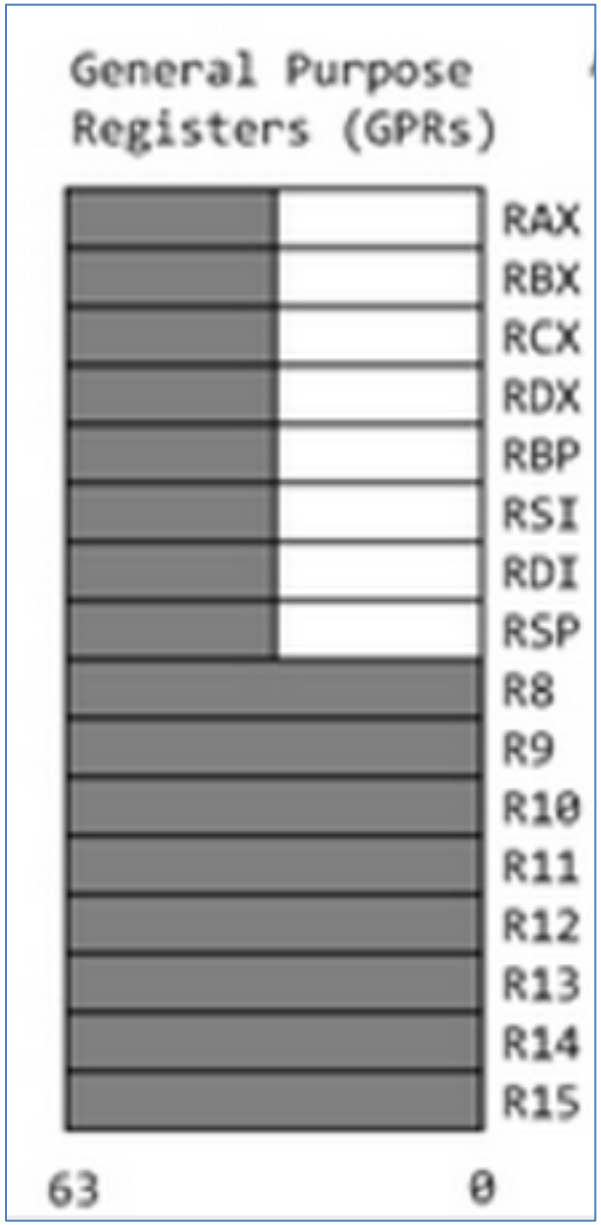
CNIT 127: Exploit Development

Lecture 7: 64-bit Assembler

Not in textbook

Rev. 3-9-17



64-bit Registers

- rip = Instruction pointer
- rsp = top of stack

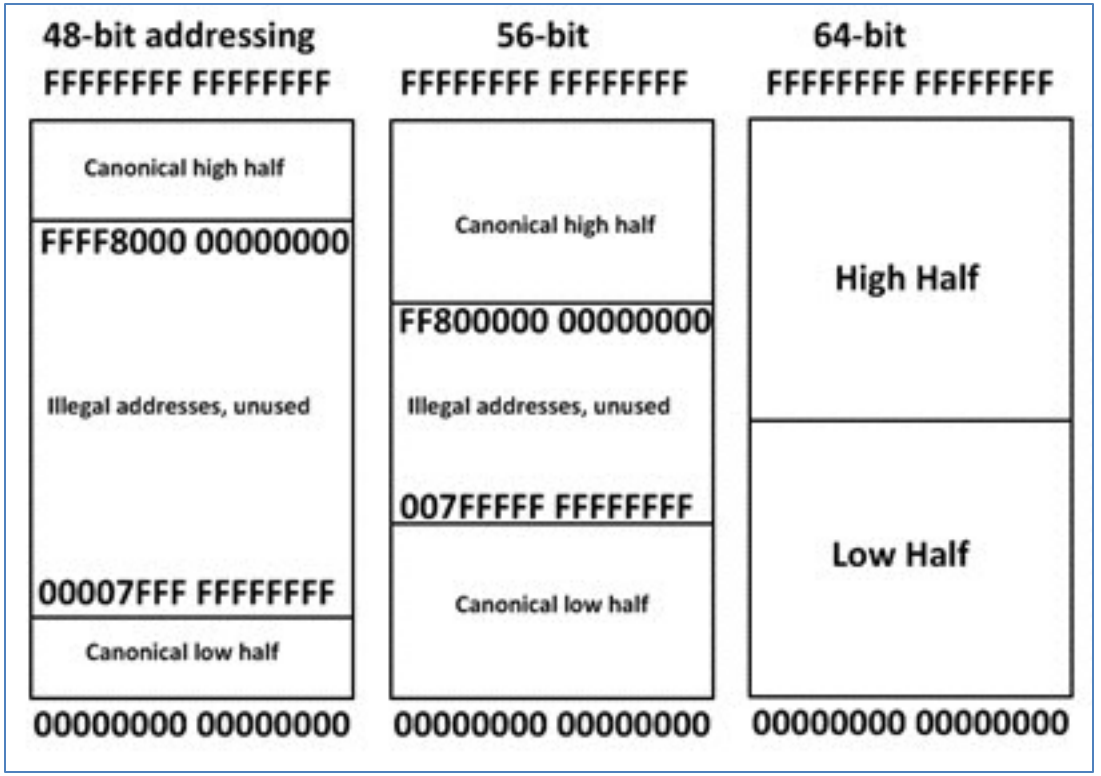
64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

Windows Limitations

- Windows doesn't implement full 64-bit addressing
- Windows 2008 Server uses 44 bits
  - Max. 16 TB RAM
- Windows 8.1, 2015 revision, uses 48 bits
  - Max. 256 TB RAM
- Links Ch L7d, L7e

OS Limitations

- OS uses top half
- User programs use lower half



System Calls

- syscall replaces INT 80

- **System Calls**
  - The kernel or system call interface uses registers `RDI`, `RSI`, `RDX`, `R10`, `R8`, `R9` for passing arguments in that order. A maximum of 6 parameters can be passed.

  - The number of the system call is passed in the register `RAX`.
  - No argument is passed on the stack.

L7h: Searchable Linux Syscall Table

Instruction: `syscall`

Return value found in: `%rax`

Syscalls are implemented in functions named as in the *Entry point* column, or with the `DEFINE_SYSCALLx(%name%` macro.

Relevant man pages: `syscall(2)`, `syscalls(2)`

Double click on a row to reveal the arguments list. Search using the fuzzy filter box.

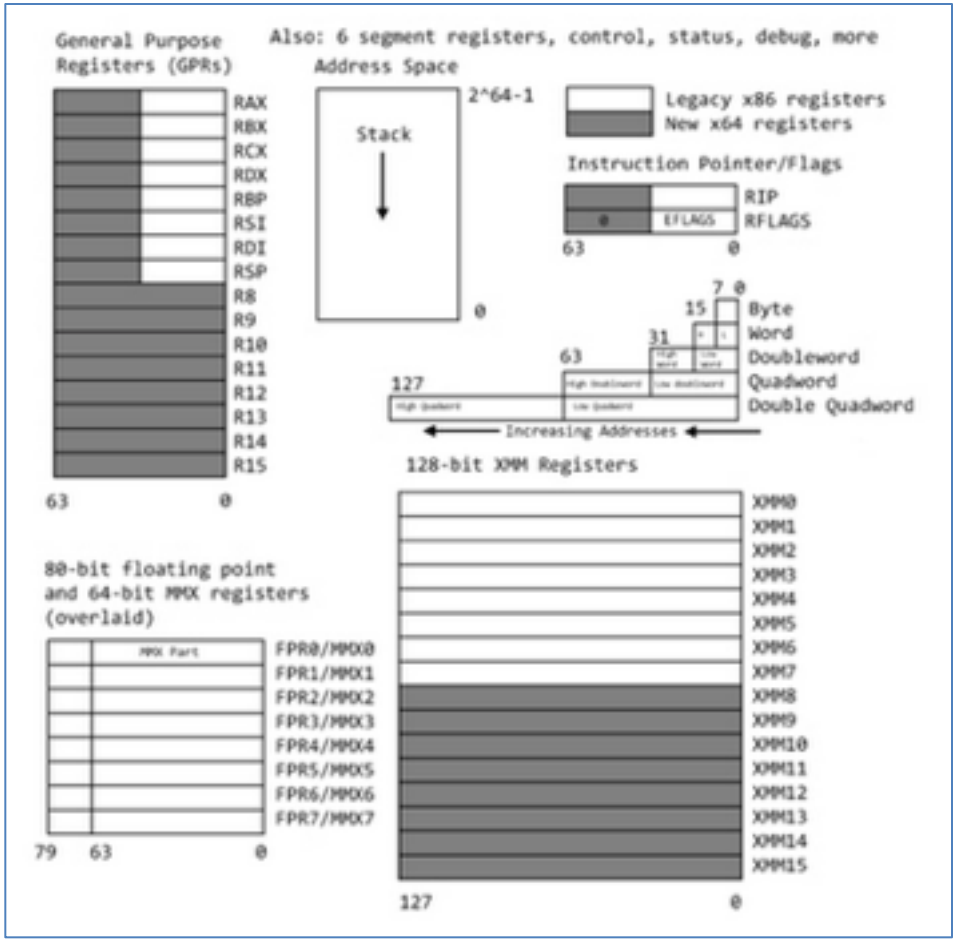
Filter:

%rax	Name	Entry point	Implementation
0	read	sys_read	fs/read_write.c
1	write	sys_write	fs/read_write.c

%rdi	%rsi	%rdx
unsigned int fd	**const char __user *** buf	**size_t** count

L7c: Introduction to x64 Assembly Intel Developer Zone

- More details about registers



Common Opcodes

Table 4 - Common Opcodes

Opcode	Meaning	Opcode	Meaning
MOV	Move to/from/between memory and registers	AND/OR/XOR/NOT	Bitwise operations
CMOV*	Various conditional moves	SHR/SAR	Shift right logical/arithmetic
XCHG	Exchange	SHL/SAL	Shift left logical/arithmetic
BSWAP	Byte swap	ROR/ROL	Rotate right/left
PUSH/POP	Stack usage	RCR/RCL	Rotate right/left through carry bit
ADD/ADC	Add/with carry	BT/BTS/BTR	Bit test/and set/and reset
SUB/SBC	Subtract/with carry	JMP	Unconditional jump
MUL/IMUL	Multiply/unsigned	JE/JNE/JC/JNC/J*	Jump if equal/not equal/carry/not carry/ many others
DIV/IDIV	Divide/unsigned	LOOP/LOOPE/LOOPNE	Loop with ECX
INC/DEC	Increment/Decrement	CALL/RET	Call subroutine/return
NEG	Negate	NOP	No operation
CMP	Compare	CPUID	CPU information

Syscall 1: Write

Understanding Syscall 1: Write

From the Linux Syscall Table, this call is specified as:

%rax	Name	Entry point	Implementation
1	write	sys_write	fs/read_write.c

%rdi	%rsi	%rdx
unsigned int fd	**const char __user *** buf	**size_t** count

So to write text to the console, we must do these things:

- Set **rax** to **1** to specify the "write" syscall
- Set **rdi** to **1** (the file descriptor for stdout, the console)
- **Push** the string onto the stack
- Set **rsi** to **rsp** (the stack pointer)
- Set **rdx** to the length of the string
- Call **syscall**

Simplest Program: ABC

Works, then Crashes (no exit)

```
  GNU nano 2.2.6                          File: abc1.asm

section .text
    global _start

    _start:
        mov   rax, 0x4142434445464748    ; 'ABCDEFGH'
        push rax
        mov   rdx, 0x8      ; length of string is 8 bytes
        mov   rsi, rsp      ; Address of string is RSP because string is on the stack
        mov   rax, 0x1      ; syscall 1 is write
        mov   rdi, 0x1      ; stdout has a file descriptor of 1
        syscall             ; make the system call
```

```
root@kali:~/127/l7# yasm -f elf64 abc1.asm
root@kali:~/127/l7# ld -o abc1.out abc1.o
root@kali:~/127/l7# ./abc1.out
HGFEDCBASegmentation fault
root@kali:~/127/l7#
```

Exit

The Linux Syscall Table, specifies the "exit" call as:

60	exit	sys_exit	kernel/exit.c

```
%rdi
int error_code
```

So to exit, we must do these things:

- Set **rax** to **0x3c** (60 in decimal) to specify the "exit" syscall
- Call **syscall**

Works Without Crashing

```
 nano 2.6.3                          File: abc2.asm

section .text
    global _start

    _start:
        mov  rax, 0x4142434445464748    ; 'ABCDEFGH'
        push rax
        mov  rdx, 0x8      ; length of string is 8 bytes
        mov  rsi, rsp      ; Address of string is RSP because string is on the stack
        mov  rax, 0x1      ; syscall 1 is write
        mov  rdi, 0x1      ; stdout has a file descriptor of 1
        syscall            ; make the system call

        mov  rax, 0x3c     ; syscall 3c is exit
        syscall            ; make the system call
```

```
root@kali:~/127/l7# yasm -f elf64 abc2.asm
root@kali:~/127/l7# ld -o abc2.out abc2.o
root@kali:~/127/l7# ./abc2.out
HGFEDCBAroot@kali:~/127/l7#
```

Letters in Order

```
 nano 2.6.3                    File: abc3.asm                    Modified

section .text
    global _start

    _start:
        mov  rax, 0x4847464544434241    ; 'ABCDEFGH' reversed
        push rax
        mov  rdx, 0x8      ; length of string is 8 bytes
        mov  rsi, rsp      ; Address of string is RSP because string is on the stack
        mov  rax, 0x1      ; syscall 1 is write
        mov  rdi, 0x1      ; stdout has a file descriptor of 1
        syscall            ; make the system call

        mov  rax, 0x3c     ; syscall 3c is exit
        syscall            ; make the system call
```

```
root@kali:~/127/l7# yasm -f elf64 abc3.asm
root@kali:~/127/l7# ld -o abc3.out abc3.o
root@kali:~/127/l7# ./abc3.out
ABCDEFGHroot@kali:~/127/l7#
```

Using a .data section

```
  nano 2.6.3                          File: hello.asm

section .data
    string1 db  "Hello World!",10    ; '10' at end is line feed

section .text
    global _start

    _start:
        mov  rdx, 0xd                ; length of string is 13 bytes
        mov  rsi, dword string1      ; set rsi to pointer to string
        mov  rax, 0x1                ; syscall 1 is write
        mov  rdi, 0x1                ; stdout has a file descriptor of 1
        syscall                      ; make the system call

        mov  rax, 0x3c               ; syscall 3c is exit
        syscall                      ; make the system call
```

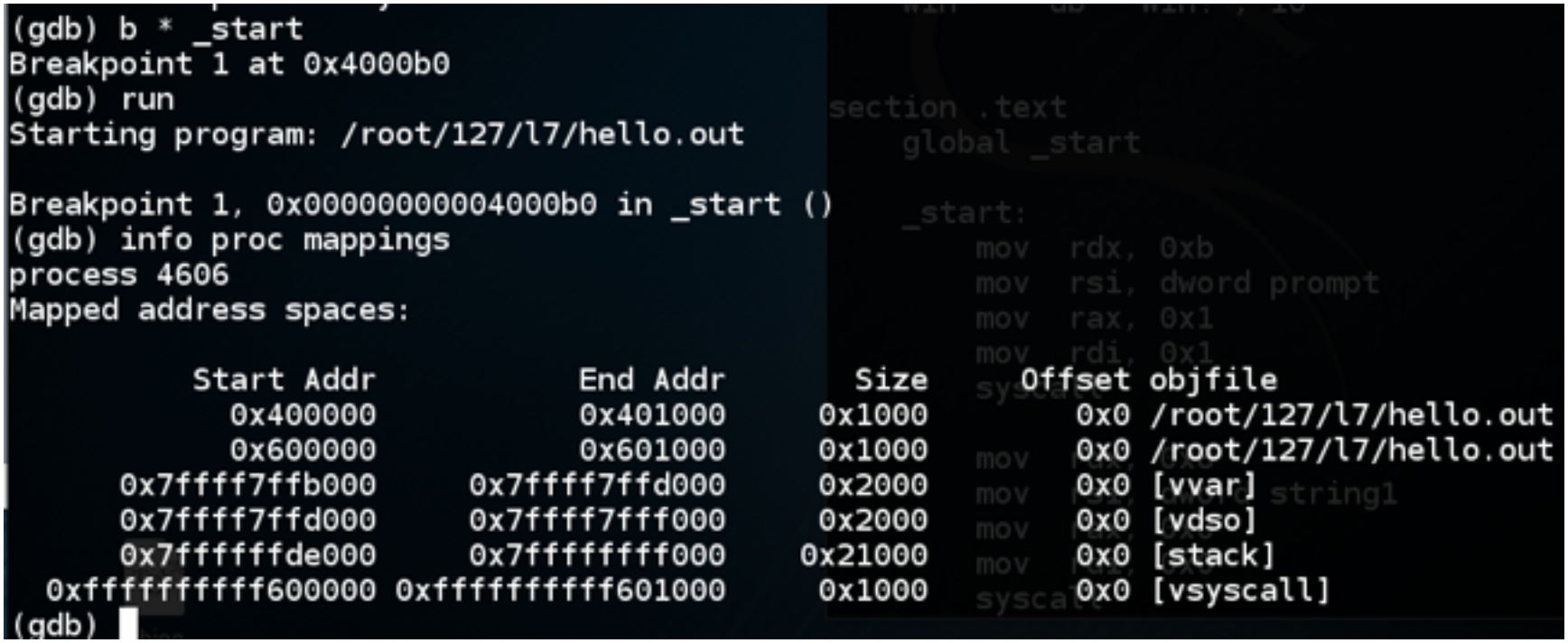
- db = "Define Byte"

Objdump

```
root@kali:~/127/l7# objdump -x hello.out

hello.out:     file format elf64-x86-64
hello.out
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00000000004000b0
```

```
SYMBOL TABLE:
00000000004000b0 l    d  .text  0000000000000000 .text
00000000006000d8 l    d  .data  0000000000000000 .data
0000000000000000 l    df *ABS*  0000000000000000 hello.asm
00000000006000d8 l       .data  0000000000000000
00000000004000b0 g       .text  0000000000000000 _start
00000000006000e5 g       .data  0000000000000000 __bss_start
00000000006000e5 g       .data  0000000000000000 _edata
00000000006000e8 g       .data  0000000000000000 _end
```

Using gdb



```
(gdb) b * _start
Breakpoint 1 at 0x4000b0
(gdb) run
Starting program: /root/127/l7/hello.out

Breakpoint 1, 0x00000000004000b0 in _start ()
(gdb) info proc mappings
process 4606
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
            0x400000           0x401000     0x1000        0x0 /root/127/l7/hello.out
            0x600000           0x601000     0x1000        0x0 /root/127/l7/hello.out
      0x7ffff7ffb000     0x7ffff7ffd000     0x2000        0x0 [vvar]
      0x7ffff7ffd000     0x7ffff7fff000     0x2000        0x0 [vdso]
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
(gdb)
```

- .data and .text sections appear the same

.text and .data Sections

```
(gdb) x/7i 0x4000b0
=> 0x4000b0 <_start>:    mov    $0xd,%rdx
   0x4000b7 <_start+7>:  mov    $0x6000d8,%rsi
   0x4000be <_start+14>:        mov    $0x1,%rax
   0x4000c5 <_start+21>:        mov    $0x1,%rdi
   0x4000cc <_start+28>:        syscall
   0x4000ce <_start+30>:        mov    $0x3c,%rax
   0x4000d5 <_start+37>:        syscall
(gdb)
```

```
(gdb) x/20c 0x6000d8
0x6000d8:       72 'H'  101 'e' 108 'l' 108 'l' 111 'o' 32 ' '  87 'W'  111 'o'
0x6000e0:       114 'r' 108 'l' 100 'd' 33 '!'  10 '\n' 0 '\000'        46 '.'  115 's'
0x6000e8:       121 'y' 109 'm' 116 't' 97 'a'
(gdb)
```

info registers

```
Breakpoint 1, 0x00000000004000b0 in _start ()
(gdb) info registers
rax            0x0      0
rbx            0x0      0
rcx            0x0      0
rdx            0x0      0
rsi            0x0      0
rdi            0x0      0
rbp            0x0      0x0
rsp            0x7fffffffe3b0   0x7fffffffe3b0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
r13            0x0      0
r14            0x0      0
r15            0x0      0
rip            0x4000b0 0x4000b0 <_start>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb)
```

Using read

"echo" with a .data section

```
  nano 2.6.3                          File: read.asm

section .data
    string1 db  "AAAABBBBCCX"          ; Reserve space for 10 characters

section .text
    global _start

    _start:
        mov  rdx, 0xa                  ; length of string is 10 bytes
        mov  rsi, dword string1        ; set rsi to pointer to string
        mov  rax, 0x0                  ; syscall 0 is read
        mov  rdi, 0x0                  ; stdin has a file descriptor of 0
        syscall                        ; make the system call

        mov  rdx, 0xa                  ; length of string is 10 bytes
        mov  rsi, dword string1        ; set rsi to pointer to string
        mov  rax, 0x1                  ; syscall 1 is write
        mov  rdi, 0x1                  ; stdout has a file descriptor of 1
        syscall                        ; make the system call

        mov  rax, 0x3c                 ; syscall 3c is exit
        syscall                        ; make the system call
```

Works with Junk at End

```
root@kali:~/127/l7# yasm -f elf64 read.asm
root@kali:~/127/l7# ld -o read.out read.o
root@kali:~/127/l7# ./read.out
APPLE
APPLE
BBCCroot@kali:~/127/l7#
```

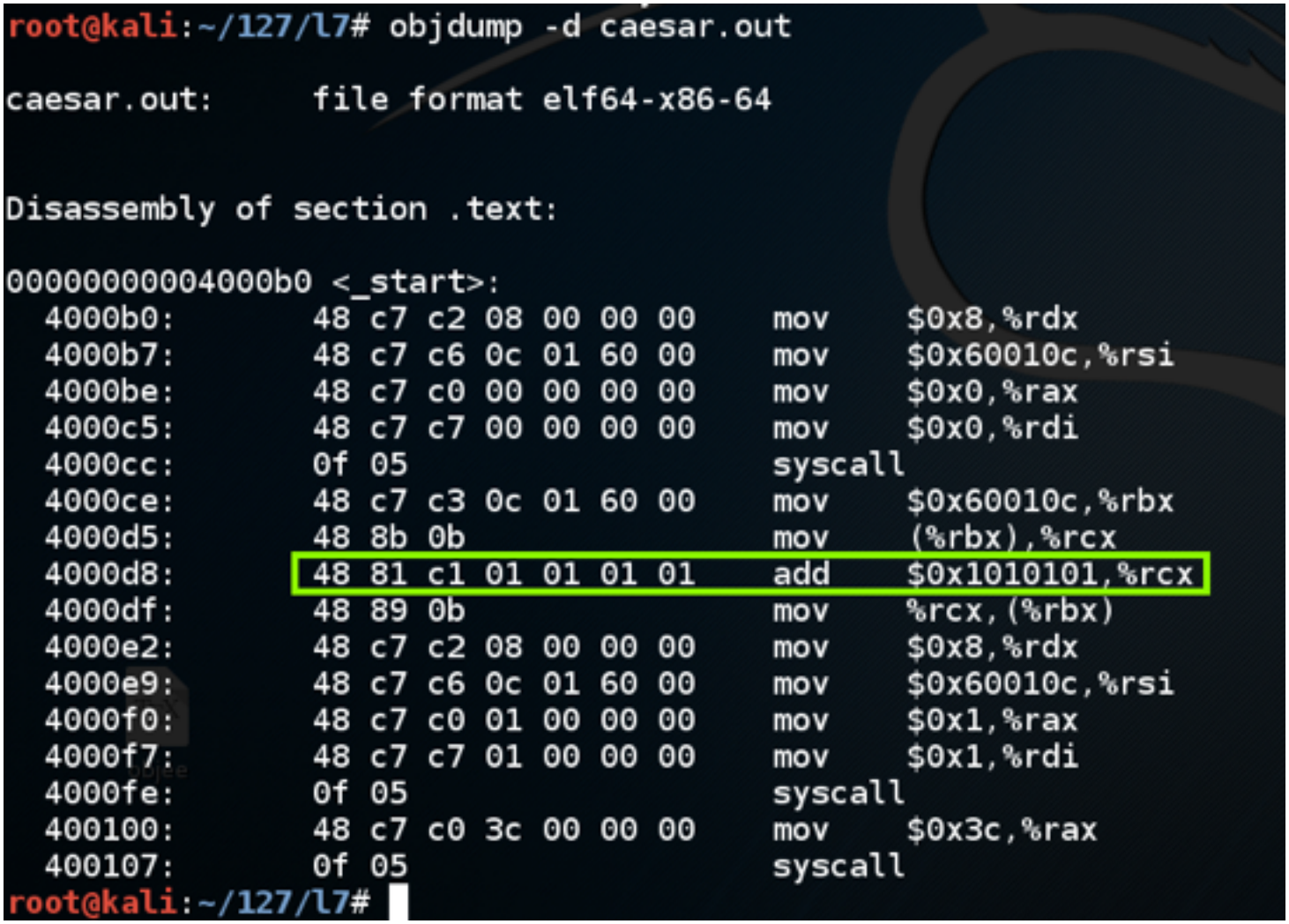
Caesar Cipher

```
 nano 2.6.3                    File: caesar.asm                          Modif
section .data
    string1 db  "AAAABBBB"              ; Reserve space for 8 characters

section .text
    global _start

    _start:
        mov  rdx, 0x8                   ; length of string is 8 bytes
        mov  rsi, dword string1         ; set rsi to pointer to string
        mov  rax, 0x0                   ; syscall 1 is read
        mov  rdi, 0x0                   ; stdin has a file descriptor of 0
        syscall                         ; make the system call

        mov  rbx, dword string1         ; set rbx to pointer to string
        mov  rcx, [rbx]                 ; Put string value into rcx
        add  rcx, 0x0101010101010101 ; Add 1 to each byte, not fixing rollover
        mov  [rbx], rcx                 ; Put modified byte on string

        mov  rdx, 0x8                   ; length of string is 8 bytes
        mov  rsi, dword string1         ; set rsi to pointer to string
        mov  rax, 0x1                   ; syscall 1 is write
        mov  rdi, 0x1                   ; stdout has a file descriptor of 1
        syscall                         ; make the system call

        mov  rax, 0x3c                  ; syscall 3c is exit
        syscall                         ; make the system call
```

Works for 4 Bytes Only

```
root@kali:~/127/l7# yasm -f elf64 caesar.asm
caesar.asm:16: warning: value does not fit in 32 bit field
root@kali:~/127/l7# ld -o caesar.out caesar.o
root@kali:~/127/l7# ./caesar.out
HELLO
IFMMO
BBroot@kali:~/127/l7#
```

Objdump Shows a 32-bit Value



Intel 64 and IA-32 Architectures Software Developer's Manual

ADD—Add

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
04 *ib*	ADD AL, *imm8*	I	Valid	Valid	Add *imm8* to AL.
05 *iw*	ADD AX, *imm16*	I	Valid	Valid	Add *imm16* to AX.
05 *id*	ADD EAX, *imm32*	I	Valid	Valid	Add *imm32* to EAX.
REX.W + 05 *id*	ADD RAX, *imm32*	I	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to RAX.
80 /0 *ib*	ADD *r/m8*, *imm8*	MI	Valid	Valid	Add *imm8* to *r/m8*.
REX + 80 /0 *ib*	ADD *r/m8**, *imm8*	MI	Valid	N.E.	Add *sign-extended imm8* to *r/m64*.
81 /0 *iw*	ADD *r/m16*, *imm16*	MI	Valid	Valid	Add *imm16* to *r/m16*.
81 /0 *id*	ADD *r/m32*, *imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.
REX.W + 81 /0 *id*	ADD *r/m64*, *imm32*	MI	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to *r/m64*.
83 /0 *ib*	ADD *r/m16*, *imm8*	MI	Valid	Valid	Add *sign-extended imm8* to *r/m16*.
83 /0 *ib*	ADD *r/m32*, *imm8*	MI	Valid	Valid	Add *sign-extended imm8* to *r/m32*.
REX.W + 83 /0 *ib*	ADD *r/m64*, *imm8*	MI	Valid	N.E.	Add *sign-extended imm8* to *r/m64*.
00 /*r*	ADD *r/m8*, *r8*	MR	Valid	Valid	Add *r8* to *r/m8*.
REX + 00 /*r*	ADD *r/m8**, *r8**	MR	Valid	N.E.	Add *r8* to *r/m8*.
01 /*r*	ADD *r/m16*, *r16*	MR	Valid	Valid	Add *r16* to *r/m16*.
01 /*r*	ADD *r/m32*, *r32*	MR	Valid	Valid	Add r32 to *r/m32*.
REX.W + 01 /*r*	ADD *r/m64*, *r64*	MR	Valid	N.E.	Add r64 to *r/m64*.
02 /*r*	ADD *r8*, *r/m8*	RM	Valid	Valid	Add *r/m8* to *r8*.
REX + 02 /*r*	ADD *r8**, *r/m8**	RM	Valid	N.E.	Add *r/m8* to *r8*.
03 /*r*	ADD *r16*, *r/m16*	RM	Valid	Valid	Add *r/m16* to *r16*.
03 /*r*	ADD *r32*, *r/m32*	RM	Valid	Valid	Add *r/m32* to *r32*.
REX.W + 03 /*r*	ADD *r64*, *r/m64*	RM	Valid	N.E.	Add *r/m64* to *r64*.

NOTES:

*In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Must use a Register

```
 nano 2.6.3                    File: caesar2.asm                    Mod
section .data
    string1 db  "AAAABBBB"              ; Reserve space for 8 characters

section .text
    global _start

    _start:
        mov  rdx, 0x8                   ; length of string is 8 bytes
        mov  rsi, dword string1         ; set rsi to pointer to string
        mov  rax, 0x0                   ; syscall 1 is read
        mov  rdi, 0x0                   ; stdin has a file descriptor of 0
        syscall                         ; make the system call

        mov  rbx, dword string1         ; set rbx to pointer to string
        mov  rcx, [rbx]                 ; Put string value into rcx
        mov  r8, 0x0101010101010101     ; Put value in r8
        add  rcx, r8                    ; Add using registers
        mov  [rbx], rcx                 ; Put modified byte on string

        mov  rdx, 0x8                   ; length of string is 8 bytes
        mov  rsi, dword string1         ; set rsi to pointer to string
        mov  rax, 0x1                   ; syscall 1 is write
        mov  rdi, 0x1                   ; stdout has a file descriptor of 1
        syscall                         ; make the system call

        mov  rax, 0x3c                  ; syscall 3c is exit
        syscall                         ; make the system call
```

Now it Works

```
root@kali:~/127/l7# yasm -f elf64 caesar2.asm
root@kali:~/127/l7# ld -o caesar2.out caesar2.o
root@kali:~/127/l7# ./caesar2.out
HELLO
IFMMP
     CCroot@kali:~/127/l7#
```

Challenge 1
"Hello from YOURNAME"

```
root@kali:~/127/l7# ./hello.out
Hello from YOURNAME
root@kali:~/127/l7# cat hello.asm
section .text
    global _start

    _start:

        mov  rax, 0x3c      ; syscall 3c is exit
        syscall             ; make the system call
```

Challenge 2
Caesar (3 steps back)

```
root@kali:~/127/l7/chal# ./caesar3.out
HELLO
EBIIL??root@kali:~/127/l7/chal# cat caesar3.asm
section .data
    string1 db  "AAAABBBB"              ; Reserve space for 8 characters

section .text
    global _start

    _start:
        mov  rax, 0x3c    ; syscall 3c is exit
        syscall           ; make the system call
root@kali:~/127/l7/chal#
```

Challenge 3: XOR Encryption

```
root@kali:~/127/l7/chal# ./xor.out
cnit127!
'!.0~usn root@kali:~/127/l7/chal#
root@kali:~/127/l7/chal# cat xor.asm
section .data
    [redacted]

section .text
    global _start

    _start:
        [redacted]
```