Chapter 15

“Zfh” and “Zfhmin” Standard Extensions for Half-Precision Floating-Point, Version 1.0

This chapter describes the Zfh standard extension for 16-bit half-precision binary floating-point instructions compliant with the IEEE 754-2008 arithmetic standard. The Zfh extension depends on the single-precision floating-point extension, F. The NaN-boxing scheme described in Section 13.2 is extended to allow a half-precision value to be NaN-boxed inside a single-precision value (which may be recursively NaN-boxed inside a double- or quad-precision value when the D or Q extension is present).

This extension primarily provides instructions that consume half-precision operands and produce half-precision results. However, it is also common to compute on half-precision data using higher intermediate precision. Although this extension provides explicit conversion instructions that suffice to implement that pattern, future extensions might further accelerate such computation with additional instructions that implicitly widen their operands—e.g., half×half+single→single—or implicitly narrow their results—e.g., half+single→half.

15.1 Half-Precision Load and Store Instructions

New 16-bit variants of LOAD-FP and STORE-FP instructions are added, encoded with a new value for the funct3 width field.

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	width	rd	opcode
12	5	3	5	7
offset[11:0]	base	H	dest	LOAD-FP

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	width	imm[4:0]	opcode
7	5	5	3	5	7
offset[11:5]	src	base	H	offset[4:0]	STORE-FP

FLH and FSH are only guaranteed to execute atomically if the effective address is naturally aligned.

FLH and FSH do not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved. FLH NaN-boxes the result written to *rd*, whereas FSH ignores all but the lower 16 bits in *rs2*.

15.2 Half-Precision Computational Instructions

A new supported format is added to the format field of most instructions, as shown in Table 15.1.

fmt field	Mnemonic	Meaning
00	S	32-bit single-precision
01	D	64-bit double-precision
10	H	16-bit half-precision
11	Q	128-bit quad-precision

Table 15.1: Format field encoding.

The half-precision floating-point computational instructions are defined analogously to their single-precision counterparts, but operate on half-precision operands and produce half-precision results.

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FADD/FSUB	H	src2	src1	RM	dest	OP-FP
FMUL/FDIV	H	src2	src1	RM	dest	OP-FP
FMIN-MAX	H	src2	src1	MIN/MAX	dest	OP-FP
FSQRT	H	0	src	RM	dest	OP-FP

31 27	26 25	24 20	19 15	14 12	11 7	6 0
rs3	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
src3	H	src2	src1	RM	dest	F[N]MADD/F[N]MSUB

15.3 Half-Precision Convert and Move Instructions

New floating-point-to-integer and integer-to-floating-point conversion instructions are added. These instructions are defined analogously to the single-precision-to-integer and integer-to-single-precision conversion instructions. FCVT.W.H or FCVT.L.H converts a half-precision floating-point number to a signed 32-bit or 64-bit integer, respectively. FCVT.H.W or FCVT.H.L converts a 32-bit or 64-bit signed integer, respectively, into a half-precision floating-point number. FCVT.WU.H, FCVT.LU.H, FCVT.H.WU, and FCVT.H.LU variants convert to or from unsigned integer values. FCVT.L[U].H and FCVT.H.L[U] are RV64-only instructions.

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FCVT.*int*.H	H	W[U]/L[U]	src	RM	dest	OP-FP
FCVT.H.*int*	H	W[U]/L[U]	src	RM	dest	OP-FP

New floating-point-to-floating-point conversion instructions are added. These instructions are defined analogously to the double-precision floating-point-to-floating-point conversion instructions. FCVT.S.H or FCVT.H.S converts a half-precision floating-point number to a single-precision floating-point number, or vice-versa, respectively. If the D extension is present, FCVT.D.H or FCVT.H.D converts a half-precision floating-point number to a double-precision floating-point number, or vice-versa, respectively. If the Q extension is present, FCVT.Q.H or FCVT.H.Q converts a half-precision floating-point number to a quad-precision floating-point number, or vice-versa, respectively.

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FCVT.S.H	S	H	src	RM	dest	OP-FP
FCVT.H.S	H	S	src	RM	dest	OP-FP
FCVT.D.H	D	H	src	RM	dest	OP-FP
FCVT.H.D	H	D	src	RM	dest	OP-FP
FCVT.Q.H	Q	H	src	RM	dest	OP-FP
FCVT.H.Q	H	Q	src	RM	dest	OP-FP

Floating-point to floating-point sign-injection instructions, FSGNJ.H, FSGNJN.H, and FSGNJX.H are defined analogously to the single-precision sign-injection instruction.

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FSGNJ	H	src2	src1	J[N]/JX	dest	OP-FP

Instructions are provided to move bit patterns between the floating-point and integer registers. FMV.X.H moves the half-precision value in floating-point register *rs1* to a representation in IEEE 754-2008 standard encoding in integer register *rd*, filling the upper XLEN-16 bits with copies of the floating-point number's sign bit.

FMV.H.X moves the half-precision value encoded in IEEE 754-2008 standard encoding from the lower 16 bits of integer register *rs1* to the floating-point register *rd*, NaN-boxing the result.

FMV.X.H and FMV.H.X do not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved.

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FMV.X.H	H	0	src	000	dest	OP-FP
FMV.H.X	H	0	src	000	dest	OP-FP

15.4 Half-Precision Floating-Point Compare Instructions

The half-precision floating-point compare instructions are defined analogously to their single-precision counterparts, but operate on half-precision operands.

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FCMP	H	src2	src1	EQ/LT/LE	dest	OP-FP

15.5 Half-Precision Floating-Point Classify Instruction

The half-precision floating-point classify instruction, FCLASS.H, is defined analogously to its single-precision counterpart, but operates on half-precision operands.

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FCLASS	H	0	src	001	dest	OP-FP

15.6 "Zfhmin" Standard Extension for Minimal Half-Precision Floating-Point Support

This section describes the Zfhmin standard extension, which provides minimal support for 16-bit half-precision binary floating-point instructions. The Zfhmin extension is a subset of the Zfh extension, consisting only of data transfer and conversion instructions. Like Zfh, the Zfhmin extension depends on the single-precision floating-point extension, F. The expectation is that Zfhmin software primarily uses the half-precision format for storage, performing most computation in higher precision.

The Zfhmin extension includes the following instructions from the Zfh extension: FLH, FSH, FMV.X.H, FMV.H.X, FCVT.S.H, and FCVT.H.S. If the D extension is present, the FCVT.D.H and FCVT.H.D instructions are also included. If the Q extension is present, the FCVT.Q.H and FCVT.H.Q instructions are additionally included.

Zfhmin does not include the FSGNJ.H instruction, because it suffices to instead use the FSGNJ.S instruction to move half-precision values between floating-point registers.

Half-precision addition, subtraction, multiplication, division, and square-root operations can be faithfully emulated by converting the half-precision operands to single-precision, performing the operation using single-precision arithmetic, then converting back to half-precision [18]. Performing half-precision fused multiply-addition using this method incurs a 1-ulp error on some inputs for the RNE and RMM rounding modes.

Conversion from 8- or 16-bit integers to half-precision can be emulated by first converting to single-precision, then converting to half-precision. Conversion from 32-bit integer can be emulated by first converting to double-precision. If the D extension is not present and a 1-ulp error under RNE or RMM is tolerable, 32-bit integers can be first converted to single-precision instead. The same remark applies to conversions from 64-bit integers without the Q extension.