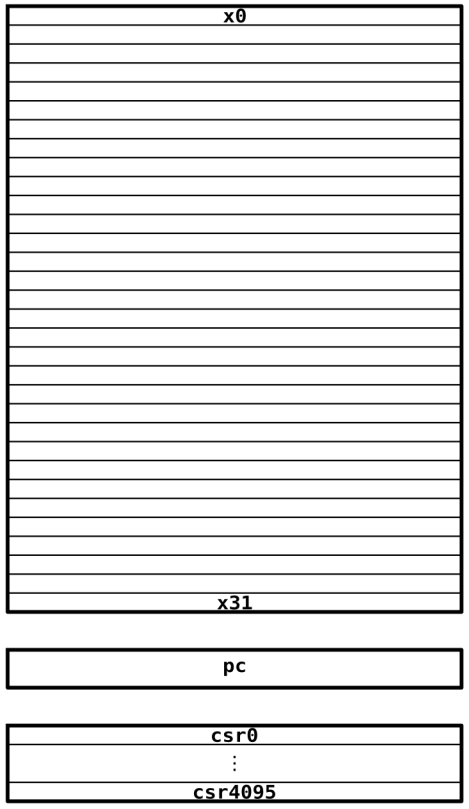
The Challenges of GNU Tool Chain Support for CORE-V

Craig Blackmore
Jeremy Bennett

RISC-V



RISC-V Processor (HART)

Memory

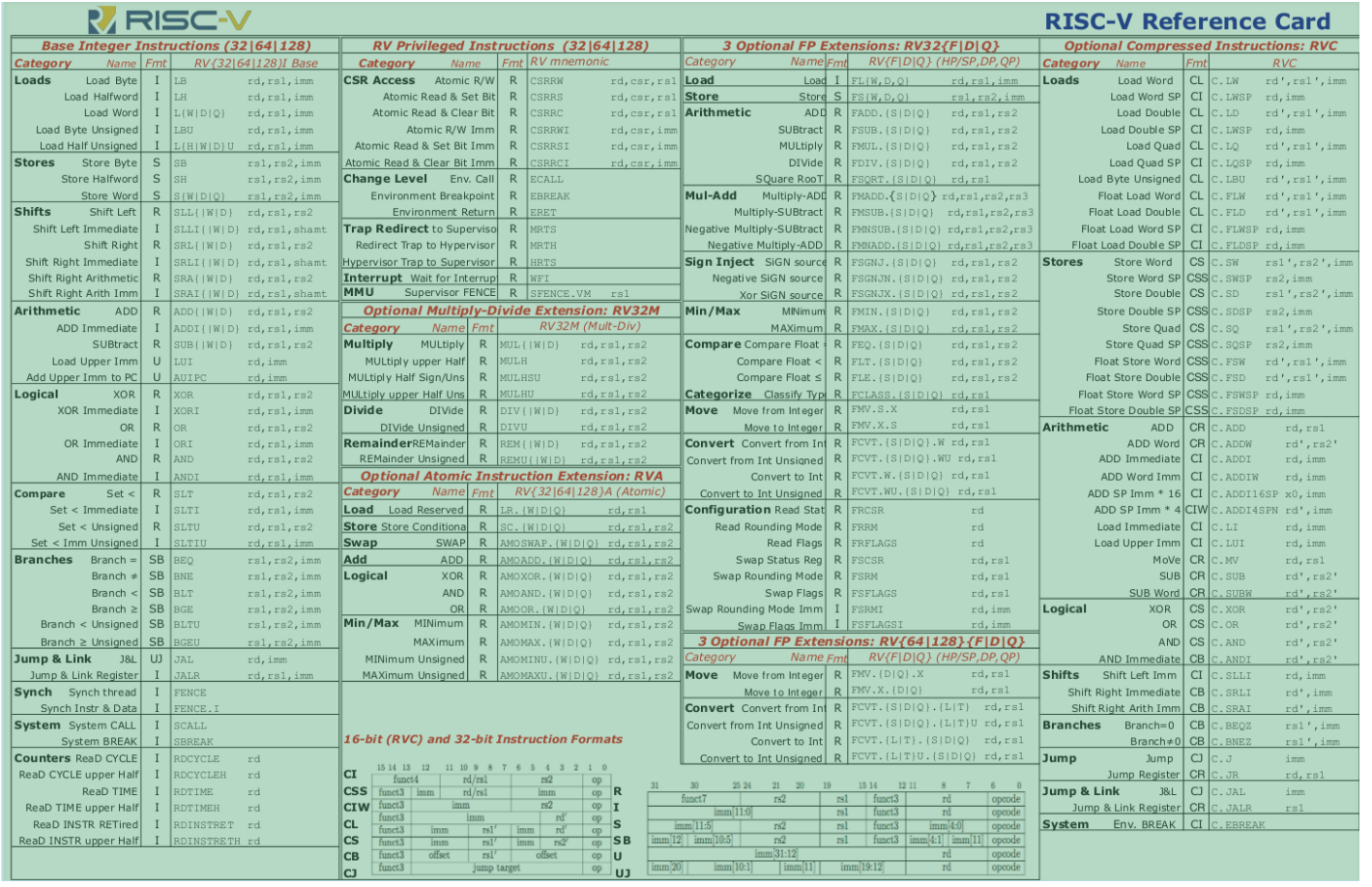
Base Instruction Set
RV32I RV64I
RV32E RV128I

Official ISA Extensions			
M	A	F	
D	Q	(G)	C
L	B	J	T
P	V	N	H
Zxxx			

Vendor ISA Extensions
PULP
Yxxx

EMBECOSM®

RISC-V Instructions “Green Card”



RISC-V Reference Card

Base Integer Instructions (32|64|128)

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

RV Privileged Instructions (32|64|128)

Category	Name	Fmt	RV mnemonic
CSR Access	Atomic R/W	R	CSRRW rd,csr,rs1
	Atomic Read & Set Bit	R	CSRRS rd,csr,rs1
	Atomic Read & Clear Bit	R	CSRRC rd,csr,rs1
	Atomic R/W Imm	R	CSRRWI rd,csr,imm
	Atomic Read & Set Bit Imm	R	CSRRSI rd,csr,imm
	Atomic Read & Clear Bit Imm	R	CSRRCI rd,csr,imm
Change Level	Env. Call	R	ECALL
	Environment Breakpoint	R	EBREAK
	Environment Return	R	ERET
Trap Redirect	to Superviso	R	MRTS
	Redirect Trap to Hypervisor	R	MRTH
	Hypervisor Trap to Supervisor	R	HRTS
Interrupt	Wait for 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

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

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

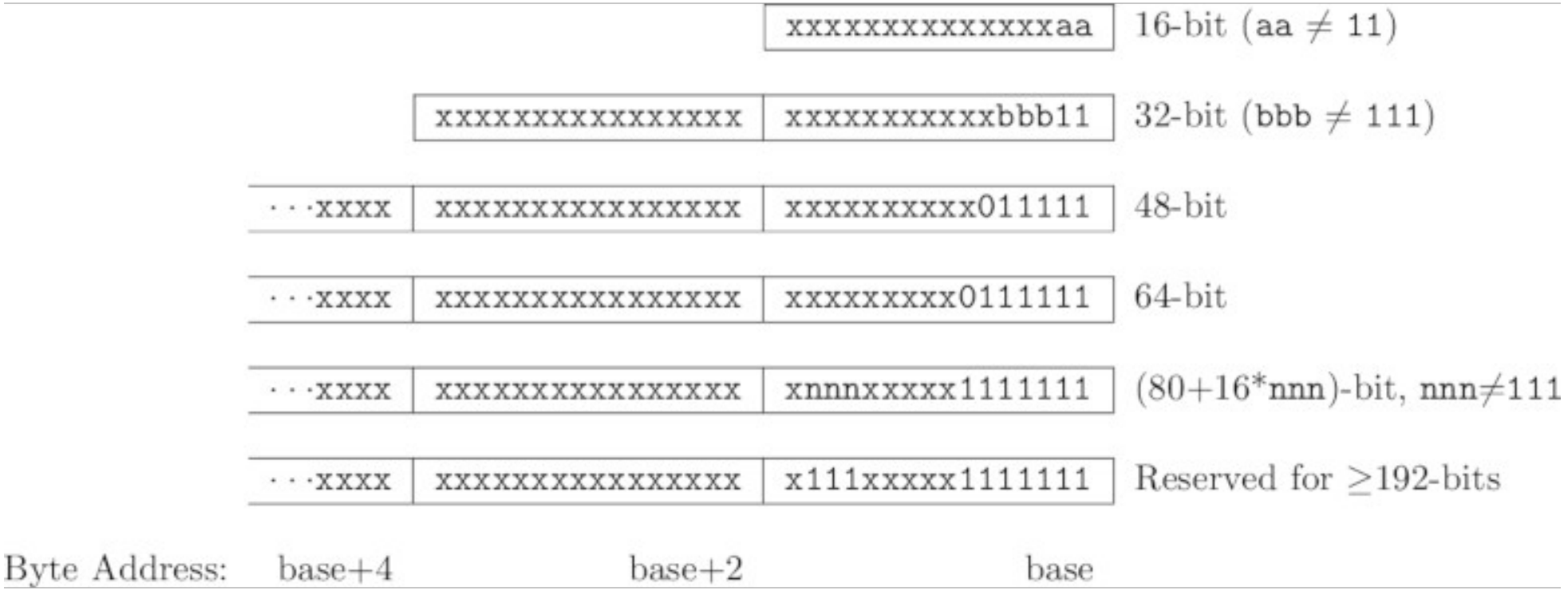
3 Optional FP Extensions: RV{64|128}{F|D|Q}

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

RISC-V Extensible Instructions



Credit: RISC-V International

RISC-V 32-Bit Instruction Formats

Format	Bit																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Upper immediate	imm[31:12]																				rd					opcode						
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	opcode						
Jump	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode						

Credit: Wikipedia

RISC-V 32-bit Opcode Groups

opcode = *xxbbb*11

bbb *xx*	000	001	010	011	100	101	110	111 (>32*b*)
00	LOAD	LOAD-FP		MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48*b*
01	STORE	STORE-FP		AMO	OP	LUI	OP-32	64*b*
10	MADD	MSUB	NMSUB	NMADD	OP-FP			48*b*
11	BRANCH	JALR		JAL	SYSTEM			≥ 80*b*

RISC-V 32-bit Opcode Groups

opcode = *xxbbb*11

bbb *xx*	000	001	010	011	100	101	110	111 (>32*b*)
00	LOAD	LOAD-FP		MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48*b*
01	STORE	STORE-FP		AMO	OP	LUI	OP-32	64*b*
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*		48*b*
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*		≥ 80*b*

RISC-V 32-bit Opcode Groups

opcode = *xxbbb*11

bbb *xx*	000	001	010	011	100	101	110	111 (>32*b*)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48*b*
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64*b*
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2*	48*b*
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3*	≥ 80*b*

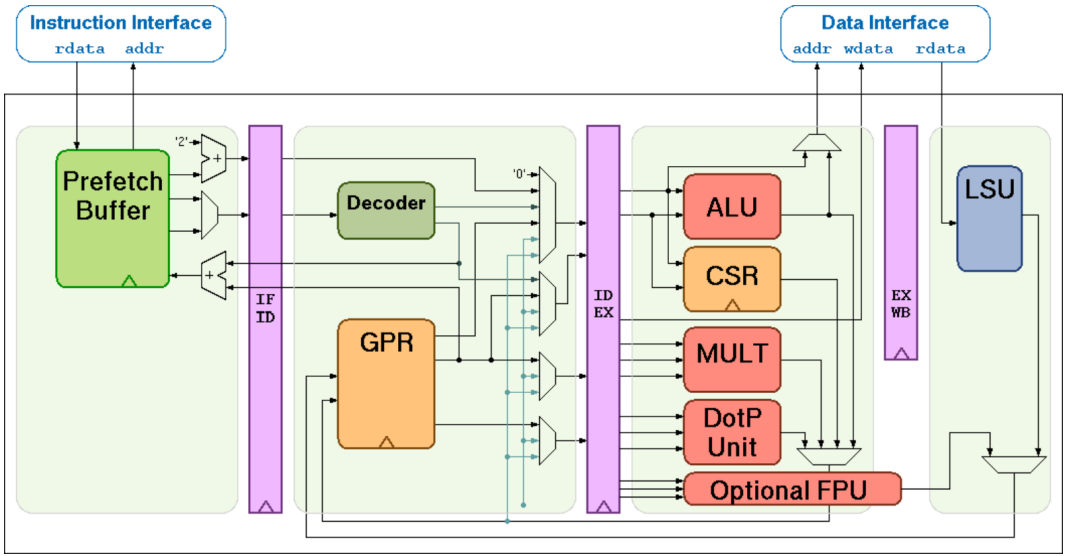
RISC-V 32-bit Opcode Groups

opcode = *xxbbb*11

bbb *xx*	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111** (>32*b*)
00	**LOAD**	**LOAD-FP**	*custom-0*	**MISC-MEM**	**OP-IMM**	**AUIPC**	**OP-IMM-32**	48*b*
01	**STORE**	**STORE-FP**	*custom-1*	**AMO**	**OP**	**LUI**	**OP-32**	64*b*
10	**MADD**	**MSUB**	**NMSUB**	**NMADD**	**OP-FP**	*reserved*	*custom-2*	48*b*
11	**BRANCH**	**JALR**	*reserved*	**JAL**	**SYSTEM**	*reserved*	*custom-3*	≥ 80*b*

custom-2 and *custom-3* are also reserved for **RV128**

Parallel Ultra Low Power (PULP)

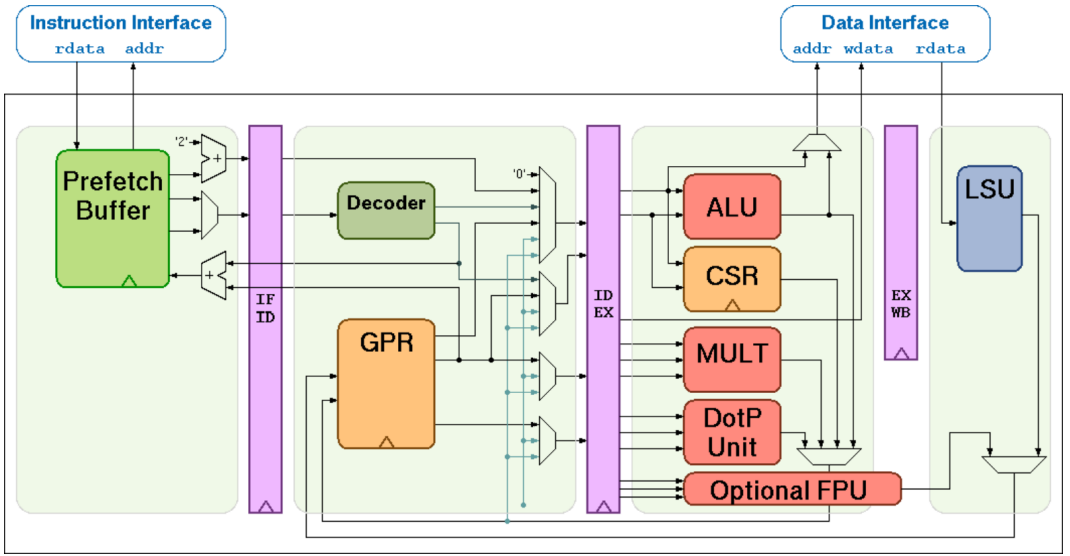


PULP RI5CY core

- 4-stage, in-order
- 32-bit RV32IMFC
- PULP custom extensions
  - post-inc load/store
  - multiply-accumulate
  - ALU extensions
  - hardware loops
  - bit manipulation
  - SIMD

RISC-V Organizations

CORE-V RV32E40P



OpenHW Group CORE-V CV32E40P

- 4-stage, in-order
- 32-bit RV32IMFC
- PULP custom extensions
  - post-inc load/store
  - multiply-accumulate
  - ALU extensions
  - hardware loops
  - bit manipulation
  - SIMD

PULP RI5CY → CORE-V CV32E40P

- Hardware
  - robust verification program

PULP RI5CY → CORE-V CV32E40P

- Hardware
  - robust verification program
- Software
  - need GNU tools, Clang/LLVM tools
  - need RTOS and full-fat OS
  - need IDEs
  - need simulators

PULP RI5CY → CORE-V CV32E40P

- Hardware
  - robust verification program
- Software
  - need GNU tools, Clang/LLVM tools
  - need RTOS and full-fat OS
  - need IDEs
  - need simulators
- Reference platform for FPGA use

CORE-V CV32E40P GNU Tools

- Full GNU tool chain supporting
  - ~150 new instruction types
  - ~350 if you count all the variants
  - based on GCC 7.1

CORE-V CV32E40P GNU Tools

- Full GNU tool chain supporting
  - ~150 new instruction types
  - ~350 if you count all the variants
  - based on GCC 7.1
- University research compiler
  - no tests (not even the upstream ones)
  - no adherence to GNU coding standards
  - tramples all over reserved RISC-V coding space

CORE-V GNU Tools Strategy

- Move instruction encoding to *custom[0123]* (hardware)
  - blocker on any upstreaming
  - leave out SIMD and bit manipulation
    - being standardized officially
- Update GNU tool chain
  - add tests to existing (2017) PULP binutils-gdb
  - roll forward to 2020 binutils-gdb
  - port PULP GCC changes to 2020 GCC

CORE-V GNU Tools Progress

- Back-porting tests to 2017 binutils-gdb failed
  - too much other research code in the way
  - RISC-V binutils-gdb port has changed much in 3 years

CORE-V GNU Tools Progress

- Back-porting tests to 2017 binutils-gdb failed
  - too much other research code in the way
  - RISC-V binutils-gdb port has changed much in 3 years
- Restart adding PULP commits to 2020 binutils-gdb
  - selective choice of just CORE-V features

CORE-V GNU Tools Progress

- Back-porting tests to 2017 binutils-gdb failed
  - too much other research code in the way
  - RISC-V binutils-gdb port has changed much in 3 years
- Restart adding PULP commits to 2020 binutils-gdb
  - selective choice of just CORE-V features
- Should we use CGEN instead?
  - already have a full RISC-V implementation

Upstreaming

- Central principle for the OpenHW Group
  - open source work must be contributed back and shared
- Choose a target “triplet” for non-official RISC-V
  - *arch-subarch-vendor-os-environment*
  - **`riscv32-corev-none-elf`** (for bare metal)
- Add architecture options **`-march=Ycorev-xxx`**
  - allows discrimination between CORE-V features
- Start with binutils-gdb, then GCC, then newlib

Poll: Is This the Right Approach?

A) Yes

B) No

Thank You

craig.blackmore@embecosm.com
jeremy.bennett@embecosm.com
openhwgroup.org

Craig Blackmore
Jeremy Bennett