Towards Automated LLVM Support and Autovectorization for RISC-V ISA Extensions

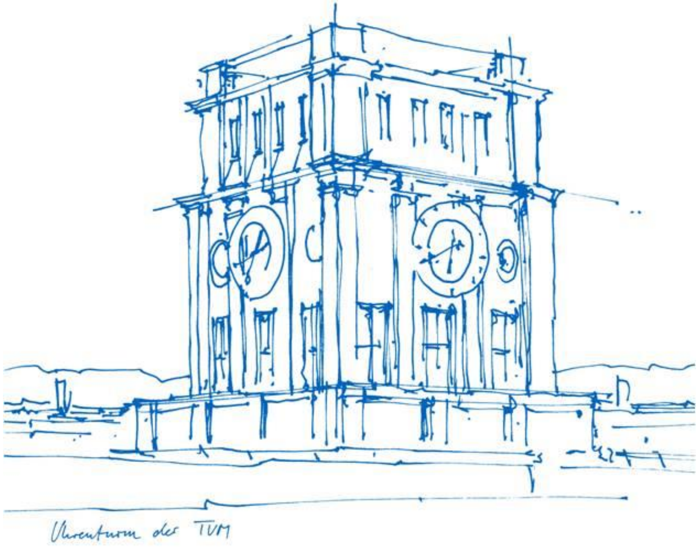
Philipp van Kempen, Mathis Salmen,

Daniel Mueller-Gritschneder, Ulf Schlichtmann

Technical University of Munich

TUM School of Computation, Information and Technology

Chair of Electronic Design Automation



Munich, 26th June 2024

This work has been developed in the project MANNHEIM-FlexKI. MANNHEIM-FlexKI is funded by the German Ministry of Education and Research (BMBF) (reference numbers: 01IS22086A-L). The authors are responsible for the content of this publication.

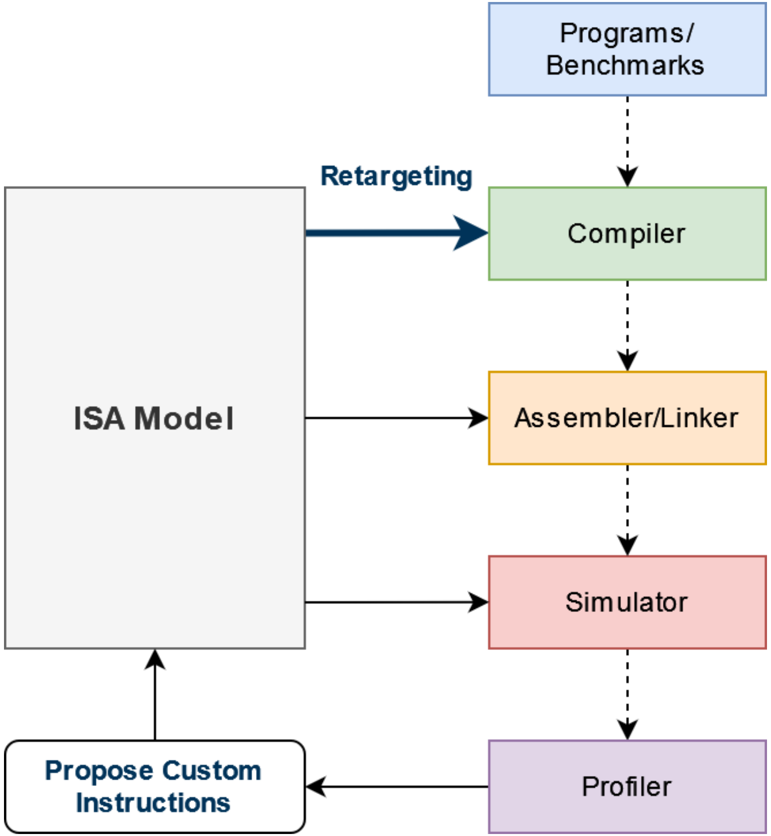
Motivation

ISA DSE

- Iterative approach
- Manual Efforts
  - Propose Instructions
  - Update Tools
  - Integration of Instructions

Here

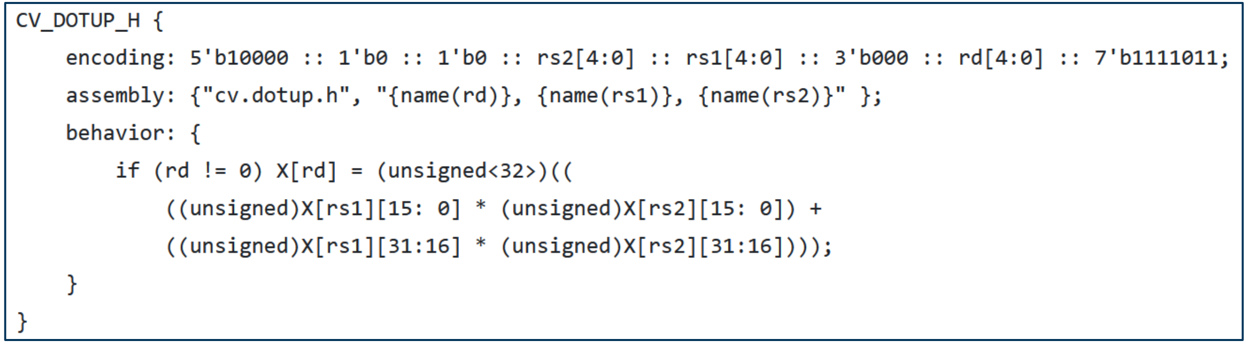
- Eliminate efforts for SW and Compiler Developers



RISC-V ISA Modeling

Contents

- Encoding
- Assembly Format
- Behavior/Semantics



```
CV_DOTUP_H {
    encoding: 5'b10000 :: 1'b0 :: 1'b0 :: rs2[4:0] :: rs1[4:0] :: 3'b000 :: rd[4:0] :: 7'b1111011;
    assembly: {"cv.dotup.h", "{name(rd)}, {name(rs1)}, {name(rs2)}" };
    behavior: {
        if (rd != 0) X[rd] = (unsigned<32>)((
            ((unsigned)X[rs1][15: 0] * (unsigned)X[rs2][15: 0]) +
            ((unsigned)X[rs1][31:16] * (unsigned)X[rs2][31:16])));
    }
}
```

Examples

- Proprietary
  - [CodAL] (Codasip)
  - [nML] (Synopsys)

- Open
  - [SAIL]: Official golden reference model
  - [CoreDSL]: Developed in Scale4Edge (S4E) project, maintained by MINRES Technologies

→ **Used here**

Retargeting SW Compilers

Related work

- Commercial
  - [Codasip] Re-targetable LLVM C/C++ compiler for RISC-V
  - [Synopsys] ASIP Designer: Optimizing C/C++ compiler

Black boxes!

- Academic
  - [TUDA] Automatic Compiler Support for Application-Specific Instruction Set Architecture Extensions
  - [DLR] Extensible Compiler (Scale4Edge)
  - [TUNI] OpenASIP 2.0: Co-Design Toolset for RISC-V Application-Specific Instruction-Set Processors

Open source!

- Other
  - [CGEN] architecture code generation used by binutils

Outdated!

Introducing Seal5

Seal5 - Semi-automated LLVM Support for RISC-V ISA Extensions (Including Autovectorization)

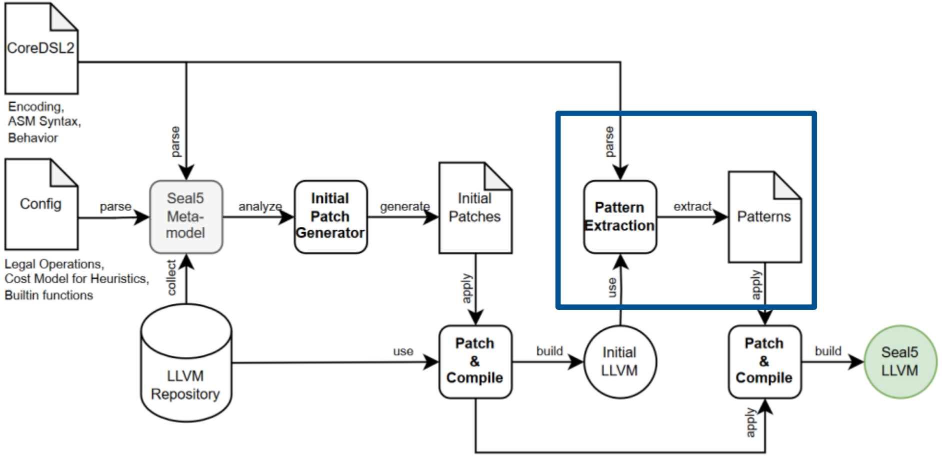
Inputs

- [CoreDSL] code for custom instructions
- Optional: YAML Settings

Outputs

- Patched [LLVM] Toolchain

Flow



Retargeting Support Levels (LLVM)

Tool	**Assembler** (Encoding, Format, Effects,…)	**Intrinsics/Builtins** (LLVM-IR, C/C++)	**CodeGen** (ISel Patterns, Legalization)	**Auto-Vectorization** (SIMD, Heuristics,…)
Extensible Compiler [DLR]	✓ (Needs user inputs)	✓	✗	✗
Thesis [TUDA]	✓	✓	✗	✗
OpenASIP 2.0 [TUNI]	✓	✓	✗	✗
Seal5 (Ours)	✓	✓	✓ (Semi-automated)	✓ (Narrow 32-bit SIMD only)

Usage by SW Developer:

```
asm("mac x3, x4, x5");
```

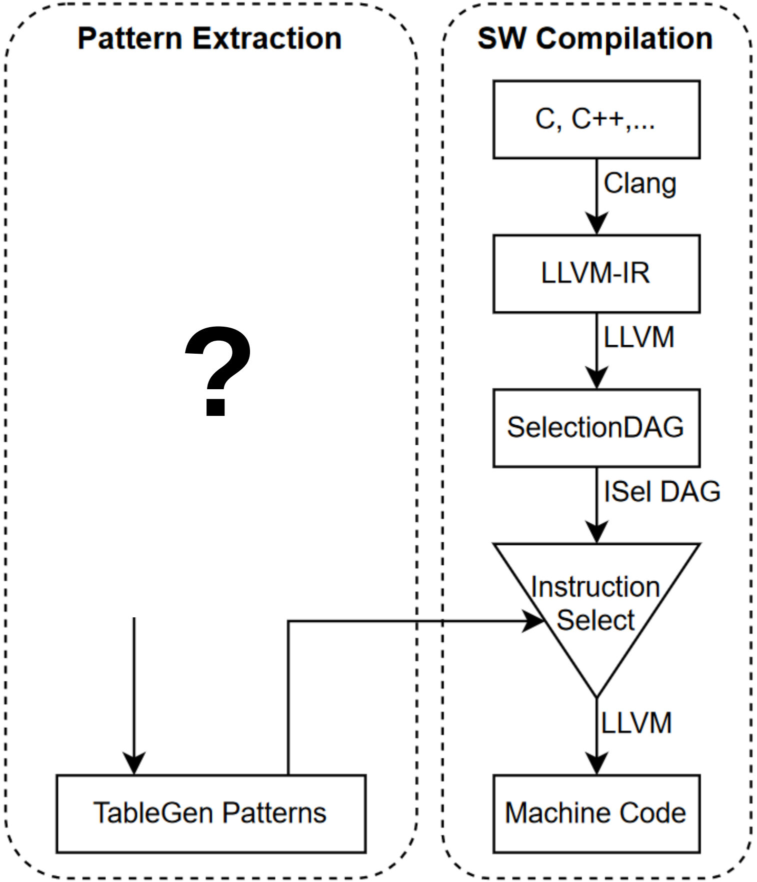
```
__builtin_mac(acc, x, y);
```

```
acc += x * y;
```

```
for (i = 0; i < n; i++) {
  acc += arr_x[i] * arr_y[i];
}
```

Why do we need patterns?

- During compilation the original program is lowered to intermediate representations (IRs) in a step-by-step fashion
- Optimizations are applied along the way
- During *Instruction Selection* Generic LLVM instructions are converted to target-specific *MachineInstructions*
- Instruction Selection depends on manually specified patterns to insert any instructions.



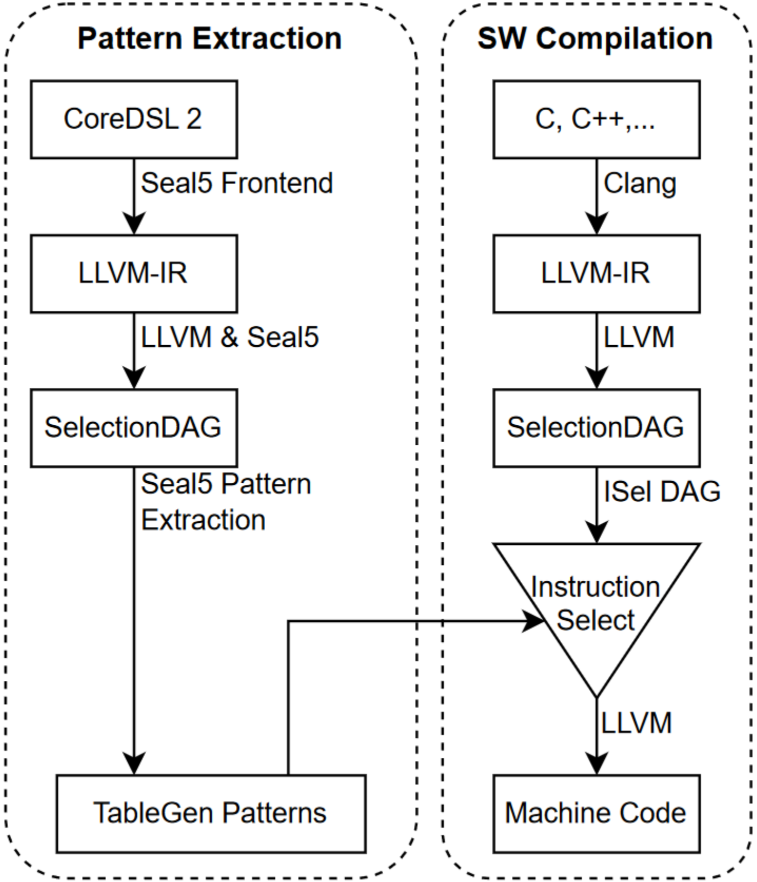
Generating ISel Patterns

Method

1. Convert CoreDSL behavior to LLVM-IR functions
2. Perform lowering in a similar way to target SW
3. Add hook to emit final DAG right before Instruction Selection would take place
4. Transform DAG nodes to TableGen code for patterns

Advantages

- Re-use existing code in LLVM
- Same optimizations → increased likeliness that extracted patterns will actually match
- SIMD-instructions are detected automatically



Seal5 Evaluation (Core-V)

Core-V Extension (OpenHW Group)

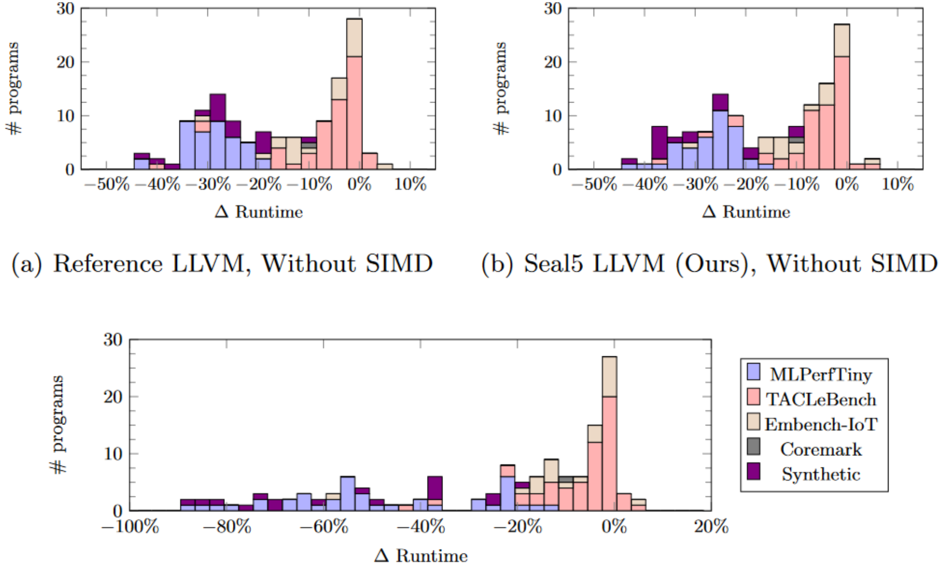
- 300+ ALU/Mem/SIMD/… instructions
- Implemented in [CV32E40P]

Configurations

1. Baseline (RV32IM)
2. Core-V Reference
3. Seal5 Generated
    a) Without SIMD
    b) With SIMD

Benchmarks

- 100+ embedded programs



(a) Reference LLVM, Without SIMD

(b) Seal5 LLVM (Ours), Without SIMD

(c) Seal5 LLVM (Ours), With SIMD

Seal5 Evaluation (SHA256)

Question: How does the Seal5 approach generalize for non Core-V instructions?

- RISC-V Bitmanipulation Extensions [Zbb] include 32-bit Rotations

- RISC-V Scalar Cryptography Extensions [Zk] includes custom SHA256 operations

→ Both can be supported effortlessly with Seal5.

→ Resulting speedup matches previous observations by [TUNI].

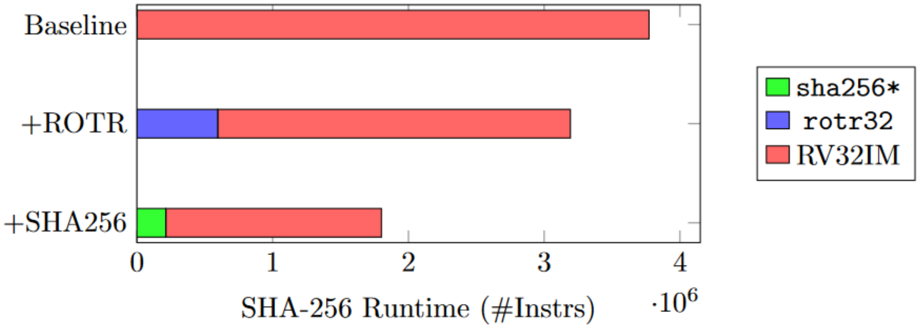
$$\mathrm{rotr32}^n(x) := (x >> n) \mid (x << (32 - n))$$

$$\mathrm{sha256sig}_0(in) := \mathrm{rotr}^7(in) \oplus \mathrm{rotr}^{18}(in) \oplus (in \gg 3)$$

$$\mathrm{sha256sig}_1(in) := \mathrm{rotr}^{17}(in) \oplus \mathrm{rotr}^{19}(in) \oplus (in \gg 10)$$

$$\mathrm{sha256sum}_0(in) := \mathrm{rotr}^2(in) \oplus \mathrm{rotr}^{13}(in) \oplus \mathrm{rotl}^{22}(in)$$

$$\mathrm{sha256sum}_1(in) := \mathrm{rotr}^6(in) \oplus \mathrm{rotr}^{11}(in) \oplus \mathrm{rotl}^{25}(in)$$



Seal5 Roadmap

Recent additions

- Migration to GlobalISel
- Support for compressed instructions
- 64-bit Targets
- Custom registers

Work in Progress

- Test-case generation
- Register-pairs

→ Generate LLVM support for RISC-V Packed Extension

Planned features

- Floating point
- Custom bit widths: 2/4/6 bits
- CSR instructions
- Hardware Loops
- uArch-aware Scheduling

Conclusion

Summary

- Exploration of custom ISA extensions is highly complex
- Retargeting is essential to eliminate manual efforts

Seal5 – Retargeting LLVM Compiler for RISC-V

- Novel approach for robust pattern generation and SIMD support
- Compared with reference Core-V vendor toolchain
- Usability demonstrated with SHA256 custom instructions

Seal5 Repository:
https://github.com/tum-ei-eda/seal5

Contributions are welcome!

References

[CodAL] Website: https://codasip.com/2021/02/26/what-is-codal/

[CoreDSL] Repo: https://github.com/Minres/CoreDSL

[SAIL] Repo: https://github.com/riscv/sail-riscv

[Codasip] Website: Re-targetable LLVM C/C++ compiler for RISC-V - https://codasip.com/2023/07/25/re-targetable-llvm-c-c-plus-plus-compiler-for-riscv/

[nML,Synopsys] Website: ASIP Designer: Optimizing C/C++ compiler - https://www.synopsys.com/dw/ipdir.php?ds=asip-designer

[TUDA] Thesis: Halkenhäuser, M. *Automatic Compiler Support for Application-Specific Instruction Set Architecture Extensions* (Master's thesis, Technische Universität).

[DLR] Paper: Schlamelcher, J., & Grüttner, K. (2022). A DSL based approach for supporting custom RISC-V instruction extensions in LLVM.
Repo: https://github.com/DLR-SE/extensible-compiler

[TUNI] Paper: Hepola, K., Multanen, J., & Jääskeläinen, P. (2022, July). OpenASIP 2.0: co-design toolset for RISC-V application-specific instruction-set processors. In *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)* (pp. 161-165). IEEE.
Repo: https://github.com/cpc/openasip

CGEN] Repo: architecture code generation used by binutils - https://github.com/stffrdhrn/cgen

[CV32E40P] Website: OpenHW Group CV32E40P User Manual - https://cv32e40p.readthedocs.io/en/latest
Repo: https://github.com/tum-ei-eda/etiss

[Zbb] Repo: Working draft of the proposed RISC-V Bitmanipulation extension - https://github.com/riscv/riscv-bitmanip

[Zk] Repo: Scalar Cryptography v1.0.1 - https://github.com/riscv/riscv-crypto

Backup slides

Seal5 First Steps

Installation (inside virtualenv)

```
pip install seal5
```

Demos/Example

```
python3 examples/demo.py
```

Documentation

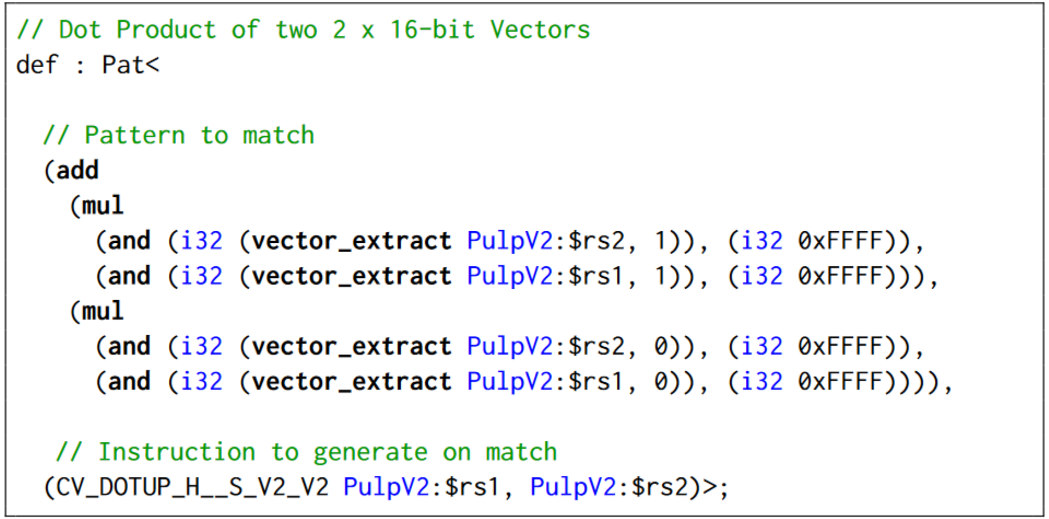
seal5.readthedocs.io

Issue Tracker

GitHub

Generated Patterns

SIMD 16-bit DOTP:

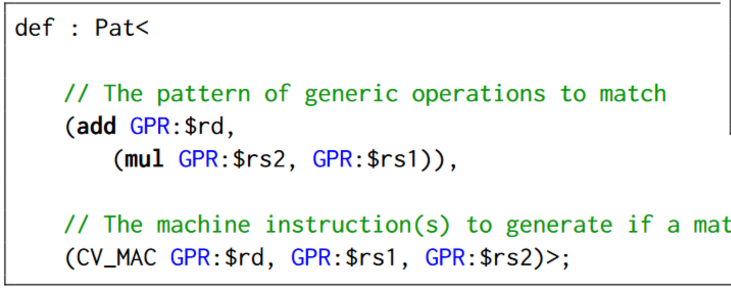


```
// Dot Product of two 2 x 16-bit Vectors
def : Pat<

  // Pattern to match
  (add
    (mul
      (and (i32 (vector_extract PulpV2:$rs2, 1)), (i32 0xFFFF)),
      (and (i32 (vector_extract PulpV2:$rs1, 1)), (i32 0xFFFF))),
    (mul
      (and (i32 (vector_extract PulpV2:$rs2, 0)), (i32 0xFFFF)),
      (and (i32 (vector_extract PulpV2:$rs1, 0)), (i32 0xFFFF)))),

   // Instruction to generate on match
  (CV_DOTUP_H__S_V2_V2 PulpV2:$rs1, PulpV2:$rs2)>;
```

Scalar 32-bit MAC:



```
def : Pat<

    // The pattern of generic operations to match
    (add GPR:$rd,
        (mul GPR:$rs2, GPR:$rs1)),

    // The machine instruction(s) to generate if a match is found
    (CV_MAC GPR:$rd, GPR:$rs1, GPR:$rs2)>;
```

Seal5 Usage

Python API

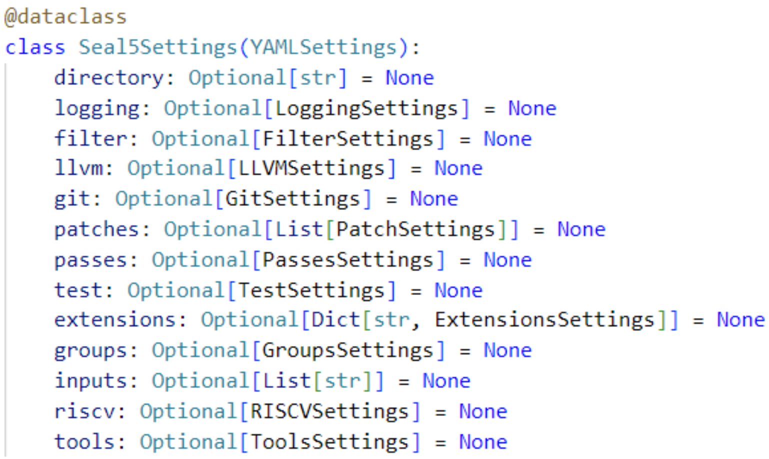
```
seal5_flow = Seal5Flow(“llvm-project”) → seal5_flow.initialize(...) → seal5_flow.setup(...)
           → seal5_flow.load([“*.core_desc”, ...]) → seal5_flow.transform(...)
           → seal5_flow.generate(...) → seal5_flow.patch(...) → seal5_flow.build(...)
           → seal5_flow.test(...) → seal5_flow.deploy(...) → seal5_flow.cleanup(...)
```

Command Line *(WIP)*

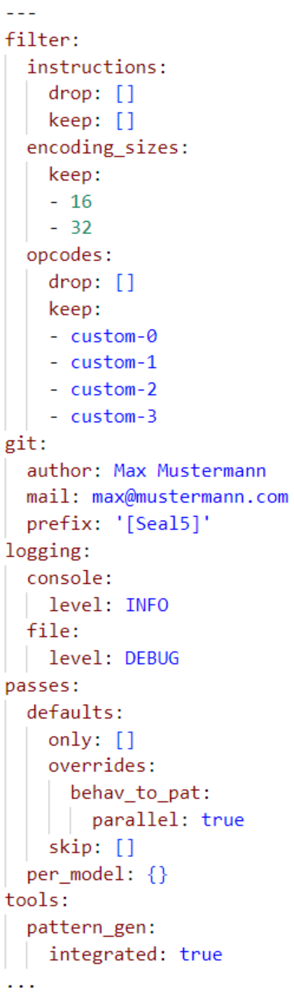
```
seal5 init llvm-project/
           → seal5 setup ... → seal5 load *.core_desc *.yml *.test.c  → seal5 transform ...
           → seal5 generate ... → seal5 patch ... → seal5 build ...
           → seal5 test ... → seal5 deploy ... → seal5 cleanup ...
```

Seal5 Configuration

Based on YAML and Python Dataclasses



```python
@dataclass
class Seal5Settings(YAMLSettings):
    directory: Optional[str] = None
    logging: Optional[LoggingSettings] = None
    filter: Optional[FilterSettings] = None
    llvm: Optional[LLVMSettings] = None
    git: Optional[GitSettings] = None
    patches: Optional[List[PatchSettings]] = None
    passes: Optional[PassesSettings] = None
    test: Optional[TestSettings] = None
    extensions: Optional[Dict[str, ExtensionsSettings]] = None
    groups: Optional[GroupsSettings] = None
    inputs: Optional[List[str]] = None
    riscv: Optional[RISCVSettings] = None
    tools: Optional[ToolsSettings] = None
```



```yaml
---
filter:
  instructions:
    drop: []
    keep: []
  encoding_sizes:
    keep:
    - 16
    - 32
  opcodes:
    drop: []
    keep:
    - custom-0
    - custom-1
    - custom-2
    - custom-3
git:
  author: Max Mustermann
  mail: max@mustermann.com
  prefix: '[Seal5]'
logging:
  console:
    level: INFO
  file:
    level: DEBUG
passes:
  defaults:
    only: []
    overrides:
      behav_to_pat:
        parallel: true
    skip: []
  per_model: {}
tools:
  pattern_gen:
    integrated: true
...
```

Seal5 Metamodel

- Based on [M2-ISA-R]
  - CoreDSL Parser
  - Provides Python-based framework for traversing behavioral descriptions
  - Components:
    - Arch: Sets, Architectural State, Encoding
    - Behav: Semantics of instructions/functions/…
- Extended with LLVM-specific information
  - Intrinsics/Builtins
  - Heuristics/Costs
  - Legalization Rules
- Added CoreDSL2 backend to export annotated and optimized Instructions