liblisa.nl explore.liblisa.nl github.com/liblisa

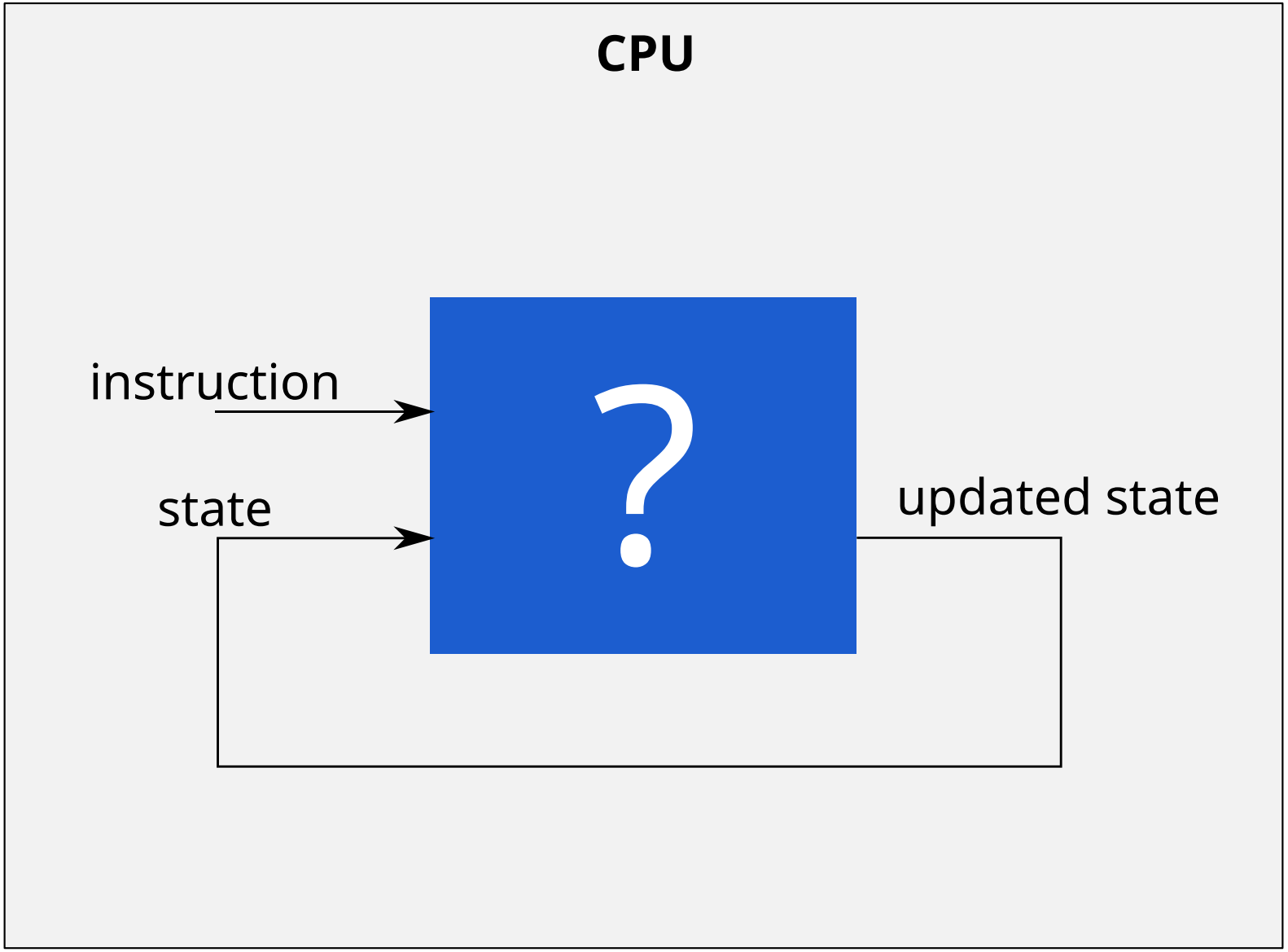
LIBLISA

Instruction Discovery and Analysis on x86-64

Jos Craaijo, Freek Verbeek, Binoy Ravindran

HARRIS'25

CPU Execution Model



Instruction Semantics

```
48 89 f8
```

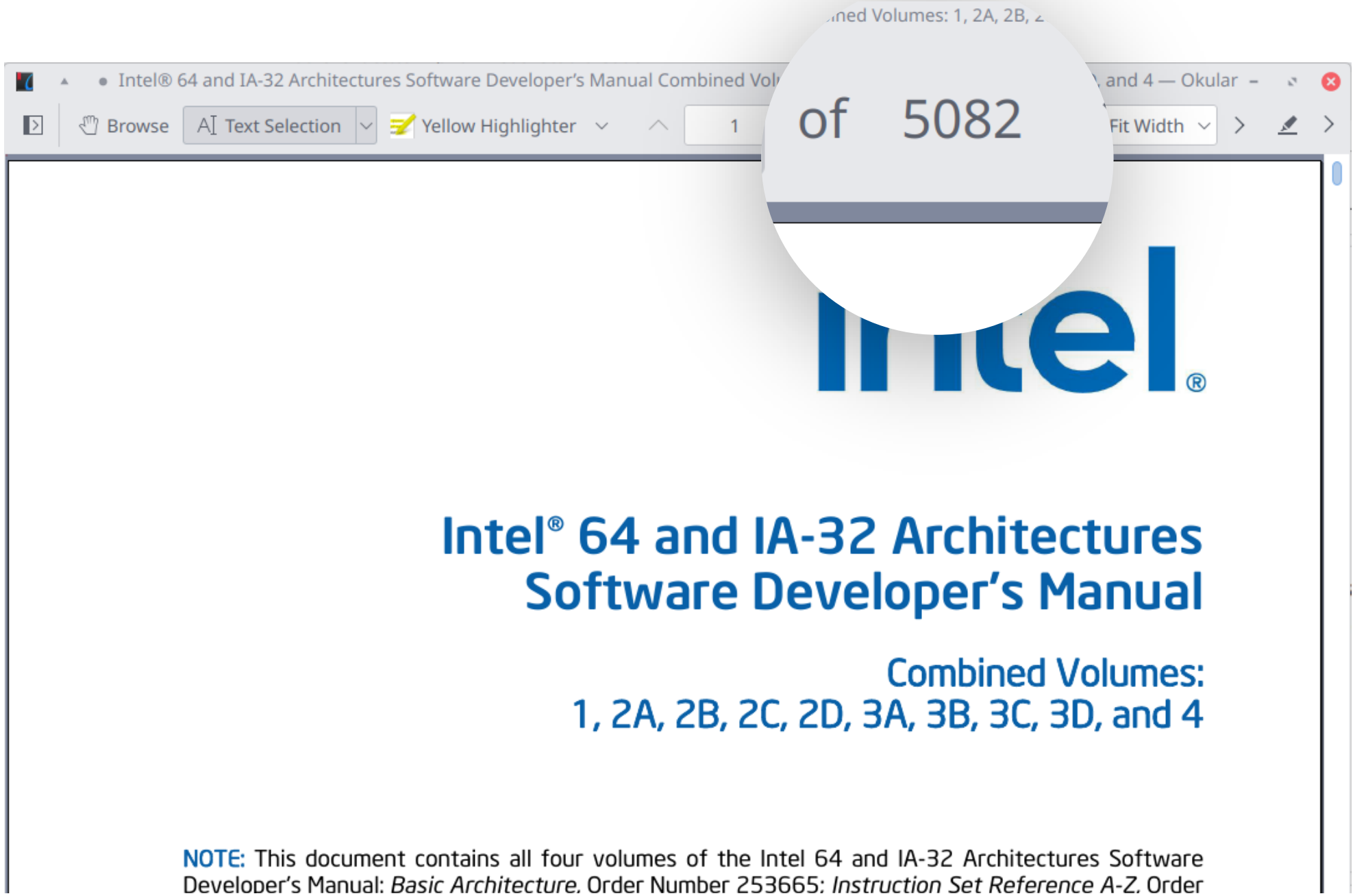
↓

Instruction Semantics

↓

- $r_{\mathrm{RDI}} := r_{\mathrm{RAX}}$
- $r_{\mathrm{RIP}} := r_{\mathrm{RIP}} + 3$

x86 Instruction Semantics



Instruction semantics are hard

Errors in Intel documentation The Intel manual only contains informal pseudo-code and an English description of the semantics, so there is no automatic way of comparing the learned formulas with the manual. However, over the over 3,800 pages of the manual, several inconsistencies majority of which is not machine-checked. Examples of these errors include:

- The upper 128 bits of the output register of `pshufb xmm0, xm` are kept unmodified on Intel hardware, but the pseudo-code in the manual incorrectly says they are cleared.
- The *Op/En* column for the `vextracti128` instruction should be MRI, but it was RMI. This causes a wrong operand ordering when assembling this instru
- One possible encoding for `xchgl eax, e` according to the manual, and its semantics are upper 32 bits of `rax`. However, `0x90` is also the encoding of `nop`, which does nothing according to the manual (and hardware).
- The textual description of `roundpd` claims to produce a single-precision floating point value, when it produces a double-precision floating point value.

We have reported all of these items and they were confirmed by Intel as errors in the manual [8].

Inconsistencies Found in the Intel Manual Here are inconsistencies found during development and testing. According to the manual, the semantics of `vpsravd %xmm3, %xmm2, %xmm1` seems to depend on the lower 100 bits of %xmm3, whereas the actual hardware execution suggests that it should depend on the lower 128 bits. Similar inconsistencies are found in instructions with mnemonics `vpsllvd`, `vpsllvq`, `vpsravd`. Also, we found misleading typos related to instructions with opcodes `vpsravw`, `vpsravd`

We also discovered cases where observed behaviors contradicts the x86 reference manual (which is unsurprising given the size and complexity of the spec). For instance, we discovered while debugging our template $T\text{-}ARI^{main}$ that the overflow `OF` flag should be set to 0 after executing `IMUL8` with 65 and 254 as inputs according to the Intel spec, while the `OF` flag is actually set to 1 after the execution of this instruction with those inputs on an Intel XEON3.7 processor.

Errors in Intel documentation

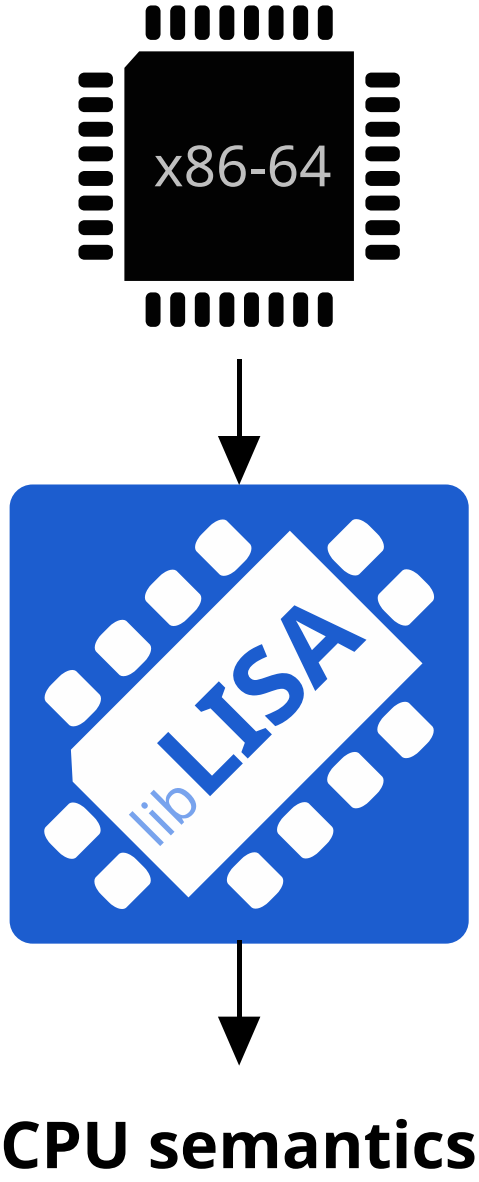
Inconsistencies Found in the Intel Manual

observed behaviors contradicts the x86 reference manual

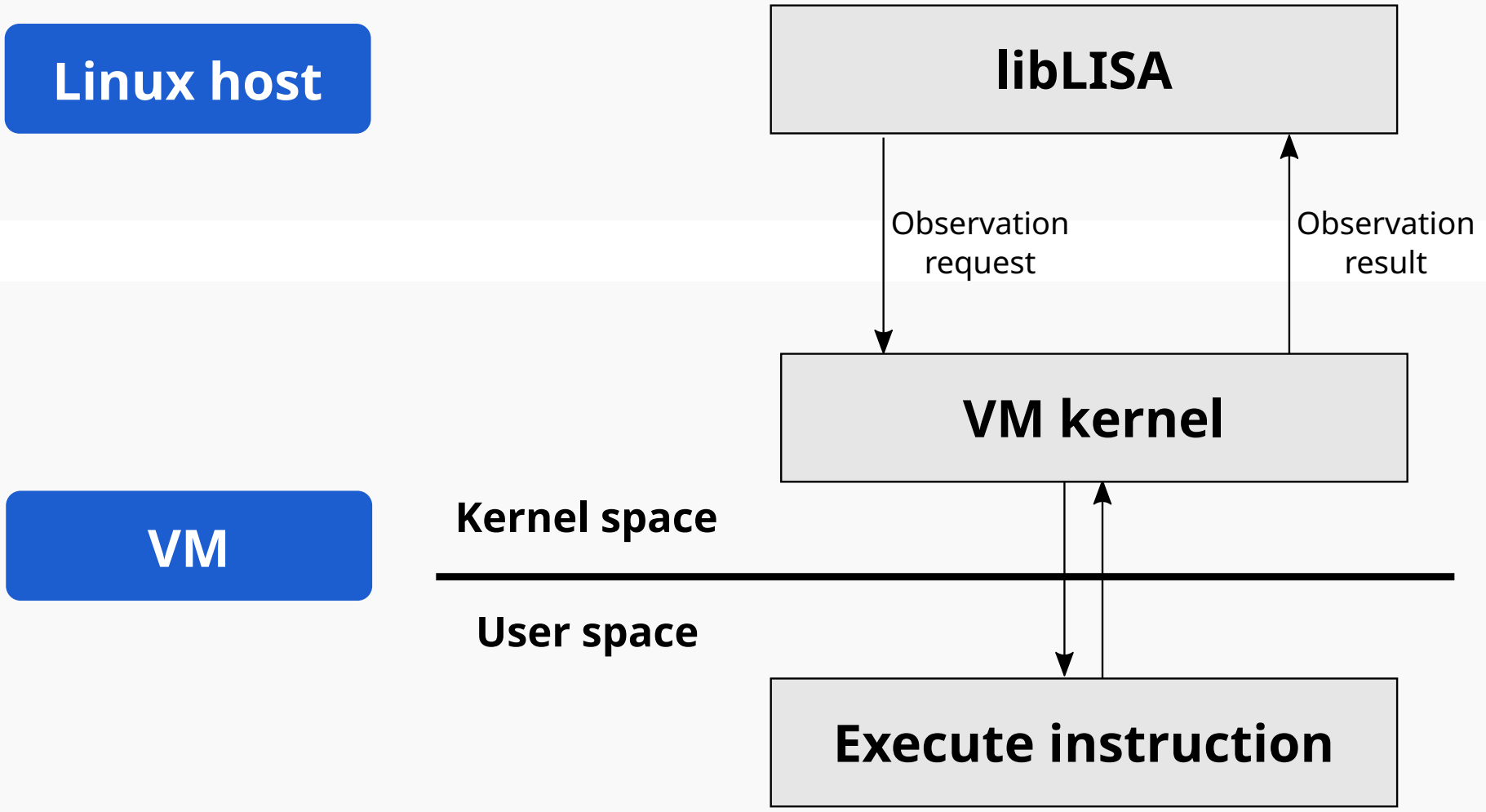
Dasgupta et al., "A complete formal semantics of x86-64 user-level instruction set architecture" (PLDI'19);
Heule et al., "Stratified Synthesis: Automatically Learning the x86-64 Instruction Set" (PLDI'16);
Godefroid and Taly, "Automated Synthesis of Symbolic Instruction Encodings from I/O Samples" (PLDI'12)

We should automate this

- **Specify only the essentials:**
  - Description of CPU state
  - CPU Observer
- **Automatically infer the rest**

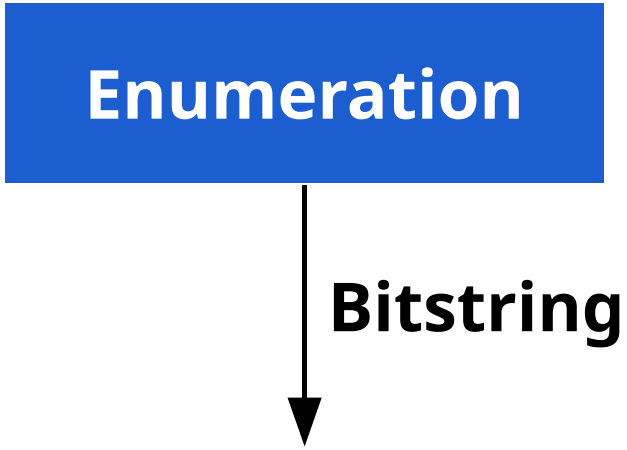


Instruction Execution Observation

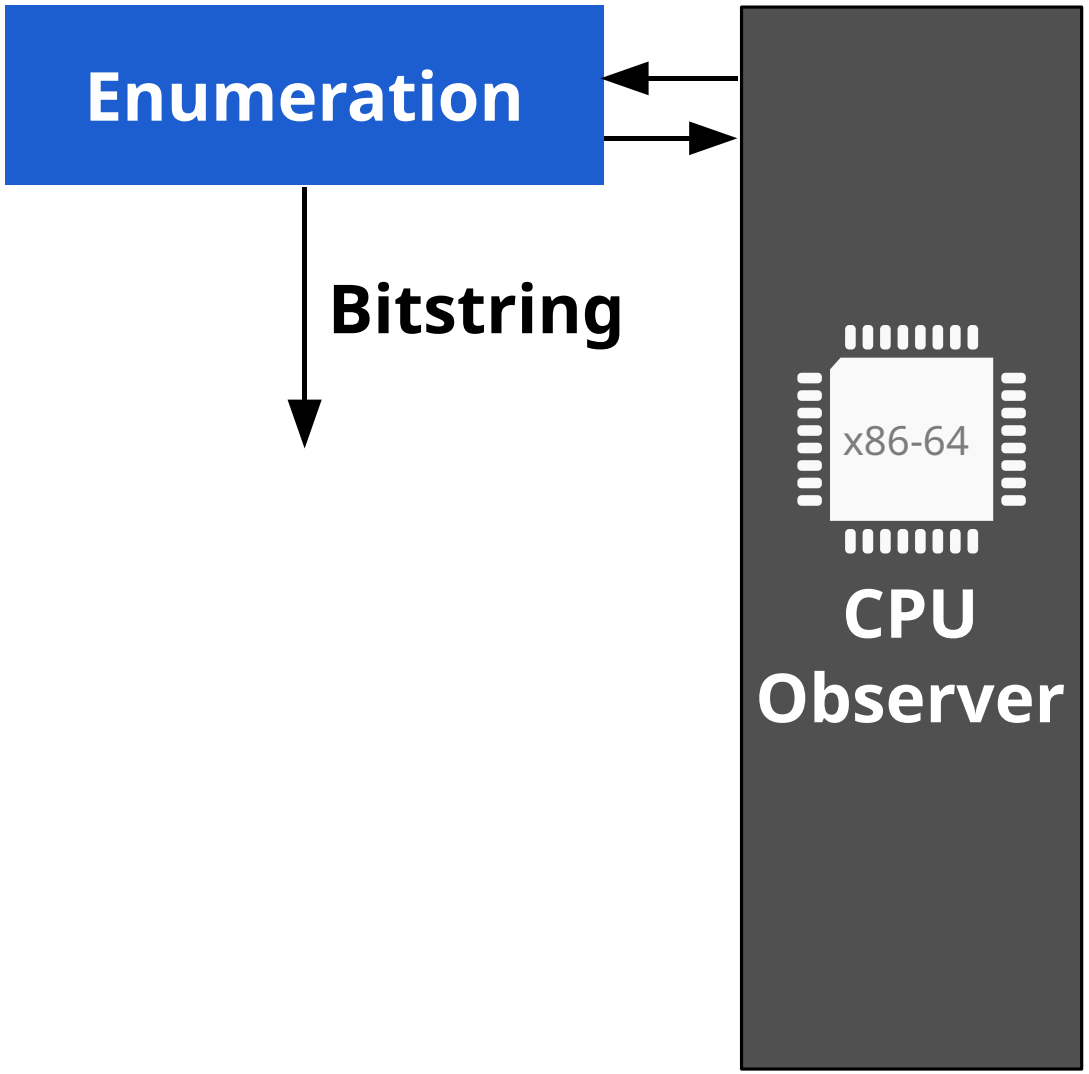


How do we automate everything?

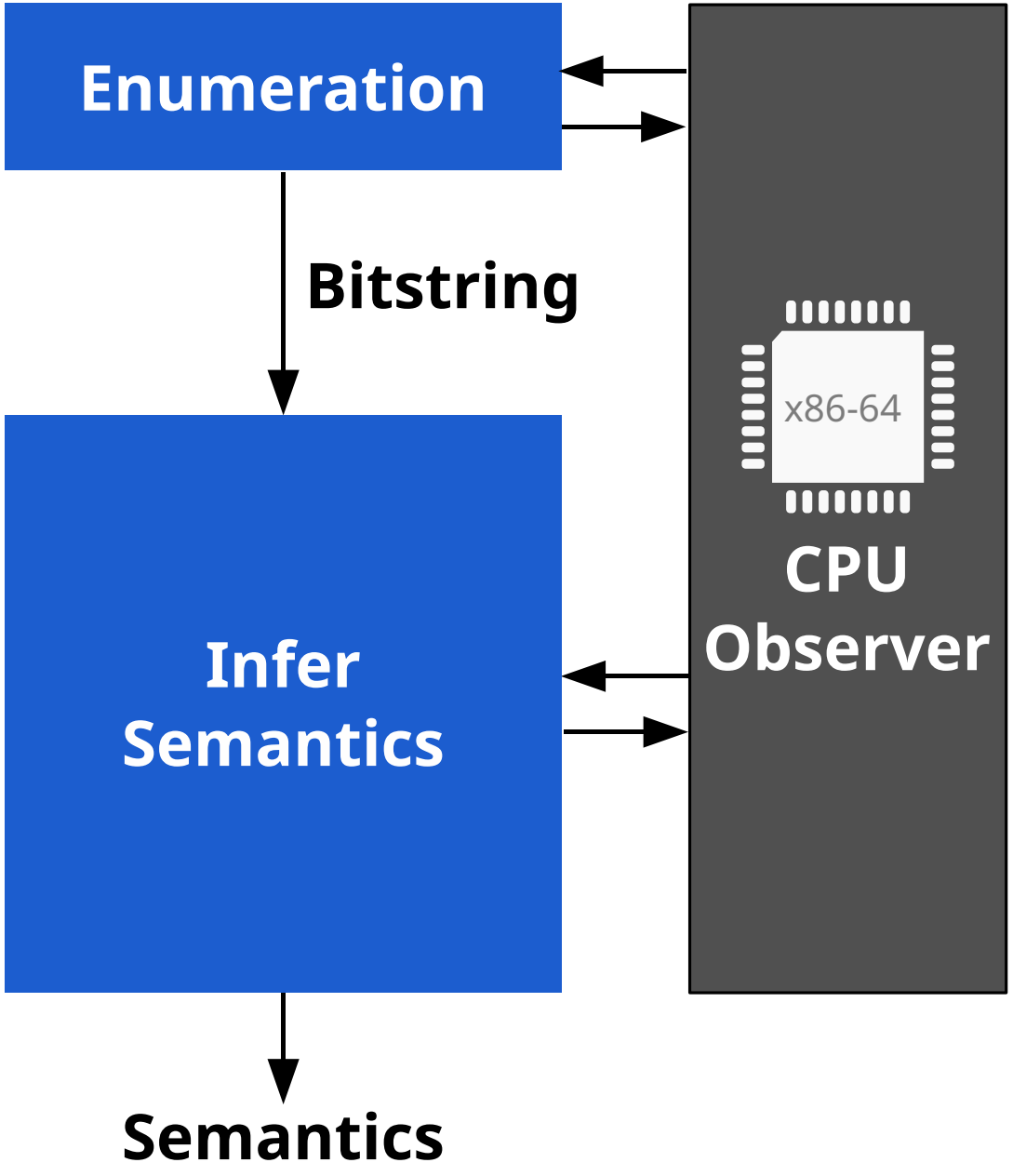
How do we automate everything?



How do we automate everything?



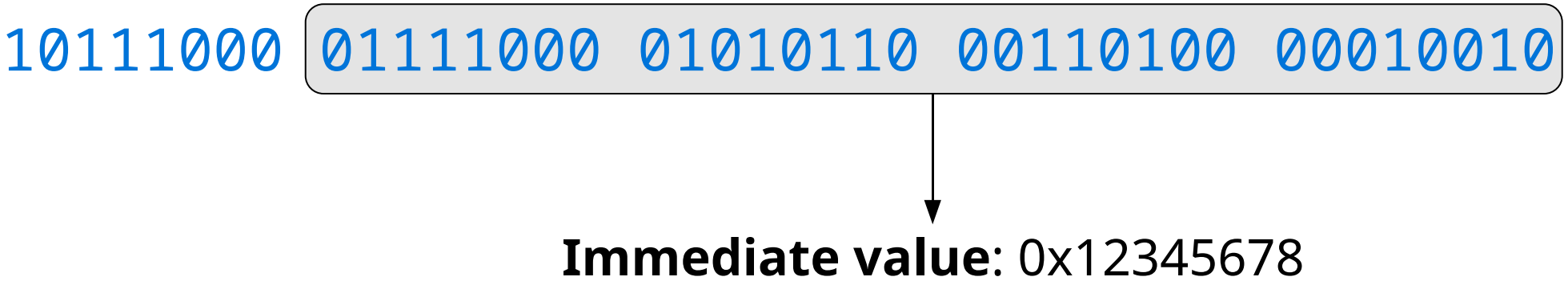
How do we automate everything?



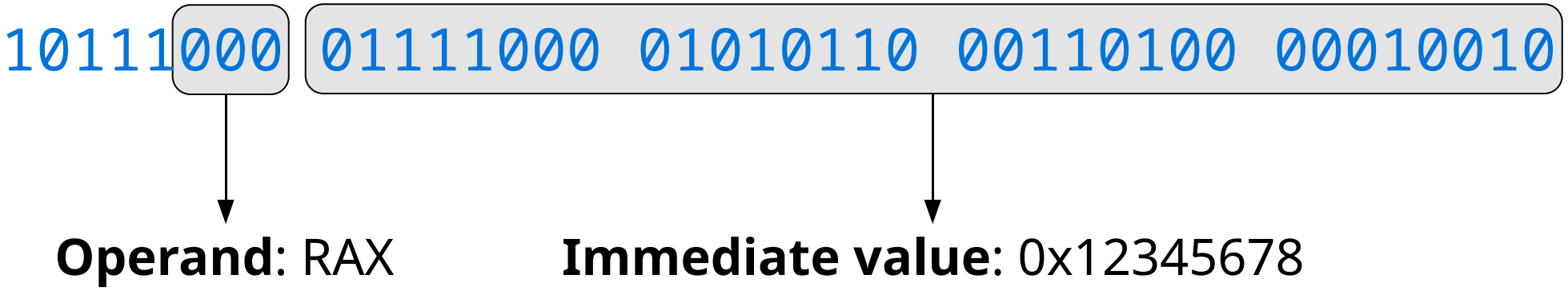
Dealing with long instructions

```
10111000 01111000 01010110 00110100 00010010
```

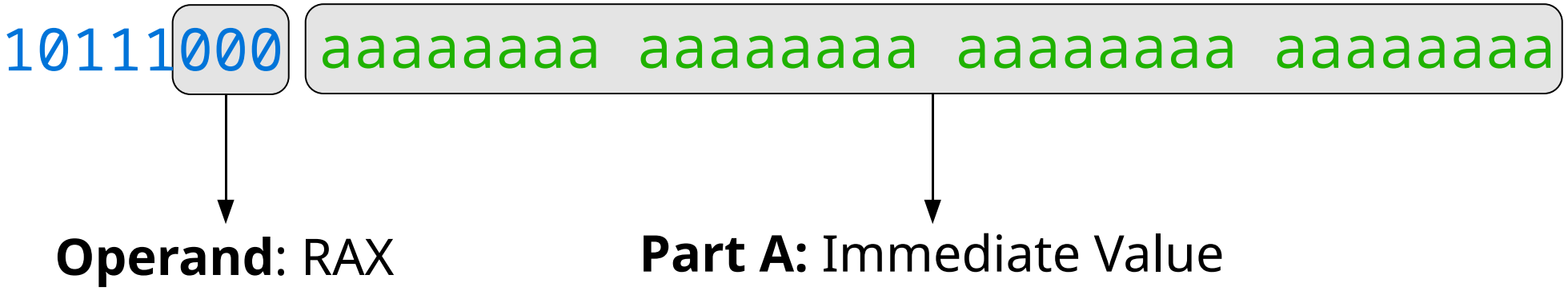
Dealing with long instructions



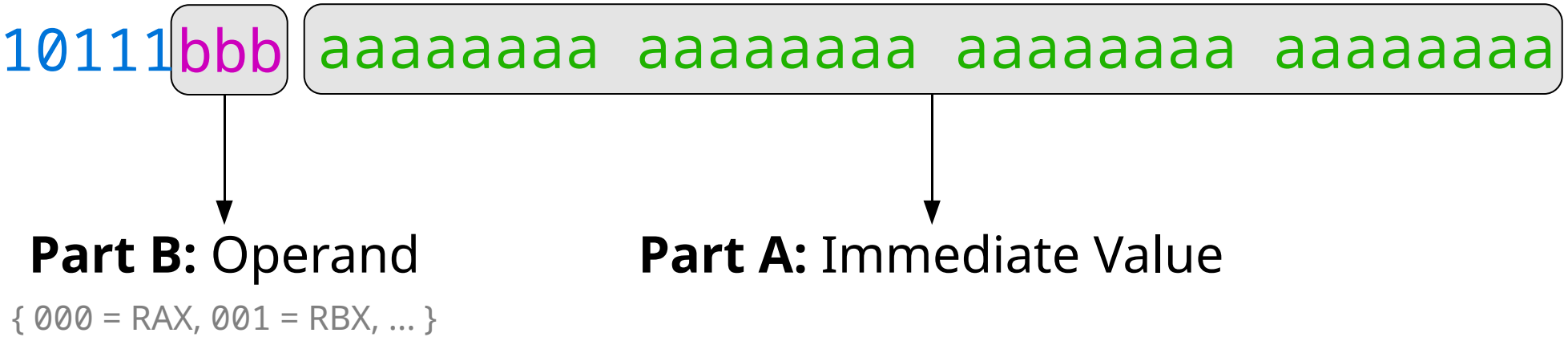
Dealing with long instructions



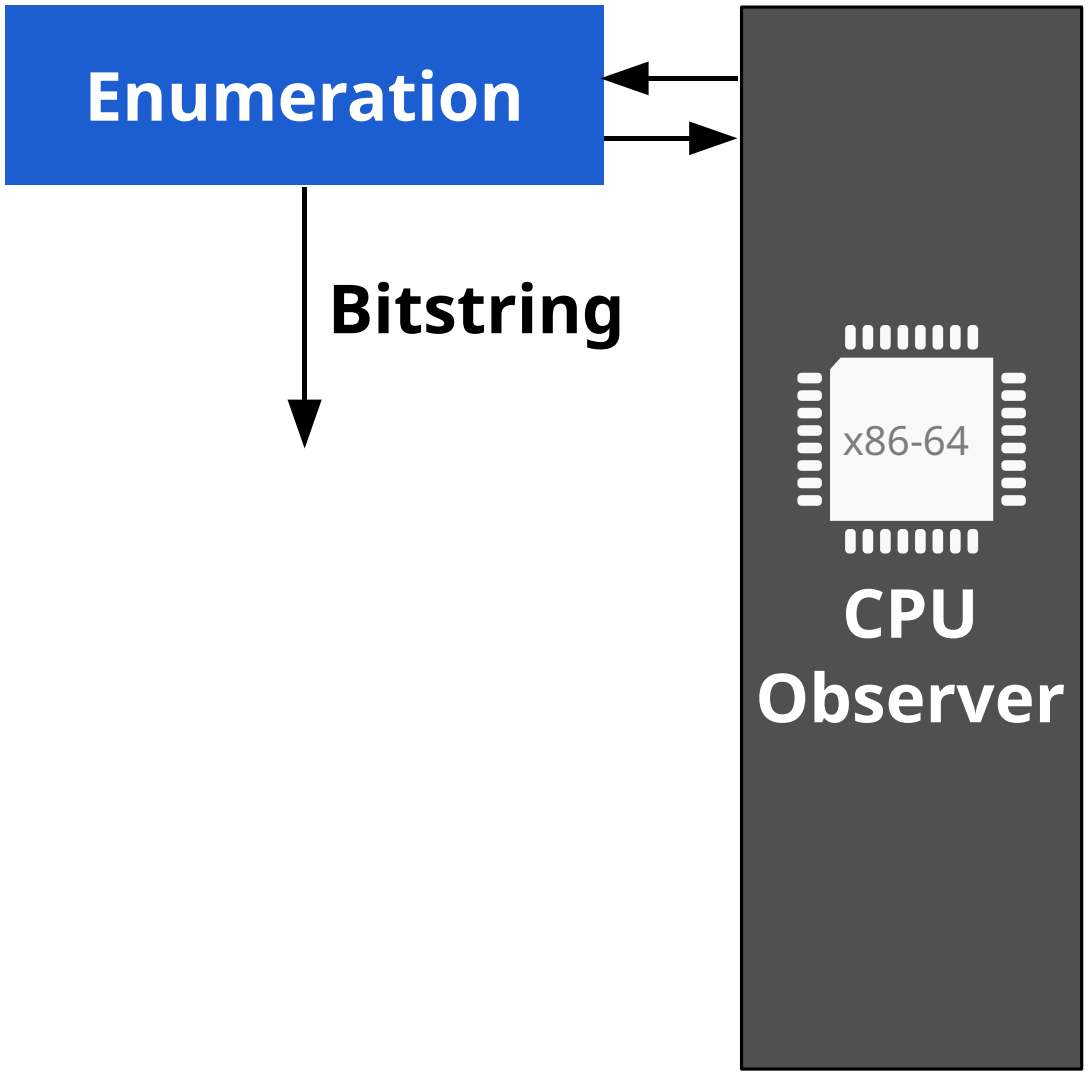
Encodings


10111000 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
Operand: RAX
Part A: Immediate Value

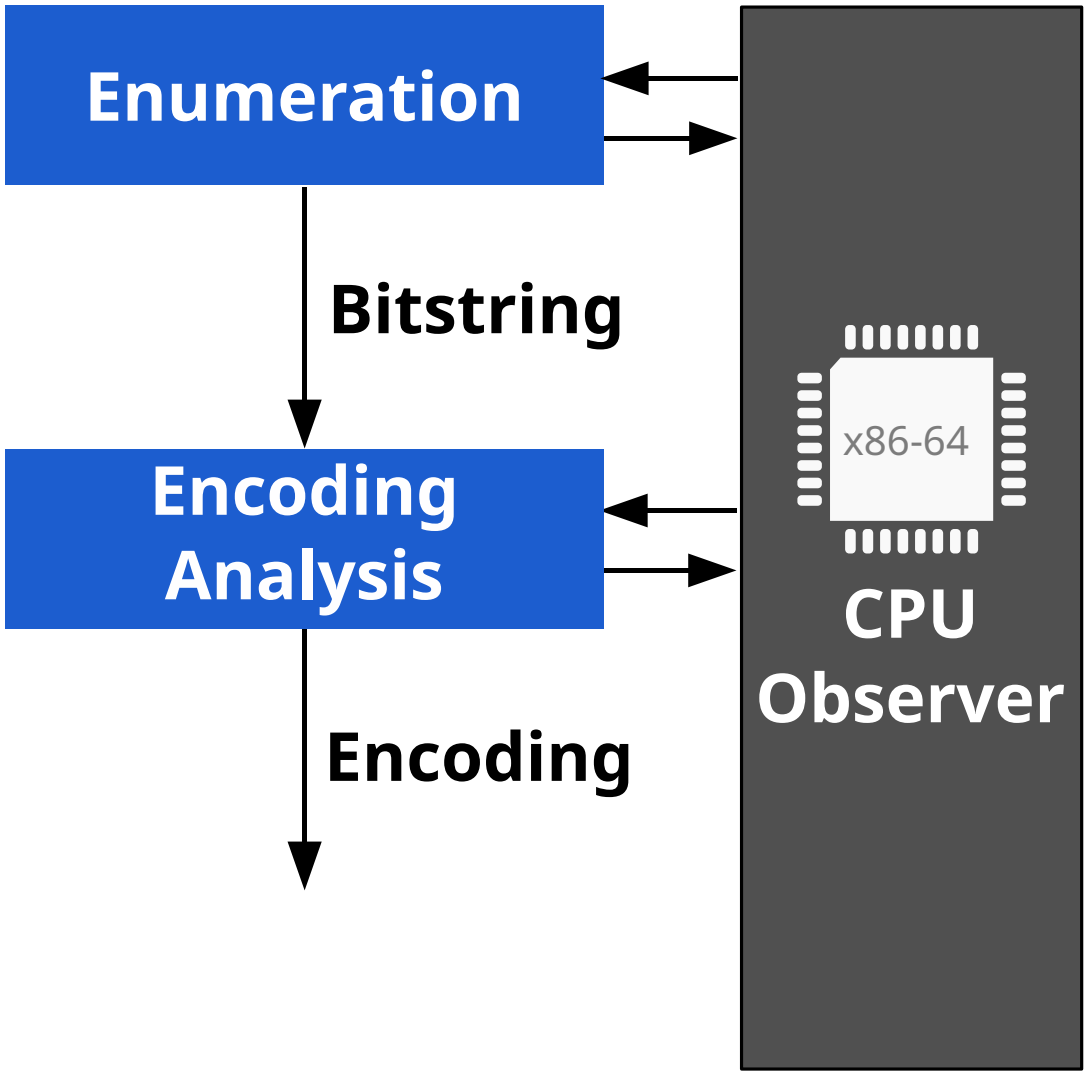
Encodings


10111bbb aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
Part B: Operand
{ 000 = RAX, 001 = RBX, ... }
Part A: Immediate Value

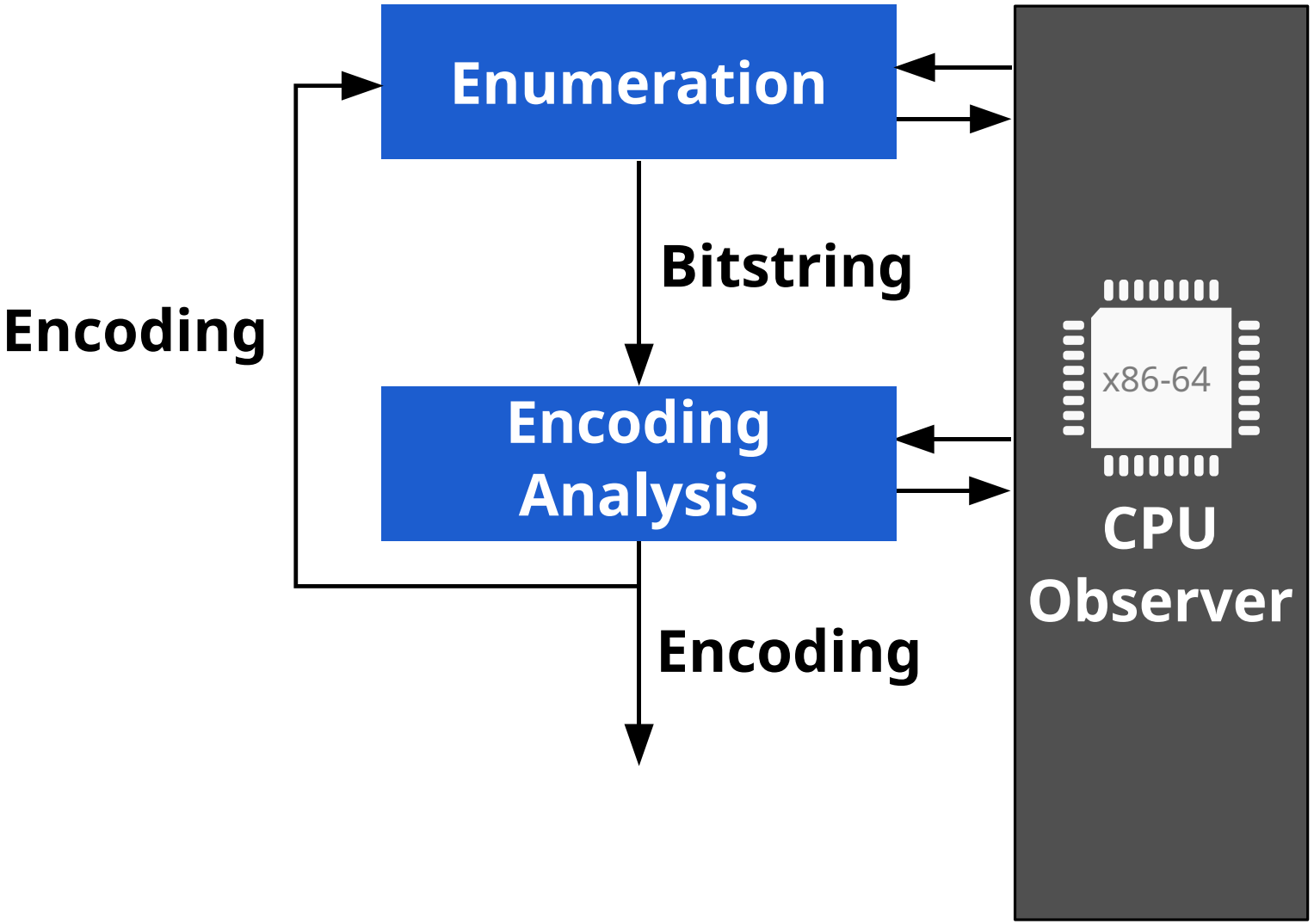
Analysis Overview



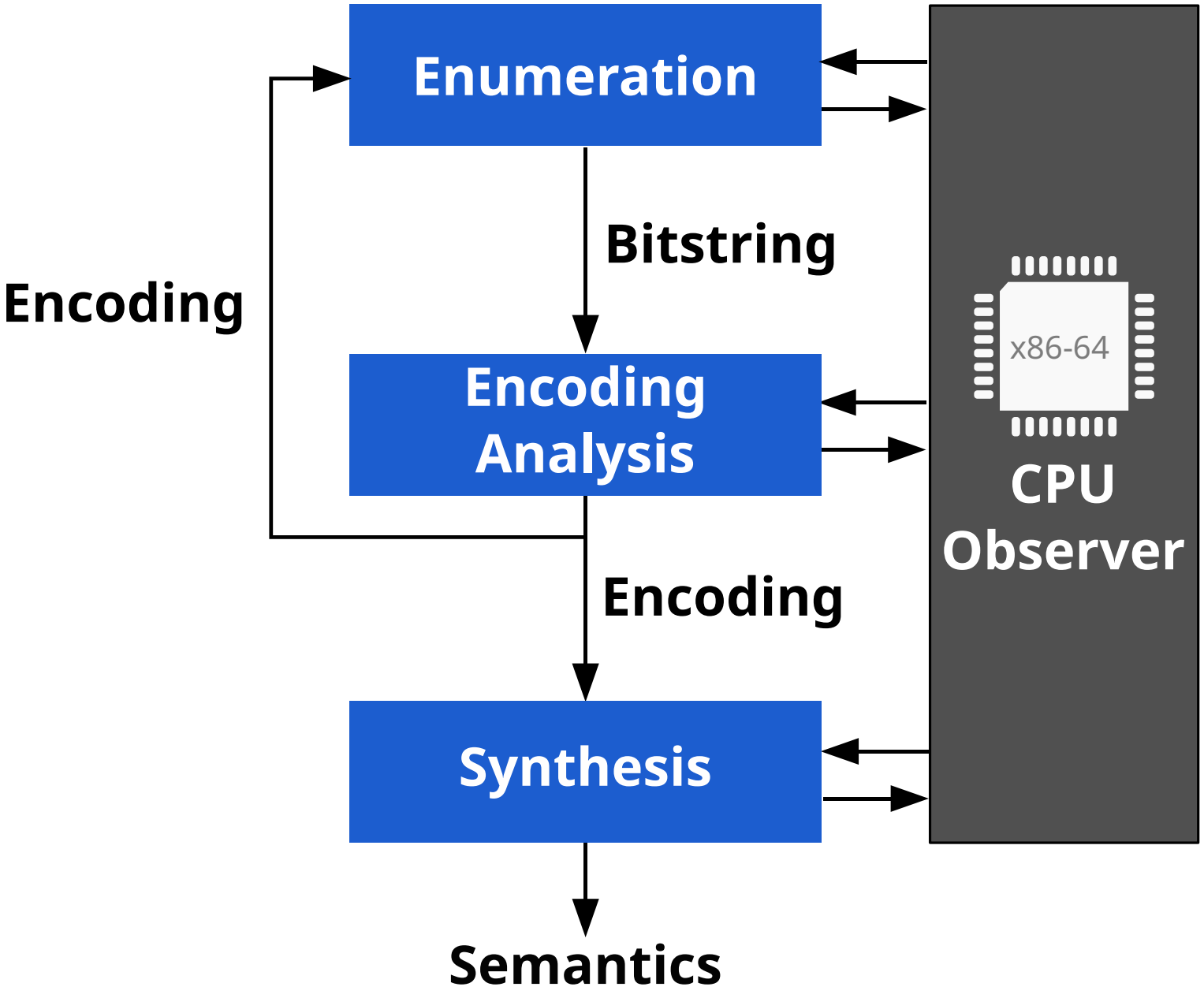
Analysis Overview



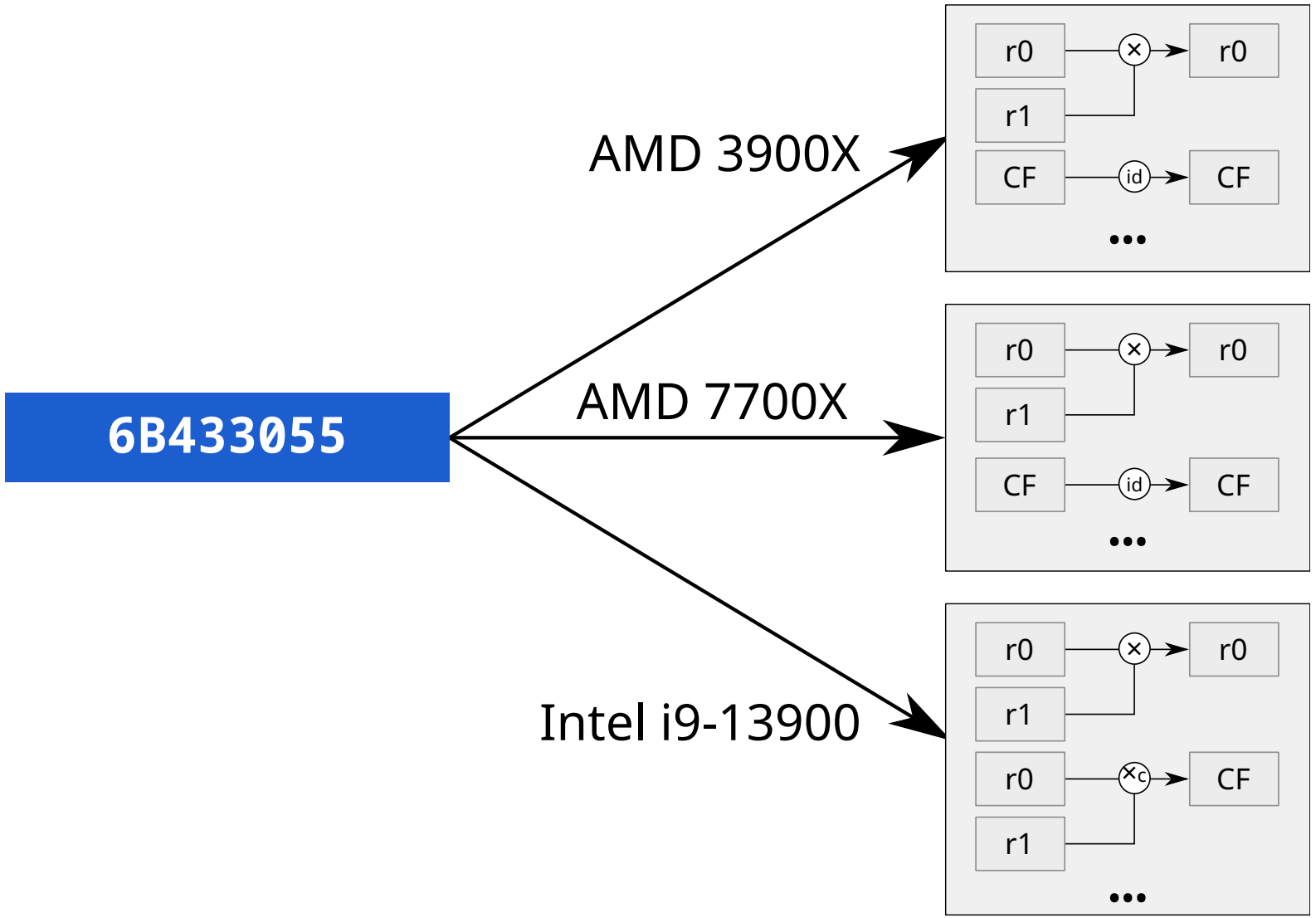
Analysis Overview



Analysis Overview



CPU-specific semantics



Analyzed CPUs

Architecture	A_0	A_1	A_2	A_3	A_4
CPU	AMD 3900X	AMD 7700X	Intel i9 13900 (p)	Intel i9 13900 (e)	Intel Xeon 4110
Cores	24T/12C	16T/8C	16T/8C	16T/16C	2x 16T/8C
Time taken	7 weeks	14 weeks	21 weeks	13 weeks	14 weeks
Encodings	118 025	118 019	117 605	118 135	117 229
Undocumented Encodings[1]	2693	3339	0	0	0
UB synthesized	91%	90%	91%	90%	90%

[1] copies of the VPERMQ* instructions with `VEX.W = 0`.

Analyzed CPUs

Architecture	A_0	A_1	A_2	A_3	A_4
CPU	AMD 3900X	AMD 7700X	Intel i9 13900 (p)	Intel i9 13900 (e)	Intel Xeon 4110
Cores	24T/12C	16T/8C	16T/8C	16T/16C	2x 16T/8C
Time taken	7 weeks	14 weeks	21 weeks	13 weeks	14 weeks
Encodings	118 025	118 019	117 605	118 135	117 229
Undocumented Encodings[1]	2693	3339	0	0	0
UB synthesized	91%	90%	91%	90%	90%

[1] copies of the VPERMQ* instructions with `VEX.W = 0`.

Analyzed CPUs

Architecture	A_0	A_1	A_2	A_3	A_4
CPU	AMD 3900X	AMD 7700X	Intel i9 13900 (p)	Intel i9 13900 (e)	Intel Xeon 4110
Cores	24T/12C	16T/8C	16T/8C	16T/16C	2x 16T/8C
Time taken	7 weeks	14 weeks	21 weeks	13 weeks	14 weeks
Encodings	118 025	118 019	117 605	118 135	117 229
Undocumented Encodings[1]	2693	3339	0	0	0
UB synthesized	91%	90%	91%	90%	90%

[1] copies of the VPERMQ* instructions with `VEX.W = 0`.

Analyzed CPUs

Architecture	A_0	A_1	A_2	A_3	A_4
CPU	AMD 3900X	AMD 7700X	Intel i9 13900 (p)	Intel i9 13900 (e)	Intel Xeon 4110
Cores	24T/12C	16T/8C	16T/8C	16T/16C	2x 16T/8C
Time taken	7 weeks	14 weeks	21 weeks	13 weeks	14 weeks
Encodings	118 025	118 019	117 605	118 135	117 229
Undocumented Encodings[1]	2693	3339	0	0	0
UB synthesized	91%	90%	91%	90%	90%

[1] copies of the VPERMQ* instructions with `VEX.W = 0`.

Comparing x86-64 implementations

Group	# 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	■	■	■	■	●
…	…	…	…	…	…	…

Comparing x86-64 implementations

Group	# 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	■	■	■	■	●
…	…	…	…	…	…	…

Comparing x86-64 implementations

Group	# 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	■	■	■	■	●
…	…	…	…	…	…	…

Comparing x86-64 implementations

Group	# 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	■	■	■	■	●
…	…	…	…	…	…	…

Comparing x86-64 implementations

Group	# 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	■	■	■	■	●
…	…	…	…	…	…	…

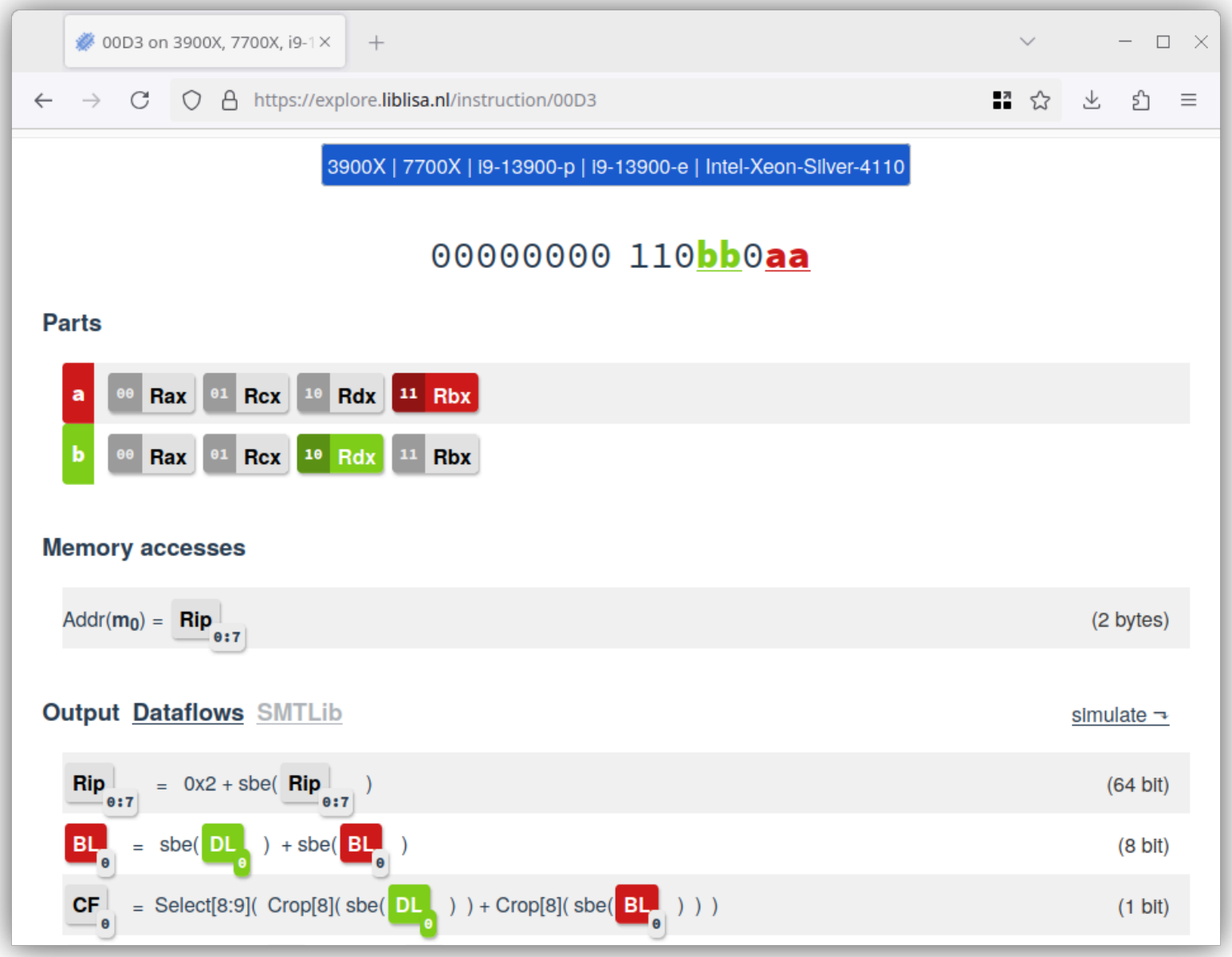
Fingerprinting CPUs

```
  xor rdx, rdx
  xor rax, rax
  add dl, 1
  mul eax
  setp r13b
  setz r14b
  lea rdi, [rip + 2f]
  xor r15b, r15b
  vpclmulhqlqdq ymm0,ymm1,ymm2
  mov r15b, 1
2:
  ...
```

Fingerprinting CPUs

```
  xor rdx, rdx
  xor rax, rax
  add dl, 1
  mul eax
  setp r13b
  setz r14b
  lea rdi, [rip + 2f]
  xor r15b, r15b
  vpclmulhqlqdq ymm0,ymm1,ymm2
  mov r15b, 1
2:
  ...
```

Semantics available on: explore.liblisa.nl



Future work

Future work: validation of existing semantics

Instruction	libLISA	Das-gupta et al.	BAP	Ghidra	IDA (?)	..?
0000000 110bb0aa	(assert (= ((_ extract 7 0) out_rRAX) ((_ extract 7 0) ((_ extract 7 0) ((_ zero_extend 120) ((_ extract 7 0) (bvadd ((_ sign_extend 120) ((_ extract 7 0) ((_ zero_extend 120) ((_ extract 7 0) rRCX)))) ((_ sign_extend 120) ((_ extract 7 0) ((_ zero_extend 120) ((_ extract 7 0) rRAX)))))))))))	(assert (= rRAX (concat ((_ extract 63 8) rRAX) ((_ extract 7 0) (bvadd (concat #b0 ((_ extract 7 0) rRCX)) (concat #b0 ((_ extract 7 0) rRAX)))))))	...	...	...	...
10111011 11bbbaaa	...	...	...	...	...	...

Validation of BAP

Comparison report

Summary

This report describes the comparison results against libLISA. Out of the **494** variants that were analyzed, errors were found in **211** variants:

ADC32mr ADC32rm ADC64mi8 ADC64rr_REV BSR32rm BSR32rr BT32mr BT64mr BTC32mr BTC64mr BTR32mr
BTR64mr BTS32mr BTS64mr CMPXCHG16B FNSTCW16m IMUL32m IMUL32r IMUL8m IMUL8r MMX_MOVD64from64rm
MMX_MOVD64mr MMX_MOVD64rm MMX_MOVD64to64rm MMX_MOVQ64mr MMX_MOVQ64rm MMX_MOVQ64rr_REV MMX_PADDBirm
MMX_PADDDirm MMX_PADDWirm MMX_PADDWirr MMX_PALIGNRrmi MMX_PALIGNRrri MMX_PANDNirm MMX_PANDirm
MMX_PAVGBirm MMX_PAVGBirr MMX_PAVGWirm MMX_PCMPEQBirm MMX_PCMPEQDirm MMX_PCMPEQWirm MMX_PCMPGTBirm
MMX_PCMPGTDirm MMX_PCMPGTDirr MMX_PMAXSWirm MMX_PMAXUBirm MMX_PMINSWirm MMX_PMINSWirr MMX_PMINUBirm
MMX_PORirm MMX_PSLLDri MMX_PSLLDrm MMX_PSLLQri MMX_PSLLQrm MMX_PSLLQrr MMX_PSLLWri MMX_PSLLWrm
MMX_PSLLWrr MMX_PSRADrm MMX_PSRLDri MMX_PSRLDrm MMX_PSRLQri MMX_PSRLQrm MMX_PSRLWrm MMX_PSRLWrr
MMX_PSUBBirm MMX_PSUBQirm MMX_PSUBWirm MMX_PUNPCKHBWirm MMX_PUNPCKHBWirr MMX_PUNPCKHDQirm
MMX_PUNPCKHWDirm MMX_PUNPCKLBWirm MMX_PUNPCKLDQirm MMX_PUNPCKLWDirm MMX_PXORirm MOV16ms MOV32rs
NEG8m NOT32m POP64rmm ROL32m1 ROL32mCL ROL32mi ROL32rCL ROL64m1 ROL64mi ROL8mCL ROL8mi ROL8ri
ROR32m1 ROR32mCL ROR32mi ROR64mi ROR8m1 ROR8mi ROR8r1 ROR8ri SAR64rCL SAR8mi SAR8rCL SBB32mr
SBB32rm SHL8mi SHLD32mri8 SHLD32rri8 SHLD64mri8 SHLD64rri8 SHR8mi SHRD32mri8 SHRD32rri8
SHRD64mri8 SHRD64rri8 VMOVDDUPrm VMOVSHDUPYrm VMOVSHDUPrm VMOVSLDUPYrm VMOVSLDUPrm VPADDBYrm
VPADDBrm VPADDDYrm VPADDQYrm VPADDQrm VPADDQrr VPADDWYrm VPADDWrm VPALIGNRYrri VPALIGNRrri
VPAVGBrm VPAVGBrr VPAVGWYrm VPMAXSWYrm VPMAXUBYrm VPMAXUBYrr VPMAXUBrm VPMAXUDrm VPMAXUDrr
VPMAXUWYrm VPMINSDYrm VPMINSWYrm VPMINSWYrr VPMINSWrm VPMINUBYrr VPMINUBrm VPMINUBrr VPMINUDYrm
VPMINUDrm VPMINUWYrm VPMINUWrm VPSHUFDYmi VPSHUFDYri VPSHUFDmi VPSHUFDri VPSLLDQYri VPSLLDYri
VPSLLDri VPSLLDrm VPSLLQYri VPSLLQYrm VPSLLQri VPSLLQrm VPSLLWYrm VPSLLWrm VPSRADYrm VPSRADrm
VPSRADrr VPSRAWYrm VPSRAWrm VPSRLDQYri VPSRLDYri VPSRLDYrm VPSRLDYrr VPSRLDri VPSRLDrm VPSRLQYri
VPSRLQYrm VPSRLQri VPSRLQrm VPSRLQrr VPSRLWYrm VPSRLWri VPSRLWrm VPSUBBYrm VPSUBBYrr VPSUBBrm
VPSUBDrm VPSUBQYrm VPSUBQrm VPSUBWYrm VPUNPCKHBWrm VPUNPCKHDQYrm VPUNPCKHDQrm VPUNPCKHQDQYrm
VPUNPCKHQDQrm VPUNPCKHWDYrm VPUNPCKHWDrm VPUNPCKHWDrr VPUNPCKLBWYrm VPUNPCKLBWrm VPUNPCKLDQYrm
VPUNPCKLDQrm VPUNPCKLDQrr VPUNPCKLQDQYrm VPUNPCKLWDYrm VPUNPCKLWDrm VXORPDrm VXORPSYrm VXORPSrm
XADD32rm XADD64rm XCHG32rm

Comparison report

Summary

. Out of the **494** variants that were analyzed, errors were found in **211** variants:

MMX_MOVD64mr MMX_MOVD64rm MMX_MOVD64to64rm MMX_MOVQ64mr MMX_MOVQ64rm MMX_MOVQ64rr_REV MMX_PADDBirm
MMX_PADDDirm MMX_PADDWirm MMX_PADDWirr MMX_PALIGNRrmi MMX_PALIGNRrri MMX_PANDNirm MMX_PANDirm
MMX_PAVGBirm MMX_PAVGBirr MMX_PAVGWirm MMX_PCMPEQBirm MMX_PCMPEQDirm MMX_PCMPEQWirm MMX_PCMPGTBirm
MMX_PCMPGTDirm MMX_PCMPGTDirr MMX_PMAXSWirm MMX_PMAXUBirm MMX_PMINSWirm MMX_PMINSWirr MMX_PMINUBirm
MMX_PORirm MMX_PSLLDri MMX_PSLLDrm MMX_PSLLQri MMX_PSLLQrm MMX_PSLLQrr MMX_PSLLWri MMX_PSLLWrm
MMX_PSLLWrr MMX_PSRADrm MMX_PSRLDri MMX_PSRLDrm MMX_PSRLQri MMX_PSRLQrm MMX_PSRLWrm MMX_PSRLWrr
MMX_PSUBBirm MMX_PSUBQirm MMX_PSUBWirm MMX_PUNPCKHBWirm MMX_PUNPCKHBWirr MMX_PUNPCKHDQirm
MMX_PUNPCKHWDirm MMX_PUNPCKLBWirm MMX_PUNPCKLDQirm MMX_PUNPCKLWDirm MMX_PXORirm MOV16ms MOV32rs
NEG8m NOT32m POP64rmm ROL32m1 ROL32mCL ROL32mi ROL32rCL ROL64m1 ROL64mi ROL8mCL ROL8mi ROL8ri
ROR32m1 ROR32mCL ROR32mi ROR64mi ROR8m1 ROR8mi ROR8r1 ROR8ri SAR64rCL SAR8mi SAR8rCL SBB32mr
SBB32rm SHL8mi SHLD32mri8 SHLD32rri8 SHLD64mri8 SHLD64rri8 SHR8mi SHRD32mri8 SHRD32rri8
SHRD64mri8 SHRD64rri8 VMOVDDUPrm VMOVSHDUPYrm VMOVSHDUPrm VMOVSLDUPYrm VMOVSLDUPrm VPADDBYrm
VPADDBrm VPADDDYrm VPADDQYrm VPADDQrm VPADDQrr VPADDWYrm VPADDWrm VPALIGNRYrri VPALIGNRrri
VPAVGBrm VPAVGBrr VPAVGWYrm VPMAXSWYrm VPMAXUBYrm VPMAXUBYrr VPMAXUBrm VPMAXUDrm VPMAXUDrr
VPMAXUWYrm VPMINSDYrm VPMINSWYrm VPMINSWYrr VPMINSWrm VPMINUBYrr VPMINUBrm VPMINUBrr VPMINUDYrm
VPMINUDrm VPMINUWYrm VPMINUWrm VPSHUFDYmi VPSHUFDYri VPSHUFDmi VPSHUFDri VPSLLDQYri VPSLLDYri
VPSLLDri VPSLLDrm VPSLLQYri VPSLLQYrm VPSLLQri VPSLLQrm VPSLLWYrm VPSLLWrm VPSRADYrm VPSRADrm
VPSRADrr VPSRAWYrm VPSRAWrm VPSRLDQYri VPSRLDYri VPSRLDYrm VPSRLDYrr VPSRLDri VPSRLDrm VPSRLQYri
VPSRLQYrm VPSRLQri VPSRLQrm VPSRLQrr VPSRLWYrm VPSRLWri VPSRLWrm VPSUBBYrm VPSUBBYrr VPSUBBrm
VPSUBDrm VPSUBQYrm VPSUBQrm VPSUBWYrm VPUNPCKHBWrm VPUNPCKHDQYrm VPUNPCKHDQrm VPUNPCKHQDQYrm
VPUNPCKHQDQrm VPUNPCKHWDYrm VPUNPCKHWDrm VPUNPCKHWDrr VPUNPCKLBWYrm VPUNPCKLBWrm VPUNPCKLDQYrm
VPUNPCKLDQrm VPUNPCKLDQrr VPUNPCKLQDQYrm VPUNPCKLWDYrm VPUNPCKLWDrm VXORPDrm VXORPSYrm VXORPSrm
XADD32rm XADD64rm XCHG32rm

Variant `VPADDQYrm`

Instruction `C55DD45941` **decoded as** `VPADDQYrm(YMM11,YMM4,RCX,0x1,Nil,0x41,Nil)`

▼ The location **Xmm4** should not be modified.

Input	Value
RCX	`00 00 00 00 00 00 00 00`
Xmm4	`ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff`
Xmm11	`00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`
m_1	`00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`

Output	Value	
Xmm4	**Expected:**	`ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff`
	Found:	`00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`

► The location **Xmm11** should be updated.

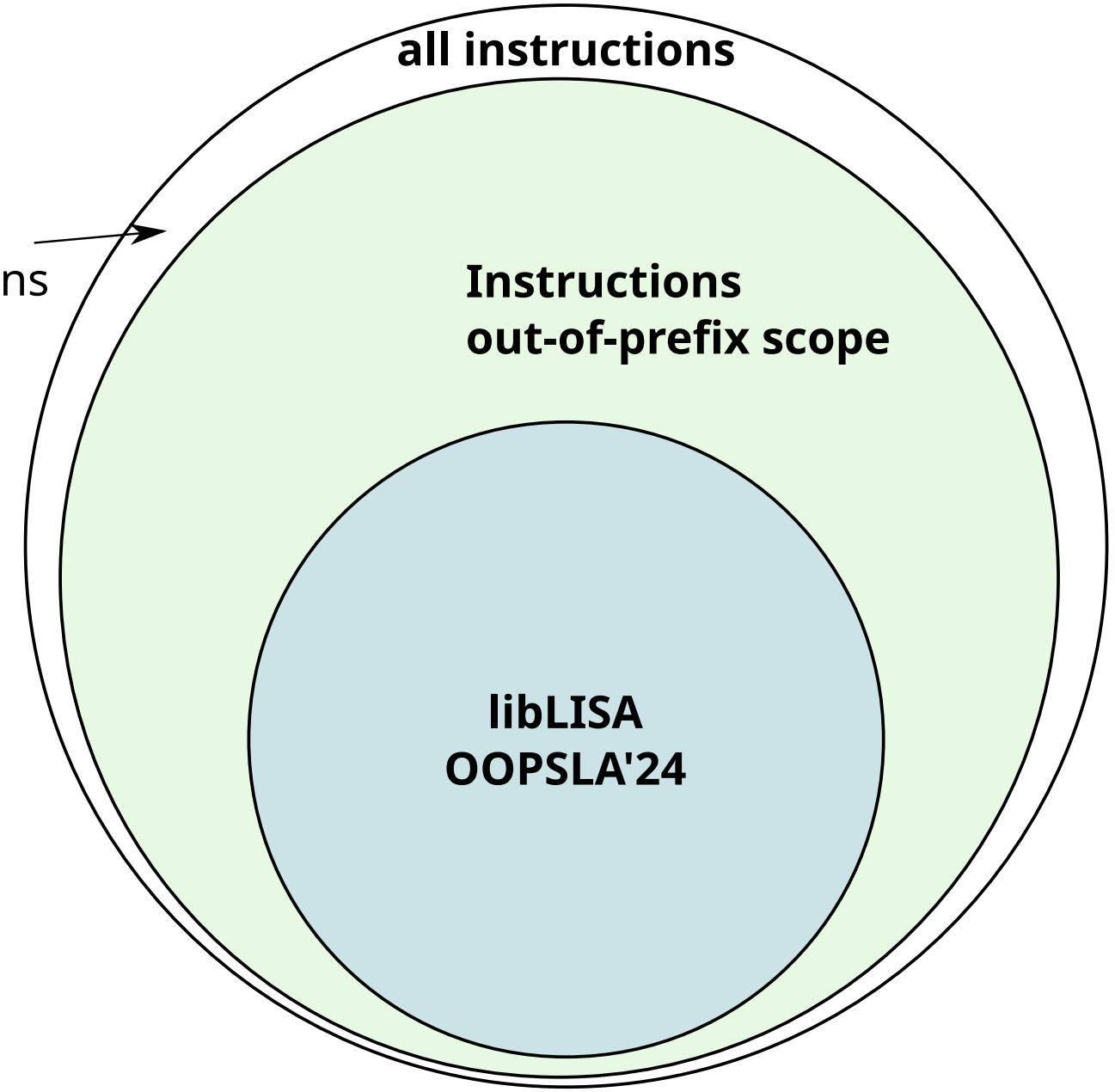
Instruction `C4E13DD4ACB3F44309F2` **decoded as** `VPADDQYrm(YMM5,YMM8,RBX,0x4,RSI,-0xdf6bc0c,Nil)`

► The location **Xmm5** should be updated.

► The location **Xmm8** should not be modified.

Future work: prefixes

Privileged instructions
Hardware instructions
Segment selector instructions
(rdtsc, xsave, out, lss, lsl, ...)



Future work: prefixes

0303 = add eax, dword ptr [rbx]

480303 = add rax, qword ptr [rbx]

660303 = add ax, word ptr [rbx]

Future work: prefixes

0303 = add eax, dword ptr [rbx]

480303 = add rax, qword ptr [rbx]

660303 = add ax, word ptr [rbx]

48486648F2F348662E3E484866**6**0303 = add ax, word ptr [rbx]

Summary

- automatically infers semantics
- analyzed 5 architectures
- semantics and code are open source