Comprehensive x86 NASM Cheat Sheet (Linux, Intel Syntax)

Introduction

This cheat sheet explains every element of x86 (32-bit) NASM assembly syntax for Linux, using Intel notation and `int 0x80` system calls. It covers instructions, directives, registers, addressing modes, flags, and system calls, with detailed descriptions and examples. Designed for Debian 13 with NASM, it aims to be a complete reference for writing and understanding assembly programs.

1 Syntax Overview

1.1 Program Structure

Term	Description	Example
section	Defines a program segment (`.text` for code, `.data` for initialized data, `.bss` for uninitialized data).	`section .text`
`global`	Declares a symbol (e.g., `_start`) visible to the linker.	`global _start`
`label`	A named address for jumps or calls (e.g., `_start`, `loop`).	`_start:` `loop:`
`;`	Comment marker; ignores rest of the line.	`; This is a comment`

1.2 Registers

Term	Description	Example
2-bit registers	General-purpose: `EAX`, `EBX`, `ECX`, `EDX`, `ESI`, `EDI`; Stack: `ESP` (stack pointer), `EBP` (base pointer).	`mov eax, ebx`
16-bit registers	Subsets of 32-bit: `AX`, `BX`, `CX`, `DX`, `SI`, `DI`, `SP`, `BP`.	`mov ax, 0x1234`
8-bit registers	Low/high bytes: `AL`, `AH` (from `AX`), `BL`, `BH`, `CL`, `CH`, `DL`, `DH`.	`mov al, 0xff`
`EIP`	Instruction pointer (not directly accessible). Holds address of next instruction.	`call` modifies `EIP`

1.3 Addressing Modes

Term	Description	Example
mmediate	Literal value (decimal or hex).	`10`, `0xff`
Register	Direct register access.	`eax`, `al`
Direct memory	Fixed memory address in square brackets.	`[0x1000]`
Indirect memory	Address in a register.	`[eax]`
Indexed memory	Register + offset.	`[eax+4]`
Size specifiers	`byte`, `word`, `dword` for memory operations.	`mov byte [eax], 1`

1.4 Restrictions

- Memory-to-memory moves are invalid (e.g., `mov [eax], [ebx]`).
- Memory operations require size specifiers if ambiguous (e.g., `byte`, `word`).

2 Data Movement Instructions

Term	Description	Example

MOV dest, src	Copies src (reg, mem, imm) to dest (reg, mem); src unchanged; not both mem.	`mov eax, 10`
`mov eax, ebx`		
`mov eax, [0x1000]`		
`mov eax, [ebx+4]`		
`PUSH item`	Pushes item (reg, imm) to stack; decrements ESP.	`push 32`
`push eax`		
`POP reg`	Pops stack to reg; increments ESP.	`pop eax`
`LEA dest, addr`	Loads effective address of addr into dest (reg only).	`lea eax, [ebx+4]`
`XCHG dest, src`	Swaps dest and src (reg or mem, not both mem).	`xchg eax, ebx`

3 Arithmetic Instructions

Term	Description	Example
ADD dest, src	dest += src; sets flags.	`add esi, 10`
`SUB dest, src`	dest -= src; sets flags.	`sub eax, ebx`
`MUL reg`	Unsigned: EDX:EAX = EAX * reg; sets flags.	`mul esi`
`DIV reg`	Unsigned: EAX = EDX:EAX ÷ reg, EDX = remainder; sets flags.	`div edi`
`INC dest`	dest += 1; sets flags.	`inc eax`
`DEC dest`	dest -= 1; sets flags.	`dec word [0x1000]`

4 Bitwise Operations

Term	Description	Example
AND dest, src	dest = dest & src; sets flags.	`and ebx, eax`
`OR dest, src`	dest = dest | src; sets flags.	`or eax, [0x2000]`
`XOR dest, src`	dest = dest ^src; sets flags.	`xor ebx, 0xffffffff`
`NOT dest`	dest = ~dest; sets flags.	`not eax`
`SHL dest, count`	dest = dest « count; sets flags.	`shl eax, 2`
`SHR dest, count`	dest = dest » count; sets flags.	`shr dword [eax], 4`

5 Control Flow

Term	Description	Example
CMP b, a	Subtracts a from b to set flags (ZF, SF, CF, OF) without storing result.	`cmp eax, 0`
`TEST reg, imm`	Bitwise AND reg & imm to set ZF; no result stored.	`test eax, 0xffff`
`JMP label`	Unconditional jump to label.	`jmp exit`
`JMP reg`	Jump to address in reg.	`jmp eax`
`JE label`	Jump if equal (ZF=1).	`je endloop`
`JNE label`	Jump if not equal (ZF=0).	`jne loopstart`
`JG label`	Jump if greater (signed, ZF=0, SF=OF).	`jg exit`
`JGE label`	Jump if greater/equal (signed, SF=OF or ZF=1).	`jge format_disk`
`JL label`	Jump if less (signed, SF≠OF).	`jl error`
`JLE label`	Jump if less/equal (signed, ZF=1 or SF≠OF).	`jle finish`
`JZ label`	Jump if zero (ZF=1, after TEST).	`jz looparound`
`JNZ label`	Jump if not zero (ZF=0, after TEST).	`jnz error`

`CALL label`	Pushes `EIP`, jumps to `label`.	`call format_disk`
`RET`	Pops `EIP`, returns.	`ret`
`LOOP label`	Decrements `ECX`, jumps to `label` if `ECX` $\neq$ `0`.	`loop repeat`

6 System Calls (Linux x86)

Term	Description	Example

write	`EAX=4`, `EBX`: file descriptor (1=stdout), `ECX`: buffer, `EDX`: length; invokes `int 0x80`.	`mov eax, 4`
		`mov ebx, 1`
		`mov ecx, msg`
		`mov edx, 13`
		`int 0x80`

exit	`EAX=1`, `EBX`: exit code; invokes `int 0x80`.	`mov eax, 1`
		`mov ebx, 0`
		`int 0x80`

read	`EAX=3`, `EBX`: file descriptor (0=stdin), `ECX`: buffer, `EDX`: length; invokes `int 0x80`.	`mov eax, 3`
		`mov ebx, 0`
		`mov ecx, buf`
		`mov edx, 64`
		`int 0x80`

`int 0x80`	Triggers Linux kernel to execute syscall in `EAX`.	`int 0x80`

7 Data Declarations

Term	Description	Example
db	Defines byte(s) (8-bit).	`msg db 'Hello', 10`
dw	Defines word(s) (16-bit).	`num dw 1234`
dd	Defines doubleword(s) (32-bit).	`num dd 12345678`
equ	Defines a constant (no memory allocation).	`msg_len equ $ - msg`
`resb n`	Reserves n bytes (uninitialized).	`buf resb 64`
`resw n`	Reserves n words (2 bytes each).	`buf resw 32`
`resd n`	Reserves n doublewords (4 bytes each).	`buf resd 16`
`$`	Current address in section.	`msg_len equ $ - msg`

8 Flags (Set by CMP, TEST, Arithmetic)

Term	Description	Example
ZF	Zero Flag: Set if result is 0.	`cmp eax, eax` sets `ZF=1`
SF	Sign Flag: Set if result is negative (MSB=1).	`sub eax, ebx` may set `SF`
CF	Carry Flag: Set if unsigned overflow.	`add eax, ebx` may set `CF`
OF	Overflow Flag: Set if signed overflow.	`add eax, ebx` may set `OF`

9 Miscellaneous Instructions

Term	Description	Example
NOP	No operation (does nothing, uses 1 cycle).	`nop`

HLT	Halts CPU until interrupt.	hlt
CLI	Clears interrupt flag (disables interrupts).	cli
STI	Sets interrupt flag (enables interrupts).	sti

10 Example: Hello, World!

```
section .data
    msg db 'Hello, World!', 10  ; String with newline
    msg_len equ $ - msg         ; Length of string

section .text
    global _start

_start:                         ; Entry point
    ; Write to stdout
    mov eax, 4                  ; Syscall number: write
    mov ebx, 1                  ; File descriptor: stdout
    mov ecx, msg                ; Buffer address
    mov edx, msg_len            ; Buffer length
    int 0x80                    ; Invoke syscall

    ; Exit
    mov eax, 1                  ; Syscall number: exit
    mov ebx, 0                  ; Exit code: 0
    int 0x80                    ; Invoke syscall
```

Assemble and Run (Debian 13):

```
nasm -f elf32 hello.asm -o hello.o
ld -m elf_i386 hello.o -o hello
./hello
```

11 Additional Examples

11.1 Loop Example

```
section .data
    counter dd 5                ; Loop 5 times

section .text
    global _start

_start:
    mov ecx, [counter]          ; Load loop count
loop_start:
    ; Do something (e.g., write)
    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, msg_len
    int 0x80
    loop loop_start             ; Decrement ECX, jump if not zero

    mov eax, 1
    mov ebx, 0
    int 0x80
```

11.2 Conditional Jump

```
section .data
    num1 dd 10
    num2 dd 20

section .text
    global _start
```

```
_start:
    mov eax, [num1]             ; Load num1
    cmp eax, [num2]             ; Compare with num2
    jg greater                  ; Jump if num1 > num2
    mov ebx, 1                  ; Else, set exit code 1
    jmp exit
greater:
    mov ebx, 0                  ; Set exit code 0
exit:
    mov eax, 1
    int 0x80
```

12 Tips

- **Assemble**: `nasm -f elf32 file.asm -o file.o`
- **Link**: `ld -m elf`$_i386 file.o - ofile$**Debug** : Usegdb(gdb `./file`, `break _start`, `run`, `nexti`).
- **Resources**: `man 2 syscall`, nasm.us, godbolt.org for testing.
- **Note**: Use `byte`, `word`, or `dword` for memory operations to avoid ambiguity.