



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 μ architecture 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

xxxxxxxxxxxxxxxxaa			16-bit (aa ≠ 11)
xxxxxxxxxxxxxxxx		xxxxxxxxxxxxbbb11	32-bit (bbb ≠ 111)
···xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxxx011111	48-bit
···xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxxx011111	64-bit
···xxxx	xxxxxxxxxxxxxxxx	xnnnxxxxx111111	(80+16*nnn)-bit, nnn≠111
···xxxx	xxxxxxxxxxxxxxxx	x111xxxxx111111	Reserved for ≥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

Download more graphics at www.psdgraphics.com

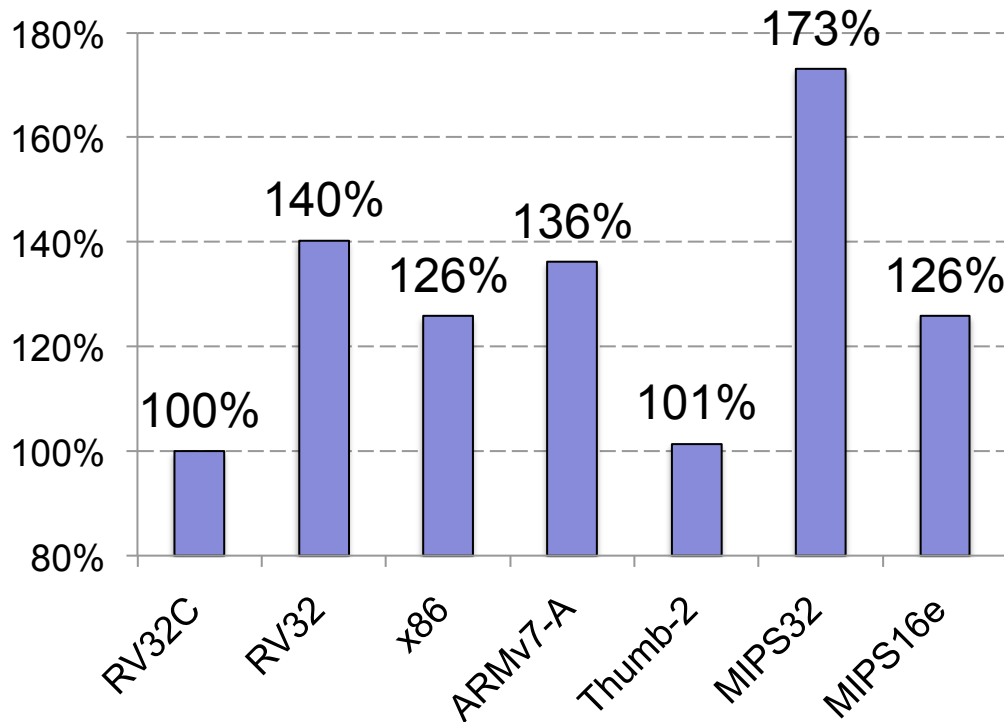
- Compressed code important for:
 - low-end embedded to save static code space
 - high-end commercial workloads to reduce cache footprint
- C extension adds 16-bit compressed instructions
 - Some 2-address forms with all 32 registers
 - More 2-address forms with most frequent 8 registers
- 1 compressed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



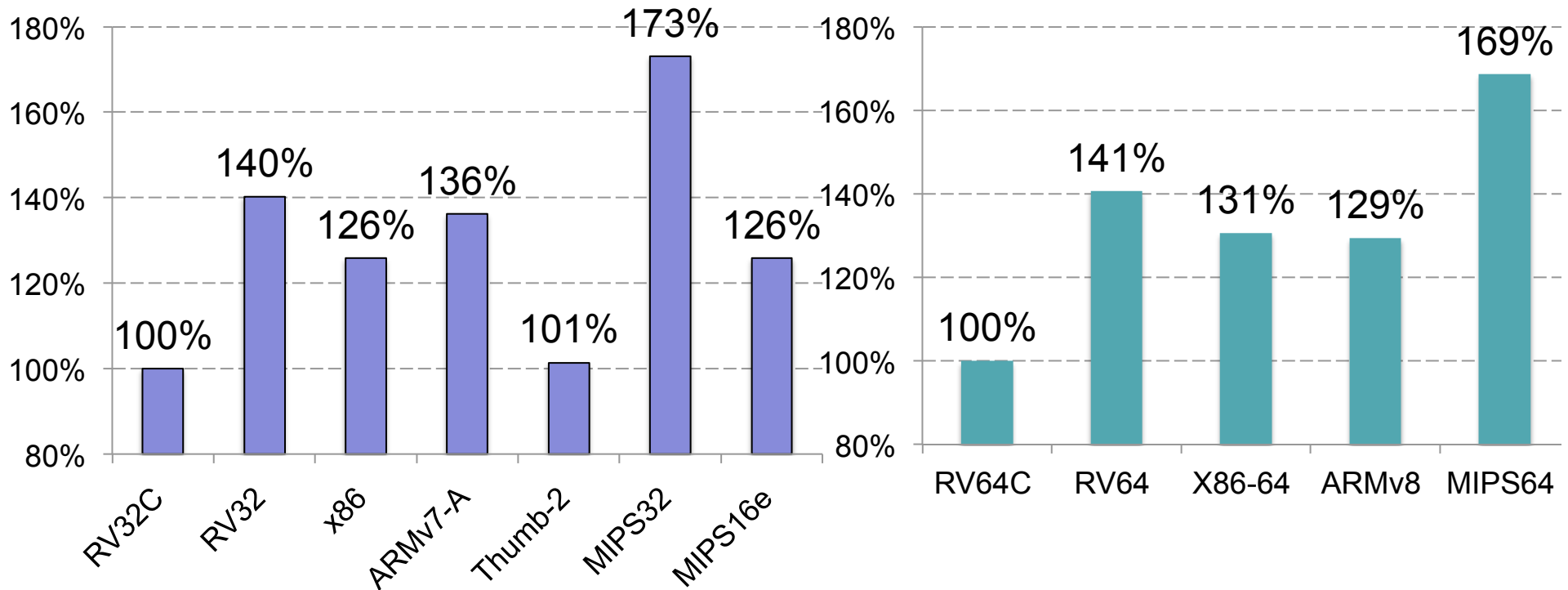


SPECint2006 compressed code size with save/restore optimization (relative to “standard” RVC)

32-bit Address



64-bit Address



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

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	LW{W D Q} rd,rs1,imm
	Load Byte Unsigned	I	LBU rd,rs1,imm
	Load Half Unsigned	I	LHU{W D}U rd,rs1,imm
Stores	Store Byte	S	SB rs1,rs2,imm
	Store Halfword	S	SH rs1,rs2,imm
	Store Word	S	SW{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

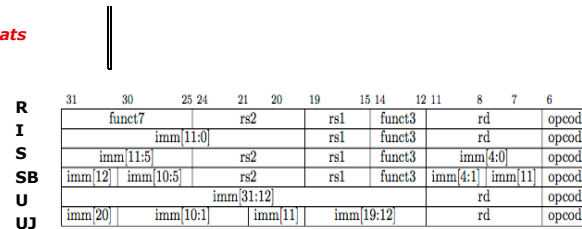
+ 8 for M

+ 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 Reference Card ④

Base Integer Instructions (32 64 128)				RV Privileged Instructions (32 64 128)				3 Optional FP Extensions: RV32{F D Q}				Optional Compressed Instructions: RVC																																																																								
Category	Name	Fmt	RV{32 64 128}I Base	Category	Name	Fmt	RV mnemonic	Category	Name	Fmt	RV{F D Q} (HP/SP,DP,QP)	Category	Name	Fmt	RVC																																																																					
Loads	Load Byte	I	LB rd,rs1,imm	CSR Access	Atomic R/W	R	CSR{RW} rd,csr,rs1	Load	Load	I	FL{W,D,Q} rd,rs1,imm	Loads	Load Word	CL	C.LW rd',rs1',imm																																																																					
	Load Halfword	I	LH rd,rs1,imm		Atomic Read & Set Bit	R	CSR{RS} rd,csr,rs1		Store	Store	S		FS{W,D,Q} rs1,rs2,imm	Load Word SP	CI	C.LWSP rd,imm																																																																				
	Load Word	I	LW{D Q} rd,rs1,imm		Atomic Read & Clear Bit	R	CSR{RC} rd,csr,rs1			Arithmetic	ADD		R	FADD.{S D Q} rd,rs1,rs2	Load Double	CI	C.LD rd',rs1',imm																																																																			
	Load Byte Unsigned	I	LBU rd,rs1,imm		Atomic R/W Imm	R	CSR{RWI} rd,csr,imm				SUBtract		R	FSUB.{S D Q} rd,rs1,rs2	Load Double SP	CI	C.LWSP rd,imm																																																																			
	Load Half Unsigned	I	LHU{D Q} rd,rs1,imm		Atomic Read & Set Bit Imm	R	CSR{RSI} rd,csr,imm				MULTiply		R	FMUL.{S D Q} rd,rs1,rs2	Load Quad	CL	C.LQ rd',rs1',imm																																																																			
Stores	Store Byte	S	SB rs1,rs2,imm	Atomic Read & Clear Bit Imm	R	CSR{RCI} rd,csr,imm	DIVide	R			FDIV.{S D Q} rd,rs1,rs2	Load Quad SP	CI	C.LQSP rd,imm																																																																						
	Store Halfword	S	SH rs1,rs2,imm	Change Level	Env. Call	R	ECALL	SQure Root	R		FSQRT.{S D Q} rd,rs1	Load Byte Unsigned	CL	C.LBU rd',rs1',imm																																																																						
	Store Word	S	SW{D Q} rs1,rs2,imm		Environment Breakpoint	R	EBREAK	Mul-Add	MULTiply-ADD	R	FMADD.{S D Q} rd,rs1,rs2,rs3	Float Load Word	CL	C.FLW rd',rs1',imm																																																																						
Shifts	Shift Left	R	SLL{W D} rd,rs1,rs2		Environment Return	R	ERET		MULTiply-SUBtract	R	FMSUB.{S D Q} rd,rs1,rs2,rs3	Float Load Double	CL	C.FLD rd',rs1',imm																																																																						
	Shift Left Immediate	I	SLLI{W D} rd,rs1,shamt	Trap Redirect	Redirect Trap to Supervisor	R	MRTS		Negative MULTiply-SUBtract	R	FMNSUB.{S D Q} rd,rs1,rs2,rs3	Float Load Word SP	CI	C.FLWSP rd,imm																																																																						
	Shift Right	R	SRL{W D} rd,rs1,rs2		Hypervisor Trap to Supervisor	R	MRTS	Negative MULTiply-ADD	R	FMNADD.{S D Q} rd,rs1,rs2,rs3	Float Load Double SP	CI	C.FLDSP rd,imm																																																																							
	Shift Right Immediate	I	SRLI{W D} rd,rs1,shamt	Interrupt	Wait for Interrupt	R	WFI	Sign Inject	SIGN source	R	FSGNJ.{S D Q} rd,rs1,rs2	Stores	Store Word	CS	C.SW rs1',rs2',imm																																																																					
	Shift Right Arithmetic	R	SRA{W D} rd,rs1,rs2		Supervisor FENCE	R	SFENCE.VM rs1		Negative SIGN source	R	FSGNJN.{S D Q} rd,rs1,rs2		Store Word SP	CSS	C.SWSP rs2,imm																																																																					
Shift Right Arith Imm	I	SRAI{W D} rd,rs1,shamt	MMU	Supervisor FENCE	R	SFENCE.VM rs1	Xor SIGN source	R	FSGNJX.{S D Q} rd,rs1,rs2	Store Double	CS		C.SD rs1',rs2',imm																																																																							
Arithmetic	ADD	R		ADD{W D} rd,rs1,rs2	Optional Multiply-Divide Extension: RV32M				Min/Max	MINimum	R		FMIN.{S D Q} rd,rs1,rs2	Store Double SP	CSS	C.SDSP rs2,imm																																																																				
	ADD Immediate	I		ADDI{W D} rd,rs1,imm	Category Name Fmt RV32M (Mult-Div)					MAXimum	R		FMAX.{S D Q} rd,rs1,rs2	Store Quad	CS	C.SQ rs1',rs2',imm																																																																				
	SUBtract	R		SUB{W D} rd,rs1,rs2	Multiply	MULTiply	R	MUL{W D} rd,rs1,rs2		Compare	Compare Float <	R	FEQ.{S D Q} rd,rs1,rs2	Store Quad SP	CSS	C.SQSP rs2,imm																																																																				
	Load Upper Imm	U		LUI rd,imm		MULTiply upper Half	R	MULHU rd,rs1,rs2			Compare Float <=	R	FLT.{S D Q} rd,rs1,rs2	Float Store Word	CSS	C.FSW rd',rs1',imm																																																																				
ADD Upper Imm to PC	U	AUIPC rd,imm		MULTiply Half Sign/Uns	R	MULHSU rd,rs1,rs2	Compare Float <=	R	FLE.{S D Q} rd,rs1,rs2	Float Store Double	CSS	C.FSD rd',rs1',imm																																																																								
Logical	XOR	R		XOR rd,rs1,rs2	MULTiply upper Half Uns	R	MULHU rd,rs1,rs2	Categorize	Classify Typ	R	FCLASS.{S D Q} rd,rs1	Float Store Word SP	CSS	C.FSWSP rd,imm																																																																						
	XOR Immediate	I		XORI rd,rs1,imm	Divide	DIVide	R		DIV{W D} rd,rs1,rs2	Move	Move from Integer	R	FMV.S.X rd,rs1	Float Store Double SP	CSS	C.FSDSP rd,imm																																																																				
	OR	R		OR rd,rs1,rs2		DIVide Unsigned	R	DIVU rd,rs1,rs2	Move to Integer		R	FMV.X.S rd,rs1	Convert	Convert from Int Unsigned	R	FCVT.{S D Q}.W rd,rs1																																																																				
	OR Immediate	I		ORI rd,rs1,imm	Remainder	REMAinder	R	REM{W D} rd,rs1,rs2	Convert from Int	R	FCVT.W.{S D Q} rd,rs1	Convert to Int Unsigned		R	FCVT.WU.{S D Q} rd,rs1																																																																					
	AND	R	AND rd,rs1,rs2	REMAinder Unsigned		R	REMU{W D} rd,rs1,rs2	Optional Atomic Instruction Extension: RVA				Configuration		Read Stat	R	FRCSR rd																																																																				
AND Immediate	I	ANDI rd,rs1,imm	Category Name Fmt RV{32 64 128}A (Atomic)				Read Rounding Mode	R	FRRM rd	Read Flags	R			FRFLAGS rd																																																																						
Compare	Set <	R	SLT rd,rs1,rs2	Load	Load Reserved	R	LR{W D Q} rd,rs1	Swap Status Req	R	FSCSR rd,rs1	Swap Rounding Mode			R	FSRM rd,rs1																																																																					
	Set < Immediate	I	SLTI rd,rs1,imm		Store	Store Conditiona	R	SC{W D Q} rd,rs1,rs2	Swap Flags	R	FSFLAG rd,rs1		Swap Rounding Mode Imm	I	FSRMI rd,imm																																																																					
	Set < Unsigned	R	SLTU rd,rs1,rs2			Swap	SWAP	R	AMOSWAP{W D Q} rd,rs1,rs2	Swap Rounding Mode Imm	I		FSRMI rd,imm	Swap Flans Imm	I	FSPT.AGST rd,imm																																																																				
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm				Add	ADD	R	AMOADD{W D Q} rd,rs1,rs2	Configuration	Read Stat	R	FRCSR rd																																																																						
Branches	Branch =	SB	BNE rs1,rs2,imm	Logical				XOR	R	AMOXOR{W D Q} rd,rs1,rs2		Read Rounding Mode	R	FRRM rd																																																																						
	Branch ≠	SB	BNE rs1,rs2,imm		AND			R	AMOAND{W D Q} rd,rs1,rs2	Swap Status Req		R	FSCSR rd,rs1																																																																							
	Branch <	SB	BLT rs1,rs2,imm		OR	R		AMOOR{W D Q} rd,rs1,rs2	Swap Rounding Mode	R		FSRM rd,rs1																																																																								
	Branch >	SB	BGE rs1,rs2,imm		Min/Max	MINimum	R	AMOHIN{W D Q} rd,rs1,rs2	Swap Flags	R		FSFLAG rd,rs1																																																																								
	Branch < Unsigned	SB	BLTU rs1,rs2,imm			MAXimum	R	AMOMAX{W D Q} rd,rs1,rs2	Swap Rounding Mode Imm	I	FSRMI rd,imm																																																																									
Branch ≥	SB	BGEU rs1,rs2,imm	MINimum Unsigned	R	AMOHINU{W D Q} rd,rs1,rs2	Swap Flans Imm	I	FSPT.AGST rd,imm																																																																												
Branch ≥ Unsigned	SB	BGEU rs1,rs2,imm	MAXimum Unsigned	R	AMOMAXU{W D Q} rd,rs1,rs2	16-bit (RVC) and 32-bit Instruction Formats																																																																														
Jump & Link	J&L	UJ	JAL rd,imm	CI	<table border="1"> <tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr> <tr><td colspan="4">funct4</td><td colspan="4">rd/rs1</td><td colspan="4">rs2</td><td colspan="4">op</td></tr> </table>				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	funct4				rd/rs1				rs2				op				CS	<table border="1"> <tr><td>31</td><td>30</td><td>25</td><td>24</td><td>21</td><td>20</td><td>19</td><td>15</td><td>14</td><td>12</td><td>11</td><td>8</td><td>7</td><td>6</td><td>0</td></tr> <tr><td colspan="4">funct7</td><td colspan="4">rs2</td><td colspan="4">rs1</td><td colspan="4">funct3</td><td colspan="4">rd</td><td colspan="4">opcode</td></tr> </table>				31	30	25	24	21	20	19	15	14	12	11	8	7	6	0	funct7				rs2				rs1				funct3				rd				opcode			
	15	14	13		12	11	10	9	8	7	6	5	4	3	2	1	0																																																																			
funct4				rd/rs1				rs2				op																																																																								
31	30	25	24	21	20	19	15	14	12	11	8	7	6	0																																																																						
funct7				rs2				rs1				funct3				rd				opcode																																																																
Jump & Link Register	I	JALR rd,rs1,imm	CSS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rd/rs1</td><td colspan="4">imm</td><td colspan="4">op</td></tr> </table>				funct3				imm				rd/rs1				imm				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs2</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs2				op																																							
funct3				imm				rd/rs1				imm				op																																																																				
funct3				imm				rs2				op																																																																								
Synch	Synch thread	I	FENCE	CL	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rd'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rd'				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs1'</td><td colspan="4">imm</td><td colspan="4">rd'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs1'				imm				rd'				op																																		
	funct3				imm				rd'				op																																																																							
funct3				imm				rs1'				imm				rd'				op																																																																
Synch Instr & Data	I	FENCE.I	CB	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs1'</td><td colspan="4">imm</td><td colspan="4">rs2'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs1'				imm				rs2'				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs1'</td><td colspan="4">imm</td><td colspan="4">rs2'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs1'				imm				rs2'				op																											
funct3				imm				rs1'				imm				rs2'				op																																																																
funct3				imm				rs1'				imm				rs2'				op																																																																
System	System CALL	I	SCALL	CJ	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">offset</td><td colspan="4">rs1'</td><td colspan="4">offset</td><td colspan="4">op</td></tr> </table>				funct3				offset				rs1'				offset				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">offset</td><td colspan="4">rs1'</td><td colspan="4">offset</td><td colspan="4">op</td></tr> </table>				funct3				offset				rs1'				offset				op																																		
	funct3				offset				rs1'				offset				op																																																																			
funct3				offset				rs1'				offset				op																																																																				
System BREAK	I	SBREAK	CI	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">jump target</td><td colspan="4">op</td></tr> </table>				funct3				jump target				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">jump target</td><td colspan="4">op</td></tr> </table>				funct3				jump target				op																																																			
funct3				jump target				op																																																																												
funct3				jump target				op																																																																												
Counters	Read CYCLE	I	RDCYCLE rd	CI	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rd/rs1</td><td colspan="4">imm</td><td colspan="4">op</td></tr> </table>				funct3				imm				rd/rs1				imm				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs2</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs2				op																																						
	funct3				imm				rd/rs1				imm				op																																																																			
funct3				imm				rs2				op																																																																								
Read CYCLE upper Half	I	RDCYLEH rd	CL	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rd'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rd'				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs1'</td><td colspan="4">imm</td><td colspan="4">rd'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs1'				imm				rd'				op																																			
funct3				imm				rd'				op																																																																								
funct3				imm				rs1'				imm				rd'				op																																																																
Read TIME	I	RDTIME rd	CB	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs1'</td><td colspan="4">imm</td><td colspan="4">rs2'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs1'				imm				rs2'				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs1'</td><td colspan="4">imm</td><td colspan="4">rs2'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs1'				imm				rs2'				op																											
funct3				imm				rs1'				imm				rs2'				op																																																																
funct3				imm				rs1'				imm				rs2'				op																																																																
Read TIME upper Half	I	RDTIMEH rd	CJ	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">offset</td><td colspan="4">rs1'</td><td colspan="4">offset</td><td colspan="4">op</td></tr> </table>				funct3				offset				rs1'				offset				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">offset</td><td colspan="4">rs1'</td><td colspan="4">offset</td><td colspan="4">op</td></tr> </table>				funct3				offset				rs1'				offset				op																																			
funct3				offset				rs1'				offset				op																																																																				
funct3				offset				rs1'				offset				op																																																																				
Read INSTR RETired	I	RDINSTRET rd	CI	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs1'</td><td colspan="4">imm</td><td colspan="4">rs2'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs1'				imm				rs2'				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">imm</td><td colspan="4">rs1'</td><td colspan="4">imm</td><td colspan="4">rs2'</td><td colspan="4">op</td></tr> </table>				funct3				imm				rs1'				imm				rs2'				op																											
funct3				imm				rs1'				imm				rs2'				op																																																																
funct3				imm				rs1'				imm				rs2'				op																																																																
Read INSTR upper Half	I	RDINSTRETH rd	CJ	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">offset</td><td colspan="4">rs1'</td><td colspan="4">offset</td><td colspan="4">op</td></tr> </table>				funct3				offset				rs1'				offset				op				CS	<table border="1"> <tr><td colspan="4">funct3</td><td colspan="4">offset</td><td colspan="4">rs1'</td><td colspan="4">offset</td><td colspan="4">op</td></tr> </table>				funct3				offset				rs1'				offset				op																																			
funct3				offset				rs1'				offset				op																																																																				
funct3				offset				rs1'				offset				op																																																																				

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



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

RISC-V “Green Card”



①

②

③

RISC-V Reference Card ④

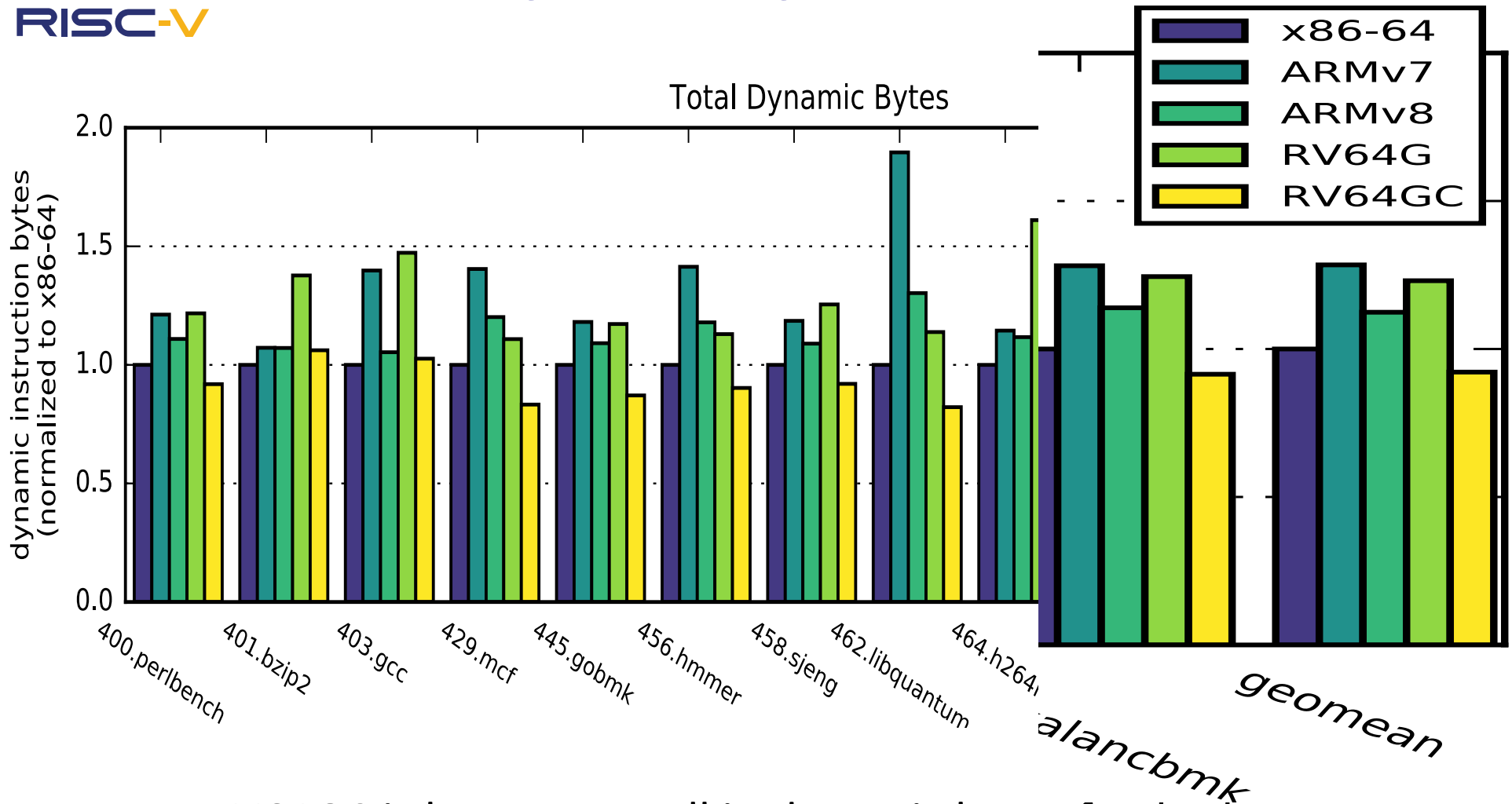
Base Integer Instructions (32 64 128)				RV Privileged Instructions (32 64 128)				3 Optional FP Extensions: RV32{F D Q}				Optional Compressed Instructions: RVC											
Category	Name	Fmt	RV{32 64 128}I Base	Category	Name	Fmt	RV mnemonic	Category	Name	Fmt	RV{F D Q} (HP/SP,DP,QP)	Category	Name	Fmt	RVC								
Loads	Load Byte	I	LB rd,rs1,imm	CSR Access	Atomic R/W	R	CSRRW rd,csr,rs1	Load	Load	I	FL{W,D,Q} rd,rs1,imm	Loads	Load Word	CL	C.LW rd',rs1',imm								
	Load Halfword	I	LH rd,rs1,imm		Atomic Read & Set Bit	R	CSRRS rd,csr,rs1		Store	Store	S		FS{W,D,Q} rs1,rs2,imm	Load Word SP	CI	C.LWSP rd,imm							
	Load Word	I	LD{W D Q} rd,rs1,imm		Atomic Read & Clear Bit	R	CSRRC rd,csr,rs1			Arithmetic	ADD		R	FADD.{S D Q} rd,rs1,rs2	Load Double	CI	C.LD rd',rs1',imm						
	Load Byte Unsigned	I	LBUI rd,rs1,imm		Atomic R/W Imm	R	CSRRI rd,csr,rs1				SUBtract		R	FSUB.{S D Q} rd,rs1,rs2	Load Double SP	CI	C.LWSP rd,imm						
	Load Half Unsigned	I	LH{W D U} rd,rs1,imm		Atomic Read & Set Bit Imm	R	CSRRII rd,csr,imm				MULTiply		R	FMUL.{S D Q} rd,rs1,rs2	Load Quad	CL	C.LQ rd',rs1',imm						
				Atomic Read & Clear Bit Imm	R	CSRRCI rd,csr,imm	DIVide	R			FDIV.{S D Q} rd,rs1,rs2	Load Quad SP	CI	C.LQSP rd,imm									
Stores	Store Byte	S	SB rs1,rs2,imm	Change Level	Env. Call	R	ECALL	Mul-Add	MULTiply-ADD		R	FMADD.{S D Q} rd,rs1,rs2,rs3	Stores	Store Word	CS	C.SW rs1',rs2',imm							
	Store Halfword	S	SH rs1,rs2,imm		Environment Breakpoint	R	EBREAK		Compare	MULTiply-SUBtract	R	FMSUB.{S D Q} rd,rs1,rs2,rs3		Store Word SP	CSS	C.SWSP rs2,imm							
	Store Word	S	SD{W D Q} rs1,rs2,imm		Environment Return	R	ERET			Negative MULTiply-SUBtract	R	FMNSUB.{S D Q} rd,rs1,rs2,rs3		Store Double	CS	C.SD rs1',rs2',imm							
Shifts	Shift Left	R	SLL{W D} rd,rs1,rs2	Trap Redirect	Redirect Trap to Supervisor	R	MRTS	Sign Inject		SIGN source	R	FSGNJ.{S D Q} rd,rs1,rs2	Min/Max	Minimum	R	FMIN.{S D Q} rd,rs1,rs2							
	Shift Left Immediate	I	SLLI{W D} rd,rs1,shamt		Redirect Trap to Hypervisor	R	MRTH		Negative MULTiply-ADD	Negative SIGN source	R	FSGNJN.{S D Q} rd,rs1,rs2		Configuration	Read Status	R	FRCSR rd						
	Shift Right	R	SRL{W D} rd,rs1,rs2		Hypervisor Trap to Supervisor	R	MRTS			Convert	Convert from Integer	R			FCVT.{S D Q}.W rd,rs1	Shifts	Shift Left Imm	CI	C.SLLI rd,imm				
	Shift Right Immediate	I	SRLI{W D} rd,rs1,shamt		Interrupt Wait for Interrupt	R	WFI				Move	Convert from Int Unsigned			R		FCVT.W.{S D Q} rd,rs1	Branches	Branch=0	CB	C.BEQ rs1',imm		
	Shift Right Arithmetic	R	SRA{W D} rd,rs1,rs2		MMU Supervisor FENCE	R	SFENCE.VM rs1					Convert			Convert to Integer		R		FCVT.WU.{S D Q} rd,rs1	Jump	Jump	CJ	C.J imm
Shift Right Arith Imm	I	SRAI{W D} rd,rs1,shamt				3 Optional FP Extensions: RV{64 128}{F D Q}	Convert to Int Unsigned	R					FCVT.WU.{S D Q} rd,rs1		Jump & Link		Jump & Link Register		CR		C.JAL rd,imm		
Arithmetic	ADD	R	ADD{W D} rd,rs1,rs2	Optional Multiply-Divide Extension: RV32M				Configuration	Read Rounding Mode				R	FRRM rd			System		Env. BREAK		CI	C.EBREAK	
	ADD Immediate	I	ADDI{W D} rd,rs1,imm	Category	Name		Fmt		RV32M (Mult-Div)	Convert			Read Flags	R		FRFLAGS rd			Jump & Link		Jump & Link Register	CR	C.JALR rs1
	SUBtract	R	SUB{W D} rd,rs1,rs2	MULTiply	MULTiply		R		MUL{W D} rd,rs1,rs2		Convert		Swap Status Req	R		FSCSR rd,rs1		System					
	Load Upper Imm	U	LUI rd,imm	MULTiply upper Half	R		MULH rd,rs1,rs2		Convert			Swap Rounding Mode	R	FSRM rd,rs1		System							
	Add Upper Imm to PC	U	AUIPC rd,imm	MULTiply Half Sign/Uns	R	MULHSU rd,rs1,rs2	Convert					Swap Flags	R	FSPFLAGS rd,rs1	System								
				MULTiply upper Half Uns	R	MULHU rd,rs1,rs2		Convert				Swap Rounding Mode Imm	I	FSRMI rd,imm			System						
Logical	XOR	R	XOR rd,rs1,rs2	DIVide	DIVide	R				DIV{W D} rd,rs1,rs2		Convert	Swap Flags Imm	I					FSPLAGSI rd,imm	System			
	XOR Immediate	I	XORI rd,rs1,imm	REMAinder	REMAinder	R				REM{W D} rd,rs1,rs2	Convert								System				
	OR	R	OR rd,rs1,rs2	REMAinder Unsigned	R	REMU{W D} rd,rs1,rs2			Convert							System							
	OR Immediate	I	ORI rd,rs1,imm	Optional Atomic Instruction Extension: RVA						Convert								System					
	AND	R	AND rd,rs1,rs2	Category	Name	Fmt	RV{32 64 128}A (Atomic)	Convert									System						
AND Immediate	I	ANDI rd,rs1,imm	Load	Load Reserved	R	LR.{W D Q} rd,rs1	Convert								System								
Compare	Set <	R	SLT rd,rs1,rs2	Store	Store Conditiona	R					SC.{W D Q} rd,rs1,rs2	Convert								System			
	Set < Immediate	I	SLTI rd,rs1,imm	SWAP	SWAP	R			AMOSWAP.{W D Q} rd,rs1,rs2		Convert								System				
	Set < Unsigned	R	SLTU rd,rs1,rs2	ADD	ADD	R			AMOADD.{W D Q} rd,rs1,rs2	Convert								System					
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm	Logical	XOR	R		AMOXOR.{W D Q} rd,rs1,rs2	Convert								System						
					AND	R	AMOAND.{W D Q} rd,rs1,rs2	Convert								System							
				OR	R	AMOOR.{W D Q} rd,rs1,rs2	Convert								System								
Branches	Branch =	SB	BNE rs1,rs2,imm	MIN/Max	MINimum	R					AMOMIN.{W D Q} rd,rs1,rs2	Convert								System			
	Branch ≠	SB	BNE rs1,rs2,imm	MAXimum	R	AMOMAX.{W D Q} rd,rs1,rs2				Convert								System					
	Branch <	SB	BLT rs1,rs2,imm	MINimum Unsigned	R	AMOMINU.{W D Q} rd,rs1,rs2			Convert								System						
	Branch ≥	SB	BGE rs1,rs2,imm	MAXimum Unsigned	R	AMOMAXU.{W D Q} rd,rs1,rs2		Convert								System							
	Branch < Unsigned	SB	BLTU rs1,rs2,imm				Convert								System								
Branch ≥ Unsigned	SB	BGEU rs1,rs2,imm				Convert								System									
Jump & Link	J&L	UJ	JAL rd,imm	16-bit (RVC) and 32-bit Instruction Formats						Convert								System					
	Jump & Link Register	I	JALR rd,rs1,imm	CI	<pre> 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 func4 rd/rs1 rs2 op </pre>				Convert								System						
Synch	Synch thread	I	FENCE	CSS	<pre> func3 imm rd/rs1 imm op func3 imm rs2 op </pre>						Convert								System				
	Synch Instr & Data	I	FENCE.I	CL	<pre> func3 imm rs1' imm rd' op func3 imm rs1' imm rd' op </pre>							Convert								System			
	System	System CALL	I	SCALL	CS	<pre> func3 imm rs1' imm rs2' op func3 offset rs1' offset op </pre>							Convert								System		
		System BREAK	I	SBREAK	CB	<pre> func3 offset rs1' offset op </pre>				Convert								System					
		Counters	Read CYCLE	I	RDCYCLE rd	CJ	<pre> func3 jump target op </pre>							Convert								System	
Read CYCLE upper Half	I		RDCYCLEH rd					Convert								System							
Read TIME	I	RDTIME rd					Convert								System								
Read TIME upper Half	I	RDTIMEH rd							Convert								System						
Read INSTR RETired	I	RDINSTRET rd								Convert								System					
Read INSTR upper Half	I	RDINSTRETH rd									Convert								System				



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

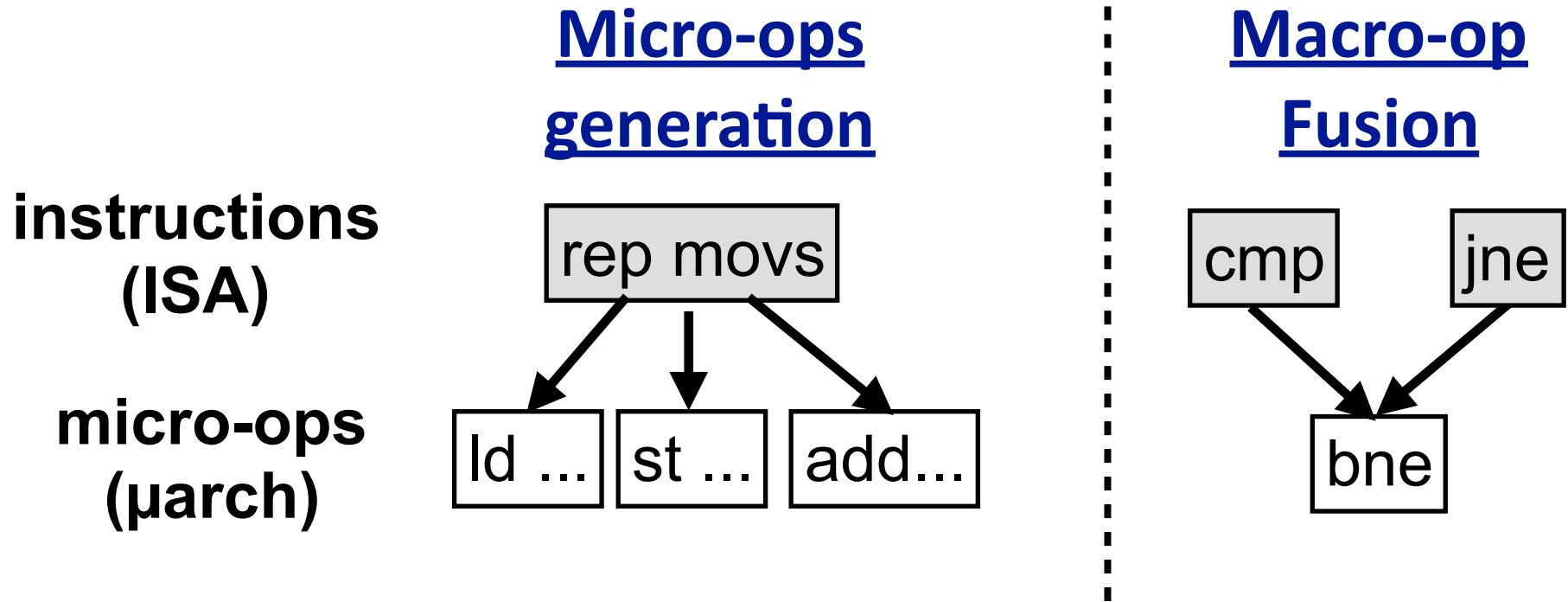


- RV64GC is lowest overall in dynamic bytes fetched
 - Despite current lack of support for vector operations



Converting Instructions to Microops

Microops are measure of microarchitectural work performed



Multiple microinstructions from one macroinstruction
Or one microinstruction from multiple macroinstructions

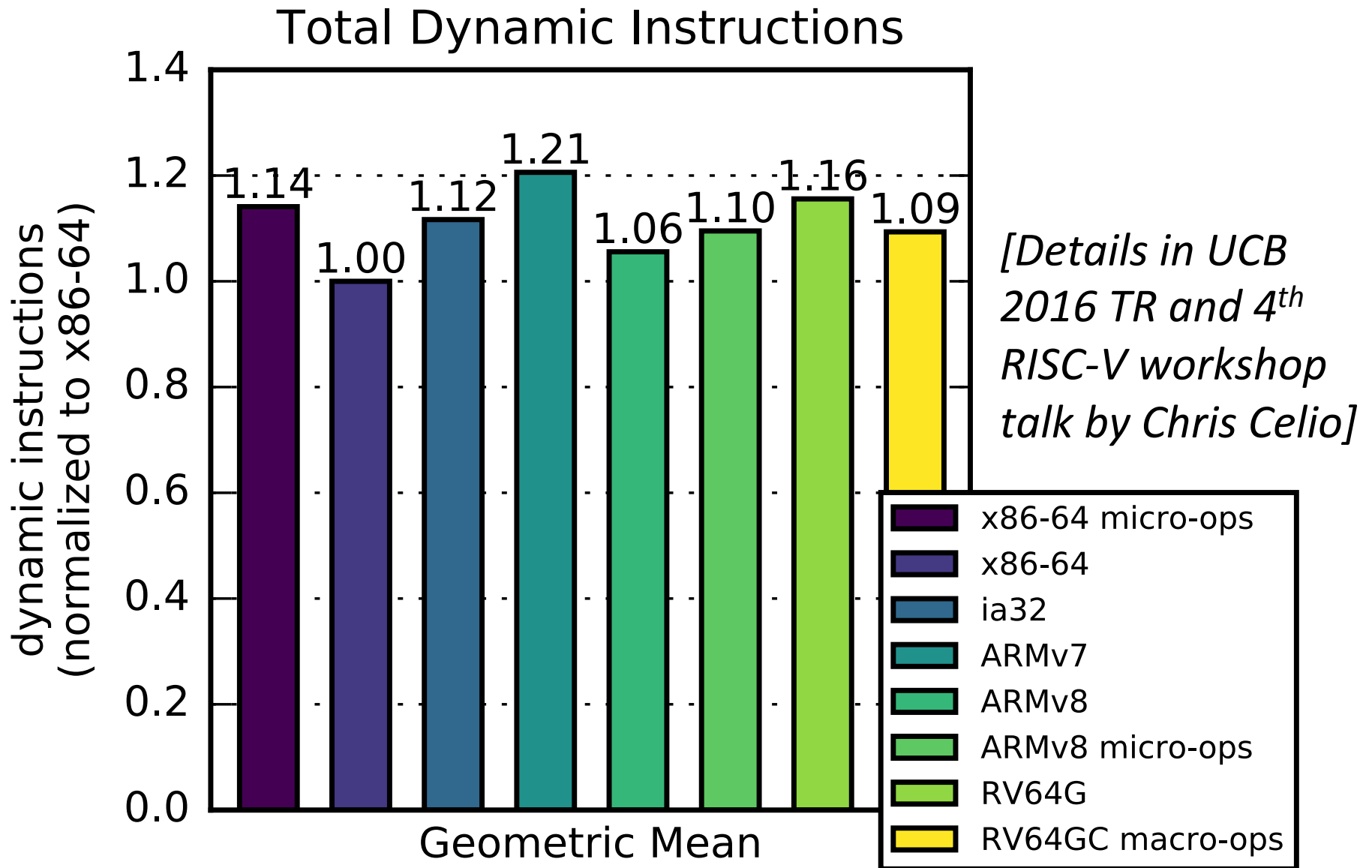


RISC-V Macro-Op Fusion Examples

- “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`
`srlr 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 μ arch Effort after Fusion





RISC-V ISA Quality

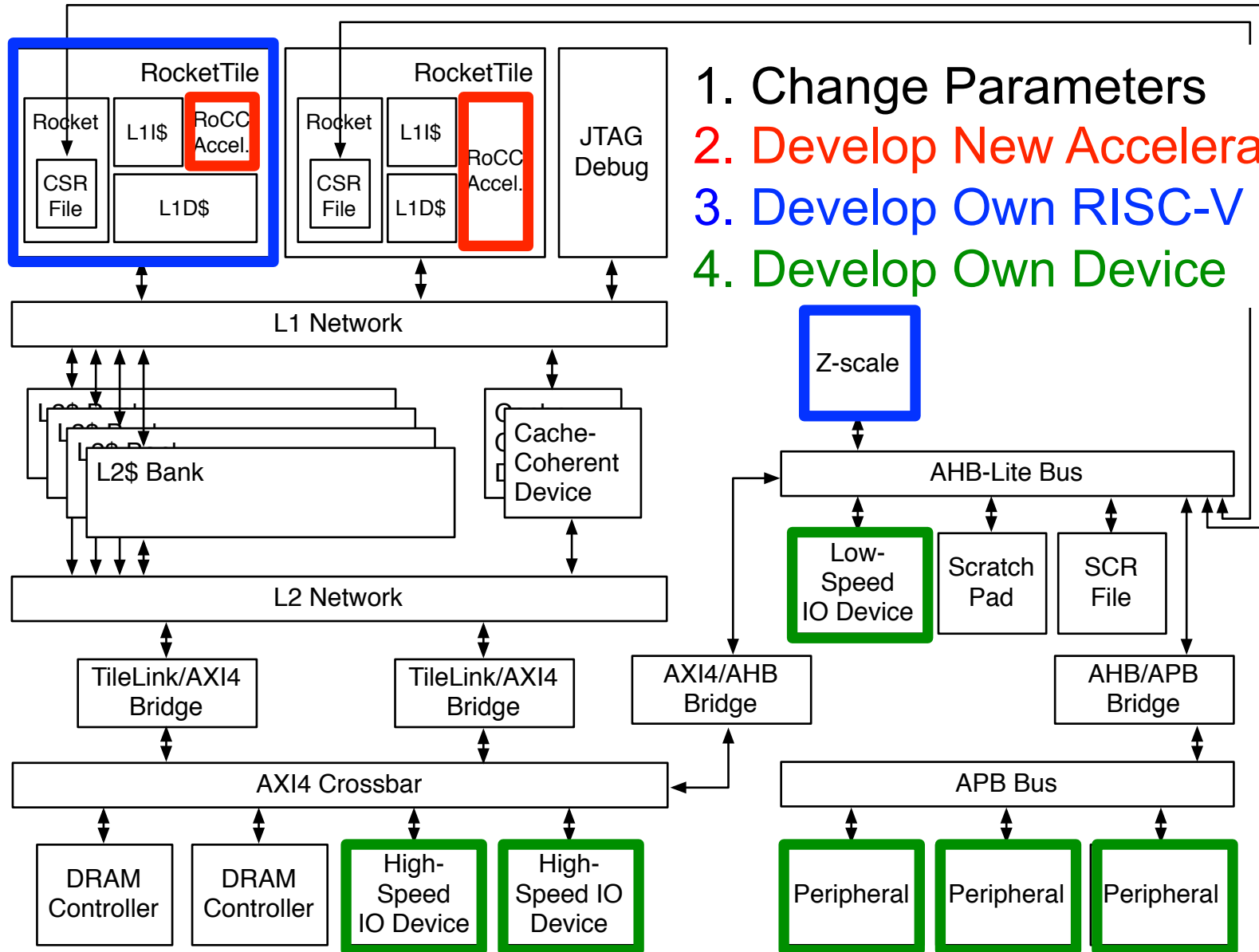
- Smallest static code size
- Fewest dynamic bytes fetched
- Comparable microarchitectural work per program
- While being the simplest ISA by far



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

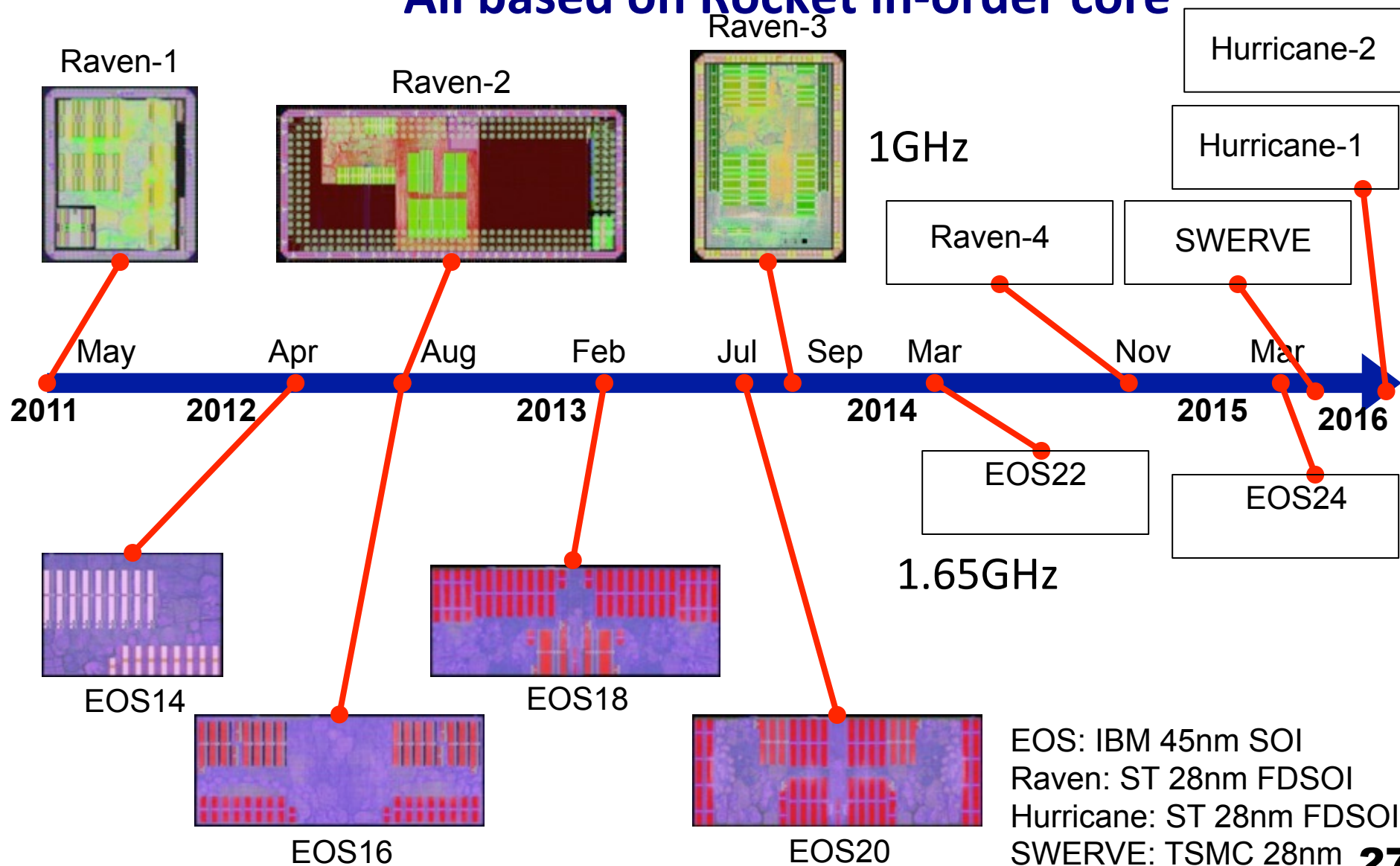
Rocket Chip Generator



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

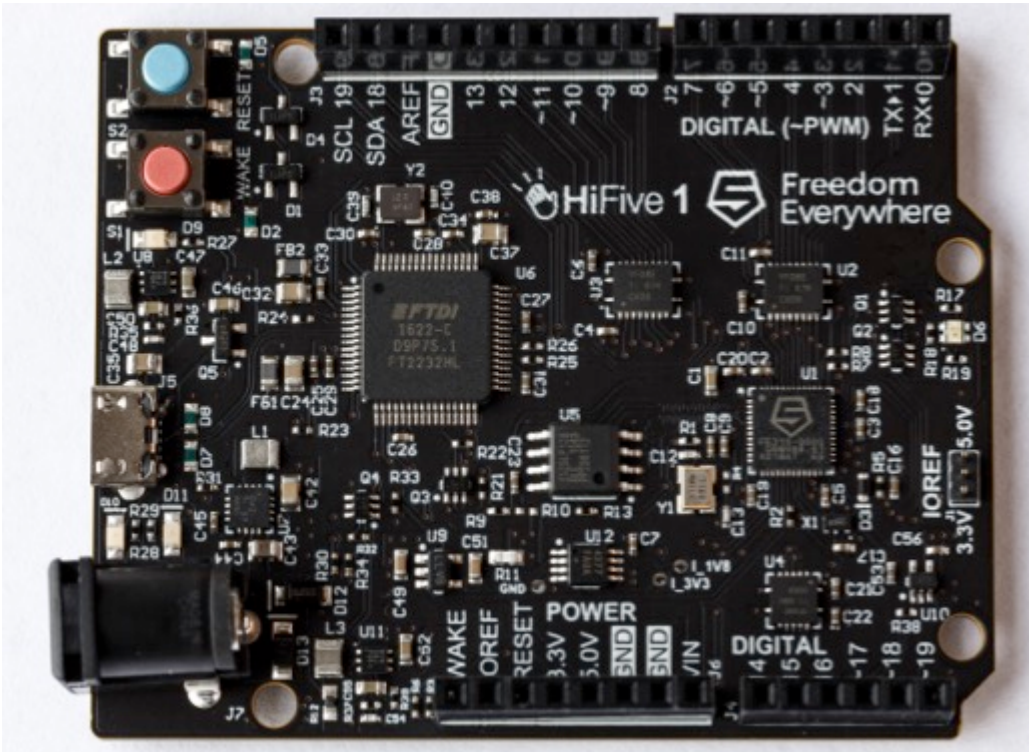


UC Berkeley RISC-V Cores: Seven 28nm & Six 45nm RISC-V Chips Tapeouts All based on Rocket in-order core



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

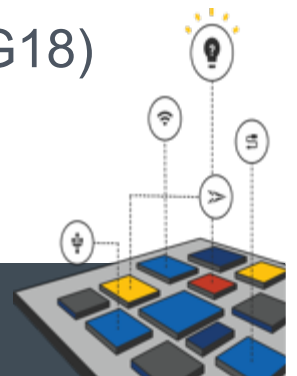
<https://www.crowdsupply.com/sifive/hifive1>



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

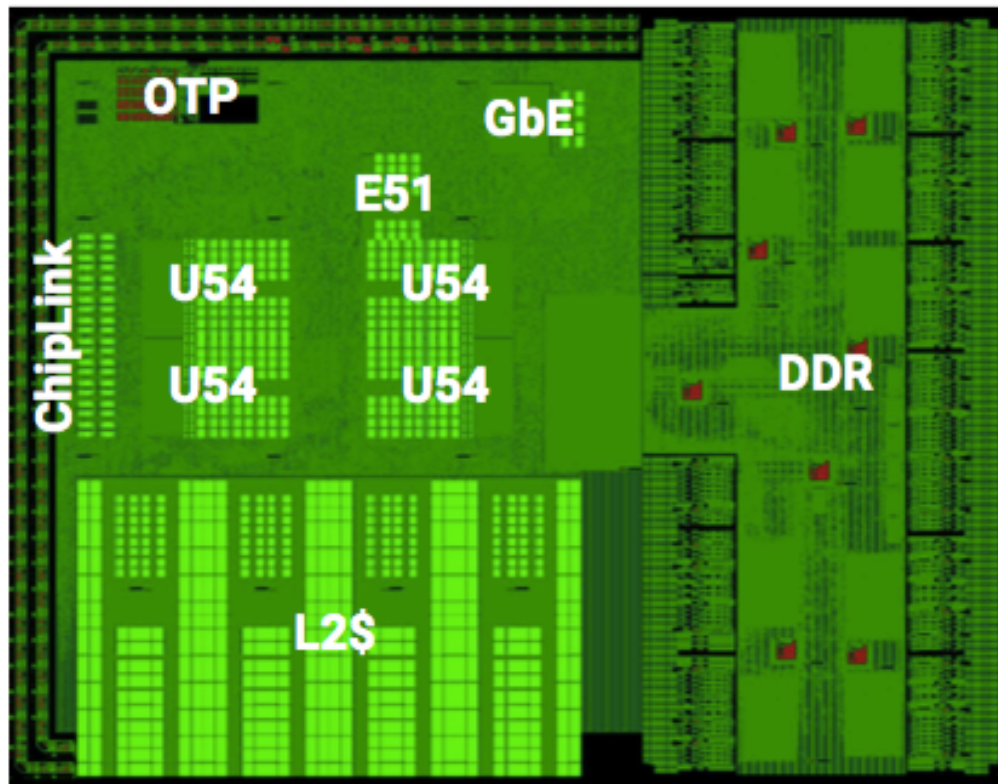
- Adopted as “standard ISA” for **India**
 - IIT-Madras \$90M funding to build 6 different open-source cores
 - C-DAC \$45M funding to build 2GHz quad-core
- **NVIDIA** selected RISC-V for on-chip microcontrollers
- **LowRISC** project based in Cambridge, UK producing open-source RISC-V Rocket-based 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, Cudasip, 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





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



Platinum:

Foundation Members (70+)



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

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

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

COMPUTER SYSTEMS DESIGN
COMPUTER HARDWARE



PATTERSON
HENNESSY

COMPUTER ORGANIZATION AND DESIGN

RISC-V
EDITION

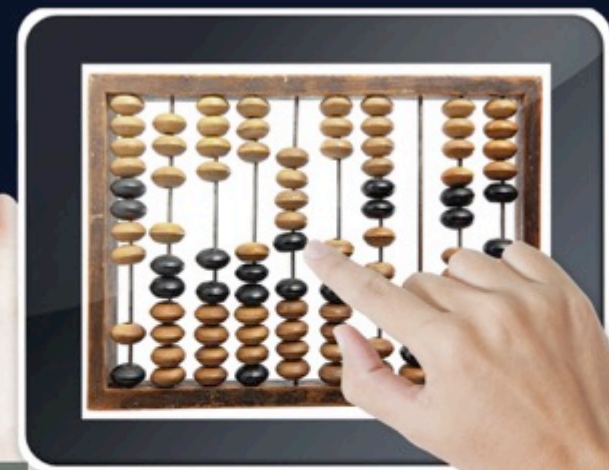
MK

MK
MORGAN KAUFMANN

COMPUTER ORGANIZATION AND DESIGN

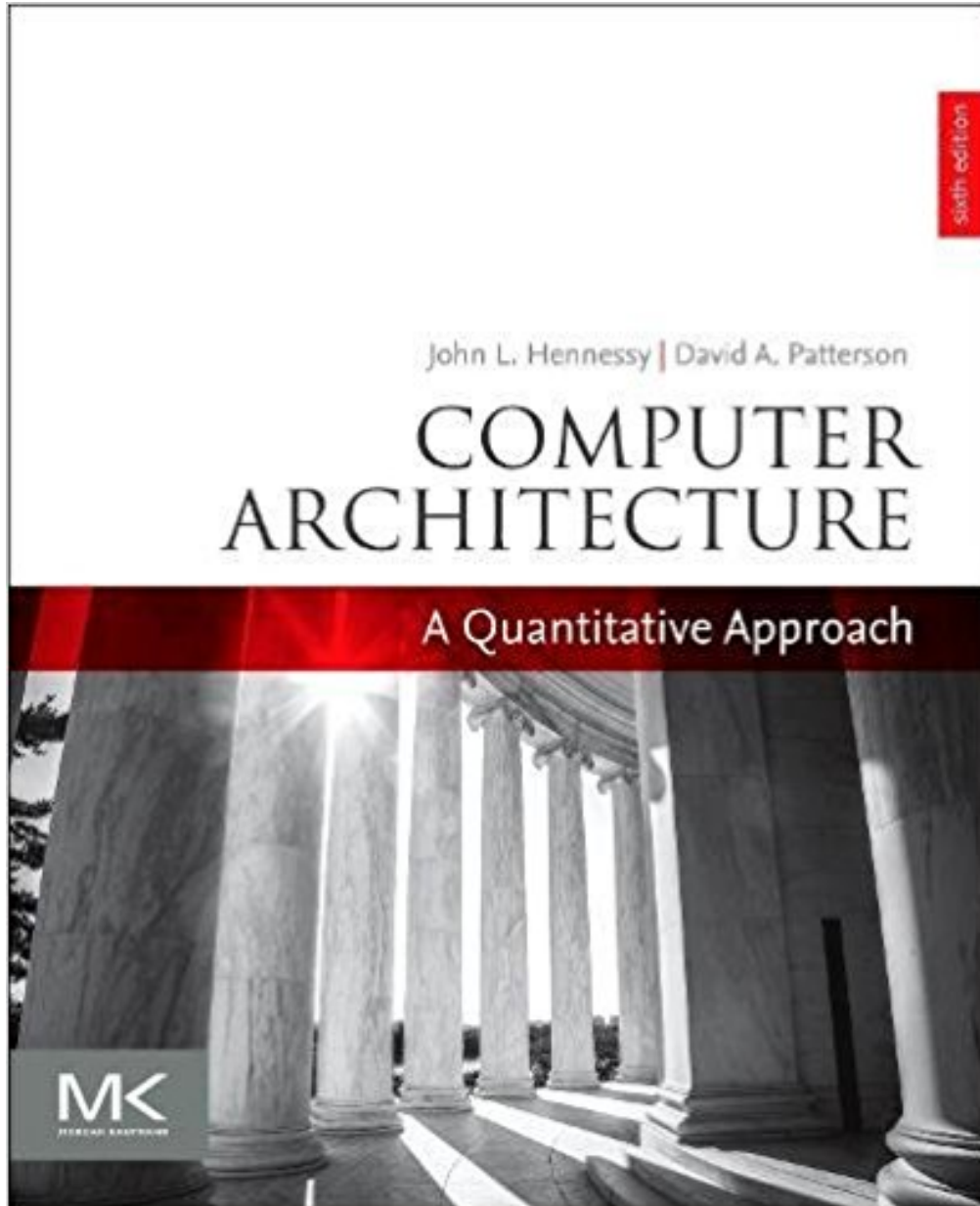
THE HARDWARE/SOFTWARE INTERFACE

RISC-V EDITION



DAVID A. PATTERSON
JOHN L. HENNESSY

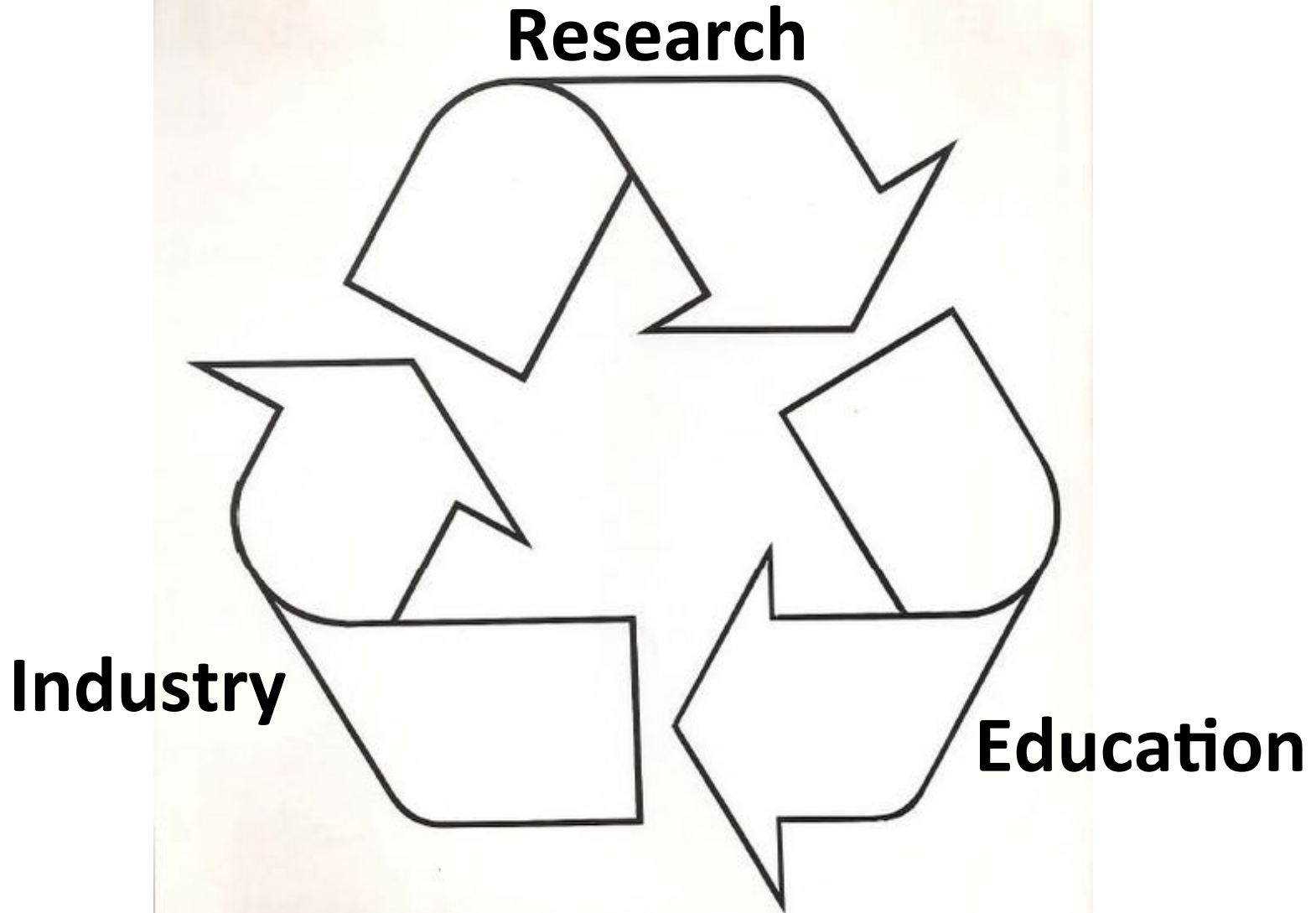
Hennessy & Patterson, 6th Edition



- 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