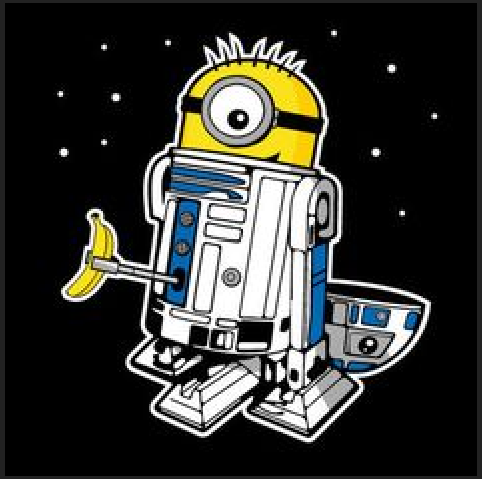
R2M2

RADARE2 + MIASM2 = ♥

@guedou - 18/11/2016

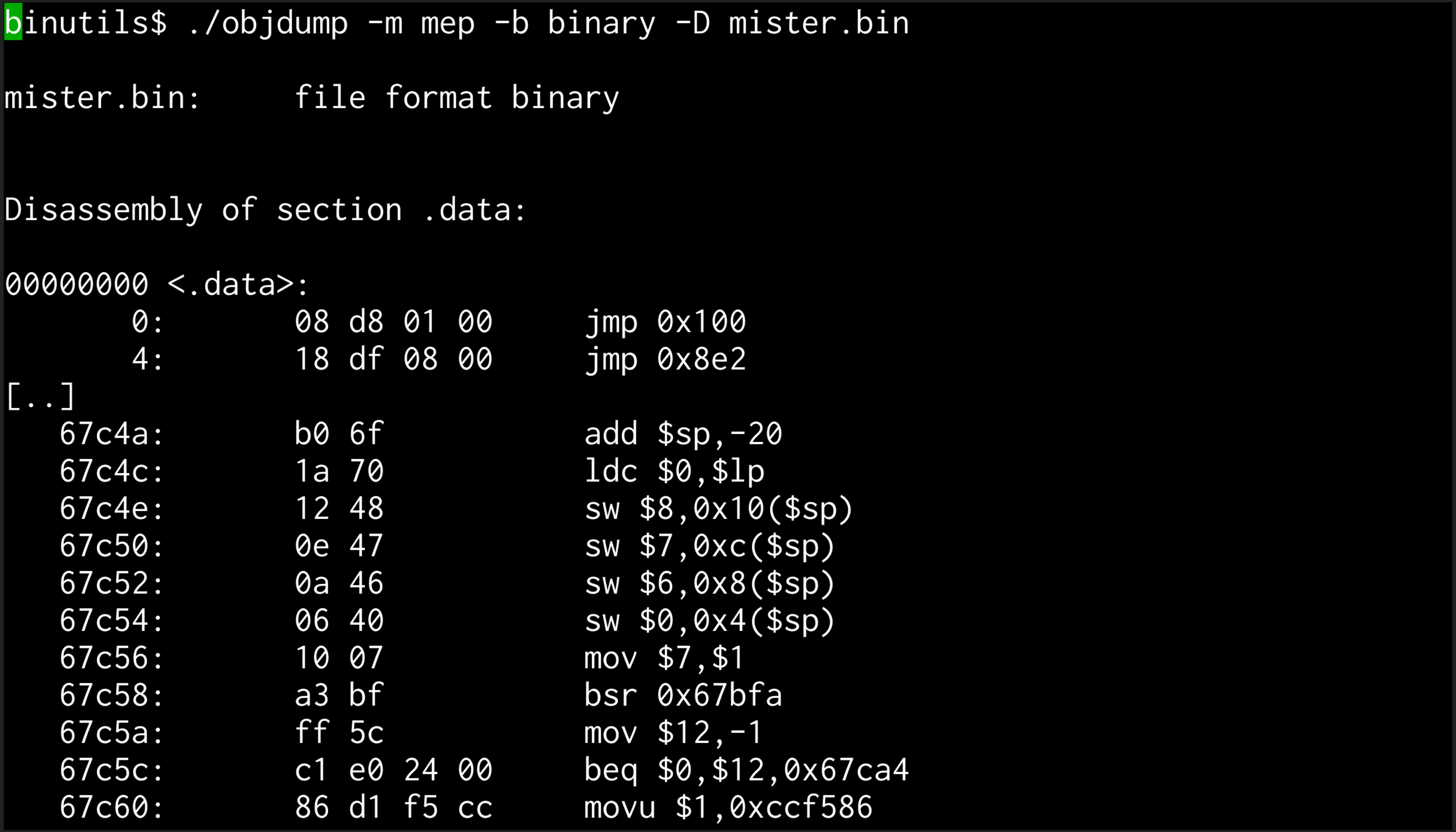


@GUEDOU?

- French
- hobbyist reverser
- network security researcher
  - IPv6, DNS, TLS, BGP, DDoS mitigation, ...
- Scapy co-maintainer
  - Python-based packet manipulation program & library
- neither a radare2 nor miasm2 power user

I needed to implement a rare CPU architecture **easily**

Back in December 2015, only `objdump` knew this architecture



```
binutils$ ./objdump -m mep -b binary -D mister.bin

mister.bin:     file format binary


Disassembly of section .data:

00000000 <.data>:
       0:       08 d8 01 00     jmp 0x100
       4:       18 df 08 00     jmp 0x8e2
[..]
   67c4a:       b0 6f           add $sp,-20
   67c4c:       1a 70           ldc $0,$lp
   67c4e:       12 48           sw $8,0x10($sp)
   67c50:       0e 47           sw $7,0xc($sp)
   67c52:       0a 46           sw $6,0x8($sp)
   67c54:       06 40           sw $0,0x4($sp)
   67c56:       10 07           mov $7,$1
   67c58:       a3 bf           bsr 0x67bfa
   67c5a:       ff 5c           mov $12,-1
   67c5c:       c1 e0 24 00     beq $0,$12,0x67ca4
   67c60:       86 d1 f5 cc     movu $1,0xccf586
```

R2M2 GOALS?

r2m2 is a radare2 plugin that aims to:

- use radare2 as a frontend to miasm2
  - tools, GUI, shortcuts, ...
- use miasm2 as a backend to radare2
  - asm/dis engine, symbolic execution, ...
- be architecture independent

ADDING A NEW ARCHITECTURE TO MIASM

HIGH-LEVEL CHECKLIST

1. registers in *miasm2/arch/ARCH/regs.py*
2. opcodes in *miasm2/arch/ARCH/arch.py*
3. semantic in *miasm2/arch/ARCH/sem.py*

ADDING A NEW OPCODE IN ARCH.PY

	MIPS ADDIU
Encoding	001001 ss ssst tttt iiii iiii iiii iiii

The opcode is defined as:

```
addop("addiu", [bs("001001"), rs, rt, s16imm], [rt, rs, s16imm])
```

The arguments are defined as:

```
rs = bs(l=5, cls=(mips32_gpreg,))
rt = bs(l=5, cls=(mips32_gpreg,))
s16imm = bs(l=16, cls=(mips32_s16imm,))
```

*mips32_** objects implement `encode()` and `decode()` methods that return miasm expressions!

ADDING A NEW OPCODE IN SEM.PY

Solution#1 - Implement the logic with miasm expressions

```
def addiu(ir, instr, reg_dst, reg_src, imm16):

    expr_src = ExprOp("+", reg_src, imm16.zeroExtend(32))

    return [ExprAff(reg_dst, expr_src)], []
```

Solution#2 - Be lazy, and implement using the *sembuilder*

```
@sbuild.parse
def addiu(reg_dst, reg_src, imm16):
    reg_dst = reg_src + imm16
```

The resulting expression is:

```
>>> ir.get_ir(instr)  # instr being the IR of "ADDIU A0, A1, 2"
([ExprAff(ExprId('A0', 32), ExprOp('+', ExprId('A1', 32),
                            ExprInt(uint32(0x2L))))], [])
```

R2M2

(at last!)

WHAT IS R2M2?

- keeps most of the smart logics in miasm2
  - r2m2 aims to be architecture independent
  - uses the R2M2_ARCH env variable to specify the arch
- provides two r2 plugins:
  - ad: assembly & disassembly
  - Ae: Analysis & esil

```
r2m2$ rasm2 -L |grep r2m2
adAe  32         r2m2        LGPL3   miasm2 backend
```

R2M2_AD - THE EASY PLUGIN

- simple CFFI / C wrapper around a miasm2 `Machine()`
- provides miasm2 assembly & disassembly features to radare2

MIPS32 assembly/disassembly with rasm2:

```
r2m2$ export R2M2_ARCH=mips32l; rasm2 -a r2m2 'addiu a0, a1, 2' |rasm2
ADDIU       A0, A1, 0x2
```

miasm2 MSP430 in r2 with random instructions:

```
r2m2$ R2M2_ARCH=msp430 r2 -a r2m2 -qc 'woR; pd 5' -
            0x00000000      07fa           and.w      R10, R7
            0x00000002      47ad           dadd.b     R13, R7
            0x00000004      f05e0778       add.b      @R14+, 0x7807(PC)
            0x00000008      f46d81ed       addc.b     @R13+, 0xED81(R4)
            0x0000000c      3fdc           bis.w      @R12+, R15
```

miasm2 x86-64 on `/bin/ls`:

```
r2m2$ R2M2_ARCH=x86_64 r2 -a r2m2 /bin/ls -qc 'pd 7 @0x00404a1c'
            0x00404a1c      4883f80e       CMP          RAX, 0xE
            0x00404a20      4889e5         MOV          RBP, RSP
            0x00404a23      761b           JBE          0x1D
            0x00404a25      b800000000     MOV          EAX, 0x0
            0x00404a2a      4885c0         TEST         RAX, RAX
            0x00404a2d      7411           JZ           0x13
            0x00404a2f      5d             POP          RBP
```

Where does these jumps go?

R2M2_AE - THE CHALLENGING ONE

Use miasm2 to **automatically**

- find branches
- find function calls
- split blocks
- emulate instructions
- ...

HOW?

Step#1 - use miasm2 expressions and internal methods

- `breakflow()`, `dstflow()`, `is_subcall()`

```
# r2m2 incomplete example
if instr.is_subcall():
    if isinstance(instr.arg, ExprInt):
        analop.type = R_ANAL_OP_TYPE_CALL
        analop.jump = address + int(instr.arg)
    else:
        analop.type = R_ANAL_OP_TYPE_UCALL
```

A simple MIPS32 output

```
r2m2$ R2M2_ARCH=mips32b rasm2 -a r2m2 'j 0x4; nop' -B > j_nop.bin

r2m2$ R2M2_ARCH=mips32b r2 -a r2m2 j_nop.bin -qc 'pd 2'
        ,=< 0x00000000      08000001       J          0x4
        `-> 0x00000004      00000000       NOP
```

A more complex output - r2 vs r2m2

```
r2$ r2 /bin/ls -qc 'pd 12 @0x00404a1c'
            0x00404a1c      4883f80e       cmp rax, 0xe
            0x00404a20      4889e5         mov rbp, rsp
        ,=< 0x00404a23      761b           jbe 0x404a40
        |   0x00404a25      b800000000     mov eax, 0
        |   0x00404a2a      4885c0         test rax, rax
       ,==< 0x00404a2d      7411           je 0x404a40
       ||   0x00404a2f      5d             pop rbp
       ||   0x00404a30      bf60e66100     mov edi, loc._edata
       ||   0x00404a35      ffe0           jmp rax
       ||   0x00404a37      660f1f840000.  nop word [rax + rax]
       ``-> 0x00404a40      5d             pop rbp
            0x00404a41      c3             ret
```

```
r2m2$ R2M2_ARCH=x86_64 r2 -a r2m2 /bin/ls -qc 'pd 12 @0x00404a1c'
            0x00404a1c      4883f80e       CMP         RAX, 0xE
            0x00404a20      4889e5         MOV         RBP, RSP
        ,=< 0x00404a23      761b           JBE         0x1D
        |   0x00404a25      b800000000     MOV         EAX, 0x0
        |   0x00404a2a      4885c0         TEST        RAX, RAX
       ,==< 0x00404a2d      7411           JZ          0x13
       ||   0x00404a2f      5d             POP         RBP
       ||   0x00404a30      bf60e66100     MOV         EDI, loc._edata
       ||   0x00404a35      ffe0           JMP         RAX
       ||   0x00404a37      660f1f840000.  NOP         WORD PTR [RAX*0x2
       ``-> 0x00404a40      5d             POP         RBP
            0x00404a41      c3             RET
```

Step#2 - convert miasm2 expression to radare2 ESIL

- both achieve the same goal: express instructions semantics
- simple automatic conversions are possible

```
m2 expr -> ExprAff(ExprId("R0", 32), ExprInt(0x2807, 32))
r2 esil -> 0x2807,r0,=
```

- need to dynamically define the radare2 registers profile
- some instructions are problematic, as their semantics are complex

A simple MIPS32 output

```
r2m2$ R2M2_ARCH=mips32b rasm2 -a r2m2 'j 0x4; nop' -B > j_nop.bin

r2m2$ R2M2_ARCH=mips32b r2 -a r2m2 j_nop.bin -qc 'e asm.emu=true; pd 2'
        ,=< 0x00000000      08000001       J          0x4              ;
        `-> 0x00000004      00000000       NOP    n
```

A more complex output

```
R2M2_ARCH=x86_64 r2 -a r2m2 /bin/ls -qc 'e asm.emu=true; pd 12 @0x00404
            0x00404a1c      4883f80e       CMP         RAX, 0xE         ;
            0x00404a20      4889e5         MOV         RBP, RSP         ;
        ,=< 0x00404a23      761b           JBE         0x1D             ;
        |   0x00404a25      b800000000     MOV         EAX, 0x0         ;
        |   0x00404a2a      4885c0         TEST        RAX, RAX         ;
       ,==< 0x00404a2d      7411           JZ          0x13             ;
       ||   0x00404a2f      5d             POP         RBP              ;
       ||   0x00404a30      bf60e66100     MOV         EDI, loc._edata ;
       ||   0x00404a35      ffe0           JMP         RAX              ;
       ||   0x00404a37      660f1f840000.  NOP         WORD PTR [RAX*0x2
       ``-> 0x00404a40      5d             POP         RBP              ;
            0x00404a41      c3             RET                          ;
```

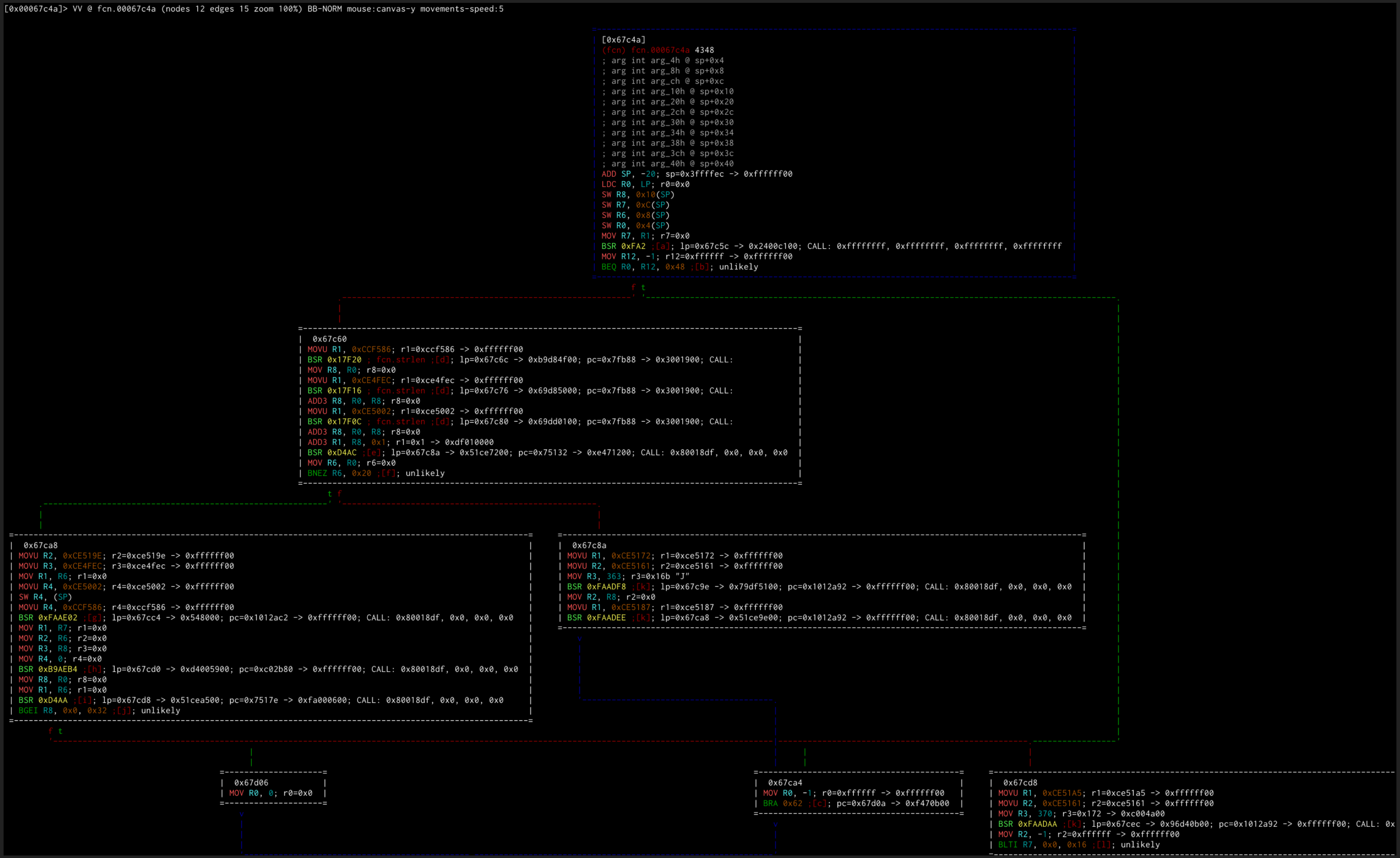
CONCLUDING REMARKS

- miasm2 and radare2 are powerful tools
  - combining them turned out to be efficient
- r2m2 is more than *"PoC that works on my laptop"*

  ```
  $ docker run --rm -it -e 'R2M2_ARCH=mips32l' guedou/r
  "rasm2 -a r2m2 'addiu a0, a1, 2'"
  ```

- too good to be true?
  - could be, yet r2m2 is better than nothing

Today, allows me to get call graphs



```
[0x00067c4a]> VV @ fcn.00067c4a (nodes 12 edges 15 zoom 100%) BB-NORM mouse:canvas-y movements-speed:5
```

```
[0x67c4a]
(fcn) fcn.00067c4a 4348
; arg int arg_4h @ sp+0x4
; arg int arg_8h @ sp+0x8
; arg int arg_ch @ sp+0xc
; arg int arg_10h @ sp+0x10
; arg int arg_20h @ sp+0x20
; arg int arg_2ch @ sp+0x2c
; arg int arg_30h @ sp+0x30
; arg int arg_34h @ sp+0x34
; arg int arg_38h @ sp+0x38
; arg int arg_3ch @ sp+0x3c
; arg int arg_40h @ sp+0x40
ADD SP, -20; sp=0x3ffffec -> 0xffffff00
LDC R0, LP; r0=0x0
SW R8, 0x10(SP)
SW R7, 0xC(SP)
SW R6, 0x8(SP)
SW R0, 0x4(SP)
MOV R7, R1; r7=0x0
BSR 0xFA2 ;[a]; lp=0x67c5c -> 0x2400c100; CALL: 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
MOV R12, -1; r12=0xffffff -> 0xffffff00
BEQ R0, R12, 0x48 ;[b]; unlikely
```
f t

```
 0x67c60
MOVU R1, 0xCCF586; r1=0xccf586 -> 0xffffff00
BSR 0x17F20 ; fcn.strlen ;[d]; lp=0x67c6c -> 0xb9d84f00; pc=0x7fb88 -> 0x3001900; CALL:
MOV R8, R0; r8=0x0
MOVU R1, 0xCE4FEC; r1=0xce4fec -> 0xffffff00
BSR 0x17F16 ; fcn.strlen ;[d]; lp=0x67c76 -> 0x69d85000; pc=0x7fb88 -> 0x3001900; CALL:
ADD3 R8, R0, R8; r8=0x0
MOVU R1, 0xCE5002; r1=0xce5002 -> 0xffffff00
BSR 0x17F0C ; fcn.strlen ;[d]; lp=0x67c80 -> 0x69dd0100; pc=0x7fb88 -> 0x3001900; CALL:
ADD3 R8, R0, R8; r8=0x0
ADD3 R1, R8, 0x1; r1=0x1 -> 0xdf010000
BSR 0xD4AC ;[e]; lp=0x67c8a -> 0x51ce7200; pc=0x75132 -> 0xe471200; CALL: 0x80018df, 0x0, 0x0, 0x0
MOV R6, R0; r6=0x0
BNEZ R6, 0x20 ;[f]; unlikely
```
t f

```
 0x67ca8
MOVU R2, 0xCE519E; r2=0xce519e -> 0xffffff00
MOVU R3, 0xCE4FEC; r3=0xce4fec -> 0xffffff00
MOV R1, R6; r1=0x0
MOVU R4, 0xCE5002; r4=0xce5002 -> 0xffffff00
SW R4, (SP)
MOVU R4, 0xCCF586; r4=0xccf586 -> 0xffffff00
BSR 0xFAAE02 ;[g]; lp=0x67cc4 -> 0x548000; pc=0x1012ac2 -> 0xffffff00; CALL: 0x80018df, 0x0, 0x0, 0x0
MOV R1, R7; r1=0x0
MOV R2, R6; r2=0x0
MOV R3, R8; r3=0x0
MOV R4, 0; r4=0x0
BSR 0xB9AEB4 ;[h]; lp=0x67cd0 -> 0xd4005900; pc=0xc02b80 -> 0xffffff00; CALL: 0x80018df, 0x0, 0x0, 0x0
MOV R8, R0; r8=0x0
MOV R1, R6; r1=0x0
BSR 0xD4AA ;[i]; lp=0x67cd8 -> 0x51cea500; pc=0x7517e -> 0xfa000600; CALL: 0x80018df, 0x0, 0x0, 0x0
BGEI R8, 0x0, 0x32 ;[j]; unlikely
```
f t

```
 0x67c8a
MOVU R1, 0xCE5172; r1=0xce5172 -> 0xffffff00
MOVU R2, 0xCE5161; r2=0xce5161 -> 0xffffff00
MOV R3, 363; r3=0x16b "J"
BSR 0xFAADF8 ;[k]; lp=0x67c9e -> 0x79df5100; pc=0x1012a92 -> 0xffffff00; CALL: 0x80018df, 0x0, 0x0, 0x0
MOV R2, R8; r2=0x0
MOVU R1, 0xCE5187; r1=0xce5187 -> 0xffffff00
BSR 0xFAADEE ;[k]; lp=0x67ca8 -> 0x51ce9e00; pc=0x1012a92 -> 0xffffff00; CALL: 0x80018df, 0x0, 0x0, 0x0
```

```
 0x67d06
MOV R0, 0; r0=0x0
```

```
 0x67ca4
MOV R0, -1; r0=0xffffff -> 0xffffff00
BRA 0x62 ;[c]; pc=0x67d0a -> 0xf470b00
```

```
 0x67cd8
MOVU R1, 0xCE51A5; r1=0xce51a5 -> 0xffffff00
MOVU R2, 0xCE5161; r2=0xce5161 -> 0xffffff00
MOV R3, 370; r3=0x172 -> 0xc004a00
BSR 0xFAADAA ;[k]; lp=0x67cec -> 0x96d40b00; pc=0x1012a92 -> 0xffffff00; CALL: 0x
MOV R2, -1; r2=0xffffff -> 0xffffff00
BLTI R7, 0x0, 0x16 ;[l]; unlikely
```

Questions? Comments? Issues? Beers?

https://github.com/guedou/r2m2