***COSC 519***

***Group Number: 2***

***05/12/2025***

***Group Members:***

Mikiyas Semu

***Long Mode Bootloader Report***

# **Objective**

The goal of this project is to get hands-on experience with how modern x86-64 computers start up by writing our own custom bootloader in NASM. This bootloader takes the system through all the major CPU states — starting in Real Mode (the initial state from BIOS), then moving into Protected Mode, and finally entering 64-bit Long Mode. Along the way, we learned how to manage memory at a very low level, set up paging, and directly interact with hardware. It’s a deep dive into how an operating system really begins behind the scenes.

# **Implementation Overview** We designed our bootloader in two stages to manage the transition from power-on to full 64-bit operation.

## **Stage 1 (boot.asm)** This is the very first code that runs, loaded by the BIOS into memory at address 0x7C00. Its job is simple but crucial:

* It sets up basic segment registers and the stack.
* Then it prints a short boot message using BIOS services.
* After that, it uses a BIOS disk interrupt to load the second stage (two sectors) from disk into memory at 0x7E00.
* If everything goes smoothly, it jumps to that location to hand off control to Stage 2.

**Stage 2 (loader.asm)** This part of the code picks up where Stage 1 left off, but it's responsible for all the heavy lifting. It:

* Starts again in Real Mode at 0x7E00, resets segment and stack settings just to be safe.
* Loads a 32-bit Global Descriptor Table (GDT), which is required for Protected Mode.
* Switches into Protected Mode.
* Builds paging structures using PAE (Physical Address Extension).
* Enables Long Mode by setting a special bit in the EFER MSR register.
* Turns on paging, then does a far jump to switch into 64-bit Long Mode.
* Finally, it prints messages directly to the screen and executes a 64-bit instruction to confirm we made it into Long Mode.

# **Key Transitions**

## **Real Mode to Protected Mode** To move into Protected Mode, we first load a GDT that defines our code and data segments. Then we enable the PE (Protection Enable) bit in the CR0 register. But that’s not enough — to actually switch modes, we also need to do a far jump. This flushes the instruction pipeline and updates the code segment (CS) to use the new GDT settings, officially putting the CPU into Protected Mode.

mov eax, cr0  
or eax, 1  
mov cr0, eax  
  
db 0xEA  
dw protected\_mode  
dw 0x08

Protected Mode to Long Mode

Paging is enabled using PAE. We create a minimal identity-mapped page table:

mov dword [0x1000], 0x2003 ; PML4 -> PDPT  
mov dword [0x2000], 0x83 ; PDPT -> 1GB page  
mov cr3, 0x1000 ; Set CR3 to PML4 base

Enable PAE and Long Mode:

mov eax, cr4  
or eax, 1 << 5  
mov cr4, eax  
  
mov ecx, 0xC0000080 ; EFER MSR  
rdmsr  
or eax, 1 << 8 ; Enable LME  
wrmsr

Enable paging and jump into Long Mode:

mov eax, cr0  
or eax, 0x80000000  
mov cr0, eax  
  
lgdt [gdt64\_descriptor]  
jmp 0x08:long\_mode

## **Long Mode Execution**

In 64-bit mode, we:

- Clear segment registers

- Disable cursor blinking via VGA ports

- Print messages directly to video memory at 0xB8000

- Execute and display a 64-bit instruction

64-bit instruction executed:

mov rsi, 0x123456789ABCDEF0  
call print\_hex

The print\_hex function takes the 64-bit value in the RSI register, breaks it down one nibble (4 bits) at a time, converts each part into a readable hexadecimal character, and then writes it to the screen using stosw.

## **Build and Execution Instructions**

To automate and facilitate the build and running process, we used a Makefile that takes care of compiling, image creation, and launching the bootloader in QEMU.

The make command builds the bootloader and creates a bootable image. make run launches the image in QEMU for testing, and make clean removes all generated files to start fresh.

Commands:

make

nasm -f bin boot.asm -o boot.bin  
nasm -f bin loader.asm -o loader.bin  
dd if=/dev/zero of=boot.img bs=512 count=2880  
dd if=boot.bin of=boot.img conv=notrunc  
dd if=loader.bin of=boot.img bs=512 seek=1 conv=notrunc

make run:

qemu-system-x86\_64 -drive format=raw,file=boot.img

make clean:

rm -f \*.bin \*.img

# 

# **Conclusion**

This project successfully shows how a computer transitions from Real Mode to Protected Mode and ultimately into 64-bit Long Mode — all through a custom bootloader we wrote. We handled every part of the process manually, including setting up paging, configuring the GDT, and working directly with hardware memory. We didn’t rely on GRUB or any pre-made operating system tools. In the end, the on-screen output and the execution of real 64-bit instructions clearly confirmed that we had entered Long Mode as intended, fully meeting the goals of the project.