We have to write a simple bootloader which would be booted via a legacy bias. On an x86 machine BIOS selects the boot device and copies the first sector of the device into the physical memory address 0x7C00 , this is called boot sector(size 512 bytes). These 512 bytes contain the boot loader code, a partition table, the disk signature, as well as a magic number(by which boot loader identifies if this sector is bootable or not).
As we are going to shift from 16 bit real mode to 32 bit protected mode so we need to tell the assembler whether it should generate 16 bit or 32 bit instructions. For this we use bits 16 and bits 32 directive.
The org 0x7c00 (Origin) , tells the assembler where the code will be in memory after it has been loaded . Now in 16 bit real mode we have 20 bit address bus hence we can access upto 2^16 (~1MB) of memory , but when we switch to protected mode we have 32 bit address bus so theoretically we can access 2^32 (~4GB) of memory but for this we have to manually turn them on to make use of the higher memory. More precisely, one has to turn on the 21st address line in order to make all higher address lines available. This address line is known as the A20 line, since you start counting the lines from zero. This is done by mov ax, 0x2401
    int 0x15.
And int means interrupt and int 0x10 (interrupt used for screen manipulation) and command mov ax, 3 int 0x10 means set the video mode to TextMode (so that we can write on screen).
Now in order to switch to 32 protected mode we have to perform following steps:
  a . cli means disable the interrupts.
  b . Load the GDT descriptor into the GDT register using ldgt instruction (GDT is a data structure used by Intel x86 -family processors in order to define the characteristics of the various memory areas used during program execution, including the base address, the size, and access privileges like executability and writability. These memory areas are called segments in Intel terminology).
  c . Enable protected mode in control register cr0.

Jump to code segment label boot_segment_32 . Here we first move our data segment hello_world to esi register (it is a short of global register) .In protected mode we cannot directly access the BIOS but we can write VGA text buffer directly. This is memory mapped to location 0xb8000.
After this we call label .cr0_label and loop . In cr0_label we first print the value of the cr0 register and then print the string databyte hello world.
After this we assemble the code from our text file into a raw binary file of machine-code instructions. With the **-f bin** flag, we tell NASM that we want a plain binary file (not a complicated Linux executable - we want it as plain as possible!). The **-o boot.bin** part tells NASM to generate the resulting binary in a file called boot.bin .
Link from where I have referred - https://dev.to/frosnerd/writing-my-own-boot-loader-3mld