Instruction Sets Want to be Free!

Krste Asanovic
UC Berkeley, RISC-V Foundation, & SiFive Inc.
krste@berkeley.edu
www.riscv.org

Barcelona Supercomputer Center
June 7, 2017

Why Instruction Set Architecture matters

- **Why can't Intel sell mobile chips?**
  - 99%+ of mobile phones/tablets based on ARM v7/v8 ISA
- **Why can't ARM partners sell servers?**
  - 99%+ of laptops/desktops/servers based on AMD64 ISA (over 95%+ built by Intel)
- **How can IBM still sell mainframes?**
  - IBM 360, oldest surviving ISA (50+ years)

ISA is most important interface in computer system

where software meets hardware

Open Software/Standards Work!

Field	*Standard*	*Free, Open Impl.*	*Proprietary Impl.*
Networking	Ethernet, TCP/IP	Many	Many
OS	Posix	Linux, FreeBSD	M/S Windows
Compilers	C	gcc, LLVM	Intel icc, ARMcc
Databases	SQL	MySQL, PostgresSQL	Oracle 12C, M/S DB2
Graphics	OpenGL	Mesa3D	M/S DirectX
ISA	??????	-----------	x86, ARM, IBM360

- Why not successful free & open standards and free & open implementations, like other fields
- Dominant proprietary ISAs are not great designs

What is RISC-V?

- Fifth generation of RISC design from UC Berkeley
- A high-quality, license-free, royalty-free RISC ISA specification
- Experiencing rapid uptake in both industry and academia
- Standard maintained by non-profit RISC-V Foundation
- Both proprietary and open-source core implementations
- Supported by growing shared software ecosystem
- Appropriate for all levels of computing system, from microcontrollers to supercomputers

RISC-V Origins

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research at Berkeley, time to choose ISA for next set of projects
- Obvious choices: x86 and ARM

Intel x86 “AAA” Instruction

- ASCII Adjust After Addition
- AL register is default source and destination

- If the low nibble is > 9 decimal, or the auxiliary carry flag AF = 1, then
  - Add 6 to low nibble of AL and discard overflow
  - Increment high byte of AL
  - Set CF and AF
- Else
  - CF = AF = 0

- Single byte instruction

ARM v7 LDMIAEQ Instruction

```
LDMIAEQ SP!, {R4-R7, PC}
```

- **L**oa**D** **M**ultiple, **I**ncrement-**A**ddress
- Writes to 7 registers from 6 loads
- Only executes if **EQ** condition code is set
- Writes to the PC (a conditional branch)
- Can change instruction sets

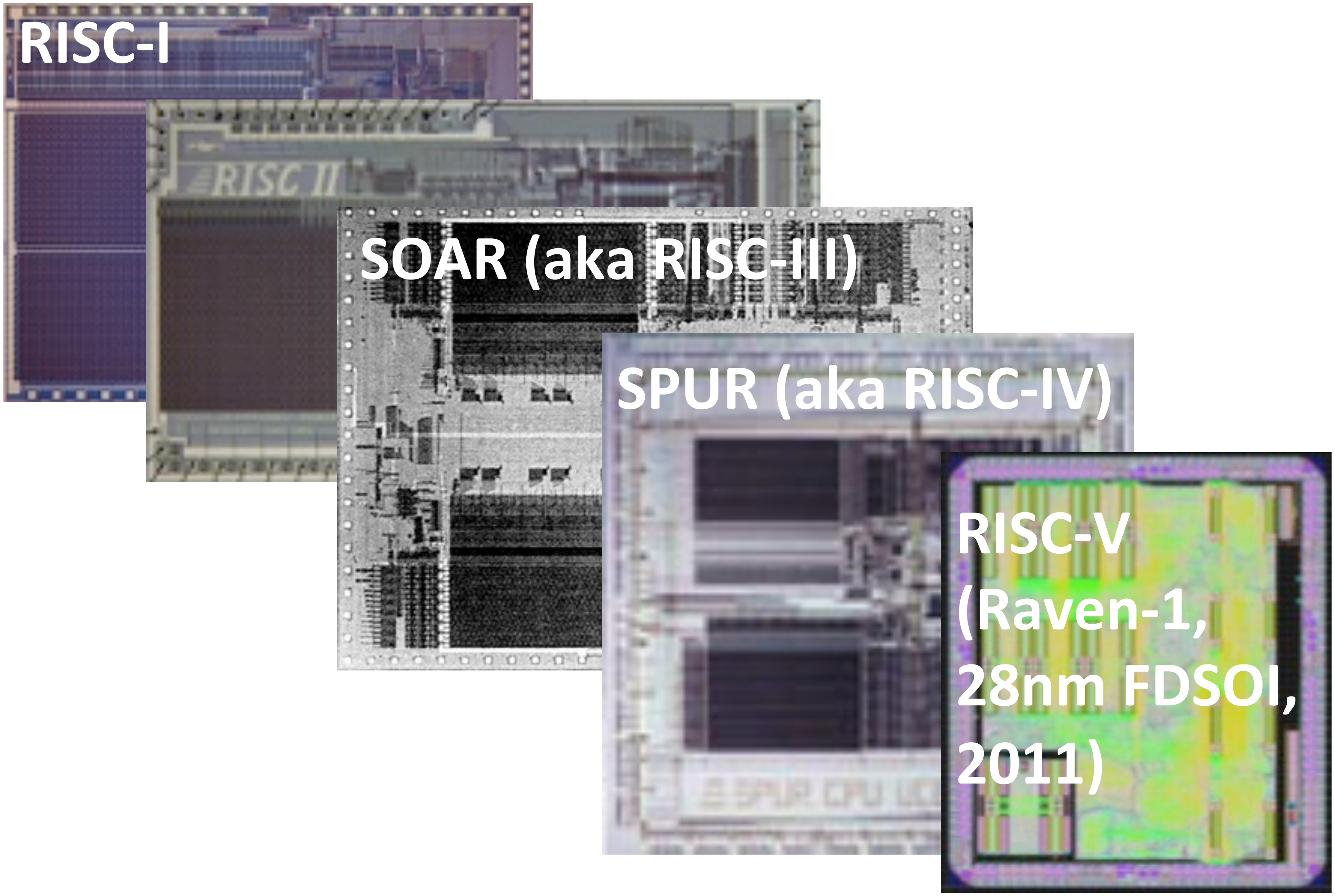
- Idiom for "stack pop and return from a function call"

RISC-V Origin Story

- x86 impossible –IP issues, too complex
- ARM mostly impossible – no 64-bit, IP issues, complex
- So we started "3-month project" in summer 2010 to develop our own clean-slate ISA
  - Andrew Waterman, Yunsup Lee, Dave Patterson, Krste Asanovic principal designers
- Four years later, we released frozen base user spec
  - First public specification released in May 2011
  - Many tapeouts and several publications along the way

Why are outsiders complaining about changes to RISC-V in Berkeley classes?

Why is name RISC-V?
(pronounced “risk-five”)



Why Didn’t Other Open ISAs Take Off?

- ***SPARC V8*** - To its credit, Sun Microsystems made SPARC V8 an IEEE standard in 1994
  - Sun, Gaisler offered open-source cores
  - ISA now owned by Oracle
- ***OpenRISC*** - GNU open-source effort started in 1999, based on DLX from *Computer Architecture: AQA*
  - 64-bit ISA was in progress in 2010
  - Didn’t separate Architecture and Implementation

- Competing in microprocessor era – now in SoC era
- Don’t meet the needs of a universal ISA

Universal ISA Requirements

- Works well with existing software stacks, languages
- Is native hardware ISA, not virtual machine/ANDF
- Suits all sizes of processor, from smallest microcontroller to largest supercomputer
- Suits all implementation technologies, FPGA, ASIC, full-custom, future device technologies…
- Efficient for all microarchitecture styles: microcoded, in-order, decoupled, out-of-order, single-issue, superscalar, …
- Supports extensive specialization to act as base for customized accelerators
- Stable: not changing, not disappearing

Companies and their ISAs Come and Go

Proprietary ISA fortunes tied to business fortunes and whims

- Digital Equipment Corporation
  - PDP-11, VAX, Alpha
- Intel
  - Itanium
- ARM
  - Sold to Softbank at >40% premium
  - Now 25% sold off to Abu Dhabi investment fund
- Imagination
  - MIPS being sold off

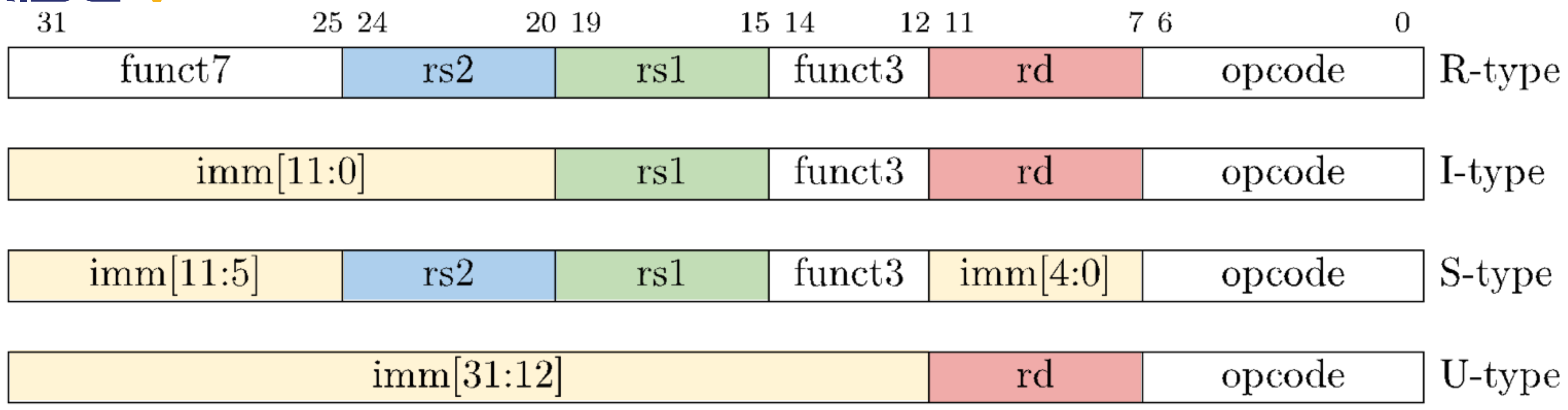
What’s Different about RISC-V?

- *Simple*
  - Far smaller than other commercial ISAs
- *Clean-slate design*
  - Clear separation between user and privileged ISA
  - Avoids µarchitecture or technology-dependent features
- A *modular* ISA
  - Small standard base ISA
  - Multiple standard extensions
- Designed for *extensibility/specialization*
  - Variable-length instruction encoding
  - Vast opcode space available for instruction-set extensions
- *Stable*
  - Base and standard extensions are frozen
  - Additions via optional extensions, not new versions

RISC-V Base Plus Standard Extensions

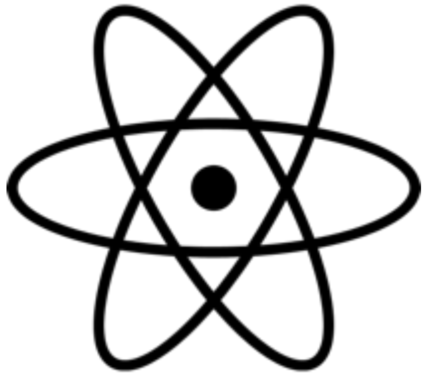
- Four base integer ISAs
  - RV32E, RV32I, RV64I, RV128I
  - RV32E is 16-register subset of RV32I
  - Only <50 hardware instructions needed for base
- Standard extensions
  - M: Integer multiply/divide
  - A: Atomic memory operations (AMOs + LR/SC)
  - F: Single-precision floating-point
  - D: Double-precision floating-point
  - G = IMAFD, “General-purpose” ISA
  - Q: Quad-precision floating-point
- All the above are a fairly standard RISC encoding in a fixed 32-bit instruction format
- Above user-level ISA components frozen in 2014
  - Supported forever after

RISC-V Standard Base ISA Details



- 32-bit fixed-width, naturally aligned instructions
- 31 integer registers x1-x31, plus x0 zero register
- rd/rs1/rs2 in fixed location, no implicit registers
- Immediate field (instr[31]) always sign-extended
- Floating-point adds f0-f31 registers plus FP CSR, also fused mul-add four-register format
- Designed to support PIC and dynamic linking

“A”: Atomic Operations Extension



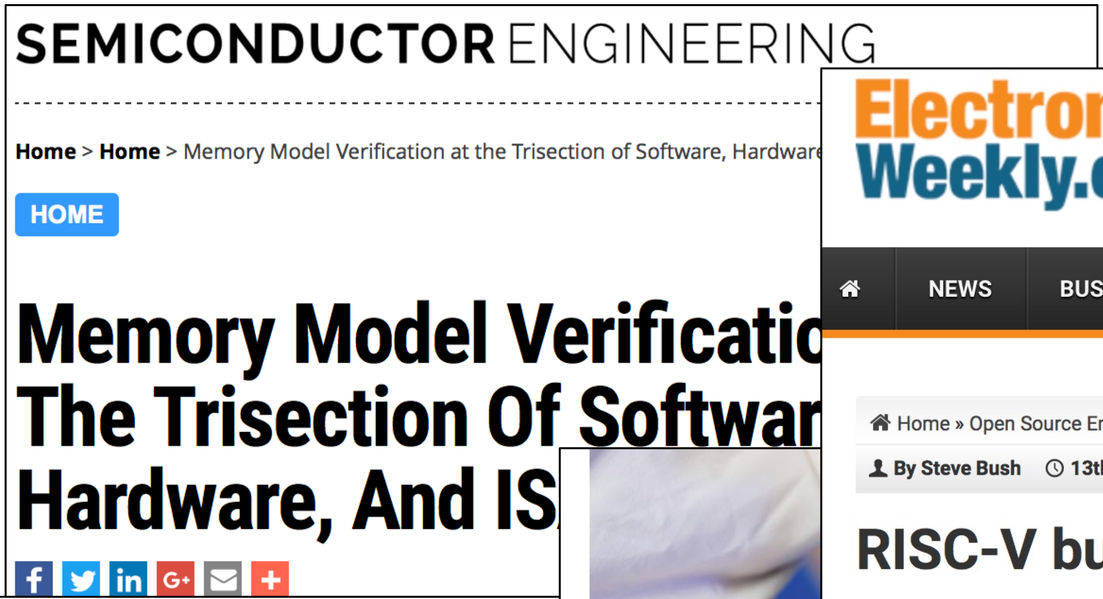
Two classes:

- Atomic Memory Operations (AMO)
  - Fetch-and-op, op=ADD,OR,XOR,MAX,MIN,MAXU,MINU
- Load–Reserved/Store Conditional
  - With forward progress guarantee for short sequences
- All atomic operations can be annotated with two bits (Acquire/Release) to implement release consistency or sequential consistency

- *Some recent press on RISC-V memory model*

RISC-V

You can tell you've made it, when FUD appears

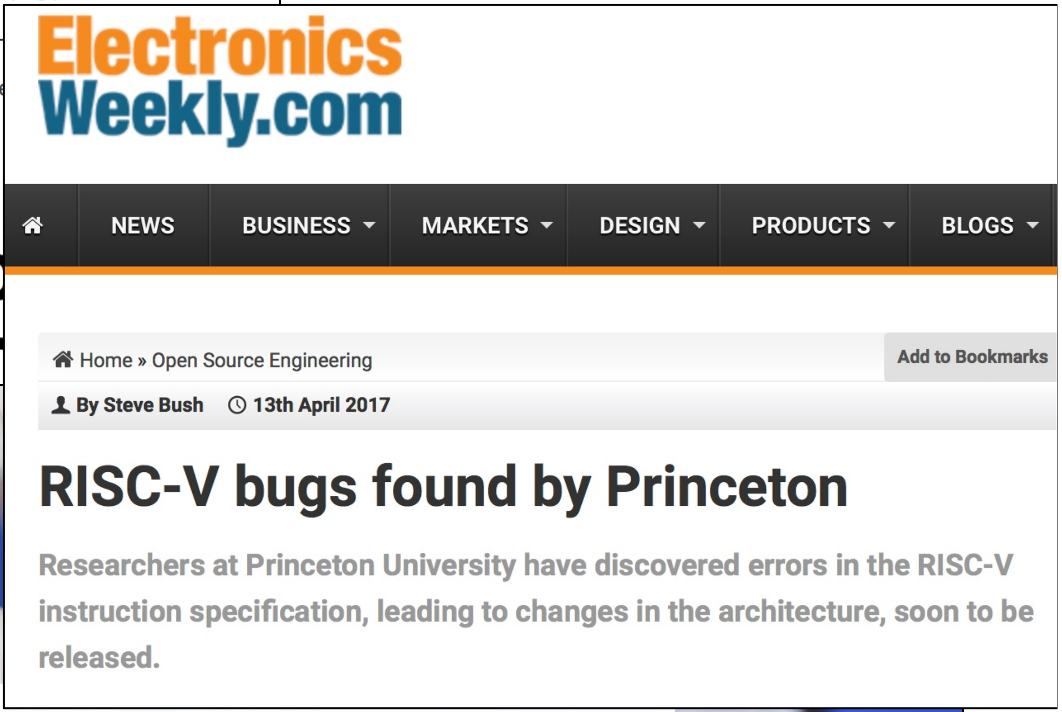


SEMICONDUCTOR ENGINEERING

Home > Home > Memory Model Verification at the Trisection of Software, Hardware

HOME

Memory Model Verificatio
The Trisection Of Softwar
Hardware, And IS



Electronics Weekly.com

NEWS BUSINESS MARKETS DESIGN PRODUCTS BLOGS

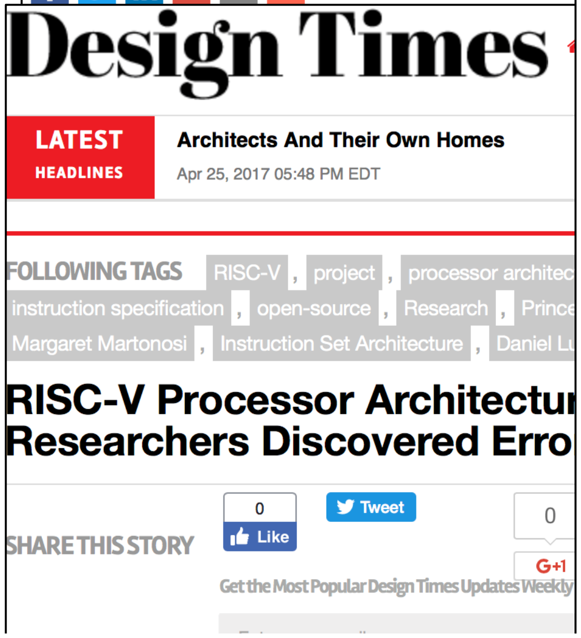
Home » Open Source Engineering

Add to Bookmarks

By Steve Bush 13th April 2017

RISC-V bugs found by Princeton

Researchers at Princeton University have discovered errors in the RISC-V instruction specification, leading to changes in the architecture, soon to be released.



Design Times

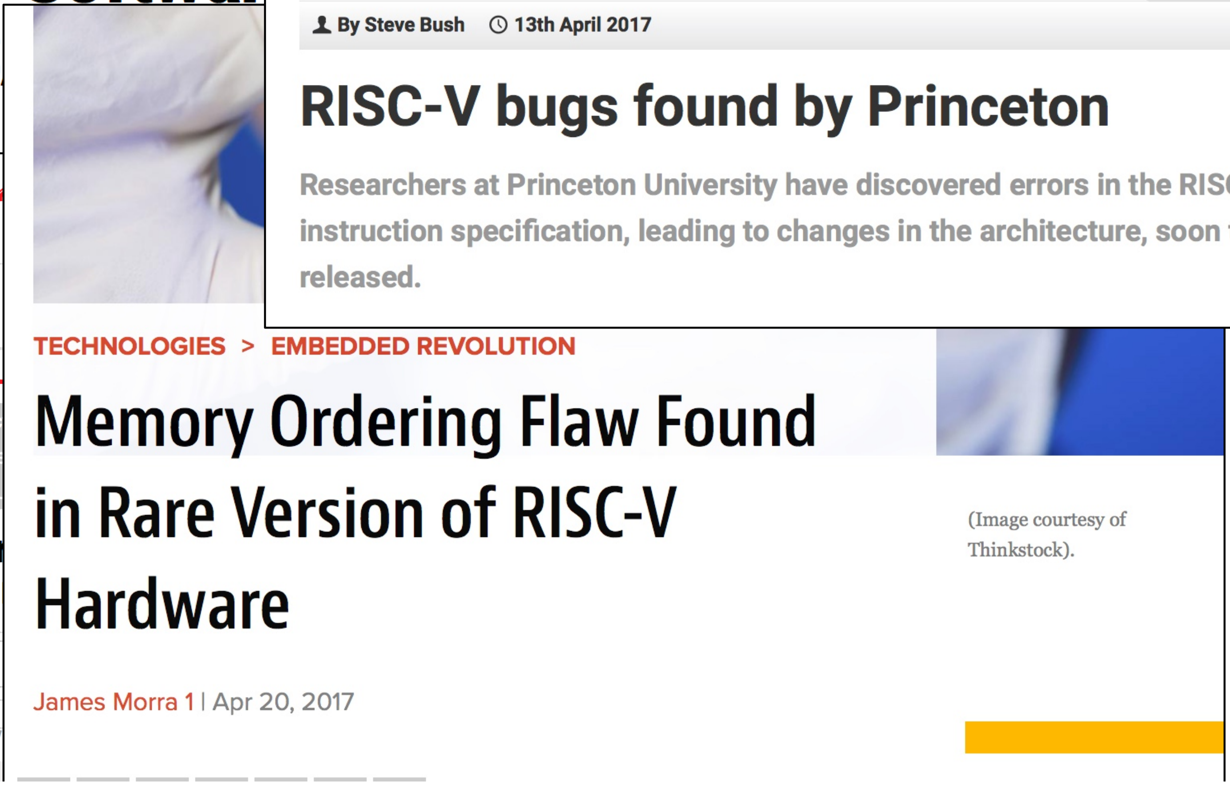
LATEST HEADLINES Architects And Their Own Homes
Apr 25, 2017 05:48 PM EDT

FOLLOWING TAGS RISC-V , project , processor architec
instruction specification , open-source , Research , Prince
Margaret Martonosi , Instruction Set Architecture , Daniel Lu

RISC-V Processor Architectu
Researchers Discovered Erro

SHARE THIS STORY 0 Like Tweet 0 G+1

Get the Most Popular Design Times Updates Weekly



TECHNOLOGIES > EMBEDDED REVOLUTION

Memory Ordering Flaw Found in Rare Version of RISC-V Hardware

(Image courtesy of Thinkstock).

James Morra 1 | Apr 20, 2017

Memory Consistency Model Update

- No existing RISC-V hardware affected
  - No implementations were that complicated yet
- Guidance released for implementations and compiler authors
  - Conservative on both sides, can be relaxed later
- Native memory model likely to be considerably tighter than original loose spec
  - Probably multi-copy atomic
  - Loads to same address are ordered
  - Probably other address/data/control dependencies honored
  - Investigating ways to support looser/tighter memory models cleanly

Variable-Length Encoding

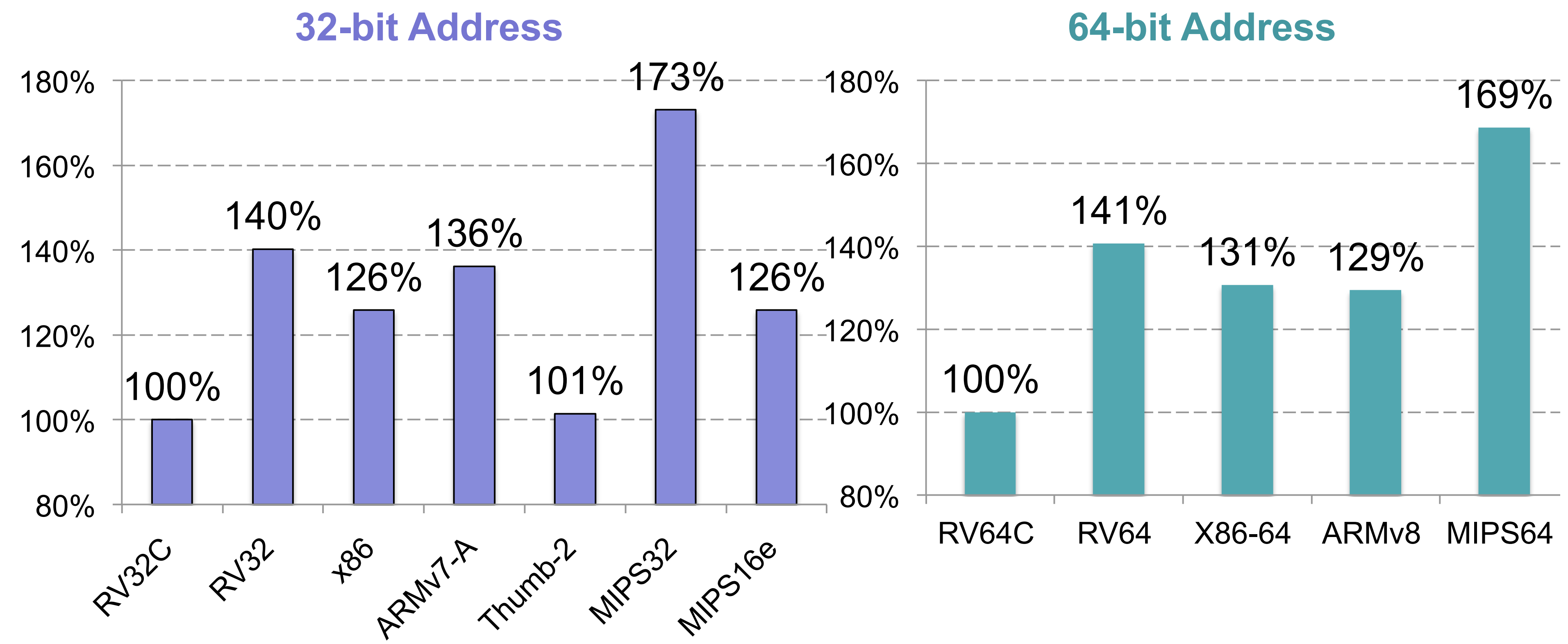
			`xxxxxxxxxxxxxxaa`	16-bit (`aa` $\neq$ `11`)
		`xxxxxxxxxxxxxxxx`	`xxxxxxxxxxxbbb11`	32-bit (`bbb` $\neq$ `111`)
	`···xxxx`	`xxxxxxxxxxxxxxxx`	`xxxxxxxxxx011111`	48-bit
	`···xxxx`	`xxxxxxxxxxxxxxxx`	`xxxxxxxxx0111111`	64-bit
	`···xxxx`	`xxxxxxxxxxxxxxxx`	`xnnnxxxxx1111111`	(80+16*`nnn`)-bit, `nnn`$\neq$`111`
	`···xxxx`	`xxxxxxxxxxxxxxxx`	`x111xxxxx1111111`	Reserved for $\geq$192-bits
Byte Address:	base+4	base+2	base	

- Extensions can use any multiple of 16 bits as instruction length
- Branches/Jumps target 16-bit boundaries even in fixed 32-bit base
  - Consumes 1 extra bit of jump/branch address

"C": Compressed Instruction Extension

- Compressed code important for:
  - low-end embedded to save static code space
  - high-end commercial workloads to reduce cache footprint
- C extension adds 16-bit compressed instructions
  - 2-address forms with all 32 registers
  - 2/3-address forms with most frequent 8 registers
- 1 compressed instruction expands to 1 base instruction
  - Assembly lang. programmer & compiler oblivious
  - RVC ⇒ RVI decoder only ~700 gates (~2% of small core)
- All original 32-bit instructions retain encoding but now can be 16-bit aligned
- 50%-60% instructions compress ⇒ 25%-30% smaller

SPECint2006 compressed code size with save/restore optimization (relative to “standard” RVC)

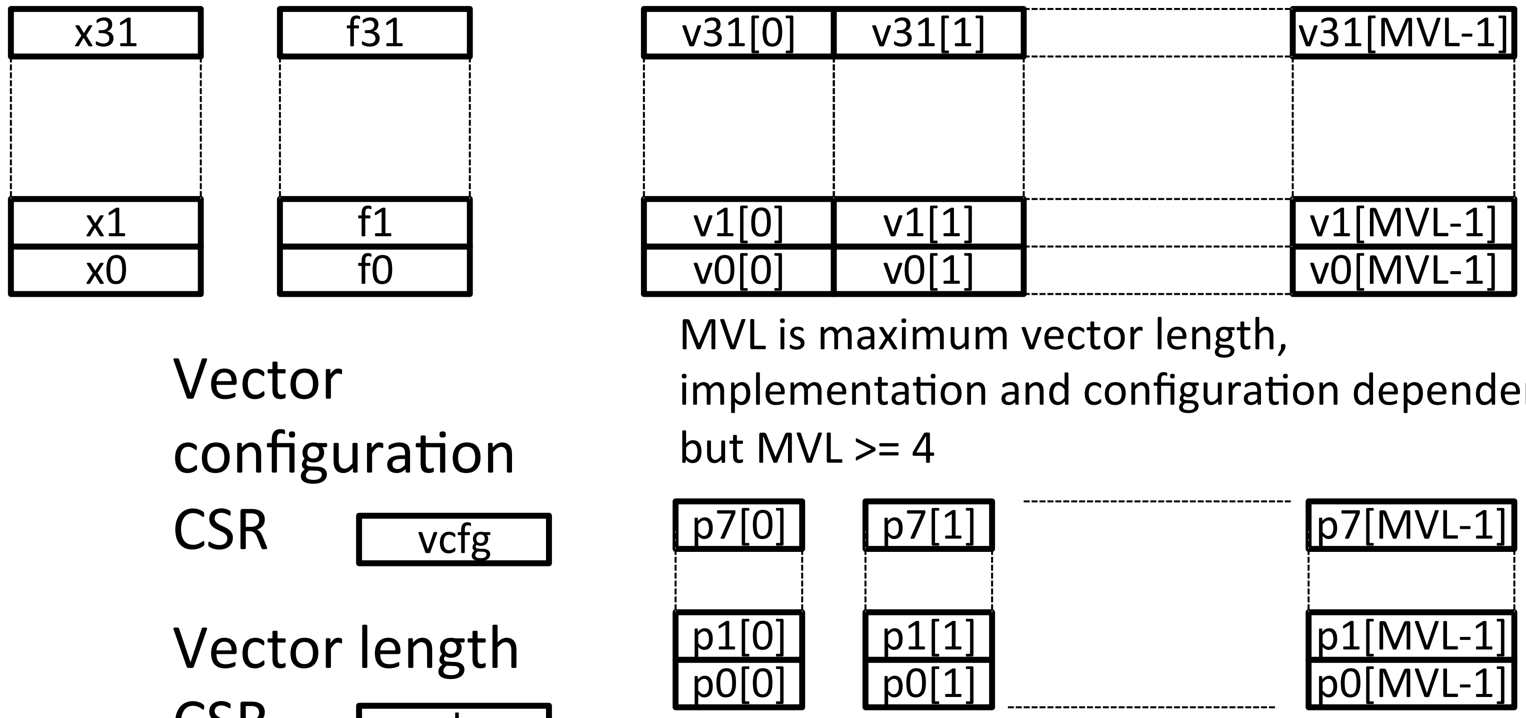


- RISC-V now smallest ISA for 32- and 64-bit addresses
- All results with same GCC compiler and options

Proposed “V” Vector Extension State

Standard RISC-V scalar x and f registers

Up to 32 vector data registers, v0-v31, of at least 4 elements each, with variable bits/element (8,16,32,64,128)



MVL is maximum vector length, implementation and configuration dependent, but MVL >= 4

Vector configuration CSR

Vector length CSR

8 vector predicate registers, with 1 bit per element

RISC-V Privileged Architecture

- Three privilege modes
  - User (U-mode)
  - Supervisor (S-mode)
  - Machine (M-mode)
- Supported combinations of modes:

– M	(simple embedded systems)
– M, U	(embedded systems with protection)
– M, S, U	(systems running Unix-style operating systems)

- Hypervisors to be run in modified S mode
  - Prioritize support for Type-2 Hypervisors like KVM
  - Can also support Type-1 Hypervisors in same model

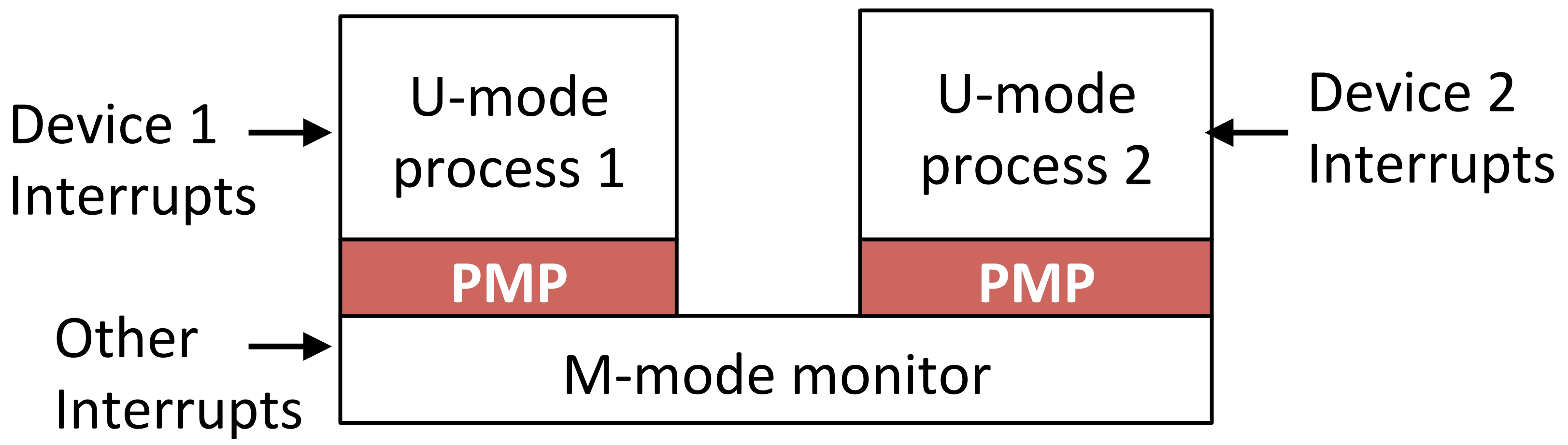
Simple Embedded Systems (M-mode only)

- No address translation/protection
  - "Mbare" bare-metal mode
  - Trap bad physical addresses precisely
- All code inherently trusted

- Low implementation cost
  - 2^7 bits of architectural state (in addition to user ISA)
  - $+2^7$ more bits for timers
  - $+2^7$ more for basic performance counters

Secure Embedded Systems (M, U modes)

- M-mode runs secure boot and runtime monitor
- Embedded code runs in U-mode
- Physical memory protection (PMP) on U-mode accesses
- Interrupt handling can be delegated to U-mode code
  - User-level interrupt support
- Provides arbitrary number of isolated subsystems
- Ongoing work to define trusted execution environments

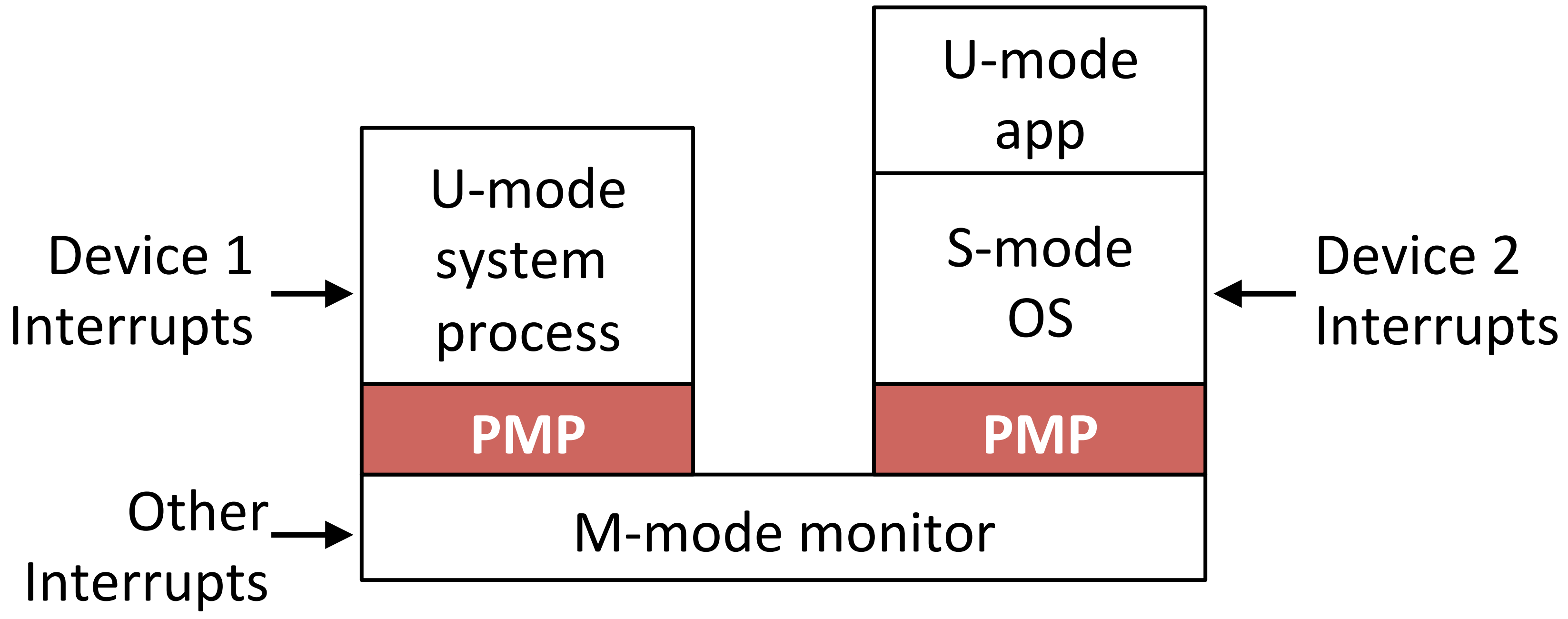


Virtual Memory Architectures (M, S, U modes)

- Designed to support current Unix-style operating systems
- Sv32 (RV32)
  - Demand-paged 32-bit virtual-address spaces
  - 2-level page table
  - 4 KiB pages, 4 MiB megapages
- Sv39 (RV64)
  - Demand-paged 39-bit virtual-address spaces
  - 3-level page table
  - 4 KiB pages, 2 MiB megapages, 1 GiB gigapages
- Sv48, Sv57, Sv64 (RV64)
  - Sv39 + 1/2/3 more page-table levels

S-Mode runs on top of M-mode

- M-mode runs secure boot and monitor
- S-mode runs OS
- U-mode runs application on top of OS or M-mode



RV32I

RISC-V ① ② ③ **RISC-V Reference Card** ④

Base Integer Instructions (32|64|128)				
Category	*Name*	*Fmt*	*RV{32|64|128}I Base*	
Loads	Load Byte	I	`LB`	`rd,rs1,imm`
	Load Halfword	I	`LH`	`rd,rs1,imm`
	Load Word	I	`L{W|D|Q}`	`rd,rs1,imm`
	Load Byte Unsigned	I	`LBU`	`rd,rs1,imm`
	Load Half Unsigned	I	`L{H|W|D}U`	`rd,rs1,imm`
Stores	Store Byte	S	`SB`	`rs1,rs2,imm`
	Store Halfword	S	`SH`	`rs1,rs2,imm`
	Store Word	S	`S{W|D|Q}`	`rs1,rs2,imm`
Shifts	Shift Left	R	`SLL{|W|D}`	`rd,rs1,rs2`
	Shift Left Immediate	I	`SLLI{|W|D}`	`rd,rs1,shamt`
	Shift Right	R	`SRL{|W|D}`	`rd,rs1,rs2`
	Shift Right Immediate	I	`SRLI{|W|D}`	`rd,rs1,shamt`
	Shift Right Arithmetic	R	`SRA{|W|D}`	`rd,rs1,rs2`
	Shift Right Arith Imm	I	`SRAI{|W|D}`	`rd,rs1,shamt`
Arithmetic	ADD	R	`ADD{|W|D}`	`rd,rs1,rs2`
	ADD Immediate	I	`ADDI{|W|D}`	`rd,rs1,imm`
	SUBtract	R	`SUB{|W|D}`	`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	`OR`	`rd,rs1,rs2`
	OR Immediate	I	`ORI`	`rd,rs1,imm`
	AND	R	`AND`	`rd,rs1,rs2`
	AND Immediate	I	`ANDI`	`rd,rs1,imm`
Compare	Set <	R	`SLT`	`rd,rs1,rs2`
	Set < Immediate	I	`SLTI`	`rd,rs1,imm`
	Set < Unsigned	R	`SLTU`	`rd,rs1,rs2`
	Set < Imm Unsigned	I	`SLTIU`	`rd,rs1,imm`
Branches	Branch =	SB	`BEQ`	`rs1,rs2,imm`
	Branch ≠	SB	`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	`JAL`	`rd,imm`
	Jump & Link Register	I	`JALR`	`rd,rs1,imm`
Synch	Synch thread	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`

+14 Privileged

+ 8 for M

+ 11 for A

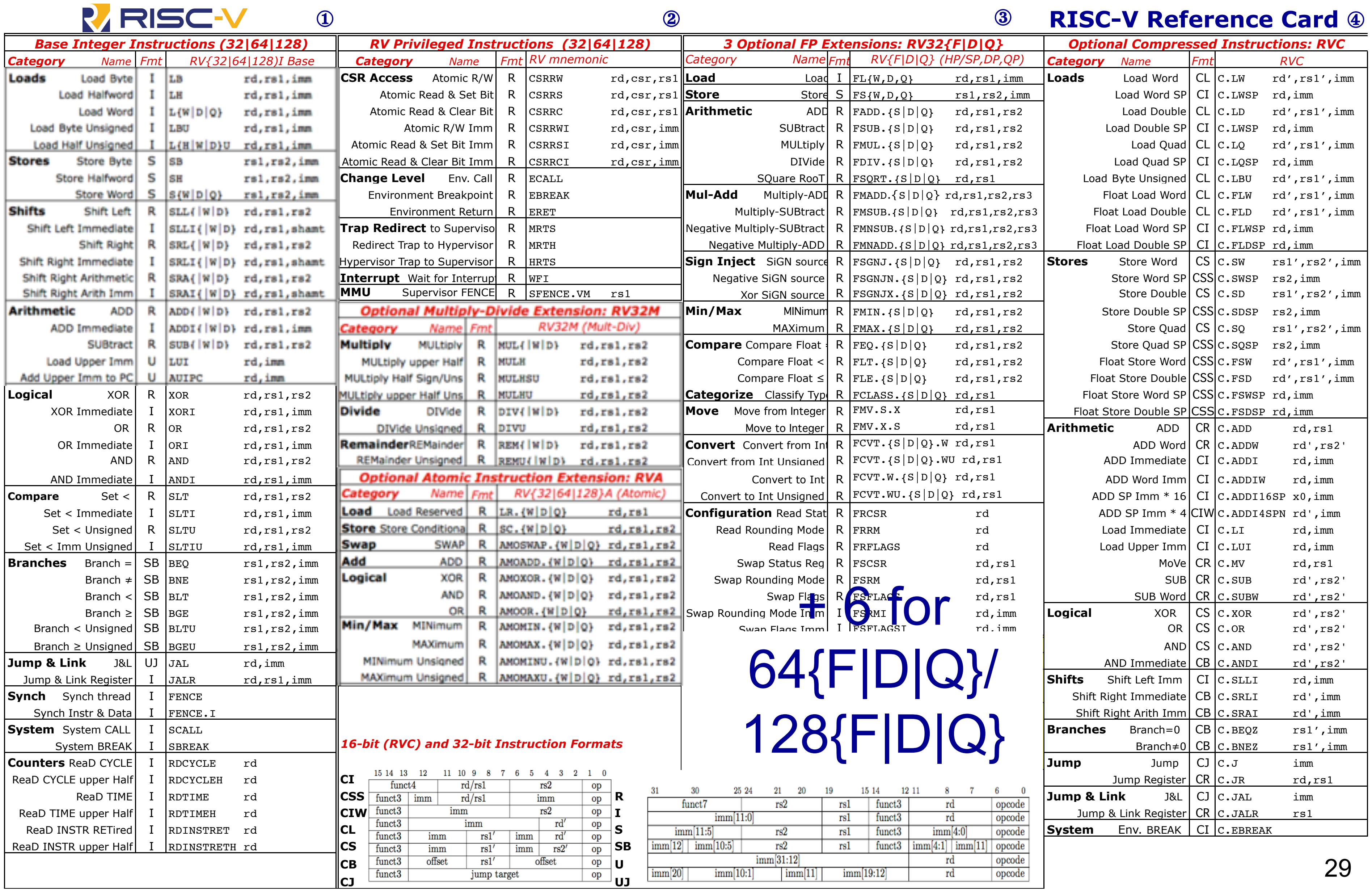
+ 34 for F, D, Q

+ 46 for C

32-bit Instruction Formats

	31	30 25	24 21	20	19 15	14 12	11 8	7	6 0
R	funct7		rs2		rs1	funct3	rd		opcode
I	imm[11:0]				rs1	funct3	rd		opcode
S	imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode
SB	imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode
U	imm[31:12]						rd		opcode
UJ	imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode

RV32I / RV64I / RV128I + M, A, F, D, Q, C



RISC-V ① ② ③ **RISC-V Reference Card ④**

Base Integer Instructions (32|64|128)

Category	Name	Fmt	RV{32|64|128)I Base	
Loads	Load Byte	I	LB	rd,rs1,imm
	Load Halfword	I	LH	rd,rs1,imm
	Load Word	I	L{W|D|Q}	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm
	Load Half Unsigned	I	L{H|W|D}U	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm
	Store Halfword	S	SH	rs1,rs2,imm
	Store Word	S	S{W|D|Q}	rs1,rs2,imm
Shifts	Shift Left	R	SLL{|W|D}	rd,rs1,rs2
	Shift Left Immediate	I	SLLI{|W|D}	rd,rs1,shamt
	Shift Right	R	SRL{|W|D}	rd,rs1,rs2
	Shift Right Immediate	I	SRLI{|W|D}	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA{|W|D}	rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI{|W|D}	rd,rs1,shamt
Arithmetic	ADD	R	ADD{|W|D}	rd,rs1,rs2
	ADD Immediate	I	ADDI{|W|D}	rd,rs1,imm
	SUBtract	R	SUB{|W|D}	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	OR	rd,rs1,rs2
	OR Immediate	I	ORI	rd,rs1,imm
	AND	R	AND	rd,rs1,rs2
	AND Immediate	I	ANDI	rd,rs1,imm
Compare	Set <	R	SLT	rd,rs1,rs2
	Set < Immediate	I	SLTI	rd,rs1,imm
	Set < Unsigned	R	SLTU	rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm
Branches	Branch =	SB	BEQ	rs1,rs2,imm
	Branch ≠	SB	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	JAL	rd,imm
	Jump & Link Register	I	JALR	rd,rs1,imm
Synch	Synch thread	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

RV Privileged Instructions (32|64|128)

Category	Name	Fmt	RV mnemonic	
CSR Access	Atomic R/W	R	CSRRW	rd,csr,rs1
	Atomic Read & Set Bit	R	CSRRS	rd,csr,rs1
	Atomic Read & Clear Bit	R	CSRRC	rd,csr,rs1
	Atomic R/W Imm	R	CSRRWI	rd,csr,imm
	Atomic Read & Set Bit Imm	R	CSRRSI	rd,csr,imm
	Atomic Read & Clear Bit Imm	R	CSRRCI	rd,csr,imm
Change Level	Env. Call	R	ECALL	
	Environment Breakpoint	R	EBREAK	
	Environment Return	R	ERET	
Trap Redirect	to Superviso	R	MRTS	
	Redirect Trap to Hypervisor	R	MRTH	
	Hypervisor Trap to Supervisor	R	HRTS	
Interrupt	Wait for Interrup	R	WFI	
MMU	Supervisor FENCE	R	SFENCE.VM	rs1

Optional Multiply-Divide Extension: RV32M

Category	Name	Fmt	RV32M (Mult-Div)	
Multiply	MULtiply	R	MUL{|W|D}	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{|W|D}	rd,rs1,rs2
	DIVide Unsigned	R	DIVU	rd,rs1,rs2
Remainder	REMainder	R	REM{|W|D}	rd,rs1,rs2
	REMainder Unsigned	R	REMU{|W|D}	rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

Category	Name	Fmt	RV{32|64|128}A (Atomic)	
Load	Load Reserved	R	LR.{W|D|Q}	rd,rs1
Store	Store Conditiona	R	SC.{W|D|Q}	rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.{W|D|Q}	rd,rs1,rs2
Add	ADD	R	AMOADD.{W|D|Q}	rd,rs1,rs2
Logical	XOR	R	AMOXOR.{W|D|Q}	rd,rs1,rs2
	AND	R	AMOAND.{W|D|Q}	rd,rs1,rs2
	OR	R	AMOOR.{W|D|Q}	rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.{W|D|Q}	rd,rs1,rs2
	MAXimum	R	AMOMAX.{W|D|Q}	rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.{W|D|Q}	rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.{W|D|Q}	rd,rs1,rs2

3 Optional FP Extensions: RV32{F|D|Q}

Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)	
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
	SQuare RooT	R	FSQRT.{S|D|Q}	rd,rs1
Mul-Add	Multiply-ADD	R	FMADD.{S|D|Q}	rd,rs1,rs2,rs3
	Multiply-SUBtract	R	FMSUB.{S|D|Q}	rd,rs1,rs2,rs3
	Negative Multiply-SUBtract	R	FMNSUB.{S|D|Q}	rd,rs1,rs2,rs3
	Negative Multiply-ADD	R	FMNADD.{S|D|Q}	rd,rs1,rs2,rs3
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
Categorize	Classify Type	R	FCLASS.{S|D|Q}	rd,rs1
Move	Move from Integer	R	FMV.S.X	rd,rs1
	Move to Integer	R	FMV.X.S	rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.W	rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.WU	rd,rs1
	Convert to Int	R	FCVT.W.{S|D|Q}	rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{S|D|Q}	rd,rs1
Configuration	Read Stat	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

+ 6 for 64{F|D|Q}/ 128{F|D|Q}

Optional Compressed Instructions: RVC

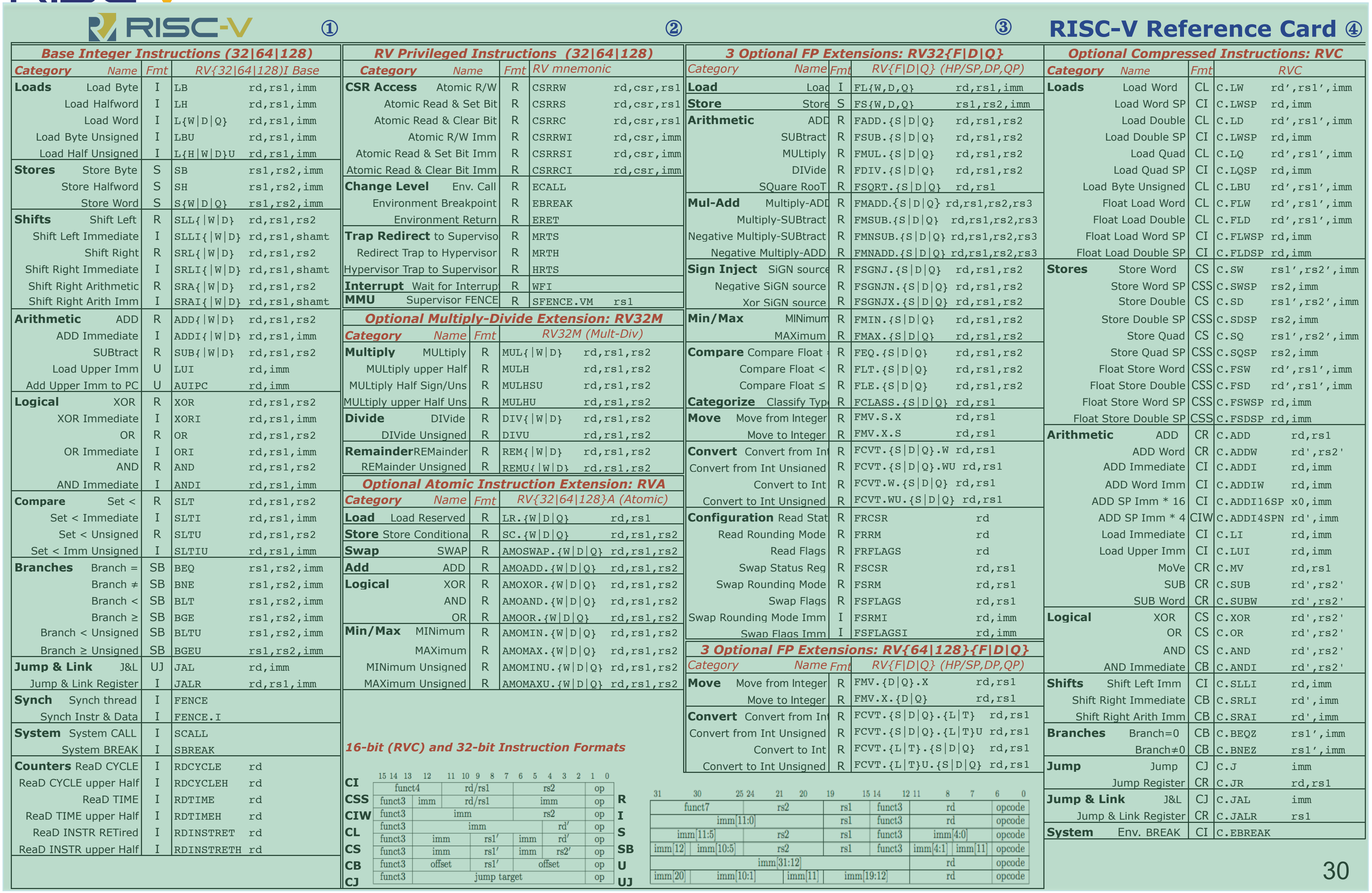
Category	Name	Fmt	RVC	
Loads	Load Word	CL	C.LW	rd′,rs1′,imm
	Load Word SP	CI	C.LWSP	rd,imm
	Load Double	CL	C.LD	rd′,rs1′,imm
	Load Double SP	CI	C.LWSP	rd,imm
	Load Quad	CL	C.LQ	rd′,rs1′,imm
	Load Quad SP	CI	C.LQSP	rd,imm
	Load Byte Unsigned	CL	C.LBU	rd′,rs1′,imm
	Float Load Word	CL	C.FLW	rd′,rs1′,imm
	Float Load Double	CL	C.FLD	rd′,rs1′,imm
	Float Load Word SP	CI	C.FLWSP	rd,imm
	Float Load Double SP	CI	C.FLDSP	rd,imm
Stores	Store Word	CS	C.SW	rs1′,rs2′,imm
	Store Word SP	CSS	C.SWSP	rs2,imm
	Store Double	CS	C.SD	rs1′,rs2′,imm
	Store Double SP	CSS	C.SDSP	rs2,imm
	Store Quad	CS	C.SQ	rs1′,rs2′,imm
	Store Quad SP	CSS	C.SQSP	rs2,imm
	Float Store Word	CSS	C.FSW	rd′,rs1′,imm
	Float Store Double	CSS	C.FSD	rd′,rs1′,imm
	Float Store Word SP	CSS	C.FSWSP	rd,imm
	Float Store Double SP	CSS	C.FSDSP	rd,imm
Arithmetic	ADD	CR	C.ADD	rd,rs1
	ADD Word	CR	C.ADDW	rd',rs2'
	ADD Immediate	CI	C.ADDI	rd,imm
	ADD Word Imm	CI	C.ADDIW	rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP	x0,imm
	ADD SP Imm * 4	CIW	C.ADDI4SPN	rd',imm
	Load Immediate	CI	C.LI	rd,imm
	Load Upper Imm	CI	C.LUI	rd,imm
	MoVe	CR	C.MV	rd,rs1
	SUB	CR	C.SUB	rd',rs2'
	SUB Word	CR	C.SUBW	rd',rs2'
Logical	XOR	CS	C.XOR	rd',rs2'
	OR	CS	C.OR	rd',rs2'
	AND	CS	C.AND	rd',rs2'
	AND Immediate	CB	C.ANDI	rd',rs2'
Shifts	Shift Left Imm	CI	C.SLLI	rd,imm
	Shift Right Immediate	CB	C.SRLI	rd',imm
	Shift Right Arith Imm	CB	C.SRAI	rd',imm
Branches	Branch=0	CB	C.BEQZ	rs1′,imm
	Branch≠0	CB	C.BNEZ	rs1′,imm
Jump	Jump	CJ	C.J	imm
	Jump Register	CR	C.JR	rd,rs1
Jump & Link	J&L	CJ	C.JAL	imm
	Jump & Link Register	CR	C.JALR	rs1
System	Env. BREAK	CI	C.EBREAK	

16-bit (RVC) and 32-bit Instruction Formats

	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CI	funct4		rd/rs1		rs2		op
CSS	funct3	imm	rd/rs1		imm		op
CIW	funct3	imm			rs2		op
CL	funct3	imm				rd′	op
	funct3	imm		rs1′	imm	rd′	op
CS	funct3	imm		rs1′	imm	rs2′	op
CB	funct3	offset		rs1′	offset		op
CJ	funct3	jump target					op

	31	30 25	24 21	20	19 15	14 12	11 8	7	6 0
R	funct7		rs2		rs1	funct3	rd		opcode
I	imm[11:0]				rs1	funct3	rd		opcode
S	imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode
SB	imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode
U	imm[31:12]						rd		opcode
UJ	imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode

RV32I / RV64I / RV128I + M, A, F, D, Q, C
RISC-V "Green Card"



RISC-V ① ② ③ **RISC-V Reference Card** ④

Base Integer Instructions (32|64|128)

Category	Name	Fmt	RV{32|64|128}I Base	
Loads	Load Byte	I	LB	rd,rs1,imm
	Load Halfword	I	LH	rd,rs1,imm
	Load Word	I	L{W|D|Q}	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm
	Load Half Unsigned	I	L{H|W|D}U	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm
	Store Halfword	S	SH	rs1,rs2,imm
	Store Word	S	S{W|D|Q}	rs1,rs2,imm
Shifts	Shift Left	R	SLL{|W|D}	rd,rs1,rs2
	Shift Left Immediate	I	SLLI{|W|D}	rd,rs1,shamt
	Shift Right	R	SRL{|W|D}	rd,rs1,rs2
	Shift Right Immediate	I	SRLI{|W|D}	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA{|W|D}	rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI{|W|D}	rd,rs1,shamt
Arithmetic	ADD	R	ADD{|W|D}	rd,rs1,rs2
	ADD Immediate	I	ADDI{|W|D}	rd,rs1,imm
	SUBtract	R	SUB{|W|D}	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	OR	rd,rs1,rs2
	OR Immediate	I	ORI	rd,rs1,imm
	AND	R	AND	rd,rs1,rs2
	AND Immediate	I	ANDI	rd,rs1,imm
Compare	Set <	R	SLT	rd,rs1,rs2
	Set < Immediate	I	SLTI	rd,rs1,imm
	Set < Unsigned	R	SLTU	rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm
Branches	Branch =	SB	BEQ	rs1,rs2,imm
	Branch ≠	SB	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	JAL	rd,imm
	Jump & Link Register	I	JALR	rd,rs1,imm
Synch	Synch thread	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

RV Privileged Instructions (32|64|128)

Category	Name	Fmt	RV mnemonic	
CSR Access	Atomic R/W	R	CSRRW	rd,csr,rs1
	Atomic Read & Set Bit	R	CSRRS	rd,csr,rs1
	Atomic Read & Clear Bit	R	CSRRC	rd,csr,rs1
	Atomic R/W Imm	R	CSRRWI	rd,csr,imm
	Atomic Read & Set Bit Imm	R	CSRRSI	rd,csr,imm
	Atomic Read & Clear Bit Imm	R	CSRRCI	rd,csr,imm
Change Level	Env. Call	R	ECALL	
	Environment Breakpoint	R	EBREAK	
	Environment Return	R	ERET	
Trap Redirect	to Supervisor	R	MRTS	
	Redirect Trap to Hypervisor	R	MRTH	
	Hypervisor Trap to Supervisor	R	HRTS	
Interrupt	Wait for Interrupt	R	WFI	
MMU	Supervisor FENCE	R	SFENCE.VM	rs1

Optional Multiply-Divide Extension: RV32M

Category	Name	Fmt	RV32M (Mult-Div)	
Multiply	MULtiply	R	MUL{|W|D}	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{|W|D}	rd,rs1,rs2
	DIVide Unsigned	R	DIVU	rd,rs1,rs2
Remainder	REMainder	R	REM{|W|D}	rd,rs1,rs2
	REMainder Unsigned	R	REMU{|W|D}	rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

Category	Name	Fmt	RV{32|64|128}A (Atomic)	
Load	Load Reserved	R	LR.{W|D|Q}	rd,rs1
Store	Store Conditional	R	SC.{W|D|Q}	rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.{W|D|Q}	rd,rs1,rs2
Add	ADD	R	AMOADD.{W|D|Q}	rd,rs1,rs2
Logical	XOR	R	AMOXOR.{W|D|Q}	rd,rs1,rs2
	AND	R	AMOAND.{W|D|Q}	rd,rs1,rs2
	OR	R	AMOOR.{W|D|Q}	rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.{W|D|Q}	rd,rs1,rs2
	MAXimum	R	AMOMAX.{W|D|Q}	rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.{W|D|Q}	rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.{W|D|Q}	rd,rs1,rs2

3 Optional FP Extensions: RV32{F|D|Q}

Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)	
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
	SQuare RooT	R	FSQRT.{S|D|Q}	rd,rs1
Mul-Add	Multiply-ADD	R	FMADD.{S|D|Q}	rd,rs1,rs2,rs3
	Multiply-SUBtract	R	FMSUB.{S|D|Q}	rd,rs1,rs2,rs3
	Negative Multiply-SUBtract	R	FMNSUB.{S|D|Q}	rd,rs1,rs2,rs3
	Negative Multiply-ADD	R	FMNADD.{S|D|Q}	rd,rs1,rs2,rs3
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
Categorize	Classify Type	R	FCLASS.{S|D|Q}	rd,rs1
Move	Move from Integer	R	FMV.S.X	rd,rs1
	Move to Integer	R	FMV.X.S	rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.W	rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.WU	rd,rs1
	Convert to Int	R	FCVT.W.{S|D|Q}	rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{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

3 Optional FP Extensions: RV{64|128}{F|D|Q}

Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)	
Move	Move from Integer	R	FMV.{D|Q}.X	rd,rs1
	Move to Integer	R	FMV.X.{D|Q}	rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.{L|T}	rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.{L|T}U	rd,rs1
	Convert to Int	R	FCVT.{L|T}.{S|D|Q}	rd,rs1
	Convert to Int Unsigned	R	FCVT.{L|T}U.{S|D|Q}	rd,rs1

Optional Compressed Instructions: RVC

Category	Name	Fmt	RVC	
Loads	Load Word	CL	C.LW	rd′,rs1′,imm
	Load Word SP	CI	C.LWSP	rd,imm
	Load Double	CL	C.LD	rd′,rs1′,imm
	Load Double SP	CI	C.LWSP	rd,imm
	Load Quad	CL	C.LQ	rd′,rs1′,imm
	Load Quad SP	CI	C.LQSP	rd,imm
	Load Byte Unsigned	CL	C.LBU	rd′,rs1′,imm
	Float Load Word	CL	C.FLW	rd′,rs1′,imm
	Float Load Double	CL	C.FLD	rd′,rs1′,imm
	Float Load Word SP	CI	C.FLWSP	rd,imm
	Float Load Double SP	CI	C.FLDSP	rd,imm
Stores	Store Word	CS	C.SW	rs1′,rs2′,imm
	Store Word SP	CSS	C.SWSP	rs2,imm
	Store Double	CS	C.SD	rs1′,rs2′,imm
	Store Double SP	CSS	C.SDSP	rs2,imm
	Store Quad	CS	C.SQ	rs1′,rs2′,imm
	Store Quad SP	CSS	C.SQSP	rs2,imm
	Float Store Word	CSS	C.FSW	rd′,rs1′,imm
	Float Store Double	CSS	C.FSD	rd′,rs1′,imm
	Float Store Word SP	CSS	C.FSWSP	rd,imm
	Float Store Double SP	CSS	C.FSDSP	rd,imm
Arithmetic	ADD	CR	C.ADD	rd,rs1
	ADD Word	CR	C.ADDW	rd′,rs2′
	ADD Immediate	CI	C.ADDI	rd,imm
	ADD Word Imm	CI	C.ADDIW	rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP	x0,imm
	ADD SP Imm * 4	CIW	C.ADDI4SPN	rd′,imm
	Load Immediate	CI	C.LI	rd,imm
	Load Upper Imm	CI	C.LUI	rd,imm
	MoVe	CR	C.MV	rd,rs1
	SUB	CR	C.SUB	rd′,rs2′
	SUB Word	CR	C.SUBW	rd′,rs2′
Logical	XOR	CS	C.XOR	rd′,rs2′
	OR	CS	C.OR	rd′,rs2′
	AND	CS	C.AND	rd′,rs2′
	AND Immediate	CB	C.ANDI	rd′,rs2′
Shifts	Shift Left Imm	CI	C.SLLI	rd,imm
	Shift Right Immediate	CB	C.SRLI	rd′,imm
	Shift Right Arith Imm	CB	C.SRAI	rd′,imm
Branches	Branch=0	CB	C.BEQZ	rs1′,imm
	Branch≠0	CB	C.BNEZ	rs1′,imm
Jump	Jump	CJ	C.J	imm
	Jump Register	CR	C.JR	rd,rs1
Jump & Link	J&L	CJ	C.JAL	imm
	Jump & Link Register	CR	C.JALR	rs1
System	Env. BREAK	CI	C.EBREAK	

16-bit (RVC) and 32-bit Instruction Formats

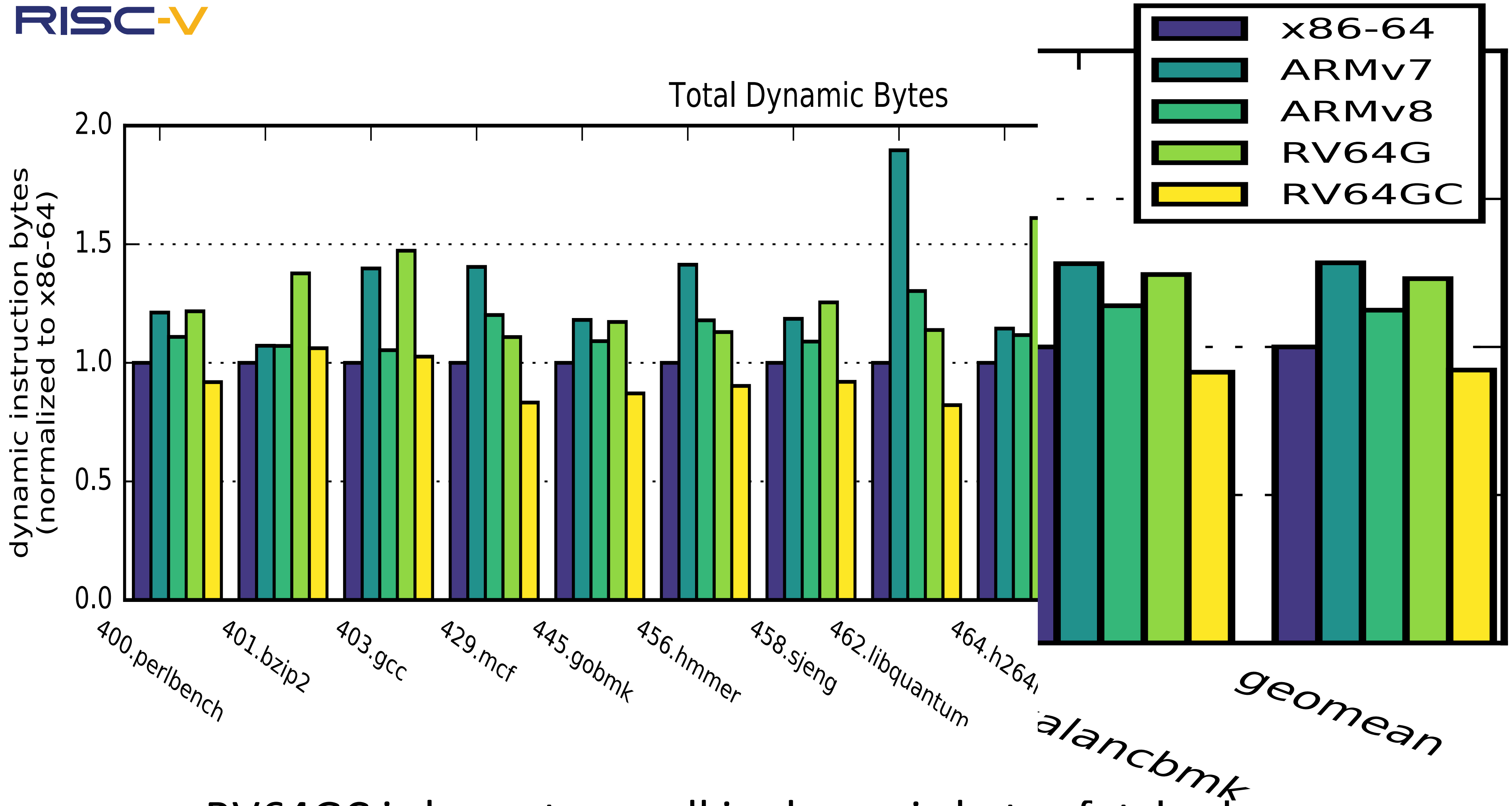
	15 14 13	12	11 10 9 8 7		6 5 4 3 2	1 0
CI	funct4		rd/rs1		rs2	op
CSS	funct3	imm	rd/rs1		imm	op
CIW	funct3	imm			rs2	op
CL	funct3	imm			rd′	op
CS	funct3	imm	rs1′	imm	rd′	op
CS	funct3	imm	rs1′	imm	rs2′	op
CB	funct3	offset	rs1′	offset		op
CJ	funct3	jump target				op

	31	30 25	24 21 20	19 15	14 12	11 8 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1] imm[11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20]	imm[10:1]	imm[11]	imm[19:12]		rd	opcode

Simplicity breeds Contempt

- How can simple ISA compete with industry monsters?
- How do measure ISA quality?
  - Static code bytes for program
  - Dynamic code bytes fetched for execution
  - Microarchitectural work generated for execution

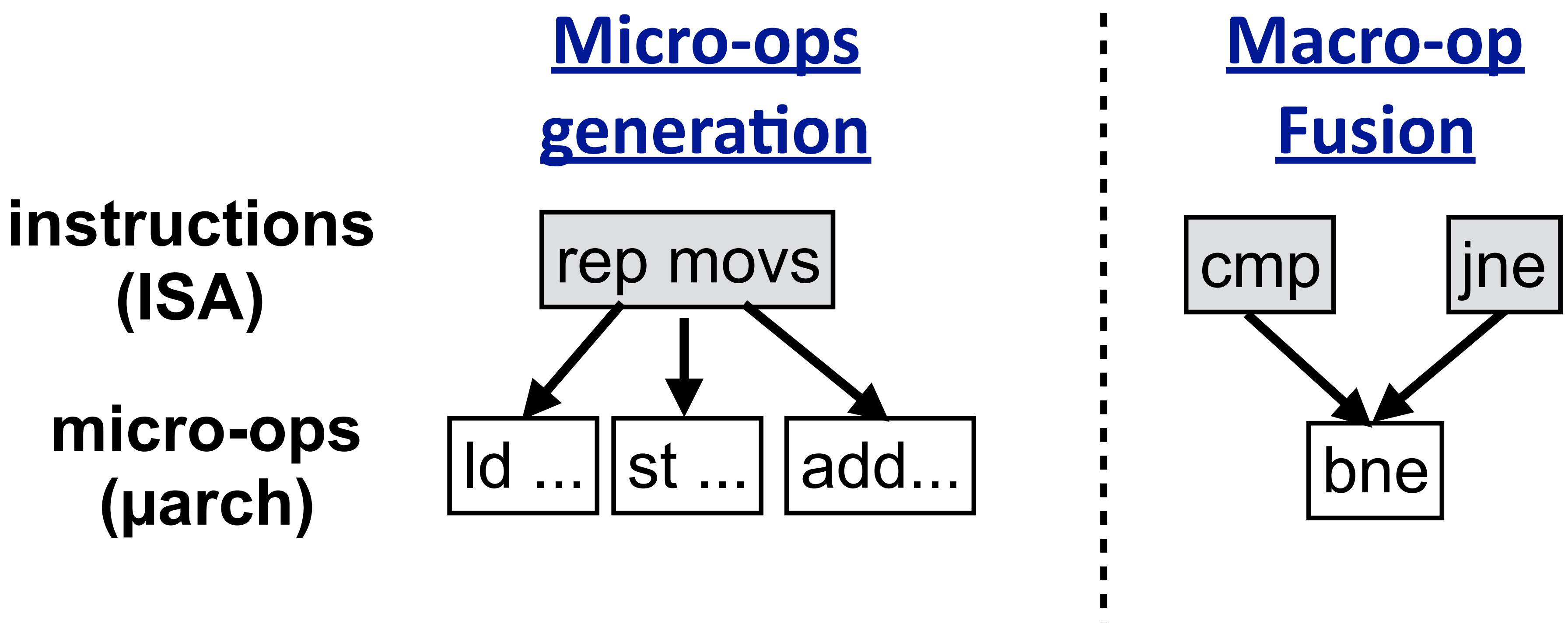
Dynamic Bytes Fetched



- RV64GC is lowest overall in dynamic bytes fetched
  - Despite current lack of support for vector operations

Converting Instructions to Microops

Microops are measure of microarchitectural work performed



Multiple microinstructions from one macroinstruction
Or one microinstruction from multiple macroinstructions

RISC-V Macro-Op Fusion Examples

- "Load effective address LEA" &(array[offset])

```
slli rd, rs1, {1,2,3}
add rd, rd, rs2
```

- "indexed load" M[rs1+rs2]

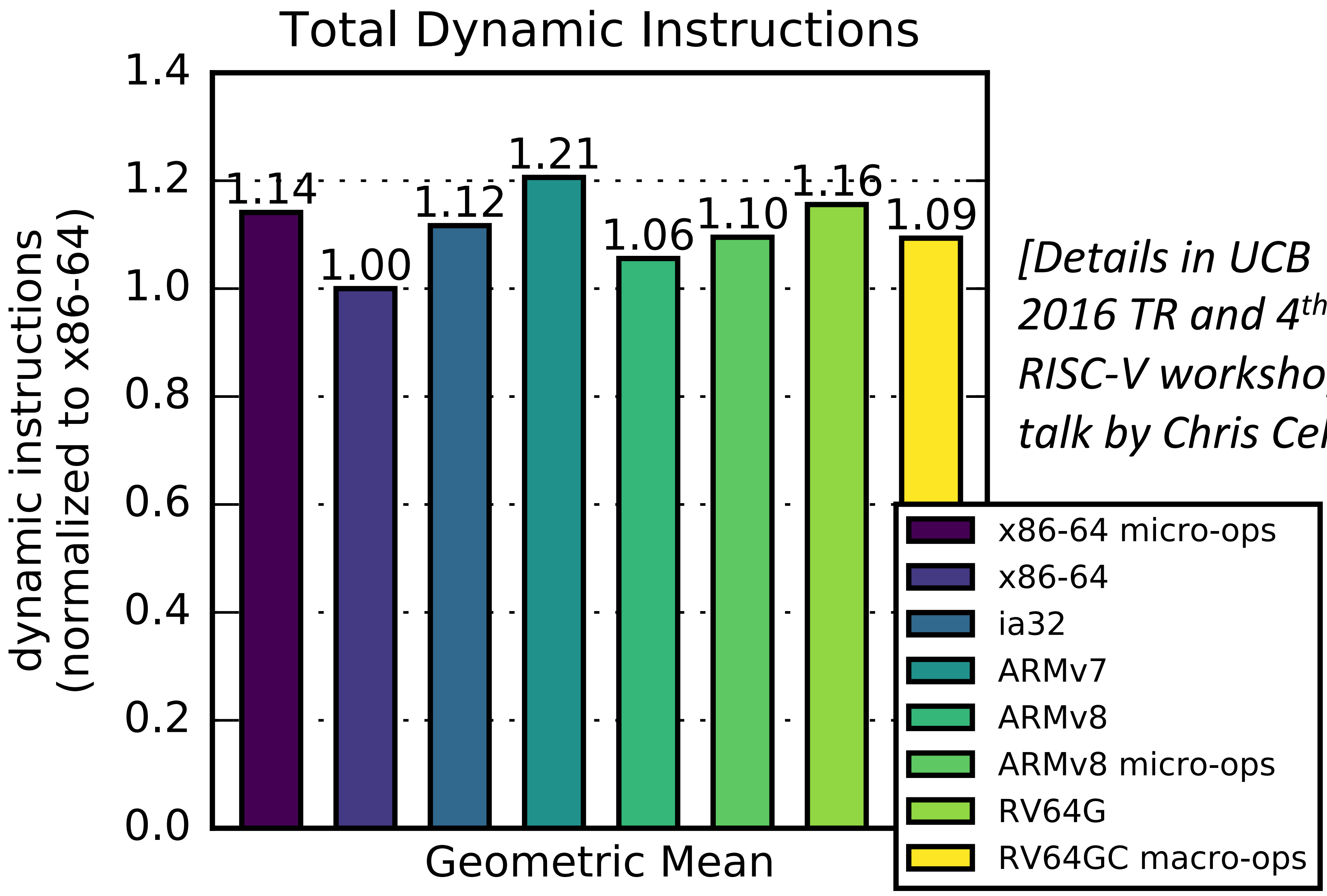
```
add rd, rs1, rs2
ld rd, 0(rd)
```

- "clear upper word" // rd = rs1 & 0xffff_ffff

```
slli rd, rs1, 32
srli rd, rd, 32
```

- Can all be fused simply in decode stage
  - Many are expressible with 2-byte compressed instructions, so effectively just adds new 4-byte instructions
- RISC-V approach: prefer macroop fusion to larger ISA

RISC-V Competitive µarch Effort after Fusion

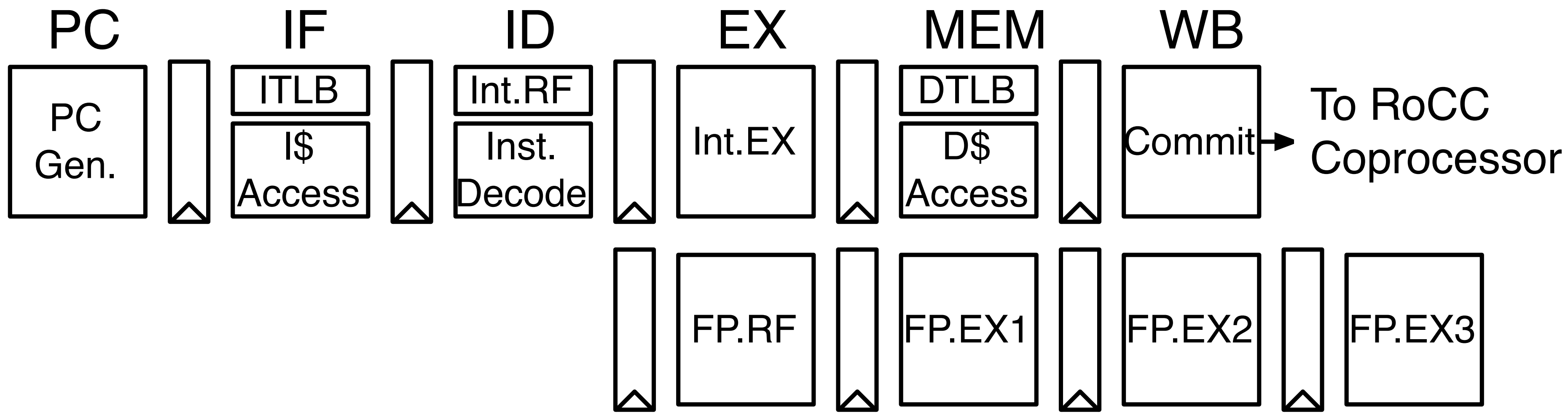


[Details in UCB 2016 TR and 4th RISC-V workshop talk by Chris Celio]

UC Berkeley RISC-V Core Generators

- **Rocket:** Family of In-order Cores
  - Supports 32-bit and 64-bit single-issue only
  - Dual-issue soon
  - Similar in spirit to ARM Cortex M-series and A5/A7/A53
- **BOOM:** Family of Out-of-Order Cores
  - Supports 64-bit single-, dual-, quad-issue
  - Similar in spirit to ARM Cortex A9/A15/A57

RISC-V Rocket In-Order Core



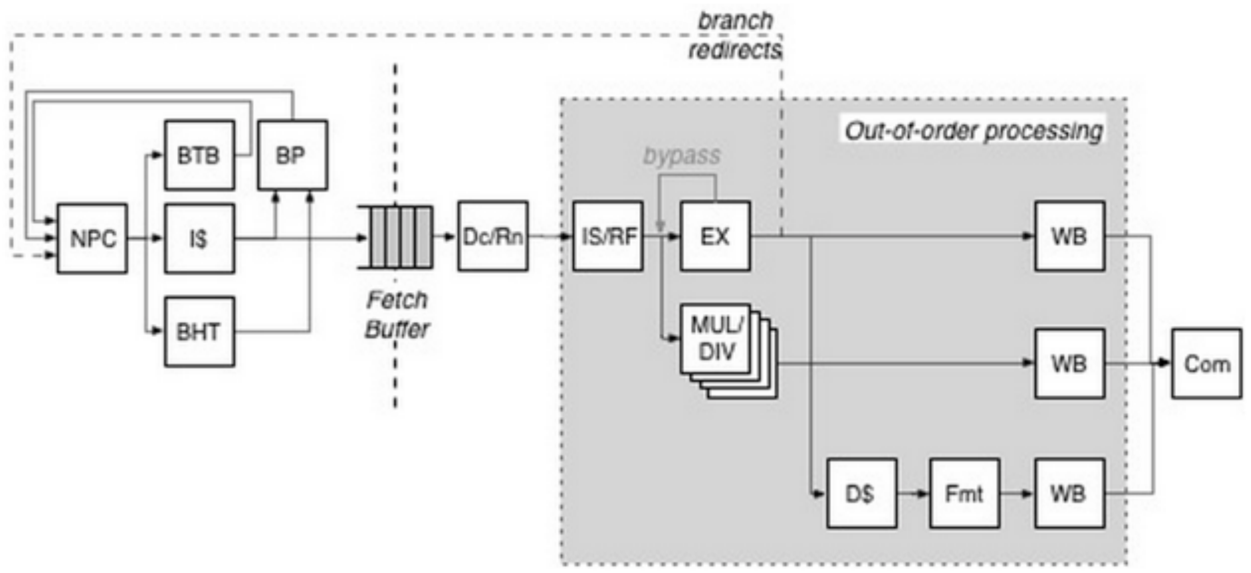
- 64-bit 5-stage single-issue in-order pipeline
- Design minimizes impact of long clock-to-output delays of compiler-generated RAMs
- 64-entry BTB, 256-entry BHT, 2-entry RAS
- MMU supports page-based virtual memory
- IEEE 754-2008-compliant FPU
  - Supports SP, DP fused multiply-adds with hardware support for subnormals
- Currently working on dual-issue in-order Rocket

ARM Cortex-A5 vs. RISC-V Rocket

Category	ARM Cortex-A5	RISC-V Rocket
ISA	32-bit ARM v7	64-bit RISC-V v2
Architecture	Single-Issue In-Order	Single-Issue In-Order 5-stage
Performance	1.57 DMIPS/MHz	1.72 DMIPS/MHz
Process	TSMC 40GPLUS	TSMC 40GPLUS
Area w/o Caches	0.27 mm^2	0.14 mm^2
Area with 16K Caches	0.53 mm^2	0.39 mm^2
Area Efficiency	2.96 DMIPS/MHz/mm^2	4.41 DMIPS/MHz/mm^2
Frequency	>1GHz	>1GHz
Dynamic Power	<0.08 mW/MHz	0.034 mW/MHz

- PPA reporting conditions
  - 85% utilization, use Dhrystone for benchmark, frequency/power at TT 0.9V 25C, all regular VT transistors
- 10% higher in DMIPS/MHz, 49% more area-efficient

RISC-V BOOM Out-of-Order Core



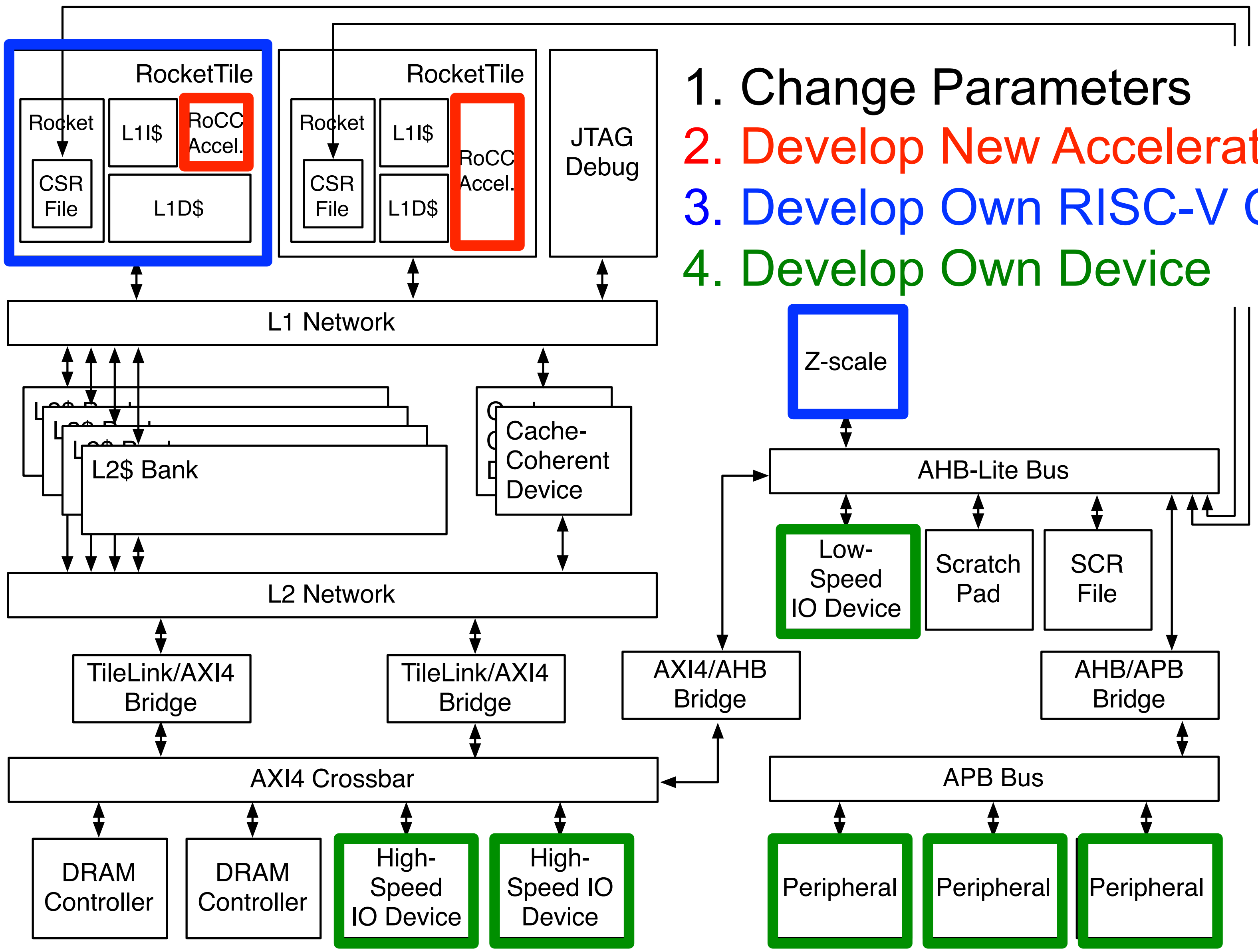
- Baseline Design
  - Superscalar (parameterizable widths)
  - Full branch speculation (BTB/BHT/RAS)
  - Load/store queue with store ordering
    - Loads execute fully OoO wrt stores, other loads
    - Store-data forwards to loads
- Estimated 2GHz+ in 45nm (<30 FO4)

ARM Cortex-A9 vs. RISC-V BOOM

Category	ARM Cortex-A9	RISC-V BOOM-2w
ISA	32-bit ARM v7	64-bit RISC-V v2 (RV64G)
Architecture	2 wide, 3+1 issue Out-of-Order 8-stage	2 wide, 3 issue Out-of-Order 6-stage
Performance	3.59 CoreMarks/MHz	3.91 CoreMarks/MHz
Process	TSMC 40GPLUS	TSMC 40GPLUS
Area with 32K caches	2.5 mm^2	1.00 mm^2
Area efficiency	1.4 CoreMarks/MHz/mm^2	3.9 CoreMarks/MHz/mm^2
Frequency	1.4 GHz	1.5 GHz

Caveats: A9 includes NEON
BOOM is 64-bit, has IEEE-2008 fused mul-add

Rocket Chip Generator

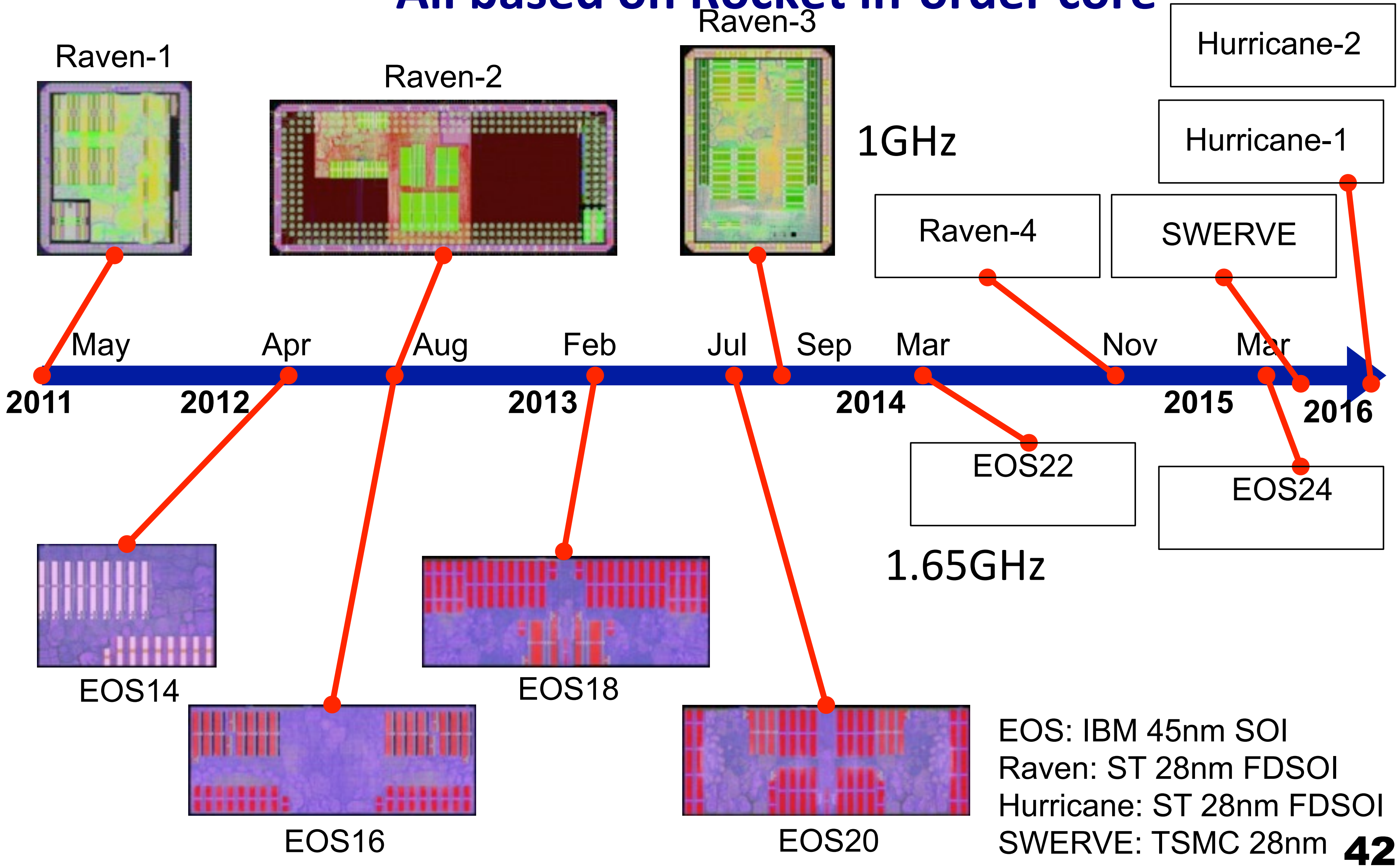


1. Change Parameters
2. Develop New Accelerators
3. Develop Own RISC-V Core
4. Develop Own Device

UC Berkeley RISC-V Cores:

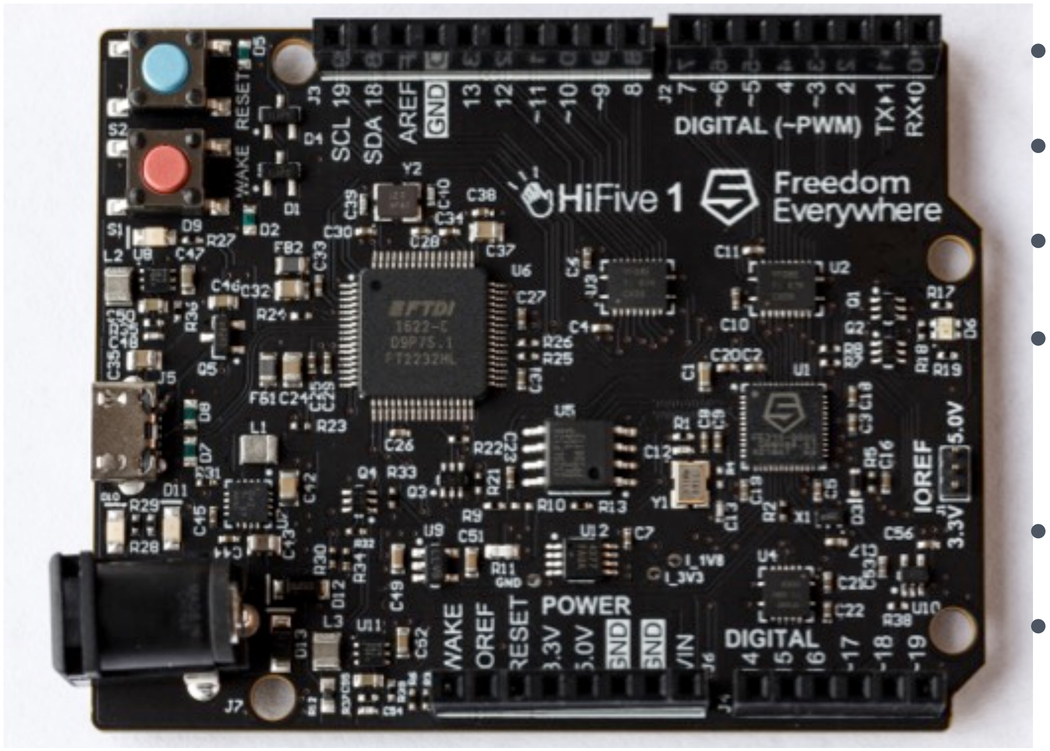
Seven 28nm & Six 45nm RISC-V Chips Tapeouts

All based on Rocket in-order core



EOS: IBM 45nm SOI
Raven: ST 28nm FDSOI
Hurricane: ST 28nm FDSOI
SWERVE: TSMC 28nm

SiFive HiFive 1



- Open-Source RTL
- Arduino-Compatible
- Freedom E SDK
- Arduino IDE Environment

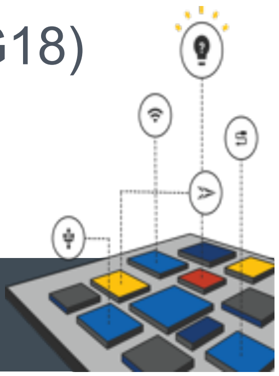
- Available for sale now!
- $59

https://www.crowdsupply.com/sifive/hifive1

RISC-V is GREAT at Perf and Power

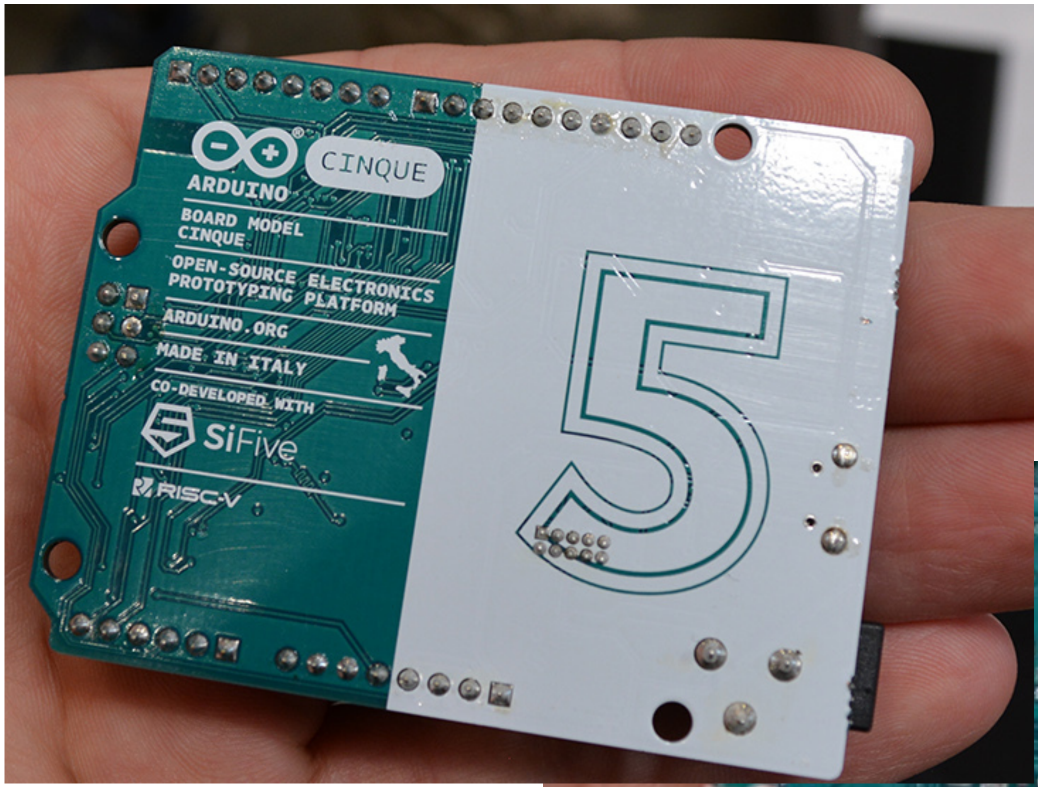
Microcontroller	CPU Core	CPU ISA	CPU Speed	DMIPs/MHz	Total Dhrystones	DMIPs/mW
Intel Curie Module	Intel Quark SE	x86	32 MHz	1.3	41.6	0.35
ATmega328P	AVR	AVR (8-bit)	16 MHz	0.30	5	0.10
ATSAMD21G18	ARM Cortex M0+	ARMv6-M	48 MHz	0.93	44.64	
Nordic NRF51	ARM Cortex M0	ARMv6-M	16 MHz	0.93	14.88	1.88
Freedom E310	SiFive E31	RISC-V RV32IMAC	200 MHz 320 MHz (max)	1.61	320.39	3.16

- 10x Faster Clock than Intel's Arduino 101 uController
- 11x More Dhrystones than ARM's Arduino Zero (ATSAMD21G18)
- 9x More Power Efficient than Intel Quark
- 2x More Power Efficient than ARM Cortex M0+

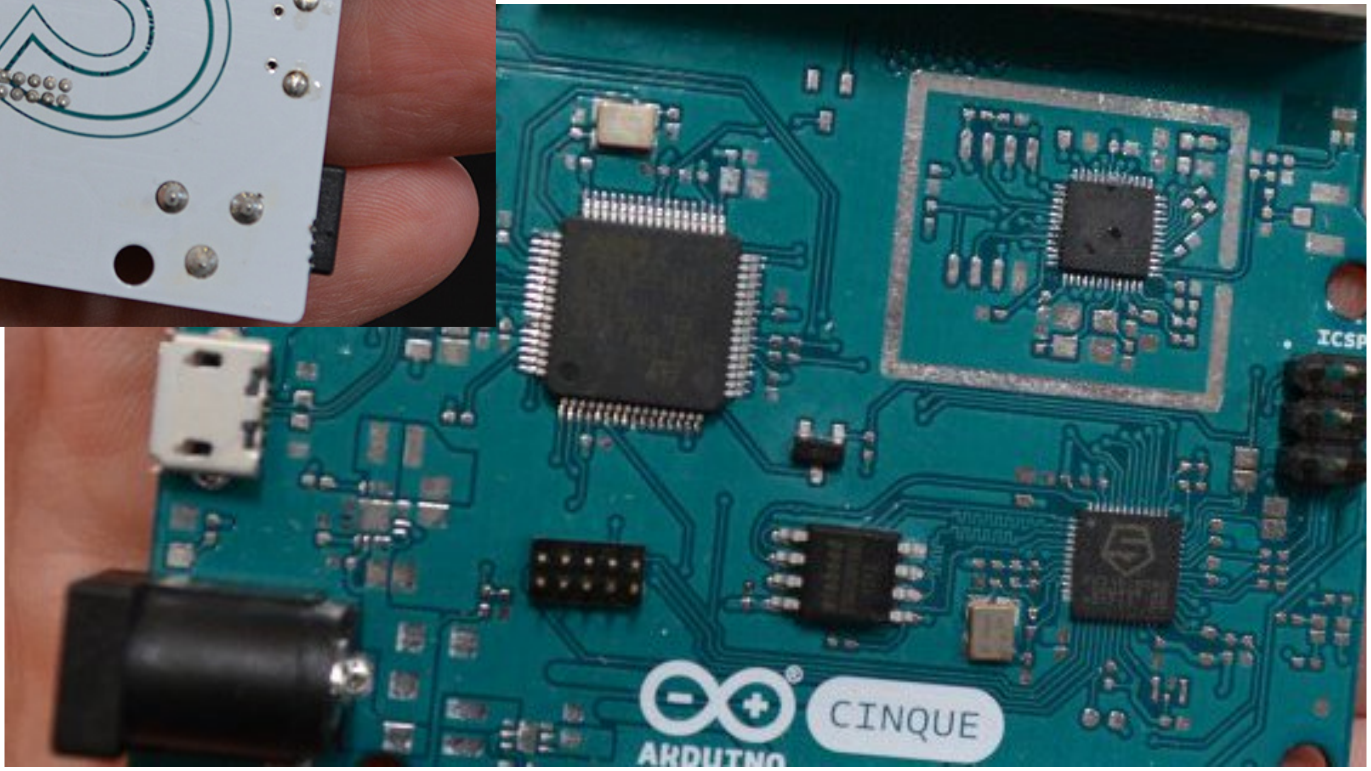


SiFive

Hot off the Press: Arduino Cinque



Announced at Bay Area Maker Faire , May 20, 2017



RISC-V Outside Berkeley

- Adopted as “standard ISA” for **India**
  - IIT-Madras $90M funding to build 6 different open-source RISC-V cores, from microcontrollers to servers
  - C-DAC $45M funding to build 2GHz quad-core
- **NVIDIA** selected RISC-V for on-chip microcontrollers
- **LowRISC** project based in Cambridge, UK producing open-source RISC-V Rocket-based SoCs
  - Led by Raspberry Pi co-founder, privately funded
- **Andes** announced 64-bit core based on RISC-V
  - Other soft core conversions to come
- **SiFive, Bluespec**, **Codasip**, **Syntacore**, + others have commercial soft cores available now
- Multiple commercial silicon implementations should be for sale later this year
- Many commercial big chip projects using small RISC-V cores
- Multiple commercial groups developing server-class cores

Software Ecosystem

- GCC, binutils upstreamed as of GCC 7.1
- Linux, glibc, gdb upstream in progress
- FreeBSD upstreamed
- LLVM upstream in progress
- QEMU user-mode upstream in progress, system-mode soon
- Zephyr, FreeRTOS ports
- Yocto embedded Linux distribution generator
- Jikes JVM port completed
- OpenJDK ported, HotSpot JVM JIT in progress
- Coreboot, Go ports by Google
- OpenOCD
- Gem5 port

RISC-V Foundation

- Mission statement
  - “to standardize, protect, and promote the free and open RISC-V instruction set architecture and its hardware and software ecosystem for use in all computing devices.”
- Established as a 501(c)(6) non-profit corporation on August 3, 2015
- Rick O’Connor recruited as Executive Director
- First year, 41+ “founding” members.
- Now over 60 members.
- Additional members welcome

RISC-V

Foundation Members (60+)

Platinum:

Berkeley Architecture Research
bluespec
cortus
DRAPER
IBM
Google
Mellanox TECHNOLOGIES
SAMSUNG
Microsemi
QUALCOMM
Microsoft
SiFive
NVIDIA
HUAWEI
Rambus Cryptography Research
WD Western Digital

Gold, Silver, Auditors:

AMD
ANDES TECHNOLOGY
antmicro EMBEDDED SYSTEMS
Blockstream
ICT 中科院计算所 INSTITUTE OF COMPUTING TECHNOLOGY, CAS
GRAY RESEARCH
codasip
Esperanto Technologies
ESPRESSIF
IDT
INTRINSIX
ETH zürich
LATTICE SEMICONDUCTOR
lowRISC
MINIMA PROCESSOR
MIT CSAIL
ROA LOGIC
Rumble
Developm
Technolution
ent
runtime.io
MEDIATEK
Sur Technology
Syntacore Custom cores and tools
VectorBlox embedded supercomputing

Learning More about RISC-V

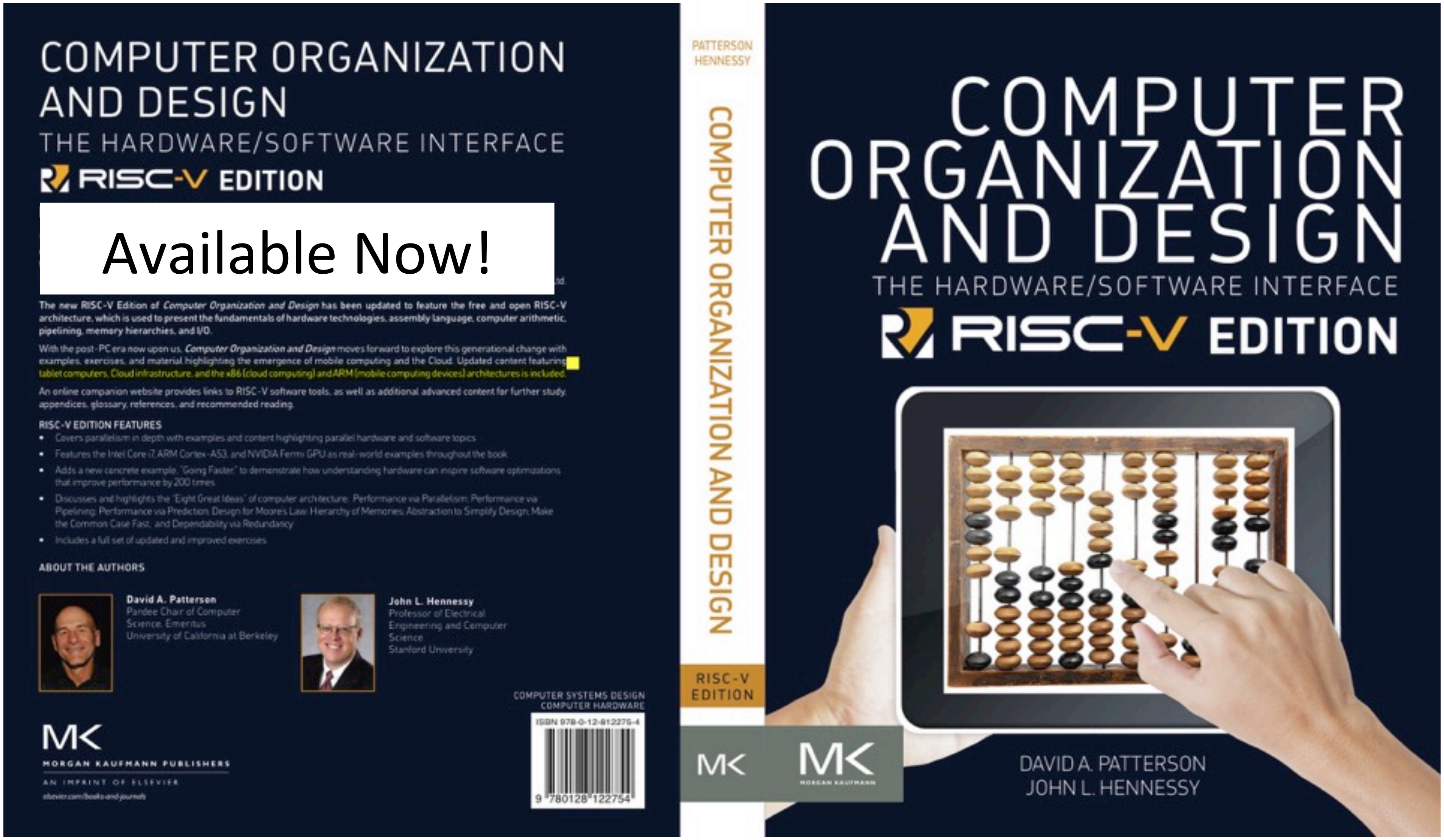
- Website **`riscv.org`** is primary resource
- Sign up for mailing lists/twitter at **`riscv.org`**
- 1st RISC-V workshop January 14-15, 2015, Monterey
- 2nd RISC-V workshop June 29-30, 2015, UC Berkeley
- 3rd RISC-V workshop January 5-6, 2016, Oracle, CA
- 4th RISC-V Workshop July 12-13, 2016, MIT, MA
- 5th RISC-V Workshop, November 29-30, 2016, Google, Mountain View, CA
- 6th RISC-V Workshop, July 8-11, 2017, Shanghai, China
- All workshops sold out!
- Material from all workshops at **`riscv.org`**

6th RISC-V Workshop May 2017

- Upcoming 7th RISC-V workshop, November, Milpitas, CA, hosted by Western Digital

RISC-V

RISC-V in Education, Patterson/Hennessy books



COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
RISC-V EDITION

Available Now!

The new RISC-V Edition of *Computer Organization and Design* has been updated to feature the free and open RISC-V architecture, which is used to present the fundamentals of hardware technologies, assembly language, computer arithmetic, pipelining, memory hierarchies, and I/O.

With the post-PC era now upon us, *Computer Organization and Design* moves forward to explore this generational change with examples, exercises, and material highlighting the emergence of mobile computing and the Cloud. Updated content featuring tablet computers, Cloud infrastructure, and the x86 (cloud computing) and ARM (mobile computing devices) architectures is included.

An online companion website provides links to RISC-V software tools, as well as additional advanced content for further study, appendices, glossary, references, and recommended reading.

RISC-V EDITION FEATURES

- Covers parallelism in depth with examples and content highlighting parallel hardware and software topics
- Features the Intel Core i7, ARM Cortex-A53, and NVIDIA Fermi GPU as real-world examples throughout the book
- Adds a new concrete example, "Going Faster," to demonstrate how understanding hardware can inspire software optimizations that improve performance by 200 times
- Discusses and highlights the "Eight Great Ideas" of computer architecture: Performance via Parallelism; Performance via Pipelining; Performance via Prediction; Design for Moore's Law; Hierarchy of Memories; Abstraction to Simplify Design; Make the Common Case Fast; and Dependability via Redundancy
- Includes a full set of updated and improved exercises

ABOUT THE AUTHORS

David A. Patterson
Pardee Chair of Computer Science, Emeritus
University of California at Berkeley

John L. Hennessy
Professor of Electrical Engineering and Computer Science
Stanford University

COMPUTER SYSTEMS DESIGN
COMPUTER HARDWARE

ISBN 978-0-12-812275-4

9 780128 122754

MK
MORGAN KAUFMANN PUBLISHERS
AN IMPRINT OF ELSEVIER
elsevier.com/books-and-journals

PATTERSON
HENNESSY
COMPUTER ORGANIZATION AND DESIGN
RISC-V EDITION
MK

COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
RISC-V EDITION

MK
MORGAN KAUFMANN

DAVID A. PATTERSON
JOHN L. HENNESSY

Graduate book 6th edition will also use RISC-V, ready in December

Why use a free and open ISA like RISC-V?

- Quality
  - Better/equal to others on main ISA metrics while simpler
- Competition
  - Pick ISA first, then pick a vendor (or roll your own)
- Flexibility
  - Design own configuration, or add own extensions
- Stability
  - No 3rd party's business decision takes RISC-V away
- Security
  - Simpler to verify, can build own cores
- Education
  - Easy to learn, and will be first (and only?) ISA for many
- Community
  - It's where the action is

RISC-V Research Project Sponsors

- DoE Isis Project
- DARPA PERFECT program
- DARPA POEM program (Si photonics)
- STARnet Center for Future Architectures (C-FAR)
- Lawrence Berkeley National Laboratory
- Industrial sponsors (ParLab + ASPIRE)
  - Intel, Google, HPE, Huawei, LG, NEC, Microsoft, Nokia, NVIDIA, Oracle, Samsung

Modest RISC-V Project Goal

Become the industry-standard ISA for all computing devices