

# CNIT 127: Exploit Development

## Lecture 7: 64-bit Assembler

Not in textbook

Rev. 3-9-17

# 64-bit Registers

General Purpose  
Registers (GPRs)

	RAX
	RBX
	RCX
	RDX
	RBP
	RSI
	RDI
	RSP
	R8
	R9
	R10
	R11
	R12
	R13
	R14
	R15

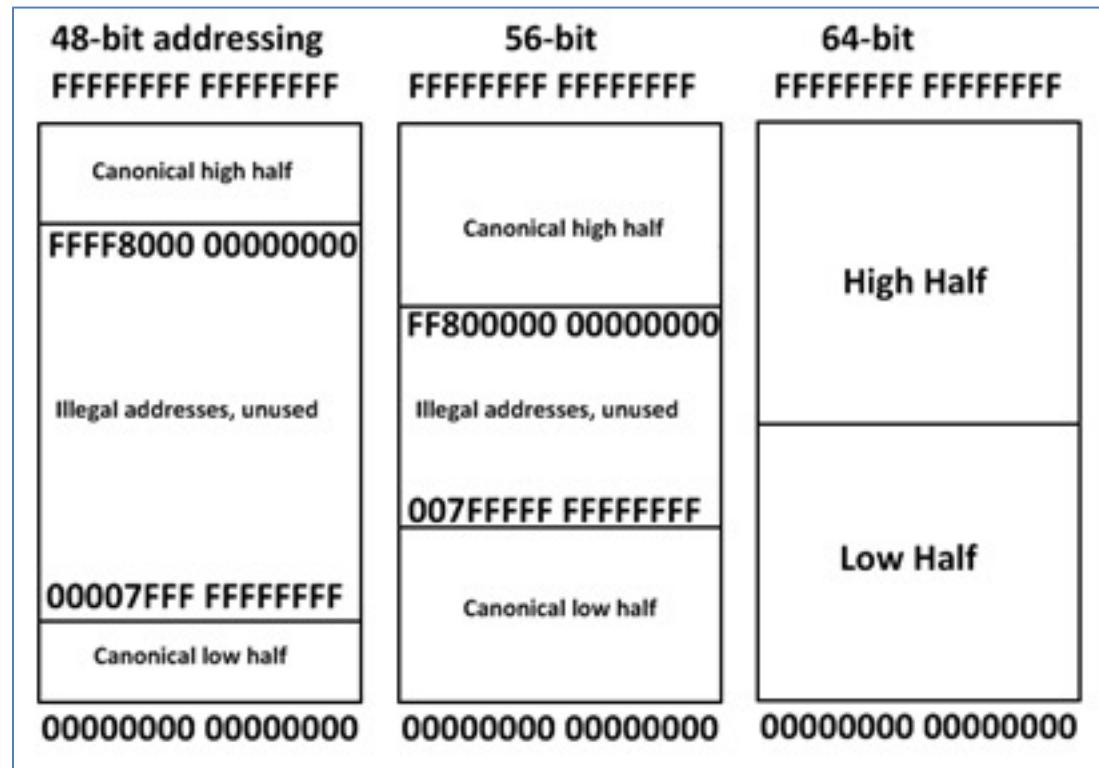
64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15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

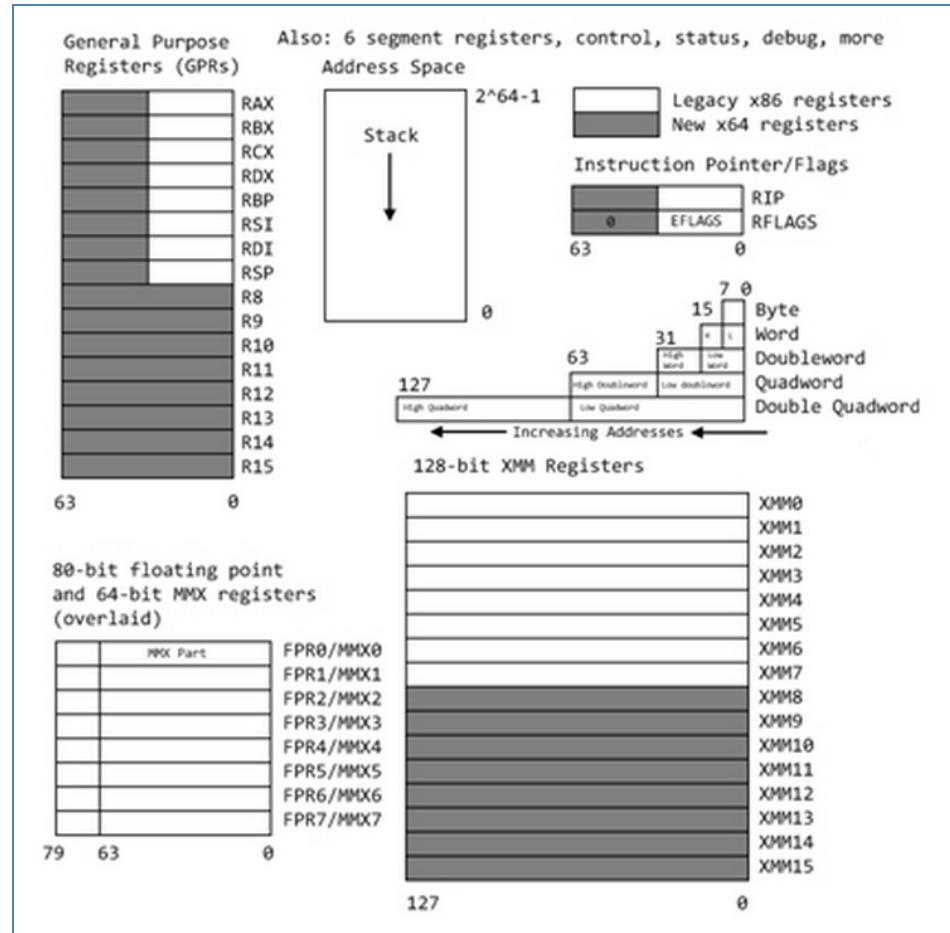
Filter:

<code>%rax</code>	Name	Entry point	Implementation
0	read	<code>sys_read</code>	<a href="#">fs/read_write.c</a>
1	write	<code>sys_write</code>	<a href="#">fs/read_write.c</a>
<code>%rdi</code> <code>unsigned int fd</code>		<code>%rsi</code> <code>const char __user * buf</code>	<code>%rdx</code> <code>size_t count</code>

# 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	<a href="#">fs/read_write.c</a>
%rdi <b>unsigned int fd</b>	%rsi <b>const char __user * buf</b>		%rdx <b>size_t count</b>

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

# Exit

The [Linux Syscall Table](#), specifies the "exit" call as:

60	exit	sys_exit	<a href="#">kernel/exit.c</a>
----	------	----------	-------------------------------

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

section .text
    global _start

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

    mov  rax, 0x3c
    syscall
                                ; syscall 3c is exit
                                ; 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 0x000000000004000b0
```

## SYMBOL TABLE:

000000000004000b0	l	d	.text	0000000000000000	.text	mov rdx,
000000000006000d8	l	d	.data	0000000000000000	.data	mov rsi,
0000000000000000	l	df	*ABS*	0000000000000000	hello.asm	mov rax,
000000000006000d8	l		.data	0000000000000000		mov rdi,
000000000004000b0	g		.text	0000000000000000	_start	syscall
000000000006000e5	g		.data	0000000000000000	_bss_start	mov rdx,
000000000006000e5	g		.data	0000000000000000	_edata	mov rsi,
000000000006000e8	g		.data	0000000000000000	_end	mov rax,

# 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
0x4000000              0x401000          0x1000
0x6000000              0x601000          0x1000
0x7ffff7ffb000         0x7ffff7ffd000      0x2000
0x7ffff7ffd000         0x7ffff7fff000      0x2000
0x7fffffffde000        0x7fffffff000      0x21000
0xfffffffff600000       0xfffffffff601000     0x1000
(gdb)
```

```
section .text
global _start

_start:
    mov    rdx, 0xb
    mov    rsi, dword prompt
    mov    rax, 0x1
    mov    rdi, 0x1
    syscall

Offset objfile
0x0 /root/127/l7/hello.out
0x0 /root/127/l7/hello.out
0x0 [vvar]
0x0 [vdso]
0x0 [stack]
0x0 [vsyscall]
```

- .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) objee
```

```
(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) objee
```

# 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	0x7fffffffef3b0	0x7fffffffef3b0
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
shared-
section .sh.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
shared-
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

```
root@kali:~/127/l7# objdump -d caesar.out

caesar.out:      file format elf64-x86-64

Disassembly of section .text:

00000000004000b0 <_start>:
4000b0:   48 c7 c2 08 00 00 00    mov    $0x8,%rdx
4000b7:   48 c7 c6 0c 01 60 00    mov    $0x60010c,%rsi
4000be:   48 c7 c0 00 00 00 00    mov    $0x0,%rax
4000c5:   48 c7 c7 00 00 00 00    mov    $0x0,%rdi
4000cc:   0f 05                 syscall
4000ce:   48 c7 c3 0c 01 60 00    mov    $0x60010c,%rbx
4000d5:   48 8b 0b               mov    (%rbx),%rcx
4000d8:   48 81 c1 01 01 01 01    add    $0x1010101,%rcx
4000df:   48 89 0b               mov    %rcx,(%rbx)
4000e2:   48 c7 c2 08 00 00 00    mov    $0x8,%rdx
4000e9:   48 c7 c6 0c 01 60 00    mov    $0x60010c,%rsi
4000f0:   48 c7 c0 01 00 00 00    mov    $0x1,%rax
4000f7:   48 c7 c7 01 00 00 00    mov    $0x1,%rdi
4000fe:   0f 05                 syscall
400100:   48 c7 c0 3c 00 00 00    mov    $0x3c,%rax
400107:   0f 05                 syscall

root@kali:~/127/l7#
```

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

## ADD—Add

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

### 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
shared-
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:
                                ; [REDACTED] - A large gray rectangular redaction box covers the assembly code body.

        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
```

```
section .text  
global _start
```

```
_start:
```