**Explanation of the Commands**

**1. NASM Command**

nasm -f elf32 -g -F stabs -l lst.l -o obj.o po1.asm

This command assembles your **po1.asm** file using the **Netwide Assembler (NASM)** into an object file.

* **nasm** → Calls the NASM assembler.
* **-f elf32** → Generates a **32-bit ELF (Executable and Linkable Format)** object file (required for linking on 32-bit systems).
* **-g** → Adds debugging information.
* **-F stabs** → Uses the **stabs** debugging format (used for debugging in GDB).
* **-l lst.l** → Generates a **listing file** (lst.l), which helps in debugging by showing assembled instructions.
* **-o obj.o** → Specifies the output file **obj.o** (the object file containing the machine code).

**2. Linking Command**

ld -m elf\_i386 -o exe.X obj.o

This command links the object file (obj.o) into an **executable** file.

* **ld** → Calls the GNU **linker**.
* **-m elf\_i386** → Specifies the **32-bit ELF format** for linking (since Linux uses ELF format).
* **-o exe.X** → Specifies the output executable file name (exe.X).

After this command, exe.X is an **executable file** that can be run using:

./exe.X

**Explanation of the Assembly Code**

section .data

section .bss

section .text

global \_start

\_start:

mov ebx,0

mov eax,1

int 0x80

**1. Sections**

* **section .data** → Reserved for initialized **data** (e.g., variables with predefined values). You didn't use it in this code.
* **section .bss** → Reserved for **uninitialized** variables. Also unused here.
* **section .text** → Contains the **executable code** (assembly instructions).

**2. \_start Label**

assembly

CopyEdit

global \_start

\_start:

* **global \_start** → Marks \_start as an entry point (needed for linking).
* **\_start:** → This is where the program begins execution.

**3. System Call to Exit**

mov ebx,0 ; Exit status (0 = success)

mov eax,1 ; System call number for sys\_exit

int 0x80 ; Invoke kernel interrupt to exit

This performs the **exit system call**, telling the Linux kernel to terminate the program.

* **mov ebx,0** → Loads 0 into EBX (exit status, 0 = success).
* **mov eax,1** → Loads 1 into EAX, which is the syscall number for sys\_exit (system call ID for exiting the program).
* **int 0x80** → Triggers a software interrupt (int 0x80) to call the **Linux kernel**, which executes the exit syscall.

**Execution Flow**

1. The program starts execution at \_start.
2. It loads 0 into EBX (exit code).
3. It loads 1 into EAX (system call number for exit).
4. It calls int 0x80, which tells the kernel to exit.

**What This Program Does**

* It simply **exits** immediately with exit code 0, meaning **successful termination**.
* It does not print anything or take user input.