Introduction to Software Reverse Engineering with Ghidra
Session 4: Ghidra Features

Hackaday U

Matthew Alt

#Outline: Advanced Features

- Ghidra Features
    - Loading external libraries
    - Patch analysis / diffing
    - Checksum Generation
    - P-Code Review
- Ghidra Extensions
    - Setting up a development environment
    - Python Scripting
- Course Wrap-up

#Course Administration

- This is our final session for the course!
  - Feedback and thoughts on course are welcome!
  - Fill out the google form to provide more feedback
    - (released later this week!)

- Office hours will be Thursday at 6:00 ET

- Questions can also be submitted through:
  - Hackaday.io chat room
  - Hackaday messaging

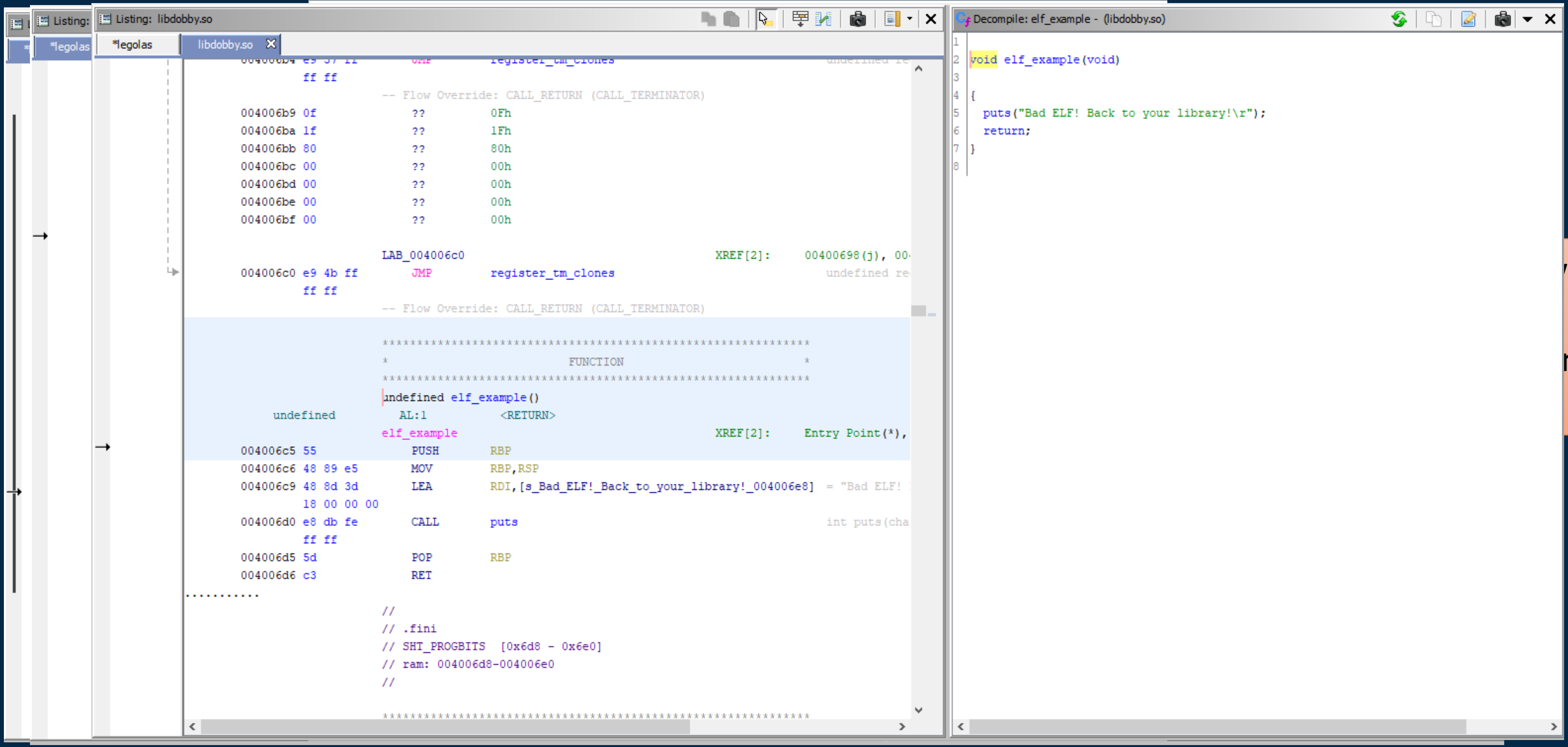
#Session Goals

- Review more Ghidra features
  - Patch analysis
  - Memory Manager
  - Extension via scripting
  - P-Code and analysis
- Ghidra Development and Scripting
  - Setting up an Eclipse based development environment
  - Scripting examples
- Course Wrap-up and conclusion
  - Review materials we’ve covered
  - Take final questions

#External Libraries

- Ghidra can be used to analyze external libraries utilized by your target binary
  - Shared objects, DLLs, etc
- Ghidra can import and analyze these external libraries
  - This can be done at load time or after initial analysis
- Loading external libraries can assist with auto analysis
  - Type recognition and function prototype generation

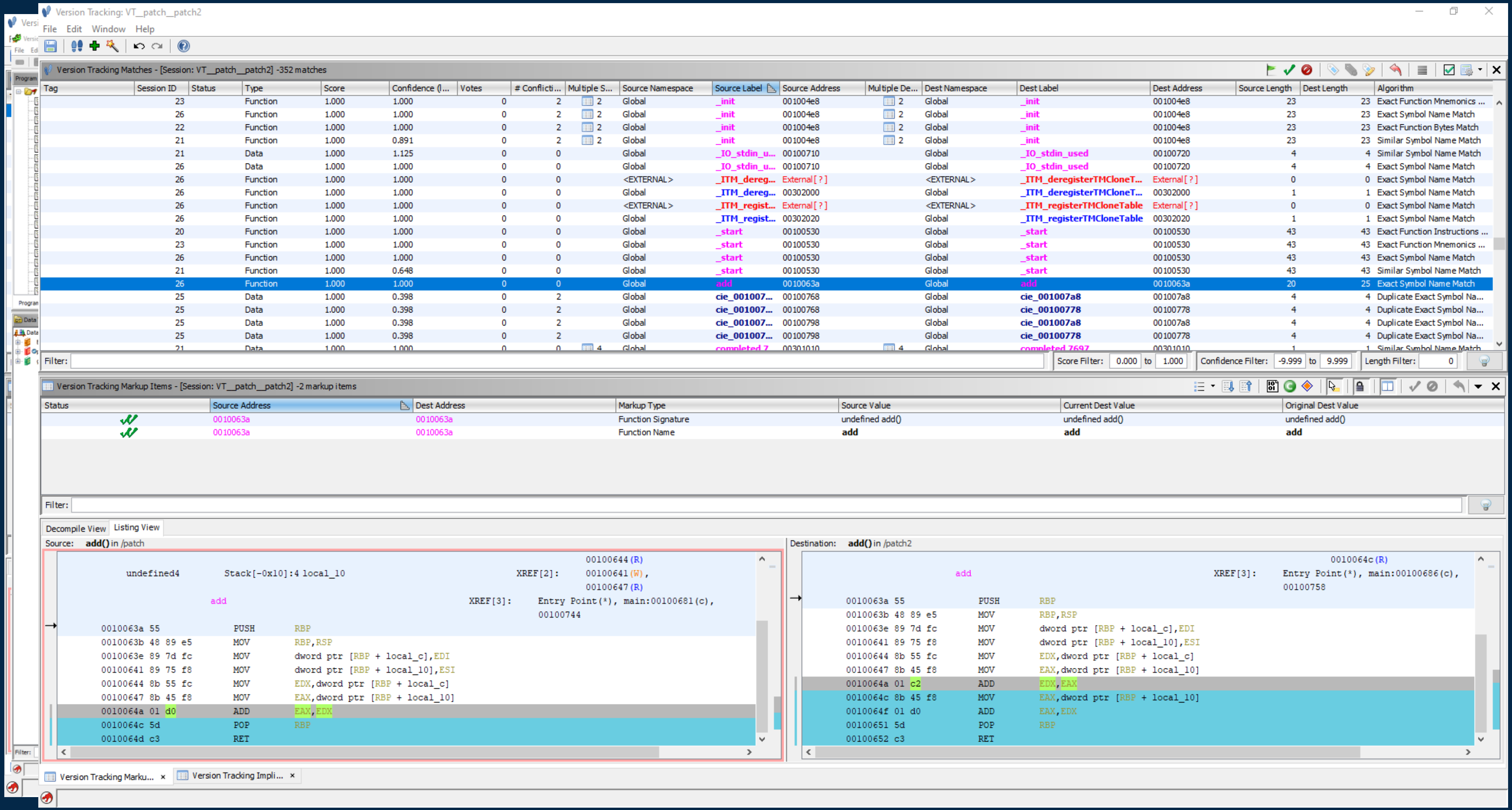
#External Libraries: Example



#Ghidra Tip: Patch Diffing

- Ghidra allows for version tracking of all imported binaries
  - This makes collaboration when reversing simpler
- When using the version tracker, files are checked in and out of the project
  - Similar to a git workflow
- This version tracking feature can also be used to track patches and changes to binaries

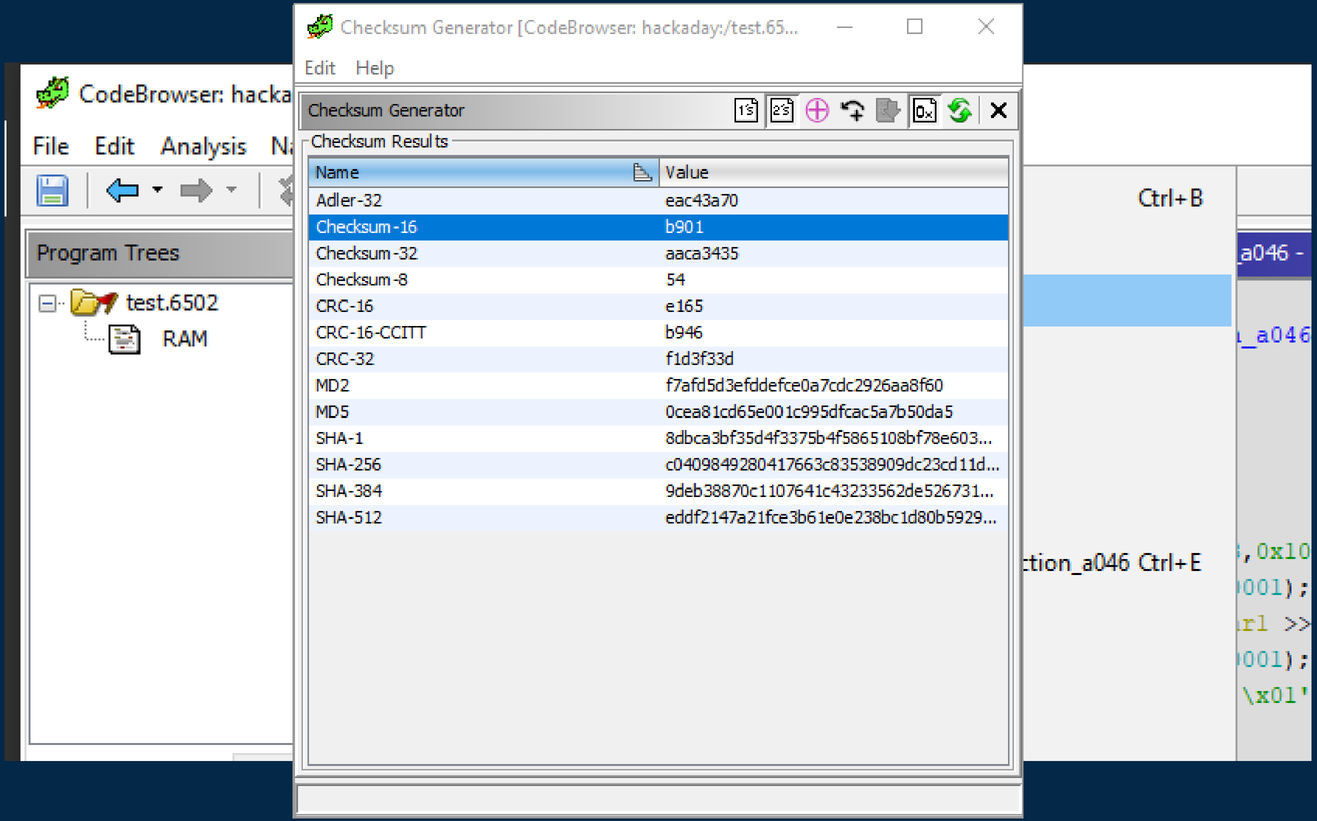
#Patch Diffing: Example



#Ghidra Tip: Checksum Tool

- Ghidra contains a built in checksum generator
- Multiple types of checksums can be generated
  - CRC-16/32
  - MD2/5
  - SHA1/256/384
- Checksums can have various operations applied to them
  - 1’s/2’s complement
  - Recompute with carry / xor

#Ghidra Tip: Checksum Tool

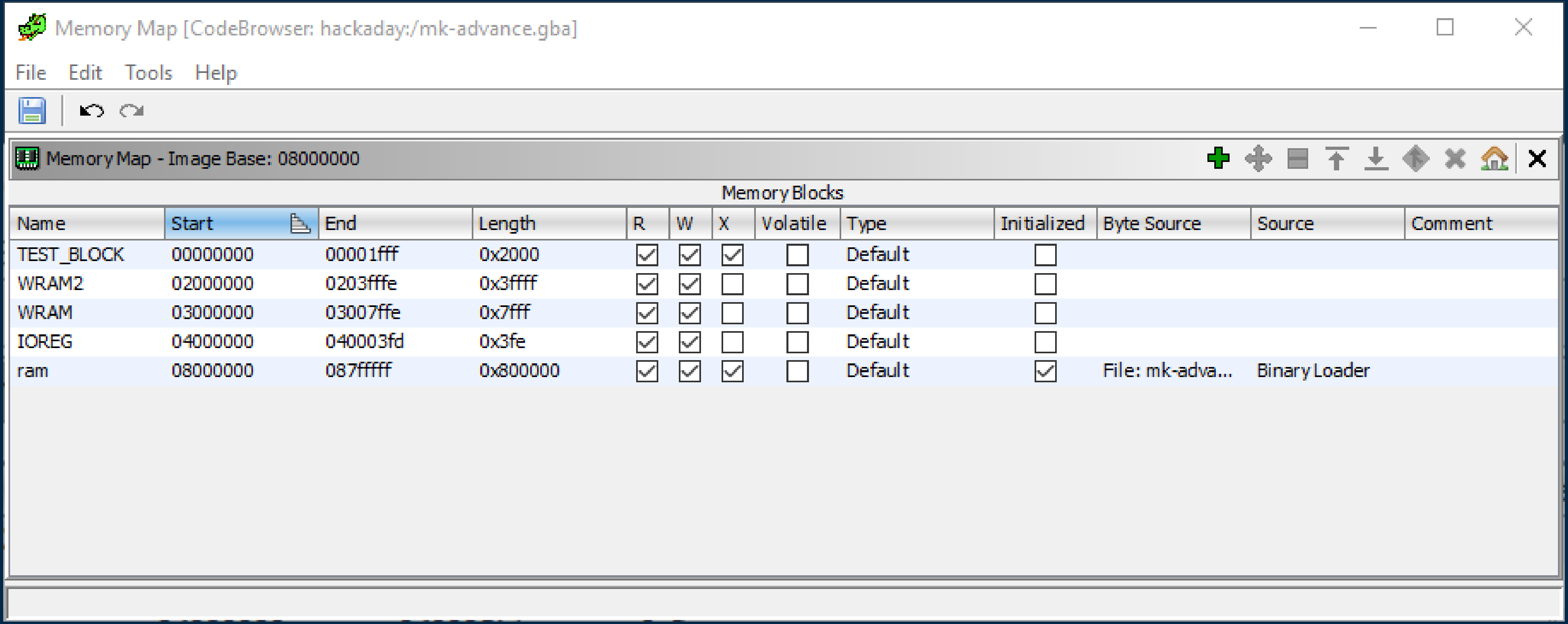


#Ghidra Tip: Memory Manager

- Memory regions can be added manually via Ghidra's Memory Map tool
  - This is often used when looking at firmware images
  - Programs can be rebased from this tool as well

- Each memory region has the following attributes in the manager
  - Start address
  - Size/End address
  - Read / Write / Execute
  - Overlay

#Ghidra Tip: Memory Manager



#Ghidra Tip: Memory Manager

- Binary files can be imported into memory regions
  - This is useful for RAM dumps if you can acquire them!
  - File -> Add to Program
- Some memory regions can be set up as overlay blocks
  - Useful for older consoles with regions that can be bank switched
- The memory manager functionality can also be instrumented via Ghidra's API
  - This is how loaders automatically generate appropriate regions!

#Ghidra: Internals

- One of Ghidra's most powerful features is having a decompiler for each processor that it supports

- To implement a processor module in Ghidra, one must first write a processor module in SLEIGH

- This processor module will handle both:
  - Decoding of bytes to assembly code (disassembly)
  - Generation of Pcode
  - Generation of decompilation based on Pcode

#SLEIGH: Overview

- SLEIGH specifies the translation from assembly code to P-Code
- SLEIGH is the language that is used to create processor modules for Ghidra
- SLEIGH modules define the processor’s instructions, registers and features
  - Examples are found at Ghidra/Processors
- SLEIGH modules are used for both disassembly and P-Code creation
  - P-Code is what is analyzed by the decompiler

#Sleigh: 6502 Example

- Each processor definition will consist of the following files:
  - CPU.cspec
  - CPU.ldefs
  - CPU.pspec
  - CPU.slaspec
- LDEFS: Language definitions
- PSPEC: Processor Specification
- CSPEC: Compiler Specification
- SLASPEC: SLEIGH Specification

#Sleigh: 6502 Example - LDEFS

```
<language processor="6502"
          endian="little"
          size="16"
          variant="default"
          version="1.0"
          slafile="6502.sla"
          processorspec="6502.pspec"
          manualindexfile="../manuals/6502.idx"
          id="6502:LE:16:default">
  <description>6502 Microcontroller Family</description>
  <compiler name="default" spec="6502.cspec" id="default"/>
  <external_name tool="IDA-PRO" name="m6502"/>
  <external_name tool="IDA-PRO" name="m65c02"/>
</language>
```

Language definitions include endianness, size – as well as definitions for the rest of the specification files for this particular processor.

#Sleigh: 6502 Example – PSPEC

```
<default_symbols>
    <symbol name="PORTA" address="0"/>
    <symbol name="PORTB" address="1"/>
    <symbol name="PORTC" address="2"/>
    <symbol name="PORTD" address="3"/>
    <symbol name="DDRA" address="4"/>
    <symbol name="DDRB" address="5"/>
    <symbol name="DDRC" address="6"/>
    <symbol name="DDRD" address="7"/>
    <symbol name="SPCR" address="A"/>
    <symbol name="SPSR" address="B"/>
    <symbol name="SPDR" address="C"/>
    <symbol name="BAUD" address="D"/>
    <symbol name="SCCR1" address="E"/>
    <symbol name="SCCR2" address="F"/>
    <symbol name="SCSR" address="10"/>
    <symbol name="SCDAT" address="11"/>
    <symbol name="TCR" address="12"/>
    <symbol name="TSR" address="13"/>
    <symbol name="ICHR" address="14"/>
    <symbol name="ICLR" address="15"/>
    <symbol name="OCHR" address="16"/>
    <symbol name="OCLR" address="17"/>
    <symbol name="CHR" address="18"/>
    <symbol name="CLR" address="19"/>
    <symbol name="ACHR" address="1A"/>
    <symbol name="ACLR" address="1B"/>
    <symbol name="NMI" address="FFFA" entry="true" type="code_ptr"/>
    <symbol name="RES" address="FFFC" entry="true" type="code_ptr"/>
    <symbol name="IRQ" address="FFFE" entry="true" type="code_ptr"/>
  </default_symbols>
```

```
<default_memory_blocks>
  <memory_block name="LOW_RAM" start_address="0x0000" length="0x0100" initialized="false"/>
  <memory_block name="STACK" start_address="0x0100" length="0x0100" initialized="false"/>
</default_memory_blocks>
```

```
<programcounter register="PC"/>
```

Processor Specification is used to define processor specific information such as:

- Program Counter
- Reset Vectors
- Interrupt Handlers
- Memory Mapped IO

#Sleigh: 6502 Example – CSPEC

```xml
<compiler_spec>
  <global>
    <range space="RAM"/>
  </global>
  <stackpointer register="SP" space="RAM" growth="negative"/>
  <returnaddress>
    <varnode space="stack" offset="1" size="2"/>
  </returnaddress>
  <default_proto>
    <prototype name="__stdcall" extrapop="2" stackshift="2" strategy="register">
      <input>
        <pentry minsize="1" maxsize="1">
          <register name="A"/>
        </pentry>
        <pentry minsize="1" maxsize="1">
          <register name="X"/>
        </pentry>
        <pentry minsize="1" maxsize="1">
          <register name="Y"/>
        </pentry>
      </input>
      <output>
        <pentry minsize="1" maxsize="1">
          <register name="A"/>
        </pentry>
      </output>
      <unaffected>
        <register name="SP"/>
      </unaffected>
    </prototype>
  </default_proto>
</compiler_spec>
```

The Compiler Specification defines things such as:

- Sizes for various data types
- Alignment rules / directives
- Stack growth / behavior
- Function prototypes
- Calling conventions

#Sleigh: 6502 Example – SLASPEC

```
define endian=little;
define alignment=1;

define space RAM      type=ram_space      size=2   default;
define space register type=register_space size=1;

define register offset=0x00  size=1  [ A X Y P ];
define register offset=0x20  size=2  [ PC      SP
define register offset=0x20  size=1  [ PCL PCH S S
define register offset=0x30  size=1 [ N V B D I Z C ];   # status bits
```

```
# Immediate
OP1: "#"imm8    is bbb=2; imm8        { tmp:1 = imm8; expor
# Zero Page
OP1: imm8       is bbb=1; imm8        { export *:1 imm8; }

:STA OP1      is (cc=1 & aaa=4) ... & OP1
{
    OP1 = A;
    resultFlags(A);
}
```

The SLASPEC file contains things such as:
- Alignment definitions
- Address space definitions
- Register definitions
- Instruction definitions

SLASPEC defines how instructions are decoded / disassembled:
- Defines the parameters for disassembly
- Defines how instructions are shown in the listing

#P-Code: Overview

- Designed to be capable of modeling a general purpose processor

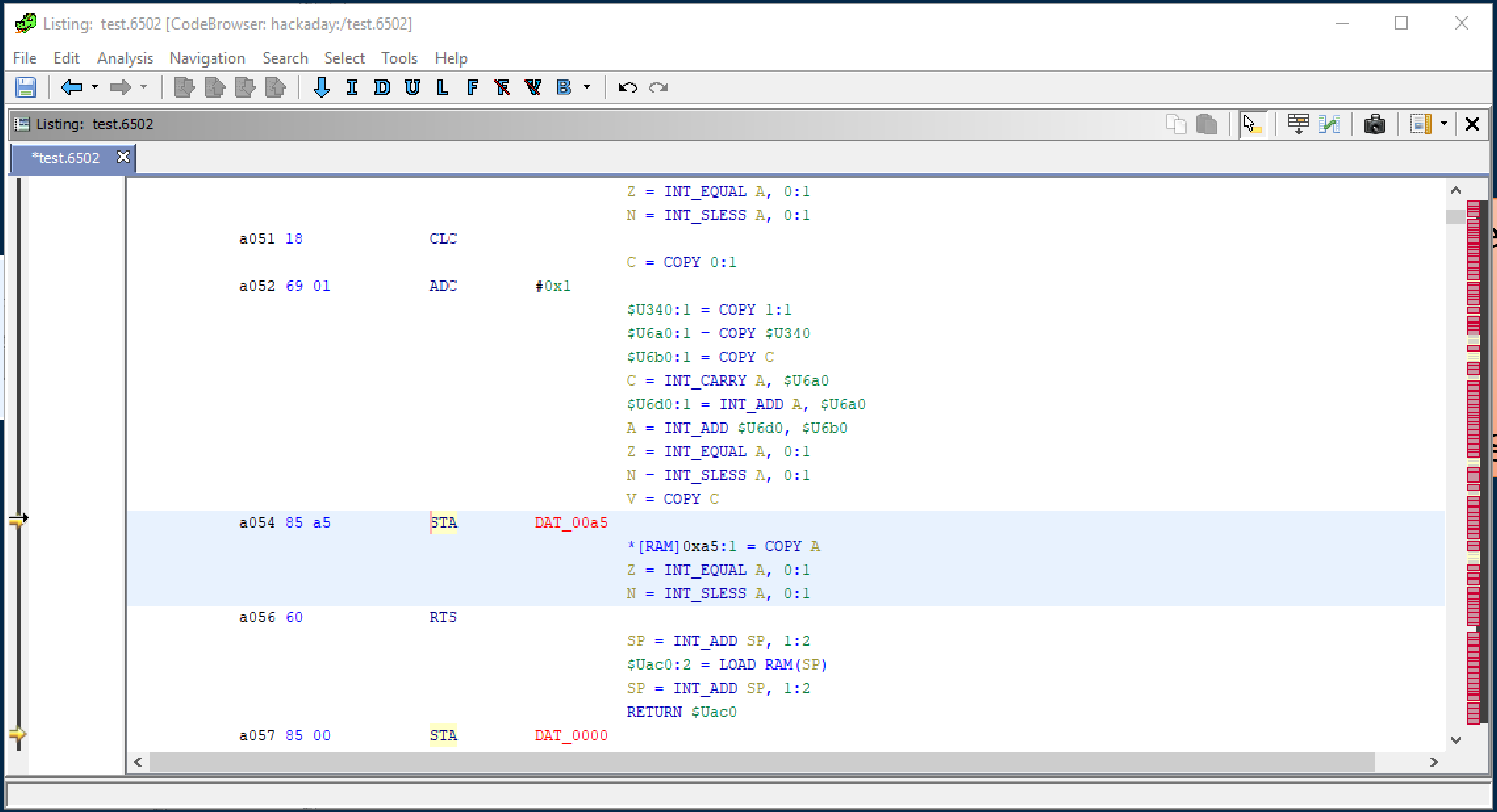
- P-Code is generated during the initial analysis process

- P-Code is a register transfer language
  - Used as an abstraction layer for CPU specific functions
  - Intermediate representation of assembly code

- P-Code is used to generate decompiled code

#P-Code: Example

a054 85 a5



ery detailed

g

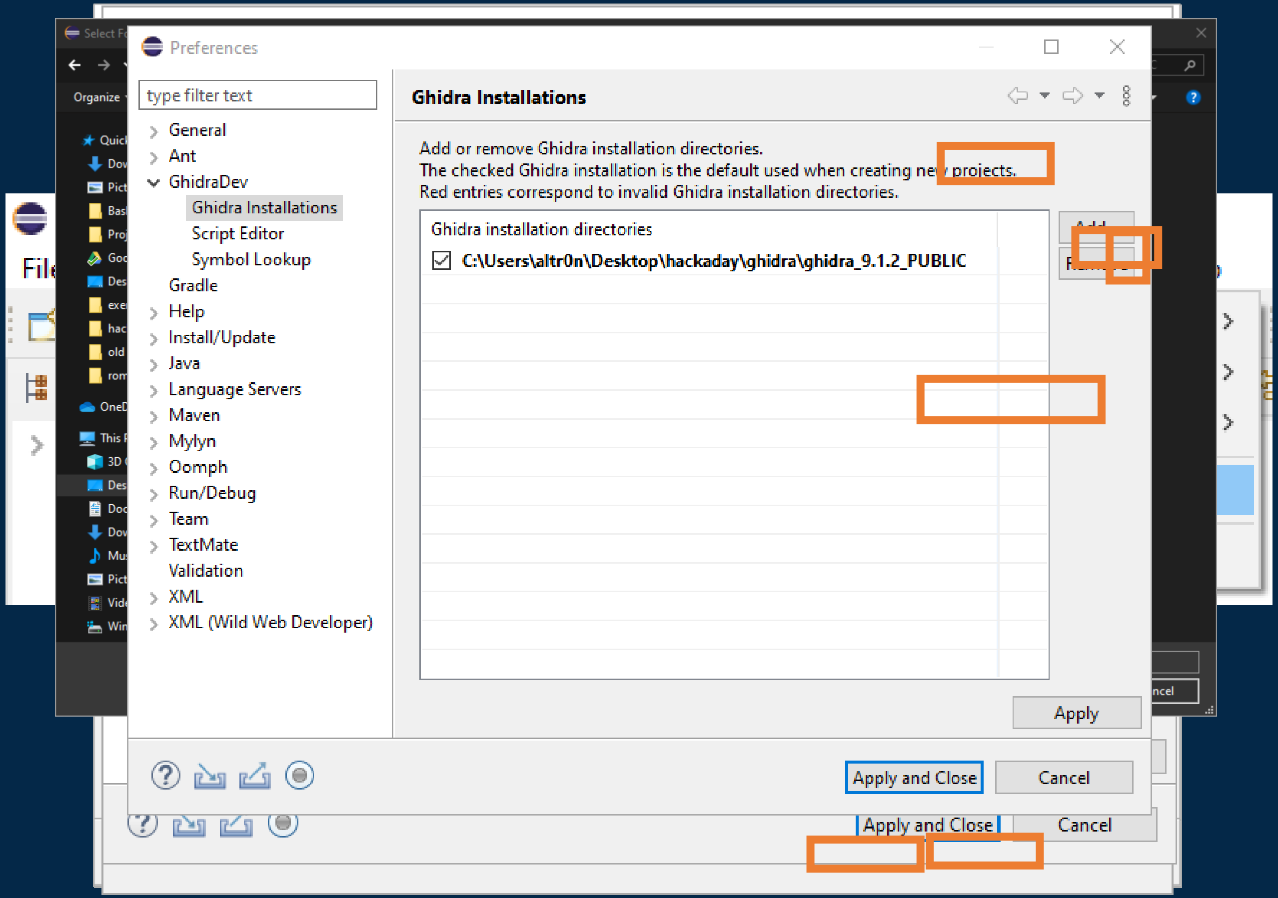
#P-Code: Exercise

- Load the exercises in session-four/exercises/script-exercises/
  - PPC
  - ARM
  - I386
  - X86_64
- Examine the P-code for each example
  - How are they different?
  - Are they similar in any way?

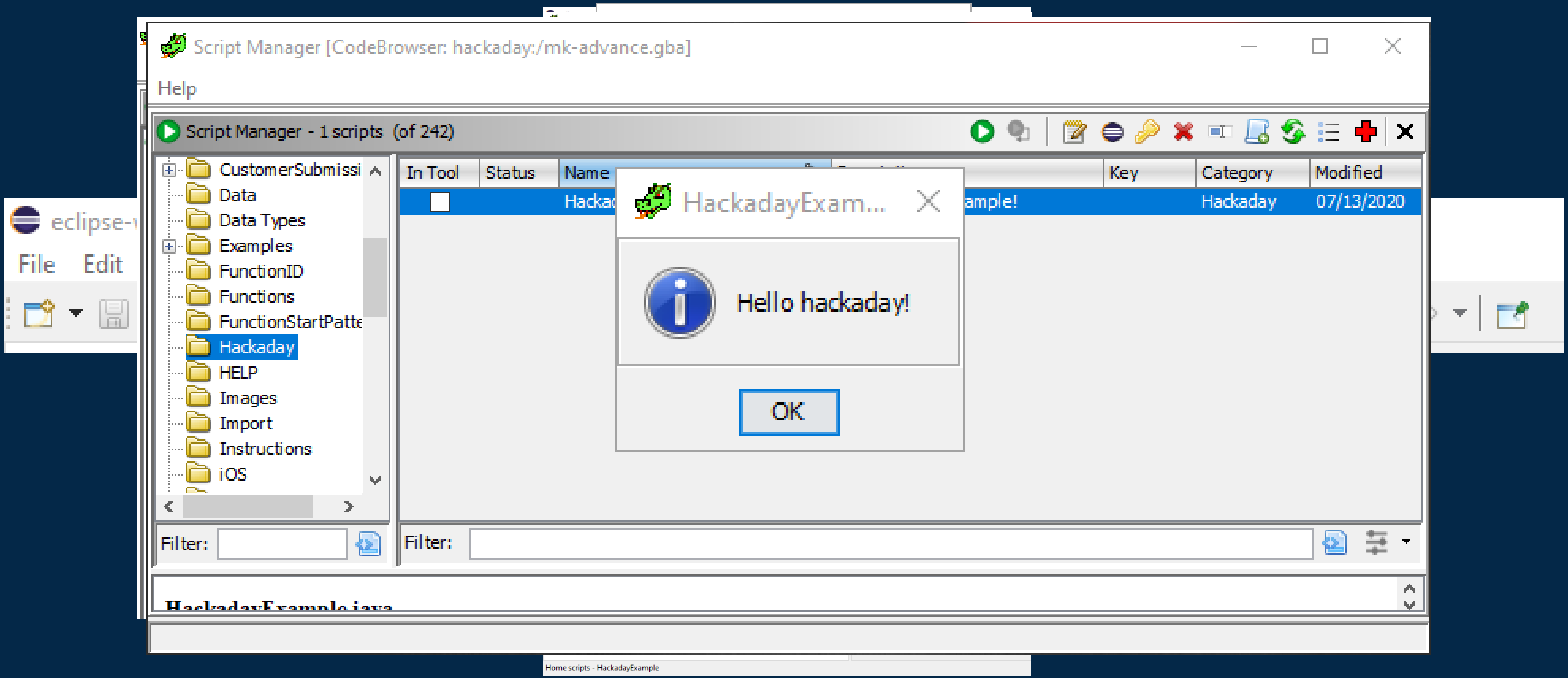
#Ghidra Extension: Eclipse

- Ghidra can be extended via plugins
  - Loaders
  - Scripts
  - Modules
- These extensions can be developed in Eclipse using the GhidraDev plugin included with Ghidra
- Ghidra provides both a Java and Python API

#Ghidra Extension: Eclipse



#Ghidra Script Project Creation



#Ghidra Scripts: Python

- Ghidra scripts can be written in Python as well as Java
  - Requires the pydev plugin for eclipse integration
- Ghidra also features a python shell
  - Window -> Python
- The Python interpreter can utilize the FlatProgramAPI
  - This is what we will focus on!

#Ghidra Scripts: Globals

- Ghidra creates multiple global variables for use while scripting
  - currentAddress
  - currentHighlight
  - currentProgram
  - currentSelection
  - currentLocation
- All of these can be used to determine information relevant to your current cursor location!

#Ghidra Scripts: Addresses

- Ghidra utilizes the Address datatype heavily within it's API
  - currentAddress returns an Address object
- Address objects can NOT be treated like integers
  - Contain special function to add and subtract
  - .add() .subtract(), etc
- Address objects can be created using the AddressFactory
  - currentProgram.getAddressFactory.getAddress("0x10000")

#Scripting: Data

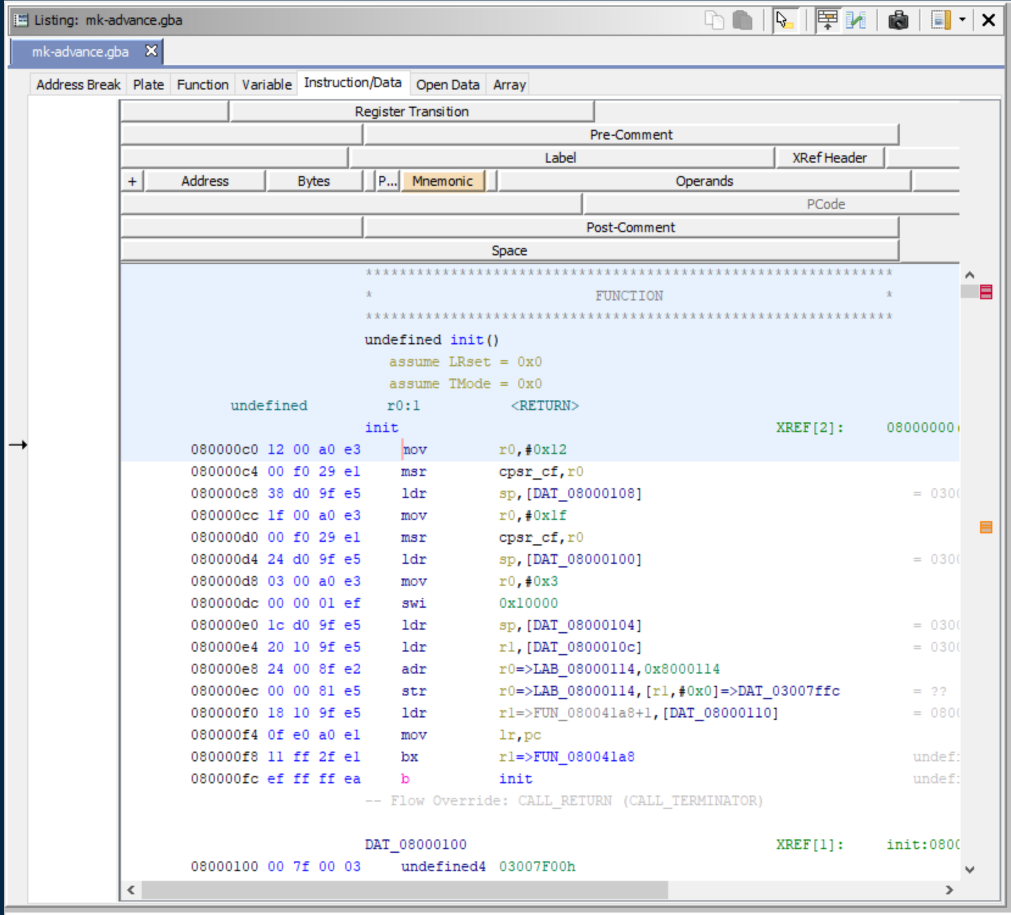
- Data can be generated automatically via scripting
  - createByte(currentAddress)
  - createChar(currentAddress)
  - createDWord(currentAddress)
  - createAsciiString(currentAddress)

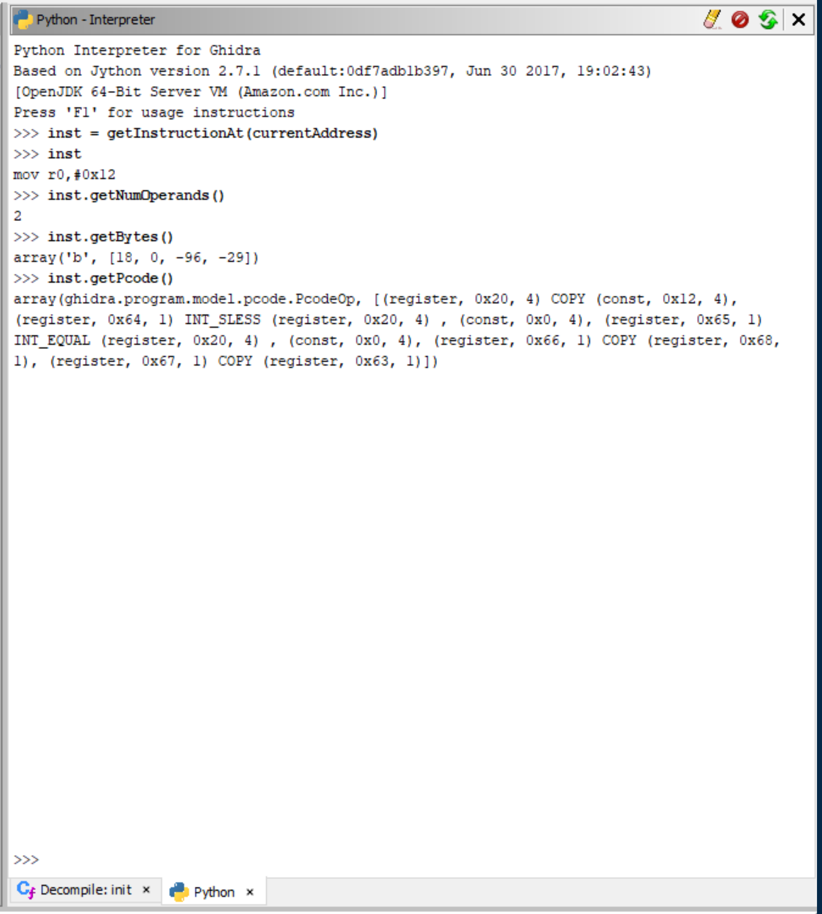
- These functions all take a Ghidra address object as an argument
  - Eq: createDWord(currentAddress)

#Scripting: Instructions

- Ghidra scripts can parse and operate at the instruction level
- getInstructionAt(address)
  - Used to get an Instruction object representing the instruction at that address
- From an instruction object multiple instruction components can be derived
  - Number of operands
  - P-code representation
  - Operand objects (scalars,etc)

#Scripting: Instructions





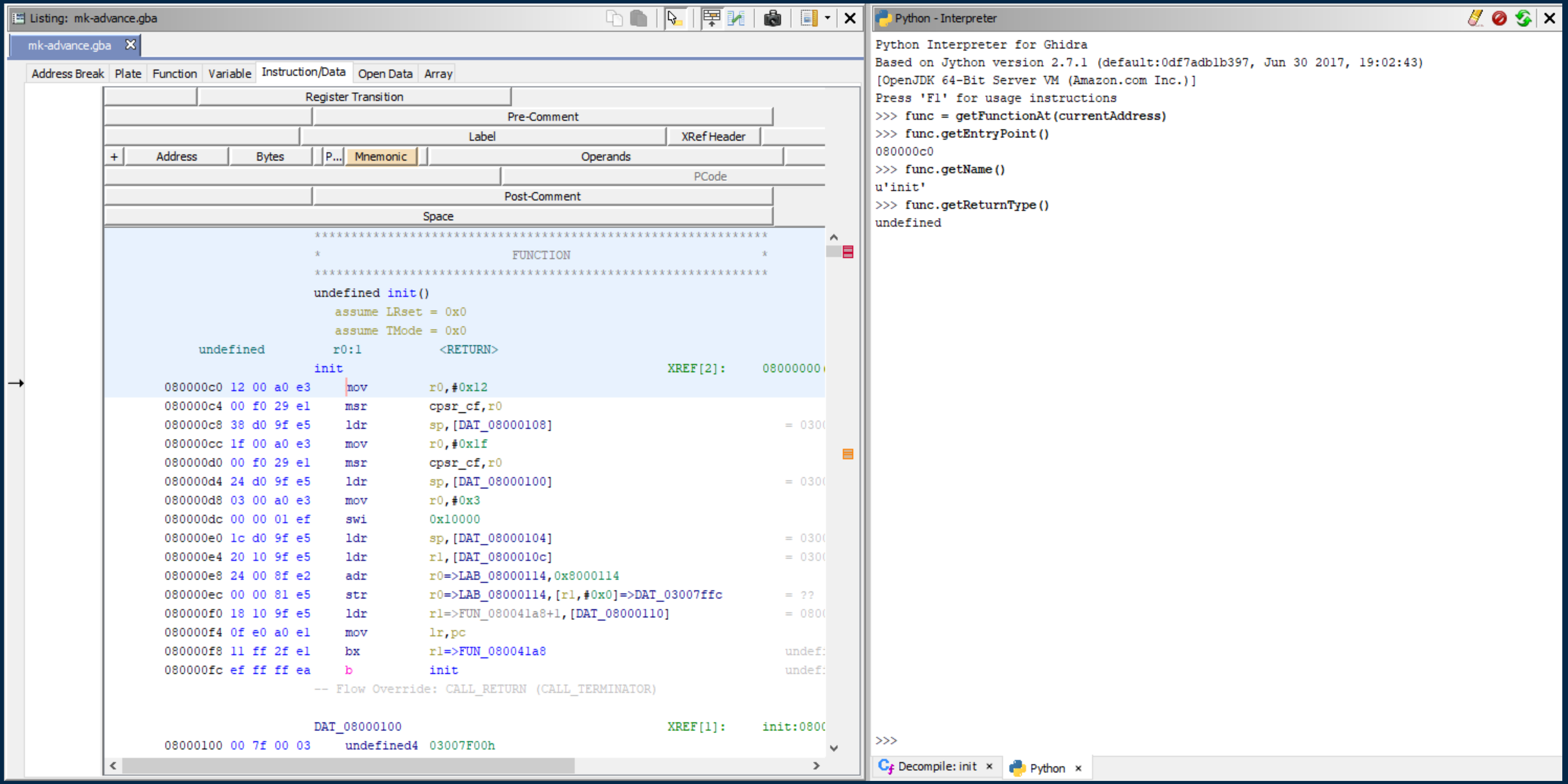
#Scripting: Functions

- Ghidra's FlatProgramAPI can be used to create and modify functions
  - Set signature, parameters, etc

- Functions can be accessed by providing an address:
  - Func = getFunctionAt(address)

- Functions can also be created at a specified address:
  - createFunction(address)

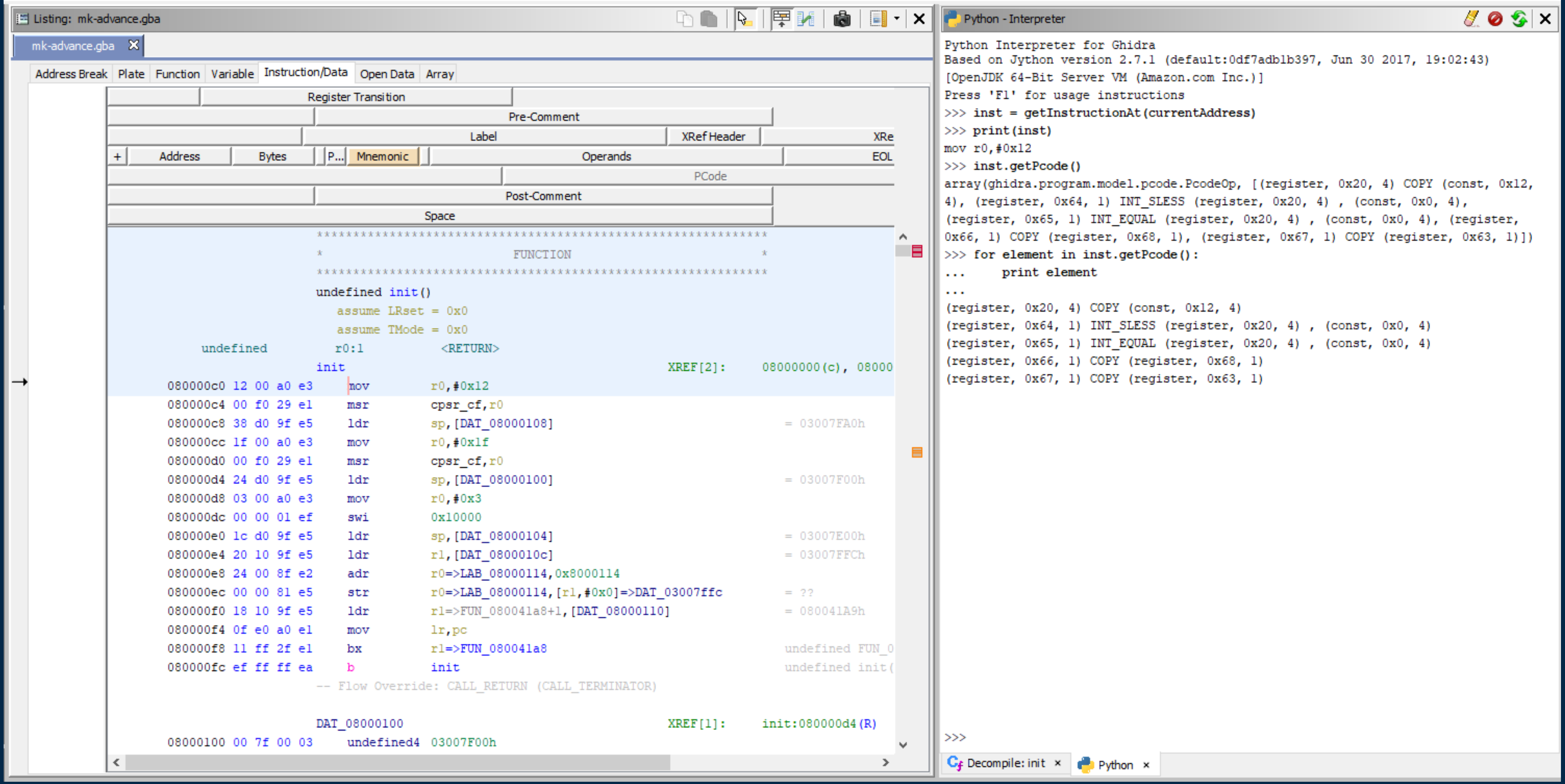
#Scripting: Functions



#Scripting: Pcode Extraction

- Pcode objects can be accessed from instruction objects
  - Inst.getPcode()
- PcodeOps describe the Pcode operations for a given instruction
- PcodeOps can be emulated using Ghidra's Pcode emulator

#Scripting: Pcode Extraction



#Final Exercises

- The final exercises for this course will be pushed shortly after class today
  - Session-four/exercises/crackmes.one
- Exercises have been pulled from crackmes.one
  - They are of ascending difficulty
  - Use this site for more reversing practice!
- Source for the exercises will be released after office hours this week

#Wrap Up

- This will be the final video for this series
  - Thank you for watching and participating!

- If you have questions or commentary, please let us know for the office hour!
  - You can also fill out the google form to give us more feedback

- Feel free to use the course chatroom to ask RE questions and Ghidra questions in the future

#Questions

?