

4.9.1 Floating-Point Exception Conditions

The following sections describe the various conditions that cause a floating-point exception to be generated and the masked response of the processor when these conditions are detected. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A & 3B*, list the floating-point exceptions that can be signaled for each floating-point instruction.

4.9.1.1 Invalid Operation Exception (#I)

The processor reports an invalid operation exception in response to one or more invalid arithmetic operands. If the invalid operation exception is masked, the processor sets the IE flag and returns an indefinite value or a QNaN. This value overwrites the destination register specified by the instruction. If the invalid operation exception is not masked, the IE flag is set, a software exception handler is invoked, and the operands remain unaltered.

See Section 4.8.3.6, "Using SNaNs and QNaNs in Applications," for information about the result returned when an exception is caused by an SNaN.

The processor can detect a variety of invalid arithmetic operations that can be coded in a program. These operations generally indicate a programming error, such as dividing ∞ by ∞ . See the following sections for information regarding the invalid-operation exception when detected while executing x87 FPU or SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.1, "Invalid Operation Exception"
- SIMD floating-point exceptions; Section 11.5.2.1, "Invalid Operation Exception (#I)"

4.9.1.2 Denormal Operand Exception (#D)

The processor reports the denormal-operand exception if an arithmetic instruction attempts to operate on a denormal operand (see Section 4.8.3.2, "Normalized and Denormalized Finite Numbers"). When the exception is masked, the processor sets the DE flag and proceeds with the instruction. Operating on denormal numbers will produce results at least as good as, and often better than, what can be obtained when denormal numbers are flushed to zero. Programmers can mask this exception so that a computation may proceed, then analyze any loss of accuracy when the final result is delivered.

When a denormal-operand exception is not masked, the DE flag is set, a software exception handler is invoked, and the operands remain unaltered. When denormal operands have reduced significance due to loss of low-order bits, it may be advisable to not operate on them. Precluding denormal operands from computations can be accomplished by an exception handler that responds to unmasked denormal-operand exceptions.

See the following sections for information regarding the denormal-operand exception when detected while executing x87 FPU or SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.2, "Denormal Operand Exception (#D)"
- SIMD floating-point exceptions; Section 11.5.2.2, "Denormal-Operand Exception (#D)"

4.9.1.3 Divide-By-Zero Exception (#Z)

The processor reports the floating-point divide-by-zero exception whenever an instruction attempts to divide a finite non-zero operand by 0. The masked response for the divide-by-zero exception is to set the ZE flag and return an infinity signed with the exclusive OR of the sign of the operands. If the divide-by-zero exception is not masked, the ZE flag is set, a software exception handler is invoked, and the operands remain unaltered.

See the following sections for information regarding the divide-by-zero exception when detected while executing x87 FPU or SSE/SSE2 instructions:

- x87 FPU; Section 8.5.3, "Divide-By-Zero Exception (#Z)"
- SIMD floating-point exceptions; Section 11.5.2.3, "Divide-By-Zero Exception (#Z)"

4.9.1.4 Numeric Overflow Exception (#O)

The processor reports a floating-point numeric overflow exception whenever the rounded result of an instruction exceeds the largest allowable finite value that will fit into the destination operand. Table 4-9 shows the threshold range for numeric overflow for each of the floating-point formats; overflow occurs when a rounded result falls at or outside this threshold range.

DATA TYPES

Table 4-9. Numeric Overflow Thresholds

Floating-Point Format	Overflow Thresholds
Single Precision	$ x \geq 1.0 * 2^{128}$
Double Precision	$ x \geq 1.0 * 2^{1024}$
Double Extended Precision	$ x \geq 1.0 * 2^{16384}$

When a numeric-overflow exception occurs and the exception is masked, the processor sets the OE flag and returns one of the values shown in Table 4-10, according to the current rounding mode. See Section 4.8.4, "Rounding."

When numeric overflow occurs and the numeric-overflow exception is not masked, the OE flag is set, a software exception handler is invoked, and the source and destination operands either remain unchanged or a biased result is stored in the destination operand (depending whether the overflow exception was generated during an SSE/SSE2/SSE3 floating-point operation or an x87 FPU operation).

Table 4-10. Masked Responses to Numeric Overflow

Rounding Mode	Sign of True Result	Result
To nearest	+	$+\infty$
	-	$-\infty$
Toward $-\infty$	+	Largest finite positive number
	-	$-\infty$
Toward $+\infty$	+	$+\infty$
	-	Largest finite negative number
Toward zero	+	Largest finite positive number
	-	Largest finite negative number

See the following sections for information regarding the numeric overflow exception when detected while executing x87 FPU instructions or while executing SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.4, "Numeric Overflow Exception (#O)"
- SIMD floating-point exceptions; Section 11.5.2.4, "Numeric Overflow Exception (#O)"

4.9.1.5 Numeric Underflow Exception (#U)

The processor detects a floating-point numeric underflow condition whenever the result of rounding with unbounded exponent (taking into account precision control for x87) is tiny; that is, less than the smallest possible normalized, finite value that will fit into the destination operand. Table 4-11 shows the threshold range for

numeric underflow for each of the floating-point formats (assuming normalized results); underflow occurs when a rounded result falls strictly within the threshold range. The ability to detect and handle underflow is provided to prevent a very small result from propagating through a computation and causing another exception (such as overflow during division) to be generated at a later time.

Table 4-11. Numeric Underflow (Normalized) Thresholds

Floating-Point Format	Underflow Thresholds*
Single Precision	$ x < 1.0 * 2^{-126}$
Double Precision	$ x < 1.0 * 2^{-1022}$
Double Extended Precision	$ x < 1.0 * 2^{-16382}$

* Where 'x' is the result rounded to destination precision with an unbounded exponent range.

How the processor handles an underflow condition, depends on two related conditions:

- creation of a tiny result
- creation of an inexact result; that is, a result that cannot be represented exactly in the destination format

Which of these events causes an underflow exception to be reported and how the processor responds to the exception condition depends on whether the underflow exception is masked:

- **Underflow exception masked** — The underflow exception is reported (the UE flag is set) only when the result is both tiny and inexact. The processor returns a denormalized result to the destination operand, regardless of inexactness.
- **Underflow exception not masked** — The underflow exception is reported when the result is tiny, regardless of inexactness. The processor leaves the source and destination operands unaltered or stores a biased result in the designating operand (depending whether the underflow exception was generated during an SSE/SSE2/SSE3 floating-point operation or an x87 FPU operation) and invokes a software exception handler.

See the following sections for information regarding the numeric underflow exception when detected while executing x87 FPU instructions or while executing SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.5, "Numeric Underflow Exception (#U)"
- SIMD floating-point exceptions; Section 11.5.2.5, "Numeric Underflow Exception (#U)"

4.9.1.6 Inexact-Result (Precision) Exception (#P)

The inexact-result exception (also called the precision exception) occurs if the result of an operation is not exactly representable in the destination format. For example, the fraction 1/3 cannot be precisely represented in binary floating-point form. This

DATA TYPES

exception occurs frequently and indicates that some (normally acceptable) accuracy will be lost due to rounding. The exception is supported for applications that need to perform exact arithmetic only. Because the rounded result is generally satisfactory for most applications, this exception is commonly masked.

If the inexact-result exception is masked when an inexact-result condition occurs and a numeric overflow or underflow condition has not occurred, the processor sets the PE flag and stores the rounded result in the destination operand. The current rounding mode determines the method used to round the result. See Section 4.8.4, "Rounding."

If the inexact-result exception is not masked when an inexact result occurs and numeric overflow or underflow has not occurred, the PE flag is set, the rounded result is stored in the destination operand, and a software exception handler is invoked.

If an inexact result occurs in conjunction with numeric overflow or underflow, one of the following operations is carried out:

- If an inexact result occurs along with masked overflow or underflow, the OE flag or UE flag and the PE flag are set and the result is stored as described for the overflow or underflow exceptions; see Section 4.9.1.4, "Numeric Overflow Exception (#O)," or Section 4.9.1.5, "Numeric Underflow Exception (#U)." If the inexact result exception is unmasked, the processor also invokes a software exception handler.
- If an inexact result occurs along with unmasked overflow or underflow and the destination operand is a register, the OE or UE flag and the PE flag are set, the result is stored as described for the overflow or underflow exceptions, and a software exception handler is invoked.

If an unmasked numeric overflow or underflow exception occurs and the destination operand is a memory location (which can happen only for a floating-point store), the inexact-result condition is not reported and the C1 flag is cleared.

See the following sections for information regarding the inexact-result exception when detected while executing x87 FPU or SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.6, "Inexact-Result (Precision) Exception (#P)"
- SIMD floating-point exceptions; Section 11.5.2.3, "Divide-By-Zero Exception (#Z)"

4.9.2 Floating-Point Exception Priority

The processor handles exceptions according to a predetermined precedence. When an instruction generates two or more exception conditions, the exception precedence sometimes results in the higher-priority exception being handled and the lower-priority exceptions being ignored. For example, dividing an SNaN by zero can potentially signal an invalid-operation exception (due to the SNaN operand) and a divide-by-zero exception. Here, if both exceptions are masked, the processor handles the higher-priority exception only (the invalid-operation exception), returning a QNaN to the destination. Alternately, a denormal-operand or inexact-result exception can

accompany a numeric underflow or overflow exception with both exceptions being handled.

The precedence for floating-point exceptions is as follows:

1. Invalid-operation exception, subdivided as follows:
 - a. stack underflow (occurs with x87 FPU only)
 - b. stack overflow (occurs with x87 FPU only)
 - c. operand of unsupported format (occurs with x87 FPU only when using the double extended-precision floating-point format)
 - d. SNaN operand
2. QNaN operand. Though this is not an exception, the handling of a QNaN operand has precedence over lower-priority exceptions. For example, a QNaN divided by zero results in a QNaN, not a zero-divide exception.
3. Any other invalid-operation exception not mentioned above or a divide-by-zero exception.
4. Denormal-operand exception. If masked, then instruction execution continues and a lower-priority exception can occur as well.
5. Numeric overflow and underflow exceptions; possibly in conjunction with the inexact-result exception.
6. Inexact-result exception.

Invalid operation, zero divide, and denormal operand exceptions are detected before a floating-point operation begins. Overflow, underflow, and precision exceptions are not detected until a true result has been computed. When an unmasked **pre-operation** exception is detected, the destination operand has not yet been updated, and appears as if the offending instruction has not been executed. When an unmasked **post-operation** exception is detected, the destination operand may be updated with a result, depending on the nature of the exception (except for SSE/SSE2/SSE3 instructions, which do not update their destination operands in such cases).

4.9.3 Typical Actions of a Floating-Point Exception Handler

After the floating-point exception handler is invoked, the processor handles the exception in the same manner that it handles non-floating-point exceptions. The floating-point exception handler is normally part of the operating system or executive software, and it usually invokes a user-registered floating-point exception handle.

A typical action of the exception handler is to store state information in memory. Other typical exception handler actions include:

- Examining the stored state information to determine the nature of the error
- Taking actions to correct the condition that caused the error