section .data // This section is for **initialized data** (like strings or constants).In our code, it’s empty because we’re not using constants or initialized variables.

section .bss // The **.bss** section is for **uninitialized** variables.

buf resb 1 // buf resb 1 tells the assembler:

* Reserve 1 **byte** of memory named buf
* This is where the program will store the character typed by the user.

section .text // .text is the **code section**, where your instructions live

global \_start // global \_start tells the linker to treat \_start as the entry point of the program (like main() in C).

\_start: // Execution Starts Here

Reading the input from the user

mov eax, 3 ;sys call //  eax holds the **syscall number**.

 3 is the syscall number for sys\_read

mov ebx, 0 //  ebx holds the **file descriptor**.

 0 = **standard input (keyboard)**

mov ecx, buf //  ecx holds the **memory address to store input**.

 We use buf — a 1-byte memory space we reserved.

mov edx, 1 // edx holds the **number of bytes to read**

int 0x80 // **Interrupt** to call the Linux kernel and perform the system call with:

* eax = 3 (read)
* ebx = 0 (stdin)
* ecx = buf (where to store)
* edx = 1 (how many bytes)

Incrementing the input by 1

mov al, Byte[buf] // Read the byte at buf into register al (the **lower 8 bits** of eax).

inc al // al = al + 1

mov byte[buf],al // Store the incremented value back into buf

Writing the output in the terminal

mov eax, 4 // syscall for sys\_write

mov ebx, 1 // file descriptor for **stdout** (the terminal)

mov ecx, buf // address of the buffer — the modified character

mov edx, 1 // we only want to write 1 byte

int 0x80 // Triggers the system call to **write** 1 byte from buf to stdout

mov eax, 1 // syscall number for sys\_exit

mov ebx, 0 // exit code 0 (meaning "OK" or success)

int 0x80 // Calls the kernel to exit the program