Seal5 Development & User Meeting

08.08.2024

Agenda (08.08.2024)

- Sync on ongoing work
  - CI/CD Integration (DLR)
  - Intrinsic/Builtins Support (DLR)
  - Support for Load/Store/Branch (TUM)
- Next meeting dates
  - ***Internal:*** *15.08.2024, 11:00 (Skype)*
  - **Public:** 12.09.2024, 11:00 (Zoom)

CI/CD Integration

- Related Repositories:
  - Private, will be made public when finished
- Status:
  - ETISS Workflow (Without M2-ISA-R) works
  - Seal5 Workflow works (Takes 1 hour)
  - Working in Target SW & Testing Workflow
- Tipps:
  - Cache GCC Download in GitHub Actions Cache, See:
    - https://github.com/tum-ei-eda/muriscv-nn/blob/master/.github/workflows/download_dependencies.yml
    - https://github.com/tum-ei-eda/muriscv-nn/blob/6eb3f32dd6f7ad1f2f1246f6379deeb73ac98ff8/.github/workflows/build_spike.yml#L21

Intrinsics/Builtins Support

- PoC
  - Reuse YAML syntax for configuration
  - Consider using CoreDSL in the future
    - Custom Syntax? → Maybe if Aliases/Pseudoinstructions (i.e. `mv rd, rs` → `addi rd, rs, 0`) can be described in CDSL
    - For simple intrinsics: using instruction/operand attributes might work
  - Not stored in Metamodel
  - Python implementation of Writer
  - 3 Patches per intrinsic
  - Cleanup (less duplication)
  - Testing with single simple intrinsic (32-bit only, no vectors)
  - Complex mappings (i.e. Core-V Shuffle) will not be covered

  → Open PR within next few weeks?

Supporting Loads/Stores/Branches

- Updates:
  - Loads/Stores
    - Problem: No way to define mem patterns in TableGen for GlobalISel due to missing mapping of relevant ISelDAG nodes to GMIR.
    - Solution: Would need support for GMIR Patterns in Tablegen (which does not exist yet, as existing ISelDAG patterns are being imported, which does only work if there is a valid mapping)
    - Workaround: Generate C++ code for Matching/Selecting these instructions…
  - Branches:
    - Problem: Branches are a conditional PC update in CDSL, which can not be represented in LLVM-IR which is generated by the CDSL2LLVM Frontend (single basic-block allowed)
    - Workaround (easy): Detect various types of Branches (without prefix) in Python and generate the Pattern (Would need to be hardcoded for detecting CondCode & Operands (REG, IMM,…))
    - Workaround (better): Detect branches in Python and translate them into a function call which could be processed by PatternGen to generate valid pattern including prefix instructions…