

General Registers

EAX	Accumulator
EBX	Base
ECX	Counter
EDX	Data

General Registers: specific values are expected when calling the kernel.

Pointer-Registers

ESP	Stackpointer
EBP	Basepointer
EIP	Instructionpointer

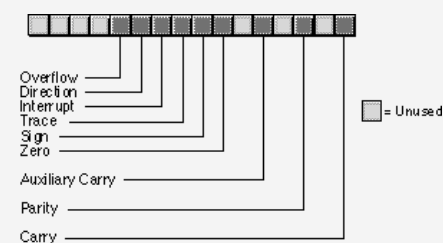
Index-Registers

ESI	Source Index
EDI	Destination Index

Segment- Registers

ECS	Code-Segment
EDS	Data-Segment
ESS	Stack-Segment
EES	Extra-Segment

Flags



NASM Basics

-f	filesystem
-g	debuginfos
-o	output

Compiling a Code

```
nasm -f elf32 -g -o filename.o
filename.nasm
ld -o filename filename.o
```

in 64bit Architecture use -f elf64

Syscall-Numbers Linux

EAX	Name(EBX, ECX, EDX)
1	exit(int)
2	fork(pointer)
3	read(uint, char*, int)
4	write(uint, char*, int)
5	open(char *, int, int)

Linux Syscall Reference

NASM Code-Sections

.text	Code
.data	initialized Data
.bss	uninitialized Data

Example

```
global _start
.data
    msg db " Hello World",0xa
    len equ $-msg
.text
_start:
    mov eax, 0x4
    mov ebx, 0x1
    mov ecx, msg
    mov edx, len
    int 0x80
exit:
    mov eax, 0x1
    mov ebx, 0x1
    int 0x80
```

Misc

int <i>Nr</i>	call Interrupt <i>Nr</i>
call <i>label</i>	jumps to <i>label</i>
ret	returns to call
nop	no operation
lea <i>dest,src</i>	load effective addr. to <i>dest</i>
int 0x80	calls the Kernel in Linux

Logical Operations

neg <i>op</i>	two-Complement
not <i>op</i>	invert each bit
and <i>dest,source</i>	<i>dest</i> = <i>dest</i> ^ <i>source</i>
or <i>dest,source</i>	<i>dest</i> = <i>dest</i> v <i>source</i>
xor <i>dest, source</i>	<i>dest</i> = <i>dest</i> XOR <i>source</i>

Control / Jumps (signed Int)

cmp <i>op1,op2</i>	Compare <i>op1</i> with <i>op2</i>
test <i>op1,op2</i>	bitwise comparison
jmp <i>dest</i>	unconditional Jump
je <i>dest</i>	Jump if equal
jne <i>dest</i>	Jump if not equal
jz <i>dest</i>	Jump if zero
jnz <i>dest</i>	Jump if not zero
kg <i>dest</i>	Jump if greater
jge <i>dest</i>	Jump if greater or equal
jl <i>dest</i>	Jump if less
jle <i>dest</i>	Jump if less or equal

For unsigned Integer use ja, jae (above) or jb, jbe (below)

Mnemonics Intel

mov <i>dest, source</i>	Moves Data
add <i>dest, value</i>	Add <i>value</i> to <i>dest</i>
sub <i>dest,value</i>	Subtract <i>value3</i> from <i>dest*</i>
inc <i>dest</i>	Increment <i>dest</i>
dec <i>dest</i>	Decrement <i>dest</i>



By Gregor Lüdi (Siniansung)
cheatography.com/siniansung/
www.ken.ch/%7elueg

Published 13th January, 2015.
 Last updated 11th May, 2016.
 Page 1 of 2.

Sponsored by [Readable.com](https://readable.com)
 Measure your website readability!
<https://readable.com>

Mnemonics Intel (cont)

`mul src` Multiply EAX and
 src

`imul dest, source` *dest = dest **
 source

General Structure:

`[label] mnemonic [operands] [;co
mment]`

Stack Operations

`push source` Insert Value onto the stack

`pop dest` Remove value from stack

Stack is a LIFO-Storage (Last In First Out)



By **Gregor Lüdi** (Siniansung)
cheatography.com/siniansung/
www.ken.ch/%7elueg

Published 13th January, 2015.

Last updated 11th May, 2016.

Page 2 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>