Computer Architecture Research with RISC-V

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

CARRV, Boston, MA
October 14, 2017

Only Two Big Mistakes Possible when Picking Research ISA

- Design your own
- Use someone else's

Promise of using commercially popular ISAs for research

- Ported applications/workloads to study
- Standard software stacks (compilers, OS)
- Real commercial hardware to experiment with
- Real commercial hardware to validate models with
- Existing implementations to study / modify
- Industry is more interested in your results

Types of projects and standard ISAs used by me or my group in last 30 years

- Experiments on real hardware platforms:
  - **Transputer** arrays, **SPARC** workstations, **MIPS** workstations, **POWER** workstations, **ARMv7** handhelds, **x86** desktops/ servers
- Research chips built around modified **MIPS** ISA:
  - T0, IRAM, STC1, Scale, Maven
- FPGA prototypes/simulations using various ISAs:
  - RAMP Blue (modified **Microblaze**), RAMP Gold/ DIABLO (**SPARC v8**)
- Experiments using software architectural simulators:
  - SimpleScalar (**PISA**), SMTsim (**Alpha**), Simics (**SPARC,x86**), Bochs (**x86**), MARSS (**x86**), Gem5(**SPARC**), PIN (**Itanium, x86**), …
- And of course, other groups used some others too.

Realities of using standard ISAs

- Everything only works if you don't change anything
  - Stock binary applications
  - Stock libraries
  - Stock compiler
  - Stock OS
  - Stock hardware implementation
- Add a new instruction, get a new non-standard ISA!
  - Need source code for the apps and recompile
    - Impossible for most real interesting applications
  - Need a new compiler?
    - Large amount of work unless just an intrinsic
- Change ISA or even just microarchitecture, need a new implementation
    - Vendors won't give you theirs to modify

Building a new implementation of standard ISA

- To really get advantage of existing software, need to build whole stack
- Interesting apps use large standard libraries
- Large standard libraries depend on standard OS
- Standard OS depends on standard privileged hardware architecture
- Need to implement all of complex ISA including privileged architecture (or fake it)

- There was an old woman who swallowed a fly…

ISA Vitality Chart

- Officially dead:
  - Transputer
  - Alpha
- Niche
  - Microblaze
- Not officially dead, but starting to smell bad:
  - Itanium
  - MIPS
  - SPARC
  - POWER
- Alive and well:
  - AMD64 (x86)
  - ARM Thumb/Thumb2/v7/v8

Surviving Popular ISAs are too complex

- No redeeming technical reasons for complexity
- Too much has to be implemented just to try to reuse software
- In particular, make poor base ISA for accelerators
  - Little opcode space left, or already at 15-byte instructions
  - Supporting base ISA too much area/power/design time

Surviving popular proprietary ISAs forbid sharing RTL implementations

- **Claim**: Without shared RTL implementations, arch community cannot make reproducible, scientifically validated progress on processor design

- **Therefore**: Community cannot use popular proprietary ISAs to make real progress in general-purpose processor design

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?

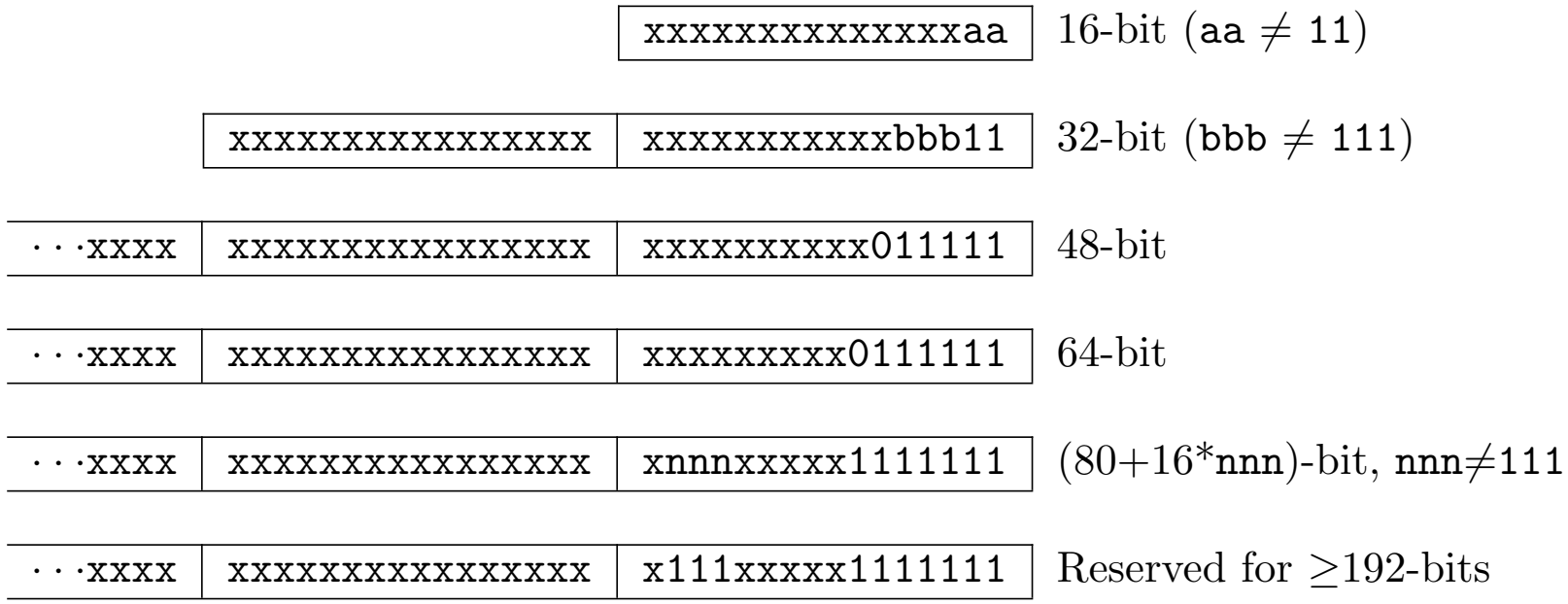
What's Different about RISC-V?

- *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

Variable-Length Encoding

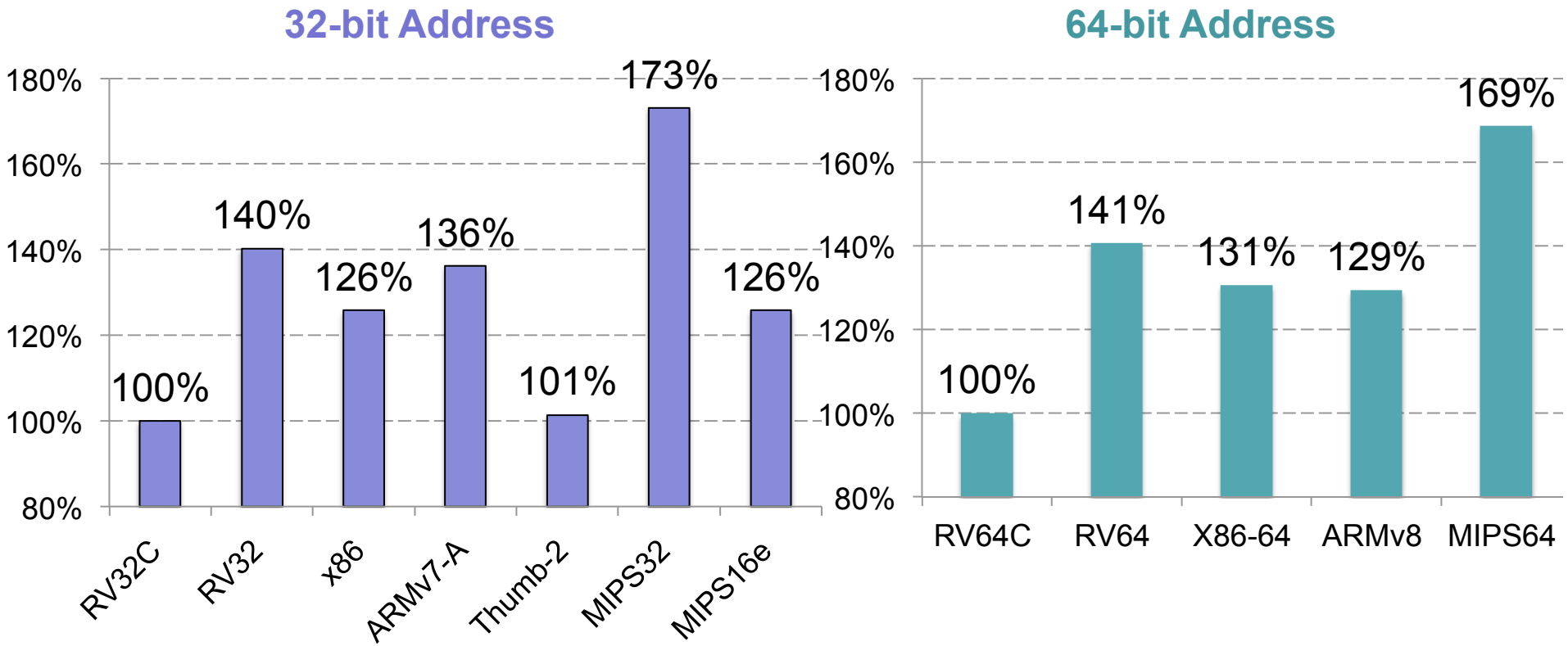


- Extensions can use any multiple of 16 bits as instruction length
- Branches/Jumps target 16-bit boundaries even in fixed 32-bit base
  - Consumes 1 extra bit of jump/branch address

"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
  - Some 2-address forms with all 32 registers
  - More 2-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

RV32I

RISC-V

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

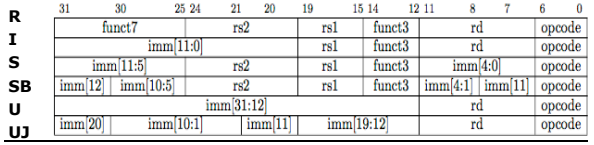
+ 8 for M

+ 34
for F, D, Q

+ 46 for C

+ 11 for A

32-bit Instruction Formats



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

RISC-V

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 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 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}

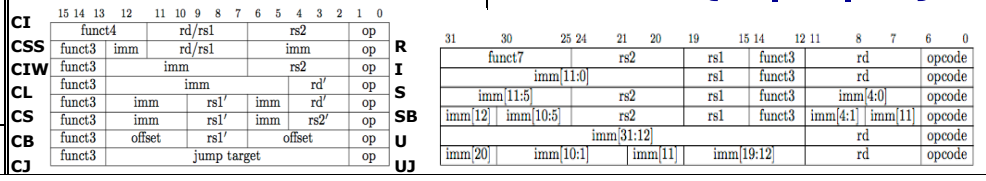
Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)	
Load	Load	I	FL{W,D,Q}	rd,rs1,imm
Store	Store	S	FS{W,D,Q}	rs1,rs2,imm
Arithmetic	ADD	R	FADD.{S|D|Q}	rd,rs1,rs2
	SUBtract	R	FSUB.{S|D|Q}	rd,rs1,rs2
	MULtiply	R	FMUL.{S|D|Q}	rd,rs1,rs2
	DIVide	R	FDIV.{S|D|Q}	rd,rs1,rs2
	SQuare RooT	R	FSQRT.{S|D|Q}	rd,rs1
Mul-Add	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 Typ	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



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 Superviso	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 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

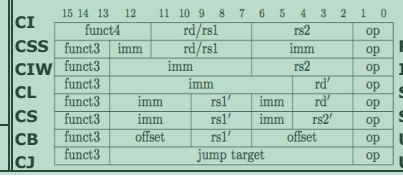
3 Optional FP Extensions: RV{64|128}{F|D|Q}

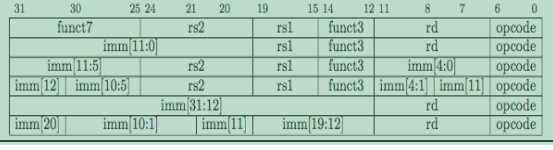
Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)	
Move	Move from Integer	R	FMV.{D|Q}.X	rd,rs1
	Move to Integer	R	FMV.X.{D|Q}	rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.{L|T}	rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.{L|T}U	rd,rs1
	Convert to Int	R	FCVT.{L|T}.{S|D|Q}	rd,rs1
	Convert to Int Unsigned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

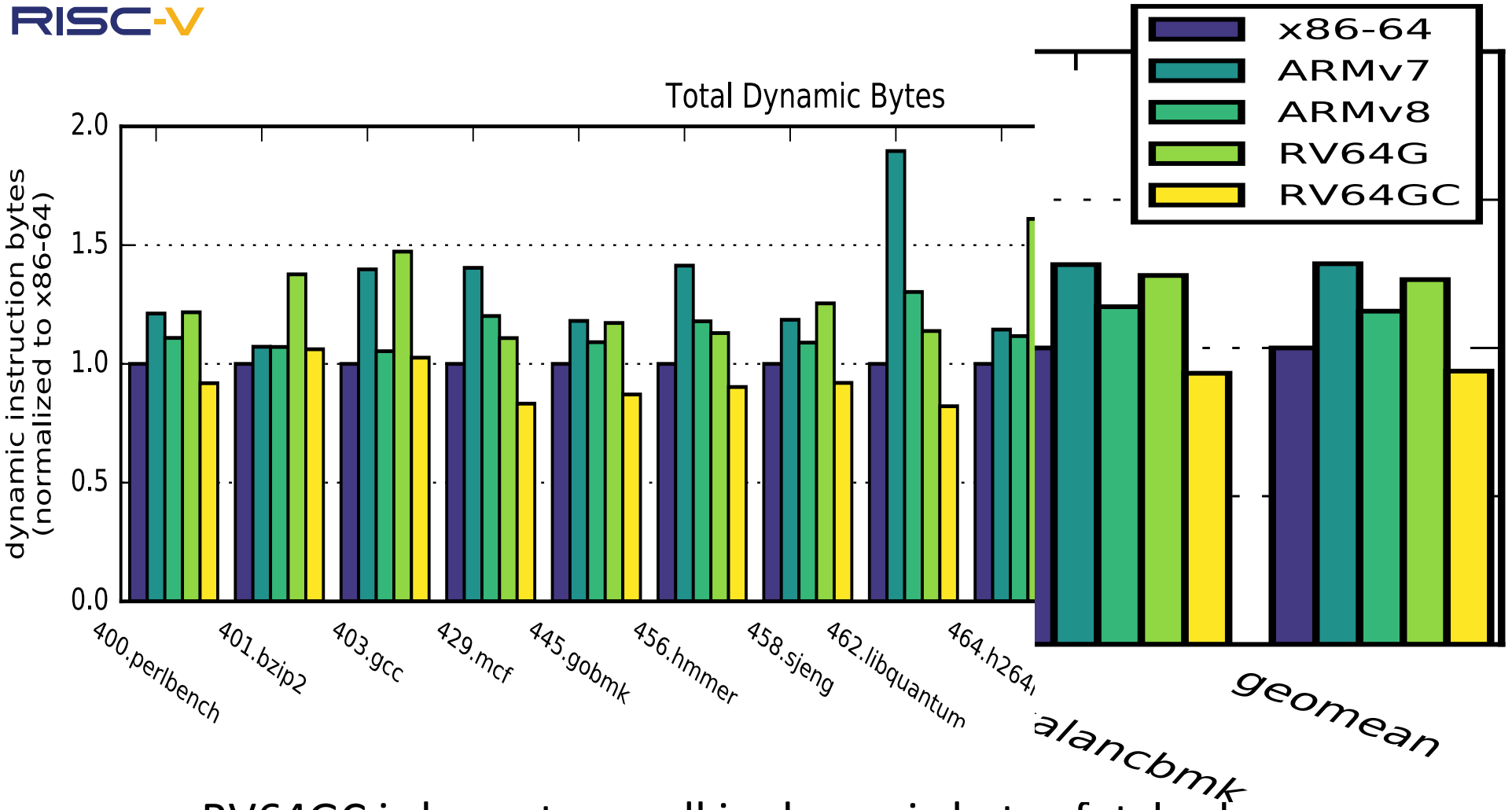




Simplicity breeds Contempt

- 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

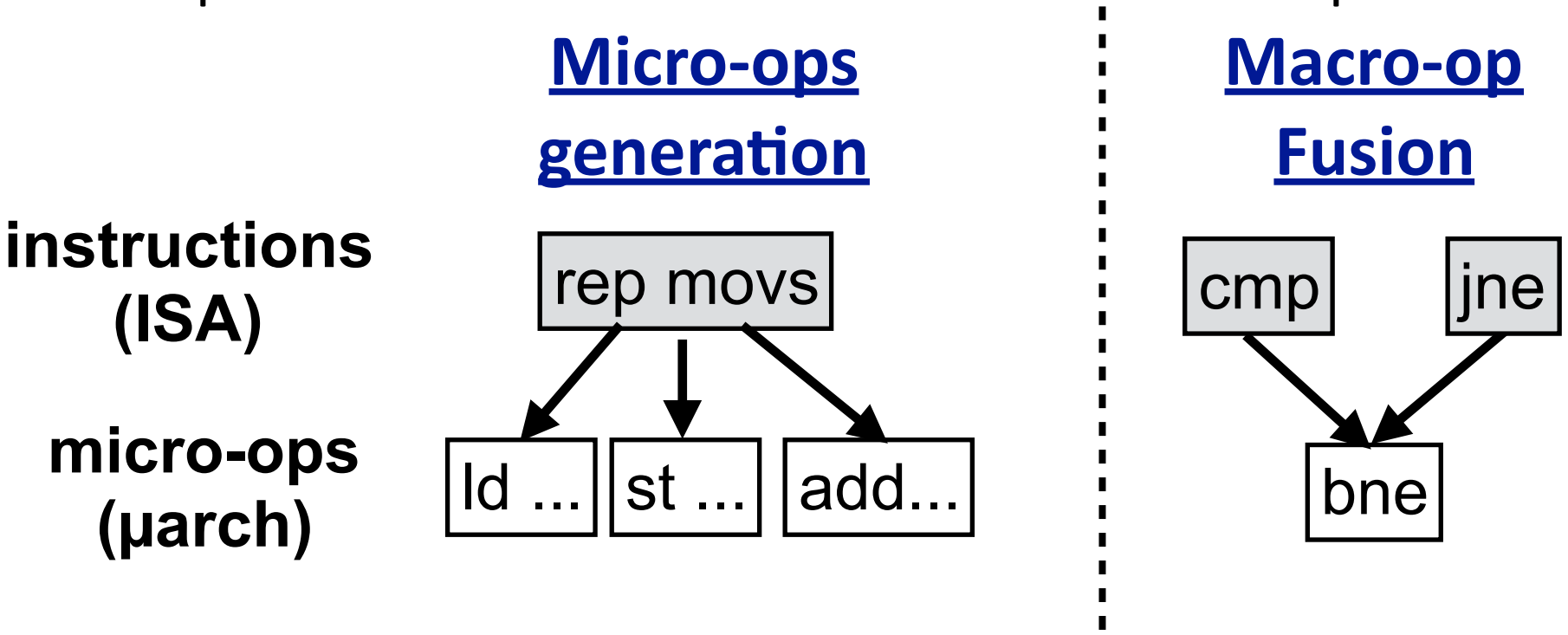
Dynamic Bytes Fetched



- 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]

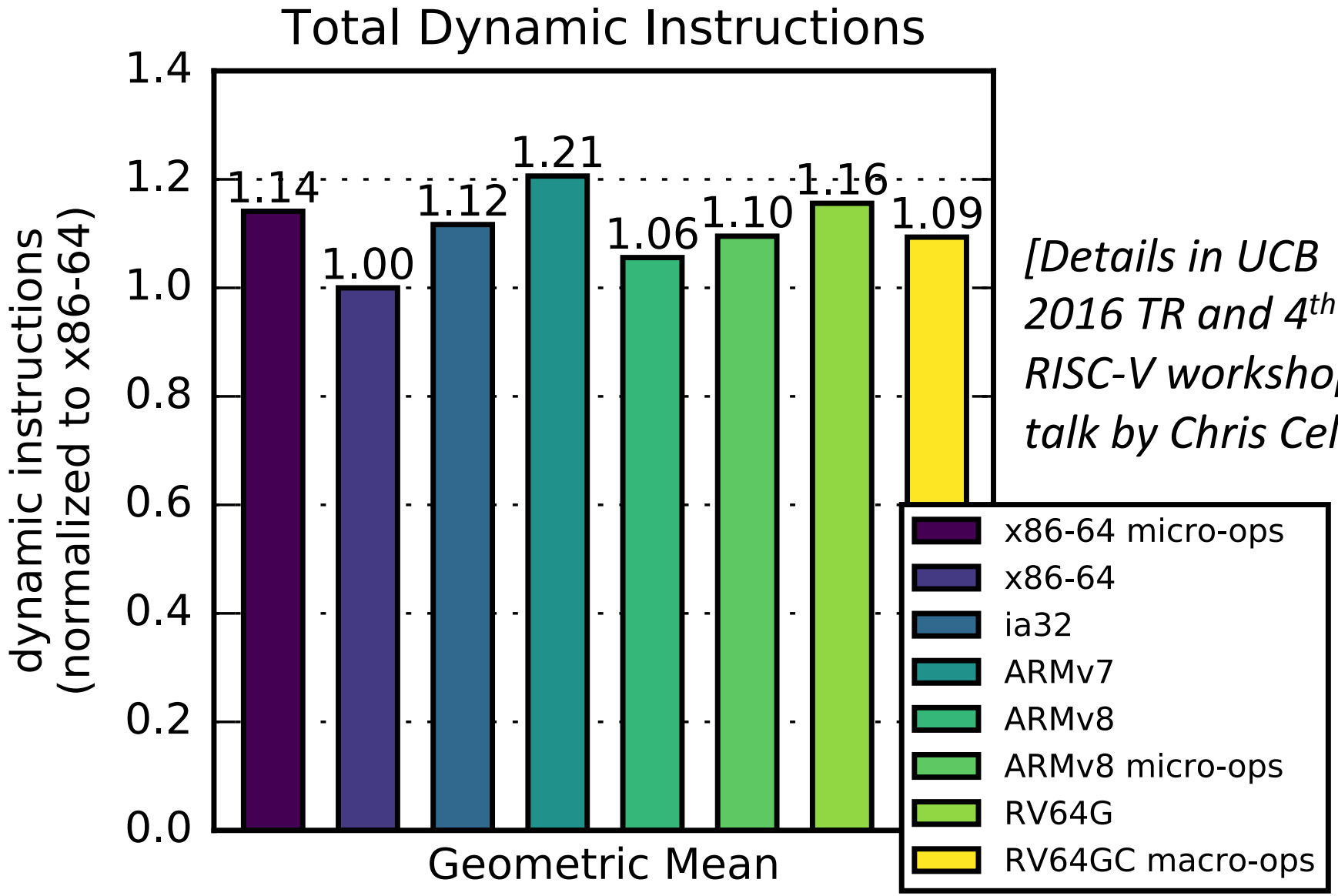
```
add rd, rs1, rs2
ld rd, 0(rd)
```

- "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]

RISC-V ISA Quality

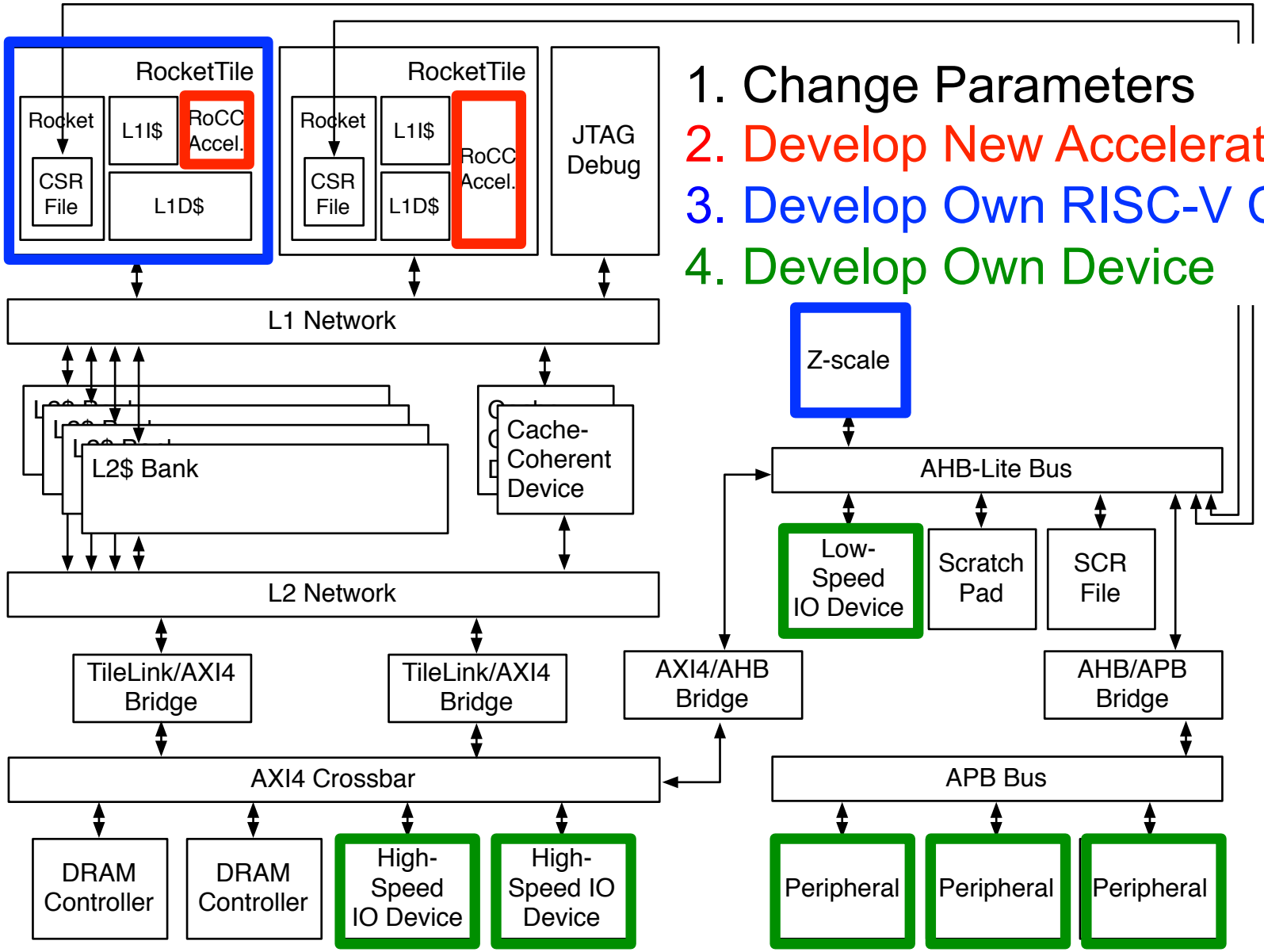
- Smallest static code size
- Fewest dynamic bytes fetched
- Comparable microarchitectural work per program

- While being the simplest ISA by far

UC Berkeley RISC-V Open-Source Core Generators

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

Rocket Chip Generator

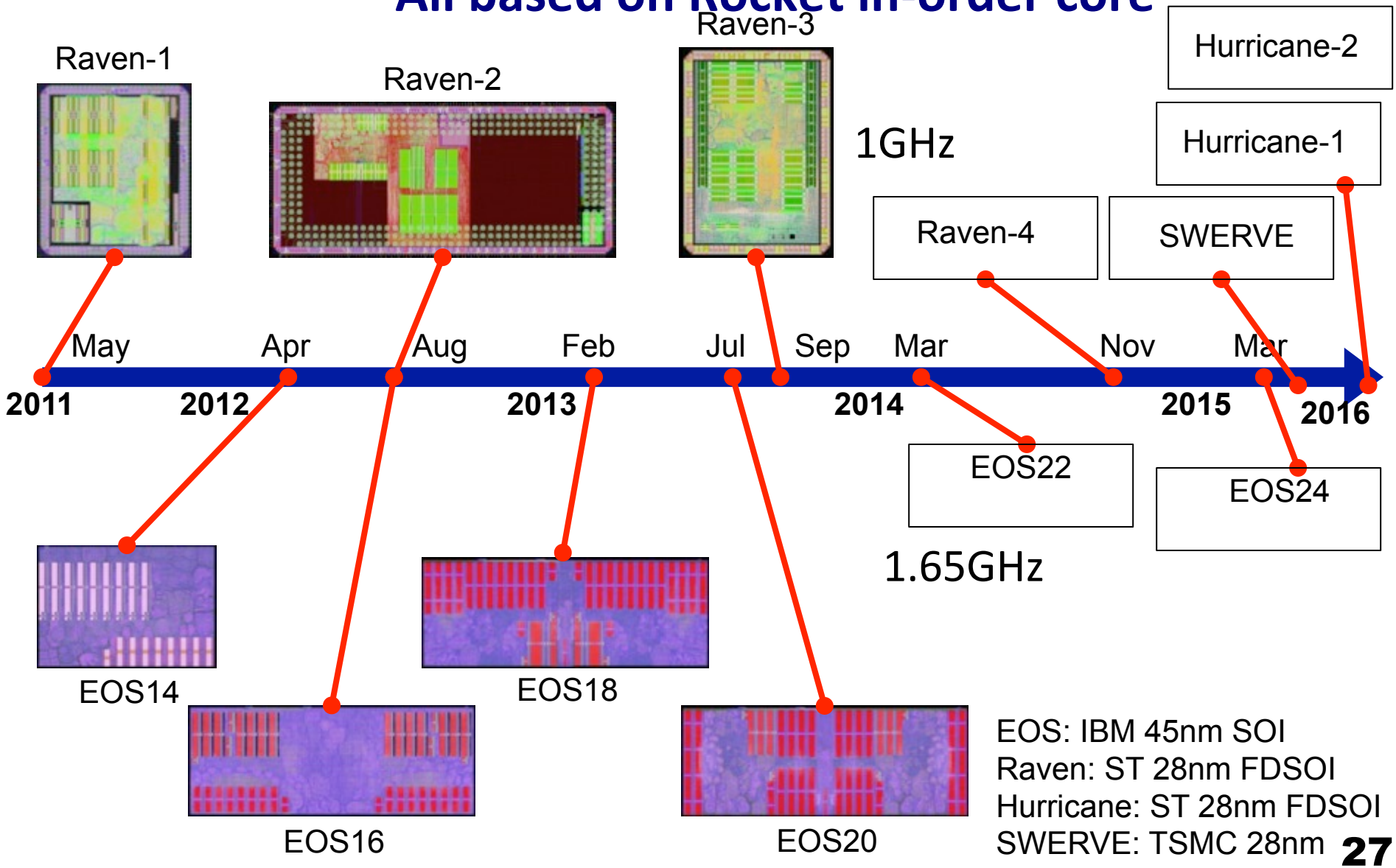


1. Change Parameters
2. Develop New Accelerators
3. Develop Own RISC-V Core
4. Develop Own Device

UC Berkeley RISC-V Cores:

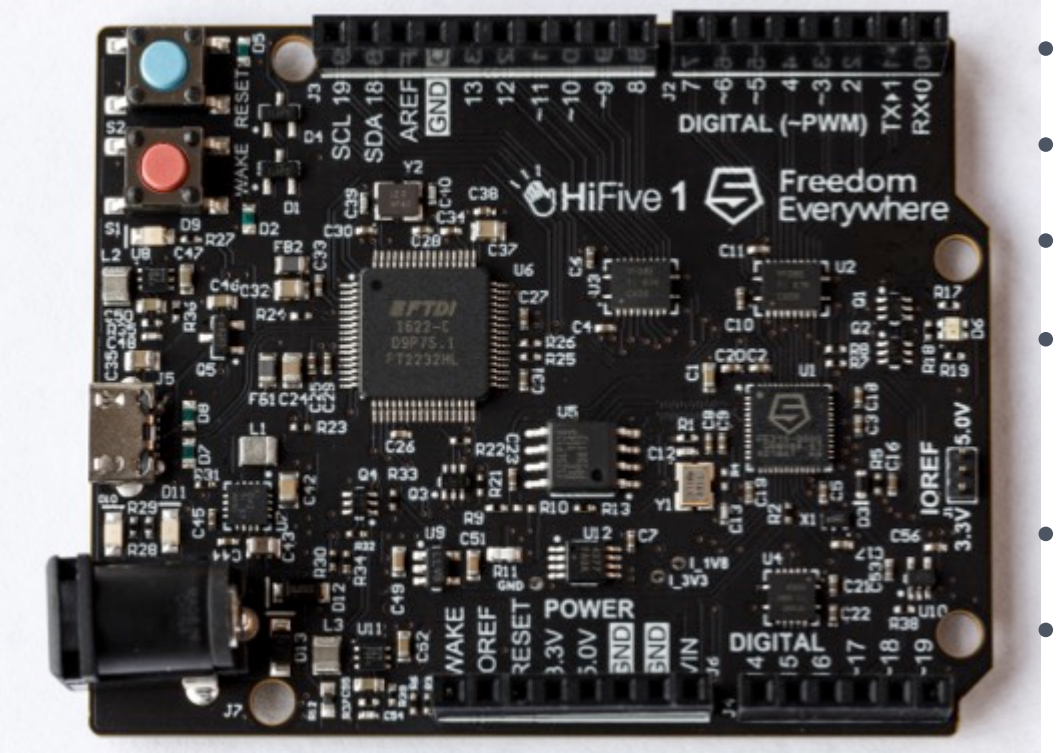
Seven 28nm & Six 45nm RISC-V Chips Tapeouts

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

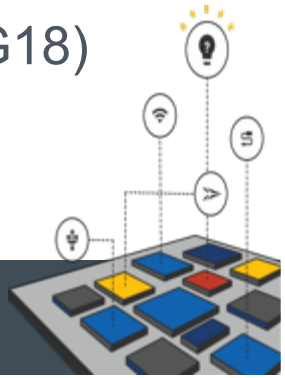
- Available for sale now!
- $59

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+



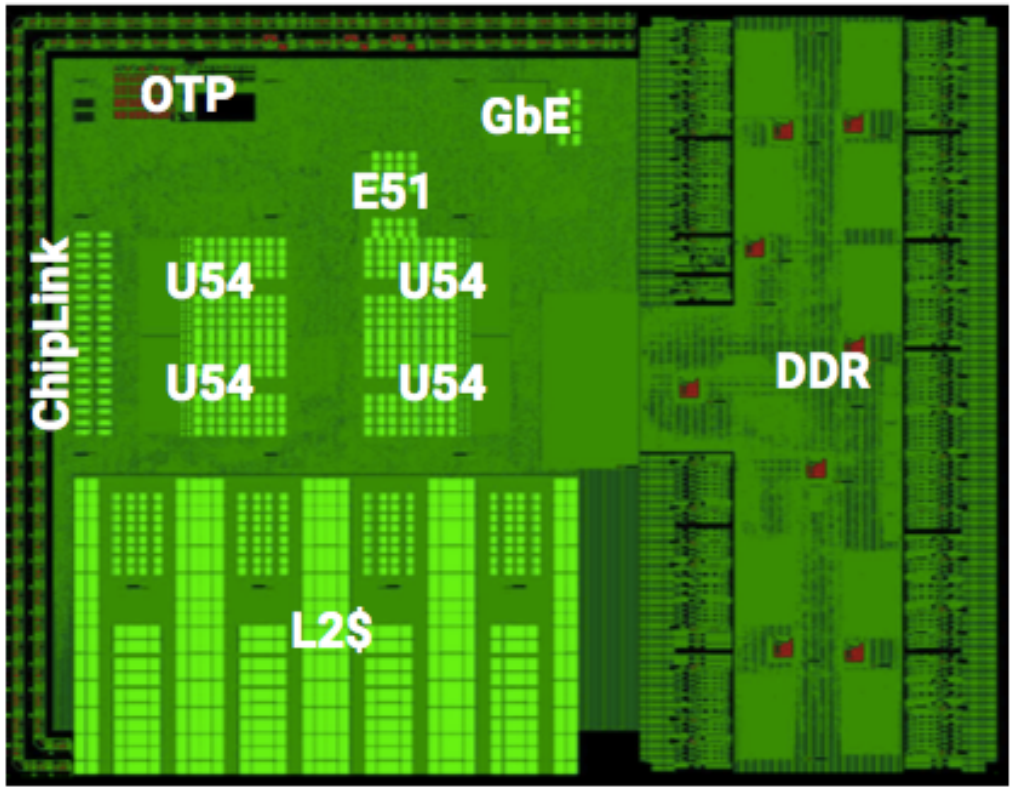
RISC-V Outside Berkeley

- Adopted as “standard ISA” for **India**
  - IIT-Madras $90M funding to build 6 different open-source cores
  - 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 32-bit/64-bit core based on RISC-V
  - Other soft core conversions to come
- **SiFive, Bluespec**, **Codasip**, **Cortus**, **Syntacore**, + others have commercial soft cores available now
- **DARPA** mandating RISC-V in SSITH BAA
- 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

SiFIve U500 Application-Processor-Class Chip

Freedom U500 Base Platform Chip

~30mm² in TSMC 28nm



- 1.5 GHz+ SiFive E51/U54 CPU
  - 1x E51: 16KB L1I$ and 8KB DTIM
  - 4x U54: 32KB L1I$ and 32KB L1D$
  - ECC support
- Banked 2MB L2$
  - ECC support
- 250M transistors
- FCBGA package
- TSMC 28HPC
- Development board available in Q1 2018

SiFive

Software Ecosystem

- GCC, binutils upstreamed as of GCC 7.1
- Linux, glibc, gdb upstream in progress
- Fedora/RedHat ported >5,000 packages
- FreeBSD upstreamed as of 11.0
- LLVM upstream in progress
- QEMU user-mode upstream in progress, system-mode soon
- ZephyrOS, 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 70 company members.
- Additional members welcome

Foundation Members (70+)

Platinum:

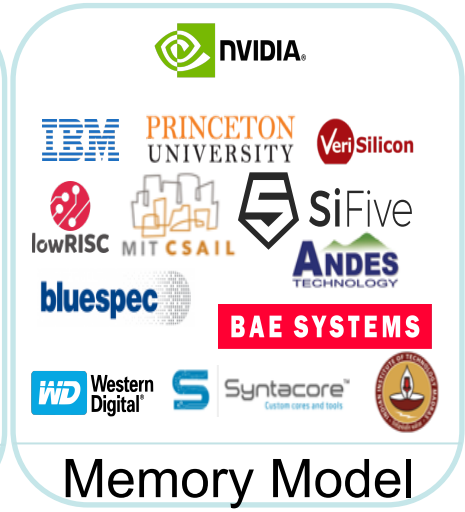
Gold, Silver, Auditors:

Foundation Working Groups (partial list)

Bit Manipulation

Compliance

Debug



Memory Model

Privileged Spec

Vector

Security

Base ISA / Opcode

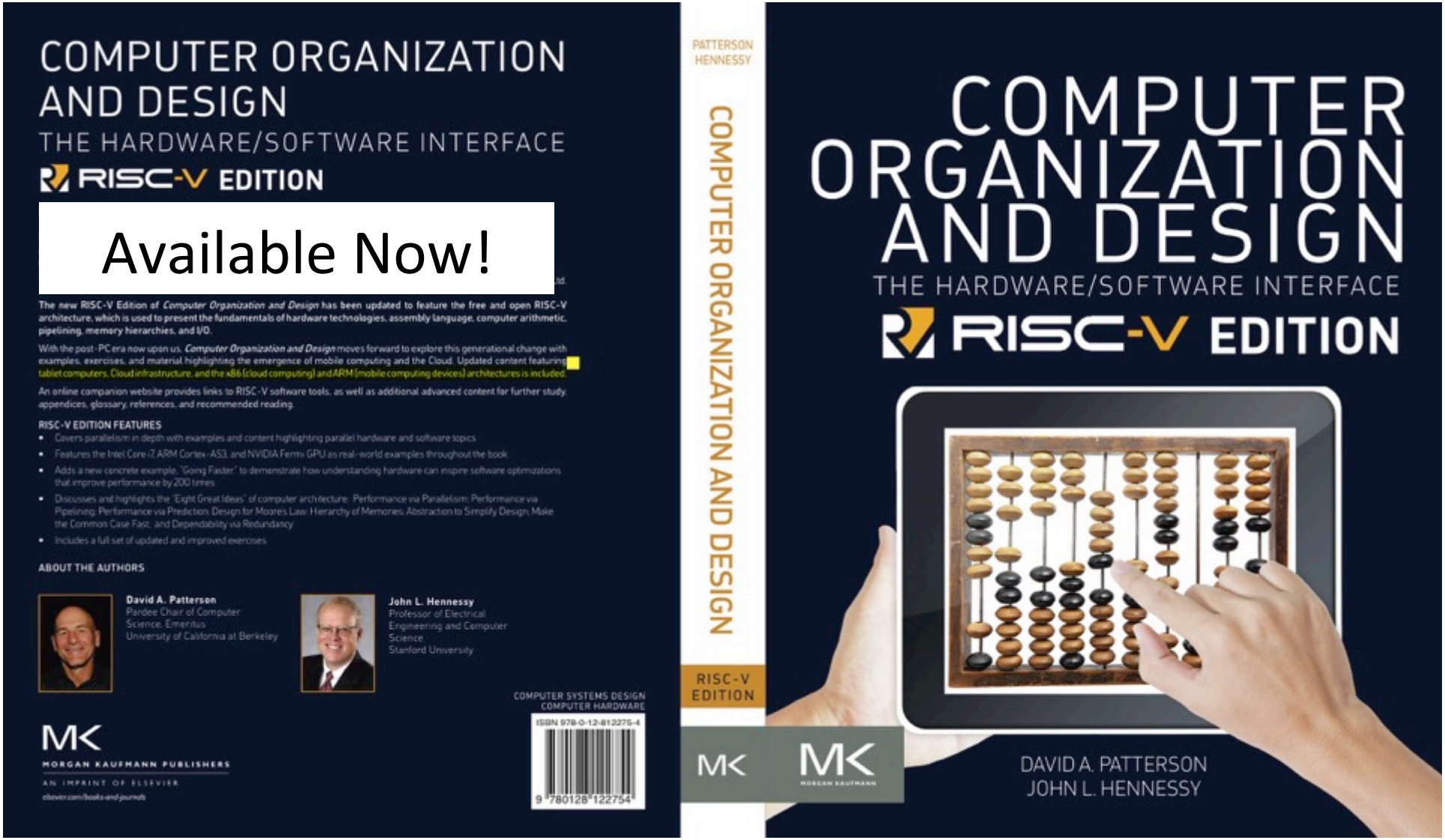
Learning More about RISC-V

- 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`**

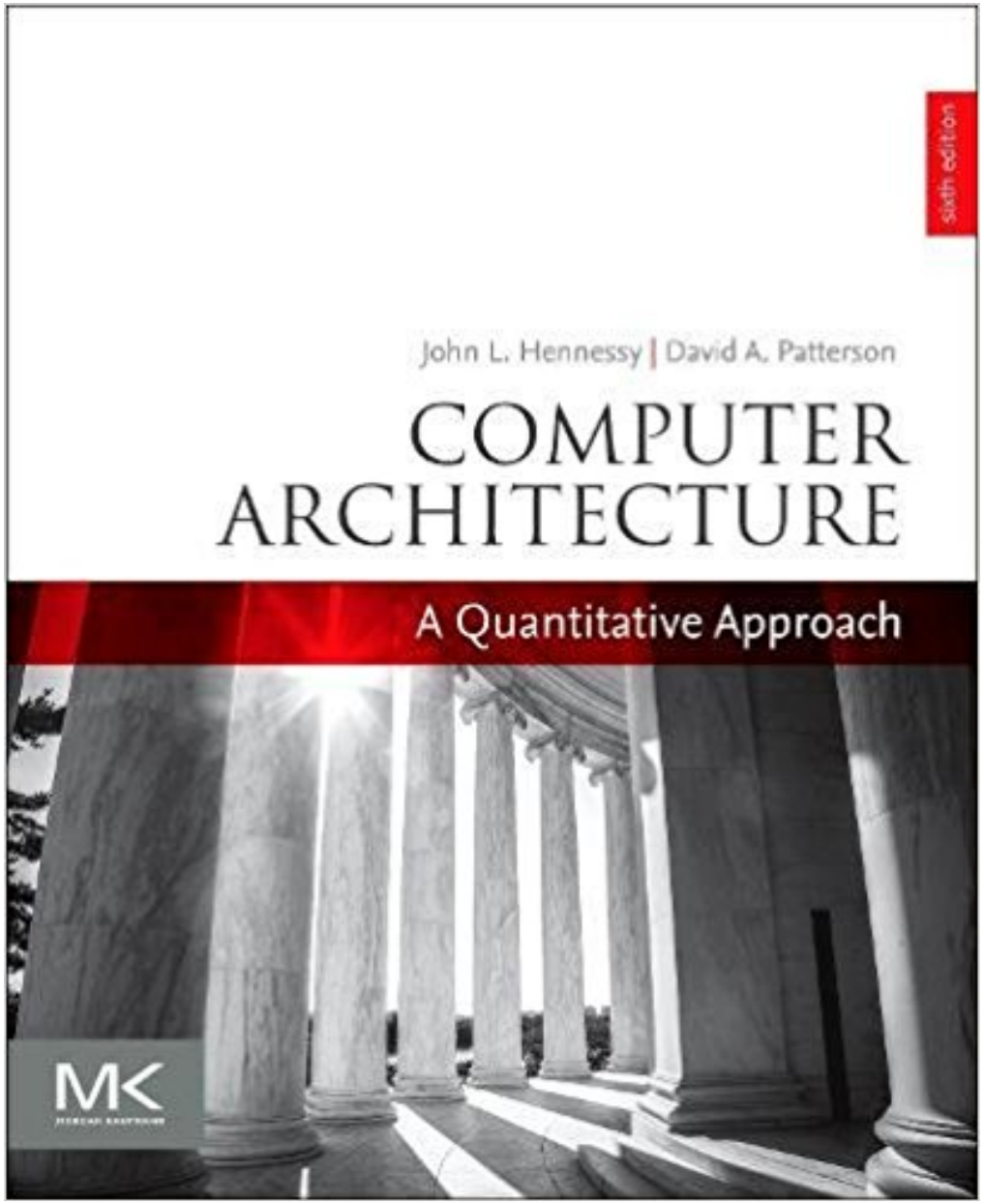
6th RISC-V Workshop
May 2017, Shanghai, China

Upcoming 7th RISC-V workshop, November 28-30, Milpitas, CA, hosted by Western Digital

RISC-V in Education, Patterson/Hennessy books

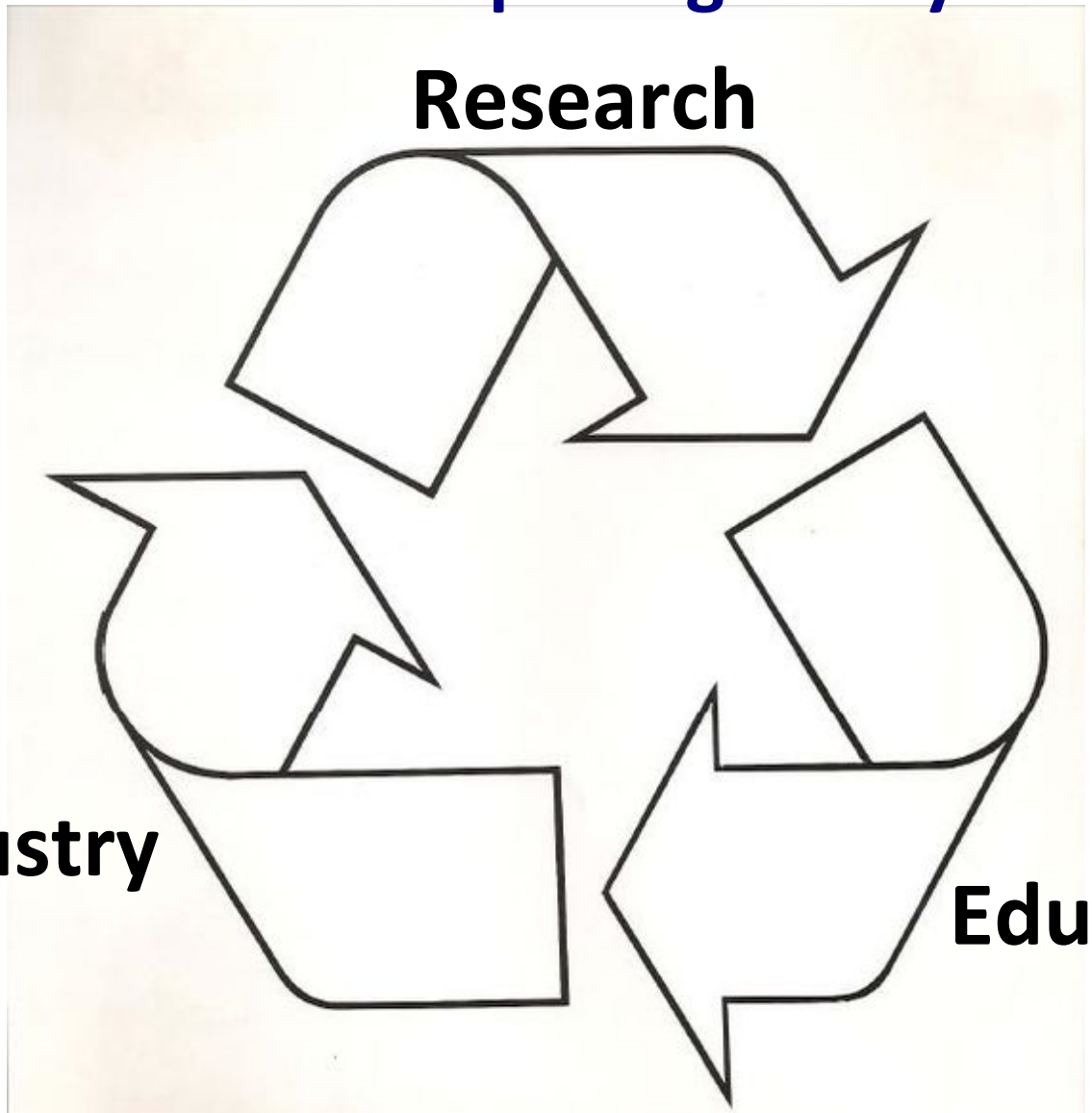


Hennessy & Patterson, 6th Edition



- Also, RISC-V based
- Released December 2017

RISC-V: Completing the Cycle



Open-source is key to keeping the virtuous cycle going

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, LGE, NEC, Microsoft, Nokia, NVIDIA, Oracle, Samsung
- Intel Science and Technology Center on Agile Design

Modest RISC-V Project Goal

Become the industry-standard ISA for all computing devices