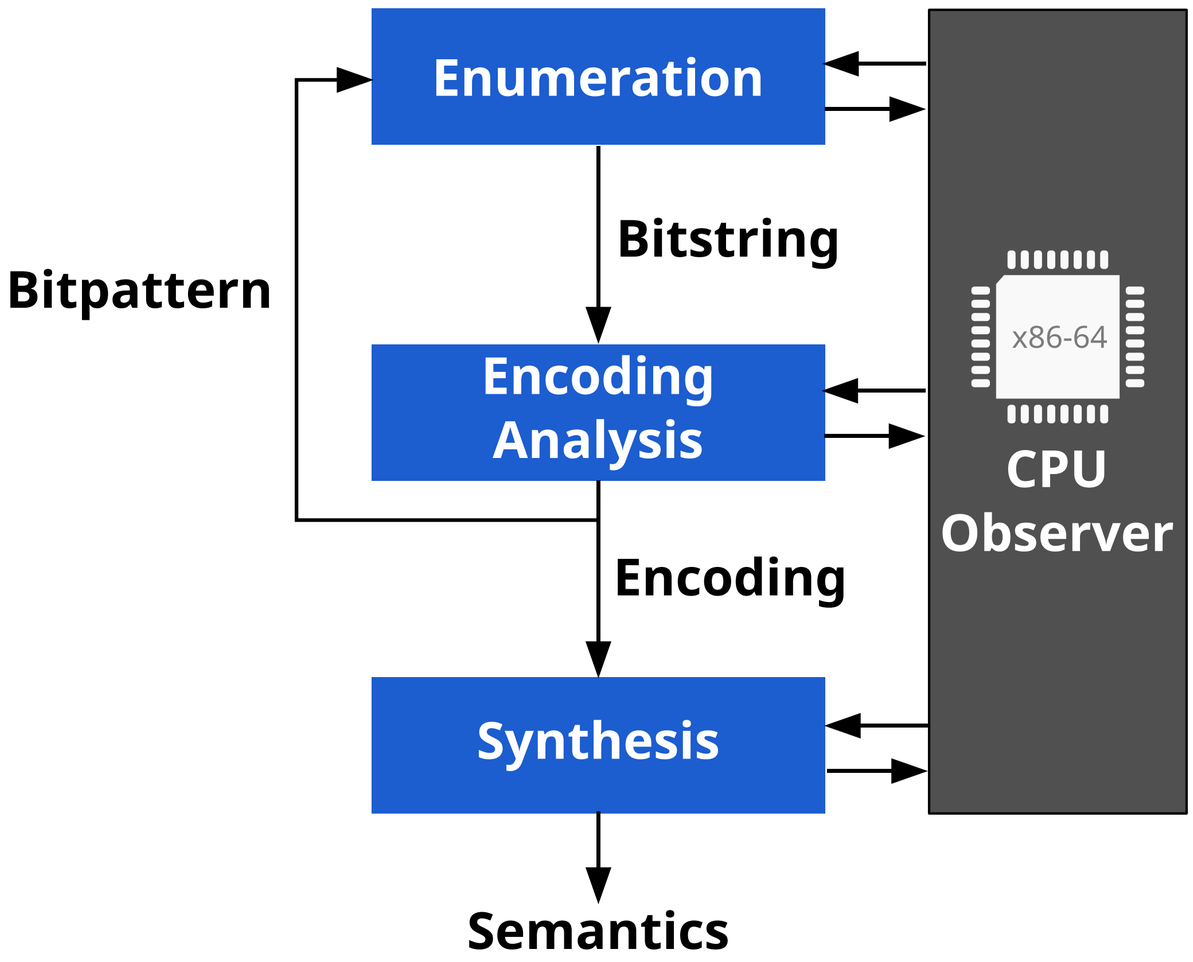
LIBLISA: Instruction Discovery and Analysis on x86-64

Jos Craaijo (Open Universiteit), **Freek Verbeek** (Open Universiteit & Virginia Tech),
Binoy Ravindran (Virginia Tech)

Analysis Overview



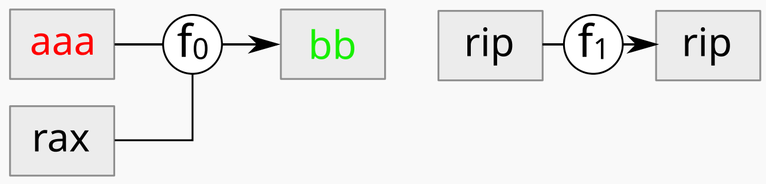
Encodings and Semantics

Encoding

Bitpattern

a0aa11bb
Part a Part b

Dataflows



Part mapping

aaa { 000 = rax, 001 = rbx, 010 = rcx, 011 = rdx, ... }
bb { 00 = rax, 01 = rbx, 10 = rcx, 11 = rdx }

Semantics

$f_0(\texttt{aaa}, \texttt{rax}) = \texttt{aaa} + \texttt{rax}$

$f_1(\texttt{rip}) = \texttt{rip} + 1$

The Problem

x86-64 still does not have a full formal model

4700 pages of semantics in the Intel reference manual

+100 new pages added to the reference manual every year

Handwritten specifications require huge amounts of manual work, which is **error prone** and **labor intensive**.

LIBLISA

We present LIBLISA, a tool for automated discovery and analysis of the instructions on a CPU.

CPU as the ground truth for the generated semantics

AUTO matic instruction enumeration and analysis

NO disassembler is used. We automatically generate *encodings*

LIBLISA observes CPU behavior and constructs semantics bottom-up.

Acknowledgements

This work is supported by the Defense Advanced Research Projects Agency (DARPA) and Naval Information Warfare Center Pacific (NIWC Pacific) under Contract No. N66001-21-C-4028.

Results

We analyzed 5 CPU architectures:

- **AMD 3900X**
- **AMD 7700X**
- **Intel Core i9-13900 (performance cores)**
- **Intel Core i9-13900 (efficiency cores)**
- **Intel Xeon Silver 4110**

21 weeks of analysis

118k encodings generated by LIBLISA's analysis

97% of all instructions in typical Linux binaries covered

89% of encodings has successfully synthesized semantics

90% of encodings with undefined behavior synthesized

All analysis results and source code are available under open source licenses.

Emulation

We implemented a proof-of-concept user-space emulator that is able to emulate some real Linux binaries.

1.2M instructions successfully emulated across 5 binaries

The emulator emulates the dynamic linker (i.e., `/lib64/ld-linux-x86-64.so.2`), all dynamically loaded libraries, as well as the binary itself.

Architecture Comparison

10% of encodings differ depending on architecture

2 instructions are sufficient to fingerprint each of the 5 architectures (group 1 + 7)

Encodings		A_0	A_1	A_2	A_3	A_4
Group 0	95170	■	■	■	■	■
Group 1	4777	■	■	●	▲	●
Group 2	2571	■	■			
Group 3	1602	■	■	●	■	●
Group 4	604	■	●	▲	▼	○
Group 5	581	■	■	●	●	●
Group 6	101	■	■	■	■	
Group 7	40		■	■	■	
Group 8	29			■	■	
Group 9	24					■
Group 10	16	■	■	■	■	●
...	...	...	...	...	...	...

Each symbol represents a different implementation for that specific group of instructions. The same symbols are re-used for each row.

Examples of differences found:

SHR when shifting by 1, AMD sets AF=1, Intel sets AF=0 (group 1)

ROL OF differs when rotating by 2 or more (group 3)

DIV only AMD CPUs modify the PF, AF, ZF and SF flags (group 5)

SHA1 not supported on architecture A_4 (group 6)

AVX 512VL-instructions not supported on A_0 + A_4 (group 7)