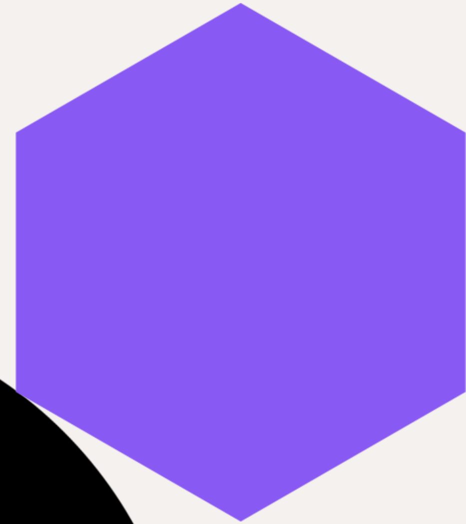
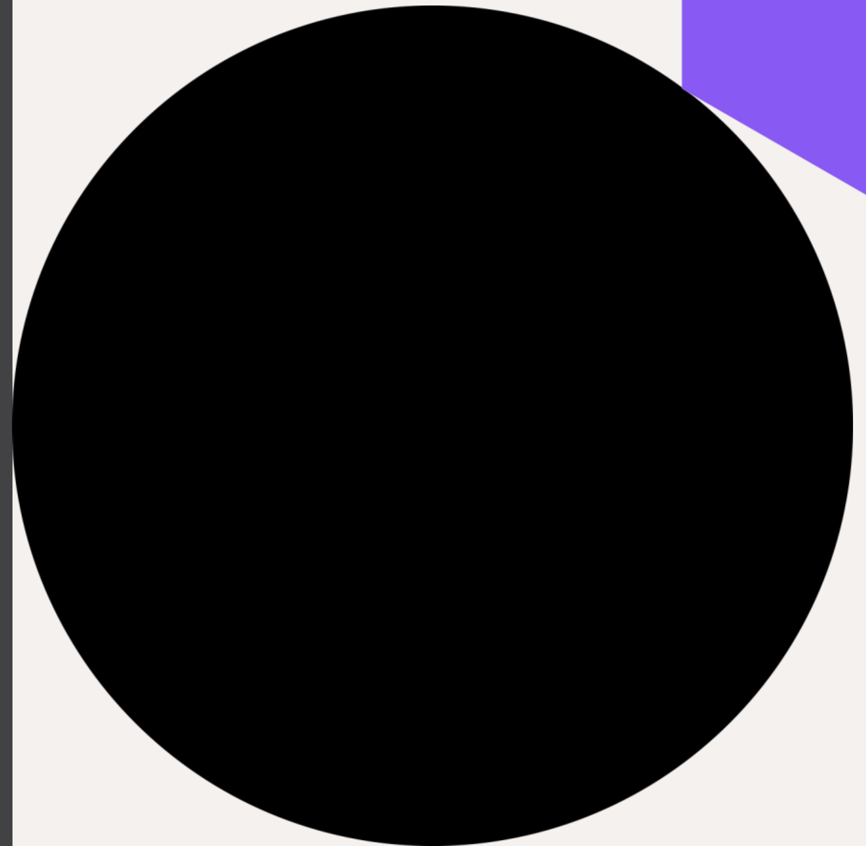
RISC-V Code Size Reduction with Zc and beyond





RISC-V Summit Barcelona

June 2023

Tariq Kurd

Codasip

Finally - Zc is ratified!

Thanks to all contributors!

© 2023 Codasip. All rights reserved.

→ Zc extensions (grey - existing, purple - new)

Codasip

Zca
Zcd
Zdf

Zcb

Zcmp

Zcmt

→ Lower power
Smaller flash sizes
Smaller chip-area
Higher performance

© 2023 Codasip. All rights reserved.

→ The Zc extensions

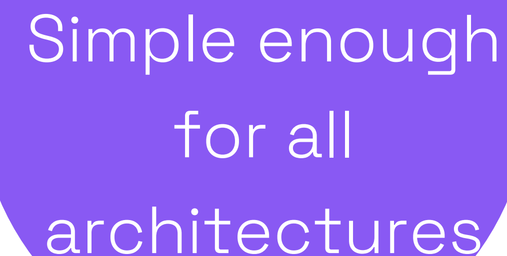
Codasip

- Zca - C extension excluding c.f* (all fp load/store)]
- Zcf - c.flw*, c.fsw*
  - C = Zca+Zcf if F is implemented
  - *Zcf encodings could be repurposed in the future*
- Zcd - c.fld*, c.fsd*
  - C = Zca+Zcf+Zcd if D is implemented (implies F)
- Zcmp/Zcmt already repurpose some of the Zcd encodings
  - *They reuse some of the code points from c.fsdsp*
- Zcb - only allocates reserved encodings

© 2023 Codasip. All rights reserved.

→ The Zcb extension

- 11 (RV32)/12(RV64) 16-bit encodings
  - Map onto existing functionality
- Included in RVA23
  - should be used wherever C is included
  - requires Zbb, M (or Zmmul) to get the full set



- Typically saves 0.5-2% of code-size and only costs (up to) 12 lines in your decoder

© 2023 Codasip. All rights reserved.

→ Zcb instruction summary

Codasip

16-bit mnemonic	32-bit target encoding	16-bit mnemonic	32-bit target encoding	16-bit mnemonic	32-bit target encoding
c.lbu	lbu	c.zext.b	andi …, 0xff	c.zext.w	add.uw …, zero (Zbb)
c.lhu	lhu	c.sext.b	sext.b (Zbb)	c.sext.h	sext.h (Zbb)
c.lh	lh	c.zext.h	zext.h (Zbb)	c.not	xori …., -1
c.sb	sb	c.sh	sh	c.mul	mul (M/Zmmul)

© 2023 Codasip. All rights reserved.

→ The Zcmp extension

- Targets function call prologues and epilogues
- Push/Pop functionality
  - Gives the expansion from 16-bit encodings to a series of existing 32-bit encodings
- Moving two argument registers at once
  - to/from a0,a1 and saved registers
- All reuse encoding space from **c.fsdsp**

- Saves 6.5% on average, but can be *much* higher

© 2023 Codasip. All rights reserved.

→ The Zcmt extension

Codasip

- Replace 32/64-bit sequences used for static function calls
- Huge benefits for large executables
  - In the v8 javascript engine: using only one table jump entry to replace every 64-bit call to one debug symbol called by every assertion saved ~400KB!
- 64-bit calls are required when the function is more than ±1MB away
- Also reuses encoding space from **c.fsdsp**

- Saves 4% on average, but many benchmarks save over 5%

© 2023 Codasip. All rights reserved.

→ Room for future 16-bit encodings

Codasip

- 16 bit load/store instructions use 1024 code-points each
  - Zcd and Zcf have 4096 code points each
- New instructions use minimal code-points
  - Zcmt uses 256
  - Zcmp uses
    - 64 - cm.push
    - 192 - cm.pop*
    - 128 - cm.mv*
- Zcd encoding space is currently 15.6% allocated
- Zcf encoding space is currently 0% allocated

→ Zcd/Zcf encoding space is only 7.8% allocated

© 2023 Codasip. All rights reserved.

→ Instruction table for custom extension

Codasip

Based on well-known dictionary compression techniques

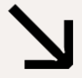
Benchmarking results show 6%-7% reduction in code-size

Very similar to Zcmt - but with table of common 32-bit instructions

Memory table of 32-bit instructions indexed with a 16-bit encoding

→ Will be available in Codasip cores later this year

© 2023 Codasip. All rights reserved.



Summary

- Zc saves about 12.5% on average
  - The actual saving varies wildly with the application
- Instruction table gets this to maybe 18%
  - Good enough to be competitive with existing commercial architectures
- Space available for more compressed encodings
- With RISC-V cores being so popular these code-size savings are really important
  - lower power, smaller flash sizes, smaller chip-area
  - higher performance due to better cache utilisation etc.

© 2023 Codasip. All rights reserved.

→ Useful links

Codasip

- Zc benchmarking
  - https://docs.google.com/spreadsheets/d/1bFMyGkuuuIBXuIaMsjBINoCWoLwObr1I9h5TAWN8s7k/edit#gid=1837831327
- Prebuilt toolchain (go to CORE-V)
  - https://www.embecosm.com/resources/tool-chain-downloads/
- Development status of SAIL, ACT etc:
  - https://github.com/riscv-admin/dev-partners/issues
  - see issues 2, 4, 5, 6
- LLVM development
  - https://github.com/plctlab/llvm-project/tree/riscv-zce-llvm14

© 2023 Codasip. All rights reserved.