x86 NASM Part 3: Random Numbers and Functions Cheat Sheet (Linux, Intel Syntax)

Introduction

This cheat sheet (Part 3) covers creating functions (stack frames, parameters, locals) and generating random numbers in x86 (32-bit) NASM assembly on Linux (Debian 13). It builds on Parts 1 (syntax) and 2 (I/O/OS). Functions use `call`/`ret` and stack management; random numbers use /dev/urandom (true random) or simple PRNG (pseudo-random). All terms explained with examples.

General Notes

- **Architecture**: x86 (32-bit), NASM Intel syntax, Linux system calls via `int 0x80`.
- **Registers**: Use `EBP` for stack frame base, `ESP` for stack pointer; `EAX` for returns.

• 1 Creating Functions

Term	Description	Example
CALL label	Pushes `EIP` (return address), jumps to `label`; sets up stack for args.	`call add_numbers`
`RET`	Pops `EIP`, returns to caller; optional `RET n` cleans stack.	`ret`
`PUSH item`	Pushes arg or local to stack (decrements `ESP` by 4).	`push 5` `push ebx`
`POP reg`	Pops value from stack to reg (increments `ESP` by 4).	`pop eax`
`ENTER n, 0`	Sets up stack frame: pushes `EBP`, `ESP` to `EBP`, allocates `n` bytes locals.	`enter 8, 0`
`LEAVE`	Tears down frame: `EBP` to `ESP`, pops `EBP`.	`leave`
`MOV EBP, ESP`	Manual frame setup (alternative to `ENTER`).	`mov ebp, esp` `sub esp, 8`

1.1 Function Parameters and Locals

– **Parameters**: Push args before `call`; access via `[EBP + 8]` (first arg at +8 from `EBP`).
– **Local Variables**: Allocate with `SUB ESP, n`; access via `[EBP - offset]`.
– **Return Value**: Place in `EAX`.

2 Random Number Generation

2.1 True Random (/dev/urandom)

Term	Description	Example

open /dev/urandom	`EAX=5`, `EBX`: ”/dev/urandom\0”, `ECX`: 0 (read), `EDX`: 0666; returns fd in `EAX`.	`mov eax, 5`
		`mov ebx, urandom_path`
		`mov ecx, 0`
		`int 0x80`

`read from fd`	`EAX=3`, `EBX`: fd, `ECX`: buffer, `EDX`: 4 (for 32-bit rand); reads bytes to buffer.	`mov eax, 3`
		`mov ebx, eax`
		`mov ecx, rand_buf`
		`mov edx, 4`
		`int 0x80`

close fd mov ebx, fd int 0x80	EAX=6, EBX: fd.	mov eax, 6

2.2 Pseudo-Random (LCG Algorithm)

Term	Description	Example
LCG Seed mul dword [a_const] add eax, c_const	Initialize seed (e.g., time or fixed); rand = (a * rand + c) mod m (a=1664525, c=1013904223, m=2^{32}).	mov eax, seed
MUL reg AND for mod	Multiplies for LCG calculation. Mod 2^{32} $via over flow (implicit)$.	mul ebx and eax, 0x7fffffff (for positive)

3 Example: Add Function

```
section .data
    num1 dd 5
    num2 dd 3

section .text
    global _start

_start:
    push dword [num2]           ; Push arg2
    push dword [num1]           ; Push arg1
    call add\_numbers           ; Call function
    add esp, 8                  ; Clean stack (2 args * 4)
    mov ebx, eax                ; Return value

    mov eax, 1
    mov ebx, eax                ; Exit with result
    int 0x80

add\_numbers:
    push ebp
    mov ebp, esp
    mov eax, [ebp + 8]          ; Arg1
    add eax, [ebp + 12]         ; Arg2
    pop ebp
    ret
```

Assemble and Run:

```
nasm -f elf32 add.asm -o add.o
ld -m elf_i386 add.o -o add
./add  ; Exit code: 8
```

4 Example: Generate Random Number (/dev/urandom)

```
section .data
    urandom\_path db '/dev/urandom', 0
    msg db 'Random: ', 0
    msg\_len equ $ - msg

section .bss
    rand\_buf resb 4             ; 32-bit random
    fd resd 1

section .text
    global _start

_start:
```

```
    ; Open /dev/urandom
    mov eax, 5
    mov ebx, urandom\_path
    mov ecx, 0                  ; Read-only
    mov edx, 0666
    int 0x80
    mov [fd], eax

    ; Read 4 bytes
    mov eax, 3
    mov ebx, [fd]
    mov ecx, rand\_buf
    mov edx, 4
    int 0x80

    ; Close
    mov eax, 6
    mov ebx, [fd]
    int 0x80

    ; Write "Random: " + number (simplified; use EAX for display)
    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, msg\_len
    int 0x80

    ; Display rand\_buf (add hex print logic here)
    mov eax, 1
    mov ebx, 0
    int 0x80
```

Assemble and Run:

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

5 Example: Pseudo-Random Function (LCG)

```
section .data
    a\_const dd 1664525
    c\_const dd 1013904223
    seed dd 12345               ; Initial seed

section .text
    global _start

_start:
    call rand\_lcg              ; Generate random
    mov ebx, eax                ; Store result

    mov eax, 1
    mov ebx, eax
    int 0x80

rand\_lcg:
    push ebp
    mov ebp, esp
    mov eax, [seed]             ; Load seed
    mul dword [a\_const]        ; eax *= a
    add eax, [c\_const]         ; + c
    mov [seed], eax             ; Update seed
    and eax, 0x7fffffff         ; Positive range
    pop ebp
    ret
```

Assemble and Run:

```
nasm -f elf32 lcg.asm -o lcg.o
```

```
ld -m elf_i386 lcg.o -o lcg
./lcg  ; Exit code: random value
```

6 Tips

– **Functions**: Always save/restore `EBP`; clean stack after `call` if passing args.
– **Random**: Use /dev/urandom for crypto; LCG for simple simulations (not secure).
– **Assemble**: `nasm -f elf32 file.asm -o file.o`
– **Link**: `ld -m elf_i386 file.o -o file`
– **Debug**: gdb `./file` for stack tracing.
– **Resources**: nasm.us, man 2 open/read, godbolt.org.