Verifying x86 Instruction Implementations

Shilpi Goel, Anna Slobodova, Rob Sumners, & Sol Swords
{shilpi,anna,rsumners,sswords}@centtech.com

Formal Verification Team @ Centaur Technology

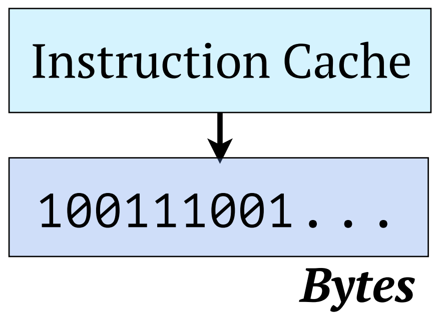
CPP 2020

Goal

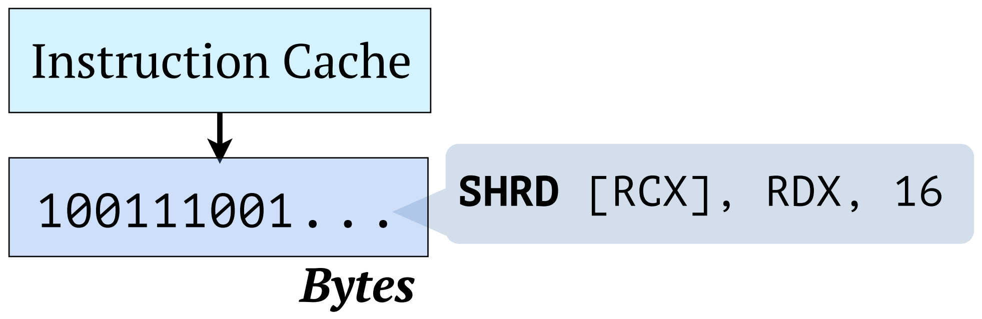
- **Broad Verification Objective:** prove that Centaur's processors correctly implement the x86 ISA.
  - Verification of all parts of the processor and microcode.

Goal

- **Broad Verification Objective:** prove that Centaur's processors correctly implement the x86 ISA.
  - Verification of all parts of the processor and microcode.
- **Focus of this Talk:** **methodology for proving that Centaur's processors execute an x86 instruction correctly.**
  - Instruction **decoding**.
  - **Translating** a legal instruction to corresponding micro-operations (***uops***).
  - **Executing** these uops.

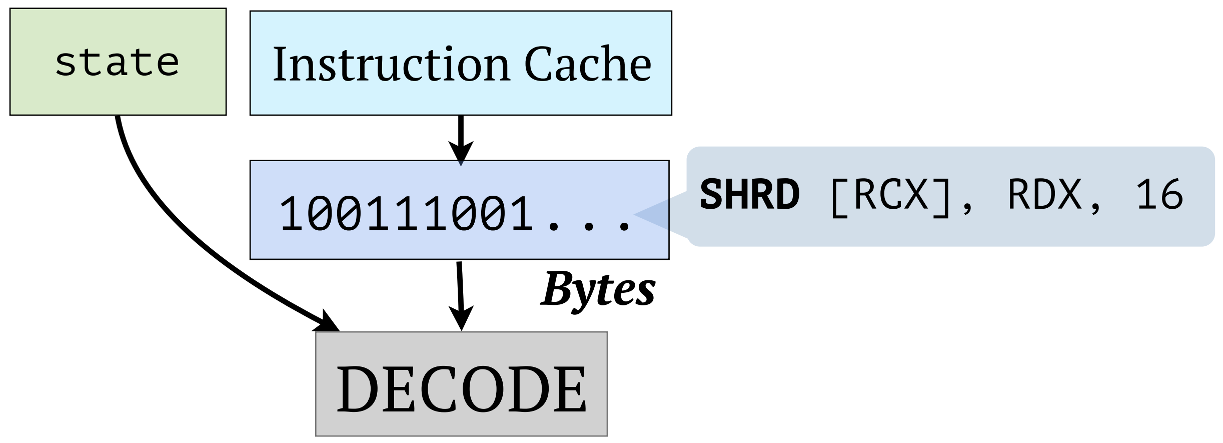


Execution of an x86 Instruction

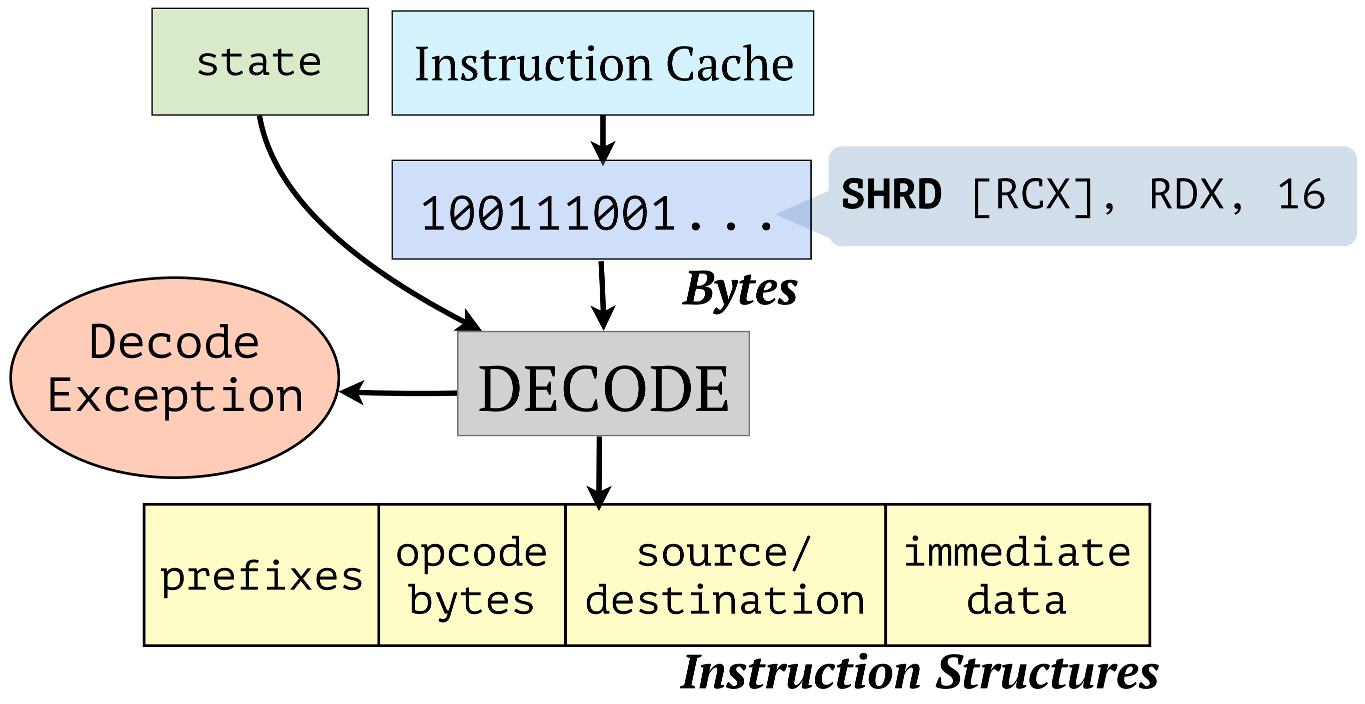


```
SHRD [RCX], RDX, 16
```

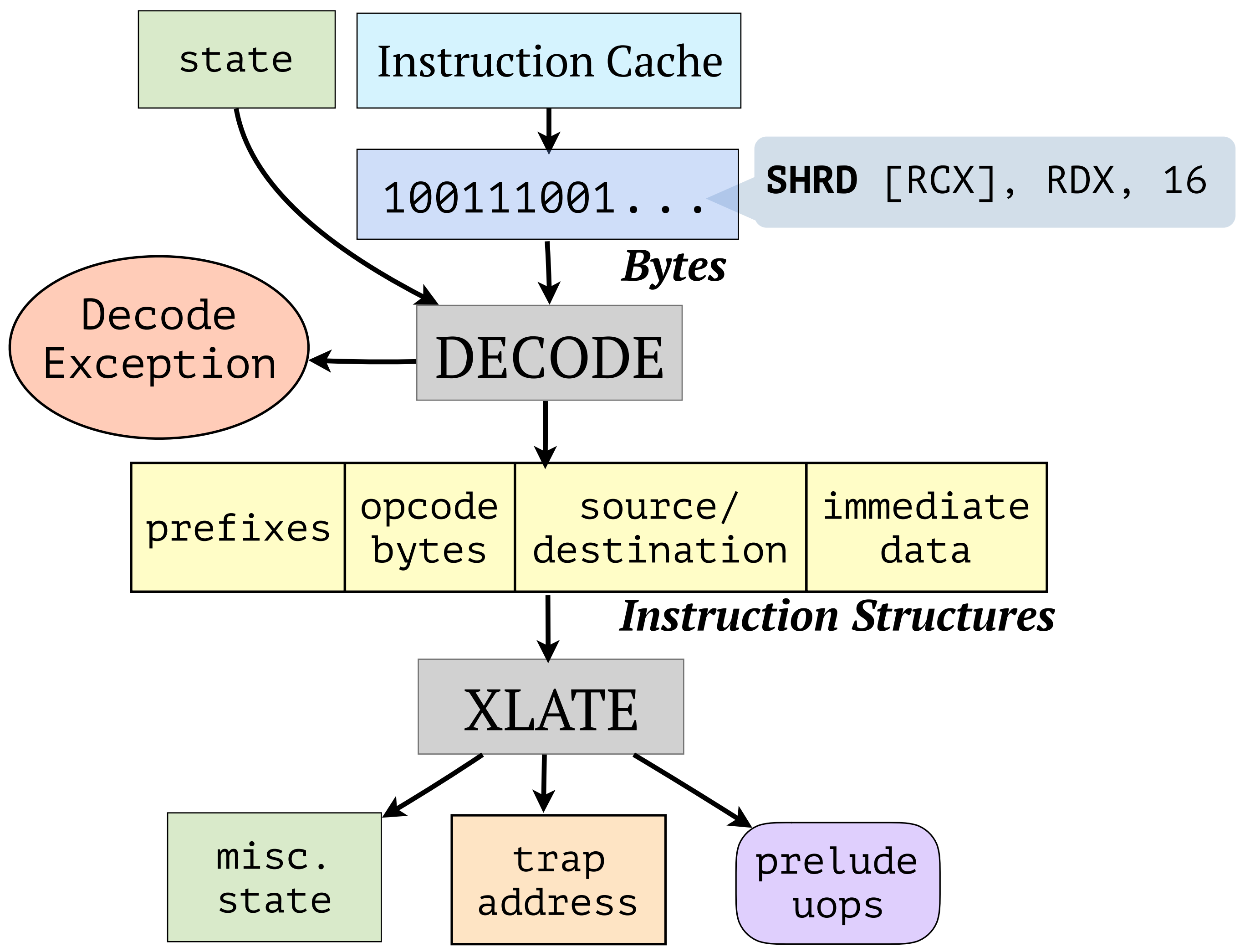
Execution of an x86 Instruction



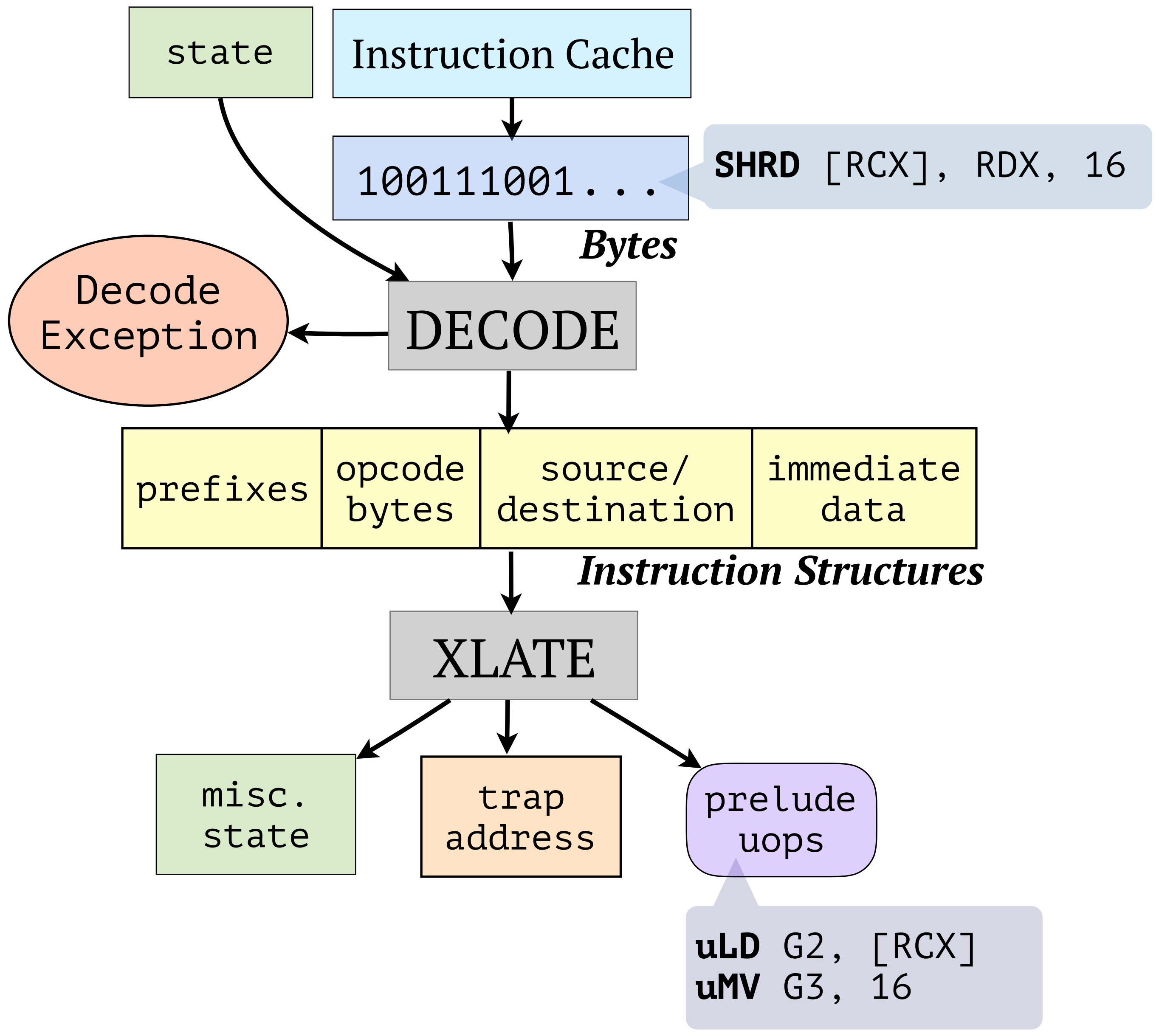
Execution of an x86 Instruction



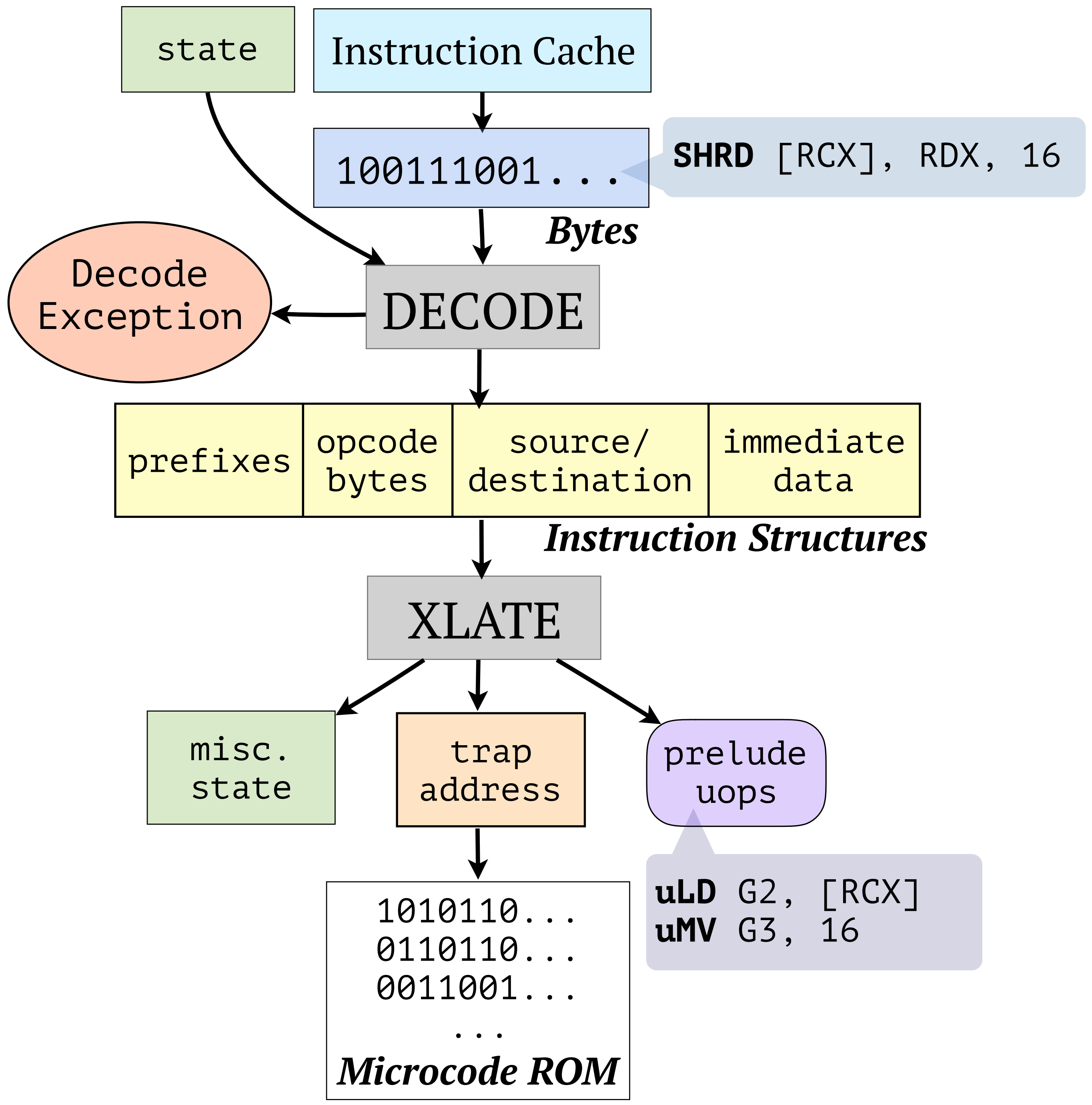
Execution of an x86 Instruction



Execution of an x86 Instruction

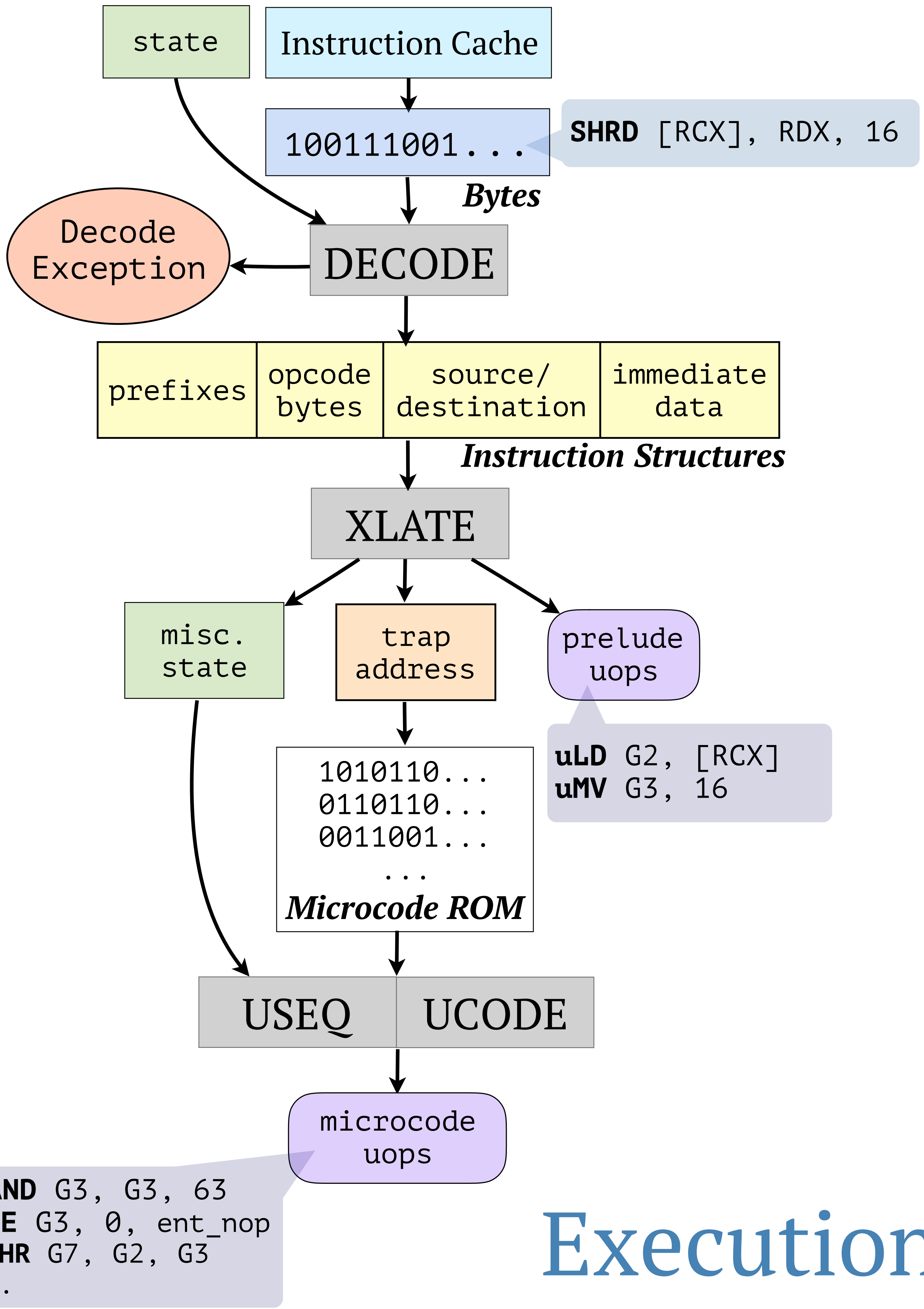


Execution of an x86 Instruction

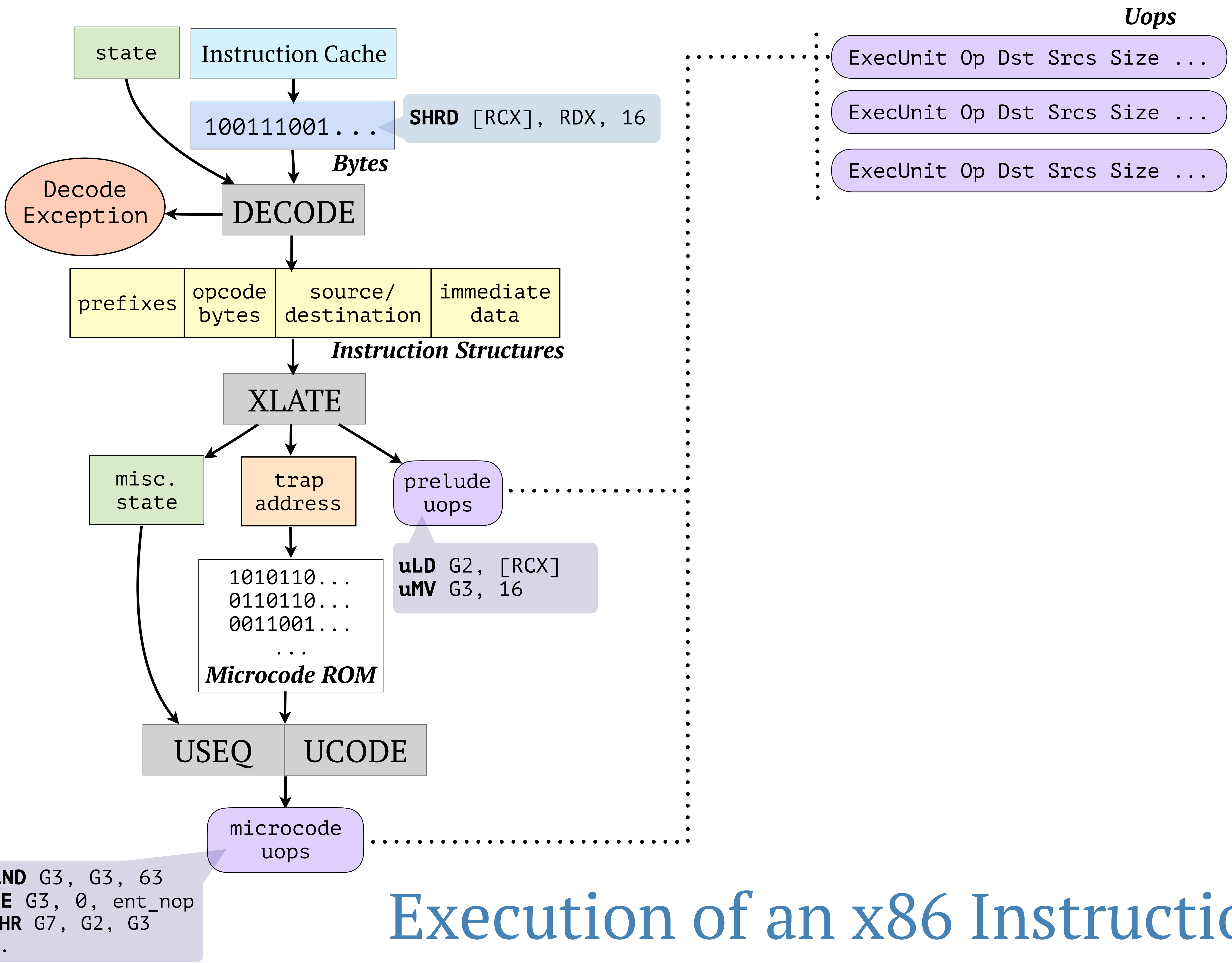


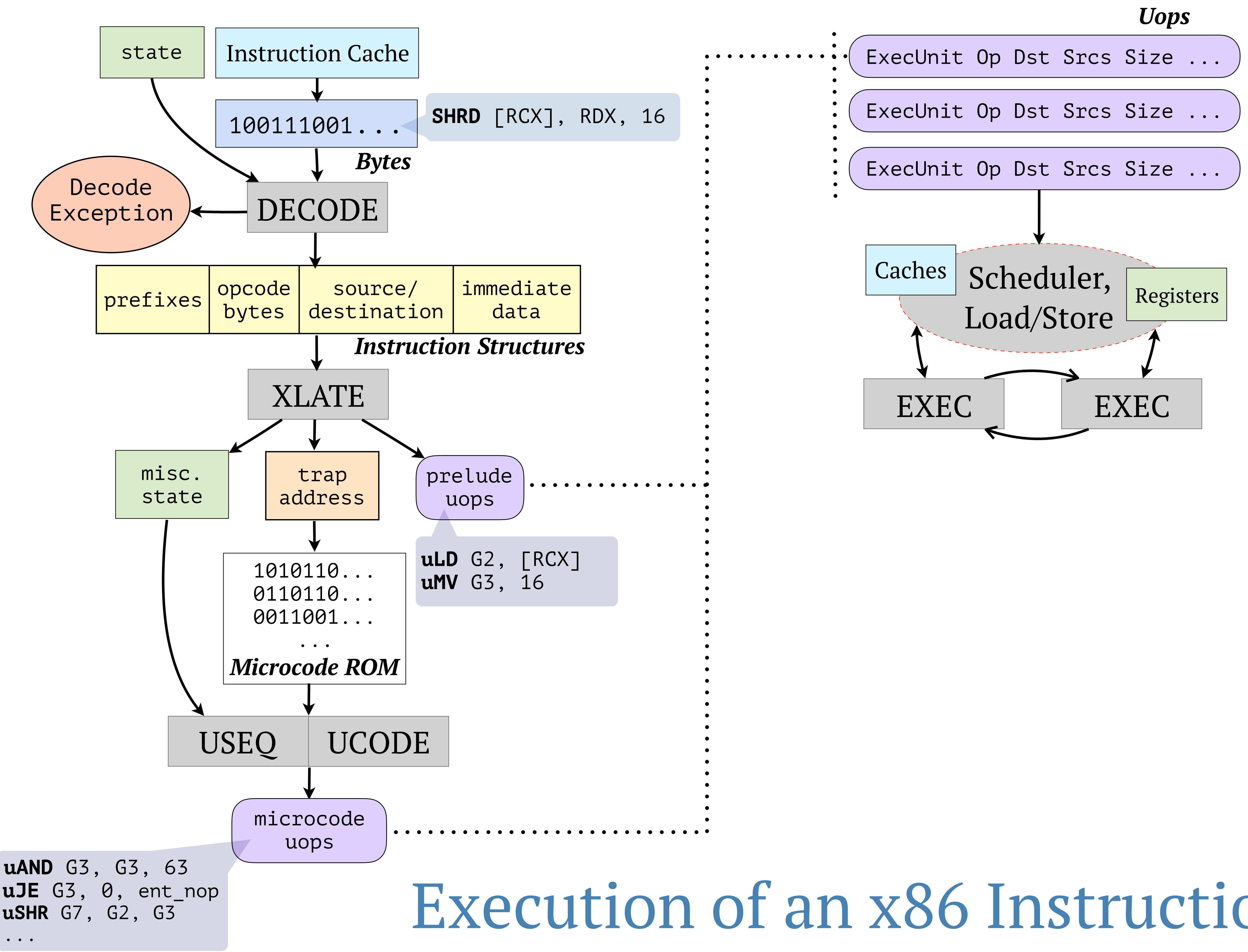
Execution of an x86 Instruction

Execution of an x86 Instruction



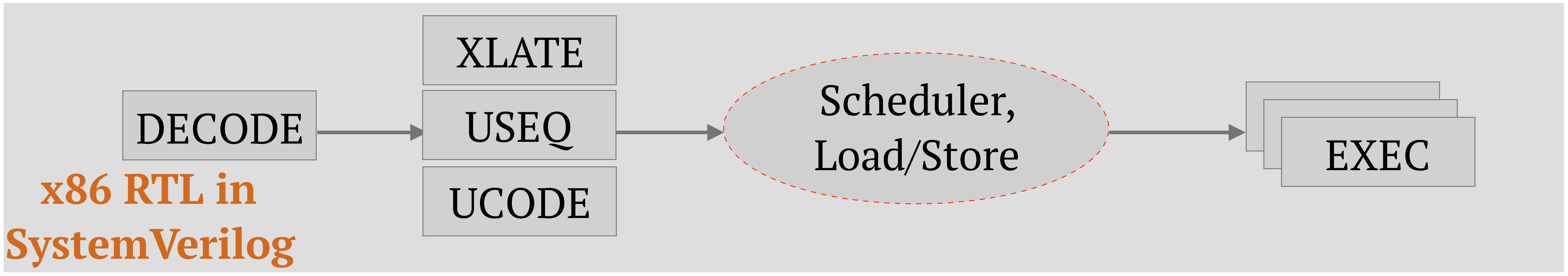
Execution of an x86 Instruction



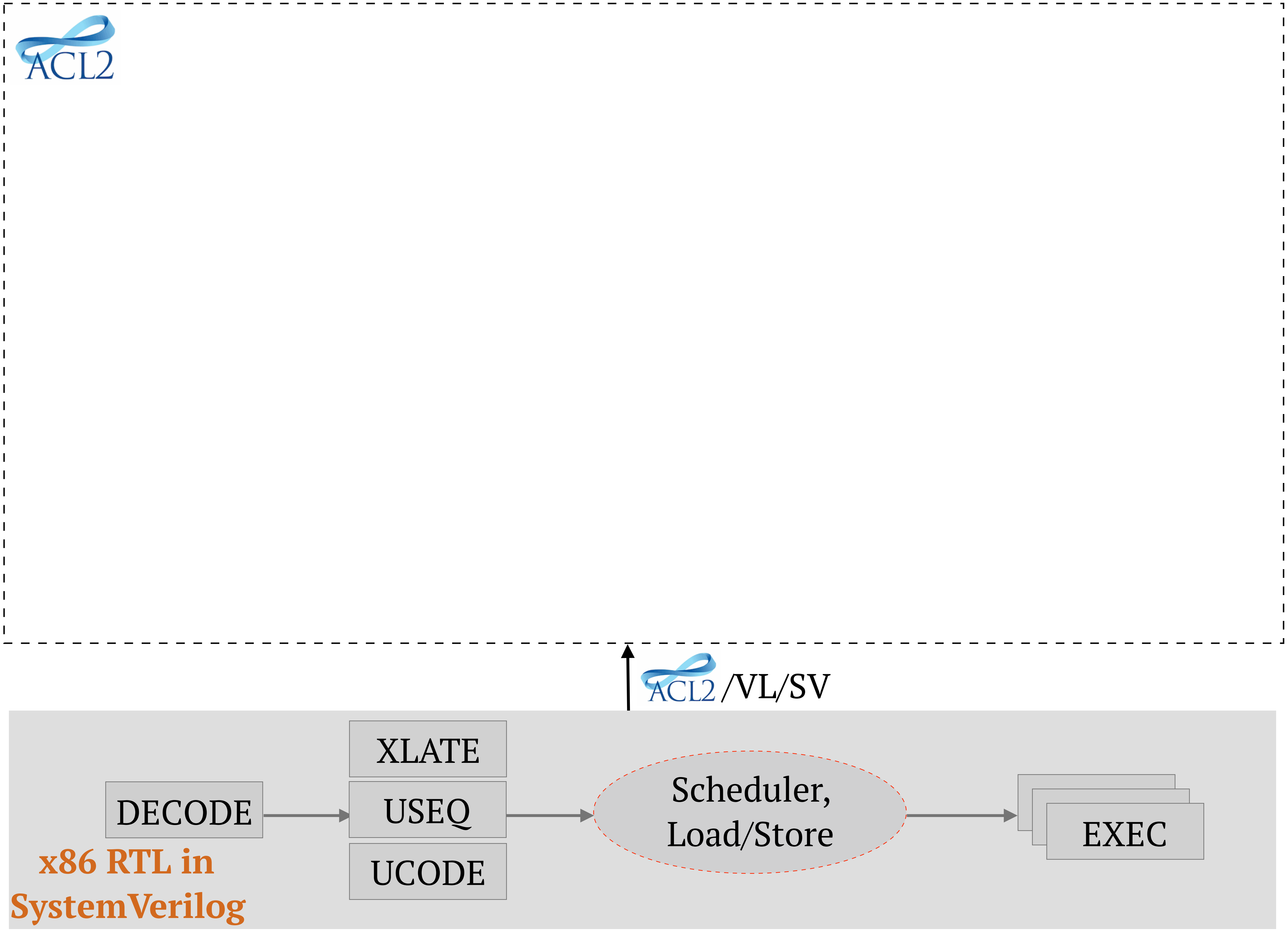


Execution of an x86 Instruction

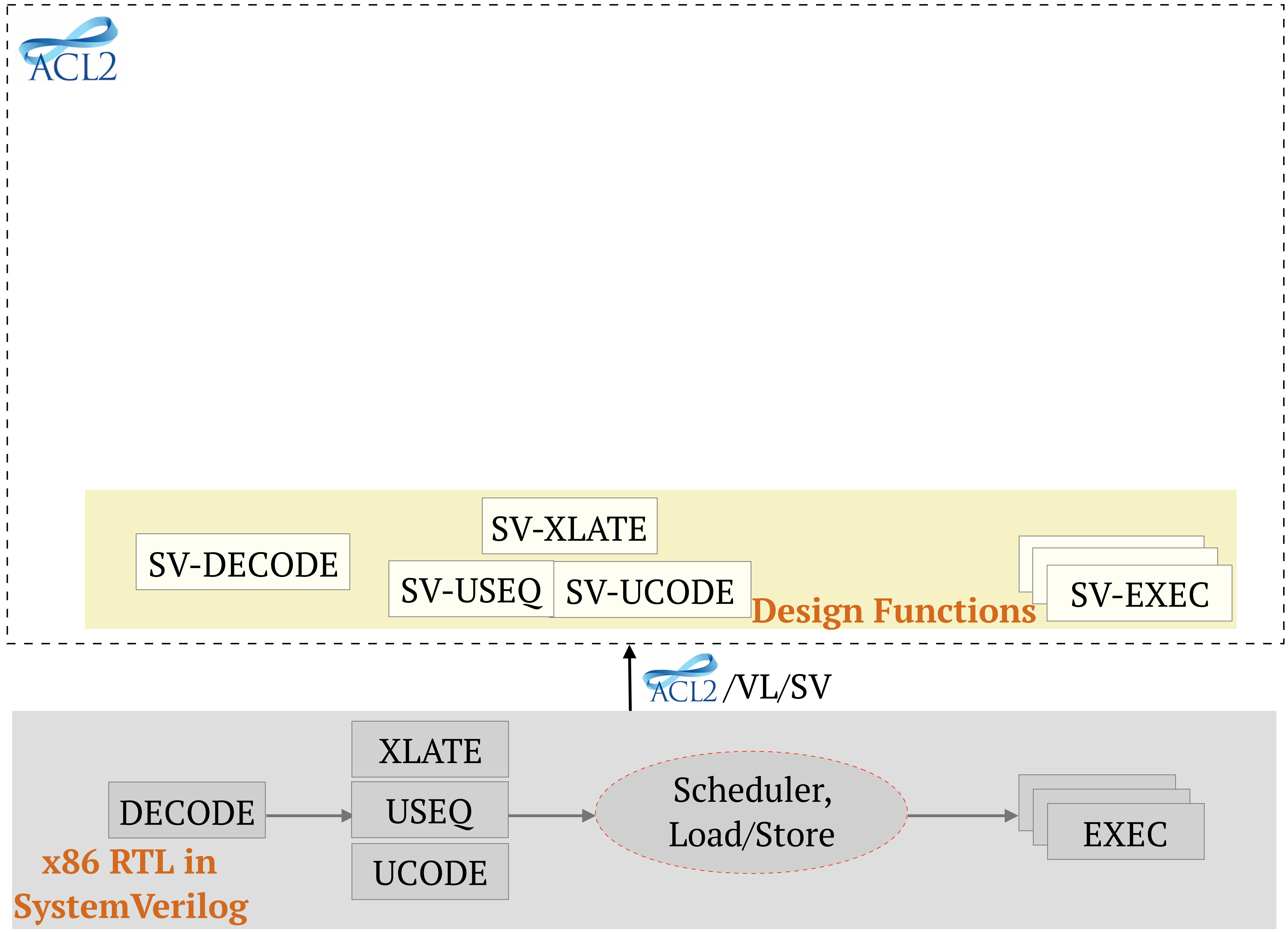
Overview


XLATE
DECODE
USEQ
UCODE
Scheduler, Load/Store
EXEC
x86 RTL in SystemVerilog

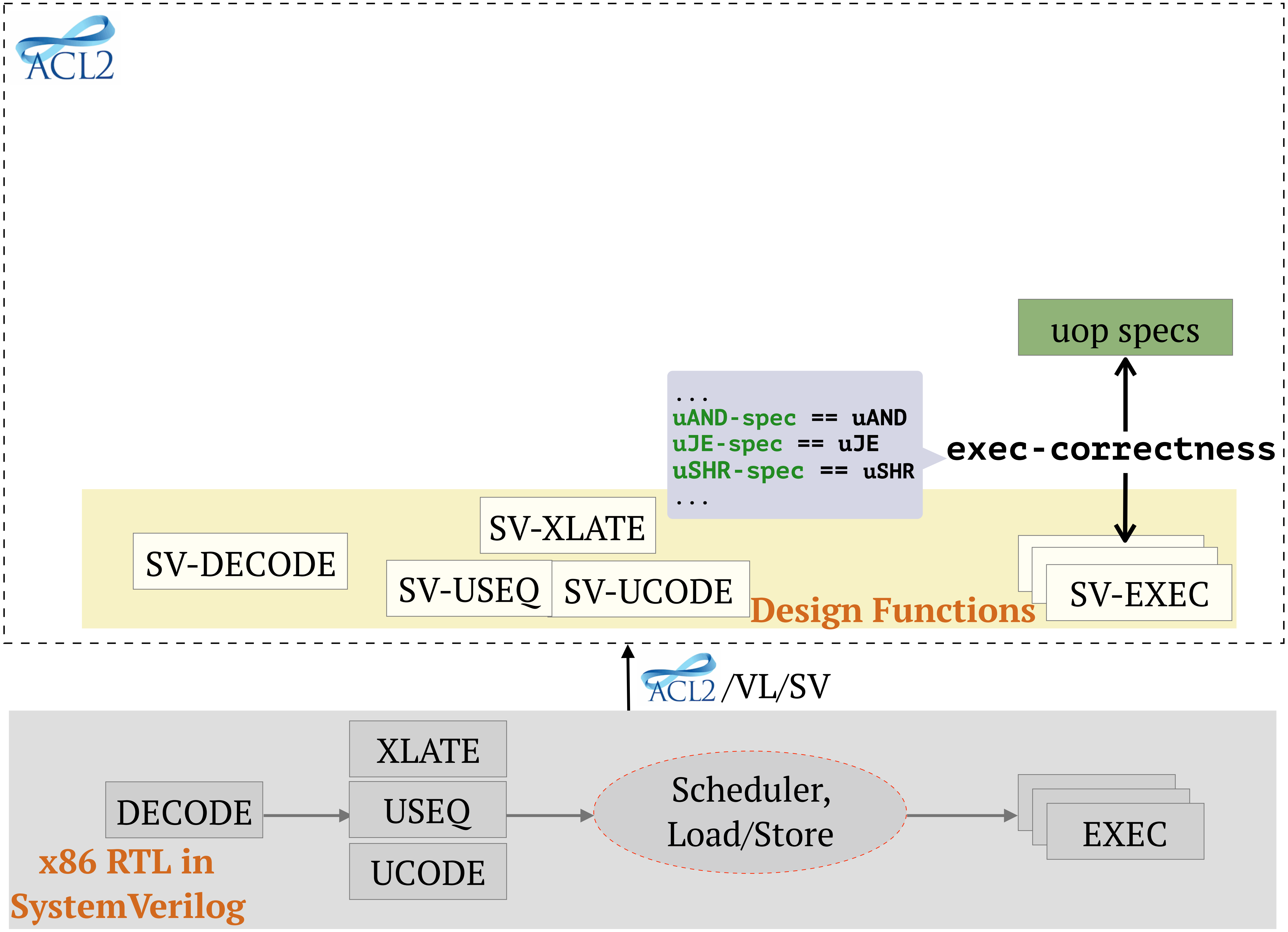
Overview


ACL2
ACL2 /VL/SV
XLATE
DECODE
USEQ
UCODE
Scheduler, Load/Store
EXEC
x86 RTL in SystemVerilog

Overview

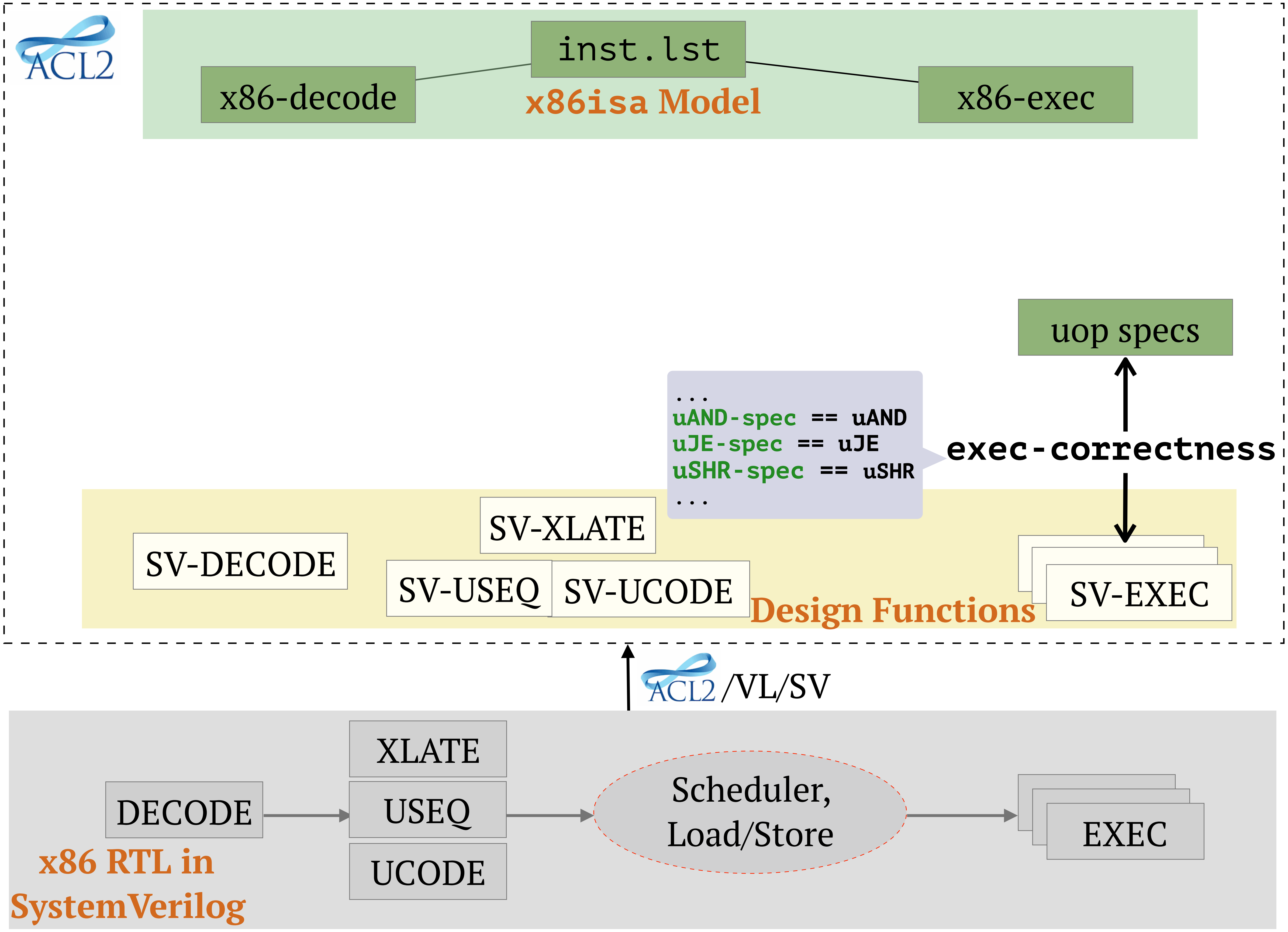

ACL2
SV-XLATE
SV-DECODE
SV-USEQ
SV-UCODE
Design Functions
SV-EXEC
ACL2 /VL/SV
XLATE
DECODE
USEQ
UCODE
Scheduler,
Load/Store
EXEC
x86 RTL in
SystemVerilog

Overview

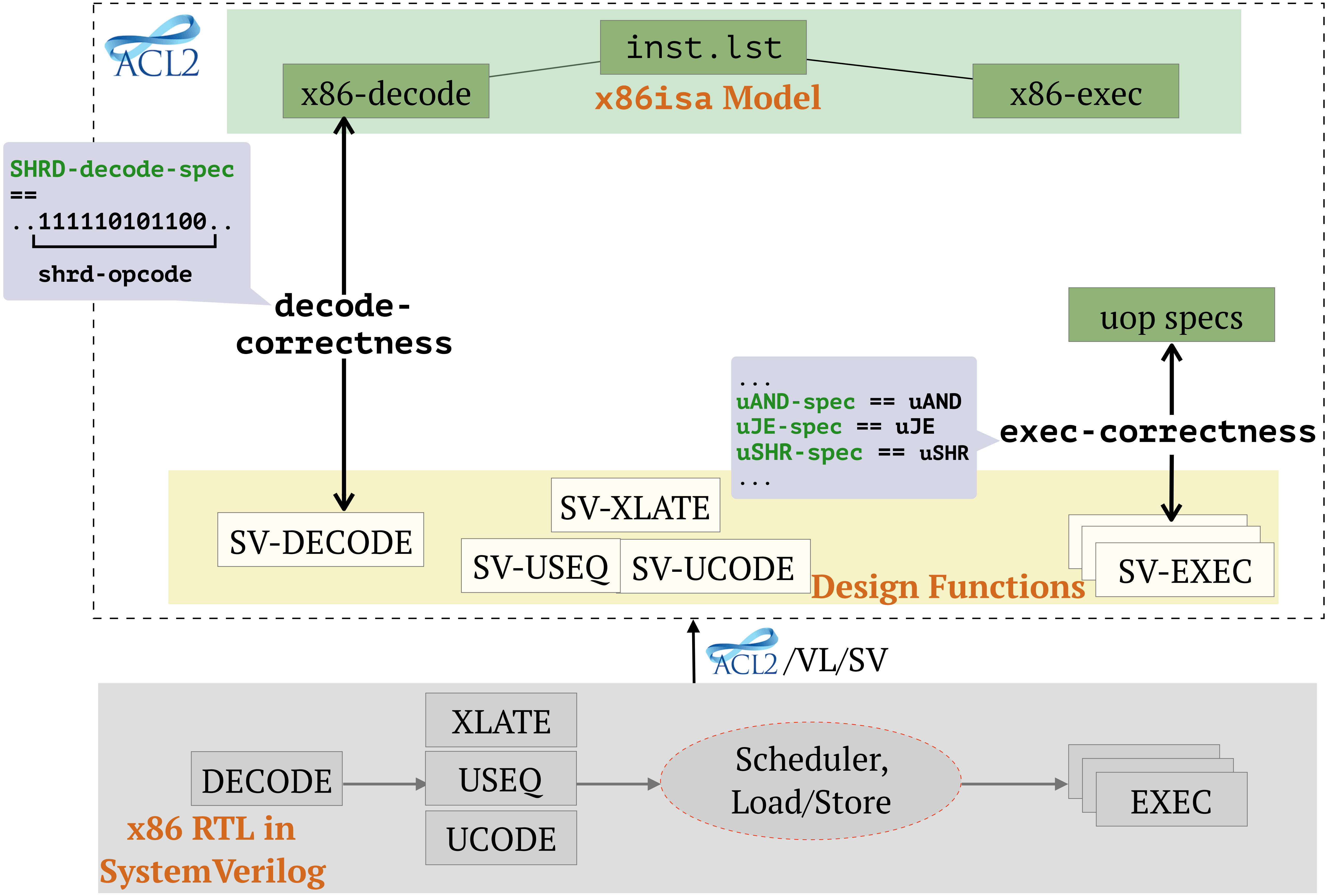


```
...
uAND-spec == uAND
uJE-spec == uJE
uSHR-spec == uSHR
...
```

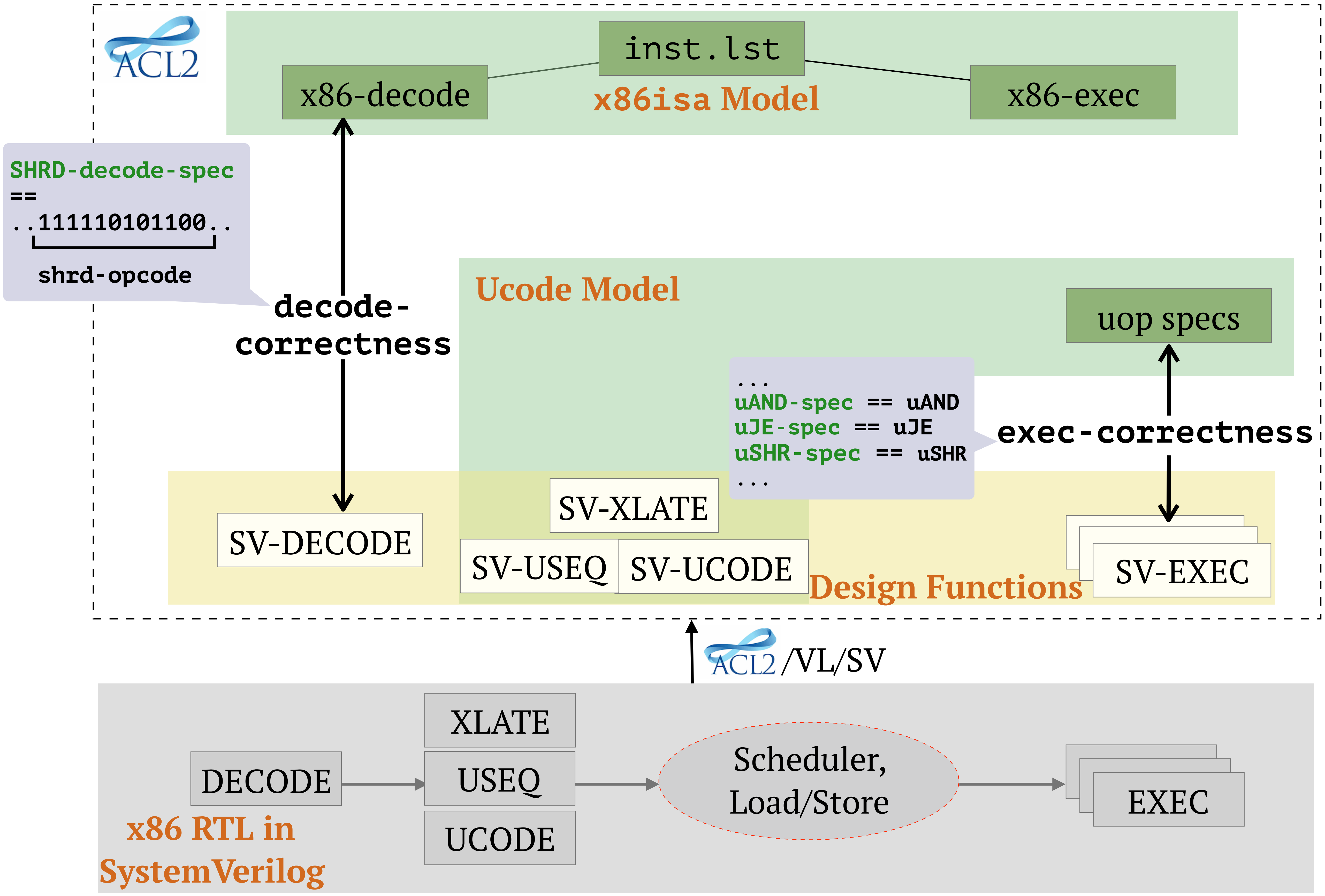
Overview


inst.lst
x86-decode
x86isa Model
x86-exec
uop specs
...
uAND-spec == uAND
uJE-spec == uJE
uSHR-spec == uSHR
...
exec-correctness
SV-XLATE
SV-DECODE
SV-USEQ
SV-UCODE
Design Functions
SV-EXEC
ACL2/VL/SV
XLATE
DECODE
USEQ
UCODE
Scheduler, Load/Store
EXEC
x86 RTL in SystemVerilog

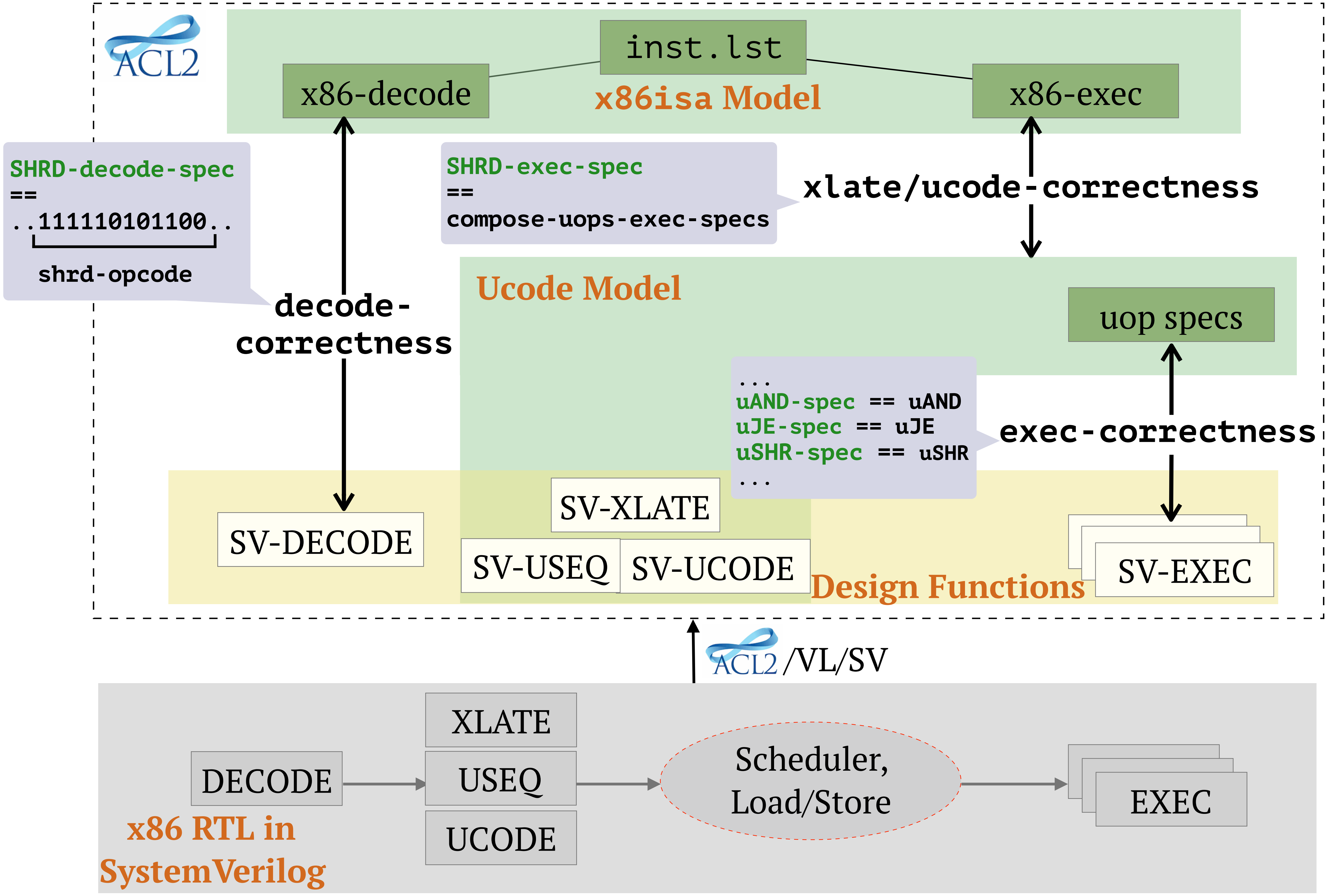
Overview


ACL2
inst.lst
x86-decode
x86isa Model
x86-exec
SHRD-decode-spec
==
..111110101100..
shrd-opcode
decode-correctness
uop specs
...
uAND-spec == uAND
uJE-spec == uJE
uSHR-spec == uSHR
...
exec-correctness
SV-XLATE
SV-DECODE
SV-USEQ
SV-UCODE
Design Functions
SV-EXEC
ACL2 /VL/SV
XLATE
DECODE
USEQ
UCODE
Scheduler, Load/Store
EXEC
x86 RTL in SystemVerilog

Overview


ACL2
inst.lst
x86-decode
x86isa Model
x86-exec
SHRD-decode-spec
==
..111110101100..
shrd-opcode
decode-correctness
Ucode Model
uop specs
...
uAND-spec == uAND
uJE-spec == uJE
uSHR-spec == uSHR
...
exec-correctness
SV-XLATE
SV-DECODE
SV-USEQ
SV-UCODE
SV-EXEC
Design Functions
ACL2 /VL/SV
XLATE
DECODE
USEQ
UCODE
Scheduler, Load/Store
EXEC
x86 RTL in SystemVerilog

Overview



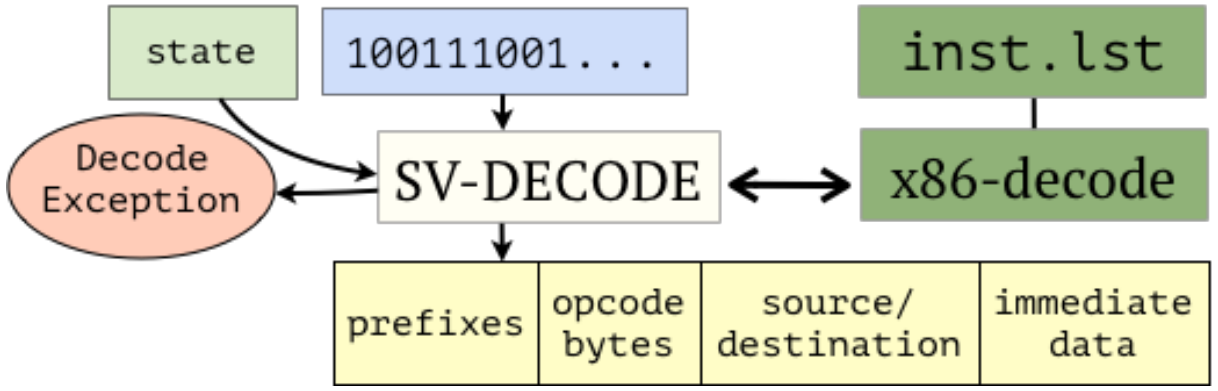
ACL2
inst.lst
x86-decode
x86isa Model
x86-exec
SHRD-decode-spec
==
..111110101100..
shrd-opcode
SHRD-exec-spec
==
compose-uops-exec-specs
xlate/ucode-correctness
decode-correctness
Ucode Model
uop specs
...
uAND-spec == uAND
uJE-spec == uJE
uSHR-spec == uSHR
...
exec-correctness
SV-XLATE
SV-DECODE
SV-USEQ
SV-UCODE
Design Functions
SV-EXEC
ACL2 /VL/SV
XLATE
DECODE
USEQ
Scheduler, Load/Store
EXEC
UCODE
x86 RTL in SystemVerilog

Proof Methodology

- **Goal:** prove that ***design functions*** are consistent with the ***x86 ISA model***.
- **Strategy:** ***decompose the problem*** into proving three kinds of lemmas.
  1. `decode-correctness`
  2. `exec-correctness`
  3. `xlate/ucode-correctness`
- All proofs are done using the ***ACL2 theorem prover***.
  - Consistent composition of intermediate results.
- Use GL library in ACL2 to ***translate each lemma into a propositional formula for bit-blasting*** using SAT, BDDs, and AIG rewriting.
  - Speed up and automate these proofs.

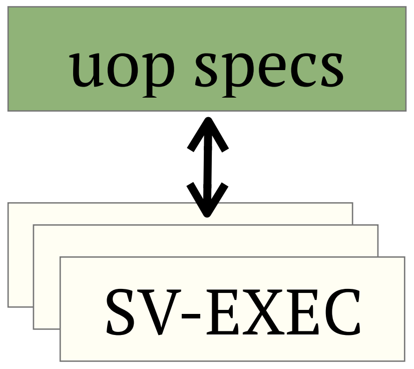
1) decode-correctness

- **Decomposition:**
  - Case splits due to parsing of the byte sequence (e.g., prefix bytes).
  - A few thousand cases across all opcodes verified in parallel on multiple machines. Each case takes ~5-10s using GL/SAT.
- **Challenges:**
  - ***Scale:*** ~3400 x86 instructions.
  - ***Complex hardware design:*** queues, feedback loops, etc.
    - *Assumption:* start from a generalized state with constraints that ensure no interference or impedance to the instruction's execution.
- **New at Centaur:** Verified all supported instructions for the most recent project.



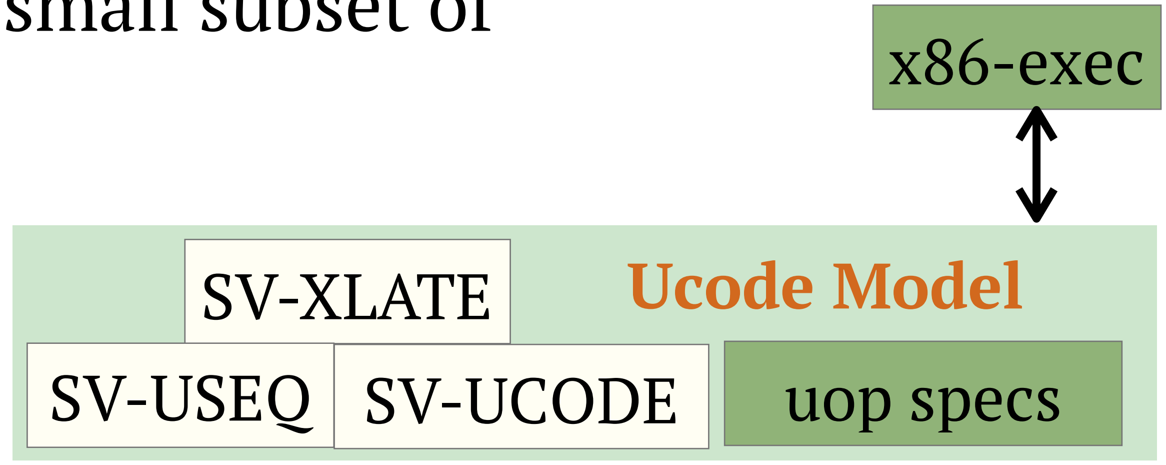
2) exec-correctness

- **Decomposition:** uop-specific (e.g., near/far paths for FP adders).
- **Challenge:** No standardized behavior — uop specifications obtained by talking to logic designers.
- **Improved ongoing work at Centaur:**
  - We now consider all uops dispatched to execution units.
    - ~600 uops in the most recent project.
  - Scope of proofs lifted to the same top-level execution unit.
    - Less susceptible to changes in internal modules.



3) xlate/ucode-correctness

- **Decomposition:** Pick a legal instruction corresponding to a fixed set of uop operations — reason about one ucode program at a time.
  - Operations fixed; other fields (e.g., register indices) symbolic.
- **Challenge:**
  - ***Uop programs:*** microcode ROM can have arbitrary-length programs with jumps and loops.
    - Use software verification techniques.
- **Improved ongoing work at Centaur:**
  - We consider both prelude and microcode uops; previously, we focused on microcode uops only.
  - This is work-in-progress; proofs of a small subset of instructions have been done so far.



Conclusions

- ***Summary:***
  - Enhanced our previous work of verifying EXEC blocks.
  - Developed a framework to verify DECODE and XLATE/UCODE blocks.
  - Composed these pieces to prove instruction implementations correct.

Conclusions

- ***Summary:***
  - Enhanced our previous work of verifying EXEC blocks.
  - Developed a framework to verify DECODE and XLATE/UCODE blocks.
  - Composed these pieces to prove instruction implementations correct.
- Found some ***bugs*** in all target design components:
  - E.g.: translator produced incorrect source operand for a uop, illegal byte sequence didn't throw an exception, etc.

Conclusions

- ***Summary:***
  - Enhanced our previous work of verifying EXEC blocks.
  - Developed a framework to verify DECODE and XLATE/UCODE blocks.
  - Composed these pieces to prove instruction implementations correct.
- Found some ***bugs*** in all target design components:
  - E.g.: translator produced incorrect source operand for a uop, illegal byte sequence didn't throw an exception, etc.
- Adopt a ***divide-and-conquer strategy***:
  - EXEC and DECODE units can be verified independently.
  - Different proof strategies can be used for different modules.

Conclusions

- ***Summary:***
  - Enhanced our previous work of verifying EXEC blocks.
  - Developed a framework to verify DECODE and XLATE/UCODE blocks.
  - Composed these pieces to prove instruction implementations correct.
- Found some ***bugs*** in all target design components:
  - E.g.: translator produced incorrect source operand for a uop, illegal byte sequence didn't throw an exception, etc.
- Adopt a ***divide-and-conquer strategy***:
  - EXEC and DECODE units can be verified independently.
  - Different proof strategies can be used for different modules.
- ***Immune to many RTL changes***:
  - No need to specify or understand instruction-to-uops translation.
  - Immune to changes in microcode ROM, assembler, internal modules.

Conclusions (contd.)

- We contribute to ***publicly-available ACL2 libraries***.
  - Formal verification of hardware in the industry is possible without needing to purchase expensive license-only tools.

- ***Work-in-Progress/Future Work:***
  - **Verify other parts of the processor:**
    - Memory operations (load/store), scheduler, etc.
  - **Increase coverage for `xlate/ucode-correctness`:**
    - Prove more instruction variants correct.
  - **Increase automation:**
    - Automatically prove the correctness of simple instructions.
    - Automatically check that component lemmas cover all possible cases.

Verifying x86 Instruction Implementations

Shilpi Goel, Anna Slobodova, Rob Sumners, & Sol Swords
`{shilpi,anna,rsumners,sswords}@centtech.com`

Formal Verification Team @ Centaur Technology

Questions?

CPP 2020