RISC-V®

Hot Chips Tutorial, Part-I: RISC-V Overview and ISA Design

RISC-V®

Krste Asanovic
Prof. EECS, UC Berkeley;
Chairman of the Board,
RISC-V Foundation;
Co-Founder and Chief Architect,
SiFive Inc.

Stanford, CA
August 18, 2019

@risc_v

Why Instruction Set Architecture matters

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

ISA is most important interface in computer system

where software meets hardware

Open Interfaces Work for Software!

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

- Why not successful free & open standards and free & open implementations, like other fields?

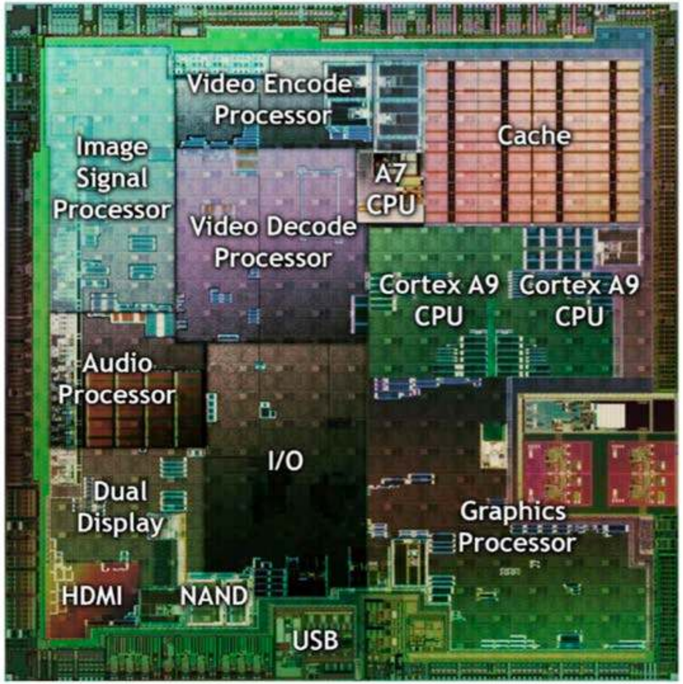
Companies and their ISAs Come and Go

Proprietary ISA fortunes tied to business fortunes and whims

- Digital Equipment Corporation
  - **PDP-11**, **VAX**, **Alpha**
- Intel
  - **i960**, **i860**, **Itanium**
- **MIPS**
  - Sold to Imagination, then bought by Wave AI startup, now opening R6?
- **SPARC**
  - Was opened by Sun, acquired by Oracle, now closed down
- **ARM**
  - Sold to Softbank at >40% premium
  - Now 25% sold off to Abu Dhabi investment fund

Today, many ISAs on one SoC

- Applications processor (usually ARM)
- Graphics processors
- Image processors
- Radio DSPs
- Audio DSPs
- Security processors
- Power-management processor
- *> dozen ISAs on some SoCs – each with unique software stack*



NVIDIA Tegra SoC

Why?

- Apps processor ISA too big, inflexible for accelerators
- IP bought from different places, each proprietary ISA
- Engineers build home-grown ISA cores

Do we need all these different ISAs?

Must they be proprietary?

Must they keep disappearing?

What if there was one stable free and open ISA everyone could use for everything?

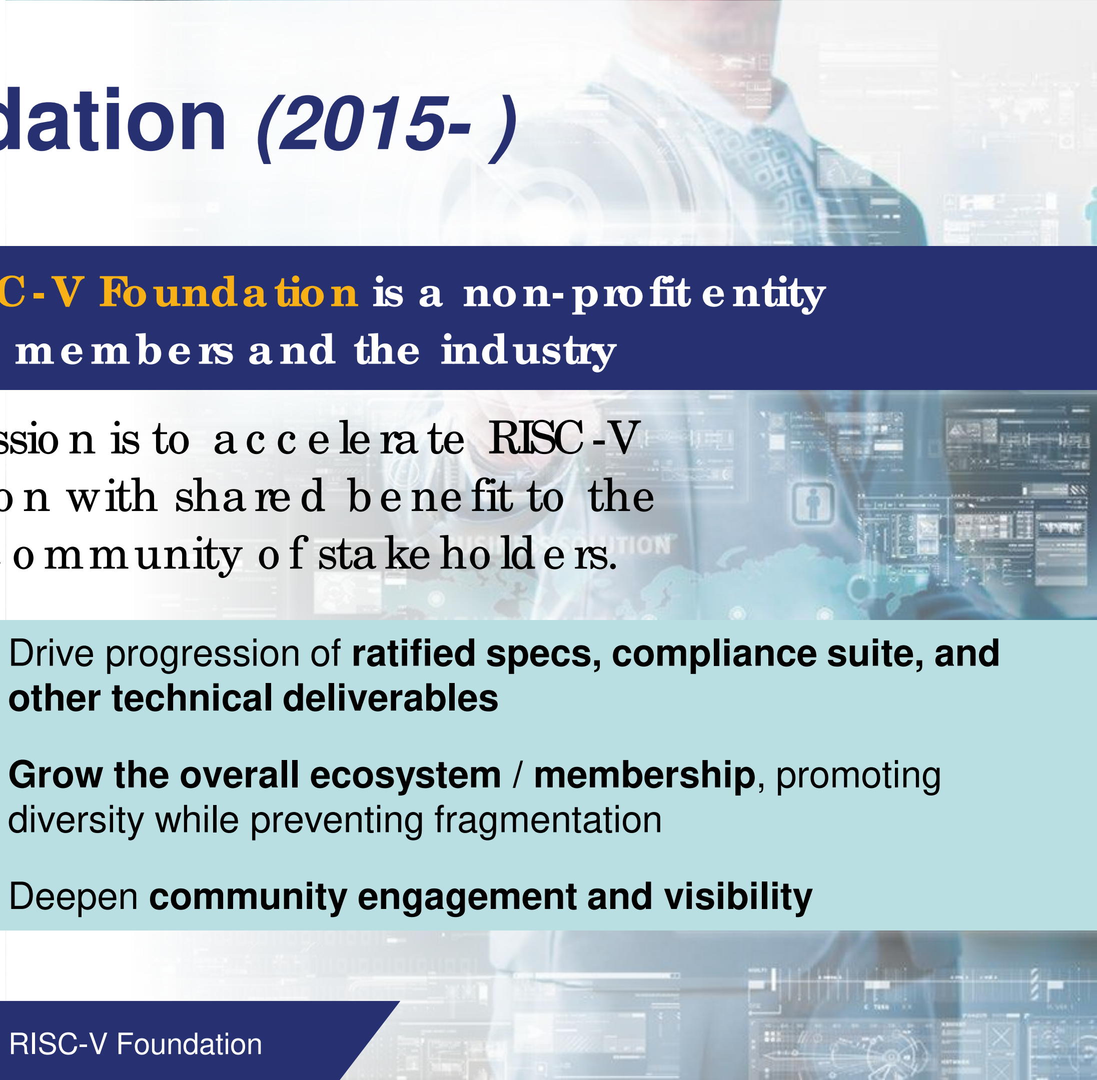
RISC-V Background

- In 2010, after many years and many research projects using MIPS, SPARC, and x86, time for architecture group at UC Berkeley to choose ISA for next set of projects
- Obvious choices: x86 and ARM
  - x86 impossible – too complex, IP issues
  - ARM mostly impossible – complex, no 64-bit in 2010, IP issues
- So we started “3-month project” during summer 2010 to develop clean-slate ISA
  - Principal designers: Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanovic
- Four years later, May 2014, released frozen base user spec
  - many tapeouts and several research publications along the way
- Name RISC-V (pronounced “risk-five”) represents fifth major Berkeley RISC ISA

Hot Chips 2014

RISC-V Foundation *(2015-)*

- **RISC-V** is the open-source hardware Instruction Set Architecture (ISA)
- Frozen base user spec released in 2014, contributed, ratified, and openly published by the RISC-V Foundation

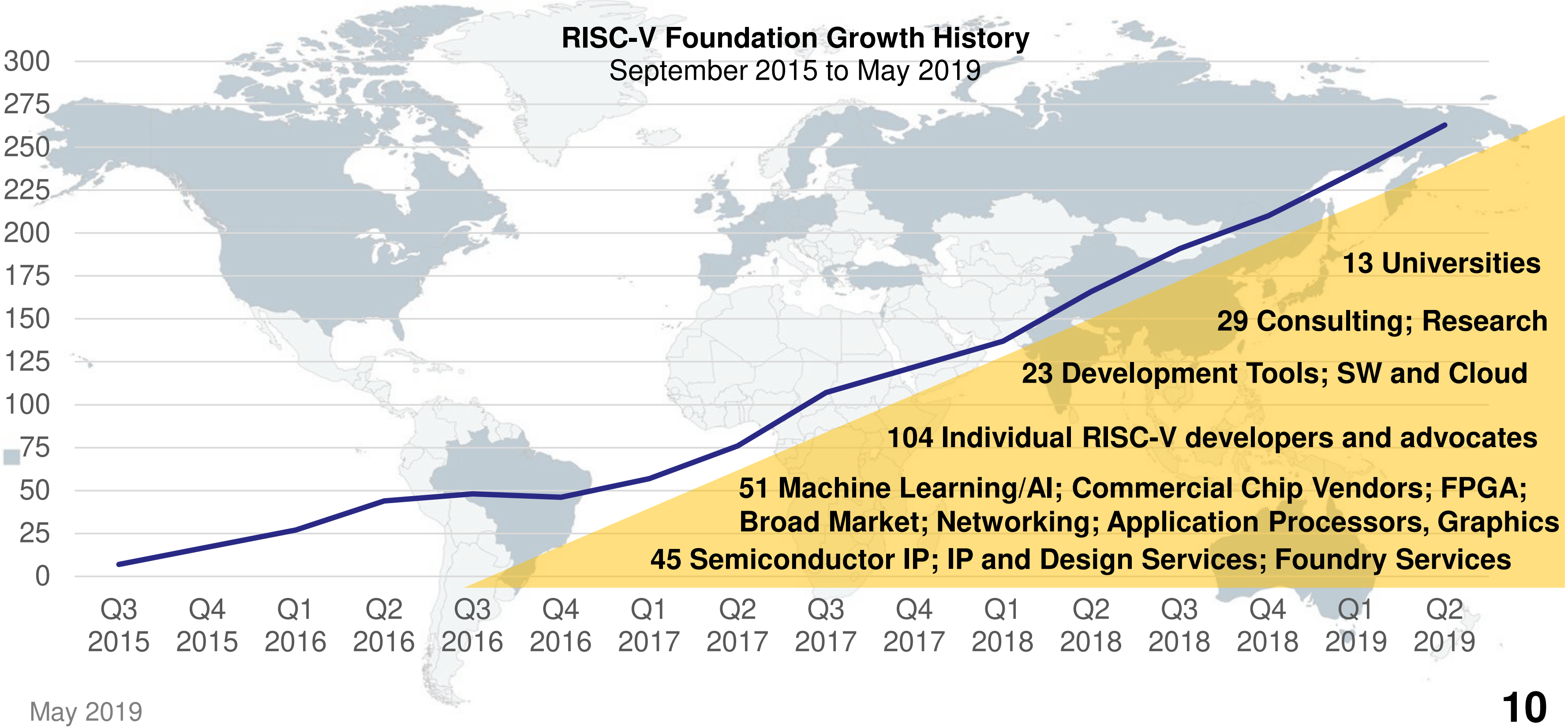


The RISC-V Foundation is a non-profit entity serving members and the industry

Our mission is to accelerate RISC-V adoption with shared benefit to the entire community of stakeholders.

- ✓ Drive progression of **ratified specs, compliance suite, and other technical deliverables**
- ✓ **Grow the overall ecosystem / membership**, promoting diversity while preventing fragmentation
- ✓ Deepen **community engagement and visibility**

More than 250 RISC-V Members in 28 Countries Around the World



13 Universities

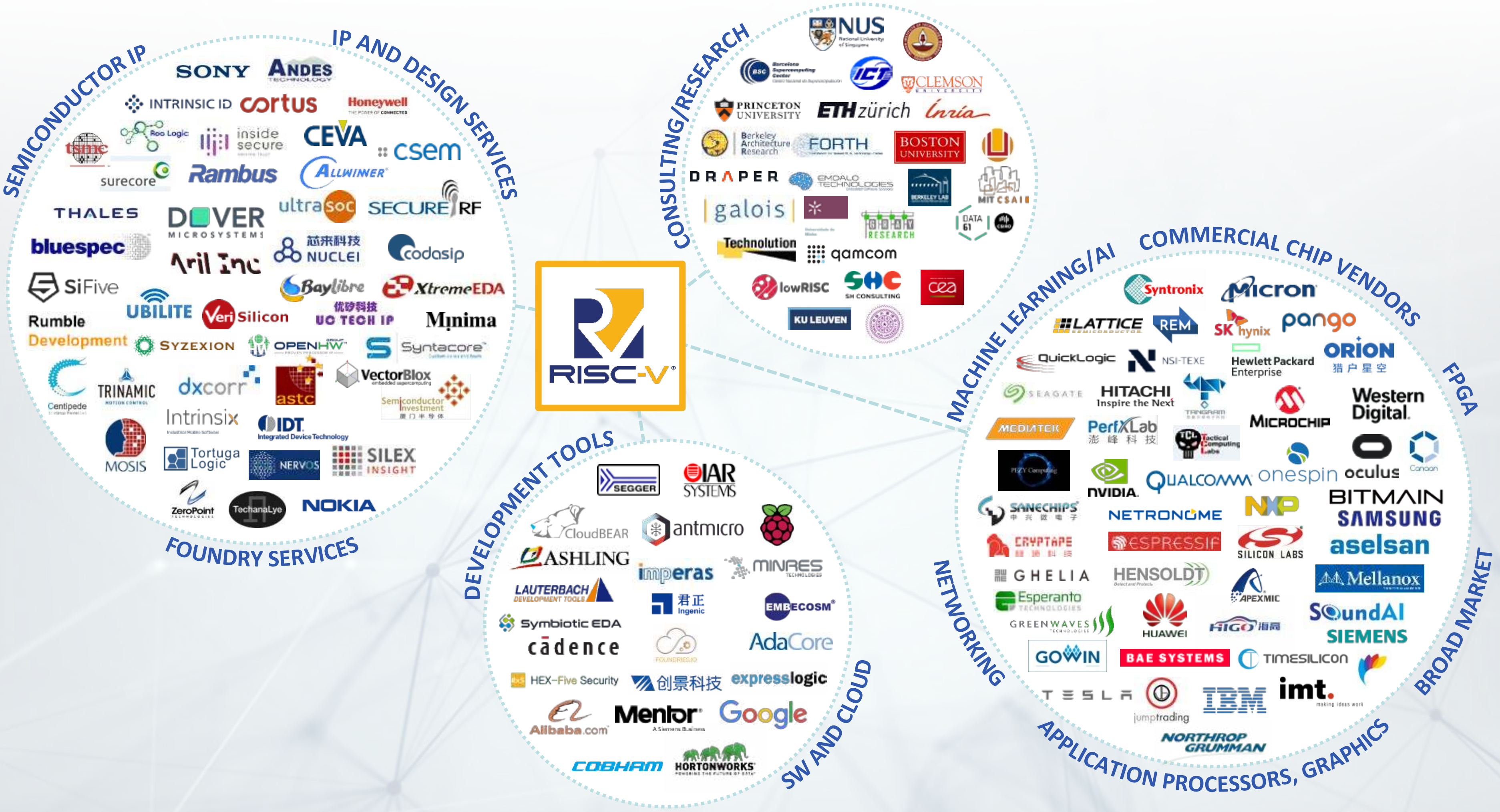
29 Consulting; Research

23 Development Tools; SW and Cloud

104 Individual RISC-V developers and advocates

51 Machine Learning/AI; Commercial Chip Vendors; FPGA; Broad Market; Networking; Application Processors, Graphics

45 Semiconductor IP; IP and Design Services; Foundry Services



SEMICONDUCTOR IP

IP AND DESIGN SERVICES

FOUNDRY SERVICES

CONSULTING/RESEARCH

DEVELOPMENT TOOLS

SW AND CLOUD

MACHINE LEARNING/AI

COMMERCIAL CHIP VENDORS

FPGA

BROAD MARKET

APPLICATION PROCESSORS, GRAPHICS

NETWORKING

Calista Redmond, CEO, RISC-V Foundation

Previously, Vice-President of IBM Z Ecosystem division; President of OpenPOWER Foundation.

Programs increase member value + engagement

Technical Deliverables

- Guard against fragmentation
- Manage and progress technical deliverables through work groups and development team
- Process and initiate technical work groups
- Develop and manage member sandbox portal

Compliance + Certification

- Develop self serve testing and compliance certification suite
- Provide visibility to additional compliance certification and verification options

Visibility

- Drive constant drumbeat of member and foundation visibility through multiple media
- Engage in industry events and host Foundation events
- Cultivate strategic visibility through industry forums, analysts, and media

Learning + Talent

- Develop multi-level learning modules
- Connect universities, professors, and course material
- Develop badge and skill certification
- Match talent via online and event forums

Advocacy + Outreach

- Establish technical advocate program
- Engage geographic and domain specific engineers via advocate-led formal and informal opportunities
- Establish alliances with other organizations

Marketplace

- Provide online marketplace of providers and products
- Offer RFP matching to members

RISC-V Ecosystem

Software

Open-source software:
Gcc, binutils, glibc, Linux, BSD, LLVM, QEMU, FreeRTOS, ZephyrOS, LiteOS, SylixOS, …

Commercial software:
Lauterbach, Segger, IAR, Micrium, ExpressLogic, Ashling, Imperas, AntMicro, …

RISC-V Foundation

ISA specification | Golden Model | Compliance

Hardware

Open-source cores:
Rocket, BOOM, RI5CY, Ariane, PicoRV32, Piccolo, SCR1, Swerv, Hummingbird, …

Commercial core providers:
Andes, Bluespec, Cloudbear, Codasip, Cortus, C-Sky, Nuclei, SiFive, Syntacore, …

Inhouse cores:
Nvidia, WDC, Alibaba, +others

Why is RISC-V so popular?

- Engineers sometimes “*don’t see forest for the trees*”
- The movement is ***not*** happening because some benchmark ran 10% faster, or some implementation was 30% lower power
- The movement ***is*** happening because ***new business model*** changes everything
  - Pick ISA first, then pick vendor or build own core
  - Add your own extension without getting permission
- Implementation features/PPA will follow
  - Whatever is broken/missing in RISC-V will get fixed

Modest RISC-V Project Goal

Become the industry-standard ISA for all computing devices

Industry Adoption Status

- Large companies adopting RISC-V for deeply embedded controllers in their SoCs (“minion cores”)
  - 2016 NVIDIA announced all future GPUs will use RISC-V
  - 2017 Western Digital announced transition of all billion cores/year to RISC-V
  - Others waiting in the wings

CTOs across entire worldwide value chain of IC suppliers, system providers, service providers, are evaluating RISC-V strategies

RISC-V: An Everyday Design Choice

- For embedded/IoT, RISC-V is already strong competitor, and other areas adopting RISC-V also
- Production ramp starting, expect “millions” of SoCs to ship with RISC-V cores in 2019
- SiFive announced >100 RISC-V IP design wins
- Andes announced 21 wins in 2018, 60 in 2019

- Message: *You won’t get fired for choosing RISC-V!*

Replacing 2nd-tier ISAs

- Smaller proprietary-ISA soft-core IP companies switching to RISC-V standard to access larger market:
  - Andes
  - Codasip
  - Cortus
  - C-Sky
  - others to announce

If you're a softcore IP provider,
you should have a RISC-V product in development

Startups

- Many startups choosing RISC-V for new products
- Esperanto announced 4,096-core 7nm RISC-V chip, with high-end OoO cores
- Fadu SSD controller announcement
- Kendryte AI microcontroller, $3 chip with two RISC-V cores from open-source Rocket codebase
- Most are stealthy so will not be visible for a while

We haven't had to tell startups about RISC-V; they find out pretty quickly when shopping for processor IP

Commercial Ecosystem Providers

- Mainstream commercial ecosystem support rapidly appearing
  - Lauterbach, Micrium, Segger, IAR, Express Logic, Imperas, UltraSOC, AntMicro, …

Demand driving supply in commercial ecosystem

Government Adoption

- India has adopted RISC-V
- US DARPA mandated RISC-V in recent security call for proposals
- Israel Innovation Authority creating GenPro incubator around RISC-V
- Shanghai Municipal Govt supporting RISC-V companies
- Other governments at various stages of investigation

If your country wishes to control security of its own information infrastructure, and promote its indigenous semiconductor industry, support RISC-V

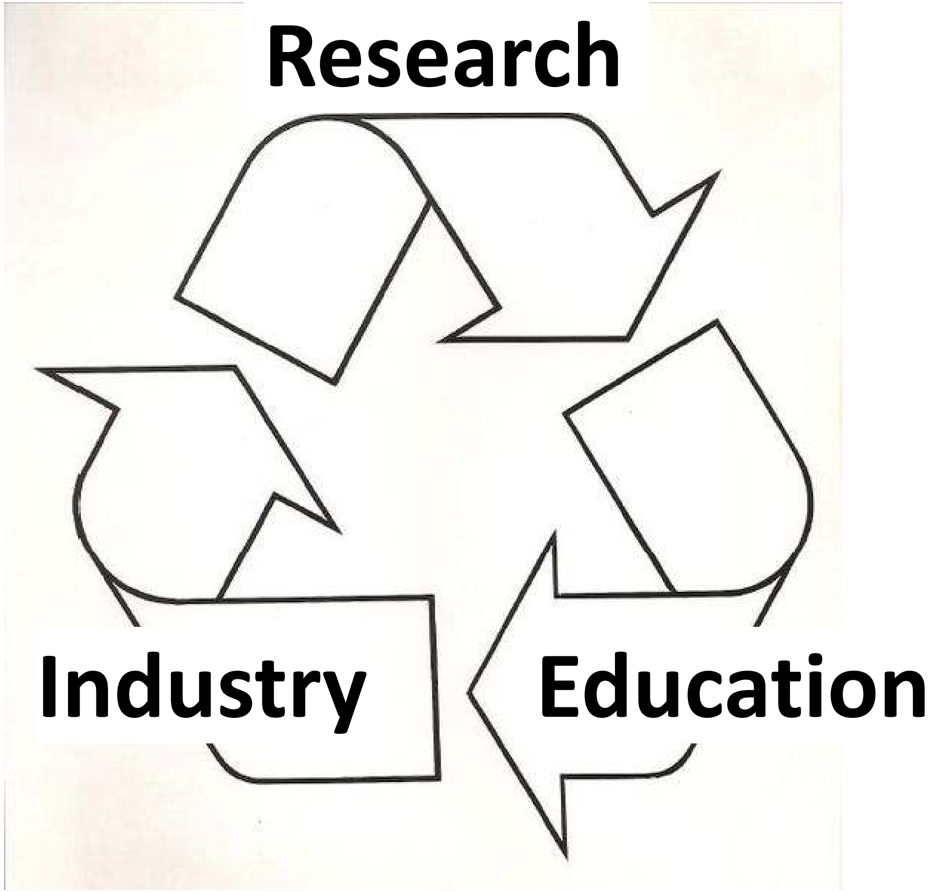
RISC-V in Education

Books available now!

In multiple languages

RISC-V spreading quickly throughout curricula of top schools

RISC-V: Completing the Innovation Cycle



Open ecosystem is key to keeping the virtuous cycle going

RISC-V ISA Tutorial

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
- *Modular* ISA designed for *extensibility/specialization*
  - Small standard base ISA, with multiple standard extensions
  - Sparse &variable-length instruction encoding for vast opcode space
- *Stable*
  - Base and first standard extensions are frozen
  - Additions via optional extensions, not new versions
- *Community designed*
  - Developed with leading industry/academic experts and software developers

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
- Above use standard RISC encoding in fixed 32-bit instruction word
- Frozen in 2014, ratified 2019, supported forever after

RISC-V ISA String Conventions

- RV32I
  - 32-bit address space, only basic integer instructions
- RV64IMAFDC (aka RV64GC)
  - 64-bit address space with integer multiply/divide, atomics, single and double precision floating-point and compressed
  - This is what current standard Linux distributions assume
- RV32EC *(RV32E not ratified yet)*
  - 32-bit address space with 16 integer registers and basic integer operations and compressed instructions
- RV128IMAFDQC *(RV128 not ratified yet)*
  - 128-bit address space with atomics, single/double/FP, and compressed instructions

RISC-V Processor Unprivileged State

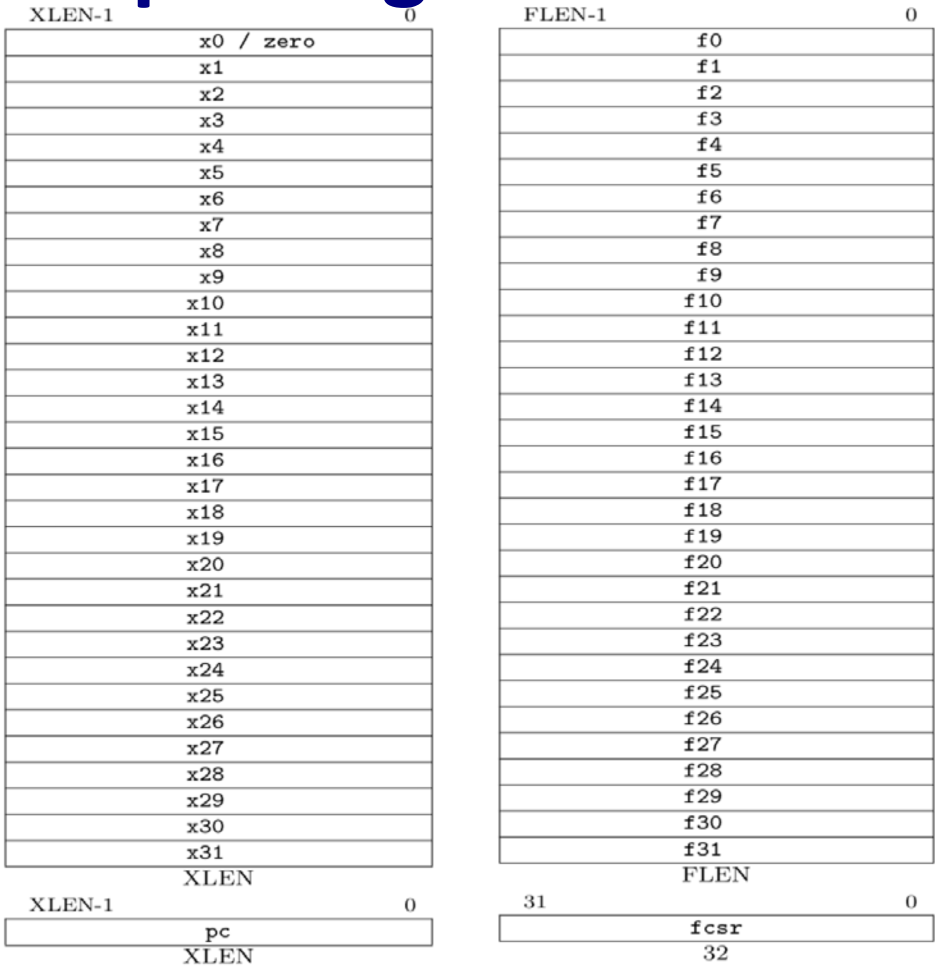
- XLEN=address width (32,64,128)

- XLEN-bit program counter (**pc**)

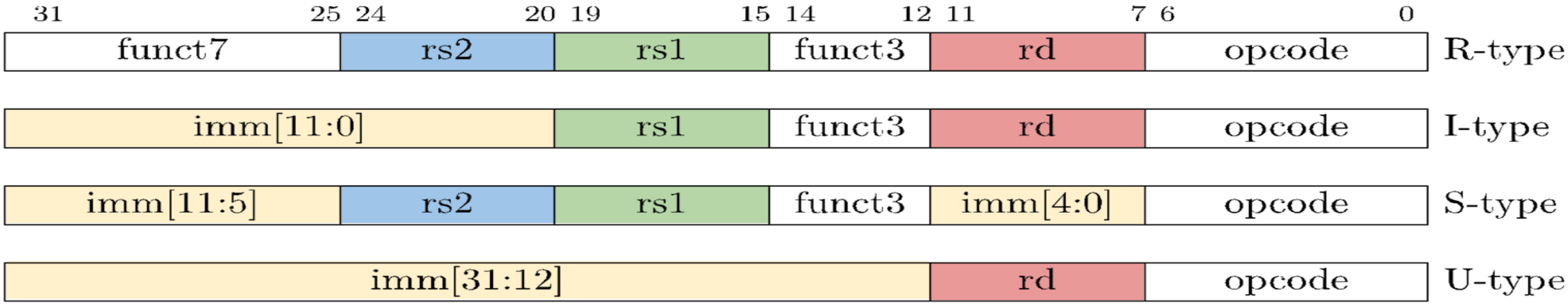
- 32 XLEN-bit integer registers (**x0-x31**)
  - **x0** always 0
  - RV32E variant has 16 registers (**x0-x15**)

- Optional 32 IEEE floating-point (FP) registers (**f0-f31**)
  - FLEN=floating-point width (extensions F=32,D=64,Q=128)

- FP status register (**fcsr**), used for FP rounding mode & exception reporting



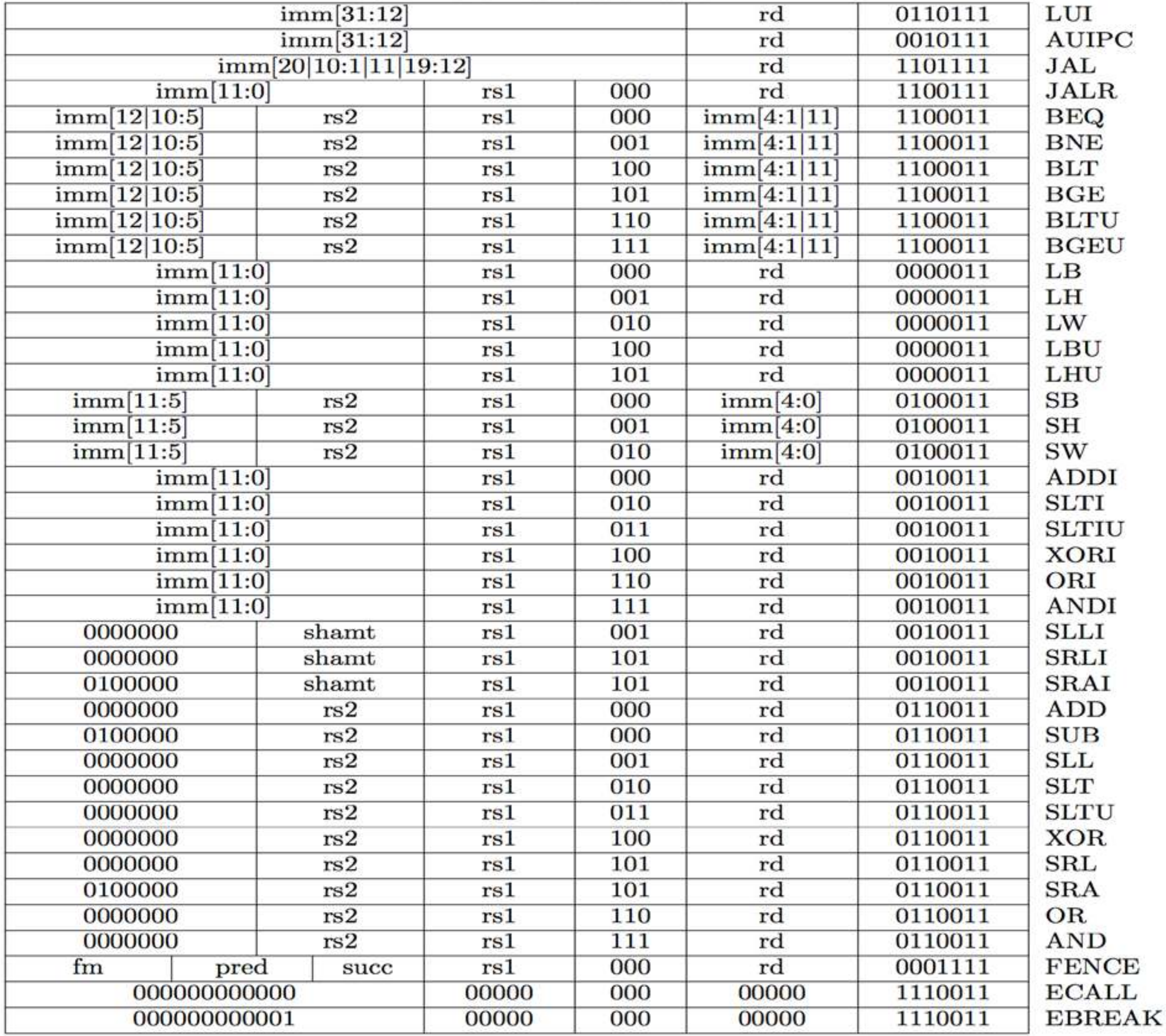
RISC-V Standard Base ISA Details



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 always sign-extended (from instr[31])
- Floating-point adds f0-f31 registers plus FP CSR, also fused mul-add four-register format
- Designed to support PIC and dynamic linking

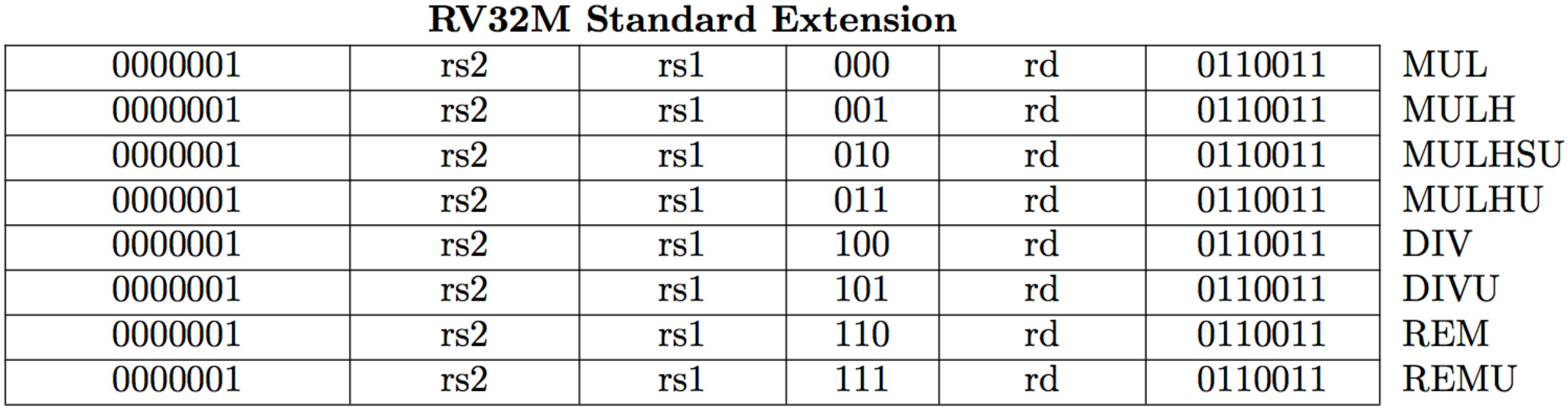
RV32I Base Unprivileged Instructions



imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm pred	succ	rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

“M” Integer Multiply-Divide Extension

- MUL returns lower XLEN of 2*XLEN multiply product
- MULH returns upper XLEN bits of signed product
- MULHU returns upper XLEN bits of unsigned product
- MULHSU returns upper XLEN bits of signed*unsigned product
- Implementation can fuse MUL+MULH{S}{U} for single microarch multiply



RV32M Standard Extension

0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

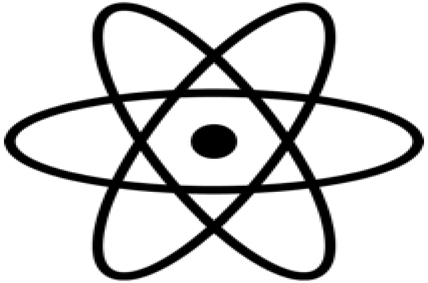
RISC-V Memory Model

- RISC-V has a base weak memory model (RVWMO)
  - Multi-copy atomic
    - store becomes visible to all other threads at same point
  - Similar to revised ARM v8 memory model
- Optional TSO extension defined (RVTSO)
  - Strictly upwards-compatible with RVWMO
  - Similar to x86 memory model
- Complete axiomatic and operational formal models available

“A”: Atomic Operations Extension

Two classes:

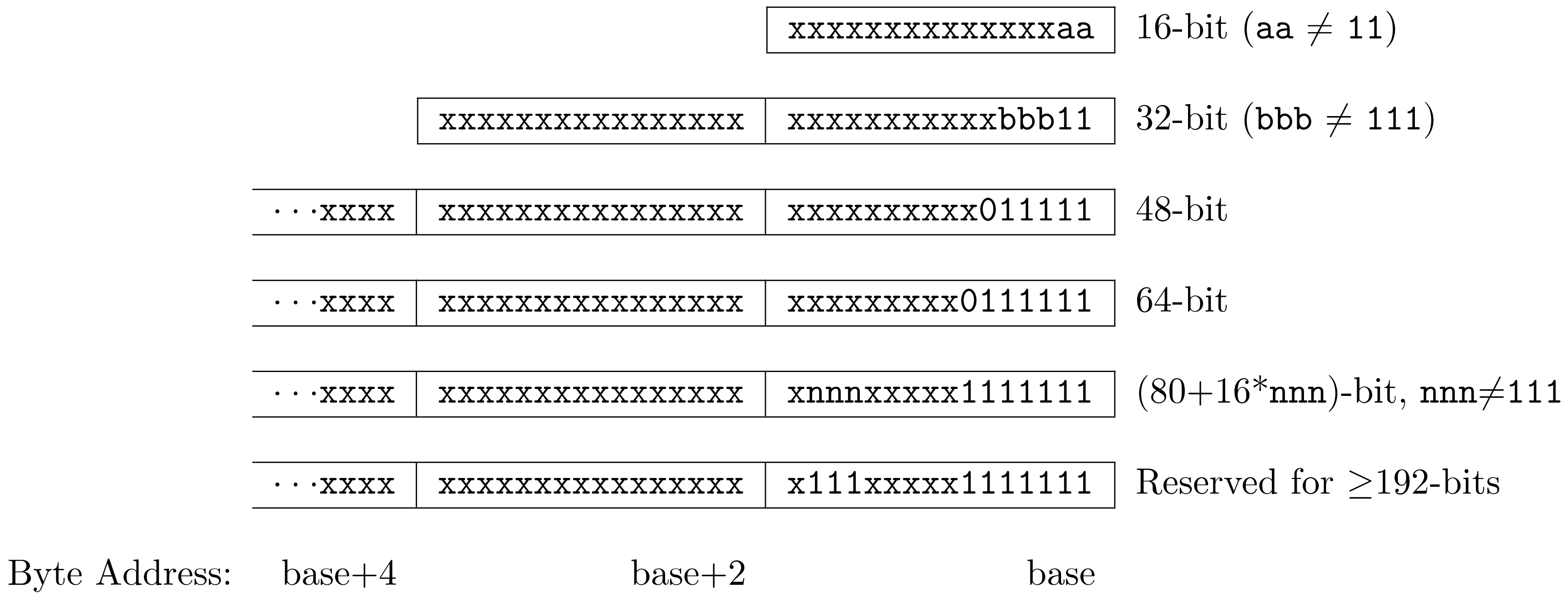
- Atomic Memory Operations (AMO)
  - Fetch-and-op, op=ADD,OR,XOR,MAX,MIN,MAXU,MINU
- Load–Reserved/Store Conditional
  - With forward-progress guarantee for short sequences
- All atomic operations can be annotated with two bits (Acquire/Release) to implement release consistency or sequential consistency



Floating-Point Extensions “F”,”D”,”Q”

- FP extensions add set of 32 FP registers f0-f31, width is FLEN
  - F = 32-bit single-precision IEEE FP (FLEN >= 32)
  - D = 64-bit double-precision IEEE FP (FLEN >= 64)
  - Q = 128-bit quad-precision IEEE FP (FLEN = 128)
  - Q implies D, D implies F
- Non-destructive fused multiply-adds supported
  - New instruction format with three sources and one destination
- Narrower FP results are “NaN-boxed” to wider FP regs
  - Result 1-extended to full FLEN width to avoid implementation-defined behavior, e.g., on RV64ID system, 32-bit FP result widened to FLEN=64 by filling upper 32 bits with all “1”s.
  - Narrower results treated as NaN if incorrectly used as source to wider FP instruction

Variable-Length Encoding



		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
Byte Address: base+4	base+2	base	

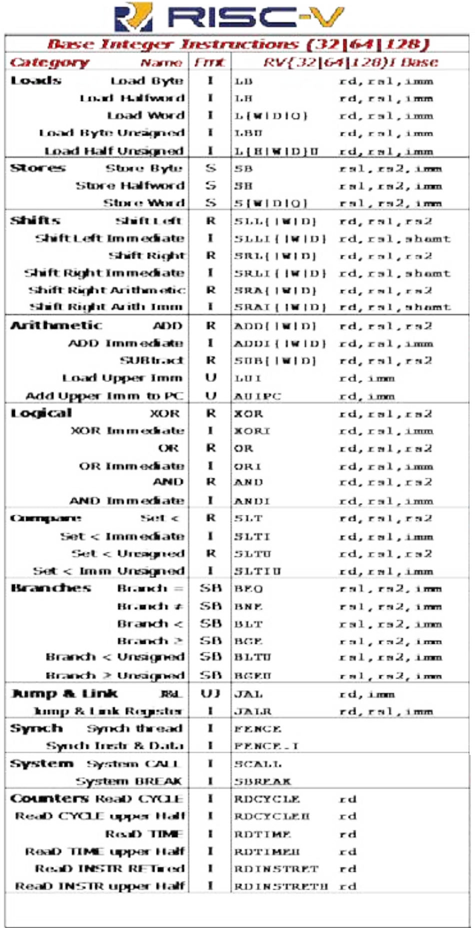
- Extensions can use any multiple of 16 bits as instruction length
- Branches/Jumps target 16-bit boundaries even in fixed 32-bit base
  - Consumes 1 extra bit of jump/branch address

“C”: Compressed Instruction Extension

- Compressed code important for:
  - low-end embedded to save static code space
  - high-end commercial workloads to reduce cache footprint
- C extension adds 16-bit compressed instructions
  - 2-address forms with all 32 registers
  - 2-address forms with most frequent 8 registers
- 1 compressed instruction expands to 1 base instruction
  - Assembly lang. programmer & compiler oblivious
  - Assembler and linker perform compression in current tool chains)
  - 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

RV32I

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

+14
Privileged

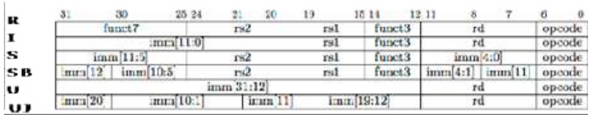
+ 8 for M

+ 11 for A

+ 34
for F, D, Q

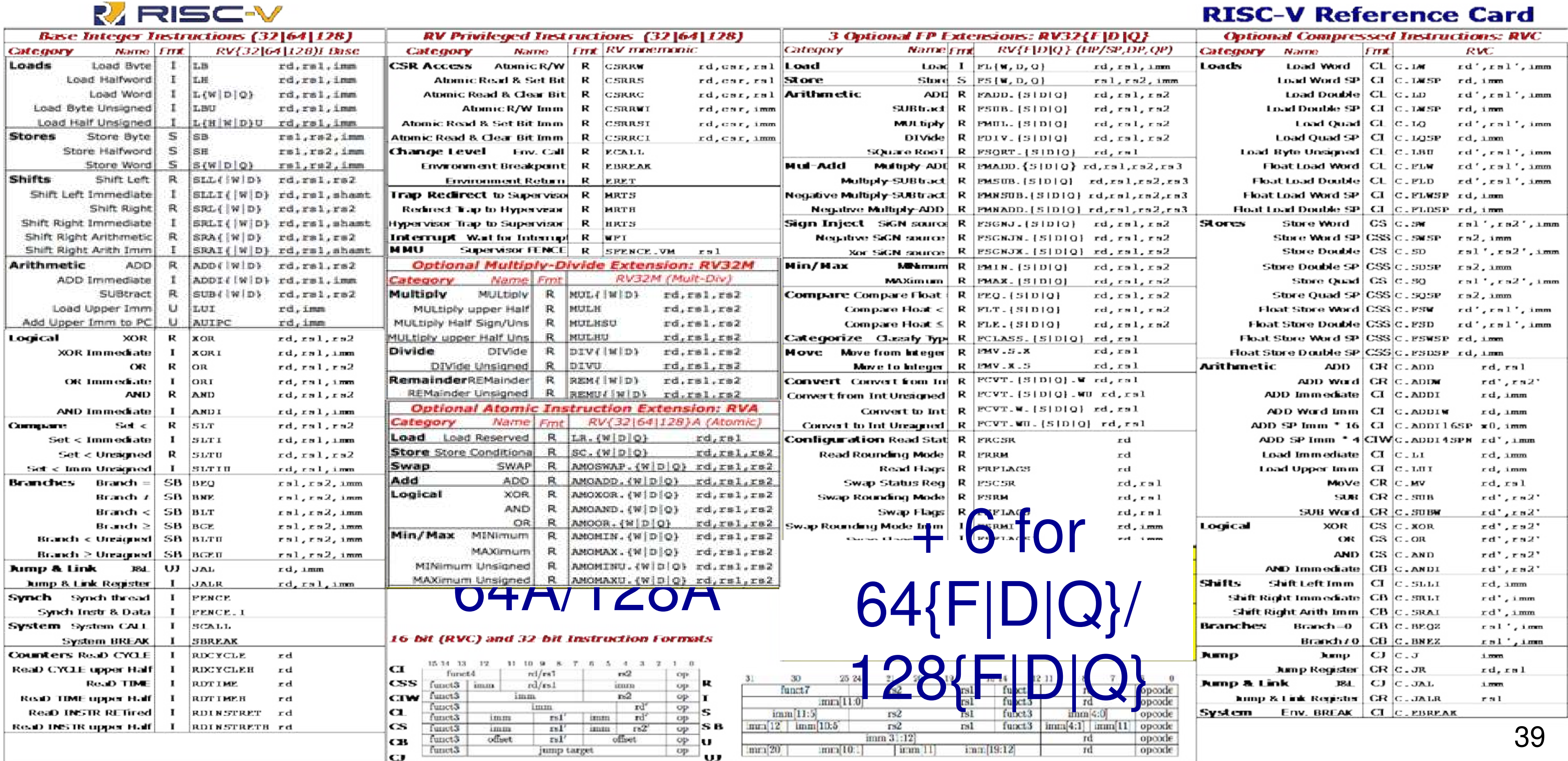
+ 46 for C

32-bit Instruction Formats



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

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



RISC-V

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 Supervisor	R	MRTS	
	Redirect Trap to Hypervisor	R	MRTH	
	Hypervisor Trap to Supervisor	R	HRTS	
Interrupt	Wait for Interrupt	R	WFI	
MMU	Supervisor FENCE	R	SFENCE.VM	rs1

Optional Multiply-Divide Extension: RV32M

Category	Name	Fmt	RV32M (Mult-Div)	
Multiply	MULtiply	R	MUL{|W|D}	rd,rs1,rs2
	MULtiply upper Half	R	MULH	rd,rs1,rs2
	MULtiply Half Sign/Uns	R	MULHSU	rd,rs1,rs2
	MULtiply upper Half Uns	R	MULHU	rd,rs1,rs2
Divide	DIVide	R	DIV{|W|D}	rd,rs1,rs2
	DIVide Unsigned	R	DIVU	rd,rs1,rs2
Remainder	REMainder	R	REM{|W|D}	rd,rs1,rs2
	REMainder Unsigned	R	REMU{|W|D}	rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

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

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

Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)	
Load	Load	I	FL{W,D,Q}	rd,rs1,imm
Store	Store	S	FS{W,D,Q}	rs1,rs2,imm
Arithmetic	ADD	R	FADD.{S|D|Q}	rd,rs1,rs2
	SUBtract	R	FSUB.{S|D|Q}	rd,rs1,rs2
	MULtiply	R	FMUL.{S|D|Q}	rd,rs1,rs2
	DIVide	R	FDIV.{S|D|Q}	rd,rs1,rs2
	SQuare RooT	R	FSQRT.{S|D|Q}	rd,rs1
Mul-Add	Multiply-ADD	R	FMADD.{S|D|Q}	rd,rs1,rs2,rs3
	Multiply-SUBtract	R	FMSUB.{S|D|Q}	rd,rs1,rs2,rs3
	Negative Multiply-SUBtract	R	FMNSUB.{S|D|Q}	rd,rs1,rs2,rs3
	Negative Multiply-ADD	R	FMNADD.{S|D|Q}	rd,rs1,rs2,rs3
Sign Inject	SiGN source	R	FSGNJ.{S|D|Q}	rd,rs1,rs2
	Negative SiGN source	R	FSGNJN.{S|D|Q}	rd,rs1,rs2
	Xor SiGN source	R	FSGNJX.{S|D|Q}	rd,rs1,rs2
Min/Max	MINimum	R	FMIN.{S|D|Q}	rd,rs1,rs2
	MAXimum	R	FMAX.{S|D|Q}	rd,rs1,rs2
Compare	Compare Float =	R	FEQ.{S|D|Q}	rd,rs1,rs2
	Compare Float <	R	FLT.{S|D|Q}	rd,rs1,rs2
	Compare Float ≤	R	FLE.{S|D|Q}	rd,rs1,rs2
Categorize	Classify Type	R	FCLASS.{S|D|Q}	rd,rs1
Move	Move from Integer	R	FMV.S.X	rd,rs1
	Move to Integer	R	FMV.X.S	rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.W	rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.WU	rd,rs1
	Convert to Int	R	FCVT.W.{S|D|Q}	rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{S|D|Q}	rd,rs1
Configuration	Read Status	R	FRCSR	rd
	Read Rounding Mode	R	FRRM	rd
	Read Flags	R	FRFLAGS	rd
	Swap Status Reg	R	FSCSR	rd,rs1
	Swap Rounding Mode	R	FSRM	rd,rs1
	Swap Flags	R	FSFLAGS	rd,rs1
	Swap Rounding Mode Imm	I	FSRMI	rd,imm
	Swap Flags Imm	I	FSFLAGSI	rd,imm

Optional Compressed Instructions: RVC

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

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

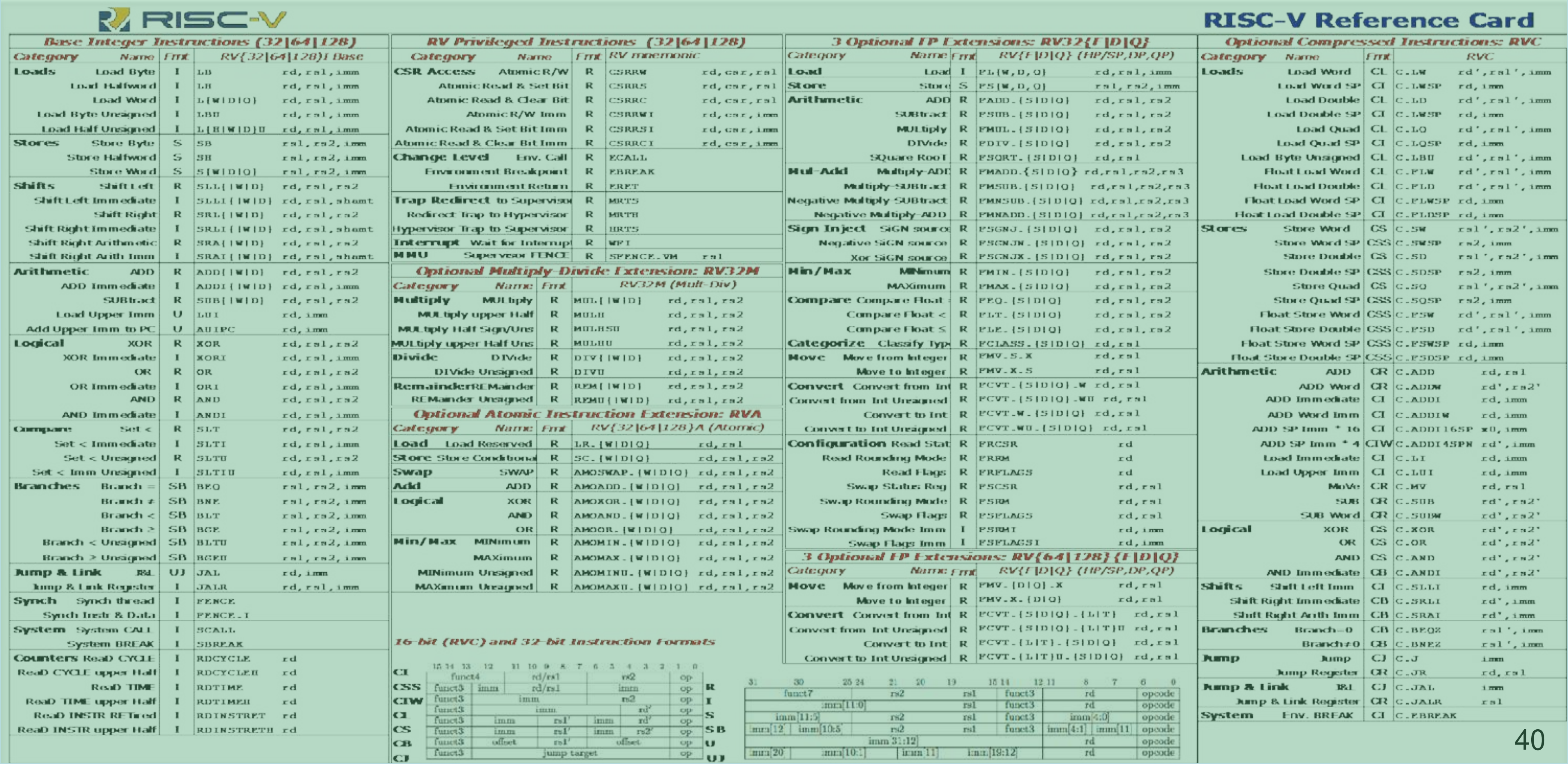
	15 14 13	12	11 10 9 8 7	6 5 4 3 2	1 0
CI	funct4		rd/rs1	rs2	op
CSS	funct3	imm	rd/rs1	imm	op
CIW	funct3	imm		rs2	op
CL	funct3	imm		rd'	op
	funct3	imm	rs1'	imm rd'	op
CS	funct3	imm	rs1'	imm rs2'	op
CB	funct3	offset	rs1'	offset	op
CJ	funct3	jump target			op

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

64A/128A

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

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



RISC-V

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 Supervisor	R	MRTS	
	Redirect Trap to Hypervisor	R	MRTH	
	Hypervisor Trap to Supervisor	R	HRTS	
Interrupt	Wait for Interrupt	R	WFI	
MMU	Supervisor FENCE	R	SFENCE.VM	rs1

Optional Multiply-Divide Extension: RV32M

Category	Name	Fmt	RV32M (Mult-Div)	
Multiply	MULtiply	R	MUL{|W|D}	rd,rs1,rs2
	MULtiply upper Half	R	MULH	rd,rs1,rs2
	MULtiply Half Sign/Uns	R	MULHSU	rd,rs1,rs2
	MULtiply upper Half Uns	R	MULHU	rd,rs1,rs2
Divide	DIVide	R	DIV{|W|D}	rd,rs1,rs2
	DIVide Unsigned	R	DIVU	rd,rs1,rs2
Remainder	REMainder	R	REM{|W|D}	rd,rs1,rs2
	REMainder Unsigned	R	REMU{|W|D}	rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

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

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

	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CI	funct4		rd/rs1		rs2		op
CSS	funct3	imm	rd/rs1		imm		op
CIW	funct3	imm			rs2		op
CI	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

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

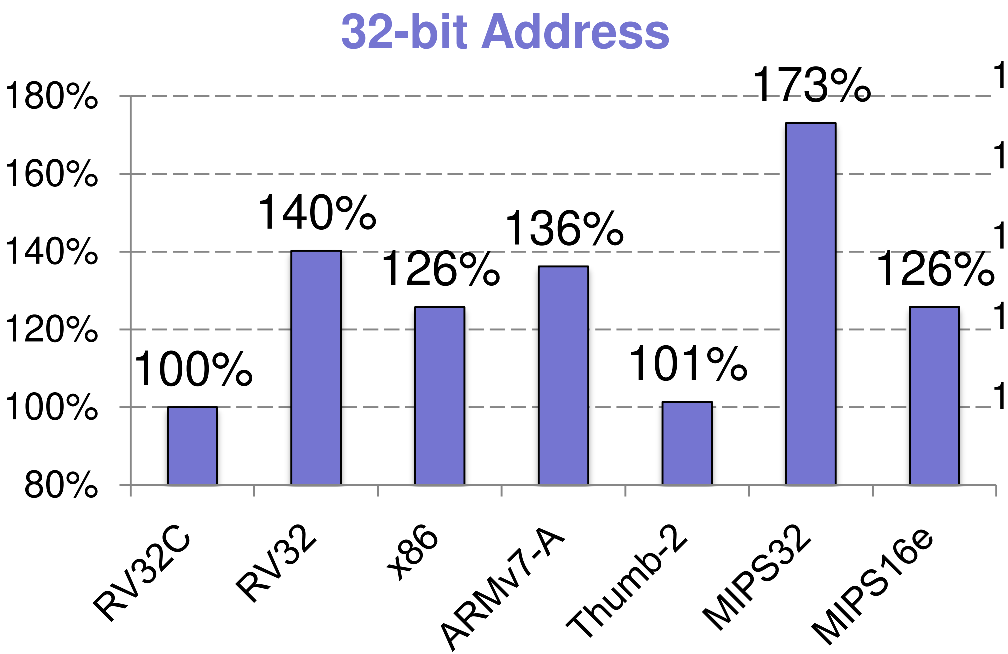
Optional Compressed Instructions: RVC

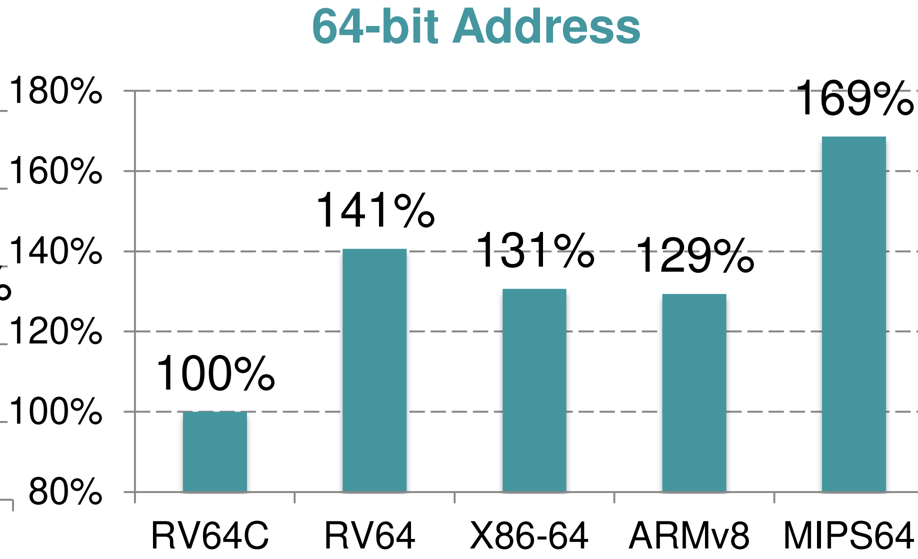
Category	Name	Fmt	RVC	
Loads	Load Word	CL	C.LW	rd′,rs1′,imm
	Load Word SP	CI	C.LWSP	rd,imm
	Load Double	CL	C.LD	rd′,rs1′,imm
	Load Double SP	CI	C.LWSP	rd,imm
	Load Quad	CL	C.LQ	rd′,rs1′,imm
	Load Quad SP	CI	C.LQSP	rd,imm
	Load Byte Unsigned	CL	C.LBU	rd′,rs1′,imm
	Float Load Word	CL	C.FLW	rd′,rs1′,imm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

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

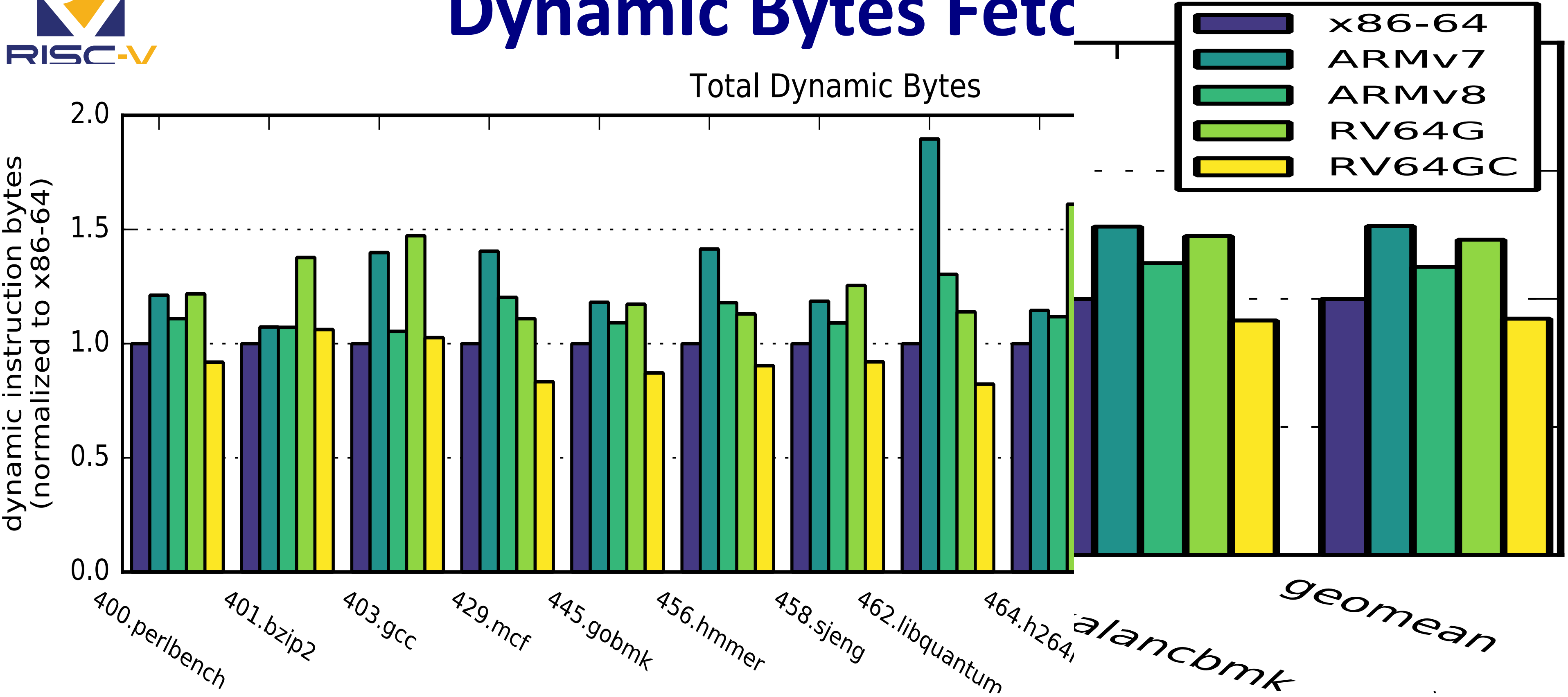
SPECint2006 compressed code size with save/restore optimization (relative to “standard” RVC)





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

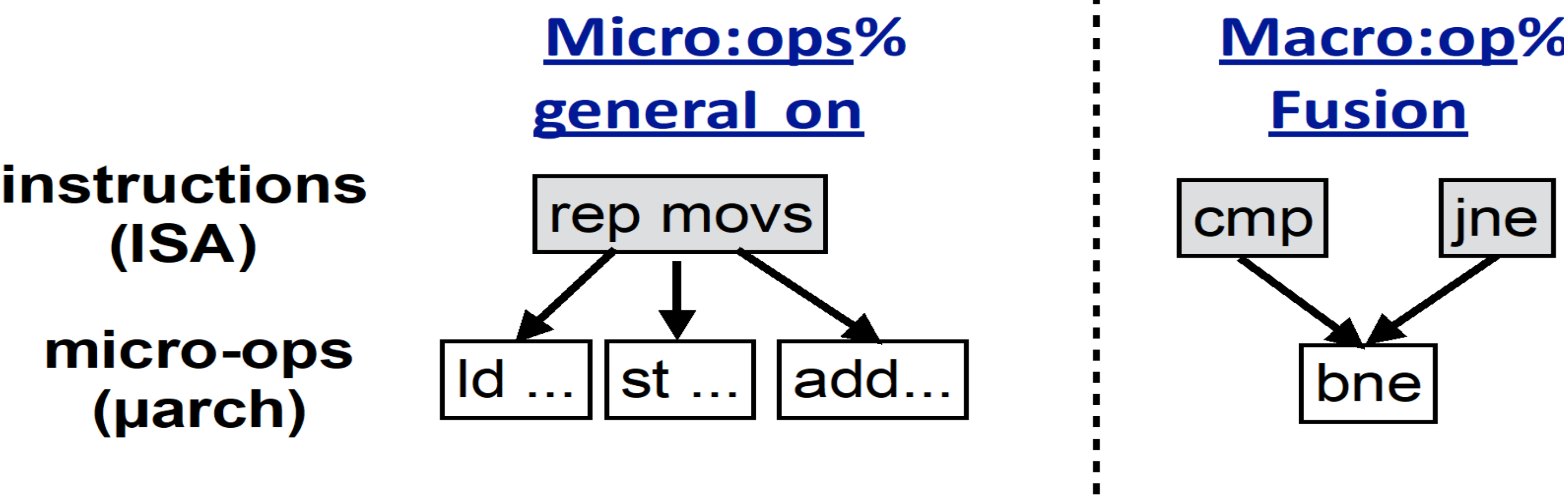
Dynamic Bytes Fetc



- 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



Micro:ops% general on

Macro:op% Fusion

instructions (ISA)

micro-ops (μarch)

rep movs

ld ...

st ...

add...

cmp

jne

bne

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]

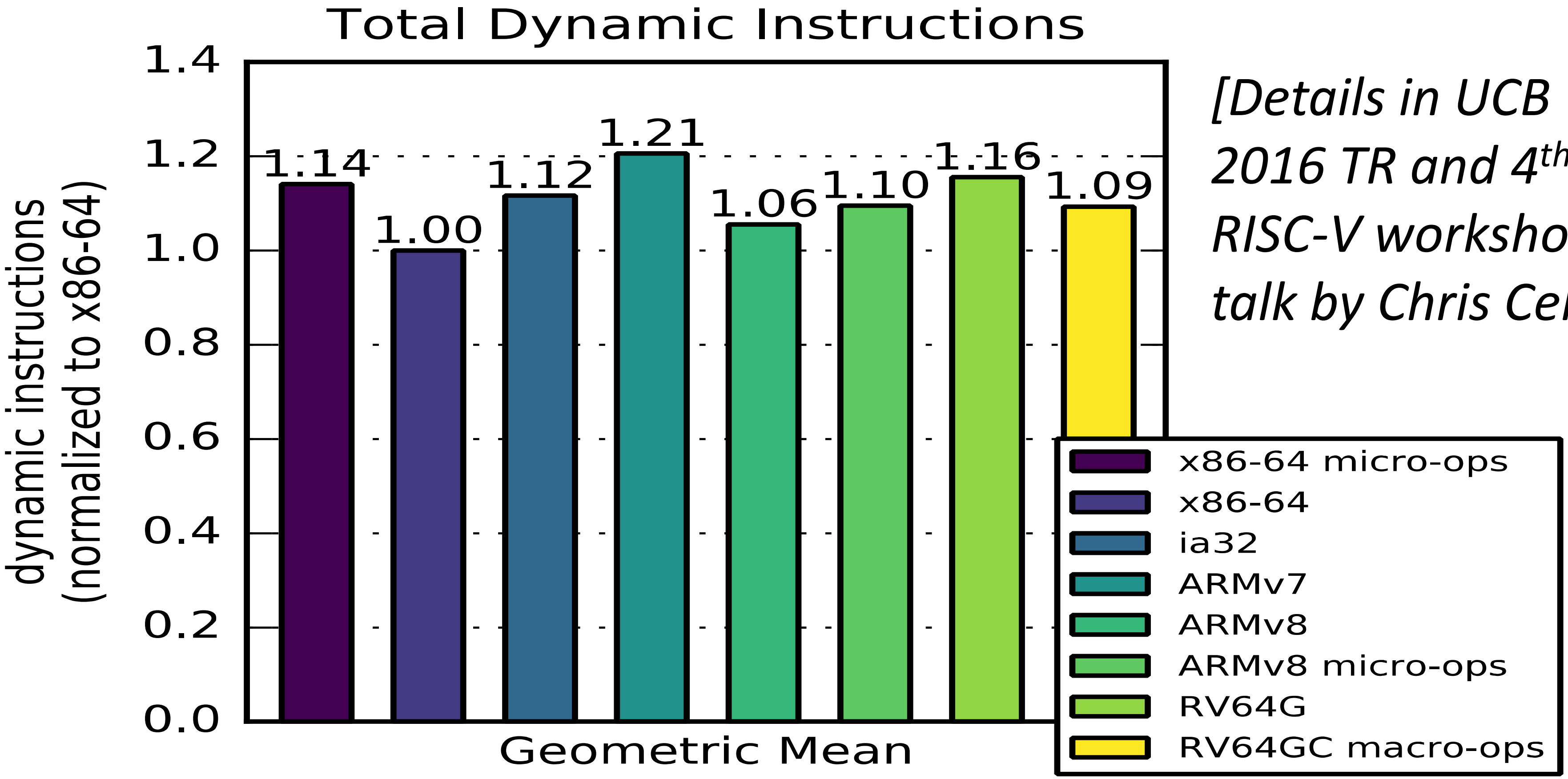
```
add rd, rs1, rs2
ld rd, 0(rd)
```

- “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: use macroop fusion, don’t grow ISA

RISC-V Competitive µarch Effort after Fusion



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

Dave Ditzel, Esperanto

RISC-V wasn’t even on the shopping list of alternatives, but the more Esperanto’s engineers looked at it, the more they realized it was more than a toy or just a teaching tool. “We assumed that RISC-V would probably lose 30% to 40% in compiler efficiency [versus Arm or MIPS or SPARC] because it’s so simple,” says Ditzel. “But our compiler guys benchmarked it, and darned if it wasn’t within 1%.”

[Article by Jim Turley, EE Journal, December 13, 2017]

Fragmentation versus Diversity

Fragmentation:

Same thing done different ways

Diversity:

Solving different problems

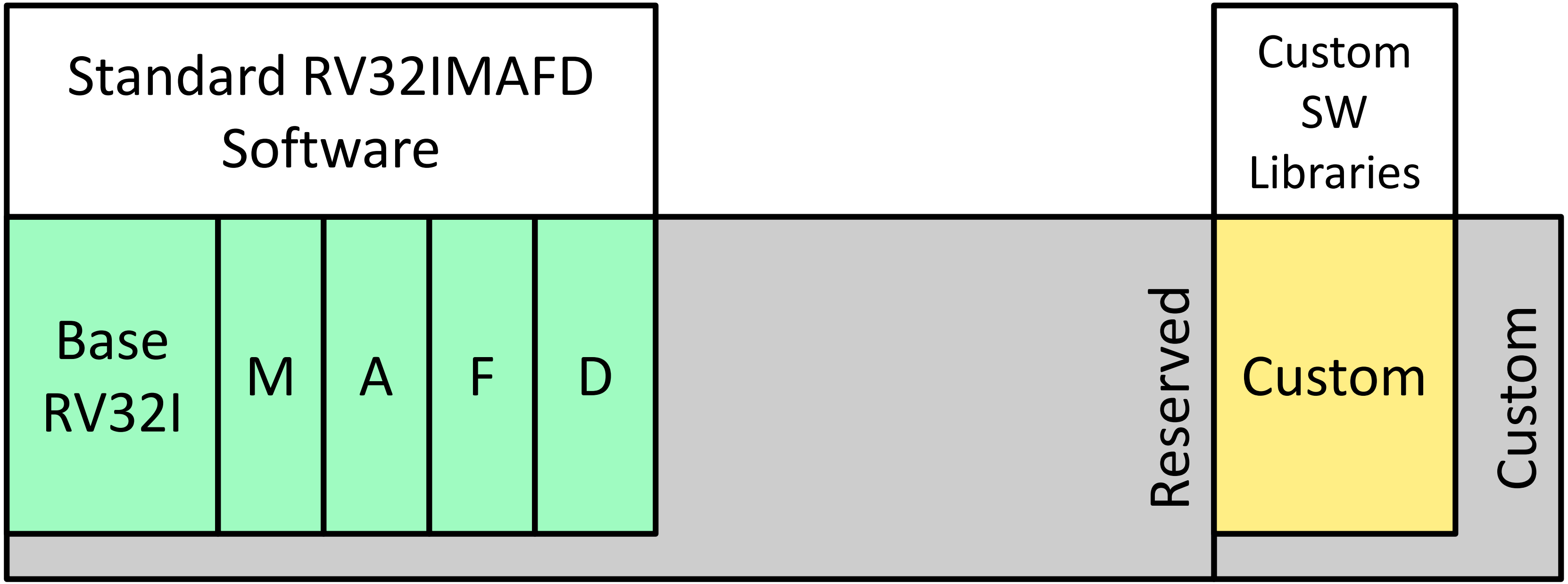
RISC-V Encoding Terminology

Standard: defined by the Foundation

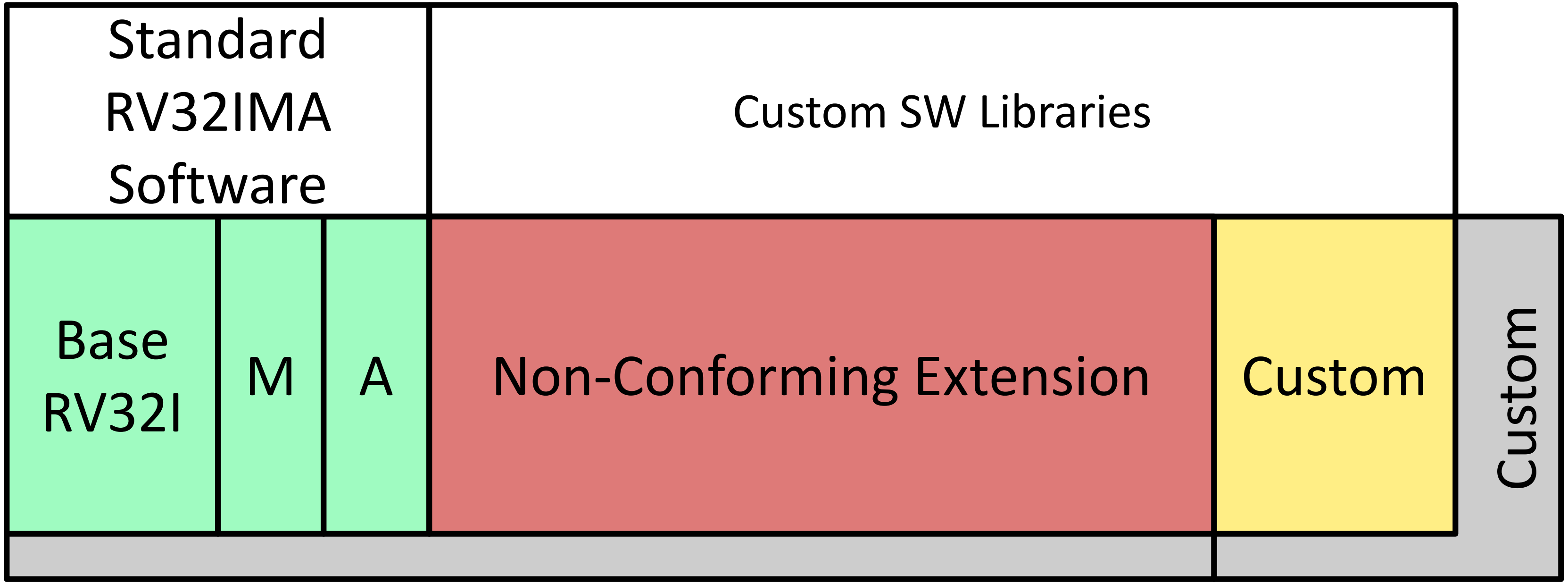
Reserved: Foundation might eventually use this space for future standard extensions

Custom: Space for implementer-specific extensions, never claimed by Foundation

RISC-V Custom Extension Example



RISC-V Custom Extension Example 2



Standard RV32IMA Software

Custom SW Libraries

Base RV32I

M

A

Non-Conforming Extension

Custom

Custom

RISC-V Privileged Architecture

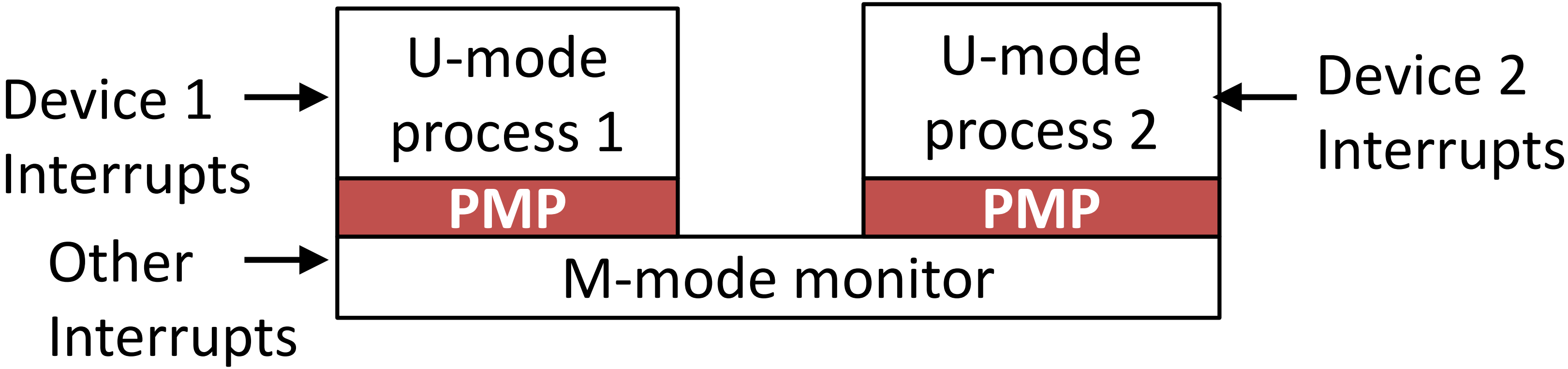
- Three privilege modes
  - User (U-mode)
  - Supervisor (S-mode)
  - Machine (M-mode)
- Supported combinations of modes:
  - M (simple embedded systems)
  - M, U (embedded systems with protection)
  - M, S, U (systems running Unix-style operating systems)
- Hypervisors run in modified S mode (HS) *(in progress)*
  - Prioritizes support for Type-2 Hypervisors like KVM
  - Can also support Type-1 Hypervisors in same model

Simple Embedded Systems (M-mode only)

- No address translation/protection
  - "Mbare" bare-metal mode
  - Trap bad physical addresses precisely
- All code inherently trusted
- Low implementation cost
  - 2^7 bits of architectural state (in addition to user ISA)
  - $+2^7$ more bits for timers
  - $+2^7$ more for basic performance counters

Secure Embedded Systems (M, U modes)

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

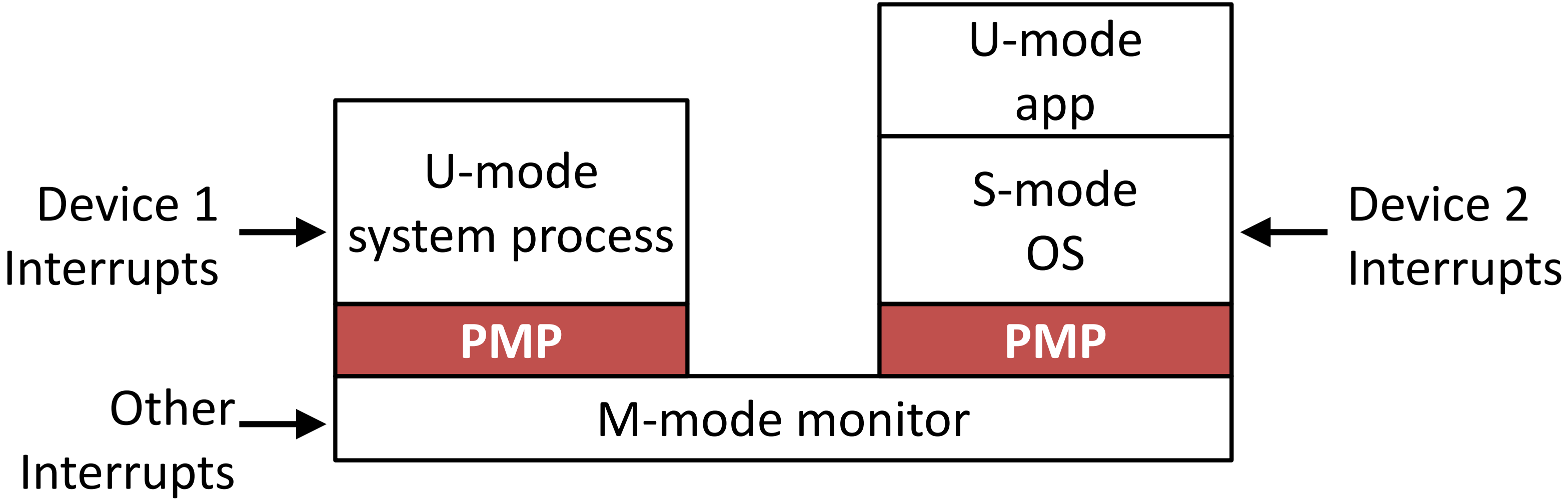


Virtual Memory Architectures (M, S, U modes)

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

S-Mode runs on top of M-mode

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



Hypervisor

- Supports Type-2 hypervisors (e.g., KVM) as well as Type-1 hypervisor (e.g., Xen)
- Current draft spec implemented in QEMU with early KVM port
- Supports recursive virtualization

RISC-V and Security

Security is one of biggest challenges in contemporary computer architecture, so which to trust?

- Simple free ISA with open implementations and publicly scrutinized security systems
- Complex proprietary ISAs with NDA-only security systems

RISC-V already the center of security architecture research

- Small set of hardware primitives support everything from embedded security to remote cloud enclaves

Foundation ISA Standards Development

- Unprivileged base and initial extensions now ***ratified***
    - RV32IMFDC, RV64IMFDC
    - ”A” extension has one minor issue to resolve (LR/SC progress)
    - User ISA stable since 2014 release
- Run/halt debug spec ***ratified***
- Memory model ***ratified***
- Privileged spec 1.11 ***ratified***
- Formal model available (SAIL)
- Vector specification 0.7.1 and tools released June 2019
    - Largest single extension to date
    - Target of advanced implementation work
- Other new ISA modules in advanced development:
    - Fast interrupts, DSP, Bit manipulation, Hypervisor, …
- ***Member-driven ISA roadmap***

RISC-V Vector Extension Overview

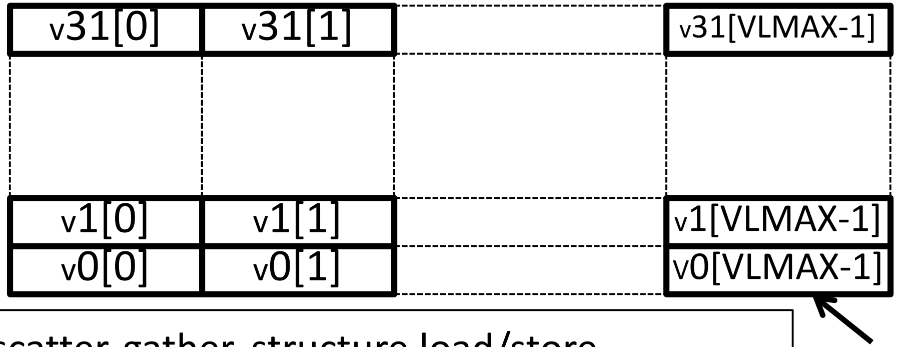
vl

Vector length CSR sets number of elements active in each instruction

vtype

Vtype sets width of element in each vector register (e.g., 32-bit, 16-bit)

32 vector registers



v31[0] v31[1] v31[VLMAX-1]

v1[0] v1[1] v1[VLMAX-1]

v0[0] v0[1] v0[VLMAX-1]

Maximum vector length (VLMAX) depends on implementation, number of vector registers used, and type of each element.

- Unit-stride, strided, scatter-gather, structure load/store instructions
- Rich set of integer, fixed-point, and floating-point instructions
- Vector-vector, vector-scalar, and vector-immediate instructions
- Multiple vector registers can be combined to form longer vectors to reduce instruction bandwidth or support mixed-precision operations (e.g., 16b*16b->32b multiply-accumulate)
- Designed for extension with custom datatypes and widths

Join and Engage!

Drive technical priorities in 20 focus areas

Compliance

Technical Deliverables

Opcode Space Mgmt Standing Committee

Software Standing Committee

Base ISA Ratification Task Group

Privileged ISA Spec Task Group

UNIX-Class Platform Spec Task Group

Formal Specification Task Group

Trusted Execution Env Spec Task Group

B Extension (Bit Manipulation) Task Group

J Extension (Dynam. Translated Lang) Task Group

P Extension (Packed-SIMD Inst) Task Group

V Extension (Vector Ops) Task Group

Cryptographic Extension Task Group

Debug Specification Task Group

Fast Interrupts Spec Task Group

Memory Model Spec Task Group

Processor Trace Spec Task Group

Compliance Task Group

+ Security Committee and
new Safety Task Group