVBYZERO)
     )

      printf( "Trouble in computation! \n" );
}
```

FP_IEEE_EXCEPTIONS_SET_ Procedure

Summary

The FP_IEEE_EXCEPTIONS_SET_ procedure sets the IEEE floating-point exception mask.

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
void FP_IEEE_EXCEPTIONS_SET_
   ( fp_ieee_exceptions new_flags );
```

Parameters

NewFlags input

INT

The 32-bit exception flags.

Exception flag values of *NewFlags* are:

FP_IEEE_INVALID	Arithmetic calculations using either positive or negative infinity as an operand, zero divided by zero, the square root of -1, the rem function with zero as a divisor (which causes divide by zero), comparisons with invalid numbers, or impossible binary-decimal conversions.
FP_IEEE_DIVBYZERO	Computing x/0, where x is finite and nonzero.
FP_IEEE_OVERFLOW	Result too large to represent as a normalized number.
FP_IEEE_UNDERFLOW	Result both inexact and too small to represent as a normalized number.
FP_IEEE_INEXACT	Result less accurate than it could have been with a larger exponent range or more fraction bits. Most commonly set when rounding off a repeating fraction such as 1.0/3.0. Also set for underflow cases and some overflow cases, but not for division by zero.

Considerations

See Considerations on page -11 for a description of considerations for this procedure.

Examples

See Example on page -11 for examples of the use of this call.

FP_IEEE_ROUND_GET_ Procedure

Summary

The FP_IEEE_ROUND_GET_ procedure reads the current rounding mode.

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
p_ieee_round FP_IEEE_ROUND_GET_(void);
```

Parameters

RoundMode input

INT

The 32-bit rounding mode code.

Rounding mode values returned by this procedure are:

FP_IEEE_ROUND_NEAREST	Round toward the representable value nearest the true result. In cases where there are two equally near values, the "even" value (the value with the least-significant bit zero) is chosen (the standard rounding mode).
FP_IEEE_ROUND_UPWARD	Round up (toward plus infinity).
FP_IEEE_ROUND_DOWNWARD	Round down (toward minus infinity).
FP_IEEE_ROUND_TOWARDZERO	Round toward zero (truncate).

FP_IEEE_ROUND_SET_ Procedure

Summary

The FP_IEEE_ROUND_SET_ procedure sets the current rounding mode.

Note. This procedure is supported in the G06.06 release and all subsequent G-series releases.

Syntax

```
#include <kfpieee.h>
void FP_IEEE_ROUND_SET_( fp_ieee_round new_mode );
```

Parameters

NewMode input

INT

The 32-bit rounding mode code.

The rounding mode can have one of the following values:

FP_IEEE_ROUND_NEAREST	Round toward the representable value nearest the true result. In cases where there are two equally near values, the "even" value (the value with the least-significant bit zero) is chosen (the standard rounding mode).
FP_IEEE_ROUND_UPWARD	Round up (toward plus infinity).
FP_IEEE_ROUND_DOWNWARD	Round down (toward minus infinity).
FP_IEEE_ROUND_TOWARDZERO	Round toward zero (truncate).