RISC-V

Instruction Sets Want to be Free!

Andrew Waterman
SiFive, Inc.
andrew@sifive.com

ASPIRE End of Project Celebration
December 5, 2017

Instruction Set Architectures don’t matter

Most of the performance and energy running software on a computer is due to:

- Algorithms
- Application code
- Compiler
- OS/Runtimes
- ISA (Instruction Set Architecture)
- Microarchitecture (core + memory hierarchy)
- Circuit design
- Physical design
- Fabrication process
- In a *system*, there’s also displays, radios, DC/DC converters, sensors, actuators, …

ISAs do matter

- Most important interface in computer system
- Large cost to port and tune all ISA-dependent parts of a modern software stack
- Large cost to recompile/port/QA all supposedly ISA-independent parts of stack
- If using proprietary closed-source, don’t have code
- Lost your own source code

- Most of the cost of developing a new chip is developing software for it

So…

If choice of ISA doesn’t have much impact on system energy/performance, and it costs a lot to use different ones

Why isn’t there a free, open standard ISA that everyone can use for everything?

Universal ISA Requirements

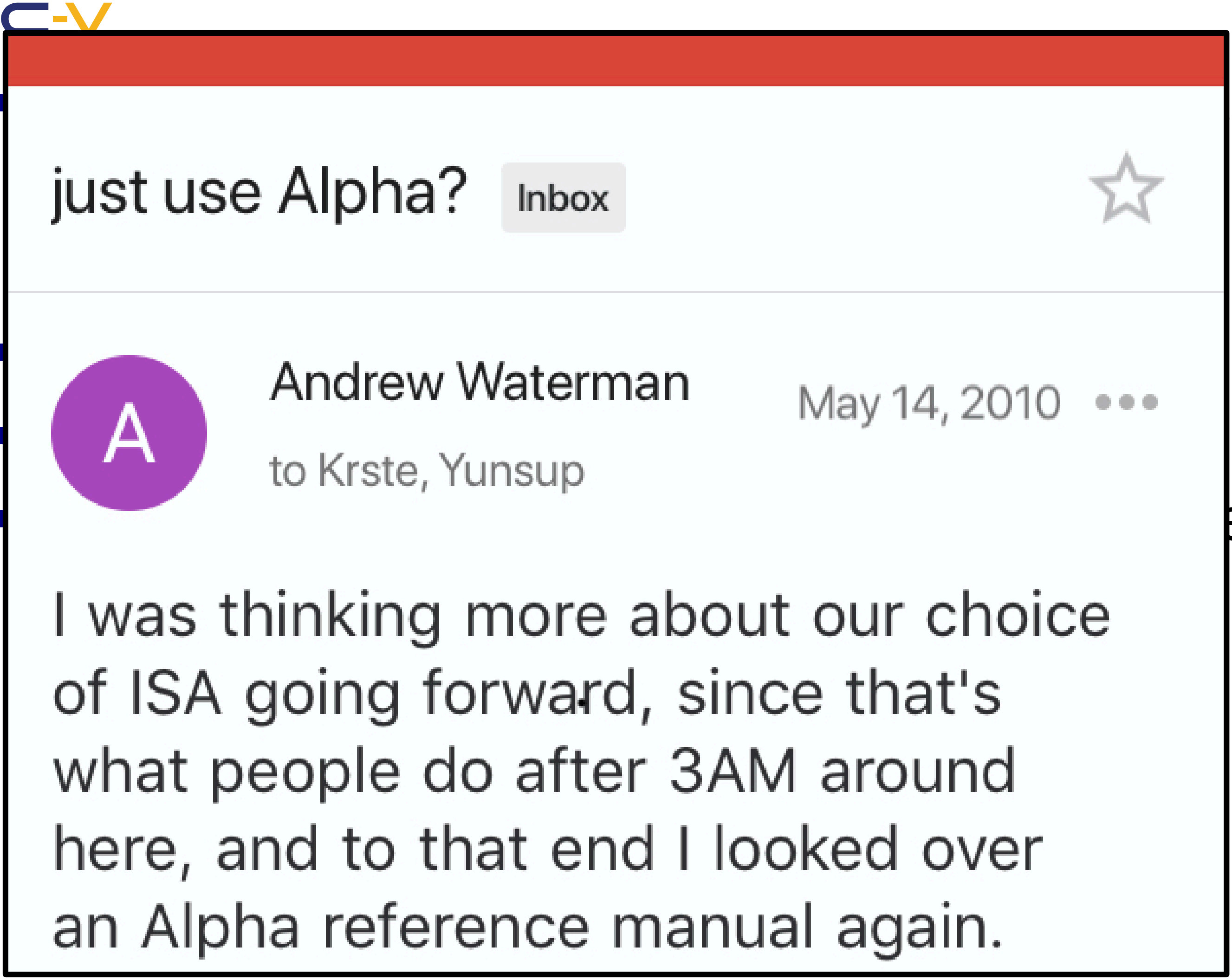
- Works well with existing software stacks, languages
- Is native hardware ISA, not virtual machine/ANDF
- Suits all sizes of processor, from smallest microcontroller to largest supercomputer
- Suits all implementation technologies, FPGA, ASIC, full-custom, future device technologies…
- Efficient for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

RISC-V Origin Story

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, time to look at ISA for next set of projects
- Obvious choices: x86 and ARM
- x86 impossible – too complex, IP issues
- ARM mostly impossible – no 64-bit, complex, IP issues

RISC-V Origin Story

- ts using
  time to
-
-
- ex, IP



just use Alpha?

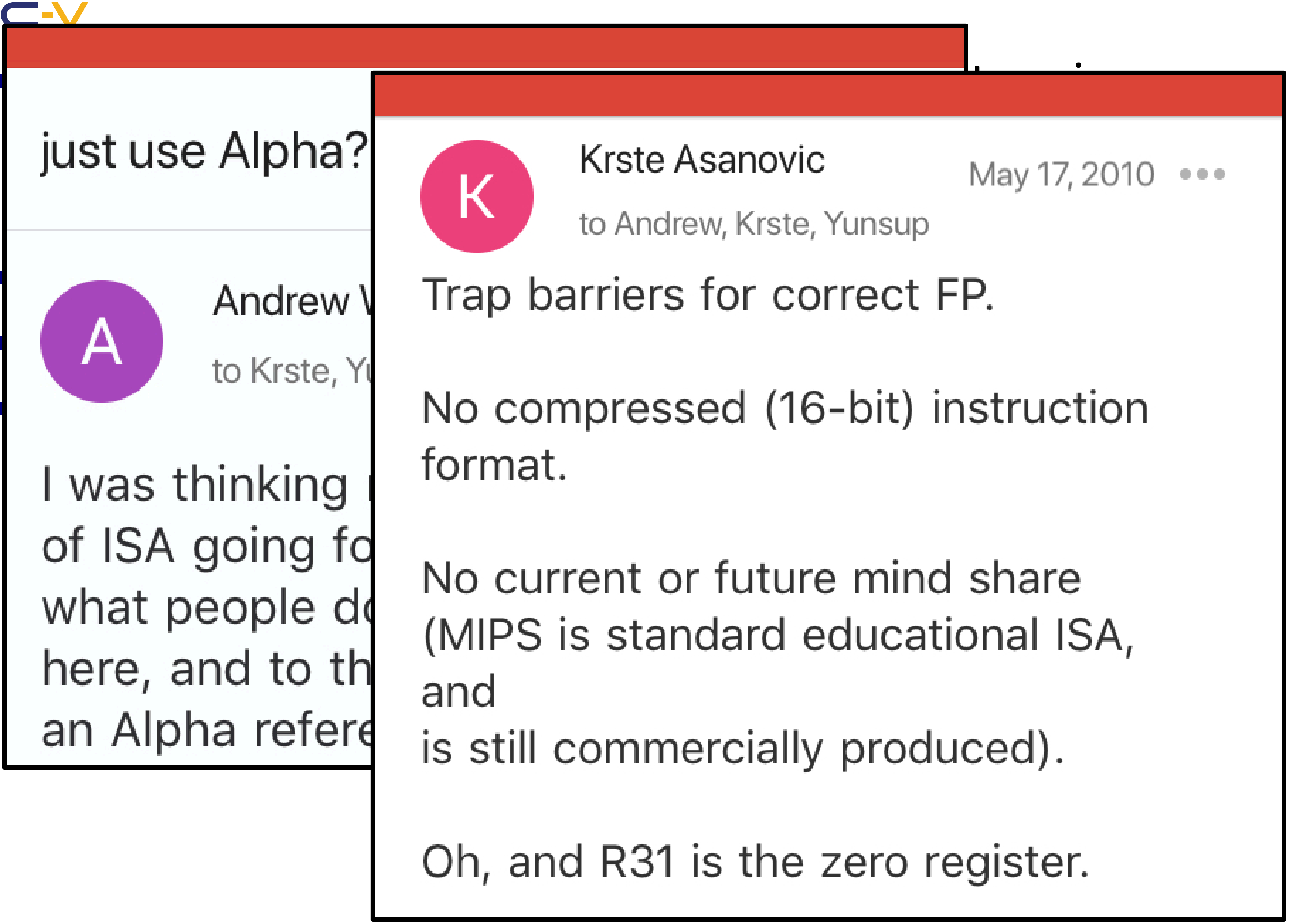
Andrew Waterman

to Krste, Yunsup

May 14, 2010

I was thinking more about our choice of ISA going forward, since that's what people do after 3AM around here, and to that end I looked over an Alpha reference manual again.

RISC-V Origin Story



just use Alpha?

Andrew W

to Krste, Yu

I was thinking
of ISA going fo
what people do
here, and to th
an Alpha refere

Krste Asanovic

May 17, 2010

to Andrew, Krste, Yunsup

Trap barriers for correct FP.

No compressed (16-bit) instruction format.

No current or future mind share (MIPS is standard educational ISA, and
is still commercially produced).

Oh, and R31 is the zero register.

RISC-V Origin Story

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, time to look at ISA for next set of projects
- Obvious choices: x86 and ARM
- x86 impossible – too complex, IP issues
- ARM mostly impossible – no 64-bit, complex, IP issues
- 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
    - But also many tapeouts and several research publications along the way

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

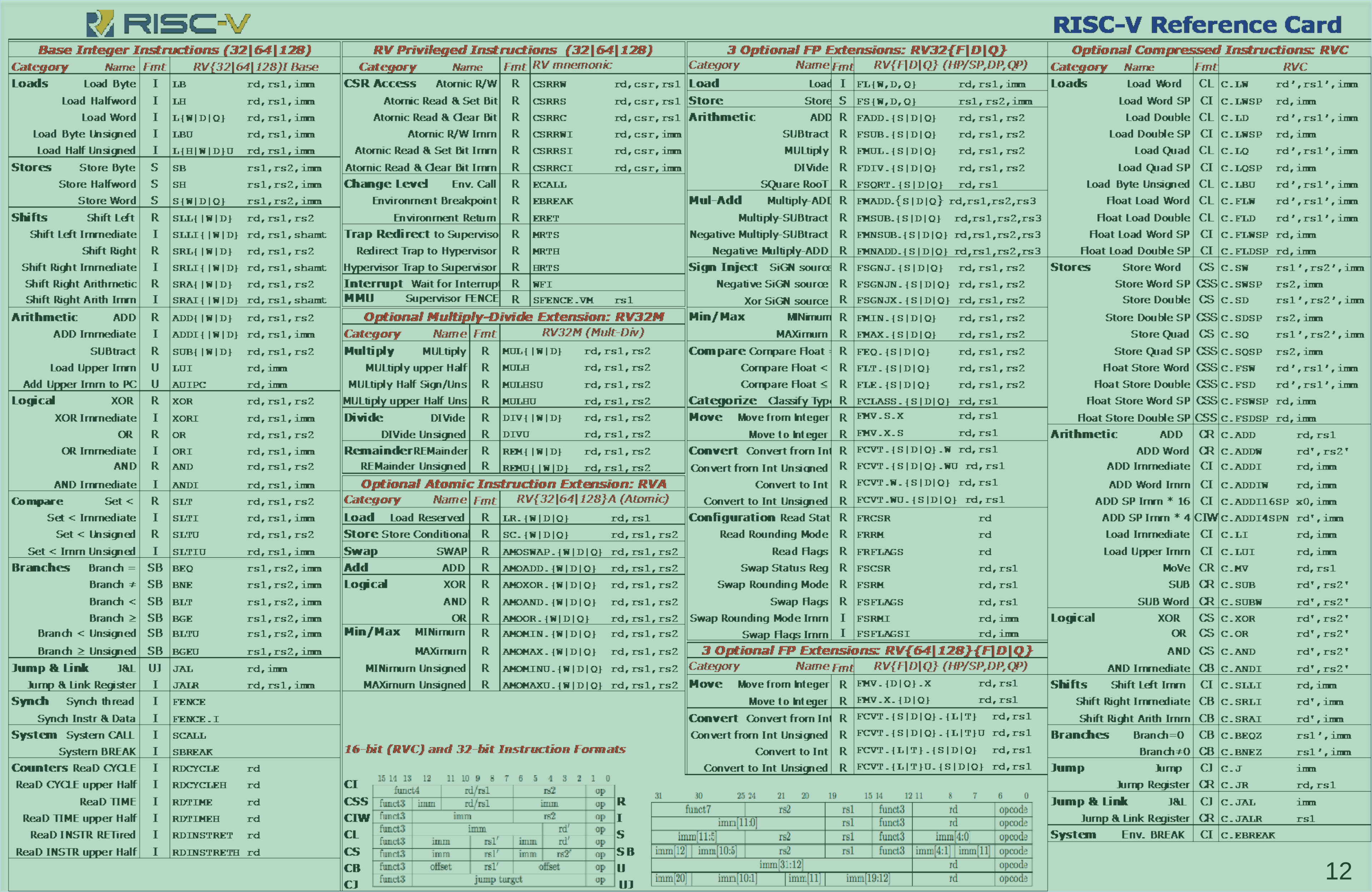
What’s Different about RISC-V?

- *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

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



RISC-V Reference Card

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 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

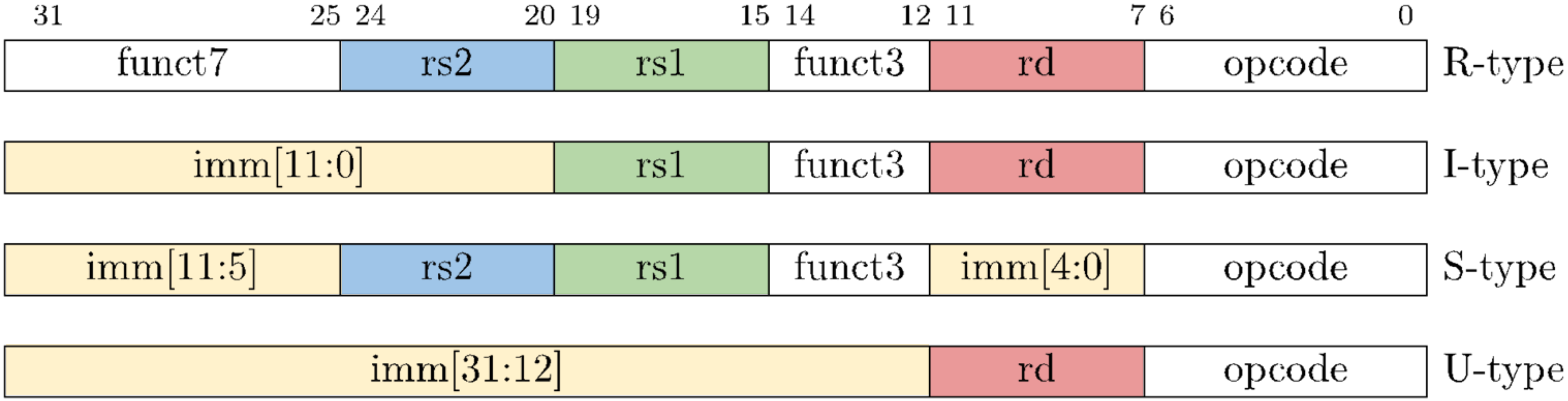
Category	Name	Fmt	RVC	
Loads	Load Word	CL	C.LW	rd′,rs1′,imm
	Load Word SP	CI	C.LWSP	rd,imm
	Load Double	CL	C.LD	rd′,rs1′,imm
	Load Double SP	CI	C.LWSP	rd,imm
	Load Quad	CL	C.LQ	rd′,rs1′,imm
	Load Quad SP	CI	C.LQSP	rd,imm
	Load Byte Unsigned	CL	C.LBU	rd′,rs1′,imm
	Float Load Word	CL	C.FLW	rd′,rs1′,imm
	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

CI	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
	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

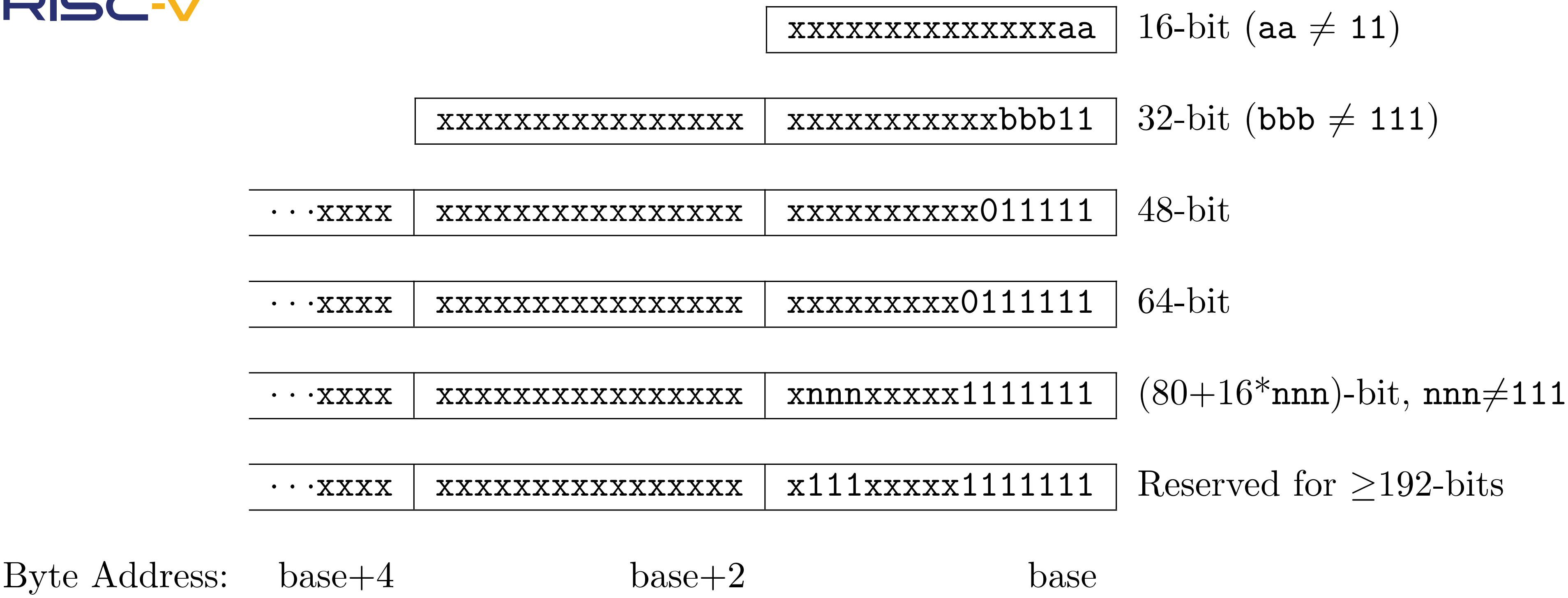
RISC-V Standard Base ISA Details



31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[31:12]				rd	opcode	U-type

- 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

Variable-Length Encoding



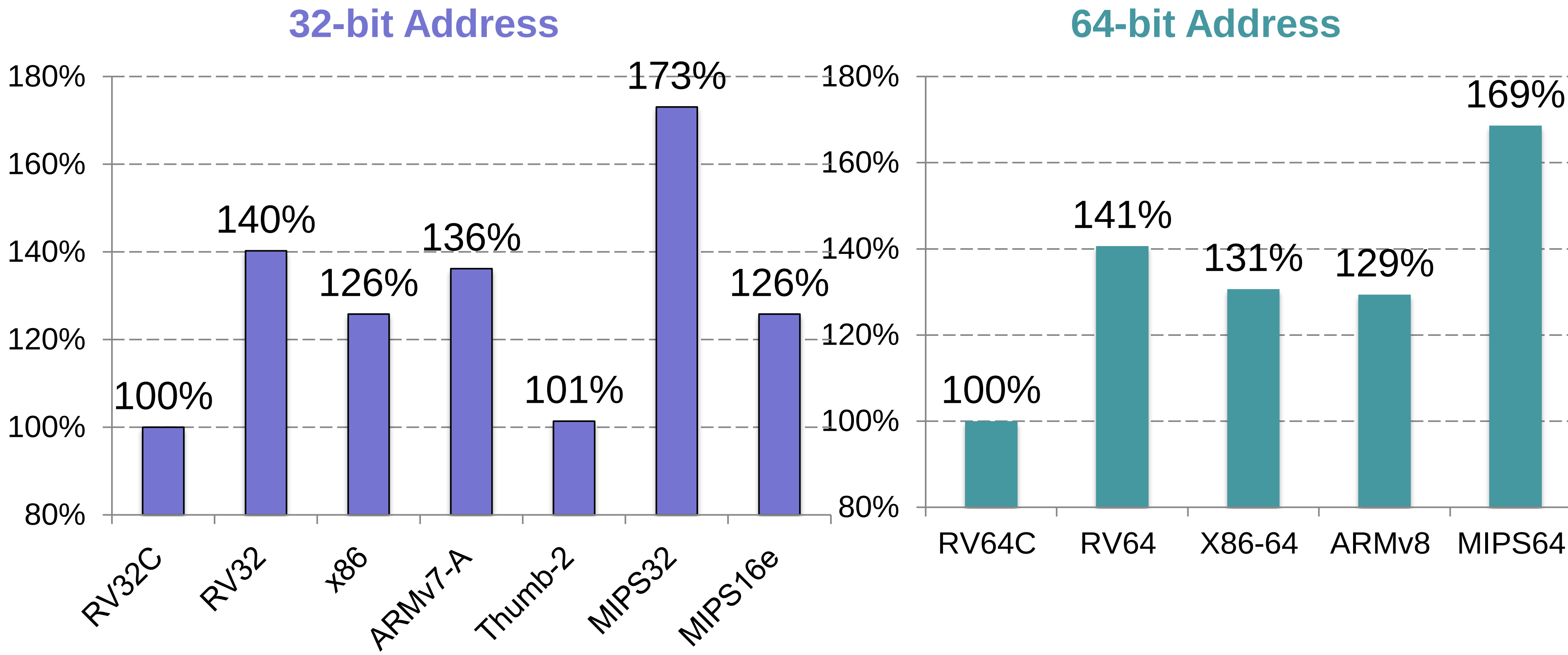
Byte Address:	base+4	base+2	base	
			xxxxxxxxxxxxxxaa	16-bit (aa ≠ 11)
		xxxxxxxxxxxxxxxx	xxxxxxxxxxxbbb11	32-bit (bbb ≠ 111)
	···xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxxx011111	48-bit
	···xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxx0111111	64-bit
	···xxxx	xxxxxxxxxxxxxxxx	xnnnxxxxx1111111	(80+16*nnn)-bit, nnn≠111
	···xxxx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

- 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

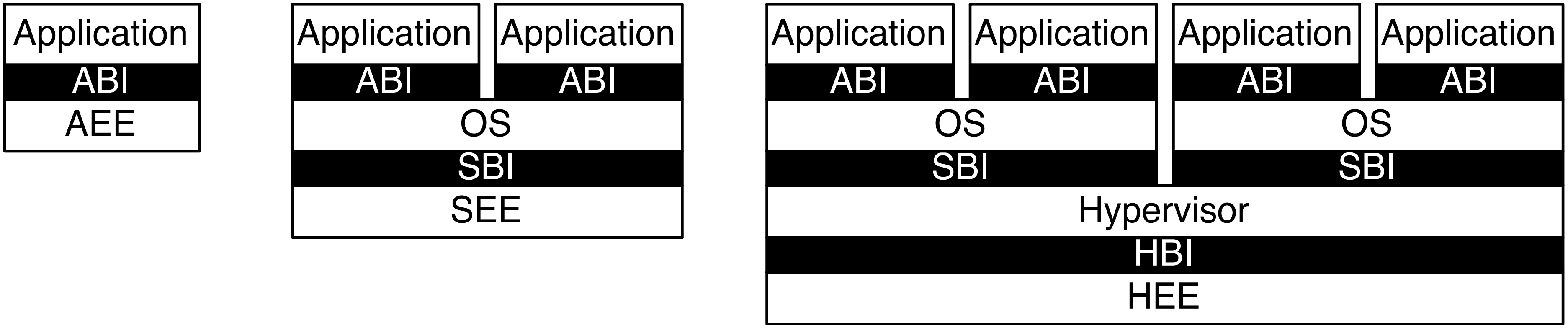
- Compressed code important for:
  - low-end embedded to save static code space
  - high-end commercial workloads to reduce cache footprint
- C extension adds 16-bit compressed instructions
  - 2-operand instructions only
  - Most instructions can only access 8 registers
- 1 compressed instruction expands to 1 base instruction
  - Assembly lang. programmer & compiler oblivious
- 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

RISC-V Privileged Architecture



- Modular design supports many classes of system
  - Simple embedded systems with no protection
  - Embedded systems with protection
  - Unix-class systems with page-based virtual memory
  - Hypervised Unix systems with two-level paging

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
- Now >100 members
- 10s of companies participating in standards definition

Berkeley Architecture Research
RISC-V
DRAPER
bluespec
C-SKY
Mentor A Siemens Business
LATTICE SEMICONDUCTOR
Mellanox TECHNOLOGIES
NXP
cortus
Rambus
Cryptography Research
SiFive
Micron
SAMSUNG
Google
IBM
Western Digital
Microsemi
NVIDIA
QUALCOMM
HUAWEI

RISC-V Foundation: 100+ Members

ESPRESSIF
MEDIATEK
ANDES TECHNOLOGY
antmicro EMBEDDED SYSTEMS
runtime.io
DOVER MICROSYSTEMS
BAE SYSTEMS
Esperanto Technologies
VeriSilicon
inside secure DRIVING TRUST
GRAHAM RESEARCH
IDT
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
ISTUARY INNOVATION GROUP
SEGGER It simply works!
NUS National University of Singapore
谨睿电子科技 WWW.GLORYSPACE.CN
GOWIN
GRIDÖÖG "Eyes on the Grid, Eyes on the Buck"
LAUTERBACH DEVELOPMENT TOOLS
INTRINSIX
ETH zürich
MIT CSAIL
ICT 中科院计算所
ASHLING THE DEVELOPMENT SYSTEMS COMPANY
codasip
imperas
Roa Logic Silicon Proven IP for FPGA and ASIC
GREENWAVES TECHNOLOGIES
SHC SH CONSULTING
MINIMA PROCESSOR
Rumble Development
TRINAMIC
XtremeEDA
VectorBlox embedded supercomputing
UBILITE Hello IoT World
SEAGATE
EMDALO TECHNOLOGIES
EMBECOSM
SECURE RF Securing the Internet of Things
BERKELEY LAB
Syntacore Custom cores and tools
lowRISC
TechanaLye
SIEMENS
优矽科技 UC TECH IP
Technolution
ultrasoc
csem
PRINCETON UNIVERSITY
Blockstream
dxcorr
GLOBALFOUNDRIES
expresslogic

Summary: Why RISC-V?

- Free and open architecture, no proprietary lock-in
- Much simpler ISA than others
- Readily and freely extensible
- Usable as base ISA for every core on SoC

- RISC-V project goal: *become the industry-standard ISA for all computing devices*

- Thank you for sponsoring this research!

Questions?