N

SPARC IEEE 754 Implementation Recommendations

A number of details in ANSI/IEEE 754 floating-point standard are left to be defined by implementations, and so are not specified in this document. In order to promote increased portability among new SPARC implementations of programs such as instruction set test vectors, the following recommendations are designed to eliminate many uncertainties, especially with regard to exceptional situations. These recommendations, perhaps modified slightly in the light of subsequent experience, are intended to be incorporated as requirements in a future SPARC revision.

N.1. Unaligned floating-point data registers

The effect of executing an instruction that refers to an unaligned floating-point register operand (double-precision operand in a register whose number is not 0 mod 2, or quadruple-precision operand in a register not 0 mod 4) is undefined in Chapter 4. An illegal_instruction trap should occur in this case.

N.2. Reading an empty FQ

The effect of reading an empty floating-point queue is not specified in Chapter 4. A trap handler which attempts to read such a queue contains a software error. A sequence_error fp_exception trap occurs in this case.

N.3. Traps inhibit results

To summarize what happens when a floating-point trap occurs, as described in Chapter 4 and elsewhere:

- The destination *f* register is unchanged
- The FSR *fcc* (floating-point condition codes) field is unchanged
- The FSR *aexc* (accrued exceptions) field is unchanged
- The FSR *cexc* (current exceptions) field is unchanged except for IEEE_754_exceptions; in that case, *cexc* contains exactly one bit which is 1, corresponding to the exception that caused the trap

These restrictions are designed to ensure a consistent state for user software. Instructions causing an fp_exception trap due to unfinished or unimplemented FPops execute as if by hardware; that a hardware trap was taken to supervisor software is undetectable by user software except possibly by timing considerations. A user-mode trap handler invoked for an IEEE 754 exception, whether as a direct result of a hardware IEEE_754_exception or as an indirect result of supervisor handling of an unfinished_FPop or unimplemented_FPop, may rely on the following:

- Supervisor software will pass it the address of the instruction which caused the exception, extracted from a deferred trap queue or elsewhere
- The destination *f* register is unchanged from its state prior to that instruction's execution
- The FSR *fcc* is unchanged
- The FSR *aexc* is unchanged
- The FSR *cexc* contains one bit on for the exception that caused the trap
- The FSR *ftt*, *qne*, *u*, and *res* fields are zero

Supervisor software is responsible for enforcing these requirements if the hardware trap mechanism does not.

N.4. NaN operand and result definitions

An untrapped floating-point result can be in a format which is either the same as, or different from, the format of the source operands. These two cases are described separately, below.

Untrapped floating-point result in different format from operands

F[sdq]TO[sdq], with quiet NaN operand: no exception caused; result is a quiet NaN. The operand is transformed as follows:

> NaN transformation: The most significant bits of the operand fraction are copied to the most significant bits of the result fraction. When converting to a narrower format, excess low order bits of the operand fraction are discarded. When converting to a wider format, excess low order bits of the result fraction are set to 0. The quiet bit (the most significant bit of the result fraction) is always set to 1, so the NaN transformation always produces a quiet NaN.

F[sdq]TO[sdq], signaling NaN operand: invalid exception, result is the signaling NaN operand processed by the NaN transformation above to produce a quiet NaN.

FCMPE[sdq] with any NaN operand: invalid exception, unordered *fcc*.

FCMP[sdq] with any signaling NaN operand: invalid exception, unordered *fcc*.

FCMP[sdq] with any quiet NaN operand but no signaling NaN operand: no exception, unordered *fcc*.

Untrapped floating-point result in same format as operands

No NaN operand: for an invalid exception such as sqrt(−1.0) or 0.0÷0.0, the result is the quiet NaN with sign = 0, exponent = all 1's, and fraction = all 1's. The sign is 0 to distinguish such results from storage initialized to all '1' bits.

One operand, quiet NaN: no exception, result is the quiet NaN operand.

One operand, signaling NaN: invalid exception, result is the signaling NaN with its quiet bit (most significant bit of fraction field) set to 1.

Two operands, both quiet: no exception, result is the rs2 (second source) operand.

Two operands, both signaling: invalid exception, result is the rs2 operand with the quiet bit set to 1.

Two operands, just one a signaling NaN: invalid exception, result is the signaling NaN operand with the quiet bit set to 1.

Two operands, neither signaling NaN, just one a quiet NaN: no exception, result is the quiet NaN operand.

In the following tabular representation of the untrapped results, NaN*n* means the NaN in rs*n*, Q means quiet, S signaling:

		rs2 operand		
		number	QNaN2	SNaN2
rs1 operand	none	IEEE 754	QNaN2	QSNaN2
	number	IEEE 754	QNaN2	QSNaN2
	QNaN1	QNaN1	QNaN2	QSNaN2
	SNaN1	QSNaN1	QSNaN1	QSNaN2

QSNaN*n* means a quiet NaN produced by the NaN transformation on a signaling NaN from rs*n*; the invalid exception is always signaled. The QNaN*n* results in the table never generate an exception, but IEEE 754 specifies a number of cases of invalid exceptions and QNaN results from operands that are both numbers.

N.5. Trapped Underflow definition (UFM=1)

Underflow occurs if the correct unrounded result has magnitude between zero and the smallest normalized number in the destination format. In terms of IEEE 754, this means "tininess detected before rounding".

Note that the wrapped exponent results intended to be delivered on trapped underflows and overflows in IEEE 754 aren't relevant to SPARC at the hardware/supervisor levels; if they are created at all, it would be by user software in a user-mode trap handler.

N.6. Untrapped underflow definition (UFM=0)

Underflow occurs if the correct unrounded result has magnitude between zero and the smallest normalized number in the destination format, and the correctly rounded result in the destination format is inexact; that result may be zero, subnormal, or the smallest normalized number. In terms of IEEE 754, this means "tininess detected before rounding" and "loss of accuracy detected as inexact".

Note that floating-point overflow is defined to be detected **after** rounding; the foregoing underflow definition simplifies hardware implementation and testing.

The following table summarizes what happens when an exact **unrounded** value u satisfying

$$0 \le |u| \le \textit{smallest normalized number}$$

would round, if no trap intervened, to a **rounded** value r which might be zero, subnormal, or the smallest normalized value. "UF" means underflow trap (with ufc set in *cexc*), "NX" means inexact trap (with nxc set in *cexc*), "uf" means untrapped underflow exception (ufc set in *cexc* and ufa in *aexc*), and "nx" means untrapped inexact exception (nxc set in *cexc* and nxa in *aexc*).

	underflow trap inexact trap	UFM=1 NXM=?	UFM=0 NXM=1	UFM=0 NXM=0
$u = r$	r is minimum normal	none	none	none
	r is subnormal	UF	none	none
	r is zero	none	none	none
$u \neq r$	r is minimum normal	UF	NX	uf nx
	r is subnormal	UF	NX	uf nx
	r is zero	UF	NX	uf nx

N.7. Integer overflow definition

F[sdq]TOi: when a NaN, infinity, large positive argument ≥ 2147483648.0, or large negative argument ≤ –2147483649.0, is converted to integer, the resulting exception is invalid. If no trap occurs and the sign bit of the operand is positive (i.e., is 0), the numerical result is 2147483647. If no trap occurs and the sign bit of the operand is negative (i.e., is 1), the numerical result is –2147483648.

N.8. Nonstandard mode

SPARC implementations are permitted but not encouraged to deviate from SPARC requirements when the nonstandard mode bit of the FSR is 1. Some implementations use that bit to provide alternative handling of subnormal floating-point operands and results that avoids unfinished_FPop traps with consequent poor performance on programs that underflow frequently.

Such traps could be avoided by proper system design. Cache misses in the CPU cause holds in the FPU, in order for extra cycles to occur to refill the cache, so that their occurrence is invisible to software and doesn't degrade performance in the normal cache hit case. Similarly "subnormal misses" in the FPU can be avoided by any of several better implementation techniques that avoid causing an unfinished_FPop trap or degrading performance in the normal case. One way is to cause subnormal misses in the FPU to hold the CPU, so that operand or result alignment can take a few extra cycles without any other effect on software. Another way to avoid extra cycles is to provide extra normalization hardware for operands and results.

So the best implementation of nonstandard mode is a no-op: nonstandard mode runs identically to the standard SPARC mode. Such implementations identify themselves in the NS bit of the FSR, which always reads 0, even after a 1 is written.

Second best is to implement nonstandard mode for subnormal operands and results as outlined below so that implementations behave uniformly:

Subnormal operands
: In nonstandard mode are replaced by zeros with the same sign. An inexact exception always arises if no other exception would, and so traps if NXM=1.

Untrapped subnormal results
: In nonstandard mode are replaced by zeros with the same sign. Underflow and inexact exceptions always arise. In terms of the previous table:

	underflow trap inexact trap	UFM=1 NXM=?	UFM=0 NXM=1	UFM=0 NXM=0
$u = r$	r is minimum normal	none	none	none
	r is zero	none	none	none
$u \neq r$	r is minimum normal	UF	NX	uf nx
	r is zero	UF	NX	uf nx