The M/o/Vfuscator

Turning 'mov' into a soul-crushing RE nightmare

{ domas, @xoreaxeaxeax

./bio

- Christopher Domas
  - Cyber Security Researcher @ Battelle Memorial Institute

- objdump –d –Mintel a.out

```
4004e9:     mov   DWORD PTR [rbp-0x8],0x0
4004f2:     push  600004
4004f8:     call  printf
4004fa:     pop   eax
4004fc:     add   DWORD PTR [rbp-0x8],0x1
400500:     cmp   DWORD PTR [rbp-0x8],0x100
400507:     jle   4004f2 <main+0xb>
```

- REMath
  (github.com/REMath)

- Stephen Dolan
- www.cl.cam.ac.uk/~sd601/papers/mov.pdf

It is well-known that the x86 instruction set is baroque, overcomplicated, and redundantly redundant. We show just how much fluff it has by demonstrating that it remains Turing-complete when reduced to just one instruction.

- Stephen Dolan

&mov destination, source

mov

- Any code we write …
- … can be written as a set of movs instead
- … *and nothing else*
- *Really?*
- That'd be tough to reverse engineer, wouldn't it?

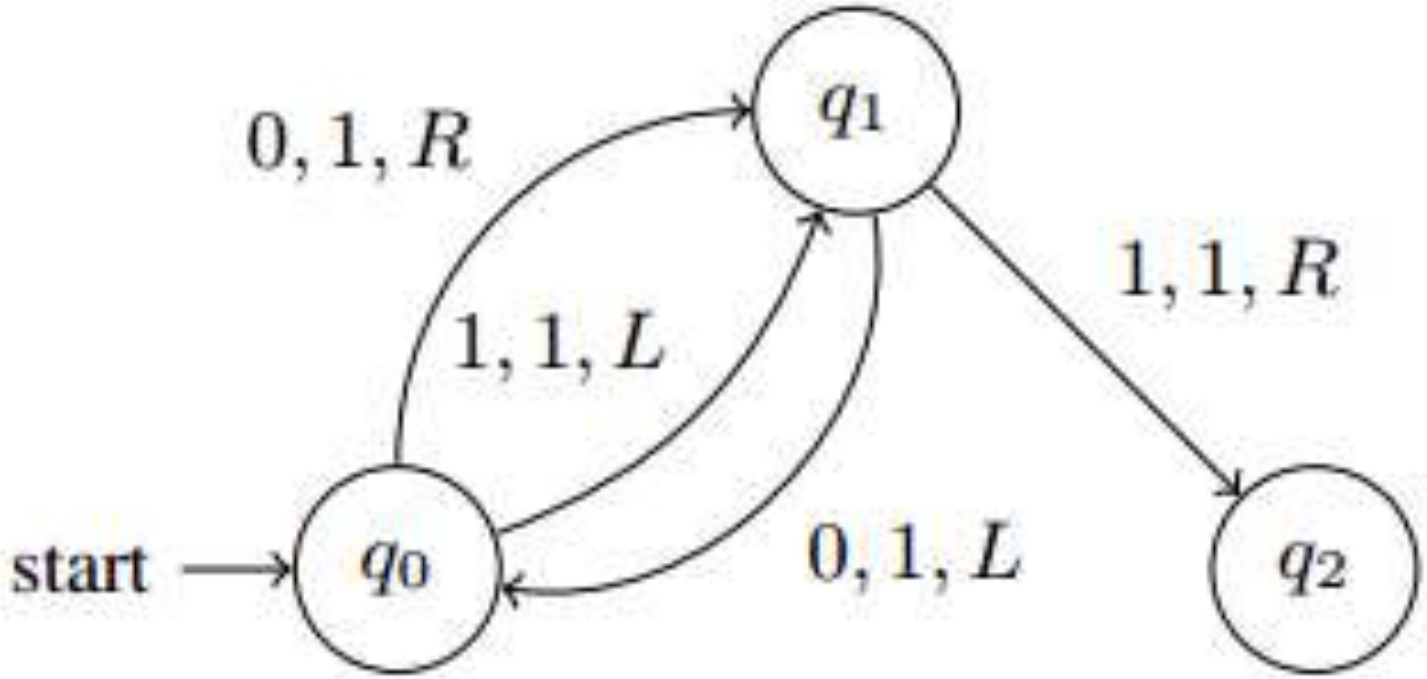
Turing Complete?

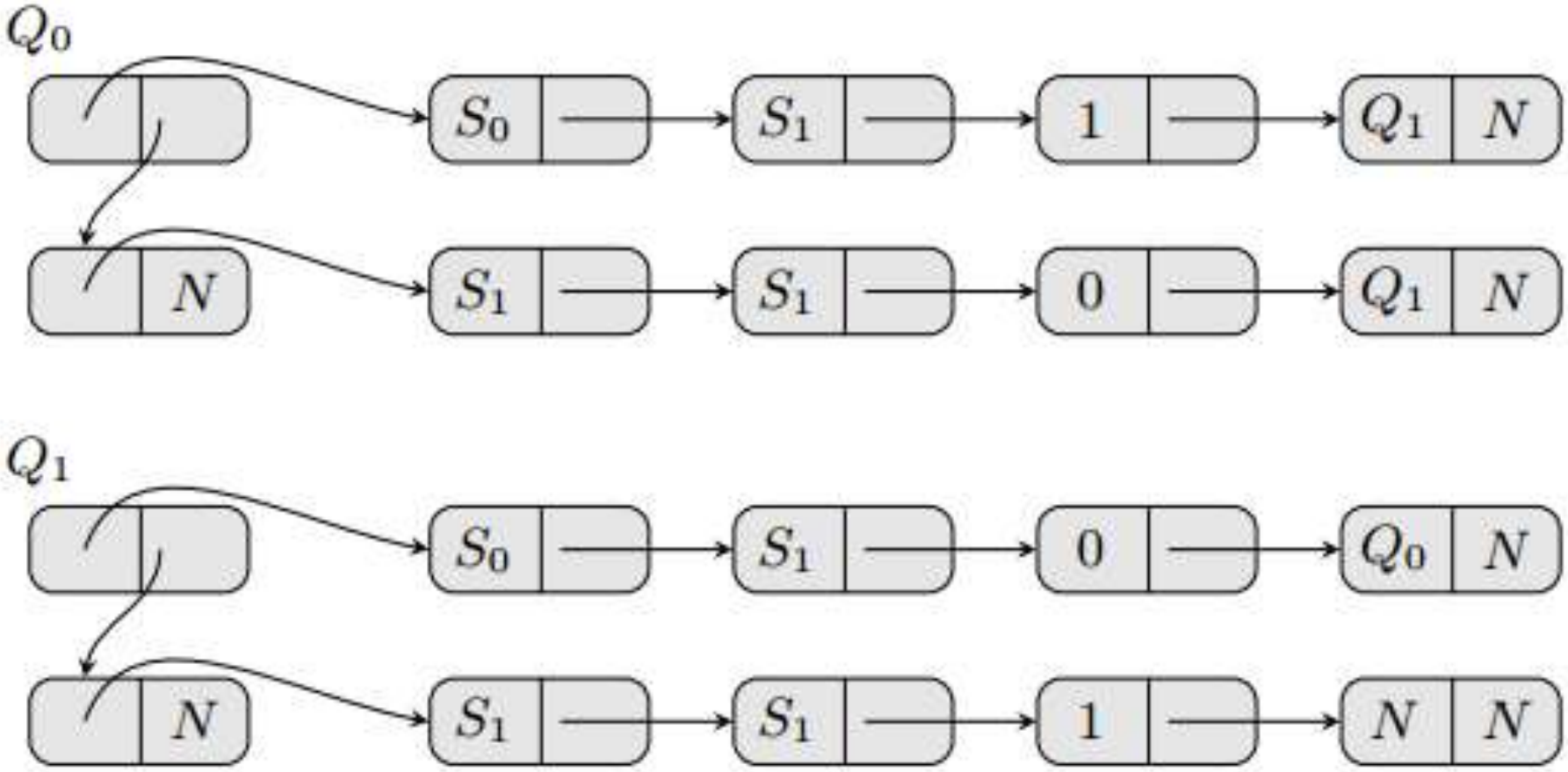
```
4004e9:   mov   DWORD PTR [rbp-0x8],0x0
4004f2:   push  600004
4004f8:   call  printf
4004fa:   pop   eax
4004fc:   add   DWORD PTR [rbp-0x8],0x1
400500:   cmp   DWORD PTR [rbp-0x8],0x100
400507:   jle   4004f2 <main+0xb>
```

```
80515bc:    mov   eax,ds:0x835d81a
80515c1:    mov   ebx,DWORD PTR [eax+0x835d6fc]
80515c7:    mov   edx,DWORD PTR ds:0x835d7da
80515cd:    mov   eax,0x0
80515d2:    mov   al,BYTE PTR [ebx+edx*1]
80515d5:    mov   al,BYTE PTR [eax+0x835dc7e]
80515db:    mov   BYTE PTR [ebx+edx*1],al
80515de:    mov   eax,ds:0x835d81a
80515e3:    mov   ebx,DWORD PTR [eax+0x835d6fc]
80515e9:    mov   edx,DWORD PTR ds:0x835d7da
80515ef:    mov   eax,0x0
80515f4:    mov   al,BYTE PTR [ebx+edx*1]
```

- $M = \langle Q, q_0, \Sigma, \sigma_0, \delta \rangle$
- A finite set of states Q
- A distinguished start state $q_0 \in Q$
- A finite set of symbols Σ
- A distinguished blank symbol $\sigma_0 \in \Sigma$
- A transition table δ, which is a partial function $Q \times \Sigma \rightarrow \Sigma \times \{L, R\} \times Q$

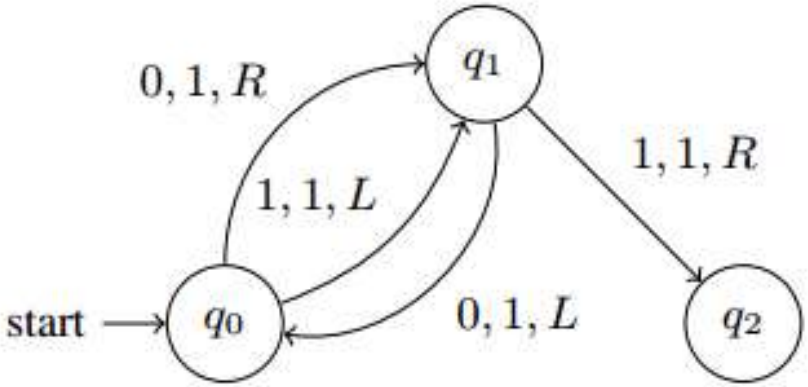
Turing Machines


q_1
$0, 1, R$
$1, 1, L$
$1, 1, R$
start
q_0
$0, 1, L$
q_2

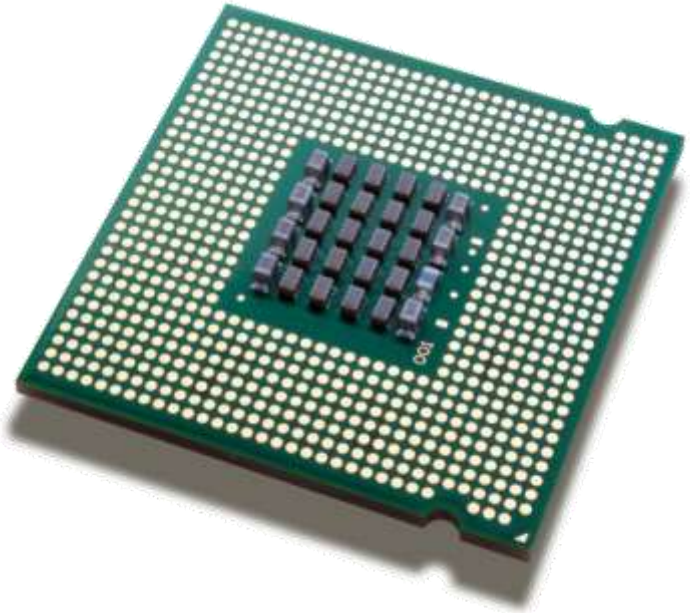

Q_0
S_0
S_1
1
Q_1
N
N
S_1
S_1
0
Q_1
N
Q_1
S_0
S_1
0
Q_0
N
N
S_1
S_1
1
N
N

- Fascinating
- But academic

So?

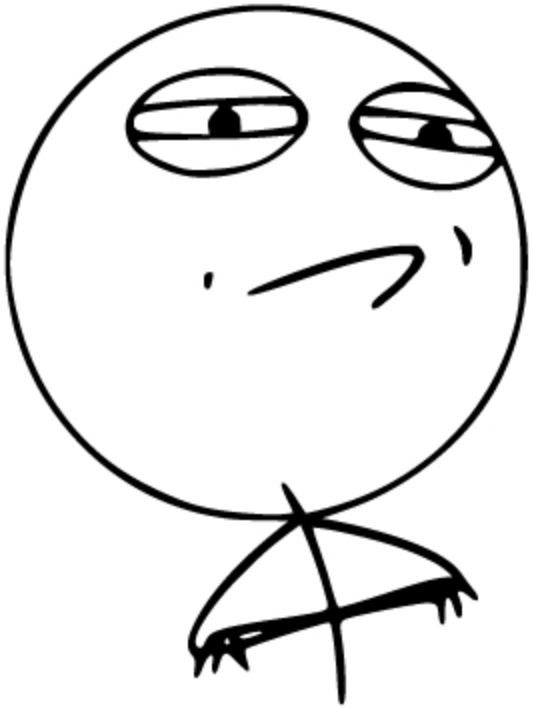


$\neq$



Removing all but the mov instruction from future iterations of the x86 architecture would have many advantages: the instruction format would be greatly simplified, the expensive decode unit would become much cheaper, and silicon currently used for complex functional units could be repurposed as even more cache. As long as someone else implements the compiler.

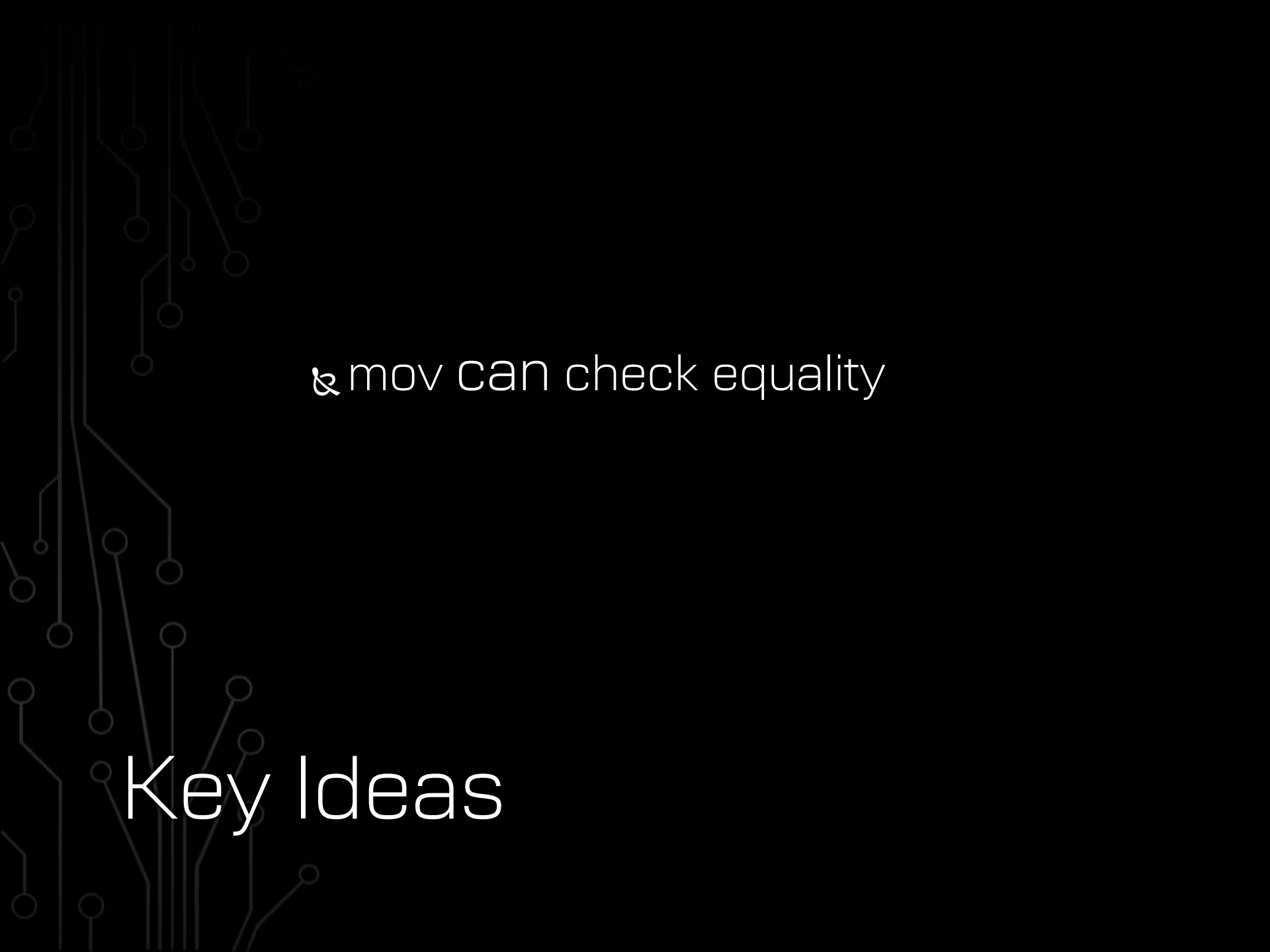
- Stephen Dolan



CHALLENGE ACCEPTED

- Where to begin…?

Key Ideas


mov can check equality
Key Ideas

- mov [x], 0
- mov [y], 1
- mov R, [x]

x==y

- x=3, y=3

- mov [x], 0
- mov [y], 1
- mov R, [x]

x==y

- x=2, y=3

- mov [x], 0
- mov [y], 1
- mov R, [x]

x==y

Key Ideas

- There is only one code path
- Designed correctly, a code block can
  - Have an effect
  - Have no effect
  - Depending on the initial state

Key Ideas

- Requires a single jmp instruction to loop back to the beginning
- Incidental
- Ideas on fixing this later

- start:
- mov …
- mov …
- mov …
- mov …
- mov …
- mov …
- mov …
- mov …
- mov …
- mov …
- mov …
- mov …
- jmp start

Key Ideas

- Dolan's Turing Machine design requires an invalid memory address, for halting

Step 2: ?

Idea…

- Build on Dolan's ideas
  - Adapt primitive TM operations for higher level logic
- Work on actual data, not abstract symbols
- Add new operations
  - If/else
  - Arithmetic
  - Logic
  - Jumps
  - Loops
  - Etc…
- Bring it closer to something we can use

Implementing if

- IF X == Y THEN
  - X = 100

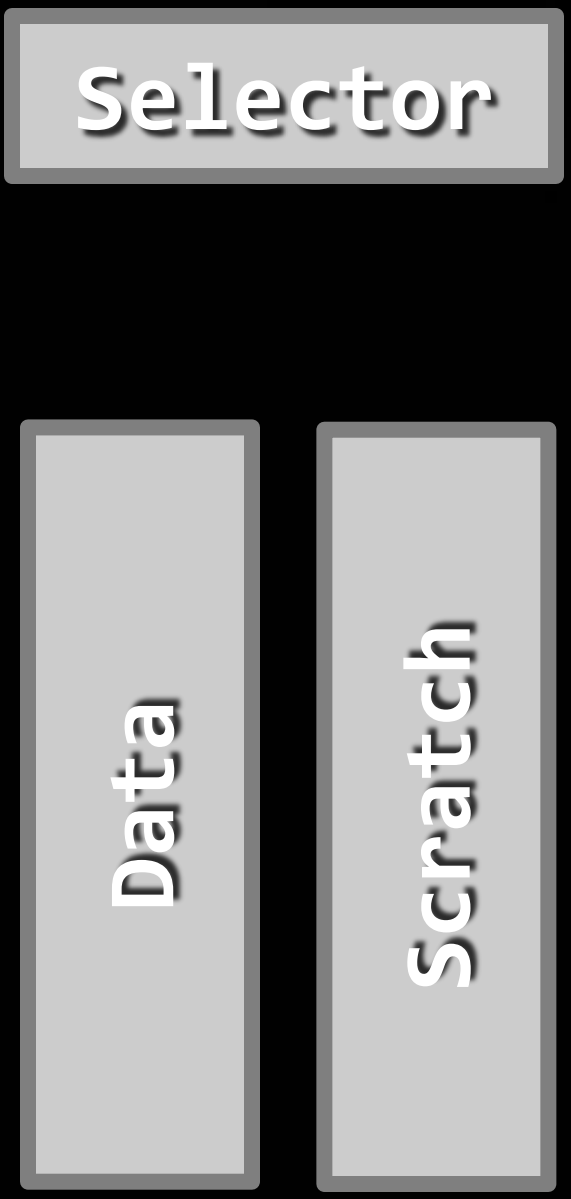
Implementing if

Implementing if

- The catch:
  - We have no branches
  - All paths execute, no matter what
- Solution:
  - Force a path to operate on “dummy” data, if we don’t want its results

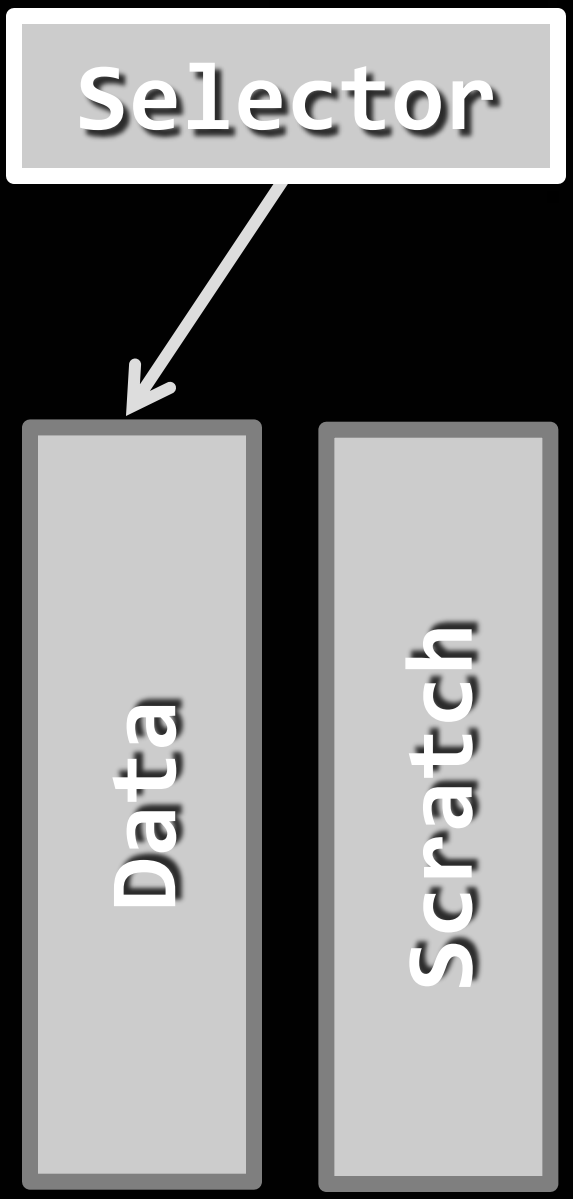
Implementing if

- IF X == Y THEN
  - X = 100



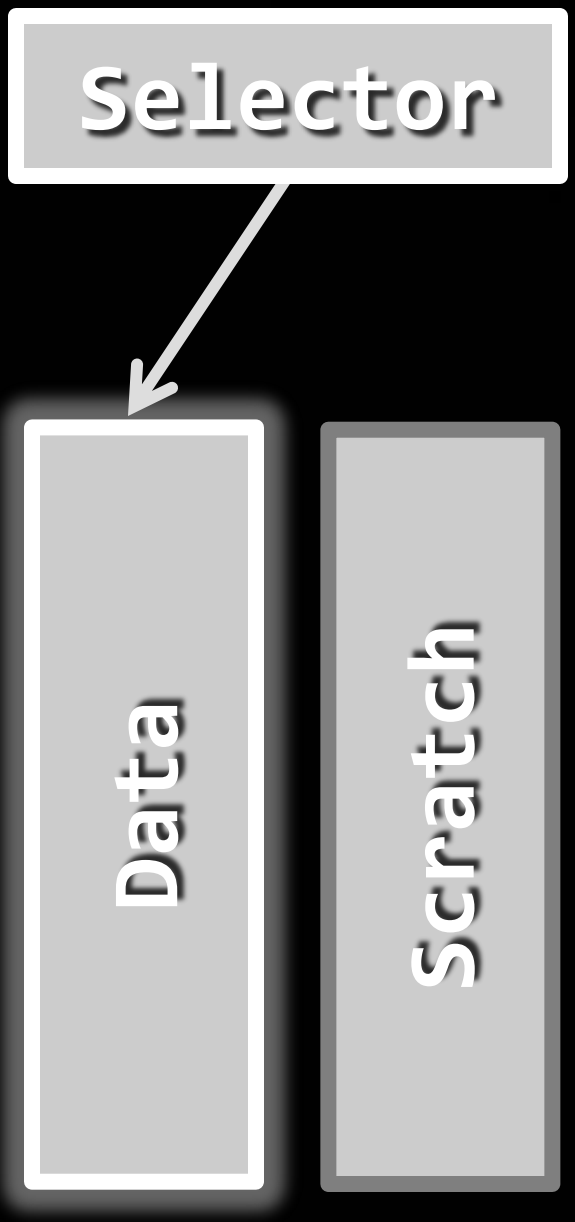
Implementing if

- IF X == Y THEN ⇐
  - X = 100 ⇐



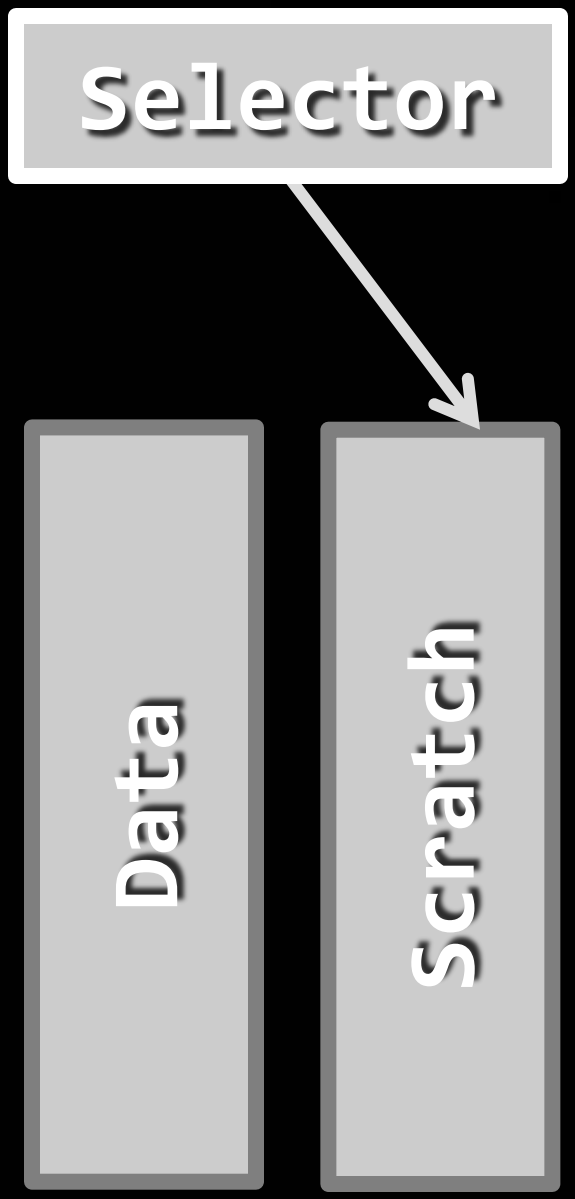
Implementing if

- IF X == Y THEN
  - X = 100



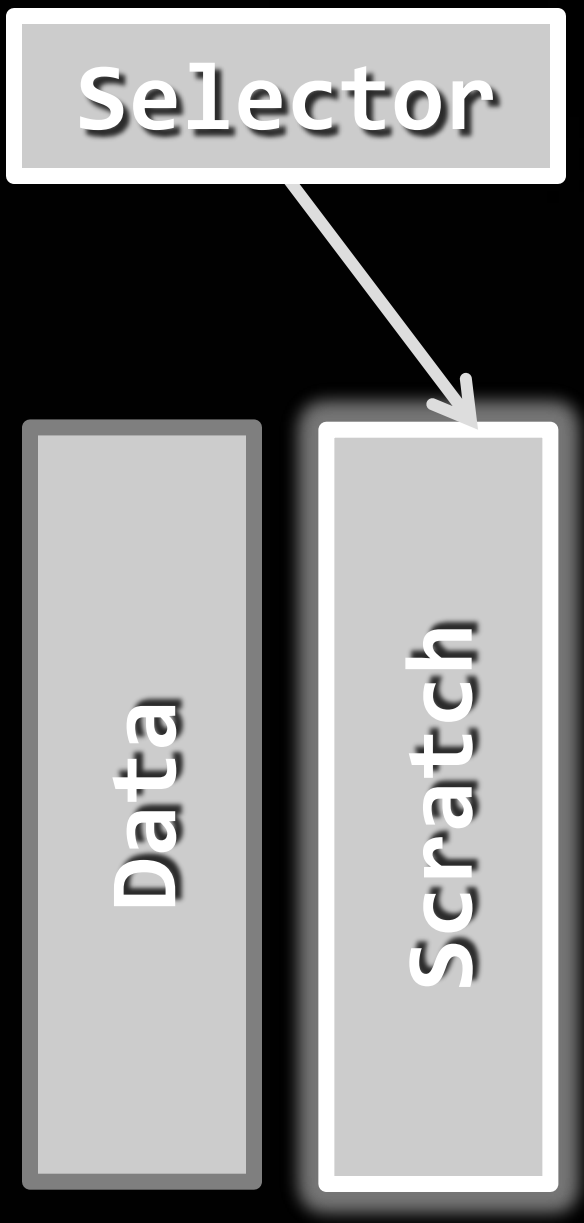
Implementing if

- IF X == Y THEN ⇐
  - X = 100 ⇐



Implementing if

- IF X == Y THEN
  - X = 100



```
IF X == Y THEN
    X = 100

int* SELECT_X[ ] = { &DUMMY_X, &X }
*SELECT_X[ X == Y ] = 100
```

Implementing if

```
section .data
X: dd 0
DUMMY_X: dd 0
Y: dd 0
DUMMY_Y: dd 0
SELECT_X: dd DUMMY_X, X
SELECT_Y: dd DUMMY_Y, Y
```

Implementing if

```
; X == Y
mov eax, [X]
mov [eax], 0
mov eax, [Y]
mov [eax], 4
mov eax, [X]
; X = 100
mov eax, [SELECT_X + eax]
mov [eax], 100
```

Implementing if

Implementing if

- Solution:
  - Add a “selector” “function” (pointer array) to all variables

Implementing if

- About that equality check…

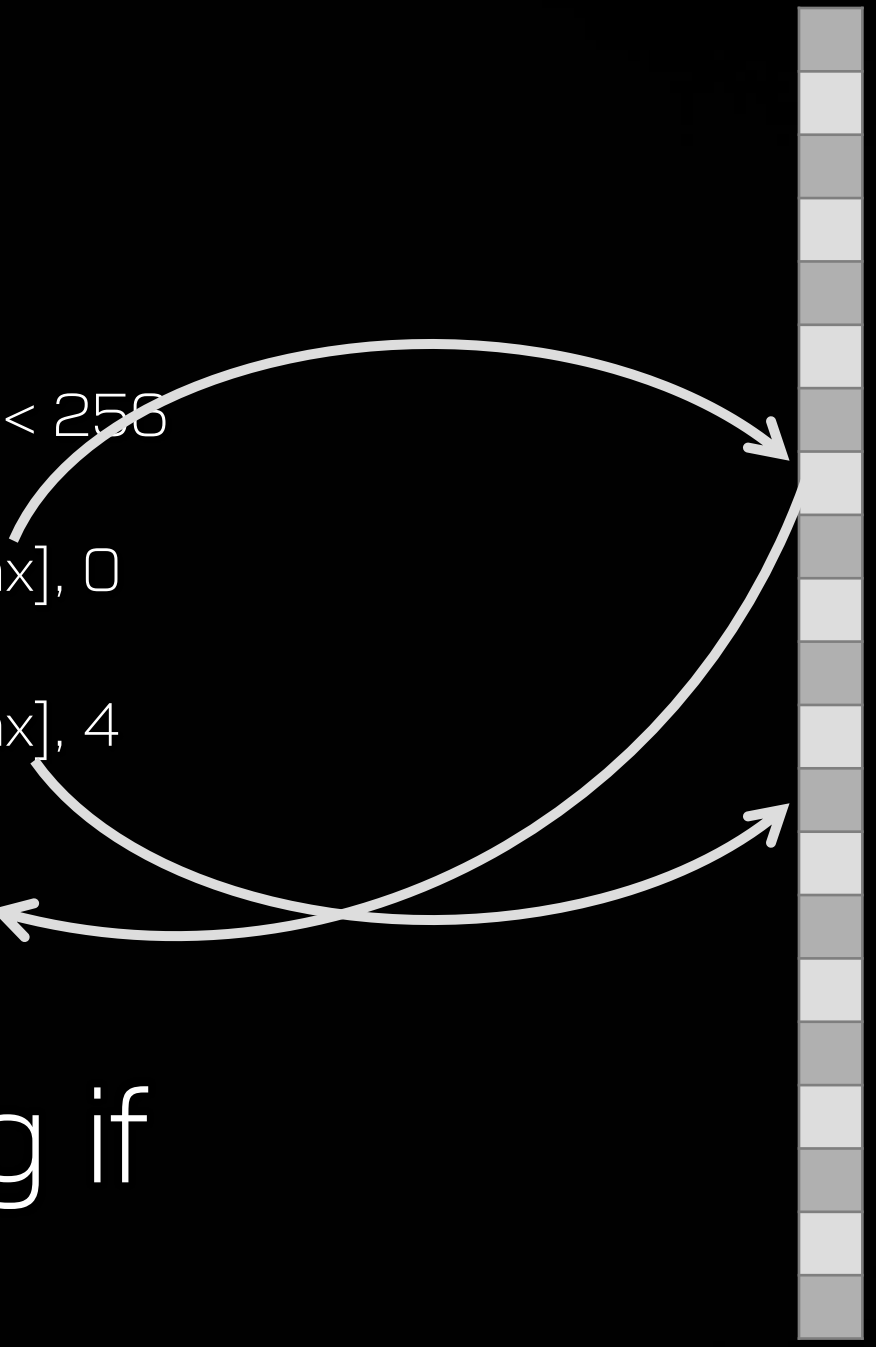
```
mov eax, [X]
mov [eax], 0
```

- We can't just write to arbitrary spots in memory (although we could with the original TM design)

Implementing if

- Several solutions
- Easiest:
  - Limit ourselves to 1 byte data
  - Create a 256 byte scratch array for equality testing

```
section .bss
EQ: resb 256

section .text
; valid on for X, Y < 256
mov al, [X]
mov byte [EQ+eax], 0
mov al, [Y]
mov byte [EQ+eax], 4
mov al, [X]
mov al, [EQ+eax]
```



Implementing if

Implementing if

- Simple extensions give us
  - if/else
  - if/elseif/else
  - Inequality checks

```
%macro eq 3
        mov eax, 0
        mov al, [%2]
        mov byte [e+eax], 0
        mov byte [e+%3], 4
        mov al, [e+eax]
        mov [%1], al
%endmacro
```

```
%macro neq 3
        mov eax, 0
        mov al, [%2]
        mov byte [e+eax], 4
        mov byte [e+%3], 0
        mov al, [e+eax]
        mov [%1], al
%endmacro
```

```
; create selector
%macro c_s 1
        %1:   dd 0
        d_%1: dd 0
        s_%1: dd d_%1, %1
%endmacro
```

Loops and branches

- Extend the if/else idea
- On each branch
  - If the branch is taken
    - Store the target address
    - Turn execution “off”
  - If the branch is not taken
    - Leave execution “on”

- On each operation
  - If execution is on
    - Run the operation on real data
  - If execution is off
    - Is current address the stored branch target?
      - Yes?
        - Turn execution “on”
        - Run operation on real data
      - No?
        - Leave execution “off”
        - Run on dummy data

Loops and branches

- start:
- 0x1000 mov ...
- 0x1004 mov ...
- 0x1008 mov ...
- 0x100c mov ...
- 0x1010 mov ...
- 0x1014 mov ...
- 0x1018 mov ...
- 0x101c mov ...
- 0x1020 mov ...
- 0x1024 mov ...
- 0x1028 mov ...
- 0x102c mov ...
- 0x1030 jmp start

- start:
- 0x1000 mov ...
- 0x1004 mov ...
- 0x1008 mov ...
- 0x100c mov ...
- 0x1010 mov ...
- 0x1014 mov ...
- 0x1018 mov ...
- 0x101c mov ...
- 0x1020 mov ... ← Implement a branch from here...
- 0x1024 mov ...
- 0x1028 mov ...
- 0x102c mov ...
- 0x1030 jmp start

- start:
- 0x1000 mov ...
- 0x1004 mov ...
- 0x1008 mov ...
- 0x100c mov ... ← ... to here
- 0x1010 mov ...
- 0x1014 mov ...
- 0x1018 mov ...
- 0x101c mov ...
- 0x1020 mov ... ← Implement a branch from here...
- 0x1024 mov ...
- 0x1028 mov ...
- 0x102c mov ...
- 0x1030 jmp start

```
start:
0x1000   mov ...
0x1004   mov ...
0x1008   mov ...
0x100c   mov ...  ← ... to here
0x1010   mov ...
0x1014   mov ...
0x1018   mov ...
0x101c   mov ...
0x1020   mov ...  ←
0x1024   mov ...
0x1028   mov ...
0x102c   mov ...
0x1030   jmp start
```

Store target
Switch to dummy data

0x100c

OFF

- start:
- 0x1000 mov ...
- 0x1004 mov ...
- 0x1008 mov ...
- 0x100c mov ... ← ... to here
- 0x1010 mov ...
- 0x1014 mov ...
- 0x1018 mov ...
- 0x101c mov ...
- 0x1020 mov ...
- 0x1024 mov ... ← Check if branch target
- 0x1028 mov ...
- 0x102c mov ...
- 0x1030 jmp start

0x100c

OFF

- start:
- 0x1000 mov ...
- 0x1004 mov ...
- 0x1008 mov ...
- 0x100c mov ... ← ... to here
- 0x1010 mov ...
- 0x1014 mov ...
- 0x1018 mov ...
- 0x101c mov ...
- 0x1020 mov ...
- 0x1024 mov ...
- 0x1028 mov ... ← Check if branch target
- 0x102c mov ...
- 0x1030 jmp start

0x100c

OFF

- start:
- 0x1000 mov ...
- 0x1004 mov ...
- 0x1008 mov ...
- 0x100c mov ... ← ... to here
- 0x1010 mov ...
- 0x1014 mov ...
- 0x1018 mov ...
- 0x101c mov ...
- 0x1020 mov ...
- 0x1024 mov ...
- 0x1028 mov ...
- 0x102c mov ... ← Check if branch target
- 0x1030 jmp start

0x100c

OFF

- start:
- 0x1000 mov ... ← Check if target
- 0x1004 mov ...
- 0x1008 mov ...
- 0x100c mov ... ← ... to here
- 0x1010 mov ...
- 0x1014 mov ...
- 0x1018 mov ...
- 0x101c mov ...
- 0x1020 mov ...
- 0x1024 mov ...
- 0x1028 mov ...
- 0x102c mov ...
- 0x1030 jmp start

0x100c

OFF

- start:
- 0x1000 mov ...
- 0x1004 mov ... ← Check if target
- 0x1008 mov ...
- 0x100c mov ... ← ... to here
- 0x1010 mov ...
- 0x1014 mov ...
- 0x1018 mov ...
- 0x101c mov ...
- 0x1020 mov ...
- 0x1024 mov ...
- 0x1028 mov ...
- 0x102c mov ...
- 0x1030 jmp start

0x100c

OFF

- start:
- 0x1000 mov ...
- 0x1004 mov ...
- 0x1008 mov ... ← Check if target
- 0x100c mov ... ← ... to here
- 0x1010 mov ...
- 0x1014 mov ...
- 0x1018 mov ...
- 0x101c mov ...
- 0x1020 mov ...
- 0x1024 mov ...
- 0x1028 mov ...
- 0x102c mov ...
- 0x1030 jmp start

0x100c

OFF

```
start:
0x1000   mov ...
0x1004   mov ...
0x1008   mov ...
0x100c   mov ... ←
0x1010   mov ...
0x1014   mov ...
0x1018   mov ...
0x101c   mov ...
0x1020   mov ...
0x1024   mov ...
0x1028   mov ...
0x102c   mov ...
0x1030   jmp start
```

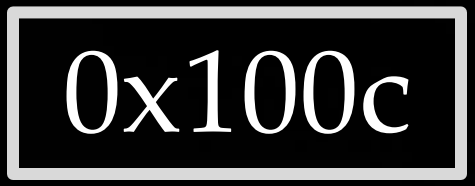
Target match
Switch to real data

0x100c

OFF

```
start:
0x1000   mov ...
0x1004   mov ...
0x1008   mov ...
0x100c   mov ...  ←
0x1010   mov ...
0x1014   mov ...
0x1018   mov ...
0x101c   mov ...
0x1020   mov ...
0x1024   mov ...
0x1028   mov ...
0x102c   mov ...
0x1030   jmp start
```

Target match
Switch to real data



Arithmetic

- Look up tables!
- We're already stuck with byte data from before, so this is pretty easy

```
unsigned char inc[]={
  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,
 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96,
 97, 98, 99,100,101,102,103,104,105,106,107,108,109,110,111,112,
113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,
129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,
145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,
161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,
177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,
193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,
209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,
225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,
241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,0
};
```

```
incb:
%assign y 1
%rep    256
        db y&0xff
        %assign y y+1
%endrep
```

```
; increment eax with mov
mov eax, [inc + eax]
```

Arithmetic

```
unsigned char dec[]={
255,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
 95, 96, 97, 98, 99,100,101,102,103,104,105,106,107,108,109,110,
111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,
127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,
143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,
159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,
175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,
191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,
207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,
223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,
239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254
};
```

```
decb:
%assign y 256-1
%rep    256
        db y&0xff
        %assign y y+1
%endrep
```

; decrement eax with mov
mov eax, [dec + eax]

Arithmetic

Logic

- Logic gates can similarly be implemented as lookup tables

```
unsigned char and[2][2]={ { 0, 0 }, {0, 1} };
unsigned char or[2][2]={ { 0, 1 }, {1, 1} };
unsigned char not[2]={ 1, 0 };

and[1][0]
or[0][1]
not[1]
```

Logic

```
o: dd o_0, o_1
o_0: dd 0, 4
o_1: dd 4, 4

%macro or 3
        mov eax, [%2]
        mov edx, [o+eax]
        mov eax, [%3]
        mov eax, [eax+edx]
        mov [%1], eax
%endmacro
```

```
a: dd a_0, a_1
a_0: dd 0, 0
a_1: dd 0, 4

%macro and 3
        mov eax, [%2]
        mov edx, [a+eax]
        mov eax, [%3]
        mov eax, [eax+edx]
        mov [%1], eax
%endmacro
```

```
n: dd 4, 0      ; not

%macro not 2
        mov eax, [%2]
        mov eax, [n+eax]
        mov [%1], eax
%endmacro
```

Halt

- Our program loops forever
- We need a way to stop it
- Dolan: a special invalid address
- Wait, that sounds familiar…
- NULL
- mov eax, [0]

```
nh: dd 0        ; halt
h: dd nh, 0

mov eax, [b]
mov eax, [h+eax]
mov eax, [eax]
```

Halt

```
eq  b, i, '+'
neq b, i, '+'
not b, off
and b, b1, b2
or  b, b1, b2
get eax, real, scratch, b
inc eax
dec eax
on  b
off b
```

Building Blocks

- With enough macros, this becomes almost doable …
  - … in assembly

Application

- A C compiler is a lofty goal
- Let’s start with something simpler

BrainF#$!

- A minimalistic esolang
- 8 instructions
- 2 registers
  - Instruction pointer
  - Data pointer
- We're going to call it BrainYuck

BrainF#$!

>	Increment the data pointer
<	Decrement the data pointer
+	Increment the byte at the data pointer
-	Decrement the byte at the data pointer
.	Output the byte at the data pointer
,	Read one byte of input, store it at the data pointer
[	If the byte at the data pointer is 0, jump forward to the matching]
]	If the byte at the data pointer is non-0, jump backward to the matching [

Halt

BrainYuck

- Print ‘1234’:

```
++++++++ ++++++++
++++++++ ++++++++
++++++++ ++++++++
+ . + . + . + . +
```

- Set the current data cell to 0:

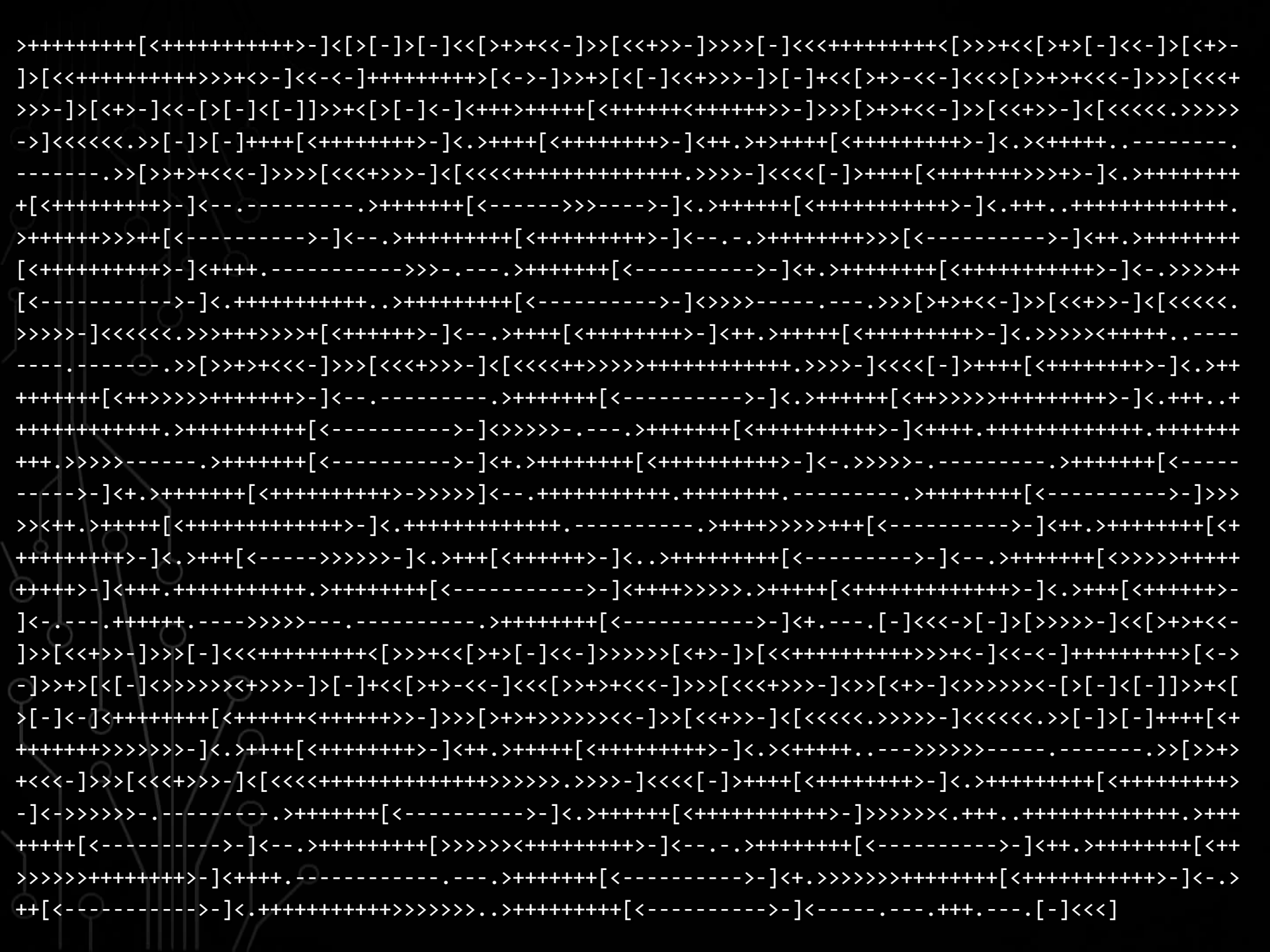
```
[ - ]
```

```
++++++++[>++++[>++>+++>+++>
+<<<<-]>+>+>->>+[<]<-]>>.>-
--.+++++++..+++.>>.<-.<.+++
.------.--------.>>+.>++.
```

Hello, world!

```
>++++++++++>+>+[[+++++[>++++++++<]
>.<++++++[>--------<-]+<<<]>.>>[[-
]<[>+<-]>>[<<+>+>-]<[>+<-[>+<-[>+<
-[>+<-[>+<-[>+<-[>+<-[>+<-[>+<-[>[
-]>+>+<<<-[>+<-]]]]]]]]]]+>>>]<<<]
```

Fibonacci Number Gen



- This is even worse than the movs!
  Why would you do this?!
- With our building blocks,
  BF ops are easy to implement with mov
- If I can get the code into BF,
  I can get it into movs
- A BASIC to BF compiler already exists

WHY!?

```
not     b, bs
and     b, b, bi
mov     eax, [b]
mov     ebx, [s_ms+eax]
mov     edx, [dp]
mov     eax, 0
mov     al, [ebx+edx]
mov     al, [incb+eax]
mov     [ebx+edx], al
```

```
not     b, bs
and     b, b, bd
mov     eax, [b]
mov     ebx, [s_ms+eax]
mov     edx, [dp]
mov     eax, 0
mov     al, [ebx+edx]
mov     al, [decb+eax]
mov     [ebx+edx], al
```

```
not     b, bs
and     b, b, bb
mov     eax, [b]
mov     ebx, [s_dp+eax]
mov     eax, [ebx]
mov     edx, 0
mov     dx, [decw+2*eax]
mov     [ebx], edx
```

<

```
not     b, bs
and     b, b, bf
mov     eax, [b]
mov     ebx, [s_dp+eax]
mov     eax, [ebx]
mov     edx, 0
mov     dx, [incw+2*eax]
mov     [ebx], edx
```

```
mov    eax, [bt]
mov    eax, [h+eax]
mov    eax, [eax]
```

#

```
not     b, bs
and     b, b, bw
mov     eax, [b]
mov     eax, [s_mz+eax]

mov     edx, [dp]
mov     al, [eax+edx]
mov     [c], al

mov     eax, 4
mov     ebx, 1
mov     ecx, c
mov     edx, 1
int     0x80
```

```
not     b, bs
and     b, b, br
mov     edx, [b]
mov     edx, [trim+edx]
mov     eax, 3
mov     ebx, 0
mov     ecx, c
int     0x80

mov     eax, [b]
mov     eax, [s_ms+eax]
mov     dl, [c]
mov     [eax], dl
```

,

```
and b, bo, bsf
mov eax, [b]
mov eax, [s_ns+eax]
mov edx, [eax]
mov dl, [incb+edx]
mov [eax], edx
```

```
and b, bo, bsb
mov eax, [b]
mov eax, [s_ns+eax]
mov edx, [eax]
mov dl, [decb+edx]
mov [eax], edx

mov [t], edx
eq b, t, 0
and b, b, bo
and b, b, bsb
mov eax, [b]
mov eax, [s_bsb+eax]
mov [eax], dword 0
```

```
mov eax, [dp]
mov edx, 0
mov dl, [m+eax]
mov [t], edx
eq t, t, 0
not b, bs
and b, b, t
and b, b, bo
mov eax, [b]
mov eax, [s_ns+eax]
mov [eax], dword 1
mov eax, [b]
mov eax, [s_bsf+eax]
mov [eax], dword 4
```

[

```
and b, bc, bsb
mov eax, [b]
mov eax, [s_ns+eax]
mov edx, [eax]
mov dl, [incb+edx]
mov [eax], edx
```

```
and b, bc, bsf
mov eax, [b]
mov eax, [s_ns+eax]
mov edx, [eax]
mov dl, [decb+edx]
mov [eax], edx

mov [t], edx
eq b, t, 0
and b, b, bc
and b, b, bsf
mov eax, [b]
mov eax, [s_bsf+eax]
mov [eax], dword 0
```

```
mov eax, [dp]
mov edx, 0
mov dl, [m+eax]
mov [t], edx
neq t, t, 0
not b, bs
and b, b, t
and b, b, bc
mov eax, [b]
mov eax, [s_ns+eax]
mov [eax], dword 1
mov eax, [b]
mov eax, [s_bsb+eax]
mov [eax], dword 4
```

]

- Compiler
- M/o/Vfuscate rot13
- objdump
- ./rot13

M/o/Vfuscator

movs

- We have two non-movs in our loop
- We can fix this by setting up the execution environment correctly

movs

- int 0x80
  - Solve with MMIO
  - mmap stdin/stdout into the process memory
  - Use mov for I/O

mov cs, ax

mov esp, [dsp]

- jmp
  - Set the loop to be its own SIGILL exception handler
  - Set the sa_nodefer flag
  - Replace the jump with an illegal mov
  - Reload the stack at each loop

```
sa: dd loop
    times 0x20 dd 0
    dd 0x40000000
    dd 0
```

movs

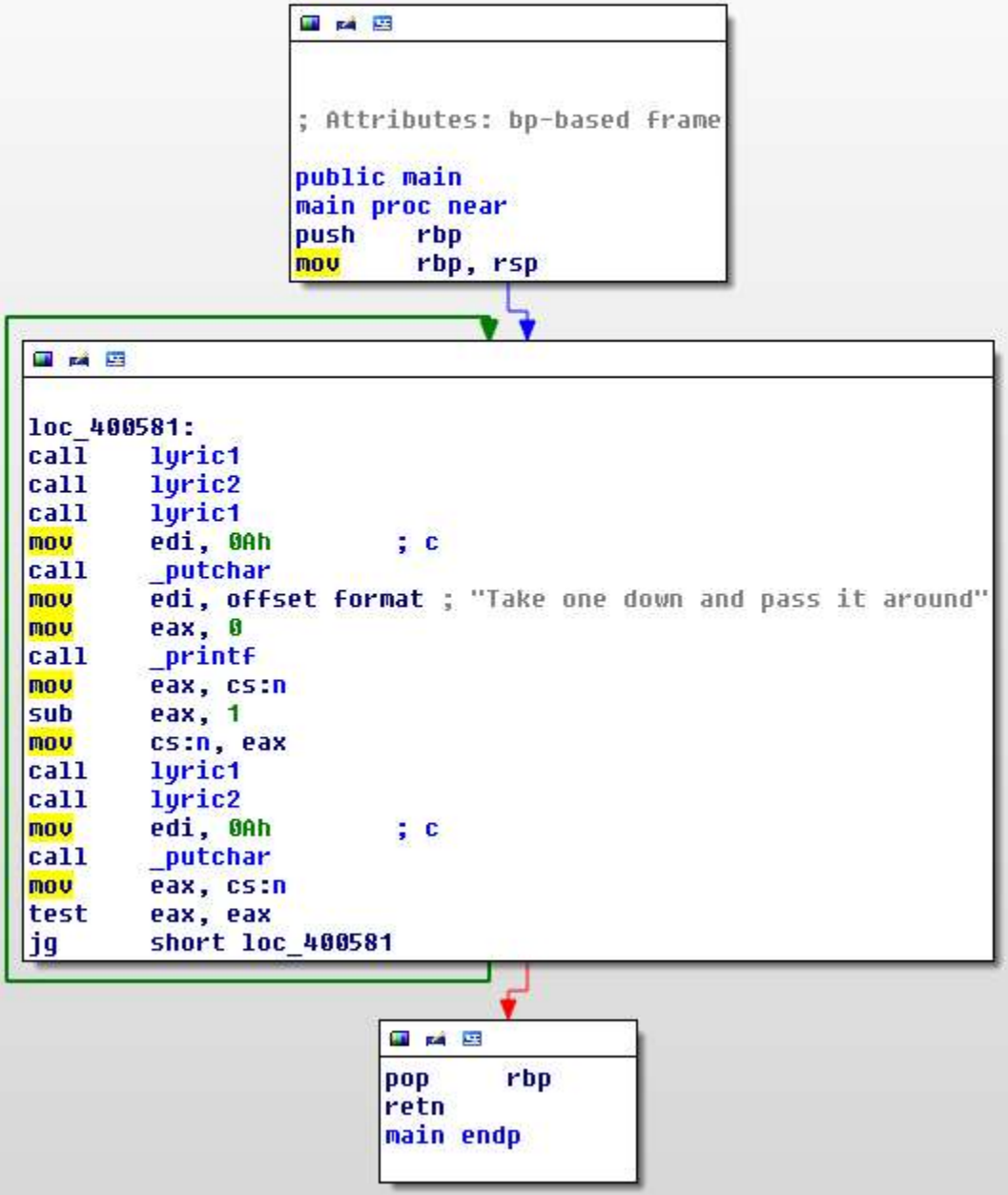
```
extern sigaction
mov dword [esp], 4
mov dword [esp+4], sa
mov dword [esp+8], 0
call sigaction
```

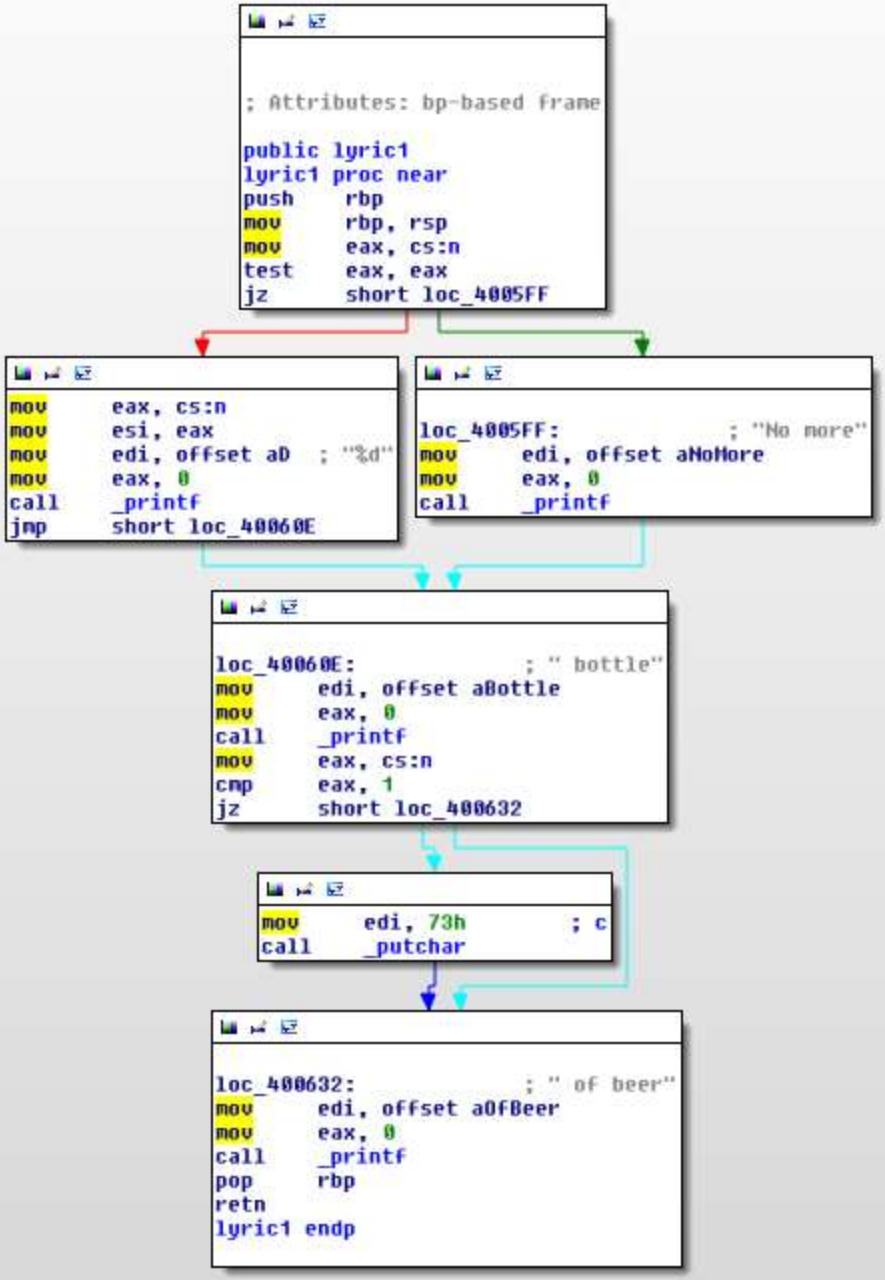
- Please, no more BF…
- HL → BF → MOV demo
- Speed?
- Anything!
  - factor 20460
  - prime
  - decss
  - Lost
  - M/o/Vfuscator

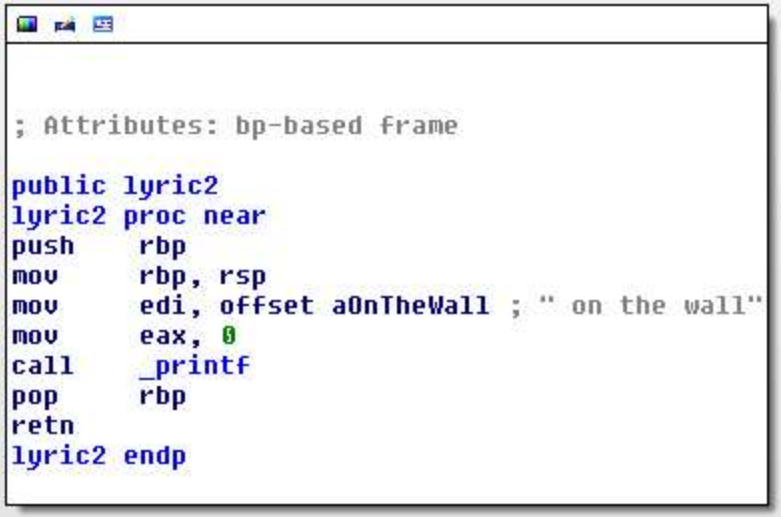
M/o/Vfuscator

- How would an experienced reverse engineer approach this?

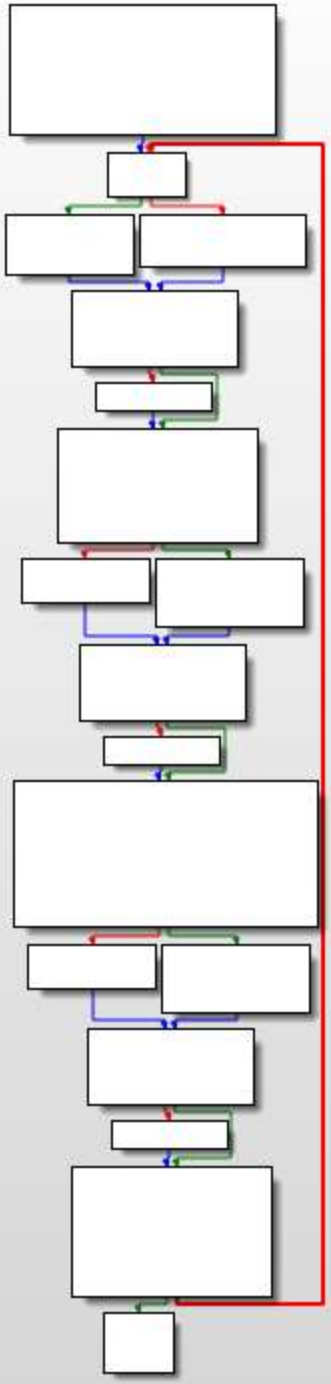

“nope”

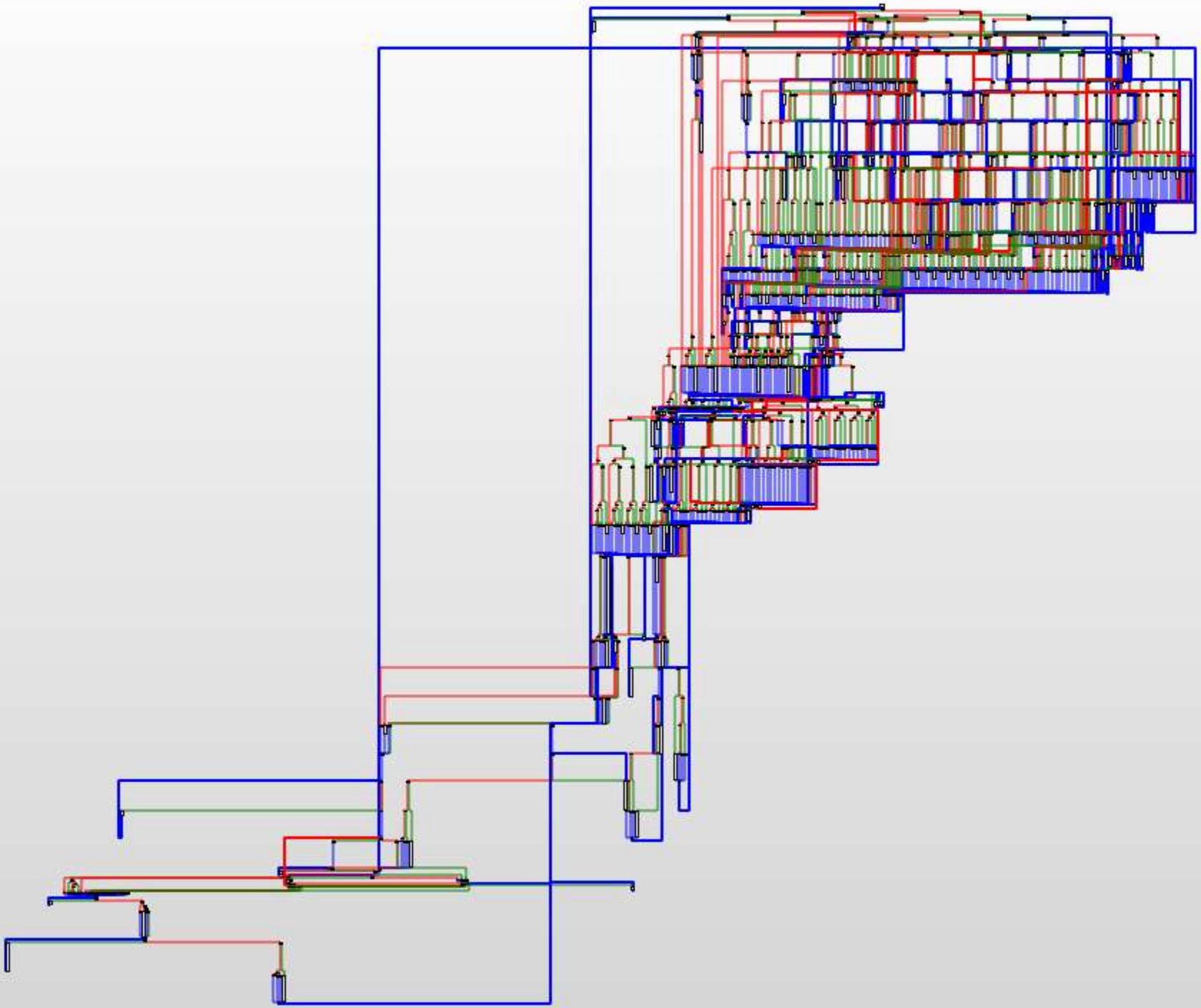

; Attributes: bp-based frame
public main
main proc near
push rbp
mov rbp, rsp
loc_400581:
call lyric1
call lyric2
call lyric1
mov edi, 0Ah ; c
call _putchar
mov edi, offset format ; "Take one down and pass it around"
mov eax, 0
call _printf
mov eax, cs:n
sub eax, 1
mov cs:n, eax
call lyric1
call lyric2
mov edi, 0Ah ; c
call _putchar
mov eax, cs:n
test eax, eax
jg short loc_400581
pop rbp
retn
main endp


; Attributes: bp-based frame
public lyric1
lyric1 proc near
push rbp
mov rbp, rsp
mov eax, cs:n
test eax, eax
jz short loc_4005FF
mov eax, cs:n
mov esi, eax
mov edi, offset aD ; "%d"
mov eax, 0
call _printf
jmp short loc_40060E
loc_4005FF: ; "No more"
mov edi, offset aNoMore
mov eax, 0
call _printf
loc_40060E: ; " bottle"
mov edi, offset aBottle
mov eax, 0
call _printf
mov eax, cs:n
cmp eax, 1
jz short loc_400632
mov edi, 73h ; c
call _putchar
loc_400632: ; " of beer"
mov edi, offset aOfBeer
mov eax, 0
call _printf
pop rbp
retn
lyric1 endp

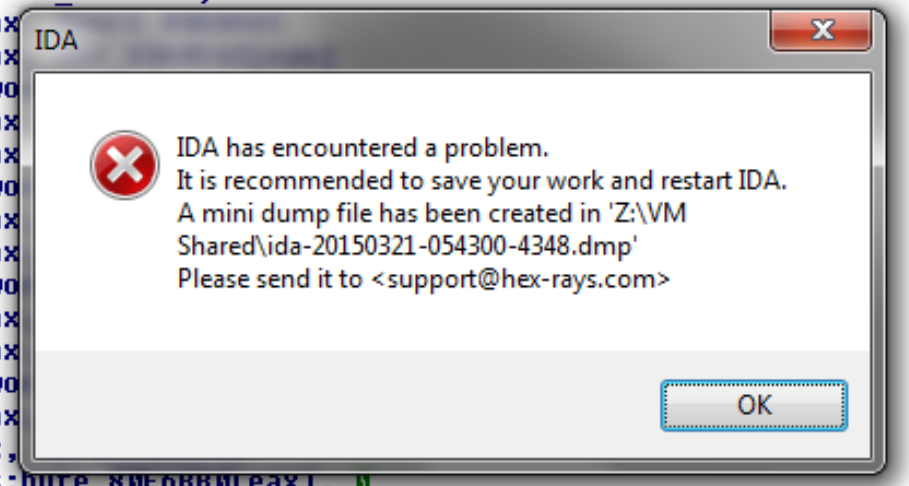


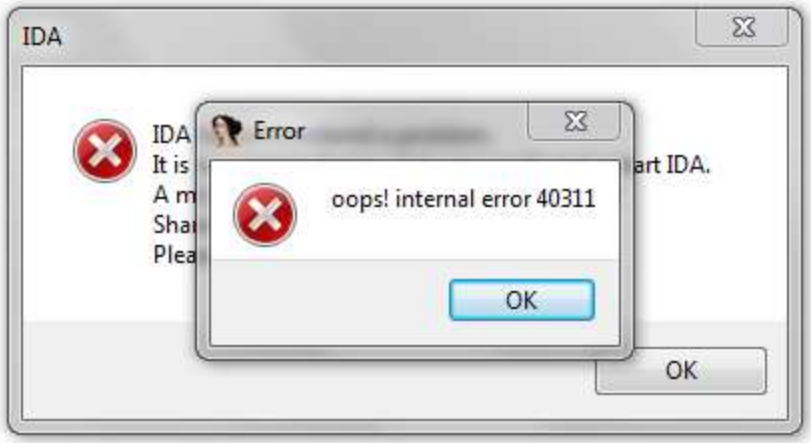
```
; Attributes: bp-based frame

public lyric2
lyric2 proc near
push    rbp
mov     rbp, rsp
mov     edi, offset aOnTheWall ; " on the wall"
mov     eax, 0
call    _printf
pop     rbp
retn
lyric2 endp
```

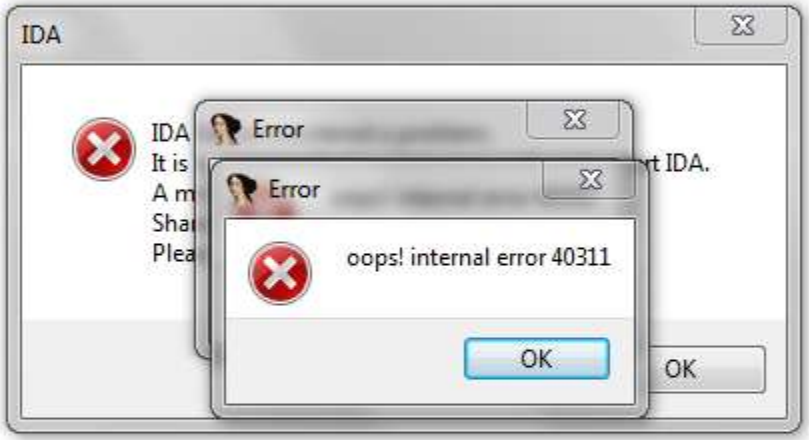




```
mov     dl, ds:byte_80E0BB0[eax]
mov     dword_80A0445, edx
mov     eax, 0
mov     ax, word ptr dword_80A0445
mov     ds:byte_80E6BB0[eax], 0
mov     ds:byte_80E6BB0, 4
mov     al, ds:byte_80E6BB0[eax]
mov     byte ptr dword_80A0445, al
mov     eax, dword_80A055A
mov     edx, off_80A0582[eax]
mov     eax, dword_80A0445
mov     eax, [eax+edx]
mov     dword_80A0441, eax
mov     eax
mov     eax
mov     dwo
mov     eax
mov     eax
mov     dwo
mov     eax
mov     eax
mov     dwo
mov     eax
mov     eax
mov     dwo
mov     eax
mov     ax,
mov     ds:byte_80E6BB0[eax], 0
mov     ds:byte_80E6BB1, 4
mov     al, ds:byte_80E6BB0[eax]
mov     byte ptr dword_80A0441, al
mov     eax, dword_80A0441
mov     edx, off_80A0582[eax]
mov     eax, dword_80A0445
mov     eax, [eax+edx]
mov     dword_80A0441, eax
mov     eax, dword_80A054A
mov     eax, dword_80A059A[eax]
mov     dword_80A0445, eax
mov     eax, dword_80A0441
mov     edx, off_80A0582[eax]
mov     eax, dword_80A0445
mov     eax, [eax+edx]
mov     dword_80A0441, eax
```

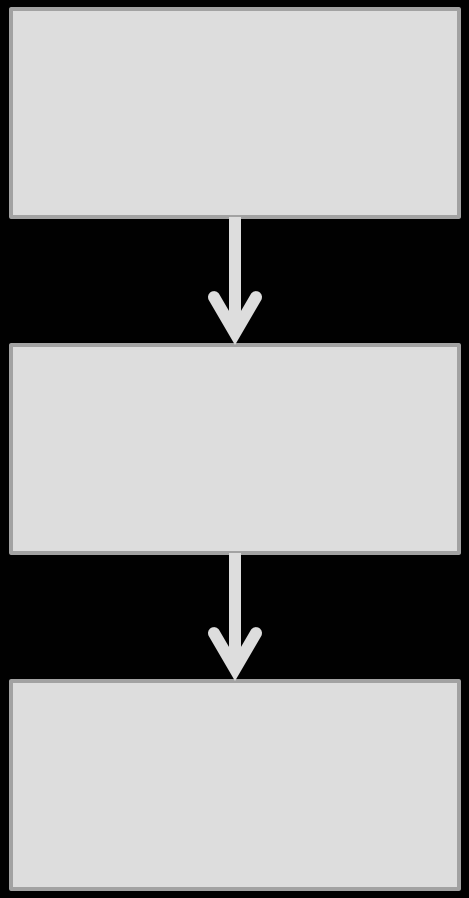



IDA
IDA
It is
A m
Shar
Plea
art IDA.
OK
Error
oops! internal error 40311
OK

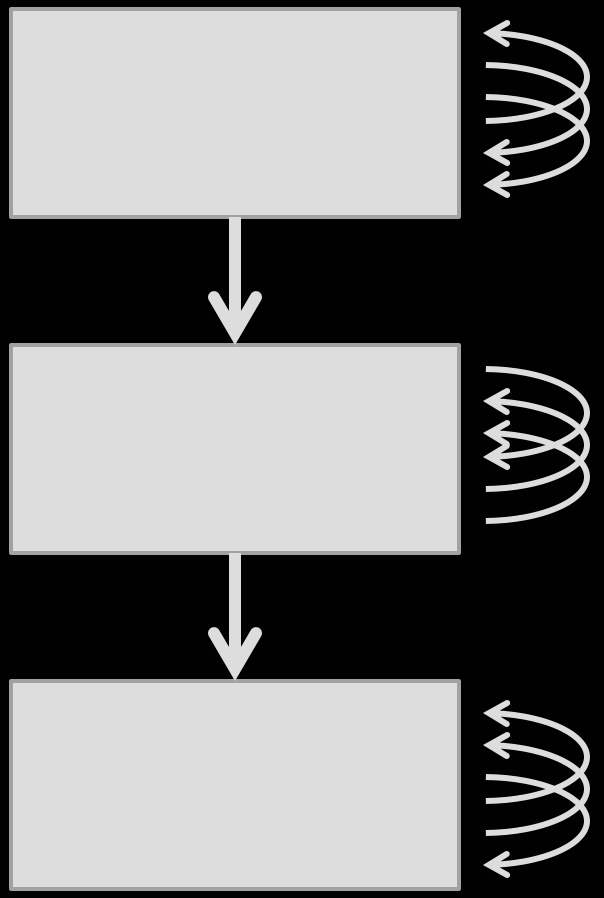

IDA
IDA
It is
A m
Shar
Plea
rt IDA.
OK
Error
Error
oops! internal error 40311
OK

Anti RE

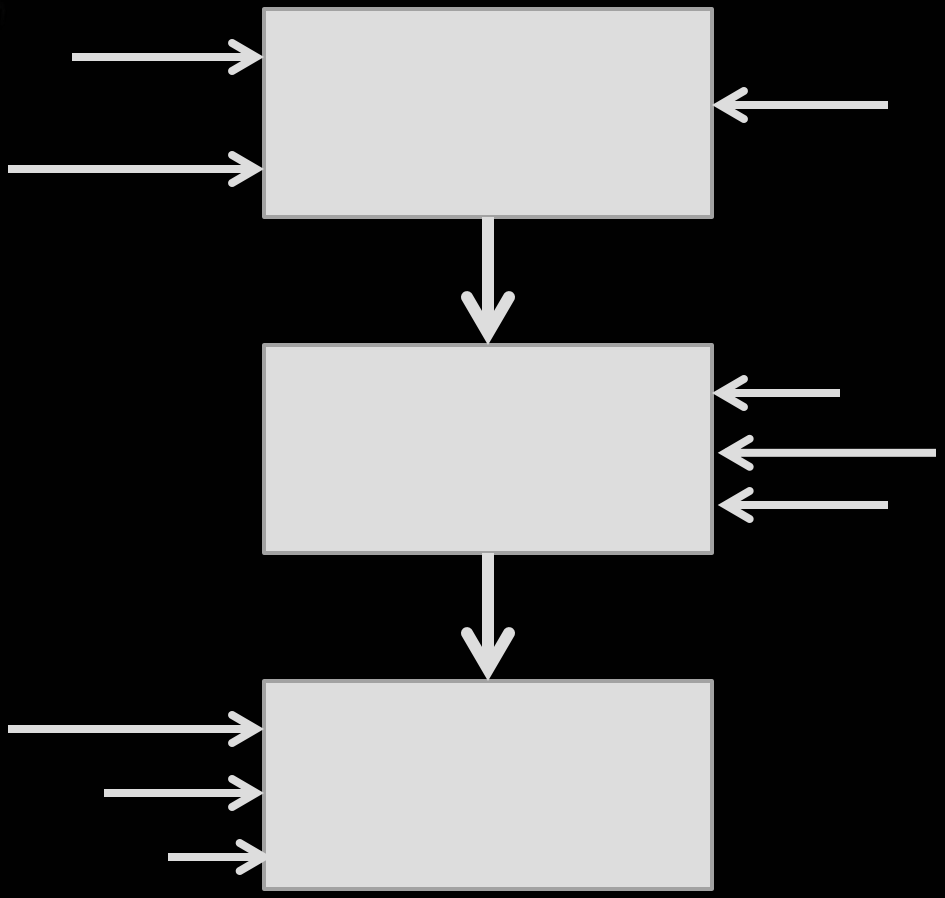
- Opposite of other obfuscators
- Every program becomes a line
- There is no dead code
  - Only code that sometimes works on fake data
  - And sometimes real data



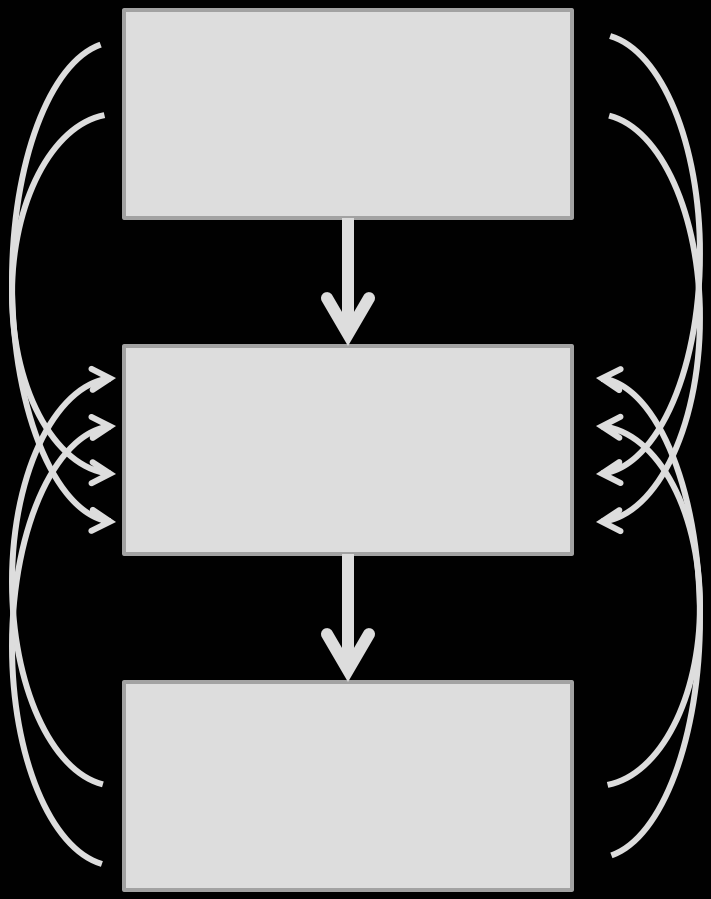
Anti RE



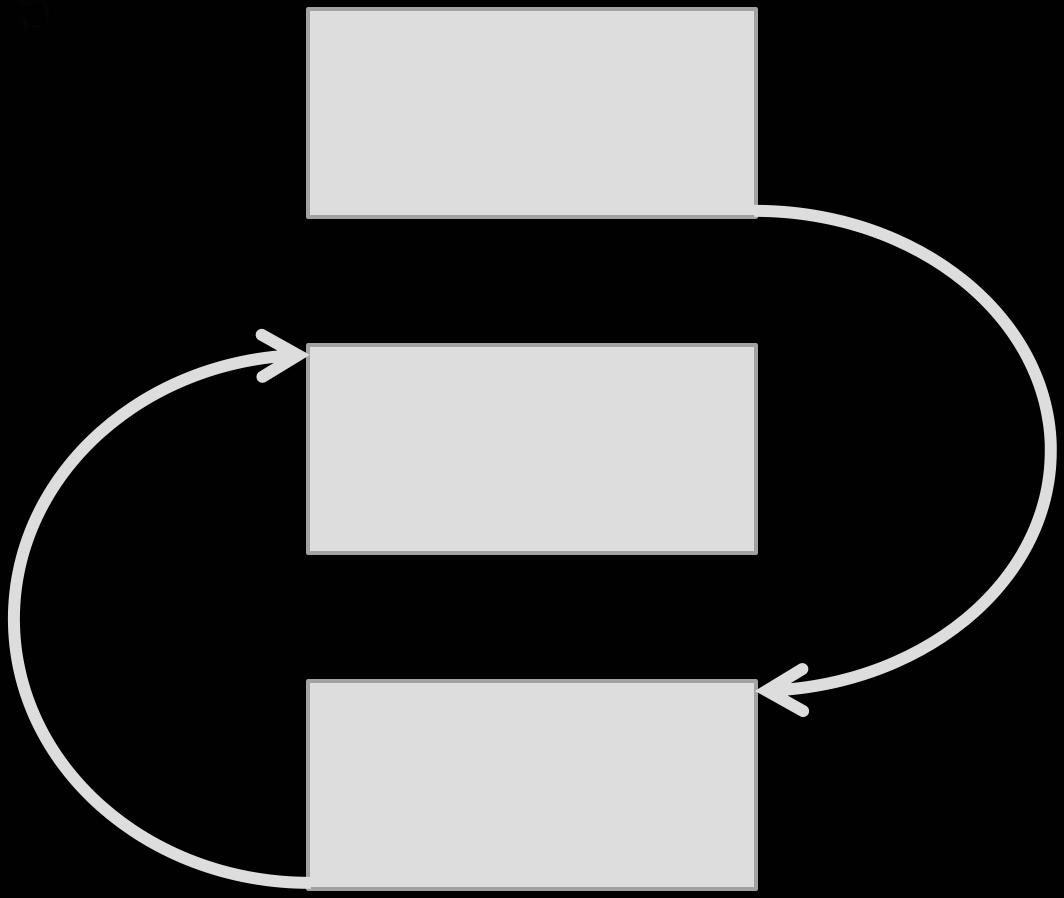
Anti RE: Permutations



Anti RE: Junk Instructions



Anti RE: Interleaving



Anti RE: Rearranging

Anti RE

- Repermute the entire program every compile
- Especially effective in the mov world

```
mov   edx,DWORD PTR ds:0x809f77d
mov   DWORD PTR [eax],0x4
mov   BYTE PTR [eax+edx*1],cl
mov   eax,DWORD PTR [eax+0x805f0f1]
mov   eax,ds:0x805f109
mov   DWORD PTR [eax],0x0
mov   dl,BYTE PTR [eax+0x80ff850]
mov   ax,WORD PTR [edx+edx*1+0x805f75d]
mov   DWORD PTR [eax],0x1
mov   eax,ds:0x805f109
mov   eax,ds:0x809f75d
mov   BYTE PTR [eax+0x80ff850],0x0
mov   dl,BYTE PTR [eax+0x809f850]
mov   cl,BYTE PTR [eax+edx*1]
mov   eax,ds:0x805f0d4
mov   eax,ds:0x805f0d4
mov   DWORD PTR ds:0x805f0d4,edx
mov   ecx,0x0
mov   edx,0x0
mov   eax,0x0
mov   eax,ds:0x805f109
mov   dl,BYTE PTR [eax+0x80ff850]
mov   eax,DWORD PTR [eax+0x805f101]
mov   BYTE PTR [eax+0x80ff850],0x0
mov   DWORD PTR [edx],eax
mov   edx,DWORD PTR ds:0x809f77d
mov   eax,DWORD PTR [eax+0x805f0c0]
mov   ds:0x805f0d4,eax
mov   ds:0x805f0d0,eax
mov   edx,DWORD PTR ds:0x805f109
mov   edx,DWORD PTR ds:0x805f0d9
mov   eax,DWORD PTR [eax+0x805f149]
mov   eax,ds:0x805f0d9
mov   eax,ds:0x805f0d0
mov   edx,0x0
mov   eax,ds:0x805f0d0
mov   ax,ds:0x805f0e9
mov   edx,DWORD PTR [eax+0x805f131]
mov   edx,0x0
mov   eax,ds:0x805f0d0
mov   edx,DWORD PTR [edx+0x809f785]
mov   DWORD PTR ds:0x805f0f9,0x0
mov   DWORD PTR ds:0x805f0d0,edx
mov   eax,0x0
mov   eax,0x0
mov   BYTE PTR ds:0x80ff850,0x4
```

```
mov   DWORD PTR [eax],0x0
mov   DWORD PTR [eax],0x1
mov   eax,ds:0x805f109
mov   DWORD PTR ds:0x805f0d4,edx
mov   eax,0x0
mov   eax,ds:0x805f109
mov   dl,BYTE PTR [eax+0x809f850]
mov   eax,0x0
mov   DWORD PTR [eax],0x4
mov   eax,DWORD PTR [eax+0x805f111]
mov   edx,DWORD PTR ds:0x805f0d9
mov   eax,ds:0x805f109
mov   BYTE PTR [eax+0x80ff850],0x0
mov   BYTE PTR ds:0x80ff850,0x4
mov   edx,DWORD PTR [eax+0x805f131]
mov   eax,DWORD PTR [eax+0x805f0c0]
mov   eax,ds:0x805f0d4
mov   DWORD PTR ds:0x805f0d0,edx
mov   eax,ds:0x805f0d9
mov   ax,ds:0x805f0d4
mov   ax,ds:0x805f0e9
mov   eax,DWORD PTR [eax+edx*1]
mov   edx,0x0
mov   dl,BYTE PTR [eax+0x80ff850]
mov   BYTE PTR [eax+0x80ff850],0x0
mov   ds:0x805f0d0,eax
mov   BYTE PTR ds:0x80ff851,0x4
mov   edx,DWORD PTR ds:0x805f109
mov   DWORD PTR ds:0x805f0d4,edx
mov   edx,DWORD PTR ds:0x809f77d
mov   eax,0x0
mov   DWORD PTR ds:0x805f0f9,0x0
mov   BYTE PTR [eax+edx*1],cl
mov   eax,DWORD PTR [eax+0x805f0f1]
mov   eax,ds:0x805f0d4
mov   edx,0x0
mov   edx,DWORD PTR [eax+0x805f131]
mov   edx,DWORD PTR ds:0x809f77d
mov   eax,DWORD PTR [eax+0x805f149]
mov   ds:0x805f0d4,eax
mov   edx,0x0
mov   cl,BYTE PTR [eax+edx*1]
mov   edx,DWORD PTR [edx+0x809f785]
mov   esp,DWORD PTR ds:0x809f84c
mov   eax,ds:0x805f0d0
mov   eax,ds:0x805f0d0
```

```
mov   DWORD PTR ds:0x805f0d4,edx
mov   eax,0x0
mov   eax,DWORD PTR [eax+0x805f0f1]
mov   eax,ds:0x805f109
mov   DWORD PTR ds:0x805f0d0,edx
mov   eax,ds:0x805f0d0
mov   edx,DWORD PTR ds:0x805f0d9
mov   eax,ds:0x805f0d9
mov   eax,0x0
mov   esp,DWORD PTR ds:0x809f84c
mov   ecx,0x0
mov   eax,DWORD PTR [eax+0x805f0c0]
mov   edx,0x0
mov   DWORD PTR [edx],eax
mov   edx,0x0
mov   edx,0x0
mov   dl,BYTE PTR [eax+0x80ff850]
mov   BYTE PTR [eax+0x80ff850],0x0
mov   eax,ds:0x809f75d
mov   eax,DWORD PTR [eax+edx*1]
mov   edx,DWORD PTR ds:0x809f77d
mov   ax,WORD PTR [edx+edx*1+0x805f75d]
mov   eax,0x0
mov   BYTE PTR ds:0x80ff850,0x4
mov   eax,ds:0x805f109
mov   eax,ds:0x805f0d0
mov   edx,DWORD PTR [eax+0x805f131]
mov   ax,ds:0x805f0e9
mov   BYTE PTR [eax+0x80ff850],0x0
mov   edx,DWORD PTR ds:0x809f77d
mov   BYTE PTR [eax+edx*1],cl
mov   cl,BYTE PTR [eax+edx*1]
mov   eax,ds:0x805f0d4
mov   BYTE PTR ds:0x80ff851,0x4
mov   eax,DWORD PTR [eax+0x805f101]
mov   edx,DWORD PTR [eax+0x805f131]
mov   dl,BYTE PTR [eax+0x80ff850]
mov   edx,DWORD PTR [edx+0x809f785]
mov   ds:0x805f0d4,eax
mov   DWORD PTR [eax],0x1
mov   eax,ds:0x805f109
mov   eax,DWORD PTR [eax+0x805f149]
mov   dl,BYTE PTR [eax+0x809f850]
mov   edx,DWORD PTR ds:0x805f109
mov   DWORD PTR [eax],0x0
mov   DWORD PTR ds:0x805f0f9,0x0
```

Anti RE

- But even without this…

Limitations

- Speed
  - Jumping switches between dummy and real data
  - Jumping forwards …
  - Jumping backwards …
  - [-]
  - REALLY slow
- Any beer left…?

- I've gone this far...
  - ... is a C compiler that much more ...?

ALU

- Can’t use lookup tables for 32 bit arithmetic…

```
%macro add32 4
    mov eax, 0
    mov ebx, 0
    mov ecx, 0
    mov edx, 0
    mov dword [%4], 0
    add8 %1+0, %2+0, %3+0, %4
    add8 %1+1, %2+1, %3+1, %4
    add8 %1+2, %2+2, %3+2, %4
    add8 %1+3, %2+3, %3+3, %4
%endmacro
```

```
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov dword [dword 0x8049576],0x0
mov al,[0x804956a]
mov bl,[dword 0x804956e]
mov cl,[dword 0x8049576]
mov dl,[eax+ecx+0x8049168]
mov al,[ebx+edx+0x8049168]
mov [0x8049572],al
mov al,[ebx+edx+0x8049369]
mov [0x8049576],al
mov al,[0x804956b]
mov bl,[dword 0x804956f]
mov cl,[dword 0x8049576]
mov dl,[eax+ecx+0x8049168]
mov al,[ebx+edx+0x8049168]
mov [0x8049573],al
mov al,[ebx+edx+0x8049369]
mov [0x8049576],al
mov al,[0x804956c]
mov bl,[dword 0x8049570]
mov cl,[dword 0x8049576]
mov dl,[eax+ecx+0x8049168]
mov al,[ebx+edx+0x8049168]
mov [0x8049574],al
mov al,[ebx+edx+0x8049369]
mov [0x8049576],al
mov al,[0x804956d]
mov bl,[dword 0x8049571]
mov cl,[dword 0x8049576]
mov dl,[eax+ecx+0x8049168]
mov al,[ebx+edx+0x8049168]
mov [0x8049575],al
mov al,[ebx+edx+0x8049369]
mov [0x8049576],al
mov eax,[0x8049572]
```

```
%macro sub32 4
    mov dword [%4], 1
    inv8 %3+0
    inv8 %3+1
    inv8 %3+2
    inv8 %3+3
    add8 %1+0, %2+0, %3+0, %4
    add8 %1+1, %2+1, %3+1, %4
    add8 %1+2, %2+2, %3+2, %4
    add8 %1+3, %2+3, %3+3, %4
%endmacro
```

```
mov eax,0x0
mov al,[0x80496fe]
mov al,[eax+0x80491f8]
mov [0x80496fe],al
mov eax,0x0
mov al,[0x80496ff]
mov al,[eax+0x80491f8]
mov [0x80496ff],al
mov eax,0x0
mov al,[0x8049700]
mov al,[eax+0x80491f8]
mov [0x8049700],al
mov eax,0x0
mov al,[0x8049701]
mov al,[eax+0x80491f8]
mov [0x8049701],al
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x80496fa]
mov bl,[dword 0x80496fe]
mov cl,[dword 0x8049706]
mov dl,[eax+ecx+0x80492f8]
mov al,[ebx+edx+0x80492f8]
mov [0x8049702],al
mov al,[ebx+edx+0x80494f9]
mov [0x8049706],al
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x80496fb]
mov bl,[dword 0x80496ff]
mov cl,[dword 0x8049706]
mov dl,[eax+ecx+0x80492f8]
mov al,[ebx+edx+0x80492f8]
mov [0x8049703],al
mov al,[ebx+edx+0x80494f9]
mov [0x8049706],al
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x80496fc]
mov bl,[dword 0x8049700]
mov cl,[dword 0x8049706]
mov dl,[eax+ecx+0x80492f8]
mov al,[ebx+edx+0x80492f8]
mov [0x8049704],al
mov al,[ebx+edx+0x80494f9]
mov [0x8049706],al
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x80496fd]
mov bl,[dword 0x8049701]
mov cl,[dword 0x8049706]
mov dl,[eax+ecx+0x80492f8]
mov al,[ebx+edx+0x80492f8]
mov [0x8049705],al
mov al,[ebx+edx+0x80494f9]
mov [0x8049706],al
mov eax,[0x8049702]
```

```
%macro shl1_8_c 2
    mov eax, 0
    mov edx, 0
    mov al, [%1]
    mov dl, [%2]
    mov eax, [shl3_8_d+eax*4]
    mov eax, [shl1_8_c_d+edx*4+eax]
    mov [%1], al
    mov [%2], ah
%endmacro

%macro shl32 3
    %rep %2
        shl1_8_c %1+0, %3
        shl1_8_c %1+1, %3
        shl1_8_c %1+2, %3
        shl1_8_c %1+3, %3
    %endrep
%endmacro
```

```
mov eax,0x0
mov edx,0x0
mov al,[0x8049e04]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e04],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e05]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e05],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e06]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e06],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e07]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e07],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e04]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e04],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e05]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e05],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e06]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e06],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e07]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e07],al
mov [dword 0x8049e08],ah
mov eax,[0x8049e04]
```

```
%macro shr1_8_c 2
    mov eax, 0
    mov edx, 0
    mov al, [%1]
    mov dl, [%2]
    mov eax, [shl3_8_d+eax*4]
    mov eax, [shr1_8_c_d+edx*4+eax]
    mov [%1], al
    mov [%2], ah
%endmacro

%macro shr32 3
    %rep %2
        shr1_8_c %1+3, %3
        shr1_8_c %1+2, %3
        shr1_8_c %1+1, %3
        shr1_8_c %1+0, %3
    %endrep
%endmacro
```

```
mov eax,0x0
mov edx,0x0
mov al,[0x8049e07]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e07],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e06]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e06],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e05]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e05],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e04]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e04],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e07]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e07],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e06]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e06],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e05]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e05],al
mov [dword 0x8049e08],ah
mov eax,0x0
mov edx,0x0
mov al,[0x8049e04]
mov dl,[dword 0x8049e08]
mov eax,[eax*4+0x8049a04]
mov eax,[eax+edx*4+0x8049204]
mov [0x8049e04],al
mov [dword 0x8049e08],ah
mov eax,[0x8049e04]
```

```
%macro mul8 4
    mov eax, 0
    mov ebx, 0
    mov ecx, 0
    mov edx, 0
    mov al, [%2]
    mov dl, [%3]
    mov ebx, [mm+4*eax]
    mov cl, [ebx+edx]
    mov ebx, [mc+4*eax]
    mov al, [ebx+edx]
    mov dl, [%4]
    mov ebx, 0
    mov bl, [su+ecx+edx]
    mov [%1], bl
    mov bl, [sc+ecx+edx]
    mov bl, [su+ebx+eax]
    mov [%4], bl
%endmacro
```

```
%macro mul32 4
    mov dword [%4], 0
    mul8 z0+0, %2+0, %3+0, %4
    mul8 z0+1, %2+1, %3+0, %4
    mul8 z0+2, %2+2, %3+0, %4
    mul8 z0+3, %2+3, %3+0, %4

    mov dword [%4], 0
    mul8 z1+1, %2+0, %3+1, %4
    mul8 z1+2, %2+1, %3+1, %4
    mul8 z1+3, %2+2, %3+1, %4

    mov dword [%4], 0
    mul8 z2+2, %2+0, %3+2, %4
    mul8 z2+3, %2+1, %3+2, %4

    mov dword [%4], 0
    mul8 z3+3, %2+0, %3+3, %4

    mov dword [%4], 0
    add8n %1+0, %4, z0+0, %4
    add8n %1+1, %4, z0+1, z1+1, %4
    add8n %1+2, %4, z0+2, z1+2, z2+2, %4
    add8n %1+3, %4, z0+3, z1+3, z2+3, z3+3, %4
%endmacro
```

```
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea10]
mov dl,[dword 0x806ea14]
mov ebx,[eax*4+0x804e200]
mov cl,[ebx+edx]
mov ebx,[eax*4+0x805e600]
mov al,[ebx+edx]
mov dl,[dword 0x806ea1c]
mov ebx,0x0
mov bl,[ecx+edx+0x8049000]
mov [dword 0x806ea00],bl
mov bl,[ecx+edx+0x8049300]
mov bl,[ebx+eax+0x8049000]
mov [dword 0x806ea1c],bl
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea11]
mov dl,[dword 0x806ea14]
mov ebx,[eax*4+0x804e200]
mov cl,[ebx+edx]
mov ebx,[eax*4+0x805e600]
mov al,[ebx+edx]
mov dl,[dword 0x806ea1c]
mov ebx,0x0
mov bl,[ecx+edx+0x8049000]
mov [dword 0x806ea01],bl
mov bl,[ecx+edx+0x8049300]
mov bl,[ebx+eax+0x8049000]
mov [dword 0x806ea1c],bl
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea12]
mov dl,[dword 0x806ea14]
mov ebx,[eax*4+0x804e200]
mov cl,[ebx+edx]
mov ebx,[eax*4+0x805e600]
mov al,[ebx+edx]
mov dl,[dword 0x806ea1c]
mov ebx,0x0
mov bl,[ecx+edx+0x8049000]
mov [dword 0x806ea02],bl
mov bl,[ecx+edx+0x8049300]
mov bl,[ebx+eax+0x8049000]
mov [dword 0x806ea1c],bl
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea13]
mov dl,[dword 0x806ea14]
mov ebx,[eax*4+0x804e200]
mov cl,[ebx+edx]
mov ebx,[eax*4+0x805e600]
mov al,[ebx+edx]
mov dl,[dword 0x806ea1c]
mov ebx,0x0
mov bl,[ecx+edx+0x8049000]
mov [dword 0x806ea03],bl
mov bl,[ecx+edx+0x8049300]
mov bl,[ebx+eax+0x8049000]
mov [dword 0x806ea1c],bl
mov dword [dword 0x806ea1c],0x0
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea10]
mov dl,[dword 0x806ea15]
mov ebx,[eax*4+0x804e200]
mov cl,[ebx+edx]
mov ebx,[eax*4+0x805e600]
mov al,[ebx+edx]
mov dl,[dword 0x806ea1c]
mov ebx,0x0
mov bl,[ecx+edx+0x8049000]
mov [dword 0x806ea05],bl
mov bl,[ecx+edx+0x8049300]
mov bl,[ebx+eax+0x8049000]
mov [dword 0x806ea1c],bl
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea11]
mov dl,[dword 0x806ea15]
mov ebx,[eax*4+0x804e200]
mov cl,[ebx+edx]
mov ebx,[eax*4+0x805e600]
mov al,[ebx+edx]
mov dl,[dword 0x806ea1c]
mov ebx,0x0
mov bl,[ecx+edx+0x8049000]
mov [dword 0x806ea06],bl
mov bl,[ecx+edx+0x8049300]
mov bl,[ebx+eax+0x8049000]
mov [dword 0x806ea1c],bl
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea12]
mov dl,[dword 0x806ea15]
mov ebx,[eax*4+0x804e200]
mov cl,[ebx+edx]
mov ebx,[eax*4+0x805e600]
mov al,[ebx+edx]
mov dl,[dword 0x806ea1c]
mov ebx,0x0
mov bl,[ecx+edx+0x8049000]
mov [dword 0x806ea07],bl
mov bl,[ecx+edx+0x8049300]
mov bl,[ebx+eax+0x8049000]
mov [dword 0x806ea1c],bl
mov dword [dword 0x806ea1c],0x0
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea10]
mov dl,[dword 0x806ea16]
mov ebx,[eax*4+0x804e200]
mov cl,[ebx+edx]
mov ebx,[eax*4+0x805e600]
mov al,[ebx+edx]
mov dl,[dword 0x806ea1c]
mov ebx,0x0
mov bl,[ecx+edx+0x8049000]
mov [dword 0x806ea0a],bl
mov bl,[ecx+edx+0x8049300]
mov bl,[ebx+eax+0x8049000]
mov [dword 0x806ea1c],bl
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea11]
mov dl,[dword 0x806ea16]
mov ebx,[eax*4+0x804e200]
mov cl,[ebx+edx]
mov ebx,[eax*4+0x805e600]
mov al,[ebx+edx]
mov dl,[dword 0x806ea1c]
mov ebx,0x0
mov bl,[ecx+edx+0x8049000]
mov [dword 0x806ea0b],bl
mov bl,[ecx+edx+0x8049300]
mov bl,[ebx+eax+0x8049000]
mov [dword 0x806ea1c],bl
mov dword [dword 0x806ea1c],0x0
mov eax,0x0
mov ebx,0x0
mov ecx,0x0
mov edx,0x0
mov al,[0x806ea01]
mov dl,[dword 0x806ea05]
mov eax,[eax*4+0x8049600]
mov edx,[edx*4+0x8049600]
mov edx,[eax+edx+0x804a200]
mov eax,0x0
mov al,[0x806ea1c]
mov edx,[edx*4+0x8049600]
mov eax,[eax*4+0x8049600]
mov edx,[eax+edx+0x804a200]
mov [dword 0x806ea19],dl
mov [dword 0x806ea1c],dh
mov eax,0x0
mov ebx,0x0
mov edx,0x0
mov al,[0x806ea02]
mov dl,[dword 0x806ea06]
mov eax,[eax*4+0x8049600]
mov edx,[edx*4+0x8049600]
mov edx,[eax+edx+0x804a200]
mov eax,0x0
mov al,[0x806ea0a]
mov edx,[edx*4+0x8049600]
mov eax,[eax*4+0x8049600]
mov edx,[eax+edx+0x804a200]
mov eax,0x0
mov al,[0x806ea1c]
mov edx,[edx*4+0x8049600]
mov eax,[eax*4+0x8049600]
mov edx,[eax+edx+0x804a200]
mov [dword 0x806ea1a],dl
mov [dword 0x806ea1c],dh
mov eax,0x0
mov ebx,0x0
mov edx,0x0
mov al,[0x806ea03]
mov dl,[dword 0x806ea07]
mov eax,[eax*4+0x8049600]
mov edx,[edx*4+0x8049600]
mov edx,[eax+edx+0x804a200]
mov eax,0x0
mov al,[0x806ea0b]
mov edx,[edx*4+0x8049600]
mov eax,[eax*4+0x8049600]
mov edx,[eax+edx+0x804a200]
mov eax,0x0
mov al,[0x806ea0f]
mov edx,[edx*4+0x8049600]
mov eax,[eax*4+0x8049600]
mov edx,[eax+edx+0x804a200]
mov eax,0x0
mov al,[0x806ea1c]
mov edx,[edx*4+0x8049600]
mov eax,[eax*4+0x8049600]
mov edx,[eax+edx+0x804a200]
mov [dword 0x806ea1b],dl
mov [dword 0x806ea1c],dh
mov eax,[0x806ea18]
```

```
mov dword [q], 0
mov dword [r], 0

%assign bb 31
%rep 32

    bit c, n, bb
    shl32 r, c

    mov dword [c], 0
    gte32 t, r, d, c

    mov eax, [t]
    mov edx, [sel_r+4*eax]
    mov [psr], edx
    mov edx, [sel_q+4*eax]
    mov [psq], edx

    mov eax, [psr]
    mov eax, [eax]
    mov [dummy_r], eax
    sub32 dummy_r, dummy_r, d, c
    mov eax, [psr]
    mov edx, [dummy_r]
    mov [eax], edx

    mov eax, [psq]
    mov eax, [eax]
    mov [dummy_q], eax
    set32 dummy_q, bb
    mov eax, [psq]
    mov edx, [dummy_q]
    mov [eax], edx

    %assign bb bb-1
%endrep

mov eax, [q]
```

- lcc
- esi/edi
- Calling convention
- Emulated stack
- 32 bit ALU
- Simplified dummy selectors
- 102 IL instructions
- Not bad!

M/o/Vfuscator 2.0

- Nibbles!
  - Source / Compile / Disas / Play

M/o/Vfuscator 2.0

- Cube!
  - Complete IEEE floating point emulator
    - ~500K movs

M/o/Vfuscator 2.0

- Calculate e, pi, sqrt, sin, cos
- AES, DES, MD5, SHA256
- Bitcoin addresses
- Sudoku, minesweeper, tic tac toe
- Mandelbrot, Julia set, Bifurcation
- Solve a maze
- Game of life
- Ray tracing engine

- With only movs ☺

M/o/Vfuscator 2.0

- First?

M/o/Vfuscator 2.0

- What other Turing-complete instructions exist…?

M/o/Vfuscator 2.0

XOR?

- Example: mov eax, 1234
  - xor eax, eax
  - xor eax, 1234

- Compile C to only XOR instructions

XORfuscator

- XORfuscator
  - Compile C to only XOR instructions
- AND/ORfuscator
  - … only AND and OR instructions
- SUBfuscator
  - … only SUB instructions
- CMPXCHG/XCHGfuscator
  - … only CMPXCHG / XCHG instructions
- RRRRRfuscator
  - … only 1-bit shifts

{.*}fuscator family

- x86 MMU is Turing-complete
  - (HT @julianbangert @sergeybratus)
- M/o/Vfuscator3: A no instruction C compiler?

M/o/Vfuscator 3.0?

Wrap up

- Maybe not a legitimate anti-RE solution
- But incredibly fun

- github.com/xoreaxeaxeax
  - M/o/Vfuscator 2.0 (+{.*}fuscators, crackme)
  - REpsych
  - x86 0-day PoC
  - Etc.

- Feedback? Ideas?

- domas
  - @xoreaxeaxeax
  - xoreaxeaxeax@gmail.com