Writeup -1

Tushar Mohan
2019393

Description :

At Bootup, BIOS selects a boot device. It can be a HDD, SSD, USB or CD-ROM. Then it copies the first sector from this device into the physical memory of the computer at the memory address 0x7C00. This boot sector has a memory limit of 512 bytes. This sector consists of bootl loader code, a Global Descriptor Table and an indicator bit which tell the BIOS whether the binary is bootable or not. The BIOS then hands over the control to the CPU after it instructs it to jump to the beginning of the boot loader code.

The boot loader implemented does the following tasks in a sequence:

1. Setup the stack
2. Switch from Real Mode (16-bit) to Protected Mode (32-bit)
3. Setup the Global Descriptor Table for the Protected mode
4. Print “Hello World”
5. Print the contents of the CR0 register
6. Finally Halt

Some Specifications as per the code:

[ORG 0x7C00] - tells the assembler to start the addressing from the Origin, 0x7C00. As BIOS loads the bootloader at the physical address 0x7C00.

[BITS 16] - is an assembler directive. This tells the assembler in which mode should the code run.

times 510 - ($ - $$) db 0 - clears out extra memory needed to load contents of the bootloader code as the size of the sector is mere 512 bytes which is too less. $ signifies current instruction address. $$ signifies the start of the program address. Hence, ($ - $$) signifies the length of the program.

DW 0xAA55 - tells the BIOS that this is a valid bootloader. If the BIOS doesn’t get 0xAA and 0x55 at the end of the bootloader then it treats it as invalid. Thus, written at the end of the code.

Flow of the code:
The stack is set up by making the base and the stack pointer point to a specific memory address. Then a switch to 32-Bit Protected Mode is initiated. So, interrupts are first diabled. Then, Global Descriptor Table is loaded. The CODE and DATA segment are fully overlapping and spanning the complete 4 GB of addressable memory. Global Descriptor Table consists of the following segments:

1. A null segment descriptor (8 zeroed byted)
2. The 4GB code segment
3. The 4GB data segment

After that, the string “Hello World” is printed and the contents of the CR0 register.

Commands:

- ***nasm -f bin boot_2019393.nasm -o boot***
    - *Creates the object file by the name boot*
- ***qemu-system-x86_64 boot***
    - *Runs the boot file in the qemu emulator.*

References:

- *https://www.viralpatel.net/taj/tutorial/hello_world_bootloader.php*
- *https://dev.to/frosnerd/writing-my-own-boot-loader-3mld*
- *https://github.com/sedflix/lame_bootloader*
- *https://stackoverflow.com/questions/65361946/print-the-contents-of-the-cr0-register*
- *http://3zanders.co.uk/2017/10/16/writing-a-bootloader2/*