

Instruction Sets Should Be Free: The Case For RISC-V

Krste Asanović
David A. Patterson

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-146

<http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html>

August 6, 2014



Copyright © 2014, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Custom systems-on-a-chip (SoCs), where the processors and caches are a small part of the chip, are becoming ubiquitous; it is rare today to find an electronics product at any scale that does not include an on-chip processor. Thus, many more companies are designing chips that include processors than in the past. Given that the industry has been revolutionized by open standards and open source software—with networking protocols like TCP/IP and operating systems (OS) like Linux—why is one of the most important interfaces proprietary?

The Case for a Free, Open ISA

While instruction set architectures (ISAs) may be proprietary for historical or business reasons, there is no good *technical* reason for the lack of free, open ISAs:

- *It's not an error of omission.* Companies with successful ISAs like ARM, IBM, and Intel have patents on quirks of their ISAs, which prevent others from using them without licenses.¹ Negotiations take 6-24 months and they can cost \$1M-\$10M, which rules out academia and others with small volumes.² An ARM license doesn't even let you design an ARM core; you just get to use *their* designs. (Only ≈15 big companies have licenses that allow new ARM cores.) Even “OpenPOWER” is an oxymoron; you must pay IBM to use its ISA. While business sound, licenses stifle competition and innovation by stopping many from designing and sharing their ISA-compatible cores.
- *Nor is it because the companies do most of the software development.* Despite the value of the software ecosystems that grow around popular ISAs, outsiders build almost all of the software for them.
- *Neither do companies exclusively have the experience needed to design a competent ISA.* While it's a lot of work, many today can design ISAs.
- *Nor are the most popular ISAs wonderful ISAs.* 80x86 and ARM aren't considered ISA exemplars.
- *Neither can only companies verify ISA compatibility.* Open organizations developed mechanisms to ensure compatibility with hardware standards long ago, such as IEEE 754 floating point, Ethernet, and PCIe. If not, open IT standards would not be so popular.
- *Finally, proprietary ISAs are not guaranteed to last.* If a company dies, it takes its ISAs with it; DEC's demise also terminated the Alpha and VAX ISAs.

Note that an ISA is really an interface specification, and not an implementation. There are three types of implementations of an ISA:

1. Private closed source, analogous to Apple iOS.
2. Licensed open source, like Wind River VxWorks.
3. Free, open source that users can change and share, like Linux.

Proprietary ISAs in practice allow the first two types of cores, but you need a free, open ISA to enable all three.

We conclude that the industry would benefit from viable freely open ISAs just as it has benefited from free

open source software. For example, it would enable a *real free open market of processor designs*, which patents on ISA quirks prevent. This could lead to:

- *Greater innovation via free-market competition* from many more designers, including open vs. proprietary implementations of the ISA.
- *Shared open core designs*, which would mean shorter time to market, lower cost from reuse, fewer errors given many more eyeballs³, and transparency that would make it hard, for example, for government agencies to add secret trap doors.
- *Processors becoming affordable for more devices*, which helps expand the Internet of Things (IoTs), which could cost as little as \$1.

The Case for RISC as the Free, Open ISA Style

For an ISA to be embraced by an open-source community, we believe it needs a proven commercial record. The first question, then, is which style of ISA has a history of success. There hasn't been a successful *stack* ISA in 30 years. Except for parts of the DSP market, *VLIWs* have failed: Multiflow went belly up and Itanium was a bust despite billions of dollars invested by HP and Intel. It's been decades since any new *CISC* ISA has been successful. The surviving *CISCs* translate from complex ISAs to easier-to-execute ISAs, which makes great sense for executing a valuable legacy code-base. A new ISA by definition won't have any legacy code, so the extra hardware cost and energy cost of translation are hard to justify; why not just use an easy-to-execute ISA in the first place? *RISC-style* load-store ISAs date back at least 50 years to Seymour Cray's CDC 6600. While the 80x86 won the PC wars, RISC dominates the tablets and smart phones of the PostPC Era; in 2013 more than 10B ARMs were shipped, as compared to 0.3B 80x86s. Repeating what we said in 1980⁴, we propose that RISC is the best choice for an (free, open) ISA.

Moreover, a new RISC ISA can be better than its predecessors by learning from their mistakes:

- *Leaving out too much:* No load/store byte or load/store half word in the initial Alpha ISA, and no floating-point load/store double in MIPS I.
- *Including too much:* The shift option for ARM instructions and register windows in SPARC.
- *Allowing current microarchitectural designs to affect the ISA:* Delayed branch in MIPS and SPARC, and floating-point trap barriers in Alpha.

To match embedded market needs, RISCs even offered solutions to the code size issue: ARM Thumb and MIPS16 added 16-bit formats to offer code smaller than 80x86. Thus, we believe there is widespread agreement on the general outline of a good RISC ISA.

The Case for Using an Existing RISC Free, Open ISA

The good news is that there are already three RISC free, open ISAs⁵:

- *SPARC V8* - To its credit, Sun Microsystems made SPARC V8 an IEEE standard in 1994.

- *OpenRISC* - This GNU open-source effort started in 2000, with the 64-bit ISA being completed in 2011.
- *RISC-V* - In 2010, partly inspired by ARM's IP restrictions together with the lack of 64-bit addresses and overall baroque-ness of ARM v7, we and our grad students Andrew Waterman and Yunsup Lee developed RISC-V⁶ (pronounced "RISC 5") for our research and classes, and made it BSD open source.

As it takes years to get the details right—the gestation period for OpenRISC was 11 years and RISC-V was 4 years—it seems wiser to start with an existing ISA than to form committees and start from scratch. RISC ISAs tend to be similar, so any one might be a good candidate.

Given ISAs can live for decades, let's first project the future IT landscape to see what features might be important to help rank the choices. Three platforms will likely dominate: 1) IoTs – billions of cheap devices with IP addresses and Internet access; 2) Personal mobile devices, such as smart phones and tablets today; 3) Warehouse-Scale Computers (WSCs). While we could have distinct ISAs for each platform, life would be simpler if we could use a single ISA design everywhere.

This landscape suggests four key requirements:

1. *Base-plus-extension ISA*.⁷ To improve efficiency and to reduce costs, SoCs add custom application-specific accelerators. To match the needs of SoCs while maintaining a stable software base, a free, open ISA should have: i) a small core set of instructions that compilers and OS's can depend upon; ii) standard but optional extensions for common ISA additions to help customize the SoC to the application; and iii) space for entirely new opcodes to invoke the application-specific accelerators.
2. *Compact instruction set encoding*. Smaller code is desirable given the cost sensitivity of IoTs and the resulting desire for smaller memory.
3. *Quadruple-precision (QP) as well as SP and DP floating point*. Some applications running in WSCs today process such large data sets that they already rely on software libraries for QP arithmetic.
4. *128-bit addressing as well as 32-bit and 64-bit*. The limited memory size of IoTs means 32-bit addressing will be important for decades to come, while 64-bit addressing is the de facto standard in anything larger. Although the WSC industry won't need 2^{128} bytes, it's plausible that within a decade WSCs might need more than 2^{64} bytes (16 exabytes) to address all of their solid-state non-volatile storage. As address size is the one ISA mistake from which it is hard to recover⁸, it's wise to plan for bigger addresses now.

The table below scores the 3 free open ISAs using these 4 criteria, plus a listing of critical compiler and OS ports.

ISA	Address			Software						
	Base+Ext	Compact Code	Quad FP	32-bit	64-bit	128-bit	GCC	LLVM	Linux	QEMU
SPARC V8			✓	✓			✓	✓	✓	✓
OpenRISC				✓	✓		✓	✓	✓	✓
RISC-V	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

The Case for RISC-V as the RISC Free, Open ISA

Our community should rally around a single ISA to test whether a free, open ISA can work. Only RISC-V meets all four requirements. RISC-V is also 10 to 20 years younger, so we had the chance to learn from and fix the mistakes of previous RISC ISAs—e.g., SPARC and OpenRISC have delayed branches—which is why RISC-V is so simple and clean (see Tables 4 and 5 and www.riscv.org). In addition to the other ISAs missing most requirements, a concern is that the 64-bit address version of SPARC (V9) is proprietary, and that OpenRISC may have lost momentum.

RISC-V has plenty of momentum. Table 1 lists other groups designing RISC-V SoCs. Thanks in part to the highly productive, open-source hardware design system Chisel⁹, Berkeley has 8 silicon chips already and more in progress. Table 2 shows one 64-bit RISC-V core that is half the area, half the power, and faster than a 32-bit ARM core with a similar pipeline in the same process.

Although it's hard to set aside biases, we believe that RISC-V is the best and safest choice for a free, open RISC ISA. Thus, we will hold workshops and tutorials¹⁰ to expand the RISC-V community and, inspired by Table 3, plan to start a non-profit foundation to certify implementations and to maintain and evolve the ISA.

Conclusion

The case is even clearer for an open ISA than for an open OS, as ISAs change very slowly, whereas algorithmic innovations and new application demands force continual OS evolution. It is also an interface standard like TCP/IP, thus simpler to maintain and evolve than an OS.

Open ISAs have been tried before, but they never became popular due to the lack of demand. The low cost and power of IoTs, the desire for a WSC alternative to the 80x86, and the fact that cores are a small but ubiquitous fraction of all SoCs combine to supply that missing demand. RISC-V is aimed at SoCs, with a base that should never change given the longevity of the basic RISC ideas; a standard set of optional extensions that will evolve slowly; and unique instructions per SoC that never need to be reused. While the first RISC-V beachhead may be IoTs or perhaps WSCs, our goal is grander: just as Linux has become the standard OS for most computing devices, we envision RISC-V becoming the standard ISA for all computing devices.

References

- ¹ MIPS letter (2002). http://brej.org/yellow_star/letter.pdf.
- ² Demerjian, C. (2013). "A long look at how ARM licenses chips: Part 1 of 2," semiaccurate.com/2013/08/07/a-long-look-at-how-arm-licenses-chips/.
- ³ Raymond, E. (1999). The Cathedral and the Bazaar. *Knowledge, Technology & Policy*, 12(3), 23-49.
- ⁴ Patterson, D. & D. Ditzel. (1980) "The Case for the Reduced Instruction Set Computer." SIGARCH Computer Architecture News 8.6, 25-33.
- ⁵ We recently learned about the new Open Core Foundation, which is planning a 64-bit open core for 2016 based on SH-4.
- ⁶ Waterman, A. *et al.* (2014). *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0*. EECS Technical Report No. UCB/EECS-2014-54, UC Berkeley.
- ⁷ Estrin, G. (1960) "Organization of computer systems: the fixed plus variable structure computer." *Western Joint IRE-AIEE-ACM Computer Conference*, 33-40.
- ⁸ Bell, G., & W. Strecker. (1976) "Computer structures: What have we learned from the PDP-11?," *3rd ISCA*, 1-14.
- ¹⁰ Bachrach, J., *et al.* (2012) "Chisel: constructing hardware in a Scala embedded language." *Proc. 49th DAC*, 1216-1225.
- ⁸ The first RISC-V workshop will be held January 14-15, 2015 in Monterey, CA. <https://www.regonline.com/riscvworkshop>.

Org	Cores	Description
IIT Madras	6	Development of a complete range of processors, ranging from micro-controllers to server/HPC grade processors. They began with the IBM Power ISA, but switched a year later to RISC-V for both technical and licensing reasons. The 6 distinct Indian processors and associated SoC components are designed to offer viable, open source alternatives to proprietary commercial processors. All implementations will be provided as patent/royalty-free, BSD-licensed open source in keeping with the RISC-V philosophy (rise.cse.iitm.ac.in/shakti.html)
Low-RISC	1	The lowRISC project (lowrisc.org) is based in Cambridge (UK) and led by one of the founders of Raspberry Pi, which is a popular \$35 computer. Their goal is to produce open source RISC-V-based SoCs, and they have plans for volume silicon manufacture and low-cost development boards.
Blue-spec	1	The EDA company Bluespec (bluespec.com) in the US has customers interested in an open ISA, so they are doing RISC-V designs in the Bluespec synthesis toolset and have ported the GDB debugger and the GNU soft-float ABI to RISC-V

Table 1. RISC-V projects beyond UC Berkeley.

ISA	Width (bits)	Frequency (GHz)	Dhrystone Performance (DMIPS/MHz)	Area mm ² (no caches)	Area mm ² (16 KB caches)	Area Efficiency (DMIPS/MHz/mm ²)	Dynamic Power (mW/MHz)
ARM	32	>1	1.57	0.27	0.53	3.0	<0.080
RISC-V	64	>1	1.72	0.14	0.39	4.4	0.034
R/A	2	1	1.1	0.5	0.7	1.5	≥0.4

Table 2. Comparison of a 32-bit ARM core (Cortex-A5) to a 64-bit RISC-V core (Rocket) built in the same TSMC process (40GPLUS). Third row is ratio of RISC-V Rocket to ARM Cortex-A5. Both use single-instruction-issue, in-order pipelines, yet the RISC-V core is faster, smaller, and uses less power. This data is from the ARM website and the paper "A 45nm 1.3GHz 16.7 Double-Precision GFLOPS/W RISC-V Processor with Vector Accelerators" by Y. Lee *et al* that will appear in the 40th European Solid-State Circuits Conference, September 22-24, 2014.

Name	Year	Description
Apache Software Foundation	1999	Provides support for the Apache community of open-source software projects, which provide software products for the public good.
Free Software Foundation	1985	Works to secure freedom for computer users by promoting the development and use of free software and documentation — particularly the GNU operating system.
Open Group	1996	A vendor and technology-neutral industry consortium, currently with over 400 member organizations. It was formed in 1996 when X/Open merged with the Open Software Foundation. Services provided include strategy, management, innovation and research, standards, certification, and test development. The Open Group is most famous as the certifying body for UNIX trademark.

Table 3. Example non-profit software foundations that maintain and evolve open source projects for decades. We presume to match the longevity of such software projects, we will need a similar organization to maintain and evolve a free, open ISA.

Category	Name	Format	RV32I Base		+RV64	+RV128																								
Loads	Load Byte	I	LB	rd,rs1,imm																										
	Load Halfword	I	LH	rd,rs1,imm																										
	Load Word	I,Cx	LW	rd,rs1,imm	LD rd,rs1,imm	LQ rd,rs2,imm																								
	Load Byte Unsigned	I	LBU	rd,rs1,imm																										
	Load Half Unsigned	I	LHU	rd,rs1,imm	LWU rd,rs1,imm	LDU rd,rs1,imm																								
Stores	Store Byte	S	SB	rs1,rs2,imm																										
	Store Halfword	S	SH	rs1,rs2,imm																										
	Store Word	S,Cx	SW	rs1,rs2,imm	SD rs1,rs2,imm	SQ rs1,rs2,imm																								
Arithmetic	ADD	R,Cx	ADD	rd,rs1,rs2	ADDW rd,rs1,rs2	ADDD rd,rs1,rs2																								
	ADD Immediate	I,Cx	ADDI	rd,rs1,imm	ADDIW rd,rs1,imm	ADDID rd,rs1,imm																								
	SUBtract	R,Cx	SUB	rd,rs1,rs2	SUBW rd,rs1,rs2	SUBD rd,rs1,rs2																								
	Load Upper Imm	U	LUI	rd,imm																										
	Add Upper Imm to PC	U	AUIPC	rd,imm																										
Logical	XOR	R	XOR	rd,rs1,rs2																										
	XOR Immediate	I	XORI	rd,rs1,imm																										
	OR	R,Cx	OR	rd,rs1,rs2																										
	OR Immediate	I	ORI	rd,rs1,imm																										
	AND	R,Cx	AND	rd,rs1,rs2																										
	AND Immediate	I	ANDI	rd,rs1,imm																										
Shifts	Shift Left	R	SLL	rd,rs1,rs2	SLLW rd,rs1,rs2	SLLD rd,rs1,rs2																								
	Shift Left Immediate	I,Cx	SLLI	rd,rs1,shamt	SLLIW rd,rs1,shamt	SLLID rd,rs1,shamt																								
	Shift Right	R	SRL	rd,rs1,rs2	SRLW rd,rs1,rs2	SRLD rd,rs1,rs2																								
	Shift Right Immediate	I	SRLI	rd,rs1,shamt	SRLIW rd,rs1,shamt	SRLID rd,rs1,shamt																								
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2	SRAW rd,rs1,rs2	SRAD rd,rs1,rs2																								
	Shift Right Arith Imm	I	SRAI	rd,rs1,shamt	SRAIW rd,rs1,shamt	SRAID rd,rs1,shamt																								
Compare	Set <	R	SLT	rd,rs1,rs2																										
	Set < Immediate	I	SLTI	rd,rs1,imm																										
	Set < Unsigned	R	SLTU	rd,rs1,rs2																										
	Set < Unsigned Imm	I	SLTIU	rd,rs1,imm																										
Branches	Branch =	SB,Cx	BEQ	rs1,rs2,imm																										
	Branch ≠	SB,Cx	BNE	rs1,rs2,imm																										
	Branch <	SB	BLT	rs1,rs2,imm																										
	Branch ≥	SB	BGE	rs1,rs2,imm																										
	Branch < Unsigned	SB	BLTU	rs1,rs2,imm																										
	Branch ≥ Unsigned	SB	BGEU	rs1,rs2,imm																										
Jump & Link	J&L	UJ,Cx	JAL	rd,imm																										
	Jump & Link Register	UJ,Cx	JALR	rd,rs1,imm																										
Synch	Synch threads	I	FENCE																											
	Synch Instr & Data	I	FENCE.I																											
System	System CALL	I	SCALL																											
	System BREAK	I	SBREAK																											
Counters	ReaD CYCLE	I	RDCYCLE	rd																										
	ReaD CYCLE upper Half	I	RDCYCLEH	rd																										
	ReaD TIME	I	RDTIME	rd																										
	ReaD TIME upper Half	I	RDTIMEH	rd																										
	ReaD INSTR RETired	I	RDINSTRET	rd																										
	ReaD INSTR upper Half	I	RDINSTRETH	rd																										
32-bit Formats							16-bit Formats																							
	31	27	26	25	24	20	19	15	14	12	11	7	6	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	funct7			rs2	rs1	funct3	rd	opcode		CI1	funct4		rd	rs1		op														
R4	rs3	funct2		rs2	rs1	funct3	rd	opcode		CI2	funct4		rd	imm[4:0]		op														
I	imm[11:0]				rs1	funct3	rd	opcode		CJ	funct4		jump target				op													
S	imm[11:5]			rs2	rs1	funct3	imm[4:0]	opcode		CI3	funct3	imm[2:0]	rd'	imm[4:3]	rs1'	op														
SB	imm[12:10:5]			rs2	rs1	funct3	imm[4:1:11]	opcode		CB	funct3	rs2'	imm[2:0]	imm[4:3]	rs1'	op														
U	imm[31:12]						rd	opcode		CR	funct3	rs2'	rd'	op	rs1'	op														
UJ	imm[20:10:11:19:12]						rd	opcode																						

Table 4. RISC-V Integer Base Instructions (RV32I/64I/128I) and instruction formats. The base has 40 classic RISC integer instructions, plus 10 miscellaneous instructions for synchronization, system calls, and counters. All RISC-V implementations must include these base instructions, and we call the 32-bit version RV32I. The 64-bit and 128-bit versions (RV64I and RV128I) expand all the registers to those widths and add 10 instructions for new data transfer and shift instructions of the wider formats. It also shows the optional compressed instruction extension: those 12 instructions with Cx formats, which are 16 bits long. There are other optional instruction extensions defined so far: Multiply-Divide, SP/DP/QP Floating Point, and Atomic. To learn more, see www.riscv.org

Category	Name	Format	RV32M (Multiply-Divide)	+RV64	+RV128
Multiply	MULTIPLY	R	MUL rd,rs1,rs2	MULW rd,rs1,rs2	MULD rd,rs1,rs2
	MULTIPLY upper Half	R	MULH rd,rs1,rs2		
	MULTIPLY Half Sign/Uns	R	MULHSU rd,rs1,rs2		
	MULTIPLY upper Half Uns	R	MULHU rd,rs1,rs2		
	Divide	DIVide	R	DIV rd,rs1,rs2	DIVW rd,rs1,rs2
	DIVide Unsigned	R	DIVU rd,rs1,rs2		
Remainder	REMAInder	R	REM rd,rs1,rs2	REMW rd,rs1,rs2	REMD rd,rs1,rs2
	REMAInder Unsigned	R	REMU rd,rs1,rs2	REMUW rd,rs1,rs2	REMUd rd,rs1,rs2
Category	Name	Format	RV32{F,D,Q} (SP,DP,QP Fl. Pt.)	+RV64	+RV128
Load	Load	I	FL{W,D,Q} rd,rs1,imm		
Store	Store	S	FS{W,D,Q} rs1,rs2,imm		
Arithmetic	ADD	R	FADD.{S,D,Q} rd,rs1,rs2		
	SUBtract	R	FSUB.{S,D,Q} rd,rs1,rs2		
	MULTIPLY	R	FMUL.{S,D,Q} rd,rs1,rs2		
	DIVide	R	FDIV.{S,D,Q} rd,rs1,rs2		
	SQure RooT	R	FSQRT.{S,D,Q} rd,rs1		
Mul-Add	MULTIPLY-ADD	R4	FMADD.{S,D,Q} rd,rs1,rs2,rs3		
	MULTIPLY-SUBtract	R4	FMSUB.{S,D,Q} rd,rs1,rs2,rs3		
	NEGATIVE MULTIPLY-SUBtract	R4	FNMSUB.{S,D,Q} rd,rs1,rs2,rs3		
	NEGATIVE MULTIPLY-ADD	R4	FNMADD.{S,D,Q} rd,rs1,rs2,rs3		
Move	Move from Integer	R	FMV.X.S rd,rs1	FMV.X.D rd,rs1	FMV.X.Q rd,rs1
	Move to Integer	R	FMV.S.X rd,rs1	FMV.D.X rd,rs1	FMV.Q.X rd,rs1
Sign Inject	SIGN source	R	FSGNJ.{S,D,Q} rd,rs1,rs2		
	NEGATIVE SIGN source	R	FSGNJN.{S,D,Q} rd,rs1,rs2		
	Xor SIGN source	R	FSGNJX.{S,D,Q} rd,rs1,rs2		
Min/Max	MINimum	R	FMIN.{S,D,Q} rd,rs1,rs2		
	MAXimum	R	FMAX.{S,D,Q} rd,rs1,rs2		
Compare	Compare Float =	R	FEQ.{S,D,Q} rd,rs1,rs2		
	Compare Float <	R	FLT.{S,D,Q} rd,rs1,rs2		
	Compare Float ≤	R	FLE.{S,D,Q} rd,rs1,rs2		
Convert	Convert from Int	R	FCVT.W.{S,D,Q} rd,rs1	FCVT.L.{S,D,Q} rd,rs1	FCVT.T.{S,D,Q} rd,rs1
	Convert from Int Unsigned	R	FCVT.WU.{S,D,Q} rd,rs1	FCVT.LU.{S,D,Q} rd,rs1	FCVT.TU.{S,D,Q} rd,rs1
	Convert to Int	R	FCVT.{S,D,Q}.W rd,rs1	FCVT.{S,D,Q}.L rd,rs1	FCVT.{S,D,Q}.T rd,rs1
	Convert to Int Unsigned	R	FCVT.{S,D,Q}.WU rd,rs1	FCVT.{S,D,Q}.LU rd,rs1	FCVT.{S,D,Q}.TU rd,rs1
Categorization	Classify Type	R	FCLASS.{S,D,Q} rd,rs1		
Configuration	Read Status	R	FRCSR rd		
	Read Rounding Mode	R	FRRM rd		
	Read Flags	R	FRFLAGS rd		
	Swap Status Reg	R	FSCSR rd,rs1		
	Swap Rounding Mode	R	FSRM rd,rs1		
	Swap Flags	R	FSFLAGS rd,rs1		
	Swap Rounding Mode Imm	I	FSRMI rd,imm		
	Swap Flags Imm	I	FSFLAGSI rd,imm		
Category	Name	Format	RV32A (Atomic)	+RV64	+RV128
Load	Load Reserved	R	LR.W rd,rs1	LR.D rd,rs1	LR.Q rd,rs1
Store	Store Conditional	R	SC.W rd,rs1,rs2	SC.D rd,rs1,rs2	SC.Q rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.W rd,rs1,rs2	AMOSWAP.D rd,rs1,rs2	AMOSWAP.Q rd,rs1,rs2
Add	ADD	R	AMOADD.W rd,rs1,rs2	AMOADD.D rd,rs1,rs2	AMOADD.Q rd,rs1,rs2
	Logical	XOR	R	AMOXOR.W rd,rs1,rs2	AMOXOR.D rd,rs1,rs2
	AND	R	AMOAND.W rd,rs1,rs2	AMOAND.D rd,rs1,rs2	AMOAND.Q rd,rs1,rs2
	OR	R	AMOOR.W rd,rs1,rs2	AMOOR.D rd,rs1,rs2	AMOOR.Q rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.W rd,rs1,rs2	AMOMIN.D rd,rs1,rs2	AMOMIN.Q rd,rs1,rs2
	MAXimum	R	AMOMAX.W rd,rs1,rs2	AMOMAX.D rd,rs1,rs2	AMOMAX.Q rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.W rd,rs1,rs2	AMOMINU.D rd,rs1,rs2	AMOMINU.Q rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.W rd,rs1,rs2	AMOMAXU.D rd,rs1,rs2	AMOMAXU.Q rd,rs1,rs2

Table 5. RISC-V Optional Extensions: Multiply-Divide, SP/DP/QP Fl. Pt., and Atomic. It further demonstrates the base-plus-extension nature of RISC-V, which has optional extensions of: 10 multiply-divide instructions (RV32M); 25 floating-point instructions each for SP, DP, or QP (RV32S, RV32D, RV32Q); and 11 optional atomic instructions (RV32A). Just as when expanding from RV32I to RV64I and RV128I, for each address-size option we need to add a few more instructions for the wider data: 4 wider multiples and divides; 6 moves and converts for floating point; and 11 wider versions of the atomic instructions. To learn more, see www.riscv.org.