Guaranteeing Correctness of LLVM RISC-V Machine Code with Fuzzing

Jocelyn Wei – University of California, San Diego
Mandeep Singh Grang, Ana Pazos – Qualcomm Innovation Center, Inc.

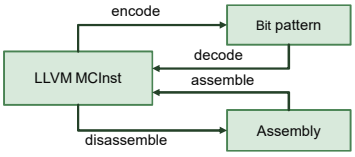
The Machine Code (MC) Layer is the foundation of LLVM. Several tools targeting various architectures are built upon it. This project seeks to validate the RISC-V MC Layer by using state-of-the-art fuzzing technology.

1. RISC-V and LLVM MC Layer

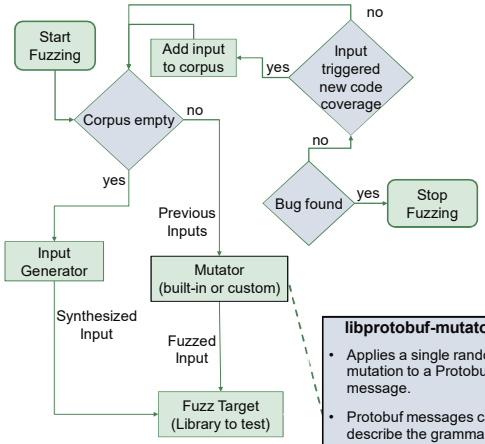
- Free and open ISA and chip design
- 32-bit, 64-bit, and 128-bit variants
- Modular specification

Standard General-Purpose ISA	
Integer	I
Integer Multiplication and Division	M
Atomics	A
Single-Precision Floating-Point	F
Double-Precision Floating-Point	D
General	G = IMAFD
Standard User-Level Extensions	
16-bit Compressed Instructions	C
…	

- Need to validate MC layer functionality with individual RISC-V extensions and extension combos.



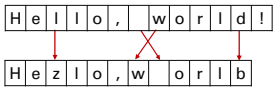
2. LLVM Fuzzing Support



libprotobuf-mutator

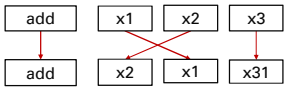
- Applies a single random mutation to a Protobuf message.
- Protobuf messages can describe the grammar for a language. E.g.: this has been done for a subset of the C++ language in clang-proto-fuzzer.

Generic Random Mutator treats input as array of bits or characters



Tools Examples:
clang-fuzzer, llvm-as-fuzzer, llvm-mc-assemble-fuzzer, llvm-mc-disassemble-fuzzer

Structured Mutator applies mutations to structured data



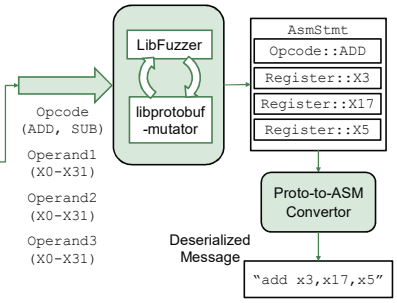
Tools Examples:
clang-proto-fuzzer, llvm-isel-fuzzer, llvm-opt-fuzzer, llvm-mc-assemble-proto-fuzzer, llvm-mc-disassemble-proto-fuzzer

3. RISC-V ASM and Encoding Protobuf Types and Convertors

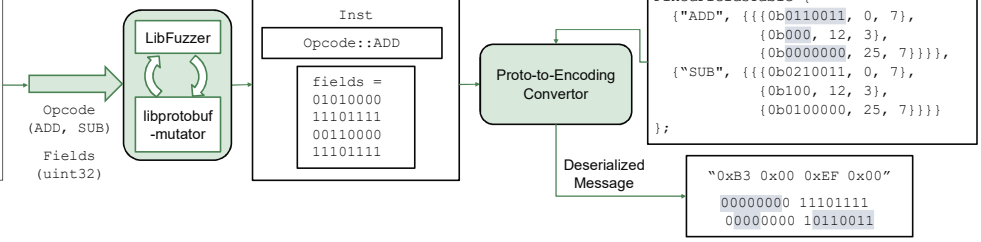
```
message Register {
  enum ValueRange {
    X0 = 1; X1 = 2; X2 = 3; ...
  };
  required ValueRange value = 1;
}
message Immediate {
  oneof imm_oneof {
    sint32 signed_imm   = 1;
    uint32 unsigned_imm = 2;
  }
}
message OpcodeType1 {
  enum ValueRange {
    ADD = 0; SUB = 1; ...
  };
  required ValueRange value = 1;
}
```

```
message OpcodeType2 {
  enum ValueRange {
    LW = 0; SW = 1; ...
  };
  required ValueRange value = 1;
}
message OperandType1 {
  required Register operand1 = 1;
  required Register operand2 = 2;
  required Register operand3 = 3;
}
message OperandType2 {
  required Register  operand1 = 1;
  required Register  operand2 = 2;
  required Immediate operand3 = 3;
}
```

```
message StmtType1 {
  required OpcodeType1  opcode   = 1;
  required OperandType1 operands = 2;
}
message StmtType2 {
  required OpcodeType2  opcode   = 1;
  required OperandType2 operands = 2;
}
message AsmStmt {
  oneof asmstatement_oneof {
    StmtType1 statement = 1;
    StmtType2 statement = 2;
  }
}
message Assembly {
  repeated AsmStmt statements = 1;
}
```

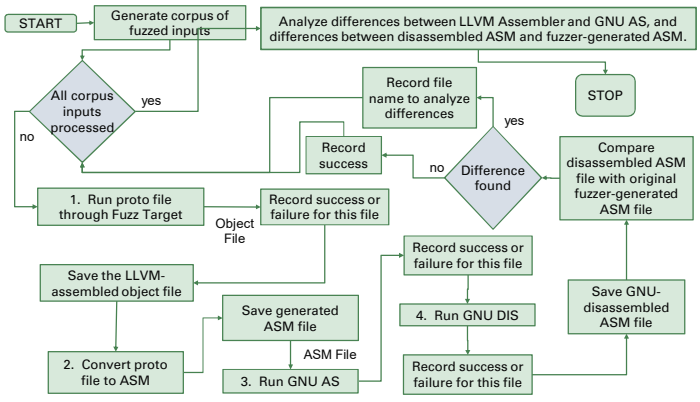



```
message Opcode {
  enum ValueRange {
    ADD = 0;
    SUB = 1;
  };
  required ValueRange value = 1;
}
message Inst {
  required Opcode opcode = 1;
  required uint32 fields = 2;
}
```




```
FixedFieldsTable {
  {"ADD", {{{0b0110011, 0, 7},
            {0b000, 12, 3},
            {0b0000000, 25, 7}}}},
  {"SUB", {{{0b0210011, 0, 7},
            {0b100, 12, 3},
            {0b0100000, 25, 7}}}}
};
```

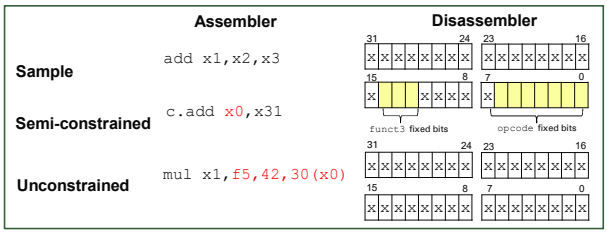
4. Driver for Testing Fuzz Targets (Assembler)



Driver for testing LLVM Disassembler has a similar work flow

LLVM AS	GNU AS	Analysis Possible Conclusions
Fail	Fail	Invalid or unimplemented instruction
Fail	Pass	LLVM MC bug or unimplemented instruction
Pass	Fail	LLVM MC bug
Pass	Pass	Success; run GNU DIS and compare output

5. Fuzzer Versions



6. Results

Area	Bugs
Assembler Parser (target independent)	1
RISC-V Assembler Parser	5
RISC-V Assembler Parser / Encoder (symbol refs)	7
RISC-V Disassembler / Decoder	4
GNU Assembler and Disassembler	4
LLVM MC compatibility issues with GNU	4