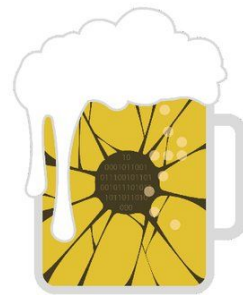


# Implementing a New CPU Architecture for Ghidra

@guedou

BeeRump



Why?

# Toshiba FlashAir W-03



Toshiba MeP-c4

# My MeP & FlashAir Tools

**cea-sec/miasm - MeP architecture in miasm**

assembly, disassembly, emulation

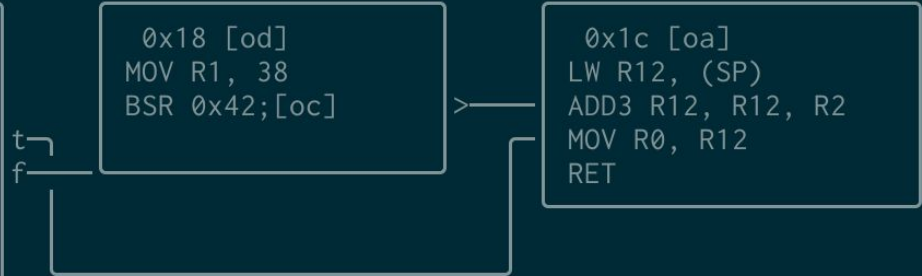
**guedou/r2m2 - miasm plugin for radare2**

graphical interface, emulation, tools

**guedou/flashre - tools to reverse FlashAir cards**

dump, telnet, fake updates...

```
[0x0]
(fcn) fcn.00000000 36
      fcn.00000000 (int32_t arg_ffffffch);
; arg int32_t arg_ffffffch @ sp+0xfffffc
LDC R0, LP
ADD SP, -4
MOV R12, 0
SW R12, (SP)
LW R12, (SP)
ADD3 R12, R12, R1
SW R12, (SP)
MOV R12, 0
MOV R11, 41
LW R10, (SP)
BNE R10, R11, 0x1C;[oa]
```



**Missing Tool: a Decompiler**  
aka output C instead of assembly

# Open Source Decompilers

**many available**

reko, snowman, r2dec, radeco, retdec...

**architecture dependent**

must describe some MeP specificities

# New Open-Source RE Tool



developed by the NSA

revealed in the Vault7 leak

released in March 2019

<https://github.com/NationalSecurityAgency/ghidra>

many features

disassembly, graphing, scripting, extensions, decompiling...



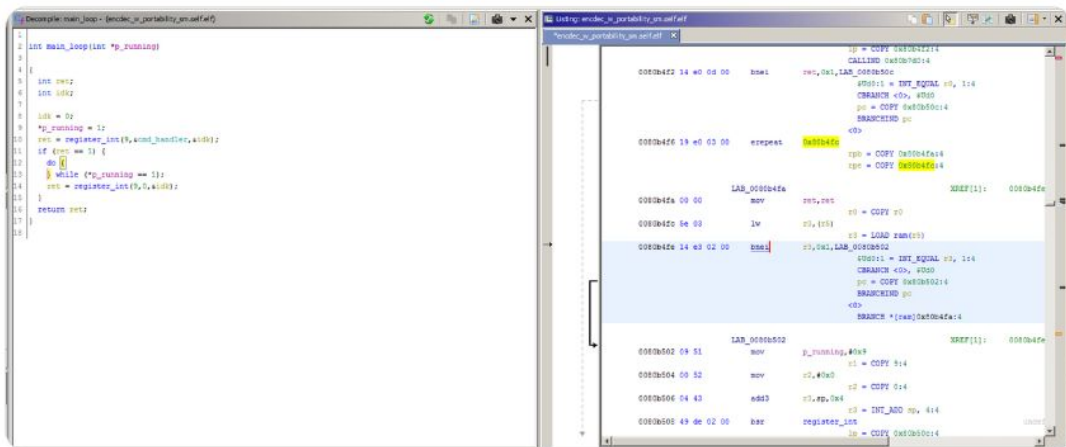


**everywhere**  
@pomfpomfpomf3

Following



## Adding Toshiba MeP to GHIDRA - can your IDA do this?



4:03 PM - 7 Mar 2019

16 Retweets 99 Likes



5



16



99



# **SLEIGH**

Ghidra Processor Specification Language

# SLEIGH?

## language derived from SLED

Specification Language for Encoding and Decoding  
architecture independent assembler ~~& disassembler~~

## ease defining instructions decoding & semantics

data-flow & decompilation analysis  
semantics converted to Ghidra IR (aka P-CODE)

## also a command-line tool

`$GHIDRA_HOME/support/sleigh`

# Mandatory Processor Structure

guedou/ghidra-processor-mep

```
$ tree Ghidra/Processors/MEP_C4/
```

```
Ghidra/Processors/MEP_C4/
```

```
|— data
|   |— languages
|       |— mep_c4.cspect
|       |— mep_c4.ldefs
|       |— mep_c4.pspec
|       |— mep_c4.sla
|       |— mep_c4.slaspec
|— Module.manifest
```

```
2 directories, 6 files
```

# Language Definitions - mep\_c4.lds

```
<language_definitions>
  <language processor="Toshiba MeP-c4"
    endian="little"
    size="32"
    variant="default"
    version="0.1"
    slafile="mep_c4.sla"
    processorspec="mep_c4.pspec"
    id="MEP_C4:LE:32:default">
    <description>Toshiba MeP-c4, little endian</description>
    <compiler name="default" spec="mep_c4.cspec" id="default"/>
  </language>
</language_definitions>
```

# Processor Specification - mep\_c4.pspec

PC, symbols (Reset, NMI handlers...)

```
<processor_spec>  
  <programcounter register="pc"/>  
</processor_spec>
```

# Compiler Specification - mep\_c4.cspec

```
<compiler_spec>
  <global>
    <range space="ram"/>
  </global>
  <stackpointer register="sp" space="ram"/>
  <default_proto>
    <prototype extrapop="0" stackshift="0" name="__stdcall">
      <input>
        <pentry minsize="1" maxsize="4">
          <register name="r1"/>
        </pentry>
      </input>
      <output>
        <pentry minsize="1" maxsize="4">
          <register name="r0"/>
        </pentry>
      </output>
    </prototype>
  </default_proto>
```

# SLEIGH Specification File - mep\_c4.slaspec

**compiled to mep\_c4.sla with sleigh**

XML version of mep\_c4.slaspec with P-CODE

## **five important concepts**

space - ram & register definition

register - names & aliases

token - instructions parts

variables - names to registers bindings

instruction - tokens composition & semantic



# Example #1

MeP-c4 16-bit MOV

```

# MOV Rn,Rm - 0000_nnnn_mmmm_0000

define register offset=0 size=4 [ r0 r1 ];

define token instr(16)
    major = (12, 15)
    rn = (8, 11)
    rm = (4, 7)
    minor = (0, 3)
;

attach variables [ rn rm ] [ r0 r1 ];

:mov rn, rm is major=0b0000 & rn & rm & minor=0b0000 {
    rn = rm;
}

```

## Example #2

MeP-c4 32-bit MOV (variant #1)

```
# MOV Rn,imm16 - 1100_nnnn_0000_0001 iii_iiii_iii_iii
```

```
define token instr(16)
```

```
    major = (12, 15)
```

```
    rn = (8, 11)
```

```
    minor8 = (0, 7)
```

```
;
```

```
define token ext(16)
```

```
    imm16 = (0, 15)
```

```
;
```

```
:mov rn, imm16 is major=0b1100 & rn & minor8=0b00000001 ; imm16 {
```

```
    rn = imm16;
```

```
}
```

# Example #3

MeP-c4 32-bit MOV (variant #2)

```
# MOVU Rn[0-7],imm24 - 1101_0nnn_IIII_IIII iii_iiii_iii_iii
```

```
define token instr(16)
```

```
    major = (12, 15)
```

```
    rn = (8, 11)
```

```
    minor = (0, 3)
```

```
;
```

```
define token ext(16)
```

```
    imm16 = (0, 15)
```

```
;
```

```
:movu rn, imm24 is major=0b1101 & rn & minor ; imm16 [ imm24 = minor + (imm16 << 8); ]
```

```
{
```

```
    rn = imm24;
```

```
}
```

# Example #4

MeP-c4 LW

```
# LW Rn,(Rm) - 0000_nnnn_mmmm_1110

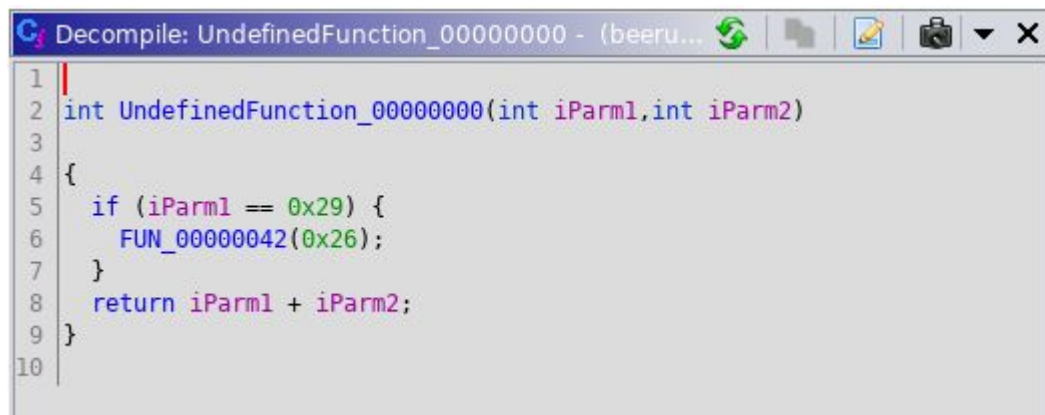
:lw rn, "("^rm^")" is major=0b0000 & rn & rm & minor=0b1110 {
    rn = *[ram]:4 rm;
}
```



# Ghidra/Processors/MEP\_C4/data/patterns/\*.xml

ease detecting functions prologues & epilogues

```
<patternlist>
  <pattern>
    <data> 0x1a 0x70 .....0000 0x6f </data>
    <!-- 1a70          LDC R0, LP
          f06f          ADD SP, -4
    -->
    <funcstart validcode="function" thunk="true"/>
  </pattern>
</patternlist>
```



The image shows a decompiler window titled "Decompile: UndefinedFunction\_00000000 - (beeru...". The window contains a C code snippet. The code defines a function named "UndefinedFunction\_00000000" that takes two integer parameters, "iParm1" and "iParm2". Inside the function, there is a conditional check: if "iParm1" is equal to the hexadecimal value "0x29", then the function calls "FUN\_00000042" with the argument "0x26". After this conditional block, the function returns the sum of "iParm1" and "iParm2". The code is displayed with syntax highlighting: keywords like "int", "if", "return", and "FUN\_" are in blue, variables and constants are in purple, and the function name is in green. Line numbers 1 through 10 are visible on the left side of the code editor.

```
1  
2 int UndefinedFunction_00000000(int iParm1,int iParm2)  
3  
4 {  
5     if (iParm1 == 0x29) {  
6         FUN_00000042(0x26);  
7     }  
8     return iParm1 + iParm2;  
9 }  
10
```

# Perspectives

## PR for ghidra-mep

add missing instructions

implement headless unit tests

## automatically generate mep.sla from miasm?

convert miasm expressions to P-CODE?

# References

[Ghidra Language Specification](#)

[Specifying Representations of Machine Instructions](#)

[The University of Queensland Binary Translator \(UQBT\) Framework](#)

[Ghidra Processors](#)

[XML schemas](#)

[SLEIGH language grammar](#)

Questions?  
Beers?