Chapter 22

"Zfinx", "Zdinx", "Zhinx", "Zhinxmin": Standard Extensions for Floating-Point in Integer Registers, Version 0.41

This chapter defines the "Zfinx" extension (pronounced "z-f-in-x") that provides instructions similar to those in the standard floating-point F extension for single-precision floating-point instructions but which operate on the `x` registers instead of the `f` registers. This chapter also defines the "Zdinx", "Zhinx", and "Zhinxmin" extensions that provide similar instructions for other floating-point precisions.

The F extension uses separate `f` *registers for floating-point computation, to reduce register pressure and simplify the provision of register-file ports for wide superscalars. However, the additional 128 B of architectural state increases the minimal implementation cost. By eliminating the* `f` *registers, the Zfinx extension substantially reduces the cost of simple RISC-V implementations with floating-point instruction-set support. Zfinx also reduces context-switch cost.*

In general, software that assumes the presence of the F extension is incompatible with software that assumes the presence of the Zfinx extension, and vice versa.

The Zfinx extension adds all of the instructions that the F extension adds, *except* for the transfer instructions FLW, FSW, FMV.W.X, FMV.X.W, C.FLW[SP], and C.FSW[SP].

Zfinx software uses integer loads and stores to transfer floating-point values from and to memory. Transfers between registers use either integer arithmetic or floating-point sign-injection instructions.

The Zfinx variants of these F-extension instructions have the same semantics, except that whenever such an instruction would have accessed an `f` register, it instead accesses the `x` register with the same number.

22.1 Processing of Narrower Values

Floating-point operands of width $w <$ XLEN bits occupy bits w-1:0 of an `x` register. Floating-point operations on w-bit operands ignore operand bits XLEN-1:w.

Floating-point operations that produce $w <$ XLEN-bit results fill bits XLEN-1:w with copies of bit w-1 (the sign bit).

The NaN-boxing scheme employed in the `f` *registers was designed to efficiently support recoded floating-point formats. Recoding is less practical for Zfinx, though, since the same registers hold both floating-point and integer operands. Hence, the need for NaN boxing is diminished.*

Sign-extending 32-bit floating-point numbers when held in RV64 `x` *registers matches the existing RV64 calling conventions, which require all 32-bit types to be sign-extended when passed or returned in* `x` *registers. To keep the architecture more regular, we extend this pattern to 16-bit floating-point numbers in both RV32 and RV64.*

22.2 Zdinx

The Zdinx extension provides analogous double-precision floating-point instructions. The Zdinx extension requires the Zfinx extension.

The Zdinx extension adds all of the instructions that the D extension adds, *except* for the transfer instructions FLD, FSD, FMV.D.X, FMV.X.D, C.FLD[SP], and C.FSD[SP].

The Zdinx variants of these D-extension instructions have the same semantics, except that whenever such an instruction would have accessed an `f` register, it instead accesses the `x` register with the same number.

22.3 Processing of Wider Values

Double-precision operands in RV32Zdinx are held in aligned `x`-register pairs, i.e., register numbers must be even. Use of misaligned (odd-numbered) registers for double-width floating-point operands is *reserved*.

Regardless of endianness, the lower-numbered register holds the low-order bits, and the higher-numbered register holds the high-order bits: e.g., bits 31:0 of a double-precision operand in RV32Zdinx might be held in register `x14`, with bits 63:32 of that operand held in `x15`.

When a double-width floating-point result is written to `x0`, the entire write takes no effect: e.g., for RV32Zdinx, writing a double-precision result to `x0` does not cause `x1` to be written.

When `x0` is used as a double-width floating-point operand, the entire operand is zero—i.e., `x1` is not accessed.

Load-pair and store-pair instructions are not provided, so transferring double-precision operands in RV32Zdinx from or to memory requires two loads or stores. Register moves need only a single FSGNJ.D instruction, however.

22.4 Zhinx

The Zhinx extension provides analogous half-precision floating-point instructions. The Zhinx extension requires the Zfinx extension.

The Zhinx extension adds all of the instructions that the Zfh extension adds, *except* for the transfer instructions FLH, FSH, FMV.H.X, and FMV.X.H.

The Zhinx variants of these Zfh-extension instructions have the same semantics, except that whenever such an instruction would have accessed an `f` register, it instead accesses the `x` register with the same number.

22.5 Zhinxmin

The Zhinxmin extension provides minimal support for 16-bit half-precision floating-point instructions that operate on the `x` registers. The Zhinxmin extension requires the Zfinx extension.

The Zhinxmin extension includes the following instructions from the Zhinx extension: FCVT.S.H and FCVT.H.S. If the Zdinx extension is present, the FCVT.D.H and FCVT.H.D instructions are also included.

In the future, an RV64Zqinx quad-precision extension could be defined analogously to RV32Zdinx. An RV32Zqinx extension could also be defined but would require quad-register groups.

22.6 Privileged Architecture Implications

In the standard privileged architecture defined in Volume II, the `mstatus` field FS is hardwired to 0 if the Zfinx extension is implemented, and FS no longer affects the trapping behavior of floating-point instructions or `fcsr` accesses.

The `misa` bits F, D, and Q are hardwired to 0 when the Zfinx extension is implemented.

A future discoverability mechanism might be used to probe the existence of the Zfinx, Zhinx, and Zdinx extensions.