



CHERI Status

0.9.6

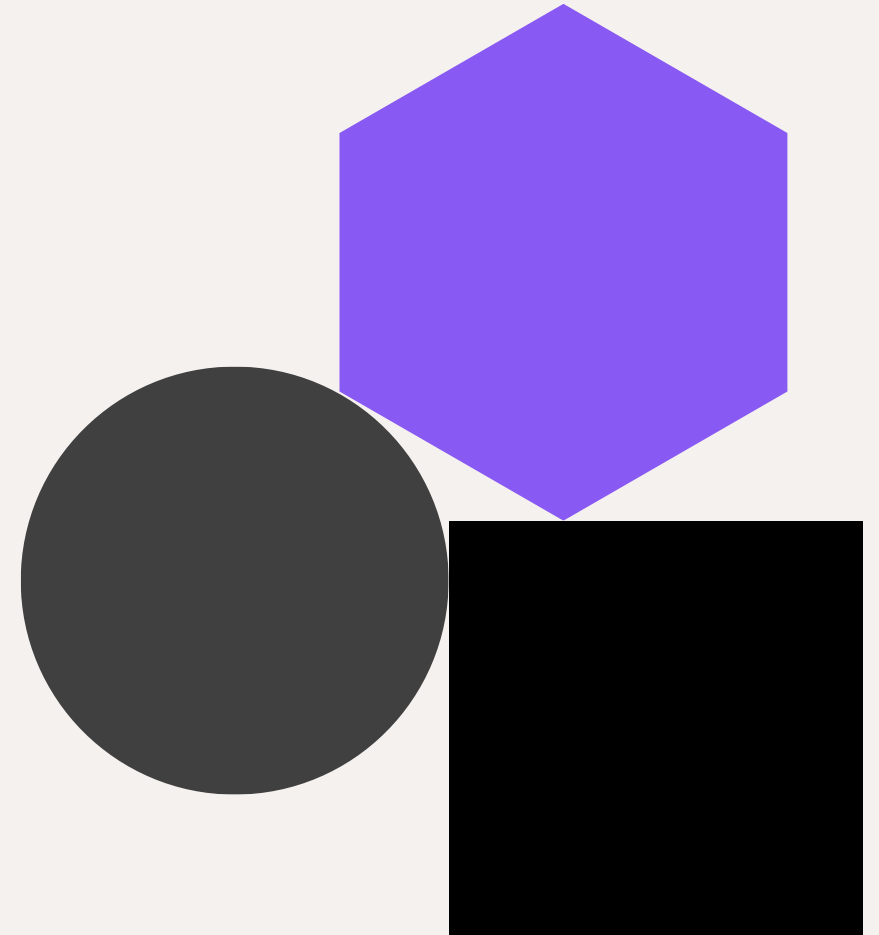
RVY base archs

What has changed and where we are now

Tariq Kurd, Cudasip

CHERITech, Manchester, Nov 2025

🔒 For public presentation



→ Agenda

- RVY bases and profile (not in 0.9.6)
- 0.9.5 to 0.9.6 changes
 - Renaming
 - ISA
 - Exceptions
 - PTEs
 - All changes are listed here: <https://github.com/riscv/riscv-cheri/issues/617>

→ RV Base Architectures

- CHERI has caused a lot of discussion in RISC-V about how to make it fit
 - Not new – this has been the situation for several years.
- The ruling is that if existing instructions change definition, then we need to declare a new base ISA
- It is possible to switch dynamically between BASE ISAs
 - E.g. RV32 / RV64 through MISA.MXL
- Other examples to be converted to bases are:
 - Endianness – BE / LE – as the mode redefines loads and stores
 - Zfinx – as it redefines floating point instructions
- We are currently waiting for guidance on naming

→ RVY Base Architectures

- CHERI is now going to be a set of base architectures
- Encoding formats change behaviour and so need a new base architecture
- We will declare base architectures along the lines of:
 - RV
 - Bit-width: 64/32
 - Y
 - Optional: E (16 registers)
 - Encoding format: University, CHERIoT, vendor encoding format etc.
- All RVY bases inherit all RVI base instructions
 - Addresses are modified to be capabilities
 - AUIPC returns a capability
 - JAL links to a capability
 - JALR jumps to and links to capabilities
 - Capability handling instructions are added like LY/SY
- All bases must specify
 - AUIPC shift
 - Enough details for base ISA instructions to be fully defined
 - R, W, X, C, LM permissions
 - Bounds encodings for tag clearing on capability ADD etc.
 - We expect new bases to be added quickly using fast-track once we have a ratified specification

→ Common base architecture – selectable encoding format

- The RV64Y encoding format:
 - Is suitable for encoding bounds for the full 64-bit address space
 - Has over 20 reserved bits so plenty of space for future expansion
- The RV32Y encoding format is more limited
 - There are already two versions
 - Unversity (better name TBD):
 - Flexible bounds covering up to the 4GB address space
 - Allows out-of-bounds representability
 - Allows hybrid mode execution
 - CHERIoT:
 - Optimised for smaller (up to 8MB) address spaces
 - Otypes
 - We expect more in the future
 - Specific accelerators will have specific requirements for bounds encoding
- Xycfg CSR says which encoding format is in use
 - To be confirmed that this should existing in the specification

→ RVY Base Architecture – new unpriv extensions / features

Extension	Status	Comment
Zys	Stable	Add sentry type for pointer sealing
Zylevels1	Stable	2-level information flow control
Zyhybrid	Stable	Allow RISC-V compatibility mode
Zabhlrsc	Stable	Byte/half LR/SC
Zyseal	Proposal (not v1.0 ratification)	Sealing / unsealing for CHERIoT

→ RVY Base Architecture – new priv extensions

Extension	Status	Comment
Svucrg	Stable	PTE addition for revocation (UAF prevention)
Machine ISA	Stable	Updates to machine ISA, e.g. mtvec is a capability, MRET updated
Supervisor ISA	Stable	Updates to supervisor ISA, e.g. stvec is a capability, SRET updated, allocate PTE.CW

→ RVY Base Architecture – modified existing extensions

Extension	Status	Comment
I	Stable	Use capability types, and add new instructions
C	Stable	Use capability types in compressed encodings
Zicsr	Stable	Allow capability CSRs
Zalrsc	Stable	Add LR.Y/SC.Y
Zaamo	Stable	Add AMOSWAP.Y
Zba	Stable	Add SH[1234]ADDY, SH[1234]ADDY.UW
Zicbo[mzp]	Stable	Add additional permission checks
Zcmp	Proposal	RV32 only – reuse RV64 stack frame layout
Zcmt	Proposal	RV32 only – more discussion required

→ RVY Base Architecture – future extension compatibility

Extension	Status	Comment
Hypervisor	Proposal	Full prototype required to propose frozen version
Vector	Stable	No complete prototype, but spec is very simple so should be ratified in v1.0
Pointer masking	In discussion	Spec needs clarification

→ RYVB profile proposal

- Start with an RVYB proposal
 - Divide and conquer before moving to RVYA
 - RVYM also to be developed
 - RVYB proposal is here: <https://github.com/riscv/riscv-cheri/issues/802>
- Identical to RVB23 except
 - Pointer masking is postponed
 - CHERI additions

→ 0.95 to 0.9.6 - ISA renaming

- All registers have the original RISC-V name
 - rs1 not cs1, rd not cd etc
 - pc not pcc
 - sp not csp
- Removing the renaming is to avoid problems converting new extensions to be CHERI compatible, and to reduce assembly porting complexity
- Capability width registers are now YLEN-bit (+ capability tag)
 - CLEN -> YLEN to match RVY

→ ISA renaming

- Most existing instructions are no longer renamed
 - E.g. AUIPC was renamed to AUIPCC, it is now always AUIPC
 - This affects AUIPC, JAL, JALR, C.ADDI16SPN, C.ADDI4SPN, C.JR, C.JAL, C.JALR
- Where instructions are renamed they always have a **Y** in them
 - The exception for the rule above is C.YMV which is renamed from C.MV to show that it's a YLEN copy not an XLEN copy

→ ISA renaming – load/store/atomic

- All mnemonics renamed
- C type becomes Y type – so all load/store use Y
- LY, SY, C.LY, C.LYSP, C.SY, C.SYSP
- LR.Y, SC.Y, AMOSWAP.Y
- HLV.Y, HSV.Y

→ ISA renaming - ADDs

- CADD, CADDI renamed to ADDY, ADDIY
- SH[1234]ADDY and SH[1234]ADDY.UW
 - All are new encodings, none are CHERI mode dependent
 - There's no Y equivalent for ADD.UW

→ ISA renaming – YLEN arithmetic

- Two YLEN instructions added:
 - PACKY – pack two XLEN values into a YLEN register
 - SRLIY – XLEN place right shift
- These are used to implement two pseudoinstructions
 - YHIW (Y-high-write – formerly SCHI)
 - YHIR (Y-high-read – formerly GCHI)
- They pave the way for more YLEN arithmetic like RV32Y-DinY
 - double precision FP in 64-bit Y registers
- Or VinY
 - Vector in Y registers

→ ISA renaming – Others

- Y-noun-verb naming
 - E.g. GCTAG (Get Capability Tag) becomes YTAGR – Y-TAG-Read
- YBNDSIW has a larger immediate (10-bit)

New	Old	New	Old
YTAGR	GCTAG	SYEQ	SCEQ
YPERMR	GCPERM	YSENTRY	SENTRY
YMV	CMV	YLT	SCSS
YHIR (pseudo)	GCHI	YBLD	CBLD
YHIW (pseudo)	SCHI	YAMASK	CRAM
YBNDSW	SCBNDS	YBASER	GCBASE
YBNDSWI	SCBNDSI	YLENR	GCLEN
YBNDSRW	SCBNDSR	YTYPER	GCTYPE
YMODEW	SCMODE	YMODESWY	MODESW.CAP
YMODER	GCMODE	YMODESWI	MODESW.INT

→ ISA renaming – Others

New	Old	New	Old
C.ADDI16SP	C.CADDI16SP	LY	LC
C.ADDI4SPN	C.CADDI4SPN	SY	SC
C.YMV	C.CMV	C.LYSP	C.LCSP
JAL	CJAL	C.SYSP	C.SCSP
C.JAL	C.CJAL	C.LY	C.LC
JALR	CJALR	C.SY	C.SC
C.JALR C.	CJALR	LR.Y	LR.C
C.JR	C.CJR	SC.Y	SC.C
		AMOSWAP.Y	AMOSWAP.C
HLV.Y	HLV.C	HSV.Y	HSV.C

→ ISA renaming – New instructions

New	Old	Comment
SH[1234]ADDY	SH[1234]ADD	No longer remapped – new encodings added
SH[1234]ADDY.UW	SH[1234]ADD.UW	
YPERMC	ACPERM	Move from ANDing permissions to clearing
SRLIY	GCHI	Pseudo for YHIR
PACKY	SCHI	Pseudo for YHIW
YSUNSEAL	N/A	New instruction

- Integer SH4ADD no longer exists

→ Extension changes

- RVY bases (post 0.9.6)
- Zys – YSENTRY – separated out
- Zish4add – removed – encodings now in Zba
 - Adding Zba gives additional instructions
- Zylevels1 added
 - No longer in base ISA

→ 16-bit RV32Y Id/st mapping unchanged – for reference

A.3.6. RV32 / RV32Y RVC load/store mapping summary

Table 48. 16-bit load/store instruction mapping in RV32I

Encoding		Supported Extensions				
[15:13]	[1:0]	Zca	Zcf	Zcd	Zcmp/ Zcmt	Zclsd
111	00	N/A	C.FSW	N/A	N/A	C.SD
011	00	N/A	C.FLW	N/A	N/A	C.LD
111	10	N/A	C.FSWSP	N/A	N/A	C.SDSP
011	10	N/A	C.FLWSP	N/A	N/A	C.LDSP
101	00	N/A	N/A	C.FSD	reserved	N/A
001	00	N/A	N/A	C.FLD	reserved	N/A
101	10	N/A	N/A	C.FSDSP	Zcmp/ Zcmt	N/A
001	10	N/A	N/A	C.FLDSP	reserved	N/A

Table 49. 16-bit load/store instruction mapping in RV32Y

Encoding		Supported Extensions		
[15:13]	[1:0]	Zca	Zcd	Zcmp/ Zcmt
111	00	C.SY		
111	10	C.SYSP		
011	10	C.LYSP		
011	00	C.LY		
101	00	N/A	C.FSD	reserved
001	00	N/A	C.FLD	reserved
101	10	N/A	C.FSDSP	Zcmp (RV32Y)/ Zcmt (RV32Y)
001	10	N/A	C.FLDSP	reserved

→ 16-bit RV64Y Id/st mapping unchanged – for reference

A.3.7. RV64 / RV64Y RVC load/store mapping summary

Table 50. 16-bit load/store instruction mapping in RV64I

Encoding		Supported Extensions		
[15:13]	[1:0]	Zca	Zcd	Zcmp/ Zcmt
111	00	C.SD	N/A	N/A
011	00	C.LD	N/A	N/A
111	10	C.SDSP	N/A	N/A
011	10	C.LDSP	N/A	N/A
101	00	N/A	C.FSD	reserved
001	00	N/A	C.FLD	reserved
101	10	N/A	C.FSDSP	Zcmp/ Zcmt
001	10	N/A	C.FLDSP	reserved

Table 51. 16-bit load/store instruction mapping in RV64Y

Encoding		Supported Extensions
[15:13]	[1:0]	Zca
111	00	C.SD
011	00	C.LD
111	10	C.SDSP
011	10	C.LDSP
101	00	C.SY
001	00	C.LY
101	10	C.SYSP
001	10	C.LYSP

→ Integrity checks

- Integrity checks were different for different instructions, split into 3 categories
 - Any reserved bits set
 - Malformed bounds
 - The permissions are legal
- They are now consistent – all or none are checked.

→ CBLD->YBLD

- YBLD output is always unsealed
 - For CHERIoT compatibility

→ Levels

- Zylevels1 extension added
 - LG-perm (replacement for EL-perm)
 - SL-perm (as before – but only 1-bit in the spec)
 - GL-flag (replacement for CL but inverted)
- This now matches the CHERIoT scheme

→ ACPERM->YPERMC

- The permission rules have been refactored to allow rules to be added by the capability encoding
- Rule added for RV64 that M requires X
- Rules for RV32 have been regenerated – but have the same overall effect

→ GCPERM / YPERMR

- GCPERM / YPERMR a bit different for RV32 – this case returns zero
- Note that SL is only 1-bit now

Quadrant 1: Executable capabilities										
bit[0] - M-bit (0-Capability Pointer Mode, 1-Integer Pointer Mode)										
Bits[4:3]	R	W	C	LM	EL	SL	X	ASR	Mode ¹	
0-1	✓	✓	✓	✓	✓	∞	✓	✓	Mode ¹	Execute + ASR (see Infinite)
2-3	✓		✓	✓	✓	∞ ²	✓		Mode ¹	Execute + Data & Cap RO
4-5	✓	✓	✓	✓	✓	∞	✓		Mode ¹	Execute + Data & Cap RW
6-7	✓	✓				0 ²	✓		Mode ¹	Execute + Data RW

Quadrant 1: Executable capabilities										
bit[0] - M-bit (0-(CHERI) Capability Mode, 1-(Non-CHERI) Address Mode)										
Field[2:0]	R	W	C	LM	LG	SL	X	ASR	Mode ¹	Notes
0-1	✓	✓	✓	✓	✓	✓	✓	✓	Mode ¹	Execute + Data & Cap RW + ASR
2-3	✓		✓	✓	✓		✓		Mode ¹	Execute + Data & Cap RO
4-5	✓	✓	✓	✓	✓	✓	✓		Mode ¹	Execute + Data & Cap RW
6-7	✓	✓					✓		Mode ¹	Execute + Data RW

→ Exceptions

- Simpler exception types

32	Writable	CHERI Instruction Access Fault
33	Writable	CHERI Load Access Fault
34	Writable	CHERI Store/AMO Access Fault
35	Writable	CHERI Load Page Fault
36	Writable	CHERI Store/AMO Page Fault

- No more sealed, permissions etc. fault types
- No more use of Xtval2
- 32 is the old PCC fault
- 33 is any fault from a load, 34 is any fault from a store/amo
- 35 and 36 are the new PTE faults

→ PTE bits

- Additional PTE bit allocated – PTE.CRW – and existing bits moved

Table 47. Summary of Store CRW and CD behavior in the PTEs

PTE.CRW	PTE.CD	PTE.CRG	Store/AMO
0	0	0	CHERI Store/AMO Page Fault if stored tag is set
0	1	X	Reserved
1	0	X	<u>CHERI Store/AMO Page Fault if stored tag is set</u> , or hardware CD update
1	1	X	Normal operation

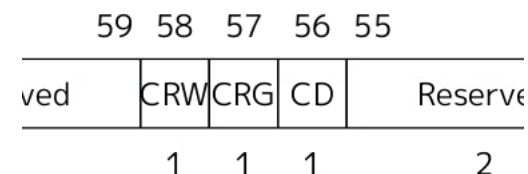


Table 46. Summary of Load CRW and CRG behavior in the PTEs

PTE.CRW	PTE.CD	PTE.CRG	PTE.U	Load/AMO
0	0	0	X	Clear loaded tag
0	0	1	X	Reserved
0	1	X	X	Reserved
1	X	≠ sstatus.UCRG	1	<u>CHERI Load Page Fault</u> , or CHERI Load Page Fault if tag is set for <i>Svucrglct</i> ¹
1	X	= sstatus.UCRG	1	Normal operation
1	X	X	0	Normal operation ²

→ CHERI enables

- CRE -> Y
- Mseccfg.CRE -> MISA.Y
- Xenvcfg.CRG -> Xenvcfg.Y

→ Illegals

- BEQ/BNE illegal if $rs1 \geq rs2$
- Reserve the behaviour for (e.g.) branch on tag value
- Ld/st/atomic with $rs1 = x0$ is still illegal
 - Question over whether this is implementable in SAIL?

→ Branch target exceptions

- No exceptions on branches, and there's only one type of PC fault
 - No ALU exceptions are possible any more
- So if JALR jumps to a bad capability then it gets installed in the PC and then the exception is taken at the target
 - This is identified as a possible problem for trapping on jumps out of a compartment
 - We may add an extension in future to re-introduce this – but for now it has gone
- If a conditional branch jumps out of bounds then the exception is also taken at the target

→ CSRs

- dincf replace by drootc and drootcsel
 - Because CHERIoT has multiple roots (i.e. multiple infinite caps – one for code and one for data)
- Xycfg added (m/s/vs/uycfg) – presence to be confirmed
 - Just a constant value – the capability encoding in use and the presence of Zylevels1