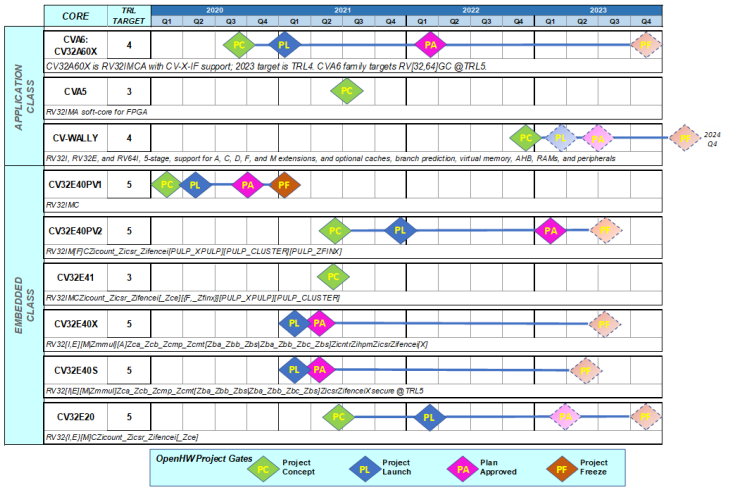
The CORE-V Software Ecosystem

Ten lessons learned from developing vendor specific compiler tool chains

Jeremy Bennett

OpenHW Group and CORE-V



CORE-V: A family of 32- and 64-bit RISC-V cores & system software

About Embecosm

- Open source software consultancy, focusing on
  - compiler tool chains
  - pre-silicon models
  - operating system bring up
  - Bayesian inference AI/ML
- R&D centers in Southampton, Paris and Nürnberg

Lesson 1: Rebase Often

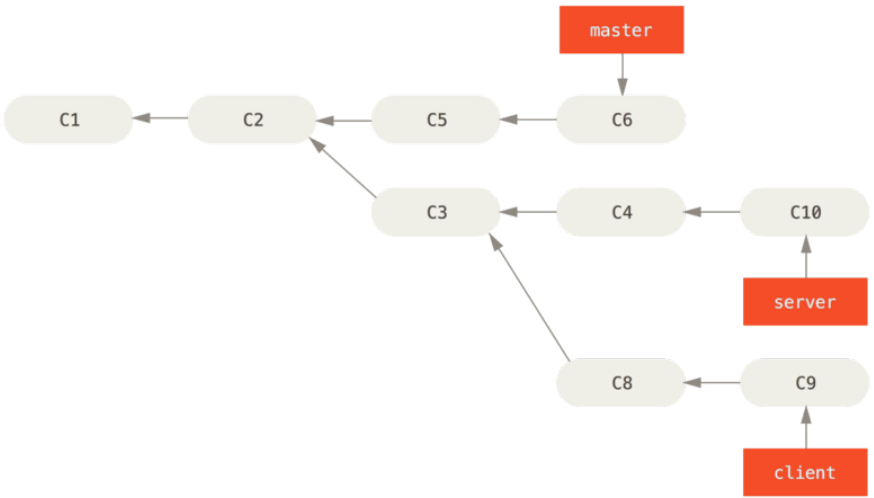


Image: git-scm.com CC-BY-NC-SA

- RISC-V tools are under active development
- Rebasing from an old mirror is laborious
- You want to build on the latest developments
- See lesson 8...

Lesson 2: Avoid Committees

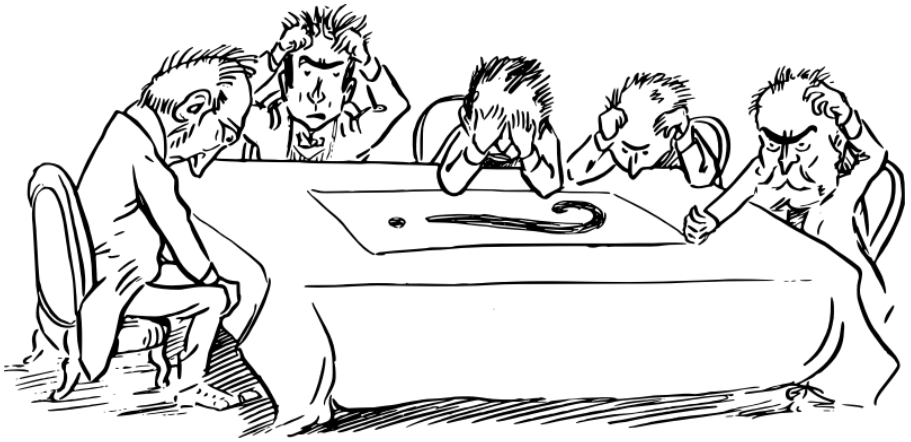


Image: Firkin at openclipart.org

- RISC-V standardization needs committees
- Committee decisions take hard work = time
- Avoid dependencies on a committee decision

Lesson 3: Single Version of Extensions

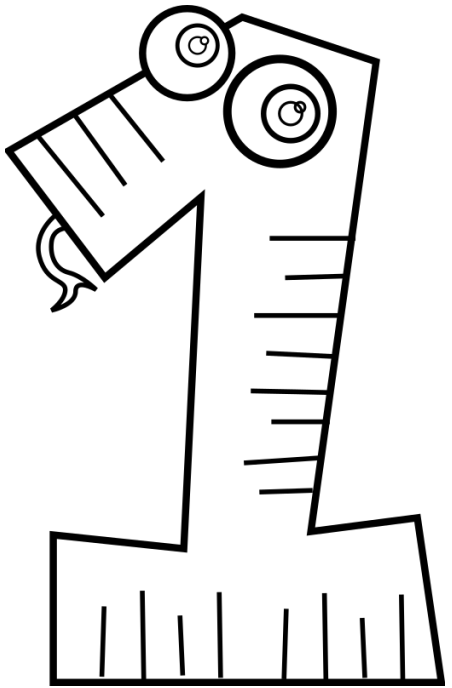


Image: horse50 at openclipart.org

- Multiple version support in tools is incomplete
- Variant instruction encodings not supported
- CORE-V had to abandon its original encodings

Lesson 4: Use Lower Case Assembler

16-Bit x 16-Bit Multiplication

```
cv.muluN     rD, rs1, rs2, Is3
cv.mulhhuN   rD, rs1, rs2, Is3
cv.mulsN     rD, rs1, rs2, Is3
cv.mulhhsN   rD, rs1, rs2, Is3
cv.muluRN    rD, rs1, rs2, Is3
cv.mulhhuRN  rD, rs1, rs2, Is3
cv.mulsRN    rD, rs1, rs2, Is3
cv.mulhhsRN  rD, rs1, rs2, Is3
```

- GNU assembler assumes lower case
  - not deliberate, but no-one had ever tried mixed-case
  - too much effort to fix
- CORE-V had to change mixed-case instructions

Lesson 5: Vendor Specific Relocations

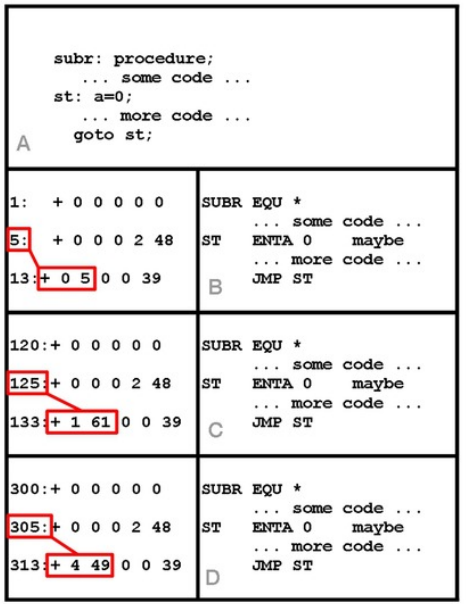


Image: Peter Flass at commons.wikimedia.org

- RISC-V psABI provides 64 vendor relocations
  - enough for one vendor, but not all vendors
- Add a new 32-bit relocation to identify vendor
  - use in pairs
- Needs standardizing
  - reference implementation in development

Lesson 6: Use Builtin Functions Wisely

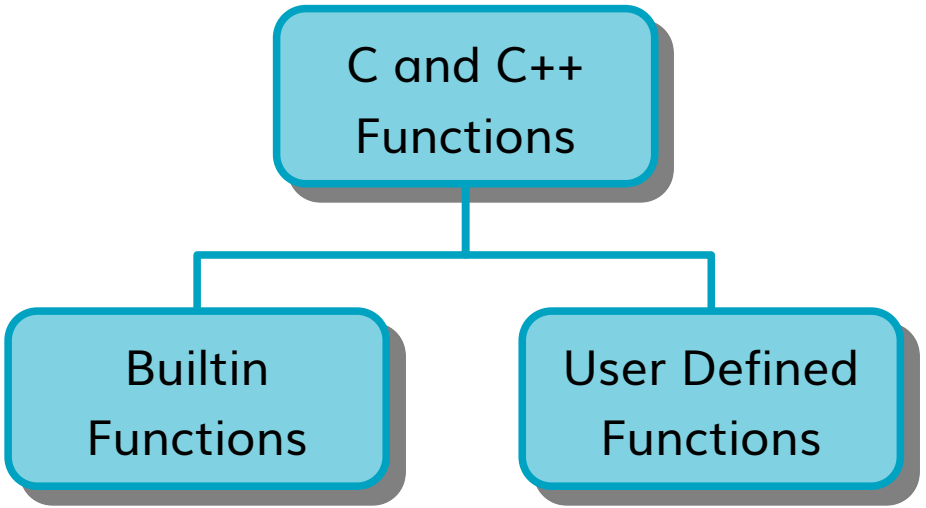


Image: Public Domain

- Don't slavishly create one builtin per ISA opcode
- Define builtins to suit the user
- Example:

  `__builtin_riscv_cv_simd_shuffle_sci_b`

  - maps to one of four CORE-V instructions

Lesson 7: Not Everything Needs Builtins

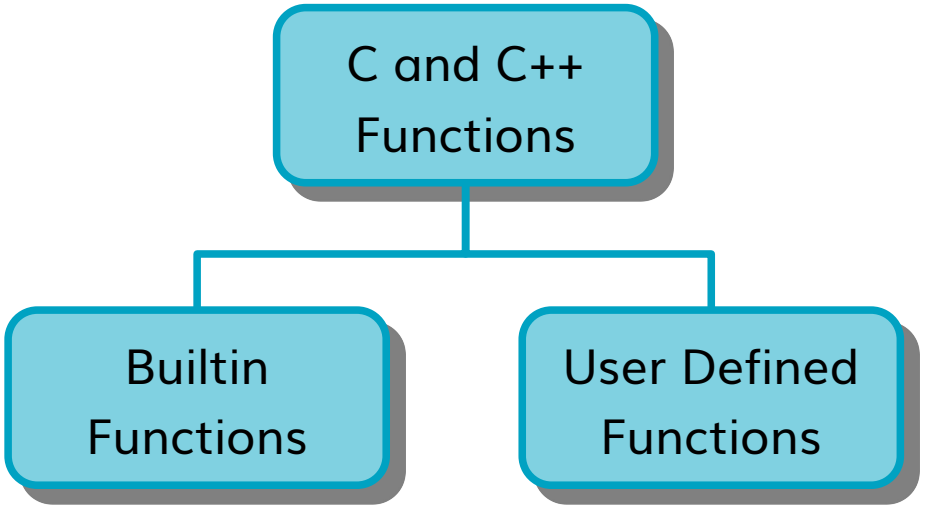


Image: Public Domain

- Reuse standard builtins where possible
  - e.g. CORE-V ALU extension reuses `__builtin_abs`
- Some operations don't work well as builtins
  - flow of control instructions
  - load/store with novel addressing modes

Lesson 8: Upstream Early & Often

Image: Josephluis at openclipart.org

- Out-of-tree is expensive
  - see Lesson 1
- Instead become part of the upstream code base
  - benefit from community interaction
- GCC and LLVM have long accepted vendor variants

Lesson 9: Use Upstream Conventions

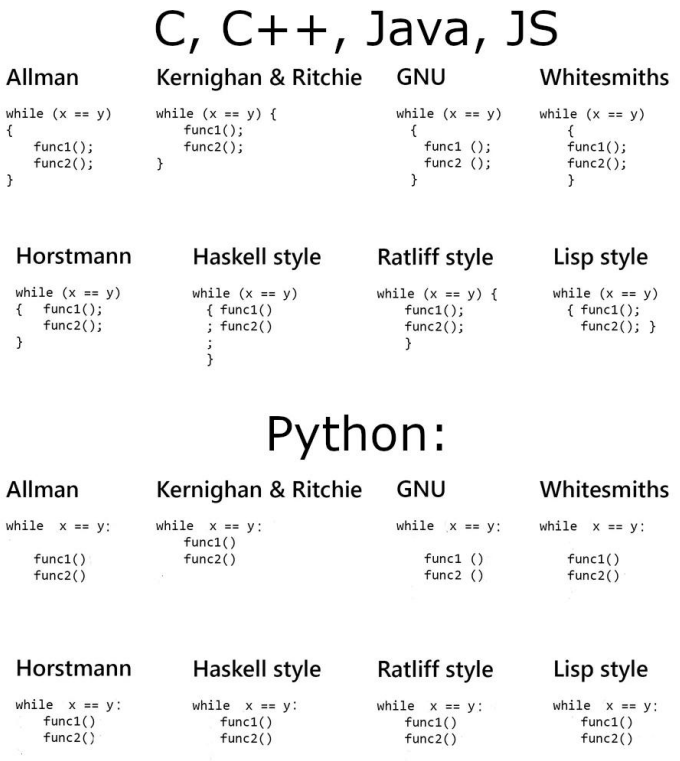


Image: u/trutheality at reddit.com

- Common coding styles make code easier for everyone
  - even if you have a better style
  - otherwise patches may be rejected
  - clang-format is your friend
- This is more than just code style
  - patch format conventions
  - submission and review process
- Use the same out-of-tree

Lesson 10: Use Proving Grounds

OPENHW®

Image: www.openhwgroup.org

- Upstream is the goal
  - tool-centric view of the world
- Initially a target centric view helps
  - gather consortia to work on the project
  - attract new engineers to tool development
  - move code upstream ASAP

Thank You

jeremy.bennett@embecosm.com

embecosm.com

openhwgroup.org