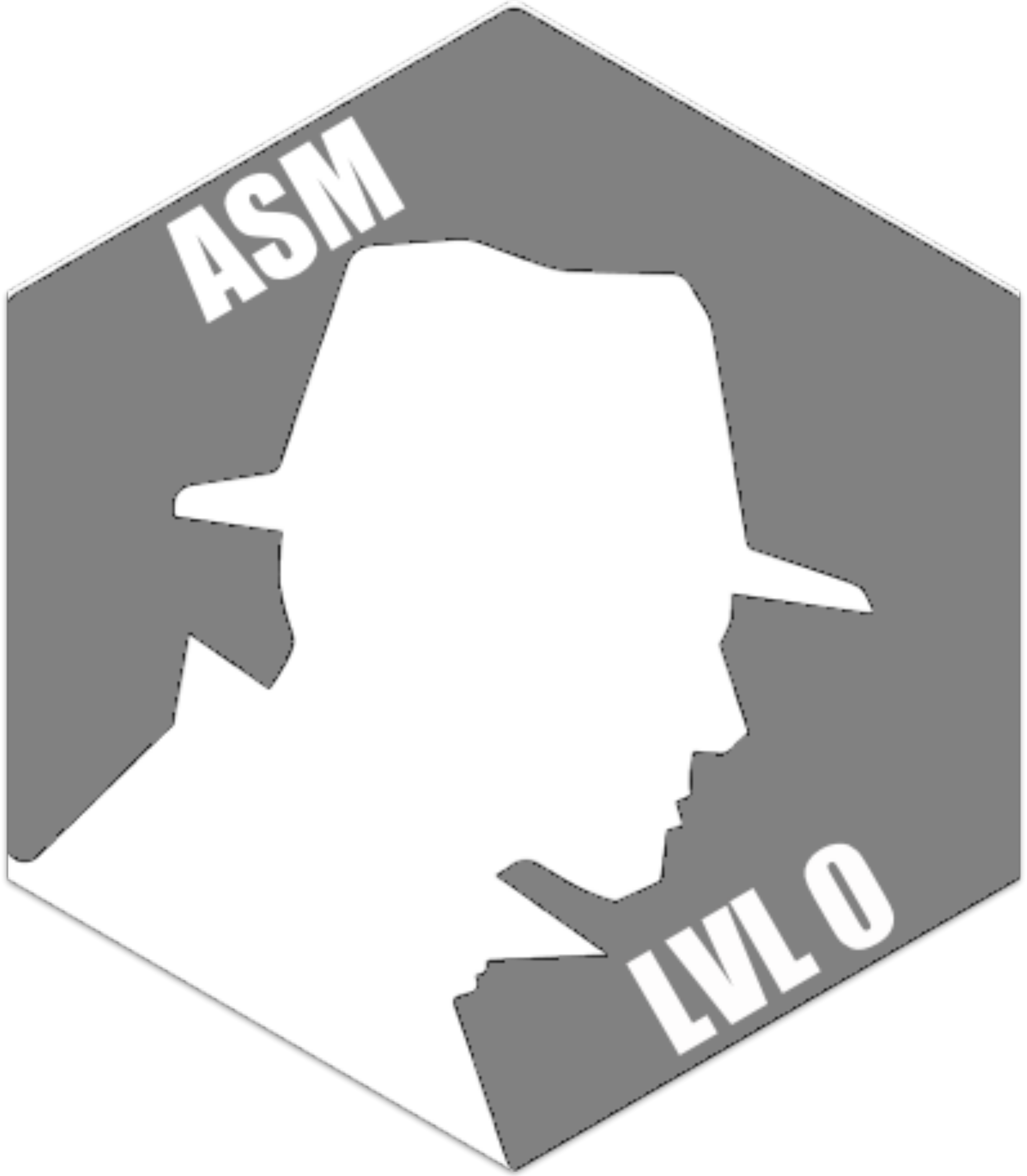


Introduction to X86-64 assembly for reverse engineering

Purposed Badges

Level 0 (Grey) = Introductory/Beginner
Level 1 (Green) = Intermediate
Level 2 (Blue) = Advanced
Level 3 (Red)   = Expert
Level 4 (Black) = 1337

Level - 0

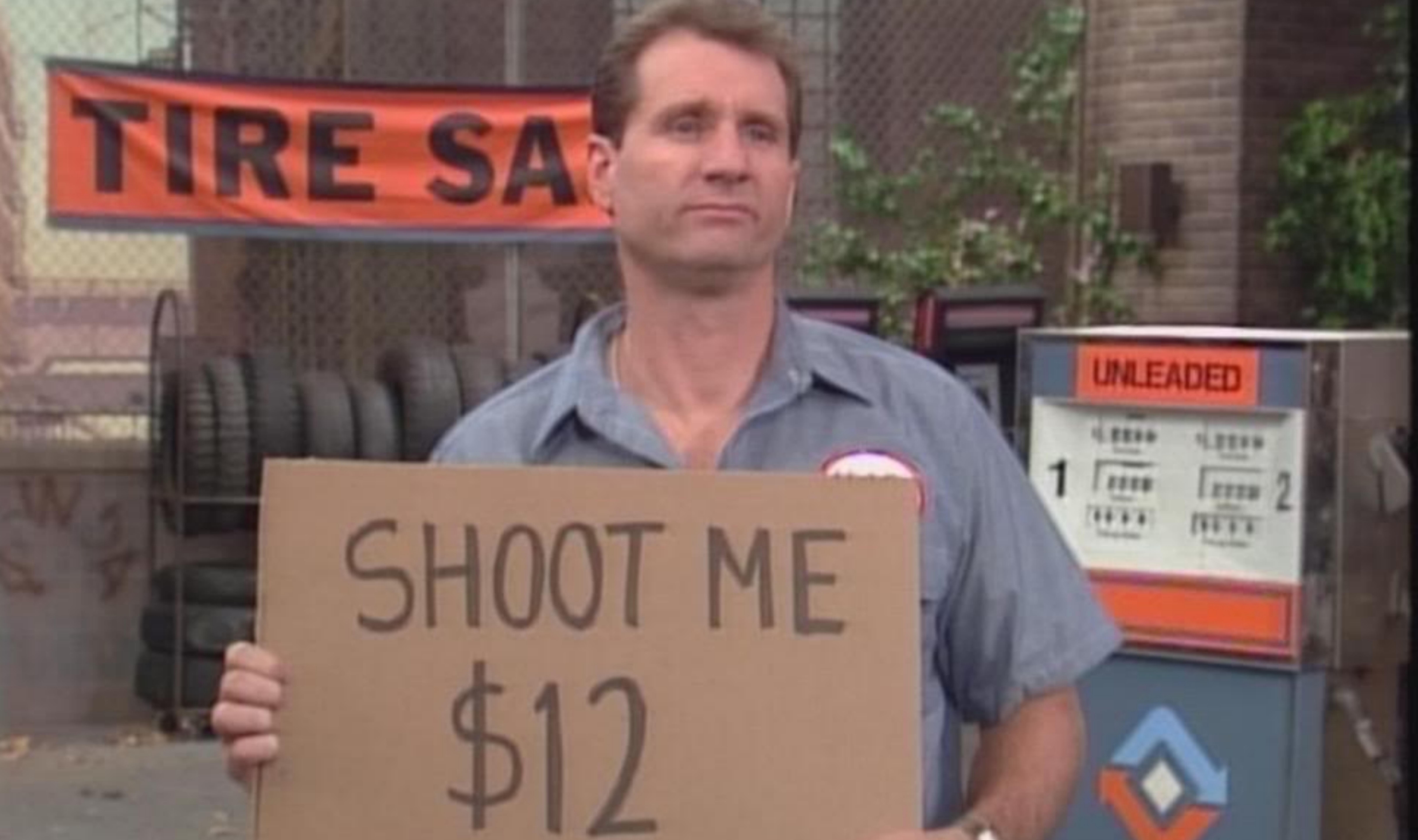
Level - 1

Level - 2

Level - 3

1337

Introduction to X86-64 assembly for reverse engineering


TIRE SA
UNLEADED
1
2
SHOOT ME
$12

lolwut?

What is assembly?

Why Learn Intel x86-64 Assembly?

Provide Learning Resources & Support

https://coggle.it/diagram/VeUyGSQQSAJrZybd

SecKC x86-64 ASM 101

You are here.

Architecture

Endianness

Registers

RAX
RBX
RCX
RDX
RSP
RSI
RDI
RSP
RBP
RIP

The Stack

Instructions

NOP
PUSH/POP
CALL/RET
MOV
ADD/SUB
IMUL
MOVZX/MOVSX
LEA
JMP/JCC
CMP/TEST
AND/OR/XOR/NOT
INC/DEC
SHR/SHL/SAR/SAL
DIV/IDIV
REP
STOS
REP
MOVS

Tools

Hopper/Radare2/IDA
GDB
Objdump
Visual Studio

http://re.seckc.org:4000

Slack | Introduction to ASM Room

https://microcorruption.com

https://bitbucket.org/nodatafound/seckc-asm-101

carnegie mellon binary bomb

http://forums.cisco.com/CertCom/game/binary_game_page.htm

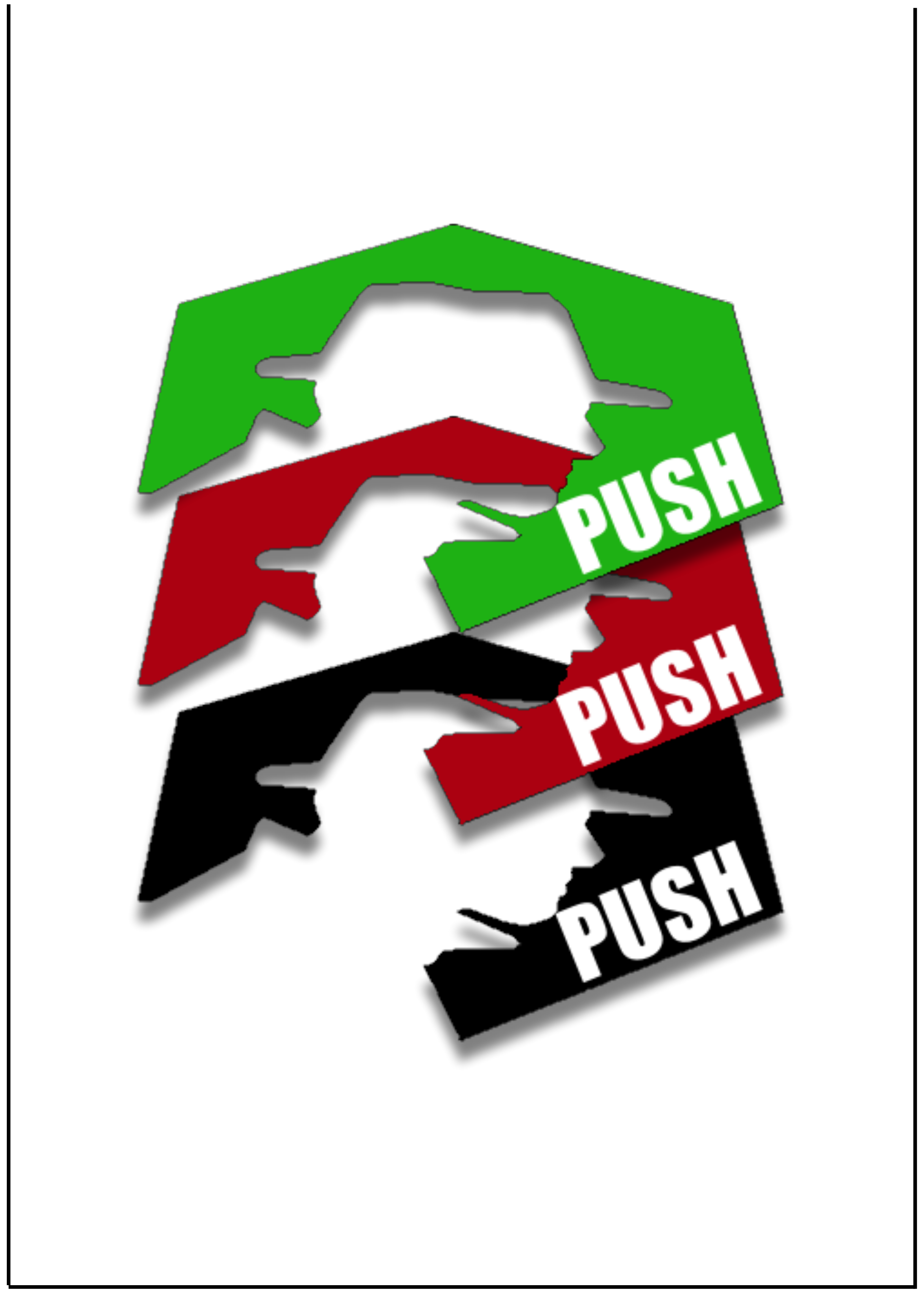
```
Instructions

PUSH, POP
```

Your first instructions

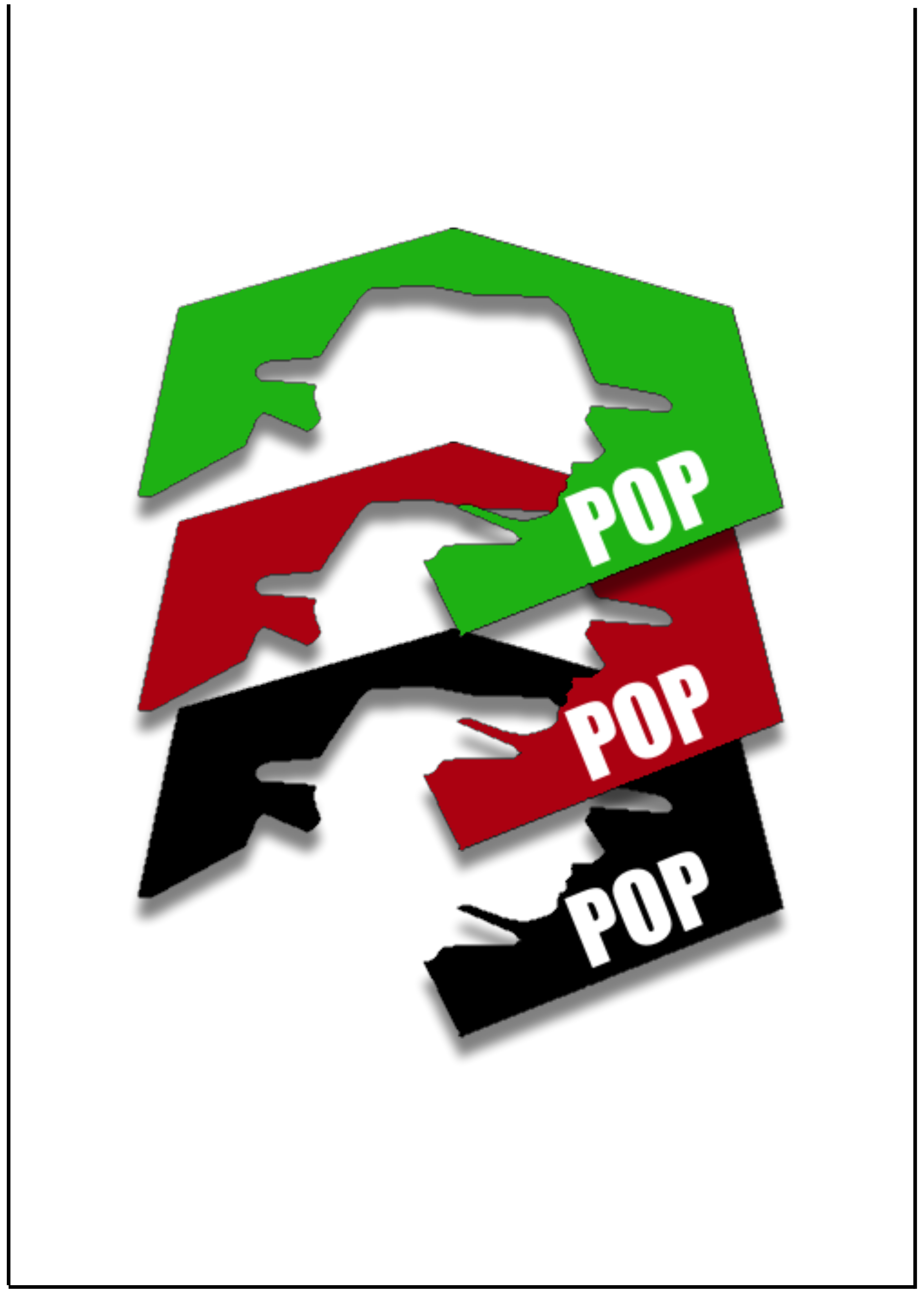
PUSH = Redo or Control Y

POP = Undo or Control Z

PUSH,POP & LIFO Stack



PUSH,POP & LIFO Stack



#justsyntaxthings

Intel: Destination <= Source(s)

Windows | Think algebra or C | y=2x+1;

```
mov rbp, rsp
add rsp, 0x1337 ; (rsp = rsp + 0x1337)
```

AT&T: Source(s) => Destination

*NIX | Think Elementary School | 1+2=3

```
mov %rsp, %rbp
add $0x1337,%rsp
```

Your first registers

RBP - Stack base pointer

RSP - Stack top pointer

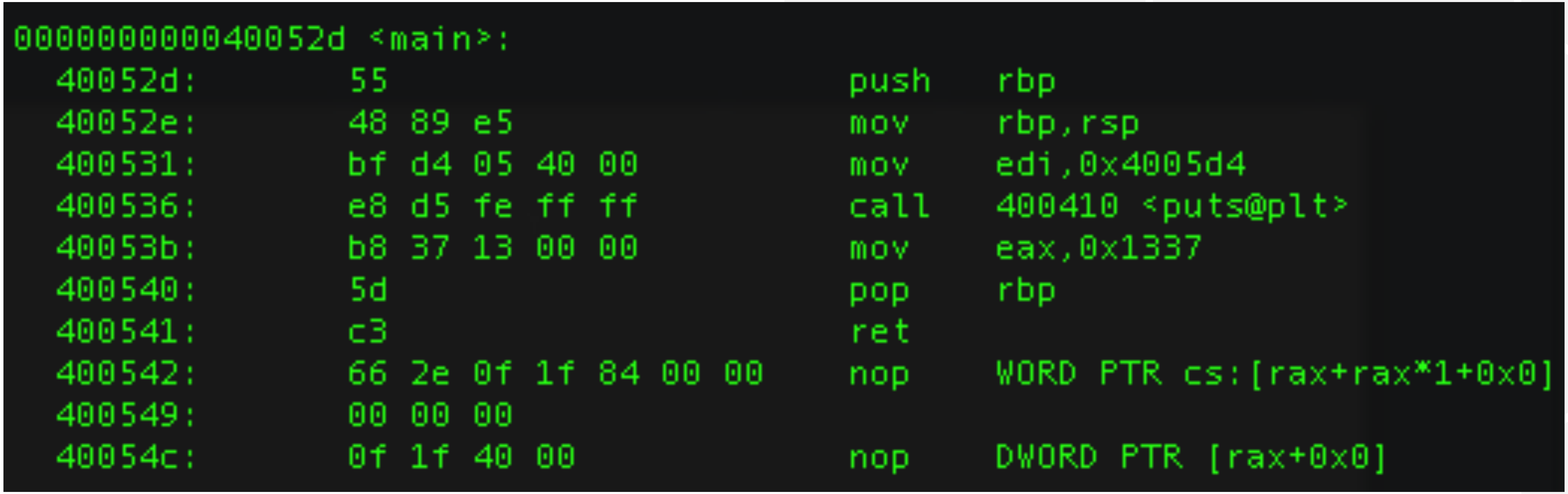
RDI - Destination pointer for string operations

RAX - Stores function return values

```c
#include <stdio.h>
int main(){
    printf("Im too SecKC for a GUI!\n");
    return 0x1337;
}
```

```
SecKC_ASM_101:
(__TEXT,__text) section
_main:
0000000100000f30    pushq   %rbp
0000000100000f31    movq    %rsp, %rbp
0000000100000f34    subq    $0x10, %rsp
0000000100000f38    leaq    0x3f(%rip), %rdi        ## literal pool for: "Im too SecKC for a GUI!\n"
0000000100000f3f    movl    $0x0, -0x4(%rbp)
0000000100000f46    movb    $0x0, %al
0000000100000f48    callq   0x100000f5e             ## symbol stub for: _printf
0000000100000f4d    movl    $0x1337, %ecx           ## imm = 0x1337
0000000100000f52    movl    %eax, -0x8(%rbp)
0000000100000f55    movl    %ecx, %eax
0000000100000f57    addq    $0x10, %rsp
0000000100000f5b    popq    %rbp
0000000100000f5c    retq
```

otool -tV SecKC_ASM_101



```
000000000040052d <main>:
  40052d:       55                      push   rbp
  40052e:       48 89 e5                mov    rbp,rsp
  400531:       bf d4 05 40 00          mov    edi,0x4005d4
  400536:       e8 d5 fe ff ff          call   400410 <puts@plt>
  40053b:       b8 37 13 00 00          mov    eax,0x1337
  400540:       5d                      pop    rbp
  400541:       c3                      ret
  400542:       66 2e 0f 1f 84 00 00    nop    WORD PTR cs:[rax+rax*1+0x0]
  400549:       00 00 00
  40054c:       0f 1f 40 00             nop    DWORD PTR [rax+0x0]
```

objdump -M intel -d SecKC | less

Challenge time! Find the base memory address.

1. Compile C

gcc -ggdb -o SecKC SecKC.c

2. View executable in assembly form

objdump -d SecKC | less

3. Debug & Set Break point

gdb SecKC

(gdb) set disassembly-flavor intel

(gdb) list

(gdb) Break main or 2

(gdb) run

(gdb) disassemble main

(gdb) si

(gdb) x/24c $edi

Contents of SecKC.c

```
#include <stdio.h>
int main(){
    printf("Im too SecKC for a GUI!\n");
    return 0x1337;
}
```

Demo

Resources!

https://goo.gl/e2v2PG

Decimal (base 10)	Binary (base 2)	Hex (base 16)
00	0000b	0x00
01	0001b	0x01
02	0010b	0x02
03	0011b	0x03
04	0100b	0x04
05	0101b	0x05
06	0110b	0x06
07	0111b	0x07
08	1000b	0x08
09	1001b	0x09
10	1010b	0x0A
11	1011b	0x0B
12	1100b	0x0C
13	1101b	0x0D
14	1110b	0x0E
15	1111b	0x0F

- First 4 parameters (from left to right) are put into RCX, RDX, R8, R9 respectively (CD89 - X86-64)

- RDI, RSI, RDX, RCX, R8, R9 (AMD64 ABI (GCC))

RAX - Stores function return values

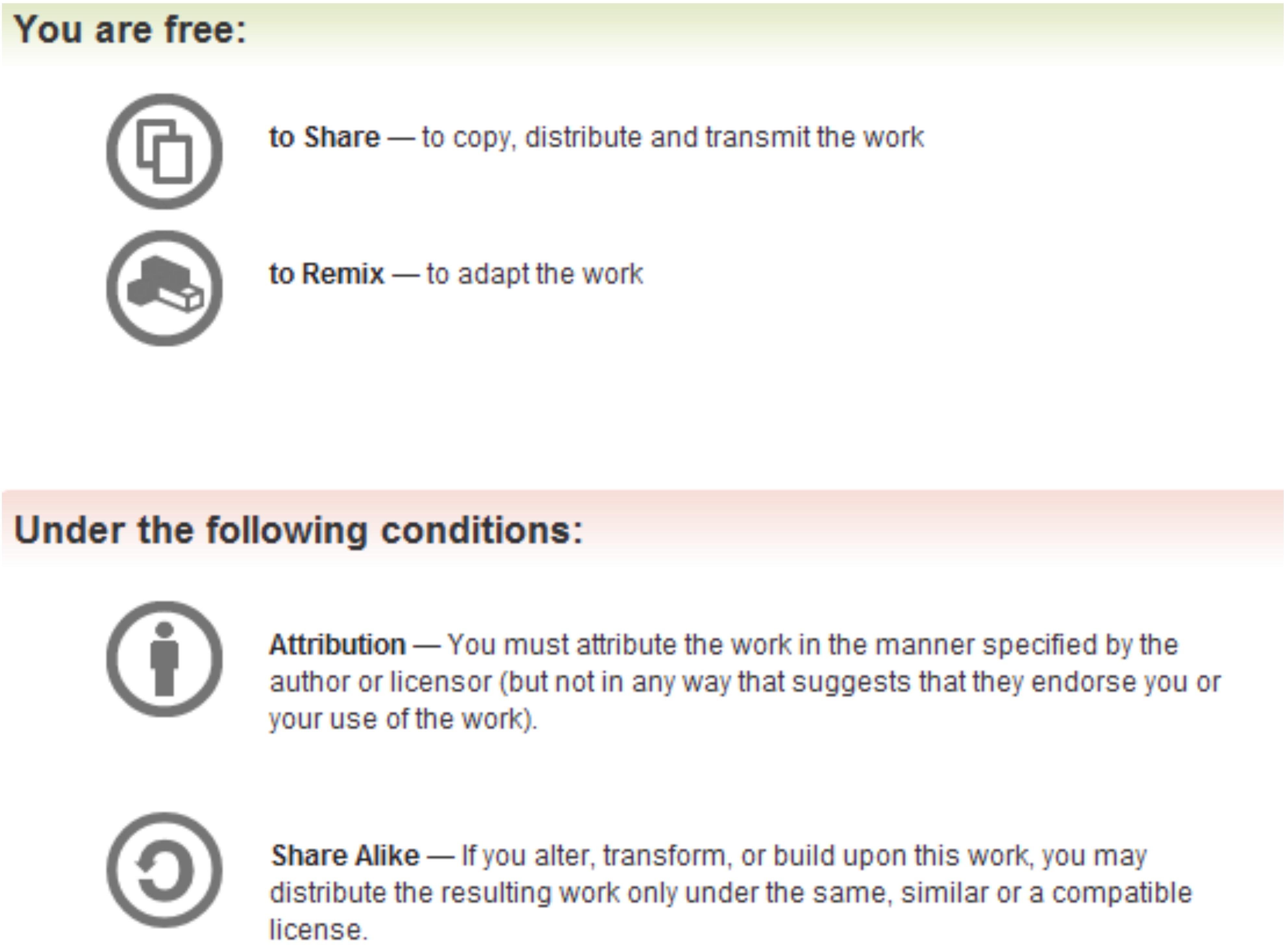
RBX - Base pointer to the data section
RCX - Counter for string and loop operations
RDX - I/O pointer
RSP - is the most critical in the class.
RSP is the pointer to the top of the stack.
RSI - Source pointer for string operations
RDI - Destination pointer for string operations
RSP - Stack top pointer
RBP - Stack frame base pointer
RIP - Pointer to next instruction to execute ("instruction pointer")

Shadow stack space calls a function:
Call **cs:__imp_printf** or **call qword ptr [__imp_printf** for example

Registers known:
NOP
PUSH/POP
CALL/RET
MOV
ADD/SUB
IMUL
MOVZX/MOVSX
LEA
JMP/Jcc (family)
CMP/TEST
AND/OR/XOR/NOT
INC/DEC
SHR/SHL/SAR/SAL
DIV/IDIV
REP STOS
REP MOVS

Architecture - Registers – 8/16/32/64 bit addressing 1

Memory Notes	Memory	Registers	Register Notes



Portions of this work were derived from Xeno Kovah's 'Intro x86-64’ class.

cory@doessteveknow.com