

A stack of papers is shown, slightly offset to the right. The top paper is blue and features several lines of binary code (0s and 1s) in a monospaced font. Below the blue paper, a white paper is visible, showing assembly instructions in a monospaced font. The instructions include 'XOR', 'MOV', 'PUSH', and 'MOV' followed by register names in red: 'KA', 'RBX', and 'RCX'. The background is a dark, textured blue.

# Hopper Disassembler

The OS X Dedicated Disassembler, Decompiler and Debugger

# About me...

- ▶ I'm Vincent Bénony, from Lille in France
- ▶ I'm a technology addict
- ▶ Interested in Maths & Physics
- ▶ Ph.D. in cryptography
  - ▶ I worked on securing future AdHoc mobile networks
- ▶ Learnt assembly on Oric (6502), then Amiga (680x0/PPC), PC (x86)...



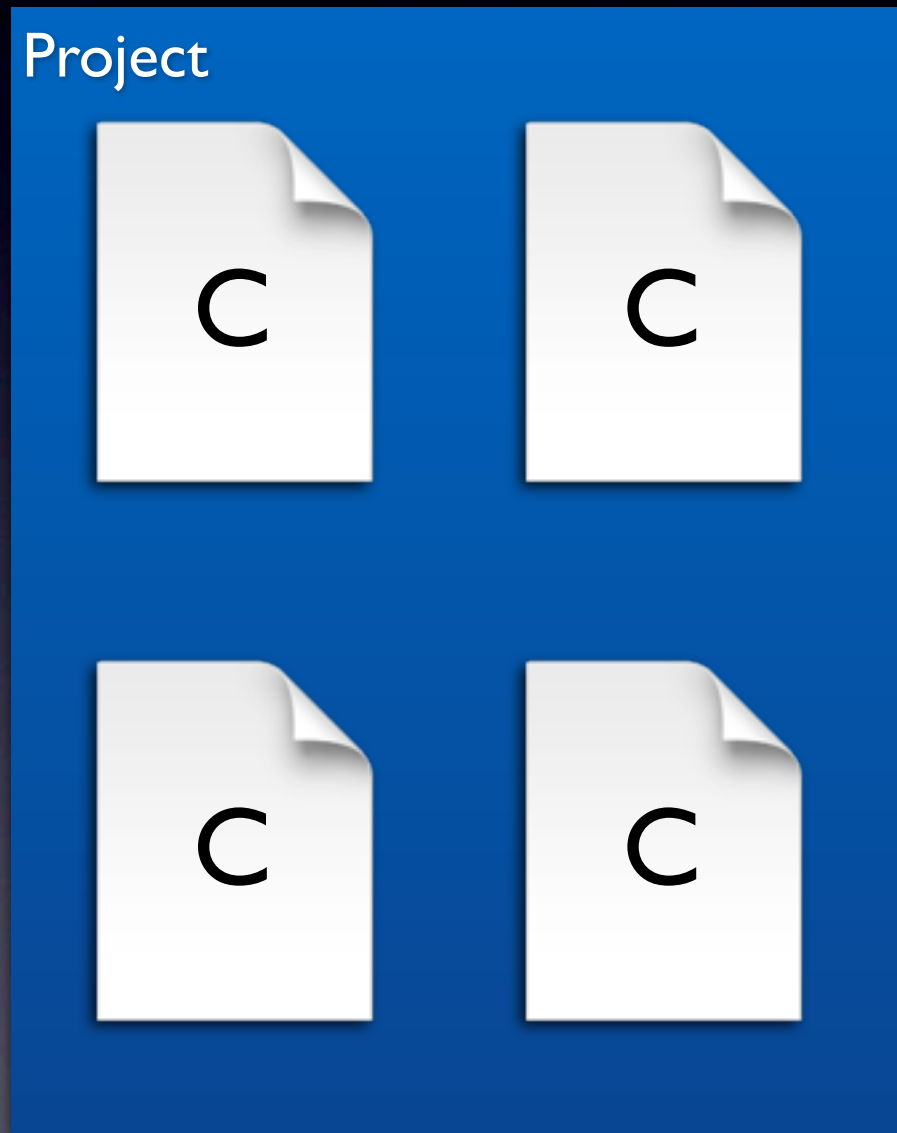
# What is Hopper?

- ▶ a disassembler
  - ▶ Intel 32 and 64 bits and ARM
  - ▶ Mach-O (Mac & iOS), PE32/32+/64, ELF 32/64
- ▶ a decompiler
- ▶ a debugger (Mac only)
- ▶ a software thought by a real Mac user!

# Why Disassembling?

- ▶ finding bugs and vulnerabilities,
- ▶ analyze stack traces from crash,
- ▶ reverse engineering a protocol,
- ▶ changing the behavior of an application,
- ▶ lost source code?
- ▶ fun :)
- ▶ ...

# Quick Intro to ASM Compiler





# Quick Intro to ASM Compiler



all information

Compilation



some type  
information

Link



only information  
needed at  
runtime

# Quick Intro to ASM

- ▶ the only language that a CPU understands
- ▶ thousand of instructions on Intel
  - ▶ a bit less on ARM (RISC)
- ▶ most of the instructions does basic things
  - ▶ write a byte in memory
  - ▶ add the value of two registers...

# Quick Intro to ASM

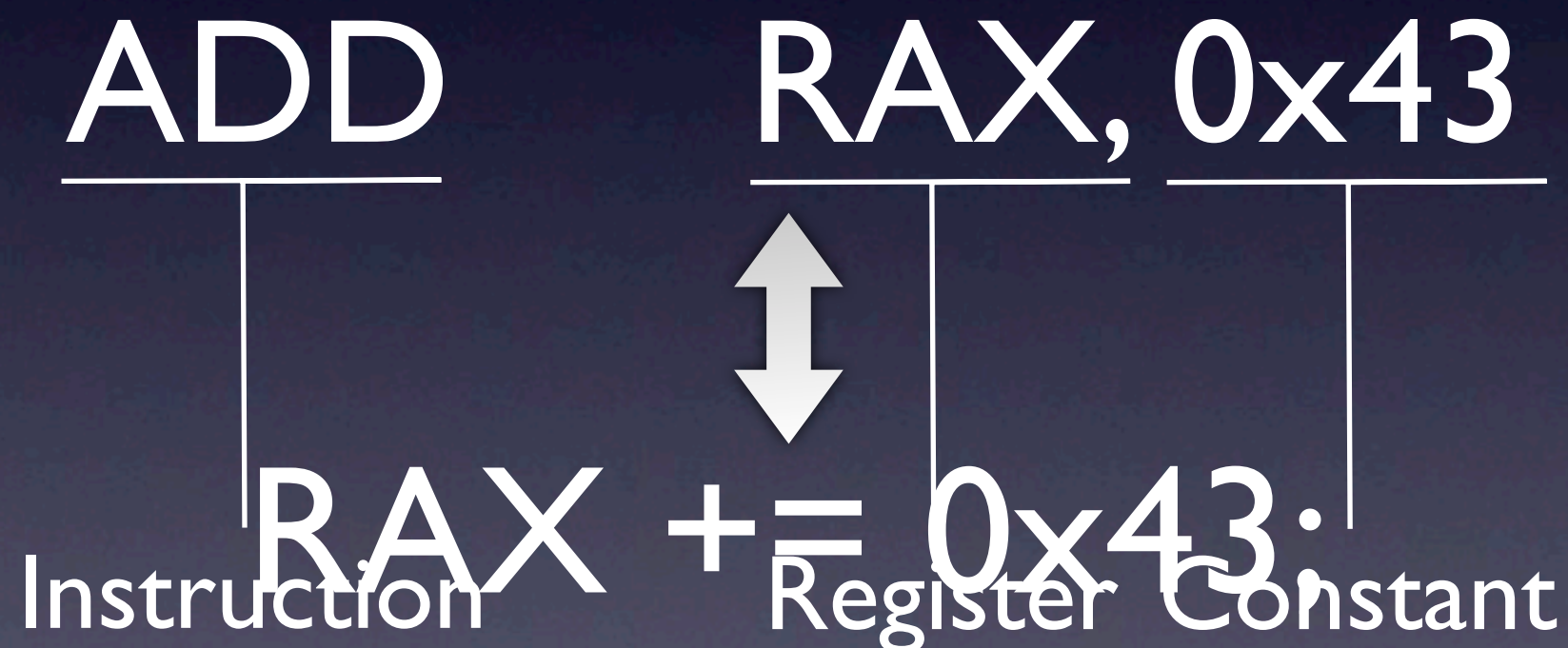
## Instruction Format





# Quick Intro to ASM

## Instruction Format



# Quick Intro to ASM

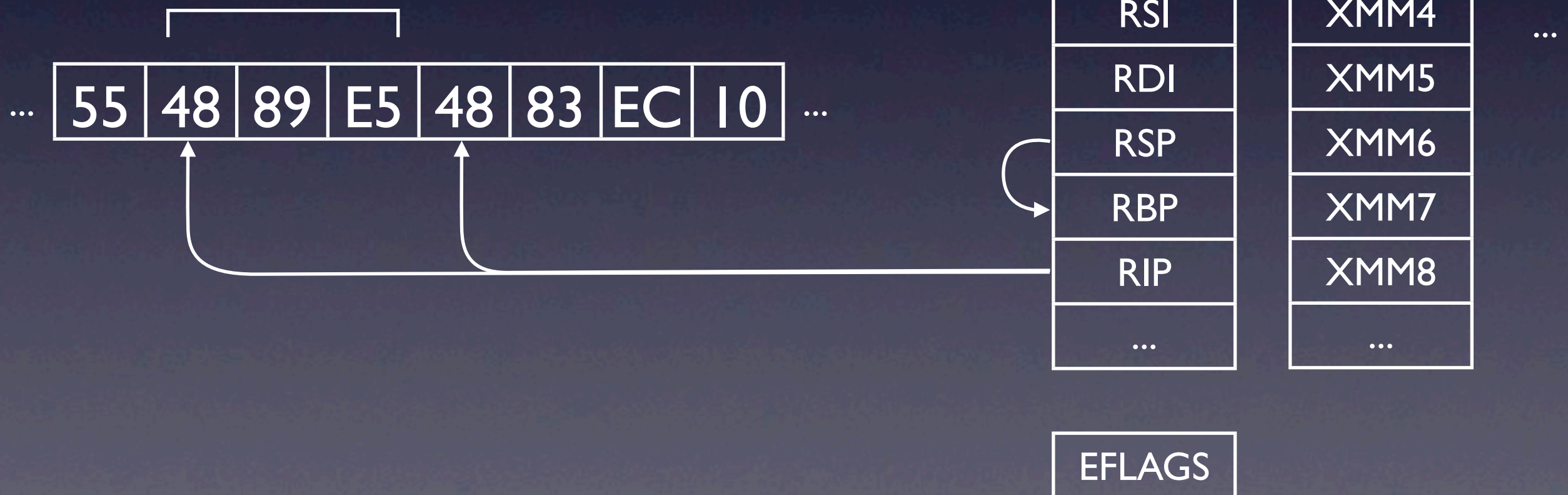
## Instruction Format

48 83 C0 43      ADD      RAX, 0x43

# Quick Intro to ASM

## The CPU

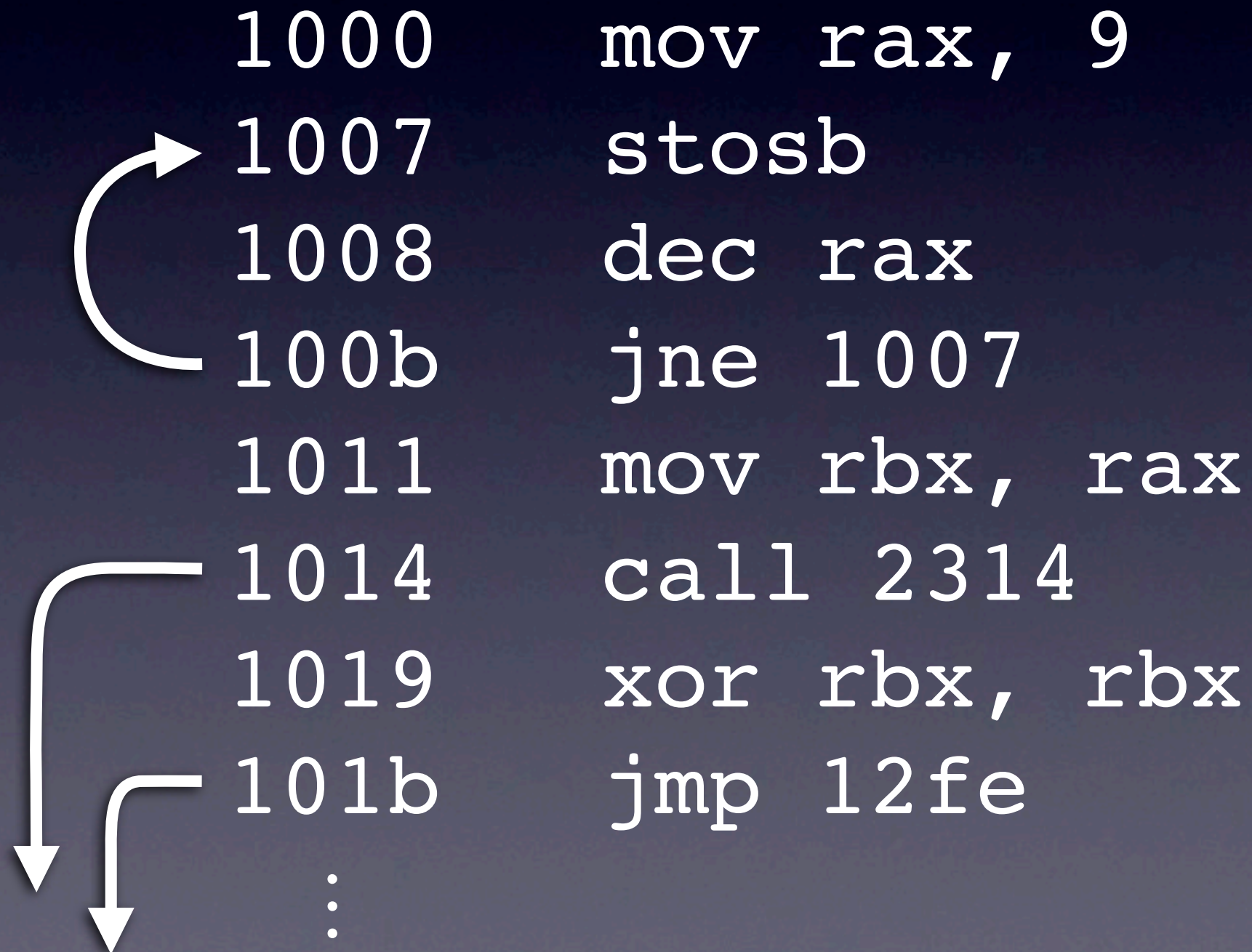
```
mov rbp, rsp
```





# Quick Intro to ASM

## Flow Control



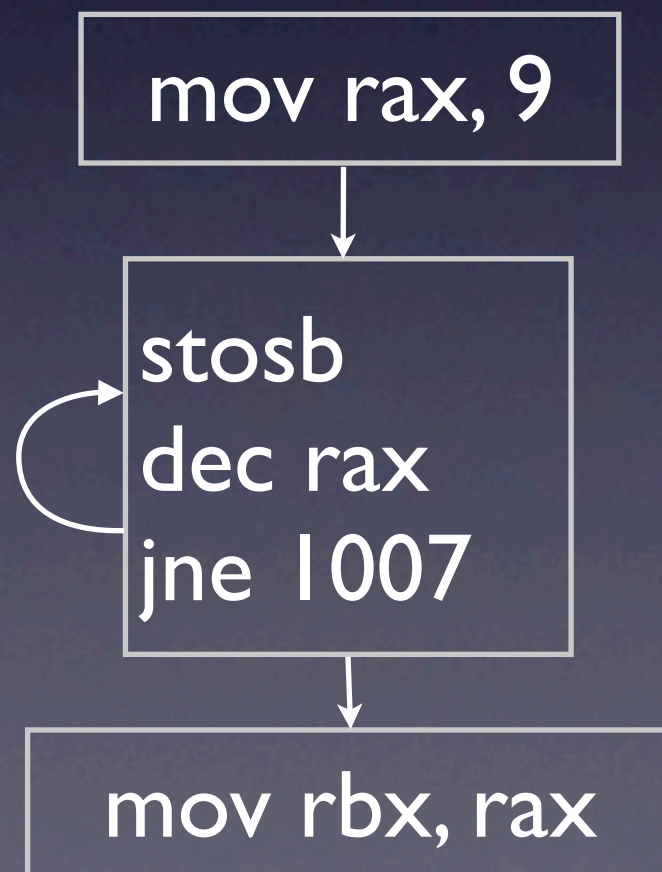
```
1000    mov rax, 9
1007    stosb
1008    dec rax
100b    jne 1007
1011    mov rbx, rax
1014    call 2314
1019    xor rbx, rbx
101b    jmp 12fe
      :
```

# Quick Intro to ASM

## Flow Control

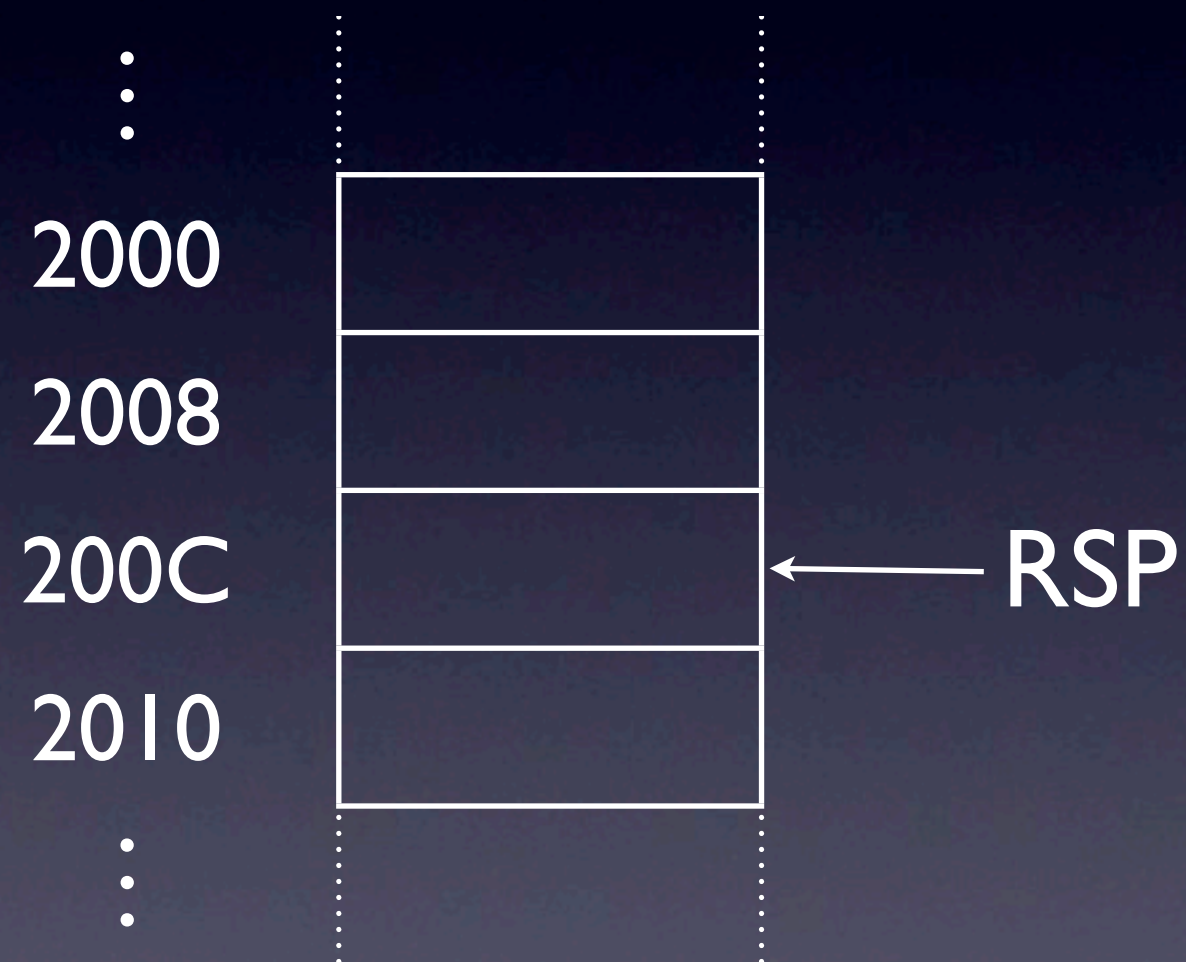
```
1000    mov rax, 9
1007    stosb
1008    dec rax
100b    jne 1007
1011    mov rbx, rax
```

```
RAX = 9;
while (RAX != 0) {
    *RDI++ = RAX--;
}
RBX = RAX;
```



# Quick Intro to ASM

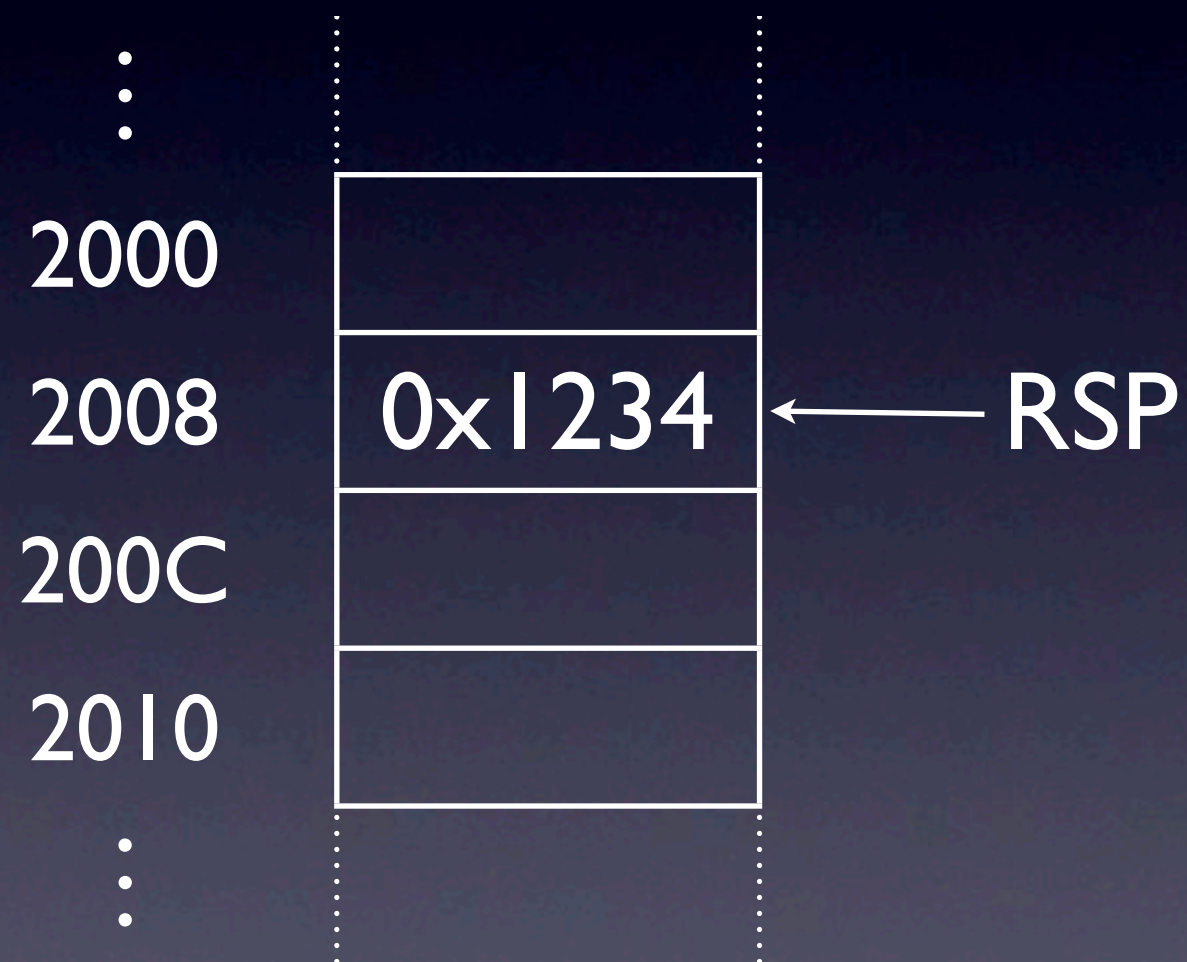
## Stack





# Quick Intro to ASM

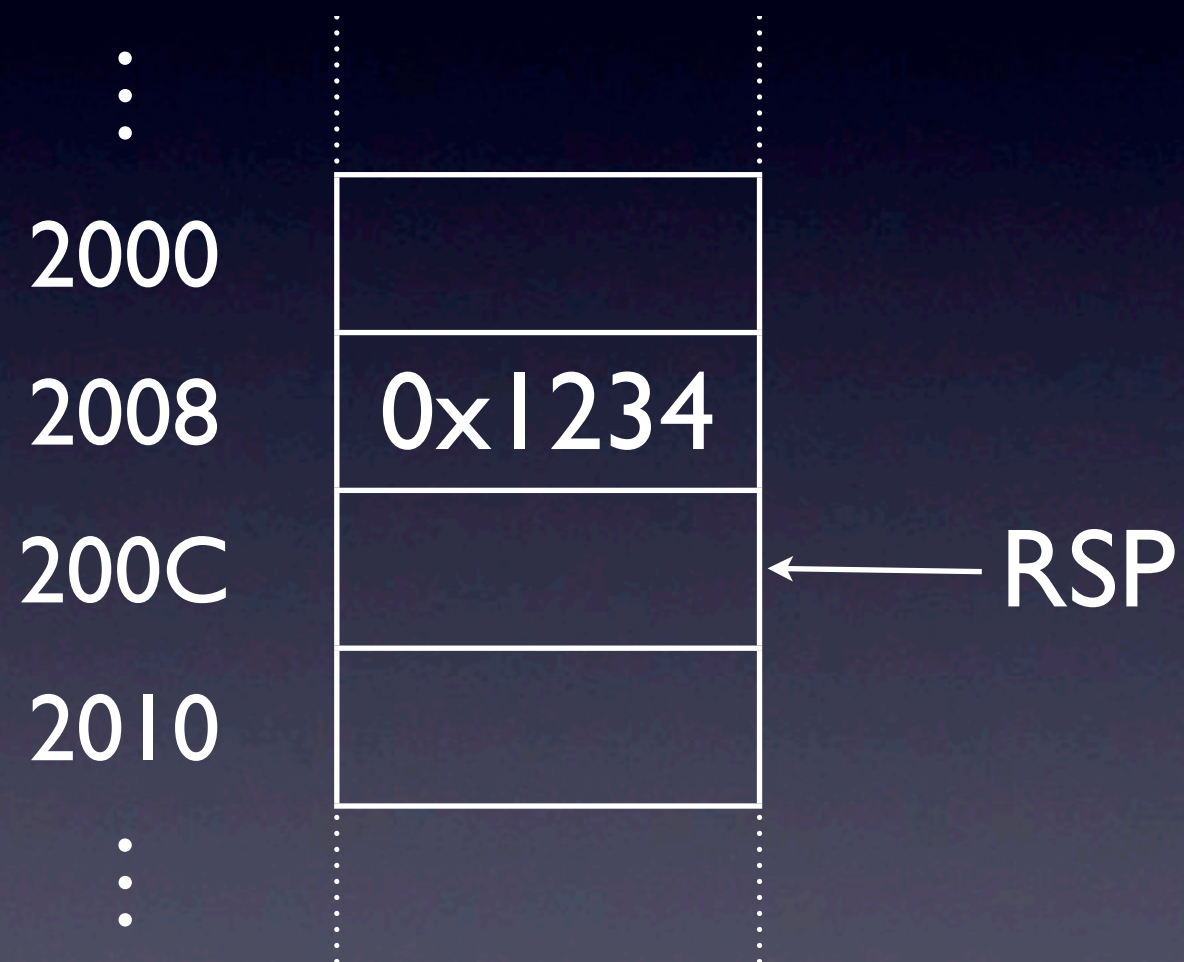
## Stack



PUSH 0x1234

# Quick Intro to ASM

## Stack



POP rax

# Quick Intro to ASM

## Stack

- ▶ stack is used to save register values
- ▶ used by “call” instruction
- ▶ used for “local variables”,
- ▶ ...



# Quick Intro to ASM

## Stack

```
push rbp
mov  rbp, rsp
sub  rbp, 124
:
pop  rbp
ret
```

```
x : [rbp + 0]
y : [rbp + 8]
z : [rbp + 12]
:
```

# Quick Intro to ASM

## x86 ABI

- ▶ no real methods in ASM
- ▶ calling convention to deal with arguments
  - ▶ `fastcall`, `stdcall`, `pascal`, `cdecl`, ...
- ▶ usually, on OS X
  - ▶ 32 bits uses stack to pass arguments
  - ▶ 64 bits uses registers `RDI`, `RSI`, `RDX`, `RCX`, ...

# Quick Intro to ASM

## x86 ABI

```
printf("result is: %d\n", x);
```

32 bits	64 bits
<pre>mov    edx, [ebp+16] push   edx push   0x164c3be call   0x18ae452</pre>	<pre>mov     rdi, 0x10000c3be mov     rsi, [rbp+16] call    0x18ae452</pre>



# Quick Intro to ASM

## Objective-C

```
[str stringByAppendingString:@" .png"];
```



```
objc_msgSend(  
    str,  
    @selector(stringByAppendingString:),  
    @" .png");
```

# Hopper

## Transformations

Hopper aimed at helping you to give a data type to each byte of a file.

```
0x55, 0x48, 0x89, 0xe5, 0x48, 0x83, 0xec, 0x10
0xc7, 0x45, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x89
0x7d, 0xf8, 0x48, 0x89, 0x75, 0xf0, 0x8b, 0x7d
0xf8, 0x48, 0x8b, 0x75, 0xf0, 0xe8, 0xcc, 0xcc
0x02, 0x00, 0x48, 0x83, 0xc4, 0x10, 0x5d, 0xc3
0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90
...
```

# Hopper

## Transformations

```
0x55, 0x48, 0x89, 0xe5, 0x48, 0x83, 0xec, 0x10  
0xc7, 0x45, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x89  
0x7d, 0xf8, 0x48, 0x89, 0x75, 0xf0, 0x8b, 0x7d  
0xf8, 0x48, 0x8b, 0x75, 0xf0, 0xe8, 0xcc, 0xcc  
0x02, 0x00, 0x48, 0x83, 0xc4, 0x10, 0x5d, 0xc3  
0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90
```

...



# Hopper

## Transformations

push		rbp					
0x48,	0x89,	0xe5,	0x48,	0x83,	0xec,	0x10	
0xc7,	0x45,	0xfc,	0x00,	0x00,	0x00,	0x00,	0x89
0x7d,	0xf8,	0x48,	0x89,	0x75,	0xf0,	0x8b,	0x7d
0xf8,	0x48,	0x8b,	0x75,	0xf0,	0xe8,	0xcc,	0xcc
0x02,	0x00,	0x48,	0x83,	0xc4,	0x10,	0x5d,	0xc3
0x90,	0x90,	0x90,	0x90,	0x90,	0x90,	0x90,	0x90

# Hopper

## Transformations

push			rbp				
0x48,	0x89,	0xe5,	0x48,	0x83,	0xec,	0x10	
0xc7,	0x45,	0xfc,	0x00,	0x00,	0x00,	0x00,	0x89
0x7d,	0xf8,	0x48,	0x89,	0x75,	0xf0,	0x8b,	0x7d
0xf8,	0x48,	0x8b,	0x75,	0xf0,	0xe8,	0xcc,	0xcc
0x02,	0x00,	0x48,	0x83,	0xc4,	0x10,	0x5d,	0xc3
0x90,	0x90,	0x90,	0x90,	0x90,	0x90,	0x90,	0x90

# Hopper

## Transformations

```
push      rbp
mov       rbp, rsp
0x48, 0x83, 0xec, 0x10
0xc7, 0x45, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x89
0x7d, 0xf8, 0x48, 0x89, 0x75, 0xf0, 0x8b, 0x7d
0xf8, 0x48, 0x8b, 0x75, 0xf0, 0xe8, 0xcc, 0xcc
0x02, 0x00, 0x48, 0x83, 0xc4, 0x10, 0x5d, 0xc3
0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90
```



# Hopper

## Transformations

```
_main:
    push    rbp
    mov     rbp, rsp
    0x48, 0x83, 0xec, 0x10
    0xc7, 0x45, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x89
    0x7d, 0xf8, 0x48, 0x89, 0x75, 0xf0, 0x8b, 0x7d
    0xf8, 0x48, 0x8b, 0x75, 0xf0, 0xe8, 0xcc, 0xcc
    0x02, 0x00, 0x48, 0x83, 0xc4, 0x10, 0x5d, 0xc3
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90
```

# Hopper

## Transformations

```
_main:  
    push    rbp  
    mov     rbp, rsp  
    0x48, 0x83, 0xec, 0x10  
    0xc7, 0x45, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x89  
    0x7d, 0xf8, 0x48, 0x89, 0x75, 0xf0, 0x8b, 0x7d  
    0xf8, 0x48, 0x8b, 0x75, 0xf0, 0xe8, 0xcc, 0xcc  
    0x02, 0x00, 0x48, 0x83, 0xc4, 0x10, 0x5d, 0xc3  
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90
```

# Hopper

## Transformations

```
_main:
    push    rbp
    mov     rbp, rsp
    dd      0x10ec8348
    0xc7, 0x45, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x89
    0x7d, 0xf8, 0x48, 0x89, 0x75, 0xf0, 0x8b, 0x7d
    0xf8, 0x48, 0x8b, 0x75, 0xf0, 0xe8, 0xcc, 0xcc
    0x02, 0x00, 0x48, 0x83, 0xc4, 0x10, 0x5d, 0xc3
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90
```



# Hopper

## Transformations

```
_main:
    push    rbp
    mov     rbp, rsp
    dd      0x10ec8348
    0xc7, 0x45, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x89
    0x7d, 0xf8, 0x48, 0x89, 0x75, 0xf0, 0x8b, 0x7d
    0xf8, 0x48, 0x8b, 0x75, 0xf0, 0xe8, 0xcc, 0xcc
    0x02, 0x00, 0x48, 0x83, 0xc4, 0x10, 0x5d, 0xc3
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90
```

# Hopper

## Transformations

```
_main:
    push    rbp
    mov     rbp, rsp
    dd     0x10ec8348
    db     0xc7, 0x45, 0xfc, 0x00
    db     0x00, 0x00, 0x00, 0x89
    db     0x7d, 0xf8, 0x48, 0x89
    db     0x75, 0xf0, 0x8b, 0x7d
    db     0xf8, 0x48, 0x8b, 0x75
    db     0xf0, 0xe8, 0xcc, 0xcc
    db     0x02, 0x00, 0x48, 0x83
    db     0xc4, 0x10, 0x5d, 0xc3
    db     0x90, 0x90, 0x90, 0x90
    db     0x90, 0x90, 0x90, 0x90
```

# Hopper

## Transformations

Hopefully, Hopper automatically gives a correct type to most bytes of the disassembled file!

But there is no magic for:

- ▶ self-generated code,
- ▶ heavily obfuscated code,
- ▶ some unknown weird construction...



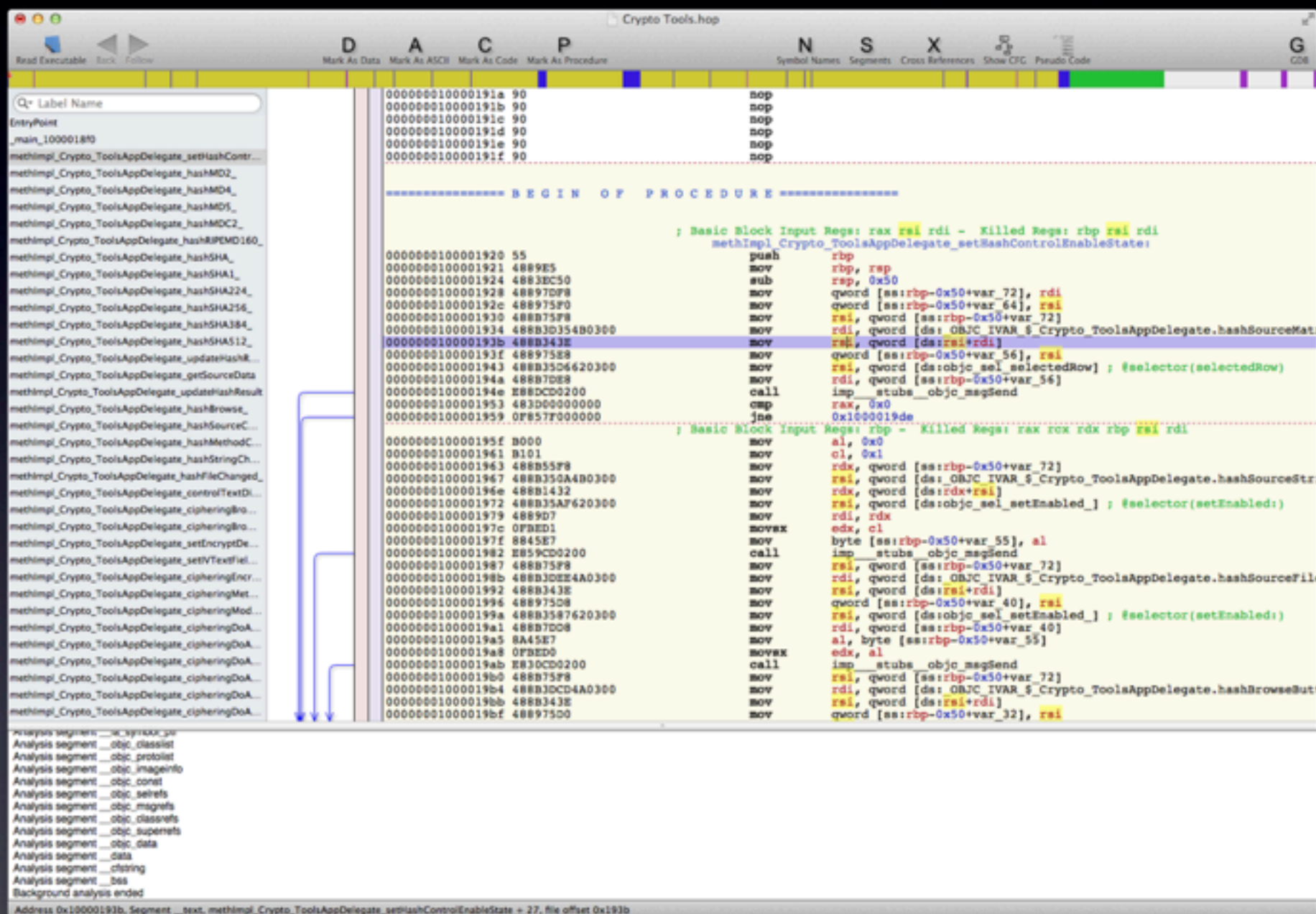
# Hopper

## Transformations

Hopper offers some useful tools:

- ▶ decompiler (pseudo-code),
- ▶ cross references (XREFs),
- ▶ control flow graph,
- ▶ bookmarks,
- ▶ searching tools (hex, ASCII, instruction)

# Hopper





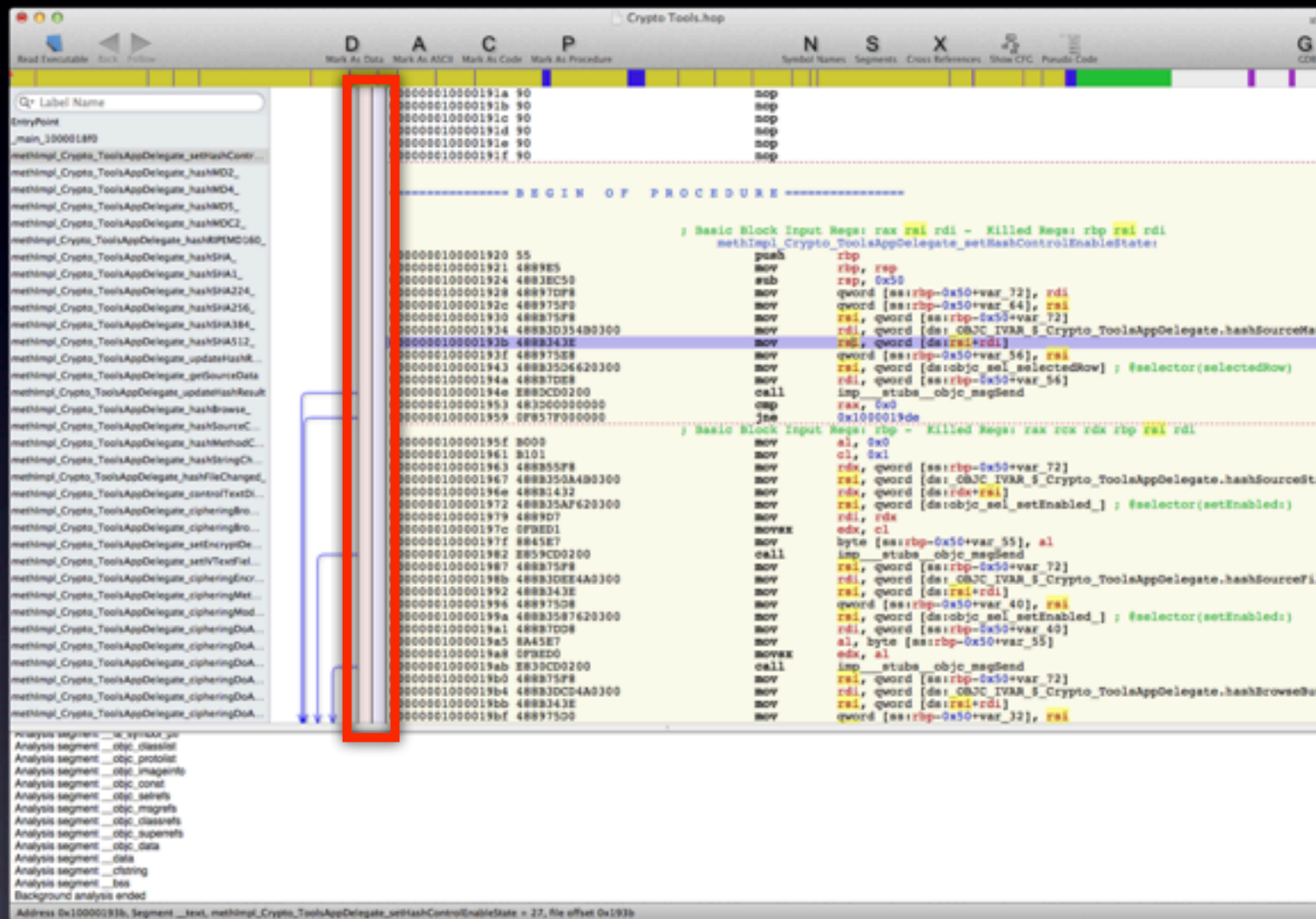
# Hopper

The screenshot displays the Hopper Disassembler's disassembly view for the procedure `methImpl_Crypto_ToolsAppDelegate_setHashControlEnabledState`. The assembly code is shown with comments and is highlighted with a red border. The code includes instructions such as `push rbp`, `mov rbp, rsp`, `sub rsp, 0x50`, and various `mov` and `call` instructions. The control flow graph (CFG) is visible on the left, showing the flow of the procedure. The bottom pane shows the control flow graph (CFG) and the control flow graph (CFG) of the procedure.

## Disassembly View

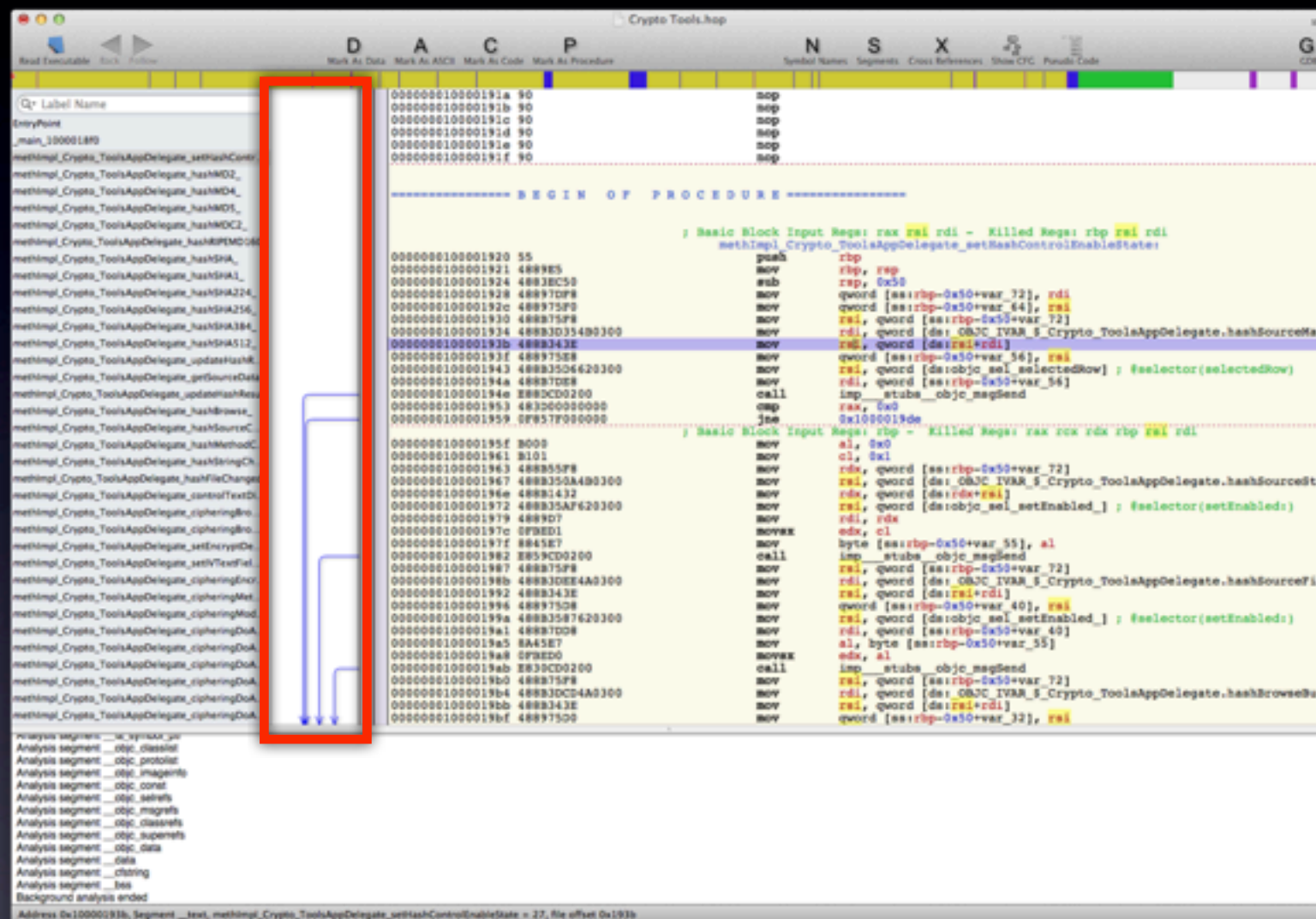


# Hopper

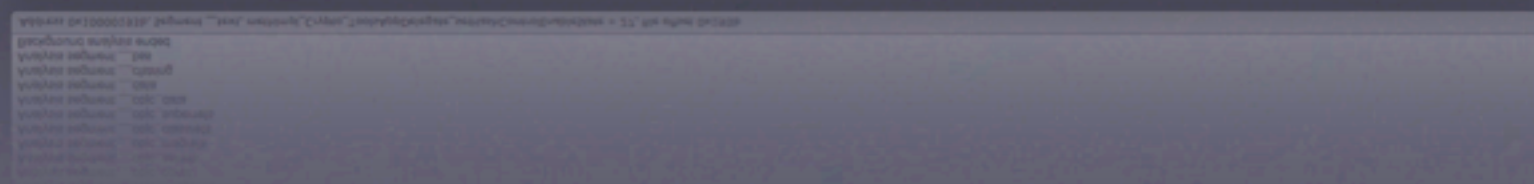


# Bookmarks and Breakpoints

# Hopper

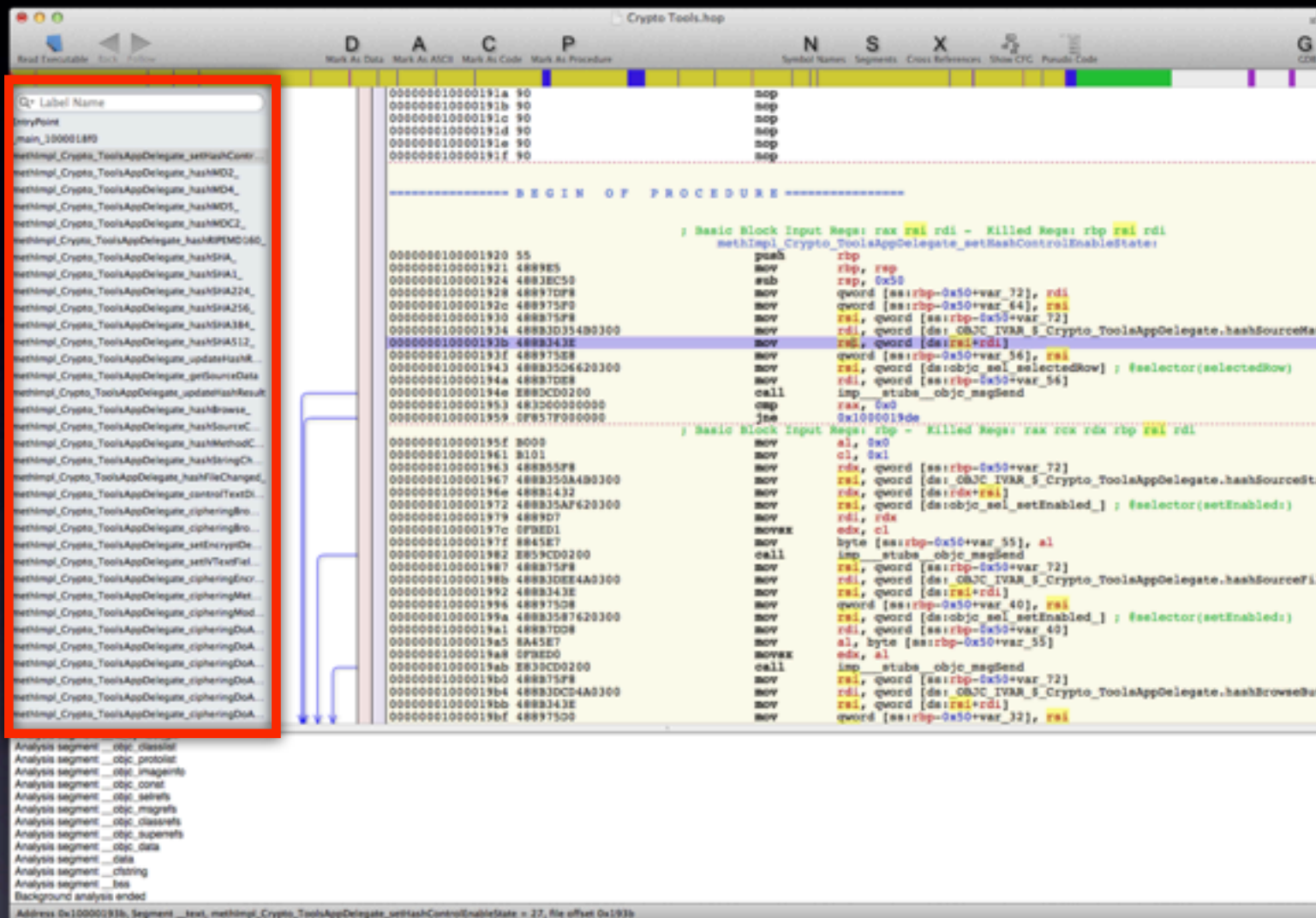


Control Flow  
Information





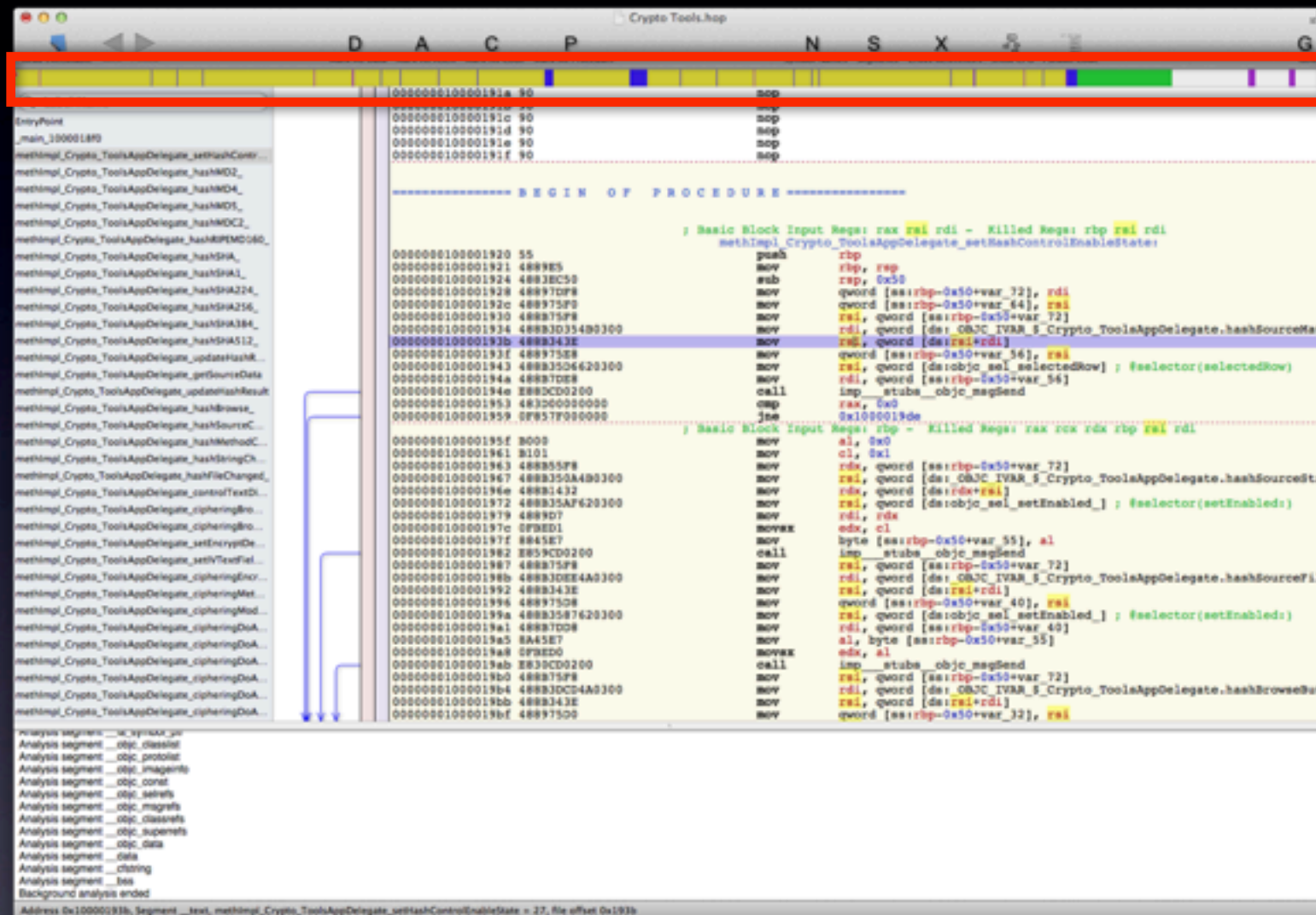
# Hopper



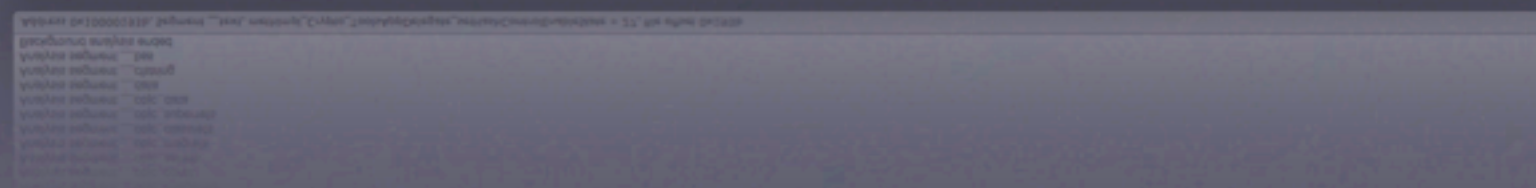
# List of labels



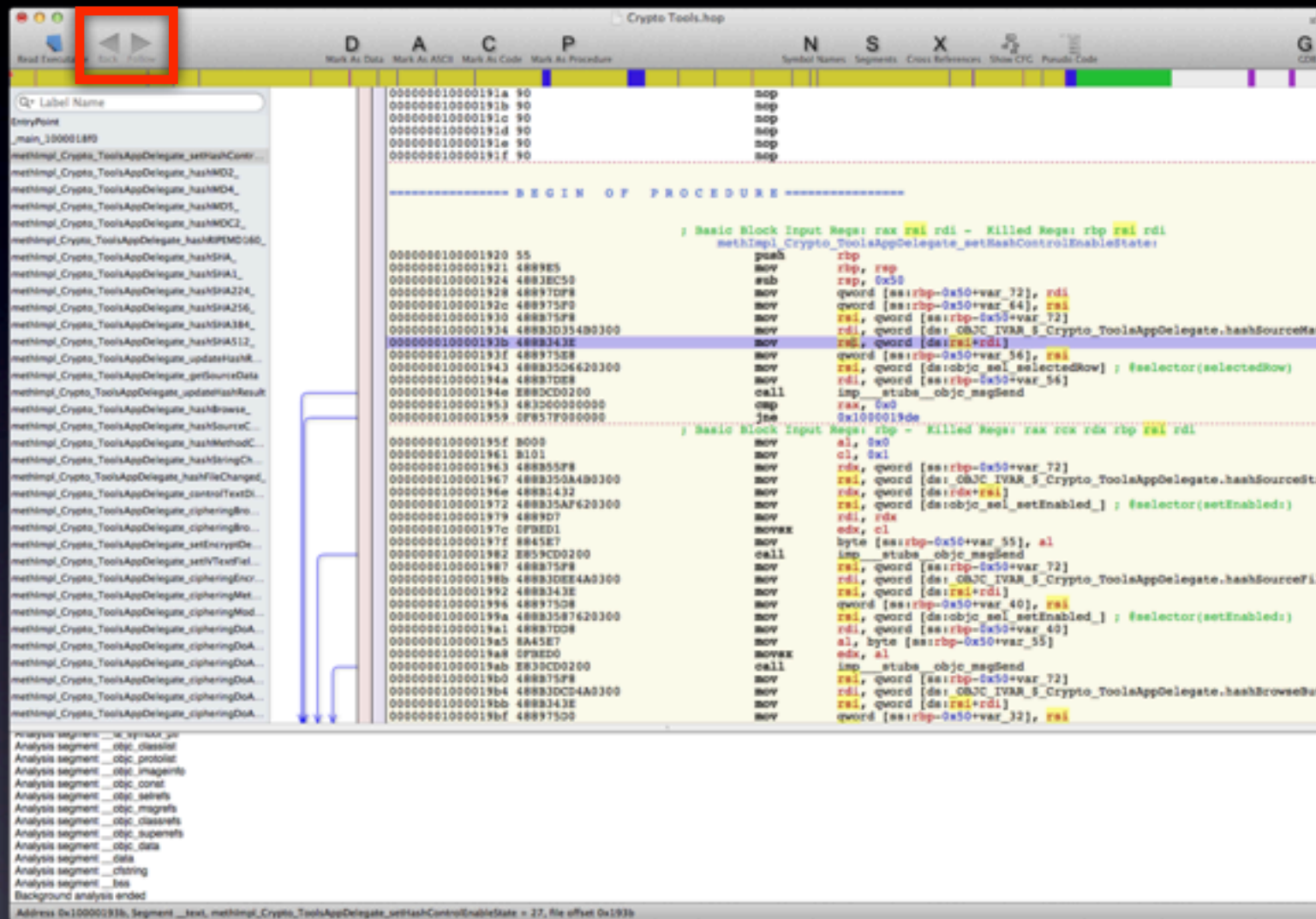
# Hopper



Binary Overview  
and Quick Navigation



# Hopper

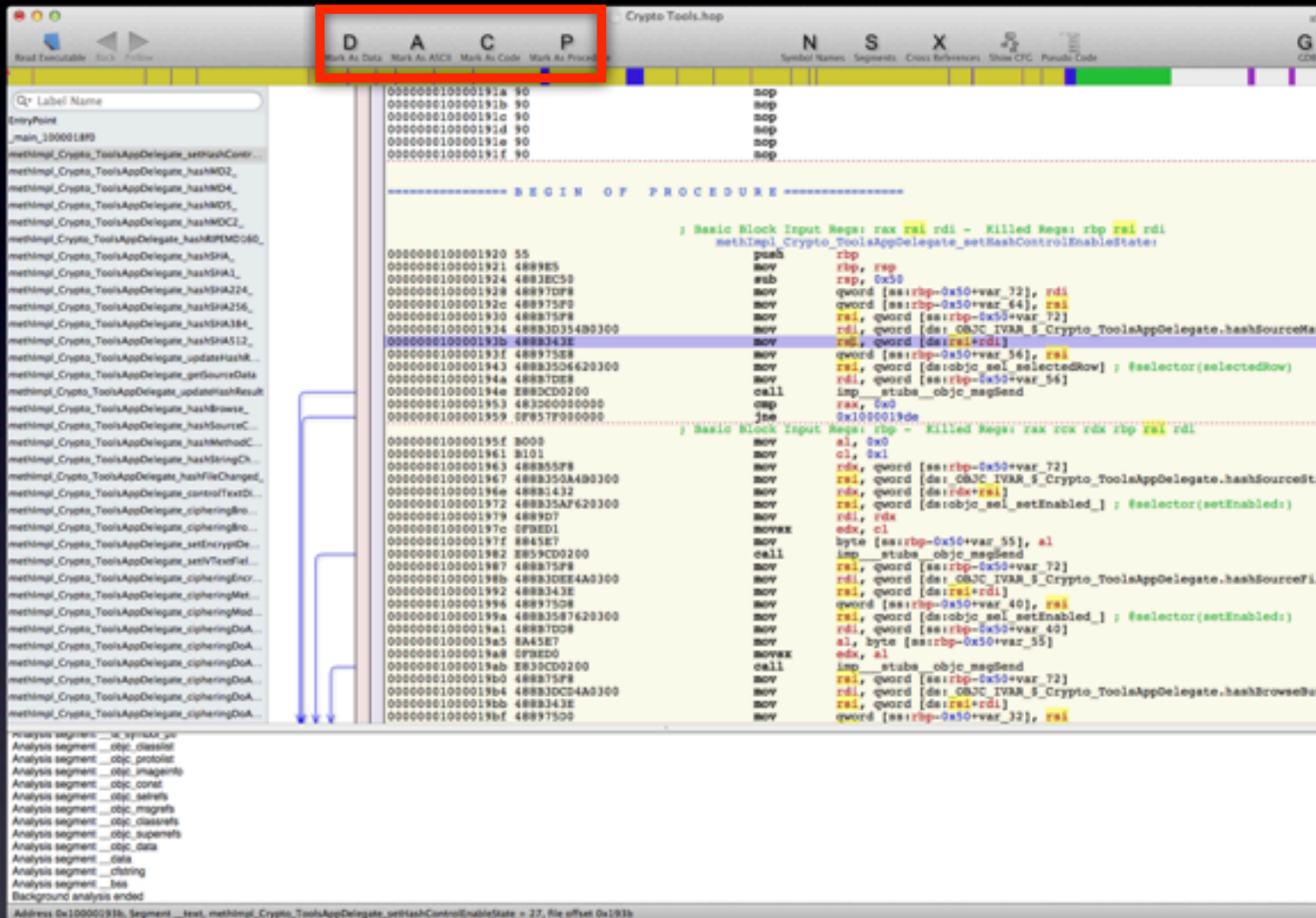


# Navigation



# Hopper

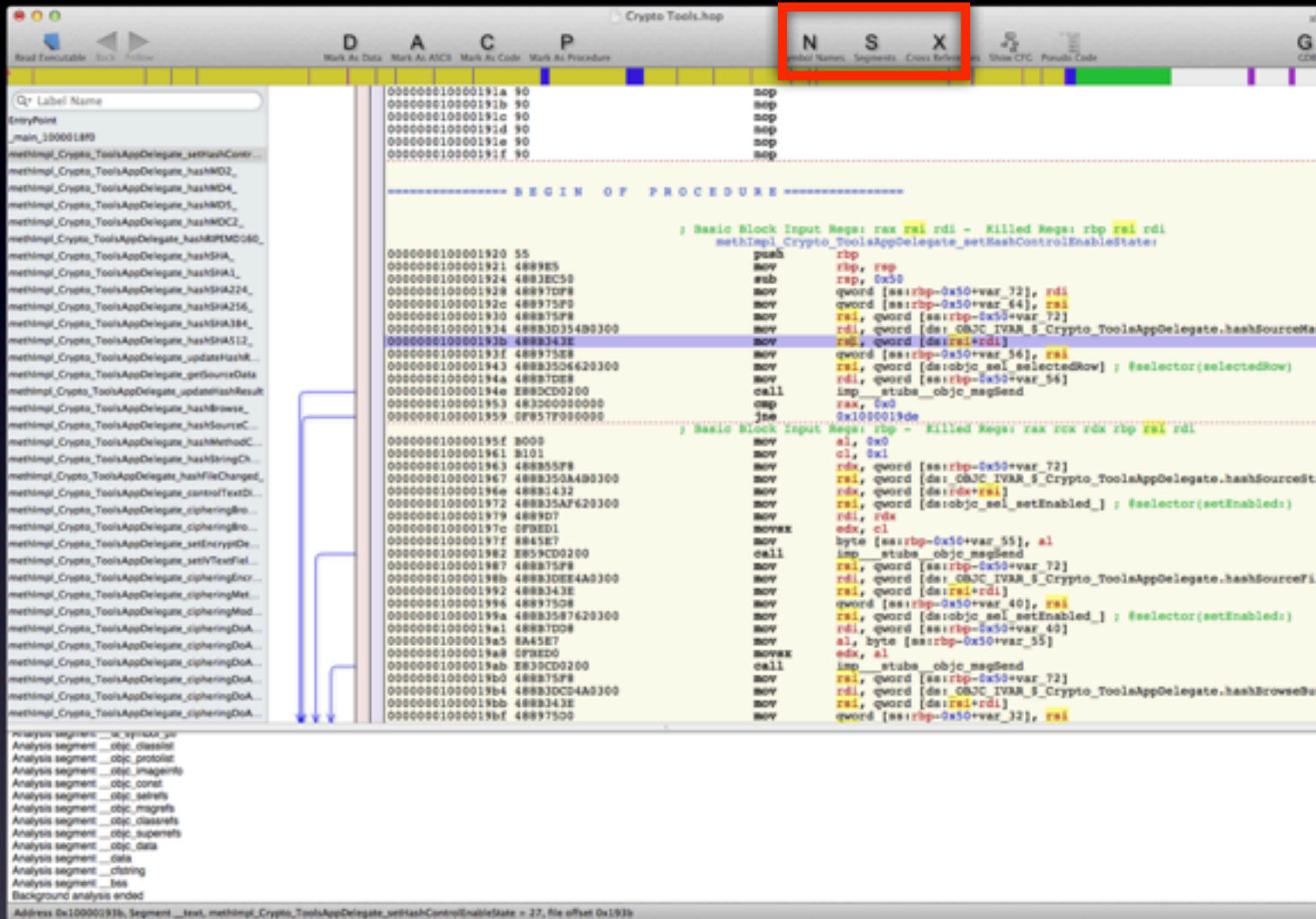
# Transformations



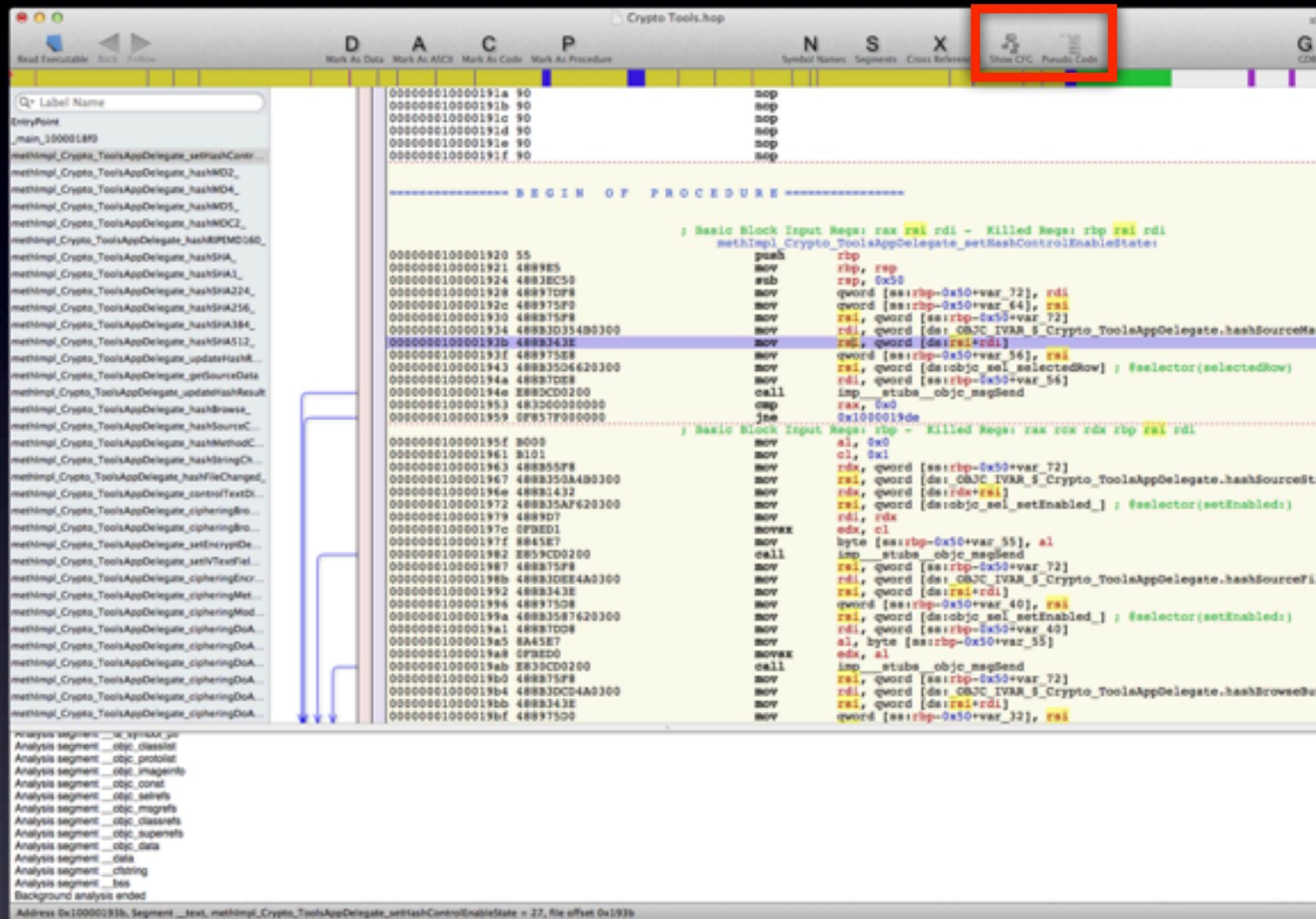


# Hopper

# Searching Tools



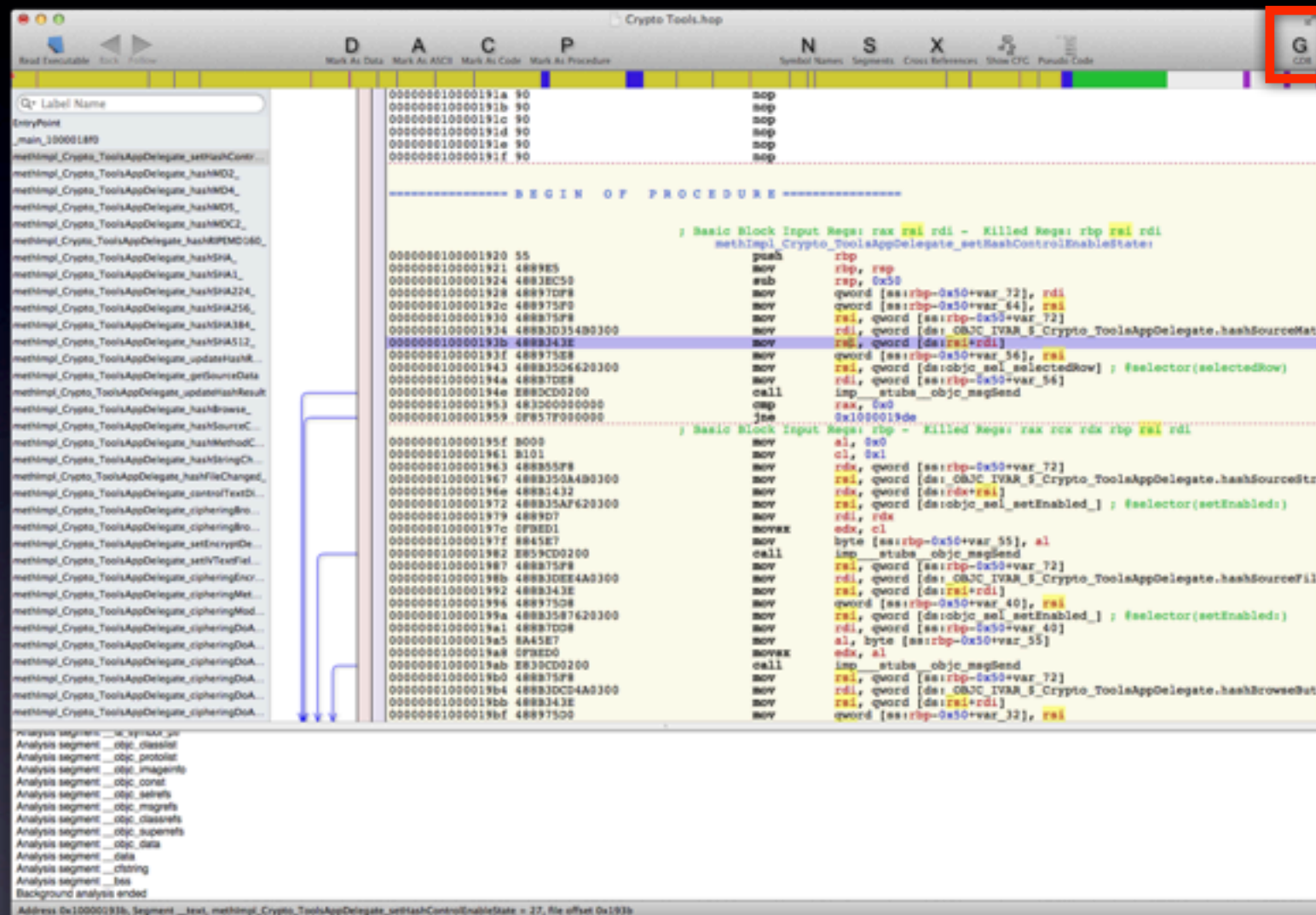
# Hopper



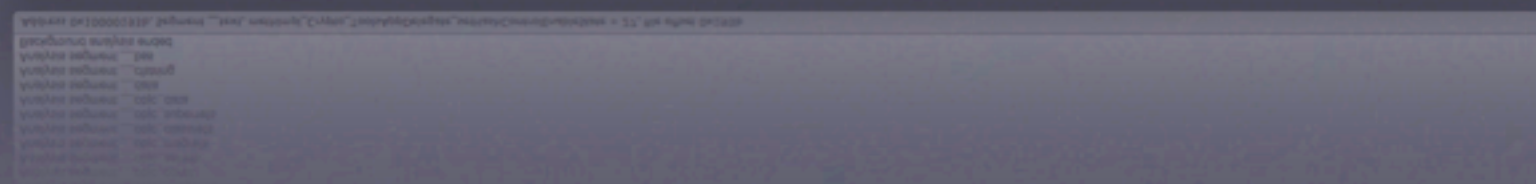
# Visualization



# Hopper



Debugger





# Hopper

# Hopper adds a “procedure” concept to the ASM

```

nop
methImpl Crypto_ToolsAppDelegate_setHashControlEnableState:
push    rbp
mov     rbp, rsp
sub     rsp, 0x50
mov     qword [ss:rbp+0xffffffffffffff8], rdi
mov     qword [ss:rbp+0xffffffffffffff0], rsi
mov     rsi, qword [ss:rbp+0xffffffffffffff8]
mov     rdi, qword [ds:_OBJC_IVAR_$_Crypto_ToolsAppDelegate.hashSourceMatrix]
mov     rsi, qword [ds:rsi+rdi]
mov     qword [ss:rbp+0xfffffffffffffe8], rsi
mov     rsi, qword [ds:objc_sel_selectedRow]; #selector(selectedRow)
mov     rdi, qword [ss:rbp+0xfffffffffffffe8]
call    imp__stubs_objc_msgSend
cmp     rax, 0x0
jne     0x10000019de
mov     al, 0x0
mov     cl, 0x1
mov     rdx, qword [ss:rbp+0xffffffffffffff8]
mov     rsi, qword [ds:_OBJC_IVAR_$_Crypto_ToolsAppDelegate.hashSourceString]
mov     rdx, qword [ds:rdx+rsi]
mov     rsi, qword [ds:objc_sel_setEnabled]; #selector(setEnabled:)
mov     rdi, rdx
movsx   edx, cl
mov     byte [ss:rbp+0xe7], al
call    imp__stubs_objc_msgSend
mov     rsi, qword [ss:rbp+0xffffffffffffff8]
mov     rdi, qword [ds:_OBJC_IVAR_$_Crypto_ToolsAppDelegate.hashSourceFile]
mov     rsi, qword [ds:rsi+rdi]
mov     qword [ss:rbp+0xfffffffffffffd8], rsi
mov     rsi, qword [ds:objc_sel_setEnabled]; #selector(setEnabled:)
mov     rdi, qword [ss:rbp+0xfffffffffffffd8]
mov     al, byte [ss:rbp+0xe7]
movsx   edx, al
call    imp__stubs_objc_msgSend
mov     rsi, qword [ss:rbp+0xffffffffffffff8]
mov     rdi, qword [ds:_OBJC_IVAR_$_Crypto_ToolsAppDelegate.hashBrowseButton]
mov     rsi, qword [ds:rsi+rdi]
mov     qword [ss:rbp+0xfffffffffffffd0], rsi
mov     rsi, qword [ds:objc_sel_setEnabled]; #selector(setEnabled:)
mov     rdi, qword [ss:rbp+0xfffffffffffffd0]
mov     byte [ss:rbp+0xe7], al
xor     eax, eax
wopd   dword [ss:rbp+0xffffffffffffe0]
wopd   dword [qs:objc_sel_setEnabled]; #selector(setEnabled:)
wopd   dword [ss:rbp+0xffffffffffffe0]
wopd   dword [qs:eax+eax]
wopd   dword [qs:_OBJC_IVAR_$_Crypto_ToolsAppDelegate.hashBrowseButton]
wopd   dword [ss:rbp+0xffffffffffffe8]
csrrl  rax, objc_msgSend
wosq   qword [ss:rbp+0xe7]
wopd   dword [ss:rbp+0xe7]
wopd   dword [ss:rbp+0xffffffffffffe0]
wopd   dword [qs:objc_sel_setEnabled]; #selector(setEnabled:)

```

# Hopper

# Hopper adds a “procedure” concept to the ASM

```

; Basic Block Input Regs: rax rsi rdi - Killed Regs: rbp rsi rdi
methImpl Crypto_ToolsAppDelegate_setHashControlEnableState:
push    rbp
mov     rbp, rsp
sub     rsp, 0x50
mov     qword [ss:rbp-0x50+var_72], rsi
mov     qword [ss:rbp-0x50+var_64], rdi
mov     rsi, qword [ss:rbp-0x50+var_72]
mov     rdi, qword [ds:_OBJC_IVAR_$_Crypto_ToolsAppDelegate.hashSourceMatrix]
mov     rsi, qword [ds:rsi+rdi]
mov     qword [ss:rbp-0x50+var_56], rsi
mov     rsi, qword [ds:objc_sel_selectedRow] ; #selector(selectedRow)
mov     rdi, qword [ss:rbp-0x50+var_56]
call    imp_stubs_objc_msgSend
cmp     rax, 0x0
jne     0x1000019de
; Basic Block Input Regs: rbp - Killed Regs: rax rcx rdx rbp rsi rdi
mov     al, 0x0
mov     cl, 0x1
mov     rdx, qword [ss:rbp-0x50+var_72]
mov     rsi, qword [ds:_OBJC_IVAR_$_Crypto_ToolsAppDelegate.hashSourceString]
mov     rdx, qword [ds:rdx+rsi]
mov     rsi, qword [ds:objc_sel_setEnabled_] ; #selector(setEnabled:)
mov     rdi, rdx
movzx   edx, cl
mov     byte [ss:rbp-0x50+var_55], al
call    imp_stubs_objc_msgSend
mov     rsi, qword [ss:rbp-0x50+var_72]
mov     rdi, qword [ds:_OBJC_IVAR_$_Crypto_ToolsAppDelegate.hashSourceFile]
mov     rsi, qword [ds:rsi+rdi]
mov     qword [ss:rbp-0x50+var_40], rsi
mov     rsi, qword [ds:objc_sel_setEnabled_] ; #selector(setEnabled:)
mov     rdi, qword [ss:rbp-0x50+var_40]
mov     al, byte [ss:rbp-0x50+var_55]
movzx   edx, al
call    imp_stubs_objc_msgSend
mov     rsi, qword [ss:rbp-0x50+var_72]
mov     rdi, qword [ds:_OBJC_IVAR_$_Crypto_ToolsAppDelegate.hashSourceString]
mov     rsi, qword [ds:rsi+rdi]
mov     qword [ss:rbp-0x50+var_40], rsi
mov     rsi, qword [ds:objc_sel_setEnabled_] ; #selector(setEnabled:)
mov     rdi, qword [ss:rbp-0x50+var_40]
mov     al, byte [ss:rbp-0x50+var_55]
movzx   edx, al
call    imp_stubs_objc_msgSend

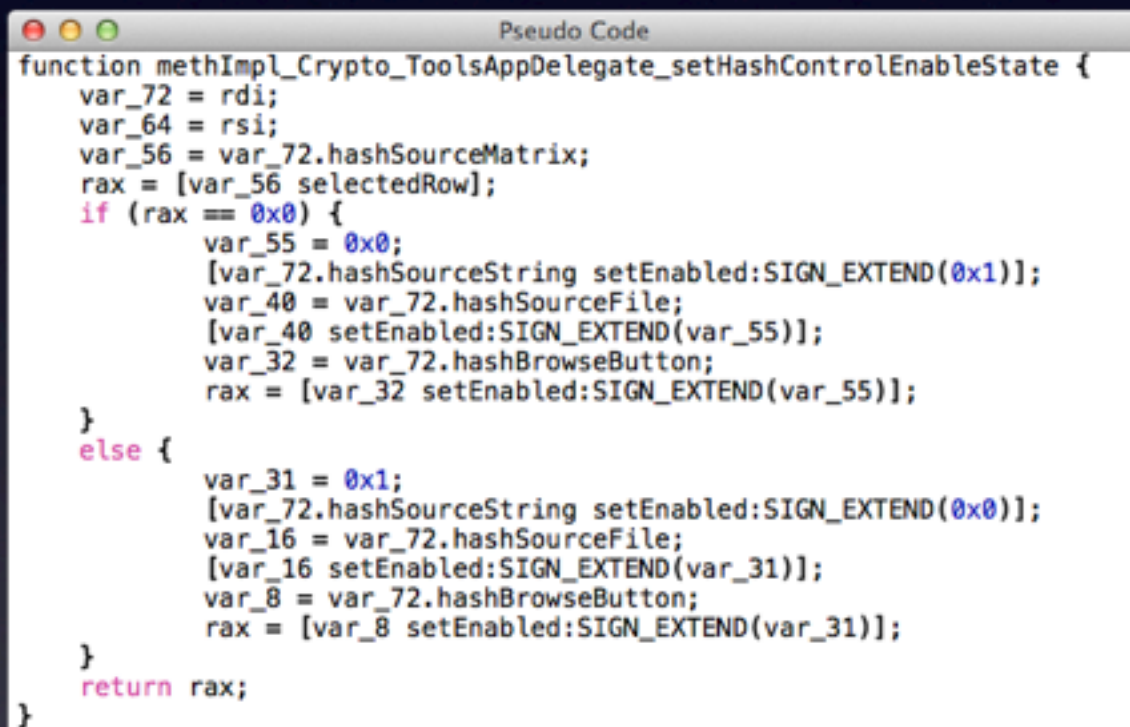
```

- ▶ Hopper tries to discover the procedure bounds,
- ▶ it slices the procedure into basic blocks,
- ▶ it computes register usage across basic blocks,
- ▶ it analyses the context and gives stack variables a name.



# Hopper

Once marked as a procedure, Hopper can decompile it



```
function methImpl_Crypto_ToolsAppDelegate_setHashControlEnableState {
    var_72 = rdi;
    var_64 = rsi;
    var_56 = var_72.hashSourceMatrix;
    rax = [var_56 selectedRow];
    if (rax == 0x0) {
        var_55 = 0x0;
        [var_72.hashSourceString setEnabled:SIGN_EXTEND(0x1)];
        var_40 = var_72.hashSourceFile;
        [var_40 setEnabled:SIGN_EXTEND(var_55)];
        var_32 = var_72.hashBrowseButton;
        rax = [var_32 setEnabled:SIGN_EXTEND(var_55)];
    }
    else {
        var_31 = 0x1;
        [var_72.hashSourceString setEnabled:SIGN_EXTEND(0x0)];
        var_16 = var_72.hashSourceFile;
        [var_16 setEnabled:SIGN_EXTEND(var_31)];
        var_8 = var_72.hashBrowseButton;
        rax = [var_8 setEnabled:SIGN_EXTEND(var_31)];
    }
    return rax;
}
```

- ▶ Hopper tries to discover as much Objective-C constructs as possible,
- ▶ there is still no magic: this is pseudo-code, not compilable code!
- ▶ but code is very readable and so useful!



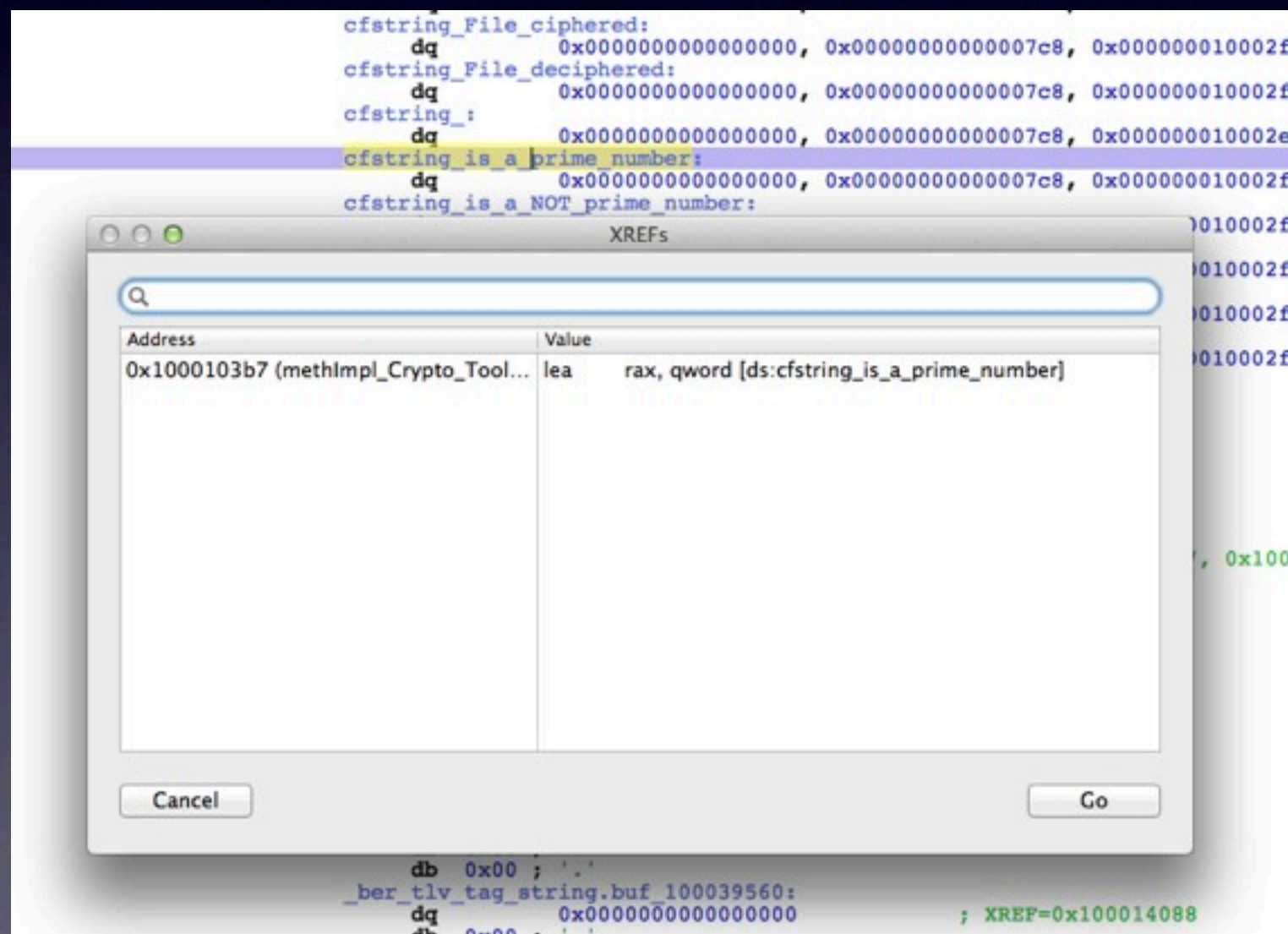
# Hopper

Methods can be displayed as a graph too:



# Hopper

Hopper can search where a variable is used



# Use Cases

*Une petite démonstration  
s'impose!*