Workshop Introductions and RISC-V Update

Krste Asanović
krste@eecs.berkeley.edu

http://www.riscv.org

2nd RISC-V Workshop, Berkeley, CA
June 29, 2015

Instruction Set Architectures don't matter

Most of performance and energy running software on a processor 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
- If catch bit rot, no longer compile own source code
- Lost your own source code

- Most of the cost of developing a new chip is developing software for it
- Most current large chips have multiple ISAs

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?

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
Architecture	--	--	x86, ARM

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

ISAs Should Be Free and Open

- ISAs proprietary for historical business reasons, no good reason for the lack of free, open ISAs:
- Not an error of omission by ISA owners
- Nor because owners do most software development
- Nor are most popular ISAs, wonderful ISAs
- Nor are companies great stewards of an ISA
- Nor can only owners verify ISA compatibility
- Not as if buying ISA protects you from patent lawsuits
- Finally, proprietary ISAs are not guaranteed to last, and many actually disappear

RISC-V Background

- 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
  - 1300 instructions, x86 ISA manual 2900 pages, instruction length 1 to 15 bytes, ...
- ARM mostly impossible - baroque, IP issues
  - 400 instructions, ARMv7 ISA manual 2700 pages
- So we started "3-month project" in summer 2010 to develop our own clean-slate ISA
- Four years later, we released frozen base user spec
  - But also many tape outs and several research publications

What is RISC-V?

- A new free and open ISA developed at UC Berkeley starting in 2010 (ParLab and ASPIRE)
  - Free as in “beer”, and free as in “speech”
- Designed for
  - research
  - education
  - commercial use
- Not just a free ISA, we also think it’s a good 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

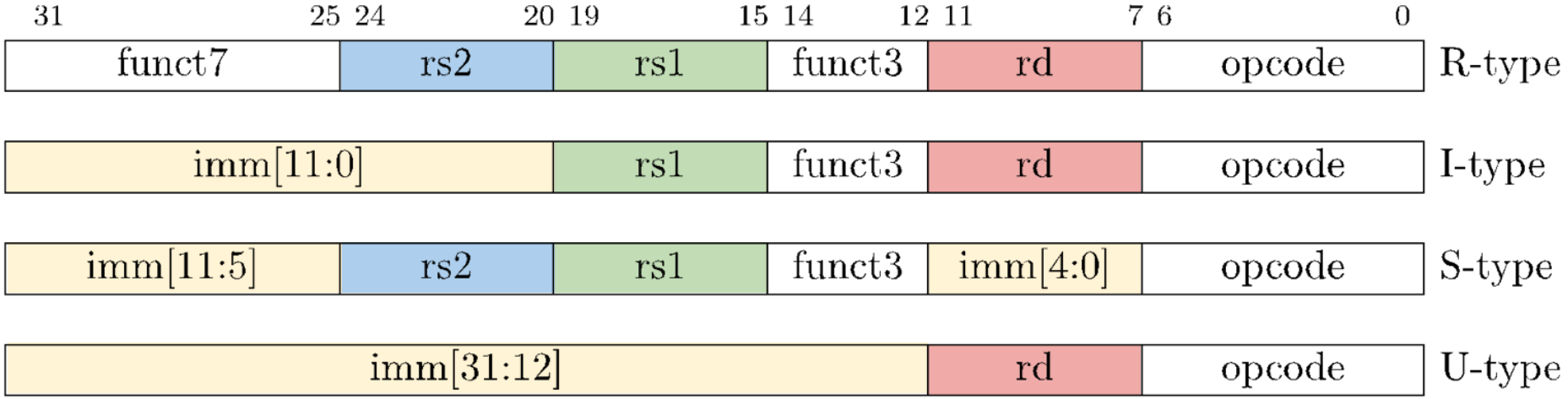
RISC-V is NOT an Open-Source Processor

- RISC-V is an ISA *specification*
- Want to encourage both open-source and proprietary implementations of the RISC-V ISA specification
- Most of cost of hardware design is software, so make sure software can be reused across many chip designs
- Expand to have open specifications for whole platforms, including I/O and accelerators

Frozen RISC-V Base + Standard Extensions

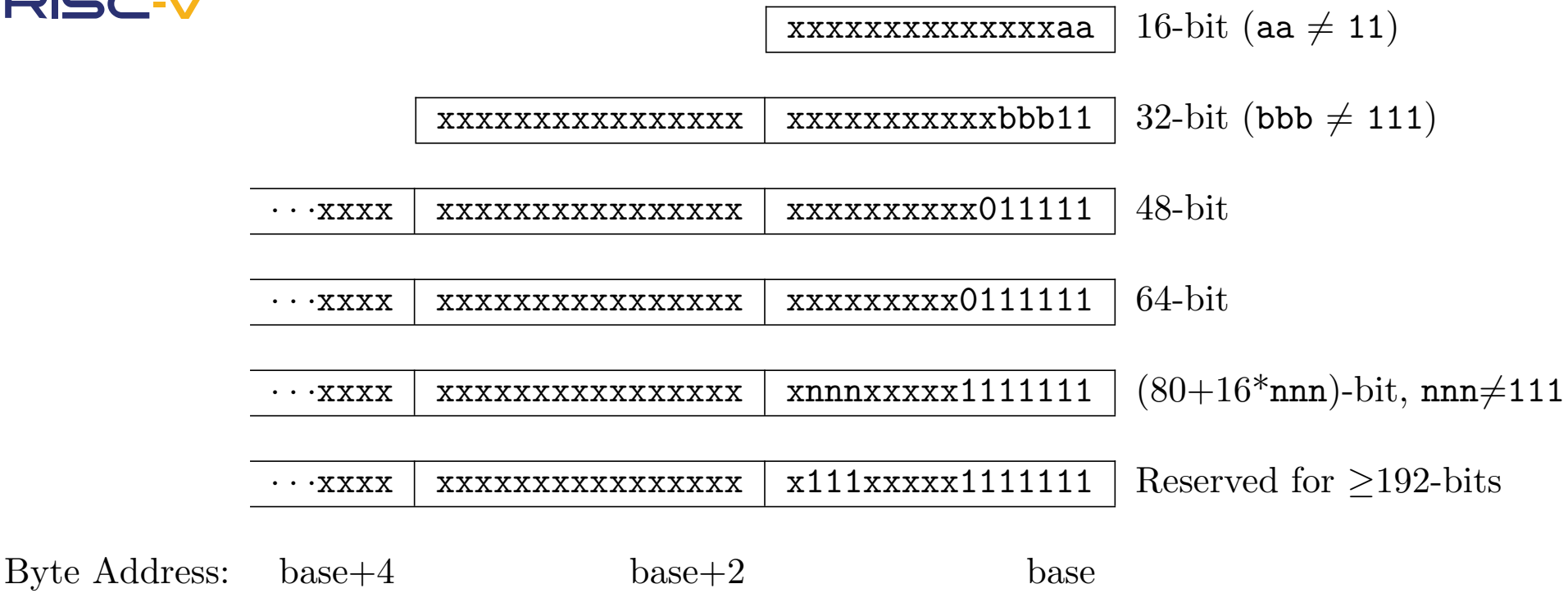
- ~~Three~~ Four base integer ISAs
  - RV32E, RV32I, RV64I, RV128I
  - RV32E is 16-register subset of RV32I
  - Only <50 hardware instructions needed
- Standard extensions
  - M: Integer multiply/divide
  - A: Atomic memory operations (AMOs + LR/SC)
  - F: Single-precision floating-point
  - D: Double-precision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

RISC-V Standard Base ISA Details



- 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



- Extensions can use any multiple of 16 bits as instruction length
- Branches/Jumps target 16-bit boundaries even in fixed 32-bit base

"C": Compressed Instruction Extension

- Compressed code important for:
  - low-end embedded to save static code space
  - high-end commercial workloads to reduce cache footprint
- Standard extension (draft proposal released 5/28) adds 16-bit compressed instructions
  - 2-address forms with all 32 registers
  - 3-address forms with most frequent 8 registers
- Each **C** instruction expands to single base **I** instruction
- Original 32-bit instructions now can be 16-bit aligned
- Approximately 25-30% reduction in code size, with performance improvement from reduced I$ misses
- Surprisingly, lots of 16-bit encode space for future extensions
- ***Talk later today***

ARMv8 ISA vs. RISC-V ISA

Category	ARMv8	RISC-V	ARM/RISC
Year announced	2011	2011	--
Address sizes	32 / 64	32 / 64 / 128	--
Instruction sizes	32	16† / 32	--
Relative code size	1	0.8†	--
Instruction formats	53	6 / 12†	4X-8X
Data addressing modes	8	1	8X
Instructions	1070	177†	6X
Min number instructions to run Linux, gcc, LLVM	359	47	8X
Backend gcc compiler size	47K LOC	10K LOC	5X
Backend LLVM compiler size	22K LOC	10K LOC	2X
ISA manual size	5428 pages	163 pages	33X

MIPS manual 700 pages
80x86 manual 2900 pages

†With optional Compressed RISC-V ISA extension

RISC-V "Green Card"

RV32

RVI Base Instructions: RV32I

Category	Name	Fmt	RV32I Base	
Loads	Load Byte	I	LB	rd,rs1,imm
	Load Halfword	I	LH	rd,rs1,imm
	Load Word	I	LW	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm
	Load Half Unsigned	I	LHU	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm
	Store Halfword	S	SH	rs1,rs2,imm
	Store Word	S	SW	rs1,rs2,imm
Arithmetic	ADD	R	ADD	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm
	Add Upper Imm to PC	U	AUIPC	rd,imm
Shifts	Shift Left	R	SLL	rd,rs1,rs2
	Shift Left Immediate	I	SLLI	rd,rs1,shamt
	Shift Right	R	SRL	rd,rs1,rs2
	Shift Right Immediate	I	SRLI	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI	rd,rs1,shamt
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 < Unsigned Imm	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	UJ	JALR	rd,rs1,imm
Synch	Synch threads	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

32-bit Formats

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
R4	rs3	funct2	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|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

+ 12 for 64I

+ 8 for M

+ 4 for 64M

+ 11 for A

+ 11 for 64A

+ 31 for C

+ 34 for F, D, Q

+ 6 for 64F, 64D, 64Q

RISC-V "Green Card"

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

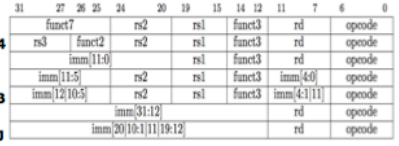
RVI Base Instructions: RV32I, RV64I, and RV128I

Category	Name	Fmt	RV32I Base	+RV64	+RV128
Loads	Load Byte	I	LB rd,rs1,imm		
	Load Halfword	I	LH rd,rs1,imm		
	Load Word	I	LW rd,rs1,imm	LD rd,rs1,imm	LQ rd,rs2,imm
	Load Byte Unsigned	I	LBU rd,rs1,imm		
	Load Half Unsigned	I	LHU rd,rs1,imm	LWU rd,rs1,imm	LDU rd,rs1,imm
Stores	Store Byte	S	SB rs1,rs2,imm		
	Store Halfword	S	SH rs1,rs2,imm		
	Store Word	S	SW rs1,rs2,imm	SD rs1,rs2,imm	SQ rs1,rs2,imm
Arithmetic	ADD	R	ADD rd,rs1,rs2	ADDW rd,rs1,rs2	ADDD rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDIW rd,rs1,imm	ADDID rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	SUBW rd,rs1,rs2	SUBD rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm		
	Add Upper Imm to PC	U	AUIPC rd,imm		
Shifts	Shift Left	R	SLL rd,rs1,rs2	SLLW rd,rs1,rs2	SLLD rd,rs1,rs2
	Shift Left Immediate	I	SLLI rd,rs1,shamt	SLLIW rd,rs1,shamt	SLLID rd,rs1,shamt
	Shift Right	R	SRL rd,rs1,rs2	SRLW rd,rs1,rs2	SRLD rd,rs1,rs2
	Shift Right Immediate	I	SRLI rd,rs1,shamt	SRLIW rd,rs1,shamt	SRLID rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRAW rd,rs1,rs2	SRAD rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI rd,rs1,shamt	SRAIW rd,rs1,shamt	SRAID rd,rs1,shamt
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 < Unsigned Imm	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	UJ	JALR rd,rs1,imm		
Synch	Synch threads	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		

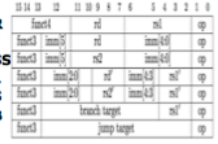
RVC Compressed Instruction Extension

Category	Name	Fmt	RVC	RVI equivalent
Loads	Load Word	CL	C.LW rd',rs1',imm	LW rd',rs1',imm*4
	Load Word SP	CI	C.LWSP rd,imm	LW rd,sp,imm*4
	Load Double	CL	C.LD rd',rs1',imm	LD rd',rs1',imm*8
	Load Double SP	CI	C.LDSP rd,imm	LD rd,sp,imm*8
	Load Quad	CL	C.LQ rd',rs1',imm	LQ rd',rs1',imm*16
	Load Quad SP	CI	C.LQSP rd,imm	LQ rd,sp,imm*16
Stores	Store Word	CS	C.SW rs1',rs2',imm	SW rs1',rs2',imm*4
	Store Word SP	CSS	C.SWSP rs2,imm	SW rs2,sp,imm*4
	Store Double	CS	C.SD rs1',rs2',imm	SD rs1',rs2',imm*8
	Store Double SP	CSS	C.SDSP rs2,imm	SD rs2,sp,imm*8
	Store Quad	CS	C.SQ rs1',rs2',imm	SQ rs1',rs2',imm*16
	Store Quad SP	CSS	C.SQSP rs2,imm	SQ rs2,sp,imm*16
Arithmetic	ADD	CR	C.ADD rd,rs1	ADD rd,rd,imm
	ADD Word	CR	C.ADDW rd,rs1	ADDW rd,rd,imm
	ADD Immediate	CI	C.ADDI rd,imm	ADDI rd,rd,imm
	ADD Word Imm	CI	C.ADDIW rd,imm	ADDIW rd,rd,imm
	ADD Imm * 4	CI	C.ADDI4 rd,imm	ADDI rd,rd,imm*4
	Load Immediate	CI	C.LI rd,imm	ADDI rd,x0,imm
	Load Upper Imm	CI	C.LUI rd,imm	LUI rd,imm
	MoVe	CR	C.MV rd,rs1	ADD rd,rs1,x0
Shifts	Shift Left Imm	CI	C.SLLI rd,imm	SLLI rd,rd,imm
Branches	Branch=0	CB	C.BEQZ rs1',imm	BEQ rs1,x0,imm
	Branch≠0	CB	C.BNEZ rs1',imm	BNE rs1,x0,imm
Jump & Link	J&L	CJ	C.J imm	JAL x0,imm
	Jump & Link Register	CR	C.JALR rd,rs1	JALR rd,rs1,0
System	Sys BREAK	CI	C.SBREAK	SBREAK

32-bit Formats



16-bit Formats



RVM Multiply-Divide Instruction Extension

Category	Name	Format	RV32M (Multiply-Divide)	+RV64	+RV128
Multiply	MULtiply	R	MUL rd,rs1,rs2	MULW rd,rs1,rs2	MULD 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 rd,rs1,rs2	DIVW rd,rs1,rs2	DIVD rd,rs1,rs2
	DIVide Unsigned	R	DIVU rd,rs1,rs2		
Remainder	REMainder	R	REM rd,rs1,rs2	REMW rd,rs1,rs2	REMD rd,rs1,rs2
	REMainder Unsigned	R	REMU rd,rs1,rs2	REMUW rd,rs1,rs2	REMUD rd,rs1,rs2

RVA Atomic Instruction Extension

Category	Name	Format	RV32A (Atomic)	+RV64	+RV128
Load	Load Reserved	R	LR.W rd,rs1	LR.D rd,rs1	LR.Q rd,rs1
Store	Store Conditional	R	SC.W rd,rs1,rs2	SC.D rd,rs1,rs2	SC.Q rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.W rd,rs1,rs2	AMOSWAP.D rd,rs1,rs2	AMOSWAP.Q rd,rs1,rs2
Add	ADD	R	AMOADD.W rd,rs1,rs2	AMOADD.D rd,rs1,rs2	AMOADD.Q rd,rs1,rs2
Logical	XOR	R	AMOXOR.W rd,rs1,rs2	AMOXOR.D rd,rs1,rs2	AMOXOR.Q rd,rs1,rs2
	AND	R	AMOAND.W rd,rs1,rs2	AMOAND.D rd,rs1,rs2	AMOAND.Q rd,rs1,rs2
	OR	R	AMOOR.W rd,rs1,rs2	AMOOR.D rd,rs1,rs2	AMOOR.Q rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.W rd,rs1,rs2	AMOMIN.D rd,rs1,rs2	AMOMIN.Q rd,rs1,rs2
	MAXimum	R	AMOMAX.W rd,rs1,rs2	AMOMAX.D rd,rs1,rs2	AMOMAX.Q rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.W rd,rs1,rs2	AMOMINU.D rd,rs1,rs2	AMOMINU.Q rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.W rd,rs1,rs2	AMOMAXU.D rd,rs1,rs2	AMOMAXU.Q rd,rs1,rs2

Three Floating-Point Instruction Extensions: RVF, RVD, & RVQ

Category	Name	Format	RV32{F,D,Q} (SP,DP,QP Fl. Pt.)	+RV64	+RV128
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	R4	FMADD.{S,D,Q} rd,rs1,rs2,rs3		
	Multiply-SUBtract	R4	FMSUB.{S,D,Q} rd,rs1,rs2,rs3		
	Negative Multiply-SUBtract	R4	FNMSUB.{S,D,Q} rd,rs1,rs2,rs3		
	Negative Multiply-ADD	R4	FNMADD.{S,D,Q} rd,rs1,rs2,rs3		
Move	Move from Integer	R	FMV.S.X rd,rs1	FMV.D.X rd,rs1	FMV.Q.X rd,rs1
	Move to Integer	R	FMV.X.S rd,rs1	FMV.X.D rd,rs1	FMV.X.Q rd,rs1
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		
Convert	Convert from Int	R	FCVT.{S,D,Q}.W rd,rs1	FCVT.{S,D,Q}.L rd,rs1	FCVT.{S,D,Q}.T rd,rs1
	Convert from Int Unsigned	R	FCVT.{S,D,Q}.WU rd,rs1	FCVT.{S,D,Q}.LU rd,rs1	FCVT.{S,D,Q}.TU rd,rs1
	Convert to Int	R	FCVT.W.{S,D,Q} rd,rs1	FCVT.L.{S,D,Q} rd,rs1	FCVT.T.{S,D,Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{S,D,Q} rd,rs1	FCVT.LU.{S,D,Q} rd,rs1	FCVT.TU.{S,D,Q} rd,rs1
Categorization	Classify Type	R	FCLASS.{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		

Simplicity breeds Contempt

- How can simple ISA compete with industry monsters?
- So far, no evidence more complex ISA justified for general code
  - Cray/RISC were right
- Many advantages to keeping base simple
  - Teaching (what profs do)
  - Learning (what engineers do)
  - Area
  - Energy
  - Quality-of-results versus Design time (HW and SW)
  - Verification
  - Security
  - Extensibility: one base ISA for all customized cores on chip

Proposed Standard "V" Vector Extension

- Traditional vector extension
  - Cray-style vector ISA, not packed-SIMD ISA or GPU
- LLVM-based compiler support for two programming models:
  - Auto-vectorization/parallelization (OpenMP)
  - Explicit SPMD (OpenCL)
- In progress
- ***Talk later today***

RISC-V Privileged Architecture

- Draft specification released for comments 5/9/2015
- Four privilege modes
  - User (U-mode)
  - Supervisor (S-mode)
  - Hypervisor (H-mode) // Not specified yet
  - 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)
- ***Talk later today***

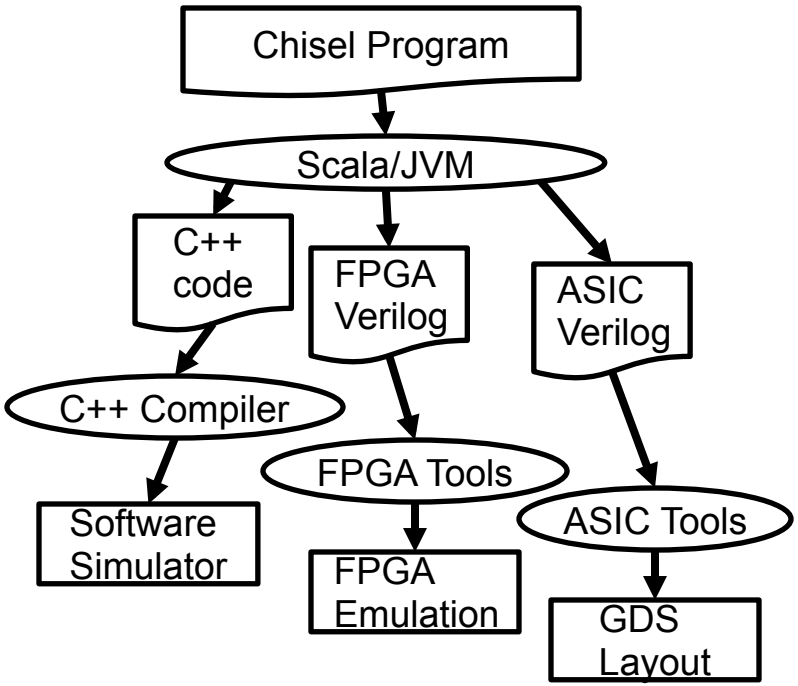
RISC-V Ecosystem

www.riscv.org

- ***Documentation***
  - User-Level ISA Spec v2
  - Privileged ISA draft
  - Compressed ISA draft
- ***Software Tools***
  - GCC/glibc/GDB
  - LLVM/Clang
  - Linux
  - Yocto
  - Verification Suite
- ***Hardware Tools***
  - Zynq FPGA Infrastructure
  - Chisel
- ***Software Implementations***
  - ANGEL, JavaScript ISA Sim.
  - Spike, In-house ISA Sim.
  - QEMU
- ***Hardware Implementations***
  - Rocket Chip Generator
    - RV64G single-issue in-order pipe
  - Sodor Processor Collection
  - External implementations

Chisel: Constructing Hardware In a Scala Embedded Language

- Embed hardware-description language in Scala, using Scala's extension facilities: Hardware module is just data structure in Scala
- Different output routines generate different types of output (C, FPGA-Verilog, ASIC-Verilog) from same hardware representation
- Full power of Scala for writing hardware generators
  - Object-Oriented: Factory objects, traits, overloading etc
  - Functional: Higher-order funcs, anonymous funcs, currying
  - Compiles to JVM: Good performance, Java interoperability



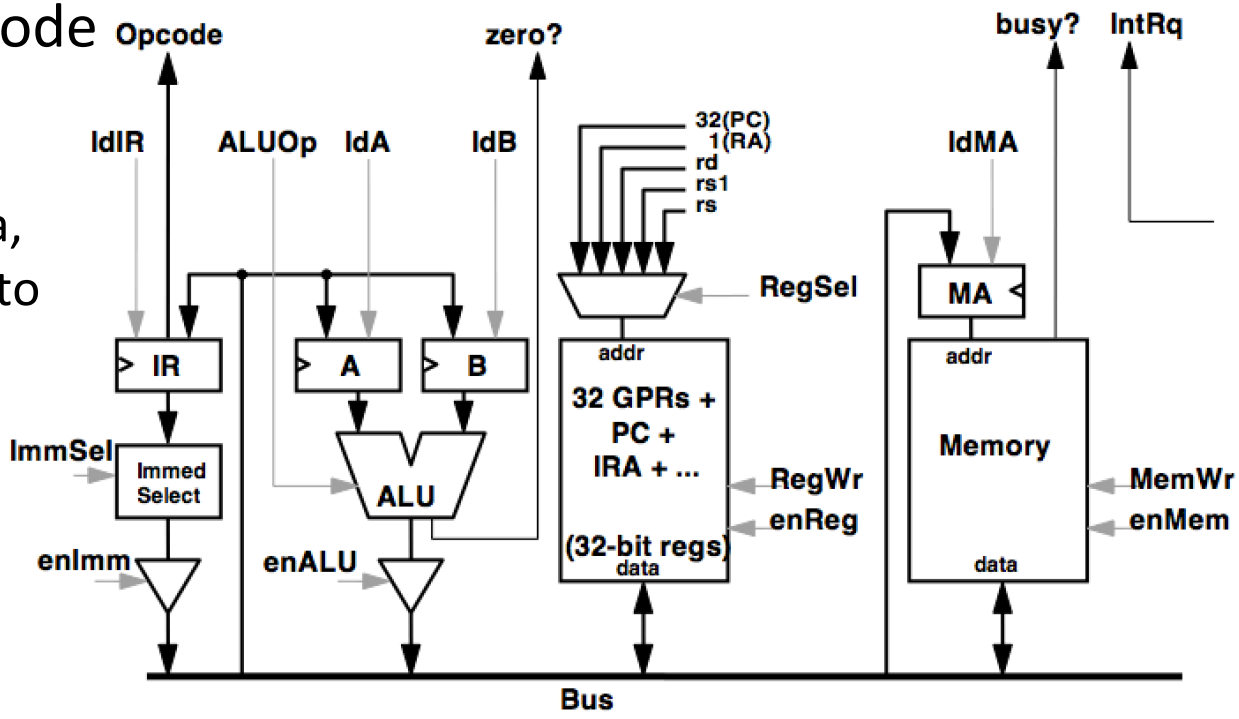
- Chisel 3.0 in development
- Based around new FIRRTL intermediate representation
- Support new language frontends, and new target backends
- Become the "LLVM for Hardware"

Berkeley RISC-V Cores

- Sodor RISC-V educational cores in Chisel
- “Rocket Chip” is our Chisel-based generator that produces complete production chip RTL (cores, uncore, interconnect)

RISC-V/Chisel in the Classroom

- Sodor Educational Processor Repository
  - https://github.com/ucb-bar/riscv-sodor/wiki
- Introduction to Chisel, RISC-V
- Menagerie of RISC-V 32b processors
  - 1-stage, 2-stage, 5-stage
  - 3-stage with synchronous memory
  - Bus-based, micro-code
    - Students write micro-code in a tiny "DSL" inside of Scala, Chisel compiles it into a micro-code ROM



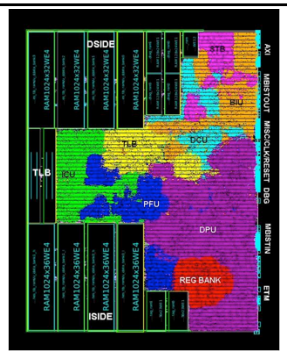
"Rocket Chip" Generator Alpha Release, Released Oct 7, 2014

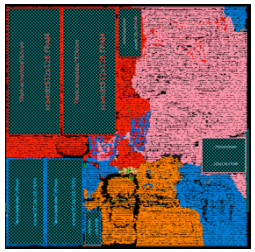
- Single-issue in-order classic 5-stage RISC pipeline
- Fully pipelined IEEE-2008 32-bit/64-bit FPU
- MMU supports Linux, other OS
- Non-blocking data cache
- BTB, BHT, RAS branch prediction
- Coprocessor interface
- Similar design point to ARM A5
- Parameterized generator
- ~5,400 LOC in Chisel for processor
- ~2,000 LOC for floating-point units
- ~4,600 LOC in Chisel for "uncore" (coherence hubs, L2 caches, networks, host/target interfaces)
- ***Updates in talk tomorrow on Z-scale***

ARM Cortex A5 vs. RISC-V Rocket

Category	ARM Cortex A5	RISC-V Rocket
ISA	32-bit ARM v7	64-bit RISC-V v2
Architecture	Single-Issue In-Order 8-stage	Single-Issue In-Order 5-stage
Performance	1.57 DMIPS/MHz	1.72 DMIPS/MHz
Process	TSMC 40GPLUS	TSMC 40GPLUS
Area w/o Caches	0.27 mm^2	0.14 mm^2
Area with 16K Caches	0.53 mm^2	0.39 mm^2
Area Efficiency	2.96 DMIPS/MHz/mm^2	4.41 DMIPS/MHz/mm^2
Frequency	>1GHz	>1GHz
Dynamic Power	<0.080 mW/MHz	0.034 mW/MHz

Rocket Area Numbers Assuming 85% Utilization, the same number ARM used to report area.
Plots are not to scale.





More Berkeley Cores!

- **Z-scale** tiny RISC-V core, ***talk tomorrow***
  - Comparable to ARM M cores
- **BOOM** out-of-order core, ***talk tomorrow***
  - Comparable to ARM A9/A15 cores
- All built within Rocket Chip generator framework

- Even if you don't use these cores, their design and performance evaluations show that RISC-V ISA is competitive

State of the RISC-V Nation

- Significant momentum since HotChips-2014 rollout
- Many companies “kicking the tires”, with varied interests
- Self-assessment:
  - If were thinking of designing own RISC ISA for project, then just use RISC-V
  - If need complete working supported core to inject into product design today, then pay $M for industry core
  - If want simple core and deadline 6 months away, then better to spend $M on RISC-V development

Concerns for RISC-V Ecosystem

- Fragmentation
  - How to stop extensible ISA from becoming 1000 different incompatible RISC-V ISAs?
- Momentum
  - Heavily Berkeley-driven so far, with extensive research funding, but don't we switch research projects frequently?
- Completeness
  - Feature X/Y/Z is missing
- FUD/Attacks
  - Fear of patent lawsuits
- Support
  - Where do I get paid help?

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*."

- A 501(c)(6) foundation in process of incorporating
- Rick O'Connor recruited as Executive Director
- Currently recruiting "founding" member companies

Foundation Principles

- The RISC-V ISA and related standards shall remain open and licence-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.
- Non-commercial use of RISC-V trademark allowed if designs pass compatibility tests.

Foundation Functions

- Official source of information about RISC-V, maintains online repository of RISC-V documents, promotes adoption of RISC-V by organizing both online and live events (such as this workshop)
- Responsible for sustaining and evolving the RISC-V instruction set architecture and surrounding hardware and software ecosystem over time in response to changes in technology and the needs and requests of the user community.
- Manages licensing of the RISC-V trademarks and provides a vehicle to decide whether a project or product can use the RISC-V trademark.
- Maintains directory of public-domain instruction set architecture and micro-architectural techniques, culled from publications and expired patents.
- Produces and sells RISC-V promotional material to raise funds

Foundation Organization

- Seven member board, initially hand-picked but replacements elected by membership
- Board ultimately responsible for fulfilling mission
- Board can amend by-laws with 2/3 vote
- Board blesses ad-hoc committees to work on RISC-V projects, and has final vote of approval on committee recommendations

Foundation Membership Levels

- Sponsor members (one vote per organization)
  - Platinum ($50K/year)
  - Gold ($25K/year)
  - Silver ($5K/year)
- Auditing Member ($2.5K/year) (no vote)
- Individual Member ($100/year) (no vote)
- Student Member ($50/year) (no vote)
- Member of sponsoring/auditing organization gets all rights of member

RISC-V Landscape

RISC-V Foundation (US non-profit)
In process

Govt/Public Open Source	Non-Profit Private Ventures	Commercial Providers
UC Berkeley (Originators, developers)	lowRISC (UK non-profit)	Bluespec
IIT Madras (Indian Govt.)		Rumble
		TenX
		startupX???
		startupY???

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, HP, Huawei, LG, NEC, Microsoft, Nokia, NVIDIA, Oracle, Samsung

Questions?