RISC-V

Instruction Sets Want to be Free!

SiFive Inc & RISC-V Foundation

Microsemi
March 6, 2017

Why Instruction Set Architecture matters

- **Why can't Intel sell mobile chips?**
  - 99%+ of mobile phones/tablets based on ARM v7/v8 ISA

- **Why can't ARM partners sell servers?**
  - 99%+ of laptops/desktops/servers based on AMD64 ISA (over 95%+ built by Intel)

- **How can IBM still sell mainframes?**
  - IBM 360, oldest surviving ISA (50+ years)

ISA is most important interface in computer system

where software meets hardware

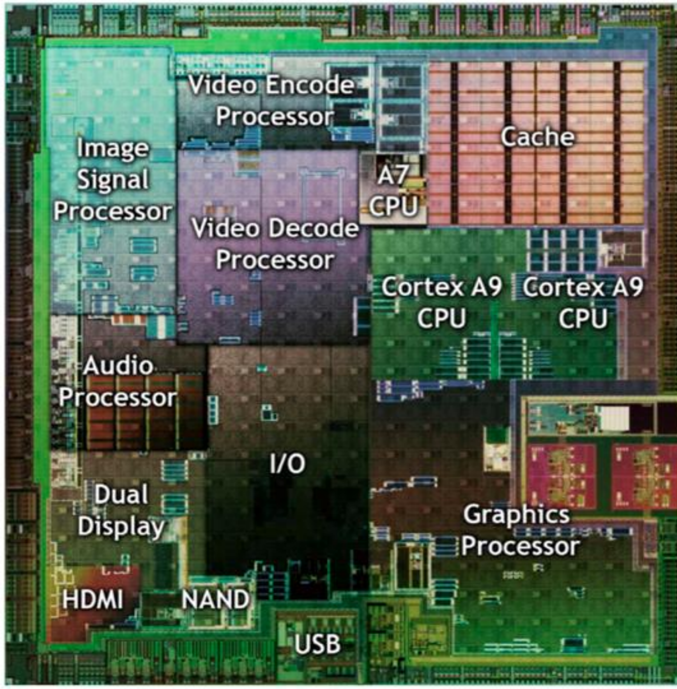
Open Software/Standards Work!

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

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

Why so many ISAs on an SoC?

- Applications processor (usually ARM)
- Graphics processors
- Image processors
- Radio DSPs
- Audio DSPs
- Security processors
- Power-management processor
- ….



NVIDIA Tegra SoC

- Apps processor ISA too large for base accelerator ISA
- IP bought from different places, each proprietary ISA
- Home-grown ISA cores
- *Over a dozen ISAs on some SoCs – each with unique software stack*

Do we need all these different ISAs?

Must they be proprietary?

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

RISC-V Origins

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

- Initial candidate was MIPS: Lots of SW
  - But, a few bad instructions
    - BRANCH DELAY SLOT
    - LOAD DELAY SLOT
  - Shows the guts of the ISA

- Obvious choices: x86 and ARM

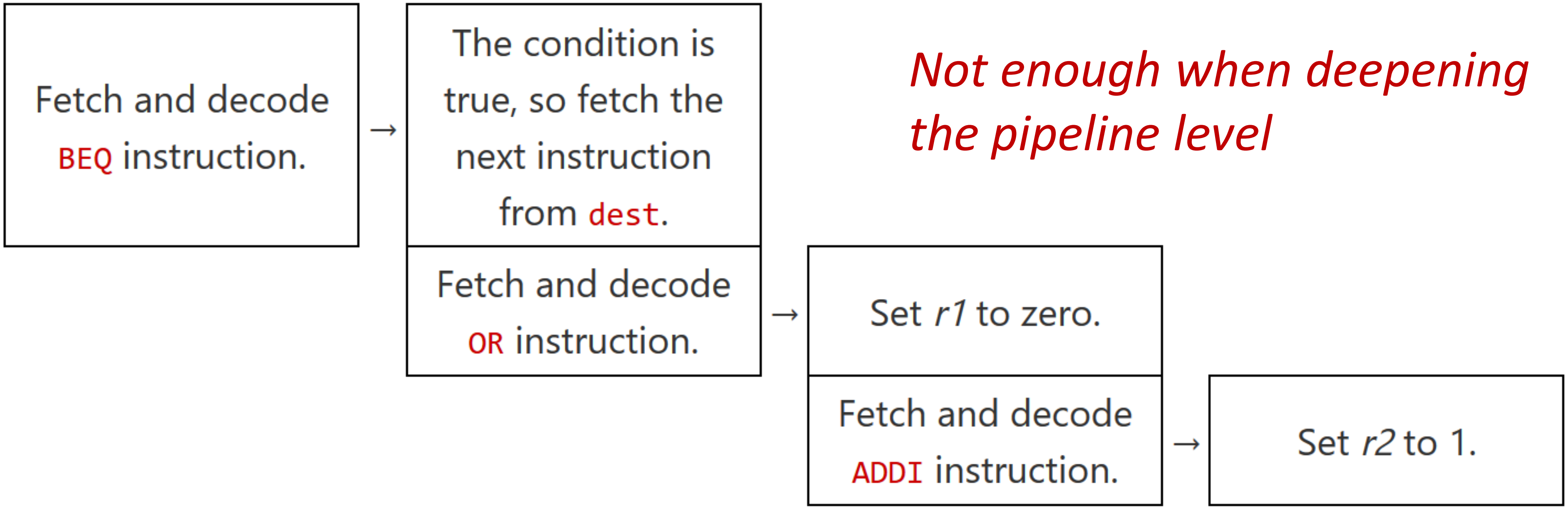
branch delay slot

- A basic example of a branch delay slot:

  (source: Microsoft)

```
    BEQ     r1, zero, dest        ; branch if r1 is equal to zero
    OR      r1, zero, zero        ; set r1 = 0; this line executes regardless
    ...
dest:
 ADDI    r2, zero, 1           ; set r2 = 1
```

- The OR instruction sits in the branch delay slot, and executes whether the branch is taken or not



Fetch and decode BEQ instruction. → The condition is true, so fetch the next instruction from dest.

Fetch and decode OR instruction. → Set *r1* to zero.

Fetch and decode ADDI instruction. → Set *r2* to 1.

Not enough when deepening the pipeline level

Intel x86 “AAA” Instruction

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

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

- Single byte instruction

ARM v7 LDMIAEQ Instruction

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

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

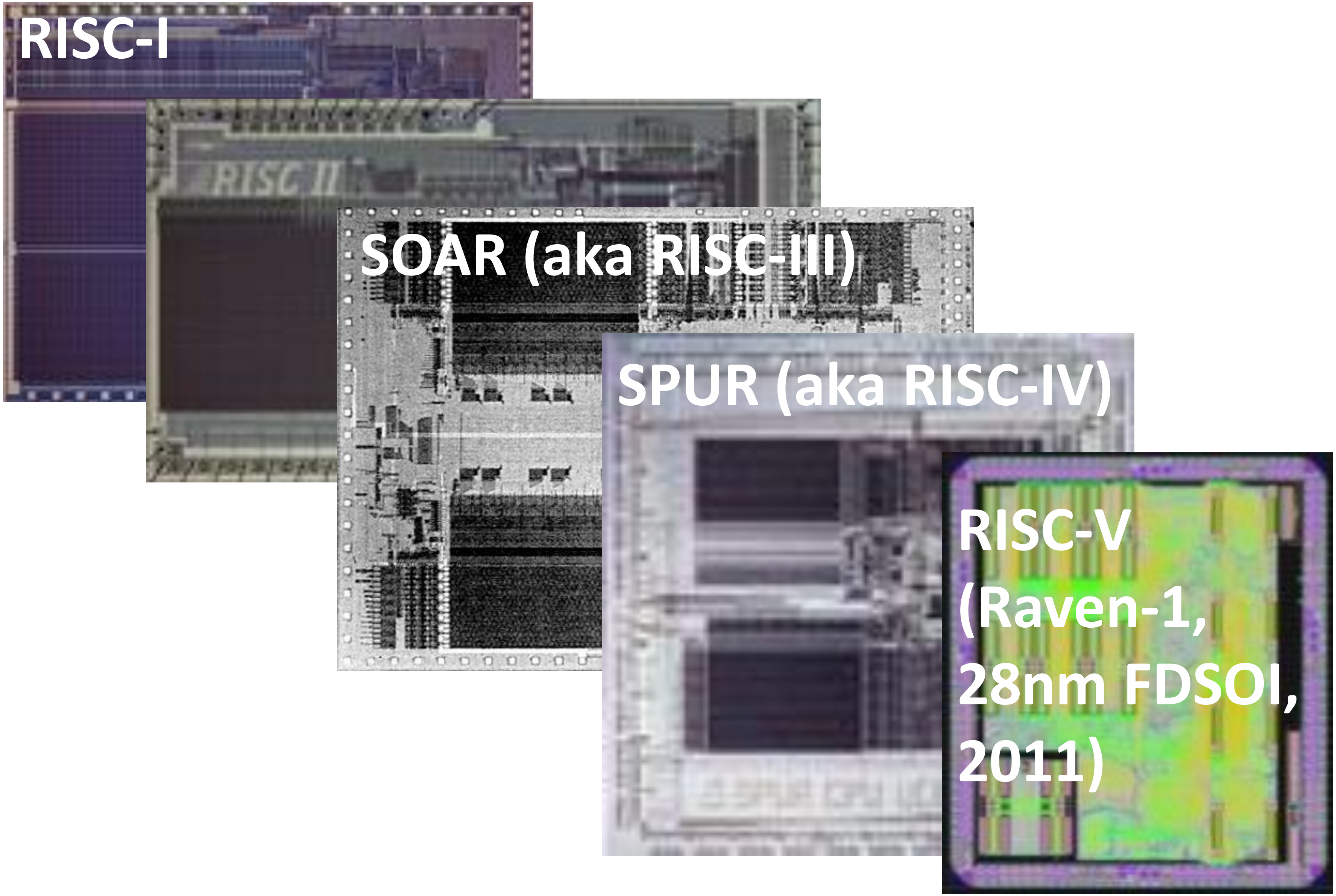
- Idiom for "stack pop and return from a function call"

RISC-V Origin Story

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

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

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



Universal ISA Requirements

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

Why Didn’t Other Open ISAs Take Off?

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

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

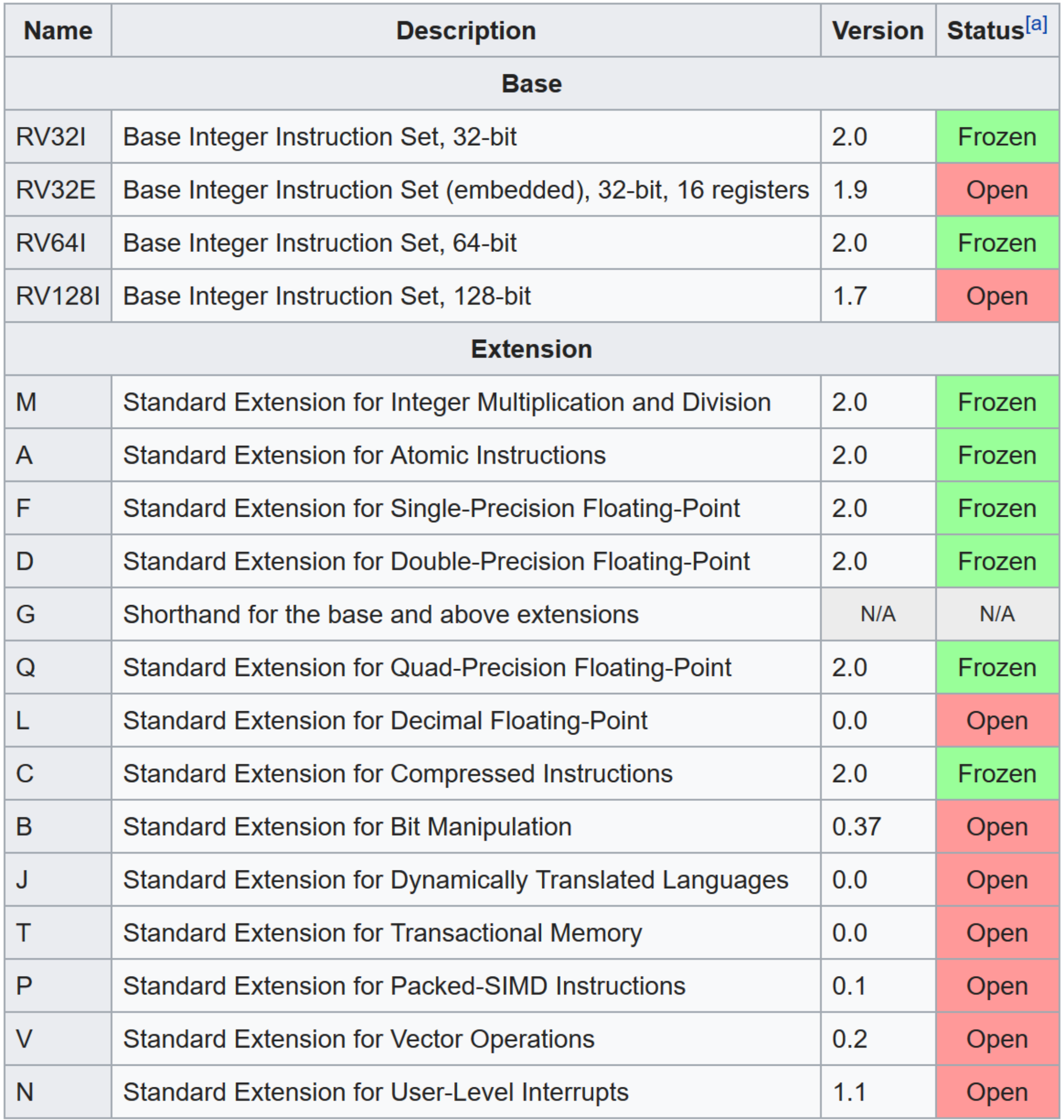
What's Different about RISC-V?

- *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

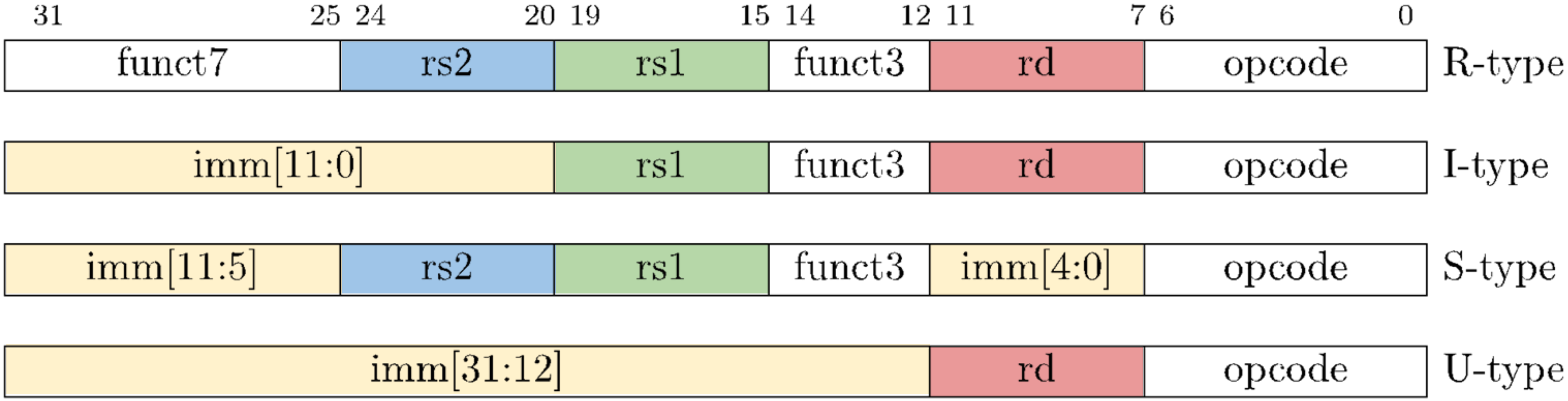
- Four base integer ISAs
  - RV32E, RV32I, RV64I, RV128I
  - RV32E is 16-register subset of RV32I
  - Only <50 hardware instructions needed for base
- Standard extensions
  - M: Integer multiply/divide
  - A: Atomic memory operations (AMOs + LR/SC)
  - F: Single-precision floating-point
  - D: Double-precision 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

Status as of “today”



Name	Description	Version	Status[a]
Base			
RV32I	Base Integer Instruction Set, 32-bit	2.0	Frozen
RV32E	Base Integer Instruction Set (embedded), 32-bit, 16 registers	1.9	Open
RV64I	Base Integer Instruction Set, 64-bit	2.0	Frozen
RV128I	Base Integer Instruction Set, 128-bit	1.7	Open
Extension			
M	Standard Extension for Integer Multiplication and Division	2.0	Frozen
A	Standard Extension for Atomic Instructions	2.0	Frozen
F	Standard Extension for Single-Precision Floating-Point	2.0	Frozen
D	Standard Extension for Double-Precision Floating-Point	2.0	Frozen
G	Shorthand for the base and above extensions	N/A	N/A
Q	Standard Extension for Quad-Precision Floating-Point	2.0	Frozen
L	Standard Extension for Decimal Floating-Point	0.0	Open
C	Standard Extension for Compressed Instructions	2.0	Frozen
B	Standard Extension for Bit Manipulation	0.37	Open
J	Standard Extension for Dynamically Translated Languages	0.0	Open
T	Standard Extension for Transactional Memory	0.0	Open
P	Standard Extension for Packed-SIMD Instructions	0.1	Open
V	Standard Extension for Vector Operations	0.2	Open
N	Standard Extension for User-Level Interrupts	1.1	Open

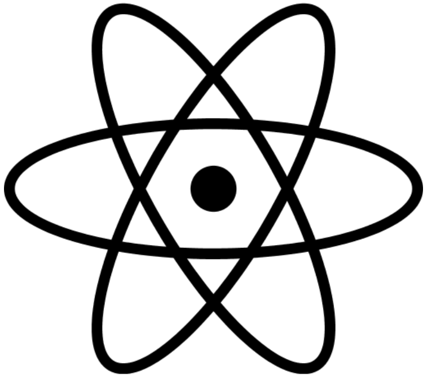
RISC-V Standard Base ISA Details



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

"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

- *Current issues in memory model being resolved*

Variable-Length Encoding

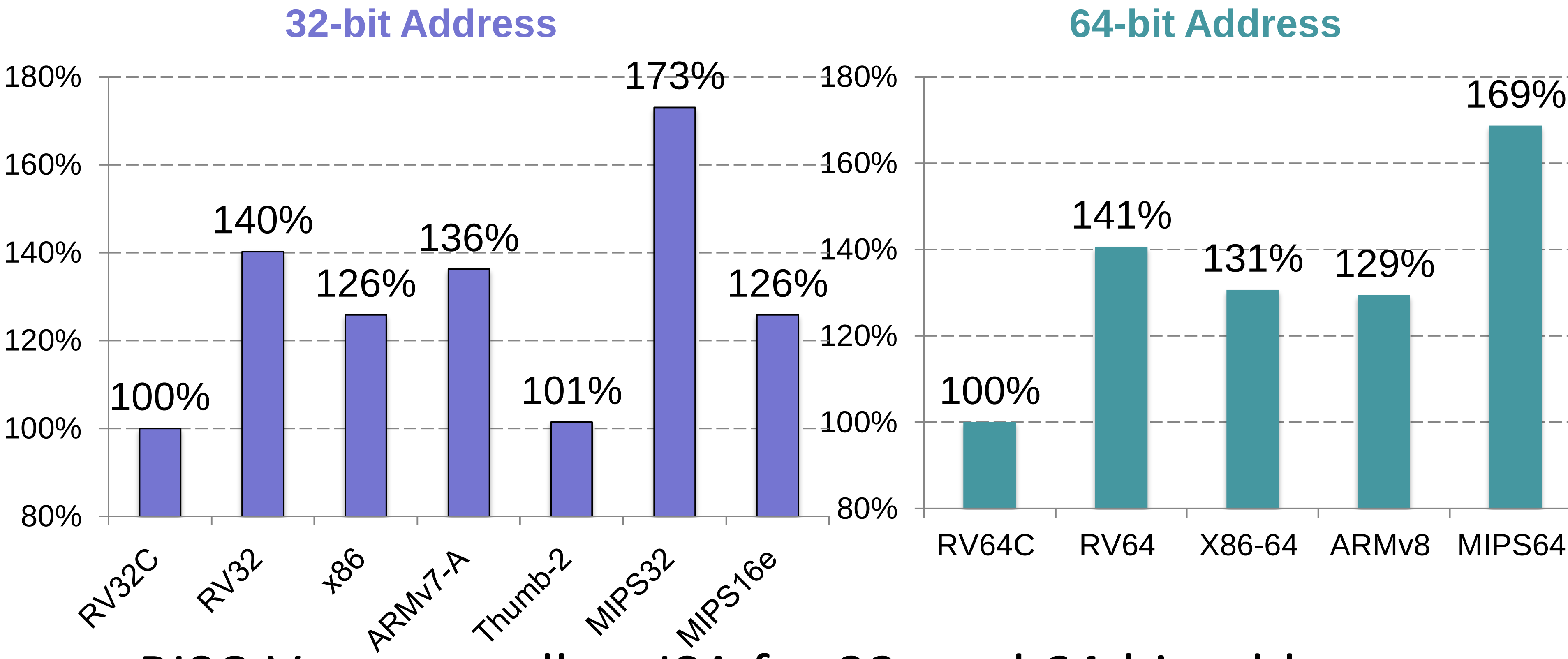
		xxxxxxxxxxxxxxaa	16-bit (aa $\neq$ 11)
	xxxxxxxxxxxxxxxx	xxxxxxxxxxxbbb11	32-bit (bbb $\neq$ 111)
$\cdots$xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxxx011111	48-bit
$\cdots$xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxx0111111	64-bit
$\cdots$xxxx	xxxxxxxxxxxxxxxx	xnnnxxxxx1111111	(80+16*nnn)-bit, nnn$\neq$111
$\cdots$xxxx	xxxxxxxxxxxxxxxx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

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

“C”: Compressed Instruction Extension

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

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



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

RISC-V Privileged Architecture

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

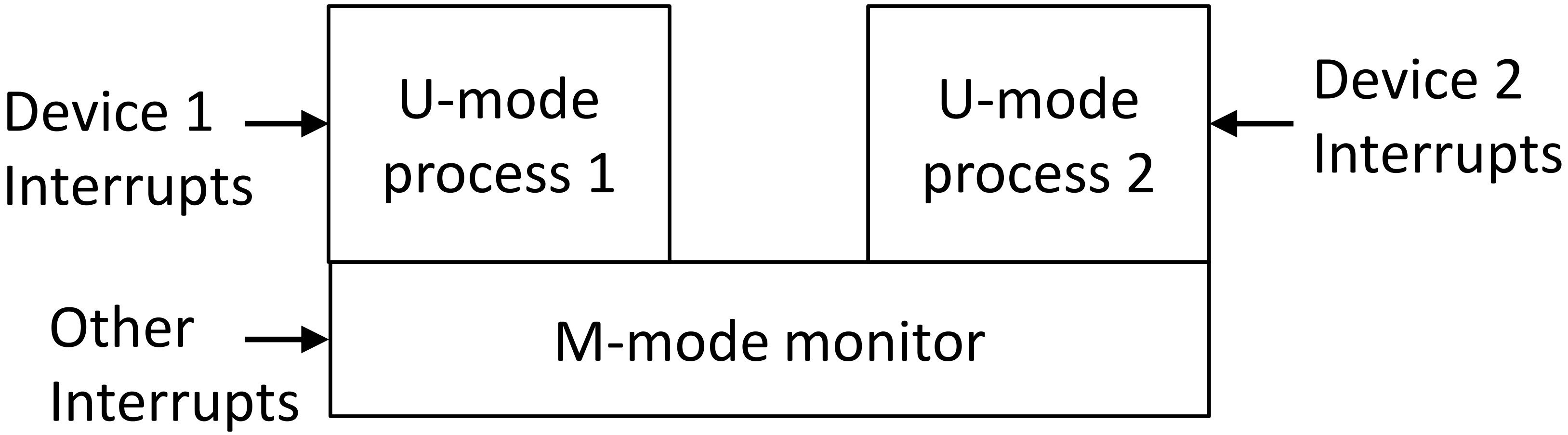
- M	(simple embedded systems)
- M, U	(embedded systems with protection)
- M, S, U	(systems running Unix-style operating systems)
- M, H, S, U	(systems running Hypervisors)

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 on U-mode accesses
- Interrupt handling can be delegated to U-mode code
  - User-level interrupt support
- Provides arbitrary number of isolated subsystems

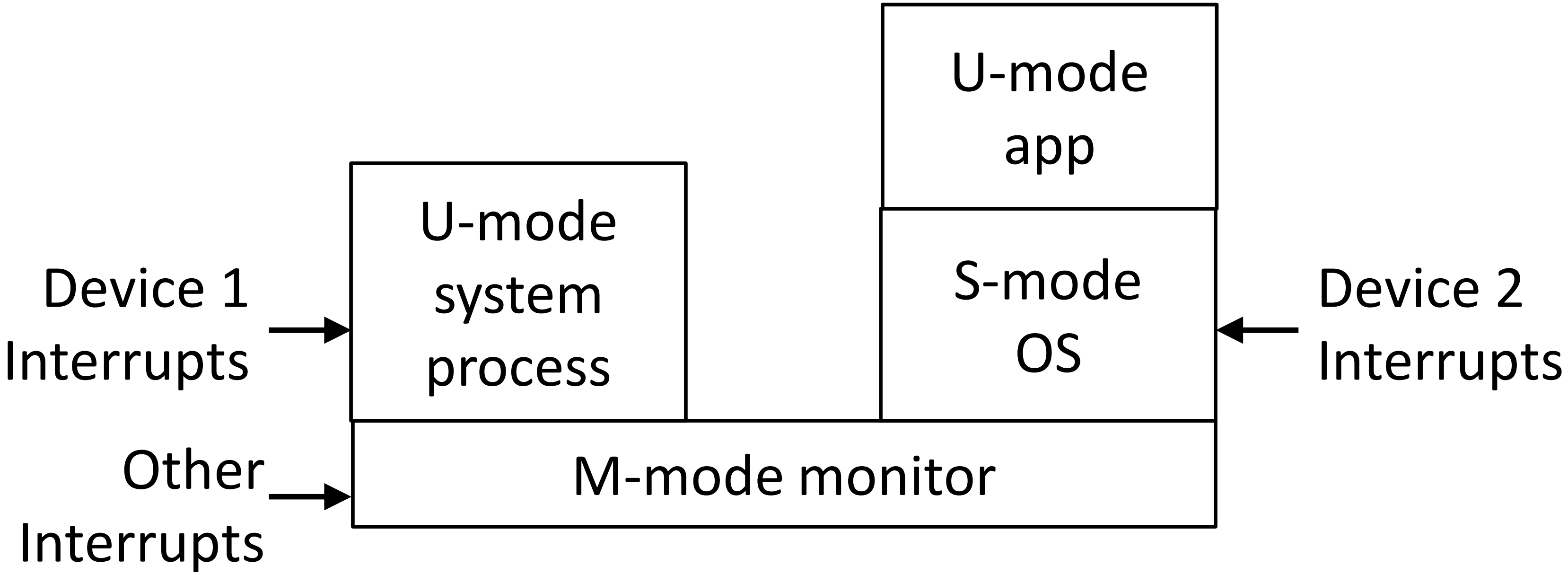


Virtual Memory Architectures (M, S, U modes)

- 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

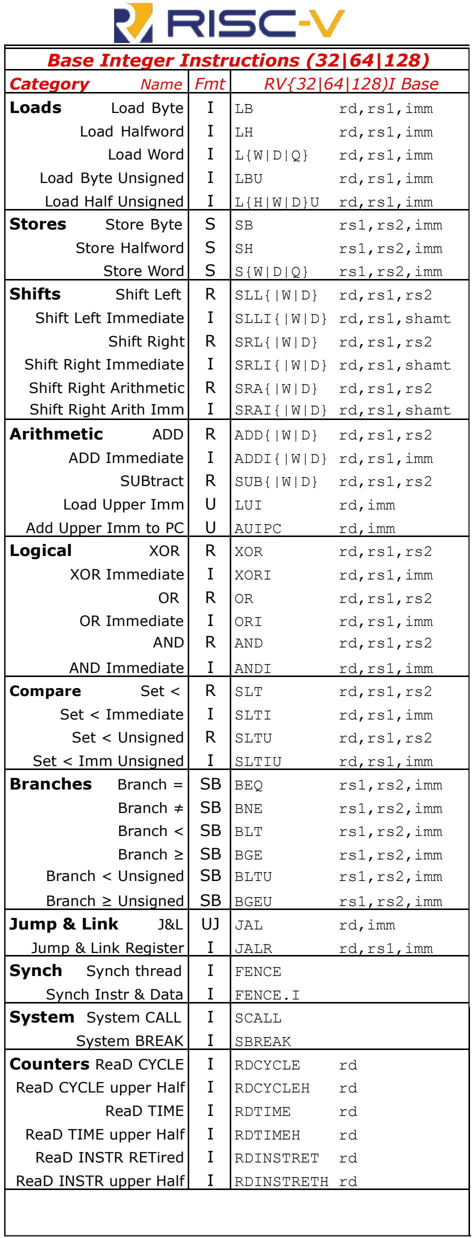
S-Mode runs on top of M-mode

- M-mode runs secure boot and monitor
- S-mode runs OS (OS always runs virtualized)
- U-mode runs application on top of OS or M-mode
- Monitor can provide physical resource partitioning, simple hypervisor (space for separate H-mode)



RV32I

RISC-V Reference Card



RISC-V

Base Integer Instructions (32|64|128)

Category	*Name*	*Fmt*	*RV{32|64|128}I Base*	
Loads	Load Byte	I	`LB`	`rd,rs1,imm`
	Load Halfword	I	`LH`	`rd,rs1,imm`
	Load Word	I	`L{W|D|Q}`	`rd,rs1,imm`
	Load Byte Unsigned	I	`LBU`	`rd,rs1,imm`
	Load Half Unsigned	I	`L{H|W|D}U`	`rd,rs1,imm`
Stores	Store Byte	S	`SB`	`rs1,rs2,imm`
	Store Halfword	S	`SH`	`rs1,rs2,imm`
	Store Word	S	`S{W|D|Q}`	`rs1,rs2,imm`
Shifts	Shift Left	R	`SLL{|W|D}`	`rd,rs1,rs2`
	Shift Left Immediate	I	`SLLI{|W|D}`	`rd,rs1,shamt`
	Shift Right	R	`SRL{|W|D}`	`rd,rs1,rs2`
	Shift Right Immediate	I	`SRLI{|W|D}`	`rd,rs1,shamt`
	Shift Right Arithmetic	R	`SRA{|W|D}`	`rd,rs1,rs2`
	Shift Right Arith Imm	I	`SRAI{|W|D}`	`rd,rs1,shamt`
Arithmetic	ADD	R	`ADD{|W|D}`	`rd,rs1,rs2`
	ADD Immediate	I	`ADDI{|W|D}`	`rd,rs1,imm`
	SUBtract	R	`SUB{|W|D}`	`rd,rs1,rs2`
	Load Upper Imm	U	`LUI`	`rd,imm`
	Add Upper Imm to PC	U	`AUIPC`	`rd,imm`
Logical	XOR	R	`XOR`	`rd,rs1,rs2`
	XOR Immediate	I	`XORI`	`rd,rs1,imm`
	OR	R	`OR`	`rd,rs1,rs2`
	OR Immediate	I	`ORI`	`rd,rs1,imm`
	AND	R	`AND`	`rd,rs1,rs2`
	AND Immediate	I	`ANDI`	`rd,rs1,imm`
Compare	Set <	R	`SLT`	`rd,rs1,rs2`
	Set < Immediate	I	`SLTI`	`rd,rs1,imm`
	Set < Unsigned	R	`SLTU`	`rd,rs1,rs2`
	Set < Imm Unsigned	I	`SLTIU`	`rd,rs1,imm`
Branches	Branch =	SB	`BEQ`	`rs1,rs2,imm`
	Branch ≠	SB	`BNE`	`rs1,rs2,imm`
	Branch <	SB	`BLT`	`rs1,rs2,imm`
	Branch ≥	SB	`BGE`	`rs1,rs2,imm`
	Branch < Unsigned	SB	`BLTU`	`rs1,rs2,imm`
	Branch ≥ Unsigned	SB	`BGEU`	`rs1,rs2,imm`
Jump & Link	J&L	UJ	`JAL`	`rd,imm`
	Jump & Link Register	I	`JALR`	`rd,rs1,imm`
Synch	Synch thread	I	`FENCE`	
	Synch Instr & Data	I	`FENCE.I`	
System	System CALL	I	`SCALL`	
	System BREAK	I	`SBREAK`	
Counters	ReaD CYCLE	I	`RDCYCLE`	`rd`
	ReaD CYCLE upper Half	I	`RDCYCLEH`	`rd`
	ReaD TIME	I	`RDTIME`	`rd`
	ReaD TIME upper Half	I	`RDTIMEH`	`rd`
	ReaD INSTR RETired	I	`RDINSTRET`	`rd`
	ReaD INSTR upper Half	I	`RDINSTRETH`	`rd`

+14 Privileged

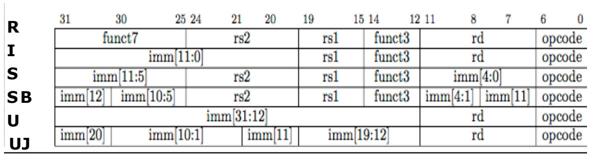
+ 8 for M

+ 34 for F, D, Q

+ 46 for C

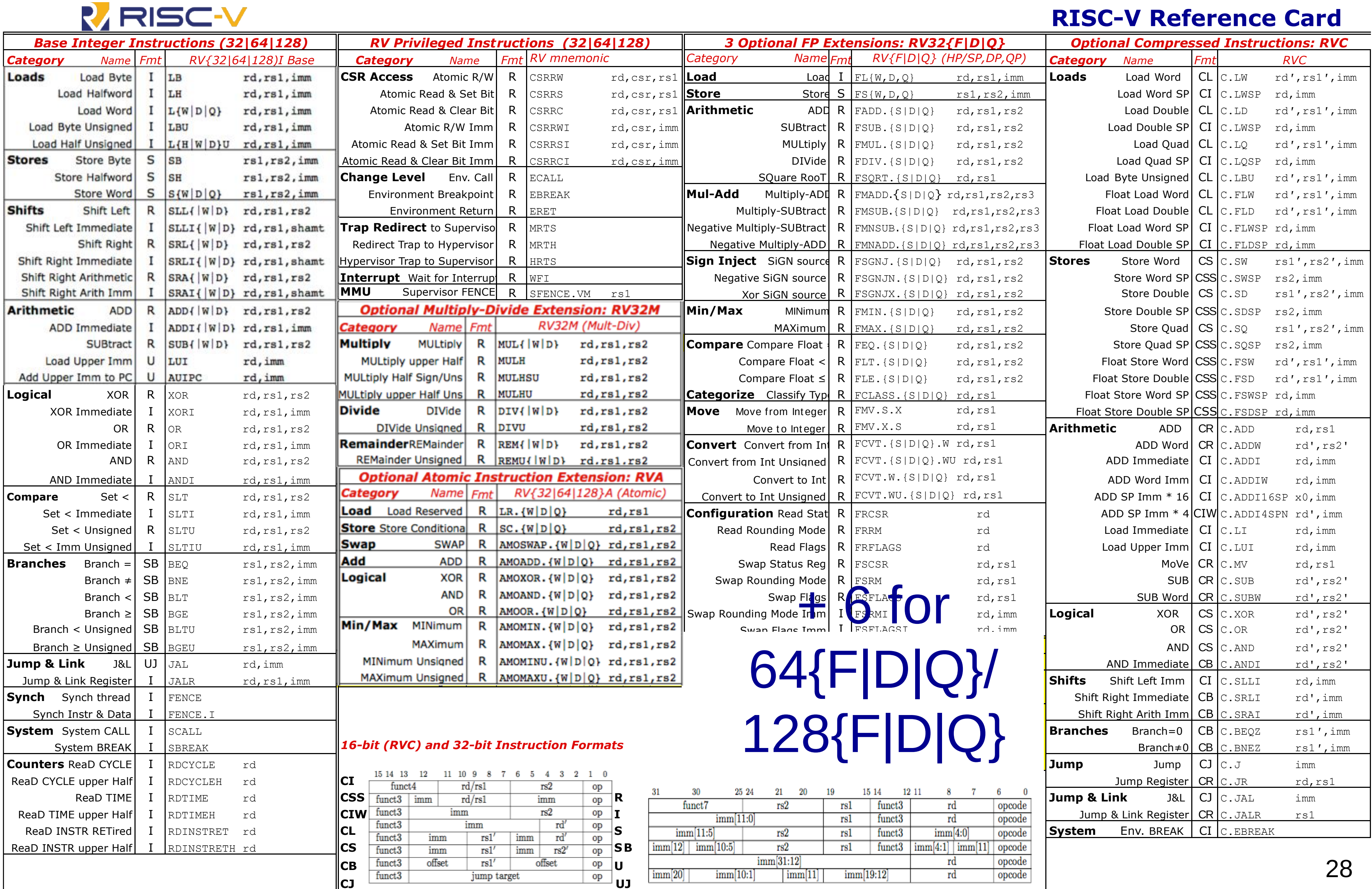
+ 11 for A

32-bit Instruction Formats



	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 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 Interrup	R	WFI	
MMU	Supervisor FENCE	R	SFENCE.VM	rs1

Optional Multiply-Divide Extension: RV32M

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

Optional Atomic Instruction Extension: RVA

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

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

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

+6 for

64{F|D|Q}/

128{F|D|Q}

Optional Compressed Instructions: RVC

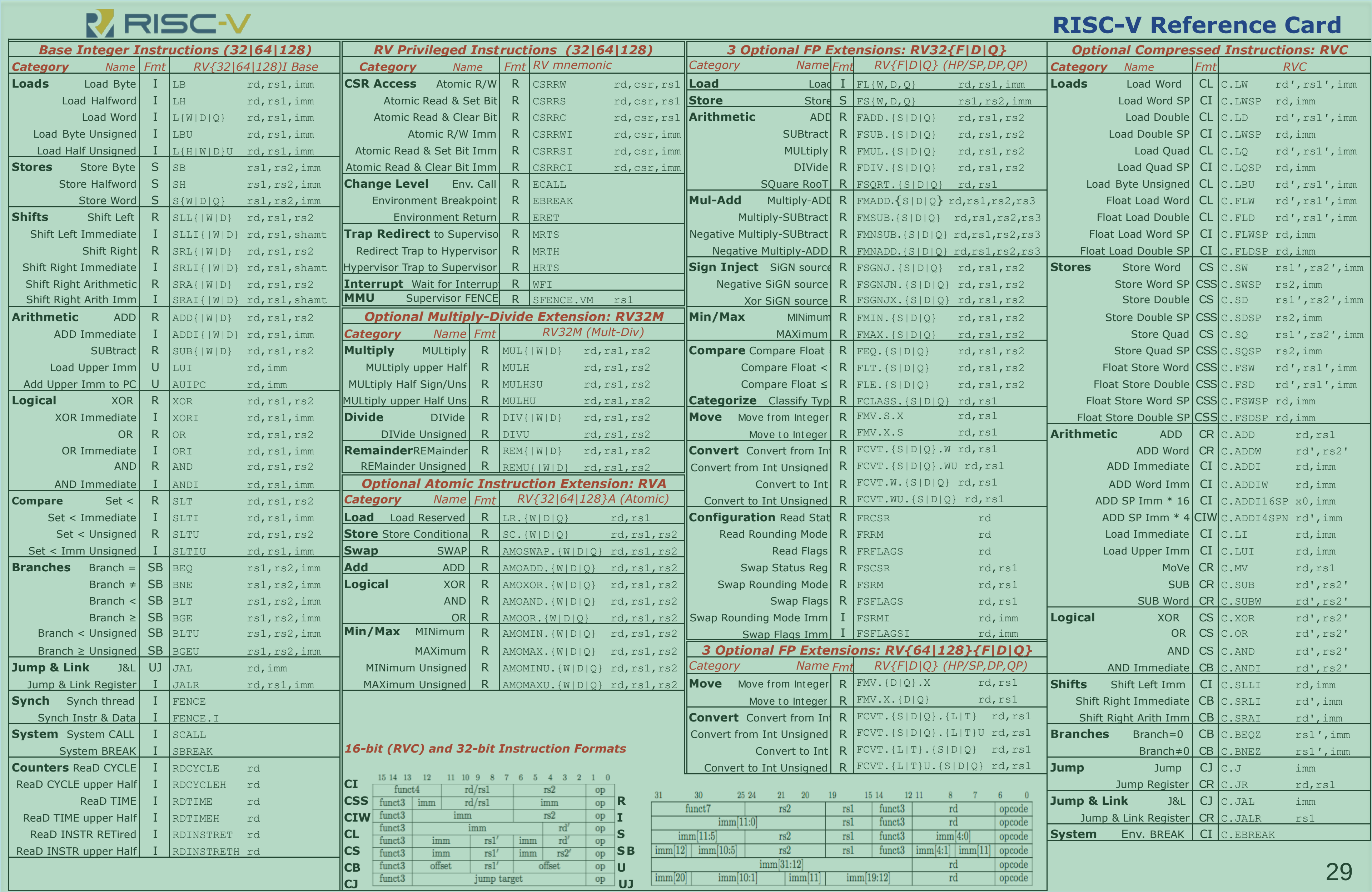
Category	Name	Fmt	RVC	
Loads	Load Word	CL	C.LW	rd',rs1',imm
	Load Word SP	CI	C.LWSP	rd,imm
	Load Double	CL	C.LD	rd',rs1',imm
	Load Double SP	CI	C.LWSP	rd,imm
	Load Quad	CL	C.LQ	rd',rs1',imm
	Load Quad SP	CI	C.LQSP	rd,imm
	Load Byte Unsigned	CL	C.LBU	rd',rs1',imm
	Float Load Word	CL	C.FLW	rd',rs1',imm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

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

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

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

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



RISC-V

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 Interrup	R	WFI	
MMU	Supervisor FENCE	R	SFENCE.VM	rs1

Optional Multiply-Divide Extension: RV32M

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

Optional Atomic Instruction Extension: RVA

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

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

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

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

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

Optional Compressed Instructions: RVC

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

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

	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CI	funct4		rd/rs1		rs2		op
CSS	funct3	imm	rd/rs1		imm		op
CIW	funct3	imm			rs2		op
CL	funct3	imm				rd′	op
	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 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. Additional members welcome:
  - Platinum ($25K/year)
  - Gold ($10K/year)
  - Silver ($5K/year)

Foundation Principles

- The RISC-V ISA and related standards shall remain open and license-free to all parties. The standard specifications shall always be publicly available as an online download.
- The compatibility test suites shall always be publicly available as a source-code download.
- To protect the standard, only members of the Foundation in good standing can use “RISC-V” and associated trademarks for commercial products, and only for devices that pass the tests in the open-source compatibility suites maintained by the Foundation.

RISC-V

Foundation Members (48+)

Platinum:

Berkeley Architecture Research

bluespec

cortus

DRAPER

Google

Hewlett Packard Enterprise

Microsemi

IBM®

Mellanox® TECHNOLOGIES

QUALCOMM®

Microsoft

ORACLE

NVIDIA®

Rambus Cryptography Research

SiFive

WD Western Digital®

Gold, Silver, Auditors:

AMD

ANDES TECHNOLOGY

antmicro EMBEDDED SYSTEMS

Blockstream

ICT 中科院计算所 INSTITUTE OF COMPUTING TECHNOLOGY,CAS

codasip

Esperanto Technologies

ESPRESSIF

GRAY RESEARCH

IDT®

INTRINSIX

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

ETH zürich

LATTICE SEMICONDUCTOR

lowRISC

PROCESSOR

MIT CSAIL

ROA LOGIC

Rumble Development

runtime.io

Sur Technology

Syntacore™ Custom cores and tools

Technolution

VectorBlox embedded supercomputing

Who is SiFive?

Best-in-class team with technology depth and breadth

Founders & Execs

Yunsup Lee
CTO

Krste Asanovic
Chief Architect

Andrew Waterman
Chief Engineer

Jack Kang
VP Product/BD

Stefan Dyckerhoff
Interim CEO, VC

Sander Arts
CMO

Key Leaders & Team

Han Chen
Chip Implementation

Ali Habibi
Verification

ARM intel NVIDIA XILINX MARVELL
QUALCOMM Berkeley UNIVERSITY OF CALIFORNIA Rambus SYNOPSYS
cadence Stanford University RISC-V

SiFive Product Offerings: CPU IP & SoCs

Common hardware platform, Standard software, Customized features

CPU Core IP

Coreplex E Series

ARM Cortex-M Series Replacement
- 32-bit CPU
- Replaces M0, M0+, M3, M4, M23, M33
- 2x Perf/Watt
- Custom Extensions

Coreplex U Series

ARM Cortex-A Series Replacement
- 32-bit and 64-bit CPUs
- Replaces A5, A7, A32, A35, A8, A9, A53,
- 2x Perf/Watt
- Custom Extensions

SiFive Freedom SoCs

Freedom Everywhere

Low cost, 32-bit microcontrollers
- Edge Computing
- Embedded
- Smart IOT
- Wearables

Freedom Unleashed

High performance, 64-bit multi-core SoCs
- Datacenter Accelerators
- Storage / SSD Controllers
- Networking / Baseband

Learning More about RISC-V

- Website **`riscv.org`** is primary resource
- Sign up for mailing lists/twitter at **`riscv.org`** to get announcements
- Ask Ted!
- 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
- All workshops sold out!
- Material from all workshops at **`riscv.org`**

5th RISC-V Workshop November 2016

- Sold out! People turned away…
- 100+ companies, 360+ attendees
- Hosted at Google, Mountain View
- SiFive announced first commercial RISC-V SoC
- Next Workshop, Shanghai, May 9-10, 2017, hosted by NVIDIA

Summary: Why RISC-V?

- Free and open architecture, no proprietary lock-in
- Much simpler ISA than others (verification, security)
- Stable, will not change or disappear
- Enables better area/power/performance than other general-purpose ISAs at all design points
- Usable as base ISA for every core on SoC
- Readily and freely extensible

Modest RISC-V Project Goal

Become the industry-standard ISA for all computing devices

RISC-V Research Project Sponsors

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

Questions?

Backup

RISC-V Architecture Roadmap

- **Nov 2016 - 5th workshop (@1.9.1 today)**
  - Agree/tweak plan, assign more leaders and doers
- **Feb 2017**
  - Priv-1.10.0
  - Debug spec ratified by Foundation
  - Calling convention fixed and documented
  - ELF format fixed and documented
  - M-mode/S-mode changes must be backwards-compatible after this date
- **March 2017**
  - Memory model changes must be backwards-compatible after this date
- **May 2017 - 6th workshop**
  - Priv-1.11.0
  - RV32EMAC RV32IMAFDQC RV64IMAFDQC ratified by Foundation
- **Aug 2017**
  - Priv-1.12.0
- **Nov 2017 - 7th workshop**
  - Priv-1.13.0 -> Priv-2.0 ratified?
  - V ratified by Foundation
  - Complete Linux/KVM, *BSD/Bhyve platform spec agreed