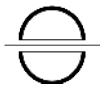
11.6 "N" Standard Extension for User-Level Interrupts

The N extension allows interrupts and exceptions that occur in user-level programs to transfer control directly to a user-level trap handler without invoking the outer execution environment. User-level interrupts are mainly intended to support secure embedded systems with only M-mode and U-mode present (Chapter 10). However, they can also support user-level trap handling in systems running Unix-like operating systems. When used in a Unix environment, conventional signal handling would likely remain, but user-level interrupts could be used as a building block for future extensions that generate user-level events such as garbage collection barriers, integer overflow, and floating-point traps.



11.7 "P" Standard Extension for Packed-SIMD Instructions

The P extension subdivides the existing architectural registers to provide data-parallel computation on smaller data types. Packed-SIMD designs represent a reasonable design point when reusing existing wide datapath resources. However, if significant additional resources are to be devoted to data-parallel execution, Chapter 8 shows that designs for vector architectures are a better choice, and architects should use the RVV extension.

11.8 "Q" Standard Extension for Quad-Precision Floating-Point

The Q extension adds 128-bit quad-precision binary floating-point instructions compliant with the IEEE 754-2008 arithmetic standard. The floating-point registers are now extended to hold either a single, double, or quad-precision floating-point value. The quad-precision binary floating-point extension requires RV64IFD.

11.9 Concluding Remarks

> *Simplify, simplify.*
>
> —Henry David Thoreau, an eminent writer of the 19th century, 1854

Having an open, standards-like committee approach to expanding RISC-V hopefully will mean that the feedback and debate will occur *before* the instructions are finalized rather than afterwards, when it's too late to change. In the ideal case, a few members will implement the proposal before it is ratified, which FPGAs make much easier to do. Proposing instruction extensions via the RISC-V Foundation committees will also be a fair amount of work, which will keep the rate of change slow, unlike what happened to x86-32 (see Figure 1.2 on page 3 in Chapter 1). Don't forget that everything in this chapter will be optional, despite how many extensions are adopted.

Our hope is that RISC-V can evolve with technological demands while maintaining its reputation as a simple, efficient ISA. If it succeeds, RISC-V will we be a significant break from the incremental ISAs of the past.

A RISC-V Instruction Listings

Coco Chanel (1883-1971) Founder of the Chanel fashion brand, her pursuit of expensive simplicity shaped 20th-century fashion.

Simplicity is the keynote of all true elegance.

—Coco Chanel, 1923

This appendix lists all the instructions for RV32/64I, all the extensions covered in this book (RVM, RVA, RVF, RVD, RVC, and RVV), and all the pseudoinstructions. Each entry has the instruction name, operands, a register-transfer level definition, instruction format type, English description, compressed versions (if any), and a figure showing the actual layout with opcodes. We think you have everything you need to understand all the instructions in these compact summaries. However, if you want even more detail, refer to the official RISC-V specifications [Waterman and Asanović 2017].

To help readers find the desired instruction in this appendix, the header of the left (even) page contains the first instruction from the top of that page and the header on the right (odd) page contains the last instruction from at the bottom of that page. The format is similar to the headers of dictionaries, which helps you search for the page that your word is on. For example, the header of the next *even* page shows **AMOADD.W**, the first instruction on the page, and the header of the following odd page shows **AMOMINU.D**, the last instruction on that page. These are the two pages where you would find any of these 10 instructions: `amoadd.w`, `amoand.d`, `amoand.w`, `amomax.d`, `amomax.w`, `amomaxu.d`, `amomaxu.w`, `amomin.d`, `amomin.w`, and `amominu.d`.