# r2cLEMENCy

Build plugins to support the cLEMENCy architecture

MaskRay

September 9, 2017

r2con 2017

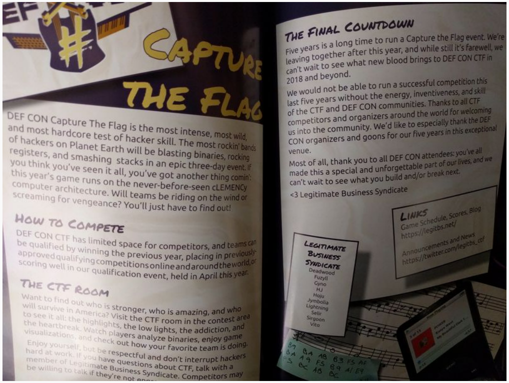
## whoami

- MaskRay (宋方睿 Sòng Fāng-ruì)
- `https://maskray.me` Twitter @HaskRay
- Software Engineer, San Francisco Bay Area, California, US
- Member of Tea Deliverers (CTF team)
- DEF CON 21∼25 CTF Finals (21∼23 blue-lotus, 24 b1o0p, 25 Tea-Deliverers)
- Sadly my RE skill has not improved much over the years...

## Tea Deliverers

- Tea Deliverers = blue-lotus + Nu1L + 110066 + Chaitin Tech
- Chinese
  `https://maskray.me/blog/2017-08-01-defcon-25-ctf`

## DEF CON 25 CTF Finals



CAPTURE THE FLAG

DEF CON Capture The Flag is the most intense, most wild, and most hardcore test of hacker skill. The most rockin' bands of hackers on Planet Earth will be blasting binaries, rocking registers, and smashing stacks in an epic three-day event. If you think you've seen it all, you've got another thing comin': this year's game runs on the never-before-seen cLEMENCy computer architecture. Will teams be riding on the wind or screaming for vengeance? You'll just have to find out!

HOW TO COMPETE

DEF CON CTF has limited space for competitors, and teams can be qualified by winning the previous year, placing in previously-approved qualifying competitions online and around the world, or scoring well in our qualification event, held in April this year.

THE CTF ROOM

Want to find out who is stronger, who is amazing, and who will survive in America? Visit the CTF room in the contest area to see it all: the highlights, the low lights, the addiction, and the heartbreak. Watch players analyze binaries, enjoy game visualizations, and check out how your favorite team is doing.

Enjoy yourself, but be respectful and don't interrupt hackers hard at work. If you have questions about CTF, talk with a member of Legitimate Business Syndicate. Competitors may be willing to talk if they're not

THE FINAL COUNTDOWN

Five years is a long time to run a Capture the Flag event. We're leaving together after this year, and while still it's farewell, we can't wait to see what new blood brings to DEF CON CTF in 2018 and beyond.

We would not be able to run a successful competition this last five years without the energy, inventiveness, and skill of the CTF and DEF CON communities. Thanks to all CTF competitors and organizers around the world for welcoming us into the community. We'd like to especially thank the DEF CON organizers and goons for our five years in this exceptional venue.

Most of all, thank you to all DEF CON attendees: you've all made this a special and unforgettable part of our lives, and we can't wait to see what you build and/or break next.

<3 Legitimate Business Syndicate

LINKS

Game Schedule, Scores, Blog
https://legitbs.net/

Announcements and News
https://twitter.com/legitbs_ctf

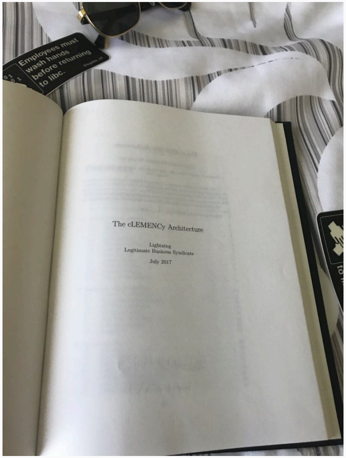
LEGITIMATE BUSINESS SYNDICATE
Deadwood
Fuzyll
Gyno
HJ
Hoju
Jymbolia
Lightning
Selir
Sirgoon
Vito

Curtain call of 5-year organizer Legitimate Business Syndicate

## cLEMENCy

- Architecture developed by Lightning
- 1 'byte' (nyte) = 9 bits
- 32 27-bit registers + 1 flags register (Zero,Carry,Overflow,Sign+others)
- ST=r29 (stack register), RA=r30 (link register), PC=r31, r28 (frame register)
- middle-endian
- `https://github.com/legitbs/cLEMENCy`
- `https://blog.legitbs.net/2017/07/def-con-ctf-2017-final-scores-and-data.html`

## cLEMENCy manual


The cLEMENCy Architecture

Lightning
Legitimate Business Syndicate

July 2017

## clemency-emu

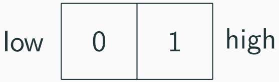
```
% cLEMENCy/cLEMENCy-emu/clemency-emu-debug -d 0 hello.bin
> t  # step
R00: 0000000    R01: 0000019    R02: 0000002    R03: 0000007
R04: 0000000    R05: 0000000    R06: 0000000    R07: 0000000
R08: 0000000    R09: 0000000    R10: 0000000    R11: 0000000
R12: 0000000    R13: 0000000    R14: 0000000    R15: 0000000
R16: 0000000    R17: 0000000    R18: 0000000    R19: 0000000
R20: 0000000    R21: 0000000    R22: 0000000    R23: 0000000
R24: 0000000    R25: 0000000    R26: 0000000    R27: 0000000
R28: 0000000     ST: 0000000     RA: 0000000     PC: 0000006
 FL: 0000000

0000006:                         5200780        smp    R00, R01, E
> db 2 3  # hexdump
0000002: 040 001 000
> u 0 2  # disassemble
0000000:                         2b0402000002b8 ldt    R01, [R00 + 0x57, 3]
0000006:                         5200780        smp    R00, R01, E
```

## Instructions

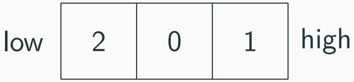
- 2,3,4,6 nytes
- `AD r0, r1, r2 # ADd`
- `ADCI r0, r1, -4 # ADd Immediate with Carry`
- M-suffixed instructions: adjacent registers as a pair
- `DVSM r3, r27, r31 # r3:r4 = (r27<<27 | r28) / (r31<<27 | r0)`
- `LD[SWT] # LoaD 1/2/3 nytes, middle-endian`
- `ST[SWT] # STore 1/2/3 nytes, middle-endian`

## Middle-endian

Word of 2 nytes: `a[1] << 9 | a[0]`

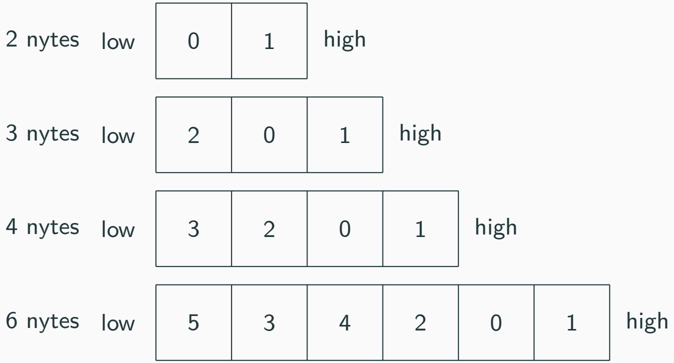


Tri-word of 3 nytes: `a[1] << 27 | a[2] << 18 | a[0]`



## Instruction decoding

Instructions consist of 3-nyte groups, with permutation in each group
Opcode in high bits



## Memory mappings

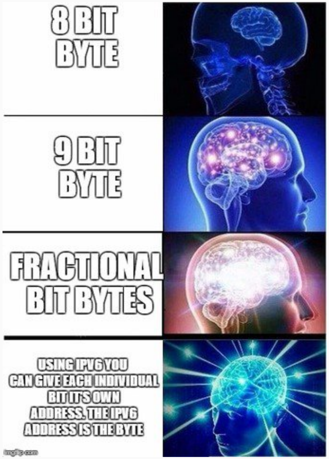
```
[0000000,4000000) Main Program Memory
[4000000,400001e) Clock IO
[4010000,4011000) Flag IO   # Capture the Flag!
[5000000,5002000) Data Received
[5002000,5002003) Data Received Size
[5010000,5012000) Data Sent
[5012000,5012003) Data Sent Size
[5100000,5104000) NFO file
[7ffff00,7ffff1c) Interrupt Pointers
[7ffff80,8000000) Processor Identification and Features
```

left-closed right-open intervals are convenient

## radare2 plugins

- `https://github.com/MaskRay/r2cLEMENCy`
- `io_9bit.so`: IO
- `core_clcy.so`: custom commands
- `bin_clcy.so`: loader
- `asm_clcy.so`: (dis)assembler
- `anal_clcy.so`: instruction semantics and emulation
- `parse_clcy.so`: C-like pseudo disassembler and `asm.varsub`
- More plugin types in `core/libs.c:r_core_loadlibs_init`
- language, filesystem, debugger, debugger breakpoint, egg

## 1 nyte = 9 bits


8 BIT BYTE
9 BIT BYTE
FRACTIONAL BIT BYTES
USING IPV6 YOU CAN GIVE EACH INDIVIDUAL BIT IT'S OWN ADDRESS. THE IPV6 ADDRESS IS THE BYTE
imgflip.com

Expand 1 nyte to 16-bit unsigned short

## r_io_plugin_clcy

```
RIOPlugin r_io_plugin_clcy = {
  .name = "clcy",
  .desc = "cLEMENCy io",
  .license = "LGPL3",
  .check = _check,
  .close = _close,
  .extend = _extend,
  .lseek = _lseek,
  .open = _open,
  .read = _read,
  .write = _write,
};
```

## io_clcy

- `.open`: file $\rightarrow$ 9-bit units $\rightarrow$ 16-bit units (2 bytes)
- `len = len_bytes*8/9; buf = malloc(len*2);`
- One address unit has 2 bytes
  `io->addrbytes = 2; // RIO::addrbytes`
- `len` argument in `read/write` still refers to bytes, not 16-bit
- Make sure buffers used by read()/write() are aware of `RIO::addrbytes`
- `.close`: 16-bit $\rightarrow$ 9-bit $\rightarrow$ file

## RIO::addrbytes

```
// A buffer of length RCore::blocksize (default: 256) contains
// blocksize/addrbytes (256/2=128) address units

// Before (every address unit is 1 byte):
while (idx < len) {
  r_anal_op (anal, &op, addr + idx, buf + idx,
      len - idx);

// After (buf access is aware of RIO::addrbytes):
while (addrbytes * idx < len) {
  r_anal_op (anal, &op, addr + idx, buf + addrbytes * idx,
      len - addrbytes * idx);
```

## Call path of a user command

- `r_core_prompt_exec`
- `r_core_cmd`
- `r_core_subst`(;,repeat,comment)
- `r_core_subst_i`
- `r_core_subst_i`(@,backquotes,double quotes,grep,pipe,redirection)
- `r_cmd_call`
- `RCorePlugin::call` / builtin commands (`RCore.cmds.cmd['p']`)

## r_core_plugin_clcy

```c
RCorePlugin r_core_plugin_clcy = {
  .name = "clcy",
  .desc = "cLEMENCy core",
  .license = "LGPL3",
  .call = r_cmd_clcy,
};

static int r_cmd_clcy(struct r_core_t *core, const char *input) {
  if (input[0] == '_') {
    ...
    case 'x': hexdump_9byte (core, input, 1); break; // "_px"
    case 'w': hexdump_18word (core, input, 1); break; // "_pw"
    case 't': hexdump_27tri (core, input, 1); break; // "_pt"
    ...
    return true;
  }
  return false;
}
```

## bin_clcy

- Create sections according to cLEMENCy memory mappings
- `.add=true, .name="Main", .paddr=0, .size=sz,`
- `.vsize=0x4000000, .srwx=R_IO_READ|R_IO_EXEC`
- Simple IO Layer creates two `RIOMap`
- file map $[0, \mathit{size})$ + null map $[\mathit{size}, \mathit{vsize})$

## om

```
[Main_Program_Memory:0x00000000]> om
10 fd: 3 +0x00000000 0x00000000 - 0x00006b67 -r-x fmap.Main_Program_Memory
 9 fd: 12 +0x00000000 0x00006b68 - 0x03ffffff -r-x mmap.Main_Program_Memory
 8 fd: 11 +0x00000000 0x00000000 - 0x0000001d -rw- mmap.Clock_IO
 7 fd: 10 +0x00000000 0x04010000 - 0x04010fff -r-- mmap.Flag_IO
 6 fd: 9 +0x00000000 0x05000000 - 0x05001fff -rw- mmap.Data_Received
 5 fd: 8 +0x00000000 0x05002000 - 0x05002001 -rw- mmap.Data_Received_Size
 4 fd: 7 +0x00000000 0x05010000 - 0x05011fff -rw- mmap.Data_Sent
 3 fd: 6 +0x00000000 0x05012000 - 0x05012001 -rw- mmap.Data_Sent_Size
#2 fd: 5 +0x00000000 0x05100000 - 0x05103fff -r-x mmap.NFO
 1 fd: 4 +0x00000000 0x07ffff00 - 0x07ffff1b -rw- mmap.Interrupt_Pointers
```

Main Program Memory has both file map (`fmap.`) and null map (`mmap.`)

## r_bin_plugin_clcy

```
RBinPlugin r_bin_plugin_clcy = {
  .name = "clcy",
  .desc = "cLEMENCy bin plugin",
  .license = "LGPL3",
  .baddr = _baddr,
  .check_bytes = _check_bytes,
  .create = _create,
  .destroy = _destroy,
  .info = _info,
  .load = _load,
  .minstrlen = 0,
  .patch_relocs = _patch_relocs,
  .sections = _sections,
};
```

## bin_clcy

- How to initialize NFO?

## bin_clcy

- How to initialize NFO?
- `.patch_relocs`

## bin_clcy

- How to initialize NFO?
- `.patch_relocs`
- Patch relocations in ELF/bFLT/CGC (Cyber Grand Challenge), especially useful for `ET_REL`

## bin_clcy

- How to initialize NFO?
- `.patch_relocs`
- Patch relocations in ELF/bFLT/CGC (Cyber Grand Challenge), especially useful for `ET_REL`
- Abuse it: create and initialize a `malloc://` map

## bin_clcy _patch_relocs

```
static RList *_patch_relocs(RBin *b) {
  ...
  RIOSection sec = {.name = "NFO", .size = n_buf * 2, .vsize = 0x4000,
    .flags = R_IO_READ | R_IO_EXEC};
  (void)r_io_create_mem_map (b->iob.io, &sec, NFO_VADDR, false);
  (void)r_io_write_at (b->iob.io, NFO_VADDR, (const ut8 *)buf, len * 2);
  ...
}
```

## asm_clcy

- IDA Pro processor in the game, `processor_t.{ana,out}`
- disassembler
- assembler
- `https://github.com/pwning/defcon25-public` by Plaid Parliament of Pwning
- X macros

## r_asm_plugin_clcy

```
static RAsmPlugin r_asm_plugin_clcy  = {
  .name = "clcy",
  .desc = "cLEMENCy asm",
  .arch = "clcy",
  .license = "LGPL3",
  .bits = 64, // in accordance with r_anal_plugin_clcy
  .disassemble = _disassemble,
  .assemble = _assemble,
};
```

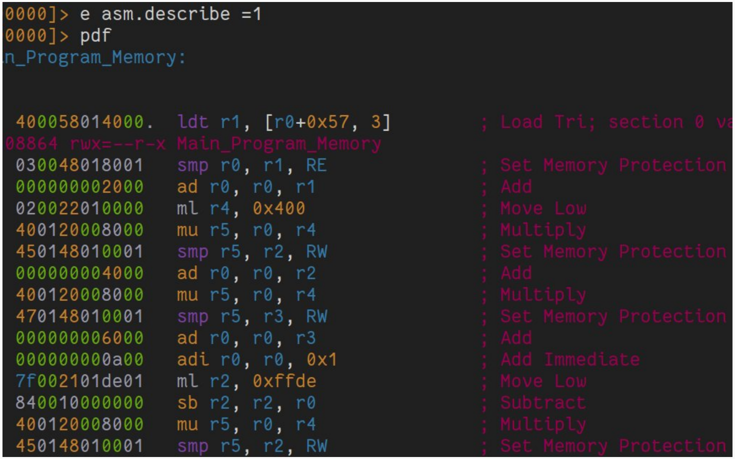
## asm_clcy struct inst_t

```
typedef struct {
  ut64 code, opcode;
  int id, size;
  ut32 pc, funct;
  st32 imm;
  ut16 cc, reg_count;
  ut8 adj_rb, arith_signed, is_imm, mem_flags, rA, rB, rC, rw, uf;
} inst_t;
```

## asm_clcy disassembler

```
// Group instructions by forms
do {
  FORMAT( R )  // assume this is an R-form instruction
  // If funct == 0b0000000 && arith_signed == 0 && is_imm == 0
  // This is ad --> break
  INS_3( ad, 0b0000000, funct, 0, arith_signed, 0, is_imm, 0 )
  // Try adc
  INS_3( adc, 0b0100000, funct, 0, arith_signed, 0, is_imm, 0 )
  // Try others
  ...
  FORMAT( R_IMM )  // assume this is an R_IMM-form instruction
  INS_2( adci, 0b0100000, arith_signed, 0, is_imm, 1 )
  ...
} while (0);

#define FORMAT(fmt) ok = decode_##fmt ...
#define INS_1(x,opc,f1,v1) if (inst.opcode==opc && inst.f1==v1) ...
#define INS_2(x,opc,f1,v1,f2,v2) if (inst.opcode==opc && inst.f1==v1 && \
  inst.f2==v2) ...
```

## pdf



```
0000]> e asm.describe =1
0000]> pdf
n_Program_Memory:


 400058014000.  ldt r1, [r0+0x57, 3]        ; Load Tri; section 0 va
08864 rwx=--r-x Main_Program_Memory
 030048018001   smp r0, r1, RE              ; Set Memory Protection
 000000002000   ad r0, r0, r1               ; Add
 020022010000   ml r4, 0x400                ; Move Low
 400120008000   mu r5, r0, r4               ; Multiply
 450148010001   smp r5, r2, RW              ; Set Memory Protection
 000000004000   ad r0, r0, r2               ; Add
 400120008000   mu r5, r0, r4               ; Multiply
 470148010001   smp r5, r3, RW              ; Set Memory Protection
 000000006000   ad r0, r0, r3               ; Add
 000000000a00   adi r0, r0, 0x1             ; Add Immediate
 7f002101de01   ml r2, 0xffde               ; Move Low
 840010000000   sb r2, r2, r0               ; Subtract
 400120008000   mu r5, r0, r4               ; Multiply
 450148010001   smp r5, r2, RW              ; Set Memory Protection
```

Descriptions: `asm/d/clcy.sdb`

## asm_clcy assembler

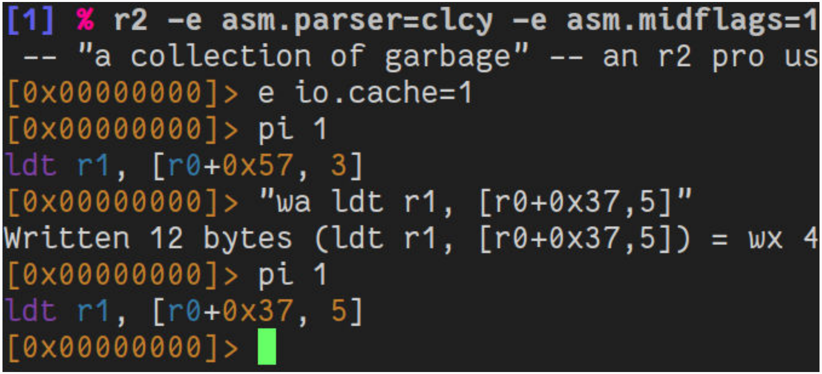
- `"wa ldt r1, [r0+0x57, 7]; ad. r0,r1,r1"`
- Recursive descent parser: `parse_{imm,rA,rB,rC,uf,comma,space,…}`
- Reuse X macros in disassembler

Suggest using a recursive descent parser in command parsing

## asm_clcy assemble_BIN_R_IMM

```c
#define FIELD(name, offset, count) | ((ut64)inst->name << \
  bit_size-count-offset)
#define FORM_BIN_R_IMM \
  FIELD(opcode, 0, 8) \
  FIELD(rA, 8, 5) \
  FIELD(imm, 13, 14)

static int assemble_BIN_R_IMM(inst_t *inst, const char **src) {
  int bit_size = 27;
  if (parse_space (inst, src)) return 1; // parse error
  if (parse_rA (inst, src)) return 1;
  if (parse_comma (inst, src)) return -1;
  if (parse_imm_st (inst, src, 14)) return 2;
  inst->imm &= (1 << 14) - 1;
  if (parse_end (src)) return 2;
  inst->size = 3; // 3 nytes
  inst->code = 0 FORM_BIN_R_IMM; // assemble all components
  return 0;
}
```

## wa



```
[1] % r2 -e asm.parser=clcy -e asm.midflags=1
 -- "a collection of garbage" -- an r2 pro us
[0x00000000]> e io.cache=1
[0x00000000]> pi 1
ldt r1, [r0+0x57, 3]
[0x00000000]> "wa ldt r1, [r0+0x37,5]"
Written 12 bytes (ldt r1, [r0+0x37,5]) = wx 4
[0x00000000]> pi 1
ldt r1, [r0+0x37, 5]
[0x00000000]>
```

## anal_clcy

- IDA Pro processor in the game, `processor_t.emu`
- Differentiate `JMP/CALL/MOV/PUSH/RET/SWI/`..., whether `COND,IND,MEM,REG,`... are used
- `R_ANAL_OP_TYPE_{JMP,COND,RCALL,RJMP,CRET,`...`}`
- `include/r_anal.h anal/p/anal_gb.c`
- Stack pointer delta (arguments, local variables), `add_stkpnt`
- ESIL translator

## ESIL

- Evaluable Strings Intermediate Language
- `anal/esil.c`
- Stack-oriented, Forth, DWARF expressions
- `mh r0, 0xffdf: 0x3ff,r0,&,10,65503,<<,|,r0,=`
- Decent support for 32/64 bits, needing work for 8/16 bits
- What if 27-bit/54-bit (register pair) + middle-endian?

## anal_clcy ESIL

- Just set `RAnal::bits` to 64 and define custom commands (`r_anal_esil_set_op`)
- `binop`: another argument for variants (carry/multi reg/imm/ signedness/update flags) + instruction family (add/sub/...)
- `addcm. r3,r2,r0` : `"r0,r2,r3,'.cm+,binop"`
- `'.cm+`
- `'` no special, arbitrary character borrowed from Lisp
- `.` update flags
- c with carry
- + add

## clcy_custom_binop

```c
r_anal_esil_set_op (esil, "binop", clcy_custom_binop);

static int clcy_custom_binop(RAnalEsil *esil) {
  bool carry = false, uf = false, mf;
  char *op = r_anal_esil_pop (esil), *op1 = op + 1,
    *rA = r_anal_esil_pop (esil), *rB = r_anal_esil_pop (esil),
    *rC = r_anal_esil_pop (esil);
  ...
  if (*op1 == '.') uf = true, op1++; // .: update flags
  if (*op1 == 'c') carry = true, op1++; // c: carry
  if (*op1 == 'm') ... // m: multi reg
  switch (*op1) {
  case '+': a = b + c; if (carry && read_fl (esil) & 2) a++; ...
  case '-': ...
  }
  if (uf) { /* update Carry/Overflow/Sign/Zero flags */ }
  ...
}
```

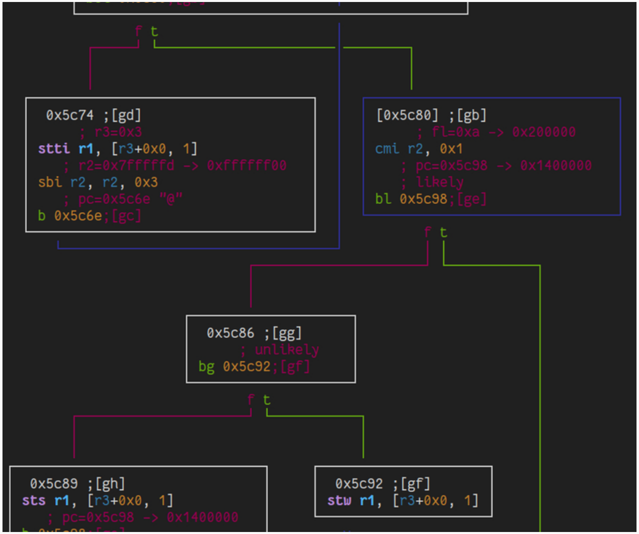
## Local variables/arguments detection

- Analysis engine detects with patterns like `0x..,st,+`
- We have custom `load/store` commands to emulate `LD[STW], ST[STW]`
- No-op `0x34,st,+,POP` to appease analysis engine

## RAnalPlugin

```c
static RAnalPlugin r_anal_plugin_clcy = {
  .name = "clcy",
  .desc = "cLEMENCy analysis",
  .license = "LGPL3",
  .arch = "clcy",
  .bits = 64, // we use 64-bit integers in esil to emulate 27-bit and 54-bit
  .esil_init = esil_clcy_init,
  .esil_fini = esil_clcy_fini,
  .esil_intr = esil_clcy_intr,
  .esil = true,
  .op = &clcy_op,
  .set_reg_profile = set_reg_profile,
};
```

# vv



```
 0x5c74 ;[gd]
     ; r3=0x3
stti r1, [r3+0x0, 1]
    ; r2=0x7ffffd -> 0xffffff00
sbi r2, r2, 0x3
    ; pc=0x5c6e "@"
b 0x5c6e;[gc]
```

```
[0x5c80] ;[gb]
     ; fl=0xa -> 0x200000
cmi r2, 0x1
    ; pc=0x5c98 -> 0x1400000
    ; likely
bl 0x5c98;[ge]
```

```
 0x5c86 ;[gg]
     ; unlikely
bg 0x5c92;[gf]
```

```
 0x5c89 ;[gh]
sts r1, [r3+0x0, 1]
    ; pc=0x5c98 -> 0x1400000
```

```
 0x5c92 ;[gf]
stw r1, [r3+0x0, 1]
```

## parse_clcy

- Bad name
  `https://github.com/radare/radare2/issues/4317`
- How to substitue variables for BP/SP offsets: `asm.varsub`
- How to generate C-like pseudo disassembly: `pdc`

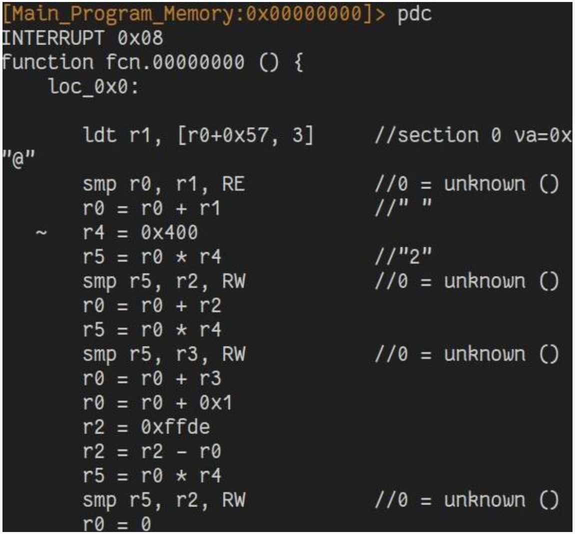
## r_parse_plugin_clcy

```
static int _parse(RParse *p, const char *src, char *dst);
static bool _varsub(RParse *p, RAnalFunction *f, ut64 addr,
  int oplen, char *src, char *dst, int len);

RParsePlugin r_parse_plugin_clcy = {
  .name = "clcy",
  .desc = "cLEMENCy",
  .parse = _parse,
  .varsub = _varsub,
};
```

## parse_clcy _parse

```
static int _parse(RParse *p, const char *src, char *dst) {
  RCore *core = p->user;
  RAsmOp op;
  int len;
  // `assemble` could be saved if we had access to metadata of previous
  // call to `assemble`
  if ((len = assemble (core->assembler->pc, &op, src)) > 0 &&
      disassemble (core->assembler->pc, &op, op.buf, len, true) > 0) {
    strcpy (dst, op.buf_asm);
  } else {
    strcpy (dst, src);
  }
  return true;
}
```

## pdc

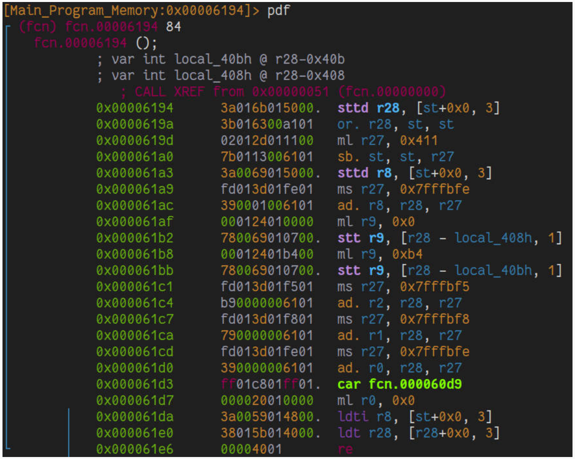


```
[Main_Program_Memory:0x00000000]> pdc
INTERRUPT 0x08
function fcn.00000000 () {
    loc_0x0:

       ldt r1, [r0+0x57, 3]      //section 0 va=0x
"@"
       smp r0, r1, RE            //0 = unknown ()
       r0 = r0 + r1              //" "
   ~   r4 = 0x400
       r5 = r0 * r4              //"2"
       smp r5, r2, RW            //0 = unknown ()
       r0 = r0 + r2
       r5 = r0 * r4
       smp r5, r3, RW            //0 = unknown ()
       r0 = r0 + r3
       r0 = r0 + 0x1
       r2 = 0xffde
       r2 = r2 - r0
       r5 = r0 * r4
       smp r5, r2, RW            //0 = unknown ()
       r0 = 0
```

## parse_clcy _varsub

```c
static bool _varsub(RParse *p, RAnalFunction *f, ut64 addr, int oplen,
    char *src, char *dst, int len) {
  ...
  // Stack register variable st+%#x
  r_list_foreach (bpargs, iter, var) {
    if (var->delta >= 0) {
      sub = r_str_newf ("[st+%#x", var->delta);
    } else {
      sub = r_str_newf ("[st-%#x", -var->delta);
    }
    // replace sub with var->name
  ...
}
```

## asm.varsub



```
[Main_Program_Memory:0x00006194]> pdf
┌ (fcn) fcn.00006194 84
│   fcn.00006194 ();
│           ; var int local_40bh @ r28-0x40b
│           ; var int local_408h @ r28-0x408
│             ; CALL XREF from 0x00000051 (fcn.00000000)
│           0x00006194      3a016b015000.  sttd r28, [st+0x0, 3]
│           0x0000619a      3b016300a101   or. r28, st, st
│           0x0000619d      02012d011100   ml r27, 0x411
│           0x000061a0      7b0113006101   sb. st, st, r27
│           0x000061a3      3a0069015000.  sttd r8, [st+0x0, 3]
│           0x000061a9      fd013d01fe01   ms r27, 0x7fffbfe
│           0x000061ac      390001006101   ad. r8, r28, r27
│           0x000061af      000124010000   ml r9, 0x0
│           0x000061b2      780069010700.  stt r9, [r28 - local_408h, 1]
│           0x000061b8      00012401b400   ml r9, 0xb4
│           0x000061bb      780069010700.  stt r9, [r28 - local_40bh, 1]
│           0x000061c1      fd013d01f501   ms r27, 0x7fffbf5
│           0x000061c4      b90000006101   ad. r2, r28, r27
│           0x000061c7      fd013d01f801   ms r27, 0x7fffbf8
│           0x000061ca      790000006101   ad. r1, r28, r27
│           0x000061cd      fd013d01fe01   ms r27, 0x7fffbfe
│           0x000061d0      390000006101   ad. r0, r28, r27
│           0x000061d3      ff01c801ff01.  car fcn.000060d9
│           0x000061d7      000020010000   ml r0, 0x0
│       │   0x000061da      3a0059014800.  ldti r8, [st+0x0, 3]
│       │   0x000061e0      38015b014000.  ldt r28, [r28+0x0, 3]
└       │   0x000061e6      00004001       re
```

See `local_*` variables. 0 offset is not handled currently

## debug_clcy

Left as exercise.

https://github.com/MaskRay/r2cLEMENCy

Questions?