2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
MARCH 4-7, 2024

Unleashing the Power of Whisper for block-level verification in high performance RISC-V CPU

Chenhui Huang, Yu Sun, Joe Rahmeh

Tenstorrent Inc.

accellera SYSTEMS INITIATIVE

What is Whisper?

- Open-Source RISC-V Instruction Set Simulator (ISS)
- Initially developed by Western Digital for SweRV Core Project
- Support different RISC-V Ext.
- Serve as Golden reference model for RISC-V
- API Supports for debugging and analysis

We extend our gratitude to Joe Rahmeh for being one of the main contributors to the 'whisper' repository, a pivotal resource available on GitHub (https://github.com/chipsalliance/).

CPU Verification Methodology Comparison

Core-level Verification

Advantages:

- Streamlined Testbench Architecture
- Straightforward Architectural Verification
- Assembly Tests are reusable cross projects

Challenges:

- Complex Debugging Process
- Limited Stimulus Control
- Not Easy for new features bring-up
- More complex Micro-Arch Verification

VS

Block-level Verification

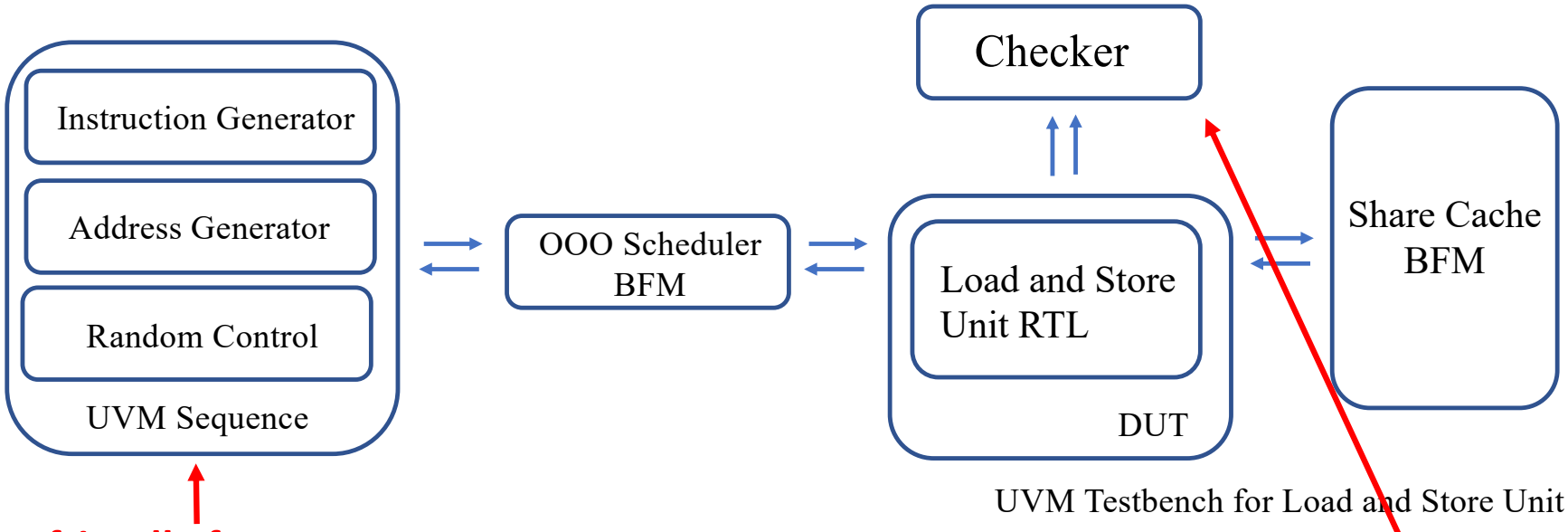
Advantages :

- Rapid Compilation Times
- Enhanced Test Control Precision
- Easier Debugging
- Straightforward to bring up new features

Challenges :

- Require more TB knowledges
- Intensive Bus Functional Model (BFM) Demands

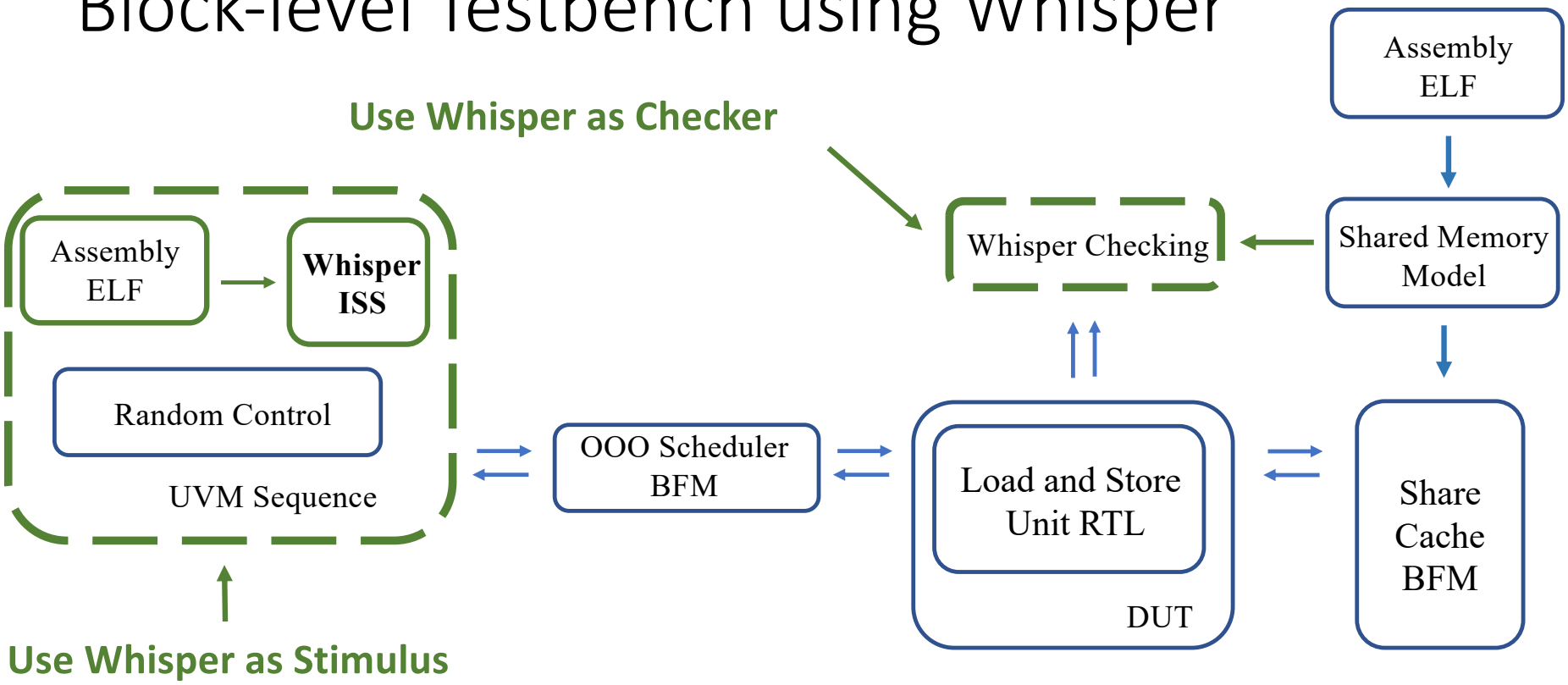
Traditional Block-level TB for Load Store Unit



Not friendly for RTL designer to bring up new features.

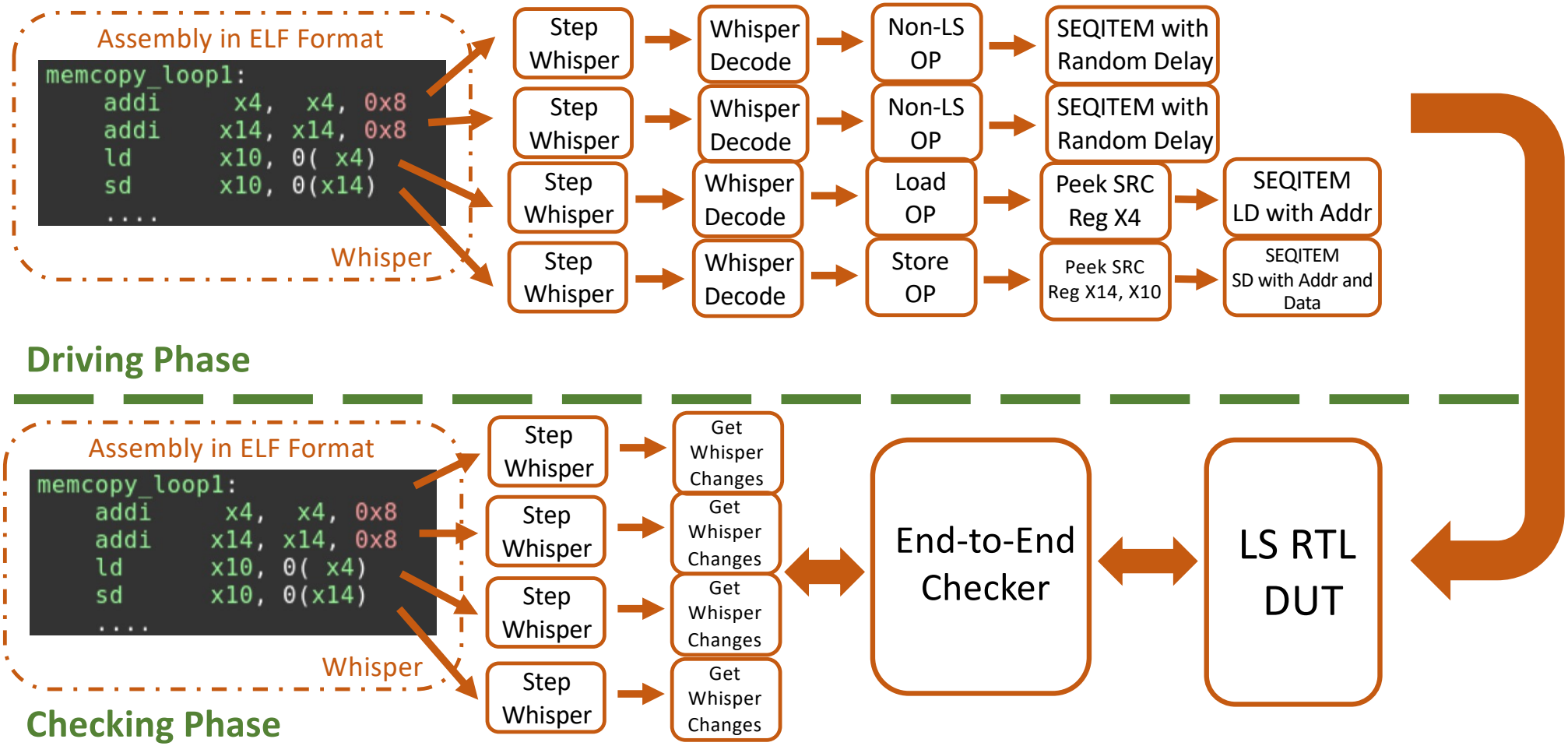
Complicated and High maintenance cost.

Block-level Testbench using Whisper



DPI Functions Whisper supports

```
import "DPI-C" function void stepWhisper(input int hart, input longint iTime, input longint instrTag
import "DPI-C" function void peekWhisperRegister(input int hart, byte resource, longint addr, output
import "DPI-C" function void peekWhisperVecRegister(input int hart, input longint addr, output byte
import "DPI-C" function void pokeWhisperRegister(input int hart, byte resource, longint addr, value,
import "DPI-C" function void pokeWhisperVecRegister(input int hart, input longint addr, input byte va
import "DPI-C" function void getWhisperChange(input int hart, output int resource, output longint ad
import "DPI-C" function void doWhisperReset(input int hart, output bit valid);
......
```


Assembly in ELF Format
memcopy_loop1:
addi x4, x4, 0x8
addi x14, x14, 0x8
ld x10, 0(x4)
sd x10, 0(x14)
....
Whisper
Step Whisper
Step Whisper
Step Whisper
Step Whisper
Whisper Decode
Whisper Decode
Whisper Decode
Whisper Decode
Non-LS OP
Non-LS OP
Load OP
Store OP
SEQITEM with Random Delay
SEQITEM with Random Delay
Peek SRC Reg X4
Peek SRC Reg X14, X10
SEQITEM LD with Addr
SEQITEM SD with Addr and Data
Driving Phase
Assembly in ELF Format
memcopy_loop1:
addi x4, x4, 0x8
addi x14, x14, 0x8
ld x10, 0(x4)
sd x10, 0(x14)
....
Whisper
Step Whisper
Step Whisper
Step Whisper
Step Whisper
Get Whisper Changes
Get Whisper Changes
Get Whisper Changes
Get Whisper Changes
End-to-End Checker
LS RTL DUT
Checking Phase

Why doing that?

Whisper can be the missing parts of your unit testbench

- Decoding the Instructions
- Data Value calculation
- Memory Model for checking
- Any architectural states

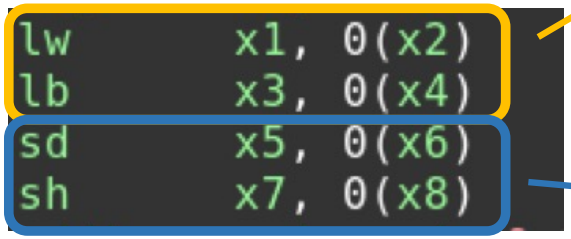
……

Whisper Memory Consistency Model (MCM)

- Whisper supports the Memory Consistency Modeling
- Checking 13 Preserved Program Order (PPO) rules for RISC-V Weak Memory Ordering (RVWMO)
- Can be adopted at the block-level testbench with multiple LS unit testbench architecture

- Overlapping-Address Orderings:
  1. b is a store, and a and b access overlapping memory addresses
  2. a and b are loads, x is a byte read by both a and b, there is no store to x between a and b in program order, and a and b return values for x written by different memory operations
  3. a is generated by an AMO or SC instruction, b is a load, and b returns a value written by a
- Explicit Synchronization
  4. There is a FENCE instruction that orders a before b
  5. a has an acquire annotation
  6. b has a release annotation
  7. a and b both have RCsc annotations
  8. a is paired with b
- Syntactic Dependencies
  9. b has a syntactic address dependency on a
  10. b has a syntactic data dependency on a
  11. b is a store, and b has a syntactic control dependency on a
- Pipeline Dependencies
  12. b is a load, and there exists some store m between a and b in program order such that m has an address or data dependency on a, and b returns a value written by m
  13. b is a store, and there exists some instruction m between a and b in program order such that m has an address dependency on a

How to check MCM in the testbench using Whisper

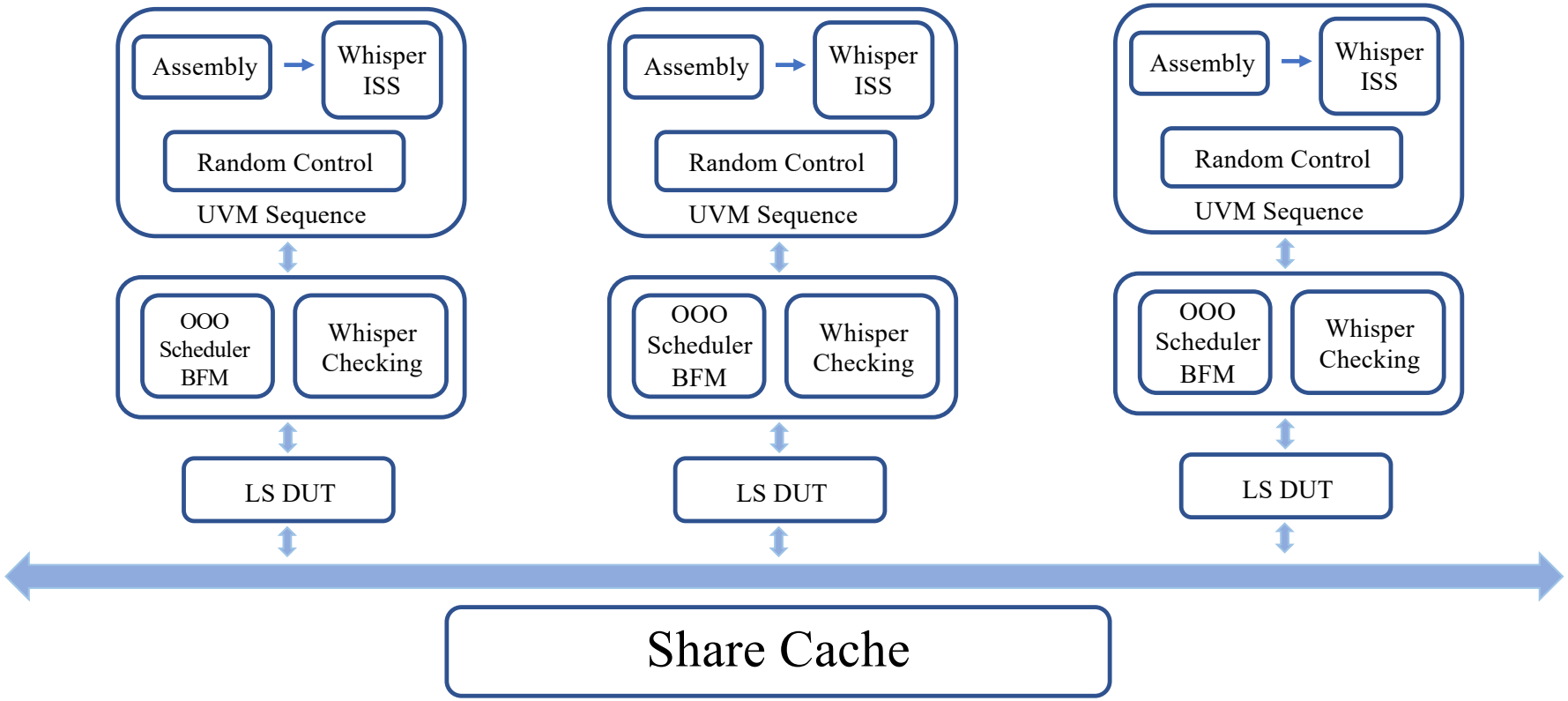


1: Time Stamp when the Load data is visible
2: Time Stamp when the Load is Retired

1: Time Stamp when the Store data is into Merge Buffer
2: Time Stamp when the Store data is drained out of Merge Buffer
3: Time Stamp when the Store is Retired
4: Time Stamp of bypass if the Store is reaching IO/uncacheable/AMO

```
import "DPI-C" function void getwhisperMcmRead(input int hart, longint mtime, longir
import "DPI-C" function bool retire(input int hart, longint mtime, longint instrTag)
import "DPI-C" function void getwhisperMcmWrite(input int hart, longint mtime,  long
import "DPI-C" function void getwhisperMcmInsert(input int hart, longint mtime, long
import "DPI-C" function void getwhisperMcmBypass(input int hart, longint mtime, long
```

Multiple LS units with Shared Cache



Whisper Python Interface

```
>>> import whisper
>>> whisper_ins = whisper.system.System64("whisper.json")
>>> whisper_ins.load_elf_files(["basic_ls.elf"])
True
>>> my_hart = whisper_ins.harts()[0]
>>> my_hart.step(1)
#1 0  M 0000000080000000 7ff0029b r 0000000000000005 00000000000007ff addiw    x5, x0, 2047
......
>>> my_hart.step(1)
#4 0  M 000000008000000c 30029073 c 0000000000000300 8000000a005fffaa csrrw    x0, mstatus, x5
>>> print(hex(my_hart.mstatus))
0x8000000a005fffaa
>>> my_hart.step(1)
#9 0  M 0000000080000020 020a5087 v 01 cafe0007cafe0006cafe0005cafe0004cafe0003cafe0002cafe0001cafe0000 vle16.v
v1, (x20) [0x80000024;0x80000026;0x80000028;0x8000002a;0x8000002c;0x8000002e;0x80000030;0x80000032;0x80000034;0x800
00036;0x80000038;0x8000003a;0x8000003c;0x8000003e;0x80000040;0x80000042]
>>> print(hex(my_hart.x2))
0x20
>>> print(my_hart.v1)
[202, 254, 0, 7, 202, 254, 0, 6, 202, 254, 0, 5, 202, 254, 0, 4, 202, 254, 0, 3, 202, 254, 0, 2, 202, 254, 0, 1,
    202, 254, 0, 0]
>>> my_hart.x2 = 0xcafe
```

Load Configuration

Load ELF File

Create object for Hart 0

Step Instruction

Peek CSR Value

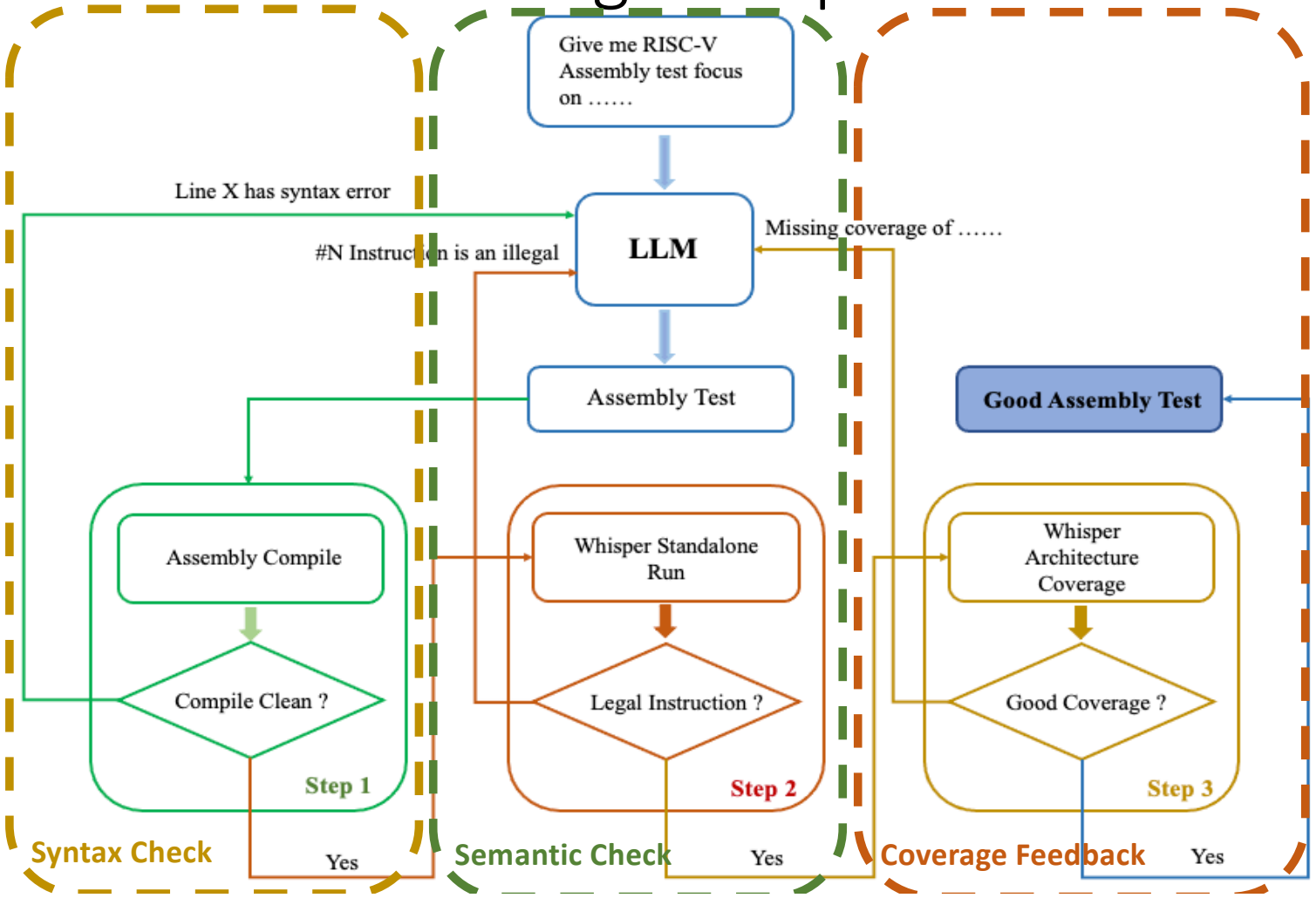
Peek Scalar Register Value

Peek Vector Register Value

Poke Register Value

We extend our gratitude to Jiahan Zhang for his support for Whisper Python Interface.

Test Generation using Whisper



Summary

Integration of Whisper in Block-Level Testbench

- Used as Stimulus:
  - Empowering RTL Designers with new capabilities
  - Shorten the development cycle during the features bring-up
- Used as Checker:
  - Inherently used for End-to-End checker
  - Memory consistency checking at block-level
- Applying Whisper with LLM for test generation:
  - Narrative Test Scenarios converted into Assembly test
  - Open-source or In-house LLM will be investigated in the future

Thank you for Listening

Questions