the life-changing magic of ida python

embedded device edition

maddie stone

madeline.stone@jhuapl.edu

@maddiestone

RECON MONTREAL 2017

who am i? – maddie stone

- reverse engineer and embedded developer at Johns Hopkins Applied Physics Lab
  - mostly embedded devices
  - merge of hardware and firmware reverse engineering
  - lead of reverse engineering working group at JHU/APL

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

reduce time required to analyze firmware of embedded devices using ida python

ida python embedded toolkit

https://github.com/maddiestone/IDAPythonEmbeddedToolkit

ida python

- "IDAPython is an IDA Pro plugin that integrates the Python programming language, allowing scripts to run in IDA Pro"
  - https://github.com/idapython/src/
  - Docs: https://www.hex-rays.com/products/ida/support/idapython_docs/
    - idc contains 98% of the functions we use

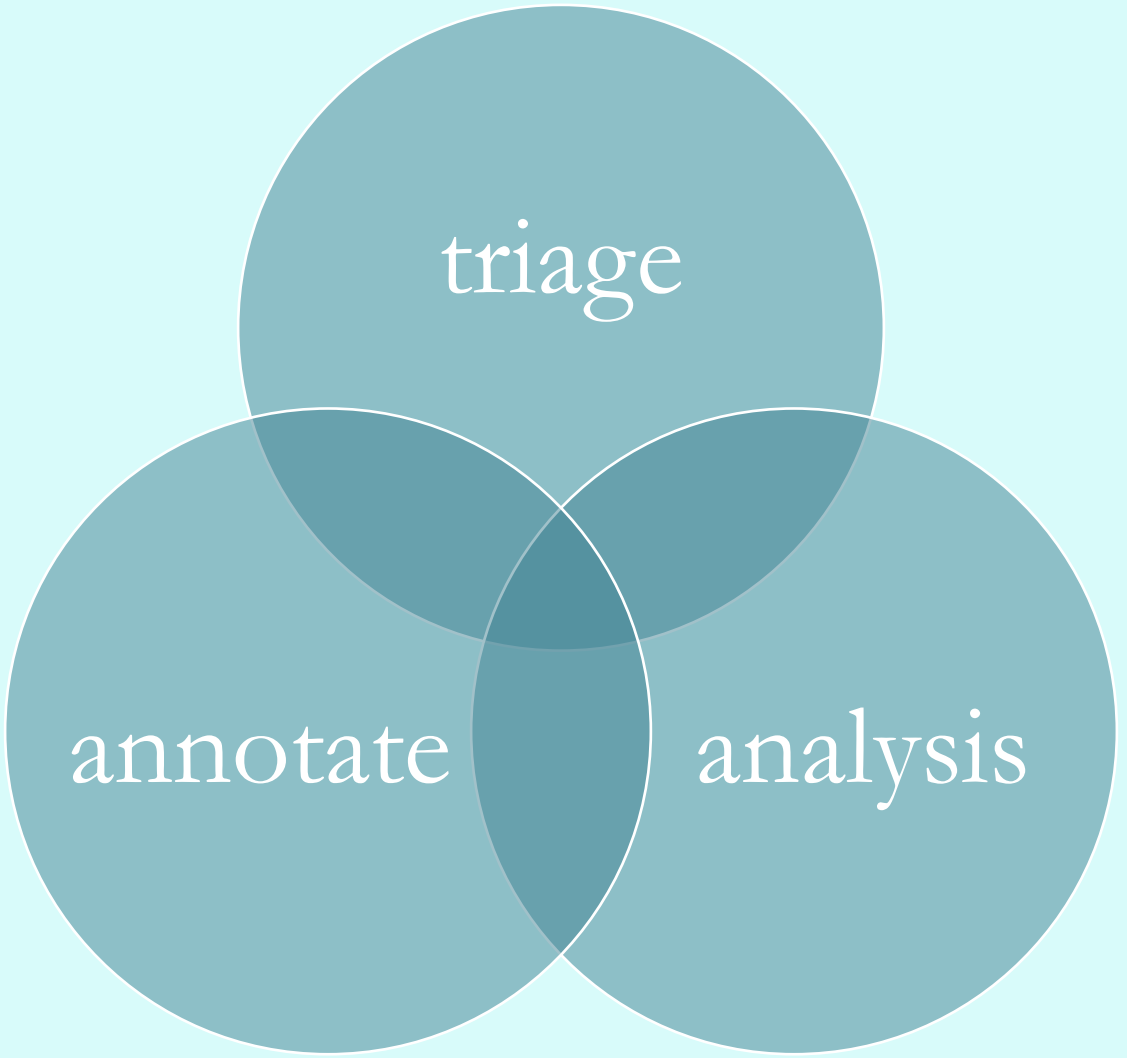
why do you care?

- current resources for ida python
  - mostly x86 or ARM based (PC applications or malware)
  - Palo Alto Networks: http://researchcenter.paloaltonetworks.com/2015/12/using-idapython-to-make-your-life-easier-part-1/
  - "The Beginner's Guide to IDAPython" by Alexander Hanel
- more embedded devices (hello, Internet of Things!)
  - microcontroller/microprocessor architectures
  - different goals of analysis than malware/application RE

important differences for embedded firmware images

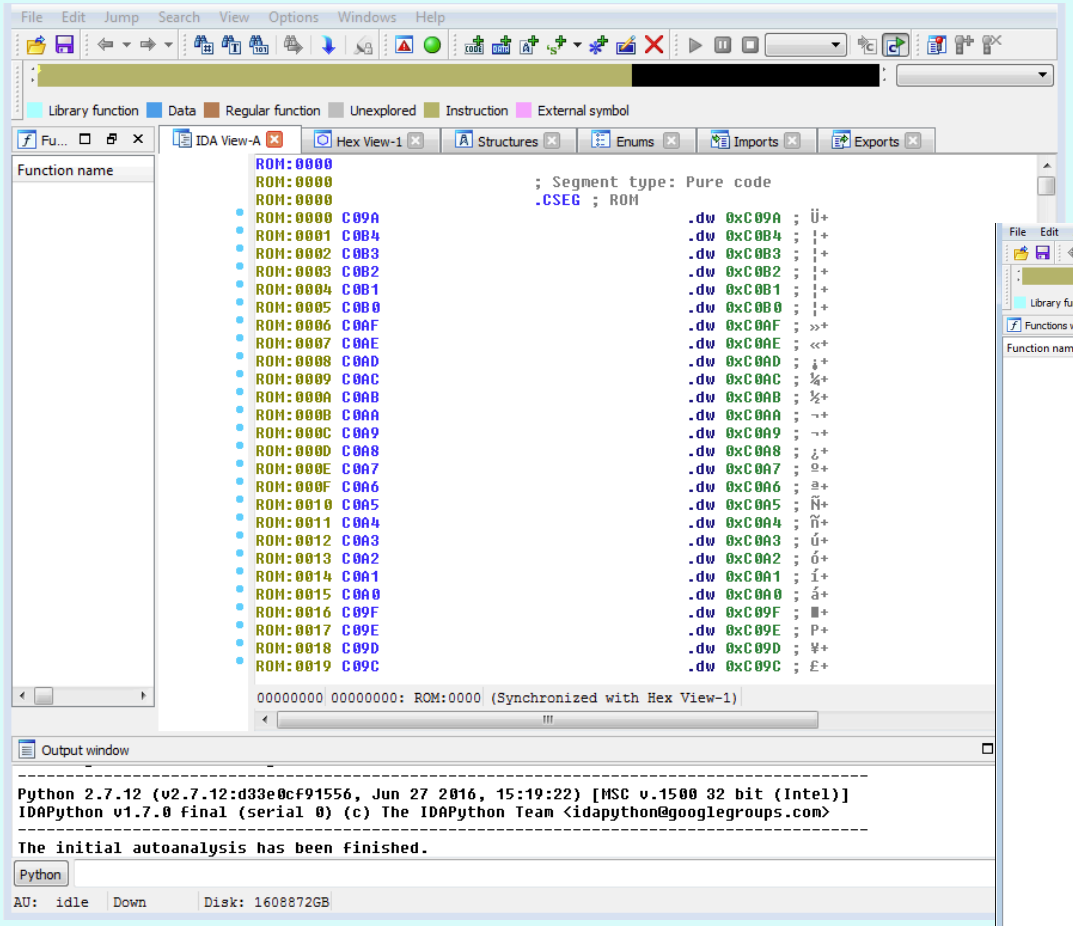
- purpose of analysis
- entire firmware image vs. application
- memory structure
- many different architectures

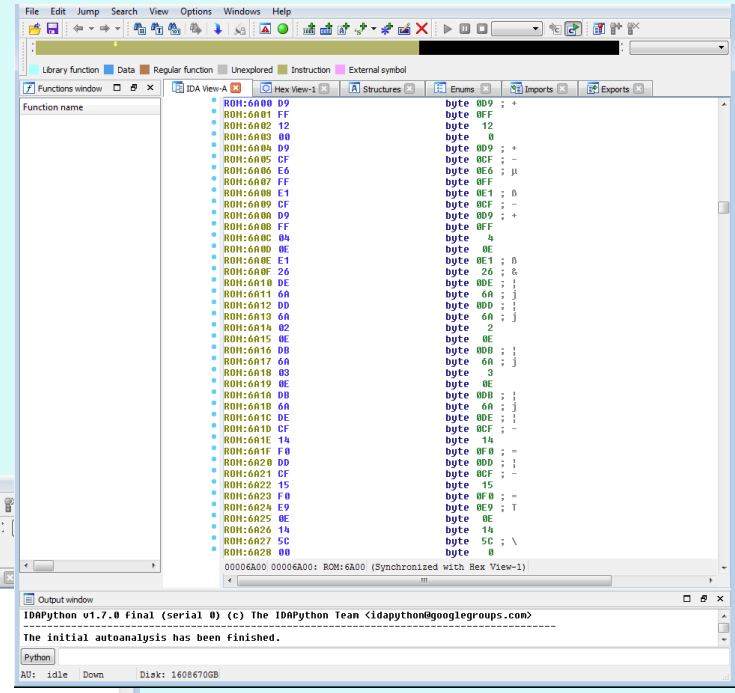
scripting the reverse engineering process



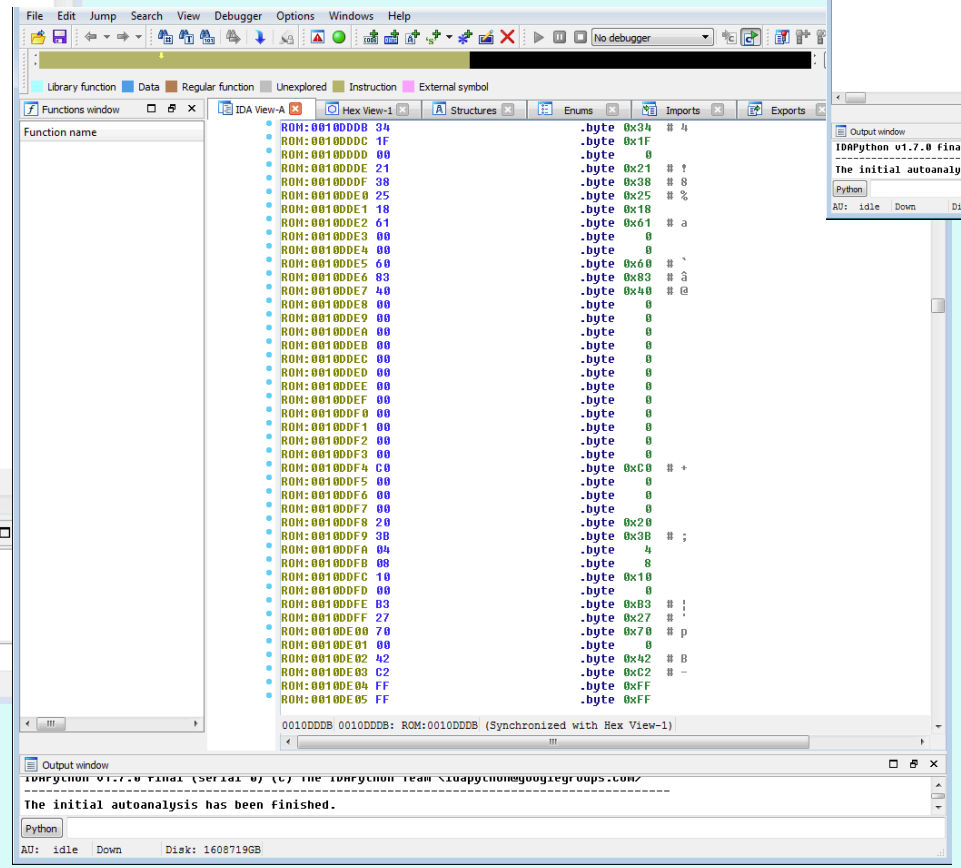
Atmel AVR

triage?





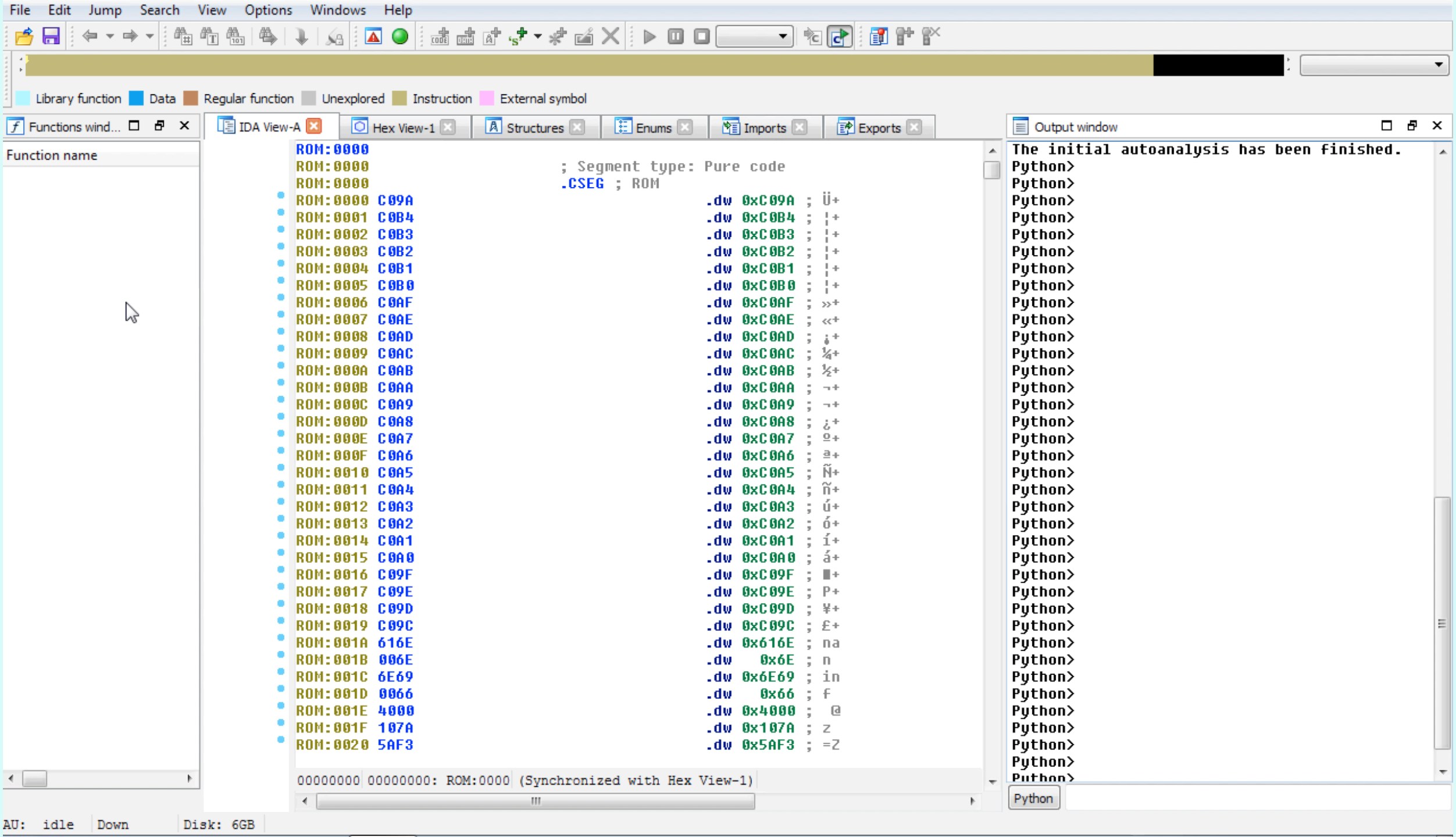
PIC18



MIPS

how ida python helps -- triage

- define_data_as_types.py
  - mass assign bytes as instructions, data, offsets
- define_code_functions.py
  - auto-assign ”unexplored” bytes as code and attempt to define functions
- make_strings.py
  - searches an address range for series of ASCII characters to define as strings

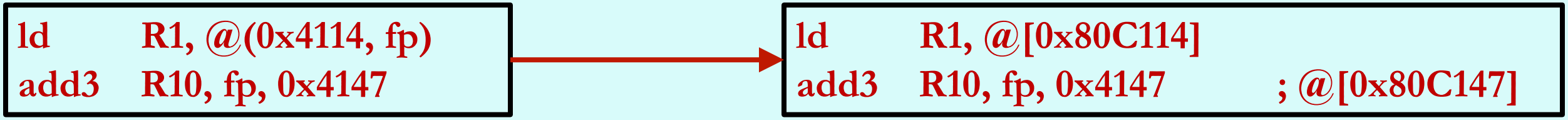

File Edit Jump Search View Options Windows Help
Library function Data Regular function Unexplored Instruction External symbol
Functions wind...
Function name
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
ROM:0000
ROM:0000 ; Segment type: Pure code
ROM:0000 .CSEG ; ROM
ROM:0000 C09A .dw 0xC09A ; Ü+
ROM:0001 C0B4 .dw 0xC0B4 ; ¦+
ROM:0002 C0B3 .dw 0xC0B3 ; ¦+
ROM:0003 C0B2 .dw 0xC0B2 ; ¦+
ROM:0004 C0B1 .dw 0xC0B1 ; ¦+
ROM:0005 C0B0 .dw 0xC0B0 ; ¦+
ROM:0006 C0AF .dw 0xC0AF ; »+
ROM:0007 C0AE .dw 0xC0AE ; «+
ROM:0008 C0AD .dw 0xC0AD ; ¡+
ROM:0009 C0AC .dw 0xC0AC ; ¼+
ROM:000A C0AB .dw 0xC0AB ; ½+
ROM:000B C0AA .dw 0xC0AA ; ¬+
ROM:000C C0A9 .dw 0xC0A9 ; ¬+
ROM:000D C0A8 .dw 0xC0A8 ; ¿+
ROM:000E C0A7 .dw 0xC0A7 ; º+
ROM:000F C0A6 .dw 0xC0A6 ; ª+
ROM:0010 C0A5 .dw 0xC0A5 ; Ñ+
ROM:0011 C0A4 .dw 0xC0A4 ; ñ+
ROM:0012 C0A3 .dw 0xC0A3 ; ú+
ROM:0013 C0A2 .dw 0xC0A2 ; ó+
ROM:0014 C0A1 .dw 0xC0A1 ; í+
ROM:0015 C0A0 .dw 0xC0A0 ; á+
ROM:0016 C09F .dw 0xC09F ; ▪+
ROM:0017 C09E .dw 0xC09E ; P+
ROM:0018 C09D .dw 0xC09D ; ¥+
ROM:0019 C09C .dw 0xC09C ; £+
ROM:001A 616E .dw 0x616E ; na
ROM:001B 006E .dw 0x6E ; n
ROM:001C 6E69 .dw 0x6E69 ; in
ROM:001D 0066 .dw 0x66 ; f
ROM:001E 4000 .dw 0x4000 ; @
ROM:001F 107A .dw 0x107A ; z
ROM:0020 5AF3 .dw 0x5AF3 ; =Z
00000000 00000000: ROM:0000 (Synchronized with Hex View-1)
Output window
The initial autoanalysis has been finished.
Python>
Python
AU: idle Down Disk: 6GB

define_code_functions.py

```
#################### USER DEFINED VALUES ####################
# Enter a regular expression for how this architecture usually
# begins and ends functions. If the architecture does not
# dictate how to start or end a function use r".*" to allow
# for any instruction.
#
# 8051 Architecture Prologue and Epilogue
smart_prolog = re.compile(r".*")
smart_epilog = re.compile(r"reti{0,1}")

# PIC18 Architecture Prologue and Epilogue
#smart_prolog = re.compile(r".*")
#smart_epilog = re.compile(r"return  0")

# Mitsubishi M32R Architecutre Prologue and Epilogue
#smart_prolog = re.compile(r"push +lr")
#smart_epilog = re.compile(r"jmp +lr.*")

# Texas Instruments TMS320C28x
#smart_prolog = re.compile(r".*")
#smart_epilog = re.compile(r"lretr")

# AVR
#smart_prolog = re.compile(r"push +r")
#smart_epilog = re.compile(r"reti{0,1}")
#############################################################
```

how ida python helps -- analysis

- find_mem_accesses.py
  - identifies all memory accesses for architectures such as 8051 which use a variable to access memory (DPTR)
- data_offset_calc.py
  - find the memory address accesses and
    - 1) create a data cross-reference to the memory address
    - 2) write the value at the memory address as a comment at the instructions
    - 3) create a file with all of the accesses memory address and the instructions accessing them



File Edit Jump Search View Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name

sub_2F38
sub_2F80
sub_3054
sub_3098
sub_30B0
sub_3308
sub_3548
sub_3678
sub_3710
sub_3754
sub_3774
sub_37C8
sub_3804
sub_38F8
sub_396C
sub_3D34
sub_3D54
sub_3D64
sub_3DB0
sub_3FCC
sub_4008
sub_409C
sub_4118
sub_4268
sub_4310
sub_431C
sub_4350
sub_44A0
sub_4590
nullsub_1
nullsub_2
sub_4694
sub_4780
sub_4B78
sub_4C8C

Line 84 of 2679

IDA View-A | Hex View-1 | Structures | Enums | Imports | Exports

```
ROM_:000039BC 60 FF                   ldi8    R0, #-1
ROM_:000039BE 50 10                   srli    R0, #0x10
ROM_:000039C0 A1 CD 41 14             ld      R1, @(0x4114, fp)
ROM_:000039C4 A2 CD 41 18             ld      R2, @(0x4118, fp)
ROM_:000039C8 01 22                   sub     R1, R2
ROM_:000039CA 12 80                   mv      R2, R0
ROM_:000039CC 01 52                   cmpu    R1, R2
ROM_:000039CE 7D 03                   bnc.s   loc_39D8
ROM_:000039D0 10 81                   mv      R0, R1
ROM_:000039D2 7F 02                   bra.s   loc_39D8
ROM_:000039D4                 ; ---------------------------------------------------------------
ROM_:000039D4
ROM_:000039D4                 loc_39D4:                               ; CODE XREF: sub_396
ROM_:000039D4 60 00 F0 00             ldi8    R0, #0 || nop
ROM_:000039D8
ROM_:000039D8                 loc_39D8:                               ; CODE XREF: sub_396
ROM_:000039D8                                                         ; sub_396C+66↑j
ROM_:000039D8 81 C8 FF FF             and3    R1, R8, #-1
ROM_:000039DC 80 C0 FF FF             and3    R0, R0, #-1
ROM_:000039E0 00 51                   cmpu    R0, R1
ROM_:000039E2 7D 06                   bnc.s   loc_39F8
ROM_:000039E4
ROM_:000039E4                 loc_39E4:                               ; CODE XREF: sub_396
ROM_:000039E4 84 AF 00 2B             add3    R4, sp, #0x2B
ROM_:000039E8 60 05                   ldi8    R0, #5
ROM_:000039EA 20 04                   stb     R0, @R4
ROM_:000039EC 61 FF F0 00             ldi8    R1, #-1 || nop
ROM_:000039F0 A1 4D 41 18             st      R1, @(0x4118, fp)
ROM_:000039F4 FF 00 00 BC             bra.l   loc_3CE4
ROM_:000039F8                 ; ---------------------------------------------------------------
ROM_:000039F8
ROM_:000039F8                 loc_39F8:                               ; CODE XREF: sub_396
ROM_:000039F8 A0 CD 41 10             ld      R0, @(0x4110, fp)
ROM_:000039FC A1 CD 41 18             ld      R1, @(0x4118, fp)
ROM_:00003A00 00 A1 F0 00             add     R0, R1 || nop
ROM_:00003A04 A0 4F 00 24             st      R0, @(0x24, sp)
ROM_:00003A08 8A AD 41 71             add3    R10, fp, #0x4171
ROM_:00003A0C 19 88 F0 00             mv      R9, R8 || nop
ROM_:00003A10 A8 2F 00 22             sth     R8, @(0x22, sp)
ROM_:00003A14 A1 BD 41 22             lduh    R1, @(0x4122, fp)
ROM_:00003A18 62 01 F0 00             ldi8    R2, #1 || nop
```

000039D8 000039D8: sub_396C:loc_39D8 (Synchronized with Hex View-1)

Output window

```
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Python>
Start Addr: 81920 End Addr: 733680
[make_code_functions] Running script on 0x14000
to 0xb31f0
[define_data_as_types.py] STARTING. start_addr:
0xB31F0, end_addr: 0xBFFFF, data_size: 4
[define_data_as_types.py] Undefined all data
between 0xB31F0 and 0xBFFFF
[define_data_as_types.py] Defining all data as
size 4
[define_data_as_types.py] FINISHED.
```

Python

AU: idle Down Disk: 6GB

data_offset_calc.py

```
operand = GetOpnd(curr_addr, 1)
-----------------------
if (offset):
    if '-' in operand :
      new_opnd = offset_var_value - int(offset[0], 16)
    else:
      new_opnd = offset_var_value + int(offset[0], 16)
    OpAlt(curr_addr, 1, new_opnd_display % new_opnd)
    result = add_dref(curr_addr, new_opnd, dr_T)
-----------------------
MakeComm(curr_addr, '0x%08x' % new_opnd)
-------------------------
curr_addr = NextHead(curr_addr)
```

index of operand to get

change how the operand is displayed

create a data cross-reference

dr_T: text
dr_R: read
dr_W: write
dr_O: offset

```
ld      R1, @(0x4114, fp)
add3    R10, fp, 0x4147
```

```
ld      R1, @[0x80C114]
add3    R10, fp, 0x4147         ; @[0x80C147]
```

how ida python helps -- annotate

- lable_funcs_with_no_xrefs.py
  - check for functions with no cross-references to them and annotate their function name with a “noXrefs” prefix
- identify_port_use_locations.py
  - searches all code for pin/port operations based on the defined regex for the architecture and lists all references in a text file and optionally labels each function

ida python functions used

AskAddr
AskFile
AskLong
AskYN
GetDisasm
GetFunctionAttr
GetFunctionName
GetOperandValue
GetOpnd
MakeByte
MakeCode
MakeComm
MakeDword
MakeFunction
MakeName
MakeStr
MakeUnkn
MakeWord
Warning
OpAlt
add_dref*
NextFunction
NextHead
PrevHead
FindUnexplored
XrefsTo*
isCode(GetFlags())
Byte
Word

all can be found in the idc module except (*)

what's next?

- ida python embedded toolkit: https://github.com/maddiestone/IDAPythonEmbeddedToolkit
- other script ideas
  - architecture independent CAN or serial identifiers
  - integrate and automate more of the triage processes
  - segment creation
  - automate architecture selection for scripts
  - other manners to display information
  - more robust examples and docs

thank you

maddie stone
madeline.stone@jhuapl edu
@maddiestone