The Linley Group

MICROPROCESSOR report

Insightful Analysis of Processor Technology

RISC-V Offers Simple, Modular ISA

New CPU Instruction Set Is Open and Extensible

By David Kanter (March 28, 2016)

RISC-V is a new general-purpose instruction-set architecture (ISA) that's BSD licensed, extensible, and royalty free. It's clean and modular with a 32-, 64-, or 128-bit integer base and various optional extensions (e.g., floating point). RISC-V is easier to implement than some alternatives—minimal RISC-V cores are roughly half the size of equivalent ARM cores—and the ISA has already gathered some support from the semiconductor industry.

The ambitious goal of RISC-V (pronounced "risk five") is to create a "universal" instruction set that is free and open to all users while providing the broad software support of successful commercial ISAs (e.g., x86 and ARM). Consider a modern smartphone processor, which typically comprises a dozen or more cores with different software stacks (e.g., an ARM application CPU, a GPU, three to five different DSPs, and a power-management core). In theory, they could all use variants of a single ISA and in many cases reuse both hardware and software. RISC-V aims to offer a practical option that could unify many of these cores under one roof.

Driving its generic nature is the desire to serve all markets, including microcontrollers as well as image, graphics, and server processors. Thus, the ISA must be consistent across microarchitectures, from an in-order scalar design to a heavily out-of-order design. Similarly, it's intended to be suitable for nearly any implementation, from FPGAs to synthesized macros to fully custom layouts. Owing to the emerging interest in accelerators (see *MPR 3/7/16,* "Accelerating Machine Learning"), extensibility is an essential part of universality.

Broad software support and extensibility are a tricky combination, as software developers want a consistent target. For example, vector instructions are optional in ARMv7 and unwanted for many low-end implementations. But when Tegra 2 omitted the Neon vector extensions, it created substantial consternation in the Android ecosystem.

To address this issue, RISC-V defines a guaranteed base integer instruction set of 32, 64, or 128 bits, a family of optional and predefined extensions, and a mechanism for creating variable-length extensions. The base ISA is cleanly architected to simplify implementation. For example, the instruction encoding is highly regular, and the memory model is straightforward and lacks complicated memory instructions. The benefit is that minimal RISC-V cores are much smaller than similar ARM or x86 cores, although the difference is not noticeable for more powerful cores (e.g., superscalar, out-of-order). To enable extensions, portions of the instruction encoding space have already been set aside for future use.

Academic Heritage, Commercial Aspirations

Krste Asanović, Andrew Waterman, and Yunsup Lee developed the RISC-V architecture at UC Berkeley, with input from David Patterson, in response to limitations of existing ISAs (see *MPR 8/18/14,* "The Case for Open Instruction Sets"). Although x86 and ARM are widely available and supported, they are complex, and the licensing model is difficult for experimental and academic use. For example, modifying the RTL of ARM-supplied cores is impossible, and even obtaining an x86 core is exceptionally unlikely. Moreover, the instruction sets are overly complicated for research, which focuses on exploring new concepts rather than on compatibility.

Created in 2015, the RISC-V Foundation manages the standard, creates compliance tests, and organizes the community. It comprises industry leaders such as Google, Mellanox, and Oracle as well as academic partners such as the Indian Institutes of Technology. Several open-source

Base ISA	Instructions	Description
RV32I	47	32-bit address space and integer instructions
RV32E	47	Subset of RV32I, restricted to 16 registers
RV64I	59	64-bit address space and integer instructions, along with several 32-bit integer instructions
RV128I	71	128-bit address space and integer instructions, along with several 64- and 32-bit instructions
Extension	**Instructions**	**Description**
M	8	Integer multiply and divide
A	11	Atomic memory operations, load-reserve/store conditional
F	26	Single-precision (32 bit) floating point
D	26	Double-precision (64 bit) floating point; requires F extension
Q	26	Quad-precision (128 bit) floating point; requires F and D extensions
C	46	Compressed integer instructions; reduces size to 16 bits

Table 1. RISC-V base instruction sets and extensions. Currently, RISC-V offers three base ISAs with several standard extensions. (Source: RISC-V)

processor cores are freely available online, including the scalar five-stage "Rocket," taking advantage of the permissive and royalty-free BSD-style license.

Modular ISA Offers Many Options

Generally, RISC-V is a three-operand load-store architecture that heavily emphasizes cleanliness and simplicity. As Table 1 shows, it effectively has three base instruction sets and six extensions. The base ISA plus MADF extensions form a general-purpose ISA (sometimes known as the G extensions) that can readily handle scalar integer or floating-point code.

The user programming model for RV32I is sparse; it comprises the program counter (PC) and 32 integer registers (x1–x31 and x0, which is hard wired to 0). It lacks explicit return address registers, but x1 holds the return address by convention.

RV32I has 47 integer instructions for loads, stores, control transfer, arithmetic, and other functions (e.g., system calls, fences, and certain counters). In addition, it has 14 privileged instructions for changing privilege level, traps, and interrupts, as well as for accessing control and status registers (CSRs) from a 12-bit address space.

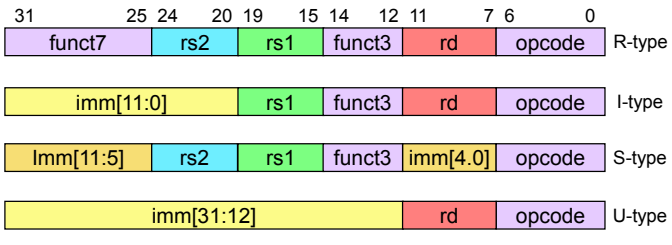

Figure 1. Basic RISC-V instruction formats. The basic RISC-V instructions are 4 bytes long and come in four different formats. These formats were specifically chosen to simplify decoding—for example, to keep register specifiers in the same fields. (Source: RISC-V)

To simplify decoding, the instruction encoding is highly regular. Basic instructions are 32 bits long, as Figure 1 illustrates, and are naturally aligned. Four different instruction formats accommodate various combinations of immediates, operands, and other specifiers. The architects took particular care to ensure consistency between formats. For example, the opcode and operands are in fixed locations, simplifying the decode process. As a result, immediates are split into two fields. Although this approach adds complexity, selecting operands is more likely to be on the critical path in practice. Immediates are sign extended, and the last bit always holds the sign.

Computational instructions operate on 32 bits and are typically in two-operand I-type format for register-immediate and the three-operand R-type format for register-register. RISC-V lacks support for integer overflow checks. The basic integer instructions are shifts, add, subtract, comparisons, and Boolean operations. The load-upper-immediate operation forms full 32-bit immediates, and an instruction to add the upper immediate to the PC is also available. The ISA specifically excludes multiplication and division by moving them into an optional extension.

Control transfer includes a jump and conditional branches. Although the base instructions are 4 bytes, RISC-V supports variable-length instructions in 2-byte multiples. Jumps employ the U-type format, where the immediate specifies a 2-byte aligned offset that's PC relative and has a ±1MB range. A two-instruction sequence accesses the full 32-bit address space. Branches use the S-type format to compare two registers and can encode a ±4KB PC-relative offset.

Simple Memory Model

The RISC-V address space is byte addressed and little-endian. Even though most other ISAs such as x86 and ARM have several potentially complex addressing modes, RISC-V only uses base+offset addressing with a 12-bit immediate to simplify load-store units. Load instructions use the I-type format; stores use S type. Loads and stores operate on byte, half-word (16 bit), and word (32 bit) data and have options to zero- or sign-extend smaller values.

All loads and stores enable misaligned accesses, but the handling is implementation dependent. Although high-performance processors with SIMD will inevitably favor fast misaligned accesses, cost-sensitive designs may simply trap and emulate slowly.

Loads and stores in a single thread are observed sequentially, but there is no explicit ordering between threads. The base ISA includes a fence instruction to enforce ordering between threads and I/O, and more-advanced synchronization primitives are available through

the optional extensions. RISC-V omits native consistency enforcement between instructions and data (e.g., for self-modifying code). Instead, a variant of the fence instruction guarantees that any subsequent instruction fetches will see all previous data stores in the same thread.

The user-level system instructions primarily handle calls, breakpoints, and counter accesses. The counters are often 64 bits long; they are accessed as two separate counters and then concatenated. The privileged instructions primarily access CSRs, trap to different execution layers, and pause for interrupts. Last, a supervisor fence instruction synchronizes page-table accesses following a page-table modification.

Expanding the Base

All other base RISC-V ISAs are variants of RV32I. The RV32E base ISA targets very low-end implementations where the register file occupies a significant portion of the die area. It reduces the user-visible state to the PC and x0–x15. Otherwise, RV32E and RV32I are identical, and instructions are still 32 bits long; the highest bit for the register numbers is set to 0.

RV64I and RV128I are straightforward 64- and 128-bit extensions that enlarge the address space and extend the 32 integer registers to the appropriate length. Most integer instructions simply operate on the appropriate data size (e.g., 64 or 128 bits). These extensions introduce new instructions that operate on the lower 32 or 64 bits while sign extending into the upper bits so that other instructions are unchanged on smaller values.

The base integer instruction set is simple and streamlined, allowing efficient minimal implementations. To complement this simplicity, RISC-V includes a number of optional extensions, listed in Table 1.

The multiply-divide extension (known as M) adds four instructions for multiplication, two for division, and two more for handling remainders. The size of the multiply and divide instructions is determined by the base integer ISA. Note that division by zero does not cause an exception or trap, keeping with the tradition of simplicity; a basic highly predictable branch can detect whether the divisor is zero.

The A extension comprises 11 synchronization instructions that enable release consistency and basic atomic operations. All include 2 bits that specify acquire and release semantics between threads; using both bits enforces sequential consistency. The A instructions fall into two groups. The first is a load reserved (LR) and store conditional (SC) for atomic memory operations on a single address. For example, LR/SC enables construction of a compare-and-swap function. The second group performs atomic integer read-modify-write operations such as ADD, OR, XOR, SWAP, MAX, and MIN. They are designed to efficiently support C11/C++11 and also facilitate sequentially consistent loads and stores.

Floating Point in Three Sizes

RISC-V includes three floating-point extensions: single precision (F), double precision (D), and quadruple precision (Q). The F extension is a prerequisite for D, which is in turn a prerequisite for Q. It introduces 32 registers (f0–f31) that are 32 bits wide, as well as a floating-point CSR (FPCSR), which has 5 bits for flags that track exceptions including invalid operations (NV), division by zero (DZ), overflow (OF), underflow (UF), and inexact (NX). Exceptions do not generate traps, and must be checked using a privileged branch. Three bits in the FPCSR specify a dynamic rounding mode. Instructions can either use a static rounding mode in the instruction encoding or select the dynamic mode indicated in the FPCSR.

FP load-and-store instructions use the same base+offset addressing as integer instructions. RISC-V computational instructions include FP add, subtract, multiply, divide, square root, and fused multiply-add (FMA). The FMA family uses four operands and a special format, shown in Figure 2, to encode the extra register operand. These instructions can negate the output of the multiplier or the input of the final adder.

The F extension also includes a family of instructions for converting and moving between integer and FP registers. FP comparisons look at two FP registers and store a result in an integer register (e.g., for branching). The FP classify instruction analyzes the data in a single FP register (e.g., infinite, negative, negative zero, positive zero, or NaN) and writes the result to an integer register. Lastly, instructions that manipulate the FPSCR are available for checking flags, rounding, and so on.

The D and Q extensions are fairly simple and increase the length of f0–f31 to 64 and 128 bits, respectively. Nearly all higher-precision instructions are analogous to the F equivalents. Conversion between FP formats (e.g., single and double) is supported. Loads and stores are defined so that if a register holding a single-precision value is written to memory using a high-precision store (e.g., FSD) and then read using a high-precision load (e.g., FLD), the original value is recreated.

The floating-point extensions are only defined for scalar quantities. The RISC-V Foundation is defining a separate vector extension that uses Cray-style vectors as opposed to the SIMD techniques that ARM, Power, and x86 employ.

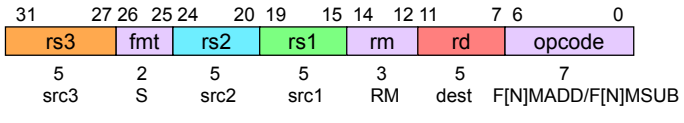

Figure 2. R32F fused-multiply-add (FMA) instruction format. The FMA family uses a unique format to encode four register operands and control rounding and negation. The format field describes whether the instruction is single, double, or quadruple precision. (Source: RISC-V)

For More Information

The RISC-V Foundation web site provides a software-tool overview at *riscv.org/software-tools*. Yunsup Lee presented a 28nm RISC-V processor with integrated voltage regulators at Hot Chips 27 (*www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.25-Tuesday-Epub/HC27.25.70-Processors-Epub/HC27.25.731-RISC-V-Lee-UCB-V4.with-backup.pdf*). The most recent version of the ISA specification can be found at *riscv.org/specifications/*; a synthesizable and configurable out-of-order core is also available (*aspire.eecs.berkeley.edu/publication/the-berkeley-out-of-order-machine-boom-an-industry-competitive-synthesizable-parameterized-risc-v-processor*).

Compression Cuts Code Size and Cost

The last integer extension (C) adds no other functions but instead encodes instructions to save storage and reduce instruction footprint. This compression extension is available for the base integer ISA as well as floating-point loads and stores. It creates 16-bit compressed instructions that are transparent to programmers and compilers thanks to a 1:1 mapping between compressed and base instructions, which can be freely intermingled. The C extension has some restrictions to reduce instruction length. Compressed instructions can use a two- or three-operand format on the most common eight registers but only a two-operand format on all 32 integer registers.

The code-size reduction for RV32C is similar to that of ARM's Thumb-2 ISA, as Figure 3 illustrates. Unlike Thumb, however, RV32C does not require separate decoders. The RISC-V team estimates that a compressed decoder requires only 700 gates. RV64C is a substantial improvement over existing 64-bit ISAs, which lack compressed-instruction support. The developers estimate that the Linux kernel text occupies 1.3MB using standard RISC-V instructions, 0.84MB using compressed instructions, and 0.65MB when compressed using bzip2. Because the last case precludes simple decoding and only reduces the size by 23%, they conclude that the C extension is very close to optimal complexity.

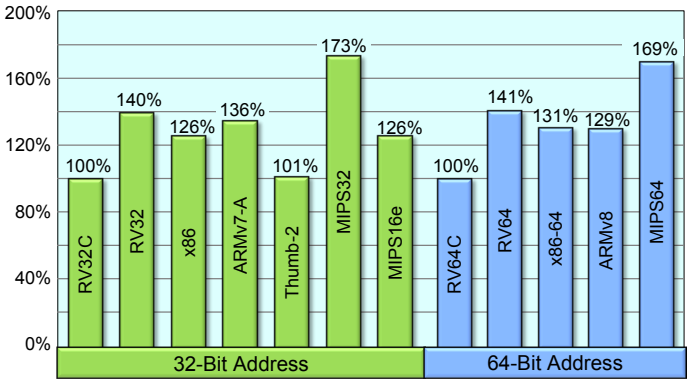

Figure 3. Code size for different ISAs. The footprint of the SPECint2006 suite for the compressed 32-bit RISC-V variants is similar to that of Thumb-2, whereas the 64-bit compressed RISC-V is denser than commercial options. (Source: RISC-V)

Although the RISC-V base ISA and the prespecified extensions are fixed length, some extensions are not (e.g., compressed instructions). The overall ISA envisions variable-length encoding with formats already specified for 2 to 22 bytes.

The RISC-V Foundation controls the standard extensions to ensure consistency throughout the community. Portions of the instruction-encoding space are guaranteed to be unused by future standard extensions, essentially carving out space for custom extensions. Such extensions can create new registers, which the OS manages through a coprocessor model (e.g., explicit save and restore on context switch).

Privilege Comes After Users

The RISC-V ISA actually comprises four privilege layers: user (U), supervisor (S), hypervisor (H), and machine (M). Although user mode has been complete since 2014, the architecture's more privileged layers remain unfinished. Between each layer is a binary interface (e.g., the ABI lies between the user application and the OS).

At the lowest level, the machine mode is mandatory, and any software running at this level has complete access to the hardware. Conceptually, it's the equivalent of Alpha PALcode or microcode, and it includes CSRs such as the *misa* register (which is similar to the x86 CPUID). All other modes are optional, and the smallest implementations (e.g., for low-cost embedded designs) may only support M mode with software running directly on the hardware. The basic architectural state for M mode is 128 bits along with another 256 bits for timers and performance counters.

The hypervisor mode is intended for virtualized systems, but few specification details are available. In part, a cleanly defined M mode obviates some use cases for a hypervisor, but it also reflects the community's initial focus on embedded design.

Traditional Unix-style operating systems are at the supervisor privilege level. The virtual-memory model for RV32 is demand paged with a two-level page table and support for 4KB and 4MB pages. The minimal 64-bit virtual-memory system uses a three-level table with 4KB, 2MB, and 1GB pages (similar to x86), and it can map 39 bits of virtual address space; each additional level increases the virtual addressing by 9 bits up to six levels for 64 bits. Generally, TLBs are expected to handle address translation in hardware, but a software page-table walker in machine mode is possible. The RISC-V supervisor binary interface (SBI) provides a platform-level abstraction (e.g., to query devices and send interrupts) and is one of the main focus areas. An SBI draft should arrive with the next privileged-ISA revision.

To reduce the cost of minimal implementations, only the M privilege level is required. But just four privilege-level combinations are supported, as Figure 4 shows. A basic embedded system might use M mode to secure several different applications written in C and running in U mode. Similarly, a hypervisor could operate in M mode by partitioning the physical address space. Systems with M, S, and U modes can run Linux and correspond to a fairly normal nonvirtualized environment, whereas adding the H mode would be suitable for datacenter servers.

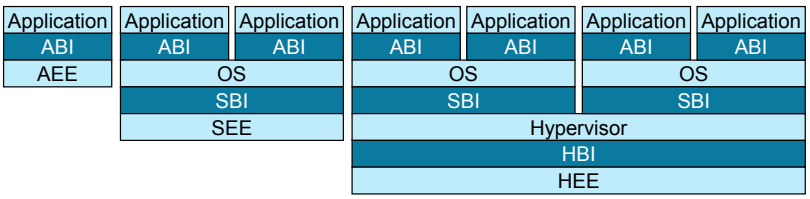

Figure 4. Supported privilege combinations. M mode is mandatory; all other privilege levels are optional. (Source: RISC-V)

An Open ISA With RISCs and Advantages

The RISC-V ISA is incomplete, but the user-mode specification indicates that it's quite promising. The clean architecture is easier to implement than ARM, specifically owing to simpler decoding. ARMv8 currently requires decoding of three different ISAs: ARMv8, ARMv7, and Thumb. In addition, RISC-V offers simpler addressing modes than ARM and omits complex instructions such as load/store multiple. This distinction is certainly noticeable; the Rocket core is roughly half the size of a similar ARM core.

But the choice of ISA has a minimal effect on a larger core (e.g., three-issue out-of-order CPU), especially once caches are considered. For example, Intel's x86 CPUs, which require complex decoders, are similar in overall size to RISC cores of comparable performance. Even for scalar microarchitectures, the die-area benefits of RISC-V are less significant when compared to a simpler RISC ISA such as ARC or MIPS.

RISC-V's greatest technical promise is the flexibility of custom extensions combined with a general-purpose architecture. Potential RISC-V extensions include Cray-style vector extensions, transactional memory, and bit manipulation. Both ARC (now Synopsys) and Tensilica (now Cadence) have offered similar extension capability for years. Custom extensions can deliver huge performance gains, but they require custom tool chains, and they limit software portability.

The RISC-V software ecosystem is in the early stages. A Linux 4.1 branch has been ported, and a full embedded Linux distribution known as Yocto is also available. The RISC-V tool chain includes the GCC compiler, LLVM and Clang, GDB, a verification suite, and several simulators. But the most important milestone is completing the supervisor interface (SBI), which will standardize the environment for Linux and BSD operating systems and thereby deliver the general-purpose aspect. Mainstream-OS support will broaden the appeal of RISC-V and enable many more applications.

On the basis of early developments, the RISC-V ISA appears promising. It offers all the basic RISC features with a few twists that simplify the implementation, thereby reducing die area and potentially power consumption. Compared with today's two most popular ISAs, RISC-V offers considerable area savings, particularly for low-end designs, and the ability to add custom extensions. Compared with ISAs such as ARC and Tensilica, RISC-V offers fewer technical advantages, but its open-source business model will continue to drive interest in the new ISA. ♦