Chapter 24

"Zfinx" Standard Extension for Floating-Point in Integer Registers, Version 0.41

This chapter defines the "Zfinx" extension (pronounced "z-f-in-x"), which provides instructions similar to those in the standard floating-point extensions, but that operate on the `x` registers instead of the `f` registers.

The baseline F extension uses separate `f` *registers for floating-point computation. This design reduces register pressure and simplifies 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 effort.*

Unlike most RISC-V extensions, the addition of Zfinx is not backwards compatible: software that uses floating-point instructions but assumes the absence of the Zfinx extension will not, in general, execute correctly on implementations with the Zfinx extension.

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.

24.1 NaN Boxing 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 of the result with ones.

To avoid the need for dedicated floating-point load instructions that fill the MSBs of an `x` *register with ones, we abandon the usual NaN-boxing requirement for floating-point operands. We retain the NaN-boxing of results to keep Zfinx as similar as possible and to ease debugging.*

24.2 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.

24.3 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.

24.4 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.

In 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.

24.5 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, Zdinx, and Zqinx extensions.