目录

前言 1.1
Capstone概述 1.2
  Capstone vs llvm 1.2.1
  其他相关 1.2.2
Capstone初始化环境 1.3
Capstone实例 1.4
  Unicorn中打印当前指令 1.4.1
Capstone使用心得 1.5
  Unicorn中Capstone使用心得 1.5.1
附录 1.6
  参考资料 1.6.1

反汇编利器：Capstone

- 最新版本： `v1.0`
- 更新时间： `20230901`

简介

整理强大好用的反汇编工具Capstone。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

HonKit源码

- crifan/ultimate_disassembler_capstone: 反汇编利器：Capstone

如何使用此HonKit源码去生成发布为电子书

详见：crifan/honkit_template: demo how to use crifan honkit template and demo

在线浏览

- 反汇编利器：Capstone book.crifan.org
- 反汇编利器：Capstone crifan.github.io

离线下载阅读

- 反汇编利器：Capstone PDF
- 反汇编利器：Capstone ePub
- 反汇编利器：Capstone Mobi

版权和用途说明

此电子书教程的全部内容，如无特别说明，均为本人原创。其中部分内容参考自网络，均已备注了出处。如发现有侵权，请通过邮箱联系我 `admin 艾特 crifan.com` ，我会尽快删除。谢谢合作。

各种技术类教程，仅作为学习和研究使用。请勿用于任何非法用途。如有非法用途，均与本人无关。

鸣谢

感谢我的老婆**陈雪**的包容理解和悉心照料，才使得我 `crifan` 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

其他

作者的其他电子书

本人 `crifan` 还写了其他 `150+` 本电子书教程，感兴趣可移步至：

crifan/crifan_ebook_readme: Crifan的电子书的使用说明

关于作者

关于作者更多介绍，详见：

关于CrifanLi李茂 – 在路上

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 23:47:49

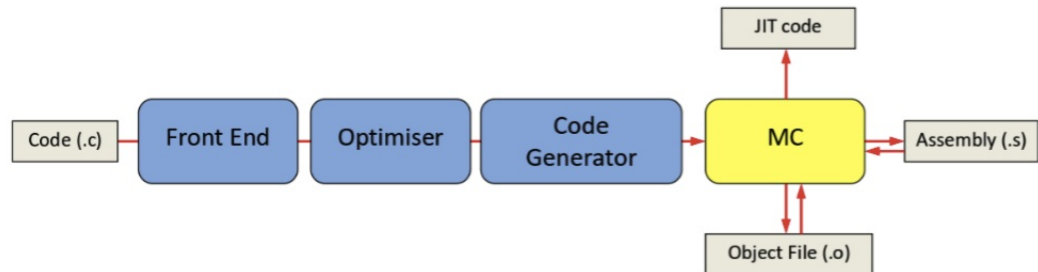
Capstone概述

- `Capstone`
  - logo
    -
  - 一句话描述
    - 终极反汇编器
      - The Ultimate Disassembler
    - 更好的下一代反汇编引擎
      - Next Generation Disassembler Engine
    - 一个轻量级的支持多平台和多架构的反汇编框架
      - a lightweight multi-platform, multi-architecture disassembly framework
  - 特点
    - lightweight=轻量级
      - 简洁的API
        - Clean/simple/lightweight/intuitive architecture-neutral API
        - 多种语言接口Bindings=提供了多种语言的编程接口
          - Clojure, F#, Common Lisp, Visual Basic, PHP, PowerShell, Haskell, Perl, Python, Ruby, C#, NodeJS, Java, GO, C++, OCaml, Lua, Rust, Delphi, Free Pascal
    - multi-platform=支持多平台=跨平台
      - `Windows` & `*nix` (with `Mac macOS`, `iOS`, `Android`, `Linux`, `*BSD` & `Solaris` confirmed)
    - multi-architecture 支持多种架构
      - `Arm`, `Arm64` (`Armv8`), `BPF`, `Ethereum Virtual Machine`, `M68K`, `M680X`, `Mips`, `MOS65XX`, `PowerPC`, `RISCV`, `Sparc`, `SystemZ`, `TMS320C64X`, `Web Assembly`, `XCore` & `X86` (include `X86_64`)
  - Capstone的强大之处
    - 反汇编 + 分析
    - 编译成中间文本形式代码，便于调试
  - 用途=应用领域
    - 安全领域
      - 二进制分析 binary analysis
      - 逆向 reversing
  - 谁用到了Capstone
    - 著名的开源逆向工具 `Radare2`
    - 商业逆向工具 `IDA Pro` 的第三方插件
    - `IntelliJ IDEA`
    - `Qemu`
    - `Binwalk`
    - `Camal` ：Coseinc恶意软件自动分析
    - `Pyew` ：Python恶意静态分析工具
    - `Cuckoo`
    - 另： `Kali Linux` 中已集成
    - 等
  - 主页
    - 官网
      - The Ultimate Disassembly Framework – Capstone – The Ultimate Disassembler

        - http://www.capstone-engine.org
  - GitHub
    - aquynh/capstone: Capstone disassembly/disassembler framework: Core (Arm, Arm64, BPF, EVM, M68K, M680X, MOS65xx, Mips, PPC, RISCV, Sparc, SystemZ, TMS320C64x, Web Assembly, X86, X86_64, XCore) + bindings.
      - https://github.com/aquynh/capstone

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 23:42:12

Capstone vs llvm

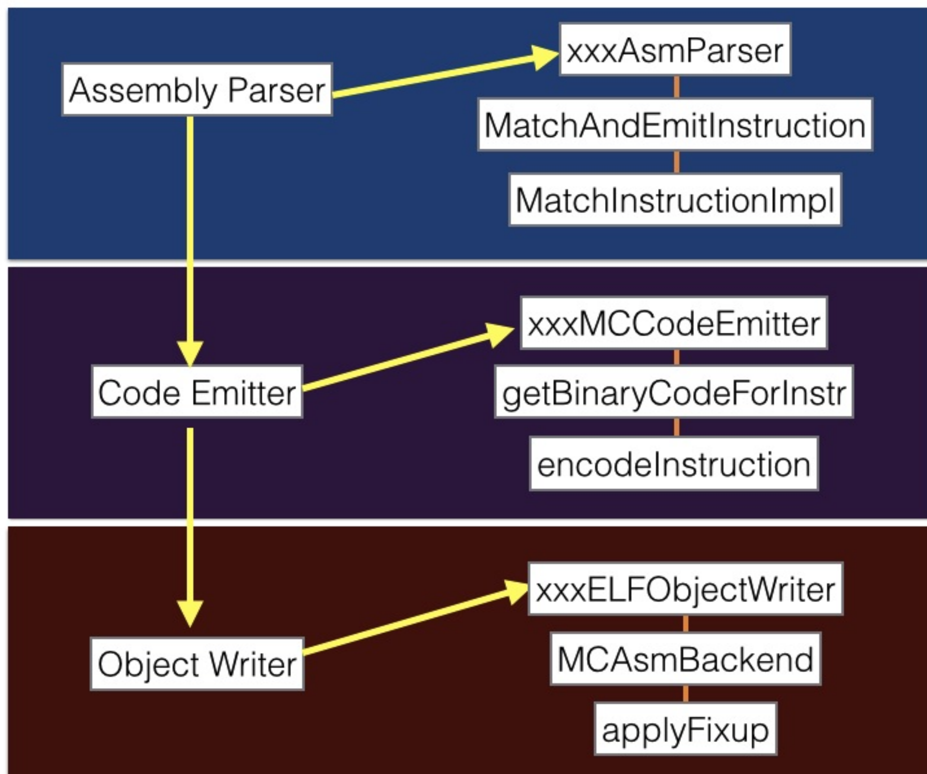
- capstone源自 `LLVM` 编译器框架中的**MC模块**
  - `MC` 模块中有个反汇编引擎叫做 `MCDisassembler`
    - `MC` = `Machine Code`
    - 机制：
      - 

  - 而 `llvm` 甚至还有个工具叫做： `llvm-mc`
    - 可以用于反汇编输入的二进制文件
- `capstone` 才用了 `MCDisassembler` 作为核心内容
  - 但又经过了大量优化改动，以适配自己的设计
  - `capstone` 在 `MCDisassembler` 在基础上加了其他的大量的功能
    - -> `MCDisassembler` 能做的 `capstone` 都能做
- `capstone` 和 `llvm-mc` 的区别
  - 详见
    - https://www.capstone-engine.org/beyond_llvm.html

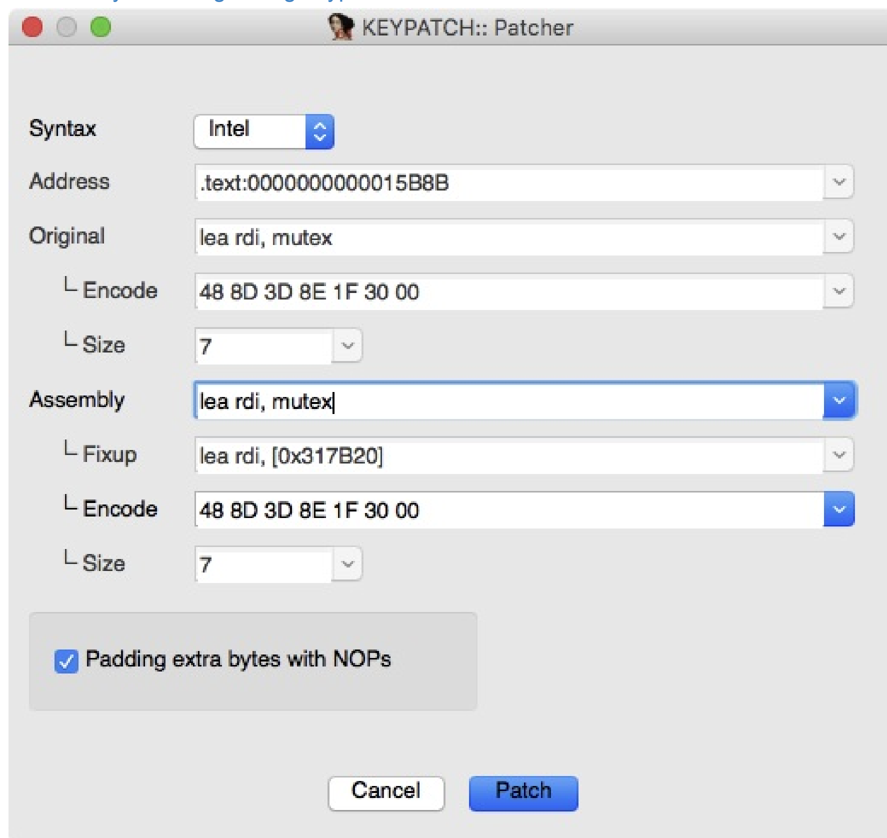
crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 23:41:45

其他相关

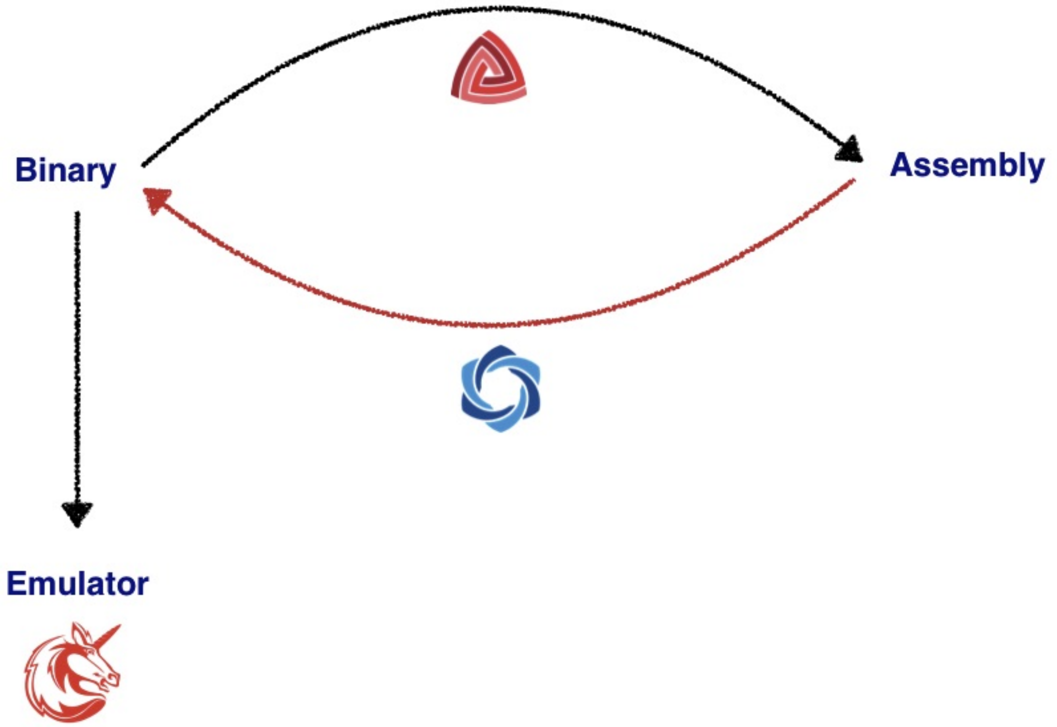
- capstone转llvm
  - chubbymaggie/capstone2llvmir: Library for Capstone instruction to LLVM IR translation
    - https://github.com/chubbymaggie/capstone2llvmir
- 成套工具
  - 3件套
    - Logo
      -

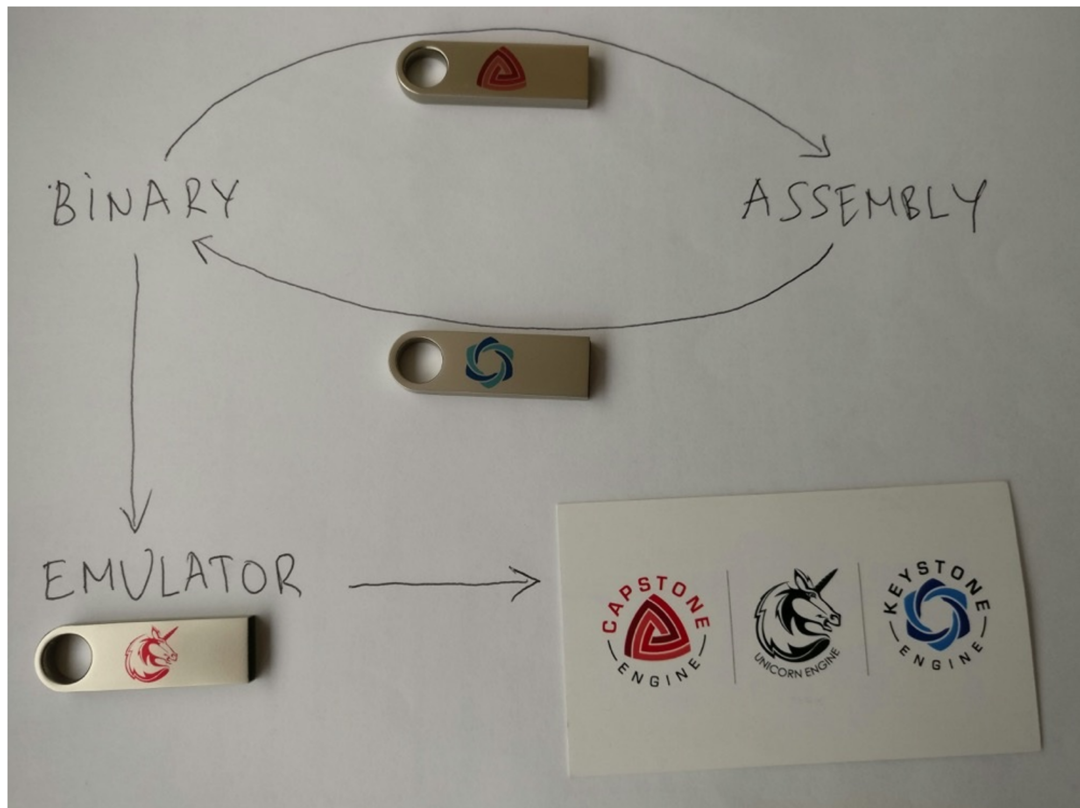
    - 3个项目
      - `Capstone disassembler`
        - Next Generation Disassembler Engine
          - http://capstone-engine.org
      - `Unicorn emulator`
        - Next Generation CPU Emulator
          - http://unicorn-engine.org
      - `Keystone assembler`
        - http://keystone-engine.org
        - 流程
          - 

        - IDA插件

- Keypatch – Keystone – The Ultimate Assembler
    - https://www.keystone-engine.org/keypatch/
    - 

- 竞品
    - `Radare2`
        - Unix-like reverse engineering framework and commandline tools
    - `Pwnypack`
        - CTF toolkit with Shellcode generator Ropper: Rop gadget and binary information tool
    - `GEF`
        - GDB plugin with enhanced features
    - `Usercorn`
        - Versatile kernel+system+userspace emulator
    - `X64dbg`
        - An open-source x64/x32 debugger for windows
    - `Liberation`
        - code injection library for iOS
    - `Demovfuscator`
        - Deobfuscator for movfuscated binaries
- 效果：
    - Fundamental frameworks for Reverse Engineering

- 

- 

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 23:44:34

Capstone初始化环境

Mac中安装Capstone

- 概述
  - Mac
    - `brew install capstone`
  - Ubuntu
    - `sudo apt-get install libcapstone3`

详解：

- 先安装Capstone的**core**
  - 命令

    ```
    brew install capstone
    ```

  - 注：查看已安装信息
    - `brew info capstone`
      - 已安装版本：`capstone: stable 4.0.2 (bottled), HEAD`
- 再安装对应的**binding**
  - Python
    - 命令

      ```
      sudo pip install capstone
      ```

    - 注：
      - 查看已安装信息
        - `pip show capstone`
      - 已安装的版本：`Capstone: 4.0.2`

测试Capstone运行正常

用Capstone的Python测试代码：

- X86

```
# test1.py
from capstone import *

CODE = b"\x55\x48\x8b\x05\xb8\x13\x00\x00"

md = Cs(CS_ARCH_X86, CS_MODE_64)
for i in md.disasm(CODE, 0x1000):
  print("0x%x:\t%s\t%s" %(i.address, i.mnemonic, i.op_str))
```

预期输出：

```
0x1000: push    rbp
0x1001: mov     rax, qword ptr [rip + 0x13b8]
```

- arm64

```
from capstone import *
from capstone.arm64 import *

CODE = b"\xe1\x0b\x40\xb9\x20\x04\x81\xda\x20\x08\x02\x8b"

md = Cs(CS_ARCH_ARM64, CS_MODE_ARM)
md.detail = True

for insn in md.disasm(CODE, 0x38):
    print("0x%x:\t%s\t%s" %(insn.address, insn.mnemonic, insn.op_str))

    if len(insn.operands) > 0:
        print("\tNumber of operands: %u" %len(insn.operands))
        c = -1
        for i in insn.operands:
            c += 1
            if i.type == ARM64_OP_REG:
                print("\t\toperands[%u].type: REG = %s" %(c, insn.reg_name(i.value.reg)))
            if i.type == ARM64_OP_IMM:
                print("\t\toperands[%u].type: IMM = 0x%x" %(c, i.value.imm))
            if i.type == ARM64_OP_CIMM:
                print("\t\toperands[%u].type: C-IMM = %u" %(c, i.value.imm))
            if i.type == ARM64_OP_FP:
                print("\t\toperands[%u].type: FP = %f" %(c, i.value.fp))
            if i.type == ARM64_OP_MEM:
                print("\t\toperands[%u].type: MEM" %c)
                if i.value.mem.base != 0:
                    print("\t\t\toperands[%u].mem.base: REG = %s" \
                        %(c, insn.reg_name(i.value.mem.base)))
                if i.value.mem.index != 0:
                    print("\t\t\toperands[%u].mem.index: REG = %s" \
                        %(c, insn.reg_name(i.value.mem.index)))
                if i.value.mem.disp != 0:
                    print("\t\t\toperands[%u].mem.disp: 0x%x" \
                        %(c, i.value.mem.disp))

            if i.shift.type != ARM64_SFT_INVALID and i.shift.value:
                print("\t\t\tShift: type = %u, value = %u" %(i.shift.type, i.shift.value))

            if i.ext != ARM64_EXT_INVALID:
                print("\t\t\tExt: %u" %i.ext)

    if insn.writeback:
        print("\tWrite-back: True")
    if not insn.cc in [ARM64_CC_AL, ARM64_CC_INVALID]:
        print("\tCode condition: %u" %insn.cc)
    if insn.update_flags:
        print("\tUpdate-flags: True")
```

预期输出：

```
0x38:	ldr	w1, [sp, #8]
	Number of operands: 2
		operands[0].type: REG = w1
		operands[1].type: MEM
			operands[1].mem.base: REG = sp
			operands[1].mem.disp: 0x8
0x3c:	cneg	x0, x1, ne
	Number of operands: 2
		operands[0].type: REG = x0
		operands[1].type: REG = x1
	Code condition: 2
0x40:	add	x0, x1, x2, lsl #2
	Number of operands: 3
		operands[0].type: REG = x0
		operands[1].type: REG = x1
		operands[2].type: REG = x2
			Shift: type = 1, value = 2
```

-> 更多测试代码，详见：

- Programming with Python language – Capstone – The Ultimate Disassembler (capstone-engine.org)
- capstone/bindings/python at master · capstone-engine/capstone · GitHub

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 23:04:26

Capstone实例

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 22:56:40

Unicorn中打印当前指令

此处介绍，在Unicorn模拟代码期间，用Capstone打印当前正在执行的指令：

自己的实际代码

模拟akd函数symbol2575

中的 `hook_code` 中的这部分的代码：

```
from capstone import *
from capstone.arm64 import *

BYTES_PER_LINE = 4

# Init Capstone instance
cs = Cs(CS_ARCH_ARM64, CS_MODE_ARM + CS_MODE_LITTLE_ENDIAN)
cs.detail = True

#-------------------- Code --------------------

# memory address where emulation starts
CODE_ADDRESS = 0x10000
# code size: 4MB
CODE_SIZE = 4 * 1024 * 1024
CODE_ADDRESS_END = (CODE_ADDRESS + CODE_SIZE) # 0x00410000


def bytesToOpcodeStr(curBytes):
    opcodeByteStr = ''.join('{:02X} '.format(eachByte) for eachByte in curBytes)
    return opcodeByteStr

# callback for tracing instructions
def hook_code(mu, address, size, user_data):

    # logging.info(">>> Tracing instruction at 0x%x, instruction size = 0x%x", address, size)
    lineCount = int(size / BYTES_PER_LINE)
    for curLineIdx in range(lineCount):
        startAddress = address + curLineIdx * BYTES_PER_LINE
        codeOffset = startAddress - CODE_ADDRESS
        opcodeBytes = mu.mem_read(startAddress, BYTES_PER_LINE)
        opcodeByteStr = bytesToOpcodeStr(opcodeBytes)
        decodedInsnGenerator = cs.disasm(opcodeBytes, address)
        # if gSingleLineCode:
        for eachDecodedInsn in decodedInsnGenerator:
            eachInstructionName = eachDecodedInsn.mnemonic
            offsetStr = "<+%d>" % codeOffset
            logging.info("--- 0x%08X %7s: %s -> %s\t%s", startAddress, offsetStr, opcodeByteStr, eachInstructionName,
 eachDecodedInsn.op_str)

...
def emulate_akd_arm64_symbol2575():
...
        mu.hook_add(UC_HOOK_CODE, hook_code, begin=CODE_ADDRESS, end=CODE_ADDRESS_END)
```

主要目的就是：

优化了log日志打印，希望打印输出的内容，尽量贴近之前Xcode调试（iOS的ObjC的）ARM汇编代码的（lldb反汇编的）显示效果：

```
libobjc.A.dylib`objc_alloc_init:
->  0x19cbd3c3c <+0>:  stp    x29, x30, [sp, #-0x10]!
    0x19cbd3c40 <+4>:  mov    x29, sp
    0x19cbd3c44 <+8>:  cbz    x0, 0x19cbd3c5c           ; <+32>
    0x19cbd3c48 <+12>: ldr    x8, [x0]
    0x19cbd3c4c <+16>: and    x8, x8, #0xffffffff8
    0x19cbd3c50 <+20>: ldrb   w8, [x8, #0x1d]
    ...
```

即，是类似于这种格式：

- 当前地址 <+偏移量>: 指令 操作数

且还希望，加上IDA中能显示opcode的信息：

- 当前地址 <+偏移量>: opcode -> 指令 操作数

所以最后经过优化，用上述代码，实现了类似Xcode中的输出效果：

```
=== 0x000113AC <+5036>: 28 01 08 0B  => add    w8, w9, w8
=== 0x000113B0 <+5040>: 08 09 01 11  => add    w8, w8, #0x42
=== 0x000113B4 <+5044>: 28 DB A8 B8  => ldrsw  x8, [x25, w8, sxtw #2]
=== 0x000113B8 <+5048>: 1F 20 03 D5  => nop
=== 0x000113BC <+5052>: 29 D4 2B 58  => ldr    x9, #0x68e40
=== 0x000113C0 <+5056>: 08 01 09 8B  => add    x8, x8, x9
=== 0x000113C4 <+5060>: 00 01 1F D6  => br     x8
```

如此，可以方便的查看到：

- 当前代码执行到哪里了== 当前的地址 == PC的值
- 函数内的偏移量
- opcode=指令的二进制值
- （借助Capstone解析后的）当前正在执行什么指令

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 22:57:52

Capstone使用心得

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 22:59:24

Unicorn中Capstone使用心得

Unicorn真正执行代码 和 Capstone反编译显示的代码 未必完全一样

Capstone反汇编出来的指令，有些细节和Xcode中不太一样

- 即，用Capstone去反汇编看到的指令，和Unicorn真正执行的指令，未必相同
  - 但是还是可以供参考的，基本上差距不大

mov vs movz

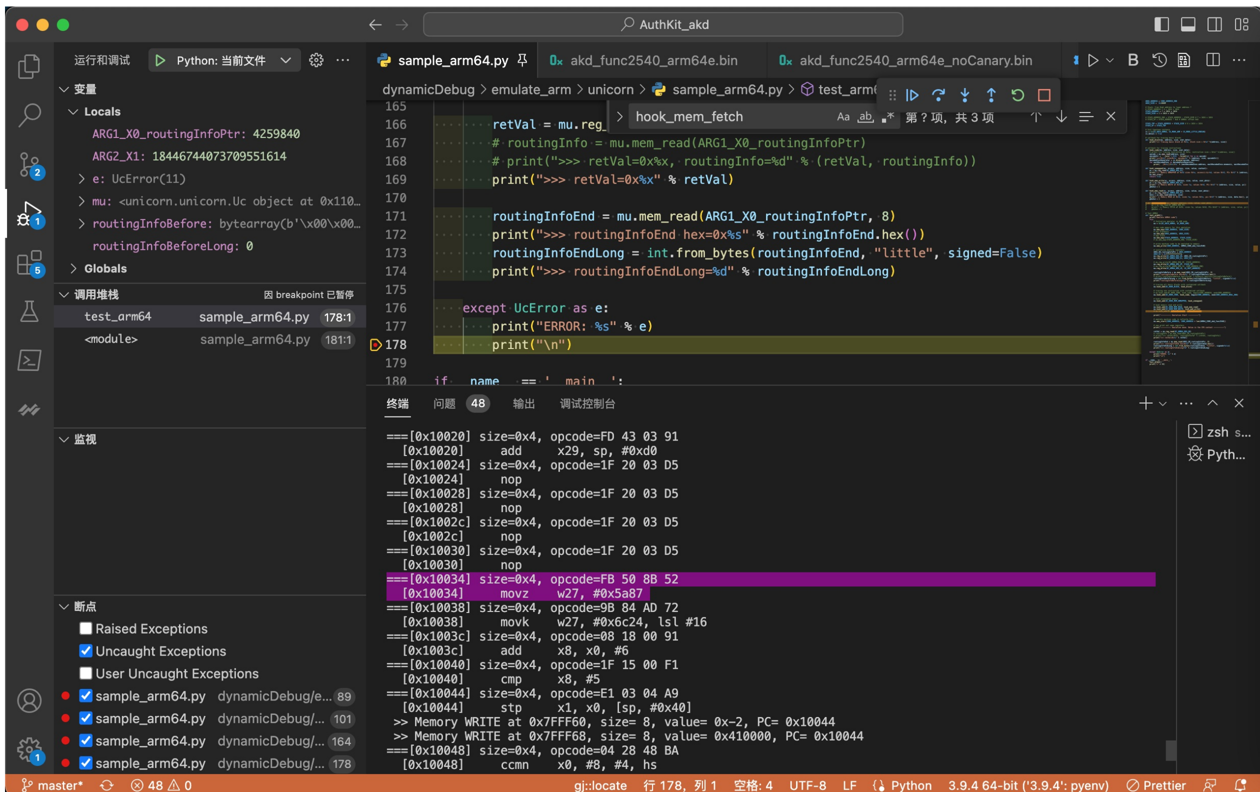
- Unicorn底层真正执行的指令 == Xcode反汇编看到的指令： `mov`

```
akd`___lldb_unnamed_symbol2540$$akd:
...
    0x1050319c0 <+52>:    mov    w27, #0x5a87
```

```
akd`___lldb_unnamed_symbol2540$$akd:
->  0x10503198c <+0>:     pacibsp
    0x105031990 <+4>:     sub    sp, sp, #0xe0             ; =0xe0
    0x105031994 <+8>:     stp    x28, x27, [sp, #0x80]
    0x105031998 <+12>:    stp    x26, x25, [sp, #0x90]
    0x10503199c <+16>:    stp    x24, x23, [sp, #0xa0]
    0x1050319a0 <+20>:    stp    x22, x21, [sp, #0xb0]
    0x1050319a4 <+24>:    stp    x20, x19, [sp, #0xc0]
    0x1050319a8 <+28>:    stp    x29, x30, [sp, #0xd0]
    0x1050319ac <+32>:    add    x29, sp, #0xd0            ; =0xd0
    0x1050319b0 <+36>:    nop
    0x1050319b4 <+40>:    ldr    x8, #0x5f1ec              ; (void *)0x00000001da7b9c60: __stack_chk_guard
    0x1050319b8 <+44>:    ldr    x8, [x8]
    0x1050319bc <+48>:    stur   x8, [x29, #-0x58]
    0x1050319c0 <+52>:    mov    w27, #0x5a87              Thread 5: breakpoint 34.1 (1)
    0x1050319c4 <+56>:    movk   w27, #0x6c24, lsl #16
    0x1050319c8 <+60>:    add    x8, x0, #0x6              ; =0x6
    0x1050319cc <+64>:    cmp    x8, #0x5                  ; =0x5
    0x1050319d0 <+68>:    stp    x1, x0, [sp, #0x40]
    0x1050319d4 <+72>:    ccmn   x0, #0x8, #0x4, hs
    0x1050319d8 <+76>:    mov    w8, #0x1
    0x1050319dc <+80>:    csel   w8, wzr, w8, eq
    0x1050319e0 <+84>:    mov    w11, #0xa57a
    0x1050319e4 <+88>:    movk   w11, #0x93db, lsl #16
```

- Capstone反编译出的指令： `movz`

```
===[0x10034] size=0x4, opcode=FB 50 8B 52
  [0x10034]     movz    w27, #0x5a87
```



—> 经过确认，其实：是一样的。

细节是：

- `MOV (wide immediate)`
  - 概述
    - Move 16-bit immediate to register.
    - This instruction is an alias of MOVZ.
  - 语法
    - `MOV Wd, #imm`
    - `MOV Xd, #imm`
  - 解释
- `MOVZ`
  - 概述
    - Move shifted 16-bit immediate to register.
    - This instruction is used by the alias MOV (wide immediate).
  - 语法
    - `MOVZ Wd, #imm{, LSL #shift}`
    - `MOVZ Xd, #imm{, LSL #shift}`
- 对比
  - 要移动的立即数imm：
    - `MOV (wide immediate)`
      - 有2种
        - 对于32位的 `Wd` ： `32位`
        - 对于64位的 `Xd` ： `64位`
      - -》和MOVZ的16bit比，32位和64位位数**更宽**，所以叫做 `wide immediate`
    - `MOVZ`
      - 16位（无符号的立即数）
  - -》而当MOVZ中 `shift=0` 时，且 `imm的值 <=65536` 即 `16位` 时：
    - `MOV (wide immediate)` == `MOVZ`
      - 举例：当 `imm=0x5a87` ，多种写法，代码逻辑是一样的
        - `mov w27, #0x5a87`

- `mov x27, #0x5a87`
- `movz w27, #0x5a87`
- `movz x27, #0x5a87`

有些值是计算后的值，而不是指令本身的值

概述：有些值是计算后的值，而不是指令本身的值

举例1：`adr x25, #0x227e4` vs `adr x25, #0x32850`

Unicorn调试期间，如果也是像我：用到Capstone去，查看当前反汇编后的ARM汇编代码

尤其要注意，对于 `adr` 等指令，其显示出的值：是**计算后**的值，而不是**原始**的值

比如：

原始ARM汇编指令是：

```
adr    x25, #0x227e4
```

而经过 `Capstone` 去反汇编出来的，却是：

```
adr     x25, #0x32850
```

其中的，加上当前PC的值，完整的log是：

```
0x0001006C  <+108 : 39 3F 11 10  -> adr     x25, #0x32850
```

可以看出：

- `x25` = `0x32850` = `当前PC` + 指令中的**原始**的偏移量 = `0x0001006C + 0x227e4`

-》否则，不小心就搞错了，以为是：

- `x25` = `0x428BC` = `当前PC` + 指令中的原始的偏移量 = `0x0001006C + 0x32850`

举例2：`LDR x8 #0x5F1EC` vs `LDR x8 #0x6F214`

- 二进制=opcode：`68 8F 2F 58`
  - 条件
    - ARM的little endian 小端
    - ARM64模式
  - 用
    - Xcode
    - 自己手动解码
    - Unicorn
    - Capstone
  - 正常解码出是
    - `LDR x8 #0x5F1EC`
  - 只不过：Unicorn/Capstone，会去继续处理：
    - 根据此处LDR (literal) == LDR (PC-relative literal)的本意：
      - `具体要加载的值` = `offset + PC的值` = `0x5F1EC` + `当前PC值是0x10028` = `0x6F214`
  - 所以，而是，Unicorn/Capstone中，显示出来的是解码后 + 解析后，最终的结果：
    - `LDR x8 #0x6F214`
  - 以为是解码错误，实际上是：解码正确的
    - 只是显示逻辑上，略有不同而已

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 23:04:56

附录

下面列出相关参考资料。

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2022-03-17 20:39:28

参考资料

- 【已解决】unicorn中用Capstone反汇编查看当前汇编代码指令
- 反汇编框架 Capstone
- 【已解决】Mac中安装和初始化Capstone去显示反汇编代码
- 【整理】Unicorn调试心得：Capstone反汇编中有些值是计算后的结果而不是原始ARM指令中的值
- 【已解决】unicorn模拟ARM指令：Capstone和Xcode的指令反汇编结果不一样
-
- 模拟akd函数symbol2575
-
- Programming with Python language – Capstone – The Ultimate Disassembler (capstone-engine.org)
- capstone/bindings/python at master · capstone-engine/capstone · GitHub
- Capstone & LLVM – Capstone – The Ultimate Disassembler
-

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2023-09-01 23:41:48