Chapter 26. "Zfa" Standard Extension for Additional Floating-Point Instructions, Version 1.0

This chapter describes the Zfa standard extension, which adds instructions for immediate loads, IEEE 754-2019 minimum and maximum operations, round-to-integer operations, and quiet floating-point comparisons. For RV32D, the Zfa extension also adds instructions to transfer double-precision floating-point values to and from integer registers, and for RV64Q, it adds analogous instructions for quad-precision floating-point values. The Zfa extension depends on the F extension.

26.1. Load-Immediate Instructions

The FLI.S instruction loads one of 32 single-precision floating-point constants, encoded in the *rs1* field, into floating-point register *rd*. The correspondence of *rs1* field values and single-precision floating-point values is shown in Table 37. FLI.S is encoded like FMV.W.X, but with *rs2*=1.

rs1	Value	Sign	Exponent	Significand
0	−1.0	1	01111111	000···000
1	*Minimum positive normal*	0	00000001	000···000
2	1.0×2^{-16}	0	01101111	000···000
3	1.0×2^{-15}	0	01110000	000···000
4	1.0×2^{-8}	0	01110111	000···000
5	1.0×2^{-7}	0	01111000	000···000
6	0.0625 (2^{-4})	0	01111011	000···000
7	0.125 (2^{-3})	0	01111100	000···000
8	0.25	0	01111101	000···000
9	0.3125	0	01111101	010···000
10	0.375	0	01111101	100···000
11	0.4375	0	01111101	110···000
12	0.5	0	01111110	000···000
13	0.625	0	01111110	010···000
14	0.75	0	01111110	100···000
15	0.875	0	01111110	110···000
16	1.0	0	01111111	000···000
17	1.25	0	01111111	010···000
18	1.5	0	01111111	100···000
19	1.75	0	01111111	110···000
20	2.0	0	10000000	000···000
21	2.5	0	10000000	010···000
22	3	0	10000000	100···000

rs1	Value	Sign	Exponent	Significand
23	4	0	10000001	000···000
24	8	0	10000010	000···000
25	16	0	10000011	000···000
26	128 (2^7)	0	10000110	000···000
27	256 (2^8)	0	10000111	000···000
28	2^{15}	0	10001110	000···000
29	2^{16}	0	10001111	000···000
30	$+\infty$	0	11111111	000···000
31	*Canonical NaN*	0	11111111	100···000

Table 35. Immediate values loaded by the FLI.S instruction.

The preferred assembly syntax for entries 1, 30, and 31 is `min`*,* `inf`*, and* `nan`*, respectively. For entries 0 through 29 (including entry 1), the assembler will accept decimal constants in C-like syntax.*

The set of 32 constants was chosen by examining floating-point libraries, including the C standard math library, and to optimize fixed-point to floating-point conversion.

Entries 8-22 follow a regular encoding pattern. No entry sets mantissa bits other than the two most significant ones.

If the D extension is implemented, FLI.D performs the analogous operation, but loads a double-precision value into floating-point register *rd*. Note that entry 1 (corresponding to the minimum positive normal value) has a numerically different value for double-precision than for single-precision. FLI.D is encoded like FLI.S, but with *fmt*=D.

If the Q extension is implemented, FLI.Q performs the analogous operation, but loads a quad-precision value into floating-point register *rd*. Note that entry 1 (corresponding to the minimum positive normal value) has a numerically different value for quad-precision. FLI.Q is encoded like FLI.S, but with *fmt*=Q.

If the Zfh or Zvfh extension is implemented, FLI.H performs the analogous operation, but loads a half-precision floating-point value into register *rd*. Note that entry 1 (corresponding to the minimum positive normal value) has a numerically different value for half-precision. Furthermore, since 2^{16} is not representable in half-precision floating-point, entry 29 in the table instead loads positive infinity—i.e., it is redundant with entry 30. FLI.H is encoded like FLI.S, but with *fmt*=H.

Additionally, since 2^{-16} and 2^{-15} are subnormal in half-precision, entry 1 is numerically greater than entries 2 and 3 for FLI.H.

The FLI.*fmt* instructions never set any floating-point exception flags.

26.2. Minimum and Maximum Instructions

The FMINM.S and FMAXM.S instructions are defined like the FMIN.S and FMAX.S instructions,

except that if either input is NaN, the result is the canonical NaN.

If the D extension is implemented, FMINM.D and FMAXM.D instructions are analogously defined to operate on double-precision numbers.

If the Zfh extension is implemented, FMINM.H and FMAXM.H instructions are analogously defined to operate on half-precision numbers.

If the Q extension is implemented, FMINM.Q and FMAXM.Q instructions are analogously defined to operate on quad-precision numbers.

These instructions are encoded like their FMIN and FMAX counterparts, but with instruction bit 13 set to 1.

These instructions implement the IEEE 754-2019 minimum and maximum operations.

26.3. Round-to-Integer Instructions

The FROUND.S instruction rounds the single-precision floating-point number in floating-point register *rs1* to an integer, according to the rounding mode specified in the instruction's *rm* field. It then writes that integer, represented as a single-precision floating-point number, to floating-point register *rd*. Zero and infinite inputs are copied to *rd* unmodified. Signaling NaN inputs cause the invalid operation exception flag to be set; no other exception flags are set. FROUND.S is encoded like FCVT.S.D, but with *rs2*=4.

The FROUNDNX.S instruction is defined similarly, but it also sets the inexact exception flag if the input differs from the rounded result and is not NaN. FROUNDNX.S is encoded like FCVT.S.D, but with *rs2*=5.

If the D extension is implemented, FROUND.D and FROUNDNX.D instructions are analogously defined to operate on double-precision numbers. They are encoded like FCVT.D.S, but with *rs2*=4 and 5, respectively,

If the Zfh extension is implemented, FROUND.H and FROUNDNX.H instructions are analogously defined to operate on half-precision numbers. They are encoded like FCVT.H.S, but with *rs2*=4 and 5, respectively,

If the Q extension is implemented, FROUND.Q and FROUNDNX.Q instructions are analogously defined to operate on quad-precision numbers. They are encoded like FCVT.Q.S, but with *rs2*=4 and 5, respectively,

The FROUNDNX.fmt instructions implement the IEEE 754-2019 roundToIntegralExact operation, and the FROUND.fmt instructions implement the other operations in the roundToIntegral family.

26.4. Modular Convert-to-Integer Instruction

The FCVTMOD.W.D instruction is defined similarly to the FCVT.W.D instruction, with the following

differences. FCVTMOD.W.D always rounds towards zero. Bits 31:0 are taken from the rounded, unbounded two's complement result, then sign-extended to XLEN bits and written to integer register *rd*. $\pm\infty$ and NaN are converted to zero.

Floating-point exception flags are raised the same as they would be for FCVT.W.D with the same input operand.

This instruction is only provided if the D extension is implemented. It is encoded like FCVT.W.D, but with the rs2 field set to 8 and the *rm* field set to 1 (RTZ). Other *rm* values are *reserved*.

The assembly syntax requires the RTZ rounding mode to be explicitly specified, i.e., `fcvtmod.w.d rd, rs1, rtz`.

The FCVTMOD.W.D instruction was added principally to accelerate the processing of JavaScript Numbers. Numbers are double-precision values, but some operators implicitly truncate them to signed integers mod 2^{32}.

26.5. Move Instructions

For RV32 only, if the D extension is implemented, the FMVH.X.D instruction moves bits 63:32 of floating-point register *rs1* into integer register *rd*. It is encoded in the OP-FP major opcode with *funct3*=0, *rs2*=1, and *funct7*=1110001.

FMVH.X.D is used in conjunction with the existing FMV.X.W instruction to move a double-precision floating-point number to a pair of x-registers.

For RV32 only, if the D extension is implemented, the FMVP.D.X instruction moves a double-precision number from a pair of integer registers into a floating-point register. Integer registers *rs1* and *rs2* supply bits 31:0 and 63:32, respectively; the result is written to floating-point register *rd*. FMVP.D.X is encoded in the OP-FP major opcode with *funct3*=0 and *funct7*=1011001.

For RV64 only, if the Q extension is implemented, the FMVH.X.Q instruction moves bits 127:64 of floating-point register *rs1* into integer register *rd*. It is encoded in the OP-FP major opcode with *funct3*=0, *rs2*=1, and *funct7*=1110011.

FMVH.X.Q is used in conjunction with the existing FMV.X.D instruction to move a quad-precision floating-point number to a pair of x-registers.

For RV64 only, if the Q extension is implemented, the FMVP.Q.X instruction moves a double-precision number from a pair of integer registers into a floating-point register. Integer registers *rs1* and *rs2* supply bits 63:0 and 127:64, respectively; the result is written to floating-point register *rd*. FMVP.Q.X is encoded in the OP-FP major opcode with *funct3*=0 and *funct7*=1011011.

26.6. Comparison Instructions

The FLEQ.S and FLTQ.S instructions are defined like the FLE.S and FLT.S instructions, except that quiet NaN inputs do not cause the invalid operation exception flag to be set.

If the D extension is implemented, FLEQ.D and FLTQ.D instructions are analogously defined to operate on double-precision numbers.

If the Zfh extension is implemented, FLEQ.H and FLTQ.H instructions are analogously defined to operate on half-precision numbers.

If the Q extension is implemented, FLEQ.Q and FLTQ.Q instructions are analogously defined to operate on quad-precision numbers.

These instructions are encoded like their FLE and FLT counterparts, but with instruction bit 14 set to 1.

We do not expect analogous comparison instructions will be added to the vector ISA, since they can be reasonably efficiently emulated using masking.