

# Get to Know Your Decompiler

0x41con 2023

@jonpalmisc

# About me

@jonpalmisc

- Florida man
- Long-time programmer
- Low-level & security enjoyer
- Present: iOS stuff at <private>
- Past: Binary Ninja developer at Vector 35 for 2 years

# History of decompilers

A concise, biased perspective

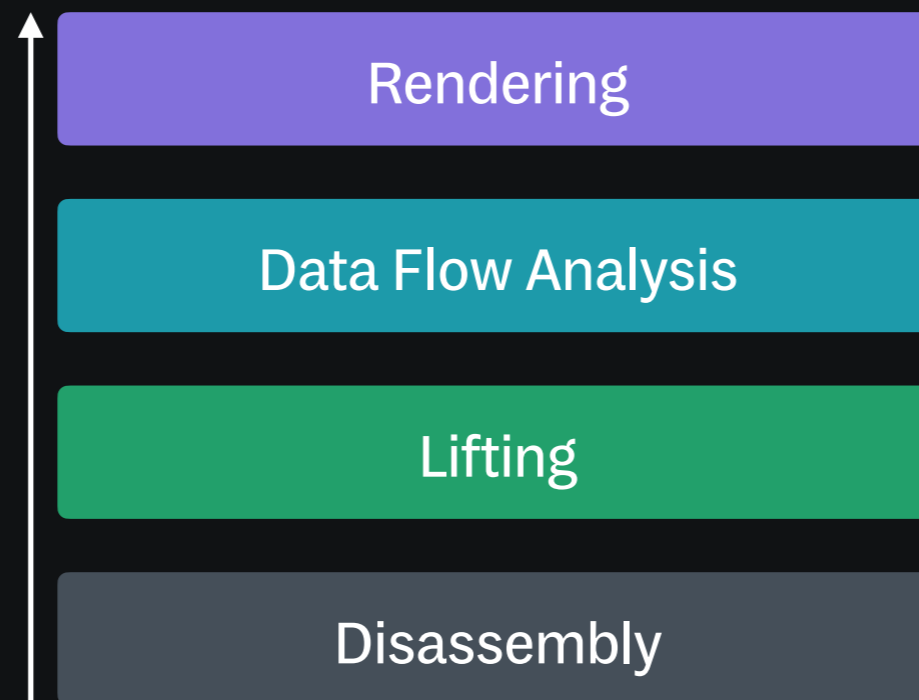
- 1994: “Reverse Compilation Techniques”
- 1995–2006: The Dark Ages
- 2007: Hex-Rays Decompiler
- 2019: Ghidra
- 2020: Binary Ninja HLIL

# Decompilation stages

Overview

# Decompilation stages

## Overview



# Decompilation stages

## Disassembly

- Natural prerequisite for decompilation
- More than just producing text

2801080b

# Decompilation stages

## Disassembly

- Natural prerequisite for decompilation
- More than just producing text

add w8, w9, w8

2801080b



# Decompilation stages

## Lifting

- Start abstracting machine code
- Need some intermediate representation
  - Numerous different approaches exist

add w8, w9, w8

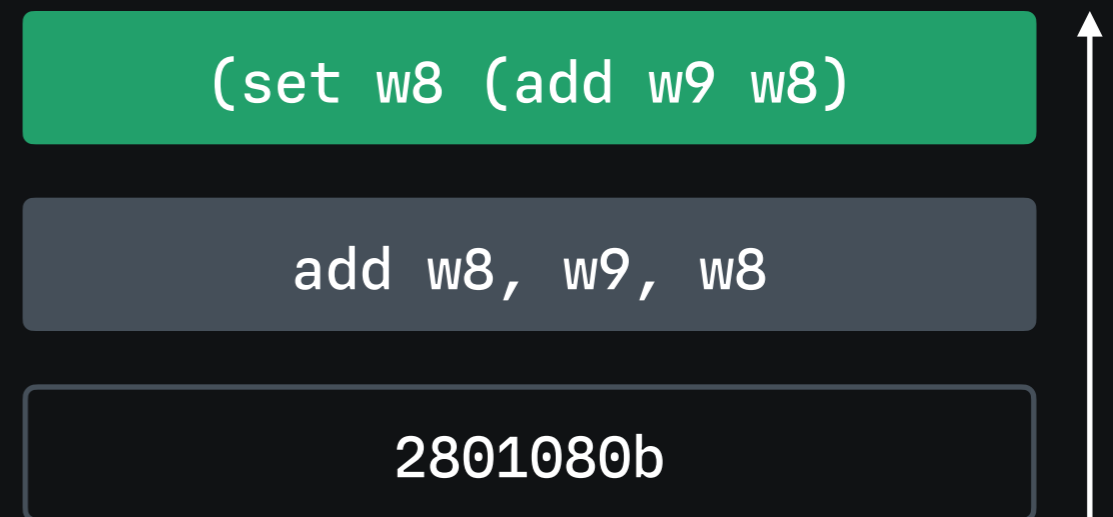
2801080b



# Decompilation stages

## Lifting

- Start abstracting machine code
- Need some intermediate representation
  - Numerous different approaches exist



# Decompilation stages

## Data flow analysis

- Where are values defined and used?
- What are the possible values of X at address Y?
- Inherent part of decompilation internally

```
(set w8 (add w9 w8))
```

```
add w8, w9, w8
```

```
2801080b
```

# Decompilation stages

## Data flow analysis

- Where are values defined and used?
- What are the possible values of X at address Y?
- Inherent part of decompilation internally

```
(set w8#2 (add w9#1 w8#1))
```

```
(set w8 (add w9 w8))
```

```
add w8, w9, w8
```

```
2801080b
```

# Decompilation stages

## Rendering

- Need to represent the abstracted code somehow
  - C-flavored syntax has become the norm
- Not “trivial”, but less difficult

```
(set w8#2 (add w9#1 w8#1))
```

```
(set w8 (add w9 w8))
```

```
add w8, w9, w8
```

```
2801080b
```

# Decompilation stages

## Rendering

- Need to represent the abstracted code somehow
  - C-flavored syntax has become the norm
- Not “trivial”, but less difficult

```
int v1 = w8 + w9;
```

```
(set w8#2 (add w9#1 w8#1))
```

```
(set w8 (add w9 w8))
```

```
add w8, w9, w8
```

```
2801080b
```

# IDA's microcode

## Overview

- Machine code transformed into “microinstructions”
  - 72 unique operations
  - Single-purpose, simple in nature
  - No side effects
- Inspired by ideas from “Reverse Compilation Techniques”

# IDA's microcode

## Microinstruction anatomy

Microinstruction

```
add x0.8, t2.8, t2.8
```

# IDA's microcode

## Microinstruction anatomy

Microinstruction

```
add x0.8, t2.8, t2.8
```

Operation

t2.8

=

add

x0.8

t2.8

# IDA's microcode

A basic example

AArch64 assembly

```
ldr x0, [x0, #8]
```

```
ret
```

# IDA's microcode

A basic example

AArch64 assembly

```
ldr x0, [x0, #8]
```



# IDA's microcode

## A basic example

AArch64 assembly

```
ldr x0, [x0, #8]
```



Generated microcode

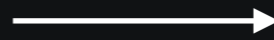
```
mov #8.8, t2.8
```

# IDA's microcode

## A basic example

AArch64 assembly

```
ldr x0, [x0, #8]
```



Generated microcode

```
mov #8.8, t2.8
```

```
add x0.8, t2.8, t2.8
```

# IDA's microcode

## A basic example

AArch64 assembly

```
ldr x0, [x0, #8]
```



Generated microcode

```
mov #8.8, t2.8
```

```
add x0.8, t2.8, t2.8
```

```
ldx t2.8, t1.8
```

# IDA's microcode

## A basic example

AArch64 assembly

```
ldr x0, [x0, #8]
```



Generated microcode

```
mov #8.8, t2.8
```

```
add x0.8, t2.8, t2.8
```

```
ldx t2.8, t1.8
```

```
mov t1.8, x0.8
```

# IDA's microcode

## Microcode lifecycle

- Unoptimized, initial lift (Level 1)
- Easy wins (Level 2)
- Local optimizations, function calls (Levels 3, 4)
- Global optimizations (Levels 5–7)
- Local variable analysis (Level 8)

# IDA's microcode

Optimizations visualized

AArch64 assembly

```
ldr x0, [x0, #8]
```



Generated microcode

```
mov #8.8, t2.8
```

```
add x0.8, t2.8, t2.8
```

```
ldx t2.8, t1.8
```

```
mov t1.8, x0.8
```

# IDA's microcode

Optimizations visualized

AArch64 assembly

```
ldr x0, [x0, #8]
```



```
ldx (x0.8 + #8.8), t1.8
```

```
mov t1.8, x0.8
```

# IDA's microcode

Optimizations visualized

AArch64 assembly

```
ldr x0, [x0, #8]
```



Optimized microcode

```
ldx (x0.8 + #8.8), x0.8
```

# IDA's microcode

Optimizations visualized

AArch64 assembly

```
ldr x0, [x0, #8]
```

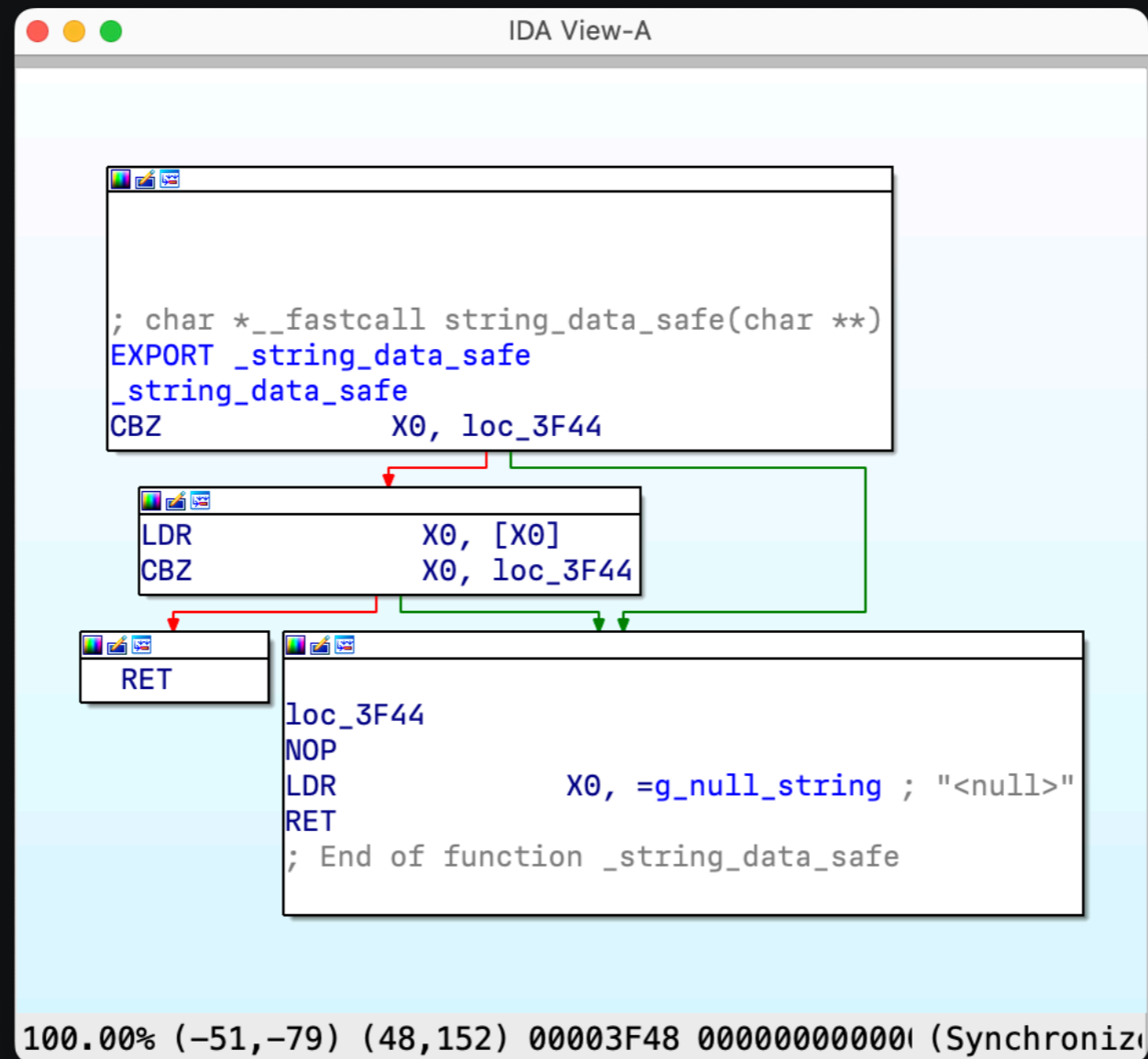


Optimized microcode with local variables

```
ldx (a1.8 + #8.8), result.8
```

# IDA's microcode

A real-world example



# IDA's microcode

## A real-world example

The image shows two windows from the IDA Pro interface. The left window, titled "IDA View-A", displays assembly code for a function named `_string_data_safe`. The code includes a `CBZ` instruction that branches to `loc_3F44` if the register `X0` is zero. Below this, there is a block of code for `loc_3F44` containing `NOP`, `LDR`, and `RET` instructions. A `RET` instruction is also shown in a separate block. The right window, titled "Microcode Explorer", displays the microcode generated for the assembly code. It shows four blocks of microcode, each corresponding to a branch or instruction in the assembly code. The microcode instructions are in a simplified format, such as `mov #0x3F44.8, t2.8` and `setz x0.8, #0.8, t0.1`. The right window also has a "Settings" panel with options like "Highlight mutual", "Show use/def", and "Sync hexrays".

```
IDA View-A
; char *__fastcall string_data_safe(char *
EXPORT _string_data_safe
_string_data_safe
CBZ          X0, loc_3F44

LDR          X0, [X0]
CBZ          X0, loc_3F44

RET

loc_3F44
NOP
LDR          X0, =g_null_stri
RET
; End of function _string_data_s

Microcode Explorer
; ????-BLOCK 0 PROP FAKE [START=00003F34 END=
; ????-BLOCK 1 PROP [START=00003F34 END=00003
1. 0 mov     #0x3F44.8, t2.8
1. 1 setz    x0.8, #0.8, t0.1
1. 2 jcnd   t0.1, $loc_3F44

; ????-BLOCK 2 PROP [START=00003F38 END=00003
2. 0 mov     #0.8, t2.8
2. 1 add     x0.8, t2.8, t2.8
2. 2 ldx     cs.2, t2.8, t1.8
2. 3 mov     t1.8, x0.8
2. 4 mov     #0x3F44.8, t2.8
2. 5 setz    x0.8, #0.8, t0.1
2. 6 jcnd   t0.1, $loc_3F44

; ????-BLOCK 3 PROP [START=00003F40 END=00003
3. 0 ret

; ????-BLOCK 4 PROP [START=00003F44 END=00003
4. 0 mov     #0x4000.8, t2.8
4. 1 ldx     cs.2, t2.8, t1.8
4. 2 mov     t1.8, x0.8
4. 3 ret

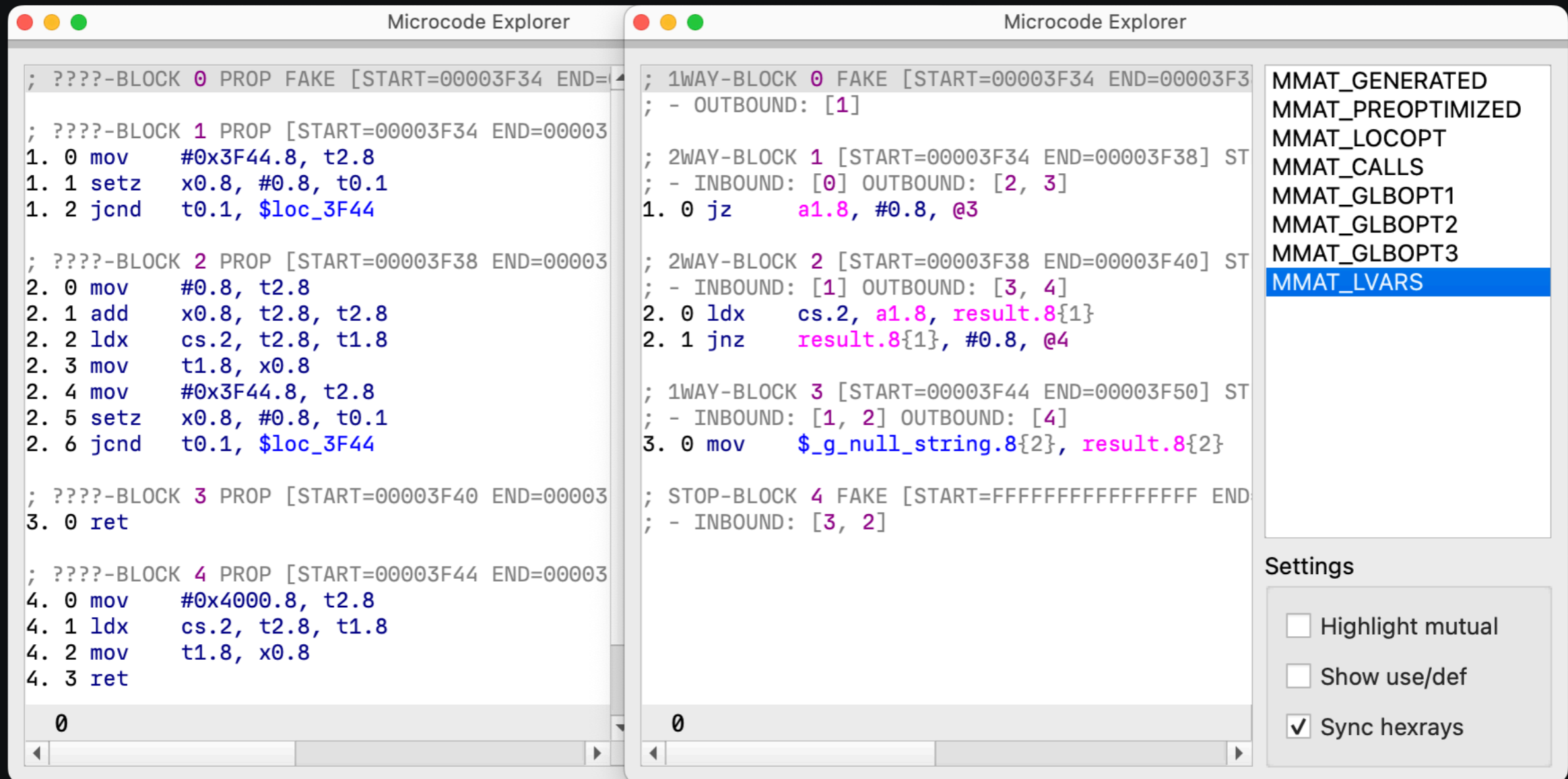
0

Settings
 Highlight mutual
 Show use/def
 Sync hexrays
```

Microcode shown with the "Lucid" plugin by Markus Gassedelen. (<https://github.com/gaasedelen/lucid>)

# IDA's microcode

## A real-world example



The image displays two side-by-side windows of the Microcode Explorer tool. The left window shows a list of microcode blocks with their instructions. The right window shows a similar list but with MMAT options visible on the right side. The MMAT options include MMAT\_GENERATED, MMAT\_PREOPTIMIZED, MMAT\_LOCOPT, MMAT\_CALLS, MMAT\_GLBOPT1, MMAT\_GLBOPT2, MMAT\_GLBOPT3, and MMAT\_LVARs (highlighted in blue). Below the MMAT options is a 'Settings' section with three checkboxes: 'Highlight mutual' (unchecked), 'Show use/def' (unchecked), and 'Sync hexrays' (checked).

```
; ????-BLOCK 0 PROP FAKE [START=00003F34 END=00003F34] ST
; - INBOUND: [0] OUTBOUND: [0]
; ????-BLOCK 1 PROP [START=00003F34 END=00003F38] ST
1. 0 mov #0x3F44.8, t2.8
1. 1 setz x0.8, #0.8, t0.1
1. 2 jcnd t0.1, $loc_3F44
; ????-BLOCK 2 PROP [START=00003F38 END=00003F40] ST
2. 0 mov #0.8, t2.8
2. 1 add x0.8, t2.8, t2.8
2. 2 ldx cs.2, t2.8, t1.8
2. 3 mov t1.8, x0.8
2. 4 mov #0x3F44.8, t2.8
2. 5 setz x0.8, #0.8, t0.1
2. 6 jcnd t0.1, $loc_3F44
; ????-BLOCK 3 PROP [START=00003F40 END=00003F44] ST
3. 0 ret
; ????-BLOCK 4 PROP [START=00003F44 END=00003F48] ST
4. 0 mov #0x4000.8, t2.8
4. 1 ldx cs.2, t2.8, t1.8
4. 2 mov t1.8, x0.8
4. 3 ret
0
```

```
; 1WAY-BLOCK 0 FAKE [START=00003F34 END=00003F34] ST
; - OUTBOUND: [1]
; 2WAY-BLOCK 1 [START=00003F34 END=00003F38] ST
; - INBOUND: [0] OUTBOUND: [2, 3]
1. 0 jz a1.8, #0.8, @3
; 2WAY-BLOCK 2 [START=00003F38 END=00003F40] ST
; - INBOUND: [1] OUTBOUND: [3, 4]
2. 0 ldx cs.2, a1.8, result.8{1}
2. 1 jnz result.8{1}, #0.8, @4
; 1WAY-BLOCK 3 [START=00003F44 END=00003F50] ST
; - INBOUND: [1, 2] OUTBOUND: [4]
3. 0 mov $_g_null_string.8{2}, result.8{2}
; STOP-BLOCK 4 FAKE [START=FFFFFFFFFFFFFFFF END=FFFFFFFFFFFFFFFF] ST
; - INBOUND: [3, 2]
0
```

MMAT\_GENERATED  
MMAT\_PREOPTIMIZED  
MMAT\_LOCOPT  
MMAT\_CALLS  
MMAT\_GLBOPT1  
MMAT\_GLBOPT2  
MMAT\_GLBOPT3  
MMAT\_LVARs

Settings

- Highlight mutual
- Show use/def
- Sync hexrays

Microcode shown with the "Lucid" plugin by Markus Gassedelen. (<https://github.com/gaasedelen/lucid>)

# Ghidra's p-code

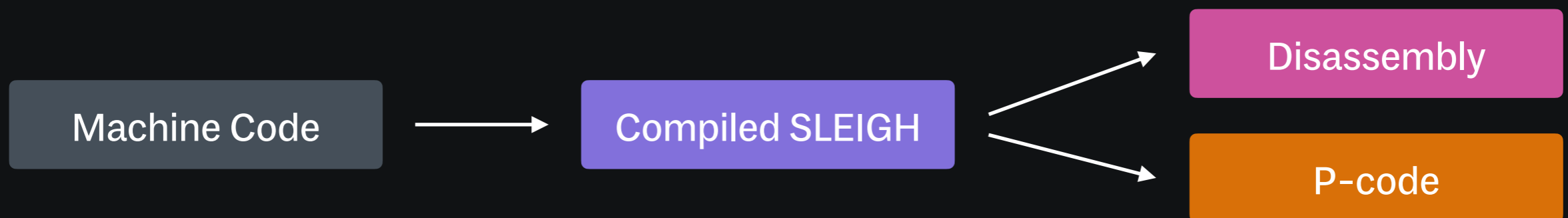
## Overview

- Self-described register transfer language
- Similar to IDA's microcode
  - 62 unique operations
  - Mostly free of side-effects
  - etc...

# Ghidra's p-code

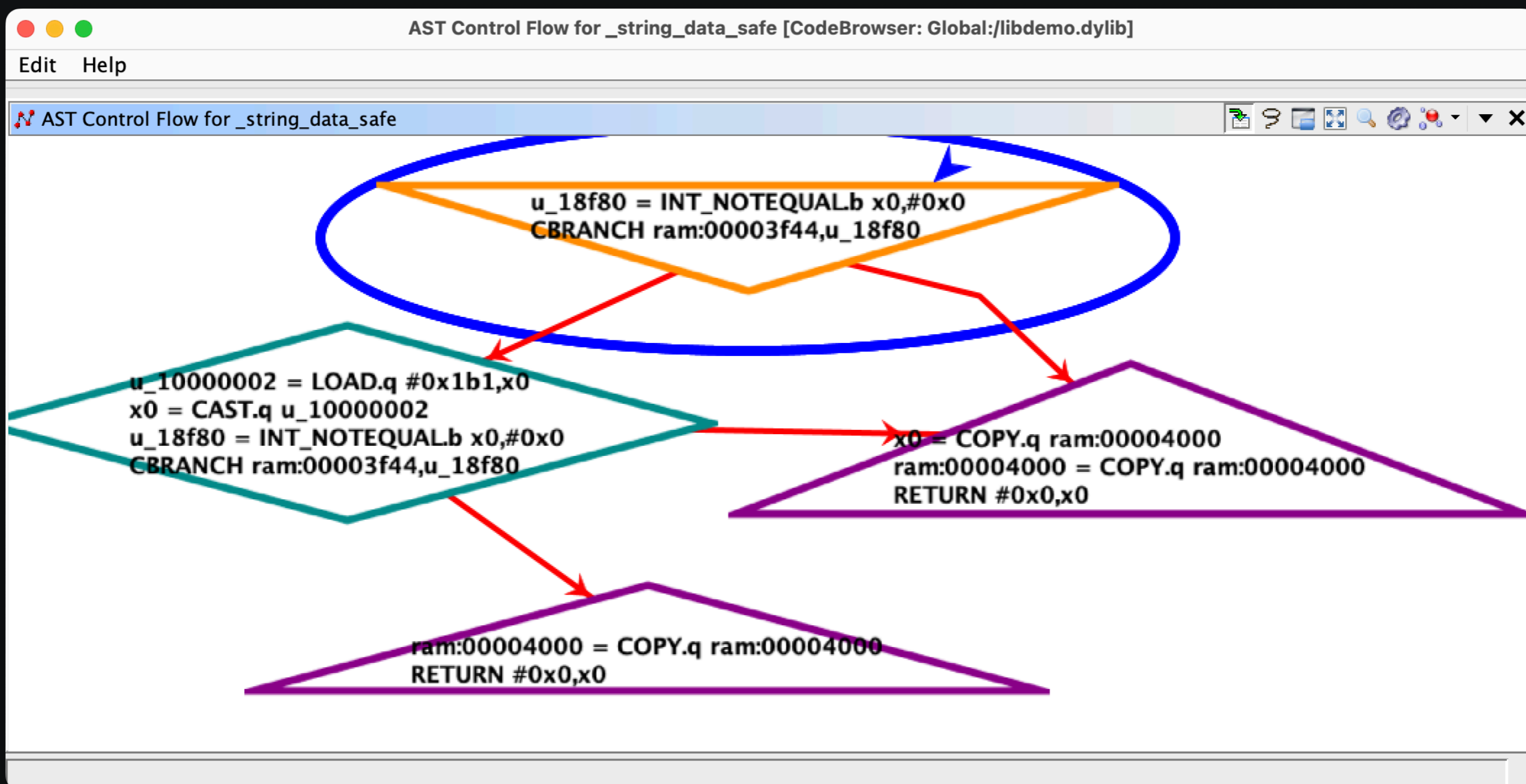
## SLEIGH

- Expressive language for describing ISAs
  - Does both disassembly and lifting
- Inspired by SLED (AT&T Labs, 1997)
- Lots of moving parts



# Ghidra's p-code

A real-world example



# Binary Ninja's ILs

## Overview

- Stack of three intermediate languages (ILs)
  - Low-level IL (LLIL)
  - Medium-level IL (MLIL)
  - High-level IL (HLIL)
- Tree/expression-based



# Binary Ninja's ILs

## Low-level IL

- 108 unique operations
- Most primitive user-facing IL, lifted from machine code
  - Internally, “Lifted IL” technically comes first
- No concept of memory
- Resolved flags & stack pointer

# Binary Ninja's ILs

## Medium-level IL

- Much more expressive than LLIL
  - Introduces variables, types, calls/parameters
- Acts as the medium for lots of important analysis
  - Type propagation
  - Data flow & value set analysis
  - Other important things

# Binary Ninja's ILs

## High-level IL

- Looks closer to C/pseudocode
- Refined through numerous simplification passes
- Introduces semantic control flow
- Adds variable merging
- Dead code elimination
- Etc...

# Binary Ninja's ILs

## Anatomy of an IL instruction

- Text form
- Tree form
  - Operation
  - Variation of operands
    - Source & destination
    - Left & right
    - Etc...

# Binary Ninja's ILs

A basic example

AArch64 assembly

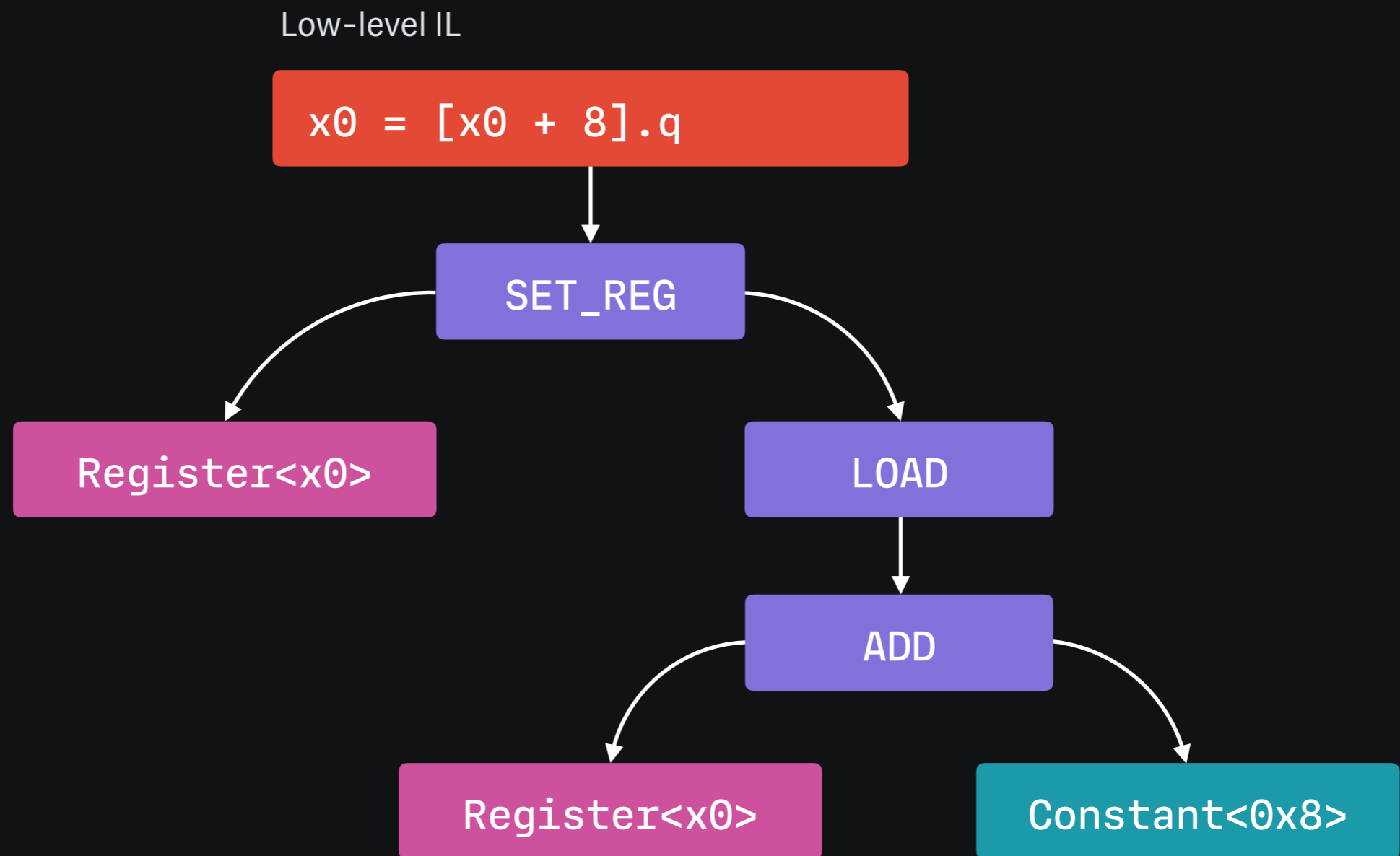
```
ldr x0, [x0, #8]
```

Low-level IL (text-form)

```
x0 = [x0 + 8].q
```

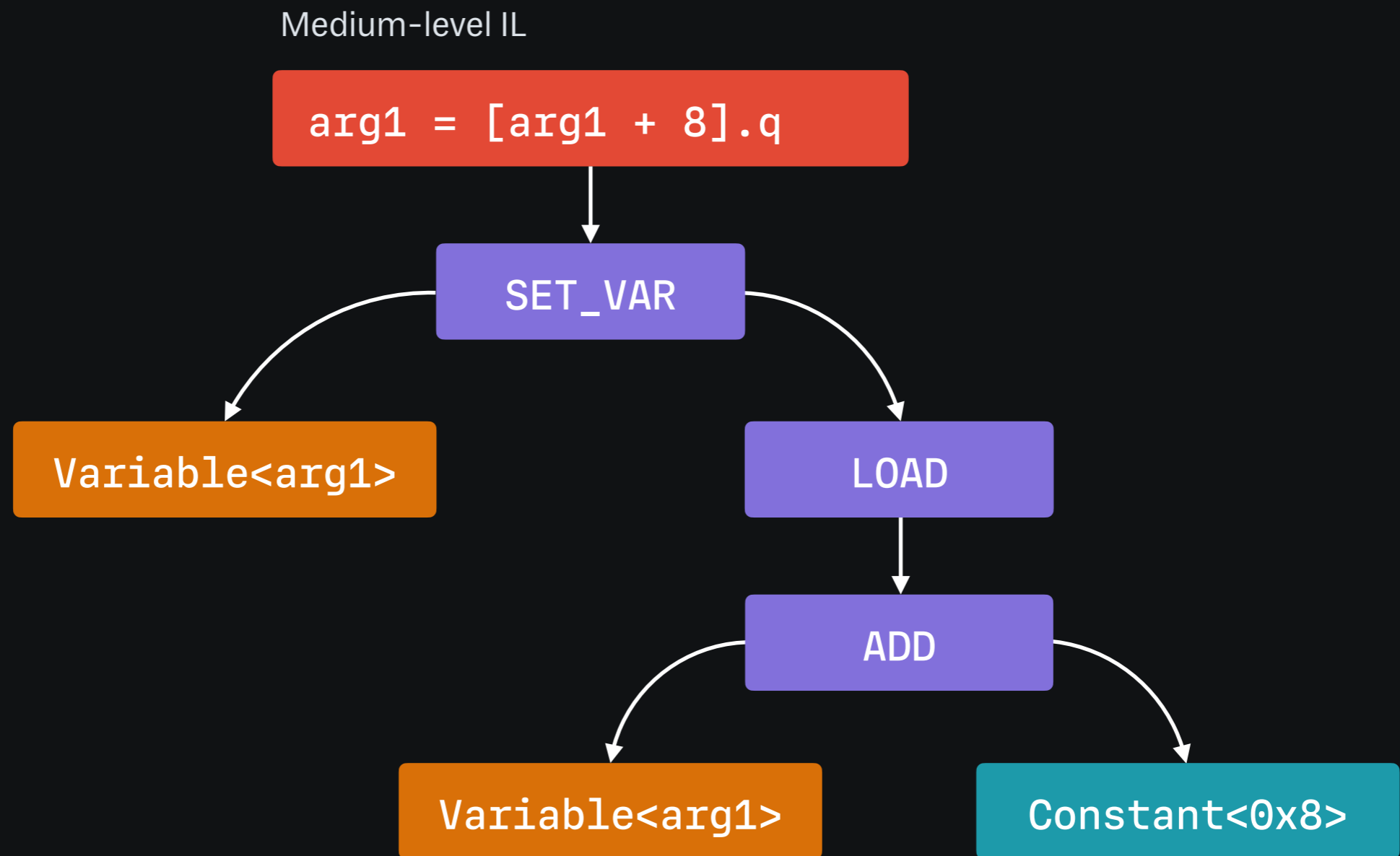
# Binary Ninja's ILs

A basic example



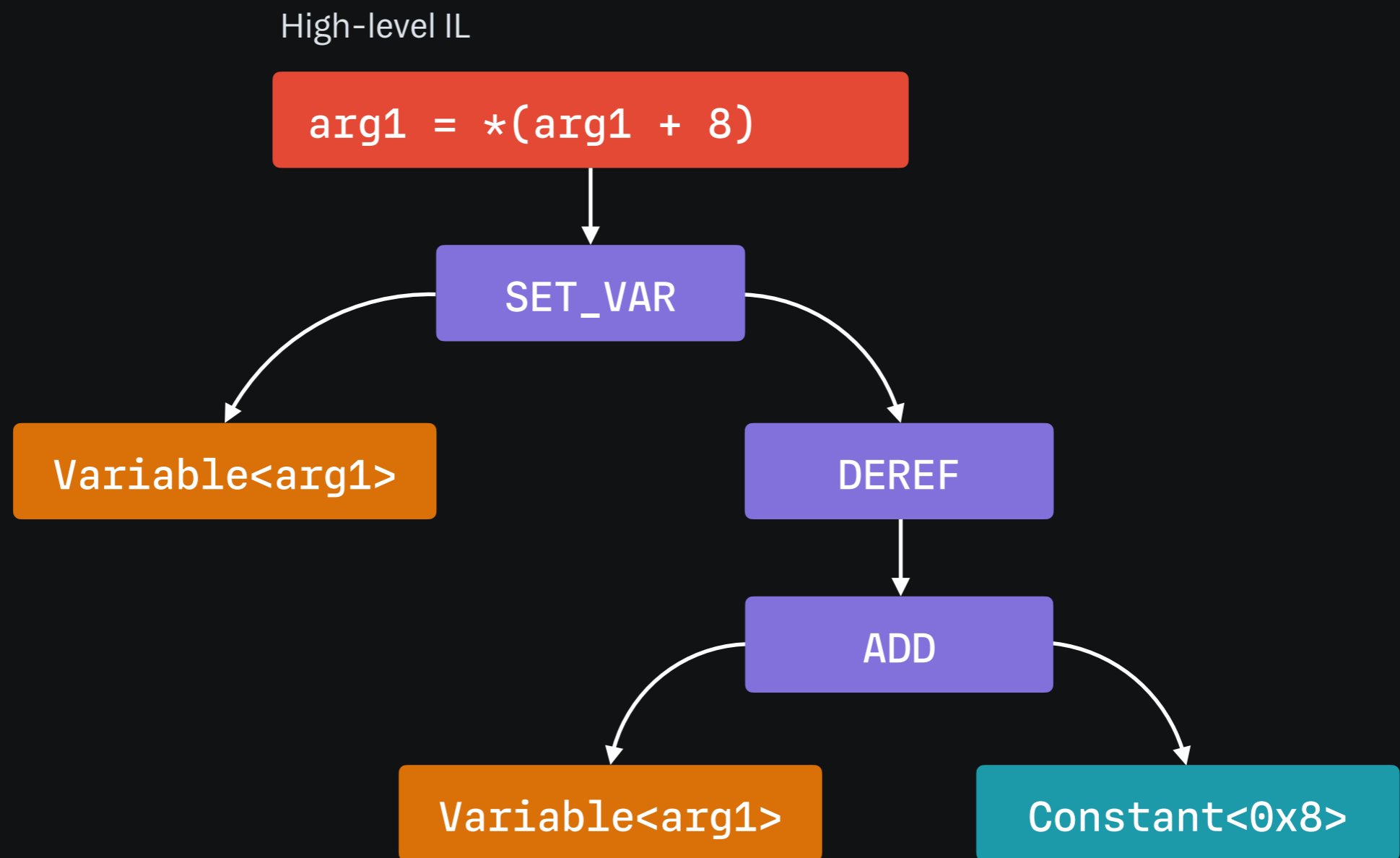
# Binary Ninja's ILs

A basic example



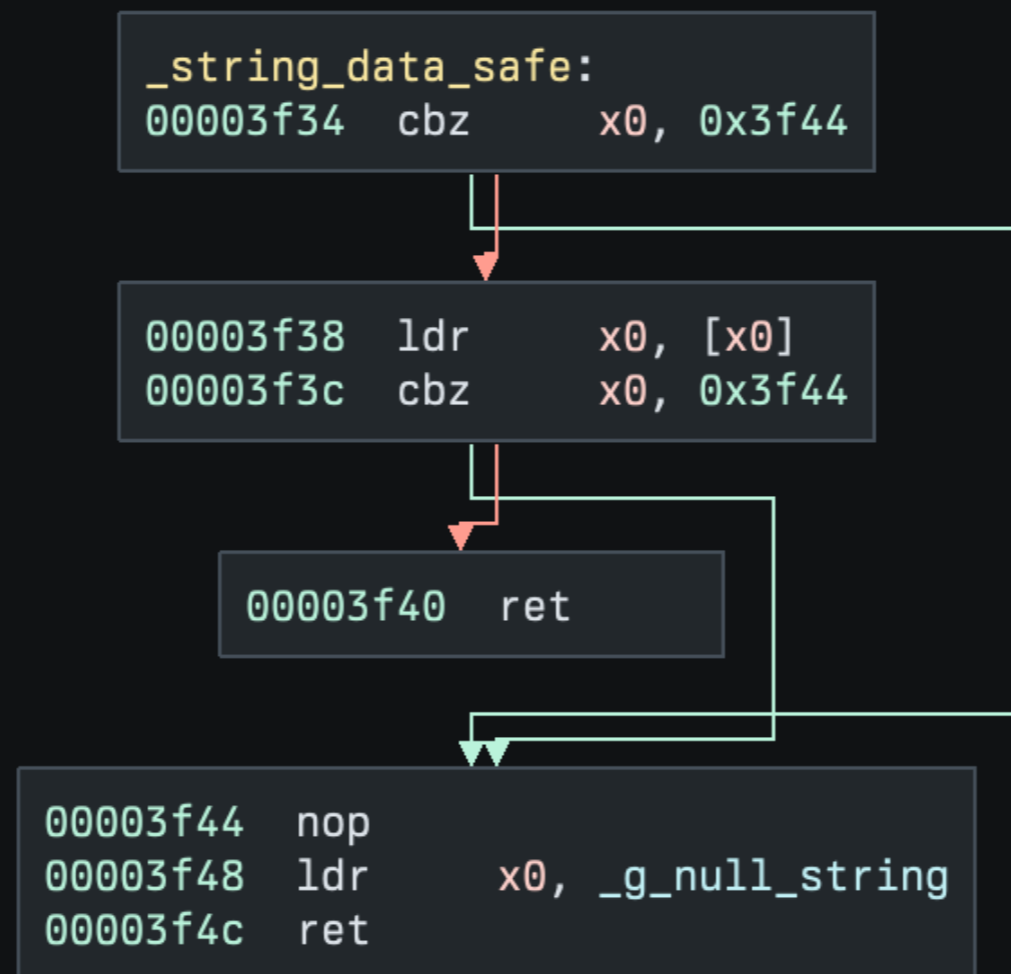
# Binary Ninja's ILs

A basic example



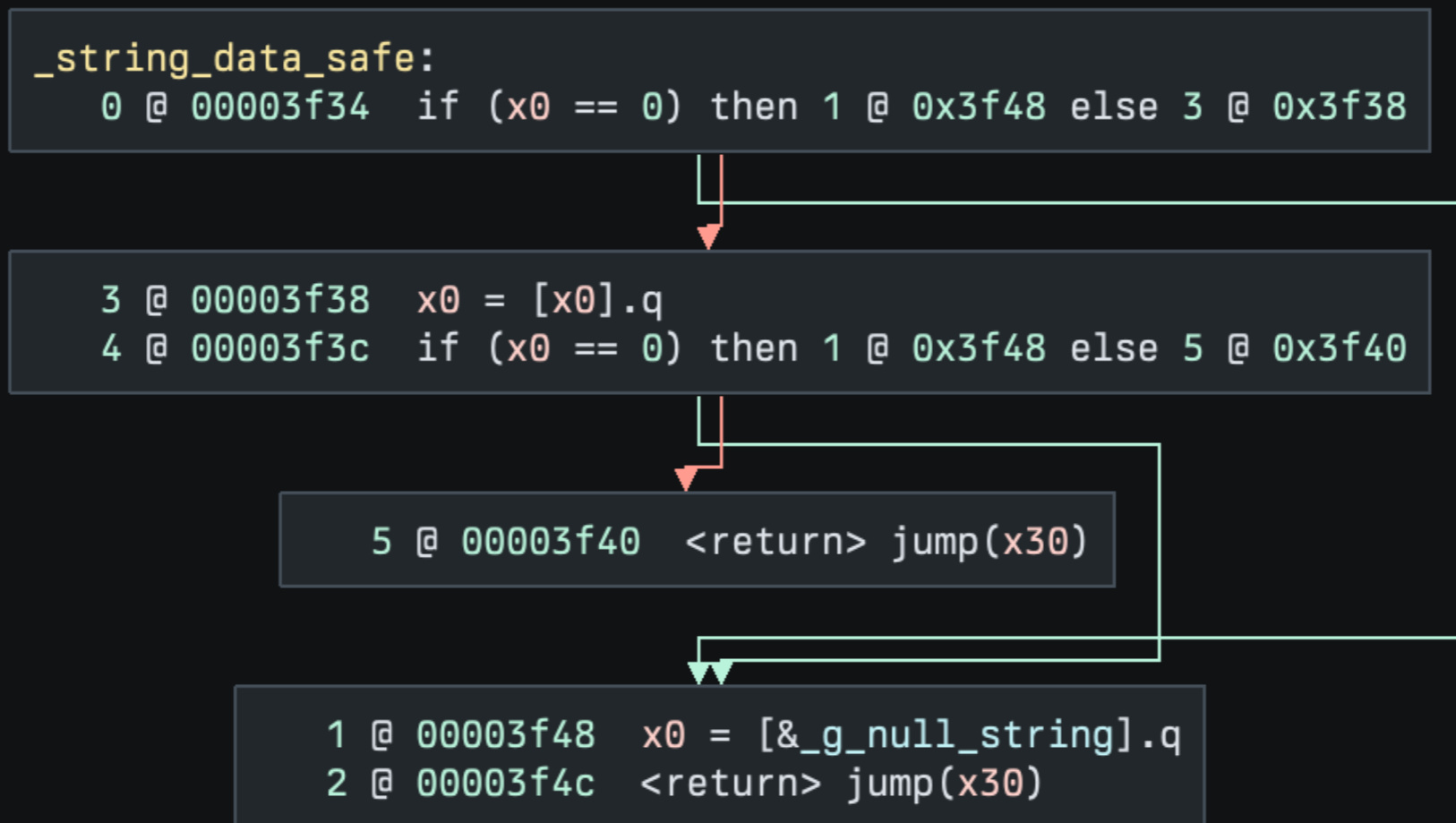
# Binary Ninja's ILs

A real-world example: disassembly



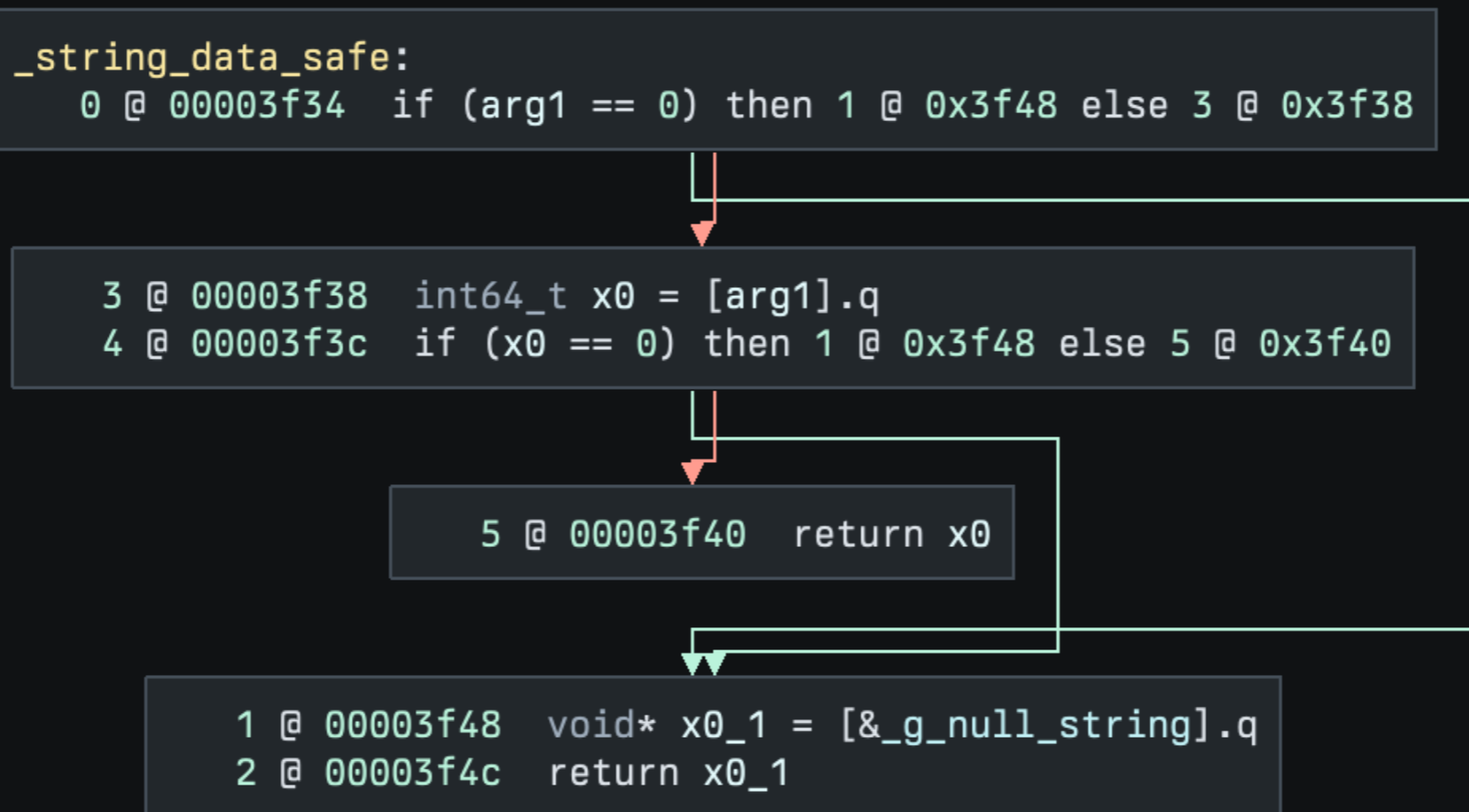
# Binary Ninja's ILs

A real-world example: LLIL



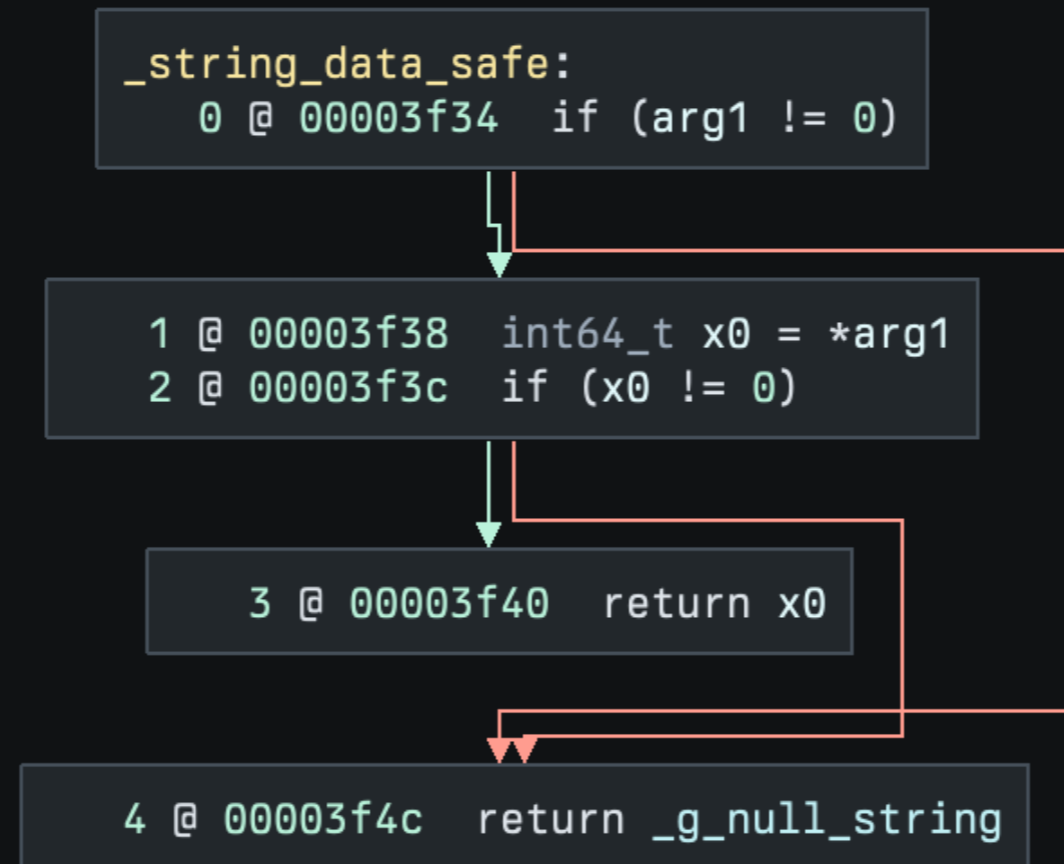
# Binary Ninja's ILs

A real-world example: MLIL



# Binary Ninja's ILs

A real-world example: HLIL



# Aside

## Immediate vs. deferred flags

- Immediate flags
  - IDA
  - Ghidra
- Deferred flags
  - Binary Ninja

# Aside

What about LLVM IR..?

- Has some apparent advantages
  - Popularity, infrastructure, maturity, etc.
- Harder to lift to straight from assembly
- Missing important decompilation-oriented semantics

# Data flow analysis

## Classic techniques (IDA)

- Tried-and-true compiler techniques
  - Value set analysis
  - Use-def chains
  - (Global) copy propagation
  - Etc...

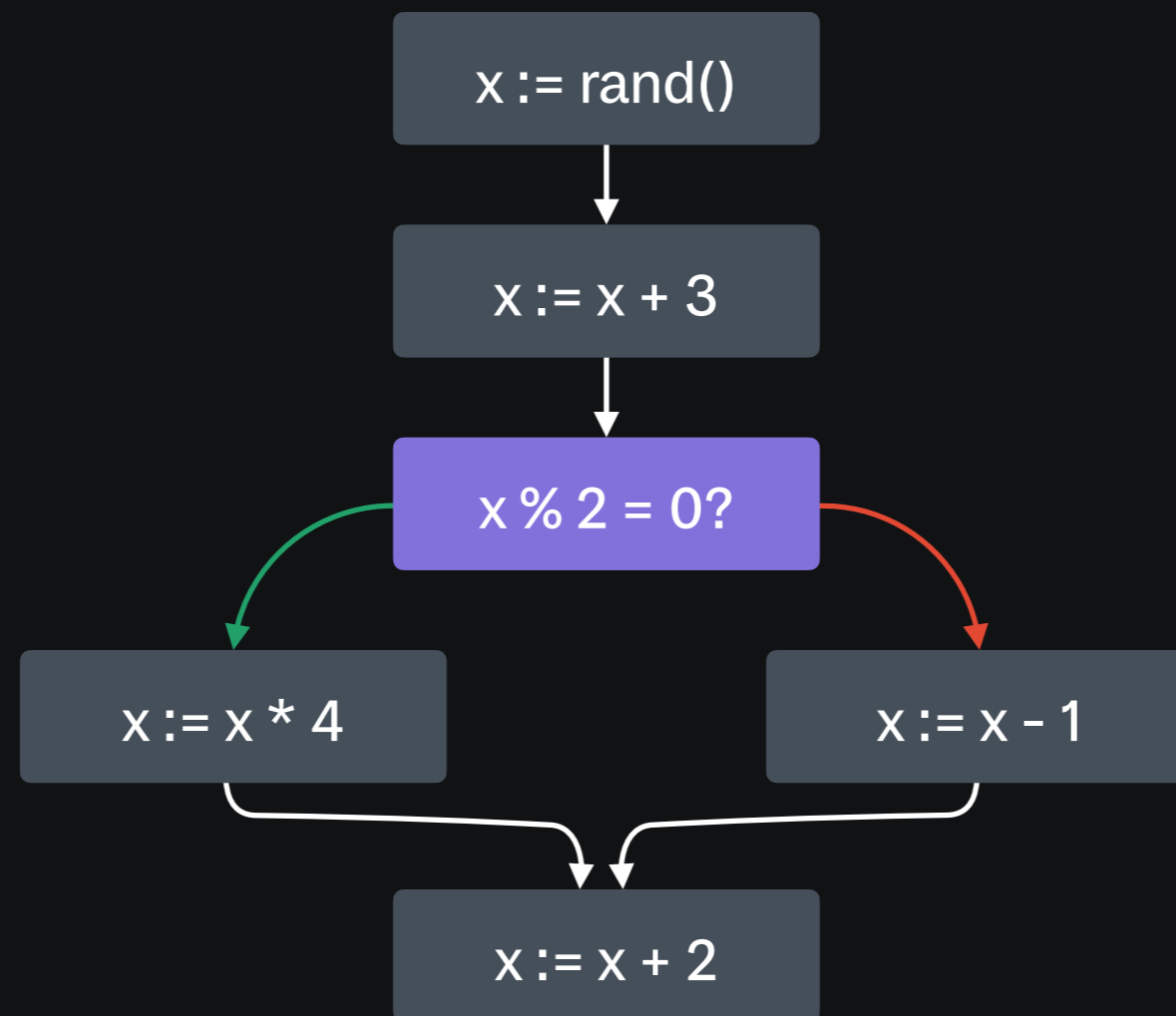
# Data flow analysis

Static single assignment form (Binary Ninja & Ghidra)

- Every variable has exactly one definition
  - Subsequent assignments labeled with subscripts
    - e.g.  $x_2$
- Joining of two live variables denoted with Phi ( $\Phi$ )
  - e.g.  $\Phi(x_2, x_3)$
- It's easier to understand visually...

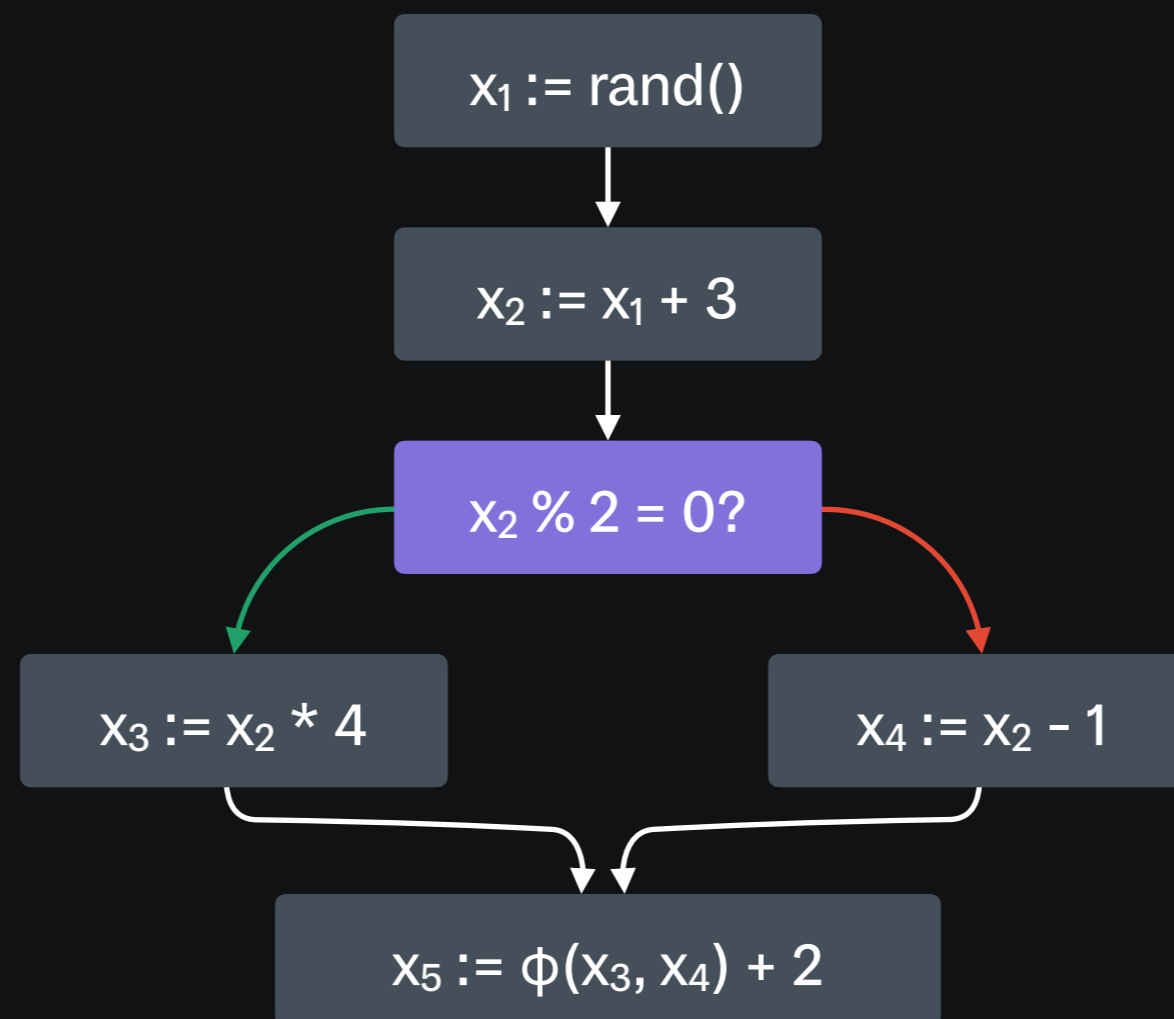
# Data flow analysis

Non-SSA form example



# Data flow analysis

## SSA form example



# Data flow analysis

## Real-world SSA form example

```
_example_ssa_small:  
00003f10  sub    sp, sp, #0x20  
00003f14  stp    x29, x30, [sp, #0x10] {__saved_x29} {__saved_x30}  
00003f18  add    x29, sp, #0x10 {__saved_x29}  
00003f1c  bl     _rand  
00003f20  stur   w0, [x29, #-0x4 {var_14}]  
00003f24  ldur   w8, [x29, #-0x4 {var_14}]  
00003f28  add    w8, w8, #0x3  
00003f2c  stur   w8, [x29, #-0x4 {var_14_1}]  
00003f30  ldur   w9, [x29, #-0x4 {var_14_1}]  
00003f34  ldur   w8, [x29, #-0x4 {var_14_1}]  
00003f38  tbnz   w9, #0, 0x3f44
```

```
00003f44  sub    w8, w8, #0x1
```

```
00003f3c  lsl    w8, w8, #0x2  
00003f40  b      0x3f48
```

```
00003f48  stur   w8, [x29, #-0x4 {var_14_2}]  
00003f4c  ldur   w0, [x29, #-0x4 {var_14_2}]  
00003f50  ldp    x29, x30, [sp, #0x10] {__saved_x29} {__saved_x30}  
00003f54  add    sp, sp, #0x20  
00003f58  ret
```

# Data flow analysis

## Real-world SSA form example

```
_example_ssa_small:
```

```
0 @ 00003f1c x0#1, mem#1 = _rand() @ mem#0  
1 @ 00003f20 var_14#1 = x0#1  
2 @ 00003f24 x8#1 = var_14#1  
3 @ 00003f28 x8_1#2 = x8#1 + 3  
4 @ 00003f2c var_14_1#2 = x8_1#2  
5 @ 00003f30 x9#1 = var_14_1#2  
6 @ 00003f34 x8_2#3 = var_14_1#2  
7 @ 00003f38 if ((x9#1 & 1) != 0) then 8 @ 0x3f44 else 10 @ 0x3f3c
```

```
8 @ 00003f44 x8_3#4 = x8_2#3 - 1  
9 @ 00003f44 goto 12 @ 0x3f48
```

```
10 @ 00003f3c x8_3#5 = x8_2#3 << 2  
11 @ 00003f40 goto 12 @ 0x3f48
```

```
12 @ 00003f48 x8_3#6 =  $\phi$  (x8_3#4, x8_3#5)  
13 @ 00003f48 var_14_2#3 = x8_3#6  
14 @ 00003f4c x0_1#2 = zx.q(var_14_2#3)  
15 @ 00003f58 return x0_1#2
```

# Closing thoughts

Modern decompilation challenges

- Unrepresentable instructions
- Inlining, outlining
- Pointer & length strings
- Pointer misconceptions
- Weird calling conventions
- Much more...

# Closing thoughts

Modern language construct support

- Modern code generation is increasingly intricate
  - Swift
  - Rust
  - Firebloom (-fbounds-safety)
- Need to filter signal from noise

# Closing thoughts

Decompile as X?

- Not everything is written in C anymore
  - Why do we try to decompile everything to C?
  - Should we still decompile everything as C?

# Closing thoughts

Diverging goals in decompilation

- Literal decompilation?
- Semantic decompilation?
- Answer: Abstraction/optimization problem

# Closing thoughts

## Modularity & tuning

- Soon-to-be necessity
- “One size fits all” decompilation is aging poorly
  - Need to be lean, adaptable
  - What will the next ten years hold?

# Demo time?

a.k.a “how long did those slides take?”

# Thank you!

- Questions?
- Find me after if you want to chat more
  - Not everything made it into the slides :)

- Twitter: [@jonpalmisc](https://twitter.com/jonpalmisc)

- Mastodon: [@jonpalmisc@infosec.exchange](https://infosec.exchange/@jonpalmisc)



```
~/Desktop/0x41conTalk — -zsh — 50x8
jp@telos:~/Desktop/0x41conTalk
[; wc Outline.md
 1049   5817  35610 Outline.md
jp@telos:~/Desktop/0x41conTalk
;
```