x86 NASM I/O and OS Commands Cheat Sheet (Linux, Intel Syntax)

Introduction

This cheat sheet covers basic I/O operations, function-related instructions, and Linux system calls for OS commands in x86 (32-bit) NASM assembly on Linux (Debian 13). It focuses on input/output (`read`, `write`), function mechanics (`_start`, `call`, `ret`), and system calls (open, `close`, etc.) using `int 0x80`. All terms are explained with practical examples.

General Notes

- **Architecture**: x86 (32-bit), NASM Intel syntax, Linux system calls via `int 0x80`.
- **Registers**: 32-bit (`EAX`, `EBX`, `ECX`, `EDX`, `ESI`, `EDI`, `EBP`, `ESP`); `EAX` holds syscall number, return value; `EBX`, `ECX`, `EDX` for arguments.
- **Memory**: Use `[addr]` for dereferencing (e.g., `mov eax, [0x1000]`).
- **Addressing**: Immediate (`10`, `0xff`), register (`eax`), memory (`[eax]`, `[0x1000]`).
- **Size Specifiers**: `byte`, `word`, `dword` for memory ops (e.g., `mov byte [eax], 1`).
- **Restrictions**: No memory-to-memory moves (e.g., `mov [eax], [ebx]` invalid).
- **Sections**: `.text` (code), `.data` (data), `.bss` (uninitialized).

1 I/O System Calls

Term	Description	Example
write	`EAX=4`, `EBX`: file descriptor (1=stdout), `ECX`: buffer address, `EDX`: length; writes data to file descriptor.	`mov eax, 4`

`mov ebx, 1`
`mov ecx, msg`
`mov edx, 13`
`int 0x80`		

read	`EAX=3`, `EBX`: file descriptor (0=stdin), `ECX`: buffer address, `EDX`: max length; reads data into buffer.	`mov eax, 3`

`mov ebx, 0`
`mov ecx, buf`
`mov edx, 64`
`int 0x80`		

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

2 File-Related System Calls

Term	Description	Example
open	`EAX=5`, `EBX`: pathname address, `ECX`: flags (0=read, 1=write, 2=read/write), `EDX`: mode (e.g., 0644); opens file, returns file descriptor in `EAX`.	`mov eax, 5`

`mov ebx, filename`
`mov ecx, 0`
`mov edx, 0644`
`int 0x80`		

close	`EAX=6`, `EBX`: file descriptor; closes file.	`mov eax, 6`
`mov ebx, 3` `int 0x80`		

3 Process Control

Term	Description	Example

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

4 Function-Related Instructions

Term	Description	Example
_start `_start:`	Label for program entry point; must be declared with `global _start`.	`global _start`
`CALL label`	Pushes `EIP` (next instruction address), jumps to `label`.	`call my_func`
`RET`	Pops `EIP`, returns to caller.	`ret`

`PUSH item`
 	
`push 32`
	Pushes `item` (reg, imm) to stack; decrements `ESP`.	`push eax`

`POP reg`	Pops stack to `reg`; increments `ESP`.	`pop eax`

5 Data Declarations for I/O

Term	Description	Example
db	Defines byte(s) for strings or data.	`msg db 'Hello', 10`
equ	Defines constant (e.g., string length).	`msg_len equ $ - msg`
`resb n`	Reserves n bytes for buffers.	`buf resb 64`
`$`	Current address in section.	`msg_len equ $ - msg`

6 Example: Read Input, Write Output

```
section .data
    prompt db 'Enter text: ', 0
    prompt_len equ $ - prompt
    msg db 'You typed: ', 0
    msg_len equ $ - msg

section .bss
    buffer resb 64              ; Buffer for input

section .text
    global _start

_start:
    ; Write prompt
    mov eax, 4                  ; Syscall: write
    mov ebx, 1                  ; fd: stdout
    mov ecx, prompt             ; Buffer
    mov edx, prompt_len         ; Length
    int 0x80

    ; Read input
    mov eax, 3                  ; Syscall: read
    mov ebx, 0                  ; fd: stdin
    mov ecx, buffer             ; Buffer
    mov edx, 64                 ; Max length
    int 0x80

    ; Write "You typed: "
    mov eax, 4
    mov ebx, 1
```

```
    mov ecx, msg
    mov edx, msg_len
    int 0x80

    ; Write input back
    mov edx, eax                ; Length read
    mov eax, 4
    mov ebx, 1
    mov ecx, buffer
    int 0x80

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

Assemble and Run:

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

7 Example: Open and Read File

```
section .data
    filename db 'test.txt', 0
    msg db 'File contents: ', 0
    msg_len equ $ - msg

section .bss
    buffer resb 64

section .text
    global _start

_start:
    ; Open file
    mov eax, 5                  ; Syscall: open
    mov ebx, filename           ; File path
    mov ecx, 0                  ; Read-only
    mov edx, 0644               ; Permissions
    int 0x80
    mov edi, eax                ; Save fd

    ; Read file
    mov eax, 3
    mov ebx, edi
    mov ecx, buffer
    mov edx, 64
    int 0x80
    mov esi, eax                ; Save length

    ; Write "File contents: "
    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, msg_len
    int 0x80

    ; Write file contents
    mov eax, 4
    mov ebx, 1
    mov ecx, buffer
    mov edx, esi
    int 0x80

    ; Close file
```

```
    mov eax, 6
    mov ebx, edi
    int 0x80

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

Assemble and Run:

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

8 Example: Function Call

```
section .data
    msg db 'Function called!', 10
    msg_len equ $ - msg

section .text
    global _start

_start:
    call print_msg              ; Call function

    mov eax, 1
    mov ebx, 0
    int 0x80

print_msg:
    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, msg_len
    int 0x80
    ret
```

Assemble and Run:

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

9 Tips

- **Assemble**: `nasm -f elf32 file.asm -o file.o`
- **Link**: `ld -m elf`$_i386 file.o - o file$**Debug** : Usegdb(gdb `./file`, `break _start`, `run`, `nexti`).
- **File Modes**: open flags: 0 (read), 1 (write), 2 (read/write); mode: 0644 (rw-r–r–).
- **Resources**: `man 2 syscall`, nasm.us, godbolt.org.